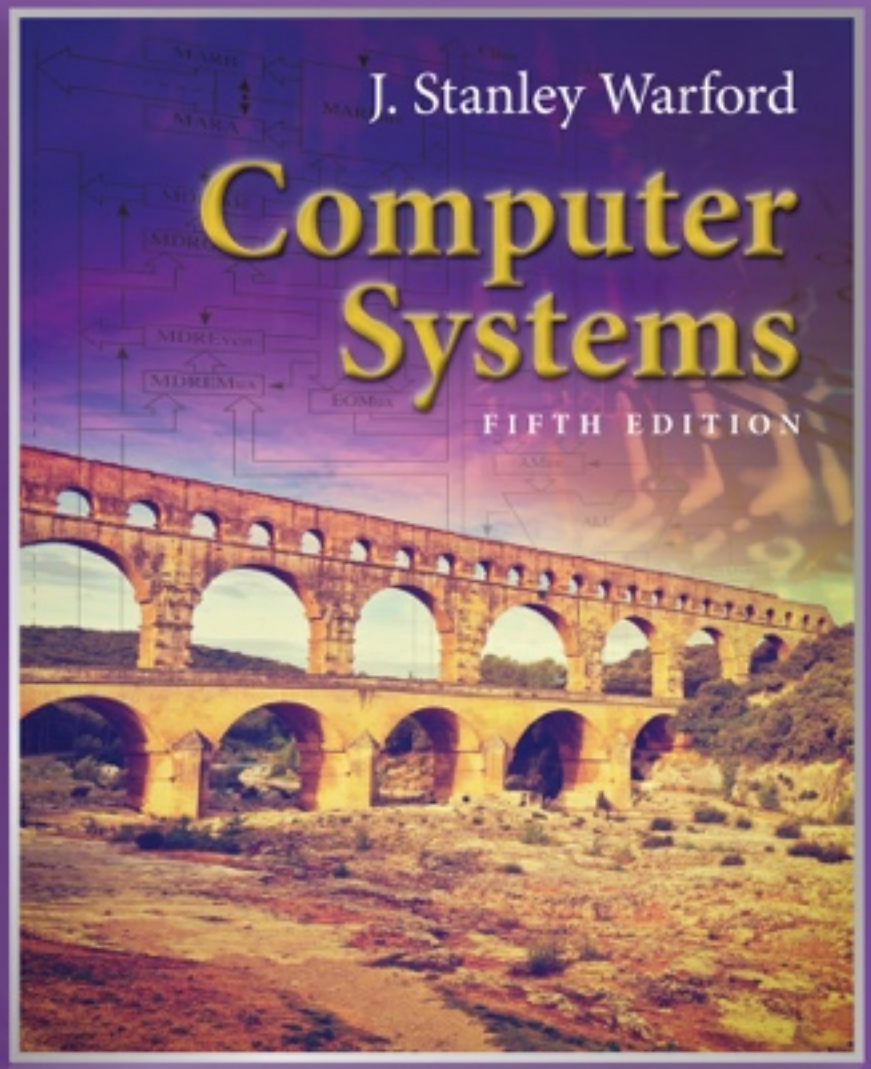
