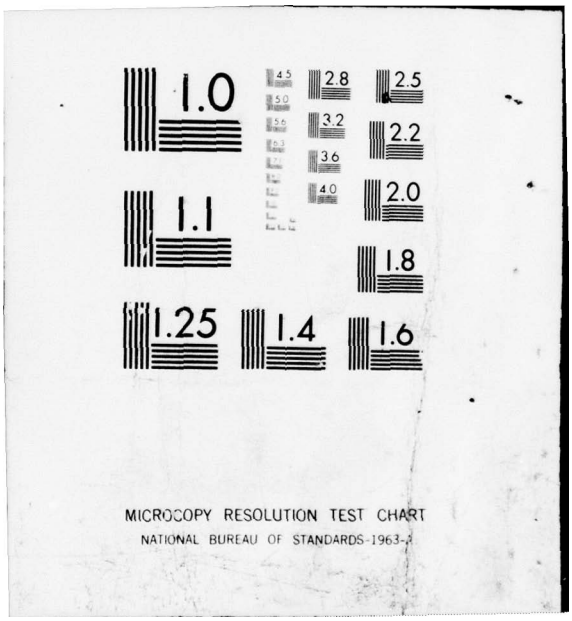
AD-A060 414 MICHIGAN UNIV ANN ARBOR SYSTEMS ENGINEERING LAB F/G 9/2
A CRAY-1 CROSS ASSEMBLER.(U)
SEP 78 W G AMES AFOSR-75-2812
UNCLASSIFIED SEL-120 AFOSR-TR-78-1406 NL

1 OF 1
AD
A060414

END
DATE FILMED
1-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

78-1406 LEVEL II

SEL Report No. 120

A CRAY-1 Cross Assembler

W. G. Ames

September 1, 1978

Sponsored Jointly by
Directorate of Mathematical and Information Sciences,
Air Force Office of Scientific Research, and
Air Force Flight Dynamics Laboratory,
Wright-Patterson Air Force Base,
under Grant 75-2812

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
SYSTEMS ENGINEERING LABORATORY
THE UNIVERSITY OF MICHIGAN, ANN ARBOR

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)
NOTICE OF TRANSMITTAL TO DDC
This technical report has been reviewed and is approved for public release IAW AFR 190-12 (7b).
Distribution is unlimited.
A. D. BLOSE
Technical Information Officer

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR-TR-78-1406
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*: A CRAY-1 CROSS ASSEMBLER - SEL REPORT # 120
5. TYPE OF REPORT & PERIOD COVERED: Interim rept.
6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): W.G. Ames
8. CONTRACT OR GRANT NUMBER(s): AFOSR-75-2812

9. PERFORMING ORGANIZATION NAME AND ADDRESS
University of Michigan
Dept. of Electrical & Computer Engineering
Ann Arbor, Michigan 48109

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F 2304/A3

11. CONTROLLING OFFICE NAME AND ADDRESS
Air Force Office of Scientific Research/NM
Bolling AFB, Washington, DC 20332

12. REPORT DATE: September 1, 1978
13. NUMBER OF PAGES: 35

14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: 38p

15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

SEL-120

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)*

Vector Processing
Parallel Processing

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

A CRAY-1 cross assembler is described. It is written in Fortran, and developed on an Amdahl 470/V6. The source code accepted is very close to that of the CRAY Research assembler, and the object code produced is compatible with the University of Michigan CRAY-1 simulator. The assembler and simulator form a powerful tool for developing assembly language programs.

DD FORM 1 JAN 73 1473

400 704

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

LEVEL II

(4)

A Cray-1 Cross Assembler

W. G. Ames

Systems Engineering Laboratory
University of Michigan
Ann Arbor, Michigan 48109

September 1, 1978

SEL Report #120

Sponsored Jointly by the Directorate of Mathematical and Information Sciences, Air Force Office of Scientific Research, and the Air Force Flight Dynamics Laboratory, under Grant 75-2812

Abstract

A CRAY-1 cross assembler is described. It is written in Fortran, and developed on an Amdahl 470/V6. The source code accepted is very close to that of the CRAY Research assembler, and the object code produced is compatible with the University of Michigan CRAY-1 simulator. The assembler and simulator form a powerful tool for developing assembly language programs.

Acknowledgements

I wish to acknowledge the help of the following people who assisted in the development of the assembler.

Professor D. A. Calahan, University of Michigan
Ed Sesek, University of Michigan
Dave Kennison, National Center for Atmospheric Research
CRAY-Research, Inc.

CONTENTS

SECTION I. INTRODUCTION

The CRAY-1 Cross Assembler has been prepared as an aid for developing CRAY-1 assembly language programs. It is designed to be used in conjunction with the CRAY-1 Simulator. Programs developed with the assembler/simulator should run with little or no modification on the CRAY-1.

The assembler will make available to the simulator the definitions of all symbols defined. This allows program control and examination and manipulation of registers and memory by referencing the same names used in the assembly language program.

Either relocatable or absolute object modules may be produced. Assembly language routines may identify multiple entry points, and reference external routines. Relocation and the resolving of external symbols is performed by the simulator.

The assembler will accept all instructions accepted by the CRAY-Research Assembler (CAL), including the special syntax forms. A subset of the pseudo ops are accepted.

SECTION II. PROGRAM STRUCTURE

The program to be assembled consists of one or more modules. Each module begins with an IDENT record and ends with and END record. Cards between the END record of a module and the IDENT record of the next module are treated as comments.

Statement Syntax

Every assembler statement consists of four fields: the label field, the result field, the operand field, and the comment field. Up to 90 columns may be used. All four fields may be omitted.

The label field is generally optional, but is required by certain pseudo ops. If present it must begin in column one or column two. It must meet the syntax requirements of symbols.

The result field must begin before column 35. If a label is present, the result field must begin at least one space after the label. If there is no label field, the result field must begin after column 2.

The operand field, if present, must begin at least one space beyond the result field and before column 35.

