AD-A054 944 SYRACUSE UNIV N Y F/G 9/2
LARGE SCALE INFORMATION SYSTEMS. VOLUME III.(U)
MAR 78 F30602-74-C-0335
UNCLASSIFIED RADC-TR-78-43-VOL-3 NL

1 OF 4
AD
A054944



AD A054944

FOR FURTHER TRAN

RADC-TR-78-43, Volume III (of four)
Final Technical Report
March 1978

LARGE SCALE INFORMATION SYSTEMS

Syracuse University

THIS DOCUMENT IS BEST QUALITY PRACTICABLE
THE COPY FURNISHED TO DDC CONTAINED A
SIGNIFICANT NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.

Approved for public release; distribution unlimited.

DDC
RECEIVED
JUN 12 1978
F

DDC FILE COPY
NO.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-78-43, Volume III (of four) has been reviewed and is approved for publication.

APPROVED: James L. Previte

JAMES L. PREVITE
Project Engineer

APPROVED: Wendall C. Bauman

WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISCA) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-78-43, Vol III (of four)	2. GOVT ACCESSION NO. 3
4. TITLE (and Subtitle): LARGE SCALE INFORMATION SYSTEMS, Volume III.	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, 3 Jun 74 — 2 Feb 77
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Syracuse University	8. CONTRACT OR GRANT NUMBER(s): F30602-74-C-0335
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Syracuse University, Syracuse New York 13210	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 55810244, 17 02
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISCA), Griffiss AFB NY 13441	12. REPORT DATE: March 1978; 13. NUMBER OF PAGES: 382
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same (12) 377p.	15. SECURITY CLASS. (of this report): UNCLASSIFIED; 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

Same

DDC RECEIVED JUN 12 1978 F

18. SUPPLEMENTARY NOTES

RADC Project Engineer:
James L. Previte (ISCA)

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

Parallel Processing
Programming Languages
Simulation
Data Base Management
Computer Architecture

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

This report describes and references work conducted by Syracuse University in four broad areas: parallel processing, programming languages, modeling, and performance evaluation of generalized data management systems.

A number of applications were evaluated for processing by an associative processor architecture including air traffic control, carryless arithmetic and simulation of high-speed random logic. An extension of the typed lambda (Cont'd)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

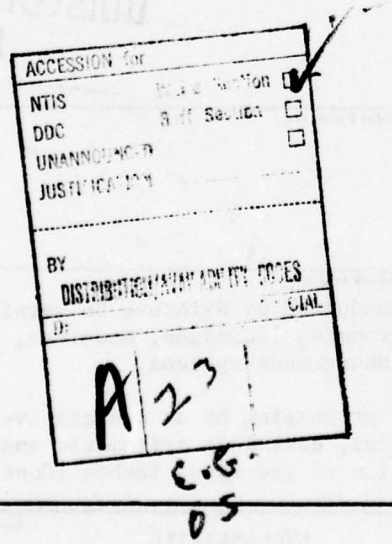
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

339 600

calculus has been developed which permits the binding and application of types. User-defined types and procedural data structures are shown to be complementary tools for data abstraction. Direct and continuation semantics of the domain of flow diagrams are formulated and the properties explored. The use of transition diagrams as a tool for structured programming has been investigated. A variety of concepts and notations have been devised to facilitate reasoning about arrays.

Work relation to various simulation tasks are reported on. A tutorial on the current statistical methods of analyzing simulation output data is provided.

A number of tasks relating to file systems are discussed and a framework is advanced for describing various file organizations and operations on files.



E.G.
05

Preface

This report describes efforts completed in the Large Scale Information Systems project at Syracuse University under RADC contract F30602-74-C-0335. The work covers the period June 3, 1974 through February 2, 1977.

The report is produced in four volumes to facilitate single volume distribution.

Contents of Volume 1

Section 1. Overview of the contract period.

Section 2. Semantics of the Domain of Flow Diagrams
by John C. Reynolds

Section 3. User-Defined Types and Procedural Data Structures as Complementary Approaches to Data Abstraction
by John C. Reynolds

Section 4. Towards a Theory of Type Structure
by John C. Reynolds

Section 5. An Introduction to Transaction Processing Systems
by Daniel Wood and Robert G. Sargent

Section 6. Analysis and Design of a Cost-Effective Associative Processor for Weather Computations
by W. Cheng and T. Feng

Section 7. AAPL: An Array Processing Language
by John G. Marzolf

Contents of Volume 2

Section 8. Reasoning about Arrays

by John C. Reynolds

Section 9. Evaluation Models for Index Sequential Files

by Amrit L. Goel and Yuan Liu

Section 10. File Organization Concepts

by Yuan Liu and Amrit L. Goel

Section 11. Concurrency in Hashed File Access

by Leo H. Groner and Amrit L. Goel

Section 12. Cascade Hashing

by Yuan Liu and Amrit L. Goel

Section 13. The Design and Implementation of APL-STARAN

by John G. Marzolf

Contents of Volume 3

Section 14. Mixed-Mode Arithmetic for STARAN

by E. P. Stabler and J. Hsu

Section 15. Parallel Arithmetic Using Serial Arithmetic Processors

by E. P. Stabler and J. Hsu

Contents of Volume 4

Section 16. Statistical Analysis of Simulation Output Data

by Robert G. Sargent

Section 14

MIXED-MODE ARITHMETIC MACROS FOR STARAN

E. P. Stabler

J. Hsu

Syracuse University

Foreward

The macros in this manual utilize the mixed mode facility of STARAN to achieve faster arithmetic speed. The basic process involves doing arithmetic on all the bits of the selected operands instead of a signle bit at a time as in the bit serial case. The average times for doing the basic arithmetic operations are shown in chapter 3. About a 2:1 improvement in add/subtract time is achieved. A mixed mode multiply takes about 1/4 the time of the bit serial version and mixed mode divide about 2/3 the time of bit serial.

In order to use these routines the programmer need know very little about their operation. He is free to intersperse bit serial and mixed mode routines in his program. In order to do this he should read chapter 1 and the beginning of chapter 2 which describe the alternate addressing for the two modes. The description of each macro includes a memory map which indicates clearly which words are selected and altered by the routine operating with mixed mode addressing. The program INIT should be executed prior to any mixed mode arithmetic. It creates various constants which are needed in the arithmetic routines.

Table of Contents

Chapter		Page
1	INTRODUCTION TO THE MIXED MODE	14-1-1
2	MIXED MODE ARITHMETIC AND SEARCH INSTRUCTIONS . .	14-2-1
	General	14-2-1
	Address Scheme of Mixed Mode Mod-32	14-2-1
	Address Scheme of Mixed Mode Mod-64	14-2-5
	Introdution to the Mixed Mode Routines	14-2-9
	Mixed Mode Operations Initiator	14-2-10
	INIT	14-2-11
	Mixed Mode Addition and Subtraction Instructions	14-2-12
	ADWM	14-2-13
	ADWMA	14-2-19
	ADCM	14-2-20
	ADCMA	14-2-25
	SUBWM	14-2-26
	SUBWMA	14-2-32
	SUBCM	14-2-33
	SUBCMA	14-2-39
	Mixed Mode Multiplication and Division Instructions	14-2-40
	MPUUM	14-2-41
	MPUUMA	14-2-47
	MPULM	14-2-48
	MPULMA	14-2-54

Chapter		Page
	MPLUM	14-2-55
	MPLUMA	14-2-61
	MPLLM	14-2-62
	MPLLMA	14-2-68
	MPCUM	14-2-69
	MPCUMA	14-2-75
	MPCLM	14-2-76
	MPCLMA	14-2-82
	DVUUM	14-2-83
	DVUUMA	14-2-89
	DVULM	14-2-90
	DVULMA	14-2-96
	DVLUM	14-2-97
	DVLUMA	14-2-103
	DVLLM	14-2-104
	DVLLMA	14-2-110
	DVCUM	14-2-111
	DVCUMA	14-2-117
	DVCLM	14-2-118
	DVCLMA	14-2-124
	Mixed Mode Search Instructions	14-2-125
	FQWM	14-2-126
	EQWMA	14-2-131
	EQCM	14-2-132

Chapter		Page
	EQCMA	14-2-137
	NEWM	14-2-138
	NEWMA	14-2-143
	NECM	14-2-144
	NECMA	14-2-149
	GTWM	14-2-150
	GTWMA	14-2-156
	GTCM	14-2-157
	GTCMA	14-2-163
	GEWM	14-2-164
	GEWMA	14-2-170
	GECM	14-2-171
	GECMA	14-2-177
	LTWM	14-2-178
	LTWMA	14-2-184
	LTCM	14-2-185
	LTCMA	14-2-191
	LEWM	14-2-192
	LEWMA	14-2-198
	LECM	14-2-199
	LECMA	14-2-205
3	EXECUTION TIMES OF THE MIXED MODE INSTRUCTIONS .	14-3-1
Appendix	LISTING OF THE MIXED MODE MACROS	14-A-1

CHAPTER 1

Introduction to the Mixed Mode

The associative arrays are the heart of the STARAN S system. There may be a maximum of 32 arrays* in one system. Each associative array contains a multidimensional-access (MDA) memory with 2^{16} bits of storage and three response-store registers X, Y, and M (Mask) register. The response-store registers in each array allow array data to be searched, restructured, and processed at a fast rate, 256 bits at a time. The array memory is arranged in a 256-bit by 256-bit square. In a standard STARAN'S system, the MDA memories in the arrays can be accessed either the vertical (bit-slice mode) or horizontal (word mode) direction and with a maximum of 256 bits transferred in a single operation. The address mode register (AMR) option allows access to the MDA memories in other modes as well. The other modes are intermediate to the bit-slice mode and the word mode since they access some bits of some words and are called mixed modes.

When an MDA memory is accessed, the program specifies two 8-bit bytes: an address A and a mode constant Z. The mode constant Z selects a certain pattern or stencil of 256 bits and the address A positions the stencil over the memory, selecting 256 memory bits for access. The address A is either specified directly in bit 8-15 of an associative machine instruction or specified indirectly as the contents of field pointers FP1, FP2 or FP3. The mode constant Z is always specified indirectly as the contents of address mode register, AMR0 or AMR1.

Let T_k refer to the kth bit of the transferred data where k is any 8-bit byte and $S_{I,J}$ refers to the Ith bit of the Jth word where I and J are 8-bit bytes. Given an address A, and a mode constant Z, the association is

$$T_k \leftrightarrow S_{\psi(Z,A,K),\psi(Z,K,A)}$$

where $\psi(Z,X,Y) = (\sim Z \wedge X) \vee (Z \wedge Y)$

Some special cases of the mixed mode are needed in the arithmetic and search operations.

(1) Mode X'00' (Bit slice mode)

Z = 00000000

then $\psi(Z,A,K) = A$
$\psi(Z,K,A) = K$

so the association becomes

$$T_K \leftrightarrow S_{A,K}$$

(2) Mode X'FF' (Word Mode)

Z = 11111111

*In the RADCAP system, there are only 4 arrays connected.

then $\psi(Z,A,K) = K$
$\psi(Z,K,A) = A$

so the association becomes

$$T_k \leftrightarrow S_{K,A}$$

(3) Mode X'1F' (Mixed Mode Mod-32)

Z = 00011111

Let B be the leftmost 3 bits of an 8-bit address A and let C be the rightmost 5 bits of A. Similarly, let L be the leftmost 3 bits of an 8-bit line index, K, and let M be the rightmost 5 bits of K. Then

$\psi(Z,A,K) = BM$
$\psi(Z,K,A) = LC$

so the association becomes

$$T_{LM} \leftrightarrow S_{BM,LC}$$

(4) Mode X'3F' (Mixed Mode Mod-64)

Z = 00111111

Let B be the leftmost 2 bits of an 8-bit address A and let C be the rightmost 6 bits of A. Similarly, let L be the leftmost 2 bits of an 8-bit line index, K, and let M be the rightmost 6 bits of K. Then

$\psi(Z,A,K) = BM$

$\psi(Z,K,A) = LC$

so the association becomes

$$T_{LM} \leftrightarrow S_{BM,LC}$$

CHAPTER 2

Mixed Mode Arithmetic and Search Instructions

General

The instructions discussed in this chapter are additional APPLE assembler mnemonics. They are implemented as macros in the MACRO/APPLE system. The language syntax and statement format rules of the APPLE assembler apply to these instructions. For detailed discussion, see APPLE programming manual and MACRO programming manual.

Address Scheme of Mixed Mode Mod-32

The mixed mode mod-32 segments each of the selected associative memory modules into eight sections, as shown in Fig. 2-1. Each section consists of 32 by 256 bits memory and 32 bits response store registers M(i), X(i) and Y(i), where $i = 0 \sim 7$. The address will select a 32-bit word from each section simultaneously. Figure 2-2 shows the address scheme of a section in the selected arrays.

For example, if under the mixed mode mod-32, the address is 4, then every 32-bit word mod-32 with number 4 in every section is selected.

Fig. 2-1. Partition of Mod-32

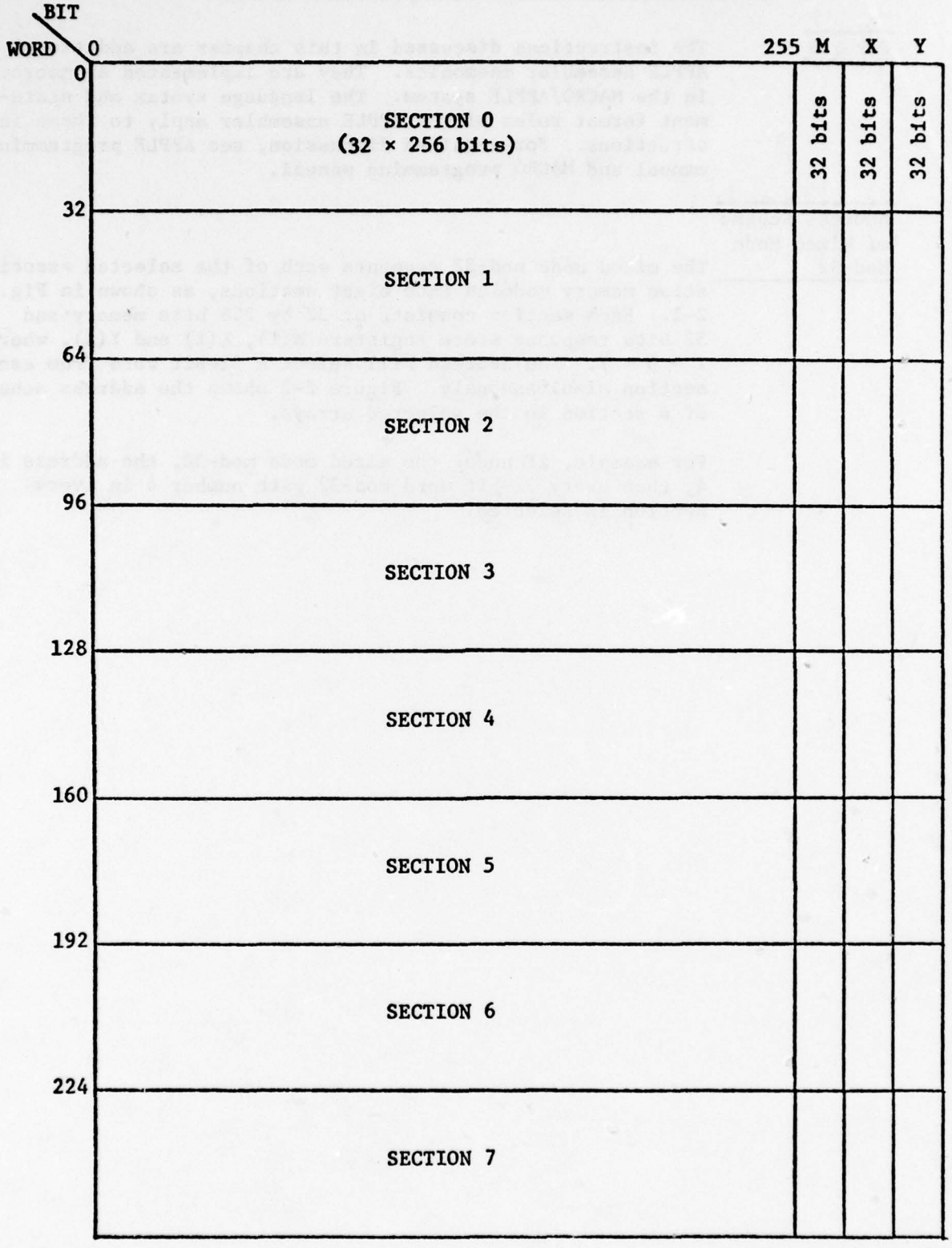


Fig. 2-2a Address Scheme in a Section of Mod-32

(Hexadecimal)

WORD \ BIT	0	32	64	96	128	160	192	224
0	0 0	2 0	4 0	6 0	8 0	A 0	C 0	E 0
1	0 1	2 1	4 1	6 1	8 1	A 1	C 1	E 1
2	0 2	2 2	4 2	6 2	8 2	A 2	C 2	E 2
3	0 3	2 3	4 3	6 3	8 3	A 3	C 3	E 3
4	0 4	2 4	4 4	6 4	8 4	A 4	C 4	E 4
5	0 5	2 5	4 5	6 5	8 5	A 5	C 5	E 5
6	0 6	2 6	4 6	6 6	8 6	A 6	C 6	E 6
7	0 7	2 7	4 7	6 7	8 7	A 7	C 7	E 7
8	0 8	2 8	4 8	6 8	8 8	A 8	C 8	E 8
9	0 9	2 9	4 9	6 9	8 9	A 9	C 9	E 9
10	0 A	2 A	4 A	6 A	8 A	A A	C A	E A
11	0 B	2 B	4 B	6 B	8 B	A B	C B	E B
12	0 C	2 C	4 C	6 C	8 C	A C	C C	E C
13	0 D	2 D	4 D	6 D	8 D	A D	C D	E D
14	0 E	2 E	4 E	6 E	8 E	A E	C E	E E
15	0 F	2 F	4 F	6 F	8 F	A F	C F	E F
16	1 0	3 0	5 0	7 0	9 0	B 0	D 0	F 0
17	1 1	3 1	5 1	7 1	9 1	B 1	D 1	F 1
18	1 2	3 2	5 2	7 2	9 2	B 2	D 2	F 2
19	1 3	3 3	5 3	7 3	9 3	B 3	D 3	F 3
20	1 4	3 4	5 4	7 4	9 4	B 4	D 4	F 4
21	1 5	3 5	5 5	7 5	9 5	B 5	D 5	F 5
22	1 6	3 6	5 6	7 6	9 6	B 6	D 6	F 6
23	1 7	3 7	5 7	7 7	9 7	B 7	D 7	F 7
24	1 8	3 8	5 8	7 8	9 8	B 8	D 8	F 8
25	1 9	3 9	5 9	7 9	9 9	B 9	D 9	F 9
26	1 A	3 A	5 A	7 A	9 A	B A	D A	F A
27	1 B	3 B	5 B	7 B	9 B	B B	D B	F B
28	1 C	3 C	5 C	7 C	9 C	B C	D C	F C
29	1 D	3 D	5 D	7 D	9 D	B D	D D	F D
30	1 E	3 E	5 E	7 E	9 E	B E	D E	F E
31	1 F	3 F	5 F	7 F	9 F	B F	D F	F F

Fig. 2-2b Address Scheme in a Section of Mod-32

(Decimal)

WORD \ BIT	0	32	64	96	128	160	192	224
0	0	32	64	96	128	160	192	224
1	1	33	65	97	129	161	193	225
2	2	34	66	98	130	162	194	226
3	3	35	67	99	131	163	195	227
4	4	36	68	100	132	164	196	228
5	5	37	69	101	133	165	197	229
6	6	38	70	102	134	166	198	230
7	7	39	71	103	135	167	199	231
8	8	40	72	104	136	168	200	232
9	9	41	73	105	137	169	201	233
10	10	42	74	106	138	170	202	234
11	11	43	75	107	139	171	203	235
12	12	44	76	108	140	172	204	236
13	13	45	77	109	141	173	205	237
14	14	46	78	110	142	174	206	238
15	15	47	79	111	143	175	207	239
16	16	48	80	112	144	176	208	240
17	17	49	81	113	145	177	209	241
18	18	50	82	114	146	178	210	242
19	19	51	83	115	147	179	211	243
20	20	52	84	116	148	180	212	244
21	21	53	85	117	149	181	213	245
22	22	54	86	118	150	182	214	246
23	23	55	87	119	151	183	215	247
24	24	56	88	120	152	184	216	248
25	25	57	89	121	153	185	217	249
26	26	58	90	122	154	186	218	250
27	27	59	91	123	155	187	219	251
28	28	60	92	124	156	188	220	252
29	29	61	93	125	157	189	221	253
30	30	62	94	126	158	190	222	254
31	31	63	95	127	159	191	223	255

Address Scheme of Mixed Mode Mod-64

The Mixed Mode Mode-64 segments the selected associative memory modules into four sections which is shown in Fig. 2-3. Each section consists of 64 by 256 bits memory and 64 bits response store registers M(i), X(i) and Y(i), where $i = 0 \sim 3$. The address will select a 64-bit word from each section simultaneously. Figure 2-4 shows the address scheme of a section in the selected arrays. For example, if under the mixed mode mode-64 the address is 3 then the 64-bit word mod-64 number 3 in each section is selected.

Fig. 2-3. Partition of Mod-64

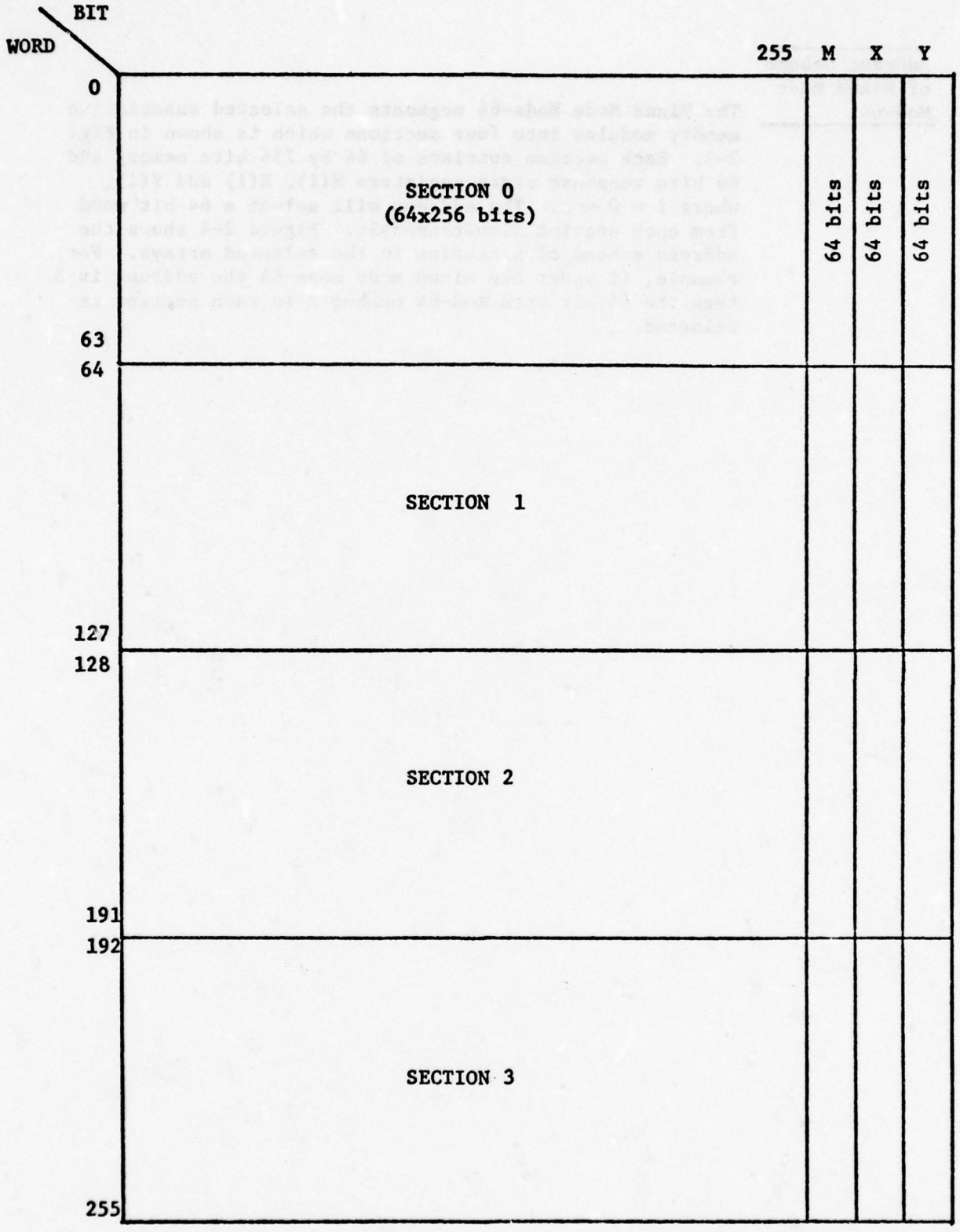


Fig. 2-4a. Address Scheme in a Section of Mixed Mode Mod-64 (Hexadecimal)

WORD \ BIT	0	64	128	192
0	0 0	4 0	8 0	C 0
1	0 1	4 1	8 1	C 1
2	0 2	4 2	8 2	C 2
3	0 3	4 3	8 3	C 3
4	0 4	4 4	8 4	C 4
5	0 5	4 5	8 5	C 5
6	0 6	4 6	8 6	C 6
7	0 7	4 7	8 7	C 7
8	0 8	4 8	8 8	C 8
9	0 9	4 9	8 9	C 9
10	0 A	4 A	8 A	C A
11	0 B	4 B	8 B	C B
12	0 C	4 C	8 C	C C
13	0 D	4 D	8 D	C D
14	0 E	4 E	8 E	C E
15	0 F	4 F	8 F	C F
16	1 0	5 0	9 0	D 0
17	1 1	5 1	9 1	D 1
18	1 2	5 2	9 2	D 2
19	1 3	5 3	9 3	D 3
20	1 4	5 4	9 4	D 4
21	1 5	5 5	9 5	D 5
22	1 6	5 6	9 6	D 6
23	1 7	5 7	9 7	D 7
24	1 8	5 8	9 8	D 8
25	1 9	5 9	9 9	D 9
26	1 A	5 A	9 A	D A
27	1 B	5 B	9 B	D B
28	1 C	5 C	9 C	D C
29	1 D	5 D	9 D	D D
30	1 E	5 E	9 E	D E
31	1 F	5 F	9 F	D F
32	2 0	6 0	A 0	E 0
33	2 1	6 1	A 1	E 1
34	2 2	6 2	A 2	E 2
35	2 3	6 3	A 3	E 3
36	2 4	6 4	A 4	E 4
37	2 5	6 5	A 5	E 5
38	2 6	6 6	A 6	E 6
39	2 7	6 7	A 7	E 7
40	2 8	6 8	A 8	E 8
41	2 9	6 9	A 9	E 9
42	2 A	6 A	A A	E A
43	2 B	6 B	A B	E B
44	2 C	6 C	A C	E C
45	2 D	6 D	A D	E D
46	2 E	6 E	A E	E E
47	2 F	6 F	A F	E F
48	3 0	7 0	B 0	F 0
49	3 1	7 1	B 1	F 1
50	3 2	7 2	B 2	F 2
51	3 3	7 3	B 3	F 3
52	3 4	7 4	B 4	F 4
53	3 5	7 5	B 5	F 5
54	3 6	7 6	B 6	F 6
55	3 7	7 7	B 7	F 7
56	3 8	7 8	B 8	F 8
57	3 9	7 9	B 9	F 9
58	3 A	7 A	B A	F A
59	3 B	7 B	B B	F B
60	3 C	7 C	B C	F C
61	3 D	7 D	B D	F D
62	3 E	7 E	B E	F E
63	3 F	7 F	B F	F F

Fig. 2-4b. Address Scheme in a Section of Mixed Mode Mod-64 (Decimal)

WORD \ BIT	0	64	128	192
0	0	64	128	192
1	1	65	129	193
2	2	66	130	194
3	3	67	131	195
4	4	68	132	196
5	5	69	133	197
6	6	70	134	198
7	7	71	135	199
8	8	72	136	200
9	9	73	137	201
10	10	74	138	202
11	11	75	139	203
12	12	76	140	204
13	13	77	141	205
14	14	78	142	206
15	15	79	143	207
16	16	80	144	208
17	17	81	145	209
18	18	82	146	210
19	19	83	147	211
20	20	84	148	212
21	21	85	149	213
22	22	86	150	214
23	23	87	151	215
24	24	88	152	216
25	25	89	153	217
26	26	90	154	218
27	27	91	155	219
28	28	92	156	220
29	29	93	157	221
30	30	94	158	222
31	31	95	159	223
32	32	96	160	224
33	33	97	161	225
34	34	98	162	226
35	35	99	163	227
36	36	100	164	228
37	37	101	165	229
38	38	102	166	230
39	39	103	167	231
40	40	104	168	232
41	41	105	169	233
42	42	106	170	234
43	43	107	171	235
44	44	108	172	236
45	45	109	173	237
46	46	110	174	238
47	47	111	175	239
48	48	112	176	240
49	49	113	177	241
50	50	114	178	242
51	51	115	179	243
52	52	116	180	244
53	53	117	181	245
54	54	118	182	246
55	55	119	183	247
56	56	120	184	248
57	57	121	185	249
58	58	122	186	250
59	59	123	187	251
60	60	124	188	252
61	61	125	189	253
62	62	126	190	254
63	63	127	191	255

Introduction to the Mixed Mode Routines

This group of associative instruction allows the programmer to perform arithmetic and search operations between associative memory words in mixed mode Mode-32/64 and between Common register with associative memory words in mixed mode Mod-32/64.

This group of instructions will operate on those associative memory modules (including response store registers) enabled via the Array Select register (AS).

Each instruction is implemented as a macro routine. Within the routine, FL1 is not used, therefore, each of these instructions can be called within either an LOOP or a RPT instruction.

The address mode register (AMR) is reset to the standard mode at the end of each instruction operations, so, the user can use this gorup of instructions with the standard APPLE instructions without considering the change of modes.

The mask response register M in each section must be either all set to 1's or all set to 0's. When the M register is set to 1's, this section will participate in the operation, otherwise, it won't.

INIT — Initiator of the mixed mode operations.

This instruction will load all the data which are required in utilizing the mixed mode operations in every array selected by the Array Select register(AS).

FORMAT

Label	Command	Argument
Symbol	INIT	Blank

- **Label** — Any valid symbol or blank.
- **Command** — INIT
- **Argument** — Blank

Note — The response store register X is used.

Mixed Mode Operations Initiator

This instruction will load the necessary data in the associative array selected to utilize the mixed mode operations.

Mnemonic	Function
INIT	Initiator of the Mixed Mode Operations.

Note

The array words mod-32 addresses from X'F4' to X'FF' are used for temporatory storage in some of the mixed mode instruction. The instruction INIT generates the required data before the mixed mode instructions are called, and the user programs should not alter the data in X'F4' through X'FF'.

Mixed Mode ADDITION and SUBTRACTION Instructions

This group of instructions allows the programmer to perform arithmetic addition and subtraction between words mod-32, and between the Common register with words mod-32. The most significant bits of all words mod-32 are considered to be the sign bits.

Mnemonic	Function
ADWM ADWMA*	Add Array Words Mod-32 to Array Words Mod-32 In Mixed Mode.
ADCM ADCMA*	Add Common Register to Array Words Mod-32 In Mixed Mode.
SUBWM SUBWMA*	Subtract Array Words Mod-32 From Array Words Mod-32 In Mixed Mode.
SUBCM SUBCMA*	Subtract Common Register From Array Words Mode-32 In Mixed Mode.

*In the "A" version of each of these routines the field pointer registers are loaded by the macro. In the other version, the programmer must preload the field pointer registers.

ADWM — **Add Array Words Mod-32 to Array Words Mod-32 In Mixed Mode (Fixed-Point)**

This instruction will add words mod-32 pointed to by FP3 to the words mod-32 pointed to by FP1 and store the resultant sums into the words mod-32 pointed to by FP2. The addresses are based on the Mixed Mode Mod-32, as shown in Fig. 2-1 and Fig. 2-2. Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

Format

Label	Command	Argument
Symbol	ADWM	Blank

. **Label** — Any valid symbol or blank.

. **Command** — ADWM

. **Argument** — Blank

Notes

(1) The response store registers X and Y in the associative memory modules which are selected by the Array Select register (AS) are used in this instruction.

(2) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address can not be used as an operand.

(3) The addresses of FP1, FP2, and FP3 can have the same values.

(4) Before this instruction is called, the registers AS, M, FP1, FP2 and FP3 must be set properly.

Example:

Assume AS = X'80000000'
FP3 = X'03'
FP1 = X'15'
FP2 = X'1E' and M is as shown in the tables then the instruction ADWM has the effect as shown in the following tables.

This instruction has the following operations:

(1) In Section 0, 19 + 34 = 53

(2) In Section 1, no operation

(3) In Section 2, no operation

(4) In Section 3, 6 + - 29 = - 23

(5) In Section 4, -10 + 18 = 8

(6) In Section 5, -13 + - 3 = - 16

(7) In Section 6, no operation

(8) In Section 7, 28 + 158 = 186

Note that the array memory locations which are not shown or are left blank are not changed.

	Word Addr. (Hex)	Before Execution Field (0 , 32) (Hex)	M	After Execution Field (0 , 32) (Hex)
Section 0	00		1	
	01		1	
	02		1	
	03	0 0 0 0 0 0 1 3	1	0 0 0 0 0 0 1 3
	04		1	
	05		1	
	06		1	
	07		1	
	08		1	
	09		1	
	0A		1	
	0B		1	
	0C		1	
	0D		1	
	0E		1	
	0F		1	
	10		1	
	11		1	
	12		1	
	13		1	
	14		1	
	15	0 0 0 0 0 0 2 2	1	0 0 0 0 0 0 2 2
	16		1	
	17		1	
	18		1	
	19		1	
	1A		1	
	1B		1	
	1C		1	
	1D		1	
	1E		1	0 0 0 0 0 0 3 5
	1F		1	
Section 1	20		0	
	21		0	
	22		0	
	23		0	
	24		0	
	25		0	
	26		0	
	27		0	
	28		0	
	29		0	
	2A		0	
	2B		0	
	2C		0	
	2D		0	
	2E		0	
	2F		0	
	30		0	
	31		0	
	32		0	
	33		0	
	34		0	
	35		0	
	36		0	
	37		0	
	38		0	
	39		0	
	3A		0	
	3B		0	
	3C		0	
	3D		0	
	3E		0	
	3F		0	

	Word Addr. (Hex)	Before Execution Field (0 , 32) (Hex)	M	After Execution Field (0 , 32) (Hex)
Section 2	40		0	
	41		0	
	42		0	
	43		0	
	44		0	
	45		0	
	46		0	
	47		0	
	48		0	
	49		0	
	4A		0	
	4B		0	
	4C		0	
	4D		0	
	4E		0	
	4F		0	
	50		0	
	51		0	
	52		0	
	53		0	
	54		0	
	55		0	
	56		0	
	57		0	
	58		0	
	59		0	
	5A		0	
	5B		0	
	5C		0	
	5D		0	
	5E		0	
	5F		0	
Section 3	60		1	
	61		1	
	62		1	
	63	0 0 0 0 0 0 0 6	1	0 0 0 0 0 0 0 6
	64		1	
	65		1	
	66		1	
	67		1	
	68		1	
	69		1	
	6A		1	
	6B		1	
	6C		1	
	6D		1	
	6E		1	
	6F		1	
	70		1	
	71		1	
	72		1	
	73		1	
	74		1	
	75	F F F F F F E 3	1	F F F F F F E 3
	76		1	
	77		1	
	78		1	
	79		1	
	7A		1	
	7B		1	
	7C		1	
	7D		1	
	7E		1	F F F F F F E 9
	7F		1	

	Word Addr. (Hex)	Before Execution Field (0 , 32) (Hex)	M	After Execution Field (0 , 32) (Hex)
Section 4	80		1	
	81		1	
	82		1	
	83	F F F F F F F 6	1	F F F F F F F 6
	84		1	
	85		1	
	86		1	
	87		1	
	88		1	
	89		1	
	8A		1	
	8B		1	
	8C		1	
	8D		1	
	8E		1	
	8F		1	
	90		1	
	91		1	
	92		1	
	93		1	
	94		1	
	95	0 0 0 0 0 0 1 2	1	0 0 0 0 0 0 1 2
	96		1	
	97		1	
	98		1	
	99		1	
	9A		1	
	9B		1	
	9C		1	
	9D		1	
	9E		1	0 0 0 0 0 0 0 8
	9F		1	
Section 5	A0		1	
	A1		1	
	A2		1	
	A3	F F F F F F F 3	1	F F F F F F F 3
	A4		1	
	A5		1	
	A6		1	
	A7		1	
	A8		1	
	A9		1	
	AA		1	
	AB		1	
	AC		1	
	AD		1	
	AE		1	
	AF		1	
	B0		1	
	B1		1	
	B2		1	
	B3		1	
	B4		1	
	B5	F F F F F F F D	1	F F F F F F F D
	B6		1	
	B7		1	
	B8		1	
	B9		1	
	BA		1	
	BB		1	
	BC		1	
	BD		1	
	BE		1	F F F F F F F 0
	BF		1	

	Word Addr. (Hex)	Before Execution Field (0 , 32) (Hex)	M	After Execution Field (0 , 32) (Hex)
Section 6	C0		0	
	C1		0	
	C2		0	
	C3		0	
	C4		0	
	C5		0	
	C6		0	
	C7		0	
	C8		0	
	C9		0	
	CA		0	
	CB		0	
	CC		0	
	CD		0	
	CE		0	
	CF		0	
	D0		0	
	D1		0	
	D2		0	
	D3		0	
	D4		0	
	D5		0	
	D6		0	
	D7		0	
	D8		0	
	D9		0	
	DA		0	
	DB		0	
	DC		0	
	DD		0	
	DE		0	
	DF		0	
Section 7	E0		1	
	E1		1	
	E2		1	
	E3	0 0 0 0 0 0 1 C	1	0 0 0 0 0 0 1 C
	E4		1	
	E5		1	
	E6		1	
	E7		1	
	E8		1	
	E9		1	
	EA		1	
	EB		1	
	EC		1	
	ED		1	
	EE		1	
	EF		1	
	F0		1	
	F1		1	
	F2		1	
	F3		1	
	F4		1	
	F5	0 0 0 0 0 0 9 E	1	0 0 0 0 0 0 9 E
	F6		1	
	F7		1	
	F8		1	
	F9		1	
	FA		1	
	FB		1	
	FC		1	
	FD		1	
	FE		1	0 0 0 0 0 0 B A
	FF		1	

ADWMA — **Add Array Words Mod-32 Array Words Mod-32 In Mixed Mode (Fixed-Point)**

This instruction will add words mod-32 pointed to by a_1 to the words mod-32 pointed to by a_2 and store the resultant sums into the words mod-32 pointed to by a_3. The addresses are based on the mixed mode mod-32, as shown in Fig. 2-1 and Fig. 2-2. Only those associative sections whose M response store register are set to 1's will participate in this instruction. Only those associative memory modulas which are selected by the Array Select register (AS) will participate in this instruction.

Format

Label	Command	Argument
Symbol	ADWMA	a_1,a_2,a_3

. **Label** — Any valid symbol or blank.

. **Command** — ADWMA

. **Argument** — Three entries are required. The first entry points to the words mod-32 which are the addends. The second entry points to the words mod-32 which are the adders. The third entry points to the words mod-32 which are the sums of the addends and the adders.

.. a_1,a_2,a_3 — a_1,a_2 and a_3 are the numbers between 0 and 255.

Notes

(1) This instruction is the same as the instruction ADWM, except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules which are selected by the AS register are used in this instruction.

(3) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

(4) The addresses of a_1,a_2 and a_3 can have the same values.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The field pointers FP1, FP2 and FP3 are used.

ADCM — **Add Common Register to Array Words Mod-32 In Mixed Mode (Fixed-Point)**

This instruction will add the 32-bit Common register to the words mod-32 pointed to by FP3 and store the resultant sums into words mod-32 pointed to by FP2. The addresses are based on the mixed mode mod-32, as shown in Fig. 2-1 and Fig. 2-2. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

Format

Label	Command	Argument
Symbol	ADCM	Blank

. **Label** — Any valid symbol or blank.

. **Command** — ADCM

. **Argument** — Blank

Notes

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address can not be used as an operand.

(3) The address of FP3 and FP2 can have the same values.

(4) Before this instruction is called, the registers AS, M, FP2 and FP3 must be set properly.

Example:

Assume AS = X'8000 0000'
FP3 = X'06'
FP2 = X'1C'
C = X'0000 0009' and M is as shown in the tables then the instruction ADCM has the effect as shown in the following tables.

This instruction has the following operations:
(1) In Section 0, 28 + 9 = 37
(2) In Section 1, no operation
(3) In Section 2, no operation
(4) In Section 3, -46 + 9 = -37
(5) In Section 4, 34 + 9 = 43
(6) In Section 5, no operation
(7) In Section 6, -5 + 9 = 4
(8) In Section 7, 13 + 9 = 22

Note that those memory locations which are not shown or are left blank are not changed.

	Word Addr. (Hex)	Before Execution Field (0 , 32) (Hex)	M	After Execution Field (0 , 32) (Hex)
Section 0	00		1	
	01		1	
	02		1	
	03		1	
	04		1	
	05		1	
	06	0 0 0 0 0 0 1 C	1	0 0 0 0 0 0 1 C
	07		1	
	08		1	
	09		1	
	0A		1	
	0B		1	
	0C		1	
	0D		1	
	0E		1	
	0F		1	
	10		1	
	11		1	
	12		1	
	13		1	
	14		1	
	15		1	
	16		1	
	17		1	
	18		1	
	19		1	
	1A		1	
	1B		1	
	1C		1	0 0 0 0 0 0 2 5
	1D		1	
	1E		1	
	1F		1	
Section 1	20		0	
	21		0	
	22		0	
	23		0	
	24		0	
	25		0	
	26		0	
	27		0	
	28		0	
	29		0	
	2A		0	
	2B		0	
	2C		0	
	2D		0	
	2E		0	
	2F		0	
	30		0	
	31		0	
	32		0	
	33		0	
	34		0	
	35		0	
	36		0	
	37		0	
	38		0	
	39		0	
	3A		0	
	3B		0	
	3C		0	
	3D		0	
	3E		0	
	3F		0	

	Word Addr. (Hex)	Before Execution Field (0 , 32) (Hex)	M	After Execution Field (0 , 32) (Hex)
Section 2	40		0	
	41		0	
	42		0	
	43		0	
	44		0	
	45		0	
	46		0	
	47		0	
	48		0	
	49		0	
	4A		0	
	4B		0	
	4C		0	
	4D		0	
	4E		0	
	4F		0	
	50		0	
	51		0	
	52		0	
	53		0	
	54		0	
	55		0	
	56		0	
	57		0	
	58		0	
	59		0	
	5A		0	
	5B		0	
	5C		0	
	5D		0	
	5E		0	
	5F		0	
Section 3	60		1	
	61		1	
	62		1	
	63		1	
	64		1	
	65		1	
	66	F F F F F F D 2	1	F F F F F F D 2
	67		1	
	68		1	
	69		1	
	6A		1	
	6B		1	
	6C		1	
	6D		1	
	6E		1	
	6F		1	
	70		1	
	71		1	
	72		1	
	73		1	
	74		1	
	75		1	
	76		1	
	77		1	
	78		1	
	79		1	
	7A		1	
	7B		1	
	7C		1	F F F F F F D B
	7D		1	
	7E		1	
	7F		1	

	Word Addr. (Hex)	Before Execution Field (0 , 32) (Hex)	M	After Execution Field (0 , 32) (Hex)
Section 4	80		1	
	81		1	
	82		1	
	83		1	
	84		1	
	85		1	
	86	0 0 0 0 0 0 2 2	1	0 0 0 0 0 0 2 2
	87		1	
	88		1	
	89		1	
	8A		1	
	8B		1	
	8C		1	
	8D		1	
	8E		1	
	8F		1	
	90		1	
	91		1	
	92		1	
	93		1	
	94		1	
	95		1	
	96		1	
	97		1	
	98		1	
	99		1	
	9A		1	
	9B		1	
	9C		1	0 0 0 0 0 0 2 B
	9D		1	
	9E		1	
	9F		1	
Section 5	A0		0	
	A1		0	
	A2		0	
	A3		0	
	A4		0	
	A5		0	
	A6		0	
	A7		0	
	A8		0	
	A9		0	
	AA		0	
	AB		0	
	AC		0	
	AD		0	
	AE		0	
	AF		0	
	B0		0	
	B1		0	
	B2		0	
	B3		0	
	B4		0	
	B5		0	
	B6		0	
	B7		0	
	B8		0	
	B9		0	
	BA		0	
	BB		0	
	BC		0	
	BD		0	
	BE		0	
	BF		0	

	Word Addr. (Hex)	Before Execution Field (0 , 32) (Hex)	M	After Execution Field (0 , 32) (Hex)
Section 6	C0		1	
	C1		1	
	C2		1	
	C3		1	
	C4		1	
	C5		1	
	C6	F F F F F F F B	1	F F F F F F F B
	C7		1	
	C8		1	
	C9		1	
	CA		1	
	CB		1	
	CC		1	
	CD		1	
	CE		1	
	CF		1	
	D0		1	
	D1		1	
	D2		1	
	D3		1	
	D4		1	
	D5		1	
	D6		1	
	D7		1	
	D8		1	
	D9		1	
	DA		1	
	DB		1	
	DC		1	0 0 0 0 0 0 0 4
	DD		1	
	DE		1	
	DF		1	
Section 7	E0		1	
	E1		1	
	E2		1	
	E3		1	
	E4		1	
	E5		1	
	E6	0 0 0 0 0 0 0 D	1	0 0 0 0 0 0 0 D
	E7		1	
	E8		1	
	E9		1	
	EA		1	
	EB		1	
	EC		1	
	ED		1	
	EE		1	
	EF		1	
	F0		1	
	F1		1	
	F2		1	
	F3		1	
	F4		1	
	F5		1	
	F6		1	
	F7		1	
	F8		1	
	F9		1	
	FA		1	
	FB		1	
	FC		1	0 0 0 0 0 0 1 6
	FD		1	
	FE		1	
	FF		1	

ADCMA — **Add Common Register to Array Words Mod-32 In Mixed Mode (Fixed-Point)**

This instruction will add the 32-bit Common register to the words mod-32 pointed to by a_1, and store the resultant sums into words mod-32 pointed to by a_2. The addresses are based on the mixed mode mod-32, as shown in Fig. 2-1 and Fig. 2-2. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

Format

Label	Command	Argument
Symbol	ADCMA	a_1, a_2

. **Label** — Any valid symbol or blank.

. **Command** — ADCMA

. **Argument** — Two entries are required. The first entry points to the words mod-32 which are the addends. The second entry points to the words mod-32 which are the sums of the addends and the Common register.

.. a_1,a_2 — a_1 and a_2 are the numbers between 0 and 255.

Notes

(1) This instruction is the same as the instruction ADCM, except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address can not be used as an operand.

(4) The addresses of a_1 and a_2 can have the same values.

(5) Before this instruction is called, the registers AS and M must be set properly.

(6) The field pointers FP3 and FP1 are used.

SUBWM Subtract Array Words Mod-32 From Array Words Mod-32 In Mixed Mode (Fixed-Point)

This instruction will subtract the words mod-32 pointed to by FP1 from the words mod-32 pointed to by FP3 and store the resultant differences into the words mod-32 pointed to by FP2. The addresses are based on the mixed mode mod-32, as shown in Fig. 2-1 and Fig. 2-2. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

Format

Label	Command	Argument
Symbol	SUBWM	Blank

. **Label** Any valid symbol or blank.

. **Command** SUBWM

. **Argument** Blank

Notes

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address can not be used as an operand.

(3) The addresses of FP1, FP2 and FP3 can have the same values.

(4) Before this instruction is called, the registers AS, M, FP1, FP2 and FP3 must be set properly.

Example

Assume AS = X'80000000'
FP3 = X'03'
FP1 = X'09'
FP2 = X'0C'

and M is as shown in the tables, then the instruction SUBWM has the effect as shown in the following tables.

This instruction has the following operations:

(1) In Section 0, 39 - 12 = 27
(2) In Section 1, 63 - -10 = 73
(3) In Section 2, no operation
(4) In Section 3, -14 - 4 = -18

(5) In Section 4, $-13 - \bar{2} = -11$
(6) In Section 5, no operation
(7) In Section 6, $-10 - \bar{24} = 14$
(8) In Section 7, no operation

Note that those associative memory locations which are not shown or are left blank are not changed.

	Word Addr. (Hex)	Before Execution Field (0,32) (Hex)	M	After Execution Field (0,32) (Hex)
Section 0	00		1	
	01		1	
	02		1	
	03	0000 0027	1	0000 0027
	04		1	
	05		1	
	06		1	
	07		1	
	08		1	
	09	0000 000C	1	0000 000C
	0A		1	
	0B		1	
	0C		1	0000 001B
	0D		1	
	0E		1	
	0F		1	
	10		1	
	11		1	
	12		1	
	13		1	
	14		1	
	15		1	
	16		1	
	17		1	
	18		1	
	19		1	
	1A		1	
	1B		1	
	1C		1	
	1D		1	
	1E		1	
	1F		1	
Section 1	20		1	
	21		1	
	22		1	
	23	0000 003F	1	0000 003F
	24		1	
	25		1	
	26		1	
	27		1	
	28		1	
	29	FFFF FFF6	1	FFFF FFF6
	2A		1	
	2B		1	
	2C		1	
	2D		1	0000 0049
	2E		1	
	2F		1	
	30		1	
	31		1	
	32		1	
	33		1	
	34		1	
	35		1	
	36		1	
	37		1	
	38		1	
	39		1	
	3A		1	
	3B		1	
	3C		1	
	3D		1	
	3E		1	
	3F		1	

	Word Addr. (Hex)	Before Execution Field (0,32) (Hex)	M	After Execution Field (0,32) (Hex)
Section 2	40		0	
	41		0	
	42		0	
	43		0	
	44		0	
	45		0	
	46		0	
	47		0	
	48		0	
	49		0	
	4A		0	
	4B		0	
	4C		0	
	4D		0	
	4E		0	
	4F		0	
	50		0	
	51		0	
	52		0	
	53		0	
	54		0	
	55		0	
	56		0	
	57		0	
	58		0	
	59		0	
	5A		0	
	5B		0	
	5C		0	
	5D		0	
	5E		0	
	5F		0	
Section 3	60		1	
	61		1	
	62		1	
	63	FFFF FFF2	1	FFFF FFF2
	64		1	
	65		1	
	66		1	
	67		1	
	68		1	
	69	0000 0004	1	0000 0004
	6A		1	
	6B		1	
	6C		1	FFFF FFEE
	6D		1	
	6E		1	
	6F		1	
	70		1	
	71		1	
	72		1	
	73		1	
	74		1	
	75		1	
	76		1	
	77		1	
	78		1	
	79		1	
	7A		1	
	7B		1	
	7C		1	
	7D		1	
	7E		1	
	7F		1	

		Before Execution		After Execution
	Word Addr. (Hex)	Field (0,32) (Hex)	M	Field (0,32) (Hex)
Section 4	80		1	
	81		1	
	82		1	
	83	FFFF FFF3	1	FFFF FFF3
	84		1	
	85		1	
	86		1	
	87		1	
	88		1	
	89	FFFF FFFE	1	FFFF FFFE
	8A		1	
	8B		1	
	8C		1	FFFF FFFS
	8D		1	
	8E		1	
	8F		1	
	90		1	
	91		1	
	92		1	
	93		1	
	94		1	
	95		1	
	96		1	
	97		1	
	98		1	
	99		1	
	9A		1	
	9B		1	
	9C		1	
	9D		1	
	9E		1	
	9F		1	
Section 5	A0		0	
	A1		0	
	A2		0	
	A3		0	
	A4		0	
	A5		0	
	A6		0	
	A7		0	
	A8		0	
	A9		0	
	AA		0	
	AB		0	
	AC		0	
	AD		0	
	AE		0	
	AF		0	
	B0		0	
	B1		0	
	B2		0	
	B3		0	
	B4		0	
	B5		0	
	B6		0	
	B7		0	
	B8		0	
	B9		0	
	BA		0	
	BB		0	
	BC		0	
	BD		0	
	BE		0	
	BF		0	

	Word Addr. (Hex)	Before Execution Field (0,32) (Hex)	M	After Execution Field (0,32) (Hex)
Section 6	C0		1	
	C1		1	
	C2		1	
	C3	FFFF FFF6	1	FFFF FFF6
	C4		1	
	C5		1	
	C6		1	
	C7		1	
	C8		1	
	C9	FFFF FFE8	1	FFFF FFE8
	CA		1	
	CB		1	
	CC		1	0000 000E
	CD		1	
	CE		1	
	CF		1	
	D0		1	
	D1		1	
	D2		1	
	D3		1	
	D4		1	
	D5		1	
	D6		1	
	D7		1	
	D8		1	
	D9		1	
	DA		1	
	DB		1	
	DC		1	
	DD		1	
	DE		1	
	DF		1	
Section 7	E0		0	
	E1		0	
	E2		0	
	E3		0	
	E4		0	
	E5		0	
	E6		0	
	E7		0	
	E8		0	
	E9		0	
	EA		0	
	EB		0	
	EC		0	
	ED		0	
	EE		0	
	EF		0	
	F0		0	
	F1		0	
	F2		0	
	F3		0	
	F4		0	
	F5		0	
	F6		0	
	F7		0	
	F8		0	
	F9		0	
	FA		0	
	FB		0	
	FC		0	
	FD		0	
	FE		0	
	FF		0	

SUBWMA Subtract Array Words Mod-32 From Array Words Mod-32 In Mixed Mode (Fixed-Point)

This instruction will subtract the words mod-32 pointed to by a_2 from the words mod-32 pointed to by a_1 and store the resultant differences into the words mod-32 pointed to by a_3. The addresses are based on the mixed mode mod-32, as shown in the Fig. 2-1, and Fig. 2-2. Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

Format

Label	Command	Argument
Symbol	SUBWMA	a_1,a_2,a_3

. **Label** Any valid symbol or blank.

. **Command** SUBWMA

. **Argument** Three entries are required. The first entry points to the words mod-32 which are the minuends. The second entry points to the words mod-32 which are the subtrahends. The third entry points to the words mod-32 which are the differences of the minuends subtract the subtrahends.

.. a_1,a_2,a_3 a_1, a_2, a_3 are the numbers between 0 and 255.

Notes

(1) This instruction is the same as the instruction SUBWM except that it loads the arguments to the full pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address can not be used as an operand.

(4) The addresses of a_1, a_2 and a_3 can have the same values.

(5) Before this instruction is called, the registers AS and M must be set properly.

(6) The field pointers FP1, FP2, and FP3 are used.

SUBCM — **Subtract Common Register From Array Words Mod-32 In Mixed Mode (Fixed-Point)**

This instruction will subtract the 32-bit Common Register from the words mod-32 pointed to by FP3 and store the resultant differences into the words mod-32 pointed to by FP2. The addresses are based on the mixed mode mod-32, as shown in Fig. 2-1 and Fig. 2-2. Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

Format

Label	Command	Arugment
Symbol	SUBCM	Blank

- **Label** — Any valid symbol or blank.
- **Command** — SUBCM
- **Argument** — Blank

Notes

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

(3) The addresses of FP3 and FP2 can have the same values.

(4) Before the instruction is called, the registers AS, M, FP2 and FP3 must be set properly.

Example:

Assume AS = X'8000 0000'
FP2 = X'04'
FP3 = X'10'
C = X'000 0013'

and M is as shown in the tables, then the instruction SUBCM has the effect as shown in the following tables.

This instruction has the following operations:

(1) In Section 0, 71 - 19 = 52
(2) In Section 1, -2 - 19 = -21
(3) In Section 2, no operation
(4) In Section 3, 8 - 19 = -11
(5) In Section 4, -23 - 19 = -42

(6) In Section 5, no operation
(7) In Section 6, no operation
(8) In Section 7, no operation

Note that those memory locations which are not shown or are left blank are not changed.

	Word Addr. (Hex)	Before Execution Field (0,32) (Hex)	M	After Execution Field (0,32) (Hex)
Section 0	00		1	
	01		1	
	02		1	
	03		1	
	04	0000 0047	1	0000 0047
	05		1	
	06		1	
	07		1	
	08		1	
	09		1	
	0A		1	
	0B		1	
	0C		1	
	0D		1	
	0E		1	
	0F		1	
	10		1	0000 0034
	11		1	
	12		1	
	13		1	
	14		1	
	15		1	
	16		1	
	17		1	
	18		1	
	19		1	
	1A		1	
	1B		1	
	1C		1	
	1D		1	
	1E		1	
	1F		1	
Section 1	20		1	
	21		1	
	22		1	
	23		1	
	24	FFFF FFFE	1	FFFF FFFE
	25		1	
	26		1	
	27		1	
	28		1	
	29		1	
	2A		1	
	2B		1	
	2C		1	
	2D		1	
	2E		1	
	2F		1	
	30		1	FFFF FFEB
	31		1	
	32		1	
	33		1	
	34		1	
	35		1	
	36		1	
	37		1	
	38		1	
	39		1	
	3A		1	
	3B		1	
	3C		1	
	3D		1	
	3E		1	
	3F		1	

	Word Addr. (Hex)	Before Execution: Field (0,32) (Hex)	M	After Execution: Field (0,32) (Hex)
Section 2	40		0	
	41		0	
	42		0	
	43		0	
	44		0	
	45		0	
	46		0	
	47		0	
	48		0	
	49		0	
	4A		0	
	4B		0	
	4C		0	
	4D		0	
	4E		0	
	4F		0	
	50		0	
	51		0	
	52		0	
	53		0	
	54		0	
	55		0	
	56		0	
	57		0	
	58		0	
	59		0	
	5A		0	
	5B		0	
	5C		0	
	5D		0	
	5E		0	
	5F		0	
Section 3	60		1	
	61		1	
	62		1	
	63		1	
	64	0000 0008	1	0000 0008
	65		1	
	66		1	
	67		1	
	68		1	
	69		1	
	6A		1	
	6B		1	
	6C		1	
	6D		1	
	6E		1	
	6F		1	
	70		1	
	71		1	
	72		1	FFFF FFF5
	73		1	
	74		1	
	75		1	
	76		1	
	77		1	
	78		1	
	79		1	
	7A		1	
	7B		1	
	7C		1	
	7D		1	
	7E		1	
	7F		1	

	Word Addr. (Hex)	Before Execution Field (0,32) (Hex)	M	After Execution Field (0,32) (Hex)
Section 4	80		1	
	81		1	
	82		1	
	83		1	
	84	FFFF FFE9	1	FFFF FFE9
	85		1	
	86		1	
	87		1	
	88		1	
	89		1	
	8A		1	
	8B		1	
	8C		1	
	8D		1	
	8E		1	
	8F		1	
	90		1	FFFF FFD6
	91		1	
	92		1	
	93		1	
	94		1	
	95		1	
	96		1	
	97		1	
	98		1	
	99		1	
	9A		1	
	9B		1	
	9C		1	
	9D		1	
	9E		1	
	9F		1	
Section 5	A0		0	
	A1		0	
	A2		0	
	A3		0	
	A4		0	
	A5		0	
	A6		0	
	A7		0	
	A8		0	
	A9		0	
	AA		0	
	AB		0	
	AC		0	
	AD		0	
	AE		0	
	AF		0	
	B0		0	
	B1		0	
	B2		0	
	B3		0	
	B4		0	
	B5		0	
	B6		0	
	B7		0	
	B8		0	
	B9		0	
	BA		0	
	BB		0	
	BC		0	
	BD		0	
	BE		0	
	BF		0	

	Word Addr. (Hex)	Before Execution: Field (0,32) (Hex)	M	After Execution: Field (0,32) (Hex)
Section 6	C0		0	
	C1		0	
	C2		0	
	C3		0	
	C4		0	
	C5		0	
	C6		0	
	C7		0	
	C8		0	
	C9		0	
	CA		0	
	CB		0	
	CC		0	
	CD		0	
	CE		0	
	CF		0	
	D0		0	
	D1		0	
	D2		0	
	D3		0	
	D4		0	
	D5		0	
	D6		0	
	D7		0	
	D8		0	
	D9		0	
	DA		0	
	DB		0	
	DC		0	
	DD		0	
	DE		0	
	DF		0	
Section 7	E0		0	
	E1		0	
	E2		0	
	E3		0	
	E4		0	
	E5		0	
	E6		0	
	E7		0	
	E8		0	
	E9		0	
	EA		0	
	EB		0	
	EC		0	
	ED		0	
	EE		0	
	EF		0	
	F0		0	
	F1		0	
	F2		0	
	F3		0	
	F4		0	
	F5		0	
	F6		0	
	F7		0	
	F8		0	
	F9		0	
	FA		0	
	FB		0	
	FC		0	
	FD		0	
	FE		0	
	FF		0	

SUBCMA — **Subtract Common Register From Array Words Mod-32 In Mixed Mode (Fixed-Point)**

This instruction will subtract the 32-bit Common register from the words mod-32 pointed to by a_1 and store the resultant differences into the words mod-32 pointed to by a_2. The addresses are based on the mixed mode mod-32, as shown in Fig. 2-1 and Fig. 2-2. Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

Format

Label	Command	Argument
Symbol	SUBCMA	a_1, a_2

. **Label** — Any valid symbol or blank.