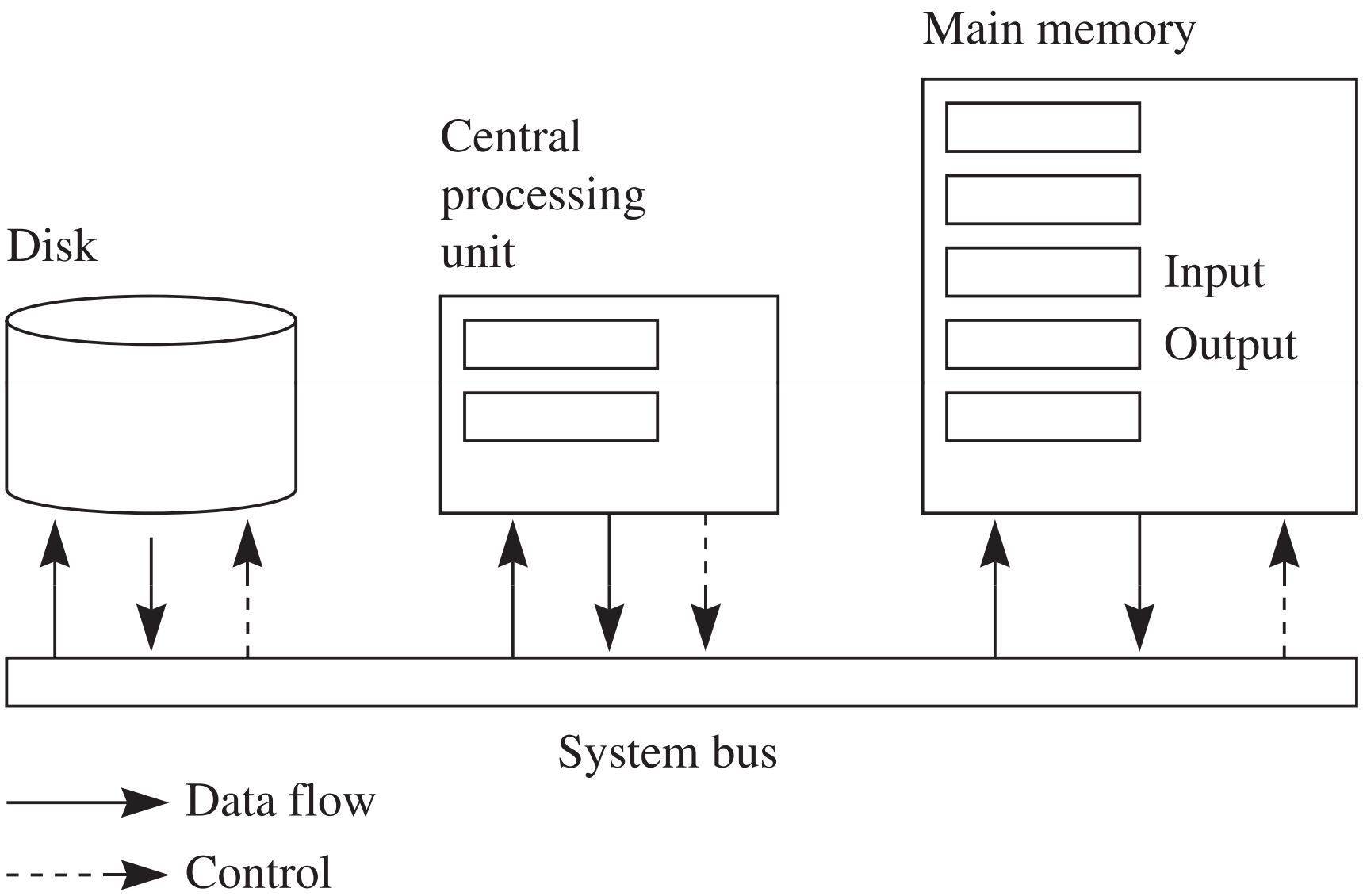
Chapter 4

