DEP

DECISION PROCESSOR

CHAPTER 6

6.0 INTRODUCTION

The DEP executes comparison and modification instructions involving two 128 word register files within the DEP. One of these register files is shared with the MAP and is used by the MAP for indexed addressing. This allows the DEP, through a logical decision process, to cause the MAP to access data from arrays stored in DATA MEMORY in a very efficient manner.

6.1 ORGANIZATION OF THE DEP

Figure 6.1 provides a general block diagram of the DEP. That part of Figure 6.1 which is in black is common to all processors and is described in Section 2.6. The elements in Figure 6.1 which are in red are those which are unique to the DEP.

The DEP is physically located on the same circuit card assembly which contains the Memory Address Processor (MAP). The I Register File shown in Figure 6.1 is shared between the DEP and the MAP in the sense that this file may be accessed by either processor. These operations of MAP and DEP are transparent and fully overlapped. See Chapter 7 for more information on this subject.

In Figure 6.1 a connection is shown in red from the Control Processor (COP) to the Processor Status Enable Register via the DATA MULTIBUS. A similar connection is shown from the COP to the Program Counter through the ADDRESS/CONTROL MULTIBUS. The contents of the Processor Status Enable Register and the Program Counter can be modified by COP program instructions when the AD 10 is in the run mode.

Sections 6.2.3 through 6.2.7 contain diagrams showing the parts of the DEP which are involved with each of the categories of DEP instructions.

6.1.1 Processor Address

The processor address for the DEP is Ø2.

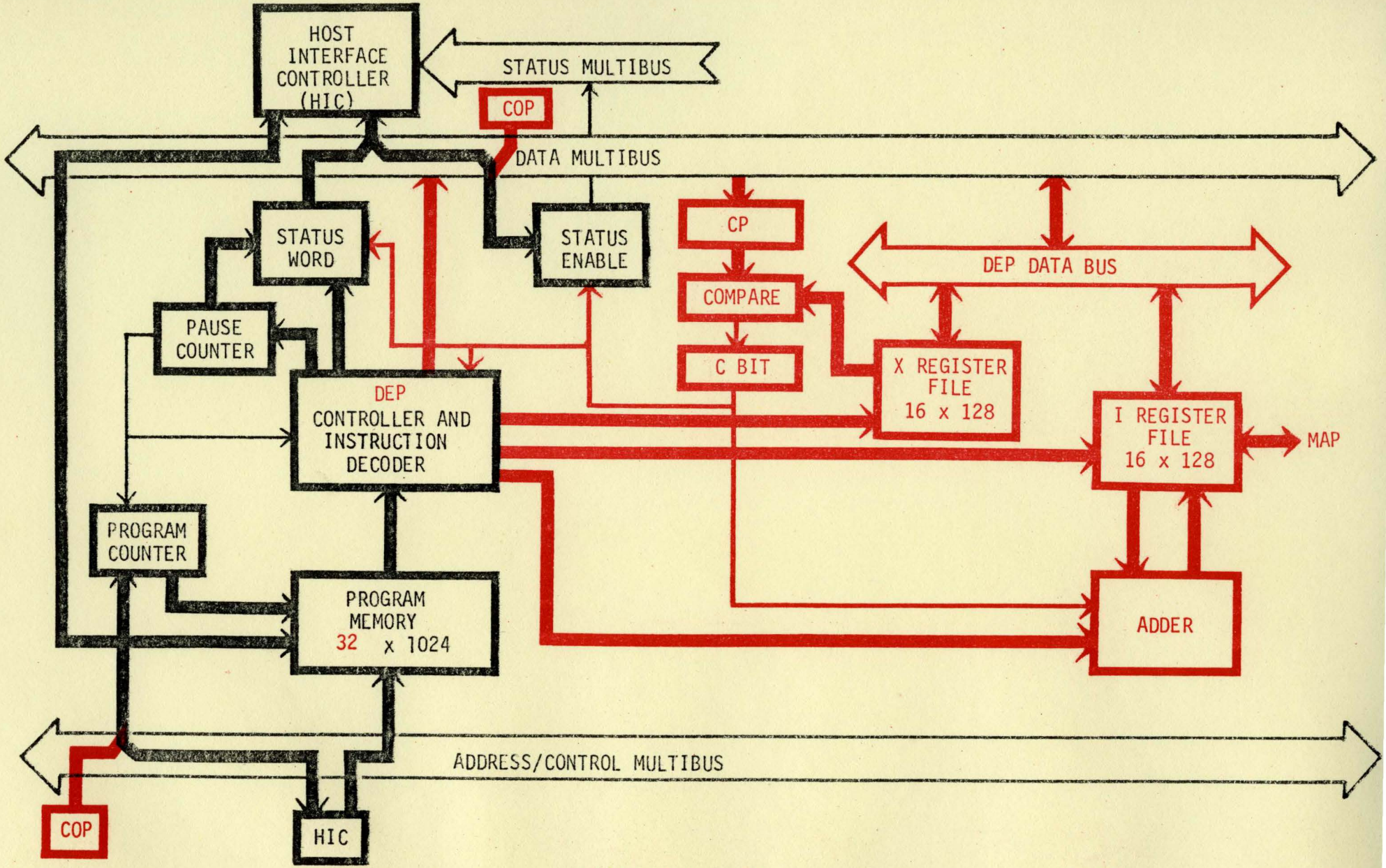
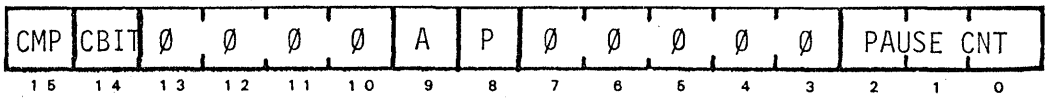


FIGURE 6.1 DEP BLOCK DIAGRAM

6.1.2 Processor Status Word

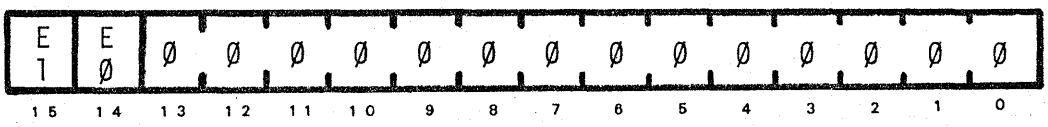


BIT	DESCRIPTION
Ø-2	Remaining Pause Count
3-7	Unassigned
8	DEP is present when set
9	DEP is active when set
1Ø-13	Unassigned
14	Condition Bit indicates the result of the last compare instruction.
15*	If a compare instruction has been performed, this bit is set.

* Once set, this bit remains set until cleared by a READ operation.

The DEP Processor Status Word (PSW) contains information on the current status of the DEP. The PSW is a read-only register.

6.1.3 Processor Status Enable



BIT	DESCRIPTION
Ø-13	Unassigned
14	Enables the condition CBIT←Ø to the AER line.
15	Enables the condition CBIT←1 to the AER line.

The Processor Status Enable Register (PSE) is a write-only register. This register has the same address as the DEP Processor Status Word.

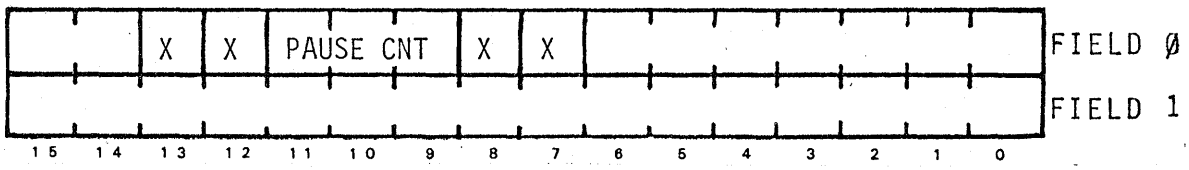
6.1.4 Program Memory

The DEP program memory contains 1,024 words. Each 32-bit instruction word is divided into two 16-bit fields. A DEP instruction is normally loaded as a sequence of two 16-bit words (one per field) from the host processor. However, each field is individually addressable and can be accessed from the host processor for a READ or WRITE operation.

6.2 THE DEP INSTRUCTION SET

6.2.1 DEP Instruction Word Format

The DEP instruction word format is:



The bits labelled "X" in the above diagram are unused. The unused bits cannot be set and are always read as a Ø.

The categories of DEP instructions are:

COMPARE Instructions
LOAD IMMEDIATE DATA Instructions
LOAD CONDITION Instructions
LOAD/STORE REGISTER Instructions
MOVE REGISTER Instructions
PAUSE/NOP Instructions

6.2.2 Microprogramming DEP Instructions

Figure 6.2 provides a summary of the DEP instruction set. This summary has been organized into several groupings of instructions to indicate as clearly as possible the microprogramming possibilities which exist.

A DEP instruction can consist of any of the following:

a.) <A> <PAUSE>
b.) <B> <PAUSE>
c.) <C> <D> <PAUSE>
d.) <C> <E> <PAUSE> (See Note 1.)
e.) <CC><DC><PAUSE>
f.) <CC><EC><PAUSE> (See Note 2.)

PAGE		GROUP	FIELD Ø																FIELD 1															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
6-9	CMM	(A)	1	0	X	X	0	0	0	X	X	←	I	N	D	E	X	→	←	I	N	C	R	E	M	E	N	T	→					
6-10	CMP	(A)	0	1	X	X	0	0	0	X	X	←	I	N	D	E	X	→	←	I	N	C	R	E	M	E	N	T	→					
6-13	LDI	(B)	1	1	X	X	0	0	0	X	X	0	0	0	0	0	1	1	←	I	M	M	E	D	I	A	T	E		D	A	T	A	→
6-14	LFI	(B)	1	1	X	X	0	0	0	X	X	0	0	0	0	0	1	0	←	I	M	M	E	D	I	A	T	E		D	A	T	A	→
6-15	LSI	(B)	1	1	X	X	0	0	0	X	X	0	0	0	0	0	0	1	←	I	M	M	E	D	I	A	T	E		D	A	T	A	→
6-25	LIF	(C)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
6-21	LTCF	(C)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
6-33	LXF	(C)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
6-29	SIF	(C)	0	0	X	X	0	0	0	X	X	←	D	E	S	T	I	N	A	T	I	O	N	→	0	1	1	1	0	0	0	0	0	0	0	0	0	0
6-37	SXF	(C)	0	0	X	X	0	0	0	X	X	←	D	E	S	T	I	N	A	T	I	O	N	→	1	0	1	1	0	0	0	0	0	0	0	0	0	0
6-19	LCF	(CC)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0
6-26	LIFC	(CC)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0
6-34	LXFC	(CC)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0
6-30	SIFC	(CC)	0	0	X	X	0	0	0	X	X	←	D	E	S	T	I	N	A	T	I	O	N	→	0	1	1	1	0	0	0	0	1	0	0	0	0	0
6-38	SXFC	(CC)	0	0	X	X	0	0	0	X	X	←	D	E	S	T	I	N	A	T	I	O	N	→	1	0	1	1	0	0	0	0	1	0	0	0	0	0
6-27	LIS	(D)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	←	S	O	U	R	C	E	→
6-22	LTCS	(D)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
6-35	LXS	(D)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	←	S	O	U	R	C	E	→
6-31	SIS	(D)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	←	D	E	S	T	I	N	A	T	I	O	N	→
6-39	SXS	(D)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	←	D	E	S	T	I	N	A	T	I	O	N	→
6-20	LCS	(DC)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
6-28	LISC	(DC)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	←	S	O	U	R	C	E	→
6-36	LXSC	(DC)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	←	S	O	U	R	C	E	→
6-32	SISC	(DC)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	←	D	E	S	T	I	N	A	T	I	O	N	→
6-40	SXSC	(DC)	0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	←	D	E	S	T	I	N	A	T	I	O	N	→
6-43	MII	(E)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	0	0	0	0	0	1	0	1	0	←	D	E	S	T	I	N	A	T	I	O	N	→
6-45	MIX	(E)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	0	0	0	0	1	0	0	1	0	←	D	E	S	T	I	N	A	T	I	O	N	→
6-47	MXI	(E)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	0	0	0	0	0	1	1	0	0	←	D	E	S	T	I	N	A	T	I	O	N	→
6-49	MXX	(E)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	0	0	0	0	1	0	1	0	0	←	D	E	S	T	I	N	A	T	I	O	N	→
6-44	MIIC	(EC)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	0	0	0	0	0	1	0	1	1	←	D	E	S	T	I	N	A	T	I	O	N	→
6-46	MIXC	(EC)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	0	0	0	0	1	0	0	1	1	←	D	E	S	T	I	N	A	T	I	O	N	→
6-48	MXIC	(EC)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	0	0	0	0	0	1	1	0	1	←	D	E	S	T	I	N	A	T	I	O	N	→
6-50	MXXC	(EC)	0	0	X	X	0	0	0	X	X	←	S	O	U	R	C	E	→	0	0	0	0	1	0	1	0	1	←	D	E	S	T	I	N	A	T	I	O	N	→
6-54	NOP		0	0	X	X	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6-53	PAUSE		0	0	X	X	←	d	→	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

FIGURE 6.2 DECISION PROCESSOR INSTRUCTION SET

Note 1. If the instruction from group C is a register Load instruction, the source register for the Move instruction must be the same numerical address as the source register for the Load instruction. Thus,

$$LIFn_s;\ MIXn_s,n_f$$

and

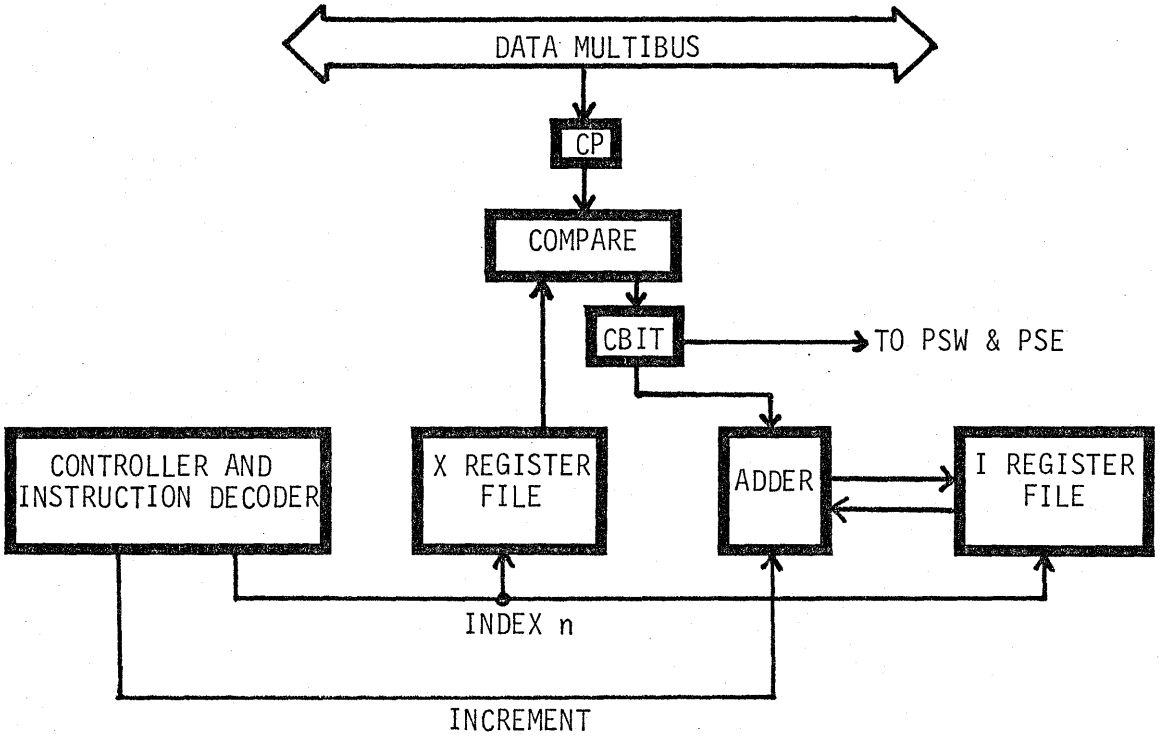
$$LXFn_s;\ MIXn_s,m_f$$

are both valid microprogrammed instructions. If the instruction from group C is a register store instruction, the source register for the Move instruction must have the same numerical address as the destination register for the Store instruction.

