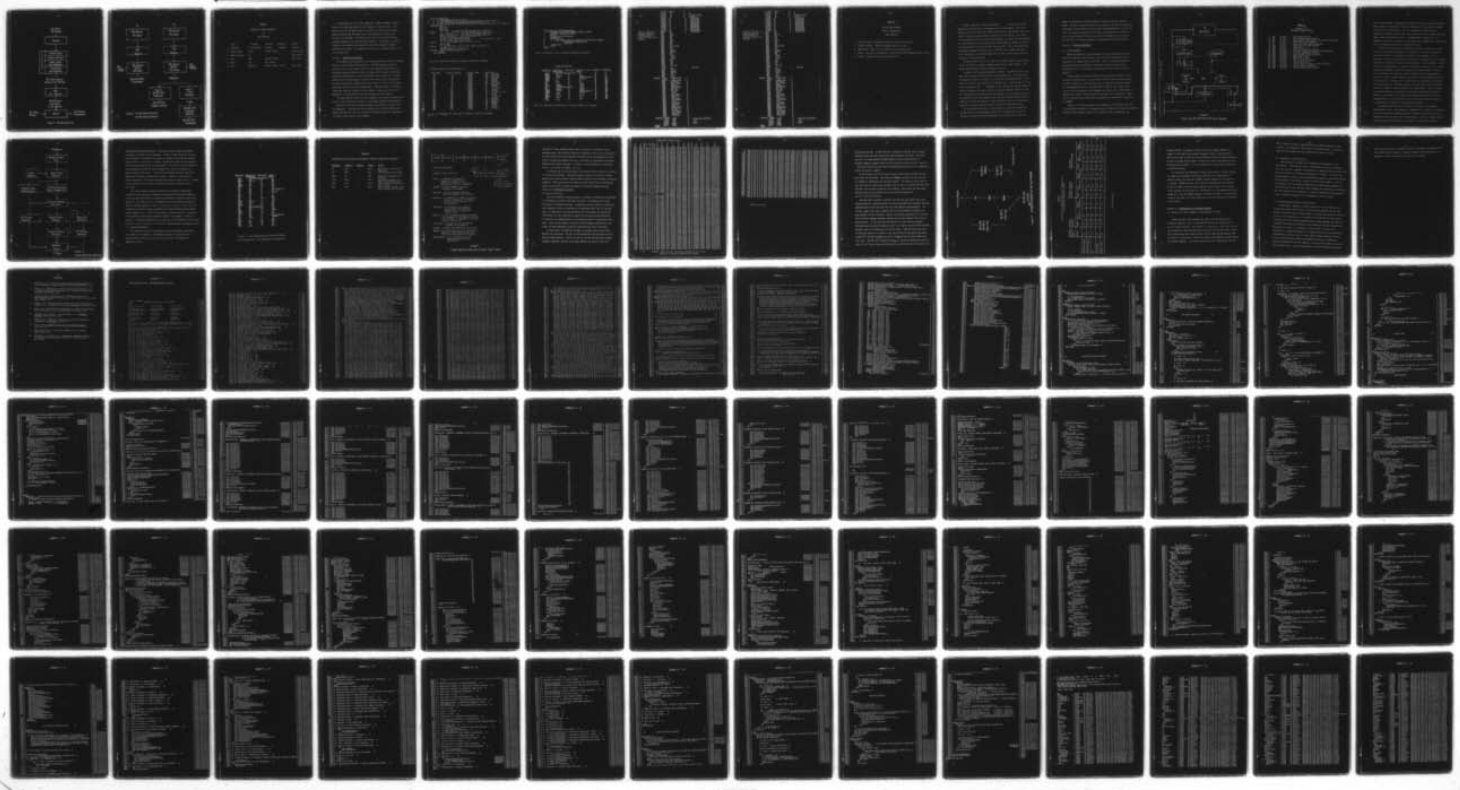
AD-A068 950 GEORGE WASHINGTON UNIV WASHINGTON D C SCHOOL OF ENGI--ETC F/G 9/2
DEVELOPMENT OF EXPERIMENTAL COMPILERS TO GENERATE EMULATORS FOR--ETC(U)
APR 79 R E MERWIN DASG60-78-C-0115

UNCLASSIFIED NL

1 OF 2
AD
A068950





AD A068950

LEVEL II

2

THE
GEORGE
WASHINGTON
UNIVERSITY

STUDENTS FACULTY STUDY R ESEARCH DEVELOPMENT FUT URE CAREER CREATIVITY CO MMUNITY LEADERSHIP TECH NOLOGY FRONTIE ESIGN ENGINEERING APP ENC GEORGE WASHIN NIV

DDC FILE COPY

DDC
RECEIVED
MAY 24 1979
C



79 05 21 137

SCHOOL OF ENGINEERING
AND APPLIED SCIENCE

THIS DOCUMENT HAS BEEN APPROVED FOR PUBLIC RELEASE AND SALE; ITS DISTRIBUTION IS UNLIMITED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: -	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Development of Experimental Compilers to Generate Emulators for the BMD DDP Test Bed From High Level Language.	5. TYPE OF REPORT & PERIOD COVERED: Final, 8/78 to 3/79
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): R.E. Merwin	8. CONTRACT OR GRANT NUMBER(s): DASG 60-78-C-0115
9. PERFORMING ORGANIZATION NAME AND ADDRESS: The George Washington University, Washington, D.C. 20052	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 12 125p
11. CONTROLLING OFFICE NAME AND ADDRESS: The Ballistic Missile Defense Advanced Technology Center Huntsville, AL	12. REPORT DATE: 1 April 1979
	13. NUMBER OF PAGES: 102
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 9 Final rept. Aug 78-Mar 79	15. SECURITY CLASS. (of this report): Unclass.
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

D D C
RECEIVED
MAY 24 1979
C

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microprogram, Compiler, High Level Language

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The use of compilers to go from high level langauges to microprograms is not commonplace. A few experimental compilers have been described but no support of this concept is provided by manufacturers of user microprogrammable computers. In this feasibility study two existing compilers accepting dialects of Pl/1 were considered for generating microprograms for computers with a horizontally encoded control word format. One of the compilers which accepts the PLM language as input and produces quadruple intermediate text

next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

153 370

formats and DEC PDP 11 assembly language statments as outputs, was selected for modification to produce micorprograms. A post processor is described which was added to this compiler to produce microcode for the DEC PDP 11/45 minicomputer. A performance analysis of this compiler concludes the report.

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. SPECIAL
A 23

UNCLASS

Development of Experimental Compilers to Generate Emulators for the BMD DDP Test Bed From High Level Languages

Final Report
April 1, 1979

The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

Sponsored by

The Ballistic Missile Defense Advanced Technology Center

Contract No. DASG60-78-C-0115

School of Engineering and Applied Science

The George Washington University

Washington, D.C. 20052

TABLE OF CONTENTS

Section	Title
1.0	Introduction
	1.1 Background
	1.2 Experimental Objectives
	1.3 Accomplishments
2.0	Compiler Description and Design
3.0	Selection of Approach
4.0	Microcode Generation
	4.1 R Quad Generator
	4.2 Microcode Generator
5.0	Performance Evaluation
	5.1 Evaluation Approach
	5.2 Quad Compiler Performance
6.0	Recommendations for Further Research
	6.1 Conversion of Input Language of Quad Compiler to PASCAL
	6.2 Optimization of Compiler
	6.3 Microprogram Host Machine Interface Study
7.0	Appendix
	7.1 Quad Compiler Listing
	7.2 PLM Representation of Test Programs
	7.3 Quad Representation of Test Programs
	7.4 DEC PDP 11/45 Microcode Representation of Test Programs
	7.5 Quad Processor Routines

FIGURES

1. TWS Functional Flow

2.(a) MPL Compiler Overview
(b) PLM Compiler Overview

3. INTERDATA MOD 3 Microcode for Greatest Element Test Program

4.(a) PLM Source Code for Greatest Element Test Program
(b) Quadruple Representation of Greatest Element Test Program
(c) DEC PDP 11 Assembly Language Representation of Greatest Element Test Program

5. Simplified DEC PDP 11/45 CPU Block Diagram

6. R Quad Processor Overview

7. R Quad Representation of Greatest Element Test Program Derived from Quads Shown in Figure 4(c)

8. R Quad Processor Register Content Table Format

9. Microcode Generated by PLM to Quad Compiler for Greatest Element Test Program

10. Alternate ways to Generate DEC PDP 11/45 Microcode

TABLES

I. Quadruple Formats Generated by Quad Compiler

II. List of Test Programs Used to Evaluate the Quad Compiler

III. DEC PDP 11/45 Microcode Control Fields

IV. Register Quads with Operands Representing Internal Registers or Addresses.

V. Comparison of Alternate Methods for Generation of Microcode for the DEC PDP 11/45

ABSTRACT

The use of compilers to go from high level languages to microprograms is not commonplace. A few experimental compilers have been described but no support of this concept is provided by manufacturers of user microprogrammable computers. In this feasibility study two existing compilers accepting dialects of PL/1 were considered for generating microprograms for computers with a horizontally encoded control word format. One of the compilers which accepts the PLM language as input and produces quadruple intermediate text formats and DEC PDP 11 assembly language statements as outputs, was selected for modification to produce microprograms. A post processor is described which was added to this compiler to produce microcode for the DEC PDP 11/45 minicomputer. A performance analysis of this compiler concludes the report.

Section 1.0 INTRODUCTION

1.1 BACKGROUND

The techniques of microprogramming provide facilities for the implementor of a computer application to optimize the performance of the hardware. Calculations which are carried out repeatedly, e.g. operating system kernels, and digital filter and graphics display routines, can all be speeded up by factors of up to ten with microprogrammed implementations. In spite of the potential performance enhancements possible through microprogramming, there hasn't been a widespread use of these techniques by the users of computers. Much of this reluctance on the part of the computer user is the inherent difficulty of writing microprograms.

Although high level languages (HLL) are widely used to relieve computer programmers from writing applications in machine or assembly language, this facility hasn't been developed to simplify the preparation of microprograms which is a more difficult task than assembly language programming. A few attempts to develop experimental compilers from a HLL to microcode have been described (1,2) but there has been no adoption of this concept by the manufacturers of "user" microprogrammed hardware. The concepts of designing such compilers haven't been extensively developed and the preparation of microcode from a HLL for computers with a horizontal encoded control word hasn't been accomplished to date. This report will describe some techniques which have been developed to establish feasibility of generation of microprograms from a HLL for a range of computer architectures.

The production of support tools to better utilize the facilities offered the application programmer has been a major object of research at

GWU. Abd-Alla and Karlgaard[3] developed one of the first algorithms for automatically optimizing microcode by scanning the object code and identifying segments to be microprogrammed. This work was extended by Moffett[4] and Evans[5]. More recently Fodor[2] implemented a microcode compiler based upon the high level microprogramming language, MPL, proposed by Eckhouse[1]. This research had two objects: one was to see if a translator writing system (TWS), i.e. XPL[6], could be used to implement a compiler for microprograms;and second to determine the efficiency of a compiler developed using a TWS. From this research it was determined that the TWS could be used to implement a compiler for a HLL which directly produces microcode. Further it was demonstrated that the resultant microcode produced by the MOD 3 compiler was nearly as efficient in storage utilization and running time as the equivalent microcode produced by hand coding.

In the later sections of this report the term host machine will be used to designate the computer hardware for which the microprograms are implemented. This terminology is widely used to differentiate the hardware in which the microprograms are executed and the computer machine language being interpreted by the microprograms. The machine language is considered as defining the target machine. In this study the target machine will be the PLM language.

The Ballistic Missile Defense (BMD) distributed data processing (DDP) test bed has a requirement to provide a flexible computational environment and the DEC VAX 11/780 computers selected for the test bed incorporates the user microprogramming feature to support this need. Since direct (hand) generation of microcode is tedious, error prone, and time consuming, it was apparent that some way of simplifying the preparation of microprograms for

these machines was desirable. The DEC VAX 11/780 has a 96 bit horizontally encoded control word and a feasibility study to determine if microcode could be produced for this machine from a HLL was initiated at The George Washington University where experimental compilers of this type (2) had previously been designed and implemented.

Two experimental compilers were available to support this feasibility study and the main effort was devoted to activating and evaluating each system to determine its applicability to the generation of microcode. It was determined that only one of the compilers was satisfactory in a selection process described below. This compiler was modified to produce microcode for the DEC PDP 11/45 as a substitute for the DEC VAX 11/780 machine since the control word field descriptions weren't available for the latter machine. The modified compiler has demonstrated an ability to generate microcode for a computer with a horizontally encoded control word as described below. A performance analysis of the compiler shows acceptable performance.

1.2 EXPERIMENTAL OBJECTIVES

The following objectives were established for this feasibility study at the outset:

(a) Modify the two available compilers described above so that they will produce microprograms for a computer with a horizontally encoded control word.

(b) Evaluate the performance of the compiler. Two criteria will be used:

1. The number of micro instructors needed to implement test cases.
2. The number of main storage references required.

These criteria will be compared between microcode produced from FORTRAN and DECPDP Assembly Language and microcode compiled by the quad compiler for the test cases.

1.3 ACCOMPLISHMENTS

(a) Both compilers which were to be used in the feasibility study were activated and test cases prepared. After some investigation it was decided to concentrate on the PLM to quadruple compiler due to its much greater flexibility at the HLL level.

(b) A post processor was developed which converts the quadruples generated by the quad compiler into special register oriented quadruples. The post processor is designed to be general purpose and capable of generating register quads for any machine architecture.

(c) A micro code generator for the DEC PDP 11/45 was implemented. It is again a generalized design capable of converting R quads into microcode for any host machine with a horizontal encoded control word.

(d) An evaluation was made relative to the criteria of the number of microinstructions and main storage references required to implement the test cases. The microinstructions and main storage references generated by the quad compiler are compared against the same criteria derived from hand coded DEC PDP 11 assembly language, and FORTRAN IV implementations of the test cases.

Section 2. COMPILER DESIGN AND DESCRIPTION

As noted above, two available compilers were considered for the generation of microprograms from a HLL. Both were implemented using a translator writer system (TWS) (6) which is based upon the XPL high level language. This system provides a methodology for producing a compiler for a high level language which can be defined in terms of a BNF grammar. It consists of a language analyzer, a prototype compiler (called skeleton), and compiler (called XCOM) which accepts a program written in the XPL high level language

and produces object code for the IBM 370 system.

The procedure for using the TWS is to define the high level language, for which a compiler is to be implemented, in a BNF format. This is entered into the language analyzer which produces a set of parsing tables for the specified language. These tables are entered into the prototype compiler and a set of routines written in XPL are inserted to carry out the compilation activity. These are: semantic routines to define the meaning of the individual phrases of the HLL input statements; code generators to convert these meanings into object code for the specified host machine; and symbol table and other data management routines. The resultant program is a compiler for the specified HLL written in the XPL language which is now compiled by the XCOM compiler into an IBM 370 object load module. This load module when loaded into a 370 computer will compile a program written in the specified HLL into object microcode for host machine. The operation of the TWS is illustrated in figure 1.

The two compilers accepted a variant of PL/1 as an input HLL. One designated the MOD 3 Compiler accepts a version of MPL (1) and produces microcode for the Interdata MOD 3 minicomputer. The second designated the Quad Compiler accepts PLM (7), a subset of PL/1 which is supported by INTEL as a HLL input to a compiler which produces object code for the 8080 series of microprocessors. This compiler produces quadruples, a form of intermediate text (8) which are translated into DEC PDP 11 machine code. An overview of the operation of these compilers is given in figure 2. A detailed program listing of the Quad compiler can be found at appendix 7.1, Table I provides a definition of the quads generated by the Quad Compiler.

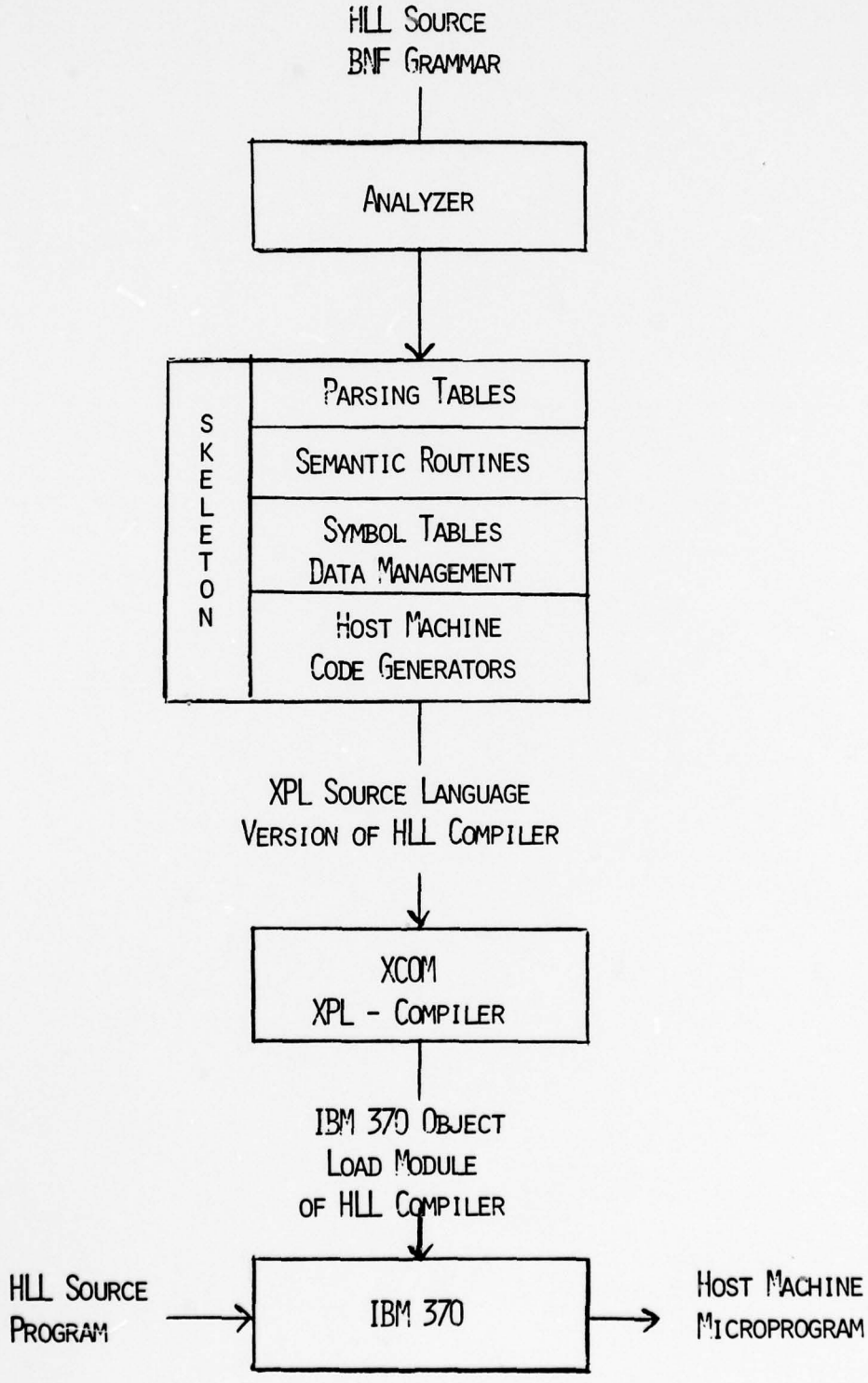


Figure 1. TWS Functional Flow

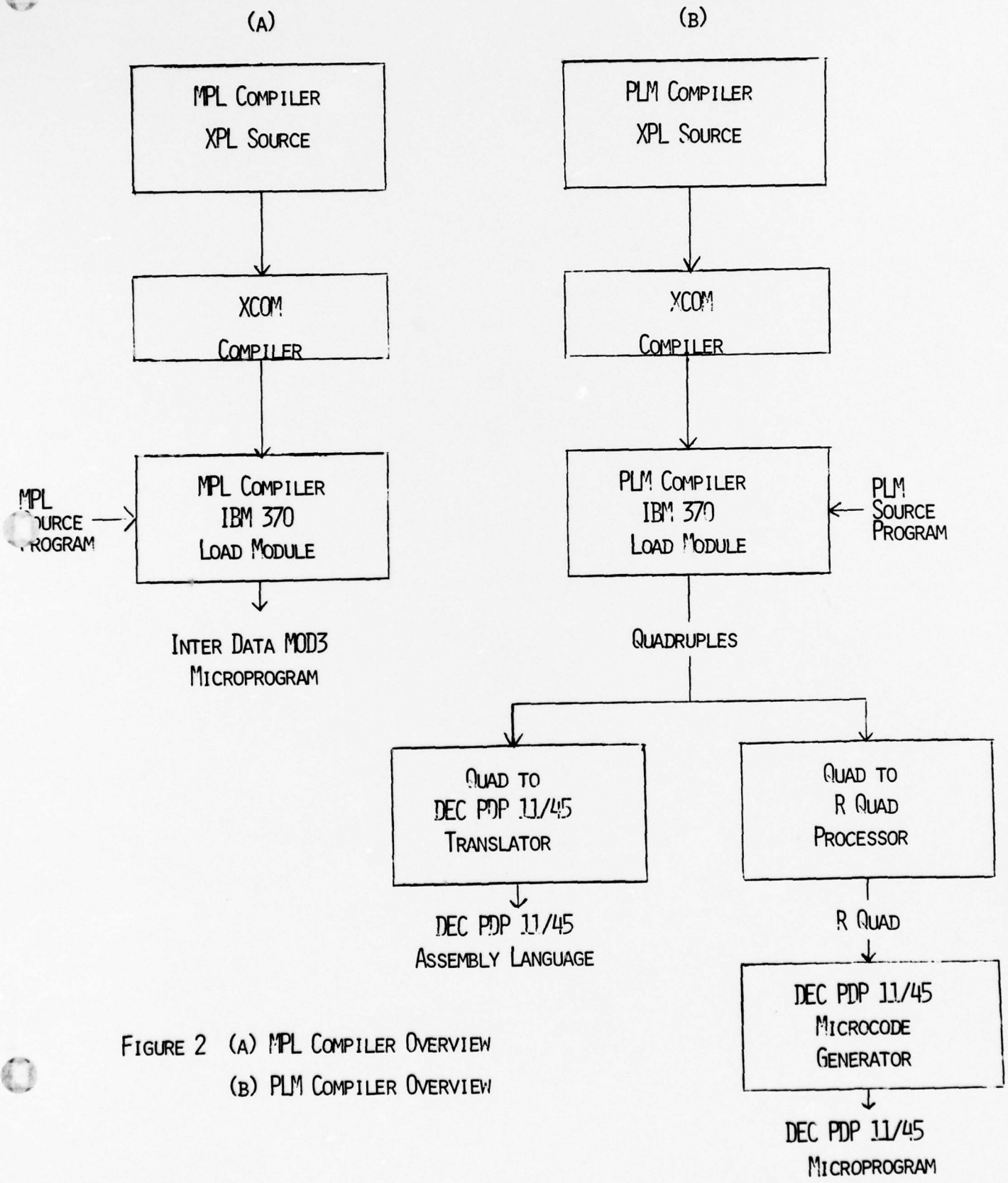


FIGURE 2 (A) MPL COMPILER OVERVIEW
(B) PLM COMPILER OVERVIEW

TABLE I

QUADRUPLE FORMATS GENERATED

BY

QUAD COMPILER

	Type	Operator	Operand 1	Operand 2	Result
1.	Arithmetic	+,-,*,GT,LT,EQ	Variable	Variable	Variable
2.	Branch (Cond)	BT,BF	Variable (Flag)	-	Addr (Label)
3.	Branch (Uncond)	BR	-	-	Addr (Label)
4.	Data	ASGN	Variable (ADDR)	-	Addr (Var)
5.	Label	LAB	Addr (Label)	-	
6.	Array	SUBS/SUBL	Address (Array)	Index	Addr (Var)

To illustrate the use of these compilers, a sample program is shown in terms of its input and output for each compiler in figure 3 and 4. Note that the input to the MOD3 compiler is written in terms of the internal machine registers and functional units even though the statements are written in a HLL format. This implies the user of this HLL must have an intimate knowledge of the Interdata MOD3 hardware. The input to the quad compiler is more general and appears much more like statements normally associated with a HLL. This generality does lead to a complexity which will be discussed below.

Section 3. SELECTION OF APPROACH

Both compilers were activated at the George Washington University Computer Center (An IBM 370 MOD 148 Installation). The MOD3 compiler was easily activated but the quad compiler which had been originally implemented on a different computer required a number of changes before becoming operational. During the implementation of the test programs shown in table II, it became apparent that the MOD3 compiler suffered from a serious lack of generality. As noted above, the HLL statements had to be written in terms of the internal functional components of the Interdata MOD3. Programs written in a machine independent form were rejected by this compiler. Based upon this lack of generality, it was decided not continue working with this compiler.

The quad compiler accepts as input programs a very natural formulation of the test case algorithms and produces an equivalent quadruple representation along with a DEC PDP 11 assembly language version which is derived from the quadruples. Some problems were encountered and solved for handling of arrays and nested DO loops and a "built in" multiply routine was implemented to permit a wider range of test programs.

```
GREATEST_ELEMENT:
        PROCEDURE OPTIONS (MAIN);
  /*  MPL PROCEDURE TO FIND THE GREATEST INTEGER IN A LIST */
        DCL (R0,R1,R2,R3,R4,R5,R6,R7,AR,DER) BIT (8);
  /*  R0 AND R1 CONTAIN POINTER TO LIST. R7 IS USED TO INDEX THE LIST */
INITIALIZE:
        R5 = 0;
        R4 = 0;
        R7 = 0;
        R6 = 49;    /* LIST IS ASSUMED TO BE 50 LONG. */
READ:               /* READ UP NEXT OPERAND IN LIST */
        MAL = R7+R1;        /* USE R1 AS BASE ADDRESS */
        MAH = R0 + CARRY;   /* DOUBLE PRECESION ADD */
        MDR = MS(MAR);      /* LOAD OPERAND  */
        IF R5 <= MDH THEN GOTO SWAP;
        IF R5 ¬= MDH THEN GOTO NEXT;          /* SHOULD BE ¬= */
        IF R4 > MDL THEN GOTO NEXT;
SWAP:       /* PLACE MDR IN R4 AND R5 */
        R4 = MDL;
        R5 = MDH;
NEXT:       /* GET NEXT ELEMENT AND CHECK IF DONE */
        R7 = R7 + 1;
        IF R7 < R6 THEN GOTO READ;
FINISH: GOTO GREATEST_ELEMENT;
        END;
   EOF EOF EOF
```

Fig 3 (a) MPL Source Program for Greatest Element Test Program

OBJECT LISTING FOR INTERDATA III

REF	LINE	MICRO LOCATION	OBJECT CODE	MNEMONICS
	8	0	5500	LI R5."00"
	9	1	5400	LI R4."00"
	10	2	5700	LI R7."00"
	11	3	5631	LI R6."31"
	12	4	0873	L AR.R7
	12	5	CC17	A MAL,R1
	13	6	CB0E	AA MAH,R0
	15	7	3100	C MEM,READ
	15	8	0853	LS AR,R5
	15	9	E807	LS AR,MDH
	16	10	1111	BR L1,17
	16	11	0853	LS AR,R5
	16	12	E807	LS AR,MDH
	17	13	1313	BR L1,19
	17	14	0843	LS AR,R4
	17	15	F847	LS AR,MDL
	18	16	1213	BR L1,19
	20	17	0443	L R4,MDL
	21	18	0503	L R5,MDH
	22	19	5801	LI AR,"01"
	22	20	C777	A R7,R7
	23	21	0873	L AR,R7
	23	22	F867	LS AR,R6
	24	23	1104	BR L1,4
	25	24	7000	BT R0,R0
	25	25	1100	BR L1,0

Fig 3 (b) INTERDATA MOD3 Microcode for Greatest Element Test Program

```
GREATEST_ELEMENT:PROCEDURE;
      DECLARE (I,N,GRTST_ELMNT,INDEX) BYTE;
      DECLARE ARRAY(10) BYTE;
      N=10;
      GRTST_ELMNT=ARRAY(1);
      INDEX=1;
      DO I=2 TO N;
            IF GRTST_ELMNT > ARRAY(I) THEN GO TO LAB1;
                  GRTST_ELMNT=ARRAY(I);
                  INDEX=I;
       LAB1:END;
      END;
EOF
```

Fig 4 (a) PLM Source Code for Greatest Element Test Program

QUADS GENERATED

OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT/ LABEL	****	RCD_NR	LOC_QUAD
LAB	GREATEST_ELEMENT	0	0		0	1
ASGN	10	0	N		0	5
SUBS	ARRAY	1	.T3		0	9
ASGN	.T3	0	GRTST_ELMNT		0	13
ASGN	1	0	INDEX		0	17
ASGN	2	0	I		0	21
LAB	.L7	0	0		0	25
GT	I	N	.T8		0	29
BT	.T8	0	.L9		0	33
SUBS	ARRAY	I	.T10		0	37
GT	GRTST_ELMNT	.T10	.T11		0	41
BT	.T11	0	LAB1		0	45
SUBS	ARRAY	I	.T13		0	49
ASGN	.T13	0	GRTST_ELMNT		0	53
ASGN	I	0	INDEX		0	57
LAB	LAB1	0	0		0	61
ADD	I	1	.T17		0	65
ASGN	.T17	0	I		0	69
BR	0	0	.L7		0	73
LAB	.L9	0	0		0	77

Fig 4 (b) Quadruple Representation of Greatest Element Test Program

```
                .ENTRY  MAIN            ;
                .WORD    0              ;       I
                .WORD    0              ;       N
                .WORD    0              ;       GRTST_ELMNT
                .WORD    0              ;       INDEX
                .WORD   0               ;         ARRAY(1)
                .WORD   0               ;         ARRAY(2)
                .WORD   0               ;         ARRAY(3)
                .WORD   0               ;         ARRAY(4)
                .WORD   0               ;         ARRAY(5)
                .WORD   0               ;         ARRAY(6)
                .WORD   0               ;         ARRAY(7)
                .WORD   0               ;         ARRAY(8)
                .WORD   0               ;         ARRAY(9)
                .WORD   0               ;         ARRAY(10)
                CLR     R4              ;       MULTIPLY
                TST     R1
                BGE     .+6.
                INC     R4
                NEG     R1
                TST     R2
                BGE     .+6.
                INC     R4
                NEG     R2
                CLR     R0
                MOV     #-16.,R3
                ASL     R0
                ROL     R1
                BCC     .+6.
                ADD     R2,R0
                ADC     R1
                INC     R3
                BNE     .-12.
                TST     R1
                BNE     .+10.
                DEC     R4
                BNE     .+4
                NEG     R0
                RTS     R7
                MOV     #2.,R6
                HALT
                CLR     R0              ;       DIVIDE
                CMP     R2,R1
                BGT     .+8.
                SUB     R2,R1
                INC     R0
                BR      .-8.
                RTS     R7
MAIN:           MOV     #4096.,R6       ;
                .WORD    6              ;
                MOV     LABELS,R5       ;
                MOV     #10.,2.         ;
                MOV     #1.,-(R6)       ;
                ASL     (R6)            ;
                ADD     #6.,(R6)        ;
                MOV     @(R6)+,4.       ;
                MOV     #1.,6.          ;
                MOV     #2.,0.          ;
                CMP     0.,2.           ;
                BLE     .+6             ;
                JMP     @6.(R5)         ;
                MOV     0.,-(R6)        ;
                ASL     (R6)            ;
                ADD     #6.,(R6)        ;
                CMP     4.,@(R6)+       ;
                BLE     .+6             ;
                JMP     @4.(R5)         ;
                MOV     0.,-(R6)        ;
                ASL     (R6)            ;
                ADD     #6.,(R6)        ;
                MOV     @(R6)+,4.       ;
                MOV     0.,6.           ;
                MOV     0.,-(R6)        ;
                ADD     #1.,(R6)        ;
                MOV     (R6)+,0.        ;
                JMP     @2.(R5)         ;
                HALT                    ;
LABELS:         .WORD   .+2             ;
                .WORD   108.            ;       GREATEST_ELEMENT
                .WORD   140.            ;       .L7
                .WORD   192.            ;       LAB1
                .WORD   208.            ;       .L9
END:            .END    MAIN            ;
SEND
```

Fig 4 (c) DEC PDP 11 Assembly Language Representation of Greatest Element Test Program

```
                .ENTRY  MAIN            ;
                .WORD   0               ;       I
                .WORD   0               ;       N
                .WORD   0               ;       GRTST_ELMNT
                .WORD   0               ;       INDEX
                .WORD   0               ;         ARRAY(1)
                .WORD   0               ;         ARRAY(2)
                .WORD   0               ;         ARRAY(3)
                .WORD   0               ;         ARRAY(4)
                .WORD   0               ;         ARRAY(5)
                .WORD   0               ;         ARRAY(6)
                .WORD   0               ;         ARRAY(7)
                .WORD   0               ;         ARRAY(8)
                .WORD   0               ;         ARRAY(9)
                .WORD   0               ;         ARRAY(10)
                CLR     R4              ;       MULTIPLY
                TST     R1
                BGE     .+6.
                INC     R4
                NEG     R1
                TST     R2
                BGE     .+6.
                INC     R4
                NEG     R2
                CLR     R0
                MOV     #-16.,R3
                ASL     R0
                ROL     R1
                BCC     .+6.
                ADD     R2,R0
                ADC     R1
                INC     R3
                BNE     .-12.
                TST     R1
                BNE     .+10.
                DEC     R4
                BNE     .+4
                NEG     R0
                RTS     R7
                MOV     #2.,R6
                HALT
                CLR     R0              ;       DIVIDE
                CMP     R2,R1
                BGT     .+8.
                SUB     R2,R1
                INC     R0
                BR      .-8.
                RTS     R7
MAIN:           MOV     #4096.,R6       ;
                .WORD   6               ;
                MOV     LABELS,R5       ;
                MOV     #10.,2.         ;
                MOV     #1.,-(R6)       ;
                ASL     (R6)            ;
                ADD     #6.,(R6)        ;
                MOV     @(R6)+,4.       ;
                MOV     #1.,6.          ;
                MOV     #2.,0.          ;
                CMP     0.,2.           ;
                BLE     .+6             ;
                JMP     @6.(R5)         ;
                MOV     0.,-(R6)        ;
                ASL     (R6)            ;
                ADD     #6.,(R6)        ;
                CMP     4.,@(R6)+       ;
                BLE     .+6             ;
                JMP     @4.(R5)         ;
                MOV     0.,-(R6)        ;
                ASL     (R6)            ;
                ADD     #6.,(R6)        ;
                MOV     @(R6)+,4.       ;
                MOV     0.,6.           ;
                MOV     0.,-(R6)        ;
                ADD     #1.,(R6)        ;
                MOV     (R6)+,0.        ;
                JMP     @2.(R5)         ;
                HALT                    ;
LABELS:         .WORD   .+2             ;       GREATEST_ELEMENT
                .WORD   108.            ;       .L7
                .WORD   140.            ;       LAB1
                .WORD   192.            ;       .L9
                .WORD   208.            ;
END:            .END    MAIN            ;
$END
```

Fig 4 (c) DEC PDP 11 Assembly Language Representation of Greatest Element Test Program

TABLE II

LIST OF TEST PROGRAMS

USED TO EVALUATE THE

QUAD COMPILER

1. Fibonacci Series: Evaluates successive terms of this series.
2. Greatest Element: Finds the largest number in a list.
3. Bubble Sort: Sorts a list of numbers into ascending order.
4. Prime: Uses Sieve of Erasthenes to identify the prime numbers between 1 and N.
5. Filter: A digital low pass filter algorithm.

The next task was to select the computer to serve as the host machine for the microcode to be generated by the quad compiler. The study contract specified the DEC VAX 11/780 system to be the host machine but we were unable to get an adequate hardware description of this system. As a result we were required to look for an alternate system. Our choice was narrowed down to a DEC machine with a horizontal control word format as similar as possible to the VAX machine (96 bit horizontal control word). Due to the availability of adequate hardware information for the DEC PDP 11/45 system, which has a 56 bit horizontal control word format with 18 control fields, this system was chosen to be the host machine.

Another reason for choice of a DEC PDP 11 computer model was the availability of DEC PDP 11 assembly language output from the quad computer. This factor facilitated the compiler evaluation activity.

A number of difficulties immediately surfaced. The DEC PDP 11/45 has no means for entering literal data at the microprogram level. This means that all addresses, indices, constants, etc. specified for program written in PLM had to be stored in main memory in the DEC PDP 11/45. There are only six registers available to the programmer in this machine and when the number of indices, constants, and other currently processed variables exceeds six then as a backup they must be resident in main storage. At any particular point in the execution of the test algorithm only currently referenced data items are stored in the internal registers. These limitations lead to the generalized approach to handling internal register contents to be described below.

An interesting side affect of the absence of a means for introducing literal data at the microprogram level was the discovery that it was not feasible to directly microcode algorithms for the DEC PDP 11/45 when the

number of constants and indices exceeded the available internal register supply. For the test cases utilized in the study, all but one (the generation of Fibonacci series values) was not directly microprogrammable in the DEC PDP 11/45. This had the undesirable side effect of eliminating one boundary of direct or hand produced microcode as a yard stick against which to measure the efficiency of the quad compiler for producing DEC PDP 11/45 microcode.

Section 4. MICROCODE GENERATION

4.1 R QUAD GENERATOR

Before describing the R Quad Generator it will be necessary to briefly describe the DEC PDP 11/45 which was selected as the host machine for the microcode to be compiled from the PLM HLL. This is a typical minicomputer architecture which was designed to efficiently interpret the DEC PDP 11 instruction set (9). An overview block diagram of this system is shown in figure 5.

As noted above, the control word for the DEC PDP 11/45 contains 56 bits and has 18 control fields. A list of these control fields and their purpose is shown in table III. Note that there are 16 registers controlled through the microprogram of which six are available for general use. Memory read access is via a BA register which transmits an address over the DEC UNIBUS to memory which returns a value one access time later over the same bus. Memory write is similar except a data value and address are transmitted to memory via the UNIBUS.

Because of the limited number of free registers in the DEC PDP 11/45 CPU, it was decided to maintain all program constants, addresses, indices and variables defined in the quadruples output by the compiler in main storage and only

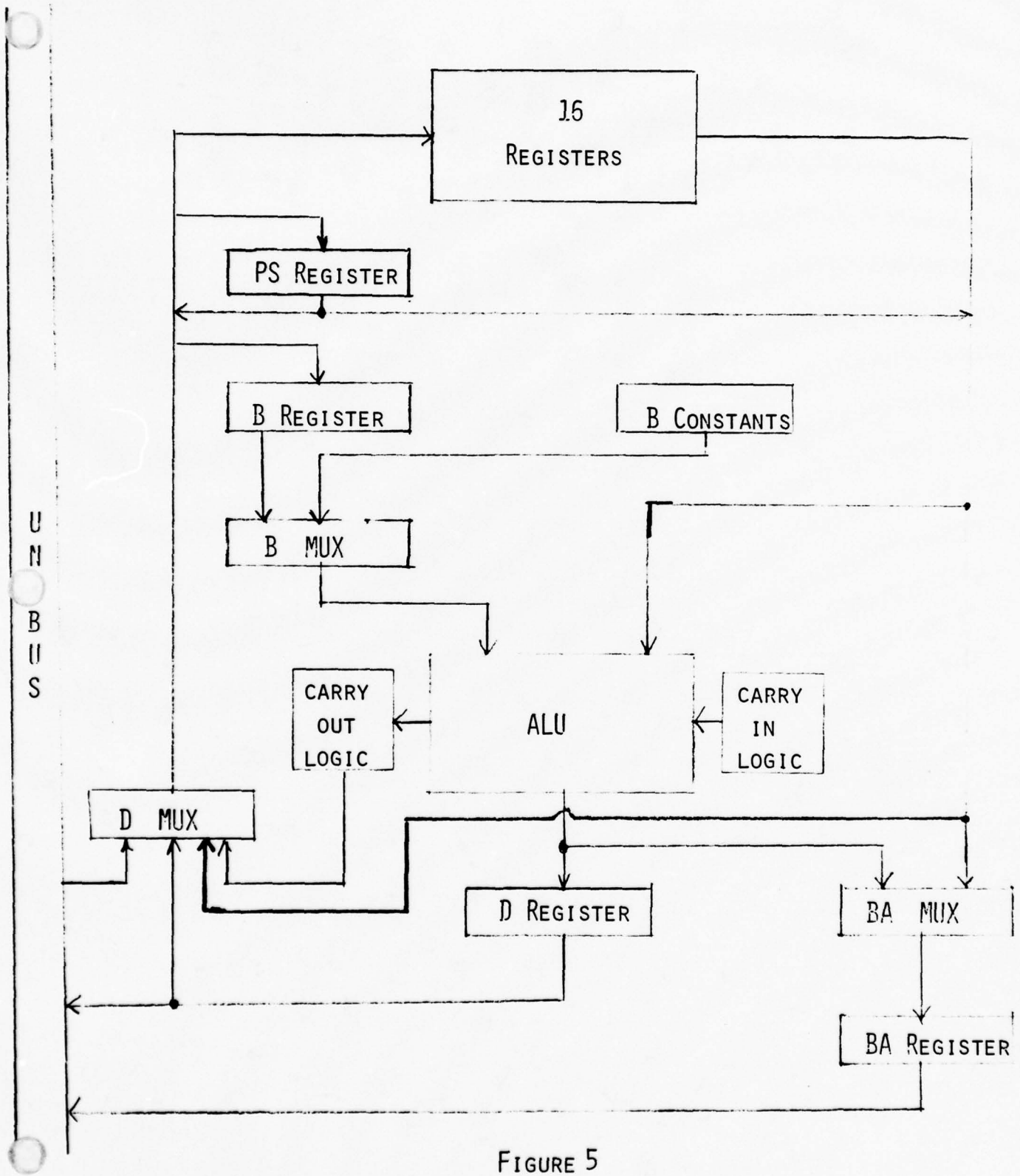


FIGURE 5

SIMPLIFIED DEC PDP 11/45 CPU BLOCK DIAGRAM

TABLE 111

DEC PDP 11/45
MICROCODE CONTROL FIELDS

1.	CLK	(3 bits)	-	Clock Control Field
2.	CIR	(1 bit)	-	Clock UNIBUS into IR Register
3.	WR	(2 bits)	-	Controls Write DMUX input to the General Registers
4.	CB	(1 bit)	-	Clock DMUX into B Register
5.	CD	(1 bit)	-	Clocks ALU Output into D Register
6.	CBA	(1 bit)	-	Clocks Data into BUS Register
7.	BUS	(3 bits)	-	BUS Control
8.	DAD	(4 bits)	-	Data Path Alteration Control
9.	SPS	(3 bits)	-	Controls Loading and Clocking of PSW Register
10.	ALU	(5 bits)	-	ALU Mode and Operation Select
11.	SBC	(4 bits)	-	Microprogram Constant Selection
12.	SBM	(4 bits)	-	BMUX Input Control
13.	SDM	(2 bits)	-	DMUX Input Control
14.	SBA	(1 bit)	-	Selects Input to BA MUX
15.	UBF	(5 bits)	-	Micro Branch Field
16.	SRX	(4 bits)	-	General Register Address Source Selection
17.	RIF	(4 bits)	-	General Register Address
18.	UPF	(8 bits)	-	Next microinstruction address

keep a limited number in registers as required at each point in the calculation. This requires keeping track of where all variables, etc. are stored in memory and which are in the registers. This is accomplished at compile time by generating a register content table which provides the required address pointers, usage, status, and register content of program data in current use. It is important to note that generation of this table doesn't impact run time execution of the microprogram but only the compilation activity.

An overview of how R quads are generated is shown at figure 6. We start with a set of quadruples generated by the compiler in which all operands are described in terms of the variables specified by the programmer of the algorithm in PLM and those variables, generated by the compiler, e.g. the indices generated to implement a DO loop or array structure. As each quadruple is read in by the R Quad generator it examines each operator and operand. The operands representing constants, indices and variables are assigned memory locations. A set of allocate and deallocate routines load and unload the program data into and out of the internal registers using read and write quadruples defined for this purpose. Operands with the same value, e.g. constants, are assigned to the same register. Variables assigned an initial value, which is already stored in a register, will be assigned to the same register. When the supply of internal registers is exhausted, an algorithm selects operands currently stored in the registers to be loaded into storage to make room for currently needed variables. As noted above, a record of all these operand assignments is being maintained during the R Quad generation process.

Two special cases must now be examined. The first is the quadruple representing a label. This quadruple is generated by the compiler in implementing a DO loop and other program structures which must serve as a point to be branched to from other parts of the microprogram. Of course, statement labels can be inserted in the HLL source as required. The second case involves

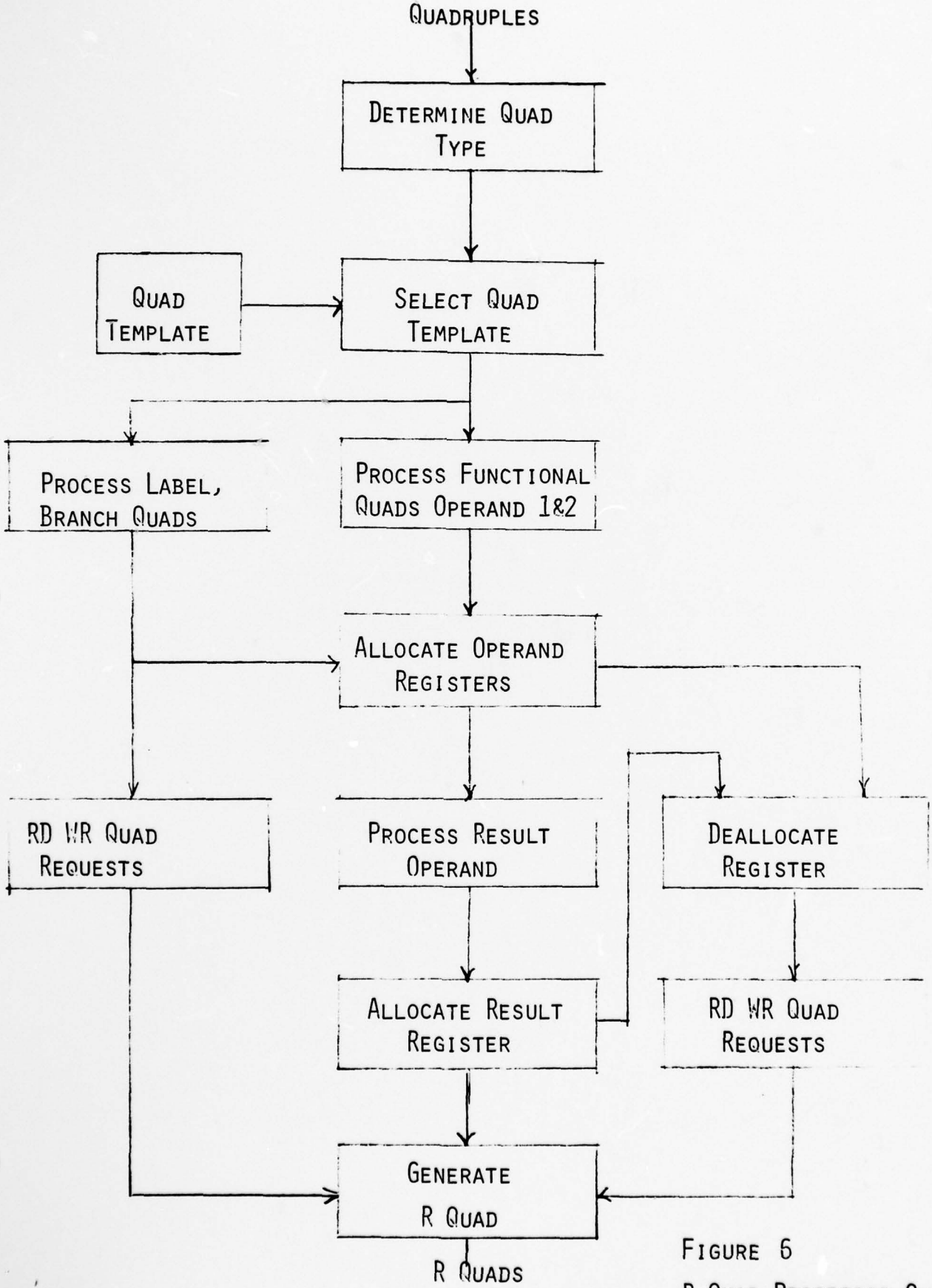


FIGURE 6
R QUAD PROCESSOR OVERVIEW

quadruples which specify branches. These may be either compiler generated or be part of the HLL source statements. In both of these cases the contents of the internal CPU registers are subject to change and the existing contents must be saved. In the case of a label, a branch may be made to this quadruple from some other quadruple in the microprogram. The register contents specified by the quadruples before the label quadruple will need to be replaced if a branch occurs to this label. In the event the program returns to this point again the old register contents must be saved. Likewise in the case of a branch quadruple it isn't known beforehand which way the branch will go. Because of the possibility of not taking the branch, the current register values are saved.

The R Quad processor assigns to registers all operands in quadruples generated by the compiler and generates the appropriate additional quadruples to move operand data between the registers and memory as required to meet this objective. The result is a set of quadruples with most operands specified in terms of internal registers. This is illustrated in figure 7, which shows the R Quads derived from the Quads representing the Greatest Element Test Program shown in Figure 4(c). A list of R Quad types is shown in table IV. The layout of the register content table required to support its operation is shown in figure 8. A detailed description of the function of each procedure of the R Quad Processor is given in appendix 7.5(a) and the R Quad generators in appendix 7.5(b).

4.2 MICROCODE GENERATOR

Each quadruple type has an operation field which designates some action to be taken by the hardware. The operands required for this action are specified and a location to store the result is given. The microcode generation routine

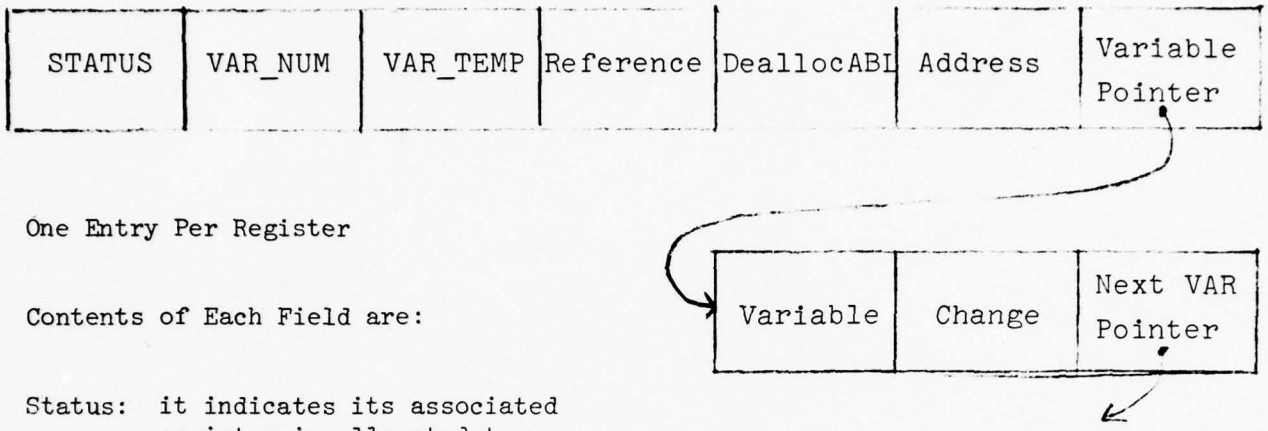
OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT LABEL
LAB	GREATEST_ELEMENT	0	0
RD	10	0	R6
RDAD	ARRAY	0	R5
RD	1	0	R4
ASL	R4	0	R1
ADD	R5	R1	R1
RDVR	R1	0	R1
RD	2	0	R3
WT	R1	0	GRTST_ELMNT
WT	R3	0	I
WT	R4	0	INDEX
WT	R6	0	N
LAB	.L7	0	0
RD	I	0	R6
RD	N	0	R5
GT	R6	R5	R1
BT	R1	0	.L9
RDAD	ARRAY	0	R4
ASL	R6	0	R1
ADD	R4	R1	R1
RDVR	R1	0	R1
RD	GRTST_ELMNT	0	R3
GT	R3	R1	R2
BT	R2	0	LAB1
ASL	R6	0	R1
ADD	R4	R1	R1
RDVR	R1	0	R1
WT	R1	0	GRTST_ELMNT
WT	R6	0	INDEX
LAB	LAB1	0	0
RD	I	0	R6
RD	1	0	R5
ADD	R6	R5	R1
WT	R1	0	I
BR	0	0	.L7
LAB	.L9	0	0

Figure 7 R Quad Representation of Greatest Element Test Program derived from Quads Shown in Figure 4(c)

TABLE IV

REGISTER QUADS WITH OPERANDS REPRESENTING INTERNAL REGISTERS OR ADDRESSES

OPERATION	OPERAND 1	OPERAND 2	RESULT	ACTION
RD	Addr	-	Reg n	Read Content of Addr into Register n
WT	Reg n	-	Addr	Write Register n into addr.
RDAD	Var	-	Reg n	Read Addr of Varaible into Register n
RDVR	Reg n	-	Reg m	Read value in Reg m from address specified in Reg n
WTAD	Reg n	-	Reg m	Write contents of Reg n into addr in Reg m
ASL	Reg n	-	Reg m	Write contents of Reg n into Reg m shifted left one place



One Entry Per Register

Contents of Each Field are:

Status: it indicates its associated register is allocated to a variable or not, it is a two valued entry 'FREE' or 'ALLOCATED'

VAR_NUM: it counts number of variables assigned to the register

VAR_TEMP: it counts number of temporary variables assigned to the register

Reference: its contents is a number to indicate the last reference to the register relative to the reference to the other registers.

Deallocabl: it indicates if the register can be used in the quad under process or not, its value is 'YES' or 'NO'

Address: it is a two valued element to indicate if the contents of the register is value of or address of a variable, it can take value of 'YES' or 'NO'

Var_Pointer: a pointer to list of variables assigned to the register

Variable: variable name assigned to the register

Change: this entry indicates if the variable has changed its value after the last time that it has been read from the memory

Next_Var_PTR: it is a pointer to the next variable associated to the same register

FIGURE 8

R QUAD PROCESSOR REGISTER CONTENT TABLE FORMAT

consists of a case statement which causes a branch to a routine for each quadruple type. This routine refers to a tabulation of the actions carried out by each control field of the DEC PDP 11/45 and selects the appropriate binary value to actuate the required function. More than one microinstruction may be generated for a given quadruple. In figure 9 the microprogram corresponding to the R Quads shown in figure 7 is shown.

To actually use the microprogram in the DEC PDP 11/45 it would be necessary to created a load module. This would require creation of a control storage address table which provides relative address values for all operands specifying addresses. This process would be similar to that required for a conventional loader routine used to create load modules for machine language programs.

Section 5.0 PERFORMANCE EVALUATION

5.1 EVALUATION APPROACH

One of the questions concerning use of a compiler to generate microprograms is the ability to produce "efficient" microcode. The meaning of the word "efficient" is somewhat arbitrary. To develop a precise meaning is cumbersome, but in general it means the generation of microgram which run in a minimum time. As noted above, two measures which have been adopted for expressing "efficiency" for this study, are the number of microinstructions generated to represent the algorithm and the corresponding number of main memory references. Since we were unable to execute the microprograms generated in this study, the more meaningful criteria of execution time wasn't obtainable.