The comment field must begin at least one space after the operand field. If the operand field is not used, the comment field must start at least one space after the result field. If the operand field for a particular instruction is optional,the comment field must start at or beyond column 35.

Symbols

Symbols may be one of six types: relocatable parcel address, relocatable word address, absolute parcel address, absolute word address, value, or undefined.

Symbols are most commonly given values through their appearance in the label field of an instruction. In this case, the symbol will be a parcel address. The symbol will additionally be relocatable or absolute, depending on the type of assembly in progress. Some pseudo ops, such as BSS and BSSZ define symbols as word addresses. A symbol can be defined as type value by using

the = or SET pseudo op, with the operand field containing a value expression.

The first character of a symbol must be alphabetic, $, % or @. The remaining characters may be alphabetic, numeric, $, % or @. Symbols may have at most eight characters. If more than eight are present, the symbol will be truncated.

Constants

Constants will be interpreted in base 10 or base 8, depending on the last BASE pseudo op encountered. This may be overriden by preceeding the constant with a D' for base 10, or an O' for base 8.

Constants are treated as type value for the purpose of expression evaluation, with one exception: an octal constant whose last digit is A, B, C or D is treated as a parcel address.

Special Symbols

Only one special symbol is allowed. This is *, and represents the parcel address of the instruction in which it appears.

Expressions

Expressions are composed of one or more constants or symbols, joined by the operators +, -, * or /. Constants or symbols joined by * or / will be combined first, then constants or symbols joined by + or -. The type of expression resulting is rather complex, and so is presented in tabular form on the following page.

Type of Expression Resulting from Symbol Combination

+	RPA	RWA	APA	AWA	V
RPA	NA	NA	RPA	RWA	RPA
RWA	NA	NA	RPA	RWA	RWA
APA	RPA	RPA	APA	APA	APA
AWA	RWA	RWA	APA	AWA	AWA
V	RPA	RWA	APA	AWA	V

-	RPA	RWA	APA	AWA	V
RPA	V	V	RPA	RPA	RPA
RWA	V	V	RPA	RWA	RWA
APA	NA	NA	V	V	APA
AWA	NA	NA	V	V	AWA
V	NA	NA	NA	NA	V

R: Relocatable
P: Parcel
W: Word
A: Address
V: Value

*,/	RPA	RWA	APA	AWA	V
RPA	NA	NA	NA	NA	NA
RWA	NA	NA	NA	NA	NA
APA	NA	NA	NA	NA	NA
AWA	NA	NA	NA	NA	NA
V	NA	NA	NA	NA	V

SECTION III. INSTRUCTIONS AVAILABLE

The cross assembler supports all of the instructions currently accepted by the CRAY-Research Assembler, including the special syntax forms. These are presented on the following pages.

In addition, two instructions are available for simulator control. These are ERT and DRT.

The ERT instruction (op code 002600) enables resource timing. Execution of this instruction is equivalent to issuing the simulator command SET TIMING=ON.

The DRT instruction (op code 002700) disables resource timing, and is equivalent to the simulator command SET TIMING=OFF.

Error messages are printed if statement syntax is not correct. If the assembler cannot recognize an instruction, two parcels of zero are generated as object code.

INSTRUCTION SUMMARY

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
000xxx	ERR		5-81	-	Error exit
†000ijk	ERR	exp	5-81	-	Error exit
†††0010jk	CA,Aj	Ak	5-82	-	Set the channel (Aj) current address to (Ak) and begin the I/O sequence
†††0011jk	CL,Aj	Ak	5-83	-	Set the channel (Aj) limit address to (Ak)
†††0012jx	CI,Aj		5-83	-	Clear channel (Aj) interrupt flag
†††0013jx	XA	Aj	5-84	-	Enter XA register with (Aj)
†††0014jx	RT	Sj	5-84	-	Enter real-time clock register with (Sj)
0020xk	VL	Ak	5-25	-	Transmit (Ak) to VL register
†0020x0	VL	1	5-25	-	Transmit 1 to VL register
0021xx	EFI		5-42	-	Enable interrupt on floating point error
0022xx	DFI		5-42	-	Disable interrupt on floating point error
003xjx	VM	Sj	5-25	-	Transmit (Sj) to VM register
†003x0x	VM	0	5-25	-	Clear VM register
004xxx	EX		5-80	-	Normal exit
†004ijk	EX	exp	5-80	-	Normal exit
005xjk	J	Bjk	5-77	-	Jump to (Bjk)
006ijkm	J	exp	5-77	-	Jump to exp
007ijkm	R	exp	5-79	-	Return jump to exp; set B00 to P
010ijkm	JAZ	exp	5-78	-	Branch to exp if (A0) = 0
011ijkm	JAN	exp	5-78	-	Branch to exp if (A0) ≠ 0
012ijkm	JAP	exp	5-78	-	Branch to exp if (A0) positive
013ijkm	JAM	exp	5-78	-	Branch to exp if (A0) negative
014ijkm	JSZ	exp	5-78	-	Branch to exp if (S0) = 0
015ijkm	JSN	exp	5-78	-	Branch to exp if (S0) ≠ 0
016ijkm	JSP	exp	5-78	-	Branch to exp if (S0) positive
017ijkm	JSM	exp	5-78	-	Branch to exp if (S0) negative
††020ijkm			5-9	-	Transmit exp - jkm to Ai
††021ijkm	Ai	exp	5-9	-	Transmit exp = 1's complement of jkm to Ai
††022ijk			5-9	-	Transmit exp = jk to Ai
023ijx	Ai	Sj	5-16	-	Transmit (Sj) to Ai
024ijk	Ai	Bjk	5-16	-	Transmit (Bjk) to Ai
025ijk	Bjk	Ai	5-22	-	Transmit (Ai) to Bjk
026ijx	Ai	PSj	5-76	Pop/LZ	Population count of (Sj) to Ai
027ijx	Ai	LSj	5-76	Pop/LZ	Leading zero count of (Sj) to Ai
030ijk	Ai	Aj+Ak	5-35	A Int Add	Integer sum of (Aj) and (Ak) to Ai
†030i0k	Ai	Ak	5-15	A Int Add	Transmit (Ak) to Ai
†030ij0	Ai	Aj+1	5-35	A Int Add	Integer sum of (Aj) and 1 to Ai
031ijk	Ai	Aj-Ak	5-36	A Int Add	Integer difference of (Aj) less (Ak) to Ai
††031i00	Ai	-1	5-9	A Int Add	Transmit -1 to Ai
031i0k	Ai	-Ak	5-15	A Int Add	Transmit the negative of (Ak) to Ai
031ij0	Ai	Aj-1	5-36	A Int Add	Integer difference of (Aj) less 1 to Ai
032ijk	Ai	Aj*Ak	5-36	A Int Mult	Integer product of (Aj) and (Ak) to Ai

x = Machine does not use this field; CAL generates zero in this position.