Note 2. The comments of Note 1 apply with each of the instructions from groups C and E replaced by their counterparts from group CC and EC respectively.

6.2.3 COMPARE INSTRUCTIONS

The DEP has two COMPARE instructions. That part of the DEP which is involved in COMPARE operations is shown below.



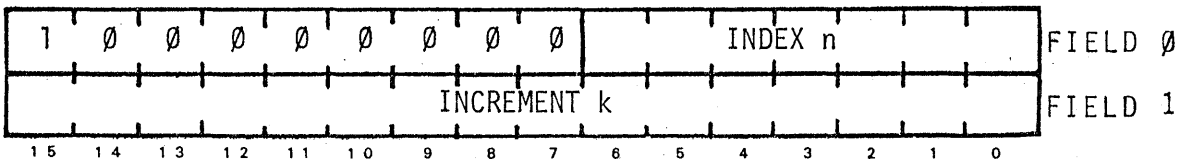
The COMPARE Instructions are:

CMM Compare and Modify

CMP Compare

CMM n,k

COMPARE AND MODIFY



OPERATION:

$CP_{.25} \leftarrow DM_{\emptyset}$

IF $CP_{.25} \leq X_{\emptyset}(INDEX)$

THEN $I_1(INDEX) \leftarrow I_{\emptyset}(INDEX)+INCREMENT$; $CBIT \leftarrow \emptyset$
ELSE IF $INCREMENT \neq \emptyset$
THEN $I_1(INDEX) \leftarrow I_{\emptyset}(INDEX)-INCREMENT$; $CBIT \leftarrow 1$
ELSE $I_1(INDEX) \leftarrow I_{\emptyset}(INDEX)-1$; $CBIT \leftarrow 1$

ERROR CONDITION(S):

CBIT if enabled to the AER line.

DESCRIPTION:

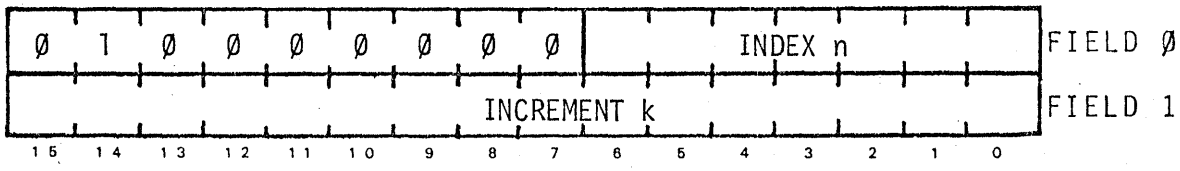
The contents of the DATA MULTIBUS FIRST are saved in register CP. The contents of the specified X register are then compared with the contents of CP. If X register is greater than or equal to CP, the I register is incremented by INCREMENT; otherwise, it is decremented. If INCREMENT is zero, this instruction will increment by zero or decrement by one.

EXAMPLE:

```
.
.
.
CMM X1,8
.
.
.
```

CMP n,k

COMPARE



OPERATION:

$CP_{.25} \leftarrow DM_{\emptyset}$

IF $CP_{.25} \leq X_{\emptyset}$(INDEX)

THEN I_1(INDEX) ← + INCREMENT; CBIT ← Ø

ELSE I_1(INDEX) ← - INCREMENT; CBIT ← 1

ERROR CONDITION(S):

CBIT if enabled to the AER line.

DESCRIPTION:

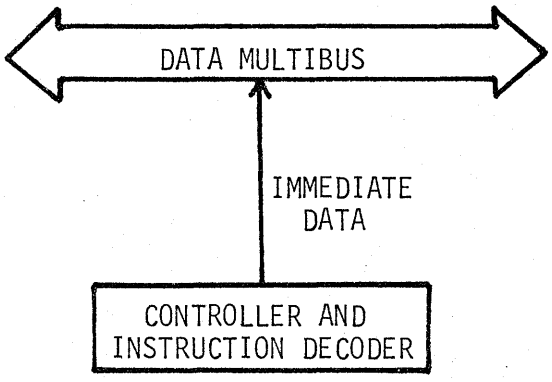
The contents of the DATA MULTIBUS FIRST are saved in register CP. The contents of the specified X register are then compared with the contents of CP. If X register is greater than or equal to CP, the I register is loaded with INCREMENT; otherwise it is loaded with -INCREMENT.

EXAMPLE:

```
.
.
.
CMP  X4,32
.
.
.
```

6.2.4 LOAD IMMEDIATE DATA INSTRUCTIONS

The DEP has three LOAD IMMEDIATE DATA instructions. That part of the DEP which is involved with these instructions is shown below.



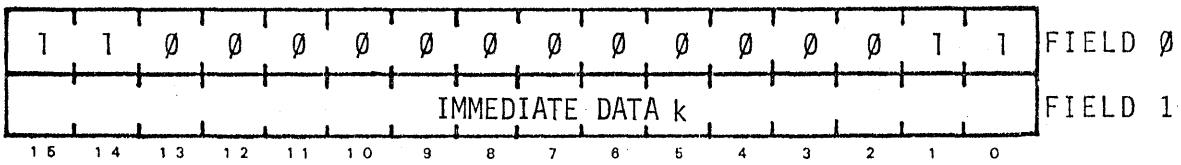
The instructions are often used to load registers in the various register files (i.e., the I and X Register Files in DEP, the Temporary Register File in ARP, and the General Register File in COP).

The LOAD IMMEDIATE DATA Instructions are:

LDI	Load Double Immediate
LFI	Load First Immediate
LSI	Load Second Immediate

LDI k

LOAD DOUBLE IMMEDIATE



OPERATION:

DM_1 ← IMMEDIATE DATA

$DM_{1.5}$ ← IMMEDIATE DATA

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

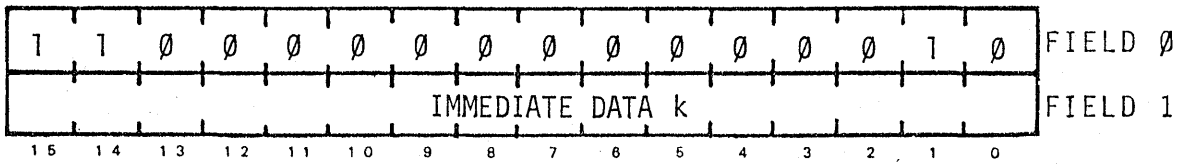
The immediate data is placed on the DATA MULTIBUS FIRST and SECOND.

EXAMPLE:

```
      .
      .
      .
LDI   DATA
      .
      .
      .
```

LFI k

LOAD FIRST IMMEDIATE



OPERATION:

DM_1 ← IMMEDIATE DATA

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

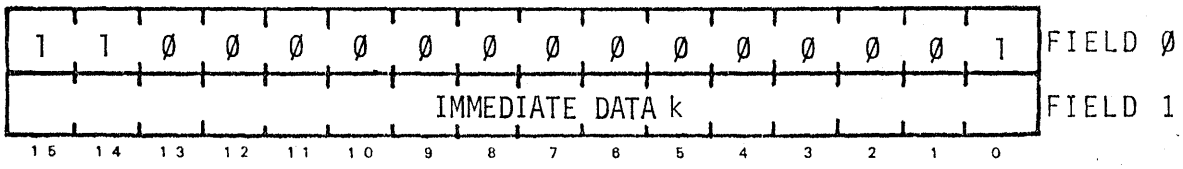
The immediate data is placed on the DATA MULTIBUS FIRST.

EXAMPLE:

```
      .
      .
      .
LFI   DATA
      .
      .
      .
```

LSI k

LOAD SECOND IMMEDIATE



OPERATION:

$DM_{1.5}$ ← IMMEDIATE DATA

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

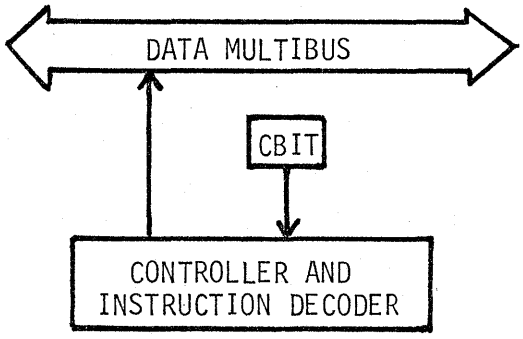
The immediate data are placed on the DATA MULTIBUS SECOND.

EXAMPLE:

```
.
.
.
LSI DATA
.
.
.
```

6.2.5 LOAD CONDITION INSTRUCTIONS

The DEP has four LOAD CONDITION instructions. These four instructions are bit load instructions (i.e., they only affect bit 15 of the DATA MULTIBUS). The part of the DEP involved in these instructions is shown below.

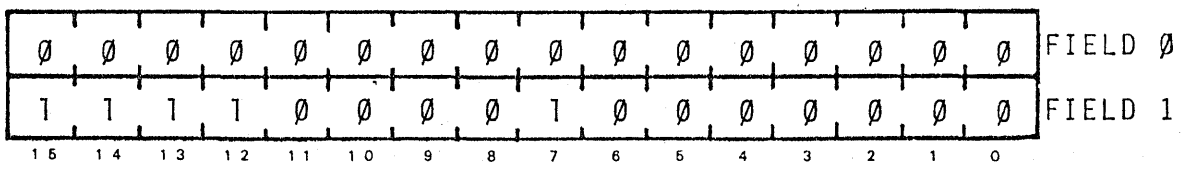


The LOAD CONDITION Instructions are:

LCF	Load Condition First
LCS	Load Condition Second
LTCF	Load True Condition First
LTCS	Load True Condition Second

LCF

LOAD CONDITION FIRST



OPERATION:

IF CBIT IS TRUE

THEN $DM_1(15) \leftarrow 1$

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUC cycle.

DESCRIPTION:

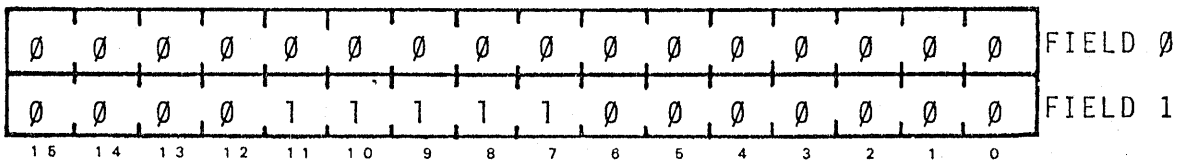
If the condition bit is true, bit 15 of the DATA MULTIBUS FIRST is set.

EXAMPLE:

```
.
.
.
LCF
.
.
.
```

LCS

LOAD CONDITION SECOND



OPERATION:

IF CBIT IS TRUE

THEN $DM_{1.5}(15) \leftarrow 1$

ERROR CONDITION(S):

BUS CONFLICT(DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

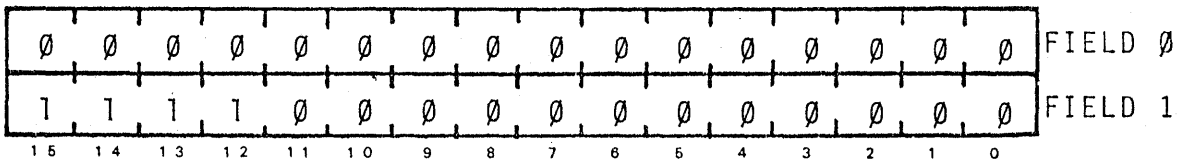
If the condition bit is true, bit 15 of the DATA MULTIBUS SECOND is set.

EXAMPLE:

```
.
.
.
LCS
.
.
.
```

LTCF

LOAD TRUE CONDITION FIRST



OPERATION:

$DM_1(15) \leftarrow 1$

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

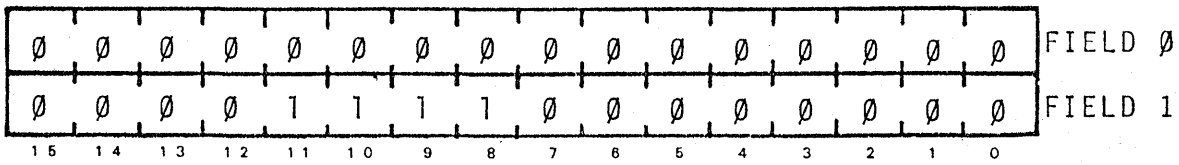
Bit 15 of the DATA MULTIBUS is set.

EXAMPLE:

```
      .
      .
      .
    LTCF
      .
      .
      .
```

LTCS

LOAD TRUE CONDITION SECOND



OPERATION:

$DM_{1.5}(15) \leftarrow 1$

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

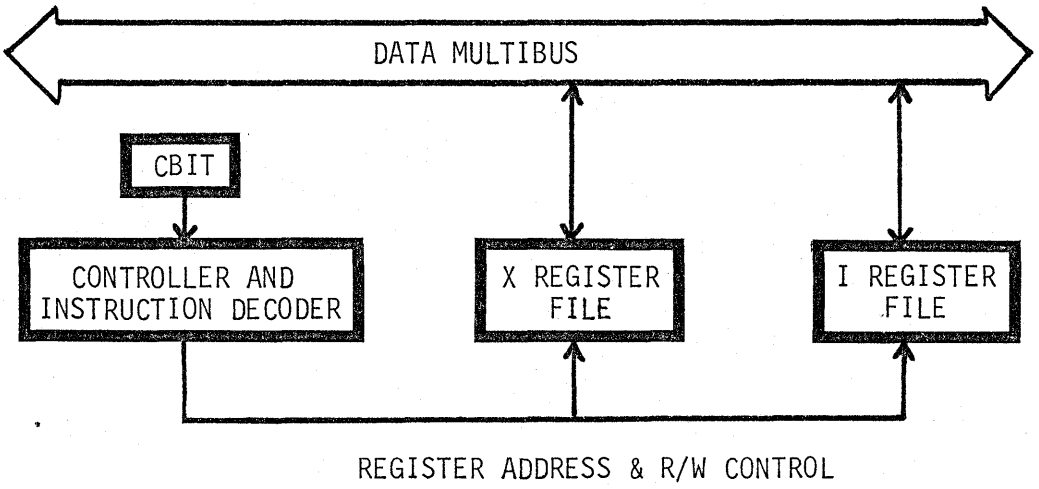
Bit 15 of the DATA MULTIBUS SECOND is set.

EXAMPLE:

```
      .
      .
      .
    LTCS
      .
      .
      .
```

6.2.6 LOAD/STORE REGISTER INSTRUCTIONS

The DEP has sixteen LOAD/STORE REGISTER instructions. The LOAD instructions are used to load data from an I register or an X register to the DATA MULTIBUS. Similarly, the STORE instructions are used to store data obtained from the DATA MULTIBUS into an I or an X register. That part of the DEP which is involved with these instructions is shown below.