As noted above, a comparison was made of alternate ways of generating DEC PDP 11/45 microcode. One was to produce DEC PDP 11/45 microcode directly using the Quad Compiler. Another way was to take the DEC PDP 11/45 assembly language statements produced by the Quad Compiler and convert these into

GENERATED MICRO WORDS

LOC	CLK	CIR	WR	CB	CD	CBA	BUS	DAD	SPS	ALU	SRC	SBM	SDM	SBA	UBF	SRX	RIF	UPF
000	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00000001
001	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00000010
002	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000011
003	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00000100
004	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00000101
005	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00000110
006	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000111
007	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00001000
010	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00001001
011	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001010
012	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00001011
013	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00001100
014	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0100	00001101
015	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00001110
016	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	00001111
017	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0101	00010000
020	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00010001
021	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0001	00010010
022	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0001	00010011
023	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00010100
024	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00010101
025	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010110
026	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	00010111
027	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00011000
030	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011001
031	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00011010
032	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	00011011
033	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00011100
034	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011101
035	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00011110
036	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0011	00011111
037	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00100000
040	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00100001
041	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00100010
042	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00100011
043	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00100100
044	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00100101
045	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00100110
046	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0110	00100111
047	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00101000
050	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00101001
051	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00101010
052	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00101011
053	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00101100
054	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00101101
055	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00101110
056	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00101111
057	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00110000
060	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00110001
061	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	00110010
062	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0110	00110011
063	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00110100
064	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	00110101
065	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00110110
066	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00111000
067	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01101100
070	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00111001
071	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00111010
072	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0110	00111011
073	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00111100
074	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	00111101
075	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0100	00111110
076	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00111111
077	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0001	01000000
100	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0001	01000001
101	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01000010
102	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01000011
103	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	01000100
104	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	01000101
105	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	01000110
106	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0011	01000111
107	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0010	01001000
110	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	01001001
111	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01001010
112	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01001100

FIGURE 9 DEC PDP 11/45 Microcode Generated by PLM to Quad Compiler for Greatest Elements Test Program

```
  3 010 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 01011011
  4 100 0 00 0 1 0 000 0000 000 01100 0000 0000 00 0 00000 0001 0110 01001101
115 010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01001110
116 010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0001 01001111
117 100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0100 01010000
120 010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01010001
121 011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0001 01010010
122 010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0001 01010011
123 111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01010100
124 010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01010101
125 010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01010110
126 101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 01010111
127 111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01011000
130 010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01011001
131 010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01011010
132 101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0110 01011011
133 011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 01011100
134 010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 01011101
135 111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01011110
136 010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01011111
137 011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01100000
140 010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0110 01100001
141 111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01100010
142 010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01100011
143 011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 1000 01100100
144 010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0101 01100101
145 010 0 00 1 0 0 000 0000 000 00000 0000 0000 00 0 00000 0001 0101 01100110
146 100 0 00 0 1 0 000 0000 000 01001 0000 0000 00 0 00000 0001 0110 01100111
147 010 0 11 0 0 0 000 0000 000 00000 0000 0000 10 0 00000 0001 0001 01101000
150 111 0 11 0 1 1 001 0000 000 01001 0010 1111 10 1 00000 0001 0000 01101001
151 010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 1000 01101010
152 010 0 00 0 0 1 000 0000 000 00000 0000 0000 00 1 00000 0001 1000 01101011
153 101 0 00 0 1 0 101 0000 000 00000 0000 0000 00 0 00000 0001 0001 00100111
154 011 0 00 0 0 1 001 0000 000 00000 0000 0000 00 1 00000 0001 0000 01101101
1   010 0 11 0 0 0 000 0000 000 00000 0000 0000 01 0 00000 0001 0000 01101110
```

FIGURE 9 Continued

equivalent microcode. A third way was to hand-code a DEC PDP 11/45 assembly language version and then convert that into equivalent microcode. The final way is to use a DEC supplied FORTRAN Compiler which produces DEC PDP 11 assembly language statments which are then converted into microcode. Figure 10 illustrates these alternate paths for generating microcode and the comparative results are shown in Table V.

As noted above, only the Fibonacci Series test problem could be directly hand coded into microcode. The number of microinstructions for this case was 11 compared to 83 generated by the Quad Compiler. It should be noted that for this test case all indices and variables could be left in internal DEC PDP 11/45 registers and no memory references were required. This shows that hand-coded microcode can be very efficient but of very limited application.

5.2 QUAD COMPILER PERFORMANCE

The data shown in Table V indicates that for the very simple test cases (Fibonacci Series, Greatest-Element) the hand-coded DEC PDP 11 assembly langauge implementation is more efficient than the quad compiler implementation. For the more complex test cases (digital filter and Prime Number Generator) the quad compiler is more efficient. Another interesting comparison is to the DEC PDP 11 machine language statments produced by the quad compiler from the quad representation. It has been well established (10) that assembly language implementations produced by compilers aren't as efficient as hand coded assembly language, which is indicated by the results shown. Going directly from quads to microcode appears to be about 4 times more efficient than going from quads to DEC PDP 11 machine language and then to microcode for the more complicated test cases. The DEC PDP 11 assembly language to microcode alternative produces from 1.5 to 2 times as many microinstructions as the direct PLM to microcode via

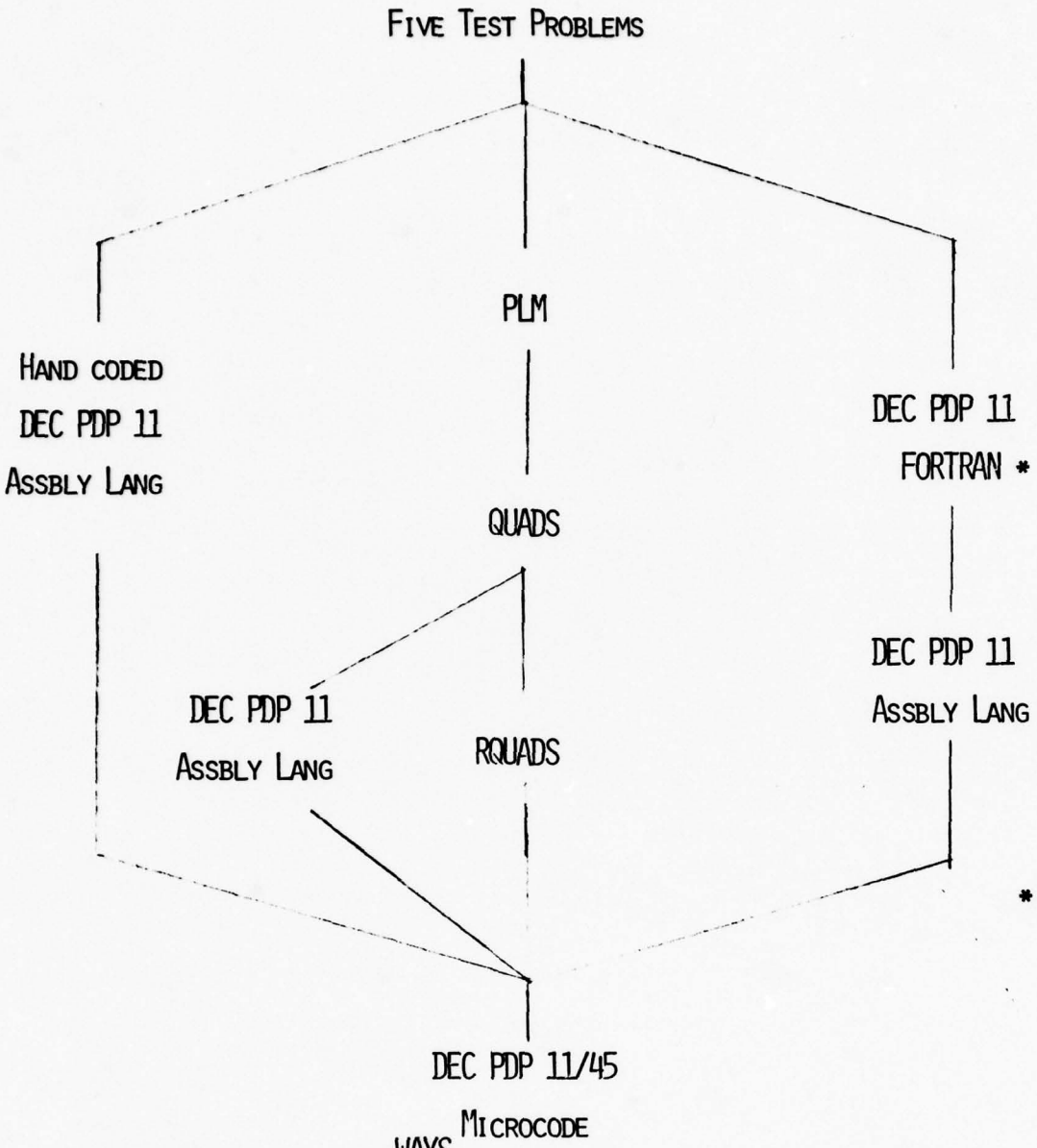


FIGURE 10 ALTERNATE WAYS TO GENERATE DEC PDP 11/45 MICROCODE

* SEE NOTE ON PG. 29

TABLE V

COMPARISON OF ALTERNATE METHODS FOR GENERATION OF MICROCODE FOR THE DEC PDP 11/45

	Compiler Generated Quads	Compiler Generated R Quads			Compiler Generated Assembly Language			Hand Coded Assembly Language		
	Number	Number	Micro Instruction	Memory Reference	Number	Micro Instruction	Memory Reference	Number	Micro Instruction	Memory Reference
Fibonacci Series	16	24	83	32	17	163	54	12	50	22
Greatest Element	20	36	109	35	26	231	91	14	77	29
Sort	31	56	165	50	45	399	171	21	141	55
Prime Number	35	57	182	50	44	473 *	181	26	235	72
Digital Filter	28	53	161	38	51	574	214	24	276	74

* See note Pg. 29

compiler method. In general the PLM to microcode compiler appears to be comparable with hand coded assembly language in spite of the necessity to move register data in and out of main storage for label and branch quadruples. To date no attempt has been made to optimize the PLM to quad compiler or the R Quad generation process. It is our intention to pursue these alternatives as ongoing research.

As indicated by the asterisks on figure 10 and table V, we had intended to compare the performance of the PLM to DEC PDP 11/45 compiler with a standard commercially available FORTRAN compiler supplied for the DEC PDP 11 systems. FORTRAN versions of the five test programs have been prepared but we are unable to get a machine language listing (object code) with any of the compilers we have access to. Accordingly, the corresponding efficiency comparison to the DEC FORTRAN compiler isn't shown in table V. We anticipate obtaining this data in the near future and it will be supplied as an addendum to this report.

SECTION 6.0 RECOMMENDATIONS FOR FURTHER RESEARCH

6.1 Conversion of Input Language of Quad Compiler to PASCAL

PASCAL has been widely accepted as a high level language used for scientific research. The U.S. Army BMD DDP Test Bed has selected PASCAL as its standard language. Accordingly, it is desirable to have this language as an input to a compiler producing microcode for the DEC VAX 11/780. PASCAL has a considerably more complex syntax structure than PLM and some problems will have to be resolved to convert the present quad compiler to accept PASCAL as an input language. It is very likely that due to limitations in the XPL

TWS, we may not be able to implement the entire PASCAL language but only a subset. The selection of this subset is a research issue. A proposal to carry out this research has been accepted and is about to begin.

6.2 Optimization of Quad Compiler

There has been no attempt made to optimize the performance of the quad compiler. There are two levels of optimization that can be attempted. The first would be to optimize the generation of quadruples by the complier and the second would be to optimize the generation of R Quads. It is felt that taking advantage of both of these optimization prospects should greatly enhance the performance of the PASCAL to Quad to DEC VAX 11/780 compiler. Since it is necessary to get the compiler converted to the PASCAL input language producing microcode for the DEC VAX 11/780, research into introducing optimization into this compiler must be deferred. A proposal indicating several approaches to quad compiler optimization will be generated in the near future.

6.3 Microprogram Host Machine Interface Study

In developing a compiler to produce microcode for a host machine with a horizontally encoded control word, it quickly became apparent that the host machine architecture had a profound impact on the ease with which the compiling process could be carried out. As noted above, the DEC PDP 11/45 has serious limitations for hand coded microprograms and it became apparent that there was a preferred host machine architecture if an intermediate program representation was going to be used, i.e., quadruple or triple format. Tuning of the architecture to optimize performance of compiled microcode via an intermediate representation could also lead to a less complex compiler internal structure. A study of these tradeoffs among host machine architecture, intermediate program representation, and the complexity of the microcode compiler would have great research value. As more understanding of the host machine architecture

and microcode interface is gained as a result of designing the PASCAL to DEC VAX 11/780 microcode compiler, a detailed research study proposal will be prepared to explore the issues noted above in more detail.

REFERENCES

1. Eckhouse, P.H., "A High Level Microprogramming Language (MPL)," PH.D. Dissertation, State University of New York at Buffalo, 1971.

2. Fodor, P.R., "Evaluation of Compiler Design and Performance for a Virtual Microprogrammed Machine," M.S. Dissertation, The George Washington University, 1976.

3. Abd-Alla, A.M. and Karlgaard, D.C., "Heuristic Synthesis of Microprogrammed Computer Architecture," IEEE TC, Vol. C-23, pp. 802-808, August, 1974.

4. Moffett, L.H., "On-line Virtual Architecture Timing Using Micro-capture," Ph.D. Dissertation, The George Washington University, 1976.

5. Evans, R.H., "Architecture Considerations in Signal Sorter Design," Ph.D. Dissertation, The George Washington University, 1976.

6. McKeeman, W.M., Horning, J.J. and Wortman, D.B., A Compiler Generator, Prentice-Hall, 1970.

7. Kildall, G.A., "High Level Language Simplifies Microcomputer Programming," Electronics, June, 1974.

8. Gries, David, Compiler Construction for Digital Computers, John Wiley 1971.

9. Digital Equip. Corp., "Processor Handbook PDP 11," Digital Equip. Corp. 1975.

10. Henriksen, J.O., Merwin, R.E., "Programming Language Efficiency in Real-Time Software Systems," AFIPS Proc. SJCC, 40 (1972) pp. 155-161.

