Technical Report 70-135
NGR-21-002-206

September 1970

A Microprogrammed Allocating-loader

by

Yaohan Chu

UNIVERSITY OF MARYLAND

COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND

Technical Report 70-135 September 1970
NGR-21-002-206

A Microprogrammed Allocating-loader

by

Yaohan Chu

This research was supported by Grant NGR-21-002-206 from the National Aeronautics and Space Administration to the Computer Science Center of the University of Maryland.

Table of Contents

Abstract

The report describes an organization for allocating and loading an executable code relative to an origin into the main memory by a 10-word microprogram. The main memory is organized into pages and their activities are kept tract of by a small-capacity but very fast associative memory. During allocation and loading, the code is not modified in any manner. The physical addresses of the instructions and operands of the code are not known until the time of execution of the code. This is called the dynamic addressing. The addressing technique, the sequence chart, and the description of the microprogram are presented in great details in the Computer Design Language.

A Microprogrammed Allocating Loader

When a program or a segment of a program is to be executed by a processor, the instructions and data of the program must first be loaded into the main memory. Before the loading, the loader must assign an absolute address to each word of the program and modify if necessary the address in the address field of the instruction accordingly. Once the program is loaded, relocation of the program is undesirable because of the task of address modification. This report describes a hardware loader which employs an addressing scheme in such a way that no address modification is required as long as the program or the segment of a program has been assembled relative to an origin. Since the loader loads a program or a segment of the program into whatever memory locations are available at the loading time, it is referred to as a dynamic loader or more descriptively a dynamic allocating-loader.

The dynamic loader faces a number of problems. It must know which memory locations are available for allocation and loading. Are the available locations large enough? If so, how can they conveniently be allocated and loaded and still be protected from other programs in the memory, since these available locations are usually scattered rather than contiguous? How is the loader to handle the allocation and loading of an additional segment of a program if the segment is needed during the execution. In addition, there is the problem of storage release, the problem of segment sharing, and the problem of segment relocation. This section describes a solution to some of these problems.

In the following, the program is assumed to consist of one or more segments, and the origin of each program or of the segment of a program is assumed to be zero.

1. Loader Organization

The organization of the dynamic loader is shown in Fig. 1 where there are a main memory, an associative memory, thirteen registers and a memory bus. The registers are: address register AR, buffer register B, next-instruction address register NI, current-segment-number register CSN, register FPA for storing the first page-address of the available segment, register NSN for storing segment number of the new segment, register NSPN for storing page number of the new segment, page counter PC, argument register A, mask register K, buffer register D, and control registers MR and MW. The first two registers are associated with the main memory M and the last five with the associative memory AM. The memory bus is attached to buffer register B.

1.1 Configuration

The above-mentioned memories and registers are described by the following CDL statements.

```
Comment, configuration of a dynamic loader                              (1)
Register,       AR(0-14),       $address register of main memory
                B(0-47),        $buffer register of main memory
                NI(0-14),       $next-instruction address register
                CSN(0-4),       $current segment-number register
                FPA(0-7),       $first page address register
                PC(0-7),        $page counter
                NSN(0-4),       $new segment number register
                NSPN(0-4),      $new segment page number register
                A(0-31),        $argument register
                K(0-31),        $mask register
                D(0-31),        $buffer register
                MR,             $match-read control bit
```