The LOAD/STORE instructions for the I registers are:

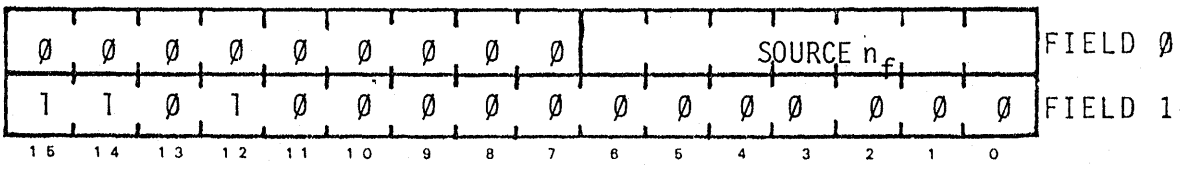
LIF	Load I First
LIFC	Load I First Conditionally
LIS	Load I Second
LISC	Load I Second Conditionally
SIF	Store I First
SIFC	Store I First Conditionally
SIS	Store I Second
SISC	Store I Second Conditionally

The LOAD/STORE instructions for the X registers are:

LXF	Load X First
LXFC	Load X First Conditionally
LXS	Load X Second
LXSC	Load X Second Conditionally
SXF	Store X First
SXFC	Store X First Conditionally
SXS	Store X Second
SXSC	Store X Second Conditionally

LIF n_f

LOAD I FIRST



OPERATION:

$DM_1 \leftarrow I_Ø(SOURCE)$

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

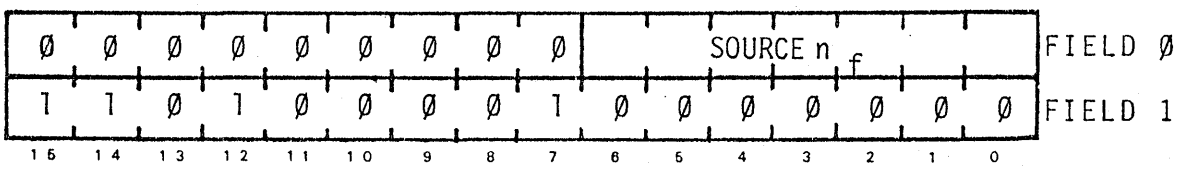
The contents of the specified I register are placed on the DATA MULTIBUS FIRST.

EXAMPLE:

```
.
.
.
LIF  I4
.
.
.
```

LIFC n_f

LOAD I FIRST CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN $DM_1 \leftarrow I_Ø(SOURCE)$

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

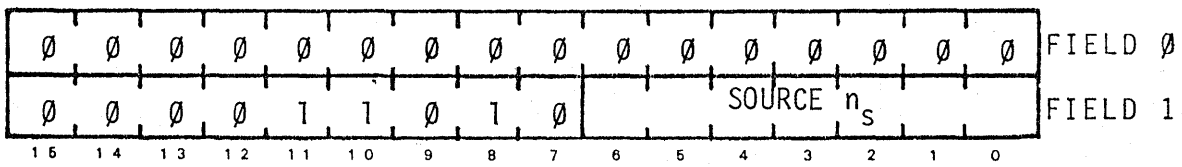
If the condition bit is true, the contents of the specified I register are placed on the DATA MULTIBUS FIRST.

EXAMPLE:

```
.
.
.
LIFC I4
.
.
.
```

LIS n_S

LOAD I SECOND



OPERATION:

$DM_{1.5} \leftarrow I_{.5}(SOURCE)$

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

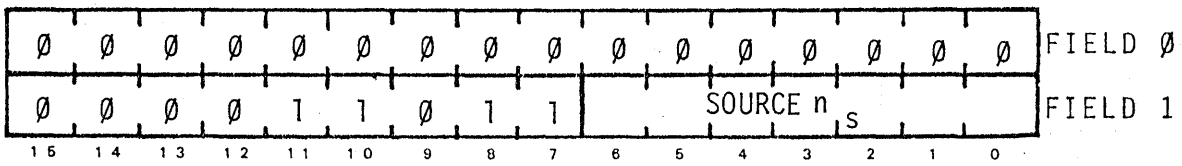
The contents of the specified I register are placed on the DATA MULTIBUS SECOND.

EXAMPLE:

```
.
.
.
LIS I4
.
.
.
```

LISC n_s

LOAD I SECOND CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN $DM_{1.5} \leftarrow I_{.5}$(SOURCE)

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

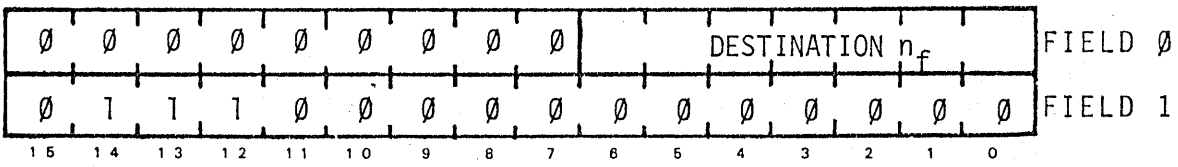
If the condition bit is true, the contents of the specified I register are placed on the DATA MULTIBUS SECOND.

EXAMPLE:

```
.
.
.
LISC I4
.
.
.
```

SIF n_f

STORE I FIRST



OPERATION:

$I_{.5}$(DESTINATION) ← $DM_Ø$

ERROR CONDITION(S):

None

DESCRIPTION:

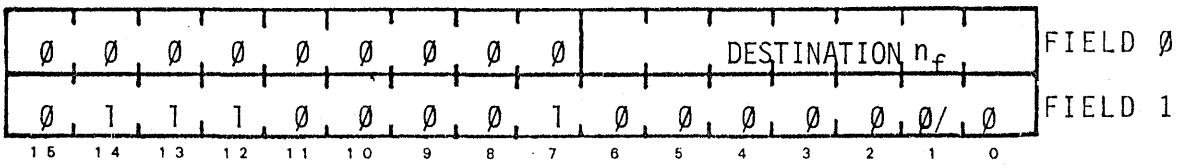
The specified I register is loaded with the data on the DATA MULTIBUS FIRST.

EXAMPLE:

```
.
.
.
SIF  I4
.
.
.
```

SIFC n_f

STORE I FIRST CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN $I_{.5}$(DESTINATION) ← $DM_Ø$

ERROR CONDITION(S):

None

DESCRIPTION:

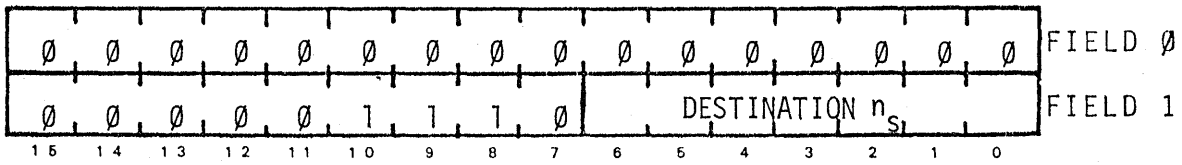
If the condition bit is true, the specified I register is loaded with the data on the DATA MULTIBUS FIRST.

EXAMPLE:

```
.
.
.
SIFC  I4
.
.
.
```

SIS n_S

STORE I SECOND



OPERATION:

I_1(DESTINATION) ← $DM_{.5}$

ERROR CONDITION(S):

None

DESCRIPTION:

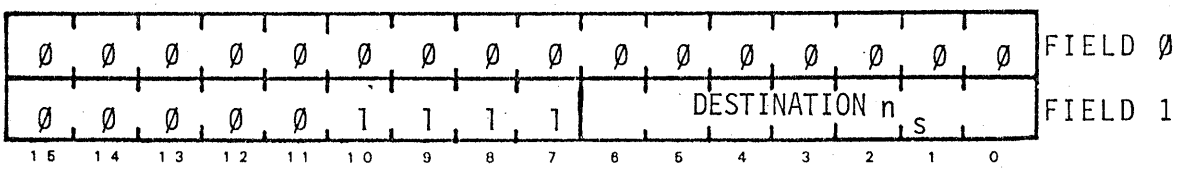
The specified I register is loaded with the data on the DATA MULTIBUS SECOND.

EXAMPLE:

```
.
.
.
SIS  I4
.
.
.
```

SISC n_s

STORE I SECOND CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN I_1 (DESTINATION) ← $DM_{.5}$

ERROR CONDITION(S):

None

DESCRIPTION:

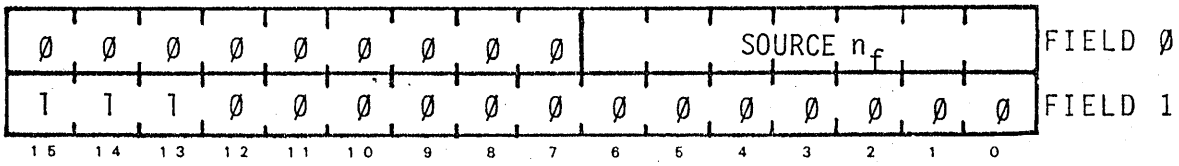
If the condition bit is true, the specified I register is loaded with the data on the DATA MULTIBUS SECOND.

EXAMPLE:

```
.
.
.
SISC  I4
.
.
.
```

LXF n_f

LOAD X FIRST



OPERATOR:

$DM_1 \leftarrow X_Ø$(SOURCE)

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in in the above specified BUS cycle.

DESCRIPTION:

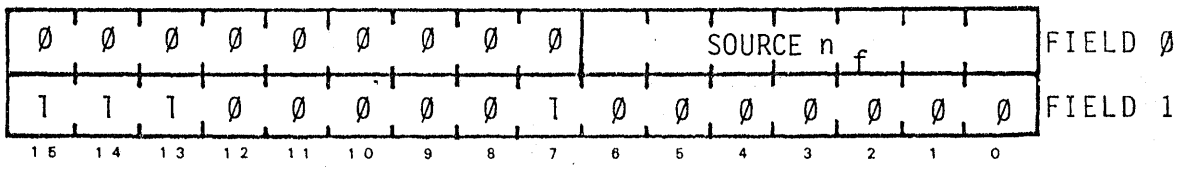
The contents of the specified X register are placed on the DATA MULTIBUS FIRST.

EXAMPLE:

```
      .
      .
      .
LXF   X4
      .
      .
      .
```

LXFC n_f

LOAD X FIRST CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN $DM_1 \leftarrow X_Ø$(SOURCE)

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

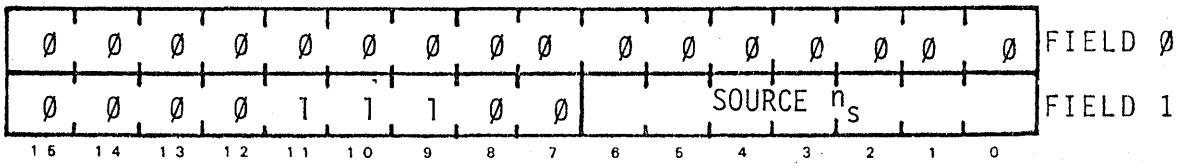
If the condition bit is true, the contents of the specified X register are placed on the DATA MULTIBUS FIRST.

EXAMPLE:

```
.
.
.
LXFC  X4
.
.
.
```

LXS n_S

LOAD X SECOND



OPERATION:

$DM_{1.5} \leftarrow X_{.5}(SOURCE)$

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

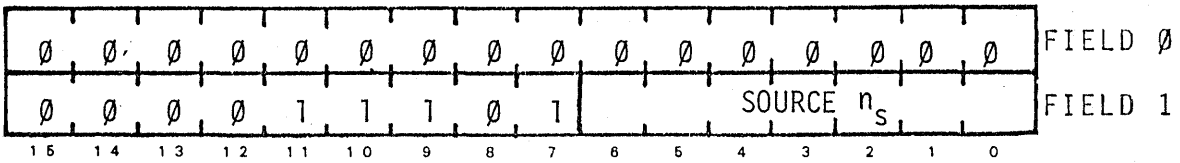
The contents of the specified X register are placed on the DATA MULTIBUS SECOND.

EXAMPLE:

```
.
.
.
LXS  X4
.
.
.
```

LXSC n_S

LOAD X SECOND CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN $DM_{1.5} \leftarrow X_{.5}$(SOURCE)

ERROR CONDITION(S):

BUS CONFLICT (DATA), if there is other data on the DM in the above specified BUS cycle.

DESCRIPTION:

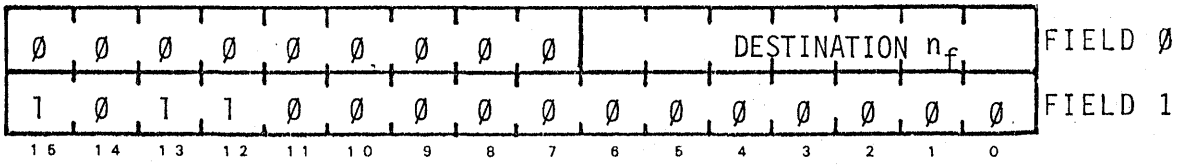
If the condition bit is true, the contents of the specified X register are placed on the DATA MULTIBUS SECOND.

EXAMPLE:

```
      .
      .
      .
LXSC  X4
      .
      .
      .
```

SXF n_f

STORE X FIRST



OPERATION:

$X_{.5}(\text{DESTINATION}) \leftarrow DM_{Ø}$

ERROR CONDITION(S):

None

DESCRIPTION:

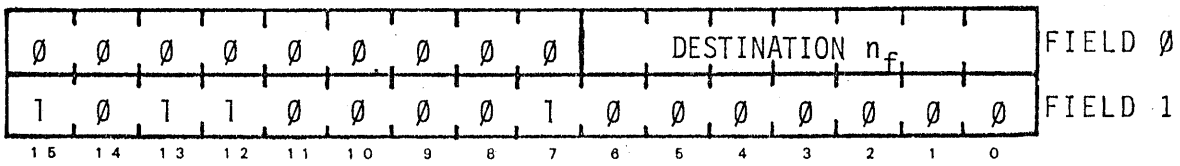
The specified X register is loaded with the data on the DATA MULTIBUS FIRST.

EXAMPLE:

```
.
.
.
SXF  X4
.
.
.
```

SXFC n_f

STORE X FIRST CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN $X_{.5}$(DESTINATION) ← $DM_{Ø}$

ERROR CONDITION(S):

None

DESCRIPTION:

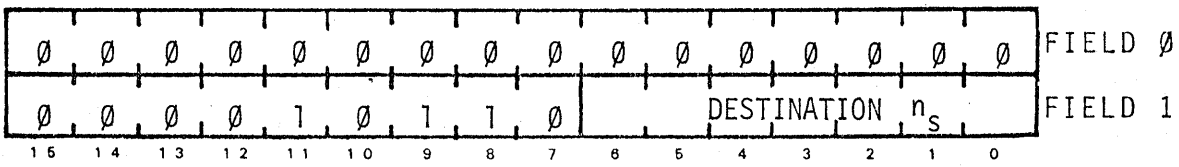
If the condition bit is true, the specified X register is loaded with the data on the DATA MULTIBUS FIRST.

EXAMPLE:

```
.
.
.
SXFC  X4
.
.
.
```

SXS n_s

STORE X SECOND



OPERATION:

X_1(DESTINATION) ← $DM_{.5}$

ERROR CONDITION(S):

None

DESCRIPTION:

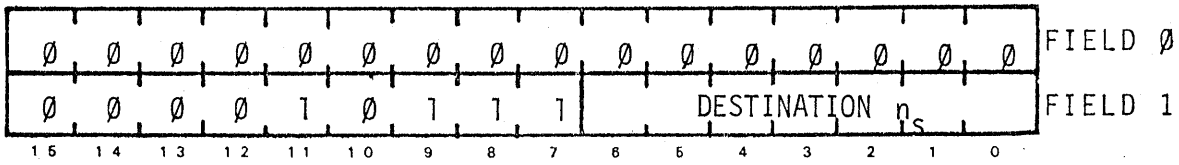
The specified X register is loaded with the data on the DATA MULTIBUS SECOND.