PLM TO DEC PDP 11/45 MICROCODE COMPILER LISTING

```
/*   SKELETON
                    THE PROTO-COMPILER OF THE XPL SYSTEM

W. M. MCKEEMAN          J. J. HORNING           D. B. WORTMAN

INFORMATION &           COMPUTER SCIENCE        COMPUTER SCIENCE
COMPUTER SCIENCE,       DEPARTMENT,             DEPARTMENT,

UNIVERSITY OF           STANFORD                STANFORD
CALIFORNIA AT           UNIVERSITY,             UNIVERSITY,

SANTA CRUZ,             STANFORD,               STANFORD,
CALIFORNIA              CALIFORNIA              CALIFORNIA
95060                   94305                   94305

DEVELOPED AT THE STANFORD COMPUTATION CENTER, CAMPUS FACILITY,  1966-69
AND THE UNIVERSITY OF CALIFORNIA COMPUTATION CENTER, SANTA CRUZ, 1969-69.

DISTRIBUTED THROUGH THE SHARE ORGANIZATION.
THIS VERSION OF SKELETON IS A SYNTAX CHECKER FOR THE FOLLOWING GRAMMAR:
 /*  <PROGRAM> ::= <STMT LIST> EOF     */
 /*  <STMT LIST> ::= <STMT>     */
 /*  <STMT LIST> ::= <STMT LIST> <STMT>     */
 /*  <STMT> ::= <BASIC STMT>     */
 /*  <STMT> ::= <IF STMT>     */
 /*  <BASIC STMT> ::= <ASSIGNMENT> ;     */
 /*  <BASIC STMT> ::= <GROUP> ;     */
 /*  <BASIC STMT> ::= <PROC DEF> ;     */
 /*  <BASIC STMT> ::= <RETURN STMT> ;     */
 /*  <BASIC STMT> ::= <CALL STMT> ;     */
 /*  <BASIC STMT> ::= <GO TO STMT> ;     */
 /*  <BASIC STMT> ::= <DECLARATION STMT> ;     */
 /*  <BASIC STMT> ::= ;     */
 /*  <BASIC STMT> ::= <LABEL DEF> <BASIC STMT>     */
 /*  <IF STMT> ::= <IF CLAUSE> <STMT>     */
 /*  <IF STMT> ::= <IF CLAUSE> <TRUE PART> <STMT>     */
 /*  <IF STMT> ::= <LABEL DEF> <IF STMT>     */
 /*  <IF CLAUSE> ::= IF <EXPR> THEN     */
 /*  <TRUE PART> ::= <BASIC STMT> ELSE     */
 /*  <GROUP> ::= <GROUP HEAD> <ENDING>     */
 /*  <GROUP HEAD> ::= DO ;     */
 /*  <GROUP HEAD> ::= DO <STEP DEF> ;     */
 /*  <GROUP HEAD> ::= DO <WHILE CLAUSE> ;     */
 /*  <GROUP HEAD> ::= DO <CASE SELECTOR> ;     */
 /*  <GROUP HEAD> ::= <GROUP HEAD> <STMT>     */
 /*  <STEP DEF> ::= <VAR> <REPLACE> <EXPR> <ITERATION CONTROL>     */
 /*  <ITERATION CONTROL> ::= TO <EXPR>     */
 /*  <ITERATION CONTROL> ::= TO <EXPR> BY <EXPR>     */
 /*  <WHILE CLAUSE> ::= WHILE <EXPR>     */
 /*  <CASE SELECTOR> ::= CASE <EXPR>     */
 /*  <PROC DEF> ::= <PROC HEAD> <STMT LIST> <ENDING>     */
 /*  <PROC HEAD> ::= <PROC NAME> ;     */
 /*  <PROC HEAD> ::= <PROC NAME> <TYPE> ;     */
 /*  <PROC HEAD> ::= <PROC NAME> <PARAMETER LIST> ;     */
 /*  <PROC HEAD> ::= <PROC NAME> <PARAMETER LIST> <TYPE> ;     */
 /*  <PROC NAME> ::= <LABEL DEF> PROCEDURE     */
 /*  <PARAMETER LIST> ::= <PARAMETER HEAD> <IDENTIFIER> )     */
```

```
 60 | /*  <PARAMETER HEAD> ::= <PARAMETER HEAD> <IDENTIFIER> ,    */
 61 | /*  <ENDING> ::= END    */
 62 | /*  <ENDING> ::= END <IDENTIFIER>    */
 63 | /*  <ENDING> ::= <LABEL DEF> <ENDING>    */
 64 | /*  <RETURN STMT> ::= RETURN    */
 65 | /*  <RETURN STMT> ::= RETURN <EXPR>    */
 66 | /*  <CALL STMT> ::= CALL <VARIABLE>    */
 67 | /*  <GO TO STMT> ::= <GO TO> <IDENTIFIER>    */
 68 | /*  <GO TO> ::= GO TO    */
 69 | /*  <GO TO> ::= GOTO    */
 70 | /*  <DECLARATION STMT> ::= DECLARE <DECLARATION ELEMENT>    */
 71 | /*  <DECLARATION ELEMENT> ::= <IDENTIFIER> LITERALLY <STRING>    */
 72 | /*  <TYPE DECLARATION> ::= <IDENTIFIER SPECIFICATION> <TYPE>    */
 73 | /*  <DECLARATION STMT> ::= <DECLARATION STMT> , <DECLARATION ELEMENT>    */
 74 | /*  <DECLARATION ELEMENT> ::= <TYPE DECLARATION>    */
 75 | /*  <TYPE DECLARATION> ::= <BOUND HEAD> <NUMBER> ) <TYPE>    */
 76 | /*  <TYPE DECLARATION> ::= <TYPE DECLARATION> <INITIAL LIST>    */
 77 | /*  <TYPE> ::= STATUS    */
 78 | /*  <TYPE> ::= REGISTER ( NUMBER )    */
 79 | /*  <TYPE> ::= MEMORY ( NUMBER )    */
 80 | /*  <BOUND HEAD> ::= <IDENTIFIER SPECIFICATION> (    */
 81 | /*  <IDENTIFIER SPECIFICATION> ::= <IDENTIFIER>    */
 82 | /*  <IDENTIFIER SPECIFICATION> ::= <IDENTIFIER LIST> <IDENTIFIER> )    */
 83 | /*  <IDENTIFIER LIST> ::= (    */
 84 | /*  <IDENTIFIER LIST> ::= <IDENTIFIER LIST> <IDENTIFIER> ,    */
 85 | /*  <INITIAL LIST> ::= <INITIAL HEAD> <CONSTANT> )    */
 86 | /*  <INITIAL HEAD> ::= INITIAL (    */
 87 | /*  <INITIAL HEAD> ::= <INITIAL HEAD> <CONSTANT> ,    */
 88 | /*  <ASSIGNMENT> ::= <VAR> <REPLACE> <EXPR>    */
 89 | /*  <ASSIGNMENT> ::= <LEFT PART> <ASSIGNMENT>    */
 90 | /*  <REPLACE> ::= =    */
 91 | /*  <LEFT PART> ::= <VARIABLE> ,    */
 92 | /*  <EXPR> ::= <LOGICAL FACTOR>    */
 93 | /*  <EXPR> ::= <EXPR> | <LOGICAL FACTOR>    */
 94 | /*  <LOGICAL FACTOR> ::= <LOGICAL SECONDARY>    */
 95 | /*  <LOGICAL FACTOR> ::= <LOGICAL FACTOR> & <LOGICAL SECONDARY>    */
 96 | /*  <LOGICAL FACTOR> ::= <LOGICAL FACTOR> XOR <LOGICAL SECONDARY>    */
 97 | /*  <LOGICAL SECONDARY> ::= <LOGICAL PRIMARY>    */
 98 | /*  <LOGICAL SECONDARY> ::= ¬ <LOGICAL PRIMARY>    */
 99 | /*  <LOGICAL PRIMARY> ::= <STRING EXPR>    */
100 | /*  <LOGICAL PRIMARY> ::= <STRING EXPR> <RELATION> <STRING EXPR>    */
101 | /*  <RELATION> ::= =    */
102 | /*  <RELATION> ::= <    */
103 | /*  <RELATION> ::= >    */
104 | /*  <RELATION> ::= ¬ =    */
105 | /*  <RELATION> ::= ¬ <    */
106 | /*  <RELATION> ::= ¬ >    */
107 | /*  <RELATION> ::= < =    */
108 | /*  <RELATION> ::= > =    */
109 | /*  <STRING EXPR> ::= <ARITH EXPR>    */
110 | /*  <STRING EXPR> ::= <STRING EXPR> || <ARITH EXPR>    */
111 | /*  <ARITH EXPR> ::= <TERM>    */
112 | /*  <ARITH EXPR> ::= <ARITH EXPR> + <TERM>    */
113 | /*  <ARITH EXPR> ::= <ARITH EXPR> - <TERM>    */
114 | /*  <ARITH EXPR> ::= + <TERM>    */
115 | /*  <ARITH EXPR> ::= - <TERM>    */
116 | /*  <TERM> ::= <PRIMARY>    */
117 | /*  <PRIMARY> ::= <CONSTANT>    */
118 | /*  <PRIMARY> ::= <VAR>    */
119 | /*  <PRIMARY> ::= ( <EXPR> )    */
120 | /*  <CONSTANT> ::= <NUMBER>    */
121 | /*  <VARIABLE> ::= <IDENTIFIER>    */
122 | /*  <VARIABLE> ::= <SUBSCRIPT HEAD> <EXPR> )    */
123 | /*  <SUBSCRIPT HEAD> ::= <IDENTIFIER> (    */
124 | /*  <SUBSCRIPT HEAD> ::= <SUBSCRIPT HEAD> <EXPR> ,    */
```

```
126 |  DECLARE V(NSY) CHARACTER INITIAL ( '<ERROR: TOKEN = 0>', ';', '(', ')', ',',
127 |     ':', '=', '<', '>', '+', '-', '*', '/', '.', 'IF', 'DO', 'GO', 'TO', ':=',
128 |     'OR', 'BY', '_|_', 'END', 'XOR', 'AND', 'NOT', 'MOD', 'HALT', 'THEN',
129 |     'ELSE', 'CASE', 'CALL', 'GOTO', 'DATA', 'BYTE', 'PLUS', 'LABEL', 'BASED',
130 |     'MINUS', 'WHILE', 'ENABLE', 'RETURN', 'DISABLE', 'DECLARE', 'ADDRESS',
131 |     'INITIAL', '<NUMBER>', '<STRING>', 'INTERRUPT', 'PROCEDURE', 'LITERALLY',
132 |     '<IDENTIFIER>', '<TO>', '<BY>', '<TYPE>', '<COMP>', '<TERM>', '<GROUP>',
133 |     '<WHILE>', '<GO TO>', '<ENDING>', '<PROGRAM>', '<REPLACE>', '<PRIMARY>',
134 |     '<VARIABLE>', '<CONSTANT>', '<RELATION>', '<STATEMENT>', '<IF CLAUSE>',
135 |     '<TRUE PART>', '<DATA LIST>', '<DATA HEAD>', '<LEFT PART>',
136 |     '<ASSIGNMENT>', '<EXPRESSION>', '<GROUP HEAD>', '<BOUND HEAD>',
137 |     '<IF STATEMENT>', '<WHILE CLAUSE>', '<INITIAL LIST>', '<INITIAL HEAD>',
138 |     '<CASE SELECTOR>', '<VARIABLE NAME>', '<CONSTANT HEAD>',
139 |     '<STATEMENT LIST>', '<CALL STATEMENT>', '<PROCEDURE HEAD>',
140 |     '<PROCEDURE NAME>', '<PARAMETER LIST>', '<PARAMETER HEAD>',
141 |     '<BASED VARIABLE>', '<LOGICAL FACTOR>', '<SUBSCRIPT HEAD>',
142 |     '<BASIC STATEMENT>', '<GO TO STATEMENT>', '<STEP DEFINITION>',
143 |     '<IDENTIFIER LIST>', '<LOGICAL PRIMARY>', '<RETURN STATEMENT>',
144 |     '<LABEL DEFINITION>', '<TYPE DECLARATION>', '<ITERATION CONTROL>',
145 |     '<LOGICAL SECONDARY>', '<LOGICAL EXPRESSION>', '<DECLARATION ELEMENT>',
146 |     '<PROCEDURE DEFINITION>', '<DECLARATION STATEMENT>',
147 |     '<ARITHMETIC EXPRESSION>', '<IDENTIFIER SPECIFICATION>');
148 |  DECLARE V_INDEX(12) BIT(8) INITIAL ( 1, 14, 21, 27, 36, 40, 42, 44, 48, 51,
149 |     51, 51, 52);
150 |  DECLARE C1(NSY) BIT(104) INITIAL (
151 |     "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00",
152 |     "(2) 02000 00000 00002 22000 02200 00202 02200 00000 22220 00002 02",
153 |     "(2) 02222 02222 22000 00222 20022 02020 00003 23020 00003 00000 00",
154 |     "(2) 00030 00000 20020 00000 00000 30000 00000 00000 00000 03000 03",
155 |     "(2) 00030 00000 20020 00000 00000 20000 00000 00000 00000 02000 03",
156 |     "(2) 02000 00000 00002 22000 02200 00200 02200 00000 22220 00002 02",
157 |     "(2) 00020 00000 20020 00000 00000 20000 00000 00000 00000 02000 02",
158 |     "(2) 00020 01010 20020 00000 00000 00000 00000 00000 00000 02000 02",
159 |     "(2) 00020 01000 20020 00000 00000 00000 00000 00000 00000 02000 02",
160 |     "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 01100 01",
161 |     "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 01100 01",
162 |     "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 01100 01",
163 |     "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 01100 01",
164 |     "(2) 00010 00000 00000 00000 00000 00000 00000 00000 00000 01100 01",
165 |     "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 01100 01",
166 |     "(2) 01000 00000 00000 00000 00000 00000 10000 00001 00000 00000 01",
167 |     "(2) 00000 00000 00000 00100 00000 00000 00000 00000 00000 01000 01",
168 |     "(2) 00020 00000 20020 00000 00000 20000 00000 00000 00000 02000 02",
169 |     "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 01100 01",
170 |     "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 01100 01",
171 |     "(2) 00020 00000 20020 00000 00000 20000 00000 00000 00000 02000 02",
172 |     "(2) 01000 00000 00001 11000 01000 00100 01100 00000 11110 01000 01",
173 |     "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 02000 01",
174 |     "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 01100 01",
175 |     "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 01100 01",
176 |     "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 01100 01",
177 |     "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 01100 01",
178 |     "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 02000 02",
179 |     "(2) 02000 00000 00002 22000 02200 00200 02200 00000 22220 00002 02",
180 |     "(2) 02000 00000 00002 22000 00000 00200 02200 00000 22220 02000 02",
181 |     "(2) 00010 00000 10010 00000 00000 10000 00000 00000 00000 01100 01",
182 |     "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 01",
183 |     "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 02000 02",
184 |     "(2) 00010 00000 00000 00000 00000 00000 00000 00000 00000 00000 00",
185 |     "(2) 02002 00000 00000 00000 00000 00000 00000 00000 00000 02000 00",
186 |     "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 01100 01",
187 |     "(2) 02002 00000 00000 00000 00000 00000 00000 00000 00000 02000 00",
188 |     "(2) 00000 01000 00000 00010 00000 00000 00000 00000 00000 02000 02",
189 |     "(2) 00010 00000 00010 00000 00000 00000 00000 00000 00000 01100 01",
190 |     "(2) 00020 00000 20020 00000 00000 20000 00000 00000 00000 02000 02",
```

```
191 |      "(2) 01000 [illegible]                                                
```

```
258 |     "(2) 02202 01111 10000 00202 20022 00020 00000 10010 00000 10000 00",     | 129
259 |     "(2) 00010 00000 00000 00000 00000 00000 00000 00001 01000 00001 00000 00");     | 128
260 |  DECLARE NCITRIPLES LITERALLY '228';     | 128
261 |  DECLARE CITRIPLES(NCITRIPLES) FIXED INITIAL ( 197373, 197386, 197389, 197401,| 128
262 |     197422, 197423, 197427, 200697, 590595, 590602, 590605, 590611, 590633,    | 128
263 |     590639, 590643, 602383, 656131, 656138, 656141, 656153, 656174, 656175,    | 128
264 |     656179, 660419, 721667, 721674, 721677, 721689, 721710, 721711, 721715,    | 128
265 |     733955, 787203, 787210, 787213, 787225, 787246, 787247, 787251, 795491,    | 128
266 |     865027, 918275, 918282, 918285, 918297, 918318, 918319, 918323, 920563,    | 128
267 |     996099, 994430, 1180419, 1180426, 1180429, 1180441, 1180462, 1180463,     | 128
268 |     1180467, 1192707, 1245955, 1245962, 1245965, 1245977, 1245981, 1245999,    | 128
269 |     1246003, 1258243, 1399315, 1392644, 1392646, 1394945, 1508114, 1508106,    | 128
270 |     1508109, 1508121, 1508142, 1508143, 1508147, 1520387, 1573614, 1573642,    | 128
271 |     1573645, 1573657, 1573678, 1573679, 1573685, 1585023, 1639171, 1639178,    | 128
272 |     1639181, 1639193, 1639214, 1639215, 1639219, 1651459, 1704713, 1704714,    | 128
273 |     1704717, 1704729, 1704750, 1704751, 1704755, 1716995, 1966851, 1966853,    | 128
274 |     1966861, 1966873, 1966884, 1966885, 1966899, 1979139, 2044675, 2294531,    | 128
275 |     2294533, 2294541, 2294552, 2294574, 2294575, 2294579, 2305811, 2491139,    | 128
276 |     2491146, 2491149, 2491161, 2491182, 2491183, 2491187, 2503427, 2687747,    | 128
277 |     2687754, 2687757, 2687769, 2687790, 2687791, 2687795, 2700035, 3015202,    | 128
278 |     3015204, 3015212, 3157505, 3408643, 3408650, 3408653, 3408662, 3408686,    | 128
279 |     3408687, 3408691, 3420931, 3474179, 3474186, 3474189, 3474201, 3474222,    | 128
280 |     3474223, 3474227, 3486467, 3801859, 3801866, 3801869, 3801881, 3801902,    | 128
281 |     3801903, 3801907, 3814147, 4054003, 4064010, 4064013, 4064025, 4064046,    | 128
282 |     4064047, 4064051, 4076291, 4326147, 4326154, 4326157, 4326171, 4326190,    | 128
283 |     4326191, 4326195, 4338435, 4469507, 4472936, 4472939, 4475137, 4490285,    | 128
284 |     4535043, 4536872, 4538374, 4540673, 4669698, 4669700, 4731651, 4734950,    | 128
285 |     4734982, 4924359, 4951508, 4951590, 4933899, 4992514, 5259522, 5259524,    | 128
286 |     5456130, 5456132, 5518383, 5521412, 5521414, 5523713, 5649185, 5652464,    | 128
287 |     5652484, 5654705, 5713457, 5730693, 5845762, 5845764, 6030883, 6031090,    | 128
288 |     6030093, 6030105, 6030124, 6030127, 6030131, 6042371, 6501124, 6504452,    | 128
289 |     6504454, 6506753, 6947963, 6947981);     | 128
290 |  DECLARE PST5(129) FIXED INITIAL (0, 5724214, 5718966, 22324, 22360, 3935,     | 128
291 |     3918, 3921, 87, 15, 73, 57, 105, 98, 85, 94, 106, 27, 40, 42, 0, 18241,    | 128
292 |     20545, 22835, 24658, 21313, 842, 23626, 33, 109, 45, 13, 51, 0, 0, 22835,  | 128
293 |     18241, 24658, 20545, 21313, 23626, 64, 51, 46, 7, 8, 0, 0, 0, 7, 0, 16, 0,| 128
294 |     0, 0, 3458, 93, 0, 0, 0, 51, 0, 0, 0, 59, 0, 13106, 0, 99, 22, 59, 93, 0,  | 128
295 |     0, 4092514, 103, 0, 27401, 27402, 27427, 27430, 10, 0, 22100, 99, 75,      | 128
296 |     14347, 14348, 14362, 0, 31, 13, 0, 13, 0, 17477, 64, 75, 66, 0, 51, 72,    | 128
297 |     3426969, 15446, 52, 56, 30, 41, 99, 0, 100, 0, 0, 26367, 26391, 0, 99, 0,  | 128
298 |     25, 0, 0, 4210250, 23620, 0, 16402, 0, 27140, 43, 27458, 0);     | 128
299 |  DECLARE PST8(129) BIT(8) INITIAL (0, 38, 39, 35, 37, 25, 26, 27, 35, 24, 6,    | 128
300 |     7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 61, 78, 41, 72, 115, 118, 122, 62,    | 128
301 |     70, 79, 119, 123, 42, 73, 43, 63, 74, 80, 120, 124, 34, 47, 48, 101, 102,  | 128
302 |     96, 83, 97, 100, 95, 84, 127, 128, 44, 21, 22, 55, 67, 69, 77, 126, 49,    | 128
303 |     68, 53, 126, 50, 100, 40, 45, 52, 76, 75, 121, 65, 64, 99, 104, 105, 106,  | 128
304 |     107, 108, 103, 84, 45, 23, 110, 111, 112, 109, 51, 117, 116, 114, 113, 19,| 128
305 |     3, 28, 18, 2, 60, 82, 31, 81, 30, 32, 33, 50, 20, 5, 66, 71, 1, 89, 89,    | 128
306 |     87, 17, 4, 93, 92, 53, 29, 91, 90, 86, 85, 57, 56, 95, 94);     | 128
307 |  DECLARE NOTS(129) BIT(8) INITIAL (0, 86, 86, 86, 86, 75, 75, 75, 86, 75, 93,  | 128
308 |     93, 93, 93, 93, 93, 93, 93, 93, 93, 70, 79, 88, 108, 63, 63, 64, 71,      | 128
309 |     76, 90, 83, 92, 59, 46, 80, 71, 96, 80, 83, 92, 72, 99, 99, 25, 65, 66,    | 128
310 |     62, 66, 55, 65, 59, 57, 93, 60, 63, 69, 59, 54, 54, 90, 58, 58, 54, 94,    | 128
311 |     55, 104, 65, 97, 60, 94, 82, 82, 64, 100, 100, 66, 107, 107, 107, 107,     | 128
312 |     107, 107, 105, 60, 57, 56, 56, 56, 56, 85, 63, 63, 63, 63, 77, 94, 75, 77,| 128
313 |     84, 104, 73, 101, 73, 101, 78, 81, 98, 77, 67, 100, 108, 61, 103, 103,     | 128
314 |     103, 93, 67, 102, 102, 104, 95, 91, 91, 74, 74, 106, 106, 97, 97);     | 128
315 |  DECLARE PSLENGTH(129) BIT(3) INITIAL (0, 4, 4, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2,  | 128
316 |     2, 2, 2, 2, 2, 2, 2, 1, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 2, 3, 3,| 128
317 |     3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 3, 2, 1, 1, 1, 2, 1, 1,| 128
318 |     1, 2, 1, 3, 1, 2, 2, 2, 2, 1, 1, 4, 2, 1, 3, 3, 3, 3, 2, 1, 3, 2, 2, 3, 3,| 128
319 |     3, 1, 2, 2, 1, 2, 1, 3, 2, 2, 2, 1, 2, 2, 4, 3, 2, 2, 2, 2, 2, 1, 2, 1, 1,| 128
320 |     3, 3, 1, 2, 1, 2, 1, 1, 4, 3, 1, 3, 1, 3, 2, 3, 1);     | 128
321 |  DECLARE CONTEXT_CASE(129) BIT(8) INITIAL (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,| 128
322 |     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,| 128
```

```
323 |     [illegible]                                                                    |
324 |     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,|
325 |     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,|
326 |     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);                          |
327 |  DECLARE LEFT_CONTEXT(4) BIT(8) INITIAL ( 107, 4, 43, 96, 87);                   |
328 |  DECLARE LEFT_INDEX(57) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,|
329 |     0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 4, 4, 4, 4, 4,|
330 |     4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5);           |
331 |  DECLARE CONTEXT_TRIPLE(0) FIXED INITIAL ( 0);                                    |
332 |  DECLARE TRIPLE_INDEX(57) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,|
333 |     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,|
334 |     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1);           |
335 |  DECLARE PR_INDEX(108) BIT(8) INITIAL ( 1, 21, 29, 35, 42, 44, 45, 49, 51, 51,|
336 |     51, 51, 51, 51, 51, 51, 51, 53, 53, 53, 54, 54, 55, 55, 55, 55, 55, 55,    |
337 |     56, 57, 57, 57, 58, 58, 59, 59, 60, 61, 61, 62, 62, 63, 63, 63, 64, 64,    |
338 |     66, 68, 69, 69, 69, 74, 74, 74, 76, 77, 83, 83, 83, 83, 86, 86, 86, 90,    |
339 |     90, 95, 95, 100, 100, 100, 101, 101, 101, 102, 108, 108, 108, 108, 110, 110,|
340 |     111, 111, 111, 112, 112, 113, 113, 113, 113, 113, 113, 113, 116, 116, 118,|
341 |     118, 118, 118, 120, 120, 120, 121, 122, 124, 126, 128, 128, 128, 130,      |
342 |     130);                                                                       |
343 |                                                                                 |
344 |  /*  END OF CARDS PUNCHED BY SYNTAX                                        */  |
345 |                                                                                 |
346 |  /*  DECLARATIONS FOR THE SCANNER                                           */  |
347 |                                                                                 |
348 |  /* TOKEN IS THE INDEX INTO THE VOCABULARY V() OF THE LAST SYMBOL SCANNED,     |
349 |     CP IS THE POINTER TO THE LAST CHARACTER SCANNED IN THE CARDIMAGE,          |
350 |     BCD IS THE LAST SYMBOL SCANNED (LITERAL CHARACTER STRING). */              |
351 |  DECLARE (TOKEN, CP) FIXED, BCD CHARACTER;                                      |
352 |                                                                                 |
353 |  /* SET UP SOME CONVENIENT ABBREVIATIONS FOR PRINTER CONTROL */                 |
354 |  DECLARE EJECT_PAGE LITERALLY 'OUTPUT(1) = PAGE',                               |
355 |     PAGE CHARACTER INITIAL ('1'), DOUBLE CHARACTER INITIAL ('0'),               |
356 |     DOUBLE_SPACE LITERALLY 'OUTPUT(1) = DOUBLE',                                |
357 |     X70 CHARACTER INITIAL ('                                                    |
358 |                    ');                                                          |
359 |                                                                                 |
360 |  /* LENGTH OF LONGEST SYMBOL IN V */                                          |
361 |  DECLARE (RESERVED_LIMIT, MARGIN_CHOP) FIXED;                                   |
362 |                                                                                 |
363 |  /* CHARTYPE() IS USED TO DISTINGUISH CLASSES OF SYMBOLS IN THE SCANNER.       |
364 |     TX() IS A TABLE USED FOR TRANSLATING FROM ONE CHARACTER SET TO ANOTHER.    |
365 |     CONTROL() HOLDS THE VALUE OF THE COMPILER CONTROL TOGGLES SET IN $ CARDS.  |
366 |     NOT_LETTER_OR_DIGIT() IS SIMILIAR TO CHARTYPE() BUT USED IN SCANNING       |
367 |     IDENTIFIERS ONLY.                                                          |
368 |                                                                                 |
369 |     ALL ARE USED BY THE SCANNER AND CONTROL() IS SET THERE.                    |
370 |  */                                                                             |
371 |  DECLARE (CHARTYPE, TX) (255) BIT(8),                                           |
372 |        (CONTROL, NOT_LETTER_OR_DIGIT)(255) BIT(1);                             |
373 |                                                                                 |
374 |  /* ALPHABET CONSISTS OF THE SYMBOLS CONSIDERED ALPHABETIC IN BUILDING         |
375 |     IDENTIFIERS     */                                                         |
376 |  DECLARE ALPHABET CHARACTER INITIAL ('ABCDEFGHIJKLMNOPQRSTUVWXYZ_$@#');         |
377 |                                                                                 |
378 |  /* BUFFER HOLDS THE LATEST CARDIMAGE,                                        |
379 |     TEXT HOLDS THE PRESENT STATE OF THE INPUT TEXT                             |
380 |     (NOT INCLUDING THE PORTIONS DELETED BY THE SCANNER),                       |
381 |     TEXT_LIMIT IS A CONVENIENT PLACE TO STORE THE POINTER TO THE END OF TEXT,  |
382 |     CARD_COUNT IS INCREMENTED BY ONE FOR EVERY SOURCE CARD READ,               |
383 |     ERROR_COUNT TABULATES THE ERRORS AS THEY ARE DETECTED,                     |
384 |     SEVERE_ERRORS TABULATES THOSE ERRORS OF FATAL SIGNIFICANCE.                |
385 |  */                                                                             |
386 |  DECLARE (BUFFER, TEXT) CHARACTER,                                              |
387 |     (TEXT_LIMIT, CARD_COUNT, ERROR_COUNT, SEVERE_ERRORS, PREVIOUS_ERROR) FIXED;|
388 |     ;                                                                           |
```

```
/* NUMBER_VALUE CONTAINS THE NUMERIC VALUE OF THE LAST CONSTANT SCANNED,
*/
DECLARE NUMBER_VALUE FIXED;

/* EACH OF THE FOLLOWING CONTAINS THE INDEX INTO V() OF THE CORRESPONDING
   SYMBOL.   WE ASK:    IF TOKEN = IDENT    ETC.    */
DECLARE (IDENT, NUMBER, DIVIDE, EOFILE) FIXED;

/* STOPIT() IS A TABLE OF SYMBOLS WHICH ARE ALLOWED TO TERMINATE THE ERROR
   FLUSH PROCESS.  IN GENERAL THEY ARE SYMBOLS OF SUFFICIENT SYNTACTIC
   HIERARCHY THAT WE EXPECT TO AVOID ATTEMPTING TO START CHECKING AGAIN
   RIGHT INTO ANOTHER ERROR PRODUCING SITUATION.  THE TOKEN STACK IS ALSO
   FLUSHED DOWN TO SOMETHING ACCEPTABLE TO A STOPIT() SYMBOL.
   FAILSOFT IS A BIT WHICH ALLOWS THE COMPILER ONE ATTEMPT AT A GENTLE
   RECOVERY.   THEN IT TAKES A STRONG HAND.   WHEN THERE IS REAL TROUBLE
   COMPILING IS SET TO FALSE, THEREBY TERMINATING THE COMPILATION.
*/
DECLARE STOPIT(100) BIT(1), (FAILSOFT, COMPILING) BIT(1);

DECLARE S CHARACTER;  /* A TEMPORARY USED VARIOUS PLACES */

/* THE ENTRIES IN PRMASK() ARE USED TO SELECT OUT PORTIONS OF CODED
   PRODUCTIONS AND THE STACK TOP FOR COMPARISON IN THE ANALYSIS ALGORITHM */
DECLARE PRMASK(5) FIXED INITIAL (0, 0, "FF", "FFFF", "FFFFFF", "FFFFFFFF");

/*THE PROPER SUBSTRING OF POINTER IS USED TO PLACE AN     UNDER THE POINT
   OF DETECTION OF AN ERROR DURING CHECKING.  IT MARKS THE LAST CHARACTER
   SCANNED.  */
DECLARE POINTER CHARACTER INITIAL ('
                                   ');
DECLARE CALLCOUNT(20) FIXED   /* COUNT THE CALLS OF IMPORTANT PROCEDURES */
   INITIAL(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);

/* RECORD THE TIMES OF IMPORTANT POINTS DURING CHECKING */
DECLARE CLOCK(5) FIXED;

/* COMMONLY USED STRINGS */
DECLARE X1 CHARACTER INITIAL(' '), X4 CHARACTER INITIAL('    ');
DECLARE PERIOD CHARACTER INITIAL ('.');

/* TEMPORARIES USED THROUGHOUT THE COMPILER */
DECLARE (I, J, K, L) FIXED;

DECLARE TRUE LITERALLY '1', FALSE LITERALLY '0', FOREVER LITERALLY 'WHILE 1';

/*  THE STACKS DECLARED BELOW ARE USED TO DRIVE THE SYNTACTIC
   ANALYSIS ALGORITHM AND STORE INFORMATION RELEVANT TO THE INTERPRETATION
   OF THE TEXT.  THE STACKS ARE ALL POINTED TO BY THE STACK POINTER SP.  */

DECLARE STACKSIZE LITERALLY '75';  /* SIZE OF STACK  */
DECLARE PARSE_STACK (STACKSIZE) BIT(8); /* TOKENS OF THE PARTIALLY PARSED
                                           TEXT */
DECLARE VAR (STACKSIZE) CHARACTER;/* EBCDIC NAME OF ITEM */
DECLARE FIXV (STACKSIZE) FIXED;   /* FIXED (NUMERIC) VALUE */

/* SP POINTS TO THE RIGHT END OF THE REDUCIBLE STRING IN THE PARSE STACK,
   MP POINTS TO THE LEFT END, AND
   MPP1 = MP+1.
*/
DECLARE (SP, MP, MPP1) FIXED;

  DECLARE SAVEREF FIXED;  /*  REFERS BACK TO A PREVIOUSLY
                             REFERENCED LABEL  */
```

```
455 |     DECLARE SLABEL(LABS) INITIAL(0);                                                    |  1
456 |     DECLARE LABELS LITERALLY '25';  /*  SIZE OF LABEL TABLE  */                        |  1
457 |     DECLARE LABID(LABELS) CHARACTER;  /*  LABEL IDENTIFIER  */                         |  1
458 |     DECLARE (LASTREF,LABDEF) (LABELS) FIXED;  /* REST OF LABEL TABLE */               |  1
459 |     DECLARE TBASIC FIXED;                                                             |  1
460 |     DECLARE SAVEQUAD FIXED,                                                           |  1
461 |             SAVEQUAD2 FIXED;                                                          |  1
462 |     DECLARE NEXTQUAD FIXED INITIAL (1);                                               |  1
463 |     DECLARE TABLE_LOC (STACKSIZE) FIXED;   /* LOCATION OF SYMBOLS IN TABLES */        |  1
464 |     DECLARE SYMBOLS LITERALLY '100'; /* SIZE OF SYMBOL TABLE */                       |  1
465 |     DECLARE SYMB(SYMBOLS) CHARACTER ; /* IDENTIFIER */                                |  1
466 |     DECLARE (LOCAT,DEF,SIZE,INIT) (SYMBOLS) FIXED; /* REST OF SYMBOL TABLE */       |  1
467 |     DECLARE SAVQUD(10) FIXED;                                                         |  1
468 |     DECLARE SAVQUDNO FIXED INITIAL(0);                                                |  1
469 |     DECLARE SAVLAB(10) CHARACTER;                                                     |  1
470 |     DECLARE SAVLABNO FIXED INITIAL(0);                                                |  1
471 | /*   QUAD TYPES */                                                                   |  1
472 |     DECLARE NVARDEF FIXED,                                                            |  1
473 |             NSYMBOL FIXED INITIAL (0),                                                |  1
474 |             NCONSTANT FIXED INITIAL (0),                                              |  1
475 |     ADD   FIXED  INITIAL  (1),                                                        |  1
476 |     MUL   FIXED  INITIAL  (2),                                                        |  1
477 |     SUB   FIXED  INITIAL  (3),                                                        |  1
478 |     DIV   FIXED  INITIAL  (4),                                                        |  1
479 |     MMOD  FIXED  INITIAL  (5),                                                        |  1
480 |     HALT  FIXED  INITIAL  (6),                                                        |  1
481 |     BR    FIXED  INITIAL  (7),                                                        |  1
482 |     BT    FIXED  INITIAL  (8),                                                        |  1
483 |     BF    FIXED  INITIAL  (9),                                                        |  1
484 |     REL   FIXED  INITIAL  (10),                                                       |  1
485 |     EQ    FIXED  INITIAL  (11),                                                       |  1
486 |     LT    FIXED  INITIAL  (12),                                                       |  1
487 |     GT    FIXED  INITIAL  (13),                                                       |  1
488 |     NE    FIXED  INITIAL  (14),                                                       |  1
489 |     LE    FIXED  INITIAL  (15),                                                       |  1
490 |     GE    FIXED  INITIAL  (16),                                                       |  1
491 |     ASSN  FIXED  INITIAL  (17),                                                       |  1
492 |     SUBS  FIXED  INITIAL  (18),                                                       |  1
493 |     BZ    FIXED  INITIAL  (19),                                                       |  1
494 |     AND   FIXED  INITIAL  (20),                                                       |  1
495 |     OR    FIXED  INITIAL  (21),                                                       |  1
496 |     UMIN  FIXED  INITIAL  (22),                                                       |  1
497 |     ZQ    FIXED  INITIAL  (23),                                                       |  1
498 |     LAB   FIXED  INITIAL  (24),                                                       |  1
499 |     SUBL  FIXED  INITIAL  (30),                                  /*NEWQUAD*/        |  1
500 |     TGOTO  FIXED  INITIAL (0);                                                        |  1
501 |     DECLARE CORE FIXED INITIAL (0);                                                   |  1
502 |     DECLARE SAVEINDEX FIXED;                                                          |  1
503 |     DECLARE LOOPLIM FIXED INITIAL (0),                                                |  1
504 |             LOOPINC FIXED INITIAL (1),                                                |  1
505 |             LOOP_INDEX FIXED INITIAL (0);                                             |  1
506 |     DECLARE SAVELOC FIXED;                                                            |  1
507 |     DECLARE SAVELAB CHARACTER;                                                        |  1
508 |     DECLARE SAVEVAR CHARACTER;                                                        |  1
509 |     DECLARE DO_SWITCH FIXED INITIAL (0);                                              |  1
510 |     DECLARE (MULLOC,DIVLOC) FIXED;                                                    |  1
511 |     DECLARE CONSTANTS LITERALLY '50'; /* SIZE OF CONSTANTS TABLE */                   |  1
512 |     DECLARE (CONLOC,CONVAL) (CONSTANTS) FIXED; /* CONSTANTS TABLE */                  |  1
513 | /*   DISK BUFFER FOR QUAD TABLE     */                                               |  1
514 |     DECLARE DISKWORDS LITERALLY '900',                                                |  1
515 |             MAXQUADS LITERALLY '220',                                                 |  1
516 |             QUADS (DISKWORDS) FIXED;                                                  |  1
517 |     DECLARE RCD_BUFF FIXED INITIAL (0);                                               |  1
518 |     DECLARE RCD_NR FIXED;                                                             |  1
519 |     DECLARE LOC_QUAD FIXED;                                                           |  1
520 |         DECLARE BR_FLAG BIT(8);                                  /*NEWQUAD*/        |  1
```

```
521 |          DECLARE CURR_VAR_VALUE(*);                                  /*NEWQUAD*/ |  12
522 |          DECLARE MIC_LOC(25) FIXED;                                  /*NEWQUAD*/ |  12
523 |          DECLARE BUF_MIC CHARACTER INITIAL(                          /*NEWQUAD*/ |  12
524 |'000 0 00 0 0 0 000 0000 000 00000 0000 0000 00 0 00000 0000 0000 00000000');   |  12
525 |          DECLARE(OPERATION,OPERAND1,OPERAND2,RESULT)CHARACTER;       /*NEWQUAD*/ |  12
526 |          DECLARE (LAVS) FIXED INITIAL(0);                            /*NEWQUAD*/ |  12
527 |          DECLARE VARIBLS(40) CHARACTER;                              /*NEWQUAD*/ |  12
528 |          DECLARE CHANGE(40) BIT(8);                                  /*NEWQUAD*/ |  12
529 |          DECLARE NEXT_VAR(40) FIXED INITIAL                          /*NEWQUAD*/ |  12
530 |          (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22, /*NEWQUAD*/ |  12
531 |          23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40);     /*NEWQUAD*/ |  12
532 |          DECLARE (RD) BIT(8);                                        /*NEWQUAD*/ |  12
533 |          DECLARE STATUS(7) BIT(8);                                   /*NEWQUAD*/ |  12
534 |      DECLARE ADDRSS(7) BIT(8);                                       /*NEWQUAD*/ |  12
535 |      DECLARE SUBSFLAG FIXED INITIAL(0);                              /*NEWQUAD*/ |  12
536 |          DECLARE VARNUM(7) FIXED;                                    /*NEWQUAD*/ |  12
537 |          DECLARE DEALLOCABL(7) BIT(8);                               /*NEWQUAD*/ |  12
538 |          DECLARE REFRENCE(7) FIXED;                                  /*NEWQUAD*/ |  12
539 |          DECLARE TEMP(7) FIXED;                                      /*NEWQUAD*/ |  12
540 |          DECLARE POINT(7) FIXED;                                     /*NEWQUAD*/ |  12
541 |          DECLARE MAXREG FIXED INITIAL(6);                            /*NEWQUAD*/ |  12
542 |          DECLARE(II,JJ,KK,LL,NN)FIXED;                               /*NEWQUAD*/ |  12
543 |          DECLARE (PREND) FIXED INITIAL (0);                          /*NEWQUAD*/ |  12
544 |          DECLARE SORTREF(6) FIXED;                                   /*NEWQUAD*/ |  12
545 |          DECLARE SORTNUM(6) FIXED;                                   /*NEWQUAD*/ |  12
546 |          DECLARE REGNO(6) FIXED;                                     /*NEWQUAD*/ |  12
547 |          DECLARE NEWQUAD(900) FIXED;                                 /*NEWQUAD*/ |  12
548 |          DECLARE NEWQUADNO FIXED INITIAL(0);                         /*NEWQUAD*/ |  12
549 |          DECLARE TEMP_CHAR CHARACTER;                                /*NEWQUAD*/ |  12
550 |          DECLARE(OPRTOR,OPRND1,OPRND2,RSLT)FIXED;                    /*NEWQUAD*/ |  12
551 |          DECLARE OPER FIXED;                                         /*NEWQUAD*/ |  12
552 |      DECLARE ALPHA(30) CHARACTER INITIAL ('WTAD',                    /*NEWQUAD*/ |  12
553 |                                          'ADD ',                     /*NEWQUAD*/ |  12
554 |                                          'MUL ',                     /*NEWQUAD*/ |  12
555 |                                          'SUB ',                     /*NEWQUAD*/ |  12
556 |                                          'DIV ',                     /*NEWQUAD*/ |  12
557 |                                          'MOD ',                     /*NEWQUAD*/ |  12
558 |                                          'HALT',                     /*NEWQUAD*/ |  12
559 |                                          'BR  ',                     /*NEWQUAD*/ |  12
560 |                                          'BT  ',                     /*NEWQUAD*/ |  12
561 |                                          'BF  ',                     /*NEWQUAD*/ |  12
562 |                                          'REL ',                     /*NEWQUAD*/ |  12
563 |                                          'EQ  ',                     /*NEWQUAD*/ |  12
564 |                                          'LT  ',                     /*NEWQUAD*/ |  12
565 |                                          'GT  ',                     /*NEWQUAD*/ |  12
566 |                                          'NE  ',                     /*NEWQUAD*/ |  12
567 |                                          'LE  ',                     /*NEWQUAD*/ |  12
568 |                                          'GE  ',                     /*NEWQUAD*/ |  12
569 |                                          'ASGN',                     /*NEWQUAD*/ |  12
570 |                                          'SUBS',                     /*NEWQUAD*/ |  12
571 |                                          'BZ  ',                     /*NEWQUAD*/ |  12
572 |                                          'AND ',                     /*NEWQUAD*/ |  12
573 |                                          'OR  ',                     /*NEWQUAD*/ |  12
574 |                                          'UMIN',                     /*NEWQUAD*/ |  12
575 |                                          'ZQ  ',                     /*NEWQUAD*/ |  12
576 |                                          'LAB ',                     /*NEWQUAD*/ |  12
577 |                                          'RD  ',                     /*NEWQUAD*/ |  12
578 |                                          'WT  ',                     /*NEWQUAD*/ |  12
579 |                                          'RDAD',                     /*NEWQUAD*/ |  12
580 |                                          'RDVR',                     /*NEWQUAD*/ |  12
581 |                                          'ASL ',                     /*NEWQUAD*/ |  12
582 |                                      'SUBL');                        /*NEWQUAD*/ |  12
583 |                                                                                |  12
584 |                                                                                |  12
585 |                                                                                |  12
586 |                                                                                |  12
```

```
588 |   /*              P R O C E D U R E S                              */
589 |
590 |
591 |
592 |PAD:
593 |    PROCEDURE (STRING, WIDTH) CHARACTER;
594 |       DECLARE STRING CHARACTER, (WIDTH, L) FIXED;
595 |
596 |       L = LENGTH(STRING);
597 |       IF L >= WIDTH THEN RETURN STRING;
598 |       ELSE RETURN STRING || SUBSTR(X70, 0, WIDTH-L);
599 |    END PAD;
600 |
601 |I_FORMAT:
602 |    PROCEDURE (NUMBER, WIDTH) CHARACTER;
603 |       DECLARE (NUMBER, WIDTH, L) FIXED, STRING CHARACTER;
604 |
605 |       STRING = NUMBER;
606 |       L = LENGTH(STRING);
607 |       IF L >= WIDTH THEN RETURN STRING;
608 |       ELSE RETURN SUBSTR(X70, 0, WIDTH-L) || STRING;
609 |    END I_FORMAT;
610 |
611 |ERROR:
612 |    PROCEDURE(MSG, SEVERITY);
613 |       /* PRINTS AND ACCOUNTS FOR ALL ERROR MESSAGES */
614 |       /* IF SEVERITY IS NOT SUPPLIED, 0 IS ASSUMED */
615 |       DECLARE MSG CHARACTER, SEVERITY FIXED;
616 |       ERROR_COUNT = ERROR_COUNT + 1;
617 |       /* IF LISTING IS SUPPRESSED, FORCE PRINTING OF THIS LINE */
618 |       IF ¬ CONTROL(BYTE('L')) THEN
619 |          OUTPUT = I_FORMAT (CARD_COUNT, 4) || ' |' || BUFFER || '|';
620 |       OUTPUT = SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP);
621 |       OUTPUT = '*** ERROR, ' || MSG ||
622 |             '.  LAST PREVIOUS ERROR WAS DETECTED ON LINE ' ||
623 |             PREVIOUS_ERROR || '.  ***';
624 |       PREVIOUS_ERROR = CARD_COUNT;
625 |       IF SEVERITY > 0 THEN
626 |          IF SEVERE_ERRORS > 25 THEN
627 |             DO;
628 |                OUTPUT = '*** TOO MANY SEVERE ERRORS, CHECKING ABORTED ***';
629 |                COMPILING = FALSE;
630 |             END;
631 |          ELSE SEVERE_ERRORS = SEVERE_ERRORS + 1;
632 |    END ERROR;
633 |
634 |
635 |
636 |
637 |
638 |
639 |  /*                  CARD IMAGE HANDLING PROCEDURE                 */
640 |
641 |
642 |GET_CARD:
643 |    PROCEDURE;
644 |       /* DOES ALL CARD READING AND LISTING                          */
645 |       DECLARE I FIXED, (TEMP, TEMP0, REST) CHARACTER, READING BIT(1);
646 |             BUFFER = INPUT;
647 |             IF LENGTH(BUFFER) = 0 THEN
648 |                DO; /* SIGNAL FOR EOF */
649 |                   CALL ERROR ('EOF MISSING OR COMMENT STARTING IN COLUMN 1.',1);
650 |                   BUFFER = PAD (' /*''/* */ EOF;END;EOF', 80);
651 |                END;
652 |             ELSE CARD_COUNT = CARD_COUNT + 1;  /* USED TO PRINT ON LISTING */
```

```
653 |       IF MARGIN_CHOP > 0 THEN                                                    | 2024 GET_CARD
654 |          DO; /* THE MARGIN CONTROL FROM DOLLAR   */                             | 2048 GET_CARD
655 |             I = LENGTH(BUFFER) - MARGIN_CHOP;                                    | 2040 GET_CARD
656 |             REST = SUBSTR(BUFFER, I);                                            | 2066 GET_CARD
657 |             BUFFER = SUBSTR(BUFFER, 0, I);                                       | 2092 GET_CARD
658 |          END;                                                                    | 2118 GET_CARD
659 |       ELSE REST = '';                                                            | 2118 GET_CARD
660 |       TEXT = BUFFER;                                                             | 2128 GET_CARD
661 |       TEXT_LIMIT = LENGTH(TEXT) - 1;                                             | 2136 GET_CARD
662 |       IF CONTROL(BYTE('M')) THEN OUTPUT = BUFFER;                                | 2162 GET_CARD
663 |       ELSE IF CONTROL(BYTE('L')) THEN                                            | 2196 GET_CARD
664 |          OUTPUT = I_FORMAT (CARD_COUNT, 4)  ||' |' || BUFFER || '|' || REST;    | 2204 GET_CARD
665 |       CP = 0;                                                                    | 2316 GET_CARD
666 |    END GET_CARD;                                                                 | 2322 GET_CARD
667 |                                                                                  | 2328
668 |                                                                                  | 2328
669 |    /*                  THE SCANNER PROCEDURES                 */                | 2328
670 |                                                                                  | 2328
671 |                                                                                  | 2328
672 |CHAR:                                                                             | 2328
673 |    PROCEDURE;                                                                    | 2328
674 |       /* USED FOR STRINGS TO AVOID CARD BOUNDARY PROBLEMS */                     | 2328 CHAR
675 |       CP = CP + 1;                                                               | 2328 CHAR
676 |       IF CP <= TEXT_LIMIT THEN RETURN;                                           | 2348 CHAR
677 |       CALL GET_CARD;                                                             | 2360 CHAR
678 |    END CHAR;                                                                     | 2370 CHAR
679 |                                                                                  | 2376
680 |                                                                                  | 2376
681 |SCAN:                                                                             | 2376
682 |    PROCEDURE;                                                                    | 2376
683 |       DECLARE (S1, S2) FIXED;                                                    | 2376 SCAN
684 |       CALLCOUNT(3) = CALLCOUNT(3) + 1;                                           | 2384 SCAN
685 |       FAILSOFT = TRUE;                                                           | 2412 SCAN
686 |       BCD = '';  NUMBER_VALUE = 0;                                               | 2420 SCAN.
687 |    SCAN1:                                                                        | 2432 SCAN
688 |       DO FOREVER;                                                                | 2432 SCAN
689 |          IF CP > TEXT_LIMIT THEN CALL GET_CARD;                                  | 2432 SCAN
690 |          ELSE                                                                    | 2448 SCAN
691 |             DO; /* DISCARD LAST SCANNED VALUE */                                | 2448 SCAN
692 |                TEXT_LIMIT = TEXT_LIMIT - CP;                                     | 2452 SCAN
693 |                TEXT = SUBSTR(TEXT, CP);                                          | 2464 SCAN
694 |                CP = 0;                                                           | 2490 SCAN
695 |             END;                                                                 | 2496 SCAN
696 |          /*  BRANCH ON NEXT CHARACTER IN TEXT                    */             | 2496 SCAN
697 |          DO CASE CHARTYPE(BYTE(TEXT));                                          | 2496 SCAN
698 |                                                                                  | 2528 SCAN
699 |             /*  CASE 0  */                                                       | 2528 SCAN
700 |                                                                                  | 2528 SCAN
701 |             /* ILLEGAL CHARACTERS FALL HERE  */                                | 2528 SCAN
702 |             CALL ERROR ('ILLEGAL CHARACTER: ' || SUBSTR(TEXT, 0, 1));           | 2528 SCAN
703 |                                                                                  | 2564 SCAN CASE 0.
704 |             /*  CASE 1  */                                                       | 2564 SCAN
705 |                                                                                  | 2564 SCAN
706 |             /*  BLANK  */                                                       | 2564 SCAN
707 |             DO;                                                                  | 2564 SCAN
708 |                CP = 1;                                                           | 2568 SCAN CASE 1.
709 |                DO WHILE BYTE(TEXT, CP) = BYTE(' ') & CP <= TEXT_LIMIT;           | 2576 SCAN
710 |                   CP = CP + 1;                                                   | 2634 SCAN C8 = 64
711 |                END;                                                              | 2646 SCAN
712 |                CP = CP - 1;                                                      | 2650 SCAN
713 |             END;                                                                 | 2662 SCAN
714 |                                                                                  | 2662 SCAN
715 |             /*  CASE 2  */                                                       | 2662 SCAN
716 |                                                                                  | 2662 SCAN
717 |          ;  /*  NOT USED IN SKELETON (BUT USED IN XCOM)  */                     | 2662 SCAN
718 |                                                                                  | 2666 SCAN CASE 2.
```

```
719 |           /*  CASE 3  */                                                    |  2658 SCAN
720 |                                                                             |  2666 SCAN
721 |       ;   /*  NOT USED IN SKELETON (BUT USED IN XCOM)  */                  |  2666 SCAN
722 |                                                                             |  2670 SCAN CASE 3
723 |           /*  CASE 4  */                                                    |  2670 SCAN
724 |                                                                             |  2670 SCAN
725 |           DO FOREVER;  /* A LETTER:  IDENTIFIERS AND RESERVED WORDS */     |  2670 SCAN
726 |              DO CP = CP + 1 TO TEXT_LIMIT;                                  |  2674 SCAN CASE 4
727 |                 IF NOT_LETTER_OR_DIGIT(BYTE(TEXT, CP)) THEN                 |  2714 SCAN
728 |                    DO;  /* END OF IDENTIFIER  */                            |  2726 SCAN
729 |                       IF CP > 0 THEN BCD = BCD || SUBSTR(TEXT, 0, CP);      |  2736 SCAN
730 |                       S1 = LENGTH(BCD);                                     |  2790 SCAN
731 |                       IF S1 > 1 THEN IF S1 <= RESERVED_LIMIT THEN           |  2812 SCAN
732 |                          /* CHECK FOR RESERVED WORDS */                     |  2844 SCAN
733 |                          DO I = V_INDEX(S1-1) TO V_INDEX(S1) - 1;           |  2844 SCAN
734 |                             IF BCD = V(I) THEN                              |  2892 SCAN
735 |                                DO;                                          |  2942 SCAN
736 |                                   TOKEN = I;                                |  2934 SCAN
737 |                                   RETURN;                                   |  2942 SCAN
738 |                                END;                                         |  2948 SCAN
739 |                          END;                                               |  2948 SCAN
740 |                       /*  RESERVED WORDS EXIT HIGHER: THEREFORE <IDENTIFIER>*/|  2952 SCAN
741 |                       TOKEN = IDENT;                                        |  2952 SCAN
742 |                       RETURN;                                               |  2960 SCAN
743 |                    END;                                                     |  2966 SCAN
744 |              END;                                                           |  2966 SCAN
745 |              /*  END OF CARD  */                                           |  2970 SCAN
746 |              BCD = BCD || TEXT;                                             |  2970 SCAN
747 |              CALL GET_CARD;                                                 |  2994 SCAN
    |              CP = -1;                                                       |  2998 SCAN
    |           END;                                                              |  3006 SCAN
750 |                                                                             |  3010 SCAN
751 |                                                                             |  3010 SCAN
752 |           /*  CASE 5  */                                                    |  3010 SCAN
753 |                                                                             |  3010 SCAN
754 |           DO;       /*  DIGIT:  A NUMBER  */                              |  3010 SCAN
755 |              TOKEN = NUMBER;                                                |  3014 SCAN CASE 5
756 |              DO FOREVER;                                                    |  3022 SCAN
757 |                 DO CP = CP TO TEXT_LIMIT;                                   |  3022 SCAN
758 |                    S1 = BYTE(TEXT, CP);                                     |  3058 SCAN
759 |                    IF S1 < "F0" THEN RETURN;                                |  3074 SCAN
760 |                    NUMBER_VALUE = 10*NUMBER_VALUE + S1 - "F0";              |  3092 SCAN
761 |                 END;                                                        |  3112 SCAN
762 |                 CALL GET_CARD;                                              |  3116 SCAN
763 |              END;                                                           |  3120 SCAN
764 |           END;                                                              |  3124 SCAN
765 |                                                                             |  3124 SCAN
766 |           /*  CASE 6  */                                                    |  3124 SCAN
767 |                                                                             |  3124 SCAN
768 |           DO;       /*  A /:  MAY BE DIVIDE OR START OF COMMENT  */       |  3124 SCAN
769 |              CALL CHAR;                                                     |  3128 SCAN CASE 6
770 |              IF BYTE(TEXT, CP) ¬= BYTE('*') THEN                            |  3132 SCAN
771 |                 DO;                                                         |  3160 SCAN
772 |                    TOKEN = DIVIDE;                                          |  3152 SCAN
773 |                    RETURN;                                                  |  3160 SCAN
774 |                 END;                                                        |  3166 SCAN
775 |              /* WE HAVE A COMMENT  */                                      |  3166 SCAN
776 |              S1, S2 = BYTE(' ');                                            |  3166 SCAN
777 |              DO WHILE S1 ¬= BYTE('*') | S2 ¬= BYTE('/');                    |  3178 SCAN
778 |                 IF S1 = BYTE('$') THEN                                      |  3228 SCAN
779 |                    DO;  /* A CONTROL CHARACTER  */                         |  3248 SCAN
780 |                       CONTROL(S2) = ¬ CONTROL(S2);                          |  3240 SCAN
781 |                       IF S2 = BYTE('T') THEN CALL TRACE;                    |  3262 SCAN
782 |                       ELSE IF S2 = BYTE('U') THEN CALL UNTRACE;             |  3268 SCAN
783 |                       ELSE IF S2 = BYTE(']') THEN                           |  3318 SCAN
784 |                          IF CONTROL(S2) THEN                                |  3342 SCAN
```

```
                                MARGIN_CHOP = TEXT_LIMIT - CP + 1;                  3356 SCAN
                             ELSE                                                   3364 SCAN
                                MARGIN_CHOP = 0;                                    3364 SCAN
                          END;                                                      3374 SCAN
                       S1 = S2;                                                     3374 SCAN
                       CALL CHAR;                                                   3382 SCAN
                       S2 = BYTE(TEXT, CP);                                         3386 SCAN
                    END;                                                            3402 SCAN
                 END;                                                               3406 SCAN
                                                                                    3406 SCAN
                 /*  CASE 7  */                                                     3406 SCAN
                 DO;       /*  SPECIAL CHARACTERS  */                             3406 SCAN
                    TOKEN = TX(BYTE(TEXT));                                         3410 SCAN CASE 7
                    CP = 1;                                                         3430 SCAN
                    RETURN;                                                         3438 SCAN
                 END;                                                               3444 SCAN
                                                                                    3444 SCAN
                 /*  CASE 8  */                                                     3444 SCAN
              ;  /*  NOT USED IN SKELETON (BUT USED IN XCOM)  */                    3444 SCAN
                                                                                    3448 SCAN CASE 8
              END;      /* OF CASE ON CHARTYPE  */                                  3448 SCAN
              CP = CP + 1;  /* ADVANCE SCANNER AND RESUME SEARCH FOR TOKEN  */      3452 SCAN
           END;                                                                     3464 SCAN
        END SCAN;                                                                   3468 SCAN
                                                                                    3474
                                                                                    3474
                                                                                    3474
                                                                                    3474
                                                                                    3474
/*                          TIME AND DATE                                    */    3474
                                                                                    3474
                                                                                    3474
PRINT_TIME:                                                                         3474
   PROCEDURE (MESSAGE, T);                                                          3474
      DECLARE MESSAGE CHARACTER, T FIXED;                                           3474 PRINT_TIME
      MESSAGE = MESSAGE || T/360000 || ':' || T MOD 360000 / 6000 || ':'            3482 PRINT_TIME
         || T MOD 6000 / 100 || '.';                                                3580 PRINT_TIME
      T = T MOD 100;  /* DECIMAL FRACTION  */                                       3662 PRINT_TIME
      IF T < 10 THEN MESSAGE = MESSAGE || '0';                                      3680 PRINT_TIME
      OUTPUT = MESSAGE || T || '.';                                                 3716 PRINT_TIME
   END PRINT_TIME;                                                                  3773 PRINT_TIME
                                                                                    3784
PRINT_DATE_AND_TIME:                                                                3784
   PROCEDURE (MESSAGE, D, T);                                                       3784
      DECLARE MESSAGE CHARACTER, (D, T, YEAR, DAY, M) FIXED;                        3784 PRINT_DATE.
      DECLARE MONTH(11) CHARACTER INITIAL ('JANUARY', 'FEBRUARY', 'MARCH',          3792 PRINT_DATE.
         'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST', 'SEPTEMBER', 'OCTOBER',          3792 PRINT_DATE.
         'NOVEMBER', 'DECEMBER'),                                                   3792 PRINT_DATE.
      DAYS(12) FIXED INITIAL (0, 31, 60, 91, 121, 152, 182, 213, 244, 274,          3792 PRINT_DATE.
         305, 335, 366);                                                            3792 PRINT_DATE.
      YEAR = D/1000 + 1900;                                                         3792 PRINT_DATE.
      DAY = D MOD 1000;                                                             3812 PRINT_DATE.
      IF (YEAR & "3") ¬= 0 THEN IF DAY > 59 THEN DAY = DAY + 1; /*   LEAP YEAR*/   3830 PRINT_DATE.
      M = 1;                                                                        3870 PRINT_DATE.
      DO WHILE DAY > DAYS(M);  M = M + 1;  END;                                     3878 PRINT_DATE.
      CALL PRINT_TIME(MESSAGE || MONTH(M-1) || X1 || DAY-DAYS(M-1) ||  ', '         3914 PRINT_DATE.
         || YEAR || '.  CLOCK TIME = ', T);                                         4002 PRINT_DATE.
   END PRINT_DATE_AND_TIME;                                                         4074 PRINT_DATE.
                                                                                    4080
/*                          INITIALIZATION                                   */    4080
                                                                                    4080
                                                                                    4080
                                                                                    4080
INITIALIZATION:                                                                     4080
   PROCEDURE;                                                                       4080
      EJECT PAGE;                                                                   4080 INITIALIZA
```

```
   |   CALL PRINT_DATE_AND_TIME ('  PL/S COMPILER FOR THE 360/370 - 05012 - BY RT | 4112 INITIALIZA
852 |WEST AND OF WILLIS ON ',DATE_OF_GENERATION,TIME_OF_GENERATION);               | 4112 INITIALIZA
853 |      DOUBLE_SPACE;                                                           | 4140 INITIALIZA
854 |      CALL PRINT_DATE_AND_TIME ('TODAY IS ', DATE, TIME);                     | 4164 INITIALIZA
855 |      DOUBLE_SPACE;                                                           | 4214 INITIALIZA
856 |      DO II=1 TO MAXREG;                                  /*NEWQUAD*/       | 4238 INITIALIZA
857 |         STATUS(II) = FALSE;                              /*NEWQUAD*/       | 4282 INITIALIZA
858 |         ADDRSS(III=FALSE;                                /*NEWQUAD*/       | 4292 INITIALIZA
859 |         END;                                             /*NEWQUAD*/       | 4302 INITIALIZA
860 |      DO I = 1 TO NT;                                                         | 4310 INITIALIZA
861 |         S = V(I);                                                            | 4346 INITIALIZA
862 |         IF S = '<NUMBER>' THEN NUMBER = I;  ELSE                             | 4362 INITIALIZA
863 |         IF S = '<IDENTIFIER>' THEN IDENT = I;  ELSE                          | 4412 INITIALIZA
864 |         IF S = '/' THEN DIVIDE = I;  ELSE                                    | 4470 INITIALIZA
865 |         IF S = '_|_' THEN EOFILE = I;  ELSE                                  | 4528 INITIALIZA
866 |         IF S = ';' THEN STOPIT(I) = TRUE;  ELSE                              | 4586 INITIALIZA
867 |         ;                                                                    | 4648 INITIALIZA
868 |      END;                                                                    | 4656 INITIALIZA
869 |      IF IDENT = NT THEN RESERVED_LIMIT = LENGTH(V(NT-1));                    | 4664 INITIALIZA
870 |      ELSE RESERVED_LIMIT = LENGTH(V(NT));                                    | 4718 INITIALIZA
871 |      V(EOFILE) = 'EOF';                                                      | 4760 INITIALIZA
872 |      STOPIT(EOFILE) = TRUE;                                                  | 4776 INITIALIZA
873 |      CHARTYPE(BYTE(' ')) = 1;                                                | 4796 INITIALIZA
874 |      DO I = 0 TO 255;                                                        | 4800 INITIALIZA
875 |         NOT_LETTER_OR_DIGIT(I) = TRUE;                                       | 4834 INITIALIZA
876 |      END;                                                                    | 4846 INITIALIZA
877 |      DO I = 0 TO LENGTH(ALPHABET) - 1;                                       | 4854 INITIALIZA
878 |         J = BYTE(ALPHABET, I);                                               | 4916 INITIALIZA
879 |         TX(J) = I;                                                           | 4934 INITIALIZA
880 |         NOT_LETTER_OR_DIGIT(J) = FALSE;                                      | 4946 INITIALIZA
881 |         CHARTYPE(J) = 4;                                                     | 4956 INITIALIZA
882 |      END;                                                                    | 4968 INITIALIZA
883 |      DO I = 0 TO 9;                                                          | 4976 INITIALIZA
884 |         J = BYTE('0123456789', I);                                           | 5010 INITIALIZA
885 |         NOT_LETTER_OR_DIGIT(J) = FALSE;                                      | 5026 INITIALIZA
886 |         CHARTYPE(J) = 5;                                                     | 5036 INITIALIZA
887 |      END;                                                                    | 5048 INITIALIZA
888 |      DO I = V_INDEX(0) TO V_INDEX(1) - 1;                                    | 5056 INITIALIZA
889 |         J = BYTE(V(I));                                                      | 5114 INITIALIZA
890 |         TX(J) = I;                                                           | 5136 INITIALIZA
891 |         CHARTYPE(J) = 7;                                                     | 5148 INITIALIZA
892 |      END;                                                                    | 5160 INITIALIZA
893 |      CHARTYPE(BYTE('/')) = 6;                                                | 5168 INITIALIZA
894 |      /* FIRST SET UP GLOBAL VARIABLES CONTROLLING SCAN, THEN CALL IT */      | 5180 INITIALIZA
895 |      CP = 0;  TEXT_LIMIT = -1;                                               | 5180 INITIALIZA
896 |      TEXT = '';                                                              | 5194 INITIALIZA
897 |      CONTROL(BYTE('L')) = TRUE;                                              | 5200 INITIALIZA
898 |      CALL SCAN;                                                              | 5212 INITIALIZA
899 |                                                                              | 5216 INITIALIZA
900 |      /* INITIALIZE THE PARSE_STACK */                                       | 5216 INITIALIZA
901 |      SP = 1;  PARSE_STACK(SP) = EOFILE;                                      | 5216 INITIALIZA
902 |                                                                              | 5236 INITIALIZA
903 |   END INITIALIZATION;                                                        | 5236 INITIALIZA
904 |                                                                              | 5242
905 |                                                                              | 5242
906 |                                                                              | 5242
907 |                                                                              | 5242
908 |                                                                              | 5242
909 |                                                                              | 5242
910 |DUMPIT:                                                                       | 5242
911 |   PROCEDURE;    /* DUMP OUT THE STATISTICS COLLECTED DURING THIS RUN  */     | 5242
912 |      DOUBLE_SPACE;                                                           | 5242 DUMPIT
913 |      /*  PUT OUT THE ENTRY COUNT FOR IMPORTANT PROCEDURES */                 | 5276 DUMPIT
914 |                                                                              | 5278 DUMPIT
915 |      OUTPUT = 'STACKING DECISIONS= ' || CALLCOUNT(1);                        | 5278 DUMPIT
916 |      OUTPUT = 'SCAN              = ' || CALLCOUNT(3);                        | 5332 DUMPIT
```

```
      OUTPUT = 'FREE STRING AREA  = ' || FREELIMIT - FREEBASE;                    5436 DUMPIT
   END DUMPIT;                                                                    5436 DUMPIT
                                                                                  5442
                                                                                  5442
STACK_DUMP:                                                                       5442
   PROCEDURE;                                                                     5442
      DECLARE LINE CHARACTER;                                                     5442 STACK_DUM
      LINE = 'PARTIAL PARSE TO THIS POINT IS: ';                                  5454 STACK_DUM
      DO I = 2 TO SP;                                                             5462 STACK_DUM
         IF LENGTH(LINE) > 105 THEN                                               5506 STACK_DUM
            DO;                                                                   5548 STACK_DUM
               OUTPUT = LINE;                                                     5540 STACK_DUM
               LINE = X4;                                                         5560 STACK_DUM
            END;                                                                  5568 STACK_DUM
         LINE = LINE || X1 || V(PARSE_STACK(I));                                  5568 STACK_DUM
      END;                                                                        5622 STACK_DUM
      OUTPUT = LINE;                                                              5630 STACK_DUM
   END STACK_DUMP;                                                                5650 STACK_DUM
                                                                                  5656
                                                                                  5656
SET_BIT:PROCEDURE(LOC);                                    /*NEWQUAD*/          5656
                                                                                  5656 SET_BIT
/*SET_BIT  SET BIT AT LOCATION LOC OF MICROWORD  */                               5656 SET_BIT
                                                                                  5656 SET_BIT
 DECLARE LOC FIXED;                                        /*NEWQUAD*/          5656 SET_BIT
 BYTE(BUF_MIC,LOC)="F1";                                   /*NEWQUAD*/          5668 SET_BIT
 END;                                                      /*NEWQUAD*/          5684 SET_BIT
SET_FIELD:PROCEDURE(VALUE,LOC1,LOC2);                      /*NEWQUAD*/          5690
                                                                                  5690 SET_FIEL
/*SET_FIELD  SET BITS AT LOCATION LOC1 TO LOC2 TO THE GIVEN VALUE */              5690 SET_FIEL
                                                                                  5690 SET_FIEL
 DECLARE (VALUE,LOC1,LOC2,A,B) FIXED;                      /*NEWQUAD*/          5690 SET_FIEL
 A=0;                                                      /*NEWQUAD*/          5702 SET_FIEL
 DO B=LOC1 TO LOC2;                                        /*NEWQUAD*/          5708 SET_FIEL
    IF VALUE MOD 2 = 1 THEN DO;                            /*NEWQUAD*/          5752 SET_FIEL
       BYTE(BUF_MIC,LOC2-A)="F1";                          /*NEWQUAD*/          5778 SET_FIEL
       END;                                                /*NEWQUAD*/          5798 SET_FIEL
    A=A+1;                                                 /*NEWQUAD*/          5798 SET_FIEL
    VALUE=VALUE/2;                                         /*NEWQUAD*/          5810 SET_FIEL
    END;                                                   /*NEWQUAD*/          5826 SET_FIEL
 END;                                                      /*NEWQUAD*/          5834 SET_FIEL
PUT_MIC:PROCEDURE(LOCATION);                               /*NEWQUAD*/          5840
                                                                                  5840 PUT_MIC
/*PUT_MIC  PRINT OUT MICROWORD AND REINITIALIZE IT TO ALL ZERO */                 5840 PUT_MIC
                                                                                  5840 PUT_MIC
 DECLARE BLK(16) FIXED INITIAL                             /*NEWQUAD*/          5840 PUT_MIC
 (3,5,8,10,12,14,18,23,27,33,38,43,46,48,54,59,64);        /*NEWQUAD*/          5852 PUT_MIC
 DECLARE (LOCATION,A,B) FIXED;                             /*NEWQUAD*/          5852 PUT_MIC
 DECLARE ADDRESS CHARACTER;                                /*NEWQUAD*/          5852 PUT_MIC
 ADDRESS='  ';                                             /*NEWQUAD*/          5852 PUT_MIC
 DO A=1 TO 3;                                              /*NEWQUAD*/          5860 PUT_MIC
    B=LOCATION MOD 8;                                      /*NEWQUAD*/          5896 PUT_MIC
    ADDRESS=B||ADDRESS;                                    /*NEWQUAD*/          5914 PUT_MIC
    LOCATION=LOCATION/8;                                   /*NEWQUAD*/          5946 PUT_MIC
    END;                                                   /*NEWQUAD*/          5962 PUT_MIC
 OUTPUT(3)='  '||ADDRESS||BUF_MIC;                         /*NEWQUAD*/          5970 PUT_MIC
 DO A=0 TO 72;                                             /*NEWQUAD*/          6044 PUT_MIC
    BYTE(BUF_MIC,A)="F0";                                  /*NEWQUAD*/          6078 PUT_MIC
    END;                                                   /*NEWQUAD*/          6094 PUT_MIC
 DO A=0 TO 16;                                             /*NEWQUAD*/          6102 PUT_MIC
    BYTE(BUF_MIC,BLK(A))="40";                             /*NEWQUAD*/          6136 PUT_MIC
    END;                                                   /*NEWQUAD*/          6160 PUT_MIC
 END;                                                      /*NEWQUAD*/          6168 PUT_MIC
OUT_MIC:PROCEDURE;                                         /*NEWQUAD*/          6174
                                                                                  6174 OUT_MIC
/*OUT_MIC  SET NEXT ADDRESS FIELD OF MICROWORD */                                 6174 OUT_MIC
```

```
 983 |                                                                                                  |  6174 OUT_MIC
 984 |  IF BR_FLAG=TRUE & NEWQUAD(II+4)=BR THEN DO;                                /*NEWQUAD*/ |  6174 OUT_MIC
 985 |     LL=NEWQUAD(II+7);                                                       /*NEWQUAD*/ |  6262 OUT_MIC
 986 |     CALL SET_FIELD(MIC_LOC(LL),65,72);                                      /*NEWQUAD*/ |  6282 OUT_MIC
 987 |     END;                                                                    /*NEWQUAD*/ |  6322 OUT_MIC
 988 |  ELSE                                                                       /*NEWQUAD*/ |  6322 OUT_MIC
 989 |     CALL SET_FIELD(CURR_MIC+1,65,72);                                       /*NEWQUAD*/ |  6322 OUT_MIC
 990 |  BR_FLAG=FALSE;                                                             /*NEWQUAD*/ |  6366 OUT_MIC
 991 |  CALL PUT_MIC(CURR_MIC);                                                    /*NEWQUAD*/ |  6372 OUT_MIC
 992 |  CURR_MIC=CURR_MIC+1;                                                       /*NEWQUAD*/ |  6388 OUT_MIC
 993 |  END;                                                                       /*NEWQUAD*/ |  6400 OUT_MIC
 994 | RO_PLUS2_READ:PROCEDURE;                                                    /*NEWQUAD*/ |  6406
 995 |                                                                                          |  6406 RO_PLUS2_REA
 996 | /*RO_PLUS2_READ   GENERATE A MICROWORD TO INITIATE AREAD OPERATION                       |  6406 RO_PLUS2_REA
 997 |                    AND UODATE RO BY 2   */                                               |  6406 RO_PLUS2_REA
 998 |                                                                                          |  6406 RO_PLUS2_REA
 999 |  CALL SET_BIT(0);                                                           /*NEWQUAD*/ |  6406 RO_PLUS2_REA
1000 |  CALL SET_BIT(1);                                                           /*NEWQUAD*/ |  6432 RO_PLUS2_REA
1001 |  CALL SET_BIT(2);                                                           /*NEWQUAD*/ |  6448 RO_PLUS2_REA
1002 |  CALL SET_BIT(6);                                                           /*NEWQUAD*/ |  6464 RO_PLUS2_REA
1003 |  CALL SET_BIT(7);                                                           /*NEWQUAD*/ |  6480 RO_PLUS2_REA
1004 |  CALL SET_BIT(11);                                                          /*NEWQUAD*/ |  6496 RO_PLUS2_REA
1005 |  CALL SET_BIT(13);                                                          /*NEWQUAD*/ |  6512 RO_PLUS2_REA
1006 |  CALL SET_BIT(17);                                                          /*NEWQUAD*/ |  6528 RO_PLUS2_REA
1007 |  CALL SET_BIT(29);                                                          /*NEWQUAD*/ |  6544 RO_PLUS2_REA
1008 |  CALL SET_BIT(32);                                                          /*NEWQUAD*/ |  6560 RO_PLUS2_REA
1009 |  CALL SET_BIT(36);                                                          /*NEWQUAD*/ |  6576 RO_PLUS2_REA
1010 |  CALL SET_BIT(39);                                                          /*NEWQUAD*/ |  6592 RO_PLUS2_REA
1011 |  CALL SET_BIT(40);                                                          /*NEWQUAD*/ |  6608 RO_PLUS2_REA
1012 |  CALL SET_BIT(41);                                                          /*NEWQUAD*/ |  6624 RO_PLUS2_REA
1013 |  CALL SET_BIT(42);                                                          /*NEWQUAD*/ |  6640 RO_PLUS2_REA
1014 |  CALL SET_BIT(44);                                                          /*NEWQUAD*/ |  6656 RO_PLUS2_REA
1015 |  CALL SET_BIT(47);                                                          /*NEWQUAD*/ |  6672 RO_PLUS2_REA
1016 |  CALL SET_BIT(58);                                                          /*NEWQUAD*/ |  6688 RO_PLUS2_REA
1017 |  CALL OUT_MIC;                                                              /*NEWQUAD*/ |  6704 RO_PLUS2_REA
1018 |  END;                                                                       /*NEWQUAD*/ |  6712 RO_PLUS2_REA
1019 | R3_UNIBUS:PROCEDURE;                                                        /*NEWQUAD*/ |  6718
1020 |                                                                                          |  6718 R3_UNIBUS
1021 | /*R3_UNIBUS   GENERATE A MICROWORD TO CLOCK UNIBUS RESULT REGISTE */                      |  6718 R3_UNIBUS
1022 |                                                                                          |  6718 R3_UNIBUS
1023 |  CALL SET_BIT(1);                                                           /*NEWQUAD*/ |  6718 R3_UNIBUS
1024 |  CALL SET_BIT(6);                                                           /*NEWQUAD*/ |  6746 R3_UNIBUS
1025 |  CALL SET_BIT(7);                                                           /*NEWQUAD*/ |  6762 R3_UNIBUS
1026 |  CALL SET_BIT(45);                                                          /*NEWQUAD*/ |  6778 R3_UNIBUS
1027 |  CALL SET_BIT(58);                                                          /*NEWQUAD*/ |  6794 R3_UNIBUS
1028 |  CALL SET_FIELD(NEWQUAD(II+3),60,63);                                       /*NEWQUAD*/ |  6810 R3_UNIBUS
1029 |  CALL OUT_MIC;                                                              /*NEWQUAD*/ |  6854 R3_UNIBUS
1030 |  END;                                                                       /*NEWQUAD*/ |  6862 R3_UNIBUS
1031 | RTEMP_UNIBUS:PROCEDURE;                                                     /*NEWQUAD*/ |  6868
1032 |                                                                                          |  6868 RTEMP_UNIBUS
1033 | /*RTEMP_UNIBUS  GENERATE A MICROWORD TO CLOCK UNIBUS TO RTEMP  */                        |  6868 RTEMP_UNIBUS
1034 |                                                                                          |  6868 RTEMP_UNIBUS
1035 |  CALL SET_BIT(1);                                                           /*NEWQUAD*/ |  6868 RTEMP_UNIBUS
1036 |  CALL SET_BIT(6);                                                           /*NEWQUAD*/ |  6896 RTEMP_UNIBUS
1037 |  CALL SET_BIT(7);                                                           /*NEWQUAD*/ |  6912 RTEMP_UNIBUS
1038 |  CALL SET_BIT(45);                                                          /*NEWQUAD*/ |  6928 RTEMP_UNIBUS
1039 |  CALL SET_BIT(59);                                                          /*NEWQUAD*/ |  6944 RTEMP_UNIBUS
1040 |  CALL SET_BIT(60);                                                          /*NEWQUAD*/ |  6960 RTEMP_UNIBUS
1041 |  CALL OUT_MIC;                                                              /*NEWQUAD*/ |  6976 RTEMP_UNIBUS
1042 |  END;                                                                       /*NEWQUAD*/ |  6984 RTEMP_UNIBUS
1043 | BA_RO_READ:PROCEDURE;                                                       /*NEWQUAD*/ |  6990
1044 |                                                                                          |  6990 BA_RO_READ
1045 |  /* BA_RO_READ  GENERATE A MICROWORD TO CLOCK RO TO BA REGISTER                          |  6990 BA_RO_READ
1046 |                  AND INITIATE A READ OPERATION  */                                       |  6990 BA_RO_READ
1047 |                                                                                          |  6990 BA_RO_READ
1048 |  CALL SET_BIT(1);                                                           /*NEWQUAD*/ |  6990 BA_RO_READ
```

```
1049 |  CALL SET_BIT(2);                                                    /*NEWQUAD*/ |  7018 BA_RO_READ