. **Command** — SUBCMA

. **Argument** — Two entries are required. The first entry points to the words mod-32 which are the minuends. The second entry points to the words mod-32 which are the differences of the minuends minus the Common register.

.. a_1,a_2 — a_1 and a_2 are the numbers between 0 and 255.

Notes

(1) This instruction is the same as the instruction SUBCM except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

(4) The addresses of a_1 and a_2 can have the same values.

(5) Before this instruction is called, the registers AS and M must be set properly.

(6) The field pointers FP3 and FP1 are used.

Mixed Mode MULTIPLICATION and DIVISION Instructions

This group of instructions allows the programmer to perform arithmetic multiplication and division between the upper/lower half of words mod-64, and between the Common register with the upper/lower half of words mod-64. The most significant bits of all 32-bit words are considered to be the sign bits.

Mnemonic	Function
MPUUM MPUUMA MPULM MPULMA MPLUM MPLUMA MPLLM MPLLMA	Multiply words by words in mixed mode mod-64.
MPCUM MPCUMA MPCLM MPCLMA	Multiply words by the Common register in mixed mode mod-64.
DVUUM DVUUMA DVULM DVULMA DVLUM DVLUMA DVLLM DVLLMA	Divide words by words in mixed mode mod-64.
DVCUM DVCUMA DVCLM DVCLMA	Divide words by the Common register in mixed mode mod-64.

MPUUM Multiply 32-bit words by 32-bit words in mixed mode mod-64 (fixed-point)

This instruction will multiply the upper 32 bits of the words mod-64 pointed to by FP3 by the upper 32 bits of the words mod-64 pointed to by FP1 and store the 64-bit resultant products into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64. (Fig.2-3 and Fig. 2-4). Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPUUM	Blank

- **LABEL** Any valid symbol or blank.
- **COMMAND** MPUUM
- **ARGUMENT** Blank

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The following addresses in the array sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(3) The addresses of FP1, FP2 and FP3 can have the same values.

(4) Before the instruction is called, the registers AS, M, FP1, FP2 and FP3 must be set properly.

(5) The addresses X'F4', X'F5', X'F7', X'F8', X'F9', X'FA', and X'FC' in the sections which participate in the instruction are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

EXAMPLE: Assume AS = X'8000 0000'
FP3 = X'04'
FP1 = X'19'
FP2 = X'2B'
and M is as shown in the tables, then the instruction MPUUM has the effect as shown in the following tables.

Note that those memory locations which are not shown or are left blank are not changed. This instruction has the following operations:
(1) In Section 0, 7 x 5 = 35
(2) In Section 1, no operation
(3) In Section 2, 7 x -8 = -56
(4) In Section 3, -6 x -15 = 90

	Word Addr. (Hex)	Before Execution: Field (0 ,32) (Hex)	Before Execution: Field (32 ,32) (Hex)	M	After Execution: Field (0 ,32) (Hex)	After Execution: Field (32 , 32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03			1		
	04	0000 0007		1	0000 0007	
	05			1		
	06			1		
	07			1		
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1		
	11			1		
	12			1		
	13			1		
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19	0000 0005		1	0000 0005	
	1A			1		
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1		
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1		
	2B			1	0000 0000	0000 0023
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1		
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32, 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 1	40			0		
	41			0		
	42			0		
	43			0		
	44			0		
	45			0		
	46			0		
	47			0		
	48			0		
	49			0		
	4A			0		
	4B			0		
	4C			0		
	4D			0		
	4E			0		
	4F			0		
	50			0		
	51			0		
	52			0		
	53			0		
	54			0		
	55			0		
	56			0		
	57			0		
	58			0		
	59			0		
	5A			0		
	5B			0		
	5C			0		
	5D			0		
	5E			0		
	5F			0		
	60			0		
	61			0		
	62			0		
	63			0		
	64			0		
	65			0		
	66			0		
	67			0		
	68			0		
	69			0		
	6A			0		
	6B			0		
	6C			0		
	6D			0		
	6E			0		
	6F			0		
	70			0		
	71			0		
	72			0		
	73			0		
	74			0		
	75			0		
	76			0		
	77			0		
	78			0		
	79			0		
	7A			0		
	7B			0		
	7C			0		
	7D			0		
	7E			0		
	7F			0		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 ,32) (Hex)	Field (32 ,32) (Hex)
Section 2	80			1		
	81			1		
	82			1		
	83			1		
	84	0 0 0 0 0 0 0 0 7		1	0 0 0 0 0 0 0 7	
	85			1		
	86			1		
	87			1		
	88			1		
	89			1		
	8A			1		
	8B			1		
	8C			1		
	8D			1		
	8E			1		
	8F			1		
	90			1		
	91			1		
	92			1		
	93			1		
	94			1		
	95			1		
	96			1		
	97			1		
	98			1		
	99	F F F F F F F 8		1	F F F F F F F 8	
	9A			1		
	9B			1		
	9C			1		
	9D			1		
	9E			1		
	9F			1		
	A0			1		
	A1			1		
	A2			1		
	A3			1		
	A4			1		
	A5			1		
	A6			1		
	A7			1		
	A8			1		
	A9			1		
	AA			1		
	AB			1	F F F F F F F F	F F F F F F C 8
	AC			1		
	AD			1		
	AE			1		
	AF			1		
	B0			1		
	B1			1		
	B2			1		
	B3			1		
	B4			1		
	B5			1		
	B6			1		
	B7			1		
	B8			1		
	B9			1		
	BA			1		
	BB			1		
	BC			1		
	BD			1		
	BE			1		
	BF			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32, 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3			1		
	C4	FFFF FFFA		1	FFFF FFFA	
	C5			1		
	C6			1		
	C7			1		
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1		
	D1			1		
	D2			1		
	D3			1		
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9	FFFF FFF1		1	FFFF FFF1	
	DA			1		
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1		
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1	00000000	0000005A
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			1		
	F6			1		
	F7			1		
	F8			1		
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

MPUUMA — Multiply 32-bit words by 32-bit words in mixed mode mod-64 (fixed-point)

This instruction will multiply the upper 32 bits of the words mod-64 pointed to by a1 by the upper 32 bits of the words mod-64 pointed to by a2 and store the 64-bit resultant products into the words mod-64 pointed to by a3. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPUUMA	a1, a2, a3

- LABEL — Any valid symbol or blank.
- COMMAND — MPUUMA
- ARGUMENT — Three entries are required. The first entry points to the words mod-64 whose upper 32 bits are the multiplicands. The second entry points to the words mod-64 whose upper 32 bits are the multipliers. The third entry points to the words mod-64 which are the 64-bit products of the multiplicands multiply the multipliers.

•• a1, a2, a3 — a1,a2 and a3 are the numbers between 0 and 255.

NOTES

(1) This instruction is the same as the instruction MPUUM except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in the array sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(4) The addresses of a1, a2, a3 can have the same values.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The field pointers FP1, FP2 and FP3 are used.

(7) The addresses X'F4', X'F5', X'F7', X'F8', X'F9', X'FA' and X'FC' in the sections which participate in the instruction are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

MPULM Multiply 32-bit words by 32-bit words in mixed mode mod-32 (Fixed-Point)

This instruction will multiply the upper 32 bits of the words mod-64 pointed to by FP3 by the lower 32 bits of the words mod-64 pointed to by FP1 and store the 64-bit resultant products into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register(AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPULM	Blank

.Label Any valid symbol or blank.

.Command MPULM

.Argument Blank

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register(AS) are used.

(2) The following addresses in the array sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(3) The addresses of FP1, FP2 and FP3 can have the same values.

(4) Before the instruction is called, the registers AS, M, FP1, FP2 and FP3 must be set properly.

(5) The addresses X'F4', X'F5', X'F7', X'F8', X'F9', X'FA' and X'FC' in the sections which participate in the instruction are used for temporate storages. If any of these addresses is used as an operand, the content will be destroyed.

EXAMPLE

Assume AS = X'8000 0000'
FP3 = X'04'
FP1 = X'19'
FP2 = X'2B'
and M is as shown in the tables, the the instruction MPULM has the following operations:

(1) In section 0, 7 x 5 = 35
(2) In section 1, No Operation.
(3) In section 2, 7 x -8 = -56
(4) In section 3, -6 x -15 = 90

Note that the associative memory locations which are not shown or are left blank are not changed in this instruction.

	Word Addr. (Hex)	Before Execution: Field (0 , 32) (Hex)	Before Execution: Field (32, 32) (Hex)	M	After Execution: Field (0 ,32) (Hex)	After Execution: Field (32,32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03			1		
	04	0 0 0 0 C 0 0 7		1	0 0 0 0 0 0 0 7	
	05			1		
	06			1		
	07			1		
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1		
	11			1		
	12			1		
	13			1		
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19		0 0 0 0 0 0 0 5	1		0 0 0 0 0 0 0 5
	1A			1		
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1		
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1		
	2B			1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 2 3
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1		
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

	Word Addr. (Hex)	Before Execution: Field (0 ,32) (Hex)	Before Execution: Field (32, 32) (Hex)	M	After Execution: Field (0 , 32) (Hex)	After Execution: Field (32 , 32) (Hex)
Section 1	40			0		
	41			0		
	42			0		
	43			0		
	44			0		
	45			0		
	46			0		
	47			0		
	48			0		
	49			0		
	4A			0		
	4B			0		
	4C			0		
	4D			0		
	4E			0		
	4F			0		
	50			0		
	51			0		
	52			0		
	53			0		
	54			0		
	55			0		
	56			0		
	57			0		
	58			0		
	59			0		
	5A			0		
	5B			0		
	5C			0		
	5D			0		
	5E			0		
	5F			0		
	60			0		
	61			0		
	62			0		
	63			0		
	64			0		
	65			0		
	66			0		
	67			0		
	68			0		
	69			0		
	6A			0		
	6B			0		
	6C			0		
	6D			0		
	6E			0		
	6F			0		
	70			0		
	71			0		
	72			0		
	73			0		
	74			0		
	75			0		
	76			0		
	77			0		
	78			0		
	79			0		
	7A			0		
	7B			0		
	7C			0		
	7D			0		
	7E			0		
	7F			0		

	Word Addr. (Hex)	Before Execution: Field (0 , 32) (Hex)	Before Execution: Field (32 , 32) (Hex)	M	After Execution: Field (0 , 32) (Hex)	After Execution: Field (32 , 32) (Hex)
Section 2	80			1		
	81			1		
	82			1		
	83			1		
	84	0000 0007		1	0000 0007	
	85			1		
	86			1		
	87			1		
	88			1		
	89			1		
	8A			1		
	8B			1		
	8C			1		
	8D			1		
	8E			1		
	8F			1		
	90			1		
	91			1		
	92			1		
	93			1		
	94			1		
	95			1		
	96			1		
	97			1		
	98			1		
	99		FFFF FFF8	1		FFFF FFF8
	9A			1		
	9B			1		
	9C			1		
	9D			1		
	9E			1		
	9F			1		
	A0			1		
	A1			1		
	A2			1		
	A3			1		
	A4			1		
	A5			1		
	A6			1		
	A7			1		
	A8			1		
	A9			1		
	AA			1		
	AB			1	FFFF FFFF	FFFFFFC8
	AC			1		
	AD			1		
	AE			1		
	AF			1		
	B0			1		
	B1			1		
	B2			1		
	B3			1		
	B4			1		
	B5			1		
	B6			1		
	B7			1		
	B8			1		
	B9			1		
	BA			1		
	BB			1		
	BC			1		
	BD			1		
	BE			1		
	BF			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 ,32) (Hex)	Field (32,32) (Hex)	M	Field (0 ,32) (Hex)	Field (32, 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3			1		
	C4	FFFF FFFA		1	FFFF FFFA	
	C5			1		
	C6			1		
	C7			1		
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1		
	D1			1		
	D2			1		
	D3			1		
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9		FFFF FFF1	1		FFFF FFF1
	DA			1		
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1		
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1	0000 0000	0000 005A
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			1		
	F6			1		
	F7			1		
	F8			1		
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

MPULMA — Multiply 32-bit words by 32-bit words in mixed mode mod-64 (fixed-point)

This instruction will multiply the upper 32 bits of the words mod-64 pointed to by a1, by the lower 32 bits of the words mod-64 pointed to by a2 and store the 64-bit resultant products into the words mod-64 pointed to by a3. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPULMA	a1, a2, a3

- LABEL — Any valid symbol or blank.
- COMMAND — MPULMA
- ARGUMENT — Three entries are required. The first entry points to the words mod-64 whose upper 32 bits are the multiplicands. The second entry points to the words mod-64 whose lower 32 bits are the multipliers. The third entry points to the words mod-64 which are the 64-bit products of the multiplicands multiply the multipliers.
  - a1, a2, a3 — a1, a2, and a3 are the numbers between 0 and 255.

NOTES

(1) This instruction is the same as the instruction MPULM, except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in the array sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(4) The addresses of a1, a2 and a3 can have the same values.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The field pointers FP1, FP2, and FP3 are used.

(7) The addresses X'F4', X'F5', X'F7', X'F8', X'F9', X'FA' and X'FC' in the sections which participate in the instruction are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

MPLUM — Multiply 32-bit words by 32-bit words in mixed mode mod-64 (Fixed-Point)

This instruction will multiply the lower 32 bits of the words mod-64 pointed to by FP3 by the upper 32 bits of the words mod-64 pointed to by FP1 and store the 64-bit resultant products into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64. (Fig. 2-3 and Fig. 2-4). Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which a.e selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPLUM	Blank

- LABEL — Any valid symbol or blank.
- COMMAND — MPLUM
- ARGUMENT — Blank.

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The following addresses in the array sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(3) The addresses of FP1, FP2 and FP3 can have the same values.

(4) Before the instruction is called, the registers AS, M, FP1, FP2 and FP3 must be set properly.

(5) The addresses X'F4', X'F5', X'F7', X'F8', X'F9', X'FA', and X'FC' in the sections which participate in the instruction are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

EXAMPLE

Assume AS = X'8000 0000'
FP3 = X'04'
FP1 = X'19'
FP2 = X'2B'
and M is as shown in the tables.
Then the instruction MPLUM has the effect as shown in the following tables:
(1) In Section 0, 7 x 5 = 35
(2) In Section 1, no operation
(3) In Section 2, 7 x -8 = -56
(4) In Section 3, -6 x -15 = 90

Note that the associative memory locations which are not shown or are left blank are not changed in this instruction.

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 ,32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03			1		
	04		0000 0007	1		0000 0007
	05			1		
	06			1		
	07			1		
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1		
	11			1		
	12			1		
	13			1		
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19	0000 0005		1	0000 0005	
	1A			1		
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1		
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1		
	2B			1	0000 0000	0000 0023
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1		
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32, 32) (Hex)
Section 1	40			0		
	41			0		
	42			0		
	43			0		
	44			0		
	45			0		
	46			0		
	47			0		
	48			0		
	49			0		
	4A			0		
	4B			0		
	4C			0		
	4D			0		
	4E			0		
	4F			0		
	50			0		
	51			0		
	52			0		
	53			0		
	54			0		
	55			0		
	56			0		
	57			0		
	58			0		
	59			0		
	5A			0		
	5B			0		
	5C			0		
	5D			0		
	5E			0		
	5F			0		
	60			0		
	61			0		
	62			0		
	63			0		
	64			0		
	65			0		
	66			0		
	67			0		
	68			0		
	69			0		
	6A			0		
	6B			0		
	6C			0		
	6D			0		
	6E			0		
	6F			0		
	70			0		
	71			0		
	72			0		
	73			0		
	74			0		
	75			0		
	76			0		
	77			0		
	78			0		
	79			0		
	7A			0		
	7B			0		
	7C			0		
	7D			0		
	7E			0		
	7F			0		

	Word Addr. (Hex)	Before Execution: Field (0 ,32) (Hex)	Before Execution: Field (32,32) (Hex)	M	After Execution: Field (0 ,32) (Hex)	After Execution: Field (32 ,32) (Hex)
Section 2	80			1		
	81			1		
	82			1		
	83			1		
	84		0000 0007	1		0000 0007
	85			1		
	86			1		
	87			1		
	88			1		
	89			1		
	8A			1		
	8B			1		
	8C			1		
	8D			1		
	8E			1		
	8F			1		
	90			1		
	91			1		
	92			1		
	93			1		
	94			1		
	95			1		
	96			1		
	97			1		
	98			1		
	99	FFFF FFF8		1	FFFF FFF8	
	9A			1		
	9B			1		
	9C			1		
	9D			1		
	9E			1		
	9F			1		
	A0			1		
	A1			1		
	A2			1		
	A3			1		
	A4			1		
	A5			1		
	A6			1		
	A7			1		
	A8			1		
	A9			1		
	AA			1		
	AB			1	FFFF FFFF	FFFF FFC8
	AC			1		
	AD			1		
	AE			1		
	AF			1		
	B0			1		
	B1			1		
	B2			1		
	B3			1		
	B4			1		
	B5			1		
	B6			1		
	B7			1		
	B8			1		
	B9			1		
	BA			1		
	BB			1		
	BC			1		
	BD			1		
	BE			1		
	BF			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 ,32) (Hex)	Field (32, 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3			1		
	C4		F F F F F F F A	1		F F F F F F F A
	C5			1		
	C6			1		
	C7			1		
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1		
	D1			1		
	D2			1		
	D3			1		
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9	F F F F F F F 1		1	F F F F F F F 1	
	DA			1		
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1		
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 5 A
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			1		
	F6			1		
	F7			1		
	F8			1		
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

MPLUMA — Multiply 32-bit words by 32-bit words in mixed mode mod-64 (Fixed-Point)

This instruction will multiply the lower 32 bits of the words mod-64 pointed to by a1, by the upper 32 bits of the words mod-64 pointed to by a2 and store the 64-bit resultant products into the words mod-64 pointed to by a3. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPLUMA	a1, a2, a3

- LABEL — Any valid symbol or blank.
- COMMAND — MPLUMA
- ARGUMENT — Three entries are required. The first entry points to the words mod-64 whose lower 32 bits are the multiplicands. The second entry points to the words mod-64 whose upper 32 bits are the multipliers. The third entry points to the words mod-64 which are the 64-bit products of the multiplicands multiply the multipliers.
- a1, a2, a3 — a1, a2 and a3 are the numbers between 0 and 255.

NOTES

(1) This instruction is the same as the instruction MPLUM, except that it loads the agruments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in the array sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(4) The addresses of a1, a2, a3 can have the same values.

(5) Before the instruction is called, the registers A6 and M must be set properly.

(6) The field pointers FP1, FP2 and FP3 are used.

(7) The addresses X'F4', X'F5', X'F7', X'F8', X'F9', X'FA' and X'FC' in the sections which participate in the instruction are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

MPLLM — Multiply 32-bit words by 32-bit words in mixed mode mod-64 (Fixed-Point).

This instruction will multiply the lower 32 bits of the words mod-64 pointed to by FP3 by the lower 32 bits of the words mod-64 pointed to by FP1 and store the 64-bit resultant products into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64. (Fig. 2-3 and Fig. 2-4). Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPLLM	Blank

- LABEL — Any valid symbol or blank.
- COMMAND — MPLLM
- ARGUMENT — Blank.

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The following addresses in the array sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(3) The addresses of FP1, FP2 and FP3 can have the same values.

(4) Before the instruction is called, the registers AS, M, FP1, FP2 and FP3 must be set properly.

(5) The addresses X'F4', X'F5', X'F7', X'F8', X'F9', X'FA', and X'FC' in the sections which participate in the instruction are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

EXAMPLE

Assume AS = X'8000 0000'
FP3 = X'04'
FP1 = X'19'
FP2 = X'2B'
and M is as shown in the tables, then the instruction MPLLM has the effect as shown in the following tables:
(1) In Section 0, 7 x 5 = 35
(2) In Section 1, no operation
(3) In Section 2, 7 x -8 = -56
(4) In Section 3, -6 x -15 = 90

Note that the associative memory locations which are not shown or are left blank are not changed in this instruction.

	Word Addr. (Hex)	Before Execution: Field (0 , 32) (Hex)	Before Execution: Field (32 , 32) (Hex)	M	After Execution: Field (0 , 32) (Hex)	After Execution: Field (32 , 32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03			1		
	04		0000 0007	1		0000 0007
	05			1		
	06			1		
	07			1		
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1		
	11			1		
	12			1		
	13			1		
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19		0000 0005	1		0000 0005
	1A			1		
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1		
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1	0000 0000	0000 0023
	2B			1		
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1		
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 1	40			0		
	41			0		
	42			0		
	43			0		
	44			0		
	45			0		
	46			0		
	47			0		
	48			0		
	49			0		
	4A			0		
	4B			0		
	4C			0		
	4D			0		
	4E			0		
	4F			0		
	50			0		
	51			0		
	52			0		
	53			0		
	54			0		
	55			0		
	56			0		
	57			0		
	58			0		
	59			0		
	5A			0		
	5B			0		
	5C			0		
	5D			0		
	5E			0		
	5F			0		
	60			0		
	61			0		
	62			0		
	63			0		
	64			0		
	65			0		
	66			0		
	67			0		
	68			0		
	69			0		
	6A			0		
	6B			0		
	6C			0		
	6D			0		
	6E			0		
	6F			0		
	70			0		
	71			0		
	72			0		
	73			0		
	74			0		
	75			0		
	76			0		
	77			0		
	78			0		
	79			0		
	7A			0		
	7B			0		
	7C			0		
	7D			0		
	7E			0		
	7F			0		

	Word Addr. (Hex)	Before Execution: Field (0 , 32) (Hex)	Before Execution: Field (32,32) (Hex)	M	After Execution: Field (0 ,32) (Hex)	After Execution: Field (32, 32) (Hex)
Section 2	80			1		
	81			1		
	82			1		
	83			1		
	84		0000 0007	1		0000 0007
	85			1		
	86			1		
	87			1		
	88			1		
	89			1		
	8A			1		
	8B			1		
	8C			1		
	8D			1		
	8E			1		
	8F			1		
	90			1		
	91			1		
	92			1		
	93			1		
	94			1		
	95			1		
	96			1		
	97			1		
	98			1		
	99		FFFF FFF8	1		FFFF FFF8
	9A			1		
	9B			1		
	9C			1		
	9D			1		
	9E			1		
	9F			1		
	A0			1		
	A1			1		
	A2			1		
	A3			1		
	A4			1		
	A5			1		
	A6			1		
	A7			1		
	A8			1		
	A9			1		
	AA			1		
	AB			1	FFFF FFFF	FFFF FFC8
	AC			1		
	AD			1		
	AE			1		
	AF			1		
	B0			1		
	B1			1		
	B2			1		
	B3			1		
	B4			1		
	B5			1		
	B6			1		
	B7			1		
	B8			1		
	B9			1		
	BA			1		
	BB			1		
	BC			1		
	BD			1		
	BE			1		
	BF			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32, 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3			1		
	C4		FFFF FFFA	1		FFFF FFFA
	C5			1		
	C6			1		
	C7			1		
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1		
	D1			1		
	D2			1		
	D3			1		
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9		FFFF FFF1	1		FFFF FFF1
	DA			1		
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1		
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1	0000 0000	0000 005A
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			1		
	F6			1		
	F7			1		
	F8			1		
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

MPLLMA — Multiply 32-bit words by 32-bit words in mixed mode mod-64 (Fixed-Point).

This instruction will multiply the lower 32 bits of the words mod-64 pointed to by a1 by the lower 32 bits of the words mod-64 pointed to by a2 and store the 64-bit resultant products into the words mod-64 pointed to by a3. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPLLMA	a1, a2, a3

- LABEL — Any valid symbol or blank.
- COMMAND — MPLLMA
- ARGUMENT — Three entries are required. The first entry points to the words mod-64 whose lower 32 bits are the multiplicands. The second entry points to the words mod-64 whose lower 32 bits are the multipliers. The third entry points to the words mod-64 which are the 64-bit products of the multiplicands multiply the multipliers.

•• a1, a2, a3 — a1, a2 and a3 are the numbers between 0 and 255.

NOTES

(1) This instruction is the same as the instruction MPLLM, except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in the array sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(4) The addresses of a1, a2, a3 can have the same values.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The field pointers FP1, FP2 and FP3 are used.

(7) The addresses X'F4', X'F5', X'F7', X'F8', X'F9', X'FA' and X'FC' in the sections which participate in the instruction are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

MPCUM — Multiply words by the common register in mixed mode mod-64 (Fixed-Point)

This instruction will multiply the upper 32 bits of the words mod-64 pointed to by FP3 by the 32-bit Common Register and store the 64-bit resultant products into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64 (Fig. 2-3 and Fig. 2-4). Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPCUM	Blank

- LABEL — Any valid symbol or blank.
- COMMAND — MPCUM
- ARGUMENT — Blank

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The following addresses in each array section selected must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE") = X'7FFF FFFF FFFF FFFF'
(X'FF") = X'FFFF FFFF 0000 0000'

(3) The field pointer FP1 is used.

(4) The addresses of FP2 and FP3 can have the same values.

(5) Before the instruction is called, the registers AS, M, FP2 and FP3 must be set properly.

(6) The addresses X'F4', X'F5', X'F9' and X'FA' are used for temporary storage. If any of these is used as an operand, the content will be destroyed.

EXAMPLE

Assume AS = X'8000 0000'
FP3 = X'5A'
FP2 = X'5F'
C = X'0000 0007'
and M is as shown in the tables,
then the instruction MPCUM has the effect as shown in the following tables.
This instruction has the following operations:
(1) In Section 0, 7 x -8 = -56
(2) In Section 1, 7 x 13 = 91
(3) In Section 2, no operation
(4) In Section 3, 7 x -6 = -42

Note that those memory locations which are not shown or are left blank are not changed.

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (64 , 32) (Hex)	Field (96,32) (Hex)	M	Field (64, 32) (Hex)	Field (96, 32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03			1		
	04			1		
	05			1		
	06			1		
	07			1		
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1		
	11			1		
	12			1		
	13			1		
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19			1		
	1A	F F F F F F F 8		1	F F F F F F F 8	
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1	F F F F F F F F	F F F F F F C 8
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1		
	2B			1		
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1		
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

	Word Addr. (Hex)	Before Execution: Field (64 ,32) (Hex)	Before Execution: Field (96 ,32) (Hex)	M	After Execution: Field (64, 32) (Hex)	After Execution: Field (96, 32) (Hex)
Section 1	40			1		
	41			1		
	42			1		
	43			1		
	44			1		
	45			1		
	46			1		
	47			1		
	48			1		
	49			1		
	4A			1		
	4B			1		
	4C			1		
	4D			1		
	4E			1		
	4F			1		
	50			1		
	51			1		
	52			1		
	53			1		
	54			1		
	55			1		
	56			1		
	57			1		
	58			1		
	59			1		
	5A	0000 000D		1	0000 000D	
	5B			1		
	5C			1		
	5D			1		
	5E			1		
	5F			1	0000 0000	0000 005B
	60			1		
	61			1		
	62			1		
	63			1		
	64			1		
	65			1		
	66			1		
	67			1		
	68			1		
	69			1		
	6A			1		
	6B			1		
	6C			1		
	6D			1		
	6E			1		
	6F			1		
	70			1		
	71			1		
	72			1		
	73			1		
	74			1		
	75			1		
	76			1		
	77			1		
	78			1		
	79			1		
	7A			1		
	7B			1		
	7C			1		
	7D			1		
	7E			1		
	7F			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (64 , 32) (Hex)	Field (96, 32) (Hex)	M	Field (64 ,32) (Hex)	Field (96 ,32) (Hex)
Section 2	80			0		
	81			0		
	82			0		
	83			0		
	84			0		
	85			0		
	86			0		
	87			0		
	88			0		
	89			0		
	8A			0		
	8B			0		
	8C			0		
	8D			0		
	8E			0		
	8F			0		
	90			0		
	91			0		
	92			0		
	93			0		
	94			0		
	95			0		
	96			0		
	97			0		
	98			0		
	99			0		
	9A			0		
	9B			0		
	9C			0		
	9D			0		
	9E			0		
	9F			0		
	A0			0		
	A1			0		
	A2			0		
	A3			0		
	A4			0		
	A5			0		
	A6			0		
	A7			0		
	A8			0		
	A9			0		
	AA			0		
	AB			0		
	AC			0		
	AD			0		
	AE			0		
	AF			0		
	B0			0		
	B1			0		
	B2			0		
	B3			0		
	B4			0		
	B5			0		
	B6			0		
	B7			0		
	B8			0		
	B9			0		
	BA			0		
	BB			0		
	BC			0		
	BD			0		
	BE			0		
	BF			0		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (64,32) (Hex)	Field (96, 32) (Hex)	M	Field (64 , 32) (Hex)	Field (96 , 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3			1		
	C4			1		
	C5			1		
	C6			1		
	C7			1		
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1		
	D1			1		
	D2			1		
	D3			1		
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9			1		
	DA	F F F F F F F A		1	F F F F F F F A	
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1	F F F F F F F F	F F F F F F D 6
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1		
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			1		
	F6			1		
	F7			1		
	F8			1		
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

MPCUMA — Multiply words by the Common Register in mixed mode mod-64 (Fixed-Point)

This instruction will multiply the upper 32 bits of the words mod-64 pointed to by a1 by the 32-bit common register and store the 64-bit resultant products into the words mod-64 pointed to by a2. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPCUMA	a1, a2

- LABEL — Any valid symbol or blank.
- COMMAND — MPCUMA
- ARGUMENT — Two entries are required. The first entry points to the words mod-64 whose upper 32 bits are the multiplicands. The second entry points to the words mod-64 which are the 64-bit resultant products of the multiplicands multiply the Common register.
  - a1, a2 — a1 and a2 are the numbers between 0 and 255.

NOTES

(1) This instruction is the same as the instruction MPCUM except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in each array section selected must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE") = X'7FFF FFFF FFFF FFFF'
(X'FF") = X'FFFF FFFF 0000 0000'

(4) The addresses of a1 and a2 can have the same values.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The addresses X'F4', X'F5', X'F9' and X'FA' are used for temporary storage. If any of these is used as an operand, the content will be destroyed.

(7) The field pointers FP1, FP2 and FP3 are used.

MPCLM — Multiply words by the Common Register in mixed mode mod-64 (Fixed-Point).

This instruction will multiply the lower 32 bits of the words mod-64 pointed to by FP3 by 32-bit Common Register and store the 64-bit resultant products into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64 (Fig. 2-3 and Fig. 2-4). Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPCLM	Blank

- LABEL — Any valid symbol or blank.
- COMMAND — MPCLM
- ARGUMENT — Blank

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The following addresses in each array section selected must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(3) The field pointer FP1 is used.

(4) The addresses of FP2 and FP3 can have the same values.

(5) Before the instruction is called, the registers AS, M, FP2 and FP3 must be set properly.

(6) The addresses X'F4', X'F5', X'F9', and X'FA' are used for temporary storage. If any of these is used as an operand, the content will be destroyed.

EXAMPLE

Assume AS = X'8000 0000'
FP3 = X'5A'
FP2 = X'5F'
C = X'0000 00C7'
and M is as shown in the tables, then the instruction MPCLM has the effect as shown in the following tables.
(1) In Section 0, 7 x -8 = -56
(2) In Section 1, 7 x 13 = 91
(3) In Section 2, no operation
(4) In Section 3, 7 x -6 = -42

Note that those memory locations which are not shown or are left blank are not changed.

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (64, 32) (Hex)	Field (96, 32) (Hex)	M	Field (64, 32) (Hex)	Field (96, 32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03			1		
	04			1		
	05			1		
	06			1		
	07			1		
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1		
	11			1		
	12			1		
	13			1		
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19			1		
	1A		FFFF FFF8	1		FFFF FFF8
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1	FFFF FFFF	FFFF FFC8
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1		
	2B			1		
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1		
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (64,32) (Hex)	Field (96, 32) (Hex)	M	Field (64 , 32) (Hex)	Field (96 , 32) (Hex)
Section 1	40			1		
	41			1		
	42			1		
	43			1		
	44			1		
	45			1		
	46			1		
	47			1		
	48			1		
	49			1		
	4A			1		
	4B			1		
	4C			1		
	4D			1		
	4E			1		
	4F			1		
	50			1		
	51			1		
	52			1		
	53			1		
	54			1		
	55			1		
	56			1		
	57			1		
	58			1		
	59			1		
	5A		0000 000D	1		0000 000D
	5B			1		
	5C			1		
	5D			1		
	5E			1		
	5F			1	0000 0000	0000 005B
	60			1		
	61			1		
	62			1		
	63			1		
	64			1		
	65			1		
	66			1		
	67			1		
	68			1		
	69			1		
	6A			1		
	6B			1		
	6C			1		
	6D			1		
	6E			1		
	6F			1		
	70			1		
	71			1		
	72			1		
	73			1		
	74			1		
	75			1		
	76			1		
	77			1		
	78			1		
	79			1		
	7A			1		
	7B			1		
	7C			1		
	7D			1		
	7E			1		
	7F			1		

	Word Addr. (Hex)	Before Execution: Field (64, 32) (Hex)	Before Execution: Field (96, 32) (Hex)	M	After Execution: Field (64, 32) (Hex)	After Execution: Field (96, 32) (Hex)
Section 2	80			0		
	81			0		
	82			0		
	83			0		
	84			0		
	85			0		
	86			0		
	87			0		
	88			0		
	89			0		
	8A			0		
	8B			0		
	8C			0		
	8D			0		
	8E			0		
	8F			0		
	90			0		
	91			0		
	92			0		
	93			0		
	94			0		
	95			0		
	96			0		
	97			0		
	98			0		
	99			0		
	9A			0		
	9B			0		
	9C			0		
	9D			0		
	9E			0		
	9F			0		
	A0			0		
	A1			0		
	A2			0		
	A3			0		
	A4			0		
	A5			0		
	A6			0		
	A7			0		
	A8			0		
	A9			0		
	AA			0		
	AB			0		
	AC			0		
	AD			0		
	AE			0		
	AF			0		
	B0			0		
	B1			0		
	B2			0		
	B3			0		
	B4			0		
	B5			0		
	B6			0		
	B7			0		
	B8			0		
	B9			0		
	BA			0		
	BB			0		
	BC			0		
	BD			0		
	BE			0		
	BF			0		

	Word Addr. (Hex)	Before Execution: Field (64, 32) (Hex)	Before Execution: Field (96, 32) (Hex)	M	After Execution: Field (64, 32) (Hex)	After Execution: Field (96, 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3			1		
	C4			1		
	C5			1		
	C6			1		
	C7			1		
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1		
	D1			1		
	D2			1		
	D3			1		
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9			1		
	DA		FFFF FFFA	1		FFFF FFFA
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1	FFFF FFFF	FFFF FFD6
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1		
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			1		
	F6			1		
	F7			1		
	F8			1		
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

MPCLMA — Multiply words by the Common Register in mixed mode mod-64 (Fixed-Point)

This instruction will multiply the lower 32 bits of the words mod-64 pointed to by a1 by the 32-bit Common Register and store the 64-bit resultant products into the words mod-64 pointed to by a2. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set to 1's will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	MPCLMA	a1 , a2

- LABEL — Any valid symbol or blank.
- COMMAND — MPCLMA
- ARGUMENT — Two entries are required. The first entry points to the words mod-64 whose lower 32 bits are the multiplicands. The second entry points to the words mod-64 which are the 64-bit resultant products of the multiplicands multiply the Common Register.
  - a1, a2 — a1 and a2 are the numbers between 0 and 255.

NOTES

(1) This instruction is the same as the instruction MPCLM except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in each array section selected must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(4) The addresses of a1 and a2 can have the same values.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The addresses X'F4', X'F5', X'F9' and X'FA' are used for temporary storage. If any of these is used as an operand, the content will be destroyed.

DVUUM — Divide Words by Words in mixed mode mod-64 (Fixed-Point)

This instruction will divide the upper 32 bits of the words mod-64 pointed to by FP3 by the upper 32 bits of the words mod-64 pointed to by FP1 and store the 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64 (Fig. 2-3 and Fig. 2-4). Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVUUM	Blank

- LABEL — Any valid symbol or blank.
- COMMAND — DVUUM
- ARGUMENT — Blank.

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The following addresses in the sections which participated in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(3) The addresses of FP1, FP2 and FP3 can have the same values.

(4) Before the instruction is called, the registers AS, M, FP3, FP1 and FP2 must be set properly.

(5) The addresses X'F4', X'F5', X'F9', X'FA', X'FC' and X'FD' in the sections which participate in the instruction are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

EXAMPLE

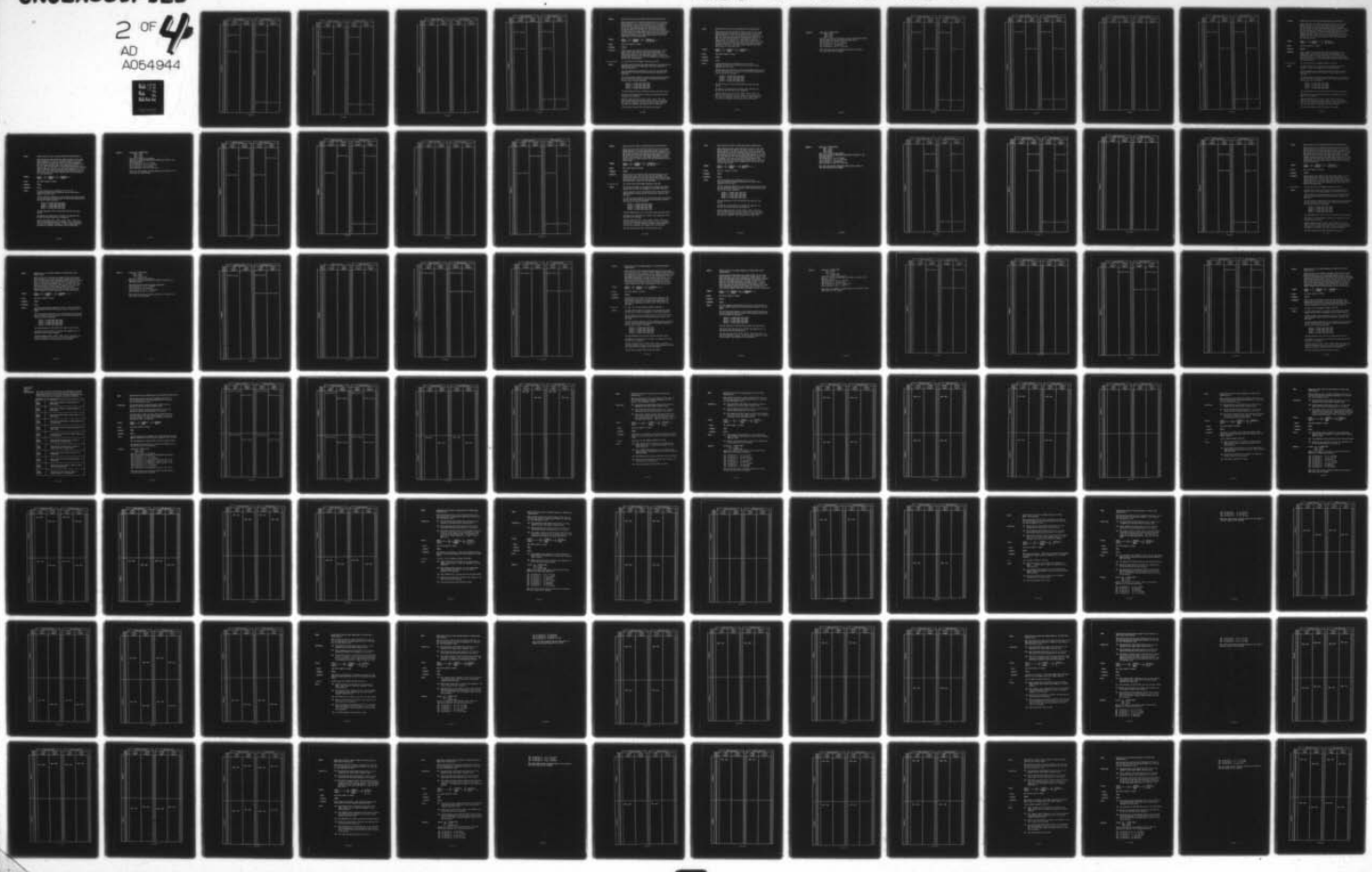
Assume AS = X'8000 0000'
FP3 = X' 07'
FP1 = X 13'
FP2 = X' 38'
and M is as shown in the tables
then the instruction DVUUM has the effect as shown in the following tables

This instruction has the operations:
(1) In Section 0, 37 ÷ 8 = R5 Q4
(2) In Section 2, -42 ÷ 5 = R-2 Q-8
(3) In Section 2, no operation
(4) In Section 3, -35 ÷ -4 = R-3 Q8

Note that those memory locations which are not shown or are left blank are not changed.

AD-A054 944 SYRACUSE UNIV N Y F/G 9/2
LARGE SCALE INFORMATION SYSTEMS. VOLUME III.(U)
MAR 78 F30602-74-C-0335
UNCLASSIFIED RADC-TR-78-43-VOL-3 NL

2 OF 4
AD
A054944



		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03			1		
	04			1		
	05			1		
	06			1		
	07	0000 0025		1	0000 0025	
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1		
	11			1		
	12			1		
	13	0000 0008		1	0000 0008	
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19			1		
	1A			1		
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1		
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1		
	2B			1		
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1	0000 0005	0000 0004
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

	Before Execution			After Execution	
Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
40			1		
41			1		
42			1		
43			1		
44			1		
45			1		
46			1		
47	FFFF FFD6		1	FFFF FFD6	
48			1		
49			1		
4A			1		
4B			1		
4C			1		
4D			1		
4E			1		
4F			1		
50			1		
51			1		
52			1		
53	0000 0005		1	0000 0005	
54			1		
55			1		
56			1		
57			1		
58			1		
59			1		
5A			1		
5B			1		
5C			1		
5D			1		
5E			1		
5F			1		
60			1		
61			1		
62			1		
63			1		
64			1		
65			1		
66			1		
67			1		
68			1		
69			1		
6A			1		
6B			1		
6C			1		
6D			1		
6E			1		
6F			1		
70			1		
71			1		
72			1		
73			1		
74			1		
75			1		
76			1		
77			1		
78			1	FFFF FFFE	FFFF FFF8
79			1		
7A			1		
7B			1		
7C			1		
7D			1		
7E			1		
7F			1		

Section 1

	Word Addr. (Hex)	Before Execution: Field (0 ,32) (Hex)	Before Execution: Field (32, 32) (Hex)	M	After Execution: Field (0 ,32) (Hex)	After Execution: Field (32 , 32) (Hex)
Section 2	80			0		
	81			0		
	82			0		
	83			0		
	84			0		
	85			0		
	86			0		
	87			0		
	88			0		
	89			0		
	8A			0		
	8B			0		
	8C			0		
	8D			0		
	8E			0		
	8F			0		
	90			0		
	91			0		
	92			0		
	93			0		
	94			0		
	95			0		
	96			0		
	97			0		
	98			0		
	99			0		
	9A			0		
	9B			0		
	9C			0		
	9D			0		
	9E			0		
	9F			0		
	A0			0		
	A1			0		
	A2			0		
	A3			0		
	A4			0		
	A5			0		
	A6			0		
	A7			0		
	A8			0		
	A9			0		
	AA			0		
	AB			0		
	AC			0		
	AD			0		
	AE			0		
	AF			0		
	B0			0		
	B1			0		
	B2			0		
	B3			0		
	B4			0		
	B5			0		
	B6			0		
	B7			0		
	B8			0		
	B9			0		
	BA			0		
	BB			0		
	BC			0		
	BD			0		
	BE			0		
	BF			0		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32, 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3			1		
	C4			1		
	C5			1		
	C6			1		
	C7	F F F F F F D D		1	F F F F F F D D	
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1		
	D1			1		
	D2			1		
	D3	F F F F F F F C		1	F F F F F F F C	
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9			1		
	DA			1		
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1		
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1		
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			1		
	F6			1		
	F7			1		
	F8			1	F F F F F F F D	0 0 0 0 0 0 0 8
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

DVUUMA — Divide words by words in mixed mode mod-64 (fixed-point)

This instruction will divide the upper 32 bits of the words mod-64 pointed to by a1 by the upper 32 bits of the words mod-64 pointed to by a2 and store the 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by a3. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVUUMA	a1, a2, a3

- LABEL — Any valid symbol or blank
- COMMAND — DVUUMA
- ARGUMENT — Three entries are required. The first entry points to the words mod-64 whose upper 32 bits are the dividends. The second entry points to the words mod-64 whose upper 32 bits are the divisors. The third entry points to the words mod-64 whose upper 32 bits are the remainders and whose lower 32 bits are the quotients.

•• a1, a2, a3 — a1, a2 and a3 are the numbers between 0 and 255.

NOTES

(1) This instruction has the same operation as the instruction DVUUM except that this instruction loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in the sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF FFFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(4) The addresses of a1, a2 and a3 can have the same values.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The addresses X'F4', X'F5', X'F9', X'FA', X'FC' and X'FD' in the sections which participate in the instruction are used for temporary storage. If any of those addresses is used as an operand, the content will be destroyed.

(7) The field pointers FP1, FP2 and FP3 are used.

DVULM — **Divide words by words in mixed mode mod-64 (Fixed-Point)**

This instruction will divide the upper 32 bits of the words mod-64 pointed to by FP3 by the lower 32 bits of the words mod-64 pointed to by FP1 and store the 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVULM	Blank

- **LABEL** — Any valid symbol or blank.
- **COMMAND** — DVULM
- **ARGUMENT** — Blank.

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The following addresses in the sections which participated in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(3) The addresses of FP1, FP2 and FP3 can have the same values.

(4) Before the instruction is called, the registers AS, M, FP3, FP1 and FP2 must be set properly.

(5) The addresses X'F4', X'F5', X'F9', X'FA', X'FC' and X'FD' in the sections which participate in the instruction are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

EXAMPLE

Assume AS = X'8000 0000'
FP3 = X'07'
FP1 = X'13'
FP2 = X'38'
and M is as shown in the tables, then the instruction DVULM has the effect as shown in the following tables:
(1) In Section 0, 37 ÷ 8 = R5 Q4
(2) In Section 1, -42 ÷ 5 = R-2 Q-8
(3) In Section 2, no operation
(4) In Section 3, -35 ÷ -4 = R-3 Q8

Note that those memory locations which are not shown or are left blank are not changed.

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 ,32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03			1		
	04			1		
	05			1		
	06			1		
	07	0000 0025		1	0000 0005	
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1		
	11			1		
	12			1		
	13		0000 0008	1		0000 0008
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19			1		
	1A			1		
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1		
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1		
	2B			1		
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1	0000 0005	0000 0004
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

	Word Addr. (Hex)	Before Execution Field (0 ,32) (Hex)	Before Execution Field (32,32) (Hex)	M	After Execution Field (0 , 32) (Hex)	After Execution Field (32 , 32) (Hex)
Section 1	40			1		
	41			1		
	42			1		
	43			1		
	44			1		
	45			1		
	46			1		
	47	F F F F F F D 6		1	F F F F F F D 6	
	48			1		
	49			1		
	4A			1		
	4B			1		
	4C			1		
	4D			1		
	4E			1		
	4F			1		
	50			1		
	51			1		
	52			1		
	53		0 0 0 0 0 0 0 5	1		0 0 0 0 0 0 0 5
	54			1		
	55			1		
	56			1		
	57			1		
	58			1		
	59			1		
	5A			1		
	5B			1		
	5C			1		
	5D			1		
	5E			1		
	5F			1		
	60			1		
	61			1		
	62			1		
	63			1		
	64			1		
	65			1		
	66			1		
	67			1		
	68			1		
	69			1		
	6A			1		
	6B			1		
	6C			1		
	6D			1		
	6E			1		
	6F			1		
	70			1		
	71			1		
	72			1		
	73			1		
	74			1		
	75			1		
	76			1		
	77			1		
	78			1	F F F F F F F E	F F F F F F F 8
	79			1		
	7A			1		
	7B			1		
	7C			1		
	7D			1		
	7E			1		
	7F			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 ,32) (Hex)	Field (32, 32) (Hex)	M	Field (0 ,32) (Hex)	Field (32 , 32) (Hex)
Section 2	80			0		
	81			0		
	82			0		
	83			0		
	84			0		
	85			0		
	86			0		
	87			0		
	88			0		
	89			0		
	8A			0		
	8B			0		
	8C			0		
	8D			0		
	8E			0		
	8F			0		
	90			0		
	91			0		
	92			0		
	93			0		
	94			0		
	95			0		
	96			0		
	97			0		
	98			0		
	99			0		
	9A			0		
	9B			0		
	9C			0		
	9D			0		
	9E			0		
	9F			0		
	A0			0		
	A1			0		
	A2			0		
	A3			0		
	A4			0		
	A5			0		
	A6			0		
	A7			0		
	A8			0		
	A9			0		
	AA			0		
	AB			0		
	AC			0		
	AD			0		
	AE			0		
	AF			0		
	B0			0		
	B1			0		
	B2			0		
	B3			0		
	B4			0		
	B5			0		
	B6			0		
	B7			0		
	B8			0		
	B9			0		
	BA			0		
	BB			0		
	BC			0		
	BD			0		
	BE			0		
	BF			0		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32, 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3			1		
	C4			1		
	C5			1		
	C6			1		
	C7	F F F F F F D D		1	F F F F F F D D	
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1		
	D1			1		
	D2			1		
	D3		F F F F F F F C	1		F F F F F F F C
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9			1		
	DA			1		
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1		
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1		
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			2		
	F6			1		
	F7			1		
	F8			1	F F F F F F F D	0 0 0 0 0 0 0 8
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

DVULMA — Divide words by words in mixed mode mod-64 (Fixed-Point)

This instruction will divide the upper 32 bits of the words mod-64 pointed to by a1 by the lower 32 bits of the words mod-64 pointed to by a2 and store the 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by a3. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVULMA	a1, a2, a3

- LABEL — Any valid symbol or blank.
- COMMAND — DVULMA
- ARGUMENT — Three entries are required. The first entry points to the words mod-64 whose upper 32 bits are the dividends. The second entry points to the words mod-64 whose lower 32 bits are the divisers. The third entry points to the words mod-64 whose upper 32 bits are the remainders and whose lower 32-bits are the quotients.
- • a1, a2, a3 — a1, a2 and a3 are the numbers between 0 and 255.

NOTES

(1) This instruction is the same as the instruction DVULM except it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in the sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(4) The addresses of a1, a2 and a3 can have the same values.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The addresses X'F4', X'F5', X'F9', X'FA', X'FC' and X'FD' in the sections which participate in the instruction are used for temporary storage. If any of those addresses is used as an operand, the content will be destroyed.

(7) The field pointers FP1, FP2 and FP3 are used.

DVLUM — Divide words by words in mixed mode mod-64 (Fixed-Point)

This instruction will divide the lower 32 bits of the words mod-64 pointed to by FP3 by the upper 32 bits of the words mod-64 pointed to by FP1 and store the 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVLUM	Blank

- LABEL — Any valid symbol or blank.
- COMMAND — DVLUM
- ARGUMENT — Blank.

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The following addresses in the sections which participated in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(3) The addresses of FP1, FP2 and FP3 can have the same values.

(4) Before the instruction is called, the registers AS, M, FP3, FP1 and FP2 must be set properly.

(5) The addresses X'F4', X'F5', X'F9', X'FA', X'FC' and X'FD' in the sections which participate in the instruction are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

EXAMPLE

Assume AS = X'8000 0000'
FP3 = X'07'
FP1 = X'13'
FP2 = X'38'
and M is as shown in the tables,
then the instruction DVLUM has the effect as shown in the following tables.

(1) In Section 0, 37 ÷ 8 = R5 Q4
(2) In Section 1, -42 ÷ 5 = R-2 Q-8
(3) In Section 2, no operation
(4) In Section 3, -35 ÷ -4 = R-3 Q8

Note that those memory locations which are not shown or are left blank are not changed.

	Word Addr. (Hex)	Before Execution: Field (0 ,32) (Hex)	Before Execution: Field (32 , 32) (Hex)	M	After Execution: Field (0 , 32) (Hex)	After Execution: Field (32 , 32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03			1		
	04			1		
	05			1		
	06			1		
	07		0000 0025	1		0000 0025
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1		
	11			1		
	12			1		
	13	0000 0008		1	0000 0008	
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19			1		
	1A			1		
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1		
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1		
	2B			1		
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1	0000 0005	0000 0004
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

	Word Addr. (Hex)	Before Execution: Field (0, 32) (Hex)	Before Execution: Field (32, 32) (Hex)	M	After Execution: Field (0, 32) (Hex)	After Execution: Field (32, 32) (Hex)
Section 1	40			1		
	41			1		
	42			1		
	43			1		
	44			1		
	45			1		
	46			1		
	47		FFFF FFD6	1		FFFF FFD6
	48			1		
	49			1		
	4A			1		
	4B			1		
	4C			1		
	4D			1		
	4E			1		
	4F			1		
	50			1		
	51			1		
	52			1		
	53	0000 0005		1	0000 0005	
	54			1		
	55			1		
	56			1		
	57			1		
	58			1		
	59			1		
	5A			1		
	5B			1		
	5C			1		
	5D			1		
	5E			1		
	5F			1		
	60			1		
	61			1		
	62			1		
	63			1		
	64			1		
	65			1		
	66			1		
	67			1		
	68			1		
	69			1		
	6A			1		
	6B			1		
	6C			1		
	6D			1		
	6E			1		
	6F			1		
	70			1		
	71			1		
	72			1		
	73			1		
	74			1		
	75			1		
	76			1		
	77			1		
	78			1	FFFF FFFE	FFFF FFF8
	79			1		
	7A			1		
	7B			1		
	7C			1		
	7D			1		
	7E			1		
	7F			1		

	Word Addr. (Hex)	Before Execution: Field (0 , 32) (Hex)	Before Execution: Field (32 , 32) (Hex)	M	After Execution: Field (0 , 32) (Hex)	After Execution: Field (32 , 32) (Hex)
Section 2	80			0		
	81			0		
	82			0		
	83			0		
	84			0		
	85			0		
	86			0		
	87			0		
	88			0		
	89			0		
	8A			0		
	8B			0		
	8C			0		
	8D			0		
	8E			0		
	8F			0		
	90			0		
	91			0		
	92			0		
	93			0		
	94			0		
	95			0		
	96			0		
	97			0		
	98			0		
	99			0		
	9A			0		
	9B			0		
	9C			0		
	9D			0		
	9E			0		
	9F			0		
	A0			0		
	A1			0		
	A2			0		
	A3			0		
	A4			0		
	A5			0		
	A6			0		
	A7			0		
	A8			0		
	A9			0		
	AA			0		
	AB			0		
	AC			0		
	AD			0		
	AE			0		
	AF			0		
	B0			0		
	B1			0		
	B2			0		
	B3			0		
	B4			0		
	B5			0		
	B6			0		
	B7			0		
	B8			0		
	B9			0		
	BA			0		
	BB			0		
	BC			0		
	BD			0		
	BE			0		
	BF			0		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 ,32) (Hex)	Field (32, 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3			1		
	C4			1		
	C5			1		
	C6			1		
	C7		F F F F F F D D	1		F F F F F F D D
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1		
	D1			1		
	D2			1		
	D3	F F F F F F F C		1	F F F F F F F C	
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9			1		
	DA			1		
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1		
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1		
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			1		
	F6			1		
	F7			1		
	F8			1	F F F F F F F D	0 0 0 0 0 0 0 8
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

DVLUMA — Divide words by words in mixed mode mod-64 (Fixed-Point)

This instruction will divide the lower 32 bits of the words mod-64 pointed to by a1 by the upper 32 bits of the words mod-64 pointed to by a2 and store the 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by a3. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVLUMA	a1, a2, a3

- **LABEL** — Any valid symbol or blank.
- **COMMAND** — DVLUMA
- **ARGUMENT** — Three entries are required. The first entry points to the words mod-64 whose lower 32 bits are the dividends. The second entry points to the words mod-64 whose upper 32 bits are the divisors. The third entry points to the words mod-64 and whose lower 32 bits are the quotients.
  - **a1, a2, a3** — a1, a2 and a3 are the numbers between 0 and 255.

NOTES

(1) This instruction is the same as the instruction DVLUM except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in the sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(4) The addresses of a1, a2 and a3 can have the same values.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The addresses X'F4', X'F5', X'F9', X'FA', X'FC' and X'FD' in the sections which participate in the instruction are used for temporary storage. If any of those addresses is used as an operand, the content will be destroyed.

(7) The field pointers FP1, FP2 and FP3 are used.

DVLLM Divide words by words in mixed mode mod-64 (Fixed-Point)

This instruction will divide the lower 32 bits of the words mod-64 pointed to by FP3 by the lower 32 bits of the words mod-64 pointed to by FP1 and store the 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVLLM	Blank

- **LABEL** Any valid symbol or blank.
- **COMMAND** DVLLM
- **ARGUMENT** Blank.

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The following addresses in the sections which participated in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(3) The addresses of FP1, FP2 and FP3 can have the same values.

(4) Before the instruction is called, the registers AS, M, FP3, FP1 and FP2 must be set properly.

(5) The addresses X'F4', X'F5', X'F9', X'FA', X'FC' and X'FD' in the sections which participate in the instruction are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

EXAMPLE

Assume AS = X'8000 0000'
FP3 = X'07'
FP1 = X'13'
FP2 = X'38'
and M is as shown in the tables,
then the instruction DVLLM has the effect as shown in the following tables.
(1) In Section 0, 37 ÷ 8 = R5 Q4
(2) In Section 1, -42 ÷ 5 = R-2 Q-8
(3) In Section 2, no operation
(4) In Section 3, -38 ÷ -4 = R-3 Q8

Note that those memory locations which are not shown or are left blank are not changed.

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32, 32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03			1		
	04			1		
	05			1		
	06			1		
	07		0000 0025	1		0000 0025
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1		
	11			1		
	12			1		
	13		0000 0008	1		0000 0008
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19			1		
	1A			1		
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1		
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1		
	2B			1		
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1	0000 0005	0000 0004
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0, 32) (Hex)	Field (32, 32) (Hex)	M	Field (0, 32) (Hex)	Field (32, 32) (Hex)
Section 1	40			1		
	41			1		
	42			1		
	43			1		
	44			1		
	45			1		
	46			1		
	47		FFFF FFD6	1		FFFF FFD6
	48			1		
	49			1		
	4A			1		
	4B			1		
	4C			1		
	4D			1		
	4E			1		
	4F			1		
	50			1		
	51			1		
	52			1		
	53		0000 0005	1		0000 0005
	54			1		
	55			1		
	56			1		
	57			1		
	58			1		
	59			1		
	5A			1		
	5B			1		
	5C			1		
	5D			1		
	5E			1		
	5F			1		
	60			1		
	61			1		
	62			1		
	63			1		
	64			1		
	65			1		
	66			1		
	67			1		
	68			1		
	69			1		
	6A			1		
	6B			1		
	6C			1		
	6D			1		
	6E			1		
	6F			1		
	70			1		
	71			1		
	72			1		
	73			1		
	74			1		
	75			1		
	76			1		
	77			1		
	78			1	FFFF FFFE	FFFF FFF8
	79			1		
	7A			1		
	7B			1		
	7C			1		
	7D			1		
	7E			1		
	7F			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 2	80			0		
	81			0		
	82			0		
	83			0		
	84			0		
	85			0		
	86			0		
	87			0		
	88			0		
	89			0		
	8A			0		
	8B			0		
	8C			0		
	8D			0		
	8E			0		
	8F			0		
	90			0		
	91			0		
	92			0		
	93			0		
	94			0		
	95			0		
	96			0		
	97			0		
	98			0		
	99			0		
	9A			0		
	9B			0		
	9C			0		
	9D			0		
	9E			0		
	9F			0		
	A0			0		
	A1			0		
	A2			0		
	A3			0		
	A4			0		
	A5			0		
	A6			0		
	A7			0		
	A8			0		
	A9			0		
	AA			0		
	AB			0		
	AC			0		
	AD			0		
	AE			0		
	AF			0		
	B0			0		
	B1			0		
	B2			0		
	B3			0		
	B4			0		
	B5			0		
	B6			0		
	B7			0		
	B8			0		
	B9			0		
	BA			0		
	BB			0		
	BC			0		
	BD			0		
	BE			0		
	BF			0		

	Word Addr. (Hex)	Before Execution: Field (0, 32) (Hex)	Before Execution: Field (32, 32) (Hex)	M	After Execution: Field (0, 32) (Hex)	After Execution: Field (32, 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3			1		
	C4			1		
	C5			1		
	C6			1		
	C7		FFFF FFDD	1		FFFF FFDD
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1		
	D1			1		
	D2			1		
	D3		FFFF FFFC	1		FFFF FFFC
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9			1		
	DA			1		
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1		
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1		
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			1		
	F6			1		
	F7			1		
	F8			1	FFFF FFFD	0000 0008
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

DVLLMA — Divide words by words in mixed mode mod-64 (Fixed-Point)

This instruction will divide the lower 32 bits of the words mod-64 pointed to by a1 by the upper 32 bits of the words mod-64 pointed to by a2 and store the 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by a3. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVLLMA	a1, a2, a3

- LABEL — Any valid symbol or blank.
- COMMAND — DVLLMA
- ARGUMENT — Three entries are required. The first entry points to the words mod-64 whose lower 32 bits are the dividends. The second entry points to the words mod-64 whose lower 32 bits are the divisors. The third entry points to the words mod-64 whose upper 32 bits are the remainders and whose lower 32 bits are the quotients.