EXAMPLE:

```
.
.
.
SXS  X4
.
.
.
```

SXSC n_s

STORE X SECOND CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN X_1(DESTINATION) ← $DM_{.5}$

ERROR CONDITION(S):

None

DESCRIPTION:

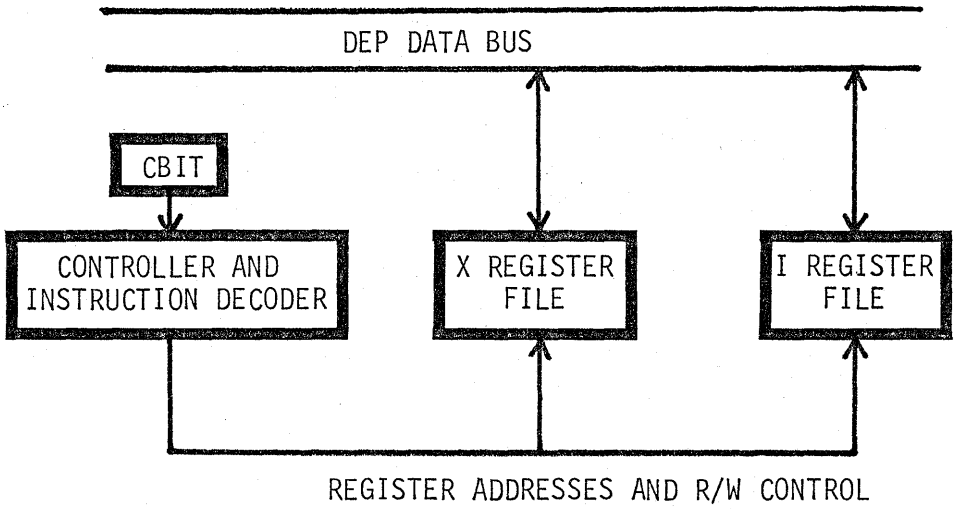
If the condition bit is true, the specified X register is loaded with the data on the DATA MULTIBUS SECOND.

EXAMPLE:

```
.
.
.
SXSC  X4
.
.
.
```

6.2.7 MOVE REGISTER INSTRUCTIONS

The DEP has eight MOVE instructions which are used to transfer the contents of one I or X register to another I or X register. These data transfers take place via the internal DEP Data Bus (DEP DB). That part of the DEP involved in these internal MOVE instructions is shown below.

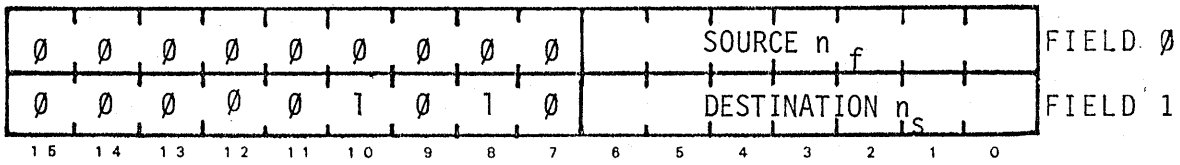


The DEP MOVE instructions are:

MII	Move I to I
MIIC	Move I to I Conditionally
MIX	Move I to X
MIXC	Move I to X Conditionally
MXI	Move X to I
MXIC	Move X to I Conditionally
MXX	Move I to X
MXXC	Move I to X Conditionally

MII n_f,n_s

MOVE I (n_f) to I (n_s)



OPERATION:

I_1(DESTINATION) ← $I_Ø$(SOURCE)

ERROR CONDITION(S):

None

DESCRIPTION:

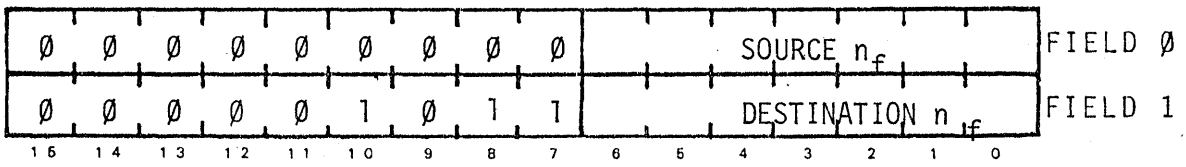
The contents of the source I register are placed in the destination I register.

EXAMPLE:

```
.
.
.
MII  I4,I5
.
.
.
```

MIIC n_f,n_s

MOVE I(n_f) to I(n_s) CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN I_1(DESTINATION) ← $I_Ø$(SOURCE)

ERROR CONDITION(S):

None

DESCRIPTION:

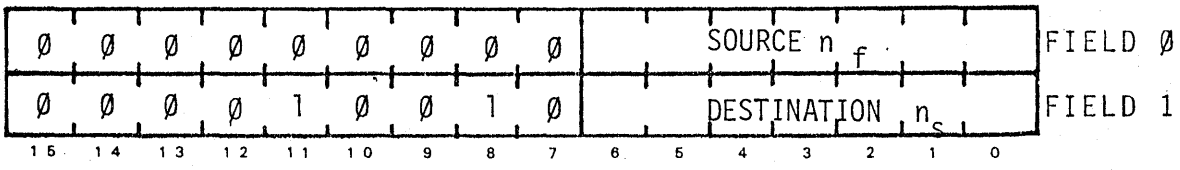
If the condition bit is true, the contents of the source I register are placed in the destination I register.

EXAMPLE:

```
.
.
.
MIIC  I4,I5
.
.
.
```

MIX n_f,n_s

MOVE $I(n_f)$ to $X(n_s)$



OPERATION:

X_1(DESTINATION) ← $I_Ø$(SOURCE)

ERROR CONDITION(S):

None

DESCRIPTION:

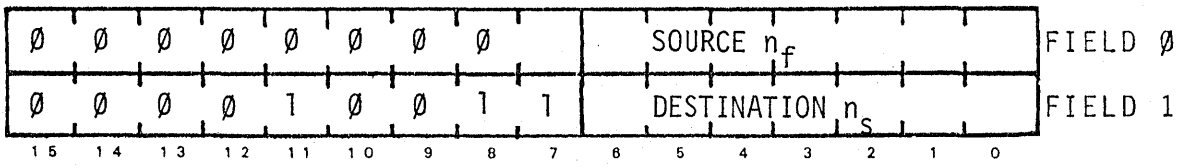
The contents of the source I register are placed in the destination X register.

EXAMPLE:

```
.
.
.
MIX  I4,X5
.
.
.
```

MIXC n_f,n_s

MOVE I(n_f) to X(n_s) CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN X_1(DESTINATION) ← $I_Ø$(SOURCE)

ERROR CONDITION(S):

None

DESCRIPTION:

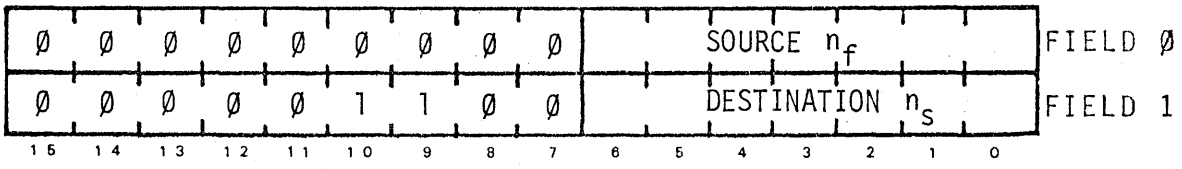
If the condition bit is true, the contents of the source I register are placed in the destination X register.

EXAMPLE:

```
.
.
.
MIXC  I4,X5
.
.
.
```

MXI n_f,n_s

MOVE $X(n_f)$ to $I(n_s)$



OPERATION:

I_1(DESTINATION) ← $X_Ø$(SOURCE)

ERROR CONDITION(S):

None

DESCRIPTION:

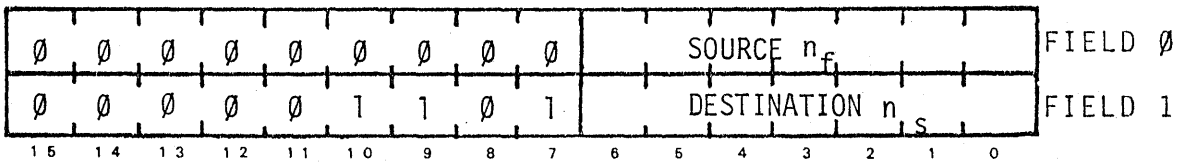
The contents of the source X register are placed in the destination I register.

EXAMPLE:

```
.
.
.
MXI  X4,I5
.
.
.
```

MXIC n_f,n_s

MOVE $X(n_f)$ to $I(n_s)$ CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN I_1(DESTINATION) ← $X_Ø$(SOURCE)

ERROR CONDITION(S):

None

DESCRIPTION:

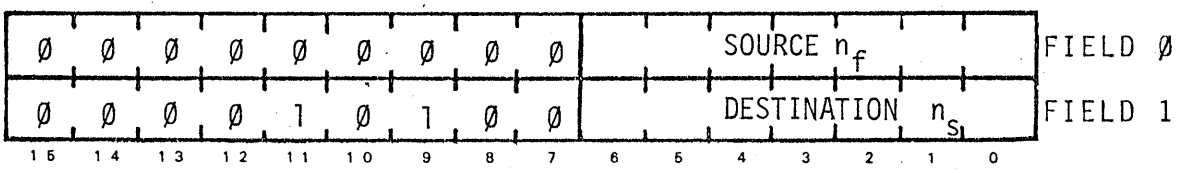
If the condition bit is true, the contents of the source X register are placed in the destination I register.

EXAMPLE:

```
.
.
.
MXIC  X4,I5
.
.
.
```

MXX n_f,n_s

MOVE $X(n_f)$ to $X(n_s)$



OPERATION:

X_1(DESTINATION) ← $X_Ø$(SOURCE)

ERROR CONDITION(S):

None

DESCRIPTION:

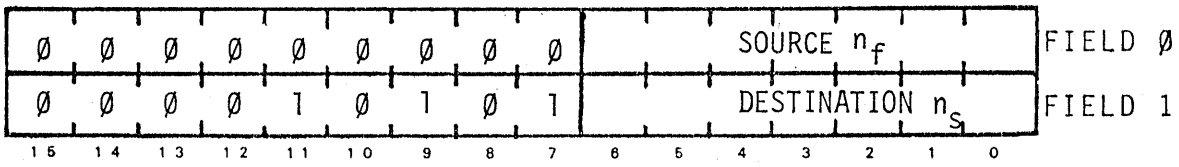
The contents of the source X register are placed in the destination X register.

EXAMPLE:

```
.
.
.
MXX  X4,X5
.
.
.
```

MXXC n_f,n_s

MOVE $X(n_f)$ to $X(n_s)$ CONDITIONALLY



OPERATION:

IF CBIT IS TRUE

THEN X_1(DESTINATION) ← $X_Ø$(SOURCE)

ERROR CONDITION(S):

None

DESCRIPTION:

If the condition bit is true, the contents of the source X register are placed in the destination X register.

EXAMPLE:

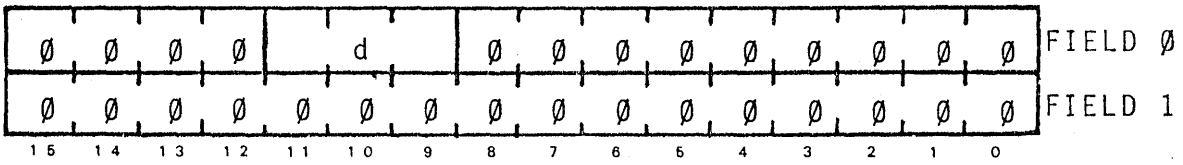
```
.
.
.
MXXC  X4,X5
.
.
.
```

6.2.8 PAUSE/NOP INSTRUCTIONS

The PAUSE instruction may be microprogrammed with any other DEP instruction or it may be issued as a stand-alone instruction. The NOP instruction is only used as a stand-alone instruction. The NOP and PAUSE instructions are defined below.

PAUSE d

PAUSE FOR d INSTRUCTION CYCLES



OPERATION:

None

ERROR CONDITION(S):

None

DESCRIPTION:

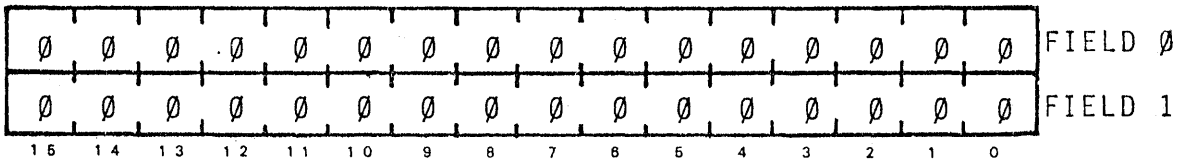
Suspend execution for d instruction cycles following the PAUSE instruction. This has the same effect as (d+1) NOP's.

EXAMPLE:

```
.
.
.
PAUSE  6
.
.
.
```

NOP

NO OPERATION



OPERATION:

None

ERROR CONDITION(S):

None

DESCRIPTION:

Suspends execution for one instruction cycle.

EXAMPLE:

```
NOP
```

7. THE MEMORY ADDRESS PROCESSOR

7.0 INTRODUCTION

The MEMORY ADDRESS PROCESSOR (MAP) initiates Data Memory read/write operations and may be viewed as the AD 10 Data Memory controller. The MAP has instructions to perform both direct addressing and indexed addressing as well as instructions for two types of address mapping, aligned and unaligned. The MAP is only concerned with addressing Data Memory and has no control over the corresponding data to be written to or read from Data Memory.

7.1 ORGANIZATION OF THE MAP

A block diagram of the MAP is shown in Figure 7.1. That part of Figure 7.1 which is in black is common to all processors and is described in Section 2.6. The elements in Figure 7.1 which are in red are those which are specific to the MAP. A more detailed diagram of key MAP elements is presented in Figure 7.2.

Data Memory read/write operations may be either indexed or non-indexed. The I register file and the adder are involved in indexed address operations. Indexed address operations are discussed in Section 7.2.3.1.

The MAP is physically located on the same circuit card assembly which contains the Decision Processor (DEP). The I register file, shown in Figures 7.1 and 7.2, is shared between MAP and DEP in the sense that it may be accessed by either processor. The MAP is limited to read-only access to the I register file. The DEP may perform either read or write operations on the I register file. See Chapter 6 for more information in this regard. NOTE: It is possible for a given register in the I register file to be accessed by both MAP and DEP in one instruction cycle. In this event, the register read operation required by MAP is performed before the register operation specified by the DEP instruction and no conflict occurs.