† Special syntax form.

†† Instruction 020ijkm, 021ijkm, 022ijk, or 031i00 is generated depending on value of exp as described in section 5.

††† Instruction is privileged to monitor mode.

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
033i0x	Ai	CI	5-16	-	Channel number to Ai (j=0)
033ij0	Ai	CA,Aj	5-17	-	Address of channel (Aj) to Ai (j≠0; k=0)
033ij1	Ai	CE,Aj	5-17	-	Error flag of channel (Aj) to Ai (j≠0; k=1)
034ijk	Bjk,Ai	,A0	5-30	Memory	Read (Ai) words to B register jk from (A0)
†034ijk	Bjk,Ai	0,A0	5-30	Memory	Read (Ai) words to B register jk from (A0)
035ijk	,A0	Bjk,Ai	5-27	Memory	Store (Ai) words at B register jk to (A0)
†035ijk	0,A0	Bjk,Ai	5-27	Memory	Store (Ai) words at B register jk to (A0)
036ijk	Tjk,Ai	,A0	5-31	Memory	Read (Ai) words to T register jk from (A0)
†036ijk	Tjk,Ai	0,A0	5-31	Memory	Read (Ai) words to T register jk from (A0)
037ijk	,A0	Tjk,Ai	5-28	Memory	Store (Ai) words at T register jk to (A0)
†037ijk	0,A0	Tjk,Ai	5-28	Memory	Store (Ai) words at T register jk to (A0)
††040ijkm	Si	exp	5-10	-	Transmit jkm to Si
††041ijkm	Si	exp	5-10	-	Transmit exp = 1's complement of jkm to Si
042ijk	Si Si	<exp #>exp	5-12	S Logical	Form 1's mask exp = 64-jk bits in Si from the right
†042i77	Si	1	5-11	S Logical	Enter 1 into Si
†042i00	Si	-1	5-11	S Logical	Enter -1 into Si
043ijk	Si Si	>exp #<exp	5-13	S Logical	Form 1's mask exp = jk bits in Si from the left
†043i00	Si	0	5-11	S Logical	Clear Si
044ijk	Si	Sj&Sk	5-55	S Logical	Logical product of (Sj) and (Sk) to Si
†044ij0	Si	Sj&SB	5-55	S Logical	Sign bit of (Sj) to Si
†044ij0	Si	SB&Sj	5-55	S Logical	Sign bit of (Sj) to Si (j≠0)
†045ijk	Si	#Sk&Sj	5-57	S Logical	Logical product of (Sj) and 1's complement of (Sk) to Si
†045ij0	Si	#SB&Sj	5-57	S Logical	(Sj) with sign bit cleared to Si
046ijk	Si	Sj\Sk	5-59	S Logical	Logical difference of (Sj) and (Sk) to Si
†046ij0	Si	Sj\SB	5-59	S Logical	Toggle sign bit of Sj, then enter into Si
†046ij0	Si	SB\Sj	5-59	S Logical	Toggle sign bit of Sj, then enter into Si (j≠0)
047ijk	Si	#Sj\Sk	5-61	S Logical	Logical equivalence of (Sk) and (Sj) to Si
†047i0k	Si	#Sk	5-19	S Logical	Transmit 1's complement of (Sk) to Si
†047ij0	Si	#Sj\SB	5-61	S Logical	Logical equivalence of (Sj) and sign bit to Si
†047ij0	Si	#SB\Sj	5-61	S Logical	Logical equivalence of (Sj) and sign bit to Si (j≠0)
†047i00	Si	#SB	5-14	S Logical	Enter 1's complement of sign bit into Si
050ijk	Si	Sj!Si&Sk	5-64	S Logical	Logical product of (Si) and (Sk) complement ORed with logical product of (Sj) and (Sk) to Si
†050ij0	Si	Sj!Si&SB	5-64	S Logical	Scalar merge of (Si) and sign bit of (Sj) to Si
051ijk	Si	Sj!Sk	5-57	S Logical	Logical sum of (Sj) and (Sk) to Si
†051i0k	Si	Sk	5-18	S Logical	Transmit (Sk) to Si
†051ij0	Si	Sj!SB	5-57	S Logical	Logical sum of (Sj) and sign bit to Si
†051ij0	Si	SB!Sj	5-57	S Logical	Logical sum of (Sj) and sign bit to Si (j≠0)
†051i00	Si	SB	5-14	S Logical	Enter sign bit into Si
052ijk	S0	Si<exp	5-68	S Shift	Shift (Si) left exp = jk places to S0
053ijk	S0	Si>exp	5-68	S Shift	Shift (Si) right exp = 64-jk places to S0
054ijk	Si	Si<exp	5-69	S Shift	Shift (Si) left exp = jk places
055ijk	Si	Si>exp	5-69	S Shift	Shift (Si) right exp = 64-jk places
056ijk	Si	Si,Sj<Ak	5-70	S Shift	Shift (Si and Sj) left (Ak) places to Si
056ij0	Si	Si,Sj<1	5-70	S Shift	Shift (Si and Sj) left one place to Si
†056i0k	Si	Si<Ak	5-70	S Shift	Shift (Si) left (Ak) places to Si

x = Machine does not use this field; CAL generates zero in this position.

† Special syntax form.