Computer Architecture



Pep/9 virtual machine

Figure 4.1



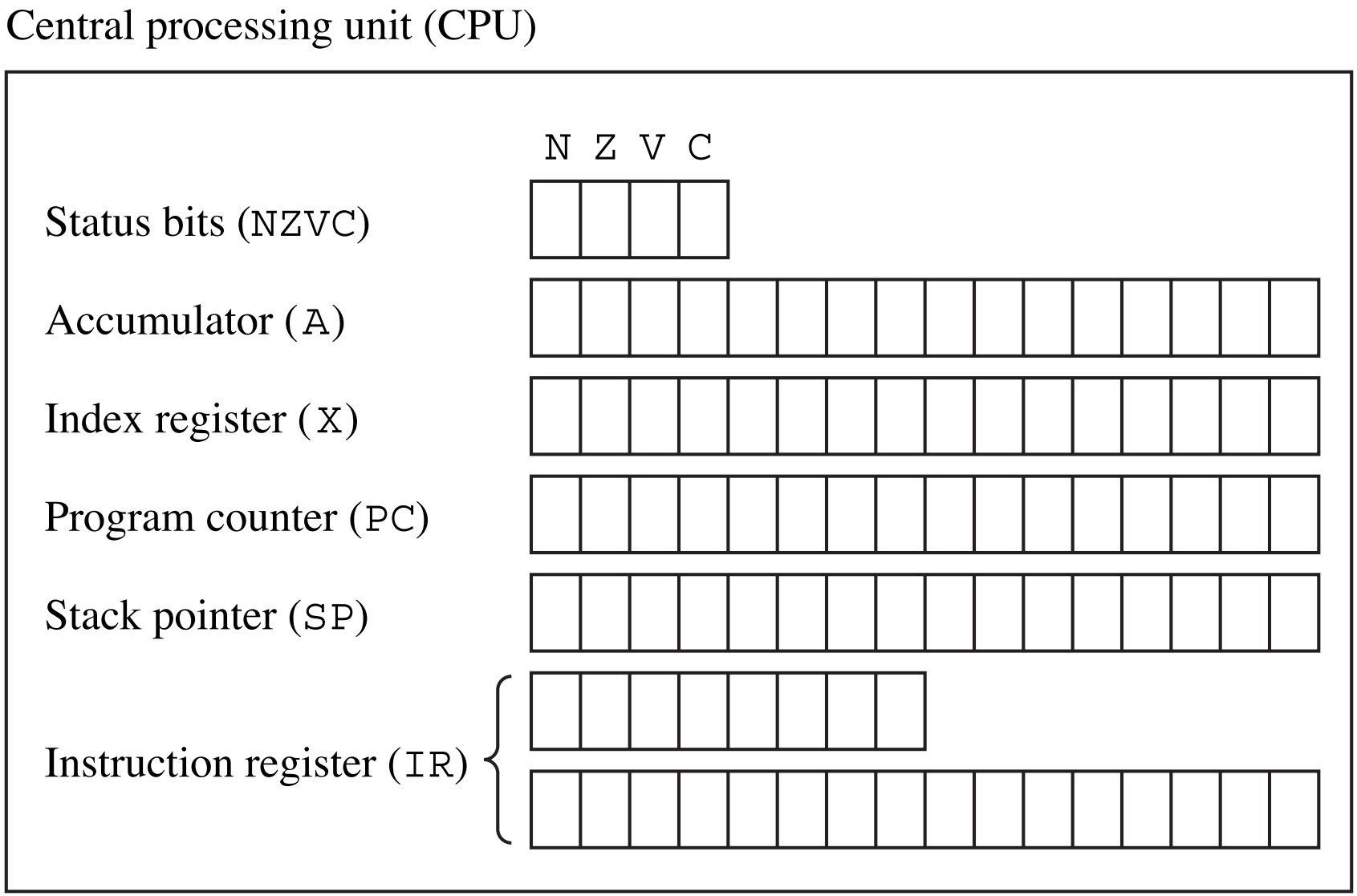


Figure 4.3

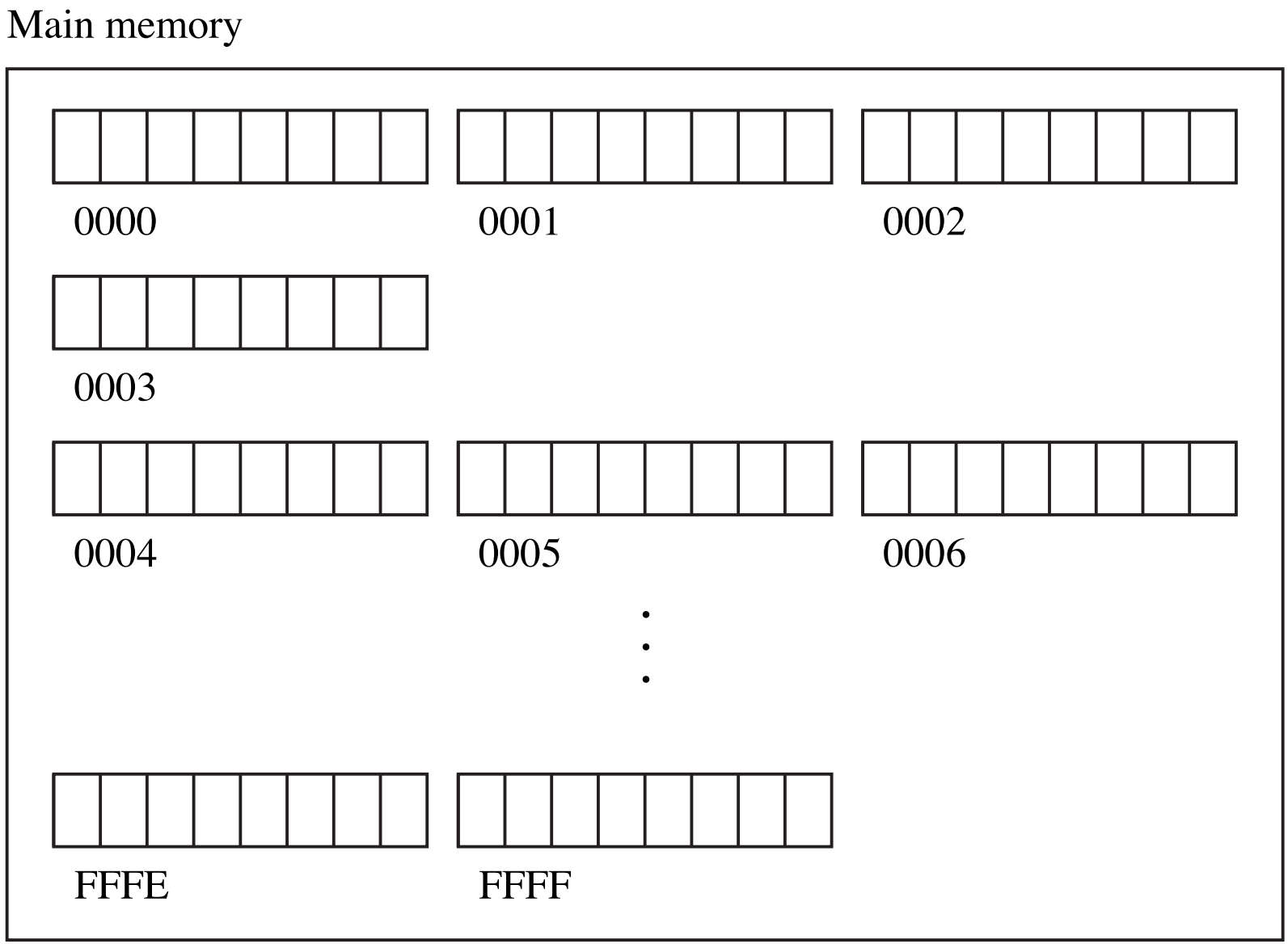


Figure 4.4