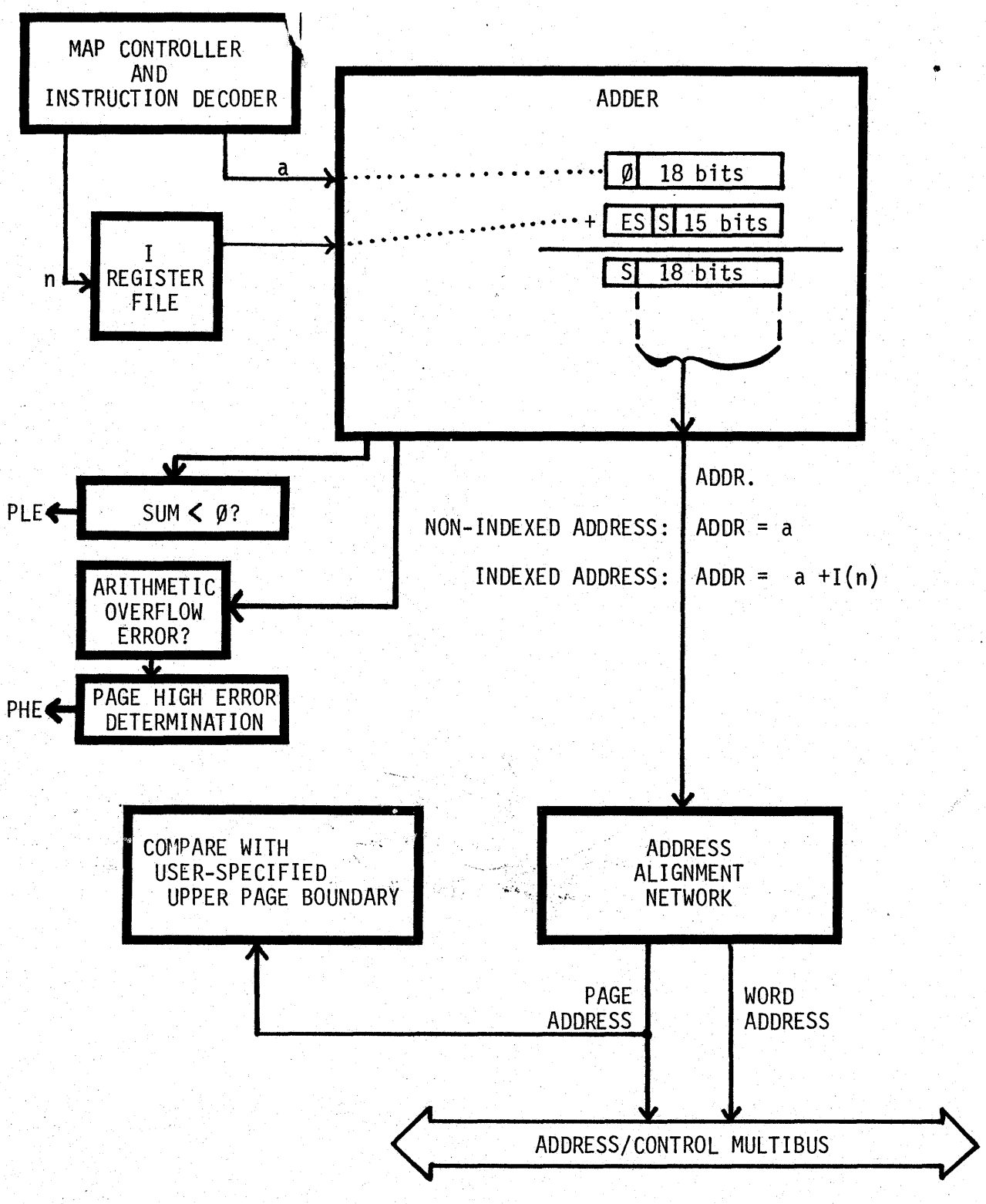


FIGURE 7.2 MAP INSTRUCTION ADDRESS OPERATIONS

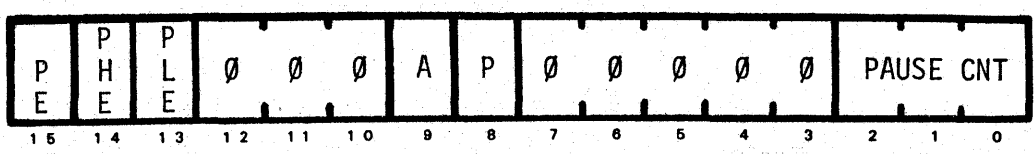
The MAP instructions allow the address output to the Data Memory to be either "aligned" or "unaligned". If the MAP instruction calls for an unaligned address read or write operation, the address specified as part of the MAP instruction or the computed indexed address is output directly to the Data Memory via the ADDRESS MULTIBUS as described in Section 7.2.3.2. Thus, an unaligned addressing operation may be viewed as an absolute addressing operation. If an aligned address read or write operation is specified, the address output from the MAP to Data Memory is obtained as the result of an operation performed by the address alignment network (see Figure 7.1) on the address specified as part of the MAP instruction or on the computed indexed address. The operation performed by the address alignment network is described in Section 7.2.3.3.

Connections from the COP to the Processor Status Enable Register via the DATA MULTIBUS and from the COP to the Program Counter via the ADDRESS/CONTROL MULTIBUS are shown in red in Figure 7.1. These connections indicate that the Processor Status Enable Register and the Program Counter can be changed by instructions in a COP program when the AD 10 is in the RUN mode.

7.1.1 Processor Address

The Processor address for the MAP is Ø1.

7.1.2 Processor Status Word

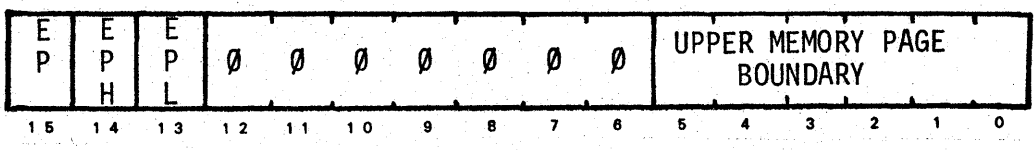


BIT	DESCRIPTION
Ø-2	Remaining PAUSE count.
3-7	Unassigned
8	MAP is present when set.
9	MAP is active when set.
10-12	Unassigned
13*	Page Low Error. Set if the page address is a negative number.
14*	Page High Error. Set if the page address exceeds the programmer specified upper page boundary or if an arithmetic overflow occurs on the computation of an indexed address.
15*	Page Error. Set if either PLE or PHE occurs.

* Once set, these bits remain set until cleared by a read operation.

The MAP Processor Status Word (PSW) contains information on the current status of the MAP. The PSW is a read-only register.

7.1.3 Processor Status Enable



BIT	DESCRIPTION
Ø-5	Upper memory page boundary. User specified upper page in Data Memory. Since there is no hardware protection in the Data Memory to indicate that a page addressed is not physically present, this allows the user to protect him/her self. NOTE: The upper memory page boundary should be specified in any instruction which set bit 14, since the hardware does not provide a default setting for these bits.
6-12	Unassigned.
13	Enable the PLE line to the AER line.
14	Enable the PHE line to the AER line.
15	Enable error on above two conditions (i.e., PE) to the AER line.

The Processor Status Enable Register (PSE) is a write-only register. This register has the same address as the MAP Processor Status Word.

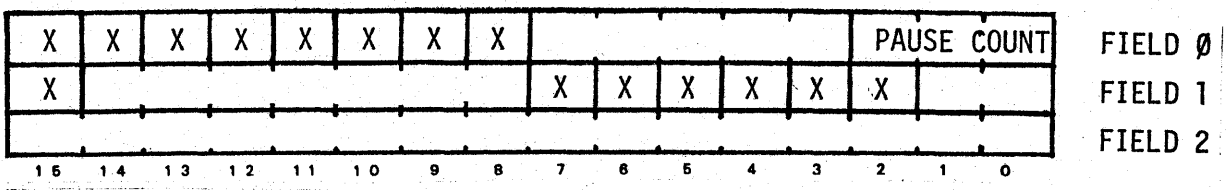
7.1.4 Program Memory

The MAP program memory contains 1,024 words. Each 48-bit instruction word is divided into three 16-bit fields. A MAP instruction is normally loaded as a sequence of three 16-bit words (one per field) from the host processor. However, each field is individually addressable and can be accessed from the host processor for a read/write operation.

7.2 THE MAP INSTRUCTION SET

7.2.1 Map Instruction Word Format

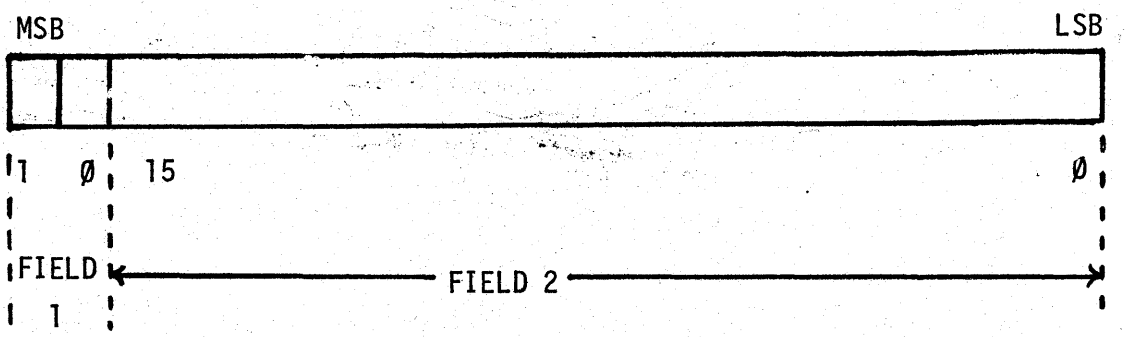
The MAP instruction word format is:



The bits labelled "X" in the above diagram are unused. The unused bits cannot be set and are always read as a Ø.

Figure 7.3 provides a summary of the MAP instruction set.

In each MAP read or write instruction, bits Ø and 1 of Field 1 are appended to Field 2 to form the address portion of the instruction. The format for this address information is:



Bits 8 through 14 of Field 1 are used to specify the address of the register in the I register file which contains the index to be used in a read or write instruction which involves indexed addressing.

7.2.2 Map Instruction Microprogramming

The only microprogramming allowed in a MAP instruction is that of a PAUSE instruction with a read instruction or a write instruction.

7.2.3 Read/Write Instructions

Figure 7.2 shows details of that part of the MAP which determines the page and word addresses to be transmitted via the ADDRESS/CONTROL MULTIBUS to Data Memory in a read or write operation.

Every read and write instruction specifies an 18-bit address "a" to be used in determining the actual address to be transmitted to Data Memory. In addition, the instruction also specifies whether or not the address "a" is to be indexed and/or aligned.

7.2.3.1 Indexed Addressing

An indexed address is obtained by adding the contents of one of the 128 registers in the I register file to the address "a" specified in the read or write instruction. An instruction which involves an indexed address contains a 7-bit number "n" which specifies the register in the I register file which is to be used.

The adder shown in Figure 2.7 is used to compute indexed addresses. Both the address "a" and the contents of the specified I register are treated as integers in two's-complement format by the adder. This allows the indexed address to lie anywhere in the range [a - 32,768] to [a + 32,767]. Depending on the specified address "a" * and the contents of the specified I register, it is possible for the computed indexed address to be a negative number. Since all valid Data Memory addresses are non-negative, a computed indexed address which is a negative number represents an error situation. The output of the adder is checked to determine whether or not the computed indexed address is negative. If it is negative, a Page Low Error (PLE) condition is generated. (See Figure 7.2).

* NOTE: The address "a" is a positive integer. See Figure 7.2.

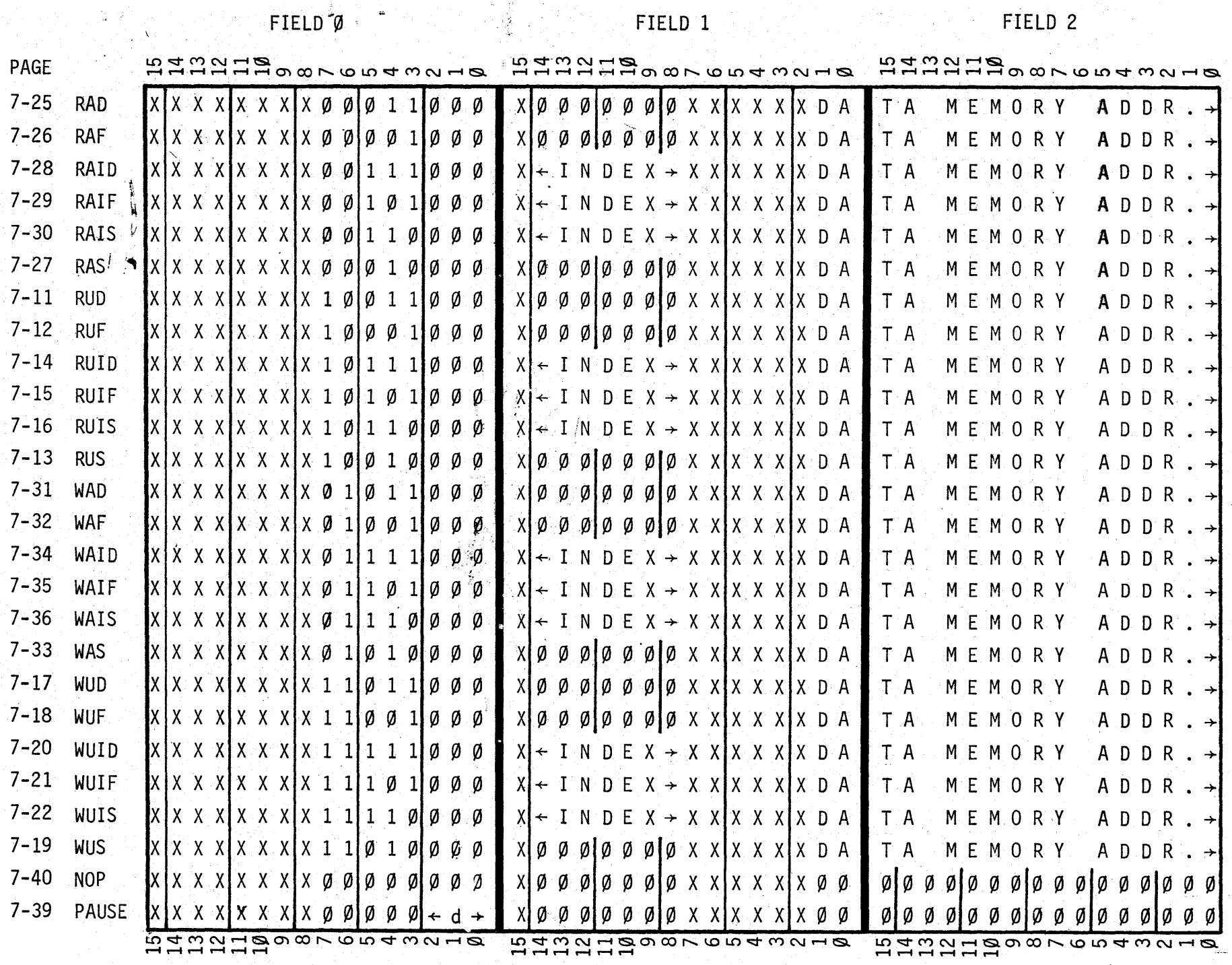


FIGURE 7.3 MEMORY ADDRESS PROCESSOR INSTRUCTION SET

The output of the adder is also checked to determine if an arithmetic overflow has occurred on the computation of an indexed address. Because of the modulo 2^{19} wraparound which occurs in an arithmetic overflow situation, the resulting page address could lie within the range of valid page addresses (i.e., between page Ø and the user specified upper memory page boundary). This is clearly an error situation as the indexed address is trying to call for a memory page whose address is greater than 63. Thus, a Page High Error (PHE) condition is generated if an arithmetic overflow occurs in the adder.