†† Instruction 040ijkm or 041ijkm is generated depending on value of exp.

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
057ijk	Si	Sj,Si Ak	5-71	S Shift	Shift (Sj and Si) right (Ak) places to Si
†057ij0	Si	Sj,Si>1	5-71	S Shift	Shift (Sj and Si) right one place to Si
†057i0k	Si	Si>Ak	5-71	S Shift	Shift (Si) right (Ak) places to Si
060ijk	Si	Sj+Sk	5-37	S Int Add	Integer sum of (Sj) and (Sk) to Si
061ijk	Si	Sj-Sk	5-39	S Int Add	Integer difference of (Sj) and (Sk) to Si
†061i0k	Si	-Sk	5-18	S Int Add	Transmit negative of (Sk) to Si
062ijk	Si	Sj+FSk	5-42	F.P. Add	Floating sum of (Sj) and (Sk) to Si
†062i0k	Si	+FSk	5-42	F.P. Add	Normalize (Sk) to Si
063ijk	Si	Sj-FSk	5-44	F.P. Add	Floating difference of (Sj) and (Sk) to Si
†063i0k	Si	-FSk	5-44	F.P. Add	Transmit normalized negative of (Sk) to Si
064ijk	Si	Sj*FSk	5-46	F.P. Mult	Floating product of (Sj) and (Sk) to Si
065ijk	Si	Sj*HSk	5-48	F.P. Mult	Half precision rounded floating product of (Sj) and (Sk) to Si
066ijk	Si	Sj*RSk	5-49	F.P. Mult	Full precision rounded floating product of (Sj) and (Sk) to Si
067ijk	Si	Sj*ISk	5-51	F.P. Mult	2 - Floating product of (Sj) and (Sk) to Si
070ijx	Si	/HSj	5-52	F.P. Rcpl	Floating reciprocal approximation of (Sj) to Si
071i0k	Si	Ak	5-19	-	Transmit (Ak) to Si with no sign extension
071i1k	Si	+Ak	5-20	-	Transmit (Ak) to Si with sign extension
071i2k	Si	+FAk	5-20	-	Transmit (Ak) to Si as unnormalized floating point number
071i3x	Si	0.6	5-11	-	Transmit constant 0.75*2**48 to Si
071i4x	Si	0.4	5-11	-	Transmit constant 0.5 to Si
071i5x	Si	1.	5-11	-	Transmit constant 1.0 to Si
071i6x	Si	2.	5-11	-	Transmit constant 2.0 to Si
071i7x	Si	4.	5-11	-	Transmit constant 4.0 to Si
072ixx	Si	RT	5-22	-	Transmit (RTC) to Si
073ixx	Si	VM	5-22	-	Transmit (VM) to Si
074ijk	Si	Tjk	5-21	-	Transmit (Tjk) to Si
075ijk	Tjk	Si	5-23	-	Transmit (Si) to Tjk
076ijk	Si	Vj,Ak	5-21	-	Transmit (Vj, element (Ak)) to Si
077ijk	Vi,Ak	Sj	5-24	-	Transmit (Sj) to Vi element (Ak)
†077i0k	Vi,Ak	0	5-14	-	Clear Vi element (Ak)
10hijkm	Ai	exp,Ah	5-32	Memory	Read from ((Ah) + exp) to Ai (A0=0)
†100ijkm	Ai	exp,0	5-32	Memory	Read from (exp) to Ai
†100ijkm	Ai	exp,	5-32	Memory	Read from (exp) to Ai
†10hi000	Ai	,Ah	5-32	Memory	Read from (Ah) to Ai
11hijkm	exp,Ah	Ai	5-28	Memory	Store (Ai) to (Ah) + exp (A0=0)
†110ijkm	exp,0	Ai	5-28	Memory	Store (Ai) to exp
†110ijkm	exp,	Ai	5-28	Memory	Store (Ai) to exp
†11hi000	,Ah	Ai	5-28	Memory	Store (Ai) to (Ah)
12hijkm	Si	exp,Ah	5-33	Memory	Read from ((Ah) + exp) to Si (A0=0)
†120ijkm	Si	exp,0	5-33	Memory	Read from exp to Si
†120ijkm	Si	exp,	5-33	Memory	Read from exp to Si
†12hi000	Si	,Ah	5-33	Memory	Read from (Ah) to Si
13hijkm	exp,Ah	Si	5-29	Memory	Store (Si) to (Ah) + exp (A0=0)
†130ijkm	exp,0	Si	5-29	Memory	Store (Si) to exp
†130ijkm	exp,	Si	5-29	Memory	Store (Si) to exp
†13hi000	,Ah	Si	5-29	Memory	Store (Si) to (Ah)
140ijk	Vi	Sj&Vk	5-56	V Logical	Logical products of (Sj) and (Vk) to Vi
†140i00	Vi	0	5-14	V Logical	Clear Vi
141ijk	Vi	Vj&Vk	5-56	V Logical	Logical products of (Vj) and (Vk) to Vi

x = Machine does not use this field; CAL generates zero in this position.

† Special syntax form.