•• a1, a2, a3 — *a1, a2 and a3 are the numbers between 0 and 255.*

NOTES

(1) This instruction is the same as the instruction DVLLM except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in the sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(4) The addresses of a1, a2 and a3 can have the same values.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The addresses X'F4', X'F5', X'F9', X'FA', X'FC' and X'FD' in the sections which participate in the instruction are used for a temporary storage. If any of those addresses is used as an operand, the content will be destroyed.

(7) The field pointers FP1, FP2 and FP3 are used.

DVCUM — Divide words by the Common Register in mixed mode mod-64 (Fixed-Point)

This instruction will divide the upper 32 bits of the words mod-64 pointed to by FP3 by the 32-bit Common register and store the resultant 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64 (Fig. 2-3 and Fig. 2-4). Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVCUM	Blank

- LABEL — Any valid symbol or blank.
- COMMAND — DVCUM
- ARGUMENT — Blank.

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The following addresses in the sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(3) The addresses of FP2 and FP3 can have the same values.

(4) Before the instruction is called, the registers AS, M, FP3 and FP2 must be set properly.

(5) The addresses X'F4', X'F5', X'FA', X'FC', and X'FD' are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

EXAMPLE

Assume AS = X'8000 0000'
FP3 = X'03'
FP2 = X'10'
C = X'0000 0007'
and M is as shown in the tables,
then the instruction DVCUM has the effect as shown in the following tables.

This instruction has the following operations:
(1) In Section 0, 76 ÷ 7 = R6 Q10
(2) In Section 1, no operation
(3) In Section 2, -97 ÷ 7 = R-6 Q-13
(4) In Section 3, 18 ÷ 7 = R4 Q2

Note that those memory locations which are not shown or are left blank are not changed.

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 ,32) (Hex)	Field (32, 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03	0000 004C		1	0000 004C	
	04			1		
	05			1		
	06			1		
	07			1		
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1	0000 0006	0000 000A
	11			1		
	12			1		
	13			1		
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19			1		
	1A			1		
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1		
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1		
	2B			1		
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1		
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32, 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 1	40			0		
	41			0		
	42			0		
	43			0		
	44			0		
	45			0		
	46			0		
	47			0		
	48			0		
	49			0		
	4A			0		
	4B			0		
	4C			0		
	4D			0		
	4E			0		
	4F			0		
	50			0		
	51			0		
	52			0		
	53			0		
	54			0		
	55			0		
	56			0		
	57			0		
	58			0		
	59			0		
	5A			0		
	5B			0		
	5C			0		
	5D			0		
	5E			0		
	5F			0		
	60			0		
	61			0		
	62			0		
	63			0		
	64			0		
	65			0		
	66			0		
	67			0		
	68			0		
	69			0		
	6A			0		
	6B			0		
	6C			0		
	6D			0		
	6E			0		
	6F			0		
	70			0		
	71			0		
	72			0		
	73			0		
	74			0		
	75			0		
	76			0		
	77			0		
	78			0		
	79			0		
	7A			0		
	7B			0		
	7C			0		
	7D			0		
	7E			0		
	7F			0		

	Word Addr. (Hex)	Before Execution: Field (0 , 32) (Hex)	Before Execution: Field (32 , 32) (Hex)	M	After Execution: Field (0 , 32) (Hex)	After Execution: Field (32 , 32) (Hex)
Section 2	80			1		
	81			1		
	82			1		
	83	F F F F F F 9 F		1	F F F F F F 9 F	
	84			1		
	85			1		
	86			1		
	87			1		
	88			1		
	89			1		
	8A			1		
	8B			1		
	8C			1		
	8D			1		
	8E			1		
	8F			1		
	90			1	F F F F F F F A	F F F F F F F 3
	91			1		
	92			1		
	93			1		
	94			1		
	95			1		
	96			1		
	97			1		
	98			1		
	99			1		
	9A			1		
	9B			1		
	9C			1		
	9D			1		
	9E			1		
	9F			1		
	A0			1		
	A1			1		
	A2			1		
	A3			1		
	A4			1		
	A5			1		
	A6			1		
	A7			1		
	A8			1		
	A9			1		
	AA			1		
	AB			1		
	AC			1		
	AD			1		
	AE			1		
	AF			1		
	B0			1		
	B1			1		
	B2			1		
	B3			1		
	B4			1		
	B5			1		
	B6			1		
	B7			1		
	B8			1		
	B9			1		
	BA			1		
	BB			1		
	BC			1		
	BD			1		
	BE			1		
	BF			1		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3	0000 0012		1	0000 0012	
	C4			1		
	C5			1		
	C6			1		
	C7			1		
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1	0000 0004	0000 0002
	D1			1		
	D2			1		
	D3			1		
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9			1		
	DA			1		
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1		
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1		
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			1		
	F6			1		
	F7			1		
	F8			1		
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

DVCUMA — Divide words by the Common Register in mixed mode mod-64 (Fixed-Point)

This instruction will divide the upper 32 bits of the words mod-64 pointed to by a1 by the 32-bit common register and store the resultant 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by a2. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVCUMA	a1, a2

- LABEL — Any valid symbol or blank.
- COMMAND — DVCUMA
- ARGUMENT — Two entries are required. The first entry points to the words mod-64 whose upper 32 bits are the dividends. The second entry points to the words mod-64 whose upper 32 bits are the remainders and whose lower 32 bits are the quotients.

•• a1, a2 — a1 and a2 are the numbers between 0 and 255.

NOTES

(1) This instruction is the same as the instruction DVCUM except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in the sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(4) The addresses of a1 and a2 can have the same value.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The addresses X'F4', X'F5', X'FA', X'FC' and X'FD' are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

(7) The field pointers FP3 and FP2 are used.

DVCLM Divide words by the Common Register in mixed mode mod-64 (Fixed-Point)

This instruction will divide the lower 32 bits of the words mod-64 pointed to by FP3 by the 32-bit common register and store the resultant 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by FP2. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVCLM	Blank

- **LABEL** Any valid symbol or blank.
- **COMMAND** DVCLM
- **ARGUMENT** Blank

NOTES

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The following addresses in the sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(3) The addresses of FP2 and FP3 can have the same values.

(4) Before the instruction is called, the registers AS, M, FP3 and FP2 must be set properly.

(5) The addresses X'F4', X'F5', X'FA', X'FC', and X'FD' are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

EXAMPLE

Assume AS = X'8000 0000'
FP3 = X'03'
FP2 = X'10'
C = X'0000 0007'
and M is as shown in the tables,
then the instruction DVCLM has the effect as shown in the following tables.

(1) In Section 0, 76 ÷ 7 = R6 Q10
(2) In Section 1, no operation
(3) In Section 2, -97 ÷ 7 = R-6 Q-13
(4) In Section 3, 18 ÷ 7 = R4 Q2

Note that those memory locations which are not shown or are left blank are not changed.

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 0	00			1		
	01			1		
	02			1		
	03		0000 004C	1		0000 004C
	04			1		
	05			1		
	06			1		
	07			1		
	08			1		
	09			1		
	0A			1		
	0B			1		
	0C			1		
	0D			1		
	0E			1		
	0F			1		
	10			1	0000 0006	0000 000A
	11			1		
	12			1		
	13			1		
	14			1		
	15			1		
	16			1		
	17			1		
	18			1		
	19			1		
	1A			1		
	1B			1		
	1C			1		
	1D			1		
	1E			1		
	1F			1		
	20			1		
	21			1		
	22			1		
	23			1		
	24			1		
	25			1		
	26			1		
	27			1		
	28			1		
	29			1		
	2A			1		
	2B			1		
	2C			1		
	2D			1		
	2E			1		
	2F			1		
	30			1		
	31			1		
	32			1		
	33			1		
	34			1		
	35			1		
	36			1		
	37			1		
	38			1		
	39			1		
	3A			1		
	3B			1		
	3C			1		
	3D			1		
	3E			1		
	3F			1		

	Word Addr. (Hex)	Before Execution: Field (0 , 32) (Hex)	Before Execution: Field (32 , 32) (Hex)	M	After Execution: Field (0 , 32) (Hex)	After Execution: Field (32 , 32) (Hex)
Section 1	40			0		
	41			0		
	42			0		
	43			0		
	44			0		
	45			0		
	46			0		
	47			0		
	48			0		
	49			0		
	4A			0		
	4B			0		
	4C			0		
	4D			0		
	4E			0		
	4F			0		
	50			0		
	51			0		
	52			0		
	53			0		
	54			0		
	55			0		
	56			0		
	57			0		
	58			0		
	59			0		
	5A			0		
	5B			0		
	5C			0		
	5D			0		
	5E			0		
	5F			0		
	60			0		
	61			0		
	62			0		
	63			0		
	64			0		
	65			0		
	66			0		
	67			0		
	68			0		
	69			0		
	6A			0		
	6B			0		
	6C			0		
	6D			0		
	6E			0		
	6F			0		
	70			0		
	71			0		
	72			0		
	73			0		
	74			0		
	75			0		
	76			0		
	77			0		
	78			0		
	79			0		
	7A			0		
	7B			0		
	7C			0		
	7D			0		
	7E			0		
	7F			0		

		Before Execution			After Execution	
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32, 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)
Section 2	80			1		
	81			1		
	82			1		
	83		FFFF FF9F	1		FFFF FF9F
	84			1		
	85			1		
	86			1		
	87			1		
	88			1		
	89			1		
	8A			1		
	8B			1		
	8C			1		
	8D			1		
	8E			1		
	8F			1		
	90			1	FFFF FFFA	FFFF FFF3
	91			1		
	92			1		
	93			1		
	94			1		
	95			1		
	96			1		
	97			1		
	98			1		
	99			1		
	9A			1		
	9B			1		
	9C			1		
	9D			1		
	9E			1		
	9F			1		
	A0			1		
	A1			1		
	A2			1		
	A3			1		
	A4			1		
	A5			1		
	A6			1		
	A7			1		
	A8			1		
	A9			1		
	AA			1		
	AB			1		
	AC			1		
	AD			1		
	AE			1		
	AF			1		
	B0			1		
	B1			1		
	B2			1		
	B3			1		
	B4			1		
	B5			1		
	B6			1		
	B7			1		
	B8			1		
	B9			1		
	BA			1		
	BB			1		
	BC			1		
	BD			1		
	BE			1		
	BF			1		

	Word Addr. (Hex)	Before Execution: Field (0, 32) (Hex)	Before Execution: Field (32, 32) (Hex)	M	After Execution: Field (0, 32) (Hex)	After Execution: Field (32, 32) (Hex)
Section 3	C0			1		
	C1			1		
	C2			1		
	C3		0000 0012	1		0000 0012
	C4			1		
	C5			1		
	C6			1		
	C7			1		
	C8			1		
	C9			1		
	CA			1		
	CB			1		
	CC			1		
	CD			1		
	CE			1		
	CF			1		
	D0			1	0000 0004	0000 0002
	D1			1		
	D2			1		
	D3			1		
	D4			1		
	D5			1		
	D6			1		
	D7			1		
	D8			1		
	D9			1		
	DA			1		
	DB			1		
	DC			1		
	DD			1		
	DE			1		
	DF			1		
	E0			1		
	E1			1		
	E2			1		
	E3			1		
	E4			1		
	E5			1		
	E6			1		
	E7			1		
	E8			1		
	E9			1		
	EA			1		
	EB			1		
	EC			1		
	ED			1		
	EE			1		
	EF			1		
	F0			1		
	F1			1		
	F2			1		
	F3			1		
	F4			1		
	F5			1		
	F6			1		
	F7			1		
	F8			1		
	F9			1		
	FA			1		
	FB			1		
	FC			1		
	FD			1		
	FE			1		
	FF			1		

DVCLMA — Divide words by the Common Register in mixed mode mod-64 (Fixed-Point)

This instruction will divide the lower 32 bits of the words mod-64 pointed to by a1 by the 32-bit Common register and store the resultant 32-bit remainders and the 32-bit quotients into the words mod-64 pointed to by a2. The addresses are based on the mixed mode mod-64, as shown in Fig. 2-3 and Fig. 2-4. Only those associative sections whose M response store bits are set will participate in this instruction. Only those associative memory modules which are selected by the Array Select register (AS) will participate in this instruction.

FORMAT

Label	Command	Argument
Symbol	DVCLMA	a1, a2

- LABEL — Any valid symbol or blank.
- COMMAND — DVCLMA
- ARGUMENT — Two entries are required. The first entry points to the words mod-64 whose lower 32 bits are the dividends. The second entry points to the words mod-64 whose upper 32 bits are the remainders and lower 32 bits are the quotients.

•• a1, a2 — a1 and a2 are the numbers between 0 and 255.

NOTES

(1) This instruction is the same as the instruction DVCLM except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The following addresses in the sections which participate in the instruction must be preset to the values indicated and can not be used as operands.

(X'F6') = X'0000 0001 0000 0000'
(X'FB') = X'7FFF FFFF 7FFF FFFF'
(X'FE') = X'7FFF FFFF FFFF FFFF'
(X'FF') = X'FFFF FFFF 0000 0000'

(4) The addresses of a1 and a2 can have the same value.

(5) Before the instruction is called, the registers AS and M must be set properly.

(6) The addresses X'F4', X'F5', X'FA', X'FC' and X'FD' are used for temporary storage. If any of these addresses is used as an operand, the content will be destroyed.

(7) The field pointers FP3 and FP2 are used.

Mixed Mode SEARCH Instructions

This group of instructions allows the programmer to perform the search operation between words mod-32, and between the Common Register with words mod-32. The most significant bits of all words mod-32 are considered to be the sign bits.

Mnemonic	Operation
EQWM EQWMA	Words mod-32 Equal to words mod-32 in Mixed Mode
EQCM EQCMA	Words mod-32 Equal to Common Register in Mixed Mode
NEWM NEWMA	Words mod-32 NOT equal to words mod-32 in Mixed Mode
NECM NECMA	Words mod-32 NOT Equal to Common Register in Mixed Mode
GTWM GTWMA	Words mod-32 Greater than words mod-32 in Mixed Mode
GTCM GTCMA	Words mod-32 Greater than Common Register in Mixed Mode
GEWM GEWMA	Words mod-32 Greater than or Equal to Words mod-32 in Mixed Mode
GECM GECMA	Words mod-32 Greater than or Equal to Common Register in Mixed Mode
LTWM LTWMA	Words mod-32 less than words mod-32 in Mixed Mode
LTCM LTCMA	Words mod-32 less than Common Register in Mixed Mode
LEWM LEWMA	Words mod-32 less than or equal to words mod-32 in Mixed Mode
LECM LECMA	Words mod-32 less than or equal to Common Register in Mixed Mode

EQWM — Words mod-32 equal to words mod-32 in Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS)

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by FP3 equals the word mod-32 pointed to by FP1 in the same section. The addresses are based on the mixed mode mod-32 (Fig. 2-1 and 2-2).

Format

Label	Command	Argument
Symbol	EQWM	Blank

- **Label** — Any valid symbol or blank.
- **Command** — EQWM
- **Argument** — Blank

Notes

(1) The response store register Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The addresses of FP3 and FP1 can have the same values.

(3) Before this instruction is called, the register AS, M, FP3 and FP1 must be set properly.

Example

Assume AS = X'8000 0000'
FP3 = X'41'
FP1 = X'A4'
and M is as shown in the tables,
then the instruction EQWM has the following operations:
(1) In Section 0, 3 = 3 is true
(2) In Section 1, 21 = -3 is false
(3) In Section 2, no operation; Y bits are set to 0's
(4) In Section 3, no operation; Y bits are set to 0's
(5) In Section 4, no operation; Y bits are set to 0's
(6) In Section 5, -2 = -2 is true
(7) In Section 6, 9 = 7 is false
(8) In Section 7, no operation; Y bits are set to 0's

Note that those memory locations which are not shown or are left blank are not changed.

	Word Addr. (Hex)	Before Execution: Field (64, 32) (Hex)	Before Execution: Field (160, 32) (Hex)	M	After Execution: Field (64, 32) (Hex)	After Execution: Field (160, 32) (Hex)	Y
Section 0	00			1			1
	01	0000 0003		1	0000 0003		1
	02			1			1
	03			1			1
	04		0000 0003	1		0000 0003	1
	05			1			1
	06			1			1
	07			1			1
	08			1			1
	09			1			1
	0A			1			1
	0B			1			1
	0C			1			1
	0D			1			1
	0E			1			1
	0F			1			1
	10			1			1
	11			1			1
	12			1			1
	13			1			1
	14			1			1
	15			1			1
	16			1			1
	17			1			1
	18			1			1
	19			1			1
	1A			1			1
	1B			1			1
	1C			1			1
	1D			1			1
	1E			1			1
	1F			1			1
Section 1	20			1			0
	21	0000 0015		1	0000 0015		0
	22			1			0
	23			1			0
	24		FFFF FFFD	1		FFFF FFFD	0
	25			1			0
	26			1			0
	27			1			0
	28			1			0
	29			1			0
	2A			1			0
	2B			1			0
	2C			1			0
	2D			1			0
	2E			1			0
	2F			1			0
	30			1			0
	31			1			0
	32			1			0
	33			1			0
	34			1			0
	35			1			0
	36			1			0
	37			1			0
	38			1			0
	39			1			0
	3A			1			0
	3B			1			0
	3C			1			0
	3D			1			0
	3E			1			0
	3F			1			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (64, 32) (Hex)	Field (160, 32) (Hex)	M	Field (64,32) (Hex)	Field (160, 32) (Hex)	Y
Section 2	40			0			0
	41	0000 0005		0	0000 0005		0
	42			0			0
	43			0			0
	44			0			0
	45		0000 0005	0		0000 0005	0
	46			0			0
	47			0			0
	48			0			0
	49			0			0
	4A			0			0
	4B			0			0
	4C			0			0
	4D			0			0
	4E			0			0
	4F			0			0
	50			0			0
	51			0			0
	52			0			0
	53			0			0
	54			0			0
	55			0			0
	56			0			0
	57			0			0
	58			0			0
	59			0			0
	5A			0			0
	5B			0			0
	5C			0			0
	5D			0			0
	5E			0			0
	5F			0			0
Section 3	60			0			0
	61	0000 0006		0	0000 0006		0
	62			0			0
	63			0			0
	64			0			0
	65		0000 0004	0		0000 0004	0
	66			0			0
	67			0			0
	68			0			0
	69			0			0
	6A			0			0
	6B			0			0
	6C			0			0
	6D			0			0
	6E			0			0
	6F			0			0
	70			0			0
	71			0			0
	72			0			0
	73			0			0
	74			0			0
	75			0			0
	76			0			0
	77			0			0
	78			0			0
	79			0			0
	7A			0			0
	7B			0			0
	7C			0			0
	7D			0			0
	7E			0			0
	7F			0			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (64, 32) (Hex)	Field (160, 32) (Hex)	M	Field (64, 32) (Hex)	Field (160, 32) (Hex)	Y
Section 4	80			0			0
	81			0			0
	82			0			0
	83			0			0
	84			0			0
	85			0			0
	86			0			0
	87			0			0
	88			0			0
	89			0			0
	8A			0			0
	8B			0			0
	8C			0			0
	8D			0			0
	8E			0			0
	8F			0			0
	90			0			0
	91			0			0
	92			0			0
	93			0			0
	94			0			0
	95			0			0
	96			0			0
	97			0			0
	98			0			0
	99			0			0
	9A			0			0
	9B			0			0
	9C			0			0
	9D			0			0
	9E			0			0
	9F			0			0
Section 5	A0			1			1
	A1	FFFF FFFE		1	FFFF FFFE		1
	A2			1			1
	A3			1			1
	A4			1			1
	A5		FFFF FFFE	1		FFFF FFFE	1
	A6			1			1
	A7			1			1
	A8			1			1
	A9			1			1
	AA			1			1
	AB			1			1
	AC			1			1
	AD			1			1
	AE			1			1
	AF			1			1
	B0			1			1
	B1			1			1
	B2			1			1
	B3			1			1
	B4			1			1
	B5			1			1
	B6			1			1
	B7			1			1
	B8			1			1
	B9			1			1
	BA			1			1
	BB			1			1
	BC			1			1
	BD			1			1
	BE			1			1
	BF			1			1

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (64, 32) (Hex)	Field (160, 32) (Hex)	M	Field (64, 32) (Hex)	Field (160, 32) (Hex)	Y
Section 6	C0			1			0
	C1	0000 0009		1	0000 0009		0
	C2			1			0
	C3			1			0
	C4			1			0
	C5		0000 0007	1		0000 0007	0
	C6			1			0
	C7			1			0
	C8			1			0
	C9			1			0
	CA			1			0
	CB			1			0
	CC			1			0
	CD			1			0
	CE			1			0
	CF			1			0
	D0			1			0
	D1			1			0
	D2			1			0
	D3			1			0
	D4			1			0
	D5			1			0
	D6			1			0
	D7			1			0
	D8			1			0
	D9			1			0
	DA			1			0
	DB			1			0
	DC			1			0
	DD			1			0
	DE			1			0
	DF			1			0
Section 7	E0			0			0
	E1			0			0
	E2			0			0
	E3			0			0
	E4			0			0
	E5			0			0
	E6			0			0
	E7			0			0
	E8			0			0
	E9			0			0
	EA			0			0
	EB			0			0
	EC			0			0
	ED			0			0
	EE			0			0
	EF			0			0
	F0			0			0
	F1			0			0
	F2			0			0
	F3			0			0
	F4			0			0
	F5			0			0
	F6			0			0
	F7			0			0
	F8			0			0
	F9			0			0
	FA			0			0
	FB			0			0
	FC			0			0
	FD			0			0
	FE			0			0
	FF			0			0

EQWMA — Words Mod-32 Equal To Words Mod-32 In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a_1 equals the word mod-32 pointed to be a_2 in the same section. The addresses are based on the mixed mode mod-32. (Fig. 2-1 and Fig. 2-2).

Format

Label	Command	Argument
Symbol	EQWMA	a_1,a_2

. Label — Any valid symbol or blank.

. Command — EQWMA

. *Argument* — *Two entries are* required. Both entries represent words mod-32 in associative memory that are compared with each other.

.. a_1,a_2 — a_1 and a_2 are the numbers between 0 and 255.

Notes

(1) This instruction is the same as the instruction EQWM, except that it loads the arguments to the field pointers.

(2) The response store register Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The addresses of a_1 and a_2 can have the same values.

(4) Before the instruction is called, the registers AS and M must be set properly.

(5) The field pointers FP3 and FP1 are used.

EQCM — Words Mod-32 Equal To Common Register In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true.

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by FP3 equals the Common register. The addresses are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	EQCM	Blank

. Label — Any valid symbol or blank.

. Command — EQCM

. Argument — Blank

Notes

(1) The response store register Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) Before this instruction is called, the register AS, M, and FP3 must be set properly.

Example:

```
Assume AS  = X'8000 0000'
       FP3 = X'05'
       C   = X'0000 0003'
```

and M is as shown in the tables, the the instruction EQCM has the following operations.

(1) In Section 0, 5 = 5 is true
(2) In Section 1, no operation
(3) In Section 2, 29 = 5 is false
(4) In Section 3, 5 = 5 is true
(5) In Section 4, no operation
(6) In Section 5, -5 = 5 is false
(7) In Section 6, no operation
(8) In Section 7, no operation

Note that those memory locations which are not shown or are left blank are not changed.

	Word Addr. (Hex)	Before Execution: Field (0,32) (Hex)	Before Execution: Field (64,32) (Hex)	M	After Execution: Field (0, 32) (Hex)	After Execution: Field (64, 32) (Hex)	Y
Section 0	00			1			1
	01			1			1
	02			1			1
	03			1			1
	04			1			1
	05	0000 0005		1	0000 0005		1
	06			1			1
	07			1			1
	08			1			1
	09			1			1
	0A			1			1
	0B			1			1
	0C			1			1
	0D			1			1
	0E			1			1
	0F			1			1
	10			1			1
	11			1			1
	12			1			1
	13			1			1
	14			1			1
	15			1			1
	16			1			1
	17			1			1
	18			1			1
	19			1			1
	1A			1			1
	1B			1			1
	1C			1			1
	1D			1			1
	1E			1			1
	1F			1			1
Section 1	20			0			0
	21			0			0
	22			0			0
	23			0			0
	24			0			0
	25	0000 001C		0	0000 001C		0
	26			0			0
	27			0			0
	28			0			0
	29			0			0
	2A			0			0
	2B			0			0
	2C			0			0
	2D			0			0
	2E			0			0
	2F			0			0
	30			0			0
	31			0			0
	32			0			0
	33			0			0
	34			0			0
	35			0			0
	36			0			0
	37			0			0
	38			0			0
	39			0			0
	3A			0			0
	3B			0			0
	3C			0			0
	3D			0			0
	3E			0			0
	3F			0			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (64 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (64 , 32) (Hex)	Y
Section 2	40			1			0
	41			1			0
	42			1			0
	43			1			0
	44			1			0
	45	0000 001D		1	0000 001D		0
	46			1			0
	47			1			0
	48			1			0
	49			1			0
	4A			1			0
	4B			1			0
	4C			1			0
	4D			1			0
	4E			1			0
	4F			1			0
	50			1			0
	51			1			0
	52			1			0
	53			1			0
	54			1			0
	55			1			0
	56			1			0
	57			1			0
	58			1			0
	59			1			0
	5A			1			0
	5B			1			0
	5C			1			0
	5D			1			0
	5E			1			0
	5F			1			0
Section 3	60			1			1
	61			1			1
	62			1			1
	63			1			1
	64			1			1
	65	0000 0005		1	0000 0005		1
	66			1			1
	67			1			1
	68			1			1
	69			1			1
	6A			1			1
	6B			1			1
	6C			1			1
	6D			1			1
	6E			1			1
	6F			1			1
	70			1			1
	71			1			1
	72			1			1
	73			1			1
	74			1			1
	75			1			1
	76			1			1
	77			1			1
	78			1			1
	79			1			1
	7A			1			1
	7B			1			1
	7C			1			1
	7D			1			1
	7E			1			1
	7F			1			1

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0, 32) (Hex)	Field (64 , 32) (Hex)	M	Field (0,32) (Hex)	Field (64,32) (Hex)	Y
Section 4	80			0			0
	81			0			0
	82			0			0
	83			0			0
	84			0			0
	85	0000 0005		0	0000 0005		0
	86			0			0
	87			0			0
	88			0			0
	89			0			0
	8A			0			0
	8B			0			0
	8C			0			0
	8D			0			0
	8E			0			0
	8F			0			0
	90			0			0
	91			0			0
	92			0			0
	93			0			0
	94			0			0
	95			0			0
	96			0			0
	97			0			0
	98			0			0
	99			0			0
	9A			0			0
	9B			0			0
	9C			0			0
	9D			0			0
	9E			0			0
	9F			0			0
Section 5	A0			1			0
	A1			1			0
	A2			1			0
	A3			1			0
	A4			1			0
	A5	FFFF FFFB		1	FFFF FFFB		0
	A6			1			0
	A7			1			0
	A8			1			0
	A9			1			0
	AA			1			0
	AB			1			0
	AC			1			0
	AD			1			0
	AE			1			0
	AF			1			0
	B0			1			0
	B1			1			0
	B2			1			0
	B3			1			0
	B4			1			0
	B5			1			0
	B6			1			0
	B7			1			0
	B8			1			0
	B9			1			0
	BA			1			0
	BB			1			0
	BC			1			0
	BD			1			0
	BE			1			0
	BF			1			0

	Word Addr. (Hex)	Before Execution: Field (0, 32) (Hex)	Before Execution: Field (64 , 32) (Hex)	Before Execution: M	After Execution: Field (0,32) (Hex)	After Execution: Field (64 ,32) (Hex)	After Execution: Y
Section 6	C0			0			0
	C1			0			0
	C2			0			0
	C3			0			0
	C4			0			0
	C5			0			0
	C6			0			0
	C7			0			0
	C8			0			0
	C9			0			0
	CA			0			0
	CB			0			0
	CC			0			0
	CD			0			0
	CE			0			0
	CF			0			0
	D0			0			0
	D1			0			0
	D2			0			0
	D3			0			0
	D4			0			0
	D5			0			0
	D6			0			0
	D7			0			0
	D8			0			0
	D9			0			0
	DA			0			0
	DB			0			0
	DC			0			0
	DD			0			0
	DE			0			0
	DF			0			0
Section 7	E0			0			0
	E1			0			0
	E2			0			0
	E3			0			0
	E4			0			0
	E5			0			0
	E6			0			0
	E7			0			0
	E8			0			0
	E9			0			0
	EA			0			0
	EB			0			0
	EC			0			0
	ED			0			0
	EE			0			0
	EF			0			0
	F0			0			0
	F1			0			0
	F2			0			0
	F3			0			0
	F4			0			0
	F5			0			0
	F6			0			0
	F7			0			0
	F8			0			0
	F9			0			0
	FA			0			0
	FB			0			0
	FC			0			0
	FD			0			0
	FE			0			0
	FF			0			0

EQCMA — Words Mod-32 Equal To Common Register In Mixed Mode. (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory, if, and only if, the following is true.

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a equals the Common register. The address are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	EQCMA	a

. Label — Any valid symbol or blank.

. Command — EQCMA

. Argument — One entry is required. This entry represents words mod-32 in associative memory that are compared with the Common register.

.. a — a is a number between 0 and 255.

Notes

(1) This instruction is the same as the instruction EQCM, except that it loads the argument to the field pointer.

(2) The response store register Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) Before this instruction is called, the registers AS and M must be set properly.

(4) The field pointer FP3 is used.

NEWM — Words Mod-32 Not Equal To Words Mod-32 In Mixed Mode. (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true.

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by FP3 is not equal to the word mod-32 pointed to by FP1 in the same section. The addresses are based on the mixed mode mod-32. (Fig. 2-1 and Fig. 2-2)

Format

Label	Command	Argument
Symbol	NEWM	Blank

. Label — Any valid symbol or blank.

. Command — NEWM

. Argument — Blank

Notes

(1) The response store register Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The addresses of FP3 and FP1 can have the same values.

(3) Before the instruction is called, the registers AS, M, FP3 and FP1 must be set properly.

Example

Assume AS = X'8000 0000'
FP3 = X'22'
FP1 = X'45'
and M is as shown in the tables, then the instruction NEWM has the following operations.

(1) In Section 0, $7 \neq 7$ is false
(2) In Section 1, $9 \neq 32$ is true
(3) In Section 2, no operation
(4) In Section 3, $-2 \neq 2$ is true
(5) In Section 4, $-4 \neq -5$ is true
(6) In Section 5, no operation
(7) In Section 6, no operation
(8) In Section 7, $7 = -1$ is true

Note that those memory locations which are not shown or left blank are not changed.

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (32,32) (Hex)	Field (64, 32) (Hex)	M	Field (32, 32) (Hex)	Field (64, 32) (Hex)	Y
Section 0	00			1			0
	01			1			0
	02	0000 0007		1	0000 0007		0
	03			1			0
	04			1			0
	05		0000 0007	1		0000 0007	0
	06			1			0
	07			1			0
	08			1			0
	09			1			0
	0A			1			0
	0B			1			0
	0C			1			0
	0D			1			0
	0E			1			0
	0F			1			0
	10			1			0
	11			1			0
	12			1			0
	13			1			0
	14			1			0
	15			1			0
	16			1			0
	17			1			0
	18			1			0
	19			1			0
	1A			1			0
	1B			1			0
	1C			1			0
	1D			1			0
	1E			1			0
	1F			1			0
Section 1	20			1			1
	21			1			1
	22	0000 0009		1	0000 0009		1
	23			1			1
	24			1			1
	25		0000 0020	1		0000 0020	1
	26			1			1
	27			1			1
	28			1			1
	29			1			1
	2A			1			1
	2B			1			1
	2C			1			1
	2D			1			1
	2E			1			1
	2F			1			1
	30			1			1
	31			1			1
	32			1			1
	33			1			1
	34			1			1
	35			1			1
	36			1			1
	37			1			1
	38			1			1
	39			1			1
	3A			1			1
	3B			1			1
	3C			1			1
	3D			1			1
	3E			1			1
	3F			1			1

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (32 , 32) (Hex)	Field (64, 32) (Hex)	M	Field (32, 32) (Hex)	Field (64, 32) (Hex)	Y
Section 2	40			0			0
	41			0			0
	42			0			0
	43			0			0
	44			0			0
	45			0			0
	46			0			0
	47			0			0
	48			0			0
	49			0			0
	4A			0			0
	4B			0			0
	4C			0			0
	4D			0			0
	4E			0			0
	4F			0			0
	50			0			0
	51			0			0
	52			0			0
	53			0			0
	54			0			0
	55			0			0
	56			0			0
	57			0			0
	58			0			0
	59			0			0
	5A			0			0
	5B			0			0
	5C			0			0
	5D			0			0
	5E			0			0
	5F			0			0
Section 3	60			1			1
	61			1			1
	62	FFFF FFFE		1	FFFF FFFE		1
	63			1			1
	64			1			1
	65		0000 0002	1		0000 0002	1
	66			1			1
	67			1			1
	68			1			1
	69			1			1
	6A			1			1
	6B			1			1
	6C			1			1
	6D			1			1
	6E			1			1
	6F			1			1
	70			1			1
	71			1			1
	72			1			1
	73			1			1
	74			1			1
	75			1			1
	76			1			1
	77			1			1
	78			1			1
	79			1			1
	7A			1			1
	7B			1			1
	7C			1			1
	7D			1			1
	7E			1			1
	7F			1			1

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (32, 32) (Hex)	Field (64, 32) (Hex)	M	Field (32,32) (Hex)	Field (64 ,32) (Hex)	Y
Section 4	80			1			1
	81			1			1
	82	FFFF FFFC		1	FFFF FFFC		1
	83			1			1
	84			1			1
	85		FFFF FFFB	1		FFFF FFFB	1
	86			1			1
	87			1			1
	88			1			1
	89			1			1
	8A			1			1
	8B			1			1
	8C			1			1
	8D			1			1
	8E			1			1
	8F			1			1
	90			1			1
	91			1			1
	92			1			1
	93			1			1
	94			1			1
	95			1			1
	96			1			1
	97			1			1
	98			1			1
	99			1			1
	9A			1			1
	9B			1			1
	9C			1			1
	9D			1			1
	9E			1			1
	9F			1			1
Section 5	A0			0			0
	A1			0			0
	A2			0			0
	A3			0			0
	A4			0			0
	A5			0			0
	A6			0			0
	A7			0			0
	A8			0			0
	A9			0			0
	AA			0			0
	AB			0			0
	AC			0			0
	AD			0			0
	AE			0			0
	AF			0			0
	B0			0			0
	B1			0			0
	B2			0			0
	B3			0			0
	B4			0			0
	B5			0			0
	B6			0			0
	B7			0			0
	B8			0			0
	B9			0			0
	BA			0			0
	BB			0			0
	BC			0			0
	BD			0			0
	BE			0			0
	BF			0			0

	Word Addr. (Hex)	Before Execution: Field (32,32) (Hex)	Before Execution: Field (64,32) (Hex)	M	After Execution: Field (32,32) (Hex)	After Execution: Field (64,32) (Hex)	Y
Section 6	C0			0			0
	C1			0			0
	C2			0			0
	C3			0			0
	C4			0			0
	C5			0			0
	C6			0			0
	C7			0			0
	C8			0			0
	C9			0			0
	CA			0			0
	CB			0			0
	CC			0			0
	CD			0			0
	CE			0			0
	CF			0			0
	D0			0			0
	D1			0			0
	D2			0			0
	D3			0			0
	D4			0			0
	D5			0			0
	D6			0			0
	D7			0			0
	D8			0			0
	D9			0			0
	DA			0			0
	DB			0			0
	DC			0			0
	DD			0			0
	DE			0			0
	DF			0			0
Section 7	E0			1			1
	E1			1			1
	E2	0000 0007		1	0000 0007		1
	E3			1			1
	E4			1			1
	E5		FFFF FFFF	1		FFFF FFFF	1
	E6			1			1
	E7			1			1
	E8			1			1
	E9			1			1
	EA			1			1
	EB			1			1
	EC			1			1
	ED			1			1
	EE			1			1
	EF			1			1
	F0			1			1
	F1			1			1
	F2			1			1
	F3			1			1
	F4			1			1
	F5			1			1
	F6			1			1
	F7			1			1
	F8			1			1
	F9			1			1
	FA			1			1
	FB			1			1
	FC			1			1
	FD			1			1
	FE			1			1
	FF			1			1

NEWMA — Words Mod-32 Not Equal To Words Mod-32 In Mixed Mode (Fixed-Point)

This instruction will set the Y response store to 1's for each section of associative memory if, and only if, the following is true.

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a_1 is not equal to the word mod-32 pointed to by a_2 in the same section. The addresses are based on the mixed mode mod-32. (Fig. 2-1 and Fig. 2-2)

Format

Label	Command	Argument
Symbol	NEWMA	a_1, a_2

. Label — Any valid symbol or blank.

. Command — NEWMA

. Argument — Two entries are required. Both entries represent words mod-32 in associative memory that are compared with each other.

.. a_1,a_2 — a_1 and a_2 are the numbers between 0 and 255.

Notes

(1) This instruction is the same as the instruction NEWMA, except that it loads the arguments to field pointers.

(2) The response store register Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The addresses of a_1 and a_2 can have the same values.

(4) Before the instruction is called, the registers AS and M must be set properly.

(5) The field pointers FP3 and FP1 are used.

NECM — Words Mod-32 Not Equal To Common Register In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true.

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by FP3 is not equal to the Common Register. The addresses are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	NECM	Blank

. **Label** — Any valid symbol or blank

. **Command** — NECM

. **Argument** — Blank

Notes

(1) The response store register Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) Before the instruction is called, the registers AS, M and FP3 must be set properly.

Example:

Assume AS = X'8000 0000'
FP3 = X'06'
C = = X'000 0005'

and M is as shown in the tables, then the instruction NECM has the following operations.

(1) In Section 0, $2 \neq 5$ is true
(2) In Section 1, $5 \neq 5$ is false
(3) In Section 2, no operation
(4) In Section 3, no operation
(5) In Section 4, $-2 \neq 5$ is true
(6) In Section 5, no operation
(7) In Section 6, no operation
(8) In Section 7, $5 \neq 5$ is false

Note that those memory locations which are not shown or are left blank are not changed.

	Word Addr. (Hex)	Before Execution: Field (0 , 32) (Hex)	Before Execution: Field (32 , 32) (Hex)	M	After Execution: Field (0 , 32) (Hex)	After Execution: Field (32 , 32) (Hex)	Y
Section 0	00			1			1
	01			1			1
	02			1			1
	03			1			1
	04			1			1
	05			1			1
	06	0000 0002		1	0000 0002		1
	07			1			1
	08			1			1
	09			1			1
	0A			1			1
	0B			1			1
	0C			1			1
	0D			1			1
	0E			1			1
	0F			1			1
	10			1			1
	11			1			1
	12			1			1
	13			1			1
	14			1			1
	15			1			1
	16			1			1
	17			1			1
	18			1			1
	19			1			1
	1A			1			1
	1B			1			1
	1C			1			1
	1D			1			1
	1E			1			1
	1F			1			1
Section 1	20			1			0
	21			1			0
	22			1			0
	23			1			0
	24			1			0
	25			1			0
	26	0000 0005		1	0000 0005		0
	27			1			0
	28			1			0
	29			1			0
	2A			1			0
	2B			1			0
	2C			1			0
	2D			1			0
	2E			1			0
	2F			1			0
	30			1			0
	31			1			0
	32			1			0
	33			1			0
	34			1			0
	35			1			0
	36			1			0
	37			1			0
	38			1			0
	39			1			0
	3A			1			0
	3B			1			0
	3C			1			0
	3D			1			0
	3E			1			0
	3F			1			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32, 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32,32) (Hex)	Y
Section 2	40			0			0
	41			0			0
	42			0			0
	43			0			0
	44			0			0
	45			0			0
	46			0			0
	47			0			0
	48			0			0
	49			0			0
	4A			0			0
	4B			0			0
	4C			0			0
	4D			0			0
	4E			0			0
	4F			0			0
	50			0			0
	51			0			0
	52			0			0
	53			0			0
	54			0			0
	55			0			0
	56			0			0
	57			0			0
	58			0			0
	59			0			0
	5A			0			0
	5B			0			0
	5C			0			0
	5D			0			0
	5E			0			0
	5F			0			0
Section 3	60			0			0
	61			0			0
	62			0			0
	63			0			0
	64			0			0
	65			0			0
	66			0			0
	67			0			0
	68			0			0
	69			0			0
	6A			0			0
	6B			0			0
	6C			0			0
	6D			0			0
	6E			0			0
	6F			0			0
	70			0			0
	71			0			0
	72			0			0
	73			0			0
	74			0			0
	75			0			0
	76			0			0
	77			0			0
	78			0			0
	79			0			0
	7A			0			0
	7B			0			0
	7C			0			0
	7D			0			0
	7E			0			0
	7F			0			0

	Word Addr. (Hex)	Before Execution: Field (0,32) (Hex)	Before Execution: Field (32 , 32) (Hex)	M	After Execution: Field (0, 32) (Hex)	After Execution: Field (32 ,32) (Hex)	Y
Section 4	80			1			1
	81			1			1
	82			1			1
	83			1			1
	84			1			1
	85			1			1
	86	FFFF FFFE		1	FFFF FFFE		1
	87			1			1
	88			1			1
	89			1			1
	8A			1			1
	8B			1			1
	8C			1			1
	8D			1			1
	8E			1			1
	8F			1			1
	90			1			1
	91			1			1
	92			1			1
	93			1			1
	94			1			1
	95			1			1
	96			1			1
	97			1			1
	98			1			1
	99			1			1
	9A			1			1
	9B			1			1
	9C			1			1
	9D			1			1
	9E			1			1
	9F			1			1
Section 5	A0			0			0
	A1			0			0
	A2			0			0
	A3			0			0
	A4			0			0
	A5			0			0
	A6			0			0
	A7			0			0
	A8			0			0
	A9			0			0
	AA			0			0
	AB			0			0
	AC			0			0
	AD			0			0
	AE			0			0
	AF			0			0
	B0			0			0
	B1			0			0
	B2			0			0
	B3			0			0
	B4			0			0
	B5			0			0
	B6			0			0
	B7			0			0
	B8			0			0
	B9			0			0
	BA			0			0
	BB			0			0
	BC			0			0
	BD			0			0
	BE			0			0
	BF			0			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (32, 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32, 32) (Hex)	Y
Section 6	C0			0			0
	C1			0			0
	C2			0			0
	C3			0			0
	C4			0			0
	C5			0			0
	C6			0			0
	C7			0			0
	C8			0			0
	C9			0			0
	CA			0			0
	CB			0			0
	CC			0			0
	CD			0			0
	CE			0			0
	CF			0			0
	D0			0			0
	D1			0			0
	D2			0			0
	D3			0			0
	D4			0			0
	D5			0			0
	D6			0			0
	D7			0			0
	D8			0			0
	D9			0			0
	DA			0			0
	DB			0			0
	DC			0			0
	DD			0			0
	DE			0			0
	DF			0			0
Section 7	E0			1			0
	E1			1			0
	E2			1			0
	E3			1			0
	E4			1			0
	E5			1			0
	E6	0000 0005		1	0000 0005		0
	E7			1			0
	E8			1			0
	E9			1			0
	EA			1			0
	EB			1			0
	EC			1			0
	ED			1			0
	EE			1			0
	EF			1			0
	F0			1			0
	F1			1			0
	F2			1			0
	F3			1			0
	F4			1			0
	F5			1			0
	F6			1			0
	F7			1			0
	F8			1			0
	F9			1			0
	FA			1			0
	FB			1			0
	FC			1			0
	FD			1			0
	FE			1			0
	FF			1			0

NECMA — Words Mod-32 Not Equal To Common Register In Mixed Mode. (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true.

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a is not equal to the Common register. The addresses are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	NECMA	a

. Label — Any valid symbol or blank

. Command — NECMA

. Argument — One entry is required. This entry represents words mod-32 in associative memory that are compared with the Common register.

.. a — a is a number between 0 and 255.

Notes

(1) This instruction is the same as the instruction NECM, except that it loads the argument to the field pointer.

(2) The response store register Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) Before the instruction is called, the registers AS and M must be set properly.

(4) The field pointer FP3 is used.

GTWM — Words Mod-32 Greater Than Words Mod-32 In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by FP3 is greater than the word mod-32 pointed to by FP1 in the same section. The addresses are based on the mixed mode mod-32. (Fig. 2-1 and Fig. 2-2).

Format

Label	Command	Argument
Symbol	GTWM	Blank

. **Label** — Any valid symbol or blank

. **Command** — GTWM

. **Argument** — Blank

Notes

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The addresses of FP3 and FP1 can have the same values.

(3) Before the instruction is called, the registers AS, M, FP3 and FP1 must be set properly.

(4) The associative array address X'FB' in the sections which participate in this instruction must be pre-set to X'7FFF FFFF' and this address can not be used as an operand.

Example:

Assume AS = X'8000 0000'
FP3 = X'10'
FP1 = X'33'
and M is as shown in the tables, then the instruction GTWM has the following operations.

(1) In Section 0, 9 > 2 is true
(2) In Section 1, no operation
(3) In Section 2, no operation
(4) In Section 3, -3 > 5 is true
(5) In Section 4, -2 > 6 is false

(6) In Section 5, 4 > 8 is false
(7) In Section 6, no operation
(8) In Section 7, 1 > -4 is true

Note that those memory locations which are not shown or are left blank are not changed.

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (32, 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32, 32) (Hex)	Y
Section 0	00			1			1
	01			1			1
	02			1			1
	03			1			1
	04			1			1
	05			1			1
	06			1			1
	07			1			1
	08			1			1
	09			1			1
	0A			1			1
	0B			1			1
	0C			1			1
	0D			1			1
	0E			1			1
	0F			1			1
	10	0000 0009		1	0000 0009		1
	11			1			1
	12			1			1
	13		0000 0002	1		0000 0002	1
	14			1			1
	15			1			1
	16			1			1
	17			1			1
	18			1			1
	19			1			1
	1A			1			1
	1B			1			1
	1C			1			1
	1D			1			1
	1E			1			1
	1F			1			1
Section 1	20			0			0
	21			0			0
	22			0			0
	23			0			0
	24			0			0
	25			0			0
	26			0			0
	27			0			0
	28			0			0
	29			0			0
	2A			0			0
	2B			0			0
	2C			0			0
	2D			0			0
	2E			0			0
	2F			0			0
	30			0			0
	31			0			0
	32			0			0
	33			0			0
	34			0			0
	35			0			0
	36			0			0
	37			0			0
	38			0			0
	39			0			0
	3A			0			0
	3B			0			0
	3C			0			0
	3D			0			0
	3E			0			0
	3F			0			0

	Word Addr. (Hex)	Before Execution: Field (0,32) (Hex)	Before Execution: Field (32, 32) (Hex)	M	After Execution: Field (0, 32) (Hex)	After Execution: Field (32,32) (Hex)	Y
Section 2	40			0			0
	41			0			0
	42			0			0
	43			0			0
	44			0			0
	45			0			0
	46			0			0
	47			0			0
	48			0			0
	49			0			0
	4A			0			0
	4B			0			0
	4C			0			0
	4D			0			0
	4E			0			0
	4F			0			0
	50			0			0
	51			0			0
	52			0			0
	53			0			0
	54			0			0
	55			0			0
	56			0			0
	57			0			0
	58			0			0
	59			0			0
	5A			0			0
	5B			0			0
	5C			0			0
	5D			0			0
	5E			0			0
	5F			0			0
Section 3	60			1			1
	61			1			1
	62			1			1
	63			1			1
	64			1			1
	65			1			1
	66			1			1
	67			1			1
	68			1			1
	69			1			1
	6A			1			1
	6B			1			1
	6C			1			1
	6D			1			1
	6E			1			1
	6F			1			1
	70	FFFF FFFD		1	FFFF FFFD		1
	71			1			1
	72			1			1
	73		FFFF FFFB	1		FFFF FFFB	1
	74			1			1
	75			1			1
	76			1			1
	77			1			1
	78			1			1
	79			1			1
	7A			1			1
	7B			1			1
	7C			1			1
	7D			1			1
	7E			1			1
	7F			1			1

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32, 32) (Hex)	M	Field (0,32) (Hex)	Field (32 ,32) (Hex)	Y
Section 4	80			1			0
	81			1			0
	82			1			0
	83			1			0
	84			1			0
	85			1			0
	86			1			0
	87			1			0
	88			1			0
	89			1			0
	8A			1			0
	8B			1			0
	8C			1			0
	8D			1			0
	8E			1			0
	8F			1			0
	90	FFFF FFFE		1	FFFF FFFE		0
	91			1			0
	92			1			0
	93		0000 0006	1		0000 0006	0
	94			1			0
	95			1			0
	96			1			0
	97			1			0
	98			1			0
	99			1			0
	9A			1			0
	9B			1			0
	9C			1			0
	9D			1			0
	9E			1			0
	9F			1			0
Section 5	A0			1			0
	A1			1			0
	A2			1			0
	A3			1			0
	A4			1			0
	A5			1			0
	A6			1			0
	A7			1			0
	A8			1			0
	A9			1			0
	AA			1			0
	AB			1			0
	AC			1			0
	AD			1			0
	AE			1			0
	AF			1			0
	B0	0000 0004		1	0000 0004		0
	B1			1			0
	B2			1			0
	B3		0000 0008	1		0000 0008	0
	B4			1			0
	B5			1			0
	B6			1			0
	B7			1			0
	B8			1			0
	B9			1			0
	BA			1			0
	BB			1			0
	BC			1			0
	BD			1			0
	BE			1			0
	BF			1			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (32, 32) (Hex)	M	Field (0, 32) (Hex)	Field (32, 32) (Hex)	Y
Section 6	C0			0			0
	C1			0			0
	C2			0			0
	C3			0			0
	C4			0			0
	C5			0			0
	C6			0			0
	C7			0			0
	C8			0			0
	C9			0			0
	CA			0			0
	CB			0			0
	CC			0			0
	CD			0			0
	CE			0			0
	CF			0			0
	D0			0			0
	D1			0			0
	D2			0			0
	D3			0			0
	D4			0			0
	D5			0			0
	D6			0			0
	D7			0			0
	D8			0			0
	D9			0			0
	DA			0			0
	DB			0			0
	DC			0			0
	DD			0			0
	DE			0			0
	DF			0			0
Section 7	E0			1			1
	E1			1			1
	E2			1			1
	E3			1			1
	E4			1			1
	E5			1			1
	E6			1			1
	E7			1			1
	E8			1			1
	E9			1			1
	EA			1			1
	EB			1			1
	EC			1			1
	ED			1			1
	EE			1			1
	EF			1			1
	F0	0000 0001		1	0000 0001		1
	F1			1			1
	F2			1			1
	F3		FFFF FFFC	1		FFFF FFFC	1
	F4			1			1
	F5			1			1
	F6			1			1
	F7			1			1
	F8			1			1
	F9			1			1
	FA			1			1
	FB			1			1
	FC			1			1
	FD			1			1
	FE			1			1
	FF			1			1

GTWMA — **Words Mod-32 Greater Than Words Mod-32 In Mixed Mode (Fixed-Point)**

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true.

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a_1 is greater than the word mod-32 pointed to by a_2 in the same section. The addresses are based on the mixed mode mod-32. (Fig. 2-1 and Fig. 2-2)

Format

Label	Command	Argument
Symbol	GTWMA	a_1, a_2

. **Label** — Any valid symbol or blank

. **Command** — GTWMA

. **Argument** — Two entries are required. Both entries represent words mod-32 in associative memory that are compared with each other.

.. a_1,a_2 — a_1 and a_2 are the numbers between 0 and 255.

Notes

(1) This instruction is the same as the instruction GTWM, except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The addresses of a_1 and a_2 can have the same values.

(4) Before the instruction is called, the registers, AS and M must be set properly.

(5) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFF' and this address can not be used as an operand.

(6) The field pointers FP3 and FP1 are used.

GTCM — Words Mod-32 Greater Than Common Register In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to be FP3 is greater than the Common register. The addresses are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	GTCM	Blank

. **Label** — Any vaild symbol or blank

. **Command** — GTCM

. **Argument** — Blank

Notes

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) Before the instruction is called, the registers AS, M and FP3 must be set properly.

(3) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operadn.

Example

Assume AS = X'8000 0000'
FP3 = X'07'
C = X'0000 0010'
and M is as shown in the following table, then the instruction GTCM has the following operations.

(1) In Section 0, 19 > 16 is true
(2) In Section 1, 16 > 16 is false
(3) In Section 2, -3 > 16 is false
(4) In Section 3, no operation
(5) In Section 4, 3 > 16 is false

(6) In Section 5, no operation
(7) In Section 6, no operation
(8) In Section 7, 256 > 16 is true

Note that those memory locations which are not shown or are left blank are not changed.

	Word Addr. (Hex)	Before Execution: Field (0,32) (Hex)	Before Execution: Field (32 ,32) (Hex)	M	After Execution: Field (0 , 32) (Hex)	After Execution: Field (32, 32) (Hex)	Y
Section 0	00			1			1
	01			1			1
	02			1			1
	03			1			1
	04			1			1
	05			1			1
	06			1			1
	07	0000 0013		1	0000 0013		1
	08			1			1
	09			1			1
	0A			1			1
	0B			1			1
	0C			1			1
	0D			1			1
	0E			1			1
	0F			1			1
	10			1			1
	11			1			1
	12			1			1
	13			1			1
	14			1			1
	15			1			1
	16			1			1
	17			1			1
	18			1			1
	19			1			1
	1A			1			1
	1B			1			1
	1C			1			1
	1D			1			1
	1E			1			1
	1F			1			1
Section 1	20			1			0
	21			1			0
	22			1			0
	23			1			0
	24			1			0
	25			1			0
	26			1			0
	27	0000 0010		1	0000 0010		0
	28			1			0
	29			1			0
	2A			1			0
	2B			1			0
	2C			1			0
	2D			1			0
	2E			1			0
	2F			1			0
	30			1			0
	31			1			0
	32			1			0
	33			1			0
	34			1			0
	35			1			0
	36			1			0
	37			1			0
	38			1			0
	39			1			0
	3A			1			0
	3B			1			0
	3C			1			0
	3D			1			0
	3E			1			0
	3F			1			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (32, 32) (Hex)	M	Field (0, 32) (Hex)	Field (32, 32) (Hex)	Y
Section 2	40			1			0
	41			1			0
	42			1			0
	43			1			0
	44			1			0
	45			1			0
	46			1			0
	47	FFFF FFFD		1	FFFF FFFD		0
	48			1			0
	49			1			0
	4A			1			0
	4B			1			0
	4C			1			0
	4D			1			0
	4E			1			0
	4F			1			0
	50			1			0
	51			1			0
	52			1			0
	53			1			0
	54			1			0
	55			1			0
	56			1			0
	57			1			0
	58			1			0
	59			1			0
	5A			1			0
	5B			1			0
	5C			1			0
	5D			1			0
	5E			1			0
	5F			1			0
Section 3	60			0			0
	61			0			0
	62			0			0
	63			0			0
	64			0			0
	65			0			0
	66			0			0
	67			0			0
	68			0			0
	69			0			0
	6A			0			0
	6B			0			0
	6C			0			0
	6D			0			0
	6E			0			0
	6F			0			0
	70			0			0
	71			0			0
	72			0			0
	73			0			0
	74			0			0
	75			0			0
	76			0			0
	77			0			0
	78			0			0
	79			0			0
	7A			0			0
	7B			0			0
	7C			0			0
	7D			0			0
	7E			0			0
	7F			0			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32,32) (Hex)	M	Field (0 ,32) (Hex)	Field (32,32) (Hex)	Y
Section 4	80			1			0
	81			1			0
	82			1			0
	83			1			0
	84			1			0
	85			1			0
	86			1			0
	87	0000 0003		1	0000 0003		0
	88			1			0
	89			1			0
	8A			1			0
	8B			1			0
	8C			1			0
	8D			1			0
	8E			1			0
	8F			1			0
	90			1			0
	91			1			0
	92			1			0
	93			1			0
	94			1			0
	95			1			0
	96			1			0
	97			1			0
	98			1			0
	99			1			0
	9A			1			0
	9B			1			0
	9C			1			0
	9D			1			0
	9E			1			0
	9F			1			0
Section 5	A0			0			0
	A1			0			0
	A2			0			0
	A3			0			0
	A4			0			0
	A5			0			0
	A6			0			0
	A7			0			0
	A8			0			0
	A9			0			0
	AA			0			0
	AB			0			0
	AC			0			0
	AD			0			0
	AE			0			0
	AF			0			0
	B0			0			0
	B1			0			0
	B2			0			0
	B3			0			0
	B4			0			0
	B5			0			0
	B6			0			0
	B7			0			0
	B8			0			0
	B9			0			0
	BA			0			0
	BB			0			0
	BC			0			0
	BD			0			0
	BE			0			0
	BF			0			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (32 ,32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	Y
Section 6	C0			0			0
	C1			0			0
	C2			0			0
	C3			0			0
	C4			0			0
	C5			0			0
	C6			0			0
	C7			0			0
	C8			0			0
	C9			0			0
	CA			0			0
	CB			0			0
	CC			0			0
	CD			0			0
	CE			0			0
	CF			0			0
	D0			0			0
	D1			0			0
	D2			0			0
	D3			0			0
	D4			0			0
	D5			0			0
	D6			0			0
	D7			0			0
	D8			0			0
	D9			0			0
	DA			0			0
	DB			0			0
	DC			0			0
	DD			0			0
	DE			0			0
	DF			0			0
Section 7	E0			1			1
	E1			1			1
	E2			1			1
	E3			1			1
	E4			1			1
	E5			1			1
	E6			1			1
	E7	0000 0100		1	0000 0100		1
	E8			1			1
	E9			1			1
	EA			1			1
	EB			1			1
	EC			1			1
	ED			1			1
	EE			1			1
	EF			1			1
	F0			1			1
	F1			1			1
	F2			1			1
	F3			1			1
	F4			1			1
	F5			1			1
	F6			1			1
	F7			1			1
	F8			1			1
	F9			1			1
	FA			1			1
	FB			1			1
	FC			1			1
	FD			1			1
	FE			1			1
	FF			1			1

GTCMA — **Words Mod-32 Greater Than Common Register In Mixed Mode (Fixed-Point)**

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a is a greater than the Common register. The addresses are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	GTCMA	a

. **Label** — Any valid symbol or blank.

. **Command** — GTCMA

. **Argument** — One entry is required. This entry represents the words mod-32 that are compared with the Common register.

.. **a** — a is a number between 0 and 255.

Notes

(1) This instruction is the same as the instruction GTCM, except that it loads the argument to a field pointer.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) Before the instruction is called, the registers AS and M must be set properly.

(4) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address can not be used as an operand.

(5) The field pointer FP3 is used.

GEWM — **Words Mod-32 Greater Than Or Equal To Words Mod-32 In Mixed Mode (Fixed-Point)**

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by FP3 is greater than or equal to the word mod-32 pointed to by FP1 in the same section. The addresses are based on the mixed mode mod-32 (Fig. 2-1 and Fig. 2-2).

Format

Label	Command	Argument
Symbol	GEWM	Blank

. Label — Any valid symbol or blank.

. Command — GEWM

. Argument — Blank

Notes

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The addresses of FP3 and FP1 can have the same values.

(3) Before the instruction is called, the registers AS, M, FP3 and FP1 must be set properly.

(4) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and the address cannot be used as an operand.