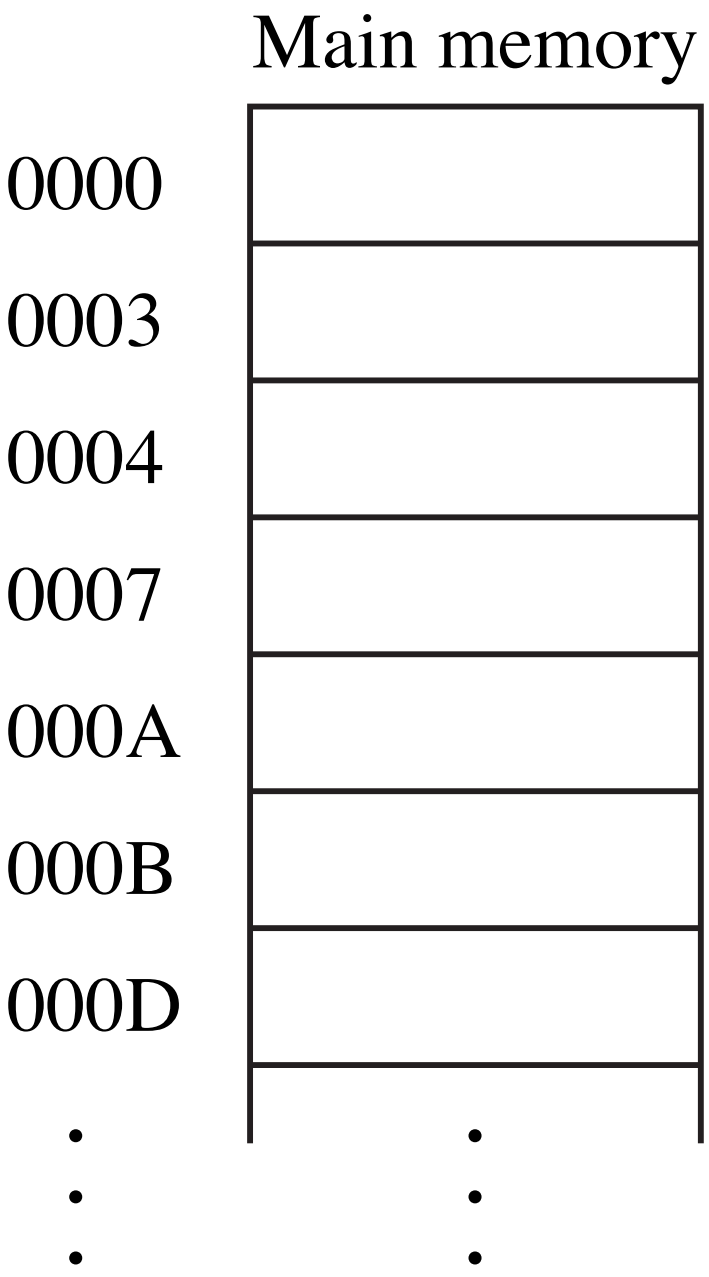
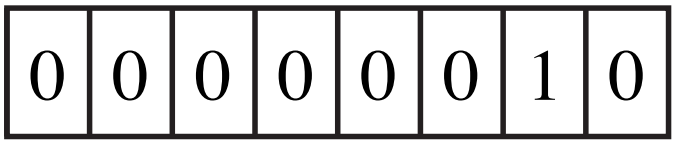
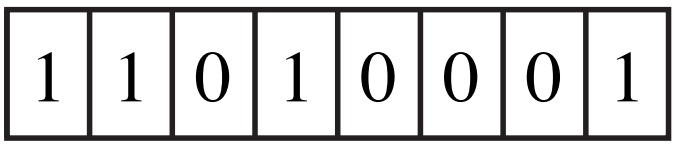


Figure 4.5



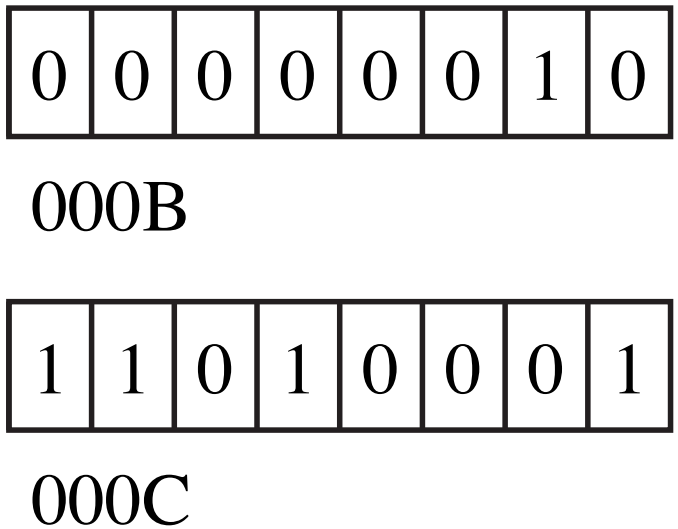
000B



000C

(a) The content in binary.

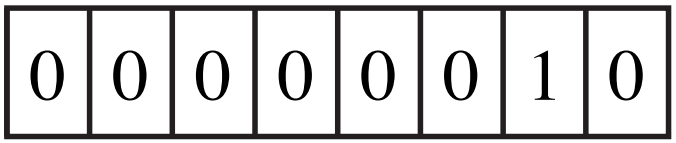
Figure 4.5



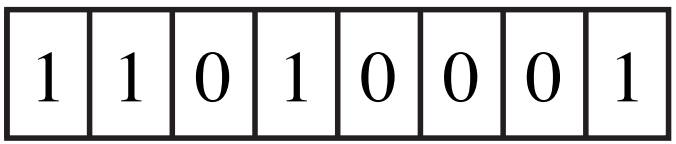
(a) The content in binary.