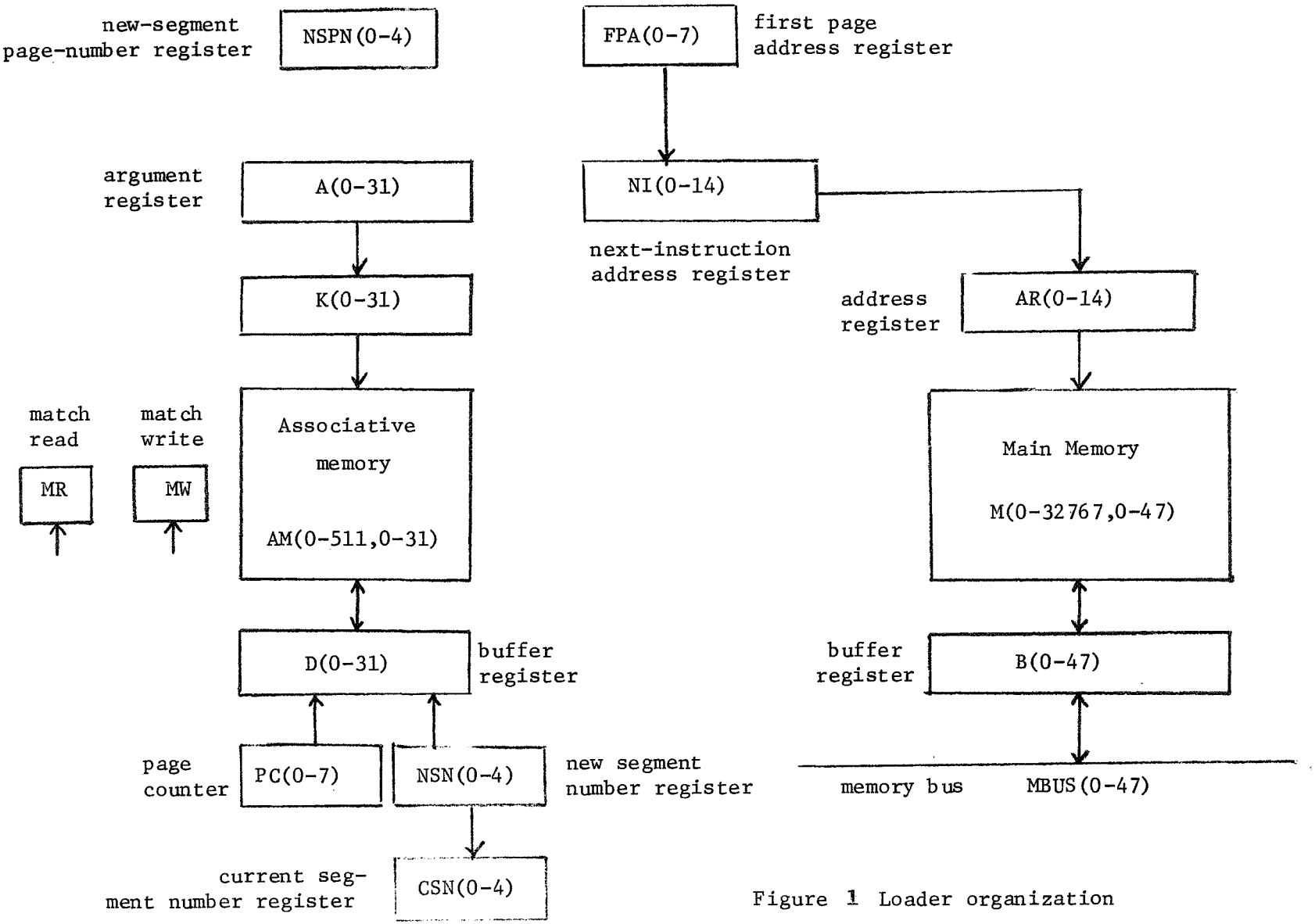


Figure 1 Loader organization

```
                MW,                          $match-write control bit
Memory,         M(AR)=M(0-32767,0-47),       $main memory
Bus,            MBUS(0-47),                  $main memory bus
Asso-memory,    AM(A,K,D)=AM(0-511,0-31),    $associative memory
Subregister,    A(X,Y,Z,SN,S)=A(0-7,8-15,16-23,24-28,29-31),
                K(X,Y,Z,SN,S)=K(0-7,8-15,16-23,24-28,29-31),
                D(X,Y,Z,SN,S)=D(0-7,8-15,16-23,24-28,29-31),
                NI(PA,LA)=NI(0-7,8-14),
                AR(PA,LA)=AR(0-7,8-14),
```

1.2 Main Memory

The main memory is conventional except that it is divided into pages. The characteristics of main memory M are shown in Table 1. The main memory has 256 pages; each page consists of 128 48-bit words located contiguously. The memory address consists of two parts, an 8-bit page address PA for each of the 256 pages and a 7-bit line address LA for each of the 128 words in a page. The pages of the main memory and the two parts of the main memory address register are shown in Fig. 8.11.

1.3 Associative Memory

The characteristics of the associative memory AM are shown in Table 2. The associative memory consists of 256 32-bit words. As illustrated in Fig. 2, each word points to one and only one page in the main memory. Each word consists of 5 fields; each field is maskable by the mask in mask register K. Each word in the associative memory is unique; therefore, multiple matches do not occur during matching. The associative memory is capable of performing two operations, match-read and match-write. The cycle time of the associative memory is 1/4 of the main-memory cycle time.

Table 1 Characteristics of the Main Memory

Characteristic	Description
word length	48 bits
page size	128 contiguous words
capacity	256 pages or 32,768 words
page address	8 bits
line address	7 bits
total address	15 bits
cycle time	1 microsecond

Table 2 Characteristics of the Associative Memory

Characteristic	Description
word length	32 bits
capacity	256 words
associativity	full associative and maskable
matching	single match only
operations	(a) match-read (b) match-write
cycle time	1/4 microsecond for match-read or write