Example:

Assume AS = X'8000 0000'
FP3 = X'07'
FP1 = X'26'
and M is as shown in the tables, then the instruction GEWM has the following operations:

(1) In Section 0, $5 \geq 2$ is true
(2) In Section 1, $19 \geq 19$ is true
(3) In Section 2, $-2 \geq 3$ is false
(4) In Section 3, no operation
(5) In Section 4, no operation

(6) In Section 5, $-4 \geq -5$ is true
(7) In Section 6, $2 \geq -1$ is true
(8) In Section 7, $-4 \geq -2$ is true

Note that those memory locations which are not shown or are left blank are not changed.

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0, 32) (Hex)	Field (32, 32) (Hex)	M	Field (0, 32) (Hex)	Field (32,32) (Hex)	Y
Section 0	00			1			1
	01			1			1
	02			1			1
	03			1			1
	04			1			1
	05			1			1
	06		0000 0002	1		0000 0002	1
	07	0000 0005		1	0000 0005		1
	08			1			1
	09			1			1
	0A			1			1
	0B			1			1
	0C			1			1
	0D			1			1
	0E			1			1
	0F			1			1
	10			1			1
	11			1			1
	12			1			1
	13			1			1
	14			1			1
	15			1			1
	16			1			1
	17			1			1
	18			1			1
	19			1			1
	1A			1			1
	1B			1			1
	1C			1			1
	1D			1			1
	1E			1			1
	1F			1			1
Section 1	20			1			1
	21			1			1
	22			1			1
	23			1			1
	24			1			1
	25			1			1
	26		0000 0013	1		0000 0013	1
	27	0000 0013		1	0000 0013		1
	28			1			1
	29			1			1
	2A			1			1
	2B			1			1
	2C			1			1
	2D			1			1
	2E			1			1
	2F			1			1
	30			1			1
	31			1			1
	32			1			1
	33			1			1
	34			1			1
	35			1			1
	36			1			1
	37			1			1
	38			1			1
	39			1			1
	3A			1			1
	3B			1			1
	3C			1			1
	3D			1			1
	3E			1			1
	3F			1			1

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	M	Field (0 , 32) (Hex)	Field (32 , 32) (Hex)	Y
Section 2	40			1			0
	41			1			0
	42			1			0
	43			1			0
	44			1			0
	45			1			0
	46		0000 0003	1		0000 0003	0
	47	FFFF FFFE		1	FFFF FFFE		0
	48			1			0
	49			1			0
	4A			1			0
	4B			1			0
	4C			1			0
	4D			1			0
	4E			1			0
	4F			1			0
	50			1			0
	51			1			0
	52			1			0
	53			1			0
	54			1			0
	55			1			0
	56			1			0
	57			1			0
	58			1			0
	59			1			0
	5A			1			0
	5B			1			0
	5C			1			0
	5D			1			0
	5E			1			0
	5F			1			0
Section 3	60			0			0
	61			0			0
	62			0			0
	63			0			0
	64			0			0
	65			0			0
	66			0			0
	67			0			0
	68			0			0
	69			0			0
	6A			0			0
	6B			0			0
	6C			0			0
	6D			0			0
	6E			0			0
	6F			0			0
	70			0			0
	71			0			0
	72			0			0
	73			0			0
	74			0			0
	75			0			0
	76			0			0
	77			0			0
	78			0			0
	79			0			0
	7A			0			0
	7B			0			0
	7C			0			0
	7D			0			0
	7E			0			0
	7F			0			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (32, 32) (Hex)	M	Field (0, 32) (Hex)	Field (32, 32) (Hex)	Y
Section 4	80			0			0
	81			0			0
	82			0			0
	83			0			0
	84			0			0
	85			0			0
	86			0			0
	87			0			0
	88			0			0
	89			0			0
	8A			0			0
	8B			0			0
	8C			0			0
	8D			0			0
	8E			0			0
	8F			0			0
	90			0			0
	91			0			0
	92			0			0
	93			0			0
	94			0			0
	95			0			0
	96			0			0
	97			0			0
	98			0			0
	99			0			0
	9A			0			0
	9B			0			0
	9C			0			0
	9D			0			0
	9E			0			0
	9F			0			0
Section 5	A0			1			1
	A1			1			1
	A2			1			1
	A3			1			1
	A4			1			1
	A5			1			1
	A6		FFFF FFFB	1		FFFF FFFB	1
	A7	FFFF FFFC		1	FFFF FFFC		1
	A8			1			1
	A9			1			1
	AA			1			1
	AB			1			1
	AC			1			1
	AD			1			1
	AE			1			1
	AF			1			1
	B0			1			1
	B1			1			1
	B2			1			1
	B3			1			1
	B4			1			1
	B5			1			1
	B6			1			1
	B7			1			1
	B8			1			1
	B9			1			1
	BA			1			1
	BB			1			1
	BC			1			1
	BD			1			1
	BE			1			1
	BF			1			1

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0, 32) (Hex)	Field (32,32) (Hex)	M	Field (0,32) (Hex)	Field (32, 32) (Hex)	Y
Section 6	C0			1			1
	C1			1			1
	C2			1			1
	C3			1			1
	C4			1			1
	C5			1			1
	C6		FFFF FFFF	1		FFFF FFFF	1
	C7	0000 0002		1	0000 0002		1
	C8			1			1
	C9			1			1
	CA			1			1
	CB			1			1
	CC			1			1
	CD			1			1
	CE			1			1
	CF			1			1
	D0			1			1
	D1			1			1
	D2			1			1
	D3			1			1
	D4			1			1
	D5			1			1
	D6			1			1
	D7			1			1
	D8			1			1
	D9			1			1
	DA			1			1
	DB			1			1
	DC			1			1
	DD			1			1
	DE			1			1
	DF			1			1
Section 7	E0			1			0
	E1			1			0
	E2			1			0
	E3			1			0
	E4			1			0
	E5			1			0
	E6		FFFF FFFE	1		FFFF FFFE	0
	E7	FFFF FFFC		1	FFFF FFFC		0
	E8			1			0
	E9			1			0
	EA			1			0
	EB			1			0
	EC			1			0
	ED			1			0
	EE			1			0
	EF			1			0
	F0			1			0
	F1			1			0
	F2			1			0
	F3			1			0
	F4			1			0
	F5			1			0
	F6			1			0
	F7			1			0
	F8			1			0
	F9			1			0
	FA			1			0
	FB			1			0
	FC			1			0
	FD			1			0
	FE			1			0
	FF			1			0

GEWMA — Words Mod-32 Greater Than Or Equal To Words Mod-32 In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a_1 is greater than or equal to the word mod-32 pointed to by a_2 in the same section. The addresses are based on the mixed mode mod-32. (Fig. 2-1 and Fig. 2-2.)

Format

Label	Command	Argument
Symbol	GEWMA	a_1, a_2

. **Label** — Any valid symbol or blank

. **Command** — GEWMA

. **Argument** — Two entries are required. Both entries represent the words mod-32 that are compared with each other.

Notes

(1) This instruction is the same as the instruction GEWM, except that it loads the argument to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The addresses of a_1 and a_2 can have the same values.

(4) Before the instruction is called, the registers AS, and M must be set properly.

(5) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

(6) The field pointers FP3 and FP1 are used.

GECM — Words Mod-32 Greater Than Or Equal To Common Register In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by FP3 is greater than or Equal to the Common register. The addresses are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	GECM	Blank

. Label — Any valid symbol or blank.

. Command — GECM

. Argument — Blank

Notes

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) Before the instruction is called, the register AS, M and FP3 must be set properly.

(3) The associative array address X'FB' in the sections which participate in this instruction must be preset ot X'7FFF FFFF' and this address cannot be used as an operand.

Example:

Assume AS = X'8000 0000'
FP3 = X'02'
C = X'0000 0002'

and M is as shown in the following tables, then the instruction GECM has the following operations:

(1) In Section 0, no operation
(2) In Section 1, $2 \geq 2$ is true
(3) In Section 2, $17 \geq 2$ is true
(4) In Section 3, no operation
(5) In Section 4, $0 \geq 2$ is false

(6) In Section 5, $-1 \geq 2$ is false
(7) In Section 6, $5 \geq 2$ is true
(8) In Section 7, $-2 \geq 2$ is false

Note that those memory locations which are not shown or are left blank are not changed.

	Word Addr. (Hex)	Before Execution: Field (0 , 32) (Hex)	Before Execution: Field (32 , 32) (Hex)	M	After Execution: Field (0,32) (Hex)	After Execution: Field (32 , 32) (Hex)	Y
Section 0	00			0			0
	01			0			0
	02			0			0
	03			0			0
	04			0			0
	05			0			0
	06			0			0
	07			0			0
	08			0			0
	09			0			0
	0A			0			0
	0B			0			0
	0C			0			0
	0D			0			0
	0E			0			0
	0F			0			0
	10			0			0
	11			0			0
	12			0			0
	13			0			0
	14			0			0
	15			0			0
	16			0			0
	17			0			0
	18			0			0
	19			0			0
	1A			0			0
	1B			0			0
	1C			0			0
	1D			0			0
	1E			0			0
	1F			0			0
Section 1	20			1			1
	21			1			1
	22	0000 0002		1	0000 0002		1
	23			1			1
	24			1			1
	25			1			1
	26			1			1
	27			1			1
	28			1			1
	29			1			1
	2A			1			1
	2B			1			1
	2C			1			1
	2D			1			1
	2E			1			1
	2F			1			1
	30			1			1
	31			1			1
	32			1			1
	33			1			1
	34			1			1
	35			1			1
	36			1			1
	37			1			1
	38			1			1
	39			1			1
	3A			1			1
	3B			1			1
	3C			1			1
	3D			1			1
	3E			1			1
	3F			1			1

	Word Addr. (Hex)	Before Execution: Field (0, 32) (Hex)	Before Execution: Field (32 , 32) (Hex)	M	After Execution: Field (0,32) (Hex)	After Execution: Field (32 , 32) (Hex)	Y
Section 2	40			1			1
	41			1			1
	42	0000 0011		1	0000 0011		1
	43			1			1
	44			1			1
	45			1			1
	46			1			1
	47			1			1
	48			1			1
	49			1			1
	4A			1			1
	4B			1			1
	4C			1			1
	4D			1			1
	4E			1			1
	4F			1			1
	50			1			1
	51			1			1
	52			1			1
	53			1			1
	54			1			1
	55			1			1
	56			1			1
	57			1			1
	58			1			1
	59			1			1
	5A			1			1
	5B			1			1
	5C			1			1
	5D			1			1
	5E			1			1
	5F			1			1
Section 3	60			0			0
	61			0			0
	62			0			0
	63			0			0
	64			0			0
	65			0			0
	66			0			0
	67			0			0
	68			0			0
	69			0			0
	6A			0			0
	6B			0			0
	6C			0			0
	6D			0			0
	6E			0			0
	6F			0			0
	70			0			0
	71			0			0
	72			0			0
	73			0			0
	74			0			0
	75			0			0
	76			0			0
	77			0			0
	78			0			0
	79			0			0
	7A			0			0
	7B			0			0
	7C			0			0
	7D			0			0
	7E			0			0
	7F			0			0

	Word Addr. (Hex)	Before Execution: Field (0 , 32) (Hex)	Before Execution: Field (32 , 32) (Hex)	M	After Execution: Field (0 , 32) (Hex)	After Execution: Field (32, 32) (Hex)	Y
Section 4	80			1			0
	81			1			0
	82	0000 0000		1	0000 0000		0
	83			1			0
	84			1			0
	85			1			0
	86			1			0
	87			1			0
	88			1			0
	89			1			0
	8A			1			0
	8B			1			0
	8C			1			0
	8D			1			0
	8E			1			0
	8F			1			0
	90			1			0
	91			1			0
	92			1			0
	93			1			0
	94			1			0
	95			1			0
	96			1			0
	97			1			0
	98			1			0
	99			1			0
	9A			1			0
	9B			1			0
	9C			1			0
	9D			1			0
	9E			1			0
	9F			1			0
Section 5	A0			1			0
	A1			1			0
	A2	FFFF FFFF		1	FFFF FFFF		0
	A3			1			0
	A4			1			0
	A5			1			0
	A6			1			0
	A7			1			0
	A8			1			0
	A9			1			0
	AA			1			0
	AB			1			0
	AC			1			0
	AD			1			0
	AE			1			0
	AF			1			0
	B0			1			0
	B1			1			0
	B2			1			0
	B3			1			0
	B4			1			0
	B5			1			0
	B6			1			0
	B7			1			0
	B8			1			0
	B9			1			0
	BA			1			0
	BB			1			0
	BC			1			0
	BD			1			0
	BE			1			0
	BF			1			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (32 , 32) (Hex)	M	Field (0,32) (Hex)	Field (32 , 32) (Hex)	Y
Section 6	C0			1			1
	C1			1			1
	C2	0000 0005		1	0000 0005		1
	C3			1			1
	C4			1			1
	C5			1			1
	C6			1			1
	C7			1			1
	C8			1			1
	C9			1			1
	CA			1			1
	CB			1			1
	CC			1			1
	CD			1			1
	CE			1			1
	CF			1			1
	D0			1			1
	D1			1			1
	D2			1			1
	D3			1			1
	D4			1			1
	D5			1			1
	D6			1			1
	D7			1			1
	D8			1			1
	D9			1			1
	DA			1			1
	DB			1			1
	DC			1			1
	DD			1			1
	DE			1			1
	DF			1			1
Section 7	E0			1			0
	E1			1			0
	E2	FFFF FFFE		1	FFFF FFFE		0
	E3			1			0
	E4			1			0
	E5			1			0
	E6			1			0
	E7			1			0
	E8			1			0
	E9			1			0
	EA			1			0
	EB			1			0
	EC			1			0
	ED			1			0
	EE			1			0
	EF			1			0
	F0			1			0
	F1			1			0
	F2			1			0
	F3			1			0
	F4			1			0
	F5			1			0
	F6			1			0
	F7			1			0
	F8			1			0
	F9			1			0
	FA			1			0
	FB			1			0
	FC			1			0
	FD			1			0
	FE			1			0
	FF			1			0

GECMA — Words Mod-32 Greater Than Or Equal To Common Register In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a is greater than or equal to the Common register. The addresses are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	GECMA	a

. Label — Any valid symbol or blank.

. Command — GECMA

. Argument — One entry is required. This entry represents the words mod-32 that are compared with the Common register.

.. a — a is a number between 0 and 255.

Notes

(1) This instruction is the same as the instruction GECM, except that it loads the argument to a field pointer.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) Before the instruction is called, the registers AS, and M must be set properly.

(4) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

(5) The field pointer FP3 is used.

LTWM — Words Mod-32 Less Tham Words Mod-32 In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by FP3 is less than the word mod-32 pointed to by FP1 in the same section. The address are based on the mixed mode mod-32 (Fig. 2-1 and Fi. 2-2).

Format

Label	Command	Argument
Symbol	LTWM	Blank

. **Label** — Any valid symbol or blank.

. **Command** — LTWM

. **Argument** — Blank

Notes

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The addresses of FP3 and FP1 can have the same values.

(3) Before the instruction is called, the registers AS, M, FP3 and FP1 must be set properly.

(4) The associative array address X'FB' is the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

Example:

Assume AS = X'8000 0000'
FP3 = X'04'
FP1 = X'67'

and M is as shown in the following tables, then the instruction LTWM has the following operations.

(1) In Section 0, 9 < 3 is false
(2) In Section 1, -5 < -2 is true
(3) In Section 2, 5 < 16 is true
(4) In Section 3, no operation
(5) In Section 4, no operation

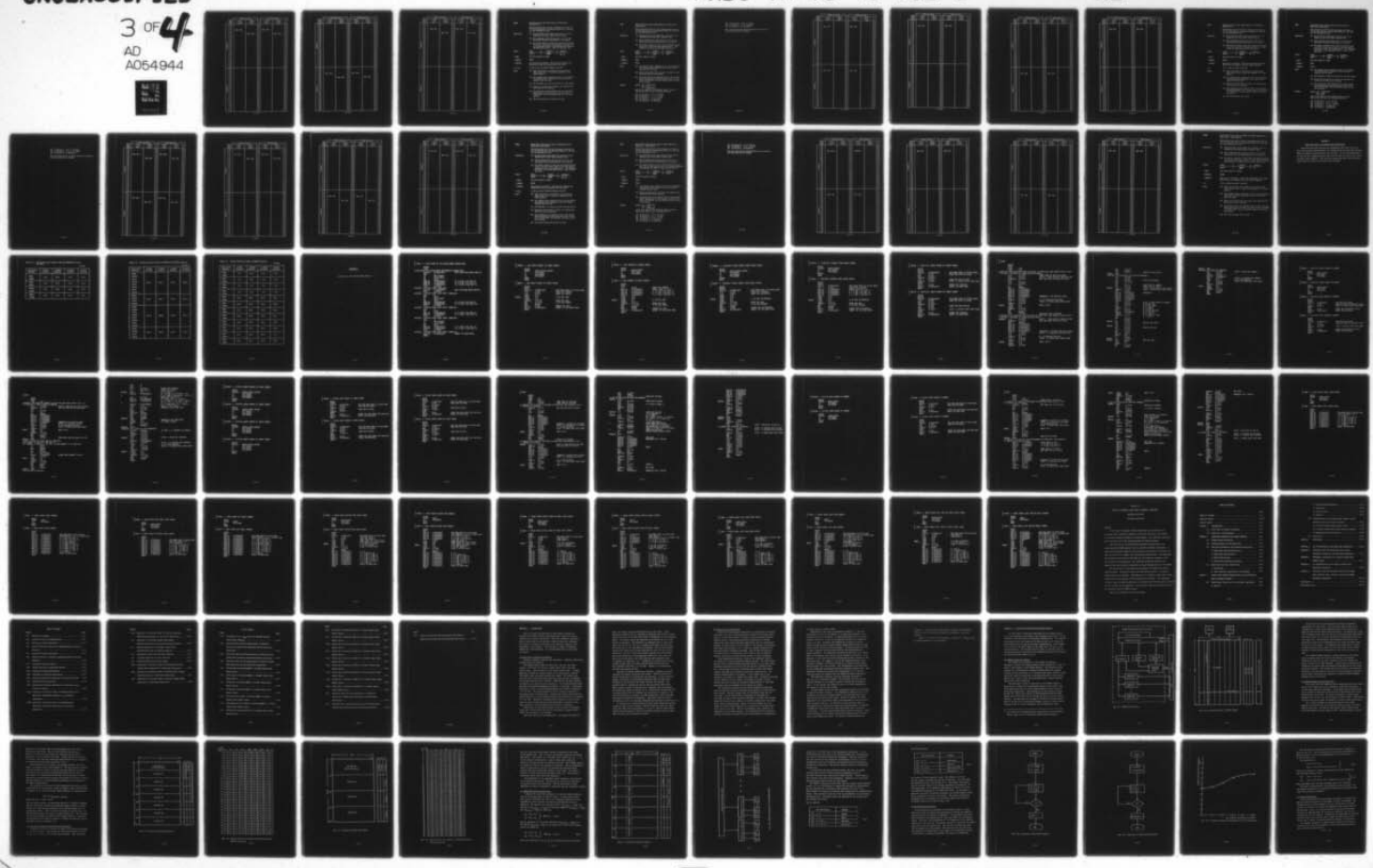
(6) In Section 5, $-2 < 9$ is true
(7) In Section 6, $4 < -4$ is false
(8) In Section 7, no operation

Note that those memory locations which are not shown or are left blank are not changed.

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (96,32) (Hex)	M	Field (0,32) (Hex)	Field (96,32) (Hex)	Y
Section 0	00			1			0
	01			1			0
	02			1			0
	03			1			0
	04	0000 0009		1	0000 0009		0
	05			1			0
	06			1			0
	07		0000 0003	1		0000 0003	0
	08			1			0
	09			1			0
	0A			1			0
	0B			1			0
	0C			1			0
	0D			1			0
	0E			1			0
	0F			1			0
	10			1			0
	11			1			0
	12			1			0
	13			1			0
	14			1			0
	15			1			0
	16			1			0
	17			1			0
	18			1			0
	19			1			0
	1A			1			0
	1B			1			0
	1C			1			0
	1D			1			0
	1E			1			0
	1F			1			0
Section 1	20			1			1
	21			1			1
	22			1			1
	23			1			1
	24	FFFF FFFB		1	FFFF FFFB		1
	25			1			1
	26			1			1
	27		FFFF FFFE	1		FFFF FFFE	1
	28			1			1
	29			1			1
	2A			1			1
	2B			1			1
	2C			1			1
	2D			1			1
	2E			1			1
	2F			1			1
	30			1			1
	31			1			1
	32			1			1
	33			1			1
	34			1			1
	35			1			1
	36			1			1
	37			1			1
	38			1			1
	39			1			1
	3A			1			1
	3B			1			1
	3C			1			1
	3D			1			1
	3E			1			1
	3F			1			1

AD-A054 944 SYRACUSE UNIV N Y F/G 9/2
LARGE SCALE INFORMATION SYSTEMS. VOLUME III.(U)
MAR 78 F30602-74-C-0335
UNCLASSIFIED RADC-TR-78-43-VOL-3 NL

3 OF 4
AD
A054944



	Word Addr. (Hex)	Before Execution: Field (0, 32) (Hex)	Before Execution: Field (96, 32) (Hex)	Before Execution: M	After Execution: Field (0,32) (Hex)	After Execution: Field (96,32) (Hex)	After Execution: Y
Section 2	40			1			1
	41			1			1
	42			1			1
	43			1			1
	44	0000 0005		1	0000 0005		1
	45			1			1
	46			1			1
	47		0000 0010	1		0000 0010	1
	48			1			1
	49			1			1
	4A			1			1
	4B			1			1
	4C			1			1
	4D			1			1
	4E			1			1
	4F			1			1
	50			1			1
	51			1			1
	52			1			1
	53			1			1
	54			1			1
	55			1			1
	56			1			1
	57			1			1
	58			1			1
	59			1			1
	5A			1			1
	5B			1			1
	5C			1			1
	5D			1			1
	5E			1			1
	5F			1			1
Section 3	60			0			0
	61			0			0
	62			0			0
	63			0			0
	64			0			0
	65			0			0
	66			0			0
	67			0			0
	68			0			0
	69			0			0
	6A			0			0
	6B			0			0
	6C			0			0
	6D			0			0
	6E			0			0
	6F			0			0
	70			0			0
	71			0			0
	72			0			0
	73			0			0
	74			0			0
	75			0			0
	76			0			0
	77			0			0
	78			0			0
	79			0			0
	7A			0			0
	7B			0			0
	7C			0			0
	7D			0			0
	7E			0			0
	7F			0			0

	Word Addr. (Hex)	Before Execution: Field (0,32) (Hex)	Before Execution: Field (96,32) (Hex)	M	After Execution: Field (0,32) (Hex)	After Execution: Field (96,32) (Hex)	Y
Section 4	80			0			0
	81			0			0
	82			0			0
	83			0			0
	84			0			0
	85			0			0
	86			0			0
	87			0			0
	88			0			0
	89			0			0
	8A			0			0
	8B			0			0
	8C			0			0
	8D			0			0
	8E			0			0
	8F			0			0
	90			0			0
	91			0			0
	92			0			0
	93			0			0
	94			0			0
	95			0			0
	96			0			0
	97			0			0
	98			0			0
	99			0			0
	9A			0			0
	9B			0			0
	9C			0			0
	9D			0			0
	9E			0			0
	9F			0			0
Section 5	A0			1			1
	A1			1			1
	A2			1			1
	A3			1			1
	A4	FFFF FFFE		1	FFFF FFFE		1
	A5			1			1
	A6			1			1
	A7		0000 0009	1		0000 0009	1
	A8			1			1
	A9			1			1
	AA			1			1
	AB			1			1
	AC			1			1
	AD			1			1
	AE			1			1
	AF			1			1
	B0			1			1
	B1			1			1
	B2			1			1
	B3			1			1
	B4			1			1
	B5			1			1
	B6			1			1
	B7			1			1
	B8			1			1
	B9			1			1
	BA			1			1
	BB			1			1
	BC			1			1
	BD			1			1
	BE			1			1
	BF			1			1

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (96, 32) (Hex)	M	Field (0, 32) (Hex)	Field (96 , 32) (Hex)	Y
Section 6	C0			1			0
	C1			1			0
	C2			1			0
	C3			1			0
	C4	0000 0004		1	0000 0004		0
	C5			1			0
	C6			1			0
	C7		FFFF FFFC	1		FFFF FFFC	0
	C8			1			0
	C9			1			0
	CA			1			0
	CB			1			0
	CC			1			0
	CD			1			0
	CE			1			0
	CF			1			0
	D0			1			0
	D1			1			0
	D2			1			0
	D3			1			0
	D4			1			0
	D5			1			0
	D6			1			0
	D7			1			0
	D8			1			0
	D9			1			0
	DA			1			0
	DB			1			0
	DC			1			0
	DD			1			0
	DE			1			0
	DF			1			0
Section 7	E0			0			0
	E1			0			0
	E2			0			0
	E3			0			0
	E4			0			0
	E5			0			0
	E6			0			0
	E7			0			0
	E8			0			0
	E9			0			0
	EA			0			0
	EB			0			0
	EC			0			0
	ED			0			0
	EE			0			0
	EF			0			0
	F0			0			0
	F1			0			0
	F2			0			0
	F3			0			0
	F4			0			0
	F5			0			0
	F6			0			0
	F7			0			0
	F8			0			0
	F9			0			0
	FA			0			0
	FB			0			0
	FC			0			0
	FD			0			0
	FE			0			0
	FF			0			0

LTWMA — Word Mod-32 Less Than Words Mod-32 In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a_1 is less than the word mod-32 pointed to by a_2 in the same section. The addresses are based on the mixed mode mod-32. (Fig. 2-1 and Fig. 2-2)

Format

Label	Command	Argument
Symbol	LTWMA	a_1, a_2

. **Label** — Any valid symbol or blank.

. **Command** — LTWMA

. **Argument** — Two entries are required. Both entries represent the words mod-32 that are compared with each other.

.. a_1, a_2 — a_1 and a_2 are the number between 0 and 255.

Notes

(1) This instruction is the same as the instruction LTWM, except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The addresses of a_1 and a_2 can have the same values.

(4) Before the instruction is called, the registers AS and M must be set properly.

(5) The associative address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

(6) The field pointers FP3 and FP1 are used.

LTCM — Words Mod-32 Less Than Common Register In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by FP3 is less than the Common register. The addresses are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	LTCM	Blank

. Label — Any valid symbol or blank.

. Command — LTCM

. Argument — Blank

Notes

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) Before the instruction is called, the registers AS, M and FP3 must be set properly.

(3) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

Example

Assume AS = X'8000 0000'
FP3 = X'04'
C = X'0000 0004'

and M is as shown in the following tables, then the instruction LTCM has the following operations.

(1) In Section 0, 9 < 4 is false
(2) In Section 1, -5 < 4 is true
(3) In Section 2, 1 < 4 is true
(4) In Section 3, no operation
(5) In Section 4, no operation

(6) In Section 5, -16 < 4 is true
(7) In Section 6, 4 < 4 is false
(8) In Section 7, no operation

Note that those memory locations which are not shown or are left blank are not changed.

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (96,32) (Hex)	M	Field (0 , 32) (Hex)	Field (96 ,32) (Hex)	Y
Section 0	00			1			0
	01			1			0
	02			1			0
	03			1			0
	04	0000 0009		1	0000 0009		0
	05			1			0
	06			1			0
	07			1			0
	08			1			0
	09			1			0
	0A			1			0
	0B			1			0
	0C			1			0
	0D			1			0
	0E			1			0
	0F			1			0
	10			1			0
	11			1			0
	12			1			0
	13			1			0
	14			1			0
	15			1			0
	16			1			0
	17			1			0
	18			1			0
	19			1			0
	1A			1			0
	1B			1			0
	1C			1			0
	1D			1			0
	1E			1			0
	1F			1			0
Section 1	20			1			1
	21			1			1
	22			1			1
	23			1			1
	24	FFFF FFFB		1	FFFF FFFB		1
	25			1			1
	26			1			1
	27			1			1
	28			1			1
	29			1			1
	2A			1			1
	2B			1			1
	2C			1			1
	2D			1			1
	2E			1			1
	2F			1			1
	30			1			1
	31			1			1
	32			1			1
	33			1			1
	34			1			1
	35			1			1
	36			1			1
	37			1			1
	38			1			1
	39			1			1
	3A			1			1
	3B			1			1
	3C			1			1
	3D			1			1
	3E			1			1
	3F			1			1

	Word Addr. (Hex)	Before Execution: Field (0, 32) (Hex)	Before Execution: Field (96, 32) (Hex)	M	After Execution: Field (0, 32) (Hex)	After Execution: Field (96, 32) (Hex)	Y
Section 2	40			1			1
	41			1			1
	42			1			1
	43			1			1
	44	0000 0001		1	0000 0001		1
	45			1			1
	46			1			1
	47			1			1
	48			1			1
	49			1			1
	4A			1			1
	4B			1			1
	4C			1			1
	4D			1			1
	4E			1			1
	4F			1			1
	50			1			1
	51			1			1
	52			1			1
	53			1			1
	54			1			1
	55			1			1
	56			1			1
	57			1			1
	58			1			1
	59			1			1
	5A			1			1
	5B			1			1
	5C			1			1
	5D			1			1
	5E			1			1
	5F			1			1
Section 3	60			0			0
	61			0			0
	62			0			0
	63			0			0
	64			0			0
	65			0			0
	66			0			0
	67			0			0
	68			0			0
	69			0			0
	6A			0			0
	6B			0			0
	6C			0			0
	6D			0			0
	6E			0			0
	6F			0			0
	70			0			0
	71			0			0
	72			0			0
	73			0			0
	74			0			0
	75			0			0
	76			0			0
	77			0			0
	78			0			0
	79			0			0
	7A			0			0
	7B			0			0
	7C			0			0
	7D			0			0
	7E			0			0
	7F			0			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0, 32) (Hex)	Field (96, 32) (Hex)	M	Field (0, 32) (Hex)	Field (96, 32) (Hex)	Y
Section 4	80			0			0
	81			0			0
	82			0			0
	83			0			0
	84			0			0
	85			0			0
	86			0			0
	87			0			0
	88			0			0
	89			0			0
	8A			0			0
	8B			0			0
	8C			0			0
	8D			0			0
	8E			0			0
	8F			0			0
	90			0			0
	91			0			0
	92			0			0
	93			0			0
	94			0			0
	95			0			0
	96			0			0
	97			0			0
	98			0			0
	99			0			0
	9A			0			0
	9B			0			0
	9C			0			0
	9D			0			0
	9E			0			0
	9F			0			0
Section 5	A0			1			1
	A1			1			1
	A2			1			1
	A3			1			1
	A4	FFFF FFF0		1	FFFF FFF0		1
	A5			1			1
	A6			1			1
	A7			1			1
	A8			1			1
	A9			1			1
	AA			1			1
	AB			1			1
	AC			1			1
	AD			1			1
	AE			1			1
	AF			1			1
	B0			1			1
	B1			1			1
	B2			1			1
	B3			1			1
	B4			1			1
	B5			1			1
	B6			1			1
	B7			1			1
	B8			1			1
	B9			1			1
	BA			1			1
	BB			1			1
	BC			1			1
	BD			1			1
	BE			1			1
	BF			1			1

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (96,32) (Hex)	M	Field (0,32) (Hex)	Field (96 , 32) (Hex)	Y
Section 6	C0			1			0
	C1			1			0
	C2			1			0
	C3			1			0
	C4	0000 0004		1	0000 0004		0
	C5			1			0
	C6			1			0
	C7			1			0
	C8			1			0
	C9			1			0
	CA			1			0
	CB			1			0
	CC			1			0
	CD			1			0
	CE			1			0
	CF			1			0
	D0			1			0
	D1			1			0
	D2			1			0
	D3			1			0
	D4			1			0
	D5			1			0
	D6			1			0
	D7			1			0
	D8			1			0
	D9			1			0
	DA			1			0
	DB			1			0
	DC			1			0
	DD			1			0
	DE			1			0
	DF			1			0
Section 7	E0			0			0
	E1			0			0
	E2			0			0
	E3			0			0
	E4			0			0
	E5			0			0
	E6			0			0
	E7			0			0
	E8			0			0
	E9			0			0
	EA			0			0
	EB			0			0
	EC			0			0
	ED			0			0
	EE			0			0
	EF			0			0
	F0			0			0
	F1			0			0
	F2			0			0
	F3			0			0
	F4			0			0
	F5			0			0
	F6			0			0
	F7			0			0
	F8			0			0
	F9			0			0
	FA			0			0
	FB			0			0
	FC			0			0
	FD			0			0
	FE			0			0
	FF			0			0

LTCMA — Words Mod-32 Less Than Common Register In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a is less than the Common register. The addresses are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	LTCMA	a

. Label — Any valid symbol or blank

. Command — LTCMA

. Argument — One entry is required. This entry represents the words mod-32 that are compared with the Common register.

.. a — a is a number between 0 and 255.

Notes

(1) This instruction is the same as the instruction LTCM, except that it loads the argument to a field pointer.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) Before the instruction is called, the registers AS and M must be set properly.

(4) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

(5) The field pointer FP3 is used.

LEWM — **Words Mod-32 Less Than Or Equal To Words Mod-32 In Mixed Mode (Fixed-Point)**

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by FP3 is less than or equal to the word mod-32 pointed to by FP1 in the same section. The addresses are based on the mixed mode mod-32. (Fig. 2-1 and Fig. 2-2)

Format

Label	Command	Argument
Symbol	LEWM	Blank

. **Label** — Any valid symbol or blank.

. **Command** — LEWM

. **Argument** — Blank

Notes

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) The addresses of FP3 and FP1 can have the same values.

(3) Before the instruction is called, the registers AS, M, FP3 and FP1 must be set properly.

(4) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

Example

Assume AS = X'8000 0000'
FP3 = X'04'
FP1 = X'67'
and M is as shown in the following tables, then the instructions LEWM has the following operations.

(1) In Section 0, $9 \leq 9$ is true
(2) In Section 1, $-5 \leq 3$ is true
(3) In Section 2, $-4 \leq -2$ is true
(4) In Section 3, no operation
(5) In Section 4, no operation

(6) In Section 5, $23 \leq -4$ is false
(7) In Section 6, $-5 \leq -9$ is false
(8) In Section 7, no operation

Note that those memory locations which are not shown or are left blank are not changed.

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0, 32) (Hex)	Field (96, 32) (Hex)	M	Field (0, 32) (Hex)	Field (96, 32) (Hex)	Y
Section 0	00			1			1
	01			1			1
	02			1			1
	03			1			1
	04	0000 0009		1	0000 0009		1
	05			1			1
	06			1			1
	07		0000 0009	1		0000 0009	1
	08			1			1
	09			1			1
	0A			1			1
	0B			1			1
	0C			1			1
	0D			1			1
	0E			1			1
	0F			1			1
	10			1			1
	11			1			1
	12			1			1
	13			1			1
	14			1			1
	15			1			1
	16			1			1
	17			1			1
	18			1			1
	19			1			1
	1A			1			1
	1B			1			1
	1C			1			1
	1D			1			1
	1E			1			1
	1F			1			1
Section 1	20			1			1
	21			1			1
	22			1			1
	23			1			1
	24	FFFF FFFB		1	FFFF FFFB		1
	25			1			1
	26			1			1
	27		0000 0003	1		0000 0003	1
	28			1			1
	29			1			1
	2A			1			1
	2B			1			1
	2C			1			1
	2D			1			1
	2E			1			1
	2F			1			1
	30			1			1
	31			1			1
	32			1			1
	33			1			1
	34			1			1
	35			1			1
	36			1			1
	37			1			1
	38			1			1
	39			1			1
	3A			1			1
	3B			1			1
	3C			1			1
	3D			1			1
	3E			1			1
	3F			1			1

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0, 32) (Hex)	Field (96, 32) (Hex)	M	Field (0, 32) (Hex)	Field (96, 32) (Hex)	Y
Section 2	40			1			1
	41			1			1
	42			1			1
	43			1			1
	44	FFFF FFFC		1	FFFF FFFC		1
	45			1			1
	46			1			1
	47		FFFF FFFE	1		FFFF FFFE	1
	48			1			1
	49			1			1
	4A			1			1
	4B			1			1
	4C			1			1
	4D			1			1
	4E			1			1
	4F			1			1
	50			1			1
	51			1			1
	52			1			1
	53			1			1
	54			1			1
	55			1			1
	56			1			1
	57			1			1
	58			1			1
	59			1			1
	5A			1			1
	5B			1			1
	5C			1			1
	5D			1			1
	5E			1			1
	5F			1			1
Section 3	60			0			0
	61			0			0
	62			0			0
	63			0			0
	64			0			0
	65			0			0
	66			0			0
	67			0			0
	68			0			0
	69			0			0
	6A			0			0
	6B			0			0
	6C			0			0
	6D			0			0
	6E			0			0
	6F			0			0
	70			0			0
	71			0			0
	72			0			0
	73			0			0
	74			0			0
	75			0			0
	76			0			0
	77			0			0
	78			0			0
	79			0			0
	7A			0			0
	7B			0			0
	7C			0			0
	7D			0			0
	7E			0			0
	7F			0			0

	Word Addr. (Hex)	Before Execution: Field (0,32) (Hex)	Before Execution: Field (96, 32) (Hex)	M	After Execution: Field (0, 32) (Hex)	After Execution: Field (96, 32) (Hex)	Y
Section 4	80			0			0
	81			0			0
	82			0			0
	83			0			0
	84			0			0
	85			0			0
	86			0			0
	87			0			0
	88			0			0
	89			0			0
	8A			0			0
	8B			0			0
	8C			0			0
	8D			0			0
	8E			0			0
	8F			0			0
	90			0			0
	91			0			0
	92			0			0
	93			0			0
	94			0			0
	95			0			0
	96			0			0
	97			0			0
	98			0			0
	99			0			0
	9A			0			0
	9B			0			0
	9C			0			0
	9D			0			0
	9E			0			0
	9F			0			0
Section 5	A0			1			0
	A1			1			0
	A2			1			0
	A3			1			0
	A4	0000 0017		1	0000 0017		0
	A5			1			0
	A6			1			0
	A7		FFFF FFFC	1		FFFF FFFC	0
	A8			1			0
	A9			1			0
	AA			1			0
	AB			1			0
	AC			1			0
	AD			1			0
	AE			1			0
	AF			1			0
	B0			1			0
	B1			1			0
	B2			1			0
	B3			1			0
	B4			1			0
	B5			1			0
	B6			1			0
	B7			1			0
	B8			1			0
	B9			1			0
	BA			1			0
	BB			1			0
	BC			1			0
	BD			1			0
	BE			1			0
	BF			1			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0,32) (Hex)	Field (96,32) (Hex)	M	Field (0,32) (Hex)	Field (96 , 32) (Hex)	Y
Section 6	C0			1			0
	C1			1			0
	C2			1			0
	C3			1			0
	C4	FFFF FFFB		1	FFFF FFFB		0
	C5			1			0
	C6			1			0
	C7		FFFF FFF7	1		FFFF FFF7	0
	C8			1			0
	C9			1			0
	CA			1			0
	CB			1			0
	CC			1			0
	CD			1			0
	CE			1			0
	CF			1			0
	D0			1			0
	D1			1			0
	D2			1			0
	D3			1			0
	D4			1			0
	D5			1			0
	D6			1			0
	D7			1			0
	D8			1			0
	D9			1			0
	DA			1			0
	DB			1			0
	DC			1			0
	DD			1			0
	DE			1			0
	DF			1			0
Section 7	E0			0			0
	E1			0			0
	E2			0			0
	E3			0			0
	E4			0			0
	E5			0			0
	E6			0			0
	E7			0			0
	E8			0			0
	E9			0			0
	EA			0			0
	EB			0			0
	EC			0			0
	ED			0			0
	EE			0			0
	EF			0			0
	F0			0			0
	F1			0			0
	F2			0			0
	F3			0			0
	F4			0			0
	F5			0			0
	F6			0			0
	F7			0			0
	F8			0			0
	F9			0			0
	FA			0			0
	FB			0			0
	FC			0			0
	FD			0			0
	FE			0			0
	FF			0			0

LEWMA — **Words Mod-32 Less Than Or Equal To Words Mod-32 In Mixed Mode (Fixed-Point)**

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a_1 is less than or equal to the word mod-32 pointed to by a_2 in the same section. The addresses are based on the mixed mode mod-32. (Fig. 2-1 and Fig. 2-2)

Format

Label	Command	Argument
Symbol	LEWMA	a_1, a_2

. **Label** — Any valid sumbol or blank.

. **Command** — LEWMA

. **Argument** — Two entries are required. Both entries represent the words mod-32 that are compared with each other.

.. a_1,a_2 — a_1 and a_2 are the numbers between 0 and 255.

Notes

(1) This instruction is the same as the instruction LEWM, except that it loads the arguments to the field pointers.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(3) The addresses of a_1 and a_2 can have the same values.

(4) Before the instruction is called, the regsiters AS and M must be set properly.

(5) The associative array address X'FB' in the section which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

(6) The field pointers FP3 and FP1 are used.

LECM — Words Mod-32 Less Than Or Equal To Common Register In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true.

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by FP3 is less than or equal to the Common register. The addresses are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	LECM	Blank

. Label — Any valid symbol or blank.

. Command — LECM

. Argument — Blank

Notes

(1) The response store registers X and Y in the associative memory modules selected by the Array Select register (AS) are used.

(2) Before the instruction is called, the registers AS, M and FP3 must be set properly.

(3) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

Example

Assume AS = X'8000 000'
FP3 = X'04'
C = X'000 0004'

and M is as shown in the following tables, then the instruction LECM has the following operation.

(1) In Section 0, $9 \leq 4$ is false
(2) In Section 1, $-5 \leq 4$ is true
(3) In Section 2, $4 \leq 4$ is true
(4) In Section 3, no operation
(5) In Section 4, no operation

(6) In Section 5, $-4 \leq 4$ is true
(7) In Section 6, $7 \leq 7$ is false
(8) In Section 7, no operation

Note that those memory locations which are not shown or are left blank are not changed.

	Word Addr. (Hex)	Before Execution: Field (0, 32) (Hex)	Before Execution: Field (96, 32) (Hex)	M	After Execution: Field (0, 32) (Hex)	After Execution: Field (96,32) (Hex)	Y
Section 0	00			1			0
	01			1			0
	02			1			0
	03			1			0
	04	0000 0009		1	0000 0009		0
	05			1			0
	06			1			0
	07			1			0
	08			1			0
	09			1			0
	0A			1			0
	0B			1			0
	0C			1			0
	0D			1			0
	0E			1			0
	0F			1			0
	10			1			0
	11			1			0
	12			1			0
	13			1			0
	14			1			0
	15			1			0
	16			1			0
	17			1			0
	18			1			0
	19			1			0
	1A			1			0
	1B			1			0
	1C			1			0
	1D			1			0
	1E			1			0
	1F			1			0
Section 1	20			1			1
	21			1			1
	22			1			1
	23			1			1
	24	FFFF FFFB		1	FFFF FFFB		1
	25			1			1
	26			1			1
	27			1			1
	28			1			1
	29			1			1
	2A			1			1
	2B			1			1
	2C			1			1
	2D			1			1
	2E			1			1
	2F			1			1
	30			1			1
	31			1			1
	32			1			1
	33			1			1
	34			1			1
	35			1			1
	36			1			1
	37			1			1
	38			1			1
	39			1			1
	3A			1			1
	3B			1			1
	3C			1			1
	3D			1			1
	3E			1			1
	3F			1			1

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0 , 32) (Hex)	Field (96 , 32) (Hex)	M	Field (0, 32) (Hex)	Field (96 , 32) (Hex)	Y
Section 2	40			1			1
	41			1			1
	42			1			1
	43			1			1
	44	0000 0004		1	0000 0004		1
	45			1			1
	46			1			1
	47			1			1
	48			1			1
	49			1			1
	4A			1			1
	4B			1			1
	4C			1			1
	4D			1			1
	4E			1			1
	4F			1			1
	50			1			1
	51			1			1
	52			1			1
	53			1			1
	54			1			1
	55			1			1
	56			1			1
	57			1			1
	58			1			1
	59			1			1
	5A			1			1
	5B			1			1
	5C			1			1
	5D			1			1
	5E			1			1
	5F			1			1
Section 3	60			0			0
	61			0			0
	62			0			0
	63			0			0
	64			0			0
	65			0			0
	66			0			0
	67			0			0
	68			0			0
	69			0			0
	6A			0			0
	6B			0			0
	6C			0			0
	6D			0			0
	6E			0			0
	6F			0			0
	70			0			0
	71			0			0
	72			0			0
	73			0			0
	74			0			0
	75			0			0
	76			0			0
	77			0			0
	78			0			0
	79			0			0
	7A			0			0
	7B			0			0
	7C			0			0
	7D			0			0
	7E			0			0
	7F			0			0

		Before Execution			After Execution		
	Word Addr. (Hex)	Field (0, 32) (Hex)	Field (96, 32) (Hex)	M	Field (0 , 32) (Hex)	Field (96 , 32) (Hex)	Y
Section 4	80			0			0
	81			0			0
	82			0			0
	83			0			0
	84			0			0
	85			0			0
	86			0			0
	87			0			0
	88			0			0
	89			0			0
	8A			0			0
	8B			0			0
	8C			0			0
	8D			0			0
	8E			0			0
	8F			0			0
	90			0			0
	91			0			0
	92			0			0
	93			0			0
	94			0			0
	95			0			0
	96			0			0
	97			0			0
	98			0			0
	99			0			0
	9A			0			0
	9B			0			0
	9C			0			0
	9D			0			0
	9E			0			0
	9F			0			0
Section 5	A0			1			1
	A1			1			1
	A2			1			1
	A3			1			1
	A4	FFFF FFFC		1	FFFF FFFC		1
	A5			1			1
	A6			1			1
	A7			1			1
	A8			1			1
	A9			1			1
	AA			1			1
	AB			1			1
	AC			1			1
	AD			1			1
	AE			1			1
	AF			1			1
	B0			1			1
	B1			1			1
	B2			1			1
	B3			1			1
	B4			1			1
	B5			1			1
	B6			1			1
	B7			1			1
	B8			1			1
	B9			1			1
	BA			1			1
	BB			1			1
	BC			1			1
	BD			1			1
	BE			1			1
	BF			1			1

	Word Addr. (Hex)	Before Execution: Field (0,32) (Hex)	Before Execution: Field (96,32) (Hex)	Before Execution: M	After Execution: Field (0,32) (Hex)	After Execution: Field (96,32) (Hex)	After Execution: Y
Section 6	C0			1			0
	C1			1			0
	C2			1			0
	C3			1			0
	C4	0000 0007		1	0000 0007		0
	C5			1			0
	C6			1			0
	C7			1			0
	C8			1			0
	C9			1			0
	CA			1			0
	CB			1			0
	CC			1			0
	CD			1			0
	CE			1			0
	CF			1			0
	D0			1			0
	D1			1			0
	D2			1			0
	D3			1			0
	D4			1			0
	D5			1			0
	D6			1			0
	D7			1			0
	D8			1			0
	D9			1			0
	DA			1			0
	DB			1			0
	DC			1			0
	DD			1			0
	DE			1			0
	DF			1			0
Section 7	E0			0			0
	E1			0			0
	E2			0			0
	E3			0			0
	E4			0			0
	E5			0			0
	E6			0			0
	E7			0			0
	E8			0			0
	E9			0			0
	EA			0			0
	EB			0			0
	EC			0			0
	ED			0			0
	EE			0			0
	EF			0			0
	F0			0			0
	F1			0			0
	F2			0			0
	F3			0			0
	F4			0			0
	F5			0			0
	F6			0			0
	F7			0			0
	F8			0			0
	F9			0			0
	FA			0			0
	FB			0			0
	FC			0			0
	FD			0			0
	FE			0			0
	FF			0			0

LECMA — Words Mod-32 Less Than Or Equal To Common Register In Mixed Mode (Fixed-Point)

This instruction will set the Y response store bits to 1's for each section of associative memory if, and only if, the following is true:

Conditions

(1) The particular array which this section is in is enabled by the Array Select register (AS).

(2) The M response store bits are set to 1's for the particular section participating in the search.

(3) The search criteria is met; the word mod-32 pointed to by a is less than or equal to the Common register. The addresses are based on the mixed mode mod-32.

Format

Label	Command	Argument
Symbol	LECMA	a

. **Label** — Any valid symbol or blank.

. **Command** — LECMA

. **Argument** — One entry is required. This entry represents the words mod-32 that are compared with the Common register.

.. **a** — a is a number between 0 and 255.

Notes

(1) This instruction is the same as the instruction LECM, except that it loads the argument to a field pointer.

(2) The response store registers X and Y in the associative memory modules selected by the Array Select (AS) are used.

(3) Before the instruction is called, the registers AS and M must be set properly.

(4) The associative array address X'FB' in the sections which participate in this instruction must be preset to X'7FFF FFFF' and this address cannot be used as an operand.

(5) The field pointer FP3 is used.

CHAPTER 3

Execution Times of the Mixed Mode Instructions

Since the mixed mode routines are implemented such that all of the bits are processed simultaneously, the average execution times will increase while the number of parallel words is increased. In the following tables, we list all the average execution times for one word, two words, four words and eight words parallel. Note that the average execution times increased slowly when the number of parallel words increased.

Table 3-1. Average Execution Times of ADD and SUBTRACT Routines (32 bits)

(μ.sec)

Instruction Mnemonic	1 Word Parallel	2 Words Parallel	4 Words Parallel	8 Words Parallel
ADWM ADWMA	11.5	13.3	14.3	15.5
ADCM ADCMA	11.3	12.4	13.6	15.6
SUBWM SUBWMA	11.3	13.0	14.0	15.8
SUBCM SUBCMA	10.9	12.0	13.7	15.2

Table 3-2. Average Execution Times of MULTIPLY and DIVIDE Routines

(μ sec.)

Instruction Mnemonic	1 Word Parallel	2 Words Parallel	4 Words Parallel	8 Words Parallel
MPUUM MPUUMA MPULM MPULMA MPLUM MPLUMA MPLLM MPLLMA	182.0	206.3	225.2	236.0
MPCUM MPCUMA MPCLM MPCLMA	105.6	108.5	113.3	117.5
DVUUM DVUUMA DVULM DVULMA DVLUM DVLUMA DVLLM DVLLMA	544.9	642.9	723.2	817.9
DVCUM DVCUMA DVCLM DVCLMA	544.3	595.8	657.7	720.0

Table 3-3. Average Execution Times of SEARCH Routines

(μ.sec)

Instruction Mnemonic	1 Word Parallel	2 Words Parallel	4 Words Parallel	8 Words Parallel
EQWM EQWMA	2.1	2.1	2.1	2.1
EQCM EQCMA	1.6	1.6	1.6	1.6
NEWM NEWMA	2.1	2.1	2.1	2.1
NECM NECMA	1.6	1.6	1.6	1.6
GTWM GTWMA	12.0	13.8	14.8	16.6
GTCM GTCMA	11.5	12.6	14.3	15.8
GEWM GEWMA	11.1	12.8	13.8	15.6
GECM GECMA	10.6	12.6	13.4	14.8
LTWM LTWMA	12.6	13.5	14.5	16.2
LTCM LTCMA	12.5	13.2	14.3	15.9
LEWM LEWMA	12.7	14.4	15.4	17.3
LECM LECMA	12.6	13.7	15.4	16.9