CRAY-1	CAL		PAGE	UNIT	DESCRIPTION
142ijk	Vi	Sj!Vk	5-58	V Logical	Logical sums of (Sj) and (Vk) to Vi
†142i0k	Vi	Vk	5-23	V Logical	Transmit (Vk) to Vi
143ijk	Vi	Vj!Vk	5-59	V Logical	Logical sums of (Vj) and (Vk) to Vi
144ijk	Vi	Sj\Vk	5-60	V Logical	Logical differences of (Sj) and (Vk) to Vi
145ijk	Vi	Vj\Vk	5-61	V Logical	Logical differences of (Vj) and (Vk) to Vi
146ijk	Vi	Sj!Vk&VM	5-65	V Logical	Transmit (Sj) if VM bit = 1; (Vk) if VM bit = 0 to Vi
†146i0k	Vi	#VM&Vk	5-67	V Logical	Vector merge of (Vk) and 0 to Vi
147ijk	Vi	Vj!Vk&VM	5-66	V Logical	Transmit (Vj) if VM bit = 1; (Vk) if VM bit = 0 to Vi
150ijk	Vi	Vj<Ak	5-72	V Shift	Shift (Vj) left (Ak) places to Vi
†150ij0	Vi	Vj<1	5-72	V Shift	Shift (Vj) left one place to Vi
151ijk	Vi	Vj>Ak	5-73	V Shift	Shift (Vj) right (Ak) places to Vi
†151ij0	Vi	Vj>1	5-73	V Shift	Shift (Vj) right one place to Vi
152ijk	Vi	Vj,Vj<Ak	5-73	V Shift	Double shift (Vj) left (Ak) places to Vi
†152ij0	Vi	Vj,Vj<1	5-73	V Shift	Double shift (Vj) left one place to Vi
153ijk	Vi	Vj,Vj>Ak	5-75	V Shift	Double shift (Vj) right (Ak) places to Vi
153ij0	Vi	Vj,Vj>1	5-75	V Shift	Double shift (Vj) right one place to Vi
154ijk	Vi	Sj+Vk	5-38	V Int Add	Integer sums of (Sj) and (Vk) to Vi
155ijk	Vi	Vj+Vk	5-38	V Int Add	Integer sums of (Vj) and (Vk) to Vi
156ijk	Vi	Sj-Vk	5-39	V Int Add	Integer differences of (Sj) and (Vk) to Vi
†156i0k	Vi	-Vk	5-40	V Int Add	Transmit negative of (Vk) to Vi
157ijk	Vi	Vj-Vk	5-40	V Int Add	Integer differences of (Vj) and (Vk) to Vi
160ijk	Vi	Sj*FVk	5-47	F.P. Mult	Floating products of (Sj) and (Vk) to Vi
161ijk	Vi	Vj*FVk	5-47	F.P. Mult	Floating products of (Vj) and (Vk) to Vi
162ijk	Vi	Sj*HVk	5-48	F.P. Mult	Half precision rounded floating products of (Sj) and (Vk) to Vi
163ijk	Vi	Vj*HVk	5-49	F.P. Mult	Half precision rounded floating products of (Vj) and (Vk) to Vi
164ijk	Vi	Sj*RVk	5-50	F.P. Mult	Rounded floating products of (Sj) and (Vk) to Vi
165ijk	Vi	Vj*RVk	5-50	F.P. Mult	Rounded floating products of (Vj) and (Vk) to Vi
166ijk	Vi	Sj*IVk	5-51	F.P. Mult	2 - floating products of (Sj) and (Vk) to Vi
167ijk	Vi	Vj*IVk	5-52	F.P. Mult	2 - floating products of (Vj) and (Vk) to Vi
170ijk	Vi	Sj+FVk	5-43	F.P. Add	Floating sums of (Sj) and (Vk) to Vi
†170i0k	Vi	+FVk	5-43	F.P. Add	Normalize (Vk) to Vi
171ijk	Vi	Vj+FVk	5-44	F.P. Add	Floating sums of (Vj) and (Vk) to Vi
172ijk	Vi	Sj-FVk	5-45	F.P. Add	Floating differences of (Sj) and (Vk) to Vi
†172i0k	Vi	-FVk	5-45	F.P. Add	Transmit normalized negatives of (Vk) to Vi
173ijk	Vi	Vj-FVk	5-46	F.P. Add	Floating differences of (Vj) and (Vk) to Vi
174ijx	Vi	/HVj	5-53	F.P. Rcpl	Floating reciprocal approximations of (Vj) to Vi
175xj0	VM	Vj,Z	5-63	V Logical	VM=1 where (Vj) = 0
175xj1	VM	Vj,N	5-63	V Logical	VM=1 where (Vj) ≠ 0
175xj2	VM	Vj,P	5-63	V Logical	VM=1 where (Vj) positive
175xj3	VM	Vj,M	5-63	V Logical	VM=1 where (Vj) negative
176ixk	Vi	,A0,Ak	5-33	Memory	Read (VL) words to Vi from (A0) incremented by (Ak)
†176ix0	Vi	,A0,1	5-33	Memory	Read (VL) words to Vi from (A0) incremented by 1
177xjk	,A0,Ak	Vj	5-30	Memory	Store (VL) words from Vj to (A0) incremented by (Ak)
†177xj0	,A0,1	Vj	5-30	Memory	Store (VL) words from Vj to (A0) incremented by 1

x = Machine does not use this field; CAL generates zero in this position.

† Special syntax form.

SECTION IV. PSEUDO OPS

The following Pseudo-Ops are currently accepted:

Pseudo Op	Meaning
sym = exp	define a symbol
ABS	absolute assembly
BASE O, D, or M	use base octal, decimal, or mixed
BSS n	reserve n words of storage
BSSZ n	reserve n words of zeroed storage
CON exp	define a full word integer constant
DATA exp	define a full word floating point constant
EJECT	list next line at top of new page
END	end of routine
ENTRY label	specify routine entry point
EXT sym	identify name of external symbol
IDENT name	specify name of routine
ORG exp	set parcel counter
sym SET exp	similar to =, but sym may be redefined
SPACE n	skip n blank lines in the listing
START label	identify program starting location
SUBTITLE 'any text'	specify sub title to appear on listing
TITLE 'any text'	specify title to appear on listing.

= PSEUDO OP

Purpose: To define a symbol

Prototype: sym = expr [,p|,w|,v]

Description:

The expression is evaluated as described in section II. If the expression is suffixed with ,P or ,W or ,V the expression is converted to type Parcel, Word or Value. The symbol in the label field is given the value, type and attributes of the expression. If the expression contains errors, the symbol will remain undefined.

Examples:

```
SY1  = 3
PADR = 4A
LEN  = W.* - W.TOP
BASE = ARRAY - IMAX - IMAX*JMAX
```

ABS PSEUDO OP

Purpose : To assemble the current routine as an absolute module

Prototype : ABS

Description:

Each routine, by default, is assembled as a relocatable module. If ABS occurs anywhere in the routine, an absolute module will be produced. The simulator will load the module at the absolute location specified by the ORG pseudo op.