1050 |  CALL SET_BIT(13);                                                   /*NEWQUAD*/ |  7034 BA_RO_READ
1051 |  CALL SET_BIT(17);                                                   /*NEWQUAD*/ |  7050 BA_RO_READ
1052 |  CALL SET_BIT(47);                                                   /*NEWQUAD*/ |  7066 BA_RO_READ
1053 |  CALL SET_BIT(58);                                                   /*NEWQUAD*/ |  7082 BA_RO_READ
1054 |  CALL OUT_MIC;                                                       /*NEWQUAD*/ |  7098 BA_RO_READ
1055 |  END;                                                                /*NEWQUAD*/ |  7106 BA_RO_READ
1056 | R3_D:PROCEDURE;                                                      /*NEWQUAD*/ |  7112
1057 |                                                                                |  7112 R3_D
1058 | /*R3_D  GENERATE A MICROWORD TO CLOCK D REGISTER TO RESULT REGISTER   */        |  7112 R3_D
1059 |                                                                                |  7112 R3_D
1060 |  CALL SET_BIT(1);                                                    /*NEWQUAD*/ |  7112 R3_D
1061 |  CALL SET_BIT(6);                                                    /*NEWQUAD*/ |  7140 R3_D
1062 |  CALL SET_BIT(7);                                                    /*NEWQUAD*/ |  7156 R3_D
1063 |  CALL SET_BIT(44);                                                   /*NEWQUAD*/ |  7172 R3_D
1064 |  CALL SET_BIT(58);                                                   /*NEWQUAD*/ |  7188 R3_D
1065 |  CALL SET_FIELD(NEWQUAD(II+3),60,63);                                /*NEWQUAD*/ |  7204 R3_D
1066 |  CALL OUT_MIC;                                                       /*NEWQUAD*/ |  7248 R3_D
1067 |  END;                                                                /*NEWQUAD*/ |  7256 R3_D
1068 | B_R2:PROCEDURE;                                                      /*NEWQUAD*/ |  7262
1069 |                                                                                |  7262 B_R2
1070 | /*B_R2  GENERATE A MICRO WORD TO CLOCK OPERAND2 REGISTER TO B REGISTER */       |  7262 B_R2
1071 |                                                                                |  7262 B_R2
1072 |  CALL SET_BIT(1);                                                    /*NEWQUAD*/ |  7262 B_R2
1073 |  CALL SET_BIT(9);                                                    /*NEWQUAD*/ |  7290 B_R2
1074 |  CALL SET_BIT(58);                                                   /*NEWQUAD*/ |  7306 B_R2
1075 |  CALL SET_FIELD(NEWQUAD(II+2),60,63);                                /*NEWQUAD*/ |  7322 B_R2
1076 |  CALL OUT_MIC;                                                       /*NEWQUAD*/ |  7366 B_R2
1077 |  END;                                                                /*NEWQUAD*/ |  7374 B_R2
1078 | RTEMP_4:PROCEDURE;                                                   /*NEWQUAD*/ |  7380
1079 |                                                                                |  7380 RTEMP_4
1080 | /*RTEMP_4   GENERATE A MICROWORD TO SET RTEMP TO 4    */                       |  7380 RTEMP_4
1081 |                                                                                |  7380 RTEMP_4
1082 |  CALL SET_BIT(0);                                                    /*NEWQUAD*/ |  7380 RTEMP_4
1083 |  CALL SET_BIT(1);                                                    /*NEWQUAD*/ |  7406 RTEMP_4
1084 |  CALL SET_BIT(6);                                                    /*NEWQUAD*/ |  7422 RTEMP_4
1085 |  CALL SET_BIT(7);                                                    /*NEWQUAD*/ |  7438 RTEMP_4
1086 |  CALL SET_BIT(11);                                                   /*NEWQUAD*/ |  7454 RTEMP_4
1087 |  CALL SET_BIT(28);                                                   /*NEWQUAD*/ |  7470 RTEMP_4
1088 |  CALL SET_BIT(29);                                                   /*NEWQUAD*/ |  7486 RTEMP_4
1089 |  CALL SET_BIT(31);                                                   /*NEWQUAD*/ |  7502 RTEMP_4
1090 |  CALL SET_BIT(34);                                                   /*NEWQUAD*/ |  7518 RTEMP_4
1091 |  CALL SET_BIT(35);                                                   /*NEWQUAD*/ |  7534 RTEMP_4
1092 |  CALL SET_BIT(36);                                                   /*NEWQUAD*/ |  7550 RTEMP_4
1093 |  CALL SET_BIT(37);                                                   /*NEWQUAD*/ |  7566 RTEMP_4
1094 |  CALL SET_BIT(39);                                                   /*NEWQUAD*/ |  7582 RTEMP_4
1095 |  CALL SET_BIT(40);                                                   /*NEWQUAD*/ |  7598 RTEMP_4
1096 |  CALL SET_BIT(41);                                                   /*NEWQUAD*/ |  7614 RTEMP_4
1097 |  CALL SET_BIT(42);                                                   /*NEWQUAD*/ |  7630 RTEMP_4
1098 |  CALL SET_BIT(44);                                                   /*NEWQUAD*/ |  7646 RTEMP_4
1099 |  CALL SET_BIT(58);                                                   /*NEWQUAD*/ |  7662 RTEMP_4
1100 |  CALL SET_BIT(60);                                                   /*NEWQUAD*/ |  7678 RTEMP_4
1101 |  CALL OUT_MIC;                                                       /*NEWQUAD*/ |  7694 RTEMP_4
1102 |  END;                                                                /*NEWQUAD*/ |  7702 RTEMP_4
1103 |  R3_0:PROCEDURE;                                                     /*NEWQUAD*/ |  7708
1104 |                                                                                |  7708 R3_0
1105 | /*R3_0   GENERATE A MICROWORD TO SET RESULT REGISTER TO ZERO   */              |  7708 R3_0
1106 |                                                                                |  7708 R3_0
1107 |  CALL SET_BIT(0);                                                    /*NEWQUAD*/ |  7708 R3_0
1108 |  CALL SET_BIT(1);                                                    /*NEWQUAD*/ |  7734 R3_0
1109 |  CALL SET_BIT(6);                                                    /*NEWQUAD*/ |  7750 R3_0
1110 |  CALL SET_BIT(7);                                                    /*NEWQUAD*/ |  7766 R3_0
1111 |  CALL SET_BIT(11);                                                   /*NEWQUAD*/ |  7782 R3_0
1112 |  CALL SET_BIT(31);                                                   /*NEWQUAD*/ |  7798 R3_0
1113 |  CALL SET_BIT(32);                                                   /*NEWQUAD*/ |  7814 R3_0
1114 |  CALL SET_BIT(44);                                                   /*NEWQUAD*/ |  7830 R3_0
```

```
1115 |  CALL SET_BIT(53);                                          /*NEWQUAD*/ |  7846 R3_0
1116 |  CALL SET_FIELD(NEWQUAD(II+3),60,63);                       /*NEWQUAD*/ |  7862 R3_0
1117 |  CALL OUT_MIC;                                              /*NEWQUAD*/ |  7906 R3_0
1118 |  END;                                                       /*NEWQUAD*/ |  7914 R3_0
1119 | RTEMP_SHIFT:PROCEDURE;                                      /*NEWQUAD*/ |  7920
1120 |                                                                       |  7920 RTEMP_SHIF
1121 | /*RTEMP_SHIFT  GENERATE A MICROWORD TO SHIFT RTEMP ONE POSITION TO LEFT */ |  7920 RTEMP_SHIF
1122 |                                                                       |  7920 RTEMP_SHIF
1123 |  CALL SET_BIT(0);                                           /*NEWQUAD*/ |  7920 RTEMP_SHIF
1124 |  CALL SET_BIT(1);                                           /*NEWQUAD*/ |  7946 RTEMP_SHIF
1125 |  CALL SET_BIT(6);                                           /*NEWQUAD*/ |  7962 RTEMP_SHIF
1126 |  CALL SET_BIT(7);                                           /*NEWQUAD*/ |  7978 RTEMP_SHIF
1127 |  CALL SET_BIT(11);                                          /*NEWQUAD*/ |  7994 RTEMP_SHIF
1128 |  CALL SET_BIT(29);                                          /*NEWQUAD*/ |  8010 RTEMP_SHIF
1129 |  CALL SET_BIT(30);                                          /*NEWQUAD*/ |  8026 RTEMP_SHIF
1130 |  CALL SET_BIT(44);                                          /*NEWQUAD*/ |  8042 RTEMP_SHIF
1131 |  CALL SET_BIT(59);                                          /*NEWQUAD*/ |  8058 RTEMP_SHIF
1132 |  CALL SET_BIT(60);                                          /*NEWQUAD*/ |  8074 RTEMP_SHIF
1133 |  CALL OUT_MIC;                                              /*NEWQUAD*/ |  8090 RTEMP_SHIF
1134 |  END;                                                       /*NEWQUAD*/ |  8098 RTEMP_SHIF
1135 | B_R1:PROCEDURE;                                             /*NEWQUAD*/ |  8104
1136 |                                                                       |  8104 B_R1
1137 | /*B_R1  GENERATE A MICROWORD TO SET B REGISTER T1 REGISTER  */         |  8104 B_R1
1138 |                                                                       |  8104 B_R1
1139 |  CALL SET_BIT(1);                                           /*NEWQUAD*/ |  8104 B_R1
1140 |  CALL SET_BIT(9);                                           /*NEWQUAD*/ |  8132 B_R1
1141 |  CALL SET_BIT(58);                                          /*NEWQUAD*/ |  8148 B_R1
1142 |  CALL SET_FIELD(NEWQUAD(II+1),60,63);                       /*NEWQUAD*/ |  8164 B_R1
1143 |  END;                                                       /*NEWQUAD*/ |  8208 B_R1
1144 | R2_SHIFT:PROCEDURE;                                         /*NEWQUAD*/ |  8214
1145 |                                                                       |  8214 R2_SHIFT
1146 | /*R2_SHIFT  GENERATE A MICROWORD TO SHIFT OPERAND2 REGISTER TO LEFT    |  8214 R2_SHIFT
1147 |             ONE POSITION     */                                       |  8214 R2_SHIFT
1148 |                                                                       |  8214 R2_SHIFT
1149 |  CALL SET_BIT(0);                                           /*NEWQUAD*/ |  8214 R2_SHIFT
1150 |  CALL SET_BIT(1);                                           /*NEWQUAD*/ |  8240 R2_SHIFT
1151 |  CALL SET_BIT(1);                                           /*NEWQUAD*/ |  8256 R2_SHIFT
1152 |  CALL SET_BIT(6);                                           /*NEWQUAD*/ |  8272 R2_SHIFT
1153 |  CALL SET_BIT(7);                                           /*NEWQUAD*/ |  8288 R2_SHIFT
1154 |  CALL SET_BIT(11);                                          /*NEWQUAD*/ |  8304 R2_SHIFT
1155 |  CALL SET_BIT(24);                                          /*NEWQUAD*/ |  8320 R2_SHIFT
1156 |  CALL SET_BIT(44);                                          /*NEWQUAD*/ |  8336 R2_SHIFT
1157 |  CALL SET_BIT(49);                                          /*NEWQUAD*/ |  8352 R2_SHIFT
1158 |  CALL SET_BIT(52);                                          /*NEWQUAD*/ |  8366 R2_SHIFT
1159 |  CALL SET_BIT(53);                                          /*NEWQUAD*/ |  8384 R2_SHIFT
1160 |  CALL SET_BIT(58);                                          /*NEWQUAD*/ |  8400 R2_SHIFT
1161 |  CALL SET_FIELD(NEWQUAD(II+2),60,63);                       /*NEWQUAD*/ |  8416 R2_SHIFT
1162 |  CALL OUT_MIC;                                              /*NEWQUAD*/ |  8460 R2_SHIFT
1163 |  END;                                                       /*NEWQUAD*/ |  8468 R2_SHIFT
1164 | NO_OP:PROCEDURE;                                            /*NEWQUAD*/ |  8474
1165 |                                                                       |  8474 NO_OP
1166 | /*NO_OP  GENERATE A NO_OP MICROWORD     */                           |  8474 NO_OP
1167 |                                                                       |  8474 NO_OP
1168 |  CALL SET_BIT(1);                                           /*NEWQUAD*/ |  8474 NO_OP
1169 |  CALL OUT_MIC;                                              /*NEWQUAD*/ |  8502 NO_OP
1170 |  END;                                                       /*NEWQUAD*/ |  8510 NO_OP
1171 | R3_R3_PLUS_B:PROCEDURE;                                     /*NEWQUAD*/ |  8516
1172 |                                                                       |  8516 R3_R3_PLUS_
1173 | /*R3_R3_PLUS_B  GENERATE A MICROWORD TO ADD RESULT REGISTER TO B REGISTER |  8516 R3_R3_PLUS_
1174 |                 AND PUT RESULT IN RESULT REGISTER   */                |  8516 R3_R3_PLUS_
1175 |                                                                       |  8516 R3_R3_PLUS_
1176 |  CALL SET_BIT(0);                                           /*NEWQUAD*/ |  8516 R3_R3_PLUS_
1177 |  CALL SET_BIT(6);                                           /*NEWQUAD*/ |  8542 R3_R3_PLUS_
1178 |  CALL SET_BIT(7);                                           /*NEWQUAD*/ |  8558 R3_R3_PLUS_
1179 |  CALL SET_BIT(11);                                          /*NEWQUAD*/ |  8574 R3_R3_PLUS_
1180 |  CALL SET_BIT(29);                                          /*NEWQUAD*/ |  8590 R3_R3_PLUS_
```

```
1181 |  CALL SET_BIT(32);                                                    /*NEWQUAD*/ |  8696 R3_R3_PLUS_B
1182 |  CALL SET_BIT(58);                                                    /*NEWQUAD*/ |  8622 R3_R3_PLUS_B
1183 |  CALL SET_FIELD(NEWQUAD(I+3),60,63);                                  /*NEWQUAD*/ |  8628 R3_R3_PLUS_B
1184 |  CALL OUT_MIC;                                                        /*NEWQUAD*/ |  8682 R3_R3_PLUS_B
1185 |  END;                                                                 /*NEWQUAD*/ |  8690 R3_R3_PLUS_B
1186 | RTEMP_MINUS_1:PROCEDURE;                                              /*NEWQUAD*/ |  8696
1187 |                                                                                 |  8696 RTEMP_MINUS_1
1188 | /*RTEMP_MINUS_1  GENERATE A MICROWORD TO SUBTRACT 1 FROM RTEMP        */         |  8696 RTEMP_MINUS_1
1189 |                                                                                 |  8696 RTEMP_MINUS_1
1190 |  CALL SET_BIT(0);                                                     /*NEWQUAD*/ |  8696 RTEMP_MINUS_1
1191 |  CALL SET_BIT(6);                                                     /*NEWQUAD*/ |  8722 RTEMP_MINUS_1
1192 |  CALL SET_BIT(7);                                                     /*NEWQUAD*/ |  8738 RTEMP_MINUS_1
1193 |  CALL SET_BIT(11);                                                    /*NEWQUAD*/ |  8754 RTEMP_MINUS_1
1194 |  CALL SET_BIT(26);                                                    /*NEWQUAD*/ |  8770 RTEMP_MINUS_1
1195 |  CALL SET_BIT(30);                                                    /*NEWQUAD*/ |  8786 RTEMP_MINUS_1
1196 |  CALL SET_BIT(31);                                                    /*NEWQUAD*/ |  8802 RTEMP_MINUS_1
1197 |  CALL SET_BIT(37);                                                    /*NEWQUAD*/ |  8818 RTEMP_MINUS_1
1198 |  CALL SET_BIT(39);                                                    /*NEWQUAD*/ |  8834 RTEMP_MINUS_1
1199 |  CALL SET_BIT(40);                                                    /*NEWQUAD*/ |  8850 RTEMP_MINUS_1
1200 |  CALL SET_BIT(41);                                                    /*NEWQUAD*/ |  8866 RTEMP_MINUS_1
1201 |  CALL SET_BIT(42);                                                    /*NEWQUAD*/ |  8882 RTEMP_MINUS_1
1202 |  CALL SET_BIT(44);                                                    /*NEWQUAD*/ |  8898 RTEMP_MINUS_1
1203 |  CALL SET_BIT(50);                                                    /*NEWQUAD*/ |  8914 RTEMP_MINUS_1
1204 |  CALL SET_BIT(52);                                                    /*NEWQUAD*/ |  8930 RTEMP_MINUS_1
1205 |  CALL SET_BIT(58);                                                    /*NEWQUAD*/ |  8946 RTEMP_MINUS_1
1206 |  CALL SET_BIT(60);                                                    /*NEWQUAD*/ |  8962 RTEMP_MINUS_1
1207 |  END;                                                                 /*NEWQUAD*/ |  8978 RTEMP_MINUS_1
1208 | MIC_GEN:PROCEDURE;                                                    /*NEWQUAD*/ |  8984
1209 |  DECLARE CASE_NUM(30) FIXED INITIAL (9,                               /*NEWQUAD*/ |  8984 MIC_GEN
1210 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1211 |                                     11,                               /*NEWQUAD*/ |  8996 MIC_GEN
1212 |                                      1,                               /*NEWQUAD*/ |  8996 MIC_GEN
1213 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1214 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1215 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1216 |                                     10,                               /*NEWQUAD*/ |  8996 MIC_GEN
1217 |                                      2,                               /*NEWQUAD*/ |  8996 MIC_GEN
1218 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1219 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1220 |                                      1,                               /*NEWQUAD*/ |  8996 MIC_GEN
1221 |                                      1,                               /*NEWQUAD*/ |  8996 MIC_GEN
1222 |                                      1,                               /*NEWQUAD*/ |  8996 MIC_GEN
1223 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1224 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1225 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1226 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1227 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1228 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1229 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1230 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1231 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1232 |                                      0,                               /*NEWQUAD*/ |  8996 MIC_GEN
1233 |                                      8,                               /*NEWQUAD*/ |  8996 MIC_GEN
1234 |                                      4,                               /*NEWQUAD*/ |  8996 MIC_GEN
1235 |                                      5,                               /*NEWQUAD*/ |  8996 MIC_GEN
1236 |                                      6,                               /*NEWQUAD*/ |  8996 MIC_GEN
1237 |                                      7,                               /*NEWQUAD*/ |  8996 MIC_GEN
1238 |                                      3,                               /*NEWQUAD*/ |  8996 MIC_GEN
1239 |                                      0);                              /*NEWQUAD*/ |  8996 MIC_GEN
1240 |  DECLARE TEMP_CURR_MIC FIXED;                                         /*NEWQUAD*/ |  8996 MIC_GEN
1241 |  DO CASE CASE_NUM(OPER);                                              /*NEWQUAD*/ |  8996 MIC_GEN
1242 |     DO;                                                               /*NEWQUAD*/ |  9028 MIC_GEN
1243 |                                                                                 |  9028 MIC_GEN CASE
1244 | /*GENERATE MICROCODE FOR ADD REGISTER QUAD  */                                  |  9028 MIC_GEN
1245 |                                                                                 |  9028 MIC_GEN
1246 |        CALL R_R2;                                                     /*NEWQUAD*/ |  9028 MIC_GEN
```

```
1247 |        CALL SET_BIT(3);                                          /*NEWQUAD*/ |  9040 MIC_GEN
1248 |        CALL SET_BIT(11);                                         /*NEWQUAD*/ |  9050 MIC_GEN
1249 |        CALL SET_BIT(29);                                         /*NEWQUAD*/ |  9066 MIC_GEN
1250 |        CALL SET_BIT(32);                                         /*NEWQUAD*/ |  9082 MIC_GEN
1251 |        CALL SET_BIT(58);                                         /*NEWQUAD*/ |  9098 MIC_GEN
1252 |        CALL SET_FIELD(NEWQUAD(II+1),60,63);                      /*NEWQUAD*/ |  9114 MIC_GEN
1253 |        CALL OUT_MIC;                                             /*NEWQUAD*/ |  9158 MIC_GEN
1254 |        BR_FLAG=TRUE;                                            /*NEWQUAD*/ |  9166 MIC_GEN
1255 |        CALL R3_D;                                                /*NEWQUAD*/ |  9174 MIC_GEN
1256 |        END;                                                      /*NEWQUAD*/ |  9182 MIC_GEN
1257 |     DO;                                                          /*NEWQUAD*/ |  9182 MIC_GEN
1258 |                                                                              |  9190 MIC_GEN CASE 1
1259 | /*GENERATE MICROCODE FOR SUB, LT, GT, EQ REGISTER QUADS   */                 |  9190 MIC_GEN
1260 |                                                                              |  9190 MIC_GEN
1261 |        IF OPER=LT THEN DO;                                       /*NEWQUAD*/ |  9190 MIC_GEN
1262 |           TEMP_CURR_MIC=NEWQUAD(II+1);                           /*NEWQUAD*/ |  9206 MIC_GEN
1263 |           NEWQUAD(II+1)=NEWQUAD(II+2);                           /*NEWQUAD*/ |  9226 MIC_GEN
1264 |           NEWQUAD(II+2)=TEMP_CURR_MIC;                           /*NEWQUAD*/ |  9258 MIC_GEN
1265 |           END;                                                   /*NEWQUAD*/ |  9278 MIC_GEN
1266 |        CALL B_R2;                                                /*NEWQUAD*/ |  9278 MIC_GEN
1267 |        CALL SET_BIT(0);                                          /*NEWQUAD*/ |  9286 MIC_GEN
1268 |        CALL SET_BIT(11);                                         /*NEWQUAD*/ |  9300 MIC_GEN
1269 |        CALL SET_BIT(19);                                         /*NEWQUAD*/ |  9316 MIC_GEN
1270 |        CALL SET_BIT(25);                                         /*NEWQUAD*/ |  9332 MIC_GEN
1271 |        CALL SET_BIT(26);                                         /*NEWQUAD*/ |  9348 MIC_GEN
1272 |        CALL SET_BIT(30);                                         /*NEWQUAD*/ |  9364 MIC_GEN
1273 |        CALL SET_BIT(31);                                         /*NEWQUAD*/ |  9380 MIC_GEN
1274 |        CALL SET_BIT(58);                                         /*NEWQUAD*/ |  9396 MIC_GEN
1275 |        CALL SET_FIELD(NEWQUAD(II+1),60,63);                      /*NEWQUAD*/ |  9412 MIC_GEN
1276 |        CALL OUT_MIC;                                             /*NEWQUAD*/ |  9456 MIC_GEN
1277 |        BR_FLAG=TRUE;                                            /*NEWQUAD*/ |  9464 MIC_GEN
1278 |        CALL R3_D;                                                /*NEWQUAD*/ |  9472 MIC_GEN
1279 |        END;                                                      /*NEWQUAD*/ |  9480 MIC_GEN
1280 |     DO;                                                          /*NEWQUAD*/ |  9480 MIC_GEN
1281 |                                                                              |  9488 MIC_GEN CASE 2.
1282 | /*GENERATE MICROCODE FOR BT REGISTER QUADS   */                              |  9488 MIC_GEN
1283 |                                                                              |  9488 MIC_GEN
1284 |        CALL SET_BIT(0);                                          /*NEWQUAD*/ |  9488 MIC_GEN
1285 |        CALL SET_BIT(11);                                         /*NEWQUAD*/ |  9502 MIC_GEN
1286 |        CALL SET_BIT(26);                                         /*NEWQUAD*/ |  9518 MIC_GEN
1287 |        CALL SET_BIT(28);                                         /*NEWQUAD*/ |  9534 MIC_GEN
1288 |        CALL SET_BIT(31);                                         /*NEWQUAD*/ |  9550 MIC_GEN
1289 |        CALL SET_BIT(32);                                         /*NEWQUAD*/ |  9566 MIC_GEN
1290 |        CALL SET_BIT(37);                                         /*NEWQUAD*/ |  9582 MIC_GEN
1291 |        CALL SET_BIT(39);                                         /*NEWQUAD*/ |  9598 MIC_GEN
1292 |        CALL SET_BIT(40);                                         /*NEWQUAD*/ |  9614 MIC_GEN
1293 |        CALL SET_BIT(41);                                         /*NEWQUAD*/ |  9630 MIC_GEN
1294 |        CALL SET_BIT(42);                                         /*NEWQUAD*/ |  9646 MIC_GEN
1295 |        CALL SET_BIT(50);                                         /*NEWQUAD*/ |  9662 MIC_GEN
1296 |        CALL SET_BIT(52);                                         /*NEWQUAD*/ |  9678 MIC_GEN
1297 |        TEMP_CURR_MIC=CURR_MIC;                                   /*NEWQUAD*/ |  9694 MIC_GEN
1298 |        IF CURR_MIC MOD 2 = 1 THEN                                /*NEWQUAD*/ |  9702 MIC_GEN
1299 |           CURR_MIC=CURR_MIC+1;                                   /*NEWQUAD*/ |  9736 MIC_GEN
1300 |        CALL SET_FIELD(CURR_MIC+1,65,72);                         /*NEWQUAD*/ |  9740 MIC_GEN
1301 |        CALL PUT_MIC(TEMP_CURR_MIC);                              /*NEWQUAD*/ |  9776 MIC_GEN
1302 |        CURR_MIC=CURR_MIC+1;                                      /*NEWQUAD*/ |  9792 MIC_GEN
1303 |        CALL SET_BIT(1);                                          /*NEWQUAD*/ |  9804 MIC_GEN
1304 |        CALL OUT_MIC;                                             /*NEWQUAD*/ |  9820 MIC_GEN
1305 |        CALL SET_BIT(1);                                          /*NEWQUAD*/ |  9828 MIC_GEN
1306 |        CALL SET_FIELD(CURR_MIC+2,65,72);                         /*NEWQUAD*/ |  9844 MIC_GEN
1307 |        CALL PUT_MIC(CURR_MIC);                                   /*NEWQUAD*/ |  9880 MIC_GEN
1308 |        CURR_MIC=CURR_MIC+1;                                      /*NEWQUAD*/ |  9896 MIC_GEN
1309 |        CALL SET_BIT(1);                                          /*NEWQUAD*/ |  9908 MIC_GEN
1310 |        LL=NEWQUAD(II+3);                                         /*NEWQUAD*/ |  9924 MIC_GEN
1311 |        CALL SET_FIELD(MIC_LOC(LL),65,72);                        /*NEWQUAD*/ |  9944 MIC_GEN
1312 |        CALL PUT_MIC(CURR_MIC);                                   /*NEWQUAD*/ |  9984 MIC_GEN
```

```
1313 |         CURR_MIC=CURR_MIC+1;                                  /*NEWQUAD*/ | 10010 MIC_GEN
1314 |         END;                                                  /*NEWQUAD*/ | 10012 MIC_GEN
1315 |     DO;                                                       /*NEWQUAD*/ | 10012 MIC_GEN
1316 |                                                                           | 10020 MIC_GEN CASE 3
1317 | /*GENERATE MICROCODE FOR ASL REGISTER QUADS   */                          | 10020 MIC_GEN
1318 |                                                                           | 10020 MIC_GEN
1319 |         CALL SET_BIT(0);                                      /*NEWQUAD*/ | 10020 MIC_GEN
1320 |         CALL SET_BIT(11);                                     /*NEWQUAD*/ | 10034 MIC_GEN
1321 |         CALL SET_BIT(29);                                     /*NEWQUAD*/ | 10050 MIC_GEN
1322 |         CALL SET_BIT(30);                                     /*NEWQUAD*/ | 10066 MIC_GEN
1323 |         CALL SET_BIT(58);                                     /*NEWQUAD*/ | 10082 MIC_GEN
1324 |         CALL SET_FIELD(NEWQUAD(II+1),60,63);                  /*NEWQUAD*/ | 10098 MIC_GEN
1325 |         CALL OUT_MIC;                                         /*NEWQUAD*/ | 10142 MIC_GEN
1326 |         CALL R3_D;                                            /*NEWQUAD*/ | 10150 MIC_GEN
1327 |         END;                                                  /*NEWQUAD*/ | 10158 MIC_GEN
1328 |     DO;                                                       /*NEWQUAD*/ | 10158 MIC_GEN
1329 |                                                                           | 10166 MIC_GEN CASE 4
1330 | /*GENERATE MICROCODE FOR RD REGISTER QUADS*/                             | 10166 MIC_GEN
1331 |                                                                           | 10166 MIC_GEN
1332 |         CALL RO_PLUS2_READ;                                   /*NEWQUAD*/ | 10166 MIC_GEN
1333 |         CALL RTEMP_UNIBUS;                                    /*NEWQUAD*/ | 10174 MIC_GEN
1334 |         CALL SET_BIT(60);                                     /*NEWQUAD*/ | 10182 MIC_GEN
1335 |         CALL BA_RO_READ;                                      /*NEWQUAD*/ | 10198 MIC_GEN
1336 |         BR_FLAG=TRUE;                                         /*NEWQUAD*/ | 10206 MIC_GEN
1337 |         CALL R3_UNIBUS;                                       /*NEWQUAD*/ | 10214 MIC_GEN
1338 |         END;                                                  /*NEWQUAD*/ | 10222 MIC_GEN
1339 |     DO;                                                       /*NEWQUAD*/ | 10222 MIC_GEN
1340 |                                                                           | 10230 MIC_GEN CASE 5
1341 | /*GENERATE MICROCODE FOR WT REGISTER QUADS   */                          | 10230 MIC_GEN
1342 |                                                                           | 10230 MIC_GEN
1343 |         CALL RO_PLUS2_READ;                                   /*NEWQUAD*/ | 10230 MIC_GEN
1344 |         CALL RTEMP_UNIBUS;                                    /*NEWQUAD*/ | 10238 MIC_GEN
1345 |         CALL SET_BIT(1);                                      /*NEWQUAD*/ | 10246 MIC_GEN
1346 |         CALL SET_BIT(13);                                     /*NEWQUAD*/ | 10262 MIC_GEN
1347 |         CALL SET_BIT(47);                                     /*NEWQUAD*/ | 10278 MIC_GEN
1348 |         CALL SET_BIT(58);                                     /*NEWQUAD*/ | 10294 MIC_GEN
1349 |         CALL SET_BIT(60);                                     /*NEWQUAD*/ | 10310 MIC_GEN
1350 |         CALL OUT_MIC;                                         /*NEWQUAD*/ | 10326 MIC_GEN
1351 |         CALL SET_BIT(0);                                      /*NEWQUAD*/ | 10334 MIC_GEN
1352 |         CALL SET_BIT(2);                                      /*NEWQUAD*/ | 10348 MIC_GEN
1353 |         CALL SET_BIT(11);                                     /*NEWQUAD*/ | 10364 MIC_GEN
1354 |         CALL SET_BIT(15);                                     /*NEWQUAD*/ | 10380 MIC_GEN
1355 |         CALL SET_BIT(17);                                     /*NEWQUAD*/ | 10396 MIC_GEN
1356 |     CALL SET_BIT(58);                                         /*NEWQUAD*/ | 10412 MIC_GEN
1357 |         CALL SET_FIELD(NEWQUAD(II+1),60,63);                  /*NEWQUAD*/ | 10428 MIC_GEN
1358 |         BR_FLAG=TRUE;                                         /*NEWQUAD*/ | 10472 MIC_GEN
1359 |         CALL OUT_MIC;                                         /*NEWQUAD*/ | 10480 MIC_GEN
1360 |         END;                                                  /*NEWQUAD*/ | 10488 MIC_GEN
1361 |     DO;                                                       /*NEWQUAD*/ | 10488 MIC_GEN
1362 |                                                                           | 10496 MIC_GEN CASE 6
1363 | /*GENERATE MICROCODE FOR RDAD REGISTER QUAD   */                         | 10496 MIC_GEN
1364 |                                                                           | 10496 MIC_GEN
1365 |         CALL RO_PLUS2_READ;                                   /*NEWQUAD*/ | 10496 MIC_GEN
1366 |         CALL R3_UNIBUS;                                       /*NEWQUAD*/ | 10504 MIC_GEN
1367 |         END;                                                  /*NEWQUAD*/ | 10512 MIC_GEN
1368 |     DO;                                                       /*NEWQUAD*/ | 10512 MIC_GEN
1369 |                                                                           | 10520 MIC_GEN CASE 7
1370 | /*GENERATE MICROCODE FOR RDVR REGISTER QUAD   */                         | 10520 MIC_GEN
1371 |                                                                           | 10520 MIC_GEN
1372 |         CALL SET_FIELD(NEWQUAD(II+1),60,63);                  /*NEWQUAD*/ | 10520 MIC_GEN
1373 |         CALL BA_RO_READ;                                      /*NEWQUAD*/ | 10564 MIC_GEN
1374 |         BR_FLAG=TRUE;                                         /*NEWQUAD*/ | 10572 MIC_GEN
1375 |         CALL R3_UNIBUS;                                       /*NEWQUAD*/ | 10580 MIC_GEN
1376 |         END;                                                  /*NEWQUAD*/ | 10588 MIC_GEN
1377 |     DO;                                                       /*NEWQUAD*/ | 10588 MIC_GEN
1378 |                                                                           | 10596 MIC_GEN CASE 8
```

```
1379 | /*GENERATE MICROCODE FOR LAR REGISTER QUAD   */                                  | 10576 MIC_GEN
1380 |                                                                                  | 10596 MIC_GEN
1381 |       CALL BA_RO_READ;                                    /*NEWQUAD*/ | 10596 MIC_GEN
1382 |       CALL SET_BIT(1);                                    /*NEWQUAD*/ | 10604 MIC_GEN
1383 |       CALL SET_BIT(6);                                    /*NEWQUAD*/ | 10620 MIC_GEN
1384 |       CALL SET_BIT(7);                                    /*NEWQUAD*/ | 10636 MIC_GEN
1385 |       CALL SET_BIT(45);                                   /*NEWQUAD*/ | 10652 MIC_GEN
1386 |       CALL SET_BIT(58);                                   /*NEWQUAD*/ | 10668 MIC_GEN
1387 |       CALL OUT_MIC;                                       /*NEWQUAD*/ | 10684 MIC_GEN
1388 |       END;                                                /*NEWQUAD*/ | 10692 MIC_GEN
1389 |  DO;                                                      /*NEWQUAD*/ | 10692 MIC_GEN
1390 |                                                                                  | 10700 MIC_GEN CASE 9
1391 | /* GENERATE MICROCODE FOR WTAD REGISTER QUAD   */                                | 10700 MIC_GEN
1392 |                                                                                  | 10700 MIC_GEN
1393 |     CALL SET_BIT(1);                                      /*NEWQUAD*/ | 10700 MIC_GEN
1394 |     CALL SET_BIT(13);                                     /*NEWQUAD*/ | 10716 MIC_GEN
1395 |     CALL SET_BIT(47);                                     /*NEWQUAD*/ | 10732 MIC_GEN
1396 |     CALL SET_BIT(58);                                     /*NEWQUAD*/ | 10748 MIC_GEN
1397 |     CALL SET_FIELD(NEWQUAD(II+1),60,63);                  /*NEWQUAD*/ | 10764 MIC_GEN
1398 |     CALL OUT_MIC;                                         /*NEWQUAD*/ | 10808 MIC_GEN
1399 |     CALL SET_BIT(0);                                      /*NEWQUAD*/ | 10816 MIC_GEN
1400 |     CALL SET_BIT(2);                                      /*NEWQUAD*/ | 10830 MIC_GEN
1401 |     CALL SET_BIT(11);                                     /*NEWQUAD*/ | 10846 MIC_GEN
1402 |     CALL SET_BIT(15);                                     /*NEWQUAD*/ | 10862 MIC_GEN
1403 |     CALL SET_BIT(17);                                     /*NEWQUAD*/ | 10878 MIC_GEN
1404 |     CALL SET_BIT(58);                                     /*NEWQUAD*/ | 10894 MIC_GEN
1405 |     CALL SET_FIELD(NEWQUAD(II+3),60,63);                  /*NEWQUAD*/ | 10910 MIC_GEN
1406 |     BR_FLAG=TRUE;                                         /*NEWQUAD*/ | 10954 MIC_GEN
1407 |     CALL OUT_MIC;                                         /*NEWQUAD*/ | 10962 MIC_GEN
1408 |     END;                                                  /*NEWQUAD*/ | 10970 MIC_GEN
1409 |                                                                                  | 10970 MIC_GEN
1410 | /*SR REGISTER QUAD*/                                                            | 10970 MIC_GEN
1411 |                                                                                  | 10970 MIC_GEN
1412 |     ;                                                     /*NEWQUAD*/ | 10970 MIC_GEN
1413 |     DO;                                                   /*NEWQUAD*/ | 10978 MIC_GEN CASE 1
1414 |                                                                                  | 10986 MIC_GEN CASE 1
1415 | /* GENERATE MICROCODE FOR MUL REGISTER QUAD  */                                  | 10986 MIC_GEN
1416 |                                                                                  | 10986 MIC_GEN
1417 |       CALL RTEMP_4;                                       /*NEWQUAD*/ | 10986 MIC_GEN
1418 |       CALL RTEMP_SHIFT;                                   /*NEWQUAD*/ | 10994 MIC_GEN
1419 |       CALL RTEMP_SHIFT;                                   /*NEWQUAD*/ | 11002 MIC_GEN
1420 |       CALL R3_0;                                          /*NEWQUAD*/ | 11010 MIC_GEN
1421 |       TEMP_CURR_MIC=CURR_MIC;                             /*NEWQUAD*/ | 11018 MIC_GEN
1422 |       IF CURR_MIC MOD 2 = 0 THEN DO;                      /*NEWQUAD*/ | 11026 MIC_GEN
1423 |          CURR_MIC=CURR_MIC+1;                             /*NEWQUAD*/ | 11052 MIC_GEN
1424 |          END;                                             /*NEWQUAD*/ | 11064 MIC_GEN
1425 |       CALL B_R1;                                          /*NEWQUAD*/ | 11064 MIC_GEN
1426 |       CALL SET_FIELD(CURR_MIC+1,65,72);                   /*NEWQUAD*/ | 11072 MIC_GEN
1427 |       CALL PUT_MIC(TEMP_CURR_MIC);                        /*NEWQUAD*/ | 11108 MIC_GEN
1428 |       CURR_MIC=CURR_MIC+1;                                /*NEWQUAD*/ | 11124 MIC_GEN
1429 |       TEMP_CURR_MIC=CURR_MIC;                             /*NEWQUAD*/ | 11136 MIC_GEN
1430 |       CALL R2_SHIFT;                                      /*NEWQUAD*/ | 11144 MIC_GEN
1431 |       CALL NO_OP;                                         /*NEWQUAD*/ | 11152 MIC_GEN
1432 |       CALL R3_R3_PLUS_8;                                  /*NEWQUAD*/ | 11160 MIC_GEN
1433 |       CALL RTEMP_MINUS_1;                                 /*NEWQUAD*/ | 11168 MIC_GEN
1434 |       CALL SET_FIELD(CURR_MIC+2,65,72);                   /*NEWQUAD*/ | 11176 MIC_GEN
1435 |       CALL PUT_MIC(CURR_MIC);                             /*NEWQUAD*/ | 11212 MIC_GEN
1436 |       CURR_MIC=CURR_MIC+2;                                /*NEWQUAD*/ | 11228 MIC_GEN
1437 |       CALL SET_BIT(1);                                    /*NEWQUAD*/ | 11240 MIC_GEN
1438 |       CALL SET_FIELD(TEMP_CURR_MIC,65,72);                /*NEWQUAD*/ | 11256 MIC_GEN
1439 |       CALL PUT_MIC(CURR_MIC);                             /*NEWQUAD*/ | 11288 MIC_GEN
1440 |       CURR_MIC=CURR_MIC+1;                                /*NEWQUAD*/ | 11304 MIC_GEN
1441 |       BR_FLAG=TRUE;                                       /*NEWQUAD*/ | 11316 MIC_GEN
1442 |       CALL NO_OP;                                         /*NEWQUAD*/ | 11324 MIC_GEN
1443 |       END;                                                /*NEWQUAD*/ | 11332 MIC_GEN
1444 |     END;                                                  /*NEWQUAD*/ | 11332 MIC_GEN
```

```
1445 |   END;                                                          /*NEWQUAD*/ | 11340 REC_SET
1446 |  PUT_NEWQUAD:PROCEDURE;                                         /*NEWQUAD*/ | 11346
1447 |                                                                             | 11346 PUT_NEWQUAD
1448 |  /*PUT_NEWQUAD  PRINT OUT REGISTER QUADS  */                                | 11346 PUT_NEWQUAD
1449 |                                                                             | 11346 PUT_NEWQUAD
1450 |   NEWQUAD(NEWQUADNO) = OPRTOR;                                  /*NEWQUAD*/ | 11346 PUT_NEWQUAD
1451 |   NEWQUAD(NEWQUADNO + 1 ) = OPRND1;                             /*NEWQUAD*/ | 11374 PUT_NEWQUAD
1452 |   NEWQUAD(NEWQUADNO + 2 ) = OPRND2;                             /*NEWQUAD*/ | 11394 PUT_NEWQUAD
1453 |   NEWQUAD(NEWQUADNO + 3 ) = RSLT;                               /*NEWQUAD*/ | 11414 PUT_NEWQUAD
1454 |   NEWQUADNO = NEWQUADNO + 4;                                    /*NEWQUAD*/ | 11434 PUT_NEWQUAD
1455 |   END;                                                          /*NEWQUAD*/ | 11446 PUT_NEWQUAD
1456 |   SYMB_INDEX:PROCEDURE;                                         /*NEWQUAD*/ | 11452
1457 |                                                                             | 11452 SYMB_INDEX
1458 |  /*SYMB_INDEX  RETURNS SYMBOL TABLE INDEX OF TEMP_CHAR   */                  | 11452 SYMB_INDEX
1459 |                                                                             | 11452 SYMB_INDEX
1460 |   DECLARE INDX FIXED;                                           /*NEWQUAD*/ | 11452 SYMB_INDEX
1461 |   INDX = 1;                                                     /*NEWQUAD*/ | 11464 SYMB_INDEX
1462 |   DO WHILE TEMP_CHAR ¬= SYMB(INDX);                             /*NEWQUAD*/ | 11472 SYMB_INDEX
1463 |      INDX = INDX + 1;                                           /*NEWQUAD*/ | 11522 SYMB_INDEX
1464 |      END;                                                       /*NEWQUAD*/ | 11534 SYMB_INDEX
1465 |   RETURN INDX;                                                  /*NEWQUAD*/ | 11542 SYMB_INDEX
1466 |   END;                                                          /*NEWQUAD*/ | 11552 SYMB_INDEX
1467 |  LAB_INDEX:PROCEDURE;                                           /*NEWQUAD*/ | 11558
1468 |                                                                             | 11558 LAB_INDEX
1469 |  /*LAB_INDEX  RETURN LABEL TABLE INDEX OF TEMP_CHAR   */                    | 11558 LAB_INDEX
1470 |                                                                             | 11558 LAB_INDEX
1471 |   DECLARE INDX FIXED;                                           /*NEWQUAD*/ | 11558 LAB_INDEX
1472 |   INDX = 1;                                                     /*NEWQUAD*/ | 11570 LAB_INDEX
1473 |   DO WHILE TEMP_CHAR ¬= LABID(INDX);                            /*NEWQUAD*/ | 11578 LAB_INDEX
1474 |      INDX = INDX + 1;                                           /*NEWQUAD*/ | 11628 LAB_INDEX
1475 |      END;                                                       /*NEWQUAD*/ | 11640 LAB_INDEX
1476 |   RETURN INDX;                                                  /*NEWQUAD*/ | 11648 LAB_INDEX
1477 |   END;                                                          /*NEWQUAD*/ | 11658 LAB_INDEX
1478 |  CONVAL_INDEX:PROCEDURE;                                        /*NEWQUAD*/ | 11664
1479 |                                                                             | 11664 CONVAL_INDE
1480 |  /*CONVAL_INDEX  RETURNS CONSTANT TABLE INDEX OF TEMP_CHAR    */             | 11664 CONVAL_INDE
1481 |                                                                             | 11664 CONVAL_INDE
1482 |   DECLARE INDX FIXED;                                           /*NEWQUAD*/ | 11664 CONVAL_INDE
1483 |   INDX=1;                                                       /*NEWQUAD*/ | 11676 CONVAL_INDE
1484 |   DO WHILE TEMP_CHAR ¬= CONVAL(INDX);                           /*NEWQUAD*/ | 11684 CONVAL_INDE
1485 |      INDX=INDX+1;                                               /*NEWQUAD*/ | 11750 CONVAL_INDE
1486 |      END;                                                       /*NEWQUAD*/ | 11762 CONVAL_INDE
1487 |      RETURN INDX;                                               /*NEWQUAD*/ | 11770 CONVAL_INDE
1488 |   END;                                                          /*NEWQUAD*/ | 11780 CONVAL_INDE
1489 |  READQUAD:PROCEDURE(QUADNO);                                    /*NEWQUAD*/ | 11786
1490 |                                                                             | 11786 READQUAD
1491 |  /*READQUAD PRINT OUT QUADS  */                                             | 11786 READQUAD
1492 |                                                                             | 11786 READQUAD
1493 |   DECLARE QUADNO FIXED;                                         /*NEWQUAD*/ | 11786 READQUAD
1494 |   DECLARE LATEST FIXED INITIAL(0);                              /*NEWQUAD*/ | 11798 READQUAD
1495 |   DECLARE(OPND1,OPND2,RES)FIXED;                                /*NEWQUAD*/ | 11798 READQUAD
1496 |   RCD_NR=(QUADNO-1)/MAXQUADS;                                   /*NEWQUAD*/ | 11798 READQUAD
1497 |   IF RCD_NR ¬= RCD_BUFF THEN DO;                                /*NEWQUAD*/ | 11820 READQUAD
1498 |      QUADS=FILE(1,RCD_BUFF);                                    /*NEWQUAD*/ | 11836 READQUAD
1499 |      RCD_BUFF=RCD_BUFF+1;                                       /*NEWQUAD*/ | 11866 READQUAD
1500 |      END;                                                       /*NEWQUAD*/ | 11878 READQUAD
1501 |   LOC_QUAD=((QUADNO-1) MOD MAXQUADS)*4 +1;                      /*NEWQUAD*/ | 11878 READQUAD
1502 |   OPER = QUADS(LOC_QUAD);                                       /*NEWQUAD*/ | 11910 READQUAD
1503 |   OPND1=QUADS(LOC_QUAD+1);                                      /*NEWQUAD*/ | 11926 READQUAD
1504 |   OPND2=QUADS(LOC_QUAD+2);                                      /*NEWQUAD*/ | 11946 READQUAD
1505 |   RES=QUADS(LOC_QUAD+3);                                        /*NEWQUAD*/ | 11966 READQUAD
1506 |   IF QUADNO > LATEST THEN DO;                                   /*NEWQUAD*/ | 11936 READQUAD
1507 |      LATEST = QUADNO;                                           /*NEWQUAD*/ | 12002 READQUAD
1508 |      IF OPER = LAB THEN                                         /*NEWQUAD*/ | 12010 READQUAD
1509 |         OPERAND1=LABID(OPND1);                                  /*NEWQUAD*/ | 12034 READQUAD
1510 |      ELSE DO;                                                   /*NEWQUAD*/ | 12042 READQUAD
```

```
1511 |         IF OPND1<0 THEN                                          /*NEWQUAD*/ | 12[illegible] READQUAD
1512 |            OPERAND1=CONVAL(-OPND1);                              /*NEWQUAD*/ | 12074 READQUAD
1513 |         ELSE                                                     /*NEWQUAD*/ | 12092 READQUAD
1514 |            IF OPND1 > 0 THEN                                     /*NEWQUAD*/ | 12092 READQUAD
1515 |               OPERAND1 = SYMB(OPND1);                            /*NEWQUAD*/ | 12124 READQUAD
1516 |            ELSE                                                  /*NEWQUAD*/ | 12132 READQUAD
1517 |               OPERAND1='0';                                      /*NEWQUAD*/ | 12132 READQUAD
1518 |         END;                                                     /*NEWQUAD*/ | 12148 READQUAD
1519 |      IF OPND2 < 0 THEN                                           /*NEWQUAD*/ | 12148 READQUAD
1520 |         OPERAND2=CONVAL(-OPND2);                                 /*NEWQUAD*/ | 12172 READQUAD
1521 |      ELSE                                                        /*NEWQUAD*/ | 12190 READQUAD
1522 |         IF OPND2 > 0 THEN                                        /*NEWQUAD*/ | 12190 READQUAD
1523 |            OPERAND2 = SYMB(OPND2);                               /*NEWQUAD*/ | 12222 READQUAD
1524 |         ELSE                                                     /*NEWQUAD*/ | 12230 READQUAD
1525 |            OPERAND2='0';                                         /*NEWQUAD*/ | 12230 READQUAD
1526 |         IF OPER > HALT THEN DO;                                  /*NEWQUAD*/ | 12246 READQUAD
1527 |            IF OPER < REL THEN                                    /*NEWQUAD*/ | 12262 READQUAD
1528 |               RESULT = LABID(RES);                               /*NEWQUAD*/ | 12286 READQUAD
1529 |            ELSE                                                  /*NEWQUAD*/ | 12294 READQUAD
1530 |               RESULT=SYMB(RES);                                  /*NEWQUAD*/ | 12294 READQUAD
1531 |      END;                                                        /*NEWQUAD*/ | 12318 READQUAD
1532 |      IF OPER <= HALT THEN DO;                                    /*NEWQUAD*/ | 12318 READQUAD
1533 |         IF RES¬=0 THEN                                           /*NEWQUAD*/ | 12334 READQUAD
1534 |            RESULT = SYMB(RES);                                   /*NEWQUAD*/ | 12358 READQUAD
1535 |            END;                                                  /*NEWQUAD*/ | 12366 READQUAD
1536 |      IF RES=0 THEN                                               /*NEWQUAD*/ | 12366 READQUAD
1537 |         RESULT='0';                                              /*NEWQUAD*/ | 12390 READQUAD
1538 |      BUFFER=PAD('    '||ALPHA(OPER),13);                         /*NEWQUAD*/ | 12390 READQUAD
1539 |      BUFFER=PAD(BUFFER||OPERAND1,24);                            /*NEWQUAD*/ | 12438 READQUAD
1540 |      BUFFER=PAD(BUFFER||OPERAND2,34);                            /*NEWQUAD*/ | 12478 READQUAD
1541 |      BUFFER=PAD(BUFFER||RESULT,48);                              /*NEWQUAD*/ | 12518 READQUAD
1542 |      BUFFER=PAD(BUFFER||RCD_NR,57);                              /*NEWQUAD*/ | 12558 READQUAD
1543 |      BUFFER=PAD(BUFFER||LOC_QUAD,80);                            /*NEWQUAD*/ | 12606 READQUAD
1544 |      OUTPUT=BUFFER;                                              /*NEWQUAD*/ | 12654 READQUAD
1545 |      END;                                                        /*NEWQUAD*/ | 12674 READQUAD
1546 |   END;                                                           /*NEWQUAD*/ | 12674 READQUAD
1547 | PRINT_NEWQUADS:PROCEDURE;                                        /*NEWQUAD*/ | 12680
1548 |                                                                            | 12680 PRINT_NEWQUAD
1549 | /*PRINT_NEWQUADS  PRINT OUT REGISTER QUADS   */                            | 12680 PRINT_NEWQUAD
1550 |                                                                            | 12680 PRINT_NEWQUAD
1551 |    DECLARE CASE_NUM(30) FIXED INITIAL(0,                         /*NEWQUAD*/ | 12680 PRINT_NEWQUAD
1552 |                                       1,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1553 |                                       1,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1554 |                                       1,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1555 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1556 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1557 |                                       3,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1558 |                                       6,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1559 |                                       2,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1560 |                                       2,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1561 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1562 |                                       1,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1563 |                                       1,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1564 |                                       1,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1565 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1566 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1567 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1568 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1569 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1570 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1571 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1572 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1573 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1574 |                                       0,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1575 |                                       4,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
1576 |                                       7,                         /*NEWQUAD*/ | 12692 PRINT_NEWQUAD
```

```
1577 |                                    8,                                  /*NEWQUAD*/ | 12692 PRINT_NEWQUA
1578 |                                    7,                                  /*NEWQUAD*/ | 12692 PRINT_NEWQUA
1579 |                                    0,                                  /*NEWQUAD*/ | 12692 PRINT_NEWQUA
1580 |                                    0,                                  /*NEWQUAD*/ | 12692 PRINT_NEWQUA
1581 |                                    0];                                 /*NEWQUAD*/ | 12692 PRINT_NEWQUA
1582 |  OUTPUT(1) = '1              NEW QUADS';                                /*NEWQUAD*/ | 12692 PRINT_NEWQUA
1583 |  DOUBLE_SPACE;                                                          /*NEWQUAD*/ | 12716 PRINT_NEWQUA
1584 |  OUTPUT = 'OPERATOR  OPERAND1/  OPERAND2  RESULT';                      /*NEWQUAD*/ | 12740 PRINT_NEWQUA
1585 |  OUTPUT = '          CONDITION            LABEL';                       /*NEWQUAD*/ | 12760 PRINT_NEWQUA
1586 |  BR_FLAG=FALSE;                                                         /*NEWQUAD*/ | 12780 PRINT_NEWQUA
1587 |  CURR_MIC=0;                                                            /*NEWQUAD*/ | 12786 PRINT_NEWQUA
1588 |  BUFFER='                          GENERATED MICRO WORDS';              /*NEWQUAD*/ | 12792 PRINT_NEWQUA
1589 |  BUFFER=PAD(BUFFER,80);                                                 /*NEWQUAD*/ | 12800 PRINT_NEWQUA
1590 |  OUTPUT(3)=BUFFER;                                                      /*NEWQUAD*/ | 12824 PRINT_NEWQUA
1591 |  BUFFER=PAD('',80);                                                     /*NEWQUAD*/ | 12848 PRINT_NEWQUA
1592 |  OUTPUT(3)=BUFFER;                                                      /*NEWQUAD*/ | 12872 PRINT_NEWQUA
1593 |  BUFFER='   L    C  C W  C C C  B   D    S    A    S'||                 /*NEWQUAD*/ | 12896 PRINT_NEWQUA
1594 |         '    S   S  S   U   S    R      U';                            /*NEWQUAD*/ | 12896 PRINT_NEWQUA
1595 |  BUFFER=PAD(BUFFER,80);                                                 /*NEWQUAD*/ | 12920 PRINT_NEWQUA
1596 |  OUTPUT(3)=BUFFER;                                                      /*NEWQUAD*/ | 12944 PRINT_NEWQUA
1597 |  BUFFER='   O    L  I R  B D B  U   A    P    L    B'||                 /*NEWQUAD*/ | 12968 PRINT_NEWQUA
1598 |         '    B   D  B   B    R    I      P';                           /*NEWQUAD*/ | 12968 PRINT_NEWQUA
1599 |  BUFFER=PAD(BUFFER,80);                                                 /*NEWQUAD*/ | 12992 PRINT_NEWQUA
1600 |  OUTPUT(3)=BUFFER;                                                      /*NEWQUAD*/ | 13016 PRINT_NEWQUA
1601 |  BUFFER='   C    K  R        A  S  D    S    U    C'||                 /*NEWQUAD*/ | 13040 PRINT_NEWQUA
1602 |         '    M   M  A   F    X    F      F';                           /*NEWQUAD*/ | 13040 PRINT_NEWQUA
1603 |  BUFFER=PAD(BUFFER,80);                                                 /*NEWQUAD*/ | 13064 PRINT_NEWQUA
1604 |  OUTPUT(3)=BUFFER;                                                      /*NEWQUAD*/ | 13088 PRINT_NEWQUA
1605 |  DO II=0 TO NEWQUADND-1 BY 4;                                           /*NEWQUAD*/ | 13112 PRINT_NEWQUA
1606 |     OPER=NEWQUAD(II);                                                   /*NEWQUAD*/ | 13136 PRINT_NEWQUA
1607 |     OPERATION=ALPHA(OPER);                                              /*NEWQUAD*/ | 13174 PRINT_NEWQUA
1608 |     DO CASE CASE_NUM(OPER);                                             /*NEWQUAD*/ | 13190 PRINT_NEWQUA
1609 |        DO;                                                              /*NEWQUAD*/ | 13222 PRINT_NEWQUA
1610 |           OPERAND1='R'||NEWQUAD(II+1);                                  /*NEWQUAD*/ | 13222 PRINT_NEWQUA
1611 |           OPERAND2='0';                                                 /*NEWQUAD*/ | 13266 PRINT_NEWQUA
1612 |           RESULT='R'||NEWQUAD(II+3);                                   /*NEWQUAD*/ | 13274 PRINT_NEWQUA
1613 |           END;                                                          /*NEWQUAD*/ | 13318 PRINT_NEWQUA
1614 |        DO;                                                              /*NEWQUAD*/ | 13318 PRINT_NEWQUA
1615 |           OPERAND1='R'||NEWQUAD(II+1);                                  /*NEWQUAD*/ | 13326 PRINT_NEWQUA
1616 |           OPERAND2='R'||NEWQUAD(II+2);                                  /*NEWQUAD*/ | 13370 PRINT_NEWQUA
1617 |           RESULT='R'||NEWQUAD(II+3);                                   /*NEWQUAD*/ | 13414 PRINT_NEWQUA
1618 |           END;                                                          /*NEWQUAD*/ | 13458 PRINT_NEWQUA
1619 |        DO;                                                              /*NEWQUAD*/ | 13458 PRINT_NEWQUA
1620 |           JJ=NEWQUAD(II+3);                                             /*NEWQUAD*/ | 13456 PRINT_NEWQUA
1621 |           OPERAND1='R'||NEWQUAD(II+1);                                  /*NEWQUAD*/ | 13486 PRINT_NEWQUA
1622 |           OPERAND2='0';                                                 /*NEWQUAD*/ | 13530 PRINT_NEWQUA
1623 |           RESULT=LABID(JJ);                                             /*NEWQUAD*/ | 13538 PRINT_NEWQUA
1624 |           END;                                                          /*NEWQUAD*/ | 13554 PRINT_NEWQUA
1625 |        DO;                                                              /*NEWQUAD*/ | 13554 PRINT_NEWQUA
1626 |           OPERAND1='0';                                                 /*NEWQUAD*/ | 13552 PRINT_NEWQUA
1627 |           OPERAND2='0';                                                 /*NEWQUAD*/ | 13570 PRINT_NEWQUA
1628 |           RESULT='0';                                                   /*NEWQUAD*/ | 13578 PRINT_NEWQUA
1629 |           END;                                                          /*NEWQUAD*/ | 13586 PRINT_NEWQUA
1630 |        DO;                                                              /*NEWQUAD*/ | 13586 PRINT_NEWQUA
1631 |           JJ=NEWQUAD(II+1);                                             /*NEWQUAD*/ | 13594 PRINT_NEWQUA
1632 |           OPERAND1=LABID(JJ);                                           /*NEWQUAD*/ | 13614 PRINT_NEWQUA
1633 |           OPERAND2='0';                                                 /*NEWQUAD*/ | 13630 PRINT_NEWQUA
1634 |           RESULT='0';                                                   /*NEWQUAD*/ | 13638 PRINT_NEWQUA
1635 |           END;                                                          /*NEWQUAD*/ | 13646 PRINT_NEWQUA
1636 |        ;                                                                /*NEWQUAD*/ | 13646 PRINT_NEWQUA
1637 |        DO;                                                              /*NEWQUAD*/ | 13654 PRINT_NEWQUA
1638 |           JJ=NEWQUAD(II+3);                                             /*NEWQUAD*/ | 13662 PRINT_NEWQUA
1639 |           OPERAND1='0';                                                 /*NEWQUAD*/ | 13652 PRINT_NEWQUA
1640 |           OPERAND2='0';                                                 /*NEWQUAD*/ | 13690 PRINT_NEWQUA
1641 |           RESULT=LABID(JJ);                                             /*NEWQUAD*/ | 13698 PRINT_NEWQUA
1642 |           END;                                                          /*NEWQUAD*/ | 13714 PRINT_NEWQUA
```

```
1643 |        DO;                                                    /*NEWQUAD*/ | 13714 PRINT_NEWQUAD
1644 |           JJ=NEWQUAD(II+1);                                   /*NEWQUAD*/ | 13722 PRINT_NEWQUAD
1645 |           IF JJ>0 THEN                                        /*NEWQUAD*/ | 13742 PRINT_NEWQUAD
1646 |              OPERAND1=SYMB(JJ);                               /*NEWQUAD*/ | 13766 PRINT_NEWQUAD
1647 |           ELSE                                                /*NEWQUAD*/ | 13774 PRINT_NEWQUAD
1648 |              OPERAND1=CONVAL(-JJ);                            /*NEWQUAD*/ | 13774 PRINT_NEWQUAD
1649 |           OPERAND2='0';                                       /*NEWQUAD*/ | 13808 PRINT_NEWQUAD
1650 |           RESULT='R'||NEWQUAD(II+3);                         /*NEWQUAD*/ | 13816 PRINT_NEWQUAD
1651 |           END;                                                /*NEWQUAD*/ | 13860 PRINT_NEWQUAD
1652 |        DO;                                                    /*NEWQUAD*/ | 13860 PRINT_NEWQUAD
1653 |           JJ=NEWQUAD(II+3);                                   /*NEWQUAD*/ | 13868 PRINT_NEWQUAD
1654 |           OPERAND1='R'||NEWQUAD(II+1);                        /*NEWQUAD*/ | 13886 PRINT_NEWQUAD
1655 |           OPERAND2='0';                                       /*NEWQUAD*/ | 13932 PRINT_NEWQUADS
1656 |           RESULT=SYMB(JJ);                                    /*NEWQUAD*/ | 13940 PRINT_NEWQUADS
1657 |           END;                                                /*NEWQUAD*/ | 13955 PRINT_NEWQUADS
1658 |        END;                                                   /*NEWQUAD*/ | 13956 PRINT_NEWQUADS
1659 |     BUFFER=PAD('    '||OPERATION,13);                         /*NEWQUAD*/ | 13964 PRINT_NEWQUADS
1660 |     BUFFER=PAD(BUFFER||OPERAND1,24);                          /*NEWQUAD*/ | 14004 PRINT_NEWQUADS
1661 |     BUFFER=PAD(BUFFER||OPERAND2,34);                          /*NEWQUAD*/ | 14044 PRINT_NEWQUADS
1662 |     BUFFER=PAD(BUFFER||RESULT,48);                            /*NEWQUAD*/ | 14084 PRINT_NEWQUADS
1663 |     OUTPUT = BUFFER;                                          /*NEWQUAD*/ | 14124 PRINT_NEWQUADS
1664 |     CALL MIC_GEN;                                             /*NEWQUAD*/ | 14144 PRINT_NEWQUADS
1665 |     END;                                                      /*NEWQUAD*/ | 14152 PRINT_NEWQUADS
1666 |  END;                                                         /*NEWQUAD*/ | 14160 PRINT_NEWQUADS
1667 | SORT:PROCEDURE( A, B);                                        /*NEWQUAD*/ | 14166
1668 |                                                                         | 14166 SORT
1669 | /*SORT   SORT SORTNUM OR SORTREF ARRAY    */                            | 14166 SORT
1670 |                                                                         | 14166 SORT
1671 |   DECLARE (B, SWITCH) BIT(8);                                 /*NEWQUAD*/ | 14166 SORT
1672 |DECLARE(A,C,D,TEMP)FIXED;                                      /*NEWQUAD*/ | 14178 SORT
1673 |  SWITCH = TRUE;                                               /*NEWQUAD*/ | 14178 SORT
1674 |  D = 0;                                                       /*NEWQUAD*/ | 14186 SORT
1675 |  DO WHILE SWITCH = TRUE;                                      /*NEWQUAD*/ | 14192 SORT
1676 |     SWITCH = FALSE;                                           /*NEWQUAD*/ | 14210 SORT
1677 |     DO C = D TO A - 1;                                        /*NEWQUAD*/ | 14216 SORT
1678 |        IF B= TRUE THEN DO;                                    /*NEWQUAD*/ | 14264 SORT
1679 |           IF SORTNUM(C) > SORTNUM( C + 1 ) THEN DO;           /*NEWQUAD*/ | 14282 SORT
1680 |              TEMP = SORTNUM(C);                               /*NEWQUAD*/ | 14313 SORT
1681 |              SORTNUM(C) = SORTNUM(C+1);                       /*NEWQUAD*/ | 14334 SORT
1682 |              SORTNUM(C+1)=TEMP;                               /*NEWQUAD*/ | 14362 SORT
1683 |              TEMP = SORTREF(C);                               /*NEWQUAD*/ | 14382 SORT
1684 |              SORTREF(C) = SORTREF(C+1);                       /*NEWQUAD*/ | 14398 SORT
1685 |              SORTREF(C+1)= TEMP;                              /*NEWQUAD*/ | 14426 SORT
1686 |              TEMP=REGNO(C);                                   /*NEWQUAD*/ | 14446 SORT
1687 |              REGNO(C)= REGNO(C+1);                            /*NEWQUAD*/ | 14462 SORT
1688 |              REGNO(C+1)=TEMP;                                 /*NEWQUAD*/ | 14490 SORT
1689 |              SWITCH=TRUE;                                     /*NEWQUAD*/ | 14510 SORT
1690 |              END;                                             /*NEWQUAD*/ | 14518 SORT
1691 |           END;                                                /*NEWQUAD*/ | 14518 SORT
1692 |        ELSE DO;                                               /*NEWQUAD*/ | 14518 SORT
1693 |           IF SORTREF(C) > SORTREF(C+1) THEN DO;               /*NEWQUAD*/ | 14526 SORT
1694 |              TEMP=SORTNUM(C);                                 /*NEWQUAD*/ | 14562 SORT
1695 |              SORTNUM(C)=SORTNUM(C+1);                         /*NEWQUAD*/ | 14578 SORT
1696 |              SORTNUM(C+1)=TEMP;                               /*NEWQUAD*/ | 14606 SORT
1697 |              TEMP=SORTREF(C);                                 /*NEWQUAD*/ | 14626 SORT
1698 |              SORTREF(C)=SORTREF(C+1);                         /*NEWQUAD*/ | 14642 SORT
1699 |              SORTREF(C+1)=TEMP;                               /*NEWQUAD*/ | 14670 SORT
1700 |              TEMP=REGNO(C);                                   /*NEWQUAD*/ | 14690 SORT
1701 |              REGNO(C)=REGNO(C+1);                             /*NEWQUAD*/ | 14706 SORT
1702 |              REGNO(C+1)=TEMP;                                 /*NEWQUAD*/ | 14734 SORT
1703 |              SWITCH=TRUE;                                     /*NEWQUAD*/ | 14754 SORT
1704 |              END;                                             /*NEWQUAD*/ | 14762 SORT
1705 |           END;                                                /*NEWQUAD*/ | 14762 SORT
1706 |        END;                                                   /*NEWQUAD*/ | 14762 SORT
1707 |     D=D+1;                                                    /*NEWQUAD*/ | 14770 SORT
1708 |     END;                                                      /*NEWQUAD*/ | 14782 SORT
```

```
1709 |     D = D;                                                          /*NEWQUAD*/ | 14796 SORT
1710 |     DO C = 0 TO A - 1;                                              /*NEWQUAD*/ | 14796 SORT
1711 |        IF B=TRUE THEN DO;                                           /*NEWQUAD*/ | 14842 SORT
1712 |           IF SORTNUM(C) = SORTNUM(C+1) THEN                         /*NEWQUAD*/ | 14860 SORT
1713 |              D = D + 1;                                             /*NEWQUAD*/ | 14904 SORT
1714 |           ELSE                                                      /*NEWQUAD*/ | 14908 SORT
1715 |              RETURN D;                                              /*NEWQUAD*/ | 14908 SORT
1716 |           END;                                                      /*NEWQUAD*/ | 14926 SORT
1717 |        ELSE DO;                                                     /*NEWQUAD*/ | 14926 SORT
1718 |           IF SORTREF(C) = SORTREF(C+1) THEN                         /*NEWQUAD*/ | 14934 SORT
1719 |              D = D + 1;                                             /*NEWQUAD*/ | 14978 SORT
1720 |           ELSE                                                      /*NEWQUAD*/ | 14992 SORT
1721 |              RETURN D;                                              /*NEWQUAD*/ | 14982 SORT
1722 |           END;                                                      /*NEWQUAD*/ | 15000 SORT
1723 |        END;                                                         /*NEWQUAD*/ | 15000 SORT
1724 |     RETURN D;                                                       /*NEWQUAD*/ | 15008 SORT
1725 |     END;                                                            /*NEWQUAD*/ | 15018 SORT
1726 | DEALLOCATE:PROCEDURE;                                               /*NEWQUAD*/ | 15024
1727 |                                                                                 | 15024 DEALLOCAT
1728 | /*DEALLOCTE  DEALLOCTES A REGISTER FROM VARIABLES ASSIGNED TO IT                 | 15024 DEALLOCAT
1729 |              1- DEALLOCATE THE REGISTER WITH NO TEMPORARY VARIABLE AND          | 15024 DEALLOCAT
1730 |                 LEAST NUMBER OF VARIABLE.                                       | 15024 DEALLOCAT
1731 |              2- IF TWO REGISTER HAVE THE SAME NUMBER OF VARIABLES ASSIGNED      | 15024 DEALLOCAT
1732 |                 TO THEM, THEN PICK THE ONE THAT HAS NOT BEEN REFRENCED           | 15024 DEALLOCAT
1733 |                 LATELY.                                                         | 15024 DEALLOCAT
1734 |              3- IF NO REGISTER IS FOUND WITH NO TEMPORARY VARIABLE, THEN        | 15024 DEALLOCAT
1735 |                 PICK THE REGISTER THAT HAS LEAST NUMBER OF TEMPORARY VARIABLE    | 15024 DEALLOCAT
1736 |                    */                                                           | 15024 DEALLOCAT
1737 |                                                                                 | 15024 DEALLOCAT
1738 |  DECLARE(TEMP,REG,MINIMUM)FIXED;                                    /*NEWQUAD*/ | 15024 DEALLOCAT
1739 |  DECLARE TEMPVARNUM(6) FIXED;                                       /*NEWQUAD*/ | 15036 DEALLOCAT
1740 |  DECLARE TEMPREGNO(6) FIXED;                                        /*NEWQUAD*/ | 15036 DEALLOCAT
1741 |  KK = 0;                                                            /*NEWQUAD*/ | 15036 DEALLOCAT
1742 |  JJ = 0;                                                            /*NEWQUAD*/ | 15042 DEALLOCAT
1743 |  DO II = 1 TO MAXREG;                                               /*NEWQUAD*/ | 15048 DEALLOCAT
1744 |     IF DEALLOCABL(II) = FALSE THEN DO;                              /*NEWQUAD*/ | 15092 DEALLOCAT
1745 |        IF TEMP(II) = 0 THEN DO;                                     /*NEWQUAD*/ | 15114 DEALLOCAT
1746 |           IF VARNUM(II) = 1 THEN DO;                                /*NEWQUAD*/ | 15138 DEALLOCAT
1747 |              LL = POINT(II);                                        /*NEWQUAD*/ | 15162 DEALLOCAT
1748 |              IF BYTE(VARIBLS(LL),0) >= BYTE('0') &                  /*NEWQUAD*/ | 15178 DEALLOCAT
1749 |                 BYTE(VARIBLS(LL),0) <= BYTE('9') THEN DO;           /*NEWQUAD*/ | 15216 DEALLOCAT
1750 |                 NEXT_VAR(LL) = LAVS;                                /*NEWQUAD*/ | 15268 DEALLOCAT
1751 |                 LAVS = LL;                                          /*NEWQUAD*/ | 15284 DEALLOCAT
1752 |                 RD = TRUE;                                          /*NEWQUAD*/ | 15292 DEALLOCAT
1753 |                 RETURN II;                                          /*NEWQUAD*/ | 15300 DEALLOCAT
1754 |                 END;                                                /*NEWQUAD*/ | 15310 DEALLOCAT
1755 |              ELSE DO;                                               /*NEWQUAD*/ | 15310 DEALLOCAT
1756 |                 IF CHANGE(LL) = FALSE THEN DO;                      /*NEWQUAD*/ | 15318 DEALLOCAT
1757 |                    NEXT_VAR(LL) = LAVS;                             /*NEWQUAD*/ | 15340 DEALLOCAT
1758 |                   LAVS = LL;                                        /*NEWQUAD*/ | 15356 DEALLOCAT
1759 |                    RD = TRUE;                                       /*NEWQUAD*/ | 15364 DEALLOCAT
1760 |                    RETURN II;                                       /*NEWQUAD*/ | 15372 DEALLOCAT
1761 |                    END;                                             /*NEWQUAD*/ | 15382 DEALLOCAT
1762 |                 SORTREF(JJ) = REFRENCE(II);                         /*NEWQUAD*/ | 15382 DEALLOCAT
1763 |                 SORTNUM(JJ) = VARNUM(II);                           /*NEWQUAD*/ | 15406 DEALLOCAT
1764 |                 REGNO(JJ) = II;                                     /*NEWQUAD*/ | 15430 DEALLOCAT
1765 |                 JJ = JJ + 1;                                        /*NEWQUAD*/ | 15446 DEALLOCAT
1766 |                 END;                                                /*NEWQUAD*/ | 15458 DEALLOCAT
1767 |              END;                                                   /*NEWQUAD*/ | 15458 DEALLOCAT
1768 |           ELSE DO;                                                  /*NEWQUAD*/ | 15458 DEALLOCAT
1769 |              SORTREF(JJ) = REFRENCE(II);                            /*NEWQUAD*/ | 15466 DEALLOCAT
1770 |              SORTNUM(JJ) = VARNUM(II);                              /*NEWQUAD*/ | 15490 DEALLOCAT
1771 |              REGNO(JJ) = II;                                        /*NEWQUAD*/ | 15514 DEALLOCAT
1772 |              JJ = JJ +1;                                            /*NEWQUAD*/ | 15530 DEALLOCAT
1773 |              END;                                                   /*NEWQUAD*/ | 15542 DEALLOCAT
1774 |           END;                                                      /*NEWQUAD*/ | 15542 DEALLOCAT
```

```
1774 |          ELSE DO;                                                    /*NEWQUAD*/ | 15542 DEALLOCATE
1775 |             TEMPVARNUM(KK) = VARNUM(II);                             /*NEWQUAD*/ | 15550 DEALLOCATE
1777 |             TEMPREGNO(KK) = II;                                      /*NEWQUAD*/ | 15574 DEALLOCATE
1778 |             KK = KK + 1;                                             /*NEWQUAD*/ | 15590 DEALLOCATE
1779 |             END;                                                     /*NEWQUAD*/ | 15602 DEALLOCATE
1780 |          END;                                                        /*NEWQUAD*/ | 15602 DEALLOCATE
1781 |       END;                                                           /*NEWQUAD*/ | 15602 DEALLOCATE
1782 |    IF JJ = 0 THEN DO;                                                /*NEWQUAD*/ | 15610 DEALLOCATE
1783 |       MINIMUM = 100;                                                 /*NEWQUAD*/ | 15626 DEALLOCATE
1784 |       DO II = 0 TO KK -1;                                            /*NEWQUAD*/ | 15634 DEALLOCATE
1785 |          IF TEMPVARNUM(II) < MINIMUM THEN DO;                        /*NEWQUAD*/ | 15650 DEALLOCATE
1786 |             MINIMUM = TEMPVARNUM(II);                                /*NEWQUAD*/ | 15704 DEALLOCATE
1787 |             REG = TEMPREGNO(II);                                     /*NEWQUAD*/ | 15720 DEALLOCATE
1788 |             END;                                                     /*NEWQUAD*/ | 15736 DEALLOCATE
1789 |          END;                                                        /*NEWQUAD*/ | 15736 DEALLOCATE
1790 |       END;                                                           /*NEWQUAD*/ | 15744 DEALLOCATE
1791 |    ELSE DO;                                                          /*NEWQUAD*/ | 15744 DEALLOCATE
1792 |       IF JJ >= 1 THEN DO;                                            /*NEWQUAD*/ | 15752 DEALLOCATE
1793 |          II = SORT(JJ-1,TRUE);                                       /*NEWQUAD*/ | 15768 DEALLOCATE
1794 |          KK= SORT(II,FALSE);                                         /*NEWQUAD*/ | 15800 DEALLOCATE
1795 |          REG = REGNO(0);                                             /*NEWQUAD*/ | 15826 DEALLOCATE
1796 |          END;                                                        /*NEWQUAD*/ | 15840 DEALLOCATE
1797 |       ELSE                                                           /*NEWQUAD*/ | 15840 DEALLOCATE
1798 |       REG = REGNO(0);                                                /*NEWQUAD*/ | 15840 DEALLOCATE
1799 |       END;                                                           /*NEWQUAD*/ | 15862 DEALLOCATE
1800 |    LL = POINT(REG);                                                  /*NEWQUAD*/ | 15862 DEALLOCATE
1801 |    DO II =  1 TO VARNUM(REG);                                        /*NEWQUAD*/ | 15878 DEALLOCATE
1802 |       IF CHANGE(LL) = TRUE THEN DO;                                  /*NEWQUAD*/ | 15930 DEALLOCATE
1803 |                                                                                  | 15952 DEALLOCATE
1804 |    /*GENERATE A MT REGISTER QUAD  */                                            | 15952 DEALLOCATE
1805 |                                                                                  | 15952 DEALLOCATE
1806 |          TEMP_CHAR=VARIBLS(LL);                                      /*NEWQUAD*/ | 15952 DEALLOCATE
1807 |          OPRTR=26;                                                   /*NEWQUAD*/ | 15968 DEALLOCATE
1808 |          OPRND1=REG;                                                 /*NEWQUAD*/ | 15976 DEALLOCATE
1809 |          OPRND2=0;                                                   /*NEWQUAD*/ | 15984 DEALLOCATE
1810 |          RSLT=SYMB_INDEX;                                            /*NEWQUAD*/ | 15990 DEALLOCATE
1811 |          CURR_MIC=CURR_MIC+4;                                        /*NEWQUAD*/ | 16002 DEALLOCATE
1812 |          CALL PUT_NEWQUAD;                                           /*NEWQUAD*/ | 16014 DEALLOCATE
1813 |          END;                                                        /*NEWQUAD*/ | 16022 DEALLOCATE
1814 |       TEMP = NEXT_VAR(LL);                                           /*NEWQUAD*/ | 16022 DEALLOCATE
1815 |       NEXT_VAR(LL) = LAVS;                                           /*NEWQUAD*/ | 16038 DEALLOCATE
1816 |       LAVS = LL;                                                     /*NEWQUAD*/ | 16054 DEALLOCATE
1817 |       LL = TEMP;                                                     /*NEWQUAD*/ | 16062 DEALLOCATE
1818 |       END;                                                           /*NEWQUAD*/ | 16070 DEALLOCATE
1819 |    RD = TRUE;                                                        /*NEWQUAD*/ | 16078 DEALLOCATE
1820 |    RETURN REG;                                                       /*NEWQUAD*/ | 16086 DEALLOCATE
1821 |    END;                                                              /*NEWQUAD*/ | 16096 DEALLOCATE
1822 | DEALLOC_TEMP:PROCEDURE(OPERAND,REG);                                 /*NEWQUAD*/ | 16102
1823 |                                                                                  | 16102 DEALLOC_TE
1824 | /*DEALLOC_TEMP   DELETE TEMPORARY VARIABLE FROM LIST OF VARIABLES                | 16102 DEALLOC_TE
1825 |                   ASSIGNED TO THE REGISTER   */                                  | 16102 DEALLOC_TE
1826 |                                                                                  | 16102 DEALLOC_TE
1827 |    DECLARE(REG,PRE_VAR)FIXED;                                        /*NEWQUAD*/ | 16102 DEALLOC_TE
1828 |    DECLARE(OPERAND)CHARACTER;                                        /*NEWQUAD*/ | 16114 DEALLOC_TE
1829 |    DEALLOCABL(REG) = FALSE;                                          /*NEWQUAD*/ | 16114 DEALLOC_TE
1830 |    IF BYTE(OPERAND,0) = BYTE('.') THEN DO;                           /*NEWQUAD*/ | 16124 DEALLOC_TE
1831 |       LL = POINT(REG);                                               /*NEWQUAD*/ | 16146 DEALLOC_TE
1832 |       DO II = 1 TO VARNUM(REG);                                      /*NEWQUAD*/ | 16162 DEALLOC_TE
1833 |          IF VARIBLS(LL) = OPERAND THEN DO;                           /*NEWQUAD*/ | 16214 DEALLOC_TE
1834 |             IF LL =POINT(REG) THEN                                   /*NEWQUAD*/ | 16264 DEALLOC_TE
1835 |                POINT(REG) = NEXT_VAR(LL);                            /*NEWQUAD*/ | 16296 DEALLOC_TE
1836 |             ELSE                                                     /*NEWQUAD*/ | 16312 DEALLOC_TE
1837 |                NEXT_VAR(PRE_VAR) = NEXT_VAR(LL);                     /*NEWQUAD*/ | 16312 DEALLOC_TE
1838 |             NEXT_VAR(LL) = LAVS;                                     /*NEWQUAD*/ | 16344 DEALLOC_TE
1839 |             LAVS = LL;                                               /*NEWQUAD*/ | 16360 DEALLOC_TE
1840 |             II = VARNUM(REG) + 1;                                    /*NEWQUAD*/ | 16368 DEALLOC_TE
```

```
1942 |         ELSE DO;                                                    /*NEWQUAD*/ | 16388 DEALLOC_TEM
1943 |            PRE_VAR = LL;                                            /*NEWQUAD*/ | 16396 DEALLOC_TEM
1944 |            LL = NEXT_VAR(LL);                                       /*NEWQUAD*/ | 16404 DEALLOC_TEM
1945 |            END;                                                     /*NEWQUAD*/ | 16420 DEALLOC_TEM
1946 |         END;                                                        /*NEWQUAD*/ | 16420 DEALLOC_TEM
1947 |      IF VARNUM(REG) ¬= 1 THEN DO;                                   /*NEWQUAD*/ | 16428 DEALLOC_TEM