APPENDIX

Listing of the Mixed Mode Macros

```
*
* INIT --- INITIATOR OF THE MIXED MODE OPERATIONS
*
         MACRO
         INIT
**INITIATE THE MIXED MODE ARITHMETIC OPERATIONS
         GEN,32    X'3801481F'          SET THE MIXED MODE MOD-32
         CLR       X
         LI        CH,X'7FFF'
         LI        CL,X'FFFF'
         SC        X(0)
         GEN,32    X'00E0CCB3'          X = X OR (-32 ROT X)
         GEN,32    X'00C0CCB3'          X = X OR (-64 ROT X)
         GEN,32    X'0080CCB3'          X = X OR (-128 ROT X)
         S,W       X,X'FB'
**X'FB' = X'7FFF FFFF' (MOD-32)
         GEN,32    X'3801483F'          SET TO MIXED MODE MOD-64
         S,W       X,X'FB'
**X'FB' = X'7FFF FFFF 7FFF FFFF' (MOD-64)
         CLR       X
         SC        X(0)
         LI        CH,X'FFFF'
         SC        X(1)
         GEN,32    X'00C0CCB3'          X = X OR (-64 ROT X)
         GEN,32    X'0080CCB3'          X = X OR (-128 ROT X)
         S,W       X,X'FE'
**X'FE' = X'7FFF FFFF FFFF FFFF' (MOD-64)
         CLR       X
         SC        X(0)
         GEN,32    X'00C0CCB3'          X = X OR (-64 ROT X)
         GEN,32    X'0080CCB3'          X = X OR (-128 ROT X)
         S,W       X,X'FF'
**X'FF' = X'FFFF FFFF 0000 0000' (MOD-64)
         CLR       X
         LI        CH,X'0000'
         LI        CL,X'0001'
         SC        X(0)
         GEN,32    X'00C0CCB3'          X = X OR (-64 ROT X)
         GEN,32    X'0080CCB3'          X = X OR (-128 ROT X)
         S,W       X,X'F6'
**X'F6' = X'0000 0001 0000 0000' (MOD-64)
         GEN,32    X'380148FF'          RESET TO WORD MODE
         MEND
```

```
*
* ADWMA --- ADD ARRAY WORDS TO ARRAY WORDS
*
         MACRO
         ADWMA     &ARG1,&ARG2,&ARG3
         LI        FP3,&ARG1
         LI        FP1,&ARG2
         LI        FP2,&ARG3
         ADWM
         MEND
*
* ADWM --- ADD ARRAY WORDS TO ARRAY WORDS
*
         MACRO
         ADWM
         GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
         L,W       X,FP3                LOAD THE ADDEND
         L,W       Y,FP1                LOAD THE ADDER
         LAND      Y,M
A1&$X    LXOR      X,Y                  X IS THE SUM
         LANDN     Y,X
         LAND,W    Y,X'FB'              ZERO THE MSB
         ROT       Y,-1,32              Y IS THE CARRY
         BRS       A1&$X
         SM,W      X,FP2                STORE THE SUM
         GEN,32    X'380148FF'          RESET TO THE WORD MODE
         MEND
```

```
*
* ADCMA --- ADD COMMON TO ARRAY WORDS
*
          MACRO
          ADCMA     &ARG1,&ARG2
          LI        FP3,&ARG1
          LI        FP2,&ARG2
          ADCM
          MEND
*
* ADCM --- ADD COMMON TO ARRAY WORDS
*
          MACRO
          ADCM
          L,W       X,FP3                LOAD THE ADDEND
          GEN,32    X'42008840'          STORE COMM TO Y(0)
          GEN,32    X'40E09952'          Y = Y OR (-32 ROT Y)
          GEN,32    X'40C09952'          Y = Y OR (-64 ROT Y)
          GEN,32    X'40809952'          Y = Y OR (-128 ROT Y)
          LAND      X,M
          LAND      Y,M
A2&$X     LXOR      X,Y                  X IS THE SUM
          LANDN     Y,X
          LAND,W    Y,X'FB'              ZERO THE MSB
          ROT       Y,-1,32              Y IS THE CARRY
          BRS       A2&$X
          SM,W      X,FP2                STORE THE SUM
          GEN,32    X'380148FF'          RESET TO THE WORD MODE
          MEND
```

```
*
* SUBWMA --- SUBTRACT ARRAY WORDS FROM ARRAY WORDS
*
          MACRO
          SUBWMA    &ARG1,&ARG2,&ARG3
          LI        FP3,&ARG1
          LI        FP1,&ARG2
          LI        FP2,&ARG3
          SUBWM
          MEND
*
* SUBWM --- SUBTRACT ARRAY WORDS FROM ARRAY WORDS
*
          MACRO
          SUBWM
          GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
          L,W       X,FP3                LOAD THE MINUEND
          L,W       Y,FP1                LOAD THE SUBTRHEND
          LAND      X,M
          LAND      Y,M
A3&$X     LXOR      X,Y                  X IS THE DIFFERENCE
          LAND      Y,X
          LAND,W    Y,X'FB'              ZERO THE MSB
          ROT       Y,-1,32              Y IS THE BORROW
          BRS       A3&$X
          SM,W      X,FP2                STORE THE DIFFERENCE
          GEN,32    X'380148FF'          RESET TO THE WORD MODE
          MEND
```

```
*
* SUBCMA --- SUBTRACT COMMON FROM ARRAY WORDS
*
         MACRO
         SUBCMA    &ARG1,&ARG2
         LI        FP3,&ARG1
         LI        FP2,&ARG2
         SUBCM
         MEND
*
* SUBCM --- SUBTRACT COMMON FROM ARRAY WORDS
*
         MACRO
         SUBCM
         GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
         L,W       X,FP3                LOAD THE MINUEND
         GEN,32    X'42008840'          STORE COMM TO Y(0)
         GEN,32    X'40E09952'          Y = Y OR (-32 ROT Y)
         GEN,32    X'40C09952'          Y = Y OR (-64 ROT Y)
         GEN,32    X'40809952'          Y = Y OR (-128 ROT Y)
         LAND      X,M
         LAND      Y,M
A4&$X    LXOR      X,Y                  X IS THE DIFFERENCE
         LAND      Y,X
         LAND,W    Y,X'FB'              ZERO THE MSB
         ROT       Y,-1,32              Y IS THE BORROW
         BRS       A4&$X
         SM,W      X,FP2                STORE THE DIFFERENCE
         GEN,32    X'380148FF'          RESET TO THE WORD MODE
         MEND
```

```
*
* MPLUM --- MULTIPLY ARRAY WORDS BY ARRAY WORDS
*
        MACRO
        MPLUM
        GEN,32      X'3801483F'          SET WORD MODE TO MIXED MODE
        L,W         X,FP3                LOAD THE MULTIPLICAND
        ROT         X,-32
        LAND,W      X,X'FF'
        L,W         Y,FP1                LOAD THE MULTIPLIER
        LAND,W      Y,X'FF'              X'FF' = X'FFFF FFFF 0000 000(
        MPWM
        SM,W        X,FP2                STORE THE PRODUCT
        GEN,32      X'380148FF'          RESET TO WORD MODE
        MEND
*
* MPLLM --- MULTIPLY ARRAY WORDS BY ARRAY WORDS
*
        MACRO
        MPLLM
        GEN,32      X'3801483F'          SET WORD MODE TO MIXED MODE
        L,W         X,FP3                LOAD THE MULTIPLICAND
        ROT         X,-32
        LAND,W      X,X'FF'
        L,W         Y,FP1                LOAD THE MULTIPLIER
        ROT         Y,-32
        LAND,W      Y,X'FF'              X'FF' = X'FFFF FFFF 0000 000(
        MPWM
        SM,W        X,FP2                STORE THE PRODUCT
        GEN,32      X'380148FF'          RESET TO WORD MODE
        MEND
```

```
*
* MPWM
*
         MACRO
         MPWM
         LAND      X,M
         LAND      Y,M
**CONVERT THE NEGATIVE MULTIPLICANDS TO POSITIVE AND STOR X'FFFF FFFF'
* TO THE V VECTORS OF THOSE MULTIPLICANDS
         S,W       X,X'F9'              TEMP STOR OF MULTIPLICAND
         S,W       Y,X'FC'              X'FC' = TEMP STOR OF MULTIPLIER
         LANDN,W   X,X'FE'              GET THE SIGN OF MULTIPLICAND
         CLR       Y
         S,W       Y,X'F5'
         S,W       Y,X'F4'
         L         Y,X
         BNR       XN1&$X
         GEN,32    X'403F99B3'
         GEN,32    X'403E99B3'
         GEN,32    X'403C99B3'
         GEN,32    X'403899B3'
         GEN,32    X'40309983'          GENERATE V OF MULTIPLICAND
         S,W       X,X'F5'
         LXOR,W    X,X'F9'              X = V XOR MULTIPLICAND
         L,W       Y,X'F6'              X'F6' = X'0000 0001 0000 0000'
         LAND,W    Y,X'F5'
A5&$X    LXOR      X,Y                  ADD Y TO X
         LANDN     Y,X
         LAND,W    Y,X'FB'
         ROT       Y,-1,32
         BR        A5&$X
         S,W       X,X'F9'              POSITIVE MULTIPLICAND
**CONVERT THE NEGATIVE MULTIPLIERS TO POSITIVE, AND STORE X'FFFF FFFF'
* TO THE V VECTORS OF THOSE MULTIPLIERS
XN1&$X   L,W       Y,X'FC'              X'FC' = TEMP STOR OF MULTIPLIER
         LANDN,W   Y,X'FE'              GET THE SIGN OF MULTIPLIER
         BNR       YN1&$X
         GEN,32    X'403F9952'
         GEN,32    X'403E9952'
         GEN,32    X'403C9952'
         GEN,32    X'40389952'
         GEN,32    X'00309952'          GENERATE V VECTOR FOR MULTIPLIER
         S,W       Y,X'F4'              X'F4' = V VECTOR OF MULTIPLIER
         L,W       X,X'FC'
         LXOR,W    X,Y                  X = V XOR MULTIPLIER
         L,W       Y,X'F6'              X'F6' = X'0000 0001 0000 0000'
         LAND,W    Y,X'F4'
A6&$X    LXOR      X,Y                  ADD Y TO X
         LANDN     Y,X
         LAND,W    Y,X'FB'
```

```
         ROT       Y,-1,32
         BRS       A6&$X
         S,W       X,X'FC'                 POSITIVE MULTIPLIER
YN1&$X   ANOP
**DO THE MULTIPLICATION OF POSITIVE NUMBERS
         L,W       X,X'F9'
         L,W       Y,X'FC'
         LI        FPE,31                  FPE IS A COUNTER
         ROT       Y,-31,64
         CLR       X
         S,W       X,X'FA'                 TEMP STOR OF PRODUCT
         S,W       X,X'F8'                 TEMP STOR OF CARRY
HE1&$X   S,W       Y,X'FC'                 X'FC' = TEMP STOR OF MULTIPLIER
         LANDN,W   Y,X'FE'                 X'FE' = X'7FFF FFFF FFFF FFFF'
         BNR       YR1&$X
         GEN,32    X'403F9952'             GENERATE V
         GEN,32    X'403E9952'
         GEN,32    X'403C9952'
         GEN,32    X'40389952'
         GEN,32    X'00309952'
         LAND,W    Y,X'F9'
         S,W       Y,X'F7'                 X'F7' = TEMP STORE OF ADDEND
         L,W       X,X'FA'                 X IS THE SUM
         L,W       Y,X'F8'                 Y IS THE CARRY
         LXOR      X,Y                     X = X XOR Y
         LXOR,W    X,X'F7'                 X = X XOR A
         S,W       X,X'FA'                 X IS THE NEW SUM
         GEN,32    X'0100AAA2'             X = Y OR NOT X
         LAND,W    X,X'F7'                 X = X AND A
         LANDN,W   Y,X'FA'                 Y = Y AND NOT SUM
         LOR       Y,X                     Y = Y OR X
         S,W       Y,X'F8'
         B         JM1&$X
YR1&$X   L,W       Y,X'F8'
         ROT       Y,1,64                  ROTATE THE CARRY
         S,W       Y,X'F8'
JM1&$X   L,W       X,X'FA'
         ROT       X,1,64                  ROTATE THE SUM
         S,W       X,X'FA'
         L,W       Y,X'FC'
         ROT       Y,1,64
         BZ,FPE    AD1&$X
         DECR      FPE
         B         HE1&$X
AD1&$X   L,W       Y,X'F8'
AG1&$X   BNR       DO1&$X                  ADD THE CARRY
         LXOR      X,Y
         LANDN     Y,X
         ROT       Y,-1,64
         B         AG1&$X
```

```
DO1&$X   ANOP
**ADJUST THE SIGNS OF PRODUCTS
         S,W       X,X'FA'              X'FA' = POSITIVE PRODUCT
         L,W       Y,X'F5'
         LXOR,W    Y,X'F4'
         GEN,32    X'00209952'
         S,W       Y,X'F4'              X'F4' = V VECTOR FOR PRODUCT
         LXOR      X,Y                  X = V XOR PRODUCT
         L,W       Y,X'F6'              X'F6' = X'0000 0001 0000 0000'
         ROT       Y,32,64
         LAND,W    Y,X'F4'
A7&$X    LXOR      X,Y
         LANDN     Y,X
         LAND,W    Y,X'FE'
         ROT       Y,-1,64
         BRS       A7&$X
         MEND
```

```
*
* MPCUMA --- MULTIPLY ARRAY WORDS BY COMMON
*
        MACRO
        MPCUMA    &ARG1,&ARG2
        LI        FP3,&ARG1
        LI        FP2,&ARG2
        MPCUM
        MEND
*
* MPCLMA --- MULTIPLY ARRAY WORDS BY COMMON
*
        MACRO
        MPCLMA    &ARG1,&ARG2
        LI        FP3,&ARG1
        LI        FP2,&ARG2
        MPCLM
        MEND
*
* MPCUM --- MULTIPLY ARRAY WORDS BY COMMON
*
        MACRO
        MPCUM
        GEN,32    X'3801483F'          SET THE MIXED MODE
        L,W       X,FP3                LOAD THE MULTIPLICAND(32 BITS)
        LAND,W    X,X'FF'              X'FF' = X'FFFF FFFF 0000 0000'
        MPCM
        SM,W      X,FP2                STORE THE PRODUCT(64 BITS)
        GEN,32    X'380148FF'          RESET TO WORD MODE
        MEND
*
* MPCLM --- MULTIPLY ARRAY WORDS BY COMMON
*
        MACRO
        MPCLM
        GEN,32    X'3801483F'          SET THE MIXED MODE
        L,W       X,FP3                LOAD THE MULTIPLICAND(32 BITS)
        ROT       X,-32,64
        LAND,W    X,X'FF'              X'FF' = X'FFFF FFFF 0000 0000'
        MPCM
        SM,W      X,FP2                STORE THE PRODUCT(64 BITS)
        GEN,32    X'380148FF'          RESET TO WORD MODE
        MEND
```

```
*
* MPCM
*
          MACRO
          MPCM
          LAND      X,M
**CONVERT THE NEGATIVE NUMBERS TO POSITIVE AND STOR X'FFFF FFFF' TO
* V VECTORS OF THOSE NEGATIVE NUMBERS
          S,W       X,X'F9'                 X'F9' = TEM STOR OF MULTIPLICANE
          LANDN,W   X,X'FE'                 GET THE SIGN OF MULTIPLICAND
          CLR       Y
          S,W       Y,X'F5'
          L         Y,X
          BNR       XN2&$X
          GEN,32    X'403F99B3'
          GEN,32    X'403E99B3'
          GEN,32    X'403C99B3'
          GEN,32    X'403899B3'
          GEN,32    X'403099B3'             GENERATE V OF MULTIPLICAND
          S,W       X,X'F5'                 X'F5' = V OF MULTIPLICAND
          LXOR,W    X,X'F9'                 X = V XOR MULTIPLICAND
          L,W       Y,X'F6'                 X'F6' = X'0000 0001 0000 0000'
          LAND,W    Y,X'F5'
A8&$X     LXOR      X,Y                     ADD Y TO X
          LANDN     Y,X
          LAND,W    Y,X'FB'
          ROT       Y,-1,32
          BRS       A8&$X
          S,W       X,X'F9'                 STOR POSI MULTIPLICAND IN X'F9'
XN2&$X    ANOP
**NOW, THE MULTIPLICANDS ARE ALL POSITIVE
**IF COMMON IS POSITIVE, SET FPE = 0
**IF COMMON IS NEGATIVE, SET FPE = 1 AND CONVERT IT TO POSITIVE
          LI        FP1,0
          LI        FPE,0
          BBZ       COM&$X
          LI        FPE,1
          LRR       BL,ASH
          LI        ASH,X'8000'
          GEN,32    X'420044A0'             STORE (NOT COMMON) TO X(0)
          L,W       Y,X'F6'
A9&$X     LXOR      X,Y
          LANDN     Y,X
          LAND,W    Y,X'FB'
          ROT       Y,-1,32
          BRS       A9&$X
          LCW       X(0)
          LRR       ASH,BL
COM&$X    ANOP
**DO THE MULTIPLICATION
```

```
          CLR       X
          CLR       Y
          S,W       X,X'FA'              CLEAR THE PRODUCT
          LI        FP1,X'1F'            LOAD 31 TO FP1
HE2&$X    LXOR      X,Y                  X = S XOR C
          GEN,32    X'46F9C3A5'          X = S XOR C IF C(FP1) = 0;
*                                        X = S XOR C XOR M(F9)  IF C = 1
          S,W       X,X'FA'              SAVE THE SUM TO P1
          GEN,32    X'4200AAA2'          X = NOT X OR Y = S OR C
          GEN,32    X'46F927A5'          X = X AND A = (S OR C) AND A
*                                        IF C = 1 ; X = 0 IF C = 0
          GEN,32    X'46FA6645'          Y = Y AND NOT P1 = C AND S
          LOR       Y,X                  THIS IS THE CARRY
          L,W       X,X'FA'              RELOAD X WITH THE SUM
          ROT       X,1,64
          DECR      FP1
          BNZ,FP1   HE2&$X
          ROT       X,1,64
          ROT       Y,1,64
AG2&$X    BNR       DO2&$X
          LXOR      X,Y                  COMPLETE THE ADDITION
          LANDN     Y,X                  Y = NOT X AND Y
          ROT       Y,-1,64
          B         AG2&$X
DO2&$X    ANOP
**ADJUST  THE SIGNS OF THE PRODUCTS
          BZ,FPE    PO2&$X               IF FPE = 0, BRANCH TO PO2&$X
          LN,W      Y,X'F5'
          LAND,W    Y,X'FF'
          SM,W      Y,X'F5'
PO2&$X    S,W       X,X'FA'              X'FA' = POSITIVE PRODUCT
          L,W       Y,X'F5'
          GEN,32    X'40209952'
          S,W       Y,X'F4'              X'F4' = V VECTOR FOR PRODUCT
          LXOR      X,Y                  X = V XOR PRODUCT
          L,W       Y,X'F6'              X'F6' = X'0000 0001 0000 0000'
          ROT       Y,32,64
          LAND,W    Y,X'F4'
AA&$X     LXOR      X,Y
          LANDN     Y,X
          LAND,W    Y,X'FE'
          ROT       Y,-1,64
          BRS       AA&$X
          MEND
```

```
*
* DVUUMA --- DIVIDE ARRAY WORDS BY ARRAY WORDS
*
        MACRO
        DVUUMA    &ARG1,&ARG2,&ARG3
        LI        FP3,&ARG1
        LI        FP1,&ARG2
        LI        FP2,&ARG3
        DVUUM
        MEND
*
* DVULMA --- DIVIDE ARRAY WORDS BY ARRAY WORDS
*
        MACRO
        DVULMA    &ARG1,&ARG2,&ARG3
        LI        FP3,&ARG1
        LI        FP1,&ARG2
        LI        FP2,&ARG3
        DVULM
        MEND
*
* DVLUMA --- DIVIDE ARRAY WORDS BY ARRAY WORDS
*
        MACRO
        DVLUMA    &ARG1,&ARG2,&ARG3
        LI        FP3,&ARG1
        LI        FP1,&ARG2
        LI        FP2,&ARG3
        DVLUM
        MEND
*
* DVLLMA --- DIVIDE ARRAY WORDS BY ARRAY WORDS
*
        MACRO
        DVLLMA    &ARG1,&ARG2,&ARG3
        LI        FP3,&ARG1
        LI        FP1,&ARG2
        LI        FP2,&ARG3
        DVLLM
        MEND
```

```
*
* DVUUM --- DIVIDE ARRAY WORDS BY ARRAY WORDS
*
        MACRO
        DVUUM
        GEN,32      X'3801483F'         SET THE WORD MODE TO MIXED MODE
        L,W         X,FP3               LOAD THE DIVIDEND
        LAND,W      X,X'FF'
        L,W         Y,FP1               LOAD THE DIVISOR
        LAND,W      Y,X'FF'
        DVWM
        SM,W        X,FP2               STORE THE REMAINDER AND QUOTIENT
        GEN,32      X'380148FF'         RESET TO WORD MODE
        MEND
*
* DVULM --- DIVIDE ARRAY WORDS BY ARRAY WORDS
*
        MACRO
        DVULM
        GEN,32      X'3801483F'         SET THE WORD MODE TO MIXED MODE
        L,W         X,FP3               LOAD THE DIVIDEND
        LAND,W      X,X'FF'
        L,W         Y,FP1               LOAD THE DIVISOR
        ROT         Y,-32,64
        DVWM
        SM,W        X,FP2               STORE THE REMAINDER AND QUOTIENT
        GEN,32      X'380148FF'         RESET TO WORD MODE
        MEND
```

```
*
* DVLUM --- DIVIDE ARRAY WORDS BY ARRAY WORDS
*
        MACRO
        DVLUM
        GEN,32      X'3801483F'          SET THE WORD MODE TO MIXED MODE
        L,W         X,FP3                LOAD THE DIVIDEND
        ROT         X,-32,64
        LAND,W      X,X'FF'
        L,W         Y,FP1                LOAD THE DIVISOR
        LAND,W      Y,X'FF'
        DVWM
        SM,W        X,FP2                STORE THE REMAINDER AND QUOTIENT
        GEN,32      X'380148FF'          RESET TO WORD MODE
        MEND
*
* DVLLM --- DIVIDE ARRAY WORDS BY ARRAY WORDS
*
        MACRO
        DVLLM
        GEN,32      X'3801483F'          SET THE WORD MODE TO MIXED MODE
        L,W         X,FP3                LOAD THE DIVIDEND
        ROT         X,-32,64
        LAND,W      X,X'FF'
        L,W         Y,FP1                LOAD THE DIVISOR
        ROT         Y,-32,64
        LAND,W      Y,X'FF'
        DVWM
        SM,W        X,FP2                STORE THE REMAINDER AND QUOTIENT
        GEN,32      X'380148FF'          RESET TO WORD MODE
        MEND
```

```
*
* DVWM
*
          MACRO
          DVWM
          LAND      X,M
          LAND      Y,M
          S,W       X,X'FA'              TEMP STOR OF DIVIDEND
          S,W       Y,X'F9'              TEMP STOR OF DIVISOR
** CONVERT THE NEGATIVE DIVIDENDS TO POSITIVE, AND GENERATE THE V
*  VECTOR OF DIVIDENDS
          LANDN,W   X,X'FE'              GET THE SIGN OF DIVIDEND
          CLR       Y
          S,W       Y,X'F5'
          S,W       Y,X'F4'
          L         Y,X
          BNR       XNO&$X
          GEN,32    X'403F99B3'
          GEN,32    X'403E99B3'
          GEN,32    X'403C99B3'
          GEN,32    X'40389B3'
          GEN,32    X'40309B3'           GENERATE V VECTOR OF DIVIDEND
          S,W       X,X'F5'              X'F5' = V VECTOR FOR DIVIDEND
          LXOR,W    X,X'FA'              X = V XOR DIVIDEND
          L,W       Y,X'F6'              X'F6' = X'0000 0001 0000 0000'
          LAND,W    Y,X'F5'
AB&$X     LXOR      X,Y                  ADD Y TO X
          LANDN     Y,X
          LAND,W    Y,X'FB'
          ROT       Y,-1,32
          BRS       AB&$X
          S,W       X,X'FA'              POSITIVE DIVIDEND
** CONVERT THE NEGATIVE DIVISORS TO POSITIVE, AND GENERATE THE V
*  VECTOR OF DIVISORS
XNO&$X    L,W       Y,X'F9'              X'F9' = TEMP STOR OF DIVISOR
          LANDN,W   Y,X'FE'              GET THE SIGN OF DIVISOR
          BNR       YN3&$X
          GEN,32    X'403F9952'
          GEN,32    X'403E9952'
          GEN,32    X'403C9952'
          GEN,32    X'40389952'
          GEN,32    X'40309952'          GENERATE V VECTOR FOR DIVISOR
          S,W       Y,X'F4'              X'F4' = V VECTOR OF DIVISOR
          L,W       X,X'F9'
          LXOR,W    X,Y                  X = V XOR DIVISOR
          L,W       Y,X'F6'              X'F6' = X'0000 0001 0000 0000'
          LAND,W    Y,X'F4'
AC&$X     LXOR      X,Y                  ADD Y TO X
          LANDN     Y,X
          LAND,W    Y,X'FB'
```

```
          ROT       Y,-1,32
          BRS       AC&$X
          S,W       X,X'F9'                 POSITIVE DIVISOR
** DO THE DIVISION OF POSITIVE NUMBERS
YN3&$X    ANOP
          L,W       X,X'FA'                 POSITIVE DIVIDEND
          CLR       Y
          S,W       Y,X'FC'                 V = X'FC'; STORE V
          ROT       X,32,64
          LORN,W    X,X'FE'
          ROT       X,-1,64
          LI        FPE,32
AG3&$X    L,W       Y,X'F9'                 LOAD THE DIVISOR
*                                           X = X ADD/SUB Y
CO3&$X    LXOR      X,Y                     X = X XOR Y
          S,W       Y,X'FD'                 SC = X'FD'; M(SC) = Y; SAVE Y
          LXOR,W    X,X'FC'                 V = X'FC'; X = X XOR M(V)
          L         Y,X                     Y = X
          LAND,W    Y,X'FD'                 Y = Y AND M(SC);
*                                           FORM CARRY AND BORROW
          LXOR,W    X,X'FC'                 X = X XOR M(V); RESTORE X
          BNR       DO3&$XP                 COP OPERATION IS COMPLETE
          LAND,W    Y,X'FE'                 X'FE' = X'7FFF FFFF FFFF FFFF'
*                                           ZERO THE SIGN BIT
          ROT       Y,-1,32                 ROTATE CARRY/BORROW VECTOR
          B         CO3&$X
DO3&$XP   ANOP
          L         Y,X
          LANDN,W   X,X'FE'                 GET SIGN
          GEN,32    X'401F99B3'             GENERATE THE V VECTOR
          GEN,32    X'401E99B3'
          GEN,32    X'401C99B3'
          GEN,32    X'401899B3'
          GEN,32    X'401099B3'
          S,W       X,X'FC'                 NEW V
          L         X,Y
          ROT       X,-1,64
          DECR      FPE
          BNZ,FPE   AG3&$X
          L         Y,X
          LANDN,W   X,X'FF'
          ROT       Y,1,64
          LAND,W    Y,X'FF'
          LOR       X,Y
          SET       Y
          S,W       Y,X'FC'                 STORE V
          L         Y,X
          LANDN,W   Y,X'FE'                 GET SIGN
          BNR       NO3&$X
          GEN,32    X'401F9972'             GENERATE THE V VECTOR
```

```
          GEN,32    X'401E9972'
          GEN,32    X'401C9972'
          GEN,32    X'40189972'
          GEN,32    X'40109972'
          LAND,W    Y,X'F9'
          S,W       X,X'FA'
          LAND,W    X,X'FF'
CO4&$X    LXOR      X,Y
          S,W       Y,X'FD'
          LXOR,W    X,X'FC'
          L         Y,X
          LAND,W    Y,X'FD'
          LXOR,W    X,X'FC'
          BNR       DO4&$X
          LAND,W    Y,X'FE'
          ROT       Y,-1,32
          B         CO4&$X
DO4&$X    ANOP
NO3&$X    ANOP
          LN,W      Y,X'FA'
          LANDN,W   Y,X'FF'
          LOR       X,Y
          S,W       X,X'FA'          X'FA' = POSITIVE (R AND Q)
** ADJUST THE SIGNS OF (R AND Q)
          L,W       X,X'F4'          X'F4' = V VECTOR FOR DIVISOR
          LXOR,W    X,X'F5'          X'F5' = V VECTOR FOR DIVIDEND
          L         Y,X
          LAND,W    Y,X'F6'          X'F6' = X'0000 0001 0000 0000'
          ROT       X,32,64
          ROT       Y,32,64
          LOR,W     X,X'F5'
          S,W       X,X'FC'
          S,W       Y,X'FD'
          L,W       Y,X'F6'
          LAND,W    Y,X'FC'
          LOR,W     Y,X'FD'
          LXOR,W    X,X'FA'
AD&$X     LXOR      X,Y
          LANDN     Y,X
          LAND,W    Y,X'FB'
          ROT       Y,-1,32
          BRS       AD&$X
          MEND
```

```
*
* DVCUMA --- DIVIDE ARRAY WORDS BY COMMON
*
        MACRO
        DVCUMA    &ARG1,&ARG2
        LI        FP3,&ARG1
        LI        FP2,&ARG2
        DVCUM
        MEND
*
* DVCLMA --- DIVIDE ARRAY WORDS BY COMMON
*
        MACRO
        DVCLMA    &ARG1,&ARG2
        LI        FP3,&ARG1
        LI        FP2,&ARG2
        DVCLM
        MEND
```

```
*
* DVCUM --- DIVIDE ARRAY WORDS BY COMMON
*
        MACRO
        DVCUM
        GEN,32      X'3801483F'          SET THE WORD MODE TO MIXED MODE
        L,W         X,FP3                LOAD THE DIVIDEND
        LAND,W      X,X'FF'
        DVCM
        SM,W        X,FP2                STORE THE REMAINDER AND QUOTIENT
        GEN,32      X'380148FF'          RESET TO WORD MODE
        MEND
*
* DVCLM --- DIVIDE ARRAY WORDS BY COMMON
*
        MACRO
        DVCLM
        GEN,32      X'3801483F'          SET THE WORD MODE TO MIXED MODE
        L,W         X,FP3                LOAD THE DIVIDEND
        ROT         X,-32,64
        LAND,W      X,X'FF'
        DVCM
        SM,W        X,FP2                STORE THE REMAINDER AND QUOTIENT
        GEN,32      X'380148FF'          RESET TO WORD MODE
        MEND.
```

```
*
* DVCM
*
          MACRO
          DVCM
          LAND      X,M
          S,W       X,X'FA'              TEMP STOR OF DIVIDEND
** CONVERT THE NEGATIVE DIVIDENDS TO POSITIVE, AND GENERATE THE
*  V VECTOR OF DIVIDENDS
          LANDN,W   X,X'FE'              GET THE SIGN OF DIVIDEND
          CLR       Y
          S,W       Y,X'F5'
          S,W       Y,X'F4'
          L         Y,X
          BNR       XNO&$X
          GEN,32    X'403F99B3'
          GEN,32    X'403E99B3'
          GEN,32    X'403C99B3'
          GEN,32    X'403899B3'
          GEN,32    X'403099B3'          GENERATE V VECTOR OF DIVIDEND
          S,W       X,X'F5'              X'F5' = V VECTOR FOR DIVIDEND
          LXOR,W    X,X'FA'              X = V XOR DIVIDEND
          L,W       Y,X'F6'              X'F6' = X'0000 0001 0000 0000'
          LAND,W    Y,X'F5'
A0&$X     LXOR      X,Y                  ADD Y TO X
          LANDN     Y,X
          LAND,W    Y,X'FB'
          ROT       Y,-1,32
          BRS       A0&$X
          S,W       X,X'FA'              POSITIVE DIVIDEND
XNO&$X    ANOP
** CONVERT THE NEGATIVE DIVISOR(COMMON) TO POSITIVE, AND GENERATE
*  THE V VECTOR OF DIVISORS
          GEN,32    X'42008840'          STORE COMM TO Y(0)
          GEN,32    X'40C09952'          Y = Y OR (-64 ROT Y)
          GEN,32    X'40809952'          Y = Y OR (-128 ROT Y)
          LAND      Y,M
          S,W       Y,X'F9'              TEMP STOR OF DIVISOR
          LANDN,W   Y,X'FE'              GET THE SIGN OF DIVISOR
          BNR       YNO&$X
          GEN,32    X'403F9952'
          GEN,32    X'403E9952'
          GEN,32    X'403C9952'
          GEN,32    X'40389952'
          GEN,32    X'40309952'          GENERATE V VECTOR FOR DIVISOR
          S,W       Y,X'F4'              X'F4' = V VECTOR OF DIVISOR
          L,W       X,X'F9'
          LXOR,W    X,Y                  X = V XOR DIVISOR
          L,W       Y,X'F6'              X'F6' = X'0000 0001 0000 0000'
          LAND,W    Y,X'F4'
```

```
A1&$X    LXOR       X,Y                 ADD Y TO X
         LANDN      Y,X
         LAND,W     Y,X'FB'
         ROT        Y,-1,32
         BRS        A1&$X
         S,W        X,X'F9'             POSITIVE DIVISOR
** DO THE DIVISION OF POSITIVE NUMBERS
YNO&$X   ANOP
         L,W        X,X'FA'             POSITIVE DIVIDEND
         CLR        Y
         S,W        Y,X'FC'             V = X'FC'; STORE V
         ROT        X,32,64
         LORN,W     X,X'FE'
         ROT        X,-1,64
         LI         FPE,32              FPE IS USED AS A COUNTER
AGA&$X   L,W        Y,X'F9'             LOAD THE DIVISOR
*                                       X = X ADD/SUB Y
COP&$X   LXOR       X,Y                 X = X XOR Y
         S,W        Y,X'FD'             SC = X'FD'; M(SC) = Y; SAVE Y
         LXOR,W     X,X'FC'             V = X'FC'; X = X XOR M(V)
         L          Y,X                 Y = X
         LAND,W     Y,X'FD'             Y = Y AND M(SC);
*                                       FORM CARRY AND BORROW
         LXOR,W     X,X'FC'             X = X XOR M(V); RESTORE X
         BNR        DCP&$X              COP OPERATION IS COMPLETE
         LAND,W     Y,X'FE'             X'FE' = X'7FFF FFFF FFFF FFFF'
*                                       ZERO THE SIGN BIT
         ROT        Y,-1,32             ROTATE CARRY/BORROW VECTOR
         B          COP&$X
DCP&$X   ANOP
         L          Y,X
         LANDN,W    X,X'FE'             GET SIGN
         GEN,32     X'401F99B3'         GENERATE THE V VECTOR
         GEN,32     X'401E99B3'
         GEN,32     X'401C99B3'
         GEN,32     X'401899B3'
         GEN,32     X'401099B3'
         S,W        X,X'FC'             NEW V
         L          X,Y
         ROT        X,-1,64
         DECR       FPE
         BNZ,FPE    AGA&$X
         L          Y,X
         LANDN,W    X,X'FF'
         ROT        Y,1,64
         LAND,W     Y,X'FF'
         LOR        X,Y
         SET        Y
         S,W        Y,X'FC'             STORE V
         L          Y,X
```

```
          LANDN,W   Y,X'FE'              GET SIGN
          BNR       NON&$X
          GEN,32    X'401F9972'          GENERATE THE V VECTOR
          GEN,32    X'401E9972'
          GEN,32    X'401C9972'
          GEN,32    X'40189972'
          GEN,32    X'40109972'
          LAND,W    Y,X'F9'
          S,W       X,X'FA'
          LAND,W    X,X'FF'
CP1&$X    LXOR      X,Y
          S,W       Y,X'FD'
          LXOR,W    X,X'FC'
          L         Y,X
          LAND,W    Y,X'FD'
          LXOR,W    X,X'FC'
          BNR       DC1&$X
          LAND,W    Y,X'FE'
          ROT       Y,-1,32
          B         CP1&$X
DC1&$X    ANOP
NON&$X    ANOP
          LN,W      Y,X'FA'
          LANDN,W   Y,X'FF'
          LOR       X,Y
          S,W       X,X'FA'              X'FA' = POSITIVE (R AND Q)
** ADJUST THE SIGNS OF (R AND Q)
          L,W       X,X'F4'              X'F4' = V VECTOR FOR DIVISOR
          LXOR,W    X,X'F5'              X'F5' = V VECTOR FOR DIVIDEND
          L         Y,X
          LAND,W    Y,X'F6'              X'F6' = X'0000 0001 0000 0000'
          ROT       X,32,64
          ROT       Y,32,64
          LOR,W     X,X'F5'
          S,W       X,X'FC'
          S,W       Y,X'FD'
          L,W       Y,X'F6'
          LAND,W    Y,X'FC'
          LOR,W     Y,X'FD'
          LXOR,W    X,X'FA'
A2&$X     LXOR      X,Y
          LANDN     Y,X
          LAND,W    Y,X'FB'
          ROT       Y,-1,32
          BRS       A2&$X
          MEND
```

```
*
* EQWMA --- ARRAY WORDS EQUAL ARRAY WORDS
*
        MACRO
        EQWMA     &ARG1,&ARG2
        LI        FP3,&ARG1
        LI        FP1,&ARG2
        EQWA
        MEND
*
* EQWM --- ARRAY WORDS EQUAL ARRAY WORDS
*
        MACRO
        EQWM
        GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
        GEN,32    X'47008845'          LOAD THE WORDS B TO Y
        GEN,32    X'47800045'          Y = A EQU B
        GEN,32    X'40102252'          Y = Y AND (-16 ROT Y)
        GEN,32    X'40182252'          Y = Y AND (-8 ROT Y)
        GEN,32    X'401C2252'          Y = Y AND (-4 ROT Y)
        GEN,32    X'401E2252'          Y = Y AND (-2 ROT Y)
        GEN,32    X'401F2252'          Y = Y AND (-1 ROT Y)
        GEN,32    X'00002241'          Y = Y AND M
        GEN,32    X'380148FF'          RESET TO WORD MODE
        MEND
```

```
*
* EQCMA --- ARRAY WORDS EQUAL COMMON
*
        MACRO
        EQCMA     &ARG1
        LI        FP3,&ARG1
        EQCM
        MEND
*
* EQCM --- ARRAY WORDS EQUAL COMMON
*
        MACRO
        EQCM
        GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
        GEN,32    X'42008840'          STOR COMMON TO Y(0), OTHERS CLRD
        GEN,32    X'40E09952'          Y = Y OR (-32 ROT Y)
        GEN,32    X'40C09952'          Y = Y OR (-64 ROT Y)
        GEN,32    X'40809952'          Y = Y OR (-128 ROT Y)
        GEN,32    X'47800045'          Y = A EQU COMMON
        GEN,32    X'40102252'          Y = Y AND (-16 ROT Y)
        GEN,32    X'40182252'          Y = Y AND (-8 ROT Y)
        GEN,32    X'401C2252'          Y = Y AND (-4 ROT Y)
        GEN,32    X'401E2252'          Y = Y AND (-2 ROT Y)
        GEN,32    X'401F2252'          Y = Y AND (-1 ROT Y)
        GEN,32    X'00002241'          Y = Y AND M
        GEN,32    X'380148FF'          RESET TO WORD MODE
        MEND
```

```
*
* NEWMA --- ARRAY WORDS NOT EQUAL ARRAY WORDS
*
        MACRO
        NEWMA     &ARG1,&ARG2
        LI        FP3,&ARG1
        LI        FP1,&ARG2
        NEWM
        MEND
*
* NEWM --- ARRAY WORDS NOT EQUAL ARRAY WORDS
*
        MACRO
        NEWM
        GEN,32    X'3801481F'         SET WORD MODE TO MIXED MODE
        GEN,32    X'47008845'         LOAD THE WORDS B TO Y
        GEN,32    X'4780CC45'         Y = A NOTEQU B
        GEN,32    X'40109952'         Y = Y OR  (-16 ROT Y)
        GEN,32    X'40189952'         Y = Y OR  (-8 ROT Y)
        GEN,32    X'401C9952'         Y = Y OR  (-4 ROT Y)
        GEN,32    X'401E9952'         Y = Y OR  (-2 ROT Y)
        GEN,32    X'401F9952'         Y = Y OR  (-1 ROT Y)
        GEN,32    X'00002241'         Y = Y AND M
        GEN,32    X'380148FF'         RESET TO WORD MODE
        MEND
```

```
*
* NECMA --- ARRAY WORDS NOT EQUAL COMMON
*
        MACRO
        NECMA     &ARG1
        LI        FP3,&ARG1
        NECM
        MEND
*
* NECM --- ARRAY WORDS NOT EQUAL COMMON
*
        MACRO
        NECM
        GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
        GEN,32    X'42008840'          STOR COMMON TO Y(0), OTHERS CLRD
        GEN,32    X'40E09952'          Y = Y OR (-32 ROT Y)
        GEN,32    X'40C09952'          Y = Y OR (-64 ROT Y)
        GEN,32    X'40809952'          Y = Y OR (-128 ROT Y)
        GEN,32    X'4780CC45'          Y = A NOTEQU COMMON
        GEN,32    X'40109952'          Y = Y OR  (-16 ROT Y)
        GEN,32    X'40189952'          Y = Y OR  (-8 ROT Y)
        GEN,32    X'401C9952'          Y = Y OR  (-4 ROT Y)
        GEN,32    X'401E9952'          Y = Y OR  (-2 ROT Y)
        GEN,32    X'401F9952'          Y = Y OR  (-1 ROT Y)
        GEN,32    X'00002241'          Y = Y AND M
        GEN,32    X'380148FF'          RESET TO WORD MODE
        MEND
```

```
*
* GTWMA --- ARRAY WORDS GRATER THAN ARRAY WORDS
*
         MACRO
         GTWMA     &ARG1,&ARG2
         LI        FP3,&ARG1
         LI        FP1,&ARG2
         GTWM
         MEND
*
* GTWM --- ARRAY WORDS GRATER THAN ARRAY WORDS
*
         MACRO
         GTWM
         GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
         L,W       Y,FP3                LOAD THE WORDS A
         L,W       X,FP1                LOAD THE WORDS B
         LAND      X,M                  ZERO THOSE NOT PARTICIPATE
         LAND      Y,M                  ZERO THOSE NOT PARTICIPATE
AR&$X    LXOR      X,Y                  X IS THE DIFFERENCE
         LAND      Y,X                  Y IS THE BORROW
         LAND,W    Y,X'FB'              X'FB' = X'7FFF FFFF'
         ROT       Y,-1,32
         BRS       AR&$X
         GEN,32    X'40008843'          Y = X
         GEN,32    X'46FB6645'          Y = Y ANDN X'FB'
         GEN,32    X'401F9952'          Y = Y OR (-1 ROT Y)
         GEN,32    X'401E9952'          Y = Y OR (-2 ROT Y)
         GEN,32    X'401C9952'          Y = Y OR (-4 ROT Y)
         GEN,32    X'40189952'          Y = Y OR (-8 ROT Y)
         GEN,32    X'40109952'          Y = Y OR (-16 ROT Y)
         GEN,32    X'00002241'          Y = Y AND M
         GEN,32    X'380148FF'          RESET TO WORD MODE
         MEND
```

```
*
* GTCMA --- ARRAY WORDS GRATER THAN COMMON
*
        MACRO
        GTCMA     &ARG1
        LI        FP3,&ARG1
        GTCM
        MEND
*
* GTCM --- ARRAY WORDS GRATER THAN COMMON
*
        MACRO
        GTCM
        GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
        GEN,32    X'420088A0'          STOR COMMON TO X(0), OTHERS CLRD
        GEN,32    X'40E099B3'          X = X OR (-32 ROT X)
        GEN,32    X'40C099B3'          X = X OR (-64 ROT X)
        GEN,32    X'408099B3'          X = X OR (-128 ROT X)
        L,W       Y,FP3                LOAD THE WORDS A
        LAND      X,M                  ZERO THOSE NOT PARTICIPATE
        LAND      Y,M                  ZERO THOSE NOT PARTICIPATE
AR&$X   LXOR      X,Y                  X IS THE DIFFERENCE
        LAND      Y,X                  Y IS THE BORROW
        LAND,W    Y,X'FB'              X'FB' = X'7FFF FFFF'
        ROT       Y,-1,32
        BRS       AR&$X
        GEN,32    X'40008843'          Y = X
        GEN,32    X'46FB6645'          Y = Y ANDN X'FB'
        GEN,32    X'401F9952'          Y = Y OR (-1 ROT Y)
        GEN,32    X'401E9952'          Y = Y OR (-2 ROT Y)
        GEN,32    X'401C9952'          Y = Y OR (-4 ROT Y)
        GEN,32    X'40189952'          Y = Y OR (-8 ROT Y)
        GEN,32    X'40109952'          Y = Y OR (-16 ROT Y)
        GEN,32    X'00002241'          Y = Y AND M
        GEN,32    X'380148FF'          RESET TO WORD MODE
        MEND
```

```
*
* GEWMA --- ARRAY WORDS GRATER THAN OR EQUAL ARRAY WORDS
*
         MACRO
         GEWMA     &ARG1,&ARG2
         LI        FP3,&ARG1
         LI        FP1,&ARG2
         GEWM
         MEND
*
* GEWM --- ARRAY WORDS GRATER THAN OR EQUAL ARRAY WORDS
*
         MACRO
         GEWM
         GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
         L,W       X,FP3                LOAD WORDS A
         L,W       Y,FP1                LOAD WORDS B
         LAND      Y,M
AR&$X    LXOR      X,Y                  X IS THE DIFFERENCE
         LAND      Y,X                  Y IS THE BORROW
         LAND,W    Y,X'FB'              X'FB' = X'7FFF FFFF'
         ROT       Y,-1,32
         BRS       AR&$X
         GEN,32    X'40004443'          Y = NOT X
         GEN,32    X'46FB6645'          Y = Y ANDN X'FB'
         GEN,32    X'401F9952'          Y = Y OR (-1 ROT Y)
         GEN,32    X'401E9952'          Y = Y OR (-2 ROT Y)
         GEN,32    X'401C9952'          Y = Y OR (-4 ROT Y)
         GEN,32    X'40189952'          Y = Y OR (-8 ROT Y)
         GEN,32    X'40109952'          Y = Y OR (-16 ROT Y)
         GEN,32    X'00002241'          Y = Y AND M
         GEN,32    X'380148FF'          RESET TO WORD MODE
         MEND
```

```
*
* GECMA --- ARRAY WORDS GRATER THAN OR EQUAL COMMON
*
        MACRO
        GECMA     &ARG1
        LI        FP3,&ARG1
        GECM
        MEND
*
* GECM --- ARRAY WORDS GRATER THAN OR EQUAL COMMON
*
        MACRO
        GECM
        GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
        L,W       X,FP3                LOAD WORDS A
        GEN,32    X'42008840'          STOR COMMON TO Y(0), OTHERS CLRD
        GEN,32    X'40E09952'          Y = Y OR (-32 ROT Y)
        GEN,32    X'40C09952'          Y = Y OR (-64 ROT Y)
        GEN,32    X'40809952'          Y = Y OR (-128 ROT Y)
        LAND      Y,M
AR&$X   LXOR      X,Y                  X IS THE DIFFERENCE
        LAND      Y,X                  Y IS THE BORROW
        LAND,W    Y,X'FB'              X'FB' = X'7FFF FFFF'
        ROT       Y,-1,32
        BRS       AR&$X
        GEN,32    X'40004443'          Y = NOT X
        GEN,32    X'46FB6645'          Y = Y ANDN X'FB'
        GEN,32    X'401F9952'          Y = Y OR (-1 ROT Y)
        GEN,32    X'401E9952'          Y = Y OR (-2 ROT Y)
        GEN,32    X'401C9952'          Y = Y OR (-4 ROT Y)
        GEN,32    X'40189952'          Y = Y OR (-8 ROT Y)
        GEN,32    X'40109952'          Y = Y OR (-16 ROT Y)
        GEN,32    X'00002241'          Y = Y AND M
        GEN,32    X'380148FF'          RESET TO WORD MODE
        MEND
```

```
*
* LTWMA --- ARRAY WORDS LESS THAN ARRAY WORDS
*
         MACRO
         LTWMA     &ARG1,&ARG2
         LI        FP3,&ARG1
         LI        FP1,&ARG2
         LTWM
         MEND
*
* LTWM --- ARRAY WORDS LESS THAN ARRAY WORDS
*
         MACRO
         LTWM
         GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
         L,W       X,FP3                LOAD WORDS A
         L,W       Y,FP1                LOAD WORDS B
         LAND      Y,M
AR&$X    LXOR      X,Y                  X IS THE DIFFERENCE
         LAND      Y,X                  Y IS THE BORROW
         LAND,W    Y,X'FB'              X'FB' = X'7FFF FFFF'
         ROT       Y,-1,32
         BRS       AR&$X
         GEN,32    X'40008843'          Y = X
         GEN,32    X'46FB6645'          Y = Y ANDN X'FB'
         GEN,32    X'401F9952'          Y = Y OR (-1 ROT Y)
         GEN,32    X'401E9952'          Y = Y OR (-2 ROT Y)
         GEN,32    X'401C9952'          Y = Y OR (-4 ROT Y)
         GEN,32    X'40189952'          Y = Y OR (-8 ROT Y)
         GEN,32    X'40109952'          Y = Y OR (-16 ROT Y)
         GEN,32    X'00002241'          Y = Y AND M
         GEN,32    X'380148FF'          RESET TO WORD MODE
         MEND
```

```
*
* LTCMA --- ARRAY WORDS LESS THAN COMMON
*
         MACRO
         LTCMA     &ARG1
         LI        FP3,&ARG1
         LTCM
         MEND
*
* LTCM --- ARRAY WORDS LESS THAN COMMON
*
         MACRO
         LTCM
         GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
         GEN,32    X'42008840'          STOR COMMON TO Y(0), OTHERS CLRD
         GEN,32    X'40E09952'          Y = Y OR (-32 ROT Y)
         GEN,32    X'40C09952'          Y = Y OR (-64 ROT Y)
         GEN,32    X'40809952'          Y = Y OR (-128 ROT Y)
         L,W       X,FP3                LOAD WORDS A
AR&$X    LXOR      X,Y                  X IS THE DIFFERENCE
         LAND      Y,X                  Y IS THE BORROW
         LAND,W    Y,X'FB'              X'FB' = X'7FFF FFFF'
         ROT       Y,-1,32
         BRS       AR&$X
         GEN,32    X'40008843'          Y = X
         GEN,32    X'46FB6645'          Y = Y ANDN X'FB'
         GEN,32    X'401F9952'          Y = Y OR (-1 ROT Y)
         GEN,32    X'401E9952'          Y = Y OR (-2 ROT Y)
         GEN,32    X'401C9952'          Y = Y OR (-4 ROT Y)
         GEN,32    X'40189952'          Y = Y OR (-8 ROT Y)
         GEN,32    X'40109952'          Y = Y OR (-16 ROT Y)
         GEN,32    X'00002241'          Y = Y AND M
         GEN,32    X'380148FF'          RESET TO WORD MODE
         MEND
```

```
*
* LEWMA --- ARRAY WORDS LESS THAN OR EQUAL ARRAY WORDS
*
         MACRO
         LEWMA     &ARG1,&ARG2
         LI        FP3,&ARG1
         LI        FP1,&ARG2
         LEWM
         MEND
*
* LEWM --- ARRAY WORDS LESS THAN OR EQUAL ARRAY WORDS
*
         MACRO
         LEWM
         GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
         L,W       Y,FP3                LOAD WORDS A
         L,W       X,FP1                LOAD WORDS B
         LAND      X,M
         LAND      Y,M
AR&$X    LXOR      X,Y                  X IS THE DIFFERENCE
         LAND      Y,X                  Y IS THE BORROW
         LAND,W    Y,X'FB'              X'FB' = X'7FFF FFFF'
         ROT       Y,-1,32
         BRS       AR&$X
         GEN,32    X'40004443'          Y = NOT X
         GEN,32    X'46FB6645'          Y = Y ANDN X'FB'
         GEN,32    X'401F9952'          Y = Y OR (-1 ROT Y)
         GEN,32    X'401E9952'          Y = Y OR (-2 ROT Y)
         GEN,32    X'401C9952'          Y = Y OR (-4 ROT Y)
         GEN,32    X'40189952'          Y = Y OR (-8 ROT Y)
         GEN,32    X'40109952'          Y = Y OR (-16 ROT Y)
         GEN,32    X'00002241'          Y = Y AND M
         GEN,32    X'380148FF'          RESET TO WORD MODE
         MEND
```

```
*
* LECMA --- ARRAY WORDS LESS THAN OR EQUAL COMMON
*
         MACRO
         LECMA     &ARG1
         LI        FP3,&ARG1
         LECM
         MEND
*
* LECM --- ARRAY WORDS LESS THAN OR EQUAL COMMON
*
         MACRO
         LECM
         GEN,32    X'3801481F'          SET WORD MODE TO MIXED MODE
         GEN,32    X'420088A0'          STOR COMMON TO X(0), OTHERS CLRD
         GEN,32    X'40E099B3'          X = X OR (-32 ROT X)
         GEN,32    X'40C099B3'          X = X OR (-64 ROT X)
         GEN,32    X'408099B3'          X = X OR (-128 ROT X)
         L,W       Y,FP3                LOAD THE WORDS A
         LAND      X,M                  ZERO THOSE NOT PARTICIPATE
         LAND      Y,M                  ZERO THOSE NOT PARTICIPATE
AR&$X    LXOR      X,Y                  X IS THE DIFFERENCE
         LAND      Y,X                  Y IS THE BORROW
         LAND,W    Y,X'FB'              X'FB' = X'7FFF FFFF'
         ROT       Y,-1,32
         BRS       AR&$X
         GEN,32    X'40004443'          Y = NOT X
         GEN,32    X'46FB6645'          Y = Y ANDN X'FB'
         GEN,32    X'401F9952'          Y = Y OR (-1 ROT Y)
         GEN,32    X'401E9952'          Y = Y OR (-2 ROT Y)
         GEN,32    X'401C9952'          Y = Y OR (-4 ROT Y)
         GEN,32    X'40189952'          Y = Y OR (-8 ROT Y)
         GEN,32    X'40109952'          Y = Y OR (-16 ROT Y)
         GEN,32    X'00002241'          Y = Y AND M
         GEN,32    X'380148FF'          RESET TO WORD MODE
         MEND
```

Section 15

PARALLEL ARITHMETIC USING SERIAL ARITHMETIC PROCESSORS

Jyh-Ming Jeremy Hsu

Syracuse University

Abstract

As the speed of computer reaches bounds due to the limitation of switching speed, parallelism seems to be the only solution to the problem of increasing demand for higher processing speeds. The arithmetic operations play a central role in the operating speeds of computer systems.

The first part of the dissertation analyzes the properties of bit serial organized STARAN computer for bit parallel arithmetic operations. A mixed mode addressing scheme of the associative array memory is utilized such that all the bits of a number can be processed simultaneously. New algorithms for arithmetic operations are described and the simulation programs in APL are included in the appendices. The execution speeds were measured and compared with the bit-serial operations by actual implementation in the STARAN.

The second part of this dissertation analyzes the signed digit number representations. Associative array structures which are part of a modified STARAN computer are proposed. Implementation of a specific signed digit number representation with radix-10 in this computer are discussed. New algorithms of signed digit arithmetic operations are simulated in APL for parallel processing and are included in the appendices. The execution times are estimated based on the comparable speed of STARAN system.

Some future extensions are also discussed.

TABLE OF CONTENTS

		PAGE
TABLE OF CONTENTS		15-i
TABLE OF FIGURES		15-iii
LIST OF TABLES		15-v
CHAPTER 1	INTRODUCTION	15-1
1-1	Prior Work of Computer Arithmetic	15-1
1-2	Objective of Investigation	15-3
CHAPTER 2	MIXED MODE OPERATIONS FOR STARAN COMPUTER	15-6
2-1	STARAN System Description	15-6
2-2	STARAN Analysis and the Mixed Mode	15-9
2-3	High Speed Arithmetic and Relational Operations	15-10
	A. Mixed Mode Addition/Subtraction	15-15
	B. Mixed Mode Multiplication	15-19
	C. Mixed Mode Division	15-23
	D. Mixed Mode Relational Operations	15-25
2-4	Speed Gains and Some Capabilities	15-25
	A. Speed Gains	15-25
	B. Some Additional Capabilities of Mixed-Mode	15-27
CHAPTER 3	SIGNED DIGIT NUMBER REPRESENTATION IN AN ASSOCIATIVE ARRAY STRUCTURE COMPUTER	15-31
3-1	Signed-Digit Numbers and its Arithmetic Algorithms	15-31
	A. Addition	15-33

B. Multiple-Operand Addition 15-37
C. Subtraction 15-40
D. Multiplication 15-40
E. Division 15-45
3-2 Implementation of the Signed Digit Numbers in the Associative Array Structure Computers 15-49
A. A Proposed Signed Digit Number System 15-49
B. A Proposed Associative Array Structure Computer . 15-50
C. Implementation Considerations 15-53
3-3 Speed Gains 15-55
CHAPTER 4 CONCLUSIONS 15-59

APPENDIX A APL DOCUMENTATION OF THE MIXED MODE OPERATIONS . . . 15-61
APPENDIX B EXECUTION TIMES AND PROCESSING RATES OF BIT-SEQUENTIAL OPERATIONS AND MIXED MODE OPERATIONS . . 15-71
APPENDIX C FUNDAMENTAL OPERATIONS OF A DECIMAL SIGNED DIGIT NUMBER SYSTEM 15-75
APPENDIX D APL DOCUMENTATION OF THE DECIMAL SIGNED DIGIT ARITHMETIC OPERATIONS 15-90
APPENDIX E EXECUTION TIMES AND PROCESSING RATES OF THE MIXED MODE OPERATIONS AND A DECIMAL SIGNED DIGIT NUMBER ARITHMETIC OPERATIONS 15-110
BIBLIOGRAPHY 15-114
BIOGRAPHICAL DATA 15-116

TABLE OF FIGURES

FIGURE		PAGE
2-1	STARAN Block Diagram	15-7
2-2	Associative Array in STARAN System	15-8
2-3	Partition of Mixed Mode Mod-32	15-11
2-4	Address Scheme in a Section of Mixed Mode Mod-32 (Decimal Notation)	15-12
2-5	Partition of Mixed Mode Mod-64	15-13
2-6	Address Scheme in a Section of Mixed Mode Mod-64 (Decimal Notation)	15-14
2-7	Mixed Mode Storage of Mod-32	15-16
2-8	Control Structure of Mixed Mode Mod-32	15-17
2-9a	Algorithm of Mixed Mode Addition	15-20
2-9b	Algorithm of Mixed Mode Subtraction	15-21
2-10	Average Carry/Borrow Propagation of 32 bits Data Words	15-22
2-11	Algorithm of Multiplication	15-24
2-12	Non-Restoring Division Algorithm for Associative Array Structure Computer	15-26
2-13a	Comparison of Execution Times of Addition/Subtraction Operations (Mixed Mode Operations vs. Bit-Serial Operations)	15-28
2-13b	Comparison of Execution Times of Multiplication Operations (Mixed Mode Operations vs. Bit-Serial Operations)	15-29

FIGURE PAGE

2-13c Comparison of Execution Times of Division Operations (Mixed Mode Operations vs. Bit-Serial Operations) 15-30

3-1 Algorithm of Four-Digit Signed Digit Number Multiplication with Two-Step Three-Operand Addition . . . 15-43

3-2 Modified Algorithm of Four-Digit Signed Digit Multiplication with Four-Operand Addition 15-44

3-3 Eight-Digit Signed Digit Division Algorithm 15-47

3-4 A Proposed Associative Array Structure 15-51

3-5 Modified STARAN System Block Diagram 15-52

3-6a Comparison of Execution Times of Addition/Subtraction (Signed Digit Operations vs. Mixed Mode Operations) . . . 15-56

3-6b Comparison of Execution Times of Multiplication (Signed Digit Operations vs. Mixed Mode Operations) 15-57

3-6c Comparison of Execution Times of Division (Signed Digit Operations vs. Mixed Mode Operations) 15-58

LIST OF TABLES

TABLE		PAGE
3-1	Relations of r, a, S_{max} and n for Multiple-Operand Signed Digit Addition	15-39
B-1	Execution Times and Processing Rates of Addition/ Subtraction (Mixed Mode Operations and Bit-Sequential Operations)	15-72
B-2	Execution Times and Processing Rates of Multiplication (Mixed Mode Operations and Bit-Sequentail Operations)	15-73
B-3	Execution Times and Processing Rates of Division (Mixed Mode Operations and Bit-Sequential Operations)	15-74
C-1	Partial Sum of Operation Ⓐ for a Decimal Signed Digit Number System	15-76
C-2	Carry Digits of Operation Ⓐ for a Decimal Signed Digit Number System	15-77
C-3	Definition of Operation Ⓑ for a Decimal Signed Digit Number System	15-78
C-4	Definition of Operation Ⓝ for a Decimal Signed Digit Number System	15-79
C-5	Partial Product Digits of Operation Ⓜ for a Decimal Signed Digit Number System	15-81
C-6	Multiplication Carry Digits of Operation Ⓜ for a Decimal Signed Digit Number System	15-82
C-7	Definition of Operation DTC for a Decimal Signed Digit Number System	15-84

TABLE		PAGE
C-8	Definition of Operation CMB for a Decimal Signed Digit Number System	15-85
C-9	Definition of Operation DNEG for a Decimal Signed Digit Number System	15-85
C-10	Definition of Operation DNZR for a Decimal Signed Digit Number System	15-85
C-11	Definition of Operation CMPRA for a Decimal Signed Digit Number System	15-86
C-12	Definition of Operation CMPRB for a Decimal Signed Digit Number System	15-86
C-13	Definition of Operation TRFM for a Decimal Signed Dignt Number System	15-86
C-14	Definition of Operation MULPP for a Decimal Signed Digit Number System	15-87
C-15	Definition of Operation SELECT for a Decimal Signed Dignt Number System	15-88
C-16	Definition of Operation SELECTZR for a Decimal Signed Digit Number System	15-89
E-1	Execution Times and Processing Rates for Addition/ Subtraction (Signed Digit Operations and Mixed Mode Operations)	15-111
E-2	Execution Times and Processing Rates for Multiplication (Signed Digit Operations and Mixed Mode Operations) . . .	15-112

TABLE PAGE

E-3 Execution Times and Processing Rates for Division (Signed Digit Operations and Mixed Mode Operations) . . . 15-113

CHAPTER 1 INTRODUCTION

One of the main considerations of the computer designer is obtaining the highest possible operating speed, subject to various technical and economic constraints. Since the arithmetic operations play a central role in the operating speed of electronic computers, the design of arithmetic operations is an important subject in the development of new systems. Over the years a number of different methods have been developed. The various approaches are described as follows.

1-1 Prior Work of Computer Arithmetic

Computer arithmetic requires four operations: Addition, Subtraction, Multiplication and Division.

The simplest binary adder is built with a one bit full adder circuit, a flip-flop for storing a single carry signal, and some control logic. This is known as a bit serial binary adder. A simple approach to design bit parallel adders is to interconnect n one-bit full adders which are called ripple-carry adders. The ripple-carry adder may have intolerable delay since the carry may have to propagate in the adder from one end to another. Today most of the adder designs are based on the carry-look-ahead principle. The carry-look-ahead adder was first described by Weinberger and Smith [1] in 1956. A systematic description of the carry-look-ahead principle was studied by MacSorley [2] in 1961. The carry-propagation time is always a restraint in the binary number representations. Signed Digit Number representations which eliminate the carry propagation chains were studied by A. Avizienis [3] in 1961. Redundant representations of numbers are used in this class of number systems and the carry propagation is limited to one digit position to the left during the operations of addition.

Subtraction is essentially similar to the operation of addition, except that in subtraction the sign of subtrahend is changed before adding the subtrahend to the minuend.

The third operation is multiplication. The general principle by

which the computer performs multiplication is very simple. Multiplication is carried out by a sequence of additions and shifts or shifts alone; the number of each and their order is determined by the multiplier digits. For binary numbers the additions are performed one at a time and not in multiples. The multiplier is examined from one end to the other, bit by bit; if a bit is a 1, addition and shift is called for; and if the bit is a 0, only shift is performed. The average multiplication would require half as many additions as there are bits in the multiplier. The use of carry-save adders improved the multiplication time by postponing the carry propagation beyond one stage until the end of all of the shifts, and then using one carry-propagate time to complete the additions [5]. Techniques for multiplication using variable length shifts and using uniform shifts of more than one were studied by MacSorley [2] in 1961. Flores [4] presented an extensive description in 1963.

The final arithmetic operation is division. Division is usually performed by repeated subtraction and shifts. Division methods are usually categorized as restoring and non-restoring. In the restoring method, the divisor is successively subtracted from the high-order positions of the dividend; the result replaces the dividend. The quotient digit is increased by one for each successful subtraction. When a negative result of subtraction is obtained, then the dividend is restored by adding the divisor to it. The dividend and quotient are then both shifted left one position and the process is repeated. The non-restoring method eliminates the time required for restore addition operations by algebraic rearrangement. Some other speed-up efforts by shifting over 0's and quotient lookahead were described by Flores [4].

In recent years, the development of large scale integrated circuits caused new logic design concepts. Several versatile arrays for arithmetic operations have been proposed [5 to 14]. Most of these efforts improve operating speed by using the iterative networks which are suitable for large scale integrated circuits.

1-2 Objective of Investigation

Given the switching speed limitation, the integration of multiple functional units into a multiprocessing or parallel processing system seems to be the only solution to obtain a higher performance digital computer system. Some existing parallel processors have already demonstrated their processing speed superiority over the conventional systems, and have proven to be the answer to the problem of processing some large scale problems [15]. There are various techniques for multiprocessing or parallel processing. Each of the techniques has its merits as well as limitations and we believe that an ultimate computer system may very well possess several of these techniques.

This dissertation describes a study of the arithmetic portion of parallel data stream processing computer systems. One method which has been used to provide higher operating speeds is to use many simple arithmetic processors and to let each one process a different data set. Such systems are suited to problem areas in which the problem requires that many data streams be processed in the same way. The Goodyear STARAN system is an example. In the STARAN each of the arithmetic processors is a simple 2-bit input unit. Arithmetic processes such as adding two 32-bit numbers are performed serially in time. However many data streams are processed simultaneously, so very high processing rates are possible for the system as a whole.

The arithmetic processes carried out by STARAN are essentially serial arithmetic processes, performed simultaneously on many separate data streams. The data streams enter the arithmetic units one bit at a time. In this dissertation, algorithms are developed for utilizing the many serial processors available to process data streams where all the bits of the numbers in the data streams enter the arithmetic process simultaneously. Hence, if a 32-bit number is to be added, the entire 32-bit number will be entered into 32 simple arithmetic units at one time. Then the 32 arithmetic units cooperate in completing the arithmetic process. The algorithms used are basically parallel arithmetic processes but they are performed by a multiplicity

of simple serial arithmetic units.

Algorithms of this type are developed in Chapter 2. The term "mixed mode" is used in the STARAN to describe the operation of the multidimensional access memory. In this memory words may be accessed serially, one bit at a time, or in parallel, all bits of a number of words, at a time. Normally, this capability is used in a kind of corner-turning operation between parallel memory loading from external devices and the internal serial processing. We have extended the term to cover arithmetic processes which utilize the multidimensional memory capability. The arithmetic processes are referred to as "mixed mode" to distinguish them from the serial arithmetic processes. It is important to note that data stream parallelism is still retained in the mixed-mode case. For example, a set of 256 serial processors could do serial arithmetic on 256 data streams or it could do 32-bit mixed mode arithmetic on 8 data streams. The implementation and evaluation of these mixed mode processing algorithms represent original work. The mixed-mode arithmetic operations were implemented [22], and the execution speeds are compared with the STARAN bit-serial operations.

The mixed mode arithmetic processes developed in Chapter 2 make use of 2's complement numbers. The speed for addition and subtraction depended on the longest necessary carry/borrow propagation chain. On the average this length is about 5 to 9 bits depending on the number of parallel data streams.

In some number systems the carry propagation length can be limited to exactly one digit position. If a STARAN-like system was able to do arithmetic on such carry-limited numbers, then the mixed mode arithmetic processes would be even faster. Chapter 3 explores the implementation of the signed-digit number representation in the associative array structure computer. An associative array structure which is the combination of associative memory and micro-processors is proposed. A STARAN-like system which uses these associative array structures is described. A decimal signed digit number system is analyzed. Arithmetic algorithms are developed for mixed mode processing of such numbers on the STARAN-like system. The speeds of these processes are

estimated. The adaption of signed digit arithmetic to STARAN-like structures and the design and evaluation of this new system are original.

Chapter 4 describes the significance of this work. Further areas of investigations are also discussed.

CHAPTER 2 MIXED MODE OPERATIONS FOR STARAN COMPUTER

In this chapter, mixed mode algorithms for the STARAN Computer [16] are developed for arithmetic and relational operations. All of the bits of the data words are processed simultaneously. Our analysis will assume a data word length of 32 bits, so that up to 8 words in each associative array may be processed simultaneously. Word lengths of 8, 16, 64 or any other length of 2's power up to 256 may be processed similarly. The average execution times are compared with the bit-serial operations provided by the Goodyear Aerospace Corporation in the standard STARAN Computer [18].

2-1 STARAN System Description

The STARAN system introduces a new concept in computers, designed to achieve very high processing rates economically. Fig. 2-1 shows a block diagram of the STARAN computer. The STARAN basically consists of a conventionally addressed control memory for program storage and data buffering, up to 32 associative memory arrays*, a control logic unit for sequencing and decoding instruction from control memory, and a control logic unit associated with a special parallel input-output (PIO) capability.

The associative arrays are the heart of the STARAN computer. There may be a maximum of 32 associative arrays in one system. The array memories provide the content-addressability and the parallel processing capabilities. The structure of an array memory is shown in Fig. 2-2. Each array contains a multi-dimensional-access (MDA) memory with 2^{16} (65,536) bits of storage and 256 one-bit processing elements. The MDA memory is arranged in a 256-bit by 256-bit square. Each of the 256 one-bit processing elements consists of three bits of response stores X, Y and M registers, and the associated logic.