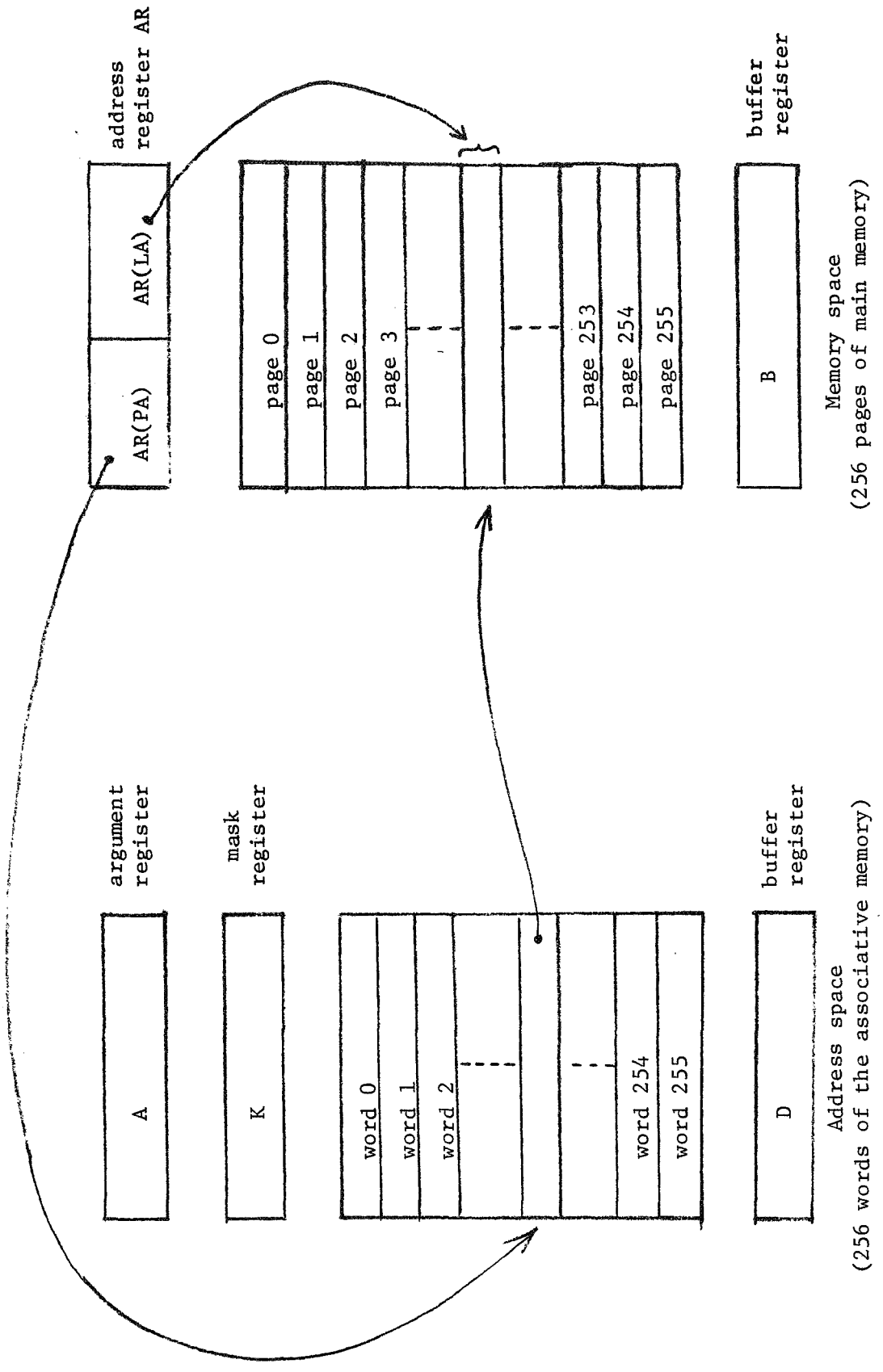


Fig. 2 Mapping from the address space to memory space

1.4 Word Formats

The word format of the associative memory word is shown in Fig. 3. There are three 8-bit fields for page addresses X, Y, and Z; a 5-bit field for segment number; and a 3-bit field for status indication. Page address X indicates the location of a page in the main memory, and page address Y the location of the next page. By means of 256 pairs of page addresses X and Y, the 256 pages of the main memory can be linked into one segment. By means of the 5-bit segment number, the 256 pages can be linked into as many as 32 segments. Page address Z indicates the order of the page in a segment for use in operand address fetch as will be further described. The status field is used to indicate the status of the page; an example of the status designation is shown in Table 3.

1.5 An Example

An associative memory map is exemplified in Fig. 4, where the main memory has only 12 pages. Program P1 (or segment P1 of a program) is loaded in three pages with page addresses 1, 5 and 9; these pages are linked into a linked list of 1-5-9 and assigned with a segment number 1. The first page of program P1 is located at page address 1. The segment number, the first page address and the program name P1 are stored in the associative memory and shown at the fourth from the last line of the associative memory map in Fig. 4. The page at page address 9 is the last page of segment number 1; this is indicated by the asterisk in the page-address Y field of the 10th line in Fig. 4. Similarly, programs P2, P3 and P4 consist of the linked lists of pages, 6-4-8, 3-11-0, and 2-10-7 with segment numbers 2, 3 and 4 respectively. By means of the segment status designations in Table 3, segment number 4 contains the pages available for loading (referred to as available segment); segment number 1 is a loaded segment; segment number 2 has been released and is ready to become a part of available segment; segment number 3 has been permanently loaded (for such use as the resident part of the supervisor program.)

Table 3 An Example of Status Designation

Status Field	Designation
000	loaded
001	released
010	reserved
011	permanently loaded
100	available for loading
101	shared
110	segment number register

page address X	page address Y	page address Z	segment number	status
8 bits	8 bits	8 bits	5 bits	3 bits

Fig. 3 Word Format of the Associative Memory

Page Address X	Page Address Y	Page Address Z	Segment Number	Status	
0	*	2	3	011	
1	5	0	1	000	
2	10	0	4	100	
3	11	0	3	011	
4	8	1	2	001	
5	9	1	1	000	
6	4	0	2	001	
7	*	2	4	100	
8	*	2	2	001	
9	*	2	1	000	
10	7	1	4	100	
11	0	1	3	011	
1	P1		1	110	segment number registers
6	P2		2	110	
3	P3		3	110	
2	P4		4	110	

Fig. 4 An Example of Associative Memory Map

The last four lines in the associative memory map of Fig. 4 represent associative memory words which function as segment-number registers as indicated by 110 in their status fields. As mentioned, the first page addresses of the segments are stored in the address X fields. The fields for page addresses Y and Z are used to store the return addresses as will be further described. The segment number and status fields store the segment number and segment status as before.

It should be noted that the order of the words in Fig. 4 is not necessarily the actual order in the associative memory; in fact, the actual order may not be known.

2. Allocation and Loading

Before the loader is initiated for loading a program segment to the main memory, the supervisor program determines that the program-segment size is smaller than the available-segment size, and then stores the number of pages of the new segment into the new-segment page-number register NSPN and the segment-number assigned to the new segment into the new-segment-number register NSN. The first page address of the available segment is in the first page address register FPA.

The loading process consists of two parts. The first part allocates the available pages of the available segment to the new segment and stores the allocation in the form of the previously-described linear list in the associative memory. The second part actually loads the program segment from a backing storage such as a disk memory into the main memory.

The loading process is shown in the sequence chart of Fig. 5. The sequence is initialized by resetting page counter PC to 0 and setting up an argument in register A and a mask in register K. The argument consists of the value in register FPA as page address X, the value of 9 as the chosen segment number for the available segment, and the value of 4 as the status designation. The mask is set up with page addresses Y and Z masked out. Next-instruction address register NI is initialized by taking the value in register FPA as the page address and the value of 0 as the line address. Current segment-number register, CSN is initialized by taking the value in new segment-number register NSN as the current segment-number. These two register NI and CSN are initialized for the second part of the loading process.