7.2.3.2 Unaligned Read/Write Instructions

In Figure 7.2, the output of the adder is labelled ADDR. For non-indexed read or write operations:

$$ADDR \leftarrow a$$

For indexed read or write instructions:

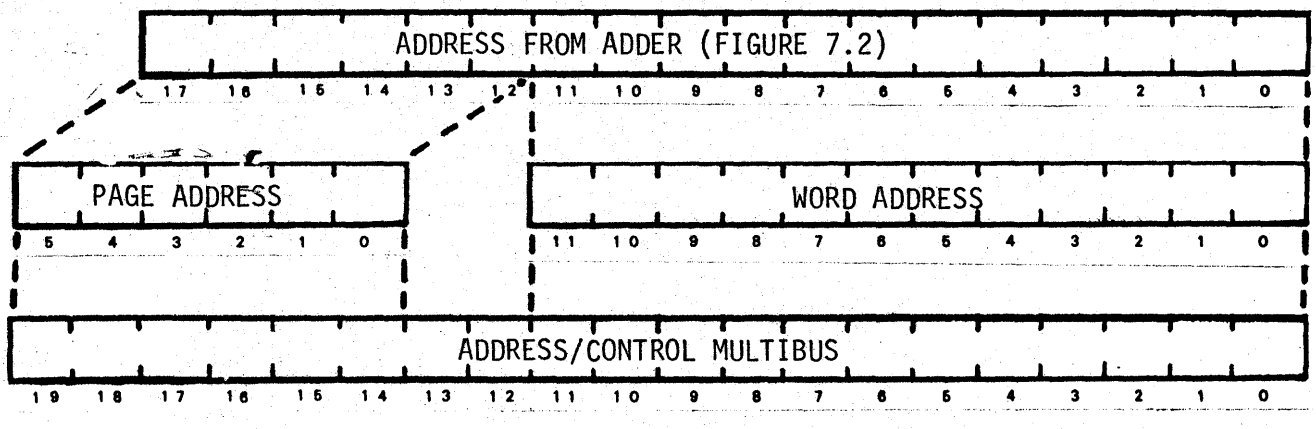
$$ADDR \leftarrow a + I(n)$$

If the read or write instruction specifies an unaligned address, the address alignment network simply outputs ADDR to the ADDRESS/CONTROL MULTIBUS as follows:

$$AM_1(14{:}19) \leftarrow ADDR_Ø(12{:}17) \quad \text{(PAGE ADDRESS)}$$

$$AM_1(Ø{:}11) \leftarrow ADDR_Ø(Ø{:}11) \quad \text{(WORD ADDRESS)}$$

The operation of the address alignment network for an unaligned read or write instruction is illustrated in the following diagram:

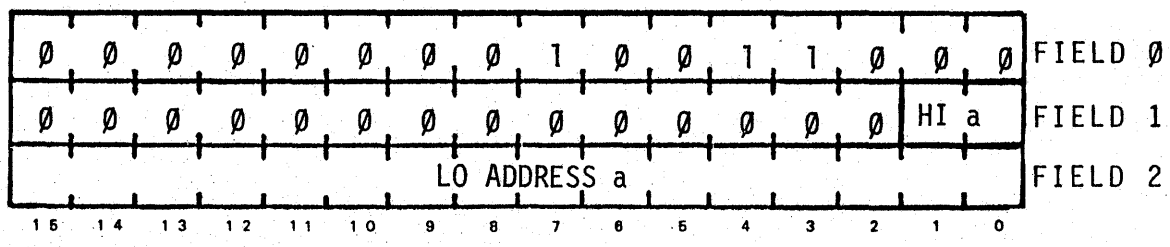


The unaligned read/write instructions are:

RUD a	READ UNALIGNED DOUBLE
RUF a	READ UNALIGNED FIRST
RUS a	READ UNALIGNED SECOND
RUID a,n	READ UNALIGNED INDEXED DOUBLE
RUIF a,n	READ UNALIGNED INDEXED FIRST
RUIS a,n	READ UNALIGNED INDEXED SECOND
WUD a	WRITE UNALIGNED DOUBLE
WUF a	WRITE UNALIGNED FIRST
WUS a	WRITE UNALIGNED SECOND
WUID a,n	WRITE UNALIGNED INDEXED DOUBLE
WUIF a,n	WRITE UNALIGNED INDEXED FIRST
WUIS a,n	WRITE UNALIGNED INDEXED SECOND

RUD a

READ UNALIGNED DOUBLE



OPERATION:

$AM_1 \leftarrow$ UNALIGNED(ADDRESS)

$AM_{1.5} \leftarrow$ UNALIGNED(ADDRESS+1)

$DM_5 \leftarrow MEM_1$(ADDRESS)

$DM_{5.5} \leftarrow MEM_{1.5}$(ADDRESS+1)

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS,DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange if enabled to AER line.

DESCRIPTION:

An address pair is sent out on the ADDRESS MULTIBUS to Data Memory.* After a delay, data will appear on the DATA MULTIBUS.

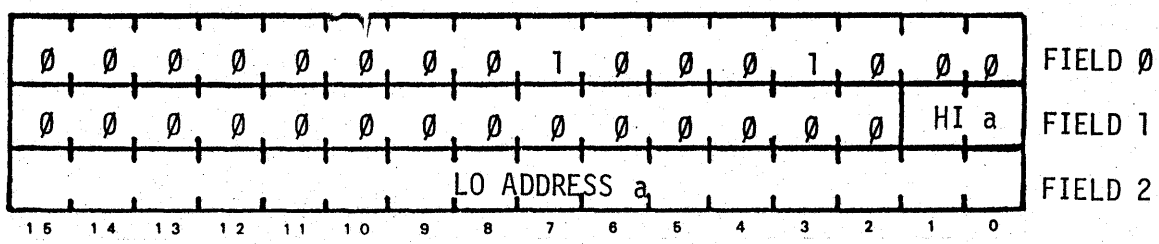
*This instruction is included primarily for maintenance purposes; a memory page access error will occur if ADDRESS is not the last address of a memory page.

EXAMPLE:

```
      .
      .
      .
RUD   FUN1
      .
      .
      .
```

RUF a

READ UNALIGNED FIRST



OPERATION:

$AM_1 \leftarrow$ UNALIGNED(ADDRESS)

$DM_5 \leftarrow MEM_1$(ADDRESS)

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS,DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange if enabled to AER line.

DESCRIPTION:

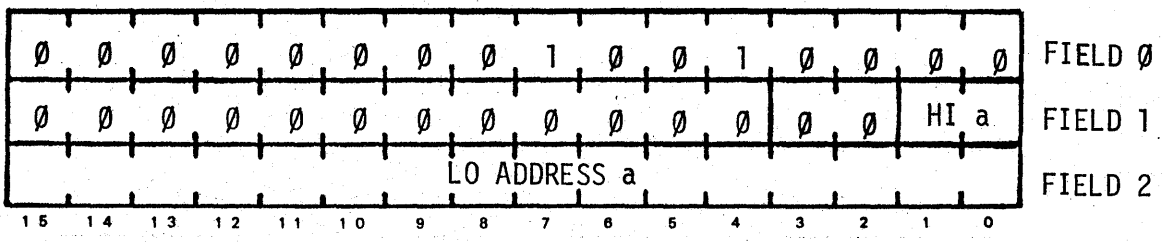
The address is sent out on the ADDRESS MULTIBUS to Data Memory. After a delay, data appears on the DATA MULTIBUS.

EXAMPLE:

```
.
.
.
RUF  FUN1
.
.
.
```

RUS a

READ UNALIGNED SECOND



OPERATION:

$AM_{1.5}$ ← UNALIGNED(ADDRESS)

$DM_{5.5}$ ← $MEM_{1.5}$(ADDRESS)

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS,DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange if enabled to AER line.

DESCRIPTION:

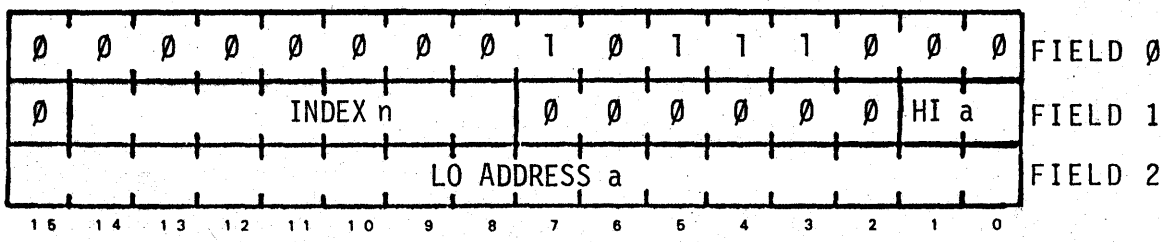
The address is sent out on the ADDRESS MULTIBUS to Data Memory. After a delay, data will appear on the DATA MULTIBUS.

EXAMPLE:

```
      .
      .
      .
RUS   FUN1
      .
      .
      .
```

RUID a,n

READ UNALIGNED INDEXED DOUBLE



OPERATION:

$AM_1 \leftarrow$ UNALIGNED(ADDRESS+$I_Ø$(INDEX))

$AM_{1.5} \leftarrow$ UNALIGNED(ADDRESS+$I_Ø$(INDEX)+1)

$DM_5 \leftarrow MEM_1$(ADDRESS+$I_Ø$(INDEX))

$DM_{5.5} \leftarrow MEM_{1.5}$(ADDRESS+$I_Ø$(INDEX)+1)

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS,DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

The address is obtained by adding the given address with the specified INDEX register. An address pair is sent out on the ADDRESS MULTIBUS Data Memory.* After a delay, data will appear on the DATA MULTIBUS.

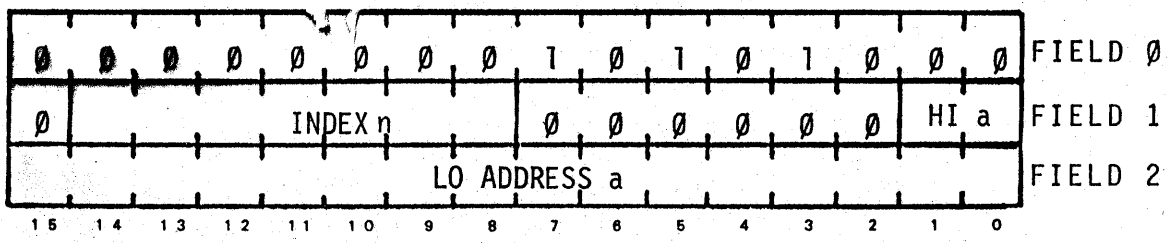
* This instruction is included primarily for maintenance purposes; a memory page access error will occur if the indexed address is not the last address of a memory page.

EXAMPLE:

```
.
.
.
RUID  FUN1,I6
.
.
.
```

RUIF a,n

READ UNALIGNED INDEXED FIRST



OPERATION:

$AM_1 \leftarrow$ UNALIGNED(ADDRESS+$I_Ø$(INDEX))

$DM_5 \leftarrow$ MEM_1(ADDRESS+$I_Ø$(INDEX))

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS,DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

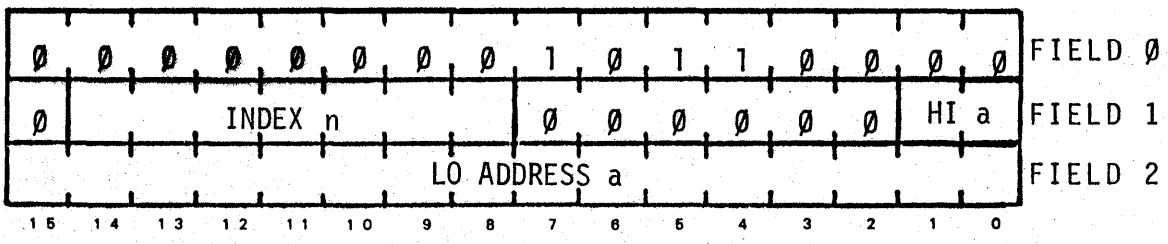
The address sent out on the ADDRESS MULTIBUS to Data Memory is obtained by adding the given address to the contents of the specified INDEX register. After a delay, data will appear on the DATA MULTIBUS.

EXAMPLE:

```
.
.
.
RUIF  FUN2,I9
.
.
.
```

RUIS a,n

READ UNALIGNED INDEXED SECOND



OPERATION:

$AM_{1.5} \leftarrow$ UNALIGNED(ADDRESS+$I_{\emptyset}$(INDEX))

$DM_{5.5} \leftarrow MEM_{1.5}$(ADDRESS+$I_{\emptyset}$(INDEX))

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS,DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

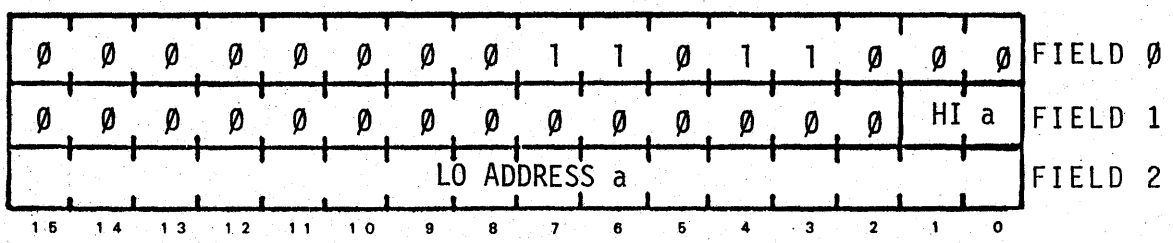
The address sent out on the ADDRESS MULTIBUS to Data Memory is obtained by adding the given address to the contents of the specified INDEX register. After a delay, data will appear on the DATA MULTIBUS.

EXAMPLE:

```
.
.
.
RUIS  A,I4
.
.
.
```

WUD a

WRITE UNALIGNED DOUBLE



OPERATION:

$AM_1 \leftarrow$ UNALIGNED(ADDRESS)

$AM_{1.5} \leftarrow$ UNALIGNED(ADDRESS+1)

MEM_5(ADDRESS) $\leftarrow DM_1$

$MEM_{5.5}$(ADDRESS+1) $\leftarrow DM_{1.5}$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange if enabled to AER line.

DESCRIPTION:

An address pair is sent out on the ADDRESS MULTIBUS to Data Memory. Data Memory takes the data pair from the DATA MULTIBUS, and, after a delay, stores the data at the specified addresses.*

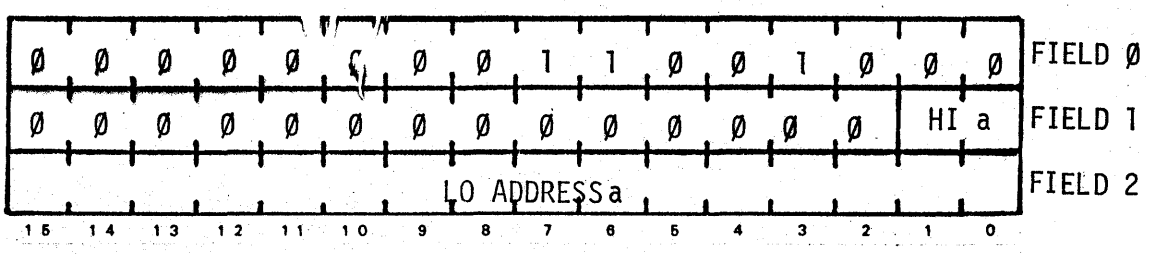
*This instruction is included primarily for maintenance purposes; a memory page access error will occur if ADDRESS is not the last address of a memory page.

EXAMPLE:

```
.
.
.
WUD  FUN1
.
.
.
```

WUF a

WRITE UNALIGNED FIRST



OPERATION:

$AM_1 \leftarrow$ UNALIGNED(ADDRESS)

MEM_5(ADDRESS) $\leftarrow DM_1$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange if enabled to AER line.

DESCRIPTION:

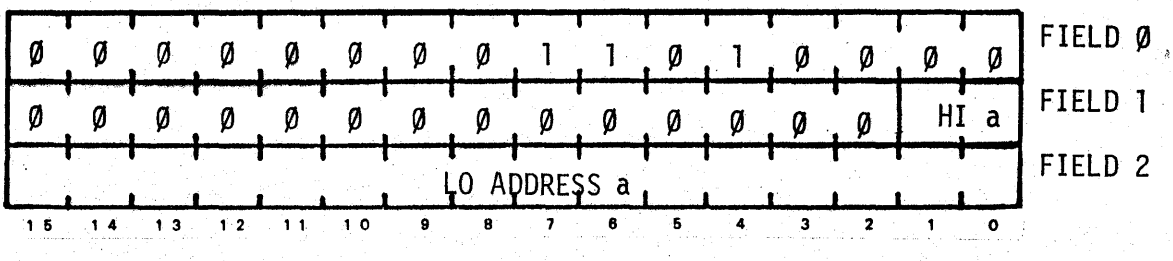
The address is sent out on the ADDRESS MULTIBUS to Data Memory. Data Memory takes the data on the DATA MULTIBUS and, after a delay, stores the data at the specified address.