---

* In the Rome Air Development Center Associative Processor (RADCAP) System, there are four associative memory arrays connected.

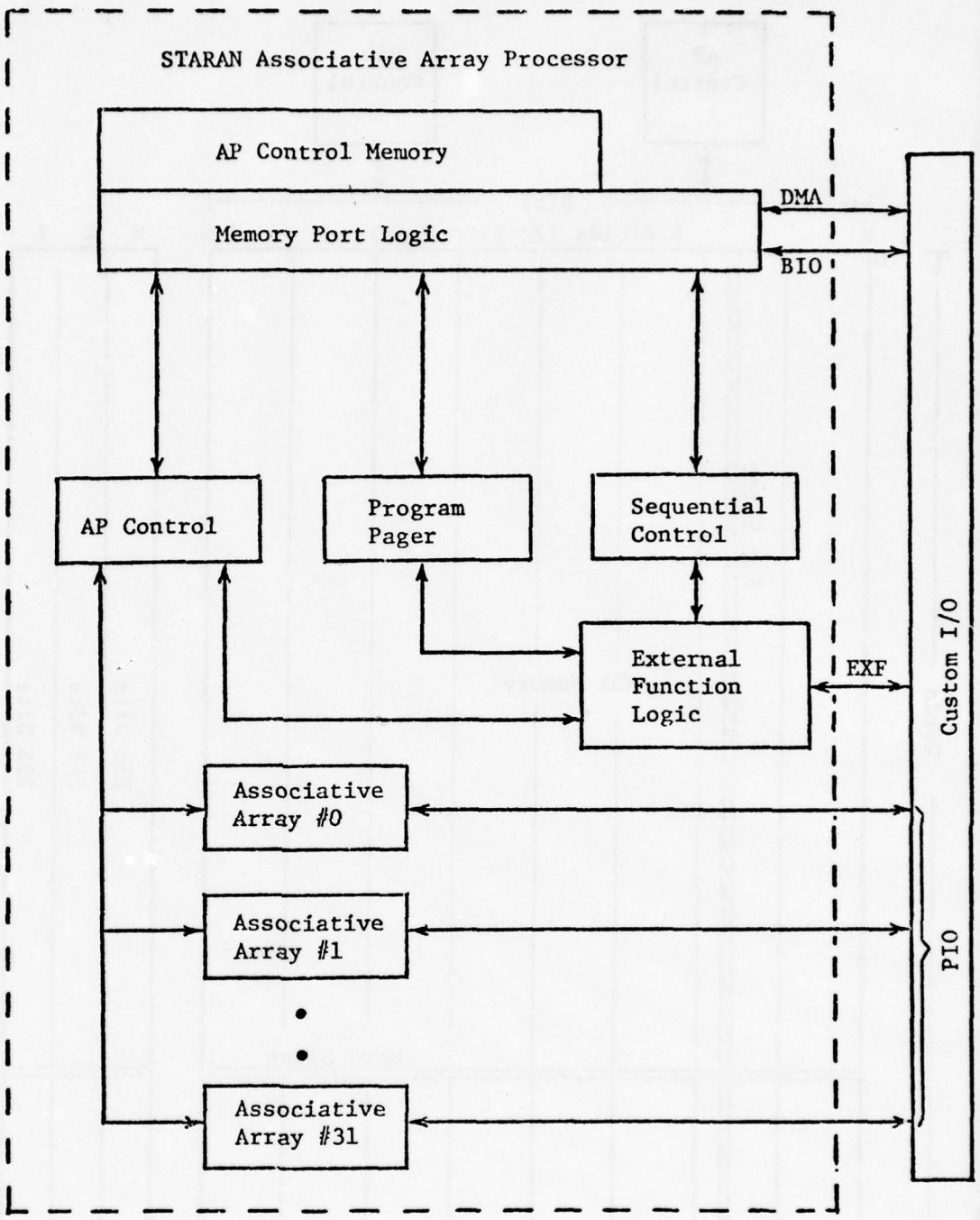


Fig. 2-1 STARAN Block Diagram

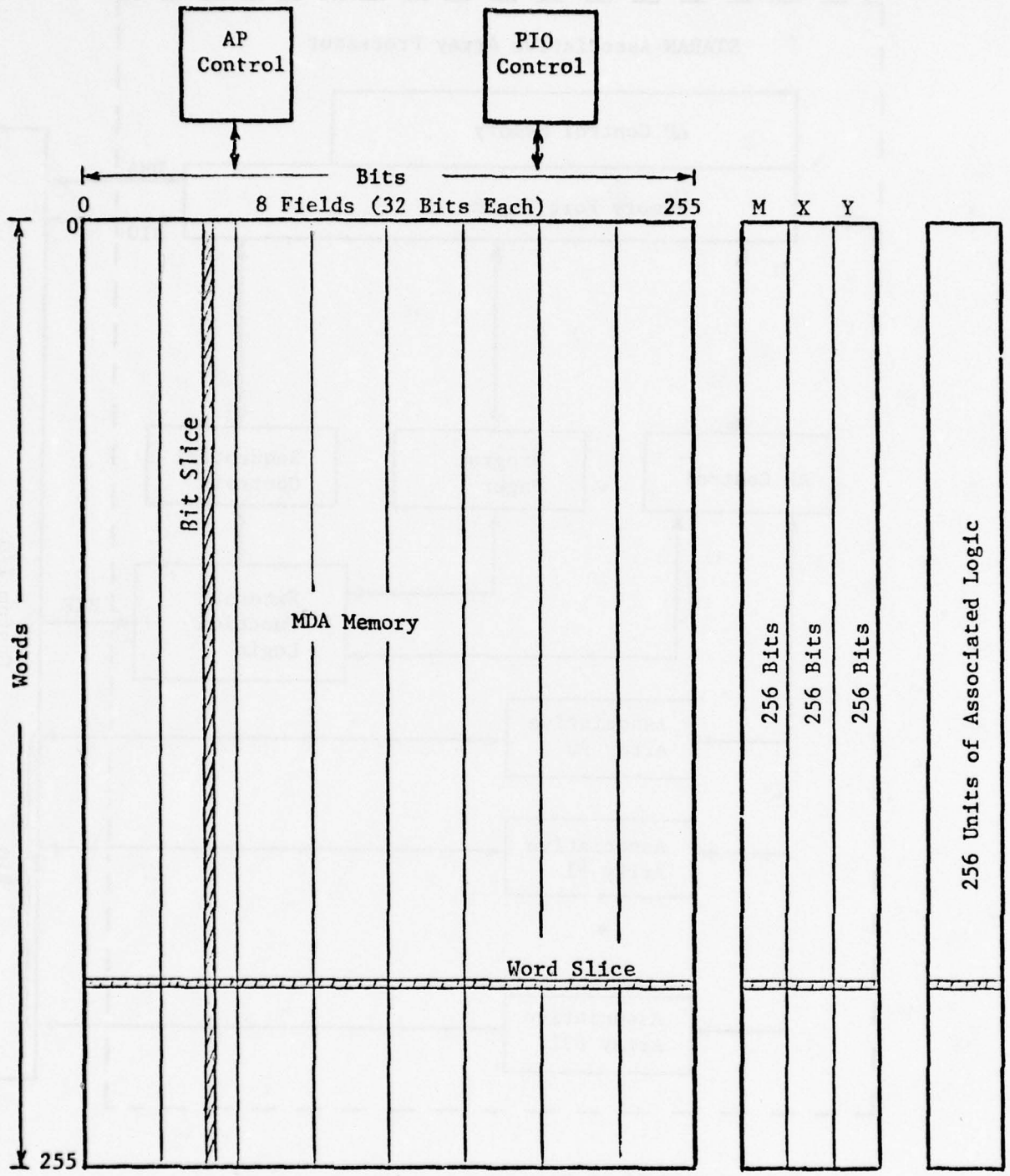


Fig. 2-2 Associative Array in STARAN System

These 256 one-bit processing elements provide the parallel capabilities for each array. Each processing element (PE) can be considered as a simple one-bit mini-processor with an associated memory of 256 bits and three one-bit registers. The associative arrays can be assigned either under the control of the Associative Processing (AP) control logic unit or under the Parallel Input/Output (PIO) control logic unit. The arithmetic and relational operations on the associative arrays can be performed while the arrays are under either AP control or PIO control. The response store registers X, Y and M in each associative array allow the array data words to be serially searched, restructured and processed at a fast rate, 256 bits at a time.

In a standard STARAN computer, the MDA memories in the arrays can be accessed either the vertical (bit-slice) or horizontal (word-slice) direction and with a maximum of 256 bits transferred within an array in a single operation. Instructions are sequenced and decoded by the control logic and executed simultaneously by all active processing elements. This allows bit-serial arithmetic and relational operations to be performed on all words of array memories in parallel. The bit-serial operations are available as standard STARAN assembler language macros [18 and 19].

2-2 STARAN Analysis and the Mixed Mode

It is clear that the STARAN bit-serial operations will have a good level of the utilization of the associative arrays when the number of data words approaches thousands. In the application problems which do not present so many data words for parallel processing, most of the array memories may be idle. It would be desirable to process simultaneously all the bits of a smaller munber of data streams. Fortunately, the associative array memory is designed to support either alternative.

In a standard STARAN, the multi-dimensional-access (MDA) memories in the arrays can be accessed two different ways: bit-slice mode and word mode. The address mode register option allows access to the MDA memories in other modes as well. The other modes are inter-

mediate to the bit-slice mode and the word mode since they access some bits of some words. There are 256 different access modes. However, to utilize the mixed mode for arithmetic and relational operations, we don't need all 256 modes. Assume the data word length of 32 bits, the mixed mode mod-32 and mixed mode mod-64 are sufficient for arithmetic and relational operations [22].

When an MDA memory is accessed, the program specifies two 8-bit bytes: an address A and a mode constant Z. The mode constant Z selects a certain pattern or stencil of 256 bits and the address A positions the stencil over the memory, selecting 256 memory bits for access. The address A is either specified directly in the associative machine instruction or specified indirectly as the content of a pointer register. The mode constant Z is always specified indirectly as the contents of an address mode register.

Let T_K refer to the Kth bit of the transferred data and $S_{I,J}$ refer to the Ith bit of the Jth word. Given an address A, and a mode constant Z, where K, I, J, A and Z are 8-bit binary numbers, the association is

$$T_K \longleftrightarrow S_{\psi(Z,A,K),\ \psi(Z,K,A)}$$

where $\psi(Z,X,Y) = (\sim Z \wedge X) \vee (Z \wedge Y)$

From the above relation, the mixed mode mod-32 (Z = '00011111')segments each of the selected associative arrays into eight sections, as shown in Fig. 2-3. Each section consists of 32 by 256 bits memory and 32 processing elements. There are 256 32-bit words in each section. The mod-32 address scheme of a section is shown in Fig. 2-4. The selection of sections is under the control of the M (Mask) register. Similarly, the partition and the address scheme of a section of mixed mode mod-64 are shown in Fig. 2-5 and Fig. 2-6.

2-3 High Speed Arithmetic and Relational Operations

Mixed Mode access is to be used in order to do parallel processing of the 32-bit numbers. The elements of 32-bit words should be stored so that all of them can be accessed simultaneously. This means that

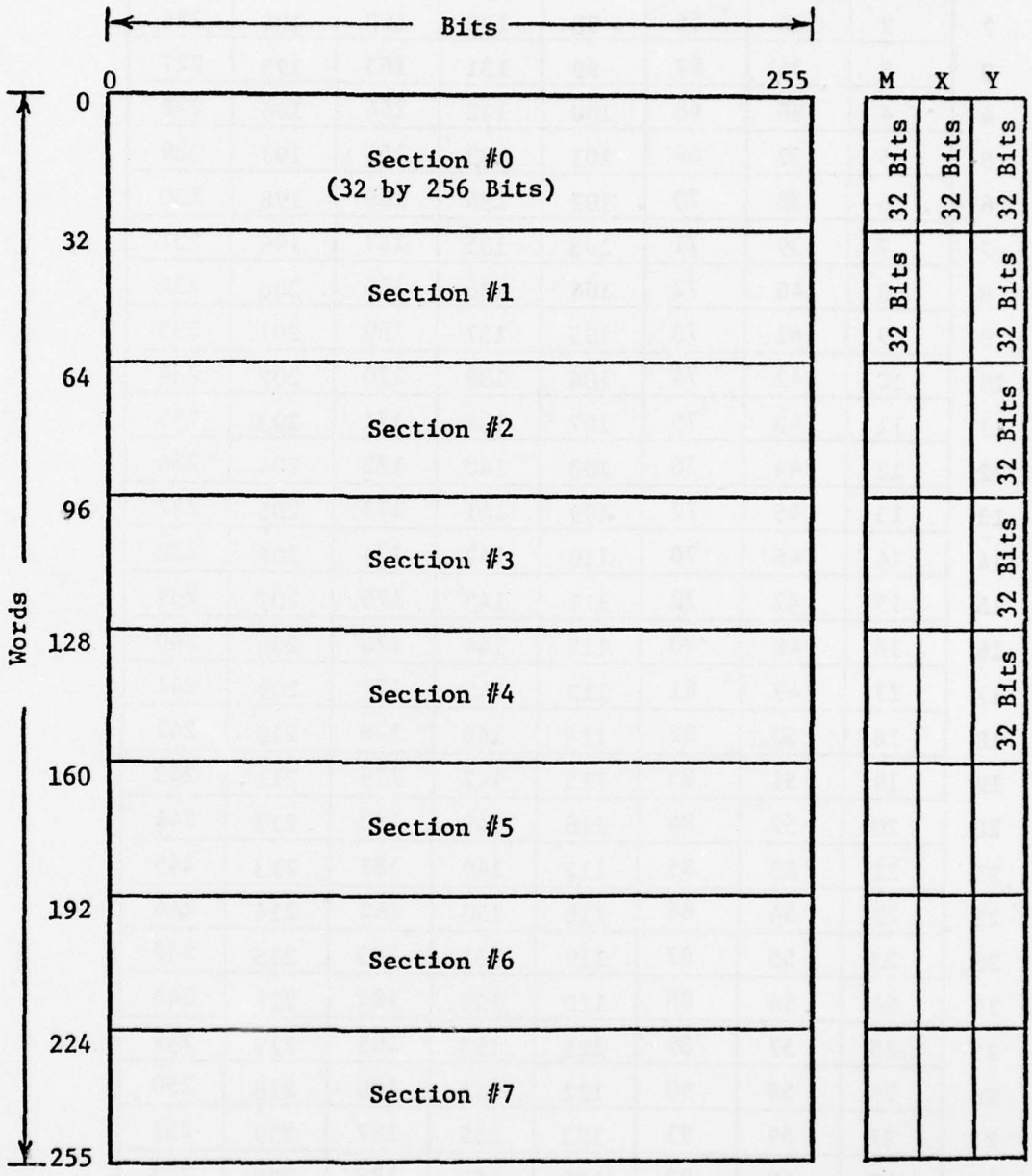


Fig. 2-3 Partition of Mixed Mode Mod-32

WORD \ BIT	0	32	64	96	128	160	192	224 255
0	0	32	64	96	128	160	192	224
1	1	33	65	97	129	161	193	225
2	2	34	66	98	130	162	194	226
3	3	35	67	99	131	163	195	227
4	4	36	68	100	132	164	196	228
5	5	37	69	101	133	165	197	229
6	6	38	70	102	134	166	198	230
7	7	39	71	103	135	167	199	231
8	8	40	72	104	136	168	200	232
9	9	41	73	105	137	169	201	233
10	10	42	74	106	138	170	202	234
11	11	43	75	107	139	171	203	235
12	12	44	76	108	140	172	204	236
13	13	45	77	109	141	173	205	237
14	14	46	78	110	142	174	206	238
15	15	47	79	111	143	175	207	239
16	16	48	80	112	144	176	208	240
17	17	49	81	113	145	177	209	241
18	18	50	82	114	146	178	210	242
19	19	51	83	115	147	179	211	243
20	20	52	84	116	148	180	212	244
21	21	53	85	117	149	181	213	245
22	22	54	86	118	150	182	214	246
23	23	55	87	119	151	183	215	247
24	24	56	88	120	152	184	216	248
25	25	57	89	121	153	185	217	249
26	26	58	90	122	154	186	218	250
27	27	59	91	123	155	187	219	251
28	28	60	92	124	156	188	220	252
29	29	61	93	125	157	189	221	253
30	30	62	94	126	158	190	222	254
31	31	63	95	127	159	191	223	255

Fig. 2-4 Address Scheme in a Section of Mixed Mode Mod-32 (Decimal Notation)

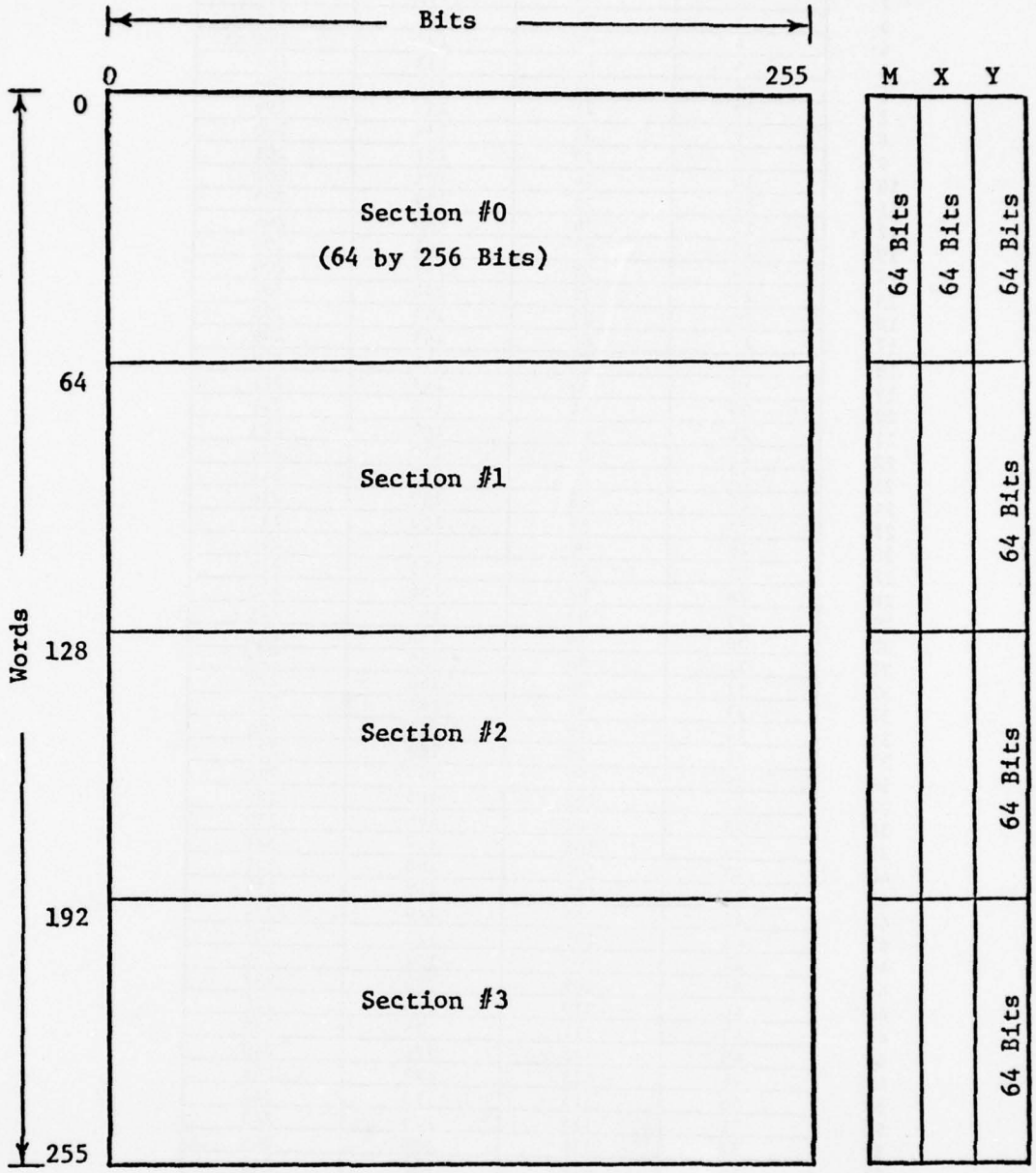


Fig. 2-5 Partition of Mixed Mode Mod-64

WORD \ BIT	0	64	128	192
0	0	64	128	192
1	1	65	129	193
2	2	66	130	194
3	3	67	131	195
4	4	68	132	196
5	5	69	133	197
6	6	70	134	198
7	7	71	135	199
8	8	72	136	200
9	9	73	137	201
10	10	74	138	202
11	11	75	139	203
12	12	76	140	204
13	13	77	141	205
14	14	78	142	206
15	15	79	143	207
16	16	80	144	208
17	17	81	145	209
18	18	82	146	210
19	19	83	147	211
20	20	84	148	212
21	21	85	149	213
22	22	86	150	214
23	23	87	151	215
24	24	88	152	216
25	25	89	153	217
26	26	90	154	218
27	27	91	155	219
28	28	92	156	220
29	29	93	157	221
30	30	94	158	222
31	31	95	159	223
32	32	96	160	224
33	33	97	161	225
34	34	98	162	226
35	35	99	163	227
36	36	100	164	228
37	37	101	165	229
38	38	102	166	230
39	39	103	167	231
40	40	104	168	232
41	41	105	169	233
42	42	106	170	234
43	43	107	171	235
44	44	108	172	236
45	45	109	173	237
46	46	110	174	238
47	47	111	175	239
48	48	112	176	240
49	49	113	177	241
50	50	114	178	242
51	51	115	179	243
52	52	116	180	244
53	53	117	181	245
54	54	118	182	246
55	55	119	183	247
56	56	120	184	248
57	57	121	185	249
58	58	122	186	250
59	59	123	187	251
60	60	124	188	252
61	61	125	189	253
62	62	126	190	254
63	63	127	191	255

Fig. 2-6 Address Scheme in a Section of Mixed Mode Mod-64 (Decimal Notation)

that the first and second numbers should be separated in the array by the modulus size. Fig. 2-7 shows the storage arrangement for mixed mode mod-32. For example, all the eight 32-bit numbers A_0, A_1, ..., A_7 can be accessed simultaneously. Each of these eight numbers is stored in the same location of different sections. The M (Mask) register is assigned to control the selection of sections. From the programmer's view, the selection of sections is controlled by the Array Select register (AS) and the Mask (M) register in each array. The control structure of mixed mode mod-32 is shown in Fig. 2-8. The control of mixed mode mod-64 can be described similarly.

The algorithms for 2's complement binary arithmetic are described in the following paragraphs. APL notation is used in the flow-chart to represent vector type operations. The APL equivalent of these algorithms is listed in Appendix A, along with some APL simulation results.

A. Mixed Mode Addition/Subtraction

A method for using 2-bit processors for parallel addition/subtraction is the process known as Mercer's adder. A carry/borrow vector and a partial sum/difference vector are repeatedly combined until the carry/borrow vectors becomes zero and the addition/subtraction is completed. The equation for the partial sum S ($S_{n-1}S_{n-2} \cdots S_1S_0$) and the carry C ($C_{n-1}C_{n-2} \cdots C_1C_0$) of two binary numbers U ($U_{n-1}U_{n-2} \cdots U_1U_0$) and V($V_{n-1}V_{n-2} \cdots V_1V_0$) are shown as:

$$\left.\begin{aligned} S_i &= (U_i \neq V_i) \\ C_i &= (U_i \wedge V_i) \end{aligned}\right\} \quad \forall\, i=0,1,\ldots,(n-1) \qquad (2.1)$$

And the equations for the partial difference D($D_{n-1}D_{n-2} \cdots D_1D_0$) and the borrow B($B_{n-1}B_{n-2} \cdots B_1B_0$) of the subtraction of two binary numbers U and V are shown as:

$$\left.\begin{aligned} D_i &= (U_i \neq V_i) \\ B_i &= (\sim U_i \wedge V_i) \end{aligned}\right\} \quad \forall\, i=0,1,\ldots,(n-1) \qquad (2.2)$$

Since the formations of S_i, C_i, D_i and B_i depend only on the two bits

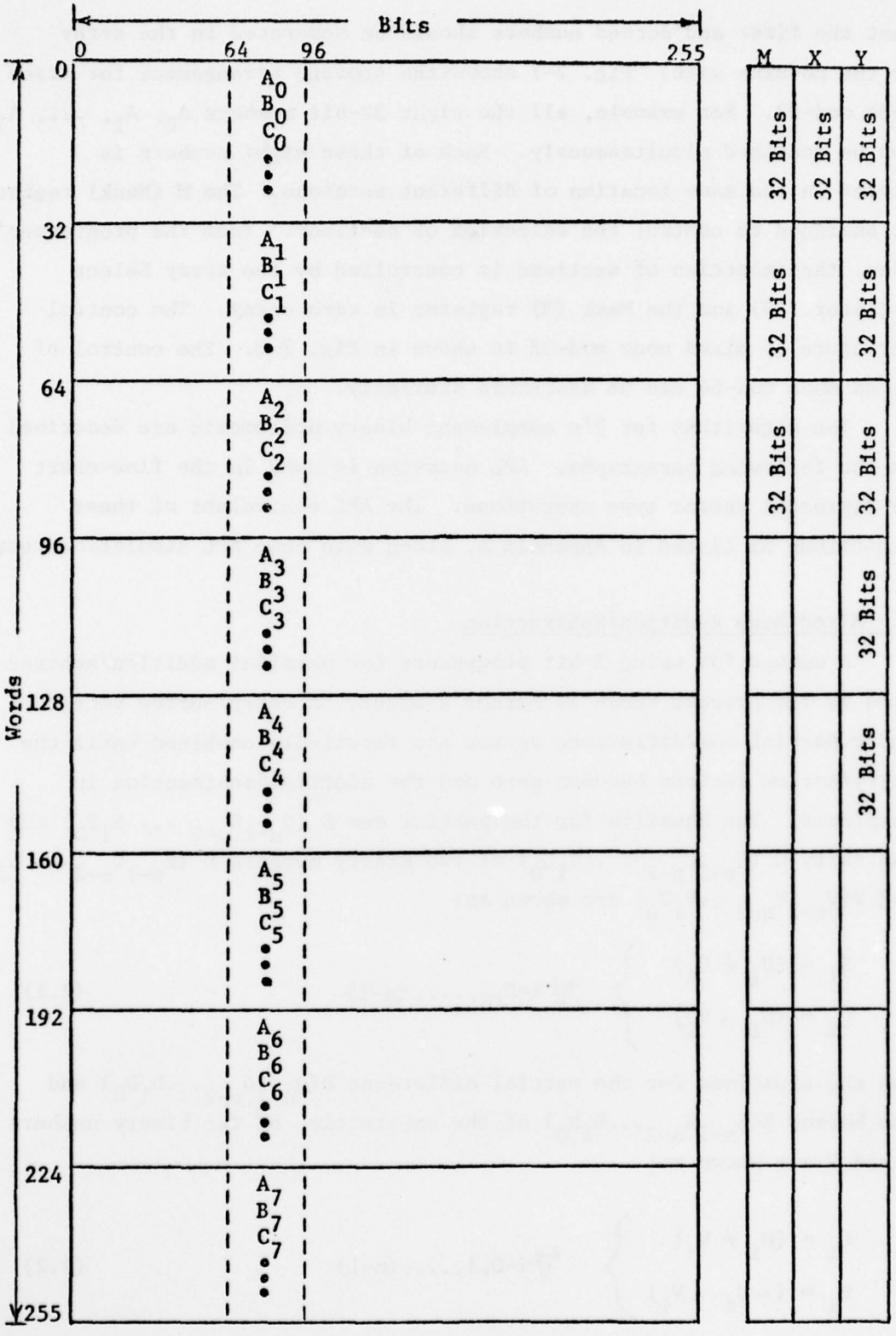


Fig. 2-7 Mixed Mode storage of Mod-32

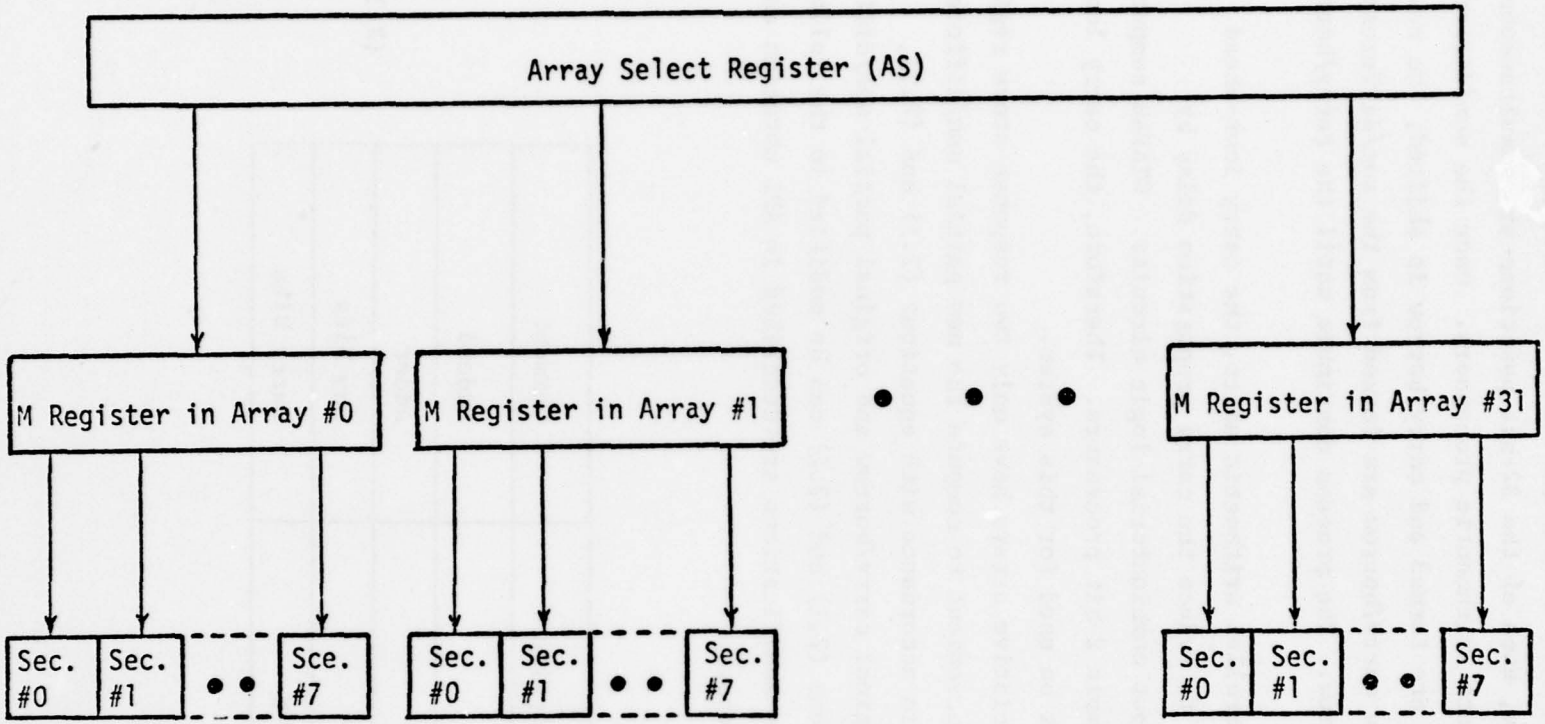


Fig. 2-8 Control Structure of Mixed Mode Mod-32

U_i and V_i, it is clear that a 2-bit processor is sufficient. In the 32-bit mixed mode case, each of the 32-bit positions are simultaneously processed by 32 separate arithmentic processors. Once the sum/difference and carry/borrow are formed and carry/borrow is shifted, the new sum/difference and new carry/borrow are formed from the sum/difference and shifted carry/borrow. The process continues until the carry/borrow becomes zero.

In conventional parallel arithmetic units, the carry look-ahead principle may be used to reduce the carry propagation delay by providing multiple-input combinatorial logic circuits. STARAN computer only provides very simple 2-bit processors. Therefore, the carry look-ahead principle cannot be used for this system.

The STARAN's associative arrays have only two response store registers X and Y, so it is inconvenient to compute the new partial sum/difference and new carry/borrow in accordance with equations (2.1) and (2.2), which require the original carry/borrow and original partial sum/difference values. These equations (2.1) and (2.2) can be modified to the following sequential steps. The modifications are described in APL notation and the operands are vectors.

(a) For addition

APL Description	Comment
[1] X ← U	Addend
[2] Y ← V	Adder
[3] X ← X ≠ Y	Sum Bits
[4] Y ← Y ∧ ∿ X	Carry Bits

(2.3)

(b) For Subtraction

APL Description	Comment
[1] X ← U	Subtrahend
[2] Y ← V	Minuend
[3] X ← X ≠ Y	Difference Bits
[4] Y ← Y ∧ X	Borrow Bits

(2.4)

As a result of above sequential steps, the contents of X and Y are the partial sum/difference and the carry/borrow. The algorithms are described in Fig. 2-9a and Fig. 2-9b. This method takes advantage of the short length of an average carry/borrow propagation. On an average basis, the carry/borrow propagation is about 5 to 6 bits for addition/subtraction of two 32-bit data words. As the number of data words processed in parallel increases, the average maximum carry/borrow propagation increases very slowly. The average carry/borrow propagation has been measured on the STARAN computer for 256 pairs of random numbers and is shown in Fig. 2-10.

B. Mixed Mode Multiplication

In the multiplication operation in associative arrays, a method which uses shifts of uniform size and permits predicting the number of cycles that will be required is preferable. The procedures described here are based on the application of carry-save adders to the multiplication. The idea is the postponement of the completion of the addition operations to the very end. The partial sum will be stored as two numbers whose sum is the true partial sum. The true sum is formed only when all the addends have been entered. The execution time of this carry-save algorithm is proportional to the length of the multiplier instead of the product of length of multiplier and length of multiplicand.

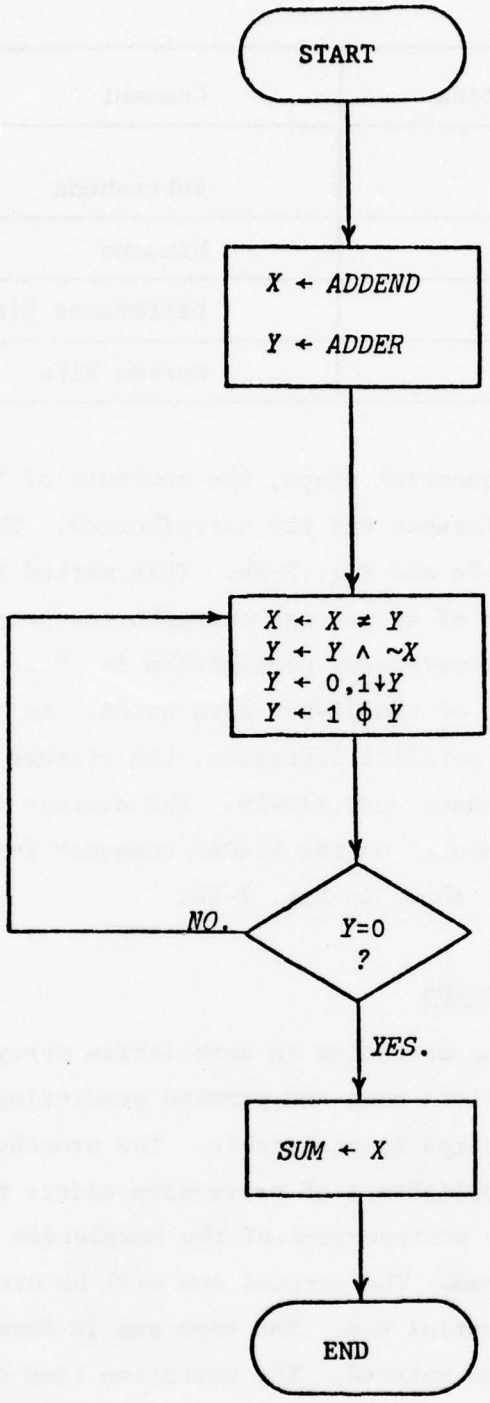


Fig. 2-9a Algorithm of Mixed Mode Addition

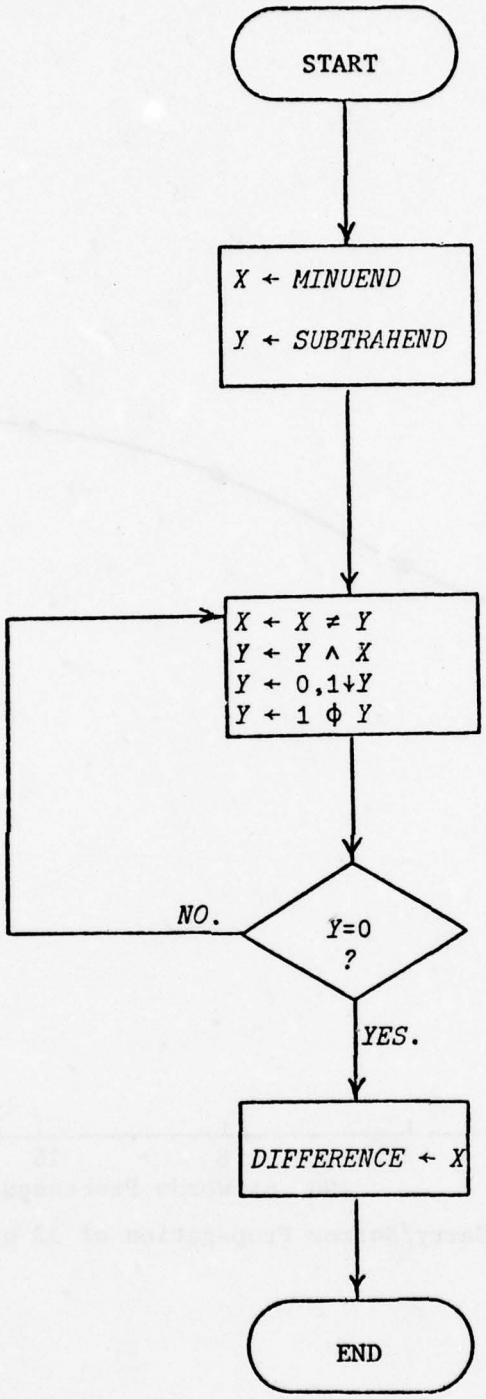


Fig. 2-9b Algorithm of Mixed Mode Subtraction

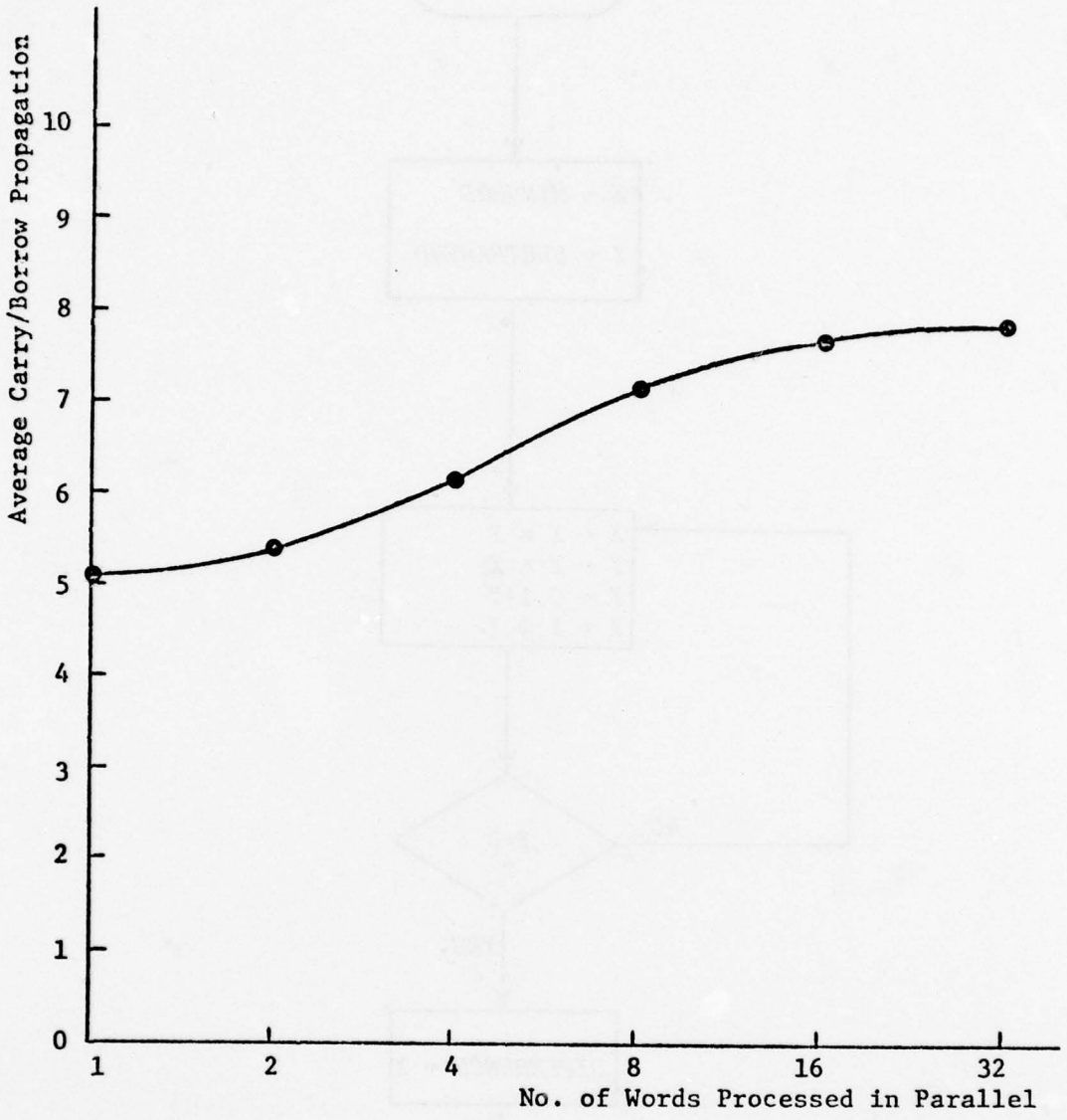


Fig. 2-10 Average Carry/Borrow Propagation of 32 bits Data Words

The multiplication algorithm for two 32-bit positive numbers is shown in Fig. 2-11. The main operation is the consecutive addition of partial product P, carry C and multiplicand A if the current bit of the multiplier is 1.

The equations are:

$$\left.\begin{aligned} &\text{new } P \leftarrow P \neq C \neq A \\ &\text{new } C \leftarrow (P \wedge C) \vee (P \wedge A) \vee (C \wedge A) \end{aligned}\right\} \qquad (2.5)$$

Since the $\neq$ operation is linear, these equations can be replaced as the following sequentail steps:

$$\left.\begin{aligned} &[1] \quad \text{new } P \leftarrow P \neq C \neq A \\ &[2] \quad \text{new } C \leftarrow ((\sim \text{new } P) \wedge C) \vee ((\sim \text{new } P) \wedge A) \vee (C \wedge A) \end{aligned}\right\} \qquad (2.6)$$

This replacement will save some execution time in the operation.

For negative multiplicands, the algorithm is still the same. But for negative multipliers, a conversion of sign will be done before and after this algorithm is applied.

C. Mixed Mode Division

A non-restoring division method which uses shifts of uniform size and also permits prediction of the number of cycles is employed. The division process is data dependent, that is, the decision on whether to add or to subtract at a given step depends on the present value of the dividend. Hence, in a parallel process in which many divisions are going on simultaneously, at a given step some will call for subtraction and some for addition. Fortunately the algorithms for addition and subtraction are so similar that a single control binary vector V can be used to switch a fixed algorithm from addition to subtraction.

For example, equations for subtraction and addition in equations (2.4) and (2.3), the only difference is that in the addition process, the carry bits are generated by:

$$Y \leftarrow Y \wedge \sim X$$

AD-A054 944 SYRACUSE UNIV N Y F/G 9/2
LARGE SCALE INFORMATION SYSTEMS. VOLUME III.(U)
MAR 78 F30602-74-C-0335
UNCLASSIFIED RADC-TR-78-43-VOL-3 NL

4 OF 4
AD
A054944



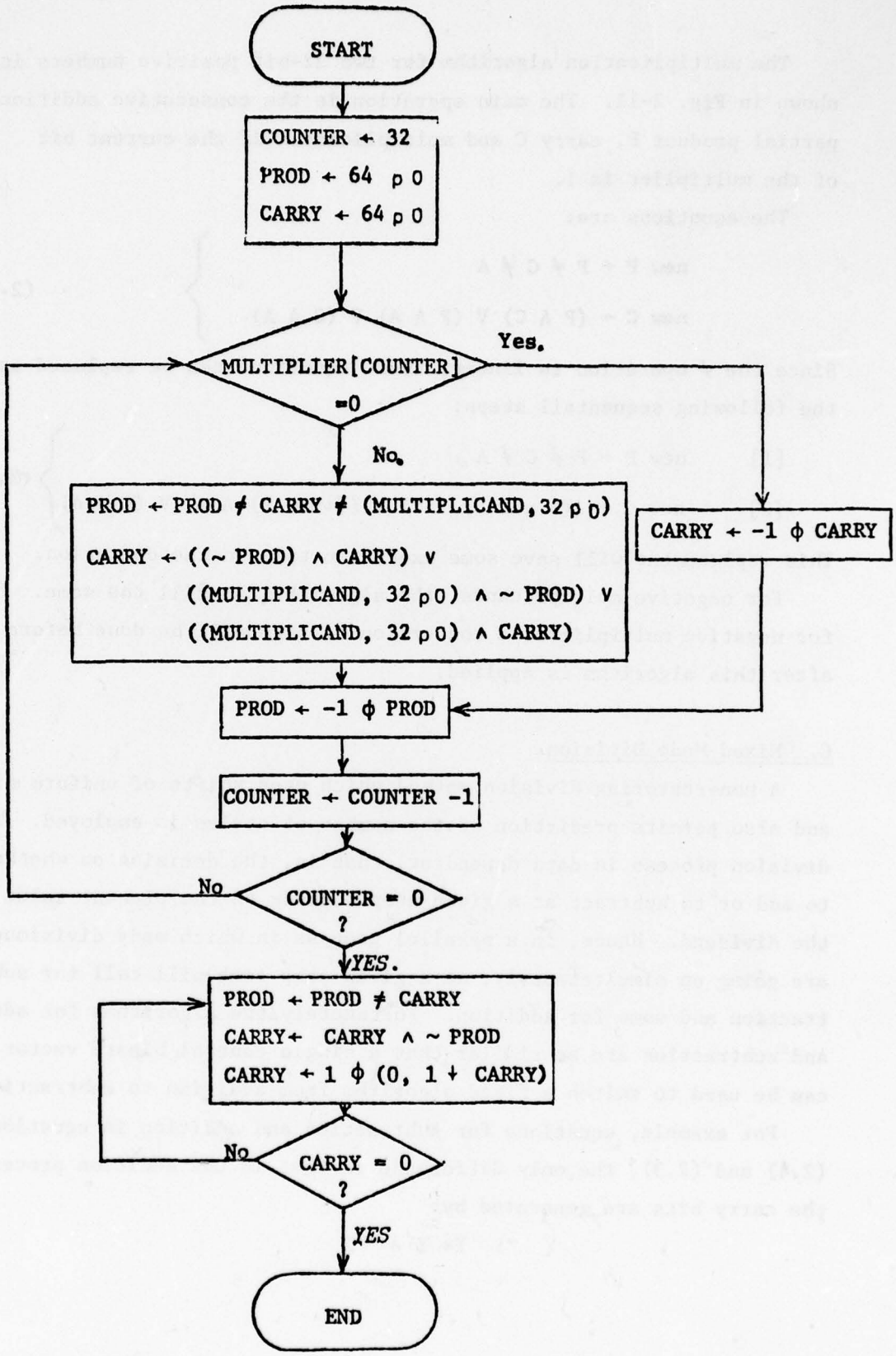


Fig. 2-11. Algorithm of Multiplication

while in the subtraction process, the borrow bits are generated by:

$$Y \leftarrow Y \wedge X$$

A single equation can be formed in which a V vector will select addition or subtraction.

The equation is as follows:

$$Y \leftarrow Y \wedge (V \neq X) \qquad (2.7)$$

An operation CO-OPERATION is designed so that it does addition or subtraction depending on a vector V. The V vector is a 256-bit vector in each active associative array. Those sections whose V vectors are all 1's will perform additions and those whose V vectors are all 0's will perform subtractions. The CO-OPERATION makes the implementation of the non-restoring division method for parallel arithmetic possible. The algorithm is described as a flow chart in Fig. 2-12. This method takes the advantage of the short average carry/borrow propagation.

D. Mixed Mode Relational Operations

Except for the Equal and Non-Equal operations, all of the relational operations can be done by subtraction of the two data words which are compared. The Equal and Non-Equal operations can be done by comparing all bits of the data words simultaneously in the mixed mode case, so these comparisons can be done in a fixed time instead of being data dependent.

2-4 Speed Gains and Some Capabilities

A. Speed Gains

The algorithms for 32-bit 2's complement numbers arithmetic have been implemented [22] and placed in the macro library of the STARAN system. The average execution speeds have been measured. The measured or estimated execution times and processing rates for the mixed mode arithmetic are included in Appendix B.

If the number of data words which can be processed in parallel is

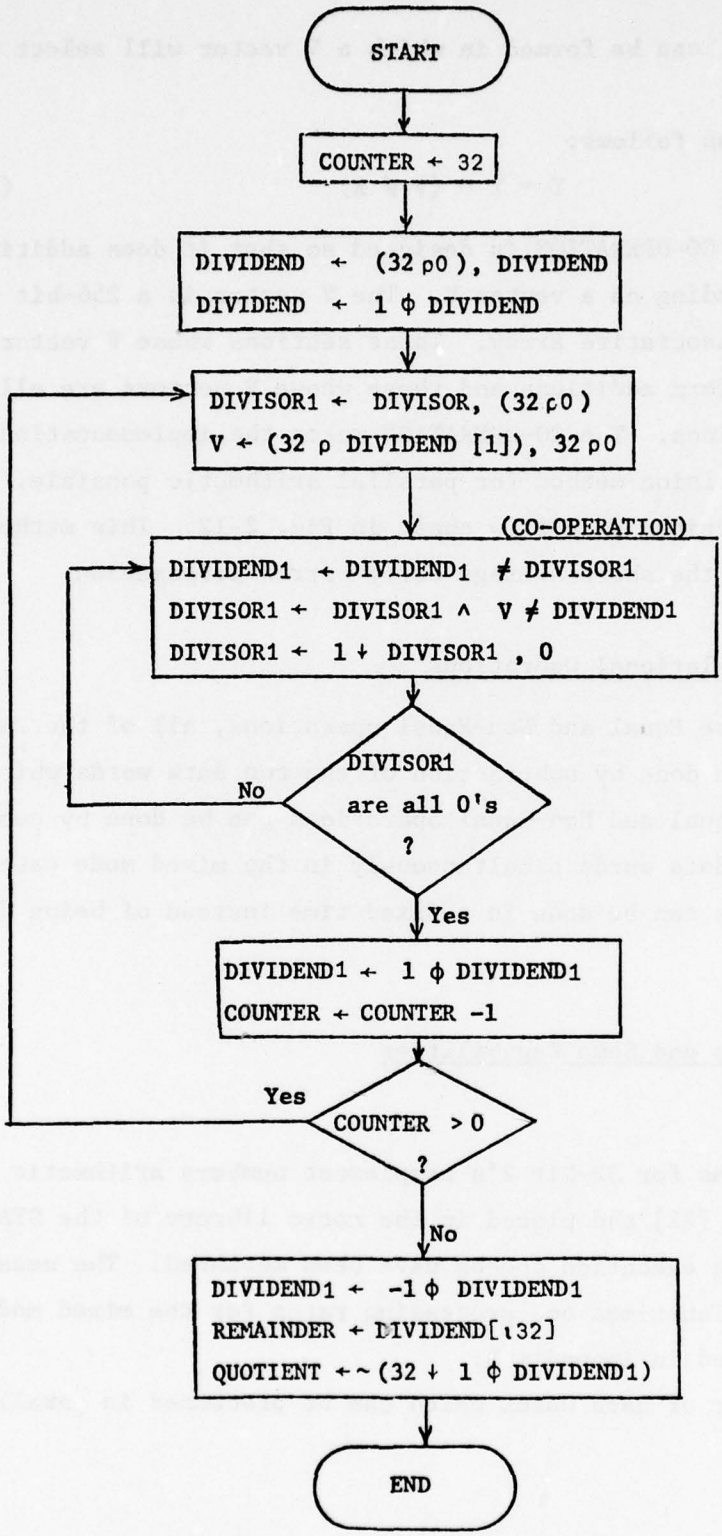


Fig. 2-12 Non-restoring Division Algorithm for Associative Array Structure Computer

less than 128, the mixed-mode algorithms are much faster than the bit-serial operations in the STARAN computer. The addition and subtraction operations are almost two times faster. The multiplication operation is five to seven times faster. The division operation is about twice as fast. Division does not gain as much in speed because the non-restoring division algorithm has to complete the addition or aubtraction in every cycle.

The speed of parallel multiplication of a set of numbers by a common multiplier is even faster than the multiplication of pairs of numbers, and is not shown here. The execution times of the mixed mode operations are compared with the bit-serial operations in Fig. 2-13a, b and c.

B, Some Additional Capabilities of Mixed-Mode

The STARAN memory control accepts 8-bit addresses so there are 256 distinct addresses available independent of the mode. In bit serial mode, the 256 addresses are addresses of individual bits in the data words. Hence the hardware allows reference to any bit in any one of 8 32-bit words. The limitation to 8 words is a severe constraint in many applications. It can be overcome by masking and data movement with some loss in time. The mixed-mode operations provide an alternative to STARAN programmers. In the mixed-mode case, the 256 distinct addresses refer to 256 distinct 32-bit words which will be adequate in most cases. However, in mixed-mode if bits must be selected from the data words masking is required. For those problems which do not present over 128 data sets for parallel processing, the mixed-mode operations provide higher processing speeds than the bit-serial operations, and more addressable words in each data stream. Some modifications of the processing hardware to support mixed-mode operations would yield further increase in speeds. For example, if carry look-ahead logic units were available in the associated logic of response store registers, the processing speeds of mixed mode operations will be further increased.

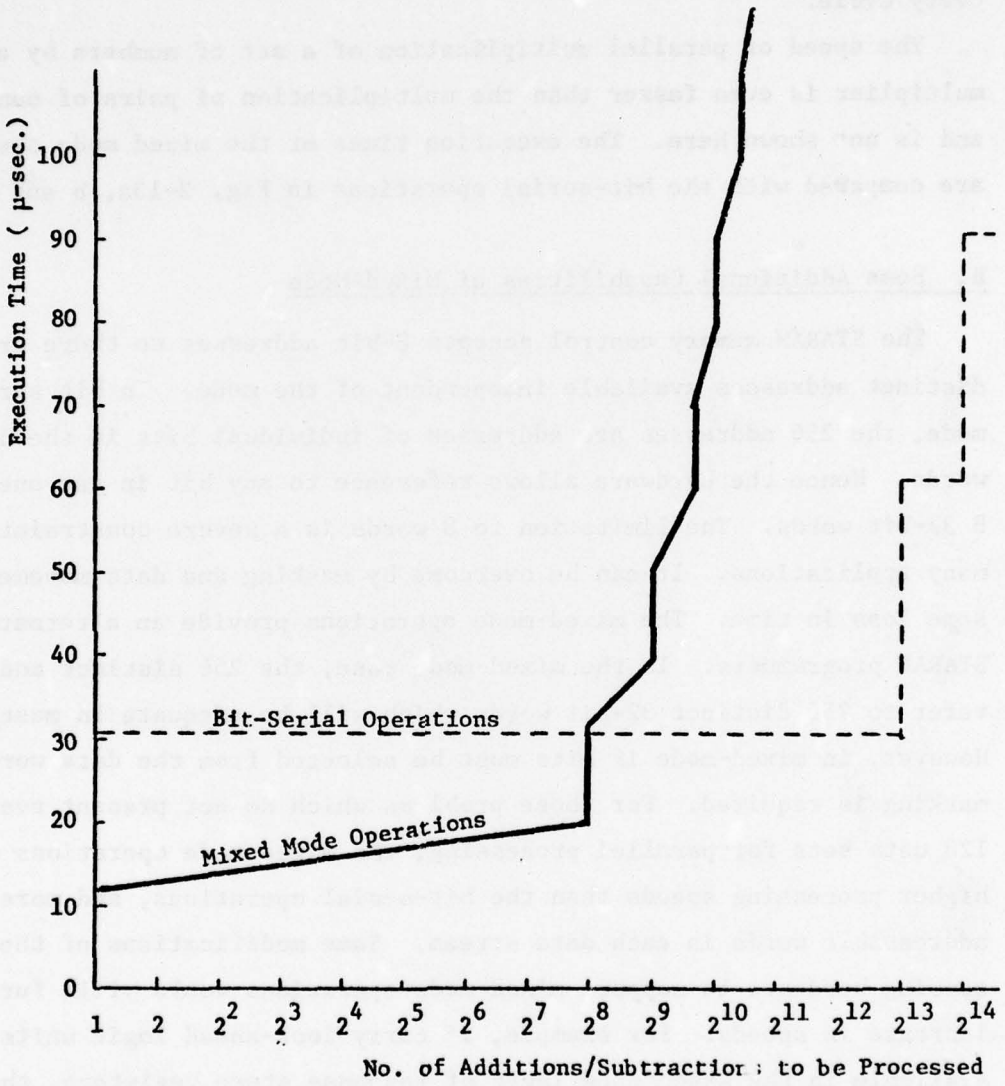


Fig. 2-13a Comparison of Execution Times of Addition/Subtraction Operations

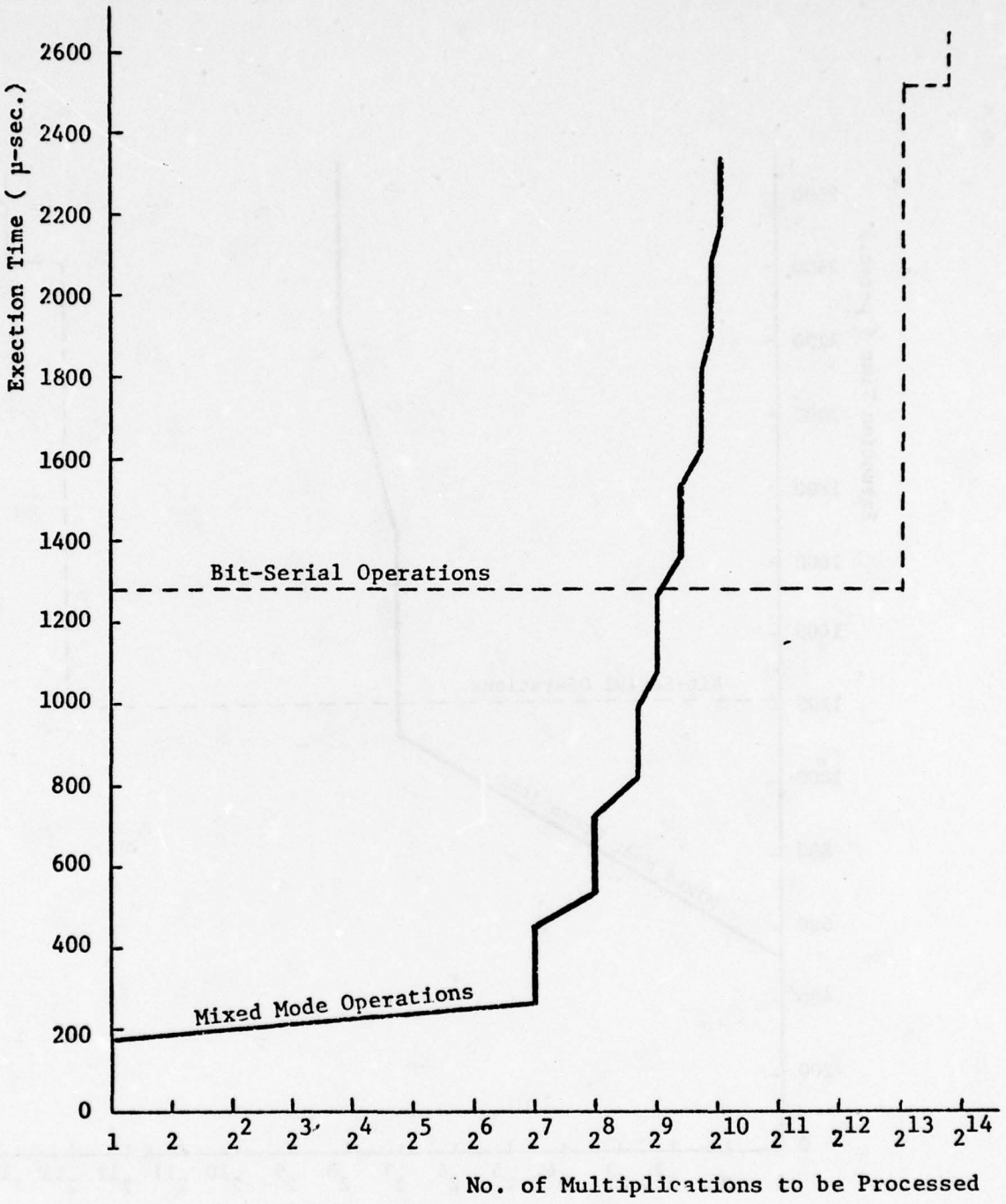


Fig. 2-13b Comparison of Execution Times of Multiplication Operations

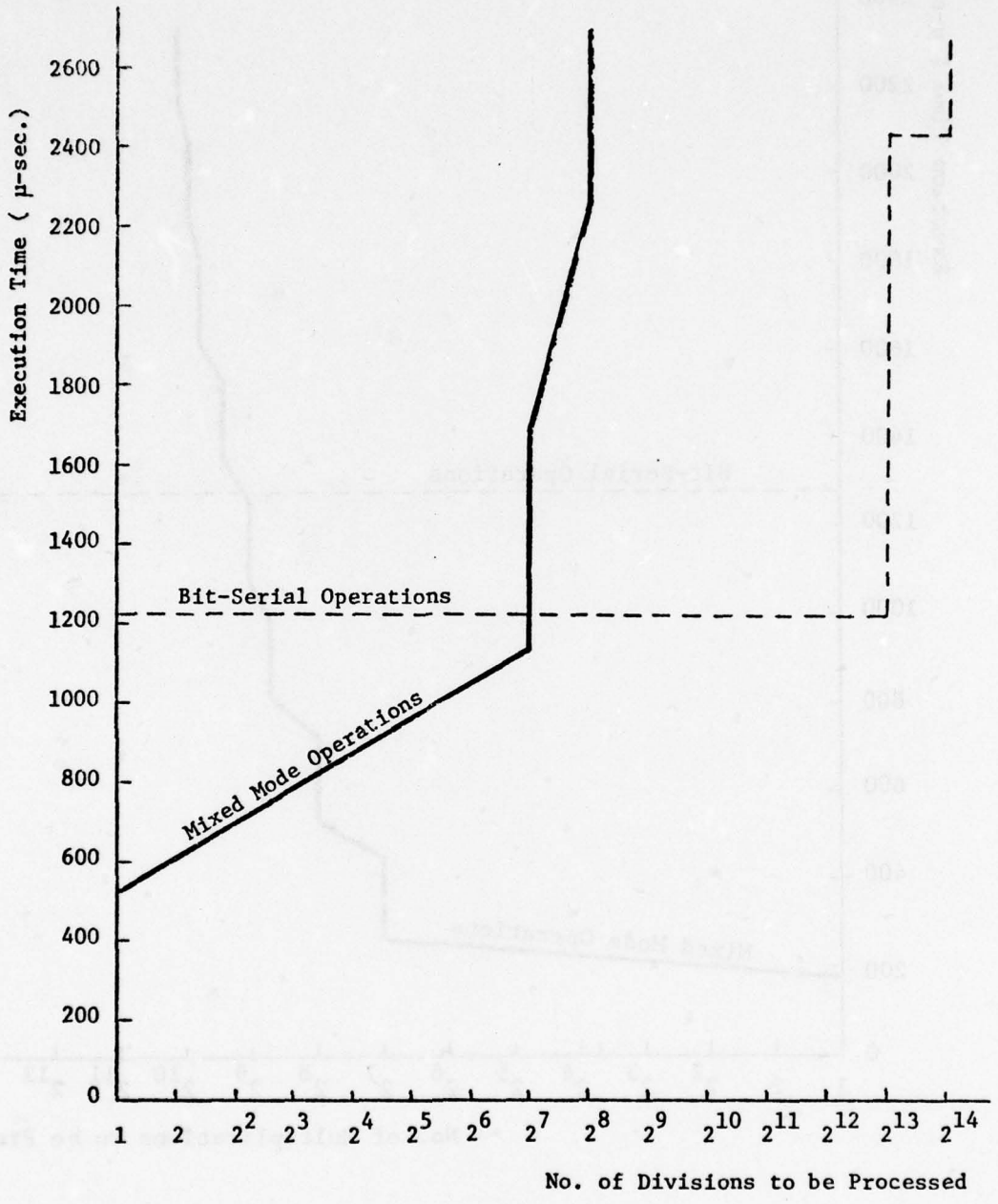


Fig. 2-13c Comparison of Execution Times of Division Operations

CHAPTER 3 SIGNED DIGIT NUMBER REPRESENTATIONS IN AN ASSOCIATIVE ARRAY STRUCTURE COMPUTER

As the cost of hardware continues to decrease, and the cost of programming and debugging continues to increase, the design convenience and the complexity of hardware are no longer the important issues. One of the important considerations is the operation speed. In the previous chapter, the carry-propagation chains in addition and subtraction as well as division are a major limiting factor in the speed of operations. In this chapter, a substantial reduction of the carry-propagation delay is achieved by the signed-digit number representations [3]. In fact, the carry-propagation length is limited to at most one digit for addition and subtraction, and to at most two digits for each addition cycle of multiplication. The structure of associative array computers is suitable for this class of representation. Some aspects of hardware requirements in the implementation of high speed operations are discussed. In order to circumvent the time consuming bit-by-bit manipulations, all the operations are considered on the basis of digit-by-digit.