Example:

```
ABS
```

BASE PSEUDO OP

Purpose : To specify the default base for numeric constants

Prototype : BASE [O|D|M|*]

Description:

The default base for constants is decimal. Occurance of a BASE O,D or M changes the base to Octal, Decimal or Mixed.

When a BASE O, D, or M is encountered, the current base is pushed on to a stack. A BASE * pseudo pops this stack. The stack can hold at most 20 entries.

Examples:

```
BASE O
BASE *
```

BSS PSEUDO OP

Purpose : To reserve words of storage

Prototype : label BSS v-expr

Description:

The parcel counter is full word aligned, if necessary. Then it is advanced v-expr words. The v-expr must be a positive value. This v-expr is evaluated during pass 1, so symbols used in the v-expr must be defined earlier in the program.

If a label is present, it is entered into the symbol table as the word address of the beginning of the block.

Examples:

```
LAB   BSS   3
      BSS   2*[DIM
```

BSSZ Pseudo OP

Purpose : To reserve words of storage with initial value zero

Prototype : [label] BSSZ [v-expr]

Description:

The parcel counter is full word aligned, if necessary. Then, v-expr full words of zero are generated. The v-expr must be a positive value. This v-expr is evaluated during pass 1, so symbols used in the v-expr must be defined earlier in the program.

If a label is present, it is entered into the symbol table as the word address of the beginning of the block.

The operand field may be omitted. If so, it is assumed to be zero. Note that full word alignment still takes place.

Examples:

```
LAB   BSSZ   3
      BSSZ
```

CON PSEUDO OP

Purpose : To define integer or character constants

Prototype : [label] CON expr[,expr ...]

Description:

The parcel counter is full word aligned. Each expr is evaluated, and a full word is generated containing its value.

If label exists, it is defined as a word address.

Examples:

```
LABL   CON   3
       CON   4,7*8+2,'CHARS'
```

DATA PSEUDO OP

Purpose : To define floating point constants
Prototype : [label] DATA v1[,v2 ...]
Description:

The parcel counter is full word aligned. Each v is evaluated as a floating point constant, and a full word of storage is generated containing it. The host machine's floating point format is used, for compatability with the simulator.

If the label exists, it is defined as a word address.

Examples:

```
PI    DATA   3.14159
      DATA   2.,3.,4.
```

EJECT PSEUDO OP

Purpose : To advance the listing to the top of a new page
Prototype : EJECT
Description:

The next line of the listing will be at the top of the next page. If the next line to be printed is already at the top of a page, the EJECT card is ignored.

Example:

```
      EJECT
```

END PSEUDO OP

Purpose : To terminate assembly of a routine

Prototype : END

Description:

Assembly of the current routine is terminated. Generation of object code is completed. The assembler is then initialized to assemble another routine following the END record. If the END record is followed by an end of file, the assembler terminates normally.

Example:

```
END
```

ENTRY PSEUDO OP

Purpose : To indicate symbols in the routine which are to be available for use by other routines.

Prototype : ENTRY symbol

Descritpion:

The symbol in the operand field is defined as an entry point to the current routine. It is available for use by other relocatable modules. It may be either a parcel or word address.

If the current routine is being assembled as an absolute module, the ENTRY record has no effect.

Example:

```
ENTRY  LOC 1
```

EXT PSEUDO OP

Purpose : To indicate symbols which will be defined by external routines

Prototype : EXT sym[,sym ...]

Description:

The symbols in the operand field are identified as externals, and made available for use in the current routine. The symbols will be resolved at load time.

If the current routine is being assembled as an absolute module, the EXT record has no effect.

Examples:

```
EXT   SUBR
EXT   LOG10,%SQRT%
```

IDENT PSEUDO OP

Purpose : To identify the name of the current module

Prototype : IDENT name

Description:

Assembly of a new routine is begun. The routine is given the name in the operand field. This name must meet the same syntax requirements as other symbols. It may be used as a label, but these uses are independent.

No statements may appear before an IDENT record. If there are any, they will be treated as comments.

Example:

```
IDENT   MAIN
```

ORG PSEUDO OP

Purpose : To give a new value to the parcel counter
Prototype : ORG expr
Description:

The expr is evaluated, and becomes the new value of the parcel counter. The expr must be defined during pass 1.

The expr must have attributes matching the type of assembly (absolute or relocatable).

The parcel counter may not be lowered using an ORG.

Examples:

```
ORG  10    (Absolute assembly)
ORG  *+20  (Relocatable assembly)
```

SET PSEUDO OP

Purpose : To define a symbol
Prototype : sym SET expr
Description:

The expression is evaluated. The symbol in the label field is given its value.

The expression must be defined during pass 1.

Symbols defined using the SET pseudo may be re-defined.

Examples:

```
A     SET   3
COUNT SET   COUNT+1
```

SPACE PSEUDO OP

Purpose : To insert blank lines into the listing

Prototype : SPACE v-expr

Description:

The v-expr is evaluated. Then, v-expr blank lines are inserted in the listing. If there are fewer than v-expr lines remaining on the current page, a simple EJECT is performed. If the next line is already to be listed at the top of a page, the SPACE record is ignored.

If the current base is mixed, decimal will be assumed.

Example:

```
SPACE   3
```

START PSEUDO OP

Purpose : To identify the current module as the module in which execution is to begin.

Prototype : START sym

Description:

The current module will be used as the first module to be executed when execution begins. The symbol in the operand field must be a parcel address, and the label of an instruction in the current module.

The START pseudo has no effect during absolute assemblies.

Example:

```
START   LTOP
```

SUBTITLE PSEUDO OP

Purpose : To specify a subtitle for the listing
Prototype : SUBTITLE 'any text'
Description:

The text specified will be printed at the bottom of each page of listing of the current routine. The text within quotes may consist of up to 64 characters.

A page eject is also performed.