2.1 Allocation

The first part of allocation now begins by decrementing the contents of register NSPN by 1. Associative memory AM match-reads the first word of the avail-

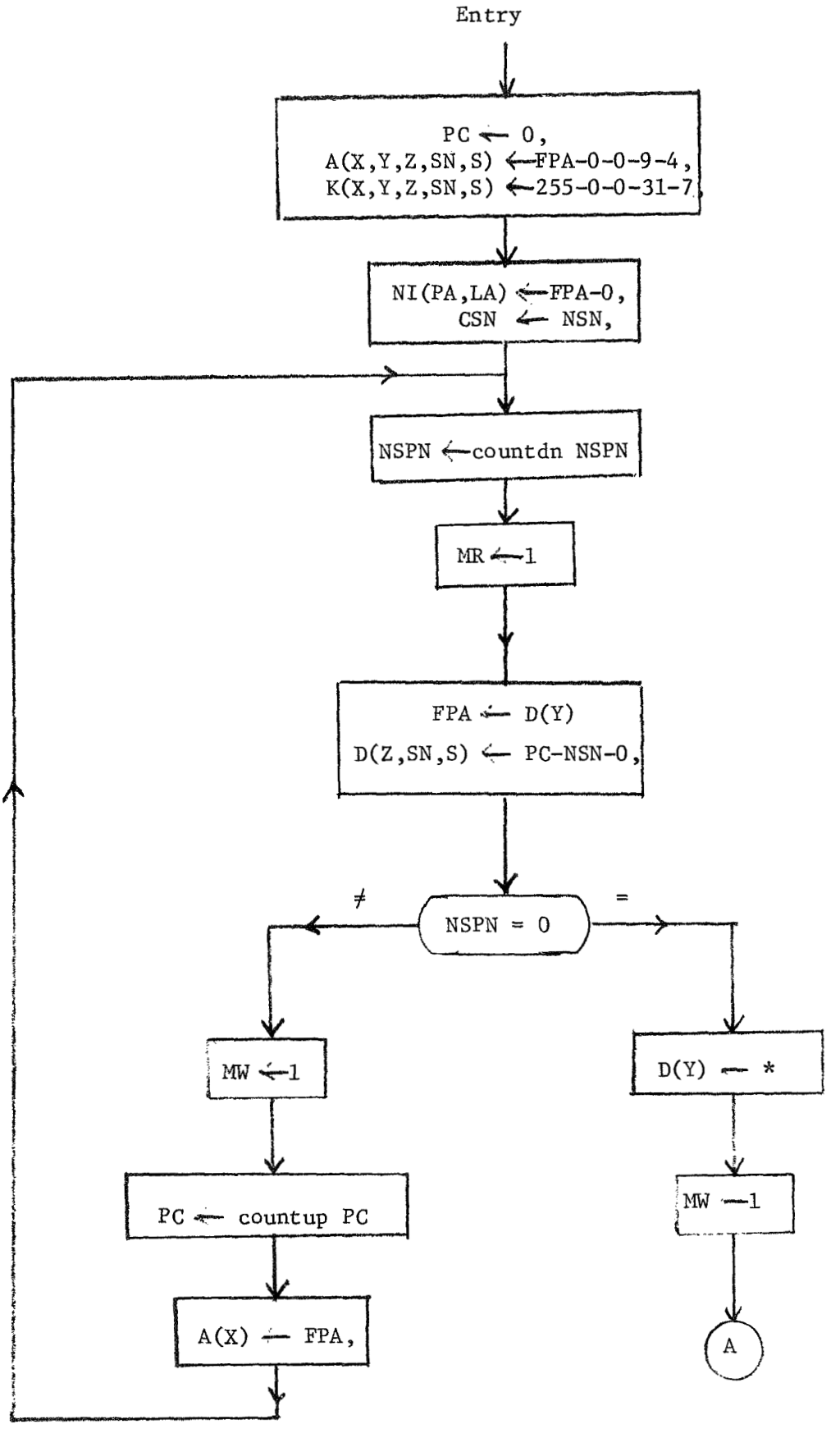


Fig. 5 Sequence chart of the dynamic loader

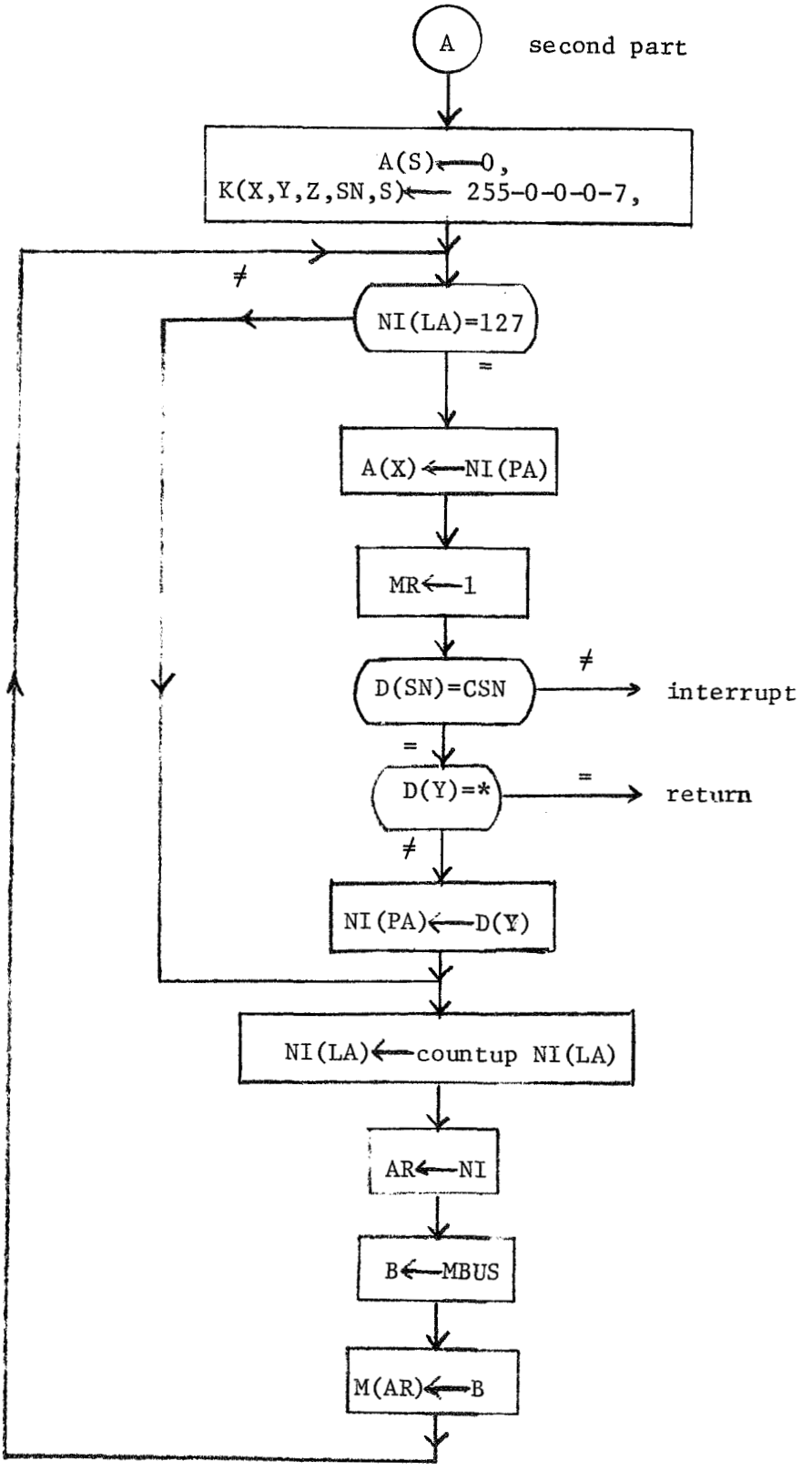


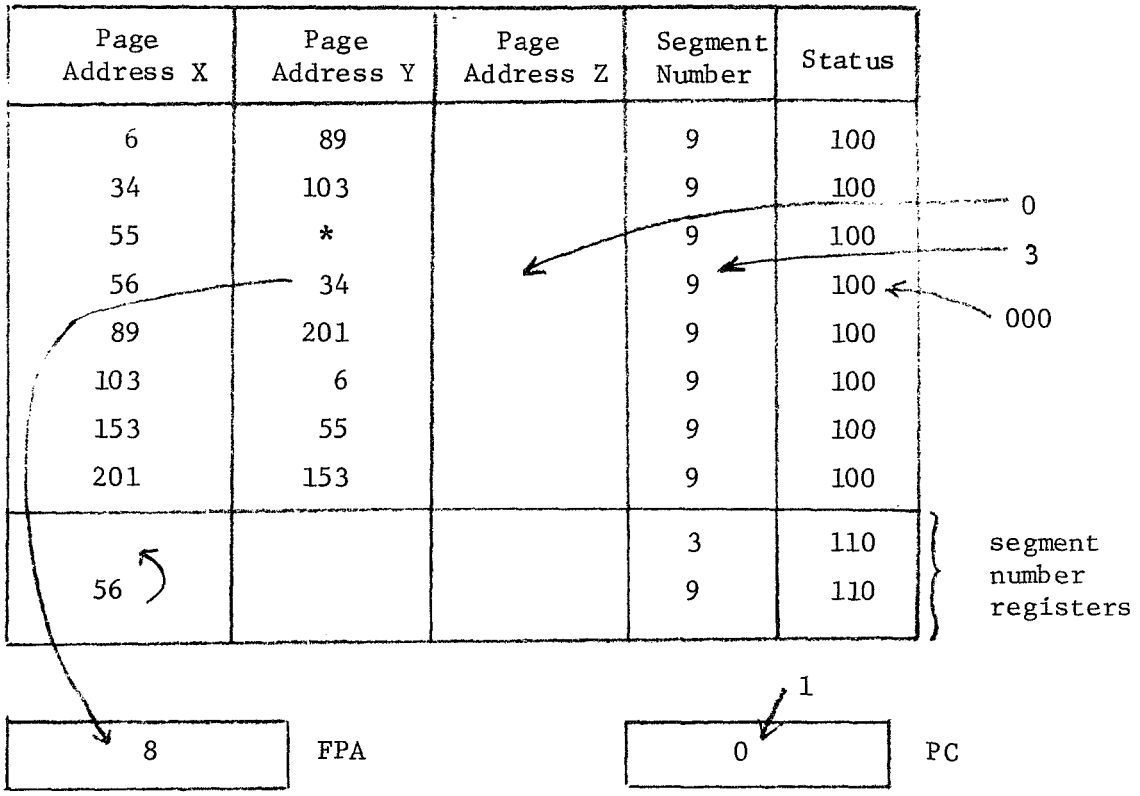
Fig. 5 (continued)

able segment out of the associative memory into buffer register D. Page address Y in register D is transferred to the first page-address register PFA. Next, a new word is formed in register D. This new word contains page address Z which is the page number in page counter PC, segment number SN taken from register NSN, and the status designation of 0 (loaded status). Subregister D(Y) of the new word should contain the segment-termination symbol "*" if register NSPN reaches 0. In either case, the new word is next match-written into the associative memory. If the new-segment page-number register NSPN were nonzero, page counter PC would be incremented by 1 and the argument would be set up again to match-read the next linked word out of the associative memory and to match-write another new word into the associative memory. This process continues until register NSPN becomes 0 (i.e. the last page of the new segment is reached). At this time, the first part of allocation is completed.