3-1 Signed Digit Numbers and its Arithmetic Algorithms

The purpose of signed-digit number representations is to eliminate the carry-propagation chains by using the redundancy in the representation of the operands [3].

In a conventional number representation with an integer radix $r > 1$, each of the digits is allowed to have exactly one representation: 0, 1, 2, ..., r-1. Any n digit number has a unique representation. In the signed-digit number representation, the digits each have a unique representation but n digit numbers may have more than one representation. This characteristic is used to limit the carry length by selecting arithmeti' processes which yield results in a representation for which the carry does not propagate.

The signed-digit number representation is a positional number representation with a constant integer radix $r \geq 3$, in which the allowed values of the individual digits Z_i are from a sequence of q $(r + 2 \leq q \leq 2r - 1)$ integers:

$(-a, \ldots, -1, 0, 1, \ldots, a)$

where a is the maximum digit magnitude. The value of a is chosen from the following range:

$$\left.\begin{array}{ll} \text{for odd radices} & r_o \geq 3,\ \frac{1}{2}(r_o + 1) \leq a \leq r_o - 1 \\ \text{for even radices} & r_e \geq 4,\ \frac{1}{2} r_e + 1 \leq a \leq r_e - 1 \end{array}\right\} \quad (3.1)$$

The value of a must be less than r - 1 in order to allow a carry of 1 without generation of another carry. For example, in adding two digits of maximum value a, and a carry in of 1, the sum must be able to be represented by a sum digit of magnitude less than a and a carry. Hence,

$$a + a + 1 \leq r + a \quad (3.2)$$

which is equivalent to

$$a \leq r - 1. \quad (3.3)$$

The other bound is the crucial one for assuming that the carry propagation is limited to one position. In summing two digits whose digit sum is a, we must represent this digit sum by a carry and a digit sum which is less than a. When this is done in every digit position, the subsequent carry propagation is limited to one position. Then

$$a \geq r - (a - 1) \quad (3.4)$$

or (maximum value) $\geq$ r - (maximum value - 1).
Combining inequalities (3.3) and (3.4), establishes the relation (3.1). We should note that equation (3.1) cannot be satisfied for r= 2, the binary case. The inequality has different forms for even and odd radices since r and a must be integers.

Both positive and negative values are allowed in the digit representations. In the general case, a signed digit number is represented by m + n + 1 digits Z, (i = m, m-1, ..., 1, 0, -1, ..., -n) and has the algebraic value

$$Z = \sum_{i=-n}^{m} Z_i\, r^i \quad (3.5)$$

where the values of Z_i and r are such that

$$r \geq 3$$

and $-a \leq Z_i \leq a$ for all i.

This representation system has the following properties:

(a) The algebraic value Z = 0 has a unique representation.

(b) The carry-propagation for addition is limited to at most one digit position.

(c) The transformation between the conventional representation and the signed-digit representation exists for every algebraic value within a specific range.

The elimination of carry-propagation chains removes a fundamental constraint of digital arithmetic operations. However, effective algorithms for the elementary arithmetic operations, addition, subtraction, multiplication and division, must be developed. The remainder of this chapter is devoted to the design of arithmetic algorithms suitable for STARAN-like system using the signed digit numbers with limited carry propagation.

The serial arithmetic vs. mixed mode arithmetic alternatives are retained. The user of the system will have the option of performing serial arithmetic on a digit by digit basis, on many parallel streams or using the same hardware in mixed mode so all the digits are processed simultaneously with limited carry propagation. Only integer arithmetic will be considered.

A. Addition

There is no carry-propagation chain in the signed-digit addition, that is, any digit of the sum is a function of only two adjacent digits of the operands. The execution time of an addition is independent of the length of the operands and is equal to the time required for adding two digits plus the time for propagating the carry one digit position. Two fundamental operations in carrying out the addition are described as follows:

(a) Operation Ⓐ

The operation Ⓐ is defined by the following rules:

$$A_i \text{ Ⓐ } B_i = rC_{i+1} + S_i \qquad (3.6)$$

where r is the radix.

and $C_{i+1} = 1$ if $A_i + B_i > S_{max}$

$C_{i+1} = 0$ if $S_{max} \geq A_i + B_i \geq -S_{max}$

$C_{i+1} = -1$ if $-S_{max} > A_i + B_i$

$C_0 = 0$ the carry into the least significant digit is 0.

The operation Ⓐ adds two digits A_i and B_i and generates two digits of signed-digit number digits C_{i+1} and S_i. C_{i+1} is the carry digit and may have the value of 1, 0, or -1 ; S_i is the partial sum digit and may have the values from the following sequence:

$$(-S_{max}, \ldots, -1, 0, 1, \ldots, S_{max})$$

where $S_{max} \leq (a - 1)$

and a is the maximum digit magnitude. It is this property of S_{max} which assures limited carry propagation.

(b) Operation Ⓑ

The operation Ⓑ is defined by the following rule:

$$S_i \text{ Ⓑ } C_i = S_i$$

The operand C_i has the value of 1, 0 or -1 ; and S_i has the maximum magnitude of S_{max} which is less than or equal to (a - 1). Therefore, the operation Ⓑ produces a sum digit S_i which has a maximum magnitude of a and is within the range of signed-digit number digits. To generalize the operations, the operands may be vectors whose elements are digits.

The algorithm for signed-digit number addition with each operand being of length n digits is described as follows:

Addend		A_{n-1}	A_{n-2}	- - -	A_2	A_1	A_0	
Adder		B_{n-1}	B_{n-2}	- - -	B_2	B_1	B_0	(Operation Ⓐ)
Partial Sum		S_{n-1}	S_{n-2}	- - -	S_2	S_1	S_0	
Shifted Carry	C_n	C_{n-1}	C_{n-2}	- - -	C_2	C_1	C_0	(Operation Ⓑ)
Sum	S_n	S_{n-1}	S_{n-2}	- - -	S_2	S_1	S_0	

The first step is to generate the partial sum and the carry digits by operation Ⓐ. Shift the carry digits one position left, then, add the partial sum digits with the shifted carry digits by operation Ⓑ, generating the sum digits S_i's which are the true sum of the signed-digit numbers A and B.

The specific definitions of operation Ⓐ and operation Ⓑ depend on which of many possible signed digit systems is in use. The selection of a specific system and the encoding of signed digit number systems is discussed later in this chapter. However, if a system of radix-3 is adopted in which the digits may have values:

$$\{ -2, -1, 0, 1, 2 \}$$

the operation Ⓐ is defined by

$$A_i \text{ Ⓐ } B_i = S_i + rC_{i+1}$$

where S_i is function of A_i and B_i and is described by the following table:

S_i:

A_i \ B_i	-2	-1	0	1	2
-2	-1	0	1	-1	0
-1	0	1	-1	0	1
0	1	-1	0	1	-1
1	-1	0	1	-1	0
2	0	1	-1	0	1

and C_j is also function of A_j and B_i and is described by the following table;

C_j:

A_j \ B_i	-2	-1	0	1	2
-2	-1	-1	-1	0	0
-1	-1	-1	0	0	0
0	-1	0	0	0	1
1	0	0	0	1	1
2	0	0	1	1	1

The operation Ⓑ is defined by

$$S_i \text{ Ⓑ } C_i = S_i$$

where S_i is function of S_i and C_i and is described by the following table:

S_i:

S_i \ C_i	-1	0	1
-1	-2	-1	0
0	-1	0	1
1	0	1	2

A simple example of adding two 4-digit numbers is illustrated as follows:

Addend A		2	1	-1	0
Adder B		1	1	-2	-2
					(Operation Ⓐ)
Partial Sum S		0	-1	0	1
Shifted Carry C	1	1	-1	-1	0
					(Operation Ⓑ)
Sum S	1	1	-2	-1	1

In this example, the addend A represents a number 60, the adder B represents a number 28, the sum S represents a number 88 which is the true sum of 60 and 28.

B. Multiple-Operand Addition

Traditionally, adders are designed to add two numbers. There are arithmetic operations, such as multiplication, which require the addition of more than two numbers. The redundancy in the signed-digit numbers representation enables us to design a multiple-operand adder without the carry-propagation chains. The operation Ⓐ can be extended to allow simultaneously adding more than two digits if values of the carry digit $|C_i| > 1$ are allowed and the radix r is sufficiently large.

A two-step multiple-operand addition method is shown as the following steps:

(1) $\textcircled{A}\ (U_i, V_i, W_i, \ldots \text{ n operands}) = r\,C_{i+1} + S_i$ (3.8)

(2) $S_i\ \textcircled{B}\ C_i = S_i$ (3.9)

The first step requires

$$na \leq C_{max} + S_{max} \tag{3.10}$$

and

$$C_{max} + S_{max} = a \tag{3.11}$$

where C_{max} is the maximum digit value of the carry digits;

S_{max} is the maximum digit value of the partial sum;

and a is the maximum digit value of signed digit numbers.

Equations (3.10) and (3.11) can be restated as:

$$n \leq r - \left(\frac{r-1}{a}\right) S_{max} \tag{3.12}$$

or

$$n \leq 1 + \left(\frac{r-1}{a}\right) C_{max} \tag{3.13}$$

The second step adds the shifted carry C_i to the partial sum S_i to get

the true sum S_j , i.e.

$$S_j \textcircled{B} C_j = S_i \tag{3.14}$$

This two-step multiple-operand addition method has a rigid restriction on the number of operands n (the number of data to be added up). For example, while r = 10, a = 7, S_{max} = 5, then C_{max} = 2, in this case, the maximum number of operands n is three. In other words, in this number representation, only three-operand addition can be performed without carry-propagation chanis.

Table 3-1 lists some typical examples of signed digit number representation to illustrate the relationships of r, a, S_{max} and n in equation (3.12).

For large values of n, a better method with three steps may be used. The three-step method is executed in three successive steps:

(1) $\textcircled{A}$ (U_i, V_i, W_i, - - - - - - - - - - -) = r C'_{i+1} + S'_i (3.15)

(2) $S'_i \textcircled{A} C'_i = C_{i+1} r + S_i$ (3.16)

(3) $S_i \textcircled{B} C_i = S_i$ (3.17)

In the first step, a partial sum S'_i and a pseudo carry C'_{i+1} which is in the range from a to -a are generated. Then the pseudo carry is shifted one position left, and added to the partial sum in the second step. The partial sum S_i and the carry C_{i+1} are generated in this step, the value of carry is restricted to the range from (a-1) to -(a-1). The third step is to add the shifted carry to the partial sum to generate the true sum. This method allows any number of operands up to the number (r+1). In the step (1) of this method, the Operation $\textcircled{A}$ is generalized in such a way that it can handle n operands, and the carry C'_{i+1} is allowed in the range from a to -a, and two operands operation is a special case.

r	a	S_{max}	n(maximum no. of operands)
4	3	2	2
8	7	6	2
8	7	5	3
8	7	4	4
8	6	5	2
8	6	4	3
8	5	4	2
10	9	8	2
10	9	7	3
10	9	6	4
10	9	5	5
10	8	7	2
10	8	6	3
10	8	5	4
10	7	6	2
10	7	5	3
10	6	5	2
12	11	6	6
16	15	8	8
31	31	16	16

Table 3-1 Relations of r, a, S_{max} and n for Multiple-Operand Signed Digit Addition

C. Subtraction

To perform subtraction, the sign of subtrahend is changed by inverting the signs of non-zero digits, and addition is performed.

A fundamental operation Ⓝ which generates the negative digits of the operand is defined as follows

$$Ⓝ\ A_i = -A_i \tag{3.18}$$

D. Multiplication

For binary numbers, multiplication can be performed as a sequence of shifts and additions. For the carry-save technique, the multiplication time of 2's complement representation is proportional to the bit-length of the multiplier. In the signed-digit representation, the digits are the basic unit of the numbers. The multiplication is performed as a sequence of single digit multiplications and signed-digit additions. Hence, the multiplication time is proportional to the digit-length of multiplier.

Given the radix $r \geq 3$, the signed-digit multiplicand A (A_{m-1} A_{m-2} A_{m-3} - - - - - - - - - - A_2 A_1 A_0) and the signed-digit multiplier B (B_{n-1} B_{n-2} - - - - - - - - - - - - B_2 B_1 B_0), the product P of A and B is shown as follows:

$$\begin{aligned} P &= A \times B \\ &= A \times \sum_{i=0}^{n-1} B_i\, r^i \\ &= \sum_{i=0}^{n-1} A \times B_i\, r^i \end{aligned} \tag{3.19}$$

The generation of product P can be rewritten into the following iterative process:

$$
\left.
\begin{aligned}
& P_0 = 0 \quad \text{initial value of partial product} \\
& P_{j+1} = (P_j + A \times B_j) \times r \\
& \qquad \text{for } j = 0, 1, 2, - - - - - - - , n-2 \\
& P_n = P_{n-1} + A \times B_{n-1} \\
& \text{where} \quad P_n = P \text{ is the product.}
\end{aligned}
\right\} \qquad (3.20)
$$

In order to carry out the signed-digit multiplication effectively, a fundamental operation Ⓜ is defined as follows:

The Operation Ⓜ multiplies two digits A_i and B_j of two signed-digit numbers A and B and produces a partial product digit P_{i+j-1} and a carry digit MC_{i+j} such that

$$
\left.
\begin{aligned}
& A_i \text{ Ⓜ } B_j = r\, MC_{i+j} + P_{i+j-1} \\
& \text{where} \quad -a \leq P_{i+j-1} \leq a \text{ (maximum digit value)} \\
& \text{and} \quad r \geq a + 1.
\end{aligned}
\right\} \qquad (3.21)
$$

The simplest signed digit multiplication method is repeatedly generating the partial products of each digit of multiplier and all digits of multiplicand. Then shift the partial products to a proper position and add them up pair by pair to obtain the true product. This multiplication method utilizes only two-operand additions and operation Ⓜ.

Assume that the number representation is chosen such that the two-step three-operand addition is possible, i.e.

$$
3 \leq r - \left(\frac{r-1}{a} \right) S_{max}
$$

or

$$
3 \leq 1 + \left(\frac{r-1}{a} \right) C_{max} .
$$

An example of multiplication is shown in Fig. 3-1 which illustrates a 4-digit number multiplication. It is seen that the three-operand addition process is just sufficient to combine the previous partial product with the new carry and new partial product resulting from $A \times B_i$. The symbols in the figure are explained as follows:

A_i : the ith digit of multiplicand.

B_i : the ith digit of multiplier.

$P_{i\,j}$: the partial product of ith multiplier digit and the jth multiplicand digit.

$MC_{i\,j}$: the carry of the product of ith multiplier digit and the (j-1)th multiplicand digit.

$S_{i\,j}$: the partial sum of $S_{(i-1)\,j}$, $P_{i\,j}$ and $MC_{i\,j}$.

$C_{i\,(j+1)}$: the addition carry of $S_{(i-1)\,j}$, $P_{i\,j}$ and $MC_{i\,j}$.

$S_{i\,j}$: the sum of $S_{i\,j}$ and $C_{i\,j}$.

The operations of each step are shown on the right of the figure.

A modified algorithm which uses the carry-save idea is described in Fig. 3-2 for 4-digit numbers. In this case, a four-operand addition is assumed. A slightly faster multiplication operation is achieved. The symbols for this figure are described as follows:

A_i : the ith digit of multiplicand.

B_i : the ith digit of multiplier.

$P_{i\,j}$: the partial product of ith multiplier digit and the jth multiplicand digit.

$MC_{i\,j}$: the carry of the product of ith multiplier digit and the (j-1)th multiplicand digit.

$S_{i\,j}$: the partial sum of $S_{(i-1)\,j}$, $C_{(i-1)\,j}$, $P_{i\,j}$, and $MC_{i\,j}$.

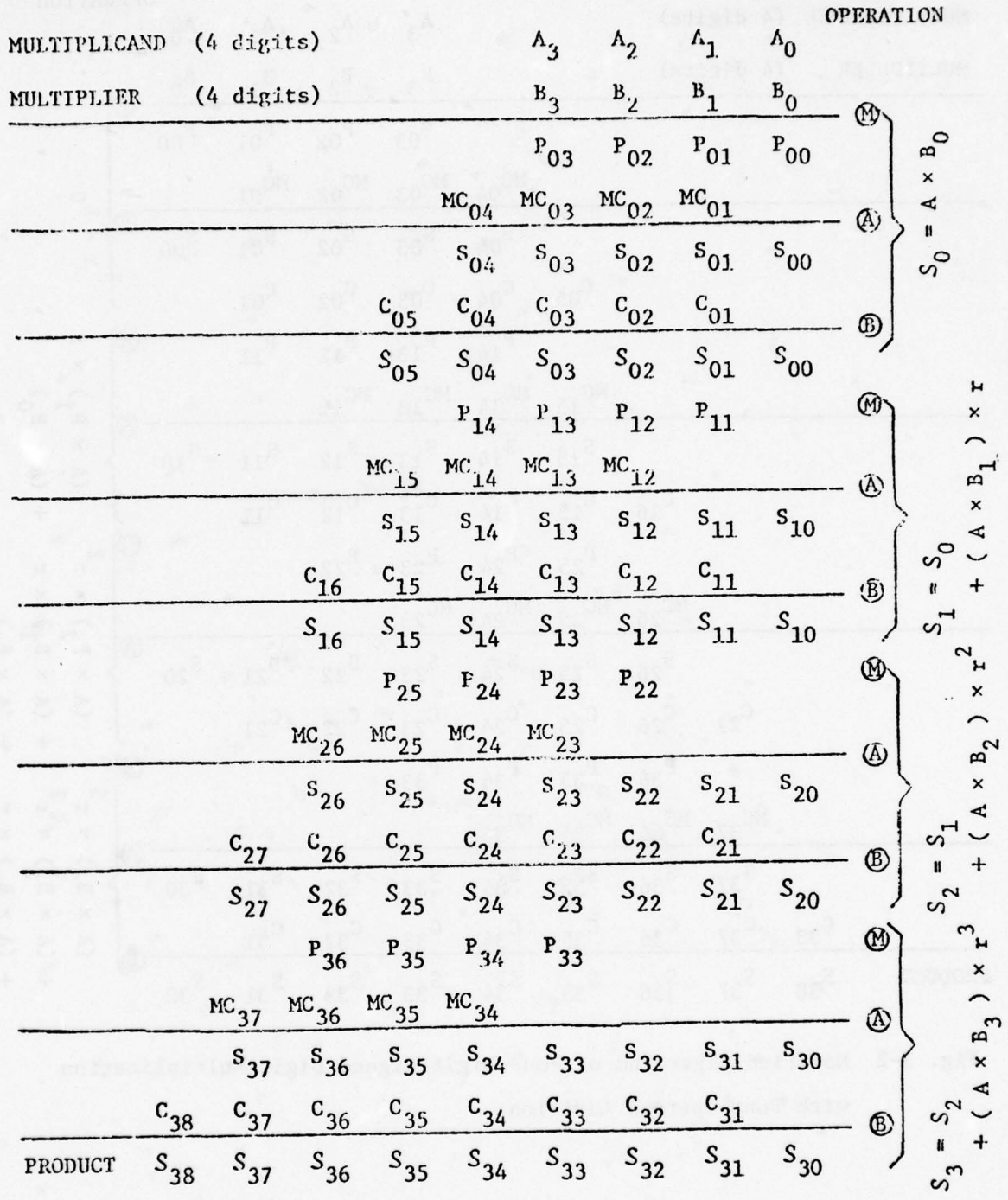


Fig. 3-1 Algorithm of Four-Digit Signed Digit Number Multiplication with Two-Step Three-Operand Addition

										OPERATION
MULTIPLICAND (4 digits)					A_3	A_2	A_1	A_0		
MULTIPLIER (4 digits)					B_3	B_2	B_1	B_0		
					P_{03}	P_{02}	P_{01}	P_{00}	M	
				MC_{04}	MC_{03}	MC_{02}	MC_{01}			
				S_{04}	S_{03}	S_{02}	S_{01}	S_{00}	A	$A \times B_0$
			C_{05}	C_{04}	C_{03}	C_{02}	C_{01}			
				P_{14}	P_{13}	P_{12}	P_{11}		M	
			MC_{15}	MC_{14}	MC_{13}	MC_{12}				
			S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	A	$(A \times B_1) \times r + (A \times B_0)$
		C_{16}	C_{15}	C_{14}	C_{13}	C_{12}	C_{11}			
			P_{25}	P_{24}	P_{23}	P_{22}			M	
		MC_{26}	MC_{25}	MC_{24}	MC_{23}					
		S_{26}	S_{25}	S_{24}	S_{23}	S_{22}	S_{21}	S_{20}	A	$(A \times B_2) \times r^2 + (A \times B_1) \times r + (A \times B_0)$
	C_{27}	C_{26}	C_{25}	C_{24}	C_{23}	C_{22}	C_{21}			
		P_{36}	P_{35}	P_{34}	P_{33}				M	
	MC_{37}	MC_{36}	MC_{35}	MC_{34}						
	S_{37}	S_{36}	S_{35}	S_{34}	S_{33}	S_{32}	S_{31}	S_{30}	A	$(A \times B_3) \times r^3 + (A \times B_2) \times r^2 + (A \times B_1) \times r + (A \times B_0)$
C_{38}	C_{37}	C_{36}	C_{35}	C_{34}	C_{33}	C_{32}	C_{31}		B	
PRODUCT S_{38}	S_{37}	S_{36}	S_{35}	S_{34}	S_{33}	S_{32}	S_{31}	S_{30}		

Fig. 3-2 Modified Algorithm of Four-Digit Signed Digit Multiplication with Four-Operand Addition

$C_{i\ (j+1)}$: the addition carry of $S_{(i-1)\ j}$, $C_{(i-1)\ j}$, $P_{i\ j}$ and $MC_{i\ j}$.

$S_{j\ j}$: the sum of $S_{i\ j}$ and $C_{i\ j}$.

E. Division

Signed-digit division is performed as a sequence of shifts and additions or subtractions. Given the dividend Z and the divisor D, the quotient Q ($Q_{n-1}\ Q_{n-2}$ - - - - - - - - $Q_2\ Q_1\ Q_0$) and remainder R are specified by the following rules:

$$\left.\begin{aligned} & Z = D \times Q + R \\ \text{and} \quad & |R| < |D| \end{aligned}\right\} \quad (3.22)$$

The above division rule can be rewritten into:

$$\left.\begin{aligned} R &= Z - D \times Q \\ &= Z - D \times (Q_{n-1}\ r^{n-1} + Q_{n-2}\ r^{n-2} + \cdots\cdots\cdots \\ &\qquad + Q_2\ r^2 + Q_1\ r^1 + Q_0) \\ &= Z - D \times Q_{n-1}\ r^{n-1} - D \times Q_{n-2}\ r^{n-2} - \cdots\cdots\cdots \\ &\qquad - D \times Q_1\ r - D \times Q_0 \end{aligned}\right\} \quad (3.23)$$

This equation shows that the signed-digit division breaks down into simpler steps:

$$\left.\begin{aligned} & R_0 = Z \\ & R_{j+1} = R_j - D \times Q_{n-j-1}\ r \quad \text{for } j = 0, 1, \cdots\cdots, n-1. \end{aligned}\right\} \quad (3.24)$$

Each of the Q_{n-j-1} must be such that

$$-a \leq Q_{n-j+1} \leq a \text{ (maximum digit magnitude)} \tag{3.25}$$

which requires that

$$-\left(\frac{a}{r-1}\right) D\, r^{n-j} \leq R_j \leq \left(\frac{a}{r-1}\right) D\, r^{n-j} \tag{3.26}$$

so it can be used in the succeeding step. The simplest method for each step is repeatedly adding or subtracting the divisor until the remainder R_j is within the specified range.

The division algorithm for 8-digit signed digit numbers is described in Fig. 3-3. In each step, the divisor is repeatedly used to decrease the magnitude of dividend until the sign of dividend is changed. Then a comparison is made between the two dividends of different signs and the one with smaller magnitude value is chosen. The quotient digit is altered accordingly. So, in the algorithm described, the condition

$$-\frac{1}{2} D\, r^{n-j} \leq R_j \leq \frac{1}{2} D\, r^{n-j}$$

is chosen instead of the general rule in equation (3.26).

A maximum of five additions or subtractions required to change the sign of dividend at any cycle. The average number of additions or subtractions is 2.04. An additional subtraction is needed to compare the two dividends. About one half of the time this comparison will call for a dividend restoration addition or subtraction. On the average, 3.54 additions or subtractions must be performed to generate one quotient digit. It is suprising that the average number of additions or subtractions required to change the sign of the dividend is so low, 2.04. This result is obtained by an analysis based upon a random distribution of divisors and dividends values.

Other methods of division have been studied for the signed-digit

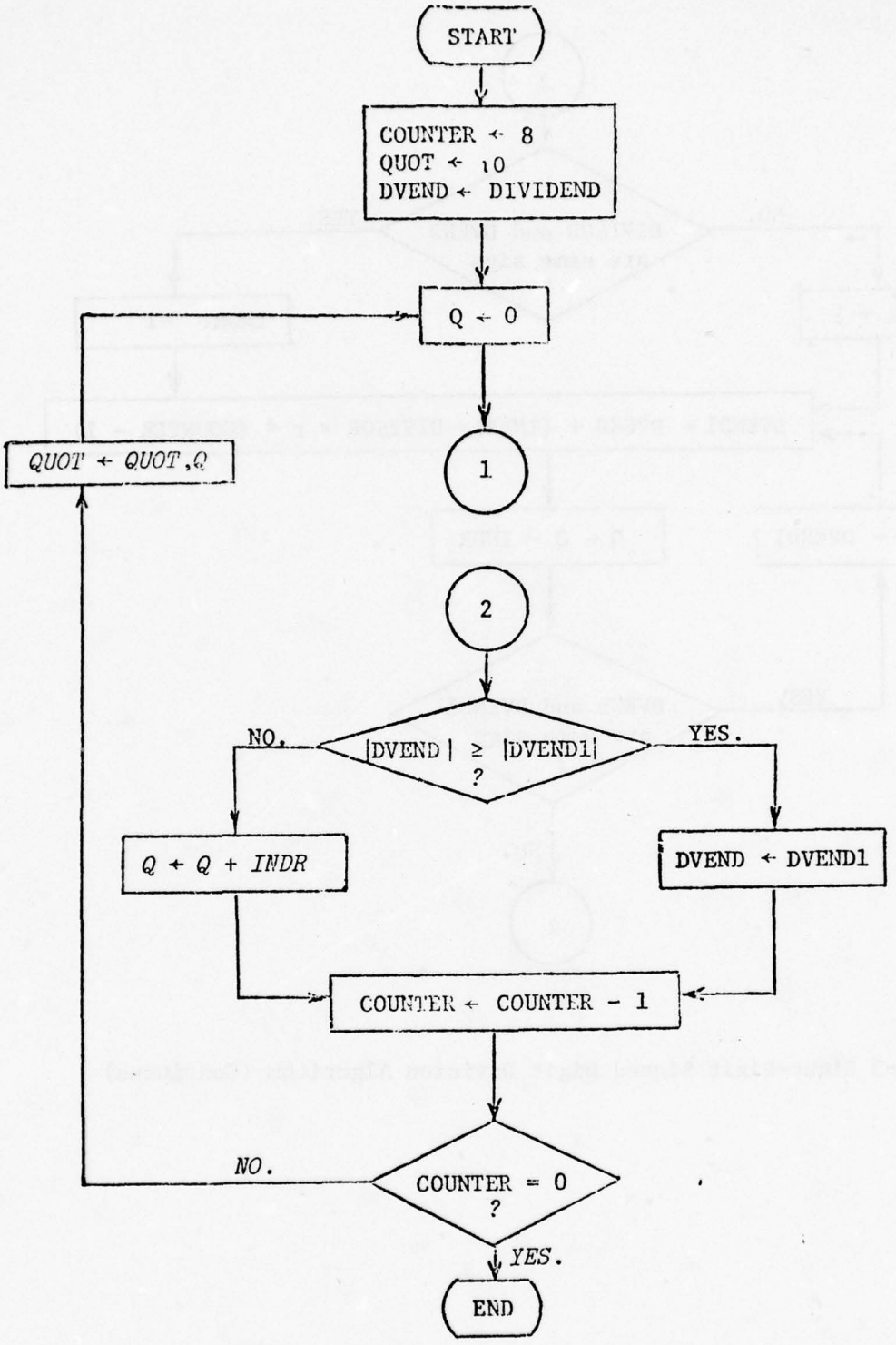


Fig. 3-3 Eight-Digit Signed Digit Division Algorithm

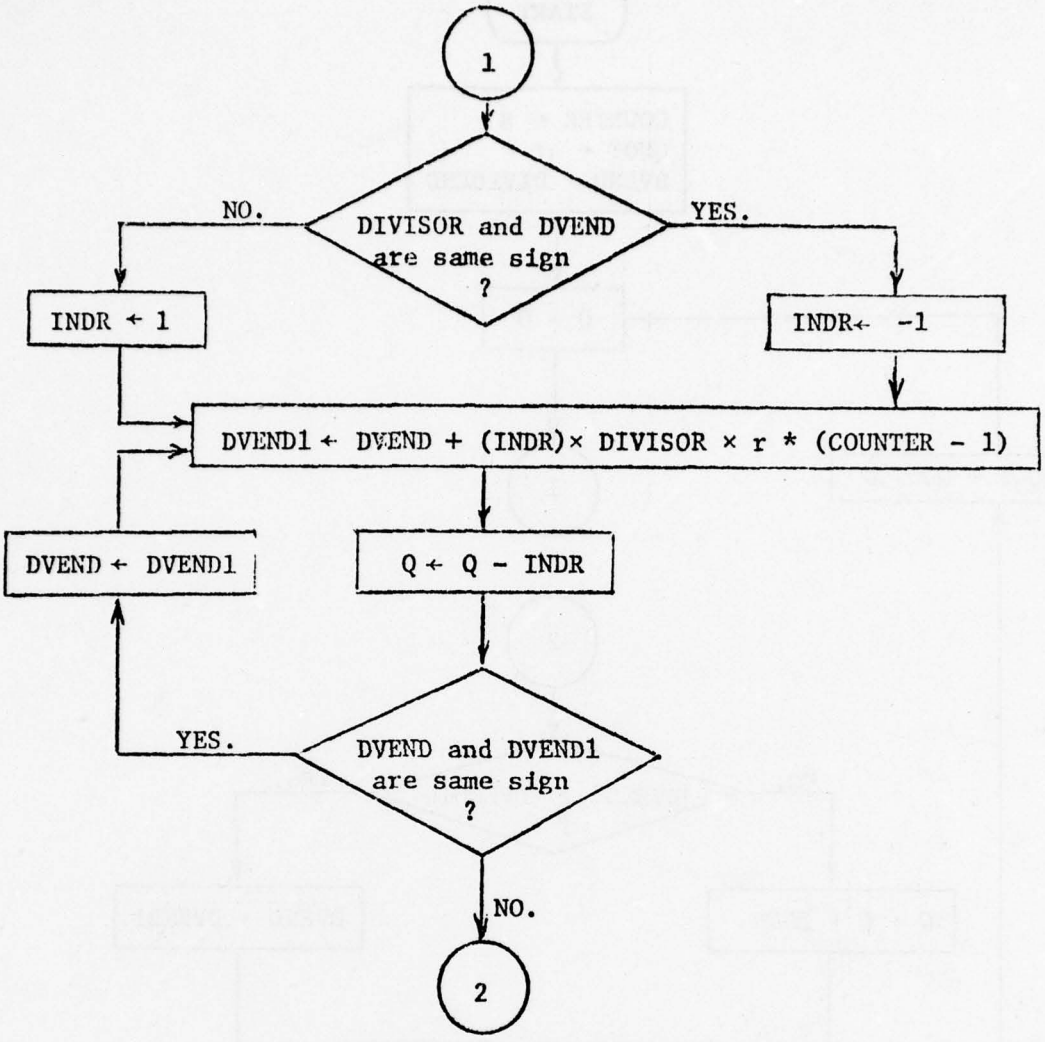


Fig. 3-3 Eight-Digit Signed Digit Division Algorithm (Continous)

number representations. One method [23] breaks the division into simpler steps, each of which is the division of an (n+1)-place number U by the n-place divisor V, which $0 \leq \left|\frac{U}{V}\right| < r$; the remainder after each step is less than r/2, so it may be used in the succeeding step. The calculation of quotient digits q_i is based on several leading digits of the operands. The method which is based on the two leading digits of U and the leading digit of V requires that

$$v_1 \geq \lfloor r/2 \rfloor \quad \text{where } v_1 \text{ is the leading digit of V.}$$

This requirement leads to an obstacle since there is no general rule which maps the signed-digit divisor into this specified range. Even if this obstacle were overcome, the method does not seem to offer much reduction in the computation required to produce a quotient digit.

3-2 Implementation of the Signed Digit Numbers in the Associative Array Structure Computers

A specific signed-digit number system is suggested here, and an associative array structure computer is proposed for this number system. The implementation of the specific signed-digit number arithmetic operations in the proposed associative array structure computer is discussed.

A. A Proposed Signed Digit Number System

There are many choices of signed digit number systems. A specific digit number representation is suggested here.

Since the decimal habit is deeply ingrained in the human society, and too large a radix number requires too much logic for encoding digits, the radix r = 10 is chosen. The allowed values of the individual digits Z_i are chosen from the sequence of integers:

(-7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7)

There are fifteen distinct states for each digit, so at least four

bits are needed to encode one digit. The 2's complement 4-bit code which provides a unique zero representation is suggested. This choice of number system does not involve complex logic and also allows the two-step three-operand addition, therefore, the two-step three-operand multiplication algorithm is applicable. The division algorithm described is also effective.

B. A Proposed Associative Array Structure Computer

In the associative array computer systems, the arithmetic operations are executed in the associative arrays. An associative array structure which is suitable for the implementation of the radix-10 signed-digit number arithmetic operations is proposed in Fig. 3-4. Each associative array consists of 256 by 256 bits square multidimensional access (MDA) memory and five response store registers M, X, Y, Z and W. The multidimensional access (MDA) memory not only can be accessed in word mode and bit-slice mode, but also can be accessed in mixed mode, i.e., some bits from some words. The mixed mode addressing scheme allows the processing of all bits or all digits of the same data words. Each of the response store registers is 256 bits long. Associated with the response store registers, there are 64 units of micro-processors which are 4-bit parallel processing units with a modest size of Read-Only Memory (ROM). User programmable Read-Only Memory would provide more flexibility. We require that the table look-up techniques are possible in ROM. Every four bits of each of the response store registers are connected to the corresponding micro-processors. The proposed structure may be considered as 64 units of mini-processors, each of which has five 4-bit registers and 256 4-bit words memory. This structure is also suitable for manipulating decimal numbers whose digits are encoded into four bits.

A modified STARAN system is shown in Fig. 3-5. The associative array structures replace the associative array memories, and the AP control and the PIO control are able to control the five response store registers in the associative array structures.

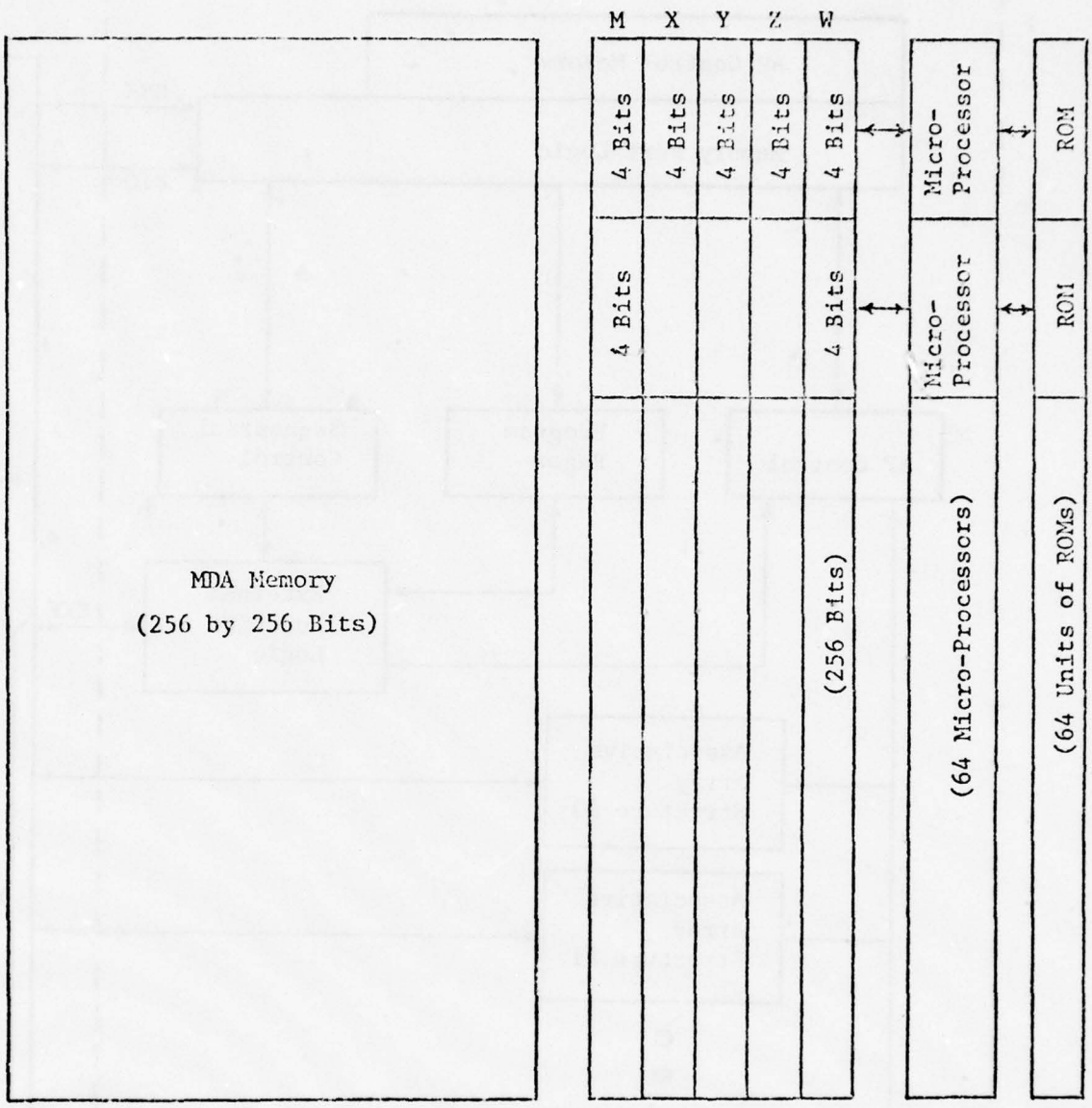


Fig. 3-4 A Proposed Associative Array Structure

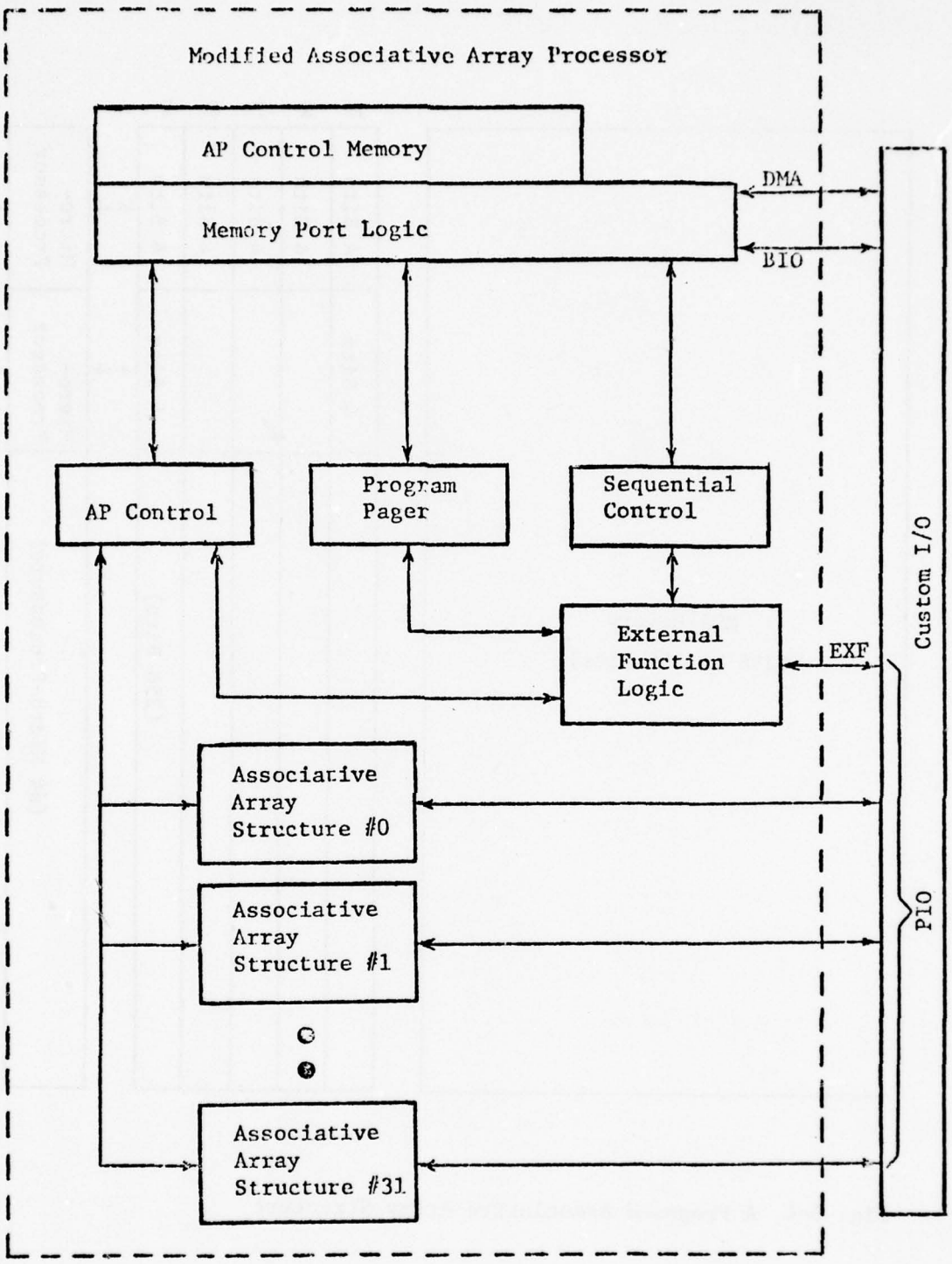


Fig. 3-5 Modified STARAN System Block Diagram

In the existing STARAN system, the instruction speeds are determined by the speed of data storage and retrieval in the MDA memory. Since the same MDA memory can be used in the proposed system, it is assumed that the memory determines the time required for the fundamental operations. This will allow a reasonably accurate estimation of the relative speeds of mixed mode signed digit operations.

C. Implementation Considerations

The signed-digit arithmeric operations are based on combining pairs of digits. Each digit is encoded into four bits. The 2's complement codes are chosen in the implementation because they have a unique zero representation. The implementation will assume a data word of length 32 bits, i.e., eight digits with four bits per digit, so that up to eight words in each associative array structure may be processed in parallel.

The fundamental operations are implemented in the micro-processors and the arithmetic algorithms are implemented in the associative array structure control programs. In order to implement the fundamental operations, some desirable features of the micro-processors are:

(1) Four four-bit general purpose registers.
(2) A four-bit arithmetic and logic unit.
(3) Instruction set includes the conditional branch.
(4) User programmable Read-Only Memory (ROM) with size 2K by 8 bits, and table look-up techniques are possible.
(5) Eight-bit parallel data channels are provided to transfer data between the Read-Only Memory Data Register (ROMDR) and the general purpose registers, also between the Read-Only Memory Address Register (ROMAR) and the general purpose registers.

The micro-processors share four general purpose registers with the MDA memory. In other words, every four bits of the response store registers X, Y, Z and W are also general purpose registers of the micro-processors. The table look-up procedure is to load two 4-bit data from two general purpose registers to the ROMAR, search the contents of the address of

ROMAR, obtain the contents in ROMDR, and then transfer the 8-bit data from ROMDR to two of the 4-bit general purpose registers. All these actions can be implemented as instruction in the micro-processors with a few micro-instructions.

The specific fundamental operations for the proposed signed-digit number arithmetic are described in Appendix C. Fundamental operations are executed in the micro-processors by table-look-up techniques. Based on the proposed number system, the fundamental operation Ⓐ needs about 256 by 8 bits of ROM for two-operand case. The same will be needed for operation Ⓜ. The other fundamental operations need smaller sizes of ROM. It is estimated that approximately 2K by 8 bits ROM is sufficient to implement all of the two-operand fundamental operations. The APL documentations of signed digit arithmetic in Appendix D are based on the assumption that only two-operand addition is possible.

Further increase in the operating speed of multiplication could be achieved by extending the addition operation to three operands, but this extension would increase the required size of ROM up to 4K words for Operation Ⓐ. The fundamental operations are the operations described earlier for arithmetic processes such as Operation Ⓐ, Operation Ⓑ, Operation Ⓜ and Operation Ⓝ. In addition, a few operations have been included which were found desirable. The individual operations used in the algorithms are limited to those included in Appendix C. APL is used to document and to simulate in detail the signed digit arithmetic processes. Appendix D includes the detailed APL description of the algorithms and some simulation runs.

Even faster speeds could be obtained if the fundamental operations are implemented with combinatorial circuitry. It has been assumed that the fundamental operations in micro-processors is executed by table look-up method. This will permit the use of general purpose high speed ROMs for signed digit arithmetic.

3-3 Speed Gains

The signed digit arithmetic operations can be implemented in the modified STARAN system which is shown in Fig. 3-5. The execution times and processing rates are estimated and are included in Appendix E. This estimation based on the assumption that the table look-up speed in the micro-processors is about the same as STARAN's memory store cycle time.

Fig. 3-6a, b and c show the comparisons of execution times between the eight-digit radix-10 signed-digit number arithmetic operations and the 32-bit 2's complement number mixed mode operations. The addition and subtraction of signed digit numbers are improved about 12 to 16 times; the multiplication operations are about 8 to 12 times; and the division operations are about 2 to 3 times. From the Fig. 3-6a, b and c, we have to note that the execution times of addition, subtraction and multiplication are independent of the number of data words in parallel if the size of associative array structure is unlimited. The execution times of signed-digit division increases far slower than those of 2's complement mixed mode operations as the number of data in parallel increases.

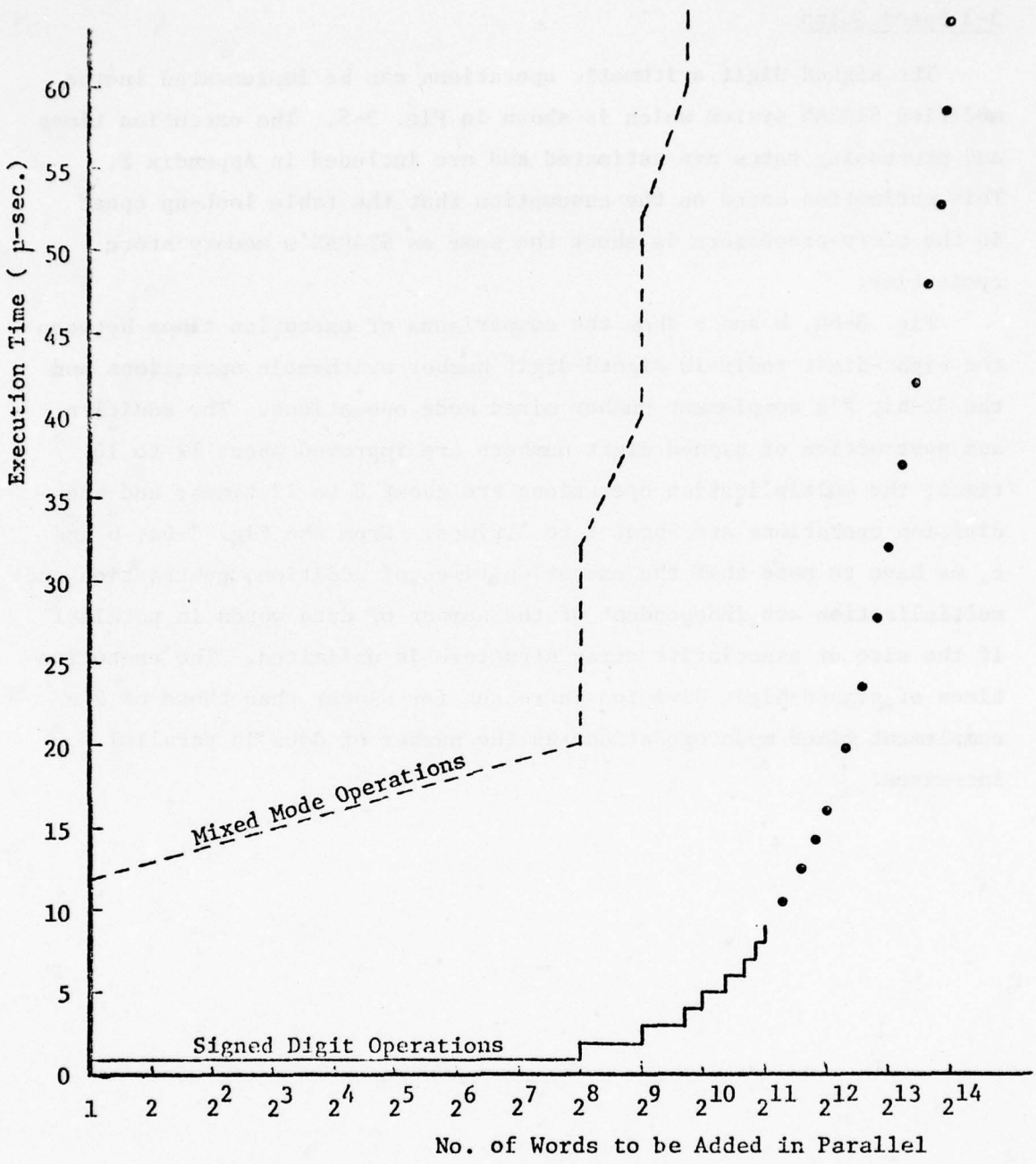


Fig. 3-6a Comparison of Execution Times of Addition/Subtraction

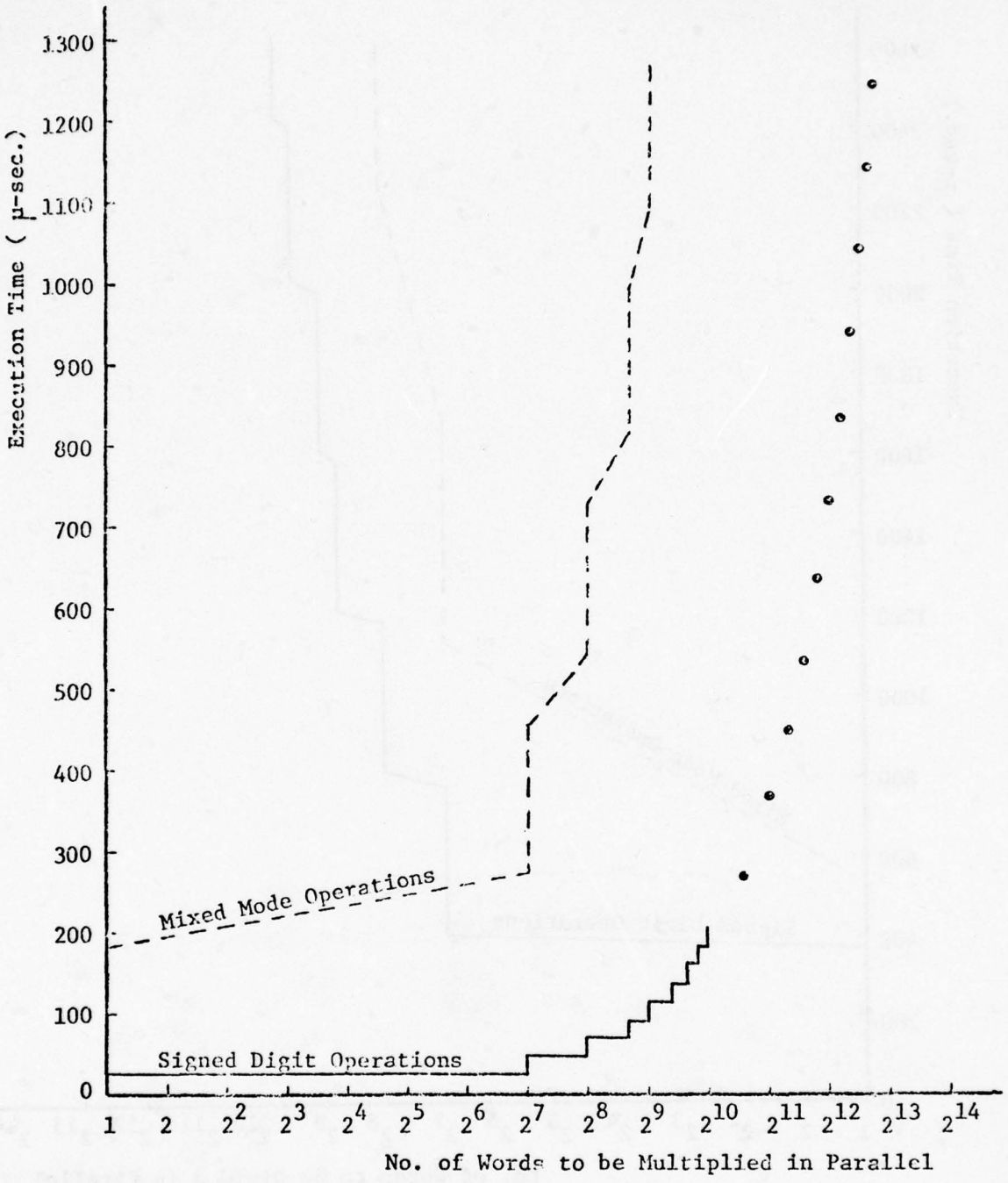


Fig. 3-6b Comparison of Execution Times of Multiplication

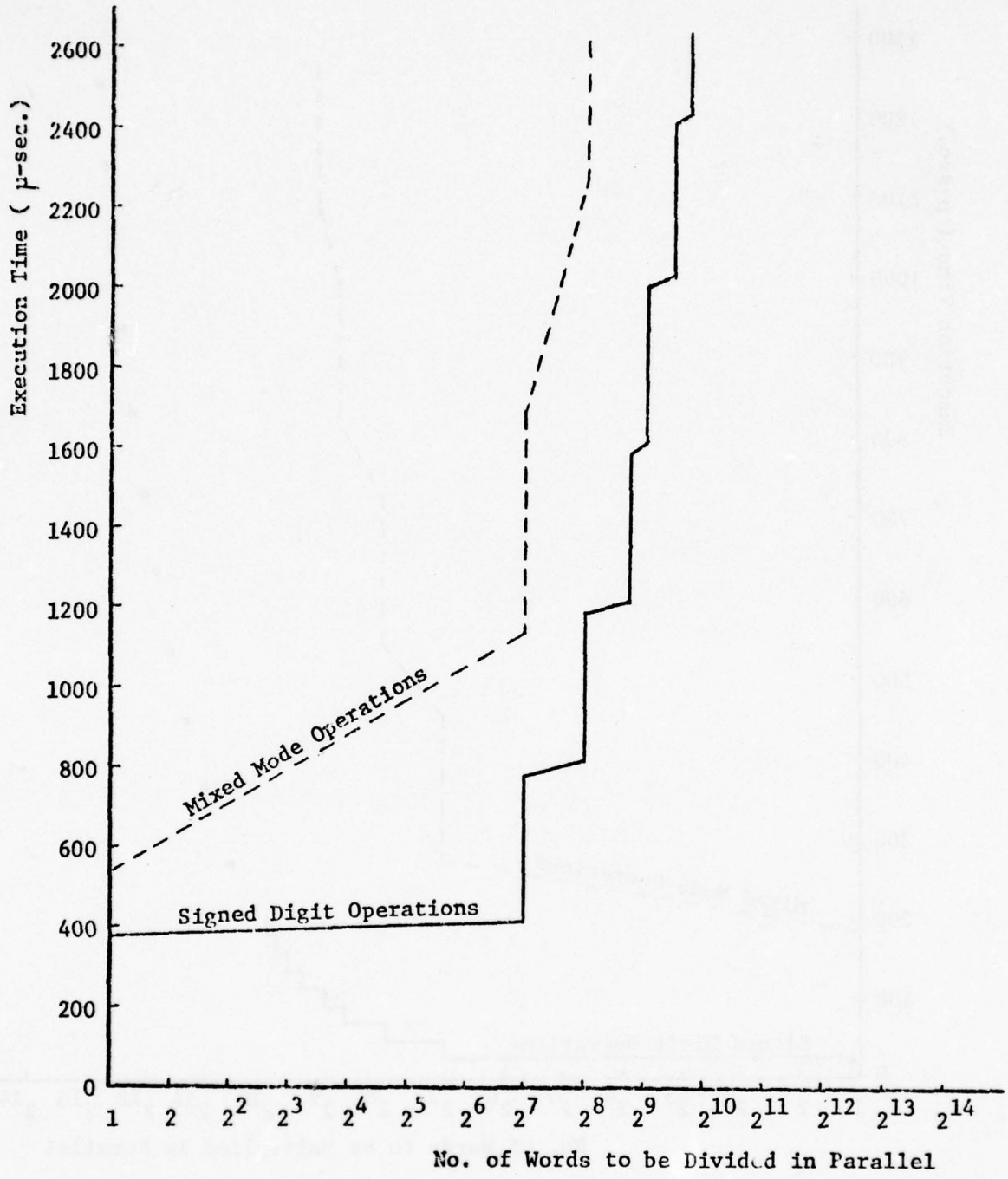


Fig. 3-6c Comparison of Execution Times of Division

CHAPTER 4 CONCLUSIONS

The design of arithmetic and logic unit is an important subject in the development of a new computer system. The use of multidimensional array memory in implementing a parallel processor seems to provide an opportunity for improving processing speed at low cost. This dissertation proposes methods for implementation of parallel arithmetic operations by utilizing the unused hardware facilities to speed up the operating speeds.