Example:

```
SUBTITLE   'FOOTER TEXT'
```

TITLE PSEUDO OP

Purpose : To specify a title for the listing
Prototype : TITLE 'any text'
Description:

The text specified will be printed at the top of each page of listing of the current routine. The text within quotes may consist of up to 64 characters.

Example:

```
TITLE   'Square Root Routine'
```

Appendix A.
Load Module Formats

1. Absolute Modules

An absolute load module is composed of a header record and one or more parcel records. The header record is composed of three free formatted fields: a starting parcel address, a module length and an optional title as shown below.

p-addr length [title-string]

Each field is separated by blanks. The p-addr field contains a modified octal (octal work address followed by an A,B,C or D parcel code) parcel address where the first parcel is stored. The length field contains the octal number of parcels following on subsequent parcel records. The title-string field contains the title that is placed in the title field of the CPACT report (also see the SET TITLE = title-string command). The title field begins with the first character after the blank which terminates the length field and may contain embedded blanks. Only the first 35 characters are retained. The LOAD command will echo the header record information to the output device.

The parcel record portion of the absolute load module contains one or more free formatted records. One or more parcels may be provided on each record in octal or modified octal format and must be separated by at least one space. A colon may be placed anywhere on a parcel record which will stop the scan of the record at that point. This allows comments to be placed to the right of the colon. Blank records in the parcel record portion are ignored. The number of parcels in the parcel record portion of the load module must correspond to the octal length specified on the header record.

All records in an absolute load module must be less than 81 characters. Since the load modules are represented in character format it is a simple matter to patch a module with the editor. If you add or delete parcels, you must update the octal length appropriately.

Example:

```
21A  3  A TITLE
:  THIS IS AN EXAMPLE OF A 3 PARCEL
:  ABSOLUTE LOAD MODULE
022133  032211  :  A1  33  ;  A2  A1*A1
004000  :  EX  O
$ENDFILE
```

2. Relocatable Modules

Relocatable modules consist of seven types of binary records. An IDEN record, one or more TXT records, zero or more RLD, EXT, ENTR, and SYM records, and an END record.

An IDEN record identifies the name of the module. The record consists of the characters IDEN, followed by 4 spaces, followed by the 8 character name of the module.

A TXT record contains the actual object code to be loaded. It consists of the letters TXT, followed by one space, followed by a four byte binary address of this portion of the module (relative to the top of the module), followed by a four byte binary length. The actual text to be loaded is on the following card.

An RLD record identifies the locations in the module which must be relocated. It consists of the letters RLD, followed by one space, followed by one or more 8 byte fields. The first 4 bytes of the field contain the binary address (relative to the top of the module) of the text to be relocated. The second 4 bytes contain a number describing the type of relocation to be performed. See RLD & EXT types, below.

An EXT record identifies the locations in the module which refer to external locations. It consists of the letters EXT, followed by 5 spaces, followed by one or more 16 byte fields. The first 8 bytes of the field contains the 8 character name of the external location referenced. The next four bytes contain the binary address (relative to the top of the module) of the text referencing the external. The last 4 bytes contain a number describing the type of reference. See RLD & EXT types, below.

An ENTR record identifies entry points in the module. It consists of the letters ENTR, collowed by 4 spaces, followed by one or more 12

byte fields. The first 8 bytes of the field contain the name of the entry point, and the last 4 bytes contain the address (relative to the top of the module) of the entry point.

A SYM record contains definitions of all symbols in the module which may be referenced by the simulator command language. It consists of the letters SYM, followed by 5 spaces, followed by one or more 16 byte fields. The first 8 bytes contain the name of the symbol. The next 4 bytes contain the value of the symbol. The last 4 bytes contain a number identifying the type of the symbol. See SYM types, below.

An END record terminates the module. It consists of the letters END, followed by 1 space. If a START pseudo op was contained in the program, the next 4 bytes contain an address (relative to the top of the module) of the starting location.

RLD and EXT types:
1: Two parcel, parcel address
2: Two parcel, word address
3: Four parcel, parcel address
4: Four parcel, word address

SYM types:
1: Parcel
2: Value
3: Word

Appendix B. Error Messages

NOTE: Warning messages are indicated by (w).

1. Pass 1 error messages

(w) LINE LONGER THAN 90 CHARACTERS TRUNCATED

An input line was too long. Characters after Column 90 will be removed.

(w) NO END CARD

An end card will be supplied.

2. Pass 2 error messages

(w) OLD STYLE B OR T BLOCK TRANSFER ENCOUNTERED

An old B or T transfer will be assembled correctly, but should be converted to the new style.

UNRECOGNIZABLE INSTRUCTION

The assembler could not recognize the instruction. It will be assembled as two parcels of zero's.

(w) SYMBOL HAS TOO MANY CHARACTERS, TRUNCATED.

The symbol being defined was more than 8 characters. Only the first 8 will be used.

INVALID SYMBOL SYNTAX

The symbol being defined starts with or contains illegal characters.

ATTEMPT TO REDEFINE SYMBOL

Only symbols defined with the SET pseudo op may be redefined.

I FIELD OUT OF RANGE: n

I field must be between 0 and 7, inclusive.

J FIELD OUT OF RANGE: n

J field must be between 0 and 7, inclusive.

K FIELD OUT OF RANGE: n

K field must be between 0 and 7, inclusive.

INVALID I FIELD

I field was neither numeric nor .SYM: or SYM was undefined; or SYM was not of type value.

INVALID J FIELD

J field was neither numeric nor .SYM; or SYM was undefined; or SYM was not of type value.

DUPLICATE I FIELDS DO NOT MATCH
An instruction requiring the appearance of the I field twice contained different values for each field.

INVALID K FIELD

K field was neither numeric nor .SYM; or SYM was undefined; or SYM was not of type value.

DUPLICATE J FIELDS DO NOT MATCH

An instruction requiring the appearance of the J field specifier twice contained different values for each field.

INVALID K FIELD

K field was neither numeric nor .SYM; or SYM was undefined; or SYM was not of type value.

JK FIELD OUT OF RANGE: n

JK field must be between 0 and 63, inclusive.