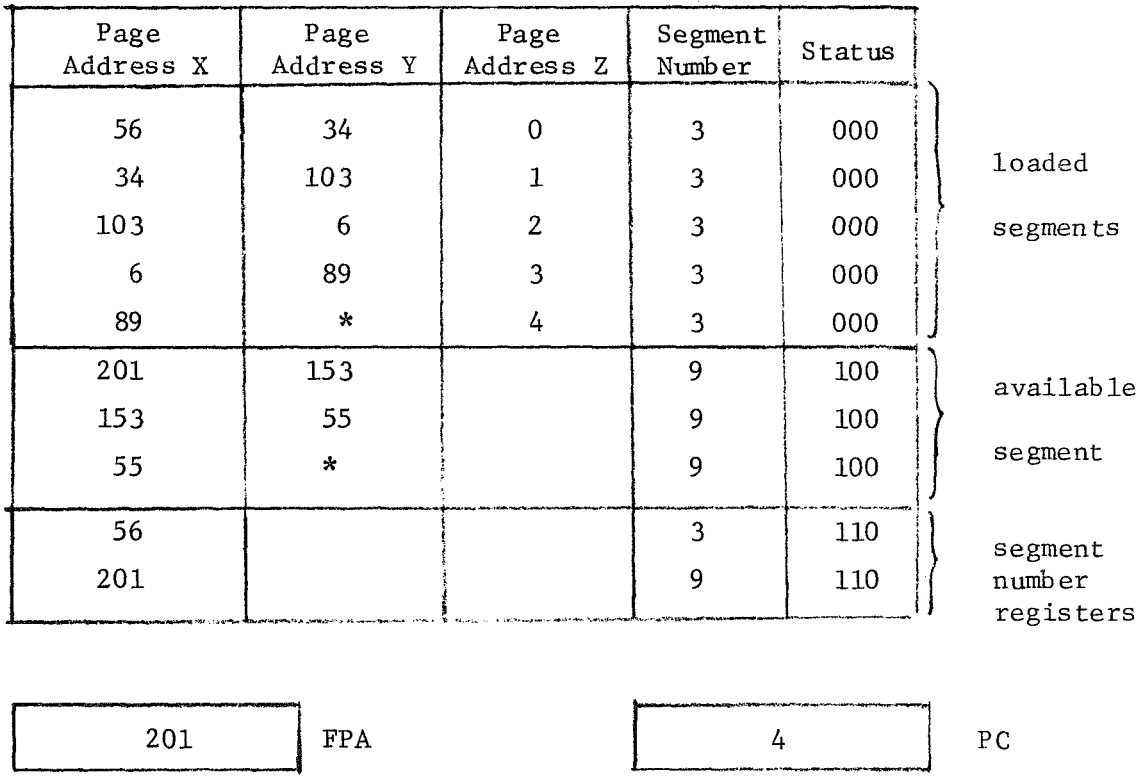
As an example, a segment of 600 words of instructions and data is to be allocated. Let 3 be the assigned segment number, and the available segment (segment number 9) having the following sequence of pages: 56-34-103-6-89-201-153-55. The associative-memory map before allocation is shown in Fig. 6(a). The first 128 words of the segment are assigned to page address 56; the required operations for this assignment are indicated in Fig. 6(a). The second 128 words are assigned to page address 34, and similarly the third, fourth, and fifth 128 words are assigned; 40 words in the fifth page are left unused. After the allocation, the associative memory map is shown in Fig.6(b).

2.2 Loading

The second part of loading the segment now begins by changing the status of the argument to 0 and masking out page addre-s Y, page address Z, and segment number SN of the argument. The line address in subregister NI(LA) is tested to



(a) before allocation



(b) after allocation

Fig. 6 An Example of Associative Memory Map

determine whether it is 127. If it is not 127, only the line address needs to be incremented by 1 and the page address needs no change. If it is 127, a new page address needs to be fetched from the associative memory; this is done by a match-reading operation with the page address in subregister NI(PA) as page address X of the argument. Page address Y in buffer register D after the match-reading operation is the new page address; this page address is now transferred to subregister NI(PA). After the match-reading operation, subregister D(SN) and D(Y) are both tested. If subregister D(SN) does not agree with the segment number in register CSN, an interrupt is initiated; this comparison serves the function of memory protection. If subregister D(Y) contains the segment-termination symbol "*", the loading of the segment is completed.

Whether subregister NI(LA) contains 127 or not, the memory address in register NI is now transferred to address register AR and the main memory word now assumed ready at memory bus MBUS is stored into the main memory. The loading now returns and again tests the line address in subregister NI(LA). This process continues until the segment-termination symbol is reached.

2.3 Statement Description

The allocating and loading sequences in the sequence chart of Fig. 5 are described by the following procedural statements.

```
Comment, allocating sequence of the dynamic loader                    (3)

Comment, initialization                                               (4)

      PC←0, A(X,Y,Z,SN,S)←FPA-0-0-9-4, K(X,Y,Z,SN,S)←255-0-0-31-7;

      NI(PA,LA)←FPA-0, CSN←NSN;

Comment, here begins the allocating loop                              (5)

/W/   NSPN←countdn NSPN, MR←1;
```

```
            FPA←—D(Y), D(Z,SN,S)←—PC-NSN-0;

            IF (NSPN=0) THEN (GOTO X) ELSE (MW←—1);

            PC←—countup PC, A(X)←—FPA, GOTO W;

/X/         D(Y)←—*;

            MW←—1;

Comment, loading sequence of the dynamic loader                    (6)

Comment, initialization                                            (7)

            A(S)←—0, K(X,Y,Z,SN,S)←—255-0-0-0-7;

Comment, here begins the loading loop                              (8)

/Y/         IF (NI(LA)=127) THEN (GOTO Z);

            A(X)←—NI(PA);

            MR←—1;

            IF (D(SNCSN) THEN (GOTO INTERRUPT);

            IF (D(Y)=*) THEN (GOTO RETURN) ELSE (NI(PA)←—D(Y));

/Z/         NI(LA)←—countup NI(LA);

            AR←—NI;

            B←—MBUS;

            M(AR)←—B, GOTO Y;

            END
```

3. Discussions

3.1 Instruction Sequencing

Instruction sequencing is controlled by the next-instruction address register NI; the lower 7-bit constitutes the line-address counter for sequencing the line address, and the upper 8-bit stores the page address. Execution of a segment begins at the first line of the first page of the segment. Within a page, the line address is advanced by the line-address counter. When the line-address counter recycles from the value of 127 to 0, a new page needs to be obtained from the associative memory. The manner that the new page and line addresses are obtained is identical to that described in the second part of the loading process. Notice that the actual memory addresses of the instructions of the segment are not known until these instructions are being executed.