(b) The content in hexadecimal.

Figure 4.5

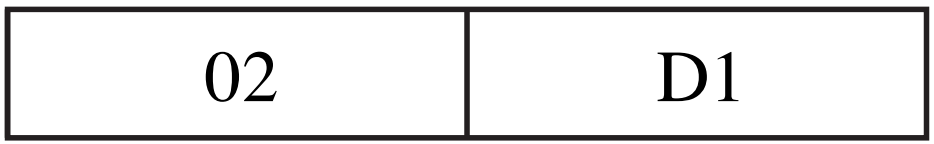


000B



000C

(a) The content in binary.



000B 000C

(b) The content in hexadecimal.

000B 02D1

(c) The content in a machine language listing.

Figure 4.6

Instruction Specifier	Instruction
0000 0000	Stop execution
0000 0001	Return from `CALL`
0000 0010	Return from trap
0000 0011	Move SP to A
0000 0100	Move NZVC flags to A⟨12..15⟩
0000 0101	Move A⟨12..15⟩ to NZVC flags
0000 011r	Bitwise invert r
0000 100r	Negate r
0000 101r	Arithmetic shift left r
0000 110r	Arithmetic shift right r
0000 111r	Rotate left r
0001 000r	Rotate right r

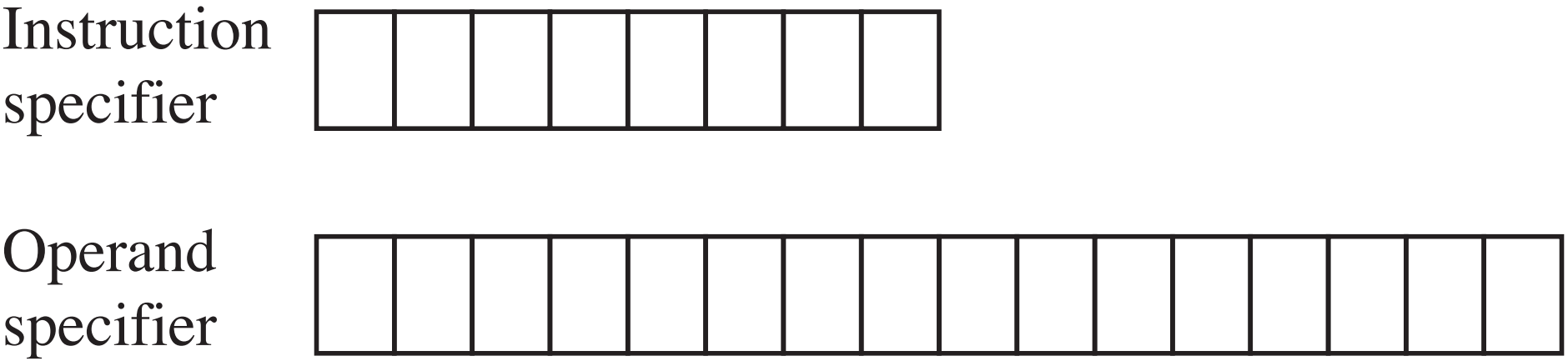
Figure 4.6 (continued)

0001 001a	Branch unconditional
0001 010a	Branch if less than or equal to
0001 011a	Branch if less than
0001 100a	Branch if equal to
0001 101a	Branch if not equal to
0001 110a	Branch if greater than or equal to
0001 111a	Branch if greater than
0010 000a	Branch if V
0010 001a	Branch if C
0010 010a	Call subroutine
0010 011n	Unimplemented opcode, unary trap
0010 1aaa	Unimplemented opcode, nonunary trap
0011 0aaa	Unimplemented opcode, nonunary trap
0011 1aaa	Unimplemented opcode, nonunary trap
0100 0aaa	Unimplemented opcode, nonunary trap
0100 1aaa	Unimplemented opcode, nonunary trap