EXAMPLE:

```
.
.
.
WUF FUN1
.
.
.
```

WUS a

WRITE UNALIGNED SECOND



OPERATION:

$AM_{1.5}$ ← UNALIGNED(ADDRESS)

$MEM_{5.5}$(ADDRESS) ← $DM_{1.5}$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange if enabled to AER line.

DESCRIPTION:

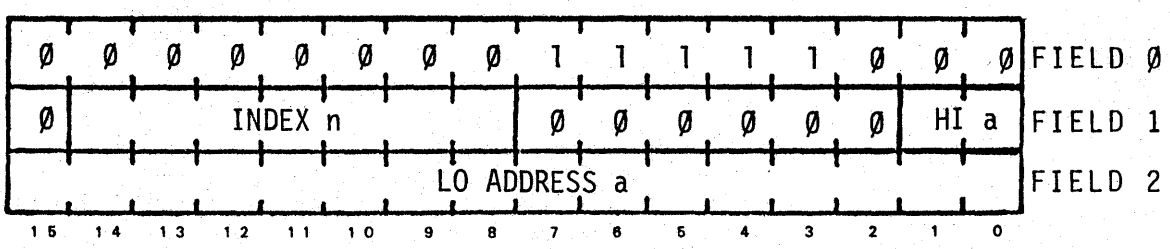
The address is sent to Data Memory on the ADDRESS MULTIBUS. Data Memory takes the data on the DATA MULTIBUS, and, after a delay, stores the data at the specified address.

EXAMPLE:

```
.
.
.
WUS FUN1
.
.
.
```

WUID a,n

WRITE UNALIGNED INDEXED DOUBLE



OPERATION:

$AM_1 \leftarrow UNALIGNED(ADDRESS+I_Ø(INDEX))$

$AM_{1.5} \leftarrow UNALIGNED(ADDRESS+I_Ø(INDEX)+1)$

$MEM_5(ADDRESS+I_Ø(INDEX)) \leftarrow DM_1$

$MEM_{5.5}(ADDRESS+I_Ø(INDEX)+1) \leftarrow DM_{1.5}$

ERROR CONDITION(S):"

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

The address sent out on the ADDRESS MULTIBUS to DATA MEMORY is obtained by adding the given address to the contents of the specified INDEX register. DATA MEMORY takes the data pair on the DATA MULTIBUS, and, after a delay, stores the data at the specified addresses.*

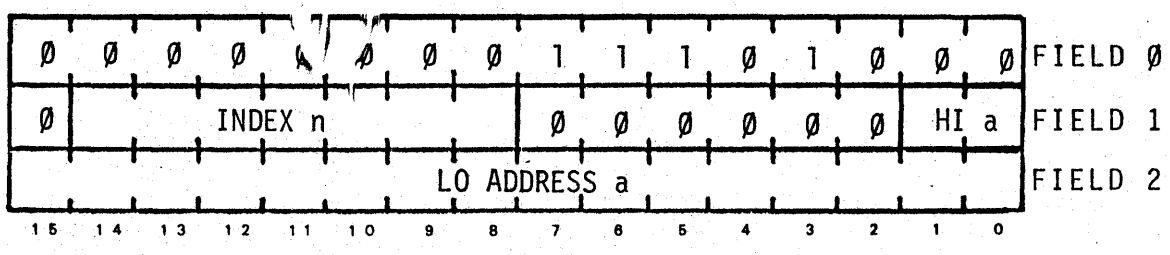
*This instruction is included primarily for maintenance purposes; a memory page access error will occur if the indexed address is not the last address of a memory page.

EXAMPLE:

```
.
.
.
WUID FUN1,I7
.
.
.
```

WUIF a,n

WRITE UNALIGNED INDIRECT FIRST



OPERATION:

$AM_1 \leftarrow UNALIGNED(ADDRESS+I_{\emptyset}(INDEX))$

$MEM_5(ADDRESS + I_{\emptyset}(INDEX)) \leftarrow DM_1$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

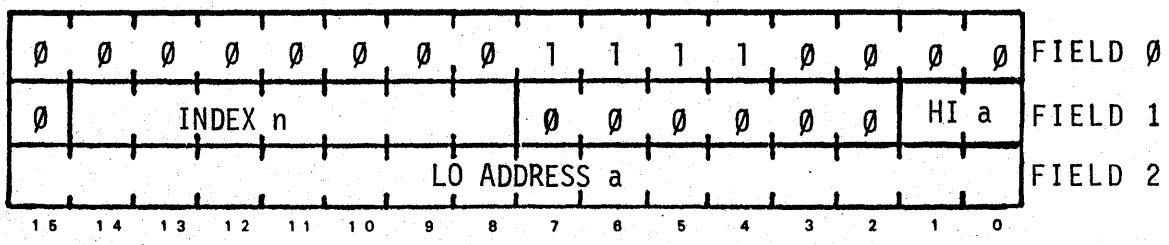
The address sent out on the ADDRESS MULTIBUS to Data Memory is obtained by adding the given address to the contents of the specified INDEX register. Data Memory takes the data on the DATA MULTIBUS, and, after a delay, stores the data at the specified address.

EXAMPLE:

```
.
.
.
WUIF FUN1,I8
.
.
.
```

WUIS a,n

WRITE UNALIGNED INDEXED SECOND



OPERATION:

$AM_{1.5} \leftarrow$ UNALIGNED(ADDRESS+$I_{\emptyset}$(INDEX))

$MEM_{5.5}$(ADDRESS + $I_{\emptyset}$ (INDEX)) $\leftarrow$ $DM_{1.5}$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

The address sent out on the ADDRESS MULTIBUS to Data Memory is obtained by adding the given address to the contents of the specified INDEX register. Data Memory takes the data on the DATA MULTIBUS, and, after a delay, stores the data at the specified address.

EXAMPLE:

```
.
.
.
WUIS FUN1,I5
.
.
.
```

7.2.3.3 Aligned Read/Write Instructions

In general, an address alignment network is a hardware device which maps the program address space into the physical address space of the memory. The purpose of an address alignment network in the AD 10 is to allow data transfers to or from Data Memory to be performed for a specific type of application in such a way as to take maximum advantage (from the standpoint of data transfer rate) of the overlapped page architecture of the Data Memory.

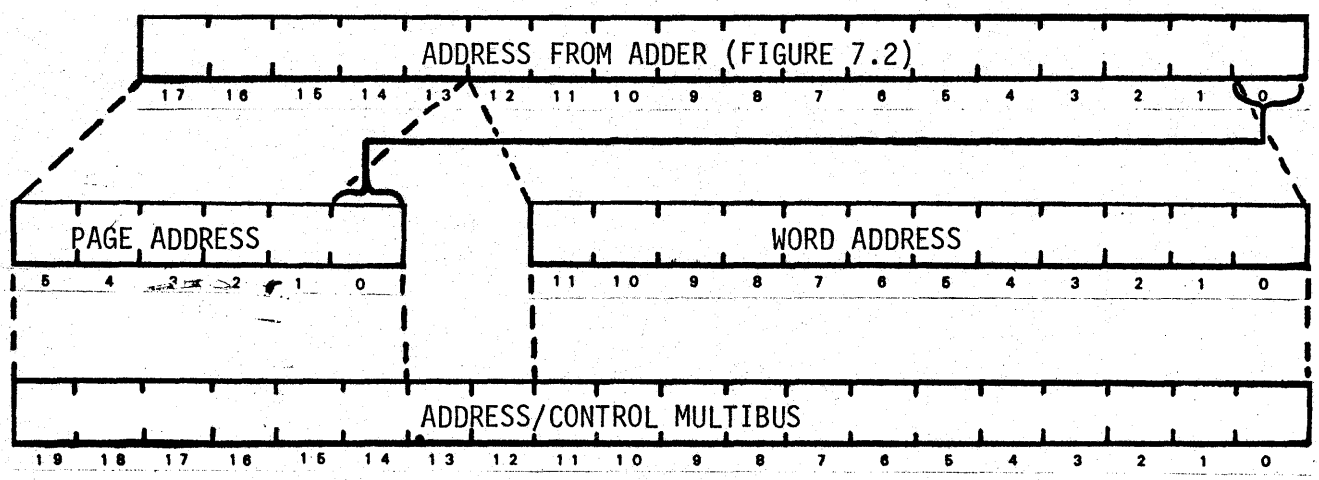
The address alignment network in MAP is designed to map a linear array in program address space into Data Memory such that any pair of words with consecutive addresses (i.e., addresses ADDR and ADDR+1) in program address space will be located in adjacent pages in Data Memory.

If a MAP read or write instruction specifies an aligned address, the address alignment network takes the program address, ADDR, (see Figure 7.2) and converts it to a Data Memory address which it outputs to the ADDRESS/CONTROL MULTIBUS. The conversion performed by the address alignment network on ADDR is:

$$AM_1(14{:}19) \leftarrow ADDR_{\emptyset}(\emptyset, 13{:}17) \quad \text{(PAGE ADDRESS)}$$

$$AM_1(\emptyset{:}11) \leftarrow ADDR_{\emptyset}(1{:}12) \quad \text{(WORD ADDRESS)}$$

The operation of the address alignment network for an aligned read or write instruction is illustrated in the diagram below.

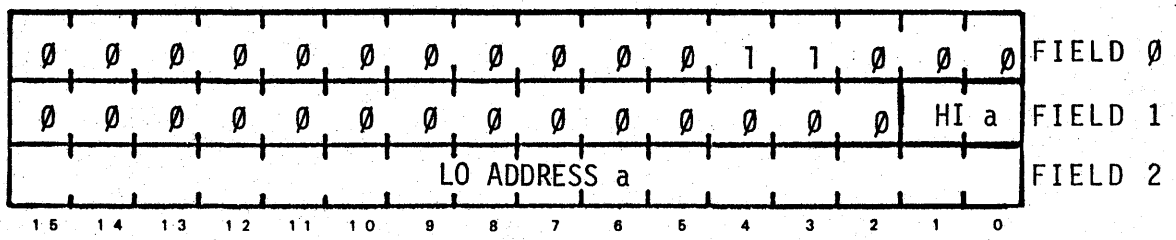


The aligned address read/write instructions are:

RAD a	READ ALIGNED DOUBLE
RAF a	READ ALIGNED FIRST
RAS a	READ ALIGNED SECOND
RAID a,n	READ ALIGNED INDEXED DOUBLE
RAIF a,n	READ ALIGNED INDEXED FIRST
RAIS a,n	READ ALIGNED INDEXED SECOND
WAD a	WRITE ALIGNED DOUBLE
WAF a	WRITE ALIGNED FIRST
WAS a	WRITE ALIGNED SECOND
WAID a,n	WRITE ALIGNED INDEXED DOUBLE
WAIF a,n	WRITE ALIGNED INDEXED FIRST
WAIS a,n	WRITE ALIGNED INDEXED SECOND

RAD a

READ ALIGNED DOUBLE



OPERATION:

$AM_1 \leftarrow$ ALIGNED(ADDRESS)

$AM_{1.5} \leftarrow$ ALIGNED(ADDRESS+1)

$DM_5 \leftarrow MEM_1$(ADDRESS)

$DM_{5.5} \leftarrow MEM_{1.5}$(ADDRESS+1)

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS,DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange if enabled to AER line.

DESCRIPTION:

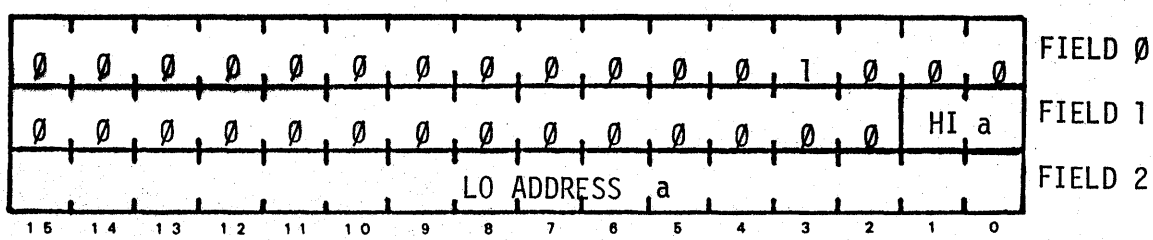
A pair of addresses is sent on the ADDRESS MULTIBUS to Data Memory. After a delay, the data will appear on the DATA MULTIBUS.

EXAMPLE:

```
.
.
.
RAD  FUN1      !GET TWO FUNCTION VALUES
.
.
.
```

RAF a

READ ALIGNED FIRST



OPERATION:

AM_1 ← ALIGNED(ADDRESS)

DM_5 ← MEM_1(ADDRESS)

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS,DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange if enabled to AER line.

DESCRIPTION:

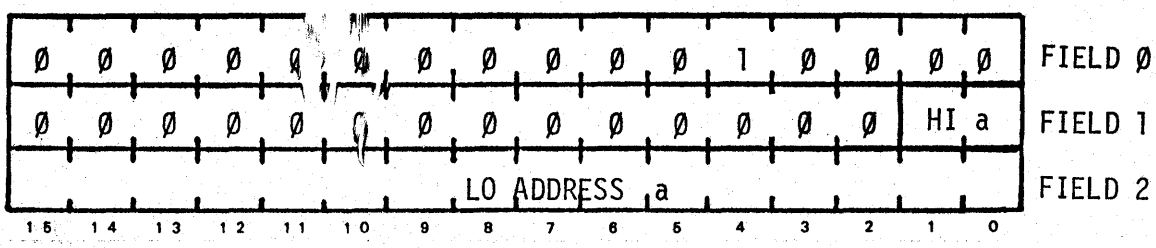
The address is sent on the ADDRESS MULTIBUS to Data Memory. After a delay, the data will appear on the DATA MULTIBUS.

EXAMPLE:

```
.
.
.
RAF FUN1      !GET DATA POINT
.
.
.
```

RAS a

READ ALIGNED SECOND



OPERATION:

$AM_{1.5} \leftarrow$ ALIGNED(ADDRESS)

$DM_{5.5} \leftarrow MEM_{1.5}$(ADDRESS)

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS,DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange if enabled to AER line.

DESCRIPTION:

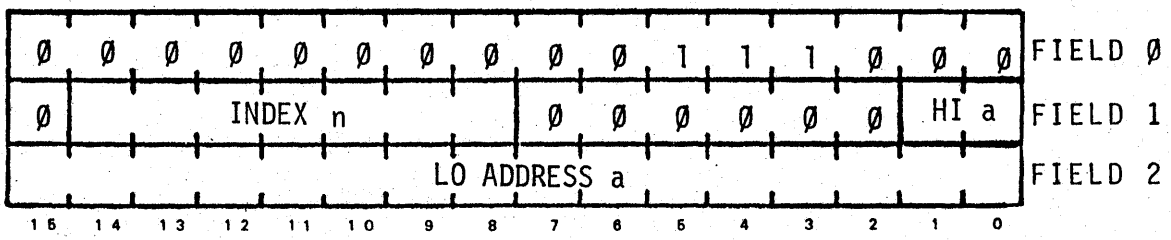
The address is sent out on the ADDRESS MULTIBUS to Data Memory. After a delay, data will appear on the DATA MULTIBUS.