Chapter 2 discusses the implementation of 2's complement binary number representations in the STARAN system. The idea of using the mixed mode addressing capability in the multidimensional access memory to implement the parallel arithmetic operations in which all the bits within the data are processed in parallel and the data streams are also processed in parallel is original. Within the capacity of the multidimensional memory, the mixed mode addition/subtraction operations are about two times faster than the bit-serial operations; the mixed mode multiplication operations are about five to seven times improvement; and the mixed mode division operations are about two times faster. The improvement in operation speeds is significant.

Chapter 3 studies the signed digit number representation in a STARAN-like associative array computer. Addition in signed digit number systems requires a carry operation of only one digit position. The arithmetic algorithms for processing all the digits within the data in parallel as well as processing the data streams in parallel are developed. The connection of micro-processors and the signed digit number systems for parallel processing is an original work. That is, the digit operations are carried out by means of hardware table look-up technique while the arithmetic algorithms themselves are developed in software. The execution times for decimal signed digit number addition/subtraction operations in STARAN-like system are about twelve to sixteen times faster than the mixed mode operations in STARAN; the

signed digit multiplication operations are about eight to twelve times faster; and the division operations are about two to three times faster. All this effort is to improve the performance for general application problems. Low cost micro-processors are employed in the implementation of a dicimal signed digit number representation, and the improvement of processing speeds is significant.

Further study of the error-detecting and error-correcting properties of this kind of number representations is needed. The development of floating point arithmetic in mixed mode is an interesting area. An improvement of the division algorithm and the completion of the logic design in Chapter 3 are also needed. Another area of further work is the general problem of full utilization of the hardware facilities for different sizes of application problems.

APPENDIX A APL DOCUMENTATION OF THE MIXED MODE OPERATIONS

The algorithms of Addition, Subtraction, Multiplication and Division are simulated in APL functions. The addition and subtraction algorithms operate on 32-bit 2's complement number representation. The multiplication and division algorithms described operate on positive numbers of 32-bit 2's complement numbers. In order to handle negative numbers, the conversion of negaitve numbers to positive, or vice verse, has to be done before and after the algorithms are applied. These algorithms are implemented on the Rome Air Development Center Associative Processor (RADCAP) system in R.A.D.C., Rome, New York [22]. The APL simulation programs are listed in the following. Every operation shown in the APL version corresponds to one or more machine operations on STARAN.

A. APL Documentation of Mixed Mode Addition on STARAN

```
      ∇MADD[□]∇
    ∇ MADD
[1]   ⍳0
[2]   'ADDEND:'
[3]   ADDEND←:□
[4]   '2''S COMPLEMENT BINARY ADDEND:'
[5]   □←X←(32⍴2)⊤ADDEND
[6]   ⍳0
[7]   'ADDER:'
[8]   ADDER←□
[9]   '2''S COMPLEMENT BINARY ADDER:'
[10]  □←Y←(32⍴2)⊤ADDER
[11]  ⍳0
[12] LA:X←X≠Y
[13]  ⍝ X IS THE PARTIAL SUM
[14]  Y←Y∧~X
[15]  ⍝ Y IS CARRY
[16]  Y←1↓Y,0
[17]  ⍝ Y IS THE SHIFTED CARRY
[18]  →LA×⍳(+/Y=1)≠0
[19]  ⍝ BRANCH TO LA IF CARRY IS NOT ALL ZEROS
[20]  '2''S COMPLEMENT BINARY SUM:'
[21]  □←SUM←X
[22]  'SUM:'
[23]  □←-⍉((2×SUM[1]-1)×(SUM[1]+(32⍴2)⊥((32⍴SUM[1])≠SUM))
    ∇
```

Example 1 of Mixed Mode Addition

```
     MADD

ADDEND:
□:
        239701
2'S COMPLEMENT BINARY ADDEND:
0  0  0  0  0  0  0  0  0  0  0  0  0  1  1
      1  1  1  1  0  1  1  0  0  1  1  1  0  1
      0  1

ADDER:
□:
        3310764
2'S COMPLEMENT BINARY ADDER:
0  0  0  0  0  0  0  0  0  0  1  1  0  0  1  0
      1  0  0  0  0  1  0  0  1  0  1  0  1  1
      0  0

2'S COMPLEMENT BINARY SUM:
0  0  0  0  0  0  0  0  0  0  1  1  0  1  1  0  0  1  1  1
      1  0  1  1  0  0  1  0  0  0  0  1
SUM:
3570465
```

Example 2 of Mixed Mode Addition

```
     MADD

ADDEND:
□:
        -9133051
2'S COMPLEMENT BINARY ADDEND:
1  1  1  1  1  1  1  1  0  1  1  1  0  1  0  0
      1  0  1  0  0  1  0  0  0  0  0  0  0  1
      0  1

ADDER:
□:
        65210
2'S COMPLEMENT BINARY ADDER:
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
      1  1  1  1  1  1  1  0  1  0  1  1  1  0
      1  0

2'S COMPLEMENT BINARY SUM:
1  1  1  1  1  1  1  1  0  1  1  1  0  1  0  1  1  0  1  0
      0  0  1  0  1  0  1  1  1  1  1  1
SUM:
-9067841
```

B. APL Documentation of Mixed Mode Subtraction on STARAN

```
      ∇MSUB[□]∇
    ∇ MSUB
[1]     ⍳0
[2]     'MINUEND:'
[3]     MINUEND←□
[4]     '2''S COMPLEMENT BINARY MINUEND:'
[5]     □←X←(32⍴2)⊤MINUEND
[6]     ⍝ 2''S COMPLEMENT OF MINUEND
[7]     ⍳0
[8]     'SUBTRAHEND:'
[9]     SUBTRAHEND←□
[10]    '2''S COMPLEMENT BINARY SUBTRAHEND:'
[11]    □←Y←(32⍴2)⊤SUBTRAHEND
[12]    ⍝ 2''S COMPLEMENT OF SUBTRAHEND
[13]    ⍳0
[14]  LA:X←X≠Y
[15]    ⍝ X IS THE PARTIAL DIFFERENCE
[16]    Y←Y∧X
[17]    ⍝ Y IS THE BORROW
[18]    Y←1↓Y,0
[19]    ⍝ Y IS THE SHIFTED BORROW
[20]    →LA×⍳(+/Y=1)≠0
[21]    ⍝ BRANCH TO LA IF BORROW IS NOT ALL ZEROS
[22]    '2''S COMPLEMENT BINARY DIFFERENCE:'
[23]    □←DIFFERENCE←X
[24]    'DIFFERENCE:'
[25]    □←-((2×DIFFERENCE[1])-1)×(DIFFERENCE[1]+(32⍴
        2)⊥((32⍴DIFFERENCE[1]≠DIFFERENCE))
    ∇
```

Example 1 of Mixed Mode Subtraction

```
        MSUB

MINUEND:
□:
        1429
2'S COMPLEMENT BINARY MINUEND:
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  1  0  1  1  0  0  1  0  1
        0  1

SUBTRAHEND:
□:
        -45532
2'S COMPLEMENT BINARY SUBTRAHEND:
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
        0  1  0  0  1  1  0  1  1  1  1  0  1  0
        0  0

2'S COMPLEMENT BINARY DIFFERENCE:
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  1
        0  1  1  1  1  0  1  0  1  1  0  1
DIFFERENCE:
47021
```

Example 2 of Mixed Mode Subtraction

```
      MSUB

MINUEND:
□:
      336842
2'S COMPLEMENT BINARY MINUEND:
0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1
      0  0  1  0  0  0  1  1  1  1  0  0  1  0
      1  0

SUBTRAHEND:
□:
      15562287
2'S COMPLEMENT BINARY SUBTRAHEND:
0  0  0  0  0  0  0  0  1  1  1  0  1  1  0  1
      0  1  1  1  0  1  1  0  0  0  1  0  1  1
      1  1

2'S COMPLEMENT BINARY DIFFERENCE:
1  1  1  1  1  1  1  1  0  0  0  1  0  1  1  1  1  0  1  0
      1  1  0  1  1  0  0  1  1  0  1  1
DIFFERENCE:
 15225445
```

C. APL Documentation of Mixed Mode Multiplication on STARAN

```
      ∇MMUL[⎕]∇
    ∇ MMUL
[1]   ⍳0
[2]   'MULTIPLICAND:'
[3]   MULTIPLICAND←⎕
[4]   'BINARY MULTIPLICAND:'
[5]   ⎕←MULTIPLICAND←(32⍴2)⊤MULTIPLICAND
[6]   ⍝ BINARY MULTIPLICAND
[7]   ⍳0
[8]   'MULTIPLIER:'
[9]   MULTIPLIER←⎕
[10]  'BINARY MULTIPLIER:'
[11]  ⎕←MULTIPLIER←(32⍴2)⊤MULTIPLIER
[12]  ⍝ BINARY MULTIPLIER
[13]  ⍳0
[14]  ⍝ SET THE INITIAL VALUES OF COUNTER, PRODUCT AND CARR
      Y
[15]  COUNTER←32
[16]  PRODUCT←64⍴0
[17]  CARRY←64⍴0
[18] LOOP:→ZERO×⍳0=MULTIPLIER[COUNTER]
[19]  PRODUCT←PRODUCT≠CARRY≠MULTIPLICAND,32⍴0
[20]  ⍝ PARTIAL PRODUCT
[21]  CARRY←(CARRY∧~PRODUCT)∨((~PRODUCT)∧MULTIPLICAND,
      32⍴0)∨(CARRY∧MULTIPLICAND,32⍴0)
[22]  →ONE
[23] ZERO:CARRY←¯1⌽CARRY
[24] ONE:PRODUCT←¯1⌽PRODUCT
[25]  COUNTER←COUNTER-1
[26]  →LOOP×⍳COUNTER≠0
[27]  ⍝ THIS LOOP ADD THE CARRY TO THE PARTIAL SUM TO GET T
      HE TRUE SUM
[28] ADD:PRODUCT←PRODUCT≠CARRY
[29]  CARRY←CARRY∧~PRODUCT
[30]  CARRY←1↓CARRY,0
[31]  →ADD×⍳(+/CARRY≠0)≠0
[32]  'BINARY PRODUCT:'
[33]  ⎕←PRODUCT
[34]  'PRODUCT:'
[35]  ⎕←(64⍴2)⊥PRODUCT
    ∇
```

Example of Mixed Mode Multiplication

```
      MMUL

MULTIPLICAND:
□:
      47732
BINARY MULTIPLICAND:
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
      1  0  1  1  1  0  1  0  0  1  1  1  0  1
      0  0

MULTIPLIER:
□:
      136
BINARY MULTIPLIER:
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0  0  1  0  0  0  1  0
      0  0

BINARY PRODUCT:
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  0  0  0  0  0  0  0  0  0  0  0  0  0
      0  0  0  0  1  1  0  0  0  1  1  0  0  0  0  1  1
      0  1  1  0  1  0  0  0  0  0
PRODUCT:
6491552
```

D. APL Documentation of Mixed Mode Division on STARAN

```
      ∇MDIV[□]∇
    ∇ MDIV
[1]   ⍳0
[2]   'DIVIDEND:'
[3]   DIVIDEND←□
[4]   'BINARY DIVIDEND:'
[5]   □←DIVIDEND←(32⍴2)⊤DIVIDEND
[6]   ⍳0
[7]   'DIVISOR:'
[8]   DIVISOR←□
[9]   'BINARY DIVISOR:'
[10]  □←DIVISOR←(32⍴2)⊤DIVISOR
[11]  ⍳0
[12]  COUNTER←32
[13]  DIVIDEND1←(32⍴0),DIVIDEND
[14]  DIVIDEND1←1⌽DIVIDEND1
[15] AGAIN:DIVISOR1←DIVISOR,32⍴0
[16]  V←(32⍴DIVIDEND1[1],32⍴0
[17]  ⍝ THE FOLLOWING BLOCK IS THE CO-OPERATION ACTION
[18] COP:DIVIDEND1←DIVIDEND1≠DIVISOR1
[19]  DIVISOR1←DIVISOR1∧V≠DIVIDEND1
[20]  DIVISOR1←1↓DIVISOR1,0
[21]  →COP×⍳(+/DIVISOR1=1)≠0
[22]  DIVIDEND1←1⌽DIVIDEND1
[23]  COUNTER←COUNTER-1
[24]  →AGAIN×⍳COUNTER>0
[25]  DIVIDEND1←¯1⌽DIVIDEND1
[26]  REMAINDER←DIVIDEND1[⍳32]
[27]  UPDATE←DIVISOR∧32⍴REMAINDER[1]
[28]  ⍝ THE FOLLOWING BLOCK ADJUST THE REMAINDER
[29] BB:REMAINDER←REMAINDER≠UPDATE
[30]  UPDATE←UPDATE∧~REMAINDER
[31]  UPDATE←1↓UPDATE,0
[32]  →BB×⍳(+/UPDATE=1)≠0
[33]  QUOTIENT←~(32↓1⌽DIVIDEND1)
[34]  'BINARY QUOTIENT:'
[35]  □←QUOTIENT
[36]  'QUOTIENT:'
[37]  □←(32⍴2)⊥QUOTIENT
[38]  ⍳0
[39]  'BINARY REMAINDER:'
[40]  □←REMAINDER
[41]  'REMAINDER:'
[42]  □←(32⍴2)⊥REMAINDER
    ∇
```

Example of Mixed Mode Division

```
      MDIV

DIVIDEND:
[]:
      4789941
BINARY DIVIDEND:
0  0  0  0  0  0  0  0  0  1  0  0  1  0  0  1
      0  0  0  1  0  1  1  0  1  0  1  1  0  1
      0  1

DIVISOR:
[]:
      3559
BINARY DIVISOR:
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
      0  0  0  0  1  1  0  1  1  1  1  0  0  1
      1  1

BINARY QUOTIENT:
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
      0  1  0  1  0  1  0  0  0  0  0  1
QUOTIENT:
1345

BINARY REMAINDER:
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
      1  1  0  0  0  0  0  0  1  1  1  0
REMAINDER:
3086
```

APPENDIX B EXECUTION TIMES AND PROCESSING RATES OF BIT-SEQUENTIAL OPERATIONS AND MIXED MODE OPERATIONS

The following tables list the execution times and processing rates of the arithmetic operations in bit-sequential operations and in mixed mode operations. These data were measured on the STARAN computer by testing on 256 pairs of random numbers.

	Bit-Sequential Operations (32-bit 2's Complement)		Mixed Mode Operations (32-bit 2's Complement)	
No. of Operations to be processed	Execution Time (μ-sec.)	Processing Rates (K op./sec.)	Execution Time (μ-sec.)	Processing Rates (k op./sec.)
1	31	32	12	83
2	31	65	13	154
2^2	31	129	14	286
2^3	31	258	15	533
2^4	31	516	16	1.000×10^3
2^5	31	1.032×10^3	17	1.882×10^3
2^6	31	2.065×10^3	18	3.556×10^3
2^7	31	4.129×10^3	19	6.737×10^3
2^8	31	8.258×10^3	20	1.280×10^4
2^9	31	1.652×10^4	40	1.280×10^4
2^{10}	31	3.303×10^4	80	1.280×10^4
2^{11}	31	6.606×10^4	160	1.280×10^4
2^{12}	31	1.321×10^5	320	1.280×10^4
2^{13}	31	2.643×10^5	640	1.280×10^4
2^{14}	64	2.643×10^5	1.28×10^3	1.280×10^4

Table B-1 Execution Times and Processing Rates of Addition/Subtraction

	Bit-Sequential Operations (32-bit 2's Complement)		Mixed Mode Operations (32-bit 2's Complement)	
No. of Operations to be processed	Execution Time (μ-sec.)	Processing Rate (op./sec.)	Execution Time (μ-sec.)	Processing Rates (op./sec.)
1	1.28×10^3	7.813×10^2	182	5.495×10^3
2	1.28×10^3	1.563×10^3	206	9.709×10^3
2^2	1.28×10^3	3.125×10^3	225	1.778×10^4
2^3	1.28×10^3	6.250×10^3	236	3.390×10^4
2^4	1.28×10^3	1.250×10^4	245	6.531×10^4
2^5	1.28×10^3	2.500×10^4	254	1.260×10^5
2^6	1.28×10^3	5.000×10^4	263	2.433×10^5
2^7	1.28×10^3	1.000×10^5	272	4.706×10^5
2^8	1.28×10^3	2.000×10^5	544	4.706×10^5
2^9	1.28×10^3	4.000×10^5	1088	4.706×10^5
2^{10}	1.28×10^3	8.000×10^5	2176	4.706×10^5
2^{11}	1.28×10^3	1.600×10^6	4352	4.706×10^5
2^{12}	1.28×10^3	3.200×10^6	8704	4.706×10^5
2^{13}	1.28×10^3	6.400×10^6	1.741×10^4	4.706×10^5
2^{14}	2.56×10^3	6.400×10^6	3.482×10^4	4.706×10^5

Table B-2 Execution Times and Processing Rates of Multiplication

	Bit-Sequential Operations (32-bit 2's Complement)		Mixed Mode Operations (32-bit 2's Complement)	
No. of Operations to be processed	Execution Time (μ-sec.)	Processing Rates (op./sec.)	Execution Time (μ-sec.)	Processing Rate (op./sec.)
1	1222	8.183×10^2	545	1.835×10^3
2	1222	1.637×10^3	643	3.110×10^3
2^2	1222	3.273×10^3	723	5.533×10^3
2^3	1222	6.547×10^3	817	9.792×10^3
2^4	1222	1.309×10^4	897	1.784×10^4
2^5	1222	2.619×10^4	978	3.272×10^4
2^6	1222	5.237×10^4	1.059×10^3	6.043×10^4
2^7	1222	1.047×10^5	1.140×10^3	1.123×10^5
2^8	1222	2.095×10^5	2.280×10^3	1.123×10^5
2^9	1222	4.190×10^5	4.560×10^3	1.123×10^5
2^{10}	1222	8.380×10^5	9.120×10^3	1.123×10^5
2^{11}	1222	1.676×10^6	1.824×10^4	1.123×10^5
2^{12}	1222	3.352×10^6	3.648×10^4	1.123×10^5
2^{13}	1222	6.704×10^6	7.296×10^4	1.123×10^5
2^{14}	2444	6.704×10^6	1.459×10^5	1.123×10^5

Table B-3 Execution Times and Processing Rates of Division

APPENDIX C FUNDAMENTAL OPERATIONS OF A DECIMAL SIGNED DIGIT NUMBER SYSTEM

The fundamental operations depend on the base. For a radix-10, digit values from 7 to -7, signed digit number system, the fundamental operations are defined for elementary operations which can be implemented in the micro-processors. The fundamental operations are described in the following tables:

A. Fundamental Operations for Addition and Subtraction

(a) Operation Ⓐ

This operation generates the partial sum and the carry of two digits of signed digit numbers. The partial sum digits are defined in Table C-1 and the carry digits are defined in Table C-2.

(b) Operation Ⓑ

This operation generates the true sum digits of partial sum and the addition carry, and is defined in Table C-3.

(c) Operation Ⓝ

This operation generates the negative digits of signed digits, and is defined in Table C-4.

A_i \ B_i	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-4	-3	-2	-1	0	1	2	3	4	-5	-4	-3	-2	-1	0
-6	-3	-2	-1	0	1	2	3	4	-5	-4	-3	-2	-1	0	1
-5	-2	-1	0	1	2	3	4	-5	-4	-3	-2	-1	0	1	2
-4	-1	0	1	2	3	4	-5	-4	-3	-2	-1	0	1	2	3
-3	0	1	2	3	4	-5	-4	-3	-2	-1	0	1	2	3	4
-2	1	2	3	4	-5	-4	-3	-2	-1	0	1	2	3	4	5
-1	2	3	4	-5	-4	-3	-2	-1	0	1	2	3	4	5	-4
0	3	4	-5	-4	-3	-2	-1	0	1	2	3	4	5	-4	-3
1	4	-5	-4	-3	-2	-1	0	1	2	3	4	5	-4	-3	-2
2	-5	-4	-3	-2	-1	0	1	2	3	4	5	-4	-3	-2	-1
3	-4	-3	-2	-1	0	1	2	3	4	5	-4	-3	-2	-1	0
4	-3	-2	-1	0	1	2	3	4	5	-4	-3	-2	-1	0	1
5	-2	-1	0	1	2	3	4	5	-4	-3	-2	-1	0	1	2
6	-1	0	1	2	3	4	5	-4	-3	-2	-1	0	1	2	3
7	0	1	2	3	4	5	-4	-3	-2	-1	0	1	2	3	4

S_i:

$$A_i \textcircled{A} B_i = S_i + r\, C_{i+1}$$

Table C-1 Partial Sum of Operation Ⓐ for a Decimal Signed Digit Number System

A_i \ B_i	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0
-6	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	0
-5	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	0	0
-4	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	0	0	0
-3	-1	-1	-1	-1	-1	0	0	0	0	0	0	0	0	0	0
-2	-1	-1	-1	-1	0	0	0	0	0	0	0	0	0	0	0
-1	-1	-1	-1	0	0	0	0	0	0	0	0	0	0	0	1
0	-1	-1	0	0	0	0	0	0	0	0	0	0	0	1	1
1	-1	0	0	0	0	0	0	0	0	0	0	0	1	1	1
2	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
3	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
4	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
5	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
6	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
7	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

C_{i+1}:

$$A_i \textcircled{A} B_i = S_i + r\, C_{i+1}$$

Table C-2 Carry Digits of Operation Ⓐ for a Decimal Signed Digit Number System

S_i:

S_i \ C_i	-1	0	1
-6	-7	-6	-5
-5	-6	-5	-4
-4	-5	-4	-3
-3	-4	-3	-2
-2	-3	-2	-1
-1	-2	-1	0
0	-1	0	1
1	0	1	2
2	1	2	3
3	2	3	4
4	3	4	5
5	4	5	6
6	5	6	7

$S_i = S_i + C_i$

Table C-3 Definition of Operation Ⓑ for a Decimal Signed Digit Number System

A_i	B_i
-7	7
-6	6
-5	5
-4	4
-3	3
-2	2
-1	1
0	0
1	-1
2	-2
3	-3
4	-4
5	-5
6	-6
7	-7

$$\textcircled{N} A_i = -A_i = B_i$$

Table C-4 Definition of Operation Ⓝ for a decimal Signed Digit Number System

B. Fundamental Operations for Multiplication

In addition to the fundamental operations for addition and subtraction, the multiplication needs the fundamental operation Ⓜ to generate the product of two digits. The fundamental operation Ⓜ generates the partial product digit and the multiplication carry digit of two signed digits. The true product digits are the sum of the partial product digits and multiplication carry digits shifted one position. The partial product digits are defined in Table C-5, and the multiplication carry digits are defined in Table C-6.

A_i \ B_j	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-1	2	5	-2	1	4	-3	0	3	-4	-1	2	-5	-2	1
-6	2	-4	0	4	-2	2	-4	0	4	-2	2	-4	0	4	-2
-5	5	0	5	0	5	0	5	0	-5	0	-5	0	-5	0	-5
-4	-2	4	0	-4	2	-2	4	0	-4	2	-2	4	0	-4	2
-3	1	-2	5	2	-1	-4	3	0	-3	4	1	-2	-5	2	-1
-2	4	2	0	-2	-4	4	2	0	-2	-4	4	2	0	-2	-4
-1	-3	-4	5	4	3	2	1	0	-1	-2	-3	-4	-5	4	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	3	4	-5	-4	-3	-2	-1	0	1	2	3	4	5	-4	-3
2	-4	-2	0	2	4	-4	-2	0	2	4	-4	-2	0	2	4
3	-1	2	-5	-2	1	4	-3	0	3	-4	-1	2	5	-2	1
4	2	-4	0	4	-2	2	-4	0	4	-2	2	-4	0	4	-2
5	-5	0	-5	0	-5	0	-5	0	5	0	5	0	5	0	5
6	-2	4	0	-4	2	-2	4	0	-4	2	-2	4	0	-4	2
7	1	-2	-5	2	-1	-4	3	0	-3	4	1	-2	5	2	-1

P_{i+j-1}:

$$A_i \textcircled{M} B_j = r\,MC_{i+j} + P_{i+j-1}$$

Table C-5 Partial Product Digits of Operation Ⓜ for a Decimal Signed Digit Number System

A_i \ B_j	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	5	4	3	3	2	1	1	0	-1	-1	-2	-3	-3	-4	-5
-6	4	4	3	2	2	1	1	0	-1	-1	-2	-2	-3	-4	-4
-5	3	3	2	2	1	1	0	0	0	-1	-1	-2	-2	-3	-3
-4	3	2	2	2	1	1	0	0	0	-1	-1	-2	-2	-2	-3
-3	2	2	1	1	1	1	0	0	0	-1	-1	-1	-1	-2	-2
-2	1	1	1	1	1	0	0	0	0	0	-1	-1	-1	-1	-1
-1	1	1	0	0	0	0	0	0	0	0	0	0	0	-1	-1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	-1	-1	0	0	0	0	0	0	0	0	0	0	0	1	1
2	-1	-1	-1	-1	-1	0	0	0	0	0	1	1	1	1	1
3	-2	-2	-1	-1	-1	-1	0	0	0	1	1	1	1	2	2
4	-3	-2	-2	-2	-1	-1	0	0	0	1	1	2	2	2	3
5	-3	-3	-2	-2	-1	-1	0	0	0	1	1	2	2	3	3
6	-4	-4	-3	-2	-2	-1	-1	0	1	1	2	2	3	4	4
7	-5	-4	-3	-2	-2	-1	-1	0	1	1	2	3	3	4	5

MC_{i+j}:

$$A_i \textcircled{M} B_j = r\, MC_{i+j} + P_{i+j-1}$$

Table C-6 Multiplication Carry Digits of Operation Ⓜ for a Decimal Signed Digit Number System

C. Fundamental Operations for Division

Table C-7 to C-16, these ten tables define all the operations which are used in the division algorithm. The logic functions of these fundamental operations are shown in the following tables.

X_i	Y_i
-7	-1
-6	-1
-5	-1
-4	-1
-3	-1
-2	-1
-1	-1
0	0
1	1
2	1
3	1
4	1
5	1
6	1
7	1

$Y_i = DTC\ X_i$

Table C-7 Definition of Operation DTC for a Decimal Signed Digit Number System

X_i:	Z_j \ W_i	1	0	-1
	-1	-1	-1	-1
	0	1	0	-1
	1	1	1	1

$X_i = Z_j$ CMB W_j

Table C-8 Definition of Operation CMB for a Decimal Signed Digit Number System

X_i:	Y_i	
	-1	1
	0	0
	1	0

X_i = DNEG Y_i

Table C-9 Definition of Operation DNEG for a Decimal Signed Digit Number System

X_i:	Y_i	
	-1	1
	0	0
	1	1

X_i = DNZR Y_i

Table C-10 Definition of Operation DNZR for a Decimal Signed Digit Number System

X_i:

Y_i \ Z_i	-1	0	1
-1	1	0	-1
0	0	0	0
1	-1	0	1

$X_i = Y_i$ CMPRA Z_i

Table C-11 Definition of Operation CMPRA for a Decimal Signed Digit Number System

X_i:

Y_i \ Z_i	-1	0	1
-1	0	1	1
0	1	0	0
1	1	0	0

$X_i = Y_i$ CMPRB Z_i

Table C-12 Definition of Operation CMPRB for a Decimal Signed Digit Number System

X_i:

Y_i	
0	1
1	-1

X_i = TRFM Y_i

Table C-13 Definition of Operation TRFM for a Decimal Signed Digit Number System

X_i:

Y_i \ Z_i	-1	0	1
-7	7	0	-7
-6	6	0	-6
-5	5	0	-5
-4	4	0	-4
-3	3	0	-3
-2	2	0	-2
-1	1	0	-1
0	0	0	0
1	-1	0	1
2	-2	0	2
3	-3	0	3
4	-4	0	4
5	-5	0	5
6	-6	0	6
7	-7	0	7

$X_i = Y_i$ MULPP Z_i

Table C-14 Definition of Operation MULPP for a Decimal Signed Digit Number System

Z_j \ Y_i	0	1
-7	0	-7
-6	0	-6
-5	0	-5
-4	0	-4
-3	0	-3
-2	0	-2
-1	0	-1
0	0	0
1	0	1
2	0	2
3	0	3
4	0	4
5	0	5
6	0	6
7	0	7

X_j:

$X_i = Y_i$ SELECT Z_i

Table C-15 Definition of Operation SELECT for a Decimal Signed Digit Number System

X_i: Z_i \ Y_i	0	1
-7	-7	0
-6	-6	0
-5	-5	0
-4	-4	0
-3	-3	0
-2	-2	0
-1	-1	0
0	0	0
1	1	0
2	2	0
3	3	0
4	4	0
5	5	0
6	6	0
7	7	0

$X_i = Y_i$ SELECTZR Z_i

Table C-16 Definition of Operation SELECTZR for a Decimal Signed Digit Number System

APPENDIX D APL DOCUMENTATION OF THE DECIMAL SIGNED DIGIT ARITHMETIC OPERATIONS

The signed digit arithmetic operations are documented by APL functions. Each word is assumed of length eight digits. The documentation programs are listed. Every operation shown in the APL documentation corresponds to one machine instruction in the proposed STARAN-like system.

A. APL Documentation of Signed Digit Addition

```
      ∇SDADD[⎕]∇
    ∇ SDADD
[1]   ⍳0
[2]   'ENTER A EIGHT DIGITS SIGNED DIGIT ADDEND:'
[3]   X←ADDEND←⎕
[4]   ⍳0
[5]   'ENTER A EIGHT DIGITS SIGNED DIGIT ADDER:'
[6]   Y←ADDER←⎕
[7]   ⍳0
[8]   Z←X ADD1 Y
[9]   ⍝ Z IS THE PARTIAL SUM OF X AND Y
[10]  Y←X ADD2 Y
[11]  ⍝ Y IS THE CARRY OF X AND Y
[12]  Y←1↓Y,0
[13]  ⍝ SHIFT THE CARRY
[14]  X←Y ADDC Z
[15]  ⍝ X IS THE TRUE SUM
[16]  'SUM:'
[17]  ⎕←SUM←X
    ∇
```

Operation Ⓐ

```
      ∇ADD1[⎕]∇
    ∇ Z←X ADD1 Y
[1]   ⍝ FUNDAMENTAL OPERATION TO GENERATE THE PARTIAL SUM
[2]   ⍝ OF X AND Y
[3]   Z←(X+Y)-((((X+Y)>5)-((X+Y)<¯5))×10)
    ∇


      ∇ADD2[⎕]∇
    ∇ W←X ADD2 Y
[1]   ⍝ FUNDAMENTAL OPERATION TO GENERATE THE CARRY DIGIT
[2]   ⍝ OF X AND Y
[3]   W←((X+Y)>5)-((X+Y)<¯5)
    ∇
```

Operation Ⓑ

```
      ∇ADDC[⎕]∇
    ∇ Z←X ADDC Y
[1]   ⍝ FUNCTIONAL DESCRIPTION OF FUNDAMENTAL OPERATION
[2]   ⍝ OUTPUT IS THE SUM OF TWO INPUTS
[3]   Z←X+Y
    ∇
```

Example 1 of Signed Digit Addition

```
        SDADD

ENTER A EIGHT DIGITS SIGNED DIGIT ADDEND:
[]:
        0 2 4 7 3 ¯5 ¯4 ¯7

ENTER A EIGHT DIGITS SIGNED DIGIT ADDER:
[]:
        2 ¯7 7 ¯6 ¯1 ¯4 ¯2 5

SUM:
2  ¯4  1  1  1  0  4  ¯2
```

Example 2 of Signed Digit Addition

```
        SDADD

ENTER A EIGHT DIGITS SIGNED DIGIT ADDEND:
[]:
        0 2 ¯5 ¯7 4 7 0 1

ENTER A EIGHT DIGITS SIGNED DIGIT ADDER:
[]:
        0 ¯7 ¯5 ¯1 4 2 2 6

SUM:
0  ¯6  ¯1  3  ¯1  ¯1  3  ¯3
```

B. APL Documentation of Signed Digit Subtraction

```
        ∇SDSUB[□]∇
     ∇ SDSUB
[1]    ⍳0
[2]    'ENTER A EIGHT DIGITS SIGNED DIGIT SUBTRAHEND:'
[3]    X←SUBTRAHEND←□
[4]    ⍳0
[5]    'ENTER A EIGHT DIGITS SIGNED DIGIT MINUEND:'
[6]    Y←NEGATION MINUEND←□
[7]    ⍝ GET THE NEGATIVE VALUE OF MINUEND
[8]    ⍳0
[9]    ⍝ THE FOLLOWING IS THE SIGNED DIGIT ADDITION
[10]   Z←X ADD1 Y
[11]   Y←X ADD2 Y
[12]   Y←1↓Y,0
[13]   X←Y ADDC Z
[14]   'DIFFERENCE:'
[15]   □←DIFF←X
     ∇
```

Operation Ⓐ

```
        ∇ADD1[□]∇
     ∇ Z←X ADD1 Y
[1]    ⍝ FUNDAMENTAL OPERATION TO GENERATE THE PARTIAL SU
[2]    ⍝ OF X AND Y
[3]    Z←(X+Y)-((((X+Y)>5)-((X+Y)<¯5))×10)
     ∇


        ∇ADD2[□]∇
     ∇ W←X ADD2 Y
[1]    ⍝ FUNDAMENTAL OPERATION TO GENERATE THE CARRY DIGIT
[2]    ⍝ OF X AND Y
[3]    W←((X+Y)>5)-((X+Y)<¯5)
     ∇
```

Operation Ⓐ

```
      ∇ADDC[⎕]∇
    ∇ Z←X ADDC Y
[1]   ⍝ FUNCTIONAL DESCRIPTION OF FUNDAMENTAL OPERATION
[2]   ⍝ OUTPUT IS THE SUM OF TWO INPUTS
[3]   Z←X+Y
    ∇
```

Operation Ⓝ

```
      ∇NEGATION[⎕]∇
    ∇ A←NEGATION B;B;A
[1]   ⍝ FUNCTIONAL DESCRIPTION OF FUNDAMENTAL OPERATION
[2]   ⍝ OUTPUT IS THE NEGATIVE VALUE OF INPUT
[3]   A←-B
    ∇
```

Example 1 of Signed Digit Subtraction

```
        SDSUB

 ENTER A EIGHT DIGITS SIGNED DIGIT SUBTRAHEND:
 []:
        2 2 -5 -3 1 -7 4 4

 ENTER A EIGHT DIGITS SIGNED DIGIT MINUEND:
 []:
        7 3 -7 1 1 -4 3 -4

 DIFFERENCE:
 -5  -1  2  -4  0  -3  2  -2
```

Example 2 of Signed Digit Subtraction

```
        SDSUB

 ENTER A EIGHT DIGITS SIGNED DIGIT SUBTRAHEND:
 []:
        2 4 -7 3 2 2 -1 5

 ENTER A EIGHT DIGITS SIGNED DIGIT MINUEND:
 []:
        -1 3 3 5 4 -3 -7 3

 DIFFERENCE:
 3  0  0  -2  -2  6  -4  2
```

C. APL Documentation of Signed Digit Multiplication

```
      ∇SDMUL[⎕]∇
    ∇ SDMUL
[1]   ⍳0
[2]   'ENTER A EIGHT DIGITS SIGNED DIGIT MULTIPLICAND:'
[3]   MULND←⎕
[4]   ⍳0
[5]   'ENTER A EIGHT DIGITS SIGNED DIGIT MULTIPLIER:'
[6]   MULER←⎕
[7]   ⍳0
[8]   I←8
[9]   X←16⍴0
[10] LA:X←(8⍴0),MULND
[11]  ⍝ Y IS THE MULTIPLICAND
[12]  Z←(8⍴0),8⍴MULER[I]
[13]  ⍝Z IS THE PROPAGATED ONE DIGIT MULTIPLIER
[14]  W←Y SDMP1 Z
[15]  ⍝ W IS THE PARTIAL PRODUCT
[16]  Z←Y SDMP2 Z
[17]  ⍝Z IS THE MULTIPLICATION CARRY
[18]  Z←1⌽Z
[19]  ⍝ SHIFT THE CARRY
[20]  ⍝ NEXT FOUR STATEMENTS ADD THE PARTIAL PRODUCT
[21]  Y←X ADD1 W
[22]  X←X ADD2 W
[23]  X←1⌽X
[24]  X←X ADDC Y
[25]  ⍝ NEXT FOUR STATEMENTS ADD THE MULTIPLICATION CARRY
[26]  Y←X ADD1 Z
[27]  X←X ADD2 Z
[28]  X←1⌽X
[29]  X←X ADDC Y
[30]  X←¯1⌽X
[31]  I←I-1
[32]  →LA×⍳(I>0)
[33]  X←¯8⌽X
[34]  'PRODUCT:'
[35]  ⎕←PRO←X
    ∇
```

Operation Ⓜ

```
        ∇SDMP1[□]∇
      ∇ Z←X SDMP1 Y;P;P1;P2
[1]     ⍝ FUNDAMENTAL OPERATION TO GENERATE THE PARTIAL
[2]     ⍝ PRODUCT OF X AND Y
[3]     P←X×Y
[4]     P1←(P≥0)×P
[5]     P2←(P<0)×P
[6]     P1←10|P1
[7]     P2←((P<0)×¯10)+10|P2
[8]     P1←((P1>5)×¯10)+P1
[9]     P2←((P2<¯5)×10)+P2
[10]    Z←P1+P2
      ∇
```

```
        ∇SDMP2[□]∇
      ∇ Z←X SDMP2 Y;P;P1;P2
[1]     ⍝ FUNDAMENTAL OPERATION TO GENERATE THE
[2]     ⍝ CARRY OF X AND Y
[3]     P←X×Y
[4]     P1←(P≥0)×P
[5]     P2←(P<0)×P
[6]     P1←10|P1
[7]     P2←((P<0)×¯10)+10|P2
[8]     P1←((P1>5)×¯10)+P1
[9]     P2←((P2<¯5)×10)+P2
[10]    Z←(P-(P1+P2))÷10
      ∇
```

Operation Ⓑ

```
        ∇ADDC[□]∇
      ∇ Z←X ADDC Y
[1]     ⍝ FUNCTIONAL DESCRIPTION OF FUNDAMENTAL OPERATION
[2]     ⍝ OUTPUT IS THE SUM OF TWO INPUTS
[3]     Z←X+Y
      ∇
```

Operation (A)

```
      ∇ADD1[□]∇
     ∇ Z←X ADD1 Y
[1]    ⍝ FUNDAMENTAL OPERATION TO GENERATE THE PARTIAL SUM
[2]    ⍝ OF X AND Y
[3]    Z←(X+Y)-((((X+Y)>5)-((X+Y)<¯5))×10)
     ∇

      ∇ADD2[□]∇
     ∇ W←X ADD2 Y
[1]    ⍝ FUNDAMENTAL OPERATION TO GENERATE THE CARRY DIGIT
[2]    ⍝ OF X AND Y
[3]    W←((X+Y)>5)-((X+Y)<¯5)
     ∇
```

Example 1 of Signed Digit Multiplication

```
      SDMUL

 ENTER A EIGHT DIGITS SIGNED DIGIT MULTIPLICAND:
 □:
      1 -5 3 7 -5 3 2 3

 ENTER A EIGHT DIGITS SIGNED DIGIT MULTIPLIER:
 □:
      0 2 3 -7 -2 0 -1 3

 PRODUCT:
 0  0  1  2  0  -5  4  -1  0  2  1  -1  -3  -3
       4  -1
```

Example 2 of Signed Digit Multiplication

```
      SDMUL

 ENTER A EIGHT DIGITS SIGNED DIGIT MULTIPLICAND:
 □:
      -2 -4 -7 7 4 -3 -1 5

 ENTER A EIGHT DIGITS SIGNED DIGIT MULTIPLIER:
 □:
      2 -7 3 -3 5 7 1 2

 PRODUCT:
 0  -3  -3  3  1  -3  -2  3  -3  -3  2  0  -4
       -2  4  0
```

D. APL Documentation of Signed Digit Division

```
      ∇SDDIV[⎕]∇
    ∇ SDDIV
[1]   ⍳0
[2]   'ENTER A EIGHT DIGITS SIGNED DIGIT DIVIDEND:'
[3]   DIVIDEND←⎕
[4]   ⍳0
[5]   'ENTER A EIGHT DIGITS SIGNED DIGIT DIVISOR:'
[6]   DIVISOR←⎕
[7]   ⍳0
[8]   X←(8⍴0),DIVIDEND
[9]   X←1⍱X
[10]  DVEND←X
[11]  Y←DIVISOR,8⍴0
[12]  DVSOR←Y
[13]  QUOT←16⍴0
[14]  ⍝ SET INITIAL VALUE OF QUOTIENT TO ZERO
[15]  COUNTER←8
[16]  ⍝ THE FOLLOWING DETERMINE TO ADD OR TO SUBTRACT
[17]  ⍝ THE DIVISOR TO OR FROM DIVIDEND
[18] RPT:V7←16⍴1
[19]  DVEND1←16⍴0
[20]  Z←DVEND
[21]  X←DETNZR Z
[22]  ⍝FUNDAMENTAL OPERATION IN MICRO-PROCESSORS
[23]  →ZR×⍳(+/X=0)=16
[24]  ⍝ BRANCH OUT IF THE DIVIDEND IS ZERO
[25]  X←DETECTION Z
[26]  ⍝ THIS SUBROUTINE USES RESPONSE STORE REGISTER W
[27]  Z←DVSOR
[28]  Y←DETECTION Z
[29]  ⍝ THIS SUBROUTINE USES RESPONSE STORE REGISTER W
[30]  X←X CMPRA Y
[31]  X←X×1,15⍴0
[32]  V3←X
[33]  Y←QUOT
[34]  Y←Y ADDC X
[35]  QUOT←Y
[36]  Z←(8⍴X[1]),8⍴0
[37]  Z←NEGATION Z
[38]  X←DVSOR
[39]  X←X MULPP Z
[40]  DVSOR1←X
```

```
[41]   ⍝ ADD OR SUBTRACT THE DIVISOR TO OR FROM DIVIDEND
[42]   ⍝ IN THE FOLLOWING STATEMENTS UNTIL THE SIGN OF
[43]   ⍝ DIVIDEND IS CHANGED
[44] LP:Y←V7
[45]   X←DVSOR1
[46]   X←Y SELECT X
[47]   DVSOR1←X
[48]   Z←DVEND
[49]   X←Y SELECTZR Z
[50]   DVEND2←X
[51]   Z←DVEND1
[52]   X←Y SELECTZR Z
[53]   DVEND3←X
[54]   Z←DVEND
[55]   X←Y SELECT Z
[56]   DVEND←X
[57]   DVEND1←DVEND ADDITION DVSOR1
[58]   DVEND1←DVEND ADDITION DVEND3
[59]   DVEND←DVEND ADDITION DVEND2
[60]   X←DVEND
[61]   Y←DETECTION X
[62]   ⍝ THIS SUBROUTINE USES RESPONSE STORE REGISTER W
[63]   V4←Y
[64]   Z←DVEND1
[65]   X←DETECTION Z
[66]   ⍝ THIS SUBROUTINE USES RESPONSE STORE REGISTER W
[67]   V5←X
[68]   X←Y CMPRB X
[69]   Y←FRFM X
[70]   Y←Y×1,15o0
[71]   ⍝ MASKING OPERATION
[72]   V6A←Y
[73]   X←X×1,15o0
[74]   ⍝ MASKING OPERATION
[75]   V6←X
[76]   →NLP×⍳(+/X=1)=1
[77]   ⍝ BRANCH IF THE SIGNED OF DIVIDEND CHANGED
[78]   Y←(8oY[1]),8o0
[79]   V7←Y
[80]   X←DVEND1[⍳8],DVEND[8+⍳8]
```

```
[81]  DVEND←X
[82]  I←16∘0
[83]  DVEND1←16∘0
[84]  Z←V3
[85]  W←QUOT
[86]  W←W ADDC Z
[87]  QUOT←W
[88]  →LP
[89]  ⍝ FOLLOWING STATEMENTS ADJUST THE DIVIDEND IF THE
[90]  ⍝ DIVISOR IS OVER ADDED OR SUBTRACTED
[91] NLP:X←V4
[92]  X←(8∘X[1]),8∘0
[93]  Y←V5
[94]  Y←(8∘Y[1]),8∘0
[95]  Z←DVEND
[96]  Z←Z MULPP X
[97]  Z←Z×(8∘1),8∘0
[98]  ⍝ MASKING OPERATION
[99]  DVEND2←Z
[100] W←DVEND1
[101] W←W MULPP Y
[102] W←W×(8∘1),8∘0
[103] W←NEGATION W
[104] DVEND3←W
[105] DIFF←DVEND2 ADDITION DVEND3
[106] ⍝ THIS SUBROUTINE USES THE RESPONSE STORE
[107] ⍝ REGISTER X, Y AND Z
[108] Z←DIFF
[109] Y←DETNEG Z
[110] V3←Y
[111] X←V3
[112] Y←X MULPP Y
[113] Y←NEGATION Y
[114] X←QUOT
[115] X←X ADDC Y
[116] QUOT←X
[117] X←V3
[118] X←(8∘X[1]),8∘0
[119] V3←X
[120] Y←DVSOR1
```

```
[121] Y←X SELECT Y
[122] Y←NEGATION Y
[123] DVSOR1A←Y
[124] X←DVEND1[⍳8],DVEND[8+⍳8]
[125] DVEND1←X
[126] DVEND←DVEND1 ADDITION DVSOR1A
[127] ⍝ THIS SUBROUTINE USES RESPONSE STORE REGISTER
[128] ⍝ X, Y AND Z
[129]ZR:COUNTER←COUNTER-1
[130] X←1⌽DVEND
[131] DVEND←X
[132] Y←1⌽QUOT
[133] →END×⍳COUNTER=0
[134] →RPT
[135]END:'QUOTIENT:'
[136] ⎕←QUOT[8+⍳8]
[137] ⍳0
[138] DVEND←¯1⌽DVEND
[139] 'REMAINDER:'
[140] ⎕←REMAINDER←DVEND[⍳8]
      ∇
```

Subroutine DETECTION

```
      ∇DETECTION[⎕]∇
    ∇ Z←DETECTION X
[1]   CNTR←8
[2]   Z←16⍴0
[3]   W←DTC X
[4]  A:Z←Z CMB W
[5]   W←1⌽W
[6]   CNTR←CNTR-1
[7]   →A×⍳(CNTR>0)
[8]   Z←Z×(1,15⍴0)
    ∇
```

Subroutine DETNEG

```
      ∇DETNEG[⎕]∇
    ∇ X←DETNEG Y
[1]   X←DETECTION Y
[2]   X←DNEG X
    ∇
```

Subroutine DETNZR

```
      ∇DETNZR[⎕]∇
    ∇ Z←DETNZR X
[1]   Z←DETECTION X
[2]   Z←DNZR Z
    ∇
```

Subroutine ADDITION

```
      ∇ADDITION[⎕]∇
    ∇ SUM←A ADDITION B
[1]   X←A
[2]   Y←B
[3]   Z←X ADD1 Y
[4]   Y←X ADD2 Y
[5]   Y←1↓Y,0
[6]   X←Y ADDC Z
[7]   SUM←X
    ∇
```

Operation TRFM

```
      ∇TRFM[□]∇
    ∇ Z←TRFM X
[1]   Z←(X=0)-(X=1)
    ∇
```

Operation DTC

```
      ∇DTC[□]∇
    ∇ Y←DTC X
[1]   Y←(X>0)-(X<0)
[2]   Y←Y×(80 1),80 0
    ∇
```

Operation CMB

```
      ∇CMB[□]∇
    ∇ X←Z CMB W
[1]   X←((Z=1)-(Z=¯1))+(Z=0)×W
    ∇
```

Operation DNEG

```
      ∇DNEG[□]∇
    ∇ Y←DNEG X
[1]   Y←X=¯1
    ∇
```

Operation DNZR

```
      ∇DNZR[□]∇
    ∇ Y←DNZR X
[1]   Y←(X=1)∨X=¯1
    ∇
```

Operation CMPRA

```
      ∇CMPRA[⎕]∇
    ∇ Z←X CMPRA Y
[1]   Z←((X×Y)>0)-(X×Y)<0
    ∇
```

Operation CMPRB

```
      ∇CMPRB[⎕]∇
    ∇ Z←X CMPRB Y
[1]   Z←((X=¯1)∧(Y≥0))∨(X≥0)∧Y=¯1
    ∇
```

Operation Ⓐ

```
      ∇ADD1[⎕]∇
    ∇ Z←X ADD1 Y
[1]   ⍝ FUNDAMENTAL OPERATION TO GENERATE THE PARTIAL SUM
[2]   ⍝ OF X AND Y
[3]   Z←(X+Y)-((((X+Y)>5)- (X+Y)<¯5))×10)
    ∇
```

```
      ∇ADD2[⎕]∇
    ∇ W←X ADD2 Y
[1]   ⍝ FUNDAMENTAL OPERATION TO GENERATE THE CARRY DIGIT
[2]   ⍝ OF X AND Y
[3]   W←((X+Y)>5)-((X+Y)<¯5)
    ∇
```

Operation Ⓑ

```
      ∇ADDC[⎕]∇
    ∇ Z←X ADDC Y
[1]   ⍝ FUNCTIONAL DESCRIPTION OF FUNDAMENTAL OPERATION
[2]   ⍝ OUTPUT IS THE SUM OF TWO INPUTS
[3]   Z←X+Y
    ∇
```

Operation MULPP

```
      ∇MULPP[⎕]∇
    ∇ Z←X MULPP Y
[1]   Z←X×Y
    ∇
```

Operation SELECT

```
      ∇SELECT[⎕]∇
    ∇ Z←X SELECT Y
[1]   Z←X×Y
    ∇
```

Operation SELECTZR

```
      ∇SELECTZR[⎕]∇
    ∇ Z←X SELECTZR Y
[1]   Z←(~X)×Y
    ∇
```

Example 1 of Signed Digit Division

```
      SDDIV

ENTER A EIGHT DIGITS SIGNED DIGIT DIVIDEND:
□:
      0 7 3 ¯5 ¯3 ¯4 2 1

ENTER A EIGHT DIGITS SIGNED DIGIT DIVISOR:
□:
      0 0 0 0 2 ¯5 2 5

QUOTIENT:
0  0  0  0  5  ¯2  ¯5  2

REMAINDER:
0  0  0  0  0  ¯2  2  1
```

Example 2 of Signed Digit Division

```
      SDDIV

ENTER A EIGHT DIGITS SIGNED DIGIT DIVIDEND:
□:
      ¯2 ¯7 ¯1 4 7 6 ¯1 2

ENTER A EIGHT DIGITS SIGNED DIGIT DIVISOR:
□:
      0 0 4 3 ¯1 ¯6 ¯3 5

QUOTIENT:
0  0  0  0  0  ¯1  4  ¯3

REMAINDER:
0  0  ¯1  3  5  2  2  ¯3
```

APPENDIX E EXECUTION TIMES AND PROCESSING RATES OF THE MIXED MODE OPERATIONS AND A DECIMAL SIGNED DIGIT NUMBER ARITHMETIC OPERATIONS

The execution times of the decimal signed digit arithmetic for eight-digit number system were estimated on the modified STARAN system by assuming that the fundamental operation speed in the connected micro-processors is about the same as STARAN's memory store cycle time. These operation speeds are compared with the mixed mode operations in Chapter 2 and are listed in Talbe E-1, E-2 and E-3.

No. of Operations to be processed	Mixed Mode Operations (2's Complement)		Signed Digit Operations (Decimal Signed Digit)	
	Execution Time (μ-sec.)	Processing Rate (K op./sec.)	Execution Time (μ-sec.)	Processing Rate (K op./sec.)
1	12	83	1	1.00×10^3
2	13	154	1	2.00×10^3
2^2	14	285	1	4.00×10^3
2^3	15	533	1	8.00×10^3
2^4	16	1.00×10^3	1	1.60×10^4
2^5	17	1.88×10^3	1	3.20×10^4
2^6	18	3.56×10^3	1	6.40×10^4
2^7	19	6.74×10^3	1	1.28×10^5
2^8	20	1.28×10^4	1	2.56×10^5
2^9	40	1.28×10^4	2	2.56×10^5
2^{10}	80	1.28×10^4	4	2.56×10^5
2^{11}	160	1.28×10^4	8	2.56×10^5
2^{12}	320	1.28×10^4	16	2.56×10^5
2^{13}	640	1.28×10^4	32	2.56×10^5

Table E-1 Execution Times and Processing Rates for Addition/ Subtraction

No. of Operations to be processed	Mixed Mode Operations (2's Complement)		Signed Digit Operations (Decimal Signed Digit)	
	Execution Time (μ-sec.)	Processing Rate (op./sec.)	Execution Time (μ-sec.)	Processing Rate (op./sec.)
1	182	5.50×10^3	23	4.35×10^4
2	206	9.71×10^3	23	8.70×10^4
2^2	225	1.78×10^4	23	1.74×10^5
2^3	236	3.39×10^4	23	3.48×10^5
2^4	245	6.53×10^4	23	6.96×10^5
2^5	254	1.26×10^5	23	1.39×10^6
2^6	263	2.43×10^5	23	2.78×10^6
2^7	272	4.71×10^5	23	5.57×10^6
2^8	544	4.71×10^5	46	5.57×10^6
2^9	1088	4.71×10^5	92	5.57×10^6
2^{10}	2176	4.71×10^5	184	5.57×10^6
2^{11}	4352	4.71×10^5	368	5.57×10^6
2^{12}	8704	4.71×10^5	736	5.57×10^6
2^{13}	17408	4.71×10^5	1472	5.57×10^6

Table E-2 Execution Time and Processing Rates for Multiplication

	Mixed Mode Operations (2's Complement)		Signed Digit Operations (Decimal Signed Digit)	
No. of Operations to be processed	Execution Time (μ-sec.)	Processing Rate (op./sec.)	Execution Time (μ-sec.)	Processing Rate (op./sec.)
1	545	1.84×10^3	380	2.63×10^3
2	643	3.11×10^3	384	5.21×10^3
2^2	723	5.53×10^3	387	1.03×10^4
2^3	817	9.79×10^3	390	2.05×10^4
2^4	897	1.78×10^4	394	4.06×10^4
2^5	978	3.27×10^4	397	8.06×10^4
2^6	1059	6.04×10^4	401	1.59×10^5
2^7	1140	1.12×10^5	406	3.15×10^5
2^8	2280	1.12×10^5	812	3.15×10^5
2^9	4560	1.12×10^5	1624	3.15×10^5
2^{10}	9120	1.12×10^5	3248	3.15×10^5
2^{11}	18240	1.12×10^5	6496	3.15×10^5
2^{12}	36480	1.12×10^5	12992	3.15×10^5
2^{13}	72960	1.12×10^5	25984	3.15×10^5

Table E-3 Execution Times and Processing Rates for Division

BIBLIOGRAPHY

[1] A. Weinberger and J. L. Smith,
"The Logical Design of a One-Microsecond Adder Using One-Megacycle Circuitry", IRE Transactions on Electronic Computers, June 1956.

[2] O. L. MacSorley,
"High-Speed Arithmetic in Binary Computers", Proceeding of the IEEE, January 1961.

[3] A. Avizienis,
"Signed-Digit Number Representations for Fast Parallel Arithmetic", IEEE Transactions on Electronic Computers, September 1961.

[4] I. Flores,
The Logic of Computer Arithmetics, Prentice-Hill, 1963.

[5] C. S. Wallace,
"A Suggesting for a Fast Multiplier", IEEE Transactions on Electronic Computers, September 1961.

[6] J. C. Hoffmann, B. Lacaze and P. Csillag,
"Iterative Logical Network for Parallel Multiplication", Electronics Letters, 3rd May 1968, Vol. 4, No. 9.

[7] K. J. Dean,
"Logical Circuits for Use in Iterative Arrays", Electronics Letters, 8th March 1968, Vol. 4, No. 5.

[8] K. J. Dean,
"Binary Division Using a Data-Dependent Iterative Array", Electronics Letters, 12th July 1968, Vol. 4, No. 14.

[9] K. J. Dean,
"Versatile Multiplier Arrays", Electronics Letters, 9th August 1968, Vo. 4, No. 16.

[10] K. J. Dean,
"Design for a full Multiplier", Proceeding of IEE, November 1968, Vol. 115, No. 11.

[11] K. J. Dean,
"Cellular Logical Array for Obtaining the Square of a Binary Number", Electronics Letters, 7th August 1969, Vol. 5, No. 16.

[12] H. H. Guild,
"Fully Iterative Fast Array for Binary Multiplication and Addition", Electronics Letters, 12th June 1969, Vol. 5, No. 12.

[13] R. de Mori,
"Suggestion for an I.C. Fast Parallel Multiplier", Electronics Letters, 6th February 1969, Vol. 5, No. 3.

[14] J. C. Majithia and R. Kital,
"An Iterative Array for Multiplication of Signed Binary Numbers" IEEE Transactions on Computers, February 1971.

[15] P. H. Enslow, Jr.,
Multiprocessors and Parallel Processing, John Wiley & Sons, 1974.

[16] STARAN S Reference Manual, Goodyear Aerospace Co., August 1973.

[17] STARAN User's Guide, Goodyear Aerospace Co., August 1973.

[18] STARAN S APPLE Programming Manual, Goodyear Aerospace Co., August 1973.

[19] STARAN S MACRO Programming Manual, Goodyear Aerospace Co., August 1973.

[20] STARAN CIOU Reference Manual, Goodyear Aerospace Co., August 1973.

[21] STARAN/HIS 645 User's Guide, Goodyear Aerospace Co., August 1973.

[22] E. P. Stabler and J. Hsu,
Mixed Mode Arithmetic Macros for STARAN, Department of Electrical and Computer Engineering, Syracuse University, Syracuse, New York, December 1974.

[23] D. E. Knuth,
The Art of Computer Programming, Vol. 2 / Seminumerical Algorithms, Addison-Wesley, 1969.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