1948 |         TEMP(REG) = TEMP(REG) - 1;                                  /*NEWQUAD*/ | 16452 DEALLOC_TEM
1949 |         VARNUM(REG) = VARNUM(REG) - 1;                              /*NEWQUAD*/ | 16480 DEALLOC_TEM
1950 |         END;                                                        /*NEWQUAD*/ | 16508 DEALLOC_TEM
1951 |      ELSE                                                           /*NEWQUAD*/ | 16508 DEALLOC_TEM
1952 |         STATUS(REG) = FALSE;                                        /*NEWQUAD*/ | 16508 DEALLOC_TEM
1953 |      END;                                                           /*NEWQUAD*/ | 16526 DEALLOC_TEM
1954 |   END;                                                              /*NEWQUAD*/ | 16526 DEALLOC_TEM
1955 | FIND_RES:PROCEDURE(RESULT);                                         /*NEWQUAD*/ | 16532
1956 |                                                                               | 16532 FIND_RES
1957 | /*FIND_RES  SEARCH REGISTER TABLE FOR RESULT VARIABLE                         | 16532 FIND_RES
1958 |             1- IF FOUND AND THE ONLY VARIABLE ASSIGNED TO THE REGISTER,       | 16532 FIND_RES
1959 |                THEN RETURN REG#                                               | 16532 FIND_RES
1960 |             2- IF FOUND AND MORE THAN ONE VARIABLE, THEN DELETE  VARIABLE     | 16532 FIND_RES
1961 |                FROM LIST OF VARIABLES ASSIGNED TO THE REGISTER AND GO TO 3.    | 16532 FIND_RES
1962 |             3- RETURN REG# OF A FREE REGISTER , IF NOT FOUND RETURN -1  */    | 16532 FIND_RES
1963 |                                                                               | 16532 FIND_RES
1964 |   DECLARE(RESULT)CHARACTER;                                         /*NEWQUAD*/ | 16532 FIND_RES
1965 |   DECLARE(PRE_VAR)FIXED;                                            /*NEWQUAD*/ | 16544 FIND_RES
1966 |   DO II = 1 TO MAXREG;                                              /*NEWQUAD*/ | 16544 FIND_RES
1967 |      IF STATUS(II) = TRUE THEN DO;                                  /*NEWQUAD*/ | 16588 FIND_RES
1968 |         LL = POINT(II);                                             /*NEWQUAD*/ | 16610 FIND_RES
1969 |         DO JJ = 1 TO VARNUM(II);                                    /*NEWQUAD*/ | 16626 FIND_RES
1970 |            IF VARIBLS(LL) = RESULT THEN DO;                         /*NEWQUAD*/ | 16678 FIND_RES
1971 |               IF VARNUM(II) ¬= 1 THEN DO;                           /*NEWQUAD*/ | 16728 FIND_RES
1972 |                  VARNUM(II) = VARNUM(II) -1;                        /*NEWQUAD*/ | 16752 FIND_RES
1973 |                  IF BYTE(RESULT,0)=BYTE('.') THEN                   /*NEWQUAD*/ | 16780 FIND_RES
1974 |                     TEMP(II) = TEMP(II) - 1;                        /*NEWQUAD*/ | 16810 FIND_RES
1975 |                  IF LL = POINT(II) THEN                             /*NEWQUAD*/ | 16830 FIND_RES
1976 |                     POINT(II) = NEXT_VAR(LL);                       /*NEWQUAD*/ | 16862 FIND_RES
1977 |                  ELSE                                               /*NEWQUAD*/ | 16878 FIND_RES
1978 |                     NEXT_VAR(PRE_VAR) = NEXT_VAR(LL);               /*NEWQUAD*/ | 16878 FIND_RES
1979 |                  NEXT_VAR(LL) = LAVS;                               /*NEWQUAD*/ | 16910 FIND_RES
1980 |                  LAVS = LL;                                         /*NEWQUAD*/ | 16926 FIND_RES
1981 |                  JJ = VARNUM(II) + 1;                               /*NEWQUAD*/ | 16934 FIND_RES
1982 |                  II = MAXREG + 1;                                   /*NEWQUAD*/ | 16954 FIND_RES
1983 |                  END;                                               /*NEWQUAD*/ | 16966 FIND_RES
1984 |               ELSE DO;                                              /*NEWQUAD*/ | 16966 FIND_RES
1985 |                  RD = FALSE;                                        /*NEWQUAD*/ | 16974 FIND_RES
1986 |                  RETURN II;                                         /*NEWQUAD*/ | 16980 FIND_RES
1987 |                  END;                                               /*NEWQUAD*/ | 16990 FIND_RES
1988 |               END;                                                  /*NEWQUAD*/ | 16990 FIND_RES
1989 |            ELSE DO;                                                 /*NEWQUAD*/ | 16990 FIND_RES
1990 |               PRE_VAR = LL;                                         /*NEWQUAD*/ | 16998 FIND_RES
1991 |               LL = NEXT_VAR(LL);                                    /*NEWQUAD*/ | 17006 FIND_RES
1992 |               END;                                                  /*NEWQUAD*/ | 17022 FIND_RES
1993 |            END;                                                     /*NEWQUAD*/ | 17022 FIND_RES
1994 |         END;                                                        /*NEWQUAD*/ | 17030 FIND_RES
1995 |      END;                                                           /*NEWQUAD*/ | 17030 FIND_RES
1996 |   RD = TRUE;                                                        /*NEWQUAD*/ | 17038 FIND_RES
1997 |   DO II = 1 TO MAXREG;                                              /*NEWQUAD*/ | 17046 FIND_RES
1998 |      IF STATUS(II) = FALSE THEN DO;                                 /*NEWQUAD*/ | 17090 FIND_RES
1999 |         RETURN II;                                                  /*NEWQUAD*/ | 17112 FIND_RES
2000 |         END;                                                        /*NEWQUAD*/ | 17122 FIND_RES
2001 |      END;                                                           /*NEWQUAD*/ | 17122 FIND_RES
2002 |   RETURN -1;                                                        /*NEWQUAD*/ | 17130 FIND_RES
2003 |   END;                                                              /*NEWQUAD*/ | 17140 FIND_RES
2004 | ALLOC_REG_RES:PROCEDURE(RESULT);                                    /*NEWQUAD*/ | 17146
2005 |                                                                               | 17146 ALLOC_REG_RE
2006 | /*ALLOC_REG_RES  ALLOCATE A REGISTER TO A RESULT VARIABLE      */              | 17146 ALLOC_REG_RE
```

```
1907 |                                                                                                    | 17140 ALLOC_REG_
1908 |  DECLARE(RESULT)CHARACTER;                                                  /*NEWQUAD*/ | 17146 ALLOC_REG_RE
1909 |  DECLARE(REG)FIXED;                                                         /*NEWQUAD*/ | 17158 ALLOC_REG_RE
1910 |  REG = FIND_RES(RESULT);                                                    /*NEWQUAD*/ | 17158 ALLOC_REG_RE
1911 |  IF REG = -1 THEN                                                           /*NEWQUAD*/ | 17178 ALLOC_REG_RE
1912 |     REG = DEALLOCATE;                                                       /*NEWQUAD*/ | 17202 ALLOC_REG_RE
1913 |  IF RD = TRUE THEN DO;                                                      /*NEWQUAD*/ | 17206 ALLOC_REG_RE
1914 |     STATUS(REG) = TRUE;                                                     /*NEWQUAD*/ | 17224 ALLOC_REG_RE
1915 |     VARNUM(REG) = 1;                                                        /*NEWQUAD*/ | 17236 ALLOC_REG_RE
1916 |     REFRENCE(REG) = REFNO;                                                  /*NEWQUAD*/ | 17252 ALLOC_REG_RE
1917 |     REFNO = REFNO + 1;                                                      /*NEWQUAD*/ | 17268 ALLOC_REG_RE
1918 |     DEALLOCABL(REG) = FALSE;                                                /*NEWQUAD*/ | 17280 ALLOC_REG_RE
1919 |     IF BYTE(RESULT,0)=BYTE('.') THEN                                        /*NEWQUAD*/ | 17290 ALLOC_REG_RE
1920 |        TEMP(REG) = 1;                                                       /*NEWQUAD*/ | 17320 ALLOC_REG_RE
1921 |     ELSE                                                                    /*NEWQUAD*/ | 17328 ALLOC_REG_RE
1922 |        TEMP(REG) =0;                                                        /*NEWQUAD*/ | 17328 ALLOC_REG_RE
1923 |     LL,POINT(REG) = LAVS;                                                   /*NEWQUAD*/ | 17350 ALLOC_REG_RE
1924 |     LAVS = NEXT_VAR(LAVS);                                                  /*NEWQUAD*/ | 17370 ALLOC_REG_RE
1925 |     VARIBLS(LL) = RESULT;                                                   /*NEWQUAD*/ | 17386 ALLOC_REG_RE
1926 |     NEXT_VAR(LL) = 0;                                                       /*NEWQUAD*/ | 17402 ALLOC_REG_RE
1927 |     CHANGE(LL) = TRUE;                                                      /*NEWQUAD*/ | 17416 ALLOC_REG_RE
1928 |     END;                                                                    /*NEWQUAD*/ | 17428 ALLOC_REG_RE
1929 |  ELSE DO;                                                                   /*NEWQUAD*/ | 17428 ALLOC_REG_RE
1930 |     LL = POINT(REG);                                                        /*NEWQUAD*/ | 17436 ALLOC_REG_RE
1931 |     CHANGE(LL) = TRUE;                                                      /*NEWQUAD*/ | 17452 ALLOC_REG_RE
1932 |     END;                                                                    /*NEWQUAD*/ | 17464 ALLOC_REG_RE
1933 |  RETURN REG;                                                                /*NEWQUAD*/ | 17464 ALLOC_REG_RE
1934 |  END;                                                                       /*NEWQUAD*/ | 17474 ALLOC_REG_RE
1935 | FIND_OP:PROCEDURE(OPERAND);                                                 /*NEWQUAD*/ | 17480
1936 |                                                                                       | 17480 FIND_OP
1937 | /*FIND_OP  SEARCH REGISTER TABLE FOR OPERAND VARIABLE                                 | 17480 FIND_OP
1938 |            1- IF FOUND RETURN REG#                                                   | 17480 FIND_OP
1939 |            2- IF NOT FOUND RETURN REG# OF A FREE REGISTER                            | 17480 FIND_OP
1940 |            3- IF NO FREE REGISTER IS AVAILABLE RETURN -1       */                   | 17480 FIND_OP
1941 |                                                                                       | 17480 FIND_OP
1942 |  DECLARE(NOT_ALLOCATED)FIXED;                                               /*NEWQUAD*/ | 17480 FIND_OP
1943 |  DECLARE(OPERAND)CHARACTER;                                                 /*NEWQUAD*/ | 17492 FIND_OP
1944 |  NOT_ALLOCATED = -1;                                                        /*NEWQUAD*/ | 17492 FIND_OP
1945 |  DO II = 1 TO MAXREG;                                                       /*NEWQUAD*/ | 17500 FIND_OP
1946 |     IF STATUS(II) = TRUE THEN DO;                                           /*NEWQUAD*/ | 17544 FIND_OP
1947 |        LL = POINT(II);                                                      /*NEWQUAD*/ | 17566 FIND_OP
1948 |        DO JJ = 1 TO VARNUM(II);                                             /*NEWQUAD*/ | 17582 FIND_OP
1949 |           IF VARIBLS(LL) = OPERAND THEN DO;                                 /*NEWQUAD*/ | 17634 FIND_OP
1950 |              RD = FALSE;                                                    /*NEWQUAD*/ | 17684 FIND_OP
1951 |              RETURN II;                                                     /*NEWQUAD*/ | 17690 FIND_OP
1952 |              END;                                                           /*NEWQUAD*/ | 17700 FIND_OP
1953 |           ELSE                                                              /*NEWQUAD*/ | 17700 FIND_OP
1954 |              LL = NEXT_VAR(LL);                                             /*NEWQUAD*/ | 17700 FIND_OP
1955 |           END;                                                              /*NEWQUAD*/ | 17724 FIND_OP
1956 |        END;                                                                 /*NEWQUAD*/ | 17732 FIND_OP
1957 |     ELSE                                                                    /*NEWQUAD*/ | 17732 FIND_OP
1958 |        NOT_ALLOCATED = II;                                                  /*NEWQUAD*/ | 17732 FIND_OP
1959 |     END;                                                                    /*NEWQUAD*/ | 17748 FIND_OP
1960 |  RD = TRUE;                                                                 /*NEWQUAD*/ | 17756 FIND_OP
1961 |  RETURN NOT_ALLOCATED;                                                      /*NEWQUAD*/ | 17764 FIND_OP
1962 |  END;                                                                       /*NEWQUAD*/ | 17774 FIND_OP
1963 | ALLOC_REG_OP:PROCEDURE(OPERAND);                                            /*NEWQUAD*/ | 17780
1964 |                                                                                       | 17780 ALLOC_REG_O
1965 | /*ALLOC_REG_OP  ALLOCATE A REGISTER TO AND OPERAND VARIABLE                           | 17780 ALLOC_REG_O
1966 |                 1- IF ALREADY ASSIGNED RETURN REG#                                    | 17780 ALLOC_REG_O
1967 |                 2- IF NOT GENERATE A RD OR RDAD REGISTER QUAD                         | 17780 ALLOC_REG_O
1968 |                    AND RETURN REG#       */                                         | 17780 ALLOC_REG_O
1969 |                                                                                       | 17780 ALLOC_REG_O
1970 |  DECLARE(REG)FIXED;                                                         /*NEWQUAD*/ | 17780 ALLOC_REG_O
1971 |  DECLARE(OPERAND)CHARACTER;                                                 /*NEWQUAD*/ | 17792 ALLOC_REG_O
1972 |  REG = FIND_OP(OPERAND);                                                    /*NEWQUAD*/ | 17792 ALLOC_REG_O
```

```
1973 |  IF REG = -1 THEN                                                    /*NEWQUAD*/ | 17812 ALLOC_REG_0
1974 |     REG = DEALLOCATE;                                                /*NEWQUAD*/ | 17836 ALLOC_REG_0
1975 |  DEALLOCABL(REG) = TRUE;                                             /*NEWQUAD*/ | 17840 ALLOC_REG_0
1976 |  REFRENCE(REG) = REFNO;                                              /*NEWQUAD*/ | 17852 ALLOC_REG_0
1977 |  REFNO = REFNO + 1;                                                  /*NEWQUAD*/ | 17868 ALLOC_REG_0
1978 |  IF RD = TRUE THEN DO;                                               /*NEWQUAD*/ | 17880 ALLOC_REG_0
1979 |     STATUS(REG) = TRUE;                                              /*NEWQUAD*/ | 17898 ALLOC_REG_0
1980 |     VARNUM(REG) = 1;                                                 /*NEWQUAD*/ | 17910 ALLOC_REG_0
1981 |     LL,POINT(REG) = LAVS;                                            /*NEWQUAD*/ | 17926 ALLOC_REG_0
1982 |     LAVS = NEXT_VAR(LAVS);                                           /*NEWQUAD*/ | 17946 ALLOC_REG_0
1983 |     VARIBLS(LL) = OPERAND;                                           /*NEWQUAD*/ | 17962 ALLOC_REG_0
1984 |     NEXT_VAR(LL) = 0;                                                /*NEWQUAD*/ | 17978 ALLOC_REG_0
1985 |     CHANGE(LL) = FALSE;                                              /*NEWQUAD*/ | 17992 ALLOC_REG_0
1986 |     IF BYTE(OPERAND,0) = BYTE('.') THEN                              /*NEWQUAD*/ | 18002 ALLOC_REG_0
1987 |        TEMP(REG) = 1;                                                /*NEWQUAD*/ | 18032 ALLOC_REG_0
1988 |     ELSE                                                             /*NEWQUAD*/ | 18040 ALLOC_REG_0
1989 |        TEMP(REG)=0;                                                  /*NEWQUAD*/ | 18040 ALLOC_REG_0
1990 |     IF OPER=SUBS | OPER=SUBL THEN DO;                                /*NEWQUAD*/ | 18062 ALLOC_REG_0
1991 |        OPRTOR=27;                                                    /*NEWQUAD*/ | 18124 ALLOC_REG_0
1992 |        TEMP_CHAR=OPERAND;                                            /*NEWQUAD*/ | 18132 ALLOC_REG_0
1993 |        OPRND1=SYMB_INDEX;                                            /*NEWQUAD*/ | 18140 ALLOC_REG_0
1994 |        OPRND2=0;                                                     /*NEWQUAD*/ | 18152 ALLOC_REG_0
1995 |        RSLT=REG;                                                     /*NEWQUAD*/ | 18158 ALLOC_REG_0
1996 |        CURR_MIC=CURR_MIC+2;                                          /*NEWQUAD*/ | 18166 ALLOC_REG_0
1997 |        CALL PUT_NEWQUAD;                                             /*NEWQUAD*/ | 18178 ALLOC_REG_0
1998 |        END;                                                          /*NEWQUAD*/ | 18186 ALLOC_REG_0
1999 |     ELSE  DO;                                                        /*NEWQUAD*/ | 18186 ALLOC_REG_0
2000 |        TEMP_CHAR=OPERAND;                                            /*NEWQUAD*/ | 18194 ALLOC_REG_0
2001 |        IF BYTE(OPERAND,0)>= BYTE('0') &                              /*NEWQUAD*/ | 18202 ALLOC_REG_0
2002 |           BYTE(OPERAND,0) <= BYTE('9') THEN                          /*NEWQUAD*/ | 18232 ALLOC_REG_0
2003 |           OPRND1=-CONVAL_INDEX;                                      /*NEWQUAD*/ | 18264 ALLOC_REG_0
2004 |        ELSE                                                          /*NEWQUAD*/ | 18290 ALLOC_REG_0
2005 |          OPRND1=SYMB_INDEX;                                          /*NEWQUAD*/ | 18290 ALLOC_REG_0
2006 |        OPRTOR=25;                                                    /*NEWQUAD*/ | 18310 ALLOC_REG_0
2007 |        OPRND2=0;                                                     /*NEWQUAD*/ | 18318 ALLOC_REG_0
2008 |        RSLT=REG;                                                     /*NEWQUAD*/ | 18324 ALLOC_REG_0
2009 |        CURR_MIC=CURR_MIC+4;                                          /*NEWQUAD*/ | 18332 ALLOC_REG_0
2010 |        CALL PUT_NEWQUAD;                                             /*NEWQUAD*/ | 18344 ALLOC_REG_0
2011 |        END;                                                          /*NEWQUAD*/ | 18352 ALLOC_REG_0
2012 |     END;                                                             /*NEWQUAD*/ | 18352 ALLOC_REG_0
2013 |  RETURN REG;                                                         /*NEWQUAD*/ | 18352 ALLOC_REG_0
2014 |  END;       ;                                                        /*NEWQUAD*/ | 18362 ALLOC_REG_0
2015 | WRITE_REGS:PROCEDURE;                                                /*NEWQUAD*/ | 18368
2016 |                                                                                  | 18368 WRITE_REGS
2017 | /*WRITE REGS  GENERATE WT REGISTER QUADS FOR VARIABLES ASSIGNED                  | 18368 WRITE_REGS
2018 |                 TO REGISTERS                 */                                  | 18368 WRITE_REGS
2019 |                                                                                  | 18368 WRITE_REGS
2020 |  DO II= 1 TO MAXREG;                                                 /*NEWQUAD*/ | 18368 WRITE_REGS
2021 |     IF STATUS(II) = TRUE THEN DO;                                    /*NEWQUAD*/ | 18424 WRITE_REGS
2022 |        LL = POINT(II);                                               /*NEWQUAD*/ | 18446 WRITE_REGS
2023 |        DO JJ = 1 TO VARNUM(II);                                      /*NEWQUAD*/ | 18462 WRITE_REGS
2024 |           IF CHANGE(LL) = TRUE THEN DO;                              /*NEWQUAD*/ | 18514 WRITE_REGS
2025 |              CHANGE(LL) = FALSE;                                     /*NEWQUAD*/ | 18536 WRITE_REGS
2026 |              TEMP_CHAR=VARIBLS(LL);                                  /*NEWQUAD*/ | 18546 WRITE_REGS
2027 |              RSLT=SYMB_INDEX;                                        /*NEWQUAD*/ | 18562 WRITE_REGS
2028 |              OPRTOR=26;                                              /*NEWQUAD*/ | 18574 WRITE_REGS
2029 |              OPRND1=II;                                              /*NEWQUAD*/ | 18582 WRITE_REGS
2030 |              OPRND2=0;                                               /*NEWQUAD*/ | 18590 WRITE_REGS
2031 |              CURR_MIC=CURR_MIC+4;                                    /*NEWQUAD*/ | 18596 WRITE_REGS
2032 |              CALL PUT_NEWQUAD;                                       /*NEWQUAD*/ | 18608 WRITE_REGS
2033 |              END;                                                    /*NEWQUAD*/ | 18616 WRITE_REGS
2034 |           LL=NEXT_VAR(LL);                                           /*NEWQUAD*/ | 18616 WRITE_REGS
2035 |           END;                                                       /*NEWQUAD*/ | 18632 WRITE_REGS
2036 |        END;                                                          /*NEWQUAD*/ | 18640 WRITE_REGS
2037 |     END;                                                             /*NEWQUAD*/ | 18640 WRITE_REGS
2038 |  END;                                                                /*NEWQUAD*/ | 18648 WRITE_REGS
```

```
2039 | NEWQUAD_GEN:PROCEDURE;                                          /*NEWQUAD*/ | 18654
2040 |                                                                             | 18654 NEWQUAD_GE
2041 | /*NEWQUAD_GEN  REGISTER QUAD GENERATOR  */                                  | 18654 NEWQUAD_GE
2042 |                                                                             | 18654 NEWQUAD_GE
2043 |  DECLARE (REG1,REG2,REG3,PRE_VAR)FIXED;                         /*NEWQUAD*/ | 18654 NEWQUAD_GE
2044 |         DECLARE CASE_NUM(30) FIXED INITIAL(0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2045 |                                            1,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2046 |                                            1,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2047 |                                            1,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2048 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2049 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2050 |                                            3,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2051 |                                            6,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2052 |                                            2,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2053 |                                            2,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2054 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2055 |                                            1,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2056 |                                            1,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2057 |                                            1,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2058 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2059 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2060 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2061 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2062 |                                            5,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2063 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2064 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2065 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2066 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2067 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2068 |                                            4,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2069 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2070 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2071 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2072 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2073 |                                            0,                   /*NEWQUAD*/ | 18666 NEWQUAD_GE
2074 |                                            5);                  /*NEWQUAD*/ | 18666 NEWQUAD_GE
2075 |  DO CASE CASE_NUM(OPER);                                        /*NEWQUAD*/ | 18666 NEWQUAD_GE
2076 |     DO;                                                         /*NEWQUAD*/ | 18698 NEWQUAD_GE
2077 |                                                                             | 18698 NEWQUAD_GE
2078 | /*PROCESS ASIGN QUAD    */                                                  | 18698 NEWQUAD_GE
2079 |                                                                             | 18698 NEWQUAD_GE
2080 |        REG1=ALLOC_REG_OP(OPERAND1);                             /*NEWQUAD*/ | 18698 NEWQUAD_GE
2081 |        REG2 = FIND_RES(RESULT);                                 /*NEWQUAD*/ | 18718 NEWQUAD_GE
2082 |        IF ADDRSS(REG2)=TRUE THEN DO;                            /*NEWQUAD*/ | 18738 NEWQUAD_GE
2083 |           OPRTOR=0;                                             /*NEWQUAD*/ | 18760 NEWQUAD_GE
2084 |           OPRND1=REG1;                                          /*NEWQUAD*/ | 18766 NEWQUAD_GE
2085 |           OPRND2=0;                                             /*NEWQUAD*/ | 18774 NEWQUAD_GE
2086 |           RSLT=REG2;                                            /*NEWQUAD*/ | 18780 NEWQUAD_GE
2087 |           CALL PUT_NEWQUAD;                                     /*NEWQUAD*/ | 18788 NEWQUAD_GE
2088 |           CURR_MIC=CURR_MIC+2;                                  /*NEWQUAD*/ | 18796 NEWQUAD_GE
2089 |           ADDRSS(REG2)=FALSE;                                   /*NEWQUAD*/ | 18808 NEWQUAD_GE
2090 |           DEALLOCABL(REG2)=FALSE;                               /*NEWQUAD*/ | 18818 NEWQUAD_GE
2091 |           END;                                                  /*NEWQUAD*/ | 18828 NEWQUAD_GE
2092 |        ELSE DO;                                                 /*NEWQUAD*/ | 18826 NEWQUAD_GE
2093 |           VARNUM(REG1)=VARNUM(REG1)+1;                          /*NEWQUAD*/ | 18836 NEWQUAD_GE
2094 |           REFRENCE(REG1)=REFNO;                                 /*NEWQUAD*/ | 18864 NEWQUAD_GE
2095 |           REFNO=REFNO+1;                                        /*NEWQUAD*/ | 18880 NEWQUAD_GE
2096 |           IF BYTE(RESULT,0)=BYTE('.') THEN                      /*NEWQUAD*/ | 18902 NEWQUAD_GE
2097 |              TEMP(REG1)=TEMP(REG1)+1;                           /*NEWQUAD*/ | 18922 NEWQUAD_GE
2098 |           LL=LAVS;                                              /*NEWQUAD*/ | 18942 NEWQUAD_GE
2099 |           LAVS=NEXT_VAR(LAVS);                                  /*NEWQUAD*/ | 18950 NEWQUAD_GE
2100 |           NEXT_VAR(LL)=POINT(REG1);                             /*NEWQUAD*/ | 18966 NEWQUAD_GE
2101 |           POINT(REG1)=LL;                                       /*NEWQUAD*/ | 18990 NEWQUAD_GE
2102 |           VARIBLS(LL)=RESULT;                                   /*NEWQUAD*/ | 19006 NEWQUAD_GE
2103 |           CHANGE(LL)=TRUE;                                      /*NEWQUAD*/ | 19022 NEWQUAD_GE
2104 |           END;                                                  /*NEWQUAD*/ | 19034 NEWQUAD_GE
```

```
2106 |         CALL DEALLOC_TEMP(OPERAND1,REG1);                    /*NEWQUAD*/ | 19044 NEWQUAD_GE
2107 |         IF RD=FALSE THEN DO;                                 /*NEWQUAD*/ | 19068 NEWQUAD_GE
2108 |            LL=POINT(REG2);                                    /*NEWQUAD*/ | 19086 NEWQUAD_GE
2109 |            NEXT_VAR(LL)=LAVS;                                 /*NEWQUAD*/ | 19102 NEWQUAD_GE
2110 |            LAVS=LL;                                           /*NEWQUAD*/ | 19118 NEWQUAD_GE
2111 |            STATUS(REG2)=FALSE;                                /*NEWQUAD*/ | 19126 NEWQUAD_GE
2112 |            END;                                               /*NEWQUAD*/ | 19136 NEWQUAD_GE
2113 |         END;                                                  /*NEWQUAD*/ | 19136 NEWQUAD_GE
2114 |      DO;                                                      /*NEWQUAD*/ | 19136 NEWQUAD_GE
2115 |                                                                         | 19144 NEWQUAD_GE
2116 | /* PROCESS ADD, SUB, MUL, LT, GT, EQ QUADS     */                       | 19144 NEWQUAD_GE
2117 |                                                                         | 19144 NEWQUAD_GE
2118 |         REG1=ALLOC_REG_OP(OPERAND1);                          /*NEWQUAD*/ | 19144 NEWQUAD_GE
2119 |         REG2=ALLOC_REG_OP(OPERAND2);                          /*NEWQUAD*/ | 19164 NEWQUAD_GE
2120 |         REG3=ALLOC_REG_RES(RESULT);                           /*NEWQUAD*/ | 19184 NEWQUAD_GE
2121 |         OPRND1=REG1;                                          /*NEWQUAD*/ | 19204 NEWQUAD_GE
2122 |         OPRND2=REG2;                                          /*NEWQUAD*/ | 19212 NEWQUAD_GE
2123 |         RSLT=REG3;                                            /*NEWQUAD*/ | 19220 NEWQUAD_GE
2124 |            OPRTOR=OPER;                                       /*NEWQUAD*/ | 19228 NEWQUAD_GE
2125 |         CALL PUT_NEWQUAD;                                     /*NEWQUAD*/ | 19236 NEWQUAD_GE
2126 |         CALL DEALLOC_TEMP(OPERAND1,REG1);                     /*NEWQUAD*/ | 19244 NEWQUAD_GE
2127 |         CALL DEALLOC_TEMP(OPERAND2,REG2);                     /*NEWQUAD*/ | 19268 NEWQUAD_GE
2128 |         IF OPER=MUL THEN DO;                                  /*NEWQUAD*/ | 19292 NEWQUAD_GE
2129 |            IF CURR_MIC MOD 2 = 1 THEN                         /*NEWQUAD*/ | 19308 NEWQUAD_GE
2130 |               CURR_MIC=CURR_MIC+12;                           /*NEWQUAD*/ | 19342 NEWQUAD_GE
2131 |            ELSE                                               /*NEWQUAD*/ | 19346 NEWQUAD_GE
2132 |               CURR_MIC=CURR_MIC+13;                           /*NEWQUAD*/ | 19346 NEWQUAD_GE
2133 |            END;                                               /*NEWQUAD*/ | 19366 NEWQUAD_GE
2134 |         ELSE                                                  /*NEWQUAD*/ | 19366 NEWQUAD_GE
2135 |            CURR_MIC=CURR_MIC+3;                               /*NEWQUAD*/ | 19366 NEWQUAD_GE
2136 |         END;                                                  /*NEWQUAD*/ | 19386 NEWQUAD_GE
2137 |      DO;                                                      /*NEWQUAD*/ | 19386 NEWQUAD_GE
2138 |                                                                         | 19394 NEWQUAD_GE
2139 | /*PROCESS BT, BF QUADS      */                                          | 19394 NEWQUAD_GE
2140 |                                                                         | 19394 NEWQUAD_GE
2141 |         REG1=ALLOC_REG_OP(OPERAND1);                          /*NEWQUAD*/ | 19394 NEWQUAD_GE
2142 |         CALL DEALLOC_TEMP(OPERAND1,REG1);                     /*NEWQUAD*/ | 19414 NEWQUAD_GE
2143 |         CALL WRITE_REGS;                                      /*NEWQUAD*/ | 19438 NEWQUAD_GE
2144 |         OPRND1=REG1;                                          /*NEWQUAD*/ | 19446 NEWQUAD_GE
2145 |         TEMP_CHAR=RESULT;                                     /*NEWQUAD*/ | 19454 NEWQUAD_GE
2146 |         RSLT=LAB_INDEX;                                       /*NEWQUAD*/ | 19462 NEWQUAD_GE
2147 |         OPRND2=0;                                             /*NEWQUAD*/ | 19474 NEWQUAD_GE
2148 |            OPRTOR=OPER;                                       /*NEWQUAD*/ | 19480 NEWQUAD_GE
2149 |         PRE_VAR= CURR_MIC MOD 2;                              /*NEWQUAD*/ | 19438 NEWQUAD_GE
2150 |         IF PRE_VAR=0 THEN                                     /*NEWQUAD*/ | 19506 NEWQUAD_GE
2151 |            CURR_MIC=CURR_MIC+4;                               /*NEWQUAD*/ | 19530 NEWQUAD_GE
2152 |         ELSE                                                  /*NEWQUAD*/ | 19534 NEWQUAD_GE
2153 |            CURR_MIC=CURR_MIC+5;                               /*NEWQUAD*/ | 19534 NEWQUAD_GE
2154 |         CALL PUT_NEWQUAD;                                     /*NEWQUAD*/ | 19554 NEWQUAD_GE
2155 |         END;                                                  /*NEWQUAD*/ | 19562 NEWQUAD_GE
2156 |      DO;                                                      /*NEWQUAD*/ | 19562 NEWQUAD_GE
2157 |                                                                         | 19570 NEWQUAD_GE
2158 | /*PROCESS HALT QUADS     */                                             | 19570 NEWQUAD_GE
2159 |                                                                         | 19570 NEWQUAD_GE
2160 |         OPRND1=0;                                             /*NEWQUAD*/ | 19570 NEWQUAD_GE
2161 |         OPRND2=0;                                             /*NEWQUAD*/ | 19576 NEWQUAD_GE
2162 |         RSLT=0;                                               /*NEWQUAD*/ | 19582 NEWQUAD_GE
2163 |            OPRTOR=OPER;                                       /*NEWQUAD*/ | 19586 NEWQUAD_GE
2164 |         CALL PUT_NEWQUAD;                                     /*NEWQUAD*/ | 19596 NEWQUAD_GE
2165 |         END;                                                  /*NEWQUAD*/ | 19604 NEWQUAD_GE
2166 |      DO;                                                      /*NEWQUAD*/ | 19604 NEWQUAD_GE
2167 |                                                                         | 19612 NEWQUAD_GE
2168 | /*PROCESS LAB QUADS                          */                         | 19612 NEWQUAD_GE
2169 |                                                                         | 19612 NEWQUAD_GE
2170 |         CALL WRITE_REGS;                                      /*NEWQUAD*/ | 19612 NEWQUAD_GE
```

```
2171 |          TEMP_CHAR=OPERAND1;                                  /*NEWQUAD*/ | 19620 NEWQUAD_GE
2172 |          OPRND1=LAB_INDEX;                                    /*NEWQUAD*/ | 19628 NEWQUAD_GE
2173 |          OPRND2=0;                                            /*NEWQUAD*/ | 19640 NEWQUAD_GE
2174 |          RSLT=0;                                              /*NEWQUAD*/ | 19646 NEWQUAD_GE
2175 |             OPRTOR=OPER;                                      /*NEWQUAD*/ | 19652 NEWQUAD_GE
2176 |          MIC_LOC(OPRND1)=CURR_MIC;                            /*NEWQUAD*/ | 19660 NEWQUAD_GE
2177 |          CURR_MIC=CURR_MIC+2;                                 /*NEWQUAD*/ | 19676 NEWQUAD_GE
2178 |          CALL PUT_NEWQUAD;                                    /*NEWQUAD*/ | 19688 NEWQUAD_GE
2179 |          DO II=1 TO MAXREG;                                   /*NEWQUAD*/ | 19696 NEWQUAD_GE
2180 |             IF STATUS(II)=TRUE THEN DO;                       /*NEWQUAD*/ | 19740 NEWQUAD_GE
2181 |                STATUS(II)=FALSE;                              /*NEWQUAD*/ | 19762 NEWQUAD_GE
2182 |                LL=POINT(II);                                  /*NEWQUAD*/ | 19772 NEWQUAD_GE
2183 |                DO JJ=1 TO VARNUM(II);                         /*NEWQUAD*/ | 19788 NEWQUAD_GE
2184 |                   PRE_VAR=NEXT_VAR(LL);                       /*NEWQUAD*/ | 19840 NEWQUAD_GE
2185 |                   NEXT_VAR(LL)=LAVS;                          /*NEWQUAD*/ | 19856 NEWQUAD_GE
2186 |                   LAVS=LL;                                    /*NEWQUAD*/ | 19872 NEWQUAD_GE
2187 |                   LL=PRE_VAR;                                 /*NEWQUAD*/ | 19880 NEWQUAD_GE
2188 |                   END;                                        /*NEWQUAD*/ | 19888 NEWQUAD_GE
2189 |                END;                                           /*NEWQUAD*/ | 19896 NEWQUAD_GE
2190 |             END;                                              /*NEWQUAD*/ | 19896 NEWQUAD_GE
2191 |          END;                                                 /*NEWQUAD*/ | 19904 NEWQUAD_GE
2192 |       DO;                                                     /*NEWQUAD*/ | 19904 NEWQUAD_GE
2193 |                                                                           | 19912 NEWQUAD_GE
2194 |    /*PROCESS SUBS,SUBL QUADS      */                                     | 19912 NEWQUAD_GE
2195 |                                                                           | 19912 NEWQUAD_GE
2196 |          REG1=ALLOC_REG_OP(OPERAND1);                         /*NEWQUAD*/ | 19912 NEWQUAD_GE
2197 |          SUBSFLAG=OPER;                                       /*NEWQUAD*/ | 19932 NEWQUAD_GE
2198 |          OPER=0;                                              /*NEWQUAD*/ | 19940 NEWQUAD_GE
2199 |          REG2=ALLOC_REG_OP(OPERAND2);                         /*NEWQUAD*/ | 19946 NEWQUAD_GE
2200 |          REG3=ALLOC_REG_RES(RESULT);                          /*NEWQUAD*/ | 19956 NEWQUAD_GE
2201 |          OPRND1=REG2;                                         /*NEWQUAD*/ | 19986 NEWQUAD_GE
2202 |          OPRND2=0;                                            /*NEWQUAD*/ | 19994 NEWQUAD_GE
2203 |          RSLT=REG3;                                           /*NEWQUAD*/ | 20000 NEWQUAD_GE
2204 |          OPRTOR=29;                                           /*NEWQUAD*/ | 20008 NEWQUAD_GE
2205 |          CALL PUT_NEWQUAD;                                    /*NEWQUAD*/ | 20016 NEWQUAD_GE
2206 |          OPRTOR=1;                                            /*NEWQUAD*/ | 20024 NEWQUAD_GE
2207 |          OPRND1=REG1;                                         /*NEWQUAD*/ | 20032 NEWQUAD_GE
2208 |          OPRND2=REG3;                                         /*NEWQUAD*/ | 20040 NEWQUAD_GE
2209 |          RSLT=REG3;                                           /*NEWQUAD*/ | 20048 NEWQUAD_GE
2210 |          CALL PUT_NEWQUAD;                                    /*NEWQUAD*/ | 20056 NEWQUAD_GE
2211 |          IF SUBSFLAG=SUBS THEN DO;                            /*NEWQUAD*/ | 20064 NEWQUAD_GE
2212 |             OPRTOR=28;                                        /*NEWQUAD*/ | 20080 NEWQUAD_GE
2213 |             OPRND1=REG3;                                      /*NEWQUAD*/ | 20088 NEWQUAD_GE
2214 |             OPRND2=0;                                         /*NEWQUAD*/ | 20096 NEWQUAD_GE
2215 |             RSLT=REG3;                                        /*NEWQUAD*/ | 20102 NEWQUAD_GE
2216 |             CALL PUT_NEWQUAD;                                 /*NEWQUAD*/ | 20110 NEWQUAD_GE
2217 |             CURR_MIC=CURR_MIC+7;                              /*NEWQUAD*/ | 20119 NEWQUAD_GE
2218 |             END;                                              /*NEWQUAD*/ | 20130 NEWQUAD_GE
2219 |          ELSE DO;                                             /*NEWQUAD*/ | 20130 NEWQUAD_GE
2220 |             CURR_MIC=CURR_MIC+5;                              /*NEWQUAD*/ | 20138 NEWQUAD_GE
2221 |             ADDRSS(REG3)=TRUE;                                /*NEWQUAD*/ | 20150 NEWQUAD_GE
2222 |             DEALLOCABL(REG3)=TRUE;                            /*NEWQUAD*/ | 20162 NEWQUAD_GE
2223 |             END;                                              /*NEWQUAD*/ | 20174 NEWQUAD_GE
2224 |          CALL DEALLOC_TEMP(OPERAND2,REG2);                    /*NEWQUAD*/ | 20174 NEWQUAD_GE
2225 |          DEALLOCABL(REG1)=FALSE;                              /*NEWQUAD*/ | 20198 NEWQUAD_GE
2226 |          END;                                                 /*NEWQUAD*/ | 20208 NEWQUAD_GE
2227 |       DO;                                                     /*NEWQUAD*/ | 20208 NEWQUAD_GE
2228 |                                                                           | 20216 NEWQUAD_GE
2229 |    /*PROCESS BR QUADS            */                                       | 20216 NEWQUAD_GE
2230 |                                                                           | 20216 NEWQUAD_GE
2231 |          CALL WRITE_REGS;                                     /*NEWQUAD*/ | 20216 NEWQUAD_GE
2232 |          OPRND1=0;                                            /*NEWQUAD*/ | 20224 NEWQUAD_GE
2233 |          OPRND2=0;                                            /*NEWQUAD*/ | 20230 NEWQUAD_GE
2234 |          TEMP_CHAR=RESULT;                                    /*NEWQUAD*/ | 20236 NEWQUAD_GE
2235 |          RSLT=LAB_INDEX;                                      /*NEWQUAD*/ | 20244 NEWQUAD_GE
2236 |             OPRTOR=OPER;                                      /*NEWQUAD*/ | 20256 NEWQUAD_GE
```

```
2237 |        CALL PUT_NEWQUAD;                                          /*NEWQUAD*/ | 20272 NEWQUAD_GEN
2238 |        END;                                                        /*NEWQUAD*/ | 20272 NEWQUAD_GEN
2239 |     END;                                                           /*NEWQUAD*/ | 20272 NEWQUAD_GEN
2240 |  END;                                                              /*NEWQUAD*/ | 20280 NEWQUAD_GEN
2241 | PROCESS_QUADS:PROCEDURE;                                           /*NEWQUAD*/ | 20286
2242 |                                                                                | 20286 PROCESS_QUAD
2243 | /*PROCESS_QUADS     PROCESS GENERATED QUADS AND GENERATE REGISTER QUADS */     | 20286 PROCESS_QUAD
2244 |                                                                                | 20286 PROCESS_QUAD
2245 |  DECLARE MM FIXED;                                                 /*NEWQUAD*/ | 20286 PROCESS_QUAD
2246 |  OUTPUT(1) = '1            QUADS GENERATED';                       /*NEWQUAD*/ | 20298 PROCESS_QUAD
2247 |  DOUBLE_SPACE;                                                     /*NEWQUAD*/ | 20322 PROCESS_QUAD
2248 |  OUTPUT='OPERATOR  OPERAND1/  OPERAND2 RESULT/' ||                 /*NEWQUAD*/ | 20346 PROCESS_QUAD
2249 |         '  ****  RCD_NR  LOC_QUAD';                               /*NEWQUAD*/ | 20346 PROCESS_QUAD
2250 |  OUTPUT='          CONDITION              LABEL';                  /*NEWQUAD*/ | 20394 PROCESS_QUAD
2251 |  DO  MM = 1 TO NEXTQUAD-1;                                         /*NEWQUAD*/ | 20404 PROCESS_QUAD
2252 |     CALL READQUAD(MM);                                             /*NEWQUAD*/ | 20452 PROCESS_QUAD
2253 |     CALL NEWQUAD_GEN;                                              /*NEWQUAD*/ | 20468 PROCESS_QUAD
2254 |     END;                                                           /*NEWQUAD*/ | 20476 PROCESS_QUAD
2255 |  CALL WRITE_REGS;                                                  /*NEWQUAD*/ | 20484 PROCESS_QUAD
2256 |  END;                                                              /*NEWQUAD*/ | 20492 PROCESS_QUAD
2257 |     /*      DEBUG PRINT OF SYMBOL TABLE      */                                | 20498
2258 |                                                                                | 20498
2259 | PRINTSYMB:                                                                     | 20498
2260 |  PROCEDURE;                                                                    | 20498
2261 |      OUTPUT(1)='1SYMBOL TABLE';                                                | 20498 PRINTSYMB
2262 |     DOUBLE_SPACE;                                                              | 20534 PRINTSYMB
2263 |     OUTPUT='INDEX  SYMBOL  LOCATION  DEFINED  SIZE  INITIAL';                  | 20558 PRINTSYMB
2264 |       DO I=1 TO NSYMBOL;                                                       | 20578 PRINTSYMB
2265 |     BUFFER=PAD('    '||I,9);                                                  | 20622 PRINTSYMB
2266 |     BUFFER=PAD(BUFFER||SYMB(I),19);                                           | 20670 PRINTSYMB
2267 |     BUFFER=PAD(BUFFER||LOCAT(I),29);                                          | 20718 PRINTSYMB
2268 |     BUFFER=PAD(BUFFER||DEF(I),36);                                            | 20774 PRINTSYMB
2269 |     BUFFER=PAD(BUFFER||SIZE(I),43);                                           | 20830 PRINTSYMB
2270 |     BUFFER=PAD(BUFFER||INIT(I),80);                                           | 20886 PRINTSYMB
2271 |     OUTPUT=BUFFER;                                                            | 20942 PRINTSYMB
2272 |     END;                                                                       | 20962 PRINTSYMB
2273 |     OUTPUT(1)='1CONSTANT TABLE';                                               | 20970 PRINTSYMB
2274 |     DOUBLE_SPACE;                                                              | 20994 PRINTSYMB
2275 |     OUTPUT='INDEX  VALUE';                                                     | 21018 PRINTSYMB
2276 |        DO I=1 TO NCONSTANT;                                                    | 21038 PRINTSYMB
2277 |     BUFFER=PAD('    '||I,9);                                                  | 21082 PRINTSYMB
2278 |     BUFFER=PAD(BUFFER||CONVAL(I),80);                                         | 21130 PRINTSYMB
2279 |     OUTPUT=BUFFER;                                                            | 21186 PRINTSYMB
2280 |        END;                                                                    | 21206 PRINTSYMB
2281 |   OUTPUT(1)='1LABEL TABLE';                                                    | 21214 PRINTSYMB
2282 |     DOUBLE_SPACE;                                                              | 21238 PRINTSYMB
2283 |     OUTPUT='INDEX  LABEL  ADDRESS  DEFINED';                                   | 21262 PRINTSYMB
2284 |     DO I=1 TO NLABEL;                                                          | 21282 PRINTSYMB
2285 |     BUFFER=PAD('    '||I,9);                                                  | 21326 PRINTSYMB
2286 |     BUFFER=PAD(BUFFER||LABID(I),16);                                          | 21374 PRINTSYMB
2287 |     BUFFER=PAD(BUFFER||LASTREF(I),27);                                        | 21422 PRINTSYMB
2288 |     BUFFER=PAD(BUFFER||LABDEF(I),80);                                         | 21478 PRINTSYMB
2289 |     OUTPUT=BUFFER;                                                            | 21534 PRINTSYMB
2290 |     END;                                                                       | 21554 PRINTSYMB
2291 |  END PRINTSYMB;                                                                | 21562 PRINTSYMB
2292 |                                                                                | 21568
2293 |     /*      STORE AN ENTIRE QUAD IN THE QUAD TABLE      */                     | 21568
2294 |                                                                                | 21568
2295 | STOREQUAD :                                                                    | 21568
2296 |  PROCEDURE (QUADNO,OPER,OPND1,OPND2,RES);                                      | 21568
2297 |     DECLARE (QUADNO,OPER,OPND1,OPND2,RES) FIXED;                               | 21568 STOREQUAD
2298 |     RCD_NR=(QUADNO-1)/MAXQUADS;                                                | 21580 STOREQUAD
2299 |     IF RCD_NR <= RCD_BUFF THEN                                                 | 21602 STOREQUAD
2300 |          DO;                                                                   | 21626 STOREQUAD
2301 |            FILE(1,RCD_BUFF)=QUADS;                                             | 21618 STOREQUAD
2302 |            RCD_BUFF=RCD_BUFF+1;                                                | 21640 STOREQUAD
```

```
2303 |          END;                                                          | 21652 STOREQUA
2304 |     LOC_QUAD=((QUADNO-1) MOD MAXQUADS)*4+1;                            | 21652 STOREQUA
2305 |     QUADS (LOC_QUAD  )=OPER;                                           | 21684 STOREQUA
2306 |     QUADS (LOC_QUAD+1)=OPND1;                                          | 21700 STOREQUA
2307 |     QUADS (LOC_QUAD+2)=OPND2;                                          | 21720 STOREQUA
2308 |     QUADS (LOC_QUAD+3)=RES;                                            | 21740 STOREQUA
2309 |     RETURN;                                                            | 21760 STOREQUA
2310 |  END STOREQUAD;                                                        | 21766 STOREQUA
2311 |                                                                        | 21772
2312 |     /*      GET QUAD - RETRIEV A FIELD FROM A QUAD    */               | 21772
2313 |                                                                        | 21772
2314 | GETQUAD:                                                               | 21772
2315 |   PROCEDURE (QUADNO,INDEX) FIXED;                                      | 21772
2316 |     DECLARE (QUADNO,INDEX) FIXED;                                      | 21772 GETQUAD
2317 |     RCD_NR=(QUADNO-1)/MAXQUADS;                                        | 21784 GETQUAD
2318 |     IF RCD_NR ¬= RCD_BUFF THEN                                         | 21806 GETQUAD
2319 |          DO;                                                           | 21830 GETQUAD
2320 |            FILE(1,RCD_BUFF)=QUADS;                                     | 21822 GETQUAD
2321 |            RCD_BUFF=RCD_NR;                                            | 21844 GETQUAD
2322 |          QUADS=FILE(1,RCD_BUFF);                                       | 21852 GETQUAD
2323 |          END;                                                          | 21882 GETQUAD
2324 |     LOC_QUAD=((QUADNO-1) MOD MAXQUADS)*4+1;                            | 21832 GETQUAD
2325 |     RETURN QUADS(LOC_QUAD+INDEX-1);                                    | 21914 GETQUAD
2326 | END GETQUAD;                                                           | 21942 GETQUAD
2327 |                                                                        | 21948
2328 |     /*      PUT A FIELD IN A QUAD     */                               | 21948
2329 |                                                                        | 21948
2330 | PUTQUAD:                                                               | 21948
2331 |  PROCEDURE (QUADNO,INDEX,VALUE);                                       | 21948
2332 |     DECLARE (QUADNO,INDEX,VALUE) FIXED;                                | 21948 PUTQUAD
2333 |     RCD_NR=(QUADNO-1)/MAXQUADS;                                        | 21960 PUTQUAD
2334 |     IF RCD_NR ¬= RCD_BUFF THEN                                         | 21982 PUTQUAD
2335 |          DO;                                                           | 22006 PUTQUAD
2336 |            FILE(1,RCD_BUFF)=QUADS;                                     | 21998 PUTQUAD
2337 |            RCD_BUFF=RCD_NR;                                            | 22020 PUTQUAD
2338 |          QUADS=FILE(1,RCD_BUFF);                                       | 22028 PUTQUAD
2339 |          END;                                                          | 22058 PUTQUAD
2340 |     LOC_QUAD=((QUADNO-1) MOD MAXQUADS)*4+1;                            | 22058 PUTQUAD
2341 |     QUADS(LOC_QUAD+INDEX-1)=VALUE;                                     | 22090 PUTQUAD
2342 |     RETURN;                                                            | 22114 PUTQUAD
2343 |  END PUTQUAD;                                                          | 22120 PUTQUAD
2344 |                                                                        | 22126
2345 |     /* PUT A TEMPORARY VARIABLE IN THE SYMBOL TABLE. THESE            | 22126
2346 |           ARE GENERATED BY THE PROGRAM AND ARE ALL UNIQUE. THE         | 22126
2347 |           ONLY ERROR IS OVERFLOW                 */                    | 22126
2348 |                                                                        | 22126
2349 | PUTTEMP:                                                               | 22126
2350 |     PROCEDURE (STACKLOC);                                              | 22126
2351 |          DECLARE STACKLOC FIXED; /* LOCATION OF TEMP IN STACK */       | 22126 PUTTEMP
2352 |          IF NSYMBOL = SYMBOLS THEN                                     | 22138 PUTTEMP
2353 |            DO;                                                         | 22162 PUTTEMP
2354 |               OUTPUT='*** SYMBOL TABLE OVERFLOW, MAX IS '||SYMBOLS;    | 22154 PUTTEMP
2355 |               CALL EXIT;                                               | 22200 PUTTEMP
2356 |            END;                                                        | 22214 PUTTEMP
2357 |          NSYMBOL=NSYMBOL+1;                                            | 22214 PUTTEMP
2358 |          TABLE_LOC(STACKLOC)=NSYMBOL;                                  | 22226 PUTTEMP
2359 |          SIZE (NSYMBOL)=0;                                             | 22242 PUTTEMP
2360 |          DEF  (NSYMBOL)=1;                                             | 22256 PUTTEMP
2361 |          LOCAT(NSYMBOL)=0;                                             | 22272 PUTTEMP
2362 |          INIT (NSYMBOL)=0;                                             | 22286 PUTTEMP
2363 |          SYMB (NSYMBOL)=VAR(STACKLOC);                                 | 22300 PUTTEMP
2364 |     RETURN;                                                            | 22324 PUTTEMP
2365 | END PUTTEMP;                                                           | 22330 PUTTEMP
2366 |                                                                        | 22336
2367 |                                                                        | 22336
2368 |     /*  FIND LABEL IN LABEL TABLE. ERROR IF NOT FOUND */              | 22336
```

```
2370 |     FINDLAB:                                                                  | 22336
2371 |                                                                               | 22336
2372 |       PROCEDURE;                                                             | 22336
2373 |     DECLARE LAB CHARACTER;                                                    | 22336 FINDLAB
2374 |         LAB = VAR(SP);                                                        | 22348 FINDLAB
2375 |       I = 1;                                                                  | 22364 FINDLAB
2376 |         DO WHILE I <= NLABEL;                                                 | 22372 FINDLAB
2377 |           IF LABID(I) = LAB THEN DO;                                          | 22388 FINDLAB
2378 |             TABLE_LOC(SP) = I;                                                | 22438 FINDLAB
2379 |           RETURN;                                                             | 22454 FINDLAB
2380 |         END;                                                                  | 22460 FINDLAB
2381 |         I = I+1;                                                              | 22460 FINDLAB
2382 |       END;                                                                    | 22472 FINDLAB
2383 |         IF NLABEL < LABELS THEN DO;                                           | 22480 FINDLAB
2384 |         NLABEL = NLABEL + 1;                                                  | 22496 FINDLAB
2385 |         LABID(NLABEL) = LAB;                                                  | 22508 FINDLAB
2386 |         LABDEF(NLABEL)=0;                                                     | 22524 FINDLAB
2387 |       TABLE_LOC(SP) = NLABEL;                                                 | 22538 FINDLAB
2388 |       RETURN;                                                                 | 22554 FINDLAB
2389 |       END;                                                                    | 22560 FINDLAB
2390 |     ELSE DO;                                                                  | 22560 FINDLAB
2391 |       OUTPUT='*** LABEL TABLE OVERFLOW, MAX IS '||LABELS;                     | 22568 FINDLAB
2392 |         CALL EXIT;                                                            | 22614 FINDLAB
2393 |       END;                                                                    | 22628 FINDLAB
2394 |     END FINDLAB;                                                              | 22628 FINDLAB
2395 |                                                                               | 22634
2396 |    /* INSERTS A BRANCH LABEL INTO THE LABEL TABLE  */                         | 22634
2397 |                                                                               | 22634
2398 |   BRANCHLAB:                                                                  | 22634
2399 |     PROCEDURE(LAB);                                                           | 22634
2400 |     DECLARE LAB CHARACTER;                                                    | 22634 BRANCHLA
2401 |       NLABEL=NLABEL+1;                                                        | 22646 BRANCHLA
2402 |         IF NLABEL=LABELS THEN DO;                                             | 22658 BRANCHLA
2403 |         OUTPUT='*** LABEL TABLE OVERFLOW';                                    | 22674 BRANCHLA
2404 |         CALL EXIT;                                                            | 22694 BRANCHLA
2405 |       END;                                                                    | 22708 BRANCHLA
2406 |       LABID(NLABEL)=LAB;                                                      | 22708 BRANCHLA
2407 |       LABDEF(NLABEL)=0;                                                       | 22724 BRANCHLA
2408 |       VAR(SP-1)=LAB;                                                          | 22738 BRANCHLA
2409 |       TABLE_LOC(SP-1)=NLABEL;                                                 | 22758 BRANCHLA
2410 |       SAVEINDEX=NLABEL;                                                       | 22778 BRANCHLA
2411 |     RETURN;                                                                   | 22786 BRANCHLA
2412 |   END BRANCHLAB;                                                              | 22792 BRANCHLA
2413 |                                                                               | 22798
2414 |                                                                               | 22798
2415 | QUADGEN:                                                                      | 22798
2416 |     PROCEDURE (TYPE);                                                         | 22798
2417 |                                                                               | 22798 QUADGEN
2418 | /*    DUMMY QUADGEN  */                                                       | 22798 QUADGEN
2419 |                                                                               | 22798 QUADGEN
2420 |           DECLARE SLOC FIXED;                                                 | 22798 QUADGEN
2421 |           DECLARE TYPE FIXED; /* QUAD TYPE */                                | 22810 QUADGEN
2422 |                                                                               | 22810 QUADGEN
2423 |           DECLARE (TNAME,OPND1,OPND2) CHARACTER;                              | 22810 QUADGEN
2424 |   /*    OPERANDS FOR STOREQUAD      */                                      | 22810 QUADGEN
2425 |     DECLARE (ITYP,IOP1,IOP2,IRES) FIXED;                                     | 22810 QUADGEN
2426 |     ITYP=TYPE;                                                               | 22810 QUADGEN
2427 |     IOP1=0;                                                                  | 22818 QUADGEN
2428 |     IOP2=0;                                                                  | 22824 QUADGEN
2429 |     IRES=0;                                                                  | 22830 QUADGEN
2430 |                                                                               | 22836 QUADGEN
2431 |           IF TYPE <= MMOD THEN DO;                                           | 22836 QUADGEN
2432 |     I=TABLE_LOC(SP-2);                                                       | 22852 QUADGEN
2433 |     IOP1=I;                                                                  | 22372 QUADGEN
2434 |      IF  I < 0 THEN                                                          | 22880 QUADGEN
```

```
2435 |            OPND1=CONVAL(-I);                                  | 22904 QUADGEN
2436 |         ELSE                                                  | 22922 QUADGEN
2437 |            OPND1=SYMB(I);                                     | 22922 QUADGEN
2438 |       I=TABLE_LOC(SP);                                        | 22946 QUADGEN
2439 |       IOP2=I;                                                 | 22952 QUADGEN
2440 |        IF  I < 0 THEN                                         | 22970 QUADGEN
2441 |            OPND2=CONVAL(-I);                                  | 22994 QUADGEN
2442 |         ELSE                                                  | 23012 QUADGEN
2443 |            OPND2=SYMB(I);                                     | 23012 QUADGEN
2444 |            TNAME='.T'||NEXTQUAD;                              | 23036 QUADGEN
2445 |          VAR(SP-2)=TNAME;                                     | 23068 QUADGEN
2446 |          CALL PUTTEMP(SP-2);                                  | 23088 QUADGEN
2447 |       IRES=TABLE_LOC(SP-2);                                   | 23108 QUADGEN
2448 |            END;                                               | 23128 QUADGEN
2449 |       ELSE IF TYPE = HALT THEN                                | 23128 QUADGEN
2450 |       DO;                                                     | 23160 QUADGEN
2451 |       END;                                                    | 23152 QUADGEN
2452 |       ELSE IF TYPE <= BE THEN                                 | 23152 QUADGEN
2453 |         DO;                                                   | 23164 QUADGEN
2454 |        IRES=TABLE_LOC(SP);                                    | 23176 QUADGEN
2455 |        IOP1=0;                                                | 23192 QUADGEN
2456 |        IOP2=0;                                                | 23198 QUADGEN
2457 |       OPND1=0;                                                | 23204 QUADGEN
2458 |       OPND2=0;                                                | 23218 QUADGEN
2459 |         END;                                                  | 23232 QUADGEN
2460 |       ELSE IF TYPE = REL THEN DO;                             | 23232 QUADGEN
2461 |            TNAME='.T'||NEXTQUAD;                              | 23256 QUADGEN
2462 |       I=TABLE_LOC(SP-2);                                      | 23288 QUADGEN
2463 |       IOP1=I;                                                 | 23308 QUADGEN
2464 |        IF  I < 0 THEN                                         | 23316 QUADGEN
2465 |            OPND1=CONVAL(-I);                                  | 23340 QUADGEN
2466 |         ELSE                                                  | 23358 QUADGEN
2467 |            OPND1=SYMB(I);                                     | 23358 QUADGEN
2468 |       I=TABLE_LOC(SP);                                        | 23382 QUADGEN
2469 |       IOP2=I;                                                 | 23398 QUADGEN
2470 |        IF  I < 0 THEN                                         | 23406 QUADGEN
2471 |            OPND2=CONVAL(-I);                                  | 23430 QUADGEN
2472 |         ELSE                                                  | 23448 QUADGEN
2473 |            OPND2=SYMB(I);                                     | 23448 QUADGEN
2474 |          VAR(SP-2)=TNAME;                                     | 23472 QUADGEN
2475 |       CALL PUTTEMP(SP-2);                                     | 23492 QUADGEN
2476 |       IRES=TABLE_LOC(SP-2);                                   | 23512 QUADGEN
2477 |       ITYP=FIXV(SP-1);                                        | 23532 QUADGEN
2478 |            END;                                               | 23552 QUADGEN
2479 |       ELSE IF TYPE = ASGN THEN                                | 23552 QUADGEN
2480 |            DO;                                                | 23584 QUADGEN
2481 |       I=TABLE_LOC(SP);                                        | 23576 QUADGEN
2482 |       IOP1=I;                                                 | 23592 QUADGEN
2483 |        IF  I < 0 THEN                                         | 23600 QUADGEN
2484 |            OPND1=CONVAL(-I);                                  | 23624 QUADGEN
2485 |         ELSE                                                  | 23642 QUADGEN
2486 |            OPND1=SYMB(I);                                     | 23642 QUADGEN
2487 |       IRES=TABLE_LOC(SP-2);                                   | 23666 QUADGEN
2488 |         END;                                                  | 23686 QUADGEN
2489 |       ELSE IF TYPE = SUBS THEN                                | 23686 QUADGEN
2490 |            DO;                                                | 23718 QUADGEN
2491 |              TNAME='.T'||NEXTQUAD;                            | 23710 QUADGEN
2492 |              I=TABLE_LOC(SP-1);                               | 23742 QUADGEN
2493 |       IOP2=I;                                                 | 23762 QUADGEN
2494 |              IF I<0 THEN                                      | 23770 QUADGEN
2495 |                 OPND2=CONVAL(-I);                             | 23794 QUADGEN
2496 |            ELSE                                               | 23812 QUADGEN
2497 |                 OPND2=SYMB(I);                                | 23812 QUADGEN
2498 |            SLOC=TABLE_LOC(SP-2);                              | 23836 QUADGEN
2499 |            IOP1=SLOC;                                         | 23856 QUADGEN
2500 |            OPND1=SYMB(SLOC);                                  | 23864 QUADGEN
```

```
2501 |          VAR(SP-2)=TNAME;                                                  | 23940 QUADGEN
2502 |          CALL PUTTEMP(SP-2);                                               | 23900 QUADGEN
2503 |     IRES=TABLE_LOC(SP-2);                                                  | 23920 QUADGEN
2504 |          LOCAT(SYMBOL)=-SLOC;                                              | 23940 QUADGEN
2505 |          END;                                                              | 23958 QUADGEN
2506 |   ELSE IF TYPE = BZ THEN DO;                                               | 23958 QUADGEN
2507 |     IRES = TABLE_LOC(SP-1);                                                | 23982 QUADGEN
2508 |     IOP1=0;                                                                | 24002 QUADGEN
2509 |     IOP2=0;                                                                | 24008 QUADGEN
2510 |     OPND1=0;                                                               | 24014 QUADGEN
2511 |     OPND2=0;                                                               | 24028 QUADGEN
2512 |  END;                                                                      | 24042 QUADGEN
2513 |   ELSE IF TYPE <= OR THEN DO;                                              | 24042 QUADGEN
2514 |     TNAME = '.T'||NEXTQUAD;                                                | 24066 QUADGEN
2515 |    I = TABLE_LOC(SP-2);                                                    | 24098 QUADGEN
2516 |   IOP1 = I;                                                                | 24118 QUADGEN
2517 |   IF I < 0 THEN                                                            | 24126 QUADGEN
2518 |     OPND1 = CONVAL(-I);                                                    | 24150 QUADGEN
2519 |   ELSE                                                                     | 24168 QUADGEN
2520 |     OPND1 = SYMB(I);                                                       | 24168 QUADGEN
2521 |   I = TABLE_LOC(SP);                                                       | 24192 QUADGEN
2522 |   IOP2 = I;                                                                | 24208 QUADGEN
2523 |   IF I < 0 THEN                                                            | 24216 QUADGEN
2524 |     OPND2 = CONVAL(-I);                                                    | 24240 QUADGEN
2525 |   ELSE                                                                     | 24258 QUADGEN
2526 |     OPND2 = SYMB(I);                                                       | 24258 QUADGEN
2527 |   VAR(SP-2)=TNAME;                                                         | 24282 QUADGEN
2528 |   CALL PUTTEMP(SP-2);                                                      | 24302 QUADGEN
2529 |  IRES=TABLE_LOC(SP-2);                                                     | 24322 QUADGEN
2530 |  END;                                                                      | 24342 QUADGEN
2531 |   ELSE IF TYPE = UMIN THEN                                                 | 24342 QUADGEN
2532 |     DO;                                                                    | 24374 QUADGEN
2533 |    I=TABLE_LOC(SP);                                                        | 24366 QUADGEN
2534 |     IOP1=I;                                                                | 24392 QUADGEN
2535 |     IOP2=0;                                                                | 24390 QUADGEN
2536 |    IF I < 0 THEN                                                           | 24396 QUADGEN
2537 |       OPND1=CONVAL(-I);                                                    | 24420 QUADGEN
2538 |     ELSE                                                                   | 24438 QUADGEN
2539 |       OPND1=SYMB(I);                                                       | 24436 QUADGEN
2540 |     OPND2='----';                                                          | 24462 QUADGEN
2541 |     TNAME = '.T'||NEXTQUAD;                                                | 24470 QUADGEN
2542 |     VAR(SP)=TNAME;                                                         | 24502 QUADGEN
2543 |     CALL PUTTEMP(SP);                                                      | 24518 QUADGEN
2544 |     IRES=TABLE_LOC(SP);                                                    | 24534 QUADGEN
2545 |  END;                                                                      | 24550 QUADGEN
2546 |   ELSE IF TYPE = ZQ THEN DO;                                               | 24550 QUADGEN
2547 |     I=TABLE_LOC(SP-1);                                                     | 24574 QUADGEN
2548 |       IOP1=I;                                                              | 24594 QUADGEN
2549 |       OPND1=I;                                                             | 24602 QUADGEN
2550 |     IOP2=0;                                                                | 24618 QUADGEN
2551 |     OPND2=0;                                                               | 24624 QUADGEN
2552 |     TNAME=' ';                                                             | 24638 QUADGEN
2553 |     IRES=0;                                                                | 24646 QUADGEN
2554 |   END;                                                                     | 24652 QUADGEN
2555 |    ELSE IF TYPE = LAB THEN                                                 | 24652 QUADGEN
2556 |          DO;                                                               | 24684 QUADGEN
2557 |            IOP1=TABLE_LOC(SP-1);                                           | 24676 QUADGEN
2558 |         OPND1=VAR(SP-1);                                                   | 24696 QUADGEN
2559 |     END;                                                                   | 24716 QUADGEN
2560 |      CALL STOREQUAD (NEXTQUAD,ITYP,IOP1,IOP2,IRES);                        | 24716 QUADGEN
2561 |    NEXTQUAD=NEXTQUAD+1;                                                    | 24764 QUADGEN
2562 | END QUADGEN;                                                               | 24776 QUADGEN
2563 |                                                                            | 24782
2564 |                                                                            | 24782
2565 |                                                                            | 24782
2566 |    /*  INITIALLY DEFINES A NEW LABEL OR DEFINES A LABEL PREVIOUSLY         | 24782
```

```
2567 |          STORED  */                                                    | 24782
2568 |                                                                        | 24782
2569 |     DEFINELAB:                                                        | 24782
2570 |                                                                        | 24782
2571 |       PROCEDURE(STACKLOC);                                            | 24782
2572 |     DECLARE LAB CHARACTER;                                             | 24782 DEFINELA
2573 |    DECLARE TEMPREF FIXED;                                              | 24794 DEFINELA
2574 |       DECLARE STACKLOC FIXED; /*  LOC IN STACK OF LABEL */             | 24794 DEFINELA
2575 |         TABLE_LOC(STACKLOC) = 0;                                       | 24794 DEFINELA
2576 |         LAB = VAR(STACKLOC);                                           | 24808 DEFINELA
2577 |           I = 1;                                                       | 24824 DEFINELA
2578 |             DO WHILE I <= NLABEL;                                      | 24832 DEFINELA
2579 |               IF LABID(I) = LAB THEN DO;                               | 24848 DEFINELA
2580 |                   LABDEF(I) = 1;                                       | 24898 DEFINELA
2581 |                 TABLE_LOC(STACKLOC) = I;                               | 24914 DEFINELA
2582 |             RETURN;                                                    | 24930 DEFINELA
2583 |             END;                                                       | 24936 DEFINELA
2584 |             I = I+1;                                                   | 24936 DEFINELA
2585 |         END;                                                           | 24948 DEFINELA
2586 |               NLABEL = NLABEL + 1;                                     | 24956 DEFINELA
2587 |                 IF NLABEL = LABELS THEN DO;                            | 24968 DEFINELA
2588 |                   OUTPUT = ' *** LABEL TABLE OVERFLOW';                | 24984 DEFINELA
2589 |                   CALL EXIT;                                           | 25004 DEFINELA
2590 |                 END;                                                   | 25018 DEFINELA
2591 |                 LABID(NLABEL)= LAB;                                    | 25018 DEFINELA
2592 |                 LABDEF(NLABEL) = 1;                                    | 25034 DEFINELA
2593 |                 TABLE_LOC(STACKLOC) = NLABEL;                          | 25050 DEFINELA
2594 |             RETURN;                                                    | 25066 DEFINELA
2595 |     END DEFINELAB;                                                     | 25072 DEFINELA
2596 |             VAR(SP-1)=LAB;                                             | 25078
2597 |    /* SET THE SIZE COLUMN OF THE SYMBOL TABLE TO THE ARRAY DIMENSION   | 25114
2598 |                OF DIMENSIONED VARIABLES   */                           | 25114
2599 |                                                                        | 25114
2600 | SETDIM:                                                                | 25114
2601 |    PROCEDURE;                                                          | 25114
2602 |          K=FIXV(SP-2);                                                 | 25114 SETDIM
2603 |          J=NSYMBOL;                                                    | 25146 SETDIM
2604 |          DO I=1 TO NVARDEF;                                            | 25154 SETDIM
2605 |            SIZE(J)=K;                                                  | 25198 SETDIM
2606 |            J=J-1;                                                      | 25214 SETDIM
2607 |          END;                                                          | 25226 SETDIM
2608 |     END SETDIM;                                                        | 25234 SETDIM
2609 |                                                                        | 25240
2610 |     /* STORE A SYMBOL IN THE SYMBOL TABLE. ERROR IF IT IS ALREADY     | 25240
2611 |           DEFINED. SET ALL DEFAULTS IN THE TABLE     */                | 25240
2612 |                                                                        | 25240
2613 | PUTIT:                                                                 | 25240
2614 |     PROCEDURE (STACKLOC);                                              | 25240
2615 |        DECLARE STACKLOC FIXED; /* LOC IN STACK OF VARIABLE IDENT */    | 25240 PUTIT
2616 |          IF NSYMBOL=SYMBOLS THEN                                       | 25252 PUTIT
2617 |            DO;                                                         | 25276 PUTIT
2618 |               OUTPUT='*** SYMBOL TABLE OVERFLOW, MAX IS '||SYMBOLS;    | 25268 PUTIT
2619 |               CALL EXIT;                                               | 25314 PUTIT
2620 |            END;                                                        | 25328 PUTIT
2621 |          TABLE_LOC(STACKLOC)=0;                                        | 25328 PUTIT
2622 |          SYMB(NSYMBOL+1)=VAR(STACKLOC);                                | 25342 PUTIT
2623 |          I=1;                                                          | 25370 PUTIT
2624 |          S=VAR(STACKLOC);                                              | 25378 PUTIT
2625 |            DO WHILE SYMB(I) ¬= S;                                      | 25394 PUTIT
2626 |               I=I+1;                                                   | 25444 PUTIT
2627 |            END;                                                        | 25456 PUTIT
2628 |          IF I <= NSYMBOL THEN                                          | 25464 PUTIT
2629 |            DO;                                                         | 25488 PUTIT
2630 |               OUTPUT='DUPLICATE IDENTIFIER (VARIABLE) NAME '||S;       | 25480 PUTIT
2631 |               RETURN;                                                  | 25518 PUTIT
2632 |            END;                                                        | 25524 PUTIT
```

```
2633 |          NSYMBOL=1;                                                    | 25524 PUTIT
2634 |          TABLE_LOC(STACKLOC)=NSYMBOL;                                  | 25532 PUTIT
2635 |          SIZE(NSYMBOL)=1;                                              | 25548 PUTIT
2636 |          LOCAT(NSYMBOL)=0;                                             | 25564 PUTIT
2637 |          DEF(NSYMBOL)=0;                                               | 25578 PUTIT
2638 |          INIT(NSYMBOL)=0;                                              | 25592 PUTIT
2639 |     RETURN;                                                            | 25606 PUTIT
2640 | END PUTIT;                                                             | 25612 PUTIT
2641 |                                                                        | 25618
2642 |     /* FIND AN IDENTIFIER IN THE SYMBOL TABLE. ERROR IF NOT FOUND */   | 25618
2643 |                                                                        | 25618
2644 | FINDIT:                                                                | 25618
2645 |     PROCEDURE (PS);                                                    | 25618
2646 |          DECLARE PS FIXED; /* LOCATION IN STACK OF SYMBOL */           | 25618 FINDIT
2647 |          S=VAR(PS);                                                    | 25630 FINDIT
2648 |          I=1;                                                          | 25646 FINDIT
2649 |          DO WHILE I <= NSYMBOL;                                        | 25654 FINDIT
2650 |            IF SYMB(I) = S THEN                                         | 25670 FINDIT
2651 |               DO;                                                      | 25728 FINDIT
2652 |                  TABLE_LOC(PS)=I;                                      | 25720 FINDIT
2653 |                  RETURN;                                               | 25736 FINDIT
2654 |               END;                                                     | 25742 FINDIT
2655 |             I=I+1;                                                     | 25742 FINDIT
2656 |            END;                                                        | 25754 FINDIT
2657 |          OUTPUT='*** VARIABLE IDENTIFIER NOT FOUND - '||S;             | 25762 FINDIT
2658 |          TABLE_LOC(PS)=0;                                              | 25800 FINDIT
2659 |     RETURN;                                                            | 25814 FINDIT