3.2 Operand Address Fetch

The instruction being executed has a 15-bit operand address. This address is one relative to the origin of the segment but not the actual memory address because the operand addresses of the instructions of the segment have not been changed during the loading. Therefore, the actual operand address where the operand is stored in the main memory must be found. The line address of this actual operand address, however, is the same as the original lower 7-bit address which, now in the buffer register B, is transferred to subregister AR(LA). The page address of the actual operand address is found from the associative memory by using page address Z in the argument (instead of page address X as in the case of instruction sequencing). After the match-reading operation, page address X in buffer regsiter D is the page address of the actual operand address which is then transferred to subregister AR(PA) for fetching the operand.

3.3 Branching, Indexing and Indirect Addressing

Branching may occur when a transfer instruction is being executed. In this case, the operand address is an instruction address. The operand address is again fetched in the same manner as described above except that the line address in buffer register B and the page address from the associative memory are now transferred to the next-instruction address register, instead of the original address in address register AR.

The above operand address fetch does not prevent the use of indexing and indirect addressing. To additionally provide indexing, the contents of the index register are added to (or subtracted from) the original relative address before the operand address is fetched. For indirect addressing, an operand address instead of an operand is fetched and the operand address fetch is repeated.

3.4 Program Return and Storage Release

If an interrupt occurs during the execution of the instructions of a segment, the supervisor program places the return address (both page and line addresses) in the segment number register assigned to this segment as illustrated by P1, P2, P3 and P4 in Fig. 8.13, and then selects another segment for execution. Whenever execution of the original segment is resumed, the supervisor program obtains the return address from the segment number register assigned to the original segment.

When execution of a segment is completed, the supervisor program may decide to release the pages of the segment. The release can be simply done by linking the pages to the available segment.

4. Microprogramming

The allocation and loading algorithm has been described in the sequence chart of Fig. 5 and in statements (3) to (8). This algorithm is now implemented by a microprogram stored in a control memory. As will be seen, it requires a microprogram of 10 control words.

4.1 Microprogram control configuration

The microprogram control has control memory CM associated with address register H and control word register F, status register STA, control register MC, reset switch RESET, clock P, and two decoders. This configuration is described below.

```
Comment, the microprogram control configuration                    (9)
Register,   H(0-7),                  $control memory address register
            F(0-31),                 $control word register
            MC(0-2),                 $main-memory-cycle register,
            STA(0-3),                $status register
Memory,     CM(H)=CM(0-255,0-29),    $control memory
Switch,     RESET(ON),
Clock,      P(0-2),
Block,      ADDRESS(IF (STA(0)+STA(1)+STA(2)+STA(3)=0) THEN (H<--countup H)
                    ELSE (H<--F(0-7), STA<--0),
Decoder,    KA(0-3)=F(10-11),
```

KN(0-3)=F(17-18),

Micro-operation ADDRESS in the above block statement specifies a two-way branching, depending on the status in the STATUS register.

4.2 Timing and control signals

Each main memory cycle is chosen to coincide with three control memory cycles; each control memory cycle to coincide with one clock cycle; each associative memory to coincide with one clock phase. Therefore, there are 3 steps in each control memory cycle and 9 steps in each main memory cycle. The control signals are now described below.

```
Comment, control signals described by the labels

/RESET(ON)/     MC<--4,

/MC(0)*P(0)/                        $begin of a mm cycle and cm cycle

/MC(0)*P(1)/

/MC(0)*P(2)/    MC<--cir MC         $end of a cm cycle

/MC(1)*P(0)/                        $begin of a cm cycle

/MC(1)*P(1)/

/MC(1)*P(2)/    MC<--cir MC         $end of a cm cycle

/MC(2)*P(0)/                        $begin of a cm cycle

/MC(2)*P(1)/

/MC(2)*P(2)/    MC<--cir MC         $end of both memory cycles
```

The three steps in each control memory cycle are sequenced by the three phases of clock P(0-2), and the three control memory cycles in each main memory cycle are sequenced by the three states of ring counter MC(0-2) which is circularly rightshifted at the end of each control memory cycle.

4.3 Control word format

Table 4 shows the control word format. The 30 bits of the control word in register F consists of two groups: field F(0-7) which contains a control memory for branching address and field (8-29) which contains the control bits. The two decoders are attached to fields F(10-11) and F(17-18). Each of bits 19, 22, 23 and 25 controls a micro-operation which tests a condition and then stores the tested result in the STATUS register.

4.4 Microprogramming the loader

With the control word format in Table 4, the sequence chart in Fig. 5 can now be described by the following CDL statements. These statements are grouped into 9 control words, each indicated by a comment statement and shown with their control memory addresses. The control word is fetched during clock phase P(0). Control memory address is transferred to address register H during clock phase P(1). Associative memory is operated also during clock phase P(1). All other micro-operations except the main-memory operation occur during clock phase P(2).

```
Comment, dynamic loader                                          (10)

/RESET(ON)/        STA<--0, H<--0, F<--0, MC<--0,

Comment, control word at H=0                                     (11)
```