INVALID JK FIELD

JK field was neither numeric nor .SYM; or SYM was undefined; or SYM was not of type value.

SHIFT COUNT OUT OF RANGE:

Explicit shift counts must be between 0 and 63, inclusive.

INVALID SHIFT COUNT

Shift count was neither numeric nor a symbol of type value.

INVALID P ADDRESS

P address specified had illegal explicit address syntax, or referenced a symbol which was undefined or was of type word address.

INVALID WORD ADDRESS

Word address specified had illegal explicit address syntax, or referenced a symbol which was undefined or was of type parcel address.

H FIELD OUT OF RANGE: n

H field must be between 0 and 7, inclusive.

INVALID H. FIELD

H field was neither numeric, nor .SYM; or SYM was undefined; or SYM was not of type value.

IJK FIELD OUT OF RANGE: n

IJK field must be between 0 and 511, inclusive.

INVALID IJK FIELD

IJK field was neither numeric nor a symbol of type value.

UNDEFINED EXPRESSION

Expression could not be evaluated.

BASE STACK OVERFLOW

The base stack exceeded the 20 entries. First stack entries will be lost.

ILLEGAL BASE

Base specified must be D, I, or M.
xxx NOT YET SUPPORTED

The pseudo op referenced is not implemented.

INVALID BSS COUNT

BSS count must be explicit number, or symbol of type value.

INVALID BSSZ COUNT

BSSZ count must be explicit number, or symbol of type value.

INVALID CONSTANT

Operand field of CON pseudo op could not be evaluated.

PROGRAMMER FORCED ERROR

ERROR pseudo op encountered.

ILLEGAL IDENT NAME

Name started with or contained invalid characters.

INVALID SPACE COUNT

Space count was neither explicit positive number nor symbol of type value.

INVALID ORG

Address must be explicit value or symbol of type value; or address is less than current address.

INVALID STARTING LOCATION

Symbol invalid or undefined.

MISSING OR BAD TITLE

Title must be surrounded by primes and contain 64 or fewer characters.

MISSING OR BAD SUBTITLE

Subtitle must be surrounded by primes and contain 64 or fewer characters.

3. Internal Assembler Errors

END OF DESCRIPTOR UNEXPECTED, ERROR IN FILL
CHARACTER MATCH FAILED
EXPECTING 4, NOT FOUND
EXPECTING 6, NOT FOUND
EXPECTING 1, NOT FOUND
EXPECTING ·, NOT FOUND
EXPECTING 2, NOT FOUND
SPECIAL DESCRIPTOR ENCOUNTERED - NOT IMPLEMENTED

Appendix C. Instruction Tables

The assembler uses a table driven parser. The tables are prepared from a file containing instruction description

1. Description file contents

Each line of the description file contains information about one instruction or pseudo-op.

The type of instruction is contained in column 1. The types are defined as follows:

1. single parcel instruction
2. two parcel instruction
3. Ai exp instruction
4. Si exp instruction
5. Pseudo Op
6. Old style block B or T transfer

The result field instruction prototype begins in Column 5 and the operand prototype begins in Column 30. Each is no more than 8 characters. An instruction prototype is formed for an instruction by replacing all digits or strings of digits with a single zero and any .SYM field with a single zero.

The result field descriptor begins in Column 15, and the operand field descriptor begins in Column 40. Each is no more than 10 characters. Each character describes the nature of the corresponding prototype character. The valid descriptor characters and meanings are:

C	exact character match with prototype
I	I field of instruction
J	J field of instruction
K	K field of instruction
L	JK field of instruction
V	shift count
P	parcel address expression

W	word address expression
H	H field of instruction
M	IJK field of instruction
F	exact character match with 4
S	exact character match with 6
O	exact character match with 1
D	exact character match with .
T	exact character match with 2
U	exact character match with 4
Q	64-JK field (shift counts)
R	JK-1 field (old B & T transfers)
↓	special, not currently used.
X	end at descriptor

The instruction op-code is in Columns 50-55. Note that pseudo ops have been given op codes for internal use.

2. Generation

A program (not strictly part of the assembler) converts the description file into a condensed binary representation which in turn is used by the assembler. The description file must first be sorted by result prototypes, with groups having identical result prototypes being sorted by operand prototypes. The mnemonic table generator reads the sorted description file from unit 5, and produces the mnemonic tables on unit 7.

Appendix D. Error Stops

The cross assembler makes a number of checks on itself during normal operation. If one of these checks fails, it indicates a problem within the assembler. The stop codes which will be printed and their meanings are presented below.

Routine Name	Stop Code	Meaning
COLDST	101	Unable to initialize dynamic arrays
STXTND	102	Unable to extend dynamic arrays
INIT1	103	Unable to create dynamic arrays
DECOD1	104	Unknown instruction type encountered
WINTR	105	End of file during write to intermediate file
COLDST	106	Unable to open intermediate file
INIT1	107	Error during empty of intermediate file
GETPAR	109	Error return from FINDST
COLDST	110	End of file during read from mnemonic file
COLDST	111	Incorrect record length from mnemonic file
PSUDO1	112	Unknown pseudo op type
PSUDO2	112	Unknown pseudo op type
GTNL2	113	End of file during read from intremediate file
DECOD2	114	Unknown instruction type encountered
ERROR	116	No message delimiter (↓)
TYCONV	117	Conversion between unknown types
FILL	119	Unknown descriptor element
COLDST	120	Mnemonic tables too small
FILLP1	121	Address field of 1 parcel instruction not zero
EMIT	122	End of file during write to object file

Appendix E. Program Availability

The cross assembler is written in Fortran-IV, and contains about 3000 lines of source code. On the Amdahl 470/V6 at the University of Michigan, about 40K (32-bit words) of run-time memory is required.

The assembler is mildly dependent on the operating system (MTS) running at the University of Michigan. The assembler and documentation of machine dependent portions are available on 9 track tape (800, 1600, or 6250 BPI) from:

Professor D. A. Calahan
2510 E. Engineering
University of Michigan
Ann Arbor, Michigan 48109