2660 | END FINDIT;                                                            | 25820 FINDIT
2661 |                                                                        | 25826
2662 |     /* FIND A NUMBER IN THE CONSTANT TABLE. IF NOT FOUND, ADD IT */    | 25826
2663 |                                                                        | 25826
2664 | FINDNO:                                                                | 25826
2665 |     PROCEDURE;                                                         | 25826
2666 |          L=FIXV(SP);                                                   | 25826 FINDNO
2667 |          CONVAL(NCONSTANT+1)=L;                                        | 25854 FINDNO
2668 |          CONLOC(NCONSTANT)=0;                                          | 25874 FINDNO
2669 |          I=1;                                                          | 25888 FINDNO
2670 |          DO WHILE CONVAL(I) ^= L;                                      | 25896 FINDNO
2671 |               I=I+1;                                                   | 25920 FINDNO
2672 |          END;                                                          | 25932 FINDNO
2673 |          TABLE_LOC(SP)=-I;                                             | 25940 FINDNO
2674 |        IF I <= NCONSTANT THEN RETURN;                                  | 25958 FINDNO
2675 |          IF NCONSTANT+1 = CONSTANTS THEN                               | 25980 FINDNO
2676 |            DO;                                                         | 26008 FINDNO
2677 |               L=CONSTANTS-1;                                           | 26000 FINDNO
2678 |               OUTPUT='*** CONSTANT TABLE OVERFLOW, MAX IS '||L;        | 26012 FINDNO
2679 |               CALL EXIT;                                               | 26058 FINDNO
2680 |          END;                                                          | 26072 FINDNO
2681 |          NCONSTANT=I;                                                  | 26072 FINDNO
2682 |     RETURN;                                                            | 26080 FINDNO
2683 | END FINDNO;                                                            | 26086 FINDNO
2684 |                                                                        | 26092
2685 |                                                                        | 26092
2686 |     /* MOVE ALL STACK CONTENTS FROM F TO T */                          | 26092
2687 |                                                                        | 26092
2688 | MOVESTACK:                                                             | 26092
2689 |     PROCEDURE (F,T);                                                   | 26092
2690 |          DECLARE (F,T) FIXED;                                          | 26092 MOVESTAC
2691 |            TABLE_LOC(T)=TABLE_LOC(F);                                  | 26104 MOVESTAC
2692 |            VAR(T)=VAR(F);                                              | 26128 MOVESTAC
2693 |            FIXV(T)=FIXV(F);                                            | 26152 MOVESTAC
2694 |          PARSE_STACK(T)=PARSE_STACK(F);                                | 26176 MOVESTAC
2695 |     END MOVESTACK;                                                     | 26194 MOVESTAC
2696 |                                                                        | 26200
2697 |                                                                        | 26200
2698 |                                                                        | 26200
```

```
2699 |    /* GENLOOP GENERATES THE QUADS NECESSARY FOR DO STATEMENTS */                    | 26200
2700 |                                                                                     | 26200
2701 |  GENLOOP:                                                                           | 26200
2702 |    PROCEDURE;                                                                       | 26200
2703 |      SP=SP-1;                                                                       | 26200 GENLOOP
2704 |      CALL QUADGEN(ASGN);                                                            | 26224 GENLOOP
2705 |      LOOP_INDEX=TABLE_LOC(SP-2);                                                    | 26240 GENLOOP
2706 |      SP=SP+1;                                                                       | 26260 GENLOOP
2707 |      SAVEVAR=VAR(SP-1);                                                             | 26272 GENLOOP
2708 |      SAVELOC=TABLE_LOC(SP-1);                                                       | 26292 GENLOOP
2709 |      CALL BRANCHLAB('.L'||NEXTQUAD);                                                | 26312 GENLOOP
2710 |      SAVLABNO=SAVLABNO+1;                                                           | 26352 GENLOOP
2711 |      SAVLAB(SAVLABNO)=VAR(SP-1);                                                    | 26364 GENLOOP
2712 |      CALL DEFINELAB(SP-1);                                                          | 26392 GENLOOP
2713 |      CALL QUADGEN(LAB);                                                             | 26412 GENLOOP
2714 |      TABLE_LOC(SP)=LOOPLIM;                                                         | 26428 GENLOOP
2715 |      TABLE_LOC(SP-2)=LOOP_INDEX;                                                    | 26444 GENLOOP
2716 |      FIXV(SP-1)=CT;                                                                 | 26464 GENLOOP
2717 |      CALL QUADGEN(REL);                                                             | 26484 GENLOOP
2718 |      SAVQUDNO=SAVQUDNO+1;                                                           | 26500 GENLOOP
2719 |      SAVQUD(SAVQUDNO)=NEXTQUAD;                                                     | 26512 GENLOOP
2720 |      CALL QUADGEN(ZQ);                                                              | 26528 GENLOOP
     |      CALL PUTQUAD(NEXTQUAD-1,1,BT);                                                 | 26544 GENLOOP
     |      I=TABLE_LOC(SP-2);                                                             | 26580 GENLOOP
2723 |      CALL PUTQUAD(NEXTQUAD-1,2,I);                                                  | 26600 GENLOOP
2724 |    RETURN;                                                                          | 26636 GENLOOP
2725 |  END GENLOOP;                                                                       | 26642 GENLOOP
2726 |                                                                                     | 26648
2727 |                                                                                     | 26648
2728 |                                                                                     | 26648
2729 |  /*                    THE SYNTHESIS ALGORITHM FOR XPL        */                    | 26648
2730 |                                                                                     | 26648
2731 |                                                                                     | 26648
2732 | SYNTHESIZE:                                                                         | 26648
2733 | PROCEDURE(PRODUCTION_NUMBER);                                                       | 26648
2734 |    DECLARE PRODUCTION_NUMBER FIXED;                                                 | 26648 SYNTHESIZE
2735 |                                                                                     | 26660 SYNTHESIZE
2736 |    /*  THIS PROCEDURE IS RESPONSIBLE FOR THE SEMANTICS (CODE SYNTHESIS), IF       | 26660 SYNTHESIZE
2737 |        ANY, OF THE SKELETON COMPILER.  ITS ARGUMENT IS THE NUMBER OF THE            | 26660 SYNTHESIZE
2738 |        PRODUCTION WHICH WILL BE APPLIED IN THE PENDING REDUCTION.  THE GLOBAL      | 26660 SYNTHESIZE
2739 |        VARIABLES MP AND SP POINT TO THE BOUNDS IN THE STACKS OF THE RIGHT PART     | 26660 SYNTHESIZE
2740 |        OF THIS PRODUCTION.                                                          | 26660 SYNTHESIZE
2741 |         NORMALLY, THIS PROCEDURE WILL TAKE THE FORM OF A GIANT CASE STATEMENT      | 26660 SYNTHESIZE
2742 |        ON PRODUCTION-NUMBER.  HOWEVER, THE SYNTAX CHECKER HAS SEMANTICS (THE       | 26660 SYNTHESIZE
2743 |        TERMINATION OF CHECKING) ONLY FOR PRODUCTION 1.                    */        | 26660 SYNTHESIZE
2744 |                                                                                     | 26660 SYNTHESIZE
2745 |                                                                                     | 26660 SYNTHESIZE
2746 |                                                                                     | 26660 SYNTHESIZE
2747 | /*  ONE STATEMENT FOR EACH PRODUCTION OF THE GRAMMER  */                            | 26660 SYNTHESIZE
2748 |                                                                                     | 26660 SYNTHESIZE
2749 |                                                                                     | 26660 SYNTHESIZE
2750 | DO CASE PRODUCTION_NUMBER;                                                          | 26660 SYNTHESIZE
2751 |      ;       /* DUMMY CASE SINCE PRODUCTIONS NUMBERED FROM 1 */                     | 26684 SYNTHESIZE
2752 | /*  <PROGRAM> ::= <STATEMENT LIST>    */                                           | 26684 SYNTHESIZE
2753 | /*  <PROGRAM>  ::=  <STATEMENT LIST>    */                                         | 26684 SYNTHESIZE
2754 |    DO;                                                                              | 26684 SYNTHESIZE
2755 |      IF MP ~= 2 THEN  /* WE DIDN'T GET HERE LEGITIMATELY  */                       | 26692 SYNTHESIZE
2756 |        DO;                                                                          | 26716 SYNTHESIZE
2757 |          CALL ERROR ('EOF AT INVALID POINT', 1);                                    | 26708 SYNTHESIZE
2758 |          CALL STACK_DUMP;                                                           | 26728 SYNTHESIZE
2759 |      END;                                                                           | 26736 SYNTHESIZE
2760 |      COMPILING = FALSE;                                                             | 26736 SYNTHESIZE
2761 |        END;                                                                         | 26742 SYNTHESIZE
2762 | /*  <STATEMENT LIST> ::= <STATEMENT>    */                                         | 26742 SYNTHESIZE
2763 |                                                                                     | 26742 SYNTHESIZE
2764 | /*  <STATEMENT LIST> ::= <STATEMENT LIST> <STATEMENT>    */                         | 26750 SYNTHESIZE
```

```
2765 |     ;                                                                    | 26750 SYNTHESIZ
2766 | /*  <STATEMENT> ::= <BASIC STATEMENT>    */                              | 26758 SYNTHESIZ
2767 |     ;                                                                    | 26758 SYNTHESIZ
2768 | /*  <STATEMENT> ::= <IF STATEMENT>    */                                | 26766 SYNTHESIZ
2769 |     ;                                                                    | 26766 SYNTHESIZ
2770 | /*  <BASIC STATEMENT> ::= <ASSIGNMENT> ;    */                          | 26774 SYNTHESIZ
2771 |   DO;                                                                    | 26774 SYNTHESIZ
2772 |     TBASIC = 1;                                                          | 26782 SYNTHESIZ
2773 |   END;                                                                   | 26790 SYNTHESIZ
2774 | /*  <BASIC STATEMENT> ::= <GROUP> ;    */                               | 26790 SYNTHESIZ
2775 |     ;                                                                    | 26790 SYNTHESIZ
2776 | /*  <BASIC STATEMENT> ::= <PROCEDURE DEFINITION> ;    */                 | 26798 SYNTHESIZ
2777 |     ;                                                                    | 26798 SYNTHESIZ
2778 | /*  <BASIC STATEMENT> ::= <RETURN STATEMENT> ;    */                     | 26806 SYNTHESIZ
2779 |     ;                                                                    | 26806 SYNTHESIZ
2780 | /*  <BASIC STATEMENT> ::= <CALL STATEMENT> ;    */                       | 26814 SYNTHESIZ
2781 |     ;                                                                    | 26814 SYNTHESIZ
2782 | /*  <BASIC STATEMENT> ::= <GO TO STATEMENT> ;    */                      | 26822 SYNTHESIZ
2783 |   DO;                                                                    | 26822 SYNTHESIZ
2784 |     TBASIC = 6;                                                          | 26830 SYNTHESIZ
2785 |   END;                                                                   | 26838 SYNTHESIZ
2786 | /*  <BASIC STATEMENT> ::= <DECLARATION STATEMENT> ;    */                 | 26838 SYNTHESIZ
2787 |     TBASIC=7;                                                            | 26838 SYNTHESIZ
2788 | /*  <BASIC STATEMENT> ::= HALT ;    */                                   | 26854 SYNTHESIZ
2789 |     DO;                                                                  | 26854 SYNTHESIZ
2790 |       CALL QUADGEN(HALT);                                                | 26862 SYNTHESIZ
2791 |       TBASIC=8;                                                          | 26878 SYNTHESIZ
2792 |     END;                                                                 | 26886 SYNTHESIZ
2793 | /*  <BASIC STATEMENT> ::= ENABLE ;    */                                | 26886 SYNTHESIZ
2794 |     ;                                                                    | 26886 SYNTHESIZ
2795 | /*  <BASIC STATEMENT> ::= DISABLE ;    */                               | 26894 SYNTHESIZ
2796 |     ;                                                                    | 26894 SYNTHESIZ
2797 | /*  <BASIC STATEMENT> ::= ;    */                                        | 26902 SYNTHESIZ
2798 |     ;                                                                    | 26902 SYNTHESIZ
2799 | /*  <BASIC STATEMENT> ::= <LABEL DEFINITION> <BASIC STATEMENT>    */     | 26910 SYNTHESIZ
2800 |     ;                                                                    | 26910 SYNTHESIZ
2801 | /*  <IF STATEMENT> ::= <IF CLAUSE> <STATEMENT>    */                     | 26918 SYNTHESIZ
2802 |   DO;                                                                    | 26918 SYNTHESIZ
2803 |     IF TBASIC = 1 THEN DO;                                               | 26926 SYNTHESIZ
2804 |     CALL BRANCHLAB('.L'||SAVQUAD(SAVQUDNO));                             | 26942 SYNTHESIZ
2805 |     CALL PUTQUAD(SAVQUAD(SAVQUDNO),4,SAVEINDEX);                         | 26990 SYNTHESIZ
2806 |     CALL PUTQUAD(SAVQUAD(SAVQUDNO),1,BF);                                | 27030 SYNTHESIZ
2807 |     SAVQUDNO=SAVQUDNO-1;                                                 | 27070 SYNTHESIZ
2808 |     CALL DEFINELAB(SP-1);                                                | 27092 SYNTHESIZ
2809 |     CALL QUADGEN(LAB);                                                   | 27102 SYNTHESIZ
2810 |   END;                                                                   | 27118 SYNTHESIZ
2811 |   IF TBASIC = 6 THEN DO;                                                 | 27118 SYNTHESIZ
2812 |     I=GETQUAD(NEXTQUAD-1,4);                                             | 27134 SYNTHESIZ
2813 |     NEXTQUAD = NEXTQUAD-1;                                               | 27166 SYNTHESIZ
2814 |     CALL PUTQUAD(SAVQUAD(SAVQUDNO),4,I);                                 | 27178 SYNTHESIZ
2815 |     CALL PUTQUAD(SAVQUAD(SAVQUDNO),1,BT);                                | 27218 SYNTHESIZ
2816 |     SAVQUDNO=SAVQUDNO-1;                                                 | 27258 SYNTHESIZ
2817 |   END;                                                                   | 27270 SYNTHESIZ
2818 |END;                                                                      | 27270 SYNTHESIZ
2819 | /*  <IF STATEMENT> ::= <IF CLAUSE> <TRUE PART> <STATEMENT>    */          | 27270 SYNTHESIZ
2820 |     IF SAVEQUAD2 ^= 0 THEN DO;                                           | 27270 SYNTHESIZ
2821 |       CALL BRANCHLAB('.L'||SAVEQUAD2);                                  | 27294 SYNTHESIZ
2822 |       CALL PUTQUAD(SAVEQUAD2,4,SAVEINDEX);                              | 27334 SYNTHESIZ
2823 |       CALL DEFINELAB(SP-1);                                             | 27366 SYNTHESIZ
2824 |       CALL QUADGEN(LAB);                                                | 27386 SYNTHESIZ
2825 |     END;                                                                 | 27402 SYNTHESIZ
2826 | /*  <IF STATEMENT> ::= <LABEL DEFINITION> <IF STATEMENT>    */           | 27402 SYNTHESIZ
2827 |     ;                                                                    | 27402 SYNTHESIZ
2828 | /*  <IF CLAUSE> ::= IF <EXPRESSION> THEN    */                          | 27410 SYNTHESIZ
2829 |     DO;                                                                  | 27410 SYNTHESIZ
2830 |     SAVQUDNO=SAVQUDNO+1;                                                 | 27418 SYNTHESIZ
```

```
2831 |      SAVQUD(SAVQUDNO)=NEXTQUAD;                                                    | 27448 SYNTHESIZ
2832 |        CALL QUADGEN(ZQ);                                                          | 27446 SYNTHESIZ
2833 |END;                                                                               | 27462 SYNTHESIZ
2834 | /*  <TRUE PART> ::= <BASIC STATEMENT> ELSE    */                                   | 27462 SYNTHESIZ
2835 | DO;                                                                               | 27462 SYNTHESIZ
2836 |    IF TBASIC = 1 THEN DO;                                                         | 27470 SYNTHESIZ
2837 |     SAVEQUAD2=NEXTQUAD;                                                           | 27486 SYNTHESIZ
2838 |     CALL QUADGEN(ZQ);                                                             | 27494 SYNTHESIZ
2839 |     CALL PUTQUAD(NEXTQUAD-1,1,BR);                                                | 27510 SYNTHESIZ
2840 |     CALL PUTQUAD(NEXTQUAD-1,2,0);                                                 | 27546 SYNTHESIZ
2841 |     CALL BRANCHLAB('.L'||SAVQUD(SAVQUDNO));                                       | 27580 SYNTHESIZ
2842 |     CALL PUTQUAD(SAVQUD(SAVQUDNO),4,SAVEINDEX);                                   | 27628 SYNTHESIZ
2843 |     CALL PUTQUAD(SAVQUD(SAVQUDNO),1,BF);                                          | 27668 SYNTHESIZ
2844 |     SAVQUDNO=SAVQUDNO-1;                                                          | 27708 SYNTHESIZ
2845 |     CALL DEFINELAB(SP-1);                                                         | 27720 SYNTHESIZ
2846 |     CALL QUADGEN(LAB);                                                            | 27740 SYNTHESIZ
2847 |    END;                                                                           | 27756 SYNTHESIZ
2848 |    IF TBASIC = 6 THEN DO ;                                                        | 27756 SYNTHESIZ
2849 |     SAVEQUAD2 = 0;                                                                | 27772 SYNTHESIZ
2850 |     I=GETQUAD(NEXTQUAD-1,4);                                                      | 27778 SYNTHESIZ
2851 |     CALL PUTQUAD(SAVQUD(SAVQUDNO),4,I);                                           | 27810 SYNTHESIZ
2852 |     CALL PUTQUAD(SAVQUD(SAVQUDNO),1,BT);                                          | 27850 SYNTHESIZ
2853 |     SAVQUDNO=SAVQUDNO-1;                                                          | 27890 SYNTHESIZ
2854 |     NEXTQUAD=NEXTQUAD-1;                                                          | 27902 SYNTHESIZ
2855 |   END;                                                                            | 27914 SYNTHESIZ
2856 |END;                                                                               | 27914 SYNTHESIZ
2857 | /*  <GROUP> ::= <GROUP HEAD> <ENDING>     */                                       | 27914 SYNTHESIZ
2858 |  DO;                                                                              | 27914 SYNTHESIZ
2859 |     SP=SP+2;                                                                      | 27922 SYNTHESIZ
2860 |     TABLE_LOC(SP)=LOOPINC;                                                        | 27934 SYNTHESIZ
2861 |     TABLE_LOC(SP-2)=LOOP_INDEX;                                                   | 27950 SYNTHESIZ
2862 |     CALL QUADGEN(ADD);                                                            | 27970 SYNTHESIZ
2863 |     TABLE_LOC(SP)=TABLE_LOC(SP-2);                                                | 27986 SYNTHESIZ
2864 |     TABLE_LOC(SP-2)=LOOP_INDEX;                                                   | 28014 SYNTHESIZ
2865 |     CALL QUADGEN(ASGN);                                                           | 28034 SYNTHESIZ
2866 |     SP=SP-2;                                                                      | 28050 SYNTHESIZ
2867 |     VAR(SP)=SAVLAB(SAVLABNO);                                                     | 28062 SYNTHESIZ
2868 |     SAVLABNO=SAVLABNO-1;                                                          | 28086 SYNTHESIZ
2869 |     CALL FINDLAB;                                                                 | 28098 SYNTHESIZ
2870 |     CALL QUADGEN(ZQ);                                                             | 28106 SYNTHESIZ
2871 |     CALL PUTQUAD(NEXTQUAD-1,1,BR);                                                | 28122 SYNTHESIZ
2872 |     CALL PUTQUAD(NEXTQUAD-1,2,0);                                                 | 28158 SYNTHESIZ
2873 |     CALL PUTQUAD(NEXTQUAD-1,4,TABLE_LOC(SP));                                     | 28192 SYNTHESIZ
2874 |     CALL BRANCHLAB('.L'||SAVQUD(SAVQUDNO));                                       | 28236 SYNTHESIZ
2875 |     CALL PUTQUAD(SAVQUD(SAVQUDNO),4,TABLE_LOC(SP-1));                             | 28284 SYNTHESIZ
2876 |     SAVQUDNO=SAVQUDNO-1;                                                          | 28336 SYNTHESIZ
2877 |     CALL DEFINELAB(SP-1);                                                         | 28348 SYNTHESIZ
2878 |     CALL QUADGEN(LAB);                                                            | 28368 SYNTHESIZ
2879 |  END;                                                                             | 28384 SYNTHESIZ
2880 | /*  <GROUP HEAD> ::= DO ;     */                                                  | 28384 SYNTHESIZ
2881 |     ;                                                                             | 28384 SYNTHESIZ
2882 | /*  <GROUP HEAD> ::= DO <STEP DEFINITION> ;     */                                | 28392 SYNTHESIZ
2883 |     ;                                                                             | 28392 SYNTHESIZ
2884 | /*  <GROUP HEAD> ::= DO <WHILE CLAUSE> ;     */                                   | 28400 SYNTHESIZ
2885 |     ;                                                                             | 28400 SYNTHESIZ
2886 | /*  <GROUP HEAD> ::= DO <CASE SELECTOR> ;     */                                  | 28408 SYNTHESIZ
2887 |     ;                                                                             | 28408 SYNTHESIZ
2888 | /*  <GROUP HEAD> ::= <GROUP HEAD> <STATEMENT>     */                             | 28416 SYNTHESIZ
2889 |     ;                                                                             | 28416 SYNTHESIZ
2890 | /*  <STEP DEFINITION> ::= <VARIABLE> <REPLACE> <EXPRESSION> <ITERATION CONTROL>|  | 28424 SYNTHESIZ
2891 |         */                                                                        | 28424 SYNTHESIZ
2892 |     CALL GENLOOP;                                                                 | 28424 SYNTHESIZ
2893 | /*  <ITERATION CONTROL> ::= <TO> <EXPRESSION>     */                             | 28440 SYNTHESIZ
2894 |  DO;                                                                              | 28440 SYNTHESIZ
2895 |     LOOPLIM=TABLE_LOC(SP);                                                        | 28448 SYNTHESIZ
2896 |     FIXV(SP)=1;                                                                   | 28464 SYNTHESIZ
```

```
2897 |     CALL FINDNO;                                                                    | 28480 SYNTHESIZ
2898 |     LOOPINC=TABLE_LOC(SP);                                                          | 28488 SYNTHESIZ
2899 |  END;                                                                               | 28504 SYNTHESIZ
2900 | /*  <ITERATION CONTROL> ::= <TO> <EXPRESSION> <BY> <EXPRESSION>     */              | 28504 SYNTHESIZ
2901 |  DO;                                                                                | 28504 SYNTHESIZ
2902 |     LOOPLIM=TABLE_LOC(SP-2);                                                        | 28512 SYNTHESIZ
2903 |     LOOPINC=TABLE_LOC(SP);                                                          | 28532 SYNTHESIZ
2904 |  END;                                                                               | 28540 SYNTHESIZ
2905 | /*  <WHILE CLAUSE> ::= <WHILE> <EXPRESSION>     */                                  | 28548 SYNTHESIZ
2906 |     ;                                                                               | 28548 SYNTHESIZ
2907 | /*  <CASE SELECTOR> ::= CASE <EXPRESSION>     */                                    | 28556 SYNTHESIZ
2908 |     ;                                                                               | 28556 SYNTHESIZ
2909 | /*  <PROCEDURE DEFINITION> ::= <PROCEDURE HEAD> <STATEMENT LIST> <ENDING>     */   | 28564 SYNTHESIZ
2910 |     ;                                                                               | 28564 SYNTHESIZ
2911 | /*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> ;     */                                  | 28572 SYNTHESIZ
2912 |     ;                                                                               | 28572 SYNTHESIZ
2913 | /*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> <TYPE> ;     */                          | 28580 SYNTHESIZ
2914 |     ;                                                                               | 28580 SYNTHESIZ
2915 | /*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> <PARAMETER LIST> ;     */                | 28588 SYNTHESIZ
2916 |     ;                                                                               | 28588 SYNTHESIZ
2917 | /*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> <PARAMETER LIST> <TYPE> ;     */         | 28596 SYNTHESIZ
2918 |     ;                                                                               | 28596 SYNTHESIZ
2919 | /*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> INTERRUPT <NUMBER> ;     */              | 28604 SYNTHESIZ
2920 |     ;                                                                               | 28604 SYNTHESIZ
2921 | /*  <PROCEDURE NAME> ::= <LABEL DEFINITION> PROCEDURE     */                       | 28612 SYNTHESIZ
2922 |     ;                                                                               | 28612 SYNTHESIZ
2923 | /*  <PARAMETER LIST> ::= <PARAMETER HEAD> <IDENTIFIER> )     */                    | 28620 SYNTHESIZ
2924 |     ;                                                                               | 28620 SYNTHESIZ
2925 | /*  <PARAMETER HEAD> ::= (     */                                                | 28628 SYNTHESIZ
2926 |     ;                                                                               | 28628 SYNTHESIZ
2927 | /*  <PARAMETER HEAD> ::= <PARAMETER HEAD> <IDENTIFIER> ,     */                    | 28636 SYNTHESIZ
2928 |     ;                                                                               | 28636 SYNTHESIZ
2929 | /*  <ENDING> ::= END     */                                                        | 28644 SYNTHESIZ
2930 |     ;                                                                               | 28644 SYNTHESIZ
2931 | /*  <ENDING> ::= END <IDENTIFIER>     */                                          | 28652 SYNTHESIZ
2932 |     ;                                                                               | 28652 SYNTHESIZ
2933 | /*  <ENDING> ::= <LABEL DEFINITION> <ENDING>     */                               | 28660 SYNTHESIZ
2934 |     ;                                                                               | 28660 SYNTHESIZ
2935 | /*  <LABEL DEFINITION> ::= <IDENTIFIER> :     */                                 | 28668 SYNTHESIZ
2936 |     DO;                                                                             | 28668 SYNTHESIZ
2937 |        CALL DEFINELAB(SP-1);                                                        | 28676 SYNTHESIZ
2938 |        CALL QUADGEN(LAB);                                                           | 28696 SYNTHESIZ
2939 |     END;                                                                            | 28712 SYNTHESIZ
2940 | /*  <LABEL DEFINITION> ::= <NUMBER> :     */                                      | 28712 SYNTHESIZ
2941 |     ;                                                                               | 28712 SYNTHESIZ
2942 | /*  <RETURN STATEMENT> ::= RETURN     */                                         | 28720 SYNTHESIZ
2943 |     ;                                                                               | 28720 SYNTHESIZ
2944 | /*  <RETURN STATEMENT> ::= RETURN <EXPRESSION>     */                            | 28728 SYNTHESIZ
2945 |     ;                                                                               | 28728 SYNTHESIZ
2946 | /*  <CALL STATEMENT> ::= CALL <VARIABLE>     */                                   | 28736 SYNTHESIZ
2947 |     ;                                                                               | 28736 SYNTHESIZ
2948 | /*  <GO TO STATEMENT> ::= <GO TO> <IDENTIFIER>     */                            | 28744 SYNTHESIZ
2949 |     DO;                                                                             | 28744 SYNTHESIZ
2950 |        CALL FINDLAB;                                                                | 28752 SYNTHESIZ
2951 |        CALL QUADGEN(BR);                                                            | 28760 SYNTHESIZ
2952 |     END;                                                                            | 28776 SYNTHESIZ
2953 | /*  <GO TO STATEMENT> ::= <GO TO> <NUMBER>     */                               | 28776 SYNTHESIZ
2954 |     ;                                                                               | 28776 SYNTHESIZ
2955 | /*  <GO TO> ::= GO TO     */                                                     | 28784 SYNTHESIZ
2956 |   DO;                                                                               | 28784 SYNTHESIZ
2957 |     TBASIC = 6;                                                                     | 28792 SYNTHESIZ
2958 |   END;                                                                              | 28800 SYNTHESIZ
2959 | /*  <GO TO> ::= GOTO     */                                                        | 28800 SYNTHESIZ
2960 |     ;                                                                               | 28800 SYNTHESIZ
2961 | /*  <DECLARATION STATEMENT> ::= DECLARE <DECLARATION ELEMENT>     */               | 28808 SYNTHESIZ
2962 |     ;                                                                               | 28808 SYNTHESIZ
```

```
2963 | /*  <DECLARATION STATEMENT> ::= <DECLARATION STATEMENT> , <DECLARATION ELEMENT>| 28810 SYNTHESIZ
2964 |         */                                                                    | 28816 SYNTHESIZ
2965 |     ;                                                                         | 28816 SYNTHESIZ
2966 | /*  <DECLARATION ELEMENT> ::= <TYPE DECLARATION>    */                         | 28824 SYNTHESIZ
2967 |     ;                                                                         | 28824 SYNTHESIZ
2968 | /*  <DECLARATION ELEMENT> ::= <IDENTIFIER> LITERALLY <STRING>    */          | 28832 SYNTHESIZ
2969 |     ;                                                                         | 28832 SYNTHESIZ
2970 | /*  <DECLARATION ELEMENT> ::= <IDENTIFIER> <DATA LIST>    */                  | 28840 SYNTHESIZ
2971 |     ;                                                                         | 28840 SYNTHESIZ
2972 | /*  <DATA LIST> ::= <DATA HEAD> <CONSTANT> )    */                           | 28848 SYNTHESIZ
2973 |     ;                                                                         | 28848 SYNTHESIZ
2974 | /*  <DATA HEAD> ::= DATA (    */                                            | 28856 SYNTHESIZ
2975 |     ;                                                                         | 28856 SYNTHESIZ
2976 | /*  <DATA HEAD> ::= <DATA HEAD> <CONSTANT> ,    */                           | 28864 SYNTHESIZ
2977 |     ;                                                                         | 28864 SYNTHESIZ
2978 | /*  <TYPE DECLARATION> ::= <IDENTIFIER SPECIFICATION> <TYPE>    */           | 28872 SYNTHESIZ
2979 |     ;                                                                         | 28872 SYNTHESIZ
2980 | /*  <TYPE DECLARATION> ::= <BOUND HEAD> <NUMBER> ) <TYPE>    */              | 28880 SYNTHESIZ
2981 |     CALL SETDIM;                                                             | 28880 SYNTHESIZ
2982 | /*  <TYPE DECLARATION> ::= <TYPE DECLARATION> <INITIAL LIST>    */          | 28896 SYNTHESIZ
2983 |     ;                                                                         | 28896 SYNTHESIZ
2984 | /*  <TYPE> ::= BYTE    */                                                    | 28904 SYNTHESIZ
2985 |     ;                                                                         | 28904 SYNTHESIZ
2986 | /*  <TYPE> ::= ADDRESS    */                                                 | 28912 SYNTHESIZ
2987 |     ;                                                                         | 28912 SYNTHESIZ
2988 | /*  <TYPE> ::= LABEL    */                                                    | 28920 SYNTHESIZ
2989 |     ;                                                                         | 28920 SYNTHESIZ
2990 | /*  <BOUND HEAD> ::= <IDENTIFIER SPECIFICATION> (    */                      | 28928 SYNTHESIZ
2991 |     ;                                                                         | 28928 SYNTHESIZ
2992 | /*  <IDENTIFIER SPECIFICATION> ::= <VARIABLE NAME>    */                     | 28936 SYNTHESIZ
2993 |         NVARDEF=1;                                                           | 28936 SYNTHESIZ
2994 | /*  <IDENTIFIER SPECIFICATION> ::= <IDENTIFIER LIST> <VARIABLE NAME> )    */ | 28952 SYNTHESIZ
2995 |     ;                                                                         | 28952 SYNTHESIZ
2996 | /*  <IDENTIFIER LIST> ::= (    */                                           | 28960 SYNTHESIZ
2997 |         NVARDEF=1;                                                           | 28960 SYNTHESIZ
2998 | /*  <IDENTIFIER LIST> ::= <IDENTIFIER LIST> <VARIABLE NAME> ,    */         | 28976 SYNTHESIZ
2999 |         NVARDEF=NVARDEF+1;                                                   | 28976 SYNTHESIZ
3000 | /*  <VARIABLE NAME> ::= <IDENTIFIER>    */                                  | 28996 SYNTHESIZ
3001 |         CALL PUTIT(SP);                                                      | 28996 SYNTHESIZ
3002 | /*  <VARIABLE NAME> ::= <BASED VARIABLE> <IDENTIFIER>    */                  | 29020 SYNTHESIZ
3003 |     ;                                                                         | 29020 SYNTHESIZ
3004 | /*  <BASED VARIABLE> ::= <IDENTIFIER> BASED    */                           | 29028 SYNTHESIZ
3005 |     ;                                                                         | 29028 SYNTHESIZ
3006 | /*  <INITIAL LIST> ::= <INITIAL HEAD> <CONSTANT> )    */                     | 29036 SYNTHESIZ
3007 |  ;                                                                           | 29036 SYNTHESIZ
3008 | /*  <INITIAL HEAD> ::= INITIAL (    */                                      | 29044 SYNTHESIZ
3009 |     ;                                                                         | 29044 SYNTHESIZ
3010 | /*  <INITIAL HEAD> ::= <INITIAL HEAD> <CONSTANT> ,    */                     | 29052 SYNTHESIZ
3011 |     ;                                                                         | 29052 SYNTHESIZ
3012 | /*  <ASSIGNMENT> ::= <VARIABLE> <REPLACE> <EXPRESSION>    */                 | 29060 SYNTHESIZ
3013 |     DO;                                                                      | 29060 SYNTHESIZ
3014 |         CALL QUADGEN(ASGN);                                                  | 29056 SYNTHESIZ
3015 |     DEFITABLE_LOC(SP-2))=CARD_COUNT;                                         | 29084 SYNTHESIZ
3016 |     END;                                                                     | 29112 SYNTHESIZ
3017 | /*  <ASSIGNMENT> ::= <LEFT PART> <ASSIGNMENT>    */                         | 29112 SYNTHESIZ
3018 |     ;                                                                         | 29112 SYNTHESIZ
3019 | /*  <REPLACE> ::= =    */                                                    | 29120 SYNTHESIZ
3020 |     DO;                                                  /*NEWQUAD*/       | 29120 SYNTHESIZ
3021 |        SUBSFLAG=GETQUAD(NEXTQUAD-1,1);                   /*NEWQUAD*/       | 29128 SYNTHESIZ
3022 |        IF SUBSFLAG=SUBS THEN                             /*NEWQUAD*/       | 29160 SYNTHESIZ
3023 |           CALL PUTQUAD(NEXTQUAD-1,1,SUBL);               /*NEWQUAD*/       | 29184 SYNTHESIZ
3024 |        END;                                              /*NEWQUAD*/       | 29212 SYNTHESIZ
3025 | /*  <LEFT PART> ::= <VARIABLE> ,    */                                      | 29212 SYNTHESIZ
3026 |     ;                                                                         | 29212 SYNTHESIZ
3027 | /*  <EXPRESSION> ::= <LOGICAL EXPRESSION>    */                             | 29220 SYNTHESIZ
3028 |     ;                                                                         | 29220 SYNTHESIZ
```

```
3029 | /*  <EXPRESSION> ::= <VARIABLE> := <LOGICAL EXPRESSION>    */                  | 29220 SYNTHESIZ
3030 |     ;                                                                       | 29228 SYNTHESIZ
3031 | /*  <LOGICAL EXPRESSION> ::= <LOGICAL FACTOR>    */                         | 29236 SYNTHESIZ
3032 |     ;                                                                       | 29236 SYNTHESIZ
3033 | /*  <LOGICAL EXPRESSION> ::= <LOGICAL EXPRESSION> OR <LOGICAL FACTOR>    */ | 29244 SYNTHESIZ
3034 |      CALL QUADGEN(OR);                                                      | 29244 SYNTHESIZ
3035 | /*  <LOGICAL EXPRESSION> ::= <LOGICAL EXPRESSION> XOR <LOGICAL FACTOR>    */ | 29268 SYNTHESIZ
3036 |     ;                                                                       | 29268 SYNTHESIZ
3037 | /*  <LOGICAL FACTOR> ::= <LOGICAL SECONDARY>    */                         | 29276 SYNTHESIZ
3038 |     ;                                                                       | 29276 SYNTHESIZ
3039 | /*  <LOGICAL FACTOR> ::= <LOGICAL FACTOR> AND <LOGICAL SECONDARY>    */    | 29284 SYNTHESIZ
3040 |      CALL QUADGEN(AND);                                                     | 29284 SYNTHESIZ
3041 | /*  <LOGICAL SECONDARY> ::= <LOGICAL PRIMARY>    */                        | 29308 SYNTHESIZ
3042 |     ;                                                                       | 29308 SYNTHESIZ
3043 | /*  <LOGICAL SECONDARY> ::= NOT <LOGICAL PRIMARY>    */                     | 29316 SYNTHESIZ
3044 |     ;                                                                       | 29316 SYNTHESIZ
3045 | /*  <LOGICAL PRIMARY> ::= <ARITHMETIC EXPRESSION>    */                     | 29324 SYNTHESIZ
3046 |     ;                                                                       | 29324 SYNTHESIZ
3047 | /*  <LOGICAL PRIMARY> ::= <ARITHMETIC EXPRESSION> <RELATION>               | 29332 SYNTHESIZ
3048 |      <ARITHMETIC EXPRESSION>    */                                          | 29332 SYNTHESIZ
3049 |   DO;                                                                       | 29332 SYNTHESIZ
3050 |      CALL QUADGEN(REL);                                                     | 29340 SYNTHESIZ
3051 |     TBASIC = 1;                                                             | 29356 SYNTHESIZ
3052 |   END;                                                                      | 29364 SYNTHESIZ
3053 | /*  <RELATION> ::= =    */                                                 | 29364 SYNTHESIZ
3054 |          FIXV(SP)=EQ;                                                       | 29364 SYNTHESIZ
3055 | /*  <RELATION> ::= <    */                                                 | 29388 SYNTHESIZ
3056 |          FIXV(SP)=LT;                                                       | 29388 SYNTHESIZ
3057 | /*  <RELATION> ::= >    */                                                 | 29412 SYNTHESIZ
3058 |          FIXV(SP)=GT;                                                       | 29412 SYNTHESIZ
3059 | /*  <RELATION> ::= <COMP>    */                                            | 29436 SYNTHESIZ
3060 |     ;                                                                       | 29436 SYNTHESIZ
3061 | /*  <COMP> ::= < >    */                                                   | 29444 SYNTHESIZ
3062 |          FIXV(SP-1)=NE;                                                     | 29444 SYNTHESIZ
3063 | /*  <COMP> ::= < =    */                                                   | 29472 SYNTHESIZ
3064 |          FIXV(SP-1)=LE;                                                     | 29472 SYNTHESIZ
3065 | /*  <COMP> ::= > =    */                                                   | 29500 SYNTHESIZ
3066 |          FIXV(SP-1)=GE;                                                     | 29500 SYNTHESIZ
3067 | /*  <ARITHMETIC EXPRESSION> ::= <TERM>    */                            | 29528 SYNTHESIZ
3068 |     ;                                                                       | 29528 SYNTHESIZ
3069 | /*  <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> + <TERM>    */     | 29536 SYNTHESIZ
3070 |          CALL QUADGEN(ADD);                                                 | 29536 SYNTHESIZ
3071 | /*  <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> - <TERM>    */     | 29560 SYNTHESIZ
3072 |          CALL QUADGEN(SUB);                                                 | 29560 SYNTHESIZ
3073 | /*  <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> PLUS <TERM>    */  | 29584 SYNTHESIZ
3074 |          CALL QUADGEN(ADD);                                                 | 29584 SYNTHESIZ
3075 | /*  <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> MINUS <TERM>    */ | 29608 SYNTHESIZ
3076 |          CALL QUADGEN(SUB);                                                 | 29608 SYNTHESIZ
3077 | /*  <ARITHMETIC EXPRESSION> ::= - <TERM>    */                           | 29632 SYNTHESIZ
3078 |      DO;                                                                    | 29632 SYNTHESIZ
3079 |       CALL QUADGEN(UMIN);                                                   | 29640 SYNTHESIZ
3080 |          CALL MOVESTACK (SP,SP-1);                                          | 29656 SYNTHESIZ
3081 |      END;                                                                   | 29684 SYNTHESIZ
3082 | /*  <TERM> ::= <PRIMARY>    */                                            | 29684 SYNTHESIZ
3083 |     ;                                                                       | 29684 SYNTHESIZ
3084 | /*  <TERM> ::= <TERM> * <PRIMARY>    */                                   | 29692 SYNTHESIZ
3085 |          CALL QUADGEN(MUL);                                                 | 29692 SYNTHESIZ
3086 | /*  <TERM> ::= <TERM> / <PRIMARY>    */                                   | 29716 SYNTHESIZ
3087 |          CALL QUADGEN(DIV);                                                 | 29716 SYNTHESIZ
3088 | /*  <TERM> ::= <TERM> MOD <PRIMARY>    */                                | 29740 SYNTHESIZ
3089 |          CALL QUADGEN(MMOD);                                                | 29740 SYNTHESIZ
3090 | /*  <PRIMARY> ::= <CONSTANT>    */                                         | 29764 SYNTHESIZ
3091 |          CALL FINDNO;                                                       | 29764 SYNTHESIZ
3092 | /*  <PRIMARY> ::= . <CONSTANT>    */                                       | 29780 SYNTHESIZ
3093 |     ;                                                                       | 29780 SYNTHESIZ
3094 | /*  <PRIMARY> ::= <CONSTANT HEAD> <CONSTANT> )    */                       | 29788 SYNTHESIZ
```

```
3095 |     ;                                                                                  | 29788 SYNTHESIZE
3096 | /*  <PRIMARY> ::= <VARIABLE>    */                                                     | 29796 SYNTHESIZE
3097 |     ;                                                                                  | 29796 SYNTHESIZE
3098 | /*  <PRIMARY> ::= . <VARIABLE>    */                                                  | 29804 SYNTHESIZE
3099 |     ;                                                                                  | 29804 SYNTHESIZE
3100 | /*  <PRIMARY> ::= ( <EXPRESSION> )    */                                             | 29812 SYNTHESIZE
3101 |         CALL MOVESTACK (SP-1,SP-2);                                                   | 29812 SYNTHESIZE
3102 | /*  <CONSTANT HEAD> ::= . (    */                                                     | 29852 SYNTHESIZE
3103 |     ;                                                                                  | 29852 SYNTHESIZE
3104 | /*  <CONSTANT HEAD> ::= <CONSTANT HEAD> <CONSTANT> ,    */                            | 29860 SYNTHESIZE
3105 |     ;                                                                                  | 29860 SYNTHESIZE
3106 | /*  <VARIABLE> ::= <IDENTIFIER>    */                                                 | 29868 SYNTHESIZE
3107 |         CALL FINDIT(SP);                                                               | 29868 SYNTHESIZE
3108 | /*  <VARIABLE> ::= <SUBSCRIPT HEAD> <EXPRESSION> )    */                              | 29892 SYNTHESIZE
3109 |     CALL QUADGEN(SUBS);                                                                | 29892 SYNTHESIZE
3110 | /*  <SUBSCRIPT HEAD> ::= <IDENTIFIER> (    */                                         | 29916 SYNTHESIZE
3111 |     DO;                                                                                | 29916 SYNTHESIZE
3112 |       CALL FINDIT(SP-1);                                                              | 29924 SYNTHESIZE
3113 |       I=TABLE_LOC(SP-1);                                                              | 29944 SYNTHESIZE
3114 |         IF SIZE(I) = 1 THEN                                                           | 29964 SYNTHESIZE
3115 |           DO;                                                                         | 29996 SYNTHESIZE
3116 |             OUTPUT=' VARIABLE '||SYMB(I)||' NOT A DIMENSIONED ARRAY';                | 29988 SYNTHESIZE
3117 |         END;                                                                          | 30050 SYNTHESIZE
3118 |     END;                                                                               | 30050 SYNTHESIZE
3119 | /*  <SUBSCRIPT HEAD> ::= <SUBSCRIPT HEAD> <EXPRESSION> ,    */                        | 30050 SYNTHESIZE
3120 |     ;                                                                                  | 30050 SYNTHESIZE
3121 | /*  <CONSTANT> ::= <STRING>    */                                                     | 30058 SYNTHESIZE
3122 |     ;                                                                                  | 30058 SYNTHESIZE
3123 | /*  <CONSTANT> ::= <NUMBER>    */                                                     | 30066 SYNTHESIZE
3124 |     ;                                                                                  | 30066 SYNTHESIZE
3125 | /*  <TO> ::= TO    */                                                                 | 30074 SYNTHESIZE
3126 | /*  <BY> ::= BY    */                                                                 | 30074 SYNTHESIZE
3127 |     ;                                                                                  | 30074 SYNTHESIZE
3128 | /*  <WHILE> ::= WHILE    */                                                           | 30082 SYNTHESIZE
3129 |     ;                                                                                  | 30082 SYNTHESIZE
3130 |   END;                                                                                 | 30050 SYNTHESIZE
3131 |END SYNTHESIZE;                                                                         | 30098 SYNTHESIZE
3132 |                                                                                        | 30104
3133 |                                                                                        | 30104
3134 |                                                                                        | 30104
3135 |                                                                                        | 30104
3136 |  /*              SYNTACTIC PARSING FUNCTIONS                                    */     | 30104
3137 |                                                                                        | 30104
3138 |                                                                                        | 30104
3139 |RIGHT_CONFLICT:                                                                         | 30104
3140 |   PROCEDURE (LEFT) BIT(1);                                                             | 30104
3141 |     DECLARE LEFT FIXED;                                                                | 30104 RIGHT_CONFLIC
3142 |     /*  THIS PROCEDURE IS TRUE IF TOKEN IS   A LEGAL RIGHT CONTEXT OF LEFT*/          | 30116 RIGHT_CONFLIC
3143 |     RETURN ("CO" & SHL(BYTE(C1(LEFT), SHR(TOKEN,2)), SHL(TOKEN,1)                     | 30116 RIGHT_CONFLIC
3144 |       & "06")) = 0;                                                                    | 30144 RIGHT_CONFLIC
3145 |   END RIGHT_CONFLICT;                                                                  | 30186 RIGHT_CONFLIC
3146 |                                                                                        | 30192
3147 |                                                                                        | 30192
3148 |RECOVER:                                                                                | 30192
3149 |   PROCEDURE;                                                                           | 30192
3150 |     /* IF THIS IS THE SECOND SUCCESSIVE CALL TO RECOVER, DISCARD ONE SYMBOL */         | 30192 RECOVER
3151 |     IF ¬ FAILSOFT THEN CALL SCAN;                                                      | 30192 RECOVER
3152 |     FAILSOFT = FALSE;                                                                  | 30230 RECOVER
3153 |     DO WHILE ¬ STOPIT(TOKEN);                                                          | 30236 RECOVER
3154 |       CALL SCAN;  /*.TO FIND SOMETHING SOLID IN THE TEXT  */                          | 30262 RECOVER
3155 |     END;                                                                               | 30266 RECOVER
3156 |     DO WHILE RIGHT_CONFLICT (PARSE_STACK(SP));                                         | 30274 RECOVER
3157 |       IF SP > 2 THEN SP = SP - 1;  /* AND IN THE STACK  */                             | 30308 RECOVER
3158 |       ELSE CALL SCAN;  /* BUT DON'T GO TOO FAR  */                                    | 30336 RECOVER
3159 |     END;                                                                               | 30348 RECOVER
3160 |     OUTPUT = 'RESUME:' || SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP+7);               | 30356 RECOVER
```

```
3161 |    END RECOVER;                                                                    | 30426 RECOVER
3162 |                                                                                    | 30434
3163 |STACKING:                                                                           | 30434
3164 |    PROCEDURE BIT(1);  /* STACKING DECISION FUNCTION */                             | 30434
3165 |       CALLCOUNT(1) = CALLCOUNT(1) + 1;                                             | 30434 STACKING
3166 |       DO FOREVER;    /* UNTIL RETURN  */                                          | 30474 STACKING
3167 |          DO CASE SHR(BYTE(C1(PARSE_STACK(SP)),SHR(TOKEN,2)),SHL(3-TOKEN,1)&6)&3;| 30474 STACKING
3168 |                                                                                    | 30546 STACKING
3169 |             /*  CASE 0  */                                                       | 30546 STACKING
3170 |             DO;    /* ILLEGAL SYMBOL PAIR  */                                   | 30546 STACKING
3171 |                CALL ERROR('ILLEGAL SYMBOL PAIR: ' || V(PARSE_STACK(SP)) || X11|| | 30546 STACKING
3172 |                   V(TOKEN), 1);                                                   | 30596 STACKING
3173 |                CALL STACK_DUMP;                                                    | 30636 STACKING
3174 |                CALL RECOVER;                                                       | 30644 STACKING
3175 |             END;                                                                   | 30652 STACKING
3176 |                                                                                    | 30652 STACKING
3177 |             /*  CASE 1  */                                                       | 30652 STACKING
3178 |                                                                                    | 30652 STACKING
3179 |             RETURN TRUE;       /*  STACK TOKEN  */                               | 30652 STACKING
3180 |                                                                                    | 30670 STACKING
3181 |             /*  CASE 2  */                                                       | 30670 STACKING
3182 |                                                                                    | 30670 STACKING
3183 |             RETURN FALSE;      /* DON'T STACK IT YET  */                         | 30670 STACKING
3184 |                                                                                    | 30686 STACKING
3185 |             /*  CASE 3  */                                                       | 30686 STACKING
3186 |                                                                                    | 30686 STACKING
3187 |             DO;       /* MUST CHECK TRIPLES  */                                | 30686 STACKING
3188 |                J = SHL(PARSE_STACK(SP-1), 16) + SHL(PARSE_STACK(SP), 8) + TOKEN;| 30694 STACKING
3189 |                I = -1;  K = NC1TRIPLES + 1;  /* BINARY SEARCH OF TRIPLES  */   | 30736 STACKING
3190 |                DO WHILE I + 1 < K;                                                 | 30756 STACKING
3191 |                   L = SHR(I+K, 1);                                                 | 30776 STACKING
3192 |                   IF C1TRIPLES(L) > J THEN K = L;                                  | 30792 STACKING
3193 |                   ELSE IF C1TRIPLES(L) < J THEN I = L;                             | 30824 STACKING
3194 |                   ELSE RETURN TRUE;  /* IT IS A VALID TRIPLE  */                  | 30864 STACKING
3195 |                END;                                                                | 30882 STACKING
3196 |                RETURN FALSE;                                                       | 30890 STACKING
3197 |             END;                                                                   | 30898 STACKING
3198 |                                                                                    | 30898 STACKING
3199 |          END;    /* OF DO CASE  */                                               | 30898 STACKING
3200 |       END;   /*  OF DO FOREVER */                                               | 30906 STACKING
3201 |    END STACKING;                                                                   | 30914 STACKING
3202 |                                                                                    | 30920
3203 |PR_OK:                                                                              | 30920
3204 |    PROCEDURE(PRD) BIT(1);                                                          | 30920
3205 |       /* DECISION PROCEDURE FOR CONTEXT CHECK OF EQUAL OR IMBEDDED RIGHT PARTS*/  | 30920 PR_OK
3206 |       DECLARE (H, I, J, PRD) FIXED;                                               | 30920 PR_OK
3207 |       DO CASE CONTEXT_CASE(PRD);                                                  | 30932 PR_OK
3208 |                                                                                    | 30962 PR_OK
3209 |          /*  CASE 0 -- NO CHECK REQUIRED  */                                     | 30962 PR_OK
3210 |                                                                                    | 30962 PR_OK
3211 |          RETURN TRUE;                                                              | 30962 PR_OK
3212 |                                                                                    | 30972 PR_OK
3213 |          /*  CASE 1 -- RIGHT CONTEXT CHECK  */                                   | 30972 PR_OK
3214 |                                                                                    | 30972 PR_OK
3215 |          RETURN ¬ RIGHT_CONFLICT (HDTB(PRD));                                     | 30972 PR_OK
3216 |                                                                                    | 31012 PR_OK
3217 |          /*  CASE 2 -- LEFT CONTEXT CHECK  */                                    | 31012 PR_OK
3218 |                                                                                    | 31012 PR_OK
3219 |          DO;                                                                       | 31012 PR_OK
3220 |             H = HDTB(PRD) - NT;                                                    | 31020 PR_OK
3221 |             I = PARSE_STACK(SP - PRLENGTH(PRD));                                  | 31038 PR_OK
3222 |             DO J = LEFT_INDEX(H-1) TO LEFT_INDEX(H) - 1;                          | 31064 PR_OK
3223 |                IF LEFT_CONTEXT(J) = I THEN RETURN TRUE;                            | 31128 PR_OK
3224 |             END;                                                                   | 31160 PR_OK
3225 |             RETURN FALSE;                                                          | 31168 PR_OK
3226 |          END;                                                                      | 31176 PR_OK
```

```
3227 |                                                                                  | 31176 PR_OK
3228 |         /*  CASE 3 -- CHECK TRIPLES  */                                        | 31176 PR_OK
3229 |                                                                                  | 31176 PR_OK
3230 |         DO;                                                                      | 31176 PR_OK
3231 |            H = HDTB(PRD) - NT;                                                  | 31184 PR_OK
3232 |            I = SHL(PARSE_STACK(SP - PRLENGTH(PRD)), 8) + TOKEN;                 | 31202 PR_OK
3233 |            DO J = TRIPLE_INDEX(H-1) TO TRIPLE_INDEX(H) - 1;                     | 31236 PR_OK
3234 |               IF CONTEXT_TRIPLE(J) = I THEN RETURN TRUE;                        | 31300 PR_OK
3235 |            END;                                                                 | 31334 PR_OK
3236 |            RETURN FALSE;                                                        | 31342 PR_OK
3237 |         END;                                                                    | 31350 PR_OK
3238 |                                                                                  | 31350 PR_OK
3239 |      END;  /* OF DO CASE  */                                                    | 31350 PR_OK
3240 |   END PR_OK;                                                                    | 31358 PR_OK
3241 |                                                                                  | 31364
3242 |                                                                                  | 31364
3243 |  /*                      ANALYSIS ALGORITHM                              */     | 31364
3244 |                                                                                  | 31364
3245 |                                                                                  | 31364
3246 |                                                                                  | 31364
3247 |REDUCE:                                                                          | 31364
3248 |   PROCEDURE;                                                                    | 31364
3249 |      DECLARE (I, J, PRD) FIXED;                                                 | 31364 REDUCE
3250 |      /* PACK STACK TOP INTO ONE WORD */                                         | 31376 REDUCE
3251 |      DO I = SP - 4 TO SP - 1;                                                   | 31376 REDUCE
3252 |         J = SHL(J, 8) + PARSE_STACK(I);                                         | 31420 REDUCE
3253 |      END;                                                                       | 31452 REDUCE
3254 |                                                                                  | 31460 REDUCE
3255 |      DO PRD = PR_INDEX(PARSE_STACK(SP)-1) TO PR_INDEX(PARSE_STACK(SP)) - 1;     | 31460 REDUCE
3256 |         IF (PPMASK(PRLENGTH(PRD)) & J) = PRTB(PRD) THEN                         | 31536 REDUCE
3257 |            IF PR_OK(PRD) THEN                                                   | 31586 REDUCE
3258 |            DO;  /* AN ALLOWED REDUCTION */                                     | 31586 REDUCE
3259 |               MP = SP - PRLENGTH(PRD) + 1; MPP1 = MP + 1;                       | 31606 REDUCE
3260 |               CALL SYNTHESIZE(PRDTB(PRD));                                      | 31642 REDUCE
3261 |               SP = MP;                                                          | 31664 REDUCE
3262 |               PARSE_STACK(SP) = HDTB(PRD);                                      | 31672 REDUCE
3263 |               RETURN;                                                           | 31690 REDUCE
3264 |            END;                                                                 | 31696 REDUCE
3265 |      END;                                                                       | 31696 REDUCE
3266 |                                                                                  | 31704 REDUCE
3267 |      /* LOOK UP HAS FAILED, ERROR CONDITION */                                  | 31704 REDUCE
3268 |      CALL ERROR('NO PRODUCTION IS APPLICABLE',1);                               | 31704 REDUCE
3269 |      CALL STACK_DUMP;                                                           | 31724 REDUCE
3270 |      FAILSOFT = FALSE;                                                          | 31732 REDUCE
3271 |      CALL RECOVER;                                                              | 31738 REDUCE
3272 |   END REDUCE;                                                                   | 31746 REDUCE
3273 |                                                                                  | 31752
3274 |COMPILATION_LOOP:                                                                | 31752
3275 |   PROCEDURE;                                                                    | 31752
3276 |                                                                                  | 31752 COMPILATION_LOO
3277 |      COMPILING = TRUE;                                                          | 31752 COMPILATION_LOO
3278 |      DO WHILE COMPILING;     /* ONCE AROUND FOR EACH PRODUCTION (REDUCTION)  */ | 31772 COMPILATION_LOO
3279 |         DO WHILE STACKING;                                                      | 31784 COMPILATION_LOO
3280 |            SP = SP + 1;                                                         | 31804 COMPILATION_LOO
3281 |            IF SP = STACKSIZE THEN                                               | 31816 COMPILATION_LOO
3282 |               DO;                                                               | 31840 COMPILATION_LOO
3283 |                  CALL ERROR ('STACK OVERFLOW *** CHECKING ABORTED ***', 2);     | 31832 COMPILATION_LO
3284 |                  RETURN;   /* THUS ABORTING CHECKING */                         | 31852 COMPILATION_LOO
3285 |               END;                                                              | 31858 COMPILATION_LOO
3286 |            PARSE_STACK(SP) = TOKEN;                                             | 31858 COMPILATION_LOO
3287 |            VAR(SP) = BCD;                                                       | 31870 COMPILATION_LOO
3288 |            FIXV(SP) = NUMBER_VALUE;                                             | 31886 COMPILATION_LOO
3289 |            CALL SCAN;                                                           | 31902 COMPILATION_LOO
3290 |         END;                                                                    | 31906 COMPILATION_LOO
3291 |                                                                                  | 31914 COMPILATION_LOO
3292 |         CALL REDUCE;                                                            | 31914 COMPILATION_LOO
```

```
3294 |    END COMPILATION_LOOP;
3295 |
3296 |
3297 |
3298 |
3299 |PRINT_SUMMARY:
3300 |    PROCEDURE;
3301 |       DECLARE I FIXED;
3302 |       CALL PRINT_DATE_AND_TIME ('END OF CHECKING ', DATE, TIME);
3303 |       OUTPUT = '';
3304 |       OUTPUT = CARD_COUNT || ' CARDS WERE CHECKED.';
3305 |       IF ERROR_COUNT = 0 THEN OUTPUT = 'NO ERRORS WERE DETECTED.';
3306 |       ELSE IF ERROR_COUNT > 1 THEN
3307 |          OUTPUT = ERROR_COUNT || ' ERRORS (' || SEVERE_ERRORS ||
3308 |                ' SEVERE) WERE DETECTED.';
3309 |       ELSE IF SEVERE_ERRORS = 1 THEN OUTPUT = 'ONE SEVERE ERROR WAS DETECTED.';
3310 |          ELSE OUTPUT = 'ONE ERROR WAS DETECTED.';
3311 |       IF PREVIOUS_ERROR > 0 THEN
3312 |          OUTPUT = 'THE LAST DETECTED ERROR WAS ON LINE ' || PREVIOUS_ERROR ||
3313 |                PERIOD;
3314 |       IF CONTROL(BYTE('D')) THEN CALL DUMPIT;
3315 |       DOUBLE_SPACE;
3316 |       CLOCK(3) = TIME;
3317 |       DO I = 1 TO 3;    /* WATCH OUT FOR MIDNIGHT */
3318 |          IF CLOCK(I) < CLOCK(I-1) THEN CLOCK(I) = CLOCK(I) +  8640000;
3319 |       END;
3320 |       CALL PRINT_TIME ('TOTAL TIME IN CHECKER     ', CLOCK(3) - CLOCK(0));
3321 |       CALL PRINT_TIME ('SET UP TIME               ', CLOCK(1) - CLOCK(0));
3322 |       CALL PRINT_TIME ('ACTUAL CHECKING TIME      ', CLOCK(2) - CLOCK(1));
3323 |       CALL PRINT_TIME ('CLEAN_UP TIME AT END      ', CLOCK(3) - CLOCK(2));
3324 |       IF CLOCK(2) > CLOCK(1) THEN    /* WATCH OUT FOR CLOCK BEING OFF */
3325 |       OUTPUT = 'CHECKING RATE: ' || 6000*CARD_COUNT/(CLOCK(2)-CLOCK(1)) ||
3326 |             ' CARDS PER MINUTE.';
3327 |    END PRINT_SUMMARY;
3328 |
3329 |MAIN_PROCEDURE:
3330 |    PROCEDURE;
3331 |       CLOCK(0) = TIME;  /* KEEP TRACK OF TIME IN EXECUTION */
3332 |       CALL INITIALIZATION;
3333 |
3334 |       CLOCK(1) = TIME;
3335 |
3336 |       CALL COMPILATION_LOOP;
3337 |
3338 |       CLOCK(2) = TIME;
3339 |
3340 |       /* CLOCK(3) GETS SET IN PRINT_SUMMARY */
3341 |       CALL PRINT_SUMMARY;
3342 |           FILE(1,RCD_BUFF)=QUADS;
3343 |           QUADS= FILE(1,0);
3344 |           RCD_BUFF=0;
3345 |       CALL PRINTSYMB;
3346 |          CALL PROCESS_QUADS;                                  /*NEWQUAD*/
3347 |       CALL PRINT_NEWQUADS;                                    /*NEWQUAD*/
3348 |
3349 |    END MAIN_PROCEDURE;
3350 |
3351 |
3352 |CALL MAIN_PROCEDURE;
3353 |RETURN SEVERE_ERRORS;
3354 |
3355 |EOF EOF EOF
```

```
*  FILE CONTROL BLOCK  39000    26000    3    2    13000    7056    12781
*  LOAD FILE WRITTEN.
END OF COMPILATION APRIL 4, 1979.  CLOCK TIME = 3:7:15.79.

3355 CARDS CONTAINING 1702 STATEMENTS WERE COMPILED.
NO ERRORS WERE DETECTED.
33054 BYTES OF PROGRAM, 18173 OF DATA. 2736 OF DESCRIPTORS. 4878 OF STRINGS.
 TOTAL CORE REQUIREMENT 58841 BYTES.

SYMBOL  TABLE  DUMP


ADD                 : FIXED      AT 3156(10),   DECLARED ON LINE 475 AND REFERENCED 3 TIMES.
ADDRSS              : BIT(8)     AT 345(9),     DECLARED ON LINE 534 AND REFERENCED 4 TIMES.
ALLOC_REG_OP        : LABEL      AT 17788(14),  DECLARED ON LINE 1963 AND REFERENCED 6 TIMES.
  PARAMETER  1      : CHARACTER  AT 2492(13),   DECLARED ON LINE 1971 AND REFERENCED 7 TIMES.
ALLOC_REG_RES       : LABEL      AT 17154(14),  DECLARED ON LINE 1904 AND REFERENCED 2 TIMES.
  PARAMETER  1      : CHARACTER  AT 2484(13),   DECLARED ON LINE 1908 AND REFERENCED 3 TIMES.
ALPHA               : CHARACTER  AT 1996(13),   DECLARED ON LINE 552 AND REFERENCED 2 TIMES.
ALPHABET            : CHARACTER  AT 912(13),    DECLARED ON LINE 376 AND REFERENCED 2 TIMES.
AND                 : FIXED      AT 3232(10),   DECLARED ON LINE 494 AND REFERENCED 1 TIMES.
ASGN                : FIXED      AT 3220(10),   DECLARED ON LINE 491 AND REFERENCED 4 TIMES.
B_R1                : LABEL      AT 8112(14),   DECLARED ON LINE 1135 AND REFERENCED 1 TIMES.
B_R2                : LABEL      AT 7270(14),   DECLARED ON LINE 1068 AND REFERENCED 2 TIMES.
BA_RO_READ          : LABEL      AT 6900(14),   DECLARED ON LINE 1043 AND REFERENCED 3 TIMES.
BCD                 : CHARACTER  AT 896(13),    DECLARED ON LINE 351 AND REFERENCED 8 TIMES.
BF                  : FIXED      AT 3188(10),   DECLARED ON LINE 483 AND REFERENCED 3 TIMES.
BR                  : FIXED      AT 3130(10),   DECLARED ON LINE 481 AND REFERENCED 4 TIMES.
BR_FLAG             : BIT(8)     AT 12(9),      DECLARED ON LINE 520 AND REFERENCED 10 TIMES.
BRANCHLAB           : LABEL      AT 22642(14),  DECLARED ON LINE 2398 AND REFERENCED 5 TIMES.
  PARAMETER  1      : CHARACTER  AT 2560(13),   DECLARED ON LINE 2400 AND REFERENCED 2 TIMES.
BT                  : FIXED      AT 3184(10),   DECLARED ON LINE 482 AND REFERENCED 3 TIMES.
BUF_PIC             : CHARACTER  AT 1808(13),   DECLARED ON LINE 523 AND REFERENCED 5 TIMES.
BUFFER              : CHARACTER  AT 916(13),    DECLARED ON LINE 386 AND REFERENCED 73 TIMES.
BZ                  : FIXED      AT 3228(10),   DECLARED ON LINE 493 AND REFERENCED 1 TIMES.
CALLCOUNT           : FIXED      AT 424(10),    DECLARED ON LINE 421 AND REFERENCED 6 TIMES.
CARD_COUNT          : FIXED      AT 260(10),    DECLARED ON LINE 387 AND REFERENCED 8 TIMES.
CHANGE              : BIT(8)     AT 123(9),     DECLARED ON LINE 528 AND REFERENCED 8 TIMES.
CHAR                : LABEL      AT 2332(14),   DECLARED ON LINE 672 AND REFERENCED 2 TIMES.
CHARTYPE            : BIT(8)     AT 3564(11),   DECLARED ON LINE 371 AND REFERENCED 6 TIMES.
CLOCK               : FIXED      AT 508(10),    DECLARED ON LINE 425 AND REFERENCED 20 TIMES.
COMPILATION_LOOP    : LABEL      AT 31760(14),  DECLARED ON LINE 3274 AND REFERENCED 1 TIMES.
COMPILING           : BIT(8)     AT 393(10),    DECLARED ON LINE 407 AND REFERENCED 4 TIMES.
CONLOC              : FIXED      AT 3296(10),   DECLARED ON LINE 512 AND REFERENCED 1 TIMES.
CONTEXT_CASE        : BIT(8)     AT 3182(11),   DECLARED ON LINE 321 AND REFERENCED 1 TIMES.
CONTEXT_TRIPLE      : FIXED      AT 3376(11),   DECLARED ON LINE 331 AND REFERENCED 1 TIMES.
CONTROL             : BIT(8)     AT 4076(11),   DECLARED ON LINE 372 AND REFERENCED 8 TIMES.
CONVAL              : FIXED      AT 3500(10),   DECLARED ON LINE 512 AND REFERENCED 16 TIMES.
CONVAL_INDEX        : LABEL      AT 11672(14),  DECLARED ON LINE 1478 AND REFERENCED 1 TIMES.
CORE                : FIXED      AT 3260(10),   DECLARED ON LINE 501 AND REFERENCED 0 TIMES.
CP                  : FIXED      AT 3552(11),   DECLARED ON LINE 351 AND REFERENCED 33 TIMES.
CURR_PIC            : FIXED      AT 16(9),      DECLARED ON LINE 521 AND REFERENCED 63 TIMES.
CI                  : CHARACTER  AT 460(13),    DECLARED ON LINE 450 AND REFERENCED 2 TIMES.
CITRIPLES           : FIXED      AT 1356(11),   DECLARED ON LINE 261 AND REFERENCED 2 TIMES.
DEALLOC_TEMP        : LABEL      AT 16110(14),  DECLARED ON LINE 1822 AND REFERENCED 5 TIMES.
  PARAMETER  1      : CHARACTER  AT 2476(13),   DECLARED ON LINE 1828 AND REFERENCED 2 TIMES.
  PARAMETER  2      : FIXED      AT 1156(8),    DECLARED ON LINE 1827 AND REFERENCED 12 TIMES.
DEALLOCABL          : BIT(8)     AT 392(9),     DECLARED ON LINE 537 AND REFERENCED 9 TIMES.
DEALLOCATE          : LABEL      AT 15032(14),  DECLARED ON LINE 1725 AND REFERENCED 2 TIMES.
DEF                 : FIXED      AT 1880(10),   DECLARED ON LINE 466 AND REFERENCED 4 TIMES.
DEFINELAB           : LABEL      AT 24790(14),  DECLARED ON LINE 2569 AND REFERENCED 6 TIMES.
  PARAMETER  1      : FIXED      AT 1580(8),    DECLARED ON LINE 2574 AND REFERENCED 4 TIMES.
DIV                 : FIXED      AT 3168(10),   DECLARED ON LINE 478 AND REFERENCED 1 TIMES.
DIVIDE              : FIXED      AT 288(10),    DECLARED ON LINE 396 AND REFERENCED 2 TIMES.
```

```
DIVIDE                : FIXED      AT 3242(10),   DECLARED ON LINE 510 AND REFERENCED 3 TIMES.
DO_SWITCH             : FIXED      AT 3234(10),   DECLARED ON LINE 509 AND REFERENCED 0 TIMES.
DOUBLE                : CHARACTER  AT 904(13),    DECLARED ON LINE 355 AND REFERENCED 9 TIMES.
DUMPIT                : LABEL      AT 5250(14),   DECLARED ON LINE 910 AND REFERENCED 1 TIMES.
EOFILE                : FIXED      AT 292(10),    DECLARED ON LINE 396 AND REFERENCED 4 TIMES.
EQ                    : FIXED      AT 3196(10),   DECLARED ON LINE 485 AND REFERENCED 1 TIMES.
ERROR                 : LABEL      AT 1526(14),   DECLARED ON LINE 611 AND REFERENCED 6 TIMES.
  PARAMETER  1        : CHARACTER  AT 2128(13),   DECLARED ON LINE 615 AND REFERENCED 1 TIMES.
  PARAMETER  2        : FIXED      AT 56(8),      DECLARED ON LINE 615 AND REFERENCED 1 TIMES.
ERROR_COUNT           : FIXED      AT 264(10),    DECLARED ON LINE 387 AND REFERENCED 5 TIMES.
FAILSOFT              : BIT(8)     AT 397(10),    DECLARED ON LINE 407 AND REFERENCED 4 TIMES.
FIND_OP               : LABEL      AT 17488(14),  DECLARED ON LINE 1935 AND REFERENCED 1 TIMES.
  PARAMETER  1        : CHARACTER  AT 2488(13),   DECLARED ON LINE 1943 AND REFERENCED 1 TIMES.
FIND_RES              : LABEL      AT 16540(14),  DECLARED ON LINE 1855 AND REFERENCED 2 TIMES.
  PARAMETER  1        : CHARACTER  AT 2480(13),   DECLARED ON LINE 1864 AND REFERENCED 2 TIMES.
FINDIT                : LABEL      AT 25626(14),  DECLARED ON LINE 2644 AND REFERENCED 2 TIMES.
  PARAMETER  1        : FIXED      AT 1624(8),    DECLARED ON LINE 2646 AND REFERENCED 3 TIMES.
FINDLAB               : LABEL      AT 22344(14),  DECLARED ON LINE 2370 AND REFERENCED 2 TIMES.
FINDNO                : LABEL      AT 25834(14),  DECLARED ON LINE 2664 AND REFERENCED 2 TIMES.
FIXV                  : FIXED      AT 624(10),    DECLARED ON LINE 445 AND REFERENCED 14 TIMES.
GE                    : FIXED      AT 3216(10),   DECLARED ON LINE 490 AND REFERENCED 1 TIMES.
GENLOOP               : LABEL      AT 26208(14),  DECLARED ON LINE 2701 AND REFERENCED 1 TIMES.
GET_CARD              : LABEL      AT 1876(14),   DECLARED ON LINE 642 AND REFERENCED 4 TIMES.
GETQUAD               : LABEL      AT 21780(14),  DECLARED ON LINE 2314 AND REFERENCED 3 TIMES.
  PARAMETER  1        : FIXED      AT 1496(8),    DECLARED ON LINE 2316 AND REFERENCED 2 TIMES.
  PARAMETER  2        : FIXED      AT 1500(8),    DECLARED ON LINE 2316 AND REFERENCED 1 TIMES.
GT                    : FIXED      AT 3204(10),   DECLARED ON LINE 487 AND REFERENCED 2 TIMES.
HALT                  : FIXED      AT 3176(10),   DECLARED ON LINE 480 AND REFERENCED 4 TIMES.
HOTB                  : BIT(8)     AT 2922(11),   DECLARED ON LINE 307 AND REFERENCED 4 TIMES.
I                     : FIXED      AT 532(10),    DECLARED ON LINE 433 AND REFERENCED 131 TIMES.
I_FORMAT              : CHARACTER  PROCEDURE AT 1400(14);  DECLARED ON LINE 501 AND REFERENCED 2 TIMES.
  PARAMETER  1        : FIXED      AT 40(8),      DECLARED ON LINE 603 AND REFERENCED 1 TIMES.
  PARAMETER  2        : FIXED      AT 44(8),      DECLARED ON LINE 603 AND REFERENCED 2 TIMES.
IDENT                 : FIXED      AT 280(10),    DECLARED ON LINE 396 AND REFERENCED 3 TIMES.
II                    : FIXED      AT 500(9),     DECLARED ON LINE 542 AND REFERENCED 96 TIMES.
INIT                  : FIXED      AT 2688(10),   DECLARED ON LINE 466 AND REFERENCED 3 TIMES.
INITIALIZATION        : LABEL      AT 4084(14),   DECLARED ON LINE 648 AND REFERENCED 1 TIMES.
J                     : FIXED      AT 536(10),    DECLARED ON LINE 433 AND REFERENCED 17 TIMES.
JJ                    : FIXED      AT 504(9),     DECLARED ON LINE 542 AND REFERENCED 31 TIMES.
K                     : FIXED      AT 540(10),    DECLARED ON LINE 433 AND REFERENCED 6 TIMES.
KK                    : FIXED      AT 508(9),     DECLARED ON LINE 542 AND REFERENCED 7 TIMES.
L                     : FIXED      AT 544(10),    DECLARED ON LINE 433 AND REFERENCED 10 TIMES.
LAB                   : FIXED      AT 3248(10),   DECLARED ON LINE 495 AND REFERENCED 9 TIMES.
LAB_INDEX             : LABEL      AT 11566(14),  DECLARED ON LINE 1467 AND REFERENCED 3 TIMES.
LABDEF                : FIXED      AT 1052(10),   DECLARED ON LINE 458 AND REFERENCED 5 TIMES.
LABID                 : CHARACTER  AT 1249(13),   DECLARED ON LINE 457 AND REFERENCED 12 TIMES.
LASTREF               : FIXED      AT 948(10),    DECLARED ON LINE 458 AND REFERENCED 1 TIMES.
LAVS                  : FIXED      AT 124(9),     DECLARED ON LINE 526 AND REFERENCED 23 TIMES.
LE                    : FIXED      AT 3212(10),   DECLARED ON LINE 489 AND REFERENCED 1 TIMES.
LEFT_CONTEXT          : BIT(8)     AT 3212(11),   DECLARED ON LINE 327 AND REFERENCED 1 TIMES.
LEFT_INDEX            : BIT(8)     AT 3317(11),   DECLARED ON LINE 328 AND REFERENCED 2 TIMES.
LL                    : FIXED      AT 512(9),     DECLARED ON LINE 542 AND REFERENCED 72 TIMES.
LOC_QUAD              : FIXED      AT 8(9),       DECLARED ON LINE 519 AND REFERENCED 15 TIMES.
LOCAT                 : FIXED      AT 1476(10),   DECLARED ON LINE 466 AND REFERENCED 4 TIMES.
LOOP_INDEX            : FIXED      AT 3276(10),   DECLARED ON LINE 505 AND REFERENCED 4 TIMES.
LOOPINC               : FIXED      AT 3272(10),   DECLARED ON LINE 504 AND REFERENCED 3 TIMES.
LOOPLIM               : FIXED      AT 3268(10),   DECLARED ON LINE 503 AND REFERENCED 3 TIMES.
LT                    : FIXED      AT 3200(10),   DECLARED ON LINE 486 AND REFERENCED 2 TIMES.
MAIN_PROCEDURE        : LABEL      AT 32842(14),  DECLARED ON LINE 3329 AND REFERENCED 1 TIMES.
MARGIN_CHOP           : FIXED      AT 3560(11),   DECLARED ON LINE 361 AND REFERENCED 6 TIMES.
MAXREG                : FIXED      AT 496(9),     DECLARED ON LINE 541 AND REFERENCED 8 TIMES.
MIC_GEN               : LABEL      AT 8992(14),   DECLARED ON LINE 1208 AND REFERENCED 1 TIMES.
MIC_LOC               : FIXED      AT 20(9),      DECLARED ON LINE 522 AND REFERENCED 3 TIMES.
MMOD                  : FIXED      AT 3172(10),   DECLARED ON LINE 479 AND REFERENCED 2 TIMES.
MOVESTACK             : LABEL      AT 26100(14),  DECLARED ON LINE 2688 AND REFERENCED 2 TIMES.
  PARAMETER  1        : FIXED      AT 1648(8),    DECLARED ON LINE 2690 AND REFERENCED 4 TIMES.
```