Table 4 Control Word Format

Control bit	Clock phase	Code	Micro-operation
F(0-7)			next control word address
F(8)	P(0)		F←CM(H)
F(9)	P(2)		DO ADDRESS
F(10-11)		00	no operation
	P(1)	01	A(X,Y,Z,SN,S) FPA-0-0-9-4,
	P(1)	10	A(X)←FPA,
	P(1)	11	A(S)←0,
F(12)	P(1)		A(X)←NI(PA),
F(13)	P(1)		K(X,Y,Z,SN,S)←255-0-0-31-7,
F(14)	P(1)		K(X,Y,Z,SN,S)←255-0-0-31-7,
F(15)	P(1)		PC←0,
F(16)	P(1)		PC←countup PC,
F(17-18)		00	no operation
	P(1)	01	NI(PA,LA)←FPA-0,
	P( )	10	NI(LA)←countup NI(LA),
	P( )	11	NI(PA)←D(Y),
F(19)	P(1)		IF (NI(LA)=127) THEN (STA(0)←0, A(X)←NI(PA)) ELSE (STA(0)←1),
F(20)	P(1)		D(Z,SN,S)←PC-NSN-0,
F(21)	P(1)		D(Y)←*
F(22)	P(1)		IF (D(Y)≠*) THEN (STA(1)←0, NI(PA)←D(Y)) ELSE (STA(1)←1),
F(23)	P(1)		IF (D(SN)=CSN) THEN (STA(2)←0) ELSE (STA(2)←1),
F(24)	P(1)		NSPN←countdn NSPN,
F(25)	P(1)		IF (NSPN(0)) THEN (STA(3)←1) ELSE (STA(3)←0),
F(26)	P(1)		FRA←D(Y)
F(27)	P(1)		CSN←NSN
F(28)	P(2)		MR←1
F(29)	P(2)		MW←1

```
/F(8)*P(0)/          F<--CM(H),

/KA(1)*P(1)/         A(X,Y,Z,SN,S)<--FPA-0-0-9-4,

/F(13)*P(2)/         K(X,Y,Z,SN,S)<--255-0-0-31-7,

/F(15)*P(1)/         PC<--0,

/KN(1)*P(1)/         NI(PA,LA)<--FPA-0,

/F(27)*P(1)/         CSN<--NSW,

/F(9)*P(2)/          DO ADDRESS,                $H<--1

Comment, here begins the allocation

Comment, control word of H=1                                    (12)

/F(8)*P(0)/          F<--CM(H),

/KA(2)*P(1)/         A(X)<--FPA,

/F(24)*P(1)/         NSPN<--countdn NSPN,

/F(28)*P(2)/         MR<--1,

/F(9)*P(2)/          DO ADDRESS,                $H<--2

Comment, control word at H=2                                    (13)

/F(8)*P(0)/          F<--CM(H)

/F(26)*P(1)/         FPA---D(Y),

/F(20)*P(1)/         D(Z,SN,S)<--PC-NSN-0,
```

```
/F(25)*P(1)/      IF (NSPN=0) THEN (STA(3)<--1) ELSE (STA(3)<--0)

/F(9)*P(2)/       DO ADDRESS                          $H<--4 or 3

Comment, control word at H=3                                        (14)

/F(8)*P(0)/       F<--CM(H)

/F(16)*P(1)/      PC<--countup PC,

/F(29)*P(2)/      MW<--1,

/F(9)*P(2)/       DO ADDRESS,                         $H<--1

Comment, here begins the loading

Comment, control word at H=4                                        (15)

/F(8)*P(0)/       F<--CM(H),

/F(21)*P(1)/      D(Y)<--*,

/F(29)*P(2)/      MW<--1,

/F(9)/*P(2)/      DO ADDRESS,                         $H<--5

Comment, control word at H=5                                        (16)

/F(8)*P(0)/       F<--CM(H),

/KA(3)*P(1)/      A(S)<--0,

/F(14)*P(1)/      K(X,Y,Z,SN,S)<--255-0-0-0-7,

/F(9)*P(2)/       DO ADDRESS,
```

```
Comment, control word at H=6                                          (17)

/F(8)*P(0)/          F<--CM(H),

/F(19)*P(1)/         IF (NZ(LA)=127) THEN (STA(0)<--0, A(X)<--NI(PA))

                                    ELSE (STA(0)<--1)

/F(31)*P(2)/         IF (STA(0)=0) THEN (MR<--1),

/F(9)*P(2)/          DO ADDRESS                $H<--7 or 9

Comment, control word at H=7                                          (18)

/F(8)*P(0)/          F<--CM(H),

/F(23)*P(1)/         IF (D(SN)=CSN) THEN (STA(2)<--0) ELSE (STA(2)<--1),

/F(9)*P(2)/          DO ADDRESS                $H<--8 or interrupt

Comment, control word at H=8                                          (19)

/F(8)*P(0)/          F<--CM(H),

/F(22)*P(1)/         IF (D(Y)≠*) THEN (STA(1)<--0, NI(PA)<--D(Y))

                                 ELSE (STA(1)<--1),

/F(9)*P(2)/          DO ADDRESS,               $H<--9 or return

Comment, control word at H=9                                          (20)

/F(8)*P(0)/          F<--CM(H),

/KN(2)*P(1)/         NI(LA)<--countup NI(LA),
```

```
/F(9)*P(2)/        DO ADDRESS, MC<--4,          $H<--6

Comment, store a word from bus MBUS into main memory M                (21)

/MC(0)*P(0)/       AR<--NI,                     $simultaneous with statements (17)

/MC(0)*P(1)/

/MC(0)*P(2)/       MC<--cir MC,

/MC(1)*P(0)/       B<--MBUS,                    $simultaneous with statements (18)

/MC(1)*P(1)/

/MC(1)*P(2)/       MC<--cir MC,

/MC(2)*P(0)/       M(AR)<--B,                   $simultaneous with statements (19)

/MC(2)*P(1)/

/MC(2)*P(2)/       MC<--0,

                   END
```

The associative memory is operated intermittently during both allocation and loading. The main memory, controlled by register MC, is not operated during allocation; it is operated intermittently during loading. However, when it is operated, the main memory cycle occurs simultaneously with control memory cycle in executing the controlwords located at addresses 6, 7 and 8.

References

1. Chu, Y., "Application of Content-Addressed Memory for Dynamic Storage Allocation," RCA Review, March 1965, pp. 140-152.
2. Chu, Y., "Direct Execution of Programs in Floating Code by Address Interpretation," IEEE Transactions on Electronic Computers, June, 1965, pp. 417-422.
3. Flores, I., "Computer Software," Prentice-Hall, Inc., 1965.
4. Barron, D. W., "Assemblers and Loaders," MacDonald and American Elsevier, 1969.
5. Lanzano, B. C., "Loader Standardization for Overlay Programs,"Com. of the ACM, October 1969, pp. 541-550.
6. Pardo, O. R., "A Loader Algorithm for Microprogramming," Computer Science Center, University of Maryland, 1970.
7. Y. Chu, "Introduction to Computer Organization," Prentice-Hall, Inc., 1970.