EXAMPLE:

```
.
.
.
RAS FUN1      !GET DATA VALUE
.
.
.
```

RAID a,n

READ ALIGNED INDEXED DOUBLE



OPERATION:

$AM_1 \leftarrow$ ALIGNED(ADDRESS+$I_Ø$(INDEX))

$AM_{1.5} \leftarrow$ ALIGNED(ADDRESS+$I_Ø$(INDEX)+1)

$DM_5 \leftarrow MEM_1$(ADDRESS+$I_Ø$(INDEX))

$DM_{5.5} \leftarrow MEM_{1.5}$(ADDRESS+$I_Ø$(INDEX)+1)

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

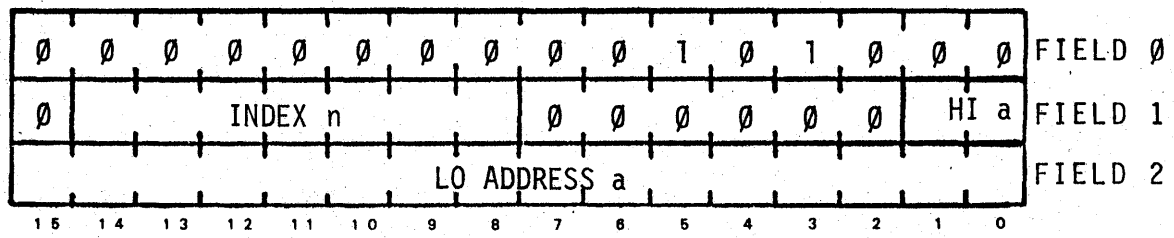
The address sent out on the ADDRESS MULTIBUS to Data Memory is obtained by adding the given address to the contents of the specified INDEX register. After a delay, data will appear on the DATA MULTIBUS.

EXAMPLE:

```
.
.
.
RAID FUN1,IØ      !GET FUNCTION PAIR
.
.
.
```

RAIF a,n

READ ALIGNED INDEXED FIRST



OPERATION:

$AM_1 \leftarrow ALIGNED(ADDRESS+I_Ø(INDEX))$

$DM_5 \leftarrow MEM_1(ADDRESS+I_Ø(INDEX))$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

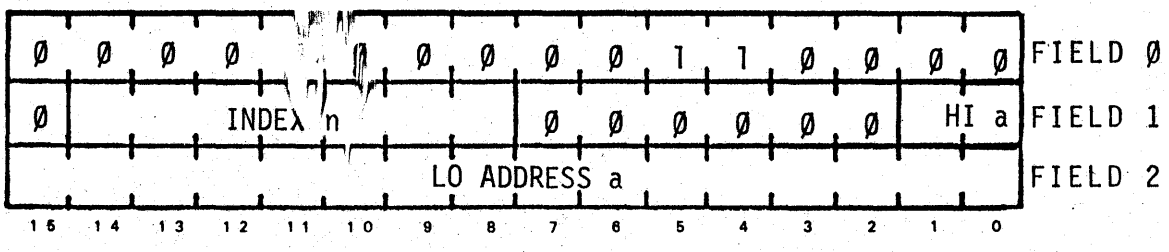
The address sent out on the ADDRESS MULTIBUS to Data Memory is obtained by adding the given address to the contents of the specified INDEX register. The address is sent out on the ADDRESS MULTIBUS to Data Memory. After a delay, data will appear on the DATA MULTIBUS.

EXAMPLE:

```
.
.
.
RAIF FUN1,I2      !GET DATA VALUE
.
.
.
```

RAIS a,n

READ ALIGNED INDEXED SECOND



OPERATION:

$AM_{1.5} \leftarrow$ ALIGNED(ADDRESS+$I_{Ø}$(INDEX))

$DM_{5.5} \leftarrow MEM_{1.5}$(ADDRESS+$I_{Ø}$(INDEX))

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

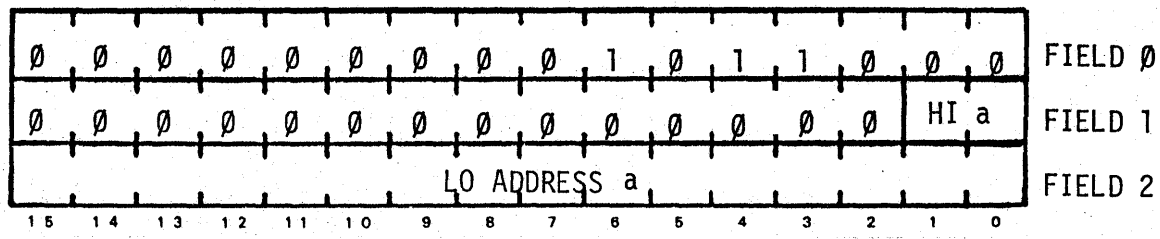
The address sent out on the ADDRESS MULTIBUS to Data Memory is obtained by adding the given address to the contents of the specified INDEX register. After a delay, data will appear on the DATA MULTIBUS.

EXAMPLE:

```
.
.
.
RAIS FUN1,I3      !GET DATA VALUE
.
.
.
```

WAD a

WRITE ALIGNED DOUBLE



OPERATION:

AM_1 ← ALIGNED(ADDRESS)

$AM_{1.5}$ ← ALIGNED(ADDRESS+1)

MEM_5(ADDRESS) ← DM_1

$MEM_{5.5}$(ADDRESS+1) ← $DM_{1.5}$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange if enabled to AER line.

DESCRIPTION:

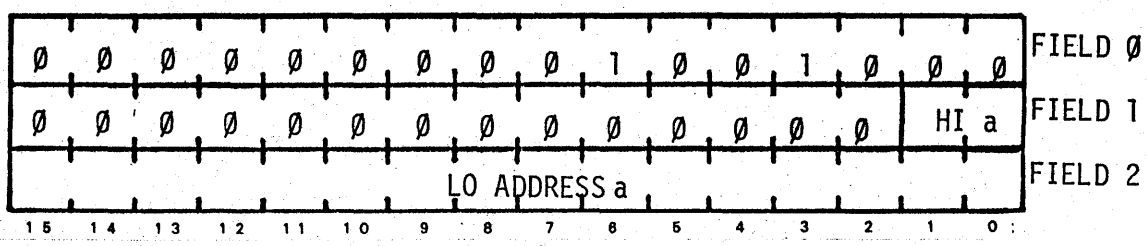
An address pair is sent out on the ADDRESS MULTIBUS to Data Memory. Data Memory takes the data pair on the DATA MULTIBUS, and, after a delay, stores the data at the specified addresses.

EXAMPLE:

```
      .
      .
      .
WAD   FUNOUT1
      .
      .
      .
```

WAF a

WRITE ALIGNED FIRST



OPERATION:

$AM_1 \leftarrow$ ALIGNED(ADDRESS)

MEM_5(ADDRESS) $\leftarrow DM_1$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous address.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange if enabled to AER line.

DESCRIPTION:

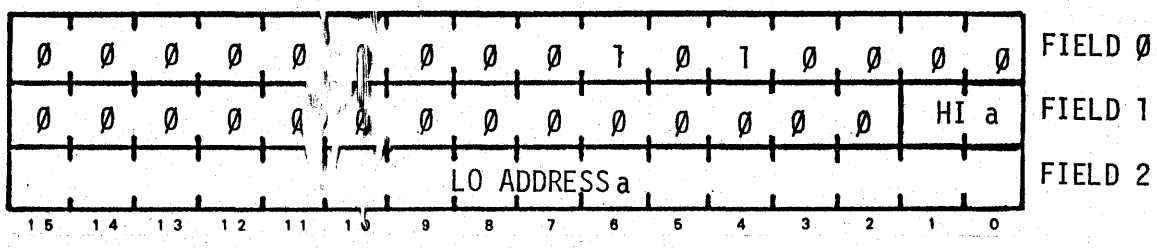
The address is sent out to DATA MEMORY on the ADDRESS MULTIBUS. Data Memory takes the data on the DATA MULTIBUS, and, after a delay, stores the data at the specified address.

EXAMPLE:

```
.
.
.
WAF  FUN2
.
.
.
```

WAS a

WRITE ALIGNED SECOND



OPERATION:

$AM_{1.5}$ ← ALIGNED(ADDRESS)

$MEM_{5.5}$(ADDRESS) ← $DM_{1.5}$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange, if enabled to AER line.

DESCRIPTION:

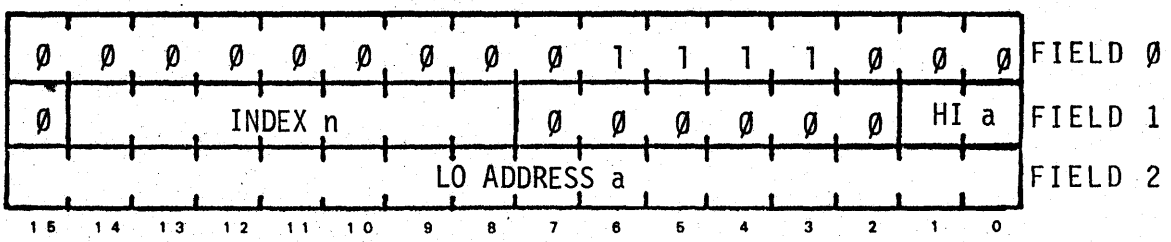
The address is sent out on the ADDRESS MULTIBUS to Data Memory. Data Memory takes the data on the DATA MULTIBUS, and, after a delay, stores the data at the specified address.

EXAMPLE:

```
.
.
.
WAS FUN1
.
.
.
```

WAID a,n

WRITE ALIGNED INDEXED DOUBLE



OPERATION:

$AM_1 \leftarrow ALIGNED(ADDRESS+I_Ø(INDEX))$

$AM_{1.5} \leftarrow ALIGNED(ADDRESS+I_Ø(INDEX)+1)$

$MEM_5(ADDRESS+I_Ø(INDEX)) \leftarrow DM_1$

$MEM_{5.5}(ADDRESS+I_Ø(INDEX)+1) \leftarrow DM_{1.5}$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

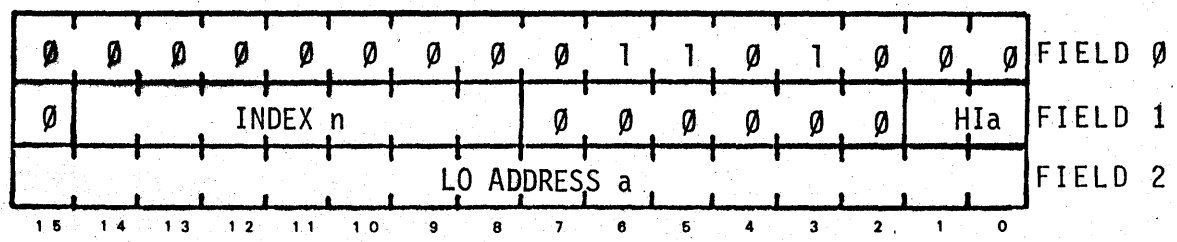
The address pair sent out on the ADDRESS MULTIBUS to Data Memory is obtained by adding the given address to the contents of the specified INDEX register. Data Memory takes the data pair on the DATA MULTIBUS, and, after a delay, stores the data at the specified addresses.

EXAMPLE:

```
.
.
.
WAID FUN1,I6
.
.
.
```

WAIF a,n

WRITE ALIGNED INDEXED FIRST



OPERATION:

$AM_1 \leftarrow ALIGNED(ADDRESS+I_Ø(INDEX))$

$MEM_5\ (ADDRESS+I_Ø(INDEX)) \leftarrow DM_1$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

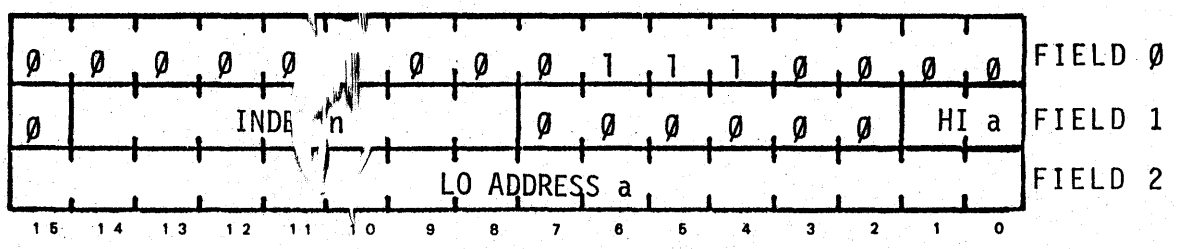
The address sent out on the ADDRESS MULTIBUS to Data Memory is obtained by adding the given address to the contents of the specified INDEX register. Data Memory takes the data on the DATA MULTIBUS, and, after a delay, stores the data at the specified address.

EXAMPLE:

```
.
.
.
WAIF FUN1,I2
.
.
.
```

WAIS a,n

WRITE ALIGNED INDEXED SECOND



OPERATION:

$AM_{1.5} \leftarrow ALIGNED(ADDRESS+I_{\emptyset}(INDEX))$

$MEM_{5.5}(ADDRESS+I_{\emptyset}(INDEX)) \leftarrow DM_{1.5}$

ERROR CONDITION(S):

1. Memory page accessed too soon after previous access.
2. BUS conflict (ADDRESS, DATA), if there is other data on the AM or the DM in the above specified BUS cycles.
3. Page address overrange/underrange if enabled to AER line.

DESCRIPTION:

The address sent out on the ADDRESS MULTIBUS to Data Memory is obtained by adding the given address to the contents of the specified INDEX register. Data Memory takes the data on the DATA MULTIBUS, and, after a delay, stores the data at the specified address.

EXAMPLE:

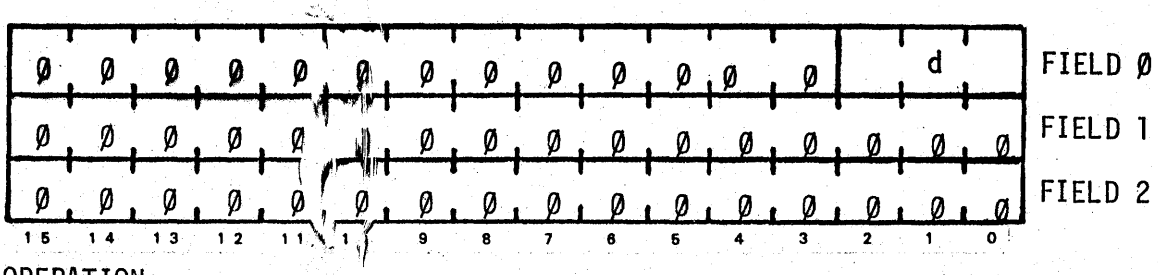
```
.
.
.
WAIS  FUN3,I1Ø
.
.
.
```

7.2.4 PAUSE/NOP INSTRUCTIONS

The PAUSE instruction may be microprogrammed with any other MAP instruction or it may be issued as a stand-alone instruction. The NOP instruction is only used as a stand-alone instruction. The NOP and PAUSE instructions are defined below.

PAUSE d

PAUSE FOR d INSTRUCTION CYCLES



OPERATION:

None

ERROR CONDITION(S):

None

DESCRIPTION:

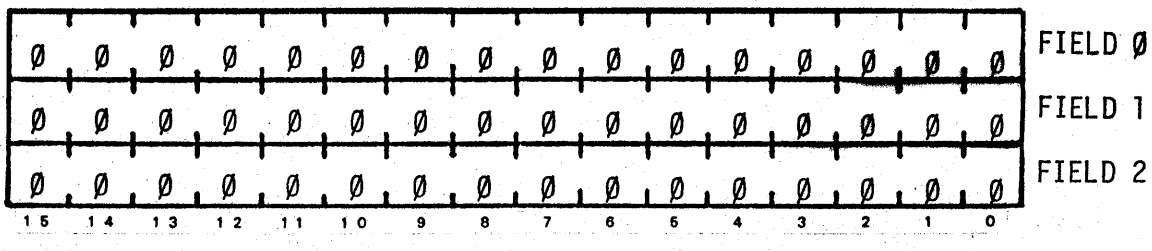
Suspend execution for d instruction cycles following the PAUSE instruction. This has the same effect as (d+1) NOP's.

EXAMPLE:

```
.
.
RAD  FUN1; PAUSE 4    !READ VALUES & WAIT FOR MEMORY
.
.
.
.
```

NOP

NO OPERATION



OPERATION:

None

ERROR CONDITION(S):

None

DESCRIPTION:

Suspends execution for one instruction cycle.

EXAMPLE:

```
NOP
```