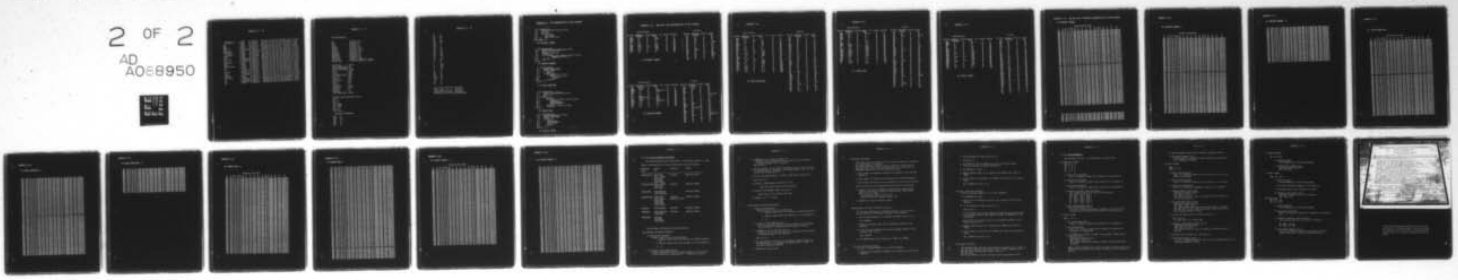
```
  PARAMETER  2      : FIXED      AT 1652(8),     DECLARED ON LINE 2691 AND REFERENCED 4 TIMES.
MP                  : FIXED      AT 932(10),     DECLARED ON LINE 451 AND REFERENCED 4 TIMES.
MPP1                : FIXED      AT 936(10),     DECLARED ON LINE 451 AND REFERENCED 1 TIMES.
MUL                 : FIXED      AT 3160(10),    DECLARED ON LINE 476 AND REFERENCED 2 TIMES.
MULLOC              : FIXED      AT 3288(10),    DECLARED ON LINE 510 AND REFERENCED 0 TIMES.
NCONSTANT           : FIXED      AT 3152(10),    DECLARED ON LINE 474 AND REFERENCED 6 TIMES.
NE                  : FIXED      AT 3208(10),    DECLARED ON LINE 488 AND REFERENCED 1 TIMES.
NEWQUAD             : FIXED      AT 608(9),      DECLARED ON LINE 547 AND REFERENCED 39 TIMES.
NEWQUAD_GEN         : LABEL      AT 18662(14),   DECLARED ON LINE 2039 AND REFERENCED 1 TIMES.
NEWQUADNO           : FIXED      AT 0(8),        DECLARED ON LINE 545 AND REFERENCED 7 TIMES.
NEXT_VAR            : FIXED      AT 172(9),      DECLARED ON LINE 529 AND REFERENCED 25 TIMES.
NEXTQUAD            : FIXED      AT 1168(10),    DECLARED ON LINE 462 AND REFERENCED 28 TIMES.
NLABEL              : FIXED      AT 944(10),     DECLARED ON LINE 455 AND REFERENCED 22 TIMES.
NN                  : FIXED      AT 516(9),      DECLARED ON LINE 542 AND REFERENCED 0 TIMES.
NO_OP               : LABEL      AT 8482(14),    DECLARED ON LINE 1164 AND REFERENCED 2 TIMES.
NOT_LETTER_OR_DIGIT: BIT(8)      AT 0(10),       DECLARED ON LINE 372 AND REFERENCED 4 TIMES.
NSYMBOL             : FIXED      AT 3148(10),    DECLARED ON LINE 473 AND REFERENCED 22 TIMES.
NUMBER              : FIXED      AT 234(10),     DECLARED ON LINE 396 AND REFERENCED 2 TIMES.
NUMBER_VALUE        : FIXED      AT 276(10),     DECLARED ON LINE 392 AND REFERENCED 4 TIMES.
NVARDEF             : FIXED      AT 3144(10),    DECLARED ON LINE 472 AND REFERENCED 5 TIMES.
OPER                : FIXED      AT 20(8),       DECLARED ON LINE 551 AND REFERENCED 22 TIMES.
OPERAND1            : CHARACTER AT 1816(13),     DECLARED ON LINE 525 AND REFERENCED 23 TIMES.
OPERAND2            : CHARACTER AT 1820(13),     DECLARED ON LINE 525 AND REFERENCED 17 TIMES.
OPERATION           : CHARACTER AT 1812(13),     DECLARED ON LINE 525 AND REFERENCED 2 TIMES.
OPRND1              : FIXED      AT 8(8),        DECLARED ON LINE 550 AND REFERENCED 16 TIMES.
OPRND2              : FIXED      AT 12(8),       DECLARED ON LINE 550 AND REFERENCED 14 TIMES.
OPRTOR              : FIXED      AT 4(8),        DECLARED ON LINE 550 AND REFERENCED 14 TIMES.
OR                  : FIXED      AT 3236(10),    DECLARED ON LINE 495 AND REFERENCED 2 TIMES.
OUT_MIC             : LABEL      AT 6182(14),    DECLARED ON LINE 980 AND REFERENCED 21 TIMES.
PAD                 : CHARACTER PROCEDURE AT 1290(14),   DECLARED ON LINE 592 AND REFERENCED 28 TIMES.
  PARAMETER  1      : CHARACTER AT 2120(13),     DECLARED ON LINE 594 AND REFERENCED 3 TIMES.
  PARAMETER  2      : FIXED      AT 28(8),       DECLARED ON LINE 594 AND REFERENCED 2 TIMES.
PAGE                : CHARACTER AT 900(13),      DECLARED ON LINE 355 AND REFERENCED 1 TIMES.
PARSE_STACK         : BIT(8)     AT 548(10),     DECLARED ON LINE 442 AND REFERENCED 16 TIMES.
PERIOD              : CHARACTER AT 940(13),      DECLARED ON LINE 430 AND REFERENCED 1 TIMES.
POINT               : FIXED      AT 464(9),      DECLARED ON LINE 540 AND REFERENCED 17 TIMES.
POINTER             : CHARACTER AT 928(13),      DECLARED ON LINE 419 AND REFERENCED 2 TIMES.
PR_INDEX            : BIT(8)     AT 3438(11),    DECLARED ON LINE 335 AND REFERENCED 2 TIMES.
PR_OK               : LABEL      AT 30928(14),   DECLARED ON LINE 3203 AND REFERENCED 1 TIMES.
  PARAMETER  1      : FIXED      AT 2244(8),     DECLARED ON LINE 3206 AND REFERENCED 6 TIMES.
PRTB                : BIT(8)     AT 2792(11),    DECLARED ON LINE 299 AND REFERENCED 1 TIMES.
PREVIOUS_ERROR      : FIXED      AT 272(10),     DECLARED ON LINE 337 AND REFERENCED 4 TIMES.
PRINT_DATE_AND_TIME: LABEL       AT 3788(14),    DECLARED ON LINE 827 AND REFERENCED 3 TIMES.
  PARAMETER  1      : CHARACTER AT 2212(13),     DECLARED ON LINE 829 AND REFERENCED 1 TIMES.
  PARAMETER  2      : FIXED      AT 212(8),      DECLARED ON LINE 829 AND REFERENCED 2 TIMES.
  PARAMETER  3      : FIXED      AT 216(8),      DECLARED ON LINE 829 AND REFERENCED 1 TIMES.
PRINT_NEWQUADS      : LABEL      AT 12688(14),   DECLARED ON LINE 1547 AND REFERENCED 1 TIMES.
PRINT_SUMMARY       : LABEL      AT 31944(14),   DECLARED ON LINE 3299 AND REFERENCED 1 TIMES.
PRINT_TIME          : LABEL      AT 3479(14),    DECLARED ON LINE 817 AND REFERENCED 5 TIMES.
  PARAMETER  1      : CHARACTER AT 2188(13),     DECLARED ON LINE 819 AND REFERENCED 5 TIMES.
  PARAMETER  2      : FIXED      AT 188(8),      DECLARED ON LINE 819 AND REFERENCED 7 TIMES.
PRINTSYMB           : LABEL      AT 20506(14),   DECLARED ON LINE 2259 AND REFERENCED 1 TIMES.
PRLENGTH            : BIT(8)     AT 3052(11),    DECLARED ON LINE 315 AND REFERENCED 4 TIMES.
PRMASK              : FIXED      AT 400(10),     DECLARED ON LINE 413 AND REFERENCED 1 TIMES.
PROCESS_QUADS       : LABEL      AT 20294(14),   DECLARED ON LINE 2241 AND REFERENCED 1 TIMES.
PRTB                : FIXED      AT 2272(11),    DECLARED ON LINE 290 AND REFERENCED 1 TIMES.
PUT_MIC             : LABEL      AT 5848(14),    DECLARED ON LINE 958 AND REFERENCED 7 TIMES.
  PARAMETER  1      : FIXED      AT 500(8),      DECLARED ON LINE 964 AND REFERENCED 3 TIMES.
PUT_NEWQUAD         : LABEL      AT 11354(14),   DECLARED ON LINE 1446 AND REFERENCED 13 TIMES.
PUTIT               : LABEL      AT 25248(14),   DECLARED ON LINE 2613 AND REFERENCED 1 TIMES.
  PARAMETER  1      : FIXED      AT 1605(8),     DECLARED ON LINE 2615 AND REFERENCED 4 TIMES.
PUTQUAD             : LABEL      AT 21956(14),   DECLARED ON LINE 2330 AND REFERENCED 18 TIMES.
  PARAMETER  1      : FIXED      AT 1508(8),     DECLARED ON LINE 2332 AND REFERENCED 2 TIMES.
  PARAMETER  2      : FIXED      AT 1512(8),     DECLARED ON LINE 2332 AND REFERENCED 1 TIMES.
  PARAMETER  3      : FIXED      AT 1516(8),     DECLARED ON LINE 2332 AND REFERENCED 1 TIMES.
PUTTEMP             : LABEL      AT 22134(14),   DECLARED ON LINE 2349 AND REFERENCED 5 TIMES.
```

```
                    : FIXED     AT 1524(5),    DECLARED ON LINE 2421 AND REFERENCED 2 TIMES.
QUADGEN             : LABEL     AT 22306(14),  DECLARED ON LINE 2415 AND REFERENCED 29 TIMES.
  PARAMETER  1      : FIXED     AT 1552(8),    DECLARED ON LINE 2421 AND REFERENCED 12 TIMES.
QUADS               : FIXED     AT 3704(10),   DECLARED ON LINE 516 AND REFERENCED 18 TIMES.
RCD_BUFF            : FIXED     AT 0(9),       DECLARED ON LINE 517 AND REFERENCED 18 TIMES.
RCD_NO              : FIXED     AT 4(9),       DECLARED ON LINE 518 AND REFERENCED 11 TIMES.
RD                  : BIT(8)    AT 336(9),     DECLARED ON LINE 532 AND REFERENCED 10 TIMES.
READQUAD            : LABEL     AT 11794(14),  DECLARED ON LINE 1489 AND REFERENCED 1 TIMES.
  PARAMETER  1      : FIXED     AT 836(8),     DECLARED ON LINE 1493 AND REFERENCED 4 TIMES.
RECOVER             : LABEL     AT 30200(14),  DECLARED ON LINE 3148 AND REFERENCED 2 TIMES.
REDUCE              : LABEL     AT 31372(14),  DECLARED ON LINE 3247 AND REFERENCED 1 TIMES.
REFNO               : FIXED     AT 520(9),     DECLARED ON LINE 543 AND REFERENCED 9 TIMES.
REFRENCE            : FIXED     AT 400(9),     DECLARED ON LINE 538 AND REFERENCED 5 TIMES.
REGNO               : FIXED     AT 580(9),     DECLARED ON LINE 545 AND REFERENCED 12 TIMES.
REL                 : FIXED     AT 3192(10),   DECLARED ON LINE 484 AND REFERENCED 4 TIMES.
RESERVED_LIMIT      : FIXED     AT 3556(11),   DECLARED ON LINE 361 AND REFERENCED 3 TIMES.
RESULT              : CHARACTER AT 1824(13),   DECLARED ON LINE 525 AND REFERENCED 21 TIMES.
RIGHT_CONFLICT      : LABEL     AT 30112(14),  DECLARED ON LINE 3139 AND REFERENCED 2 TIMES.
  PARAMETER  1      : FIXED     AT 2192(8),    DECLARED ON LINE 3141 AND REFERENCED 1 TIMES.
RSLT                : FIXED     AT 16(8),      DECLARED ON LINE 550 AND REFERENCED 14 TIMES.
RTEMP_MINUS_1       : LABEL     AT 8704(14),   DECLARED ON LINE 1186 AND REFERENCED 1 TIMES.
RTEMP_SHIFT         : LABEL     AT 7928(14),   DECLARED ON LINE 1119 AND REFERENCED 2 TIMES.
RTEMP_UNIBUS        : LABEL     AT 6876(14),   DECLARED ON LINE 1031 AND REFERENCED 2 TIMES.
RTEMP_4             : LABEL     AT 7388(14),   DECLARED ON LINE 1078 AND REFERENCED 1 TIMES.
R0_PLUS2_READ       : LABEL     AT 6414(14),   DECLARED ON LINE 994 AND REFERENCED 3 TIMES.
R2_SHIFT            : LABEL     AT 8222(14),   DECLARED ON LINE 1144 AND REFERENCED 1 TIMES.
R3_D                : LABEL     AT 7120(14),   DECLARED ON LINE 1055 AND REFERENCED 3 TIMES.
R3_R3_PLUS_B        : LABEL     AT 8524(14),   DECLARED ON LINE 1171 AND REFERENCED 1 TIMES.
R3_UNIBUS           : LABEL     AT 6726(14),   DECLARED ON LINE 1019 AND REFERENCED 3 TIMES.
R3_0                : LABEL     AT 7716(14),   DECLARED ON LINE 1103 AND REFERENCED 1 TIMES.
S                   : CHARACTER AT 924(13),    DECLARED ON LINE 409 AND REFERENCED 12 TIMES.
SAVEINDEX           : FIXED     AT 3264(10),   DECLARED ON LINE 502 AND REFERENCED 4 TIMES.
SAVELAB             : CHARACTER AT 1800(13),   DECLARED ON LINE 507 AND REFERENCED 0 TIMES.
SAVELOC             : FIXED     AT 3280(10),   DECLARED ON LINE 506 AND REFERENCED 1 TIMES.
SAVEQUAD            : FIXED     AT 1160(10),   DECLARED ON LINE 460 AND REFERENCED 0 TIMES.
SAVEQUAD2           : FIXED     AT 1164(10),   DECLARED ON LINE 461 AND REFERENCED 5 TIMES.
SAVEREF             : FIXED     AT 940(10),    DECLARED ON LINE 453 AND REFERENCED 0 TIMES.
SAVEVAR             : CHARACTER AT 1804(13),   DECLARED ON LINE 508 AND REFERENCED 1 TIMES.
SAVLAB              : CHARACTER AT 1756(13),   DECLARED ON LINE 469 AND REFERENCED 2 TIMES.
SAVLABNO            : FIXED     AT 3140(10),   DECLARED ON LINE 470 AND REFERENCED 6 TIMES.
SAVQUD              : FIXED     AT 3092(10),   DECLARED ON LINE 467 AND REFERENCED 14 TIMES.
SAVQUDNO            : FIXED     AT 3136(10),   DECLARED ON LINE 468 AND REFERENCED 28 TIMES.
SCAN                : LABEL     AT 2380(14),   DECLARED ON LINE 681 AND REFERENCED 5 TIMES.
SET_BIT             : LABEL     AT 5664(14),   DECLARED ON LINE 937 AND REFERENCED 183 TIMES.
  PARAMETER  1      : FIXED     AT 392(8),     DECLARED ON LINE 941 AND REFERENCED 1 TIMES.
SET_FIELD           : LABEL     AT 5698(14),   DECLARED ON LINE 944 AND REFERENCED 22 TIMES.
  PARAMETER  1      : FIXED     AT 400(8),     DECLARED ON LINE 948 AND REFERENCED 3 TIMES.
  PARAMETER  2      : FIXED     AT 404(8),     DECLARED ON LINE 948 AND REFERENCED 1 TIMES.
  PARAMETER  3      : FIXED     AT 408(8),     DECLARED ON LINE 948 AND REFERENCED 2 TIMES.
SETDIM              : LABEL     AT 25122(14),  DECLARED ON LINE 2600 AND REFERENCED 1 TIMES.
SEVERE_ERRORS       : FIXED     AT 268(10),    DECLARED ON LINE 387 AND REFERENCED 6 TIMES.
SIZE                : FIXED     AT 2284(10),   DECLARED ON LINE 466 AND REFERENCED 5 TIMES.
SORT                : LABEL     AT 14174(14),  DECLARED ON LINE 1667 AND REFERENCED 2 TIMES.
  PARAMETER  1      : FIXED     AT 1040(8),    DECLARED ON LINE 1672 AND REFERENCED 2 TIMES.
  PARAMETER  2      : BIT(8)    AT 1036(8),    DECLARED ON LINE 1671 AND REFERENCED 2 TIMES.
SORTNUM             : FIXED     AT 552(9),     DECLARED ON LINE 545 AND REFERENCED 14 TIMES.
SORTREF             : FIXED     AT 524(9),     DECLARED ON LINE 544 AND REFERENCED 14 TIMES.
SP                  : FIXED     AT 928(10),    DECLARED ON LINE 451 AND REFERENCED 117 TIMES.
STACK_DUMP          : LABEL     AT 5450(14),   DECLARED ON LINE 921 AND REFERENCED 3 TIMES.
STACKING            : LABEL     AT 30442(14),  DECLARED ON LINE 3163 AND REFERENCED 1 TIMES.
STATUS              : BIT(8)    AT 337(9),     DECLARED ON LINE 533 AND REFERENCED 11 TIMES.
STOPIT              : BIT(8)    AT 296(10),    DECLARED ON LINE 407 AND REFERENCED 3 TIMES.
STOREQUAD           : LABEL     AT 21576(14),  DECLARED ON LINE 2295 AND REFERENCED 1 TIMES.
  PARAMETER  1      : FIXED     AT 1472(8),    DECLARED ON LINE 2297 AND REFERENCED 2 TIMES.
  PARAMETER  2      : FIXED     AT 1476(8),    DECLARED ON LINE 2297 AND REFERENCED 1 TIMES.
  PARAMETER  3      : FIXED     AT 1480(8),    DECLARED ON LINE 2297 AND REFERENCED 1 TIMES.
```

AD-A068 950 GEORGE WASHINGTON UNIV WASHINGTON D C SCHOOL OF ENGI--ETC F/G 9/2
DEVELOPMENT OF EXPERIMENTAL COMPILERS TO GENERATE EMULATORS FOR--ETC(U)
APR 79 R E MERWIN DAS60-78-C-0115

UNCLASSIFIED NL

2 OF 2
AD
A068950



END
DATE
FILMED
7-79
DDC

```
  PARAMETER  4     : FIXED      AT 1494(8),    DECLARED ON LINE 2297 AND REFERENCED 1 TIMES.
  PARAMETER  5     : FIXED      AT 1498(8),    DECLARED ON LINE 2297 AND REFERENCED 1 TIMES.
SUB                : FIXED      AT 3164(10),   DECLARED ON LINE 477 AND REFERENCED 2 TIMES.
SUBL               : FIXED      AT 3252(10),   DECLARED ON LINE 499 AND REFERENCED 2 TIMES.
SUBS               : FIXED      AT 3224(10),   DECLARED ON LINE 492 AND REFERENCED 5 TIMES.
SUBSFLAG           : FIXED      AT 356(9),     DECLARED ON LINE 535 AND REFERENCED 4 TIMES.
SYMB               : CHARACTER  AT 1352(13),   DECLARED ON LINE 465 AND REFERENCED 23 TIMES.
SYMB_INDEX         : LABEL      AT 11460(14),  DECLARED ON LINE 1456 AND REFERENCED 4 TIMES.
SYNTHESIZE         : LABEL      AT 26656(14),  DECLARED ON LINE 2732 AND REFERENCED 1 TIMES.
  PARAMETER  1     : FIXED      AT 1664(8),    DECLARED ON LINE 2734 AND REFERENCED 1 TIMES.
TABLE_LOC          : FIXED      AT 1172(10),   DECLARED ON LINE 463 AND REFERENCED 52 TIMES.
TBASIC             : FIXED      AT 1156(10),   DECLARED ON LINE 459 AND REFERENCED 10 TIMES.
TEMP               : FIXED      AT 432(9),     DECLARED ON LINE 539 AND REFERENCED 10 TIMES.
TEMP_CHAR          : CHARACTER  AT 1992(13),   DECLARED ON LINE 549 AND REFERENCED 10 TIMES.
TEXT               : CHARACTER  AT 920(13),    DECLARED ON LINE 386 AND REFERENCED 15 TIMES.
TEXT_LIMIT         : FIXED      AT 256(10),    DECLARED ON LINE 387 AND REFERENCED 12 TIMES.
TGOTO              : FIXED      AT 3256(10),   DECLARED ON LINE 500 AND REFERENCED 0 TIMES.
TOKEN              : FIXED      AT 3548(11),   DECLARED ON LINE 351 AND REFERENCED 14 TIMES.
TRIPLE_INDEX       : BIT(8)     AT 3380(11),   DECLARED ON LINE 332 AND REFERENCED 2 TIMES.
TX                 : BIT(8)     AT 3820(11),   DECLARED ON LINE 371 AND REFERENCED 3 TIMES.
UMIN               : FIXED      AT 3240(10),   DECLARED ON LINE 496 AND REFERENCED 2 TIMES.
V                  : CHARACTER  AT 24(13),     DECLARED ON LINE 126 AND REFERENCED 9 TIMES.
V_INDEX            : BIT(8)     AT 1340(11),   DECLARED ON LINE 148 AND REFERENCED 4 TIMES.
VAR                : CHARACTER  AT 944(13),    DECLARED ON LINE 444 AND REFERENCED 20 TIMES.
VARIBLS            : CHARACTER  AT 1828(13),   DECLARED ON LINE 527 AND REFERENCED 10 TIMES.
VARNUM             : FIXED      AT 360(9),     DECLARED ON LINE 536 AND REFERENCED 22 TIMES.
WRITE_REGS         : LABEL      AT 18376(14),  DECLARED ON LINE 2015 AND REFERENCED 4 TIMES.
X1                 : CHARACTER  AT 932(13),    DECLARED ON LINE 429 AND REFERENCED 3 TIMES.
X4                 : CHARACTER  AT 936(13),    DECLARED ON LINE 429 AND REFERENCED 1 TIMES.
X70                : CHARACTER  AT 908(13),    DECLARED ON LINE 357 AND REFERENCED 3 TIMES.
ZQ                 : FIXED      AT 3244(10),   DECLARED ON LINE 497 AND REFERENCED 5 TIMES.
```

```
MACRO DEFINITIONS:

NT                      LITERALLY: 51
NSY                     LITERALLY: 108
TRUE                    LITERALLY: 1
FALSE                   LITERALLY: 0
LABELS                  LITERALLY: 25
DX_SIZE                 LITERALLY: 500
FOREVER                 LITERALLY: WHILE 1
SYMBOLS                 LITERALLY: 100
MAXQUADS                LITERALLY: 220
STACKSIZE               LITERALLY: 75
CONSTANTS               LITERALLY: 50
DISKWORDS               LITERALLY: 900
NOLTRIPLES              LITERALLY: 228
EJECT_PAGE              LITERALLY: OUTPUT(1) = PAGE
DOUBLE_SPACE            LITERALLY: OUTPUT(1) = DOUBLE


IDCOMPARES          = 343105
SYMBOL TABLE SIZE = 311
MACRO DEFINITIONS = 15
STACKING DECISIONS= 57026
SCAN                = 16603
EMITRR              = 697
EMITRX              = 8520
FORCEACCUMULATOR    = 3475
ARITHEMIT           = 541
GENSTORE            = 809
FIXBFW              = 467
FIXDATAWORD         = 12
FIXCHW              = 844
GETDATA             = 6
GETCODE             = 4
FINDADDRESS         = 1033
SHORTCFIX           = 838
LONGCFIX            = 6
SHORTDFIX           = 9
LONGDFIX            = 3
FREE STRING AREA    = 92014


REGISTER VALUES (RELATIVE TO R11):
R4 = 0
R5 = 0
R6 = 0
R7 = 0
R8 = 15852
R9 = 11640
R10 = 4332
R11 = 0
R12 = 0
R13 = 18168

 INSTRUCTION FREQUENCIES:

BALR    92
BCTR    3
BCR     133
LPR     1
LTR     14
LCR     15
NR      4
```

```
UR      8
XR      16
LR      111
CR      3
AR      30
SR      257
DR      2
ALR     4
SLR     4
STH     3
LA      737
STC     76
IC      82
EX      18
BAL     636
BC      733
LH      6
ST      1605
N       25
O       1
X       5
L       2802
C       209
A       209
S       148
M       6
D       24
AL      2
SRL     31
SLL     449
SRA     16
SRDA    25
STM     88
TM      7
DI      1
LM      89

TOTAL TIME IN COMPILER    0:6:26.58.
SET UP TIME               0:0:5.83.
ACTUAL COMPILATION TIME   0:5:52.10.
POST-COMPILATION TIME     0:0:28.65.
COMPILATION RATE: 571 CARDS PER MINUTE.
```

APPENDIX 7.2 PLM REPRESENTATION OF TEST PROGRAMS

```
 1 | FIBS:PROCEDURE;
 2 |      DECLARE (FIB,FIB1,FIB2,I,N) BYTE;
 3 |      FIB1 = 0;
 4 |      FIB2 = 1;
 5 |      N = 100;
 6 |      DO I = 1 TO N;
 7 |          FIB = FIB1;
 8 |          FIB1 = FIB2;
 9 |          FIB2 = FIB + FIB1;
10 |          END;
11 |  END;
12 |EOF EOF EOF
```

(a) FIBONACCI SERIES

```
 1 |GRT_ELM:PROCEDURE;
 2 |      DECLARE (I,N,GRTST_ELMNT,INDEX) BYTE;
 3 |      DECLARE ARRAY(10) BYTE;
 4 |      N=10;
 5 |      GRTST_ELMNT=ARRAY(1);
 6 |      INDEX=1;
 7 |      DO I=2 TO N;
 8 |          IF GRTST_ELMNT > ARRAY(I) THEN GO TO LAB1;
 9 |              GRTST_ELMNT=ARRAY(I);
10 |              INDEX=I;
11 |       LAB1:END;
12 |      END;
13 |EOF
```

(b) GREATEST ELEMENT

```
 1 | PRIME:PROCEDURE;
 2 |      DECLARE (I,J,K) BYTE;
 3 |      DECLARE A(256) BYTE;
 4 |      DO I=1 TO 256;
 5 |          A(I)=1;
 6 |          END;
 7 |      DO I=2 TO 256;
 8 |          IF A(I)=0 THEN GO TO LAB1;
 9 |              K=2*I;
10 |              DO J=K TO 256 BY I;
11 |                  A(J)=0;
12 |                  END;
13 |          LAB1:END;
14 |      END;
15 |EOF
```

(c) PRIME IDENTIFIER

```
 1 |SORT:PROCEDURE;
 2 |      DECLARE ARRAY(10) BYTE;
 3 |      DECLARE (I,J,K,TEMP,SWITCH) BYTE;
 4 |      SWITCH = 0;
 5 |      K=0;
 6 | LAB2:K = K + 1;
 7 |      DO I=K TO 9;
 8 |          J=I+1;
 9 |          IF ARRAY(I) < ARRAY(J) THEN GO TO LAB1;
10 |              SWITCH=1;
11 |              TEMP=ARRAY(I);
12 |              ARRAY(I)=ARRAY(J);
13 |              ARRAY(J)=TEMP;
14 |          LAB1:END;
15 |          IF SWITCH = 1 THEN GO TO LAB2;
16 |          END SORT;
17 |EOF
```

(d) BUBBLE SORT

```
 1 | FILTER:PROCEDURE;
 2 |      DECLARE(AT,EXAT,I,J,N,C)BYTE;
 3 |      DECLARE A(20) BYTE;
 4 |      DECLARE B(20) BYTE;
 5 |      B(1)=0;
 6 |      DO I=1 TO N;
 7 |          J=I+1;
 8 |          A(I)=A(I)*AT;
 9 |          B(J)=B(I)*EXAT;
10 |          C=A(I)+B(J);
11 |          B(J)=C;
12 |          END;
13 |      END;
14 |EOF
```

(e) DIGITAL FILTER

APPENDIX 7.3-1 QUAD AND R QUAD REPRESENTATION OF TEST PROGRAMS

QUADS GENERATED

OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT/ LABEL	****	RCD_NR	LOC_QUAD
LAB	FIBS	0	0		0	1
ASGN	0	0	FIB1		0	5
ASGN	1	0	FIB2		0	9
ASGN	100	0	N		0	13
ASGN	1	0	I		0	17
LAB	.L6	0	0		0	21
GT	I	N	.T7		0	25
BT	.T7	0	.L8		0	29
ASGN	FIB1	0	FIB		0	33
ASGN	FIB2	0	FIB1		0	37
ADD	FIB	FIB1	.T11		0	41
ASGN	.T11	0	FIB2		0	45
ADD	I	1	.T13		0	49
ASGN	.T13	0	I		0	53
BR	0	0	.L6		0	57
LAB	.L8	0	0		0	61

R QUADS

OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT LABEL
LAB	FIBS	0	0
RD	0	0	R6
RD	1	0	R5
RD	100	0	R4
WT	R4	0	N
WT	R5	0	I
WT	R5	0	FIB2
WT	R6	0	FIB1
LAB	.L6	0	0
RD	I	0	R6
RD	N	0	R5
GT	R6	R5	R1
BT	R1	0	.L8
RD	FIB1	0	R4
RD	FIB2	0	R3
ADD	R4	R3	R1
RD	1	0	R2
ADD	R6	R2	R5
WT	R1	0	FIB2
WT	R3	0	FIB1
WT	R4	0	FIB
WT	R5	0	I
BR	0	0	.L6
LAB	.L8	0	0

(a) FIBONACCI SERIES

QUADS GENERATED

OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT/ LABEL	****	RCD_NR	LOC_QUAD
LAB	GRT_ELM	0	0		0	1
ASGN	10	0	N		0	5
SUBS	ARRAY	1	.T3		0	9
ASGN	.T3	0	GRTST_ELMNT		0	13
ASGN	1	0	INDEX		0	17
ASGN	2	0	I		0	21
LAB	.L7	0	0		0	25
GT	I	N	.T8		0	29
BT	.T8	0	.L9		0	33
SUBS	ARRAY	I	.T10		0	37
GT	GRTST_ELMNT	.T10	.T11		0	41
BT	.T11	0	LAB1		0	45
SUBS	ARRAY	I	.T13		0	49
ASGN	.T13	0	GRTST_ELMNT		0	53
ASGN	I	0	INDEX		0	57
LAB	LAB1	0	0		0	61
ADD	I	1	.T17		0	65
ASGN	.T17	0	I		0	69
BR	0	0	.L7		0	73
LAB	.L9	0	0		0	77

R QUADS

OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT LABEL
LAB	GRT_ELM	0	0
RD	10	0	R6
RDAD	ARRAY	0	R5
RD	1	0	R4
ASL	R4	0	R1
ADD	R5	R1	R1
RDVR	R1	0	R3
RD	2	0	GRTST_ELMNT
WT	R1	0	I
WT	R3	0	INDEX
WT	R4	0	N
WT	R6	0	0
LAB	.L7	0	R6
RD	I	0	R5
RD	N	0	R1
GT	R6	R5	.L9
BT	R1	0	R4
RDAD	ARRAY	0	R1
ASL	R6	R1	R1
ADD	R4	0	R1
RDVR	R1	0	R3
RD	GRTST_ELMNT	0	R2
GT	R3	R1	LAB1
BT	R2	0	R1
ASL	R6	R1	R1
ADD	R4	0	R1
RDVR	R1	0	GRTST_ELMNT
WT	R1	0	INDEX
WT	R6	0	0
LAB	LAB1	0	R6
RD	I	0	R5
RD	I	R5	R1
ADD	R6	0	I
WT	R1	0	.L7
BR	0	0	0
LAB	.L9	0	

(b) GREATEST ELEMENT

APPENDIX 7.3-2

QUADS GENERATED

OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT/ LABEL	****	NCD_NR	LOC_QUAD
LAB	PRIME	0	0		0	1
ASGN	1	0	1		0	5
LAB	J	0	0		0	9
GT	I	256	.T4		0	13
BT	.T4	0	.L5		0	17
SUBL	A	1	.T6		0	21
ASGN	1	0	.T6		0	25
ADD	I	1	.T8		0	29
ASGN	.T8	0	I		0	33
HR	0	0	.L3		0	37
LAB	.L5	0	0		0	41
ASGN	2	0	I		0	45
LAB	J	0	0		0	49
GT	I	256	.T14		0	53
BT	.T14	0	.L15		0	57
SUBS	A	1	.T16		0	61
EQ	.T16	0	.T17		0	65
BT	.T17	0	LAB1		0	69
MUL	2	1	.T19		0	73
ASGN	.T19	0	K		0	77
ASGN	K	0	J		0	81
LAB	J	0	0		0	85
GT	I	256	.T23		0	89
BT	.T23	0	.L24		0	93
SUBL	A	1	.T25		0	97
ASGN	0	0	.T25		0	101
ADD	J	1	.T27		0	105
ASGN	.T27	0	J		0	109
HR	0	0	.L22		0	113
LAB	.L24	0	0		0	117
LAB	LAB1	0	0		0	121
ADD	J	1	.T32		0	125
ASGN	.T32	0	J		0	129
[illegible]	0	0	.L13		0	133
L	.L15	0	0		0	137

(C) PRIME IDENTIFIER

R QUADS

OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT LABEL
	PRIME	0	0
LAB	1	0	R6
RD	wT	R6	I
WT	LAB	.L3	0
LAB	RD	I	R6
RD	RD	256	R5
RD	GT	R6	R1
GT	BT	R1	.L5
BT	RDAD	A	R4
RDAD	ASL	R6	R1
ASL	ADD	R4	R1
ADD	RD	1	R3
RD	WTAD	R3	R1
WTAD	ADD	R6	R3
ADD	WT	R1	I
WT	HR	0	.L3
HR	LAB	.L5	0
LAB	RD	2	R6
RD	WT	R6	I
WT	LAB	.L13	0
LAB	RD	I	R6
RD	RD	256	R5
RD	GT	R6	R1
GT	BT	R1	.L15
BT	RDAD	A	R4
RDAD	ASL	R6	R1
ASL	ADD	R4	R1
ADD	RDVR	R1	R1
RDVR	RD	0	R3
RD	EW	R1	R2
EW	BT	R2	LAB1
BT	RD	2	R2
RD	MUL	R2	R1
MUL	RD	J	R6
RD	RD	256	R5
RD	GT	R6	R1
GT	BT	R1	.L24
BT	RDAD	A	R4
RDAD	ASL	R6	R1
ASL	ADD	R4	R1
ADD	RD	0	R3
RD	WTAD	R3	R1
WTAD	RD	I	R2
RD	ADD	R6	R1
ADD	WT	R1	J
WT	HR	0	.L22
HR	LAB	.L24	0
LAB	LAB1	J	0
RD	RD	I	R6
RD	ADD	R6	R5
ADD	WT	R1	J
WT	HR	0	.L13
HR	LAB	.L15	0

APPENDIX 7.3-3

QUADS GENERATED

OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT/ LABEL ****	RCD_NR	LOC_QUAD
LAB	SORT	0	0	0	1
ASGN	0	0	SWITCH	0	5
ASGN	0	0	K	0	9
LAB	LAB2	0	0	0	13
ADD	K	1	.T5	0	17
ASGN	.T5	0	K	0	21
ASGN	K	0	I	0	25
LAB	.L8	0	0	0	29
GT	I	9	.T9	0	33
BT	.T9	0	.L10	0	37
ADD	I	1	.T11	0	41
ASGN	.T11	0	J	0	45
SUBS	ARRAY	I	.T13	0	49
SUBS	ARRAY	J	.T14	0	53
LT	.T13	.T14	.T15	0	57
BT	.T15	0	LAB1	0	61
ASGN	I	0	SWITCH	0	65
SUBS	ARRAY	I	.T18	0	69
ASGN	.T18	0	TEMP	0	73
SUBL	ARRAY	I	.T20	0	77
SUBS	ARRAY	J	.T21	0	81
ASGN	.T21	0	.T20	0	85
SUBL	ARRAY	J	.T23	0	89
ASGN	TEMP	0	.T23	0	93
LAB	LAB1	0	0	0	97
ADD	I	1	.T26	0	101
ASGN	.T26	0	I	0	105
BR	0	0	.L8	0	109
LAB	.L10	0	0	0	113
EQ	SWITCH	1	.T30	0	117
BT	.T30	0	LAB2	0	121

(a) BUBBLE SORT

R QUADS

OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT LABEL
LAB	SORT	0	0
RD	0	0	R6
WT	R6	0	K
WT	R6	0	SWITCH
LAB	LAB2	0	0
RD	K	0	R6
RD	1	0	R5
ADD	R6	R5	R1
WT	R1	0	I
WT	R1	0	K
LAB	.L8	0	0
RD	I	0	R6
RD	9	0	R5
GT	R6	R5	R1
BT	R1	0	.L10
RD	1	0	R4
ADD	R6	R4	R1
RDAD	ARRAY	0	R3
ASL	R6	0	R2
ADD	R3	R2	R2
RDVR	R2	0	R2
ASL	R1	0	R4
ADD	R3	R4	R4
RDVR	R4	0	R4
LT	R2	R4	R3
WT	R1	0	J
BT	R3	0	LAB1
RD	1	0	R4
RDAD	ARRAY	0	R3
ASL	R6	0	R2
ADD	R3	R2	R2
RDVR	R2	0	R2
ASL	R6	0	R1
ADD	R3	R1	R1
RD	J	0	R5
ASL	R5	0	R6
ADD	R3	R6	R6
RDVR	R6	0	R6
WTAD	R6	0	R1
ASL	R5	0	R1
ADD	R3	R1	R1
WTAD	R2	0	R1
WT	R2	0	TEMP
WT	R4	0	SWITCH
LAB	LAB1	0	0
RD	I	0	R6
RD	1	0	R5
ADD	R6	R5	R1
WT	R1	0	I
BR	0	0	.L8
LAB	.L10	0	0
RD	SWITCH	0	R6
RD	1	0	R5
EQ	R6	R5	R1
BT	R1	0	LAB2

APPENDIX 7.3-4

QUADS GENERATED

OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT/ LABEL	****	RCD_NR	LOC_QUAD
LAR	FILTER	0	0		0	1
SUBL	H	1	.T2		0	5
ASGN	0	0	.T2		0	9
ASGN	1	0	I		0	13
LAR	.L5	0	0		0	17
GT	I	N	.T6		0	21
BT	.T6	0	.L7		0	25
ADD	I	1	I		0	29
ASGN	.T8	0	J		0	33
SUBL	A	I	.T10		0	37
SUBS	A	I	.T11		0	41
MUL	.T11	AT	.T12		0	45
ASGN	.T12	0	.T10		0	49
SUBL	H	J	.T14		0	53
SUBS	H	I	.T15		0	57
MUL	.T15	EXAT	.T16		0	61
ASGN	.T16	0	.T14		0	65
SUBS	A	I	.T18		0	69
SUBS	H	J	.T19		0	73
ADD	.T18	.T19	.T20		0	77
ASGN	.T20	0	C		0	81
SUBL	H	J	.T22		0	85
ASGN	C	0	.T22		0	89
ADD	I	1	.T24		0	93
ASGN	.T24	0	I		0	97
BR	0	0	.L5		0	101
LAR	.L7	0	0		0	105

(e) DIGITAL FILTER

R QUADS

OPERATOR	OPERAND1/ CONDITION	OPERAND2	RESULT LABEL
LAR	FILTER	0	0
RDAD	B	0	R6
RD	1	0	R5
ASL	R5	0	R1
ADD	R6	R1	R1
RD	0	0	R4
WTAD	R4	0	R1
WT	R5	0	1
LAR	.L5	0	0
RD	I	0	R6
RD	N	0	R5
GT	R6	R5	R1
BT	R1	0	.L7
RD	1	0	R4
ADD	R6	R4	R1
RDAD	A	0	R3
ASL	R6	0	R2
ADD	R3	R2	R2
ASL	R6	0	R4
ADD	R3	R4	R4
RDVR	R4	0	R4
RD	AT	0	R3
MUL	R4	R3	R5
WTAD	R5	0	R2
RDAD	B	0	R5
ASL	R1	0	R2
ADD	R5	R2	R2
ASL	R6	0	R4
ADD	R5	R4	R4
RDVR	R4	0	R4
RD	EXAT	0	R3
MUL	R4	R3	R5
WTAD	R5	0	R2
RDAD	A	0	R5
ASL	R6	0	R2
ADD	R5	R2	R2
RDVR	R2	0	R2
RDAD	B	0	R4
ASL	R1	0	R3
ADD	R4	R3	R3
RDVR	R3	0	R3
ADD	R2	R3	R4
RDAD	B	0	R3
ASL	R1	0	R2
ADD	R3	R2	R2
WTAD	R4	0	R2
RD	1	0	R2
ADD	R6	R2	R3
WT	R1	0	I
WT	R3	0	C
WT	R4	0	
BR	0	0	.L5
LAR	.L7	0	0

APPENDIX 7.4-1 DEC PDP 11/45 MICROCODE REPRESENTATION OF TEST PROGRAMS

(a) FIBONACCI SERIES

GENERATED MICRO WORDS

LOC	CLK	CIR	WF	CH	CD	CBA	BUS	DAD	SPS	ALU	SBC	SBM	SDM	SBA	UBF	SRX	RIF	UPF
000	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00000001
001	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00000010
002	111	0	11	0	1	0	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000011
003	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00000100
004	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00000101
005	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00000110
006	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000111
007	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001000
010	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00001001
011	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	0000101C
012	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00001011
013	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001100
014	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00001101
015	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00001110
016	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00001111
017	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00010000
020	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010001
021	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0100	00010010
022	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00010011
023	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00010100
024	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010101
025	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0101	00010110
026	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00010111
027	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011000
030	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00011001
031	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0101	00011010
032	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00011011
033	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011100
034	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00011101
035	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0110	00011110
036	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00011111
037	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00100000
040	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00100001
041	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00100010
[illegible]																		
044	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00100101
045	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00100110
046	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00100111
047	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00101000
050	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	00101001
051	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0110	00101010
052	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00101011
053	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	00101101
055	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00101110
056	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00110000
057	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01010010
060	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00110001
061	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00110010
062	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00110011
063	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00110100
064	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00110101
065	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00110110
066	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00110111
067	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	00111000
070	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0011	00111001
071	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0100	00111010
072	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00111011
073	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00111100
074	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00111101
075	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00111110
076	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0010	00111111
077	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0010	01000000
100	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0110	01000001
101	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0101	01000010
102	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01000011
103	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01000100
104	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	01000101
105	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	01000110
106	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01000111
107	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01001000
110	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	01001001
111	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0011	01001010
112	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01001011
113	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01001100
114	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	01001101
115	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0100	01001110
116	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01001111
117	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01010000
120	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	01010001
121	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0101	00011110
122	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	01010011
123	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	01010100

APPENDIX 7.4-2

(b) GREATEST ELEMENT - 1

GENERATED MICRO WORDS

LOC	CLK	CIR	WR	CR	CD	CBA	BUS	DAD	SPS	ALU	SBC	SHM	SDM	SBA	UBF	SRX	RIF	UPF
000	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	000000001
001	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	000000010
002	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	000000011
003	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	000000100
004	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	000000101
005	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	000000110
006	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	000000111
007	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	000001000
010	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	000001001
011	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	000001010
012	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	000001011
013	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	000001100
014	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0100	000001101
015	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	000001110
016	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	000001111
017	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0101	000010000
020	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	000010001
021	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0001	000010010
022	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0001	000010011
023	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	000010100
024	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	000010101
025	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	000010110
026	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	000010111
027	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	000011000
030	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	000011001
031	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	000011010
032	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	000011011
033	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	000011100
034	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	000011101
035	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	000011110
036	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0011	000011111
037	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	000100000
040	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	000100001
041	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	000100010
042	111	0	00	0	1	1	001	0000	000	00000	0000	0000	00	0	00000	0001	0100	000100011
043	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	000100100
044	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	000100101
045	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	000100110
046	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0110	000100111
047	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	000101000
050	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	000101001
051	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	000101010
052	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	000101011
053	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	000101100
054	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	000101101
055	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	000101110
056	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	000101111
057	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	000110000
060	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	000110001
061	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	000110010
062	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0110	000110011
063	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	000110100
064	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	000110101
065	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	000110110
066	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	000111000
067	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	011011100
070	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	000111001
071	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	001110010
072	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0110	001110011
073	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	001111100
074	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	001111101
075	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0100	001111110
076	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	001111111
077	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0001	010000000
100	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0001	010000001
101	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	010000010
102	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	010000011
103	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	010000100
104	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	010000101
105	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	010000110
106	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0011	010000111
107	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0010	010001000
110	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	010001001
111	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	010001010
112	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	010001100

APPENDIX 7.4-3

(b) GREATEST ELEMENT - 2

113	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01011011
114	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0110	01001101
115	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	01001110
116	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	01001111
117	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0100	01010000
120	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	01010001
121	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0001	01010010
122	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0001	01010011
123	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01010100
124	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01010101
125	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	01010110
126	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	01010111
127	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01011000
130	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01011001
131	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	01011010
132	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0110	01011011
133	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	01011100
134	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	01011101
135	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01011110
136	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01011111
137	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	01100000
140	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	01100001
141	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01100010
142	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01100011
143	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	01100100
144	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	01100101
145	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	01100110
146	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0110	01100111
147	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	01101000
150	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01101001
151	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01101010
152	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	01101011
153	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	00100111
154	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	01101101
155	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	01101110

APPENDIX 7.4-4

(c) PRIME IDENTIFIER 1

GENERATED MICRO WORDS

LOC	CLK	CIR	WR	CB	CD	CRA	RUS	DAD	SPS	ALU	SBC	SBM	SDM	SBA	UBF	SRX	RIF	UPF
000	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00000001
001	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00000010
002	111	0	11	0	0	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000011
003	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00000100
004	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00000101
005	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00000110
006	111	0	11	0	1	0	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000111
007	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001000
010	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00001001
011	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0110	00001010
012	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00001011
013	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00001100
014	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00001101
015	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001110
016	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00001111
017	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00010000
020	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00010001
021	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00010010
022	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010011
023	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00010100
024	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	00010101
025	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0110	00010110
026	010	0	11	0	1	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00010111
027	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	00011001
031	010	0	00	0	1	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00011010
032	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00011100
033	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00110000
034	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00011101
035	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00011110
036	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0110	00011111
037	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00100000
040	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	00100001
041	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0100	00100010
042	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00100011
[illegible]																		
044	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00100101
045	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00100110
046	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	00100111
047	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	0011	00101000
050	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	00101001
051	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0011	00101010
052	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0110	00101011
053	010	0	11	0	0	1	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00101100
054	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00101101
055	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00101110
056	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00101111
057	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	00001010
060	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00110001
061	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00110010
062	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00110011
063	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00110100
064	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00110101
065	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00110110
066	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00110111
067	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00111000
070	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00111001
071	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0110	00111010
072	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00111011
073	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00111100
074	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00111101
075	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00111110
076	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00111111
077	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	01000000
100	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01000001
101	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	01000010
102	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	01000011
103	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	01000100
104	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	01000101
105	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0110	01000110
106	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	01000111
107	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	01001001
111	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01001010
112	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01001100
113	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	10110101
114	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	01001101

APPENDIX 7.4-5

(c) PRIME IDENTIFIER - 2

APPENDIX 7.4-6

(c) PRIME IDENTIFIER = 3

237	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	10100000
240	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	10100001
241	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	01111001
242	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	10100011
243	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	10100100
244	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	10100101
245	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	10100110
246	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	10100111
247	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	10101000
250	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	10101001
251	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	10101010
252	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	10101011
253	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	10101100
254	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	10101101
255	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	10101110
256	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	10101111
257	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0110	10110000
260	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	10110001
261	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	10110010
262	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	10110011
263	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	10110100
264	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	00111010
265	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	10110110
266	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	10110111

APPENDIX 7.4-7

(d) BUBBLE SORT - 1

GENERATED MICRO WORDS

LOC	CLK	CIR	WR	CB	CD	CRA	RUS	DAD	SPS	ALU	SRC	SHM	SDM	SBA	UBF	SRX	RIF	UPF
000	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00000001
001	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00000010
002	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000011
003	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00000100
004	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00000101
005	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00000110
006	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000111
007	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001000
010	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00001001
011	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0110	00001010
012	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00001011
013	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001100
014	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00001101
015	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0110	00001110
016	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00001111
017	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00010000
020	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00010001
021	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00010010
022	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010011
023	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00010100
024	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00010101
025	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00010110
026	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010111
027	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00011000
030	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	00011001
031	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0110	00011010
032	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00011011
033	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00011100
034	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011101
035	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00011110
036	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	00011111
037	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00100000
040	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00100001
041	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00100010
042	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	00100011
043	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00100100
044	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00100101
045	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00100110
046	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00100111
047	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00101000
050	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00101001
051	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00101010
052	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00101011
053	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00101100
054	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00101101
055	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	00101110
056	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0110	00101111
057	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00110000
060	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	00110001
061	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00110010
062	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00110100
063	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	10010101
064	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00110101
065	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00110110
066	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00110111
067	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00111000
070	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0100	00111001
071	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0110	00111010
072	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00111011
073	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00111100
074	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	00111101
075	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0110	00111110
076	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0010	00111111
077	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0010	01000000
100	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0011	01000001
101	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0010	01000010
102	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0010	01000011
103	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0010	01000100
104	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0001	01000101
105	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0100	01000110
106	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0100	01000111
107	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0011	01001000
110	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0100	01001001
111	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0100	01001010
112	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	01001011

APPENDIX 7.4-8

(d) BUBBLE SORT 2

APPENDIX 7.4-9

(e) DIGITAL FILTER - 1

GENERATED MICRO WORDS

LDC	CLK	CTR	WP	CA	CB	CRA	BUS	UAD	SPS	ALU	SBC	SBM	SDM	SBA	UBF	SRX	RIF	UPF
000	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00000001
001	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00000010
002	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000011
003	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00000100
004	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00000101
005	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00000110
006	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00000111
007	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00001000
010	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0101	00001001
011	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00001010
012	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0001	00001011
013	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0110	00001100
014	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00001101
015	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00001110
016	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00001111
017	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010000
020	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00010001
021	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	0100	00010010
022	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0001	00010011
023	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00010100
024	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00010101
025	010	0	00	0	0	1	000	0000	000	00000	0000	0000	00	1	00000	0001	1000	00010110
026	101	0	00	0	1	0	101	0000	000	00000	0000	0000	00	0	00000	0001	0101	00010111
027	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0000	00011000
030	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0000	00011001
031	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00011010
032	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011011
033	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00011100
034	010	0	11	0	0	1	000	0000	000	00000	0000	0000	01	0	00000	0001	0110	00011101
035	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00011110
036	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00011111
037	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00100000
040	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0101	00100001
041	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0101	00100010
042	100	0	00	0	1	0	000	1000	011	00110	0000	0000	00	0	00000	0001	0110	00100011
043	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00100100
044	100	0	00	0	1	0	000	0000	001	10011	0001	1111	00	0	01010	0000	0000	00100101
045	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00100110
046	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	00101000
047	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	10100000
050	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00101001
051	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00101010
052	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	00101011
053	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00101100
054	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0100	00101101
055	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0110	00101110
056	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0001	00101111
057	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00110000
060	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	00110001
061	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0110	00110010
062	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0010	00110011
063	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0010	00110100
064	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0011	00110101
065	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0010	00110110
066	100	0	00	0	1	0	000	0000	000	01100	0000	0000	00	0	00000	0001	0110	00110111
067	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0100	00111000
070	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0100	00111001
071	100	0	00	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0011	00111010
072	010	0	11	0	0	0	000	0000	000	00000	0000	0000	10	0	00000	0001	0100	00111011
073	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	0100	00111100
074	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0100	00111101
075	111	0	11	0	1	1	001	0000	000	01001	0010	1111	10	1	00000	0001	0000	00111110
076	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	1000	00111111
077	011	0	00	0	0	1	001	0000	000	00000	0000	0000	00	1	00000	0001	1000	01000000
100	010	0	11	0	0	0	000	0000	000	00000	0000	0000	01	0	00000	0001	0011	01000001
101	110	0	11	0	1	0	000	0000	000	11010	1111	1111	10	0	00000	0001	1000	01000010
102	110	0	11	0	1	0	000	0000	000	01100	0000	0000	10	0	00000	0001	1000	01000011
103	110	0	11	0	1	0	000	0000	000	01100	0000	0000	10	0	00000	0001	1000	01000100
104	110	0	11	0	1	0	000	0000	000	00011	0000	0000	10	0	00000	0001	0101	01000101
105	010	0	00	1	0	0	000	0000	000	00000	0000	0000	00	0	00000	0001	0100	01000110
106	110	0	11	0	1	0	000	0000	100	00000	0000	0000	10	0	10011	0001	0011	01000111
107	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01001000
110	100	0	11	0	1	0	000	0000	000	01001	0000	0000	00	0	00000	0001	0101	01001001
111	100	0	11	0	1	0	000	0000	001	00110	0001	1111	10	0	01010	0001	1000	01001011
112	010	0	00	0	0	0	000	0000	000	00000	0000	0000	00	0	00000	0000	0000	01000110

APPENDIX 7.4-10

(e) DIGITAL FILTER - 2

7.5 (a) R. QUAD PROCESSOR PROCEDURES

The R Quad Processor has 7 procedures. The calling sequence is shown below. Following this is a brief description of each procedure.

Procedure Name	Called By	Parameter	Return
ALLOC_REG_OP	ASGN QUAD Funct. QUADS SUBS QUAD SUBL QUAD BT/BF QUADS	Variable	Register Number
ALLOC_REG_RES	Funct. QUADS SUBS QUAD SUBL QUAD	Variable	Register Number
DEALLOCATE	ALLOC_REG_OP ALLOC_REG_RES		Register Number
DEALLOC_TEMP	ASGN QUAD Funct QUADS SUBS QUAD SUBL QUAD BT/BF QUADS	Variable Register Number	Register Number
FIND_OP	ALLOC_REG_OP	Variable	Register Number
FIND_RES	ALLOC_REG_RES ASGN_QUAD	Variable	Register Number
WRITE_REG	BR QUAD LAB QUAD BT/BF QUAD		

The procedure descriptions are provided below.

Alloc_Reg_OP: Procedure (Variable)

1. Reg=Find_OP (Variable)
   (Value of Reg is one of the following:
   a. register number which the variable is already assigned to.
   b. register number which the variable is to be assigned to.
   c. -1)

2. If Reg=-1, then Reg=Deallocate.
   (routine Deallocate returns a register number to be allocated to the variable, see explanation in Deallocate routine)

3. If RDFLAG is set to 'NO', then go to 8
   (RDFLAG is set by Find_OP routine to indicate if the variable is already assigned to the register)
   Else, go to 4.

4. Put the variable in the list of variables assigned to Reg and set its STATUS entry to 'ALLOCATED' and CHANGE entry to 'NO', set other entries accordingly, go to 5.

5. If the quad under process is a SUBS or SUBL quad, then go to 6.

   Else, go to 7.

6. Generate a RDAD register quad and go to 8.

   Quad sets Reg to address of the variable

7. Generate a RD register quad, and go to 8

   Reads value of the variable into Reg.

8. RETURN with value of Reg.

Alloc_Reg_Res: Procedure (Variable)

1. Reg=Find_Res(Variable)
   (Value of Reg is one of the following
   a. register number which the variable is already assigned to.

   b. register number which the variable is to be assigned to.

   c. -1)

2. If Reg= -1, then Reg=Deallocate.
   (routine Deallocate returns a register number to be allocated to the variable, see explanation on Deallocate routine)

3. If RDFLAG is set to 'NO' then go to 5
   (RDFLAG is set by Find_Res routine to indicate if the variable is already assigned to the register)

   Else, go to 4.

4. Put the variable in the list of variables assigned to Reg, set its STATUS entry to 'ALLOCATED' and CHANGE entry to "YES", and set other entries accordingly.

5. RETURN with value of Reg.

Deallocate: Procedure

(The routine will be called only in the case no register is available to be allocated to a variable.
The routine searches register table and selects a register on the following basis, notice that the given basis are in the order of importance attached to them in coding the routine.

1. Least number of temporary variables are assigned to the selected register.

2. Least number of variables are assigned to the selected register.

3. The longest time after the last reference to the selected register.

After selection of the register following steps are re-executed)

1. Search in the list of variables assigned to the register and generate a WT register quad for those variables whose CHANGE entry is set to 'YES'.
   (the generated register quad writes
   contents of Reg into memory location Var)

2. RETURN with value of register number.

Dealloc_Temp: Procedure (Variable, Register)

(If the given variable is a temporary variable, then it will be deleted from list of variables assigned to the register)

1. If the given variable is a temporary variable, then go to 2.

   Else, RETURN.

2. Delete the variable from list of variables assigned to the register.

3. If the deleted variable was the only variable assigned to the register, then go to 4.

   Else, RETURN.

4. Set STATUS entry of the register to 'FREE' and RETURN.

Find_OP: Procedure (Variable)
(returns a register number or -1, and sets RDFLAG)

1. Search list of variables assigned to all registers for the given variable.

2. If the variable is found, then go to 3.

   Else, go to 4.

3. Set RDFLAG to 'NO' and RETURN with value of register number which the variable is already assigned to.

4. Set READFLAG to 'YES'.

5. Search register table for a register with STATUS entry equal to 'FREE'.

6. If the search is successful, then RETURN with value of the register number.

   Else, RETURN with value of -1.

Find_Res: Procedure (Variable)
(returns a register number or -1 and sets READFLAG)

1. Set READFLAG to 'YES'.

1. Search list of variables assigned to all registers for the given variable.

2. If the variable is found, then go to 3.

   Else, go to 4.

3. If the variable is the only variable assigned to the register then set the CHANGE ENTRY OF THE VARIABLE TO'YES' and READFLAG to 'NO', and RETURN.

   Else, delete the variable from list of variables assigned to the register and go to 4.

4. Search register table for a register with STATUS entry equal to 'FREE'.

5. If the search is successful then RETURN with value of the register number.

   Else, RETURN with value of -1.

Write_Regs: Procedure

(the Procedure goes through list of variables assigned to all registers, and for those variables whose CHANGE entry is 'YES', generates a WT register quad, the generated register quads are
WT Reg Ø Var
which write contents of Reg in memory location associated to Var)

7.5 (b) R QUAD GENERATORS

The generators for the six R QUAD types are shown below.

1) Functional R QUADS

```
ADD  A  B  C
SUB  A  B  C
GT   A  B  C
LT   A  B  C
EQ   A  B  C
```

1. Reg 1 =Alloc_Reg_OP(A)
   (Reg 1 is register number which variable A is assigned to)

2. Reg 2=Alloc_Reg_OP(B)
   (Reg 2 is a register number which variable B is assigned tc)

3. Reg 3=Alloc_Reg_Res(C)
   (Reg 3 is a register number which variable C is assigned to)

4. Generate a register quad accordingly
   (the register quad will be one of the following:

```
ADD  Reg 1  Reg 2  Reg 3
SUB  Reg 1  Reg 2  Reg 3
GT   Reg 1  Reg 2  Reg 3
LT   Reg 1  Reg 2  Reg 3
EQ   Reg 1  Reg 2  Reg 3)
```

5. Call Dealloc_Temp(A, Reg 1)
   Call Dealloc_Temp(B, Reg 2)
   (if A or B or both are temporary variables, then they will be deleted from list variables associated to their registers)

2) Assign R QUAD

```
ASGN  A  Ø  B
```

1. Reg 1=Alloc_Reg_OP(A);
   (Reg 1 is register number which variable A is assigned to)

2. Reg 2=Find_Res(B)
   (Reg 2 is a register number which variable B or address of variable B is assigned to)

3. If ADDRESS entry of Reg 2 is'YES', then generate a WTAD register quad, and go to 4.
   (generated register quad is
   WTAD Reg 1 Ø Reg 2
   the register quad writes contents of Reg 1 into location given by Reg 2)

   Else, insert variable B into list of variables assigned to Reg 1, set CHANGE entry of variable B to 'YES', set other entries accordingly, go to 4.

4. Delete variable B from list of variables assigned to Reg 2.

5. Call Dealloc_Temp(A, Reg 1)
   (Dealloc_Temp is a routine to delete variable A from Reg 2, if A is a temporary variable)

3) Array R QUADS

SUBS A I C
SUBL A I C

1. Reg 1= Alloc_Reg_OP(A)
   (Reg 1 is a register number to which address of first word of array A is assigned)

2. Reg 2= Alloc_Reg_OP(I)
   (Reg 2 is a register number to which variable I is assigned)

3. Reg 3= Alloc_Reg_Res(C)
   (Reg 3 is a register number to which variable C is assigned)

4. Generate an ASL register quad.
   (generated register quad is
   ASL Reg 2 Ø Reg 3
   the register quad sets Reg 3 to contents of Reg 2 shifted to left by one position)

5. Generate an ADD register quad.
   (generated register quad is
   ADD Reg 1 Reg 3 Reg 3
   the register quad adds contents of registers Reg 1 and Reg 3 and sets Reg 3 to the result of addition, after the addition Reg 3 contains address of A(I))

6. If the quad under process is SUBS, then go to 7.

   Else, go to 8.
   (in this case it is a SUBL quad)

7. Generate a RDAD register quad, go to 9.
   (generated register quad is
   RDAD Reg 3 Ø Reg 3
   the register quad sets Reg 3 to contents of location given by Reg 3.)

8. Set ADDRS entry of Reg 3 to 'YES', go to 9.

9. Call Dealloc_Temp(I, Reg 2)
   (if I is a temporary variable, then it will be deleted from list of variables assigned to Reg 2)

4) Branch R QUADS

BR Ø Ø lab

1. Call Write_Regs
   (see explanation on routine Write_Regs)

2. Generate a BR register quad.
   (the generated register is
   BR Ø Ø lab)

5) Label R QUAD

LAB lab Ø Ø

1. Call Write_Regs.
   (see explanation on routine Write_Regs)

2. Delete all variables assigned to all registers.

3. Set STATUS entry of all registers to 'FREE'.

4. Generate a LAB register quad.
   (generated register quad is the following
   LAB lab Ø Ø)

6) BT/BF Quads

BT A Ø lab
BF A Ø lab

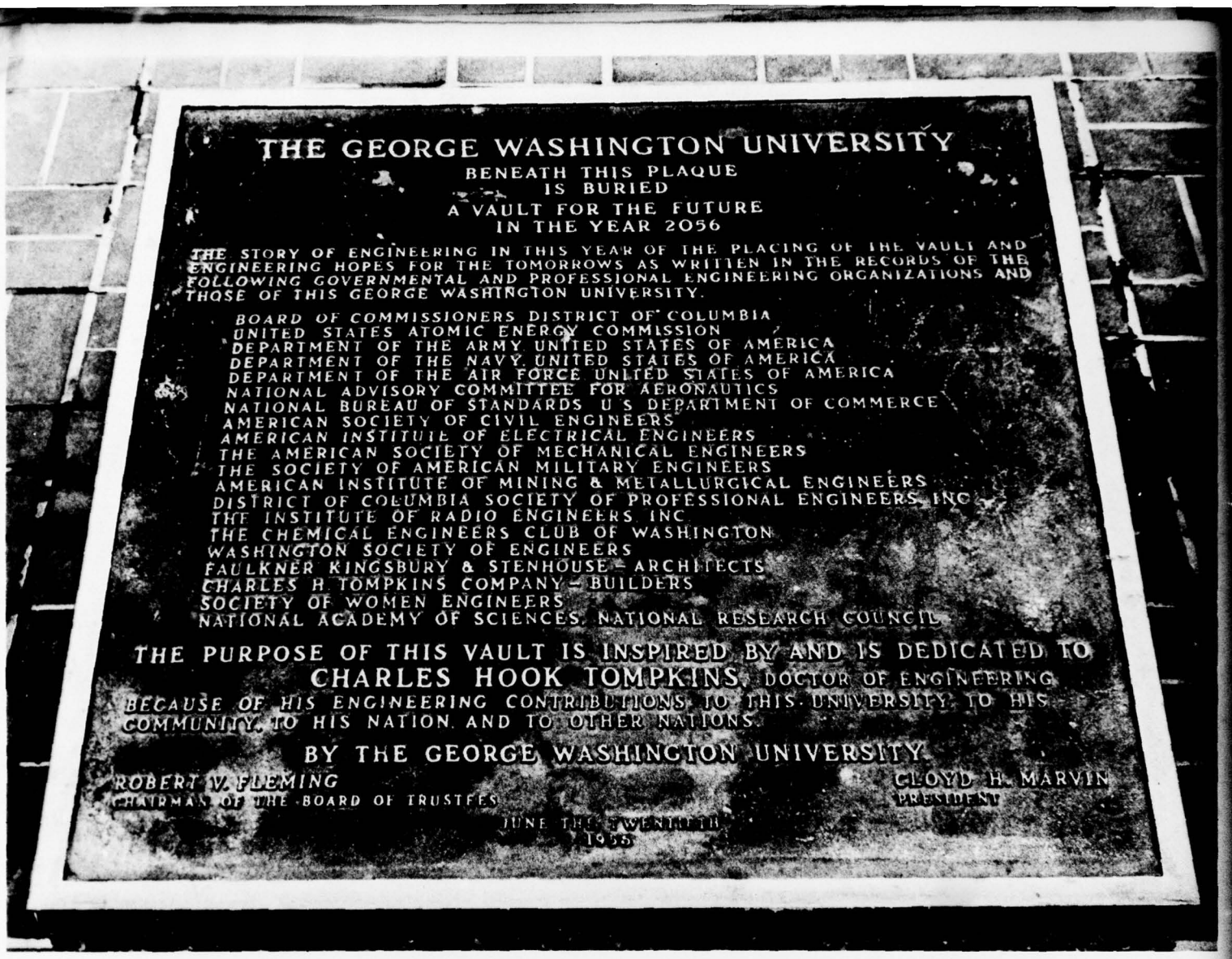
1. Call Write_Regs;
   (see explanation on routine Write_Regs)

2. Reg 1≤ Alloc_Reg_OP(A)
   (Reg 1 is a register number which variable A is assigned to.

3. Generate a register quad accordingly
   (the register quad will be one of the following

   BT Reg 1 Ø Lab
   BF Reg 2 Ø Lab)

4. Call Dealloc_Temp(A, Reg 2)
   (if A is a temporary variable then variable A will be deleted from list of variables assigned to Reg 2)



To cope with the expanding technology, our society must be assured of a continuing supply of rigorously trained and educated engineers. The School of Engineering and Applied Science is completely committed to this objective.