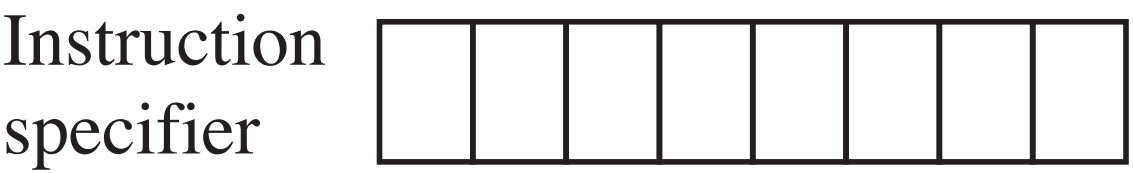
Figure 4.6 (continued)

0101 0aaa	Add to stack pointer (SP)
0101 1aaa	Subtract from stack pointer (SP)
0110 raaa	Add to r
0111 raaa	Subtract from r
1000 raaa	Bitwise AND to r
1001 raaa	Bitwise OR to r
1010 raaa	Compare word to r
1011 raaa	Compare byte to r⟨8..15⟩
1100 raaa	Load word r from memory
1101 raaa	Load byte r⟨8..15⟩ from memory
1110 raaa	Store word r to memory
1111 raaa	Store byte r⟨8..15⟩ to memory

Figure 4.7



(a) The two parts of a nonunary instruction



(b) A unary instruction

aaa	Addressing Mode
000	Immediate
001	Direct
010	Indirect
011	Stack-relative
100	Stack-relative deferred
101	Indexed
110	Stack-indexed
111	Stack-deferred indexed

(a) The addressing-aaa field.

Figure 4.8 (continued)

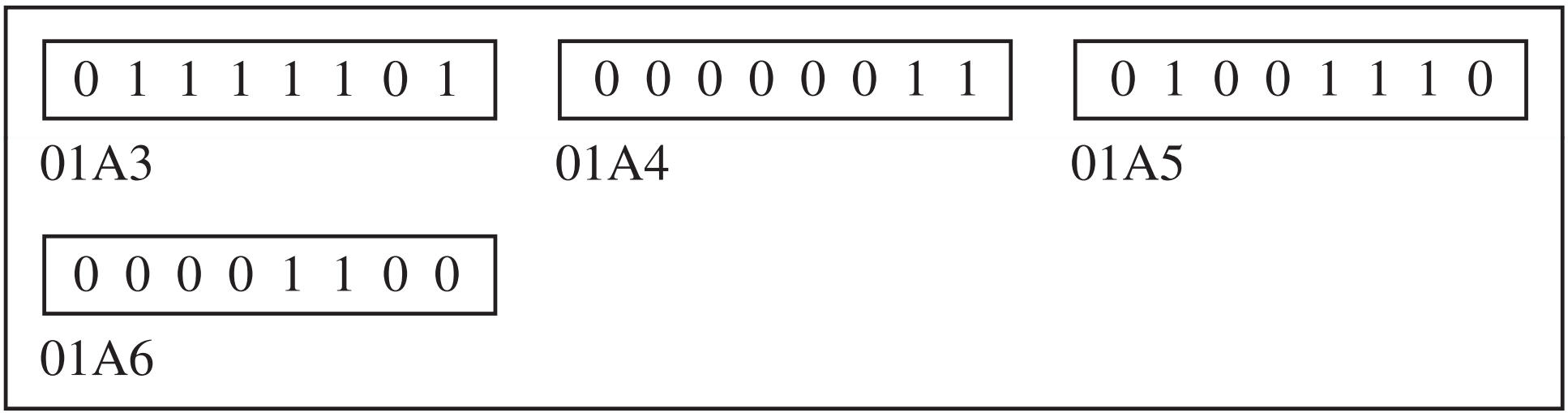
a	Addressing Mode
0	Immediate
1	Indexed

(b) The addressing-a field.

r	Register
0	Accumulator, A
1	Index register, X

(c) The register-r field.

Figure 4.9



Direct addressing

- Oprnd = Mem[OprndSpec]
- The operand specifier is the memory address of the operand.

The stop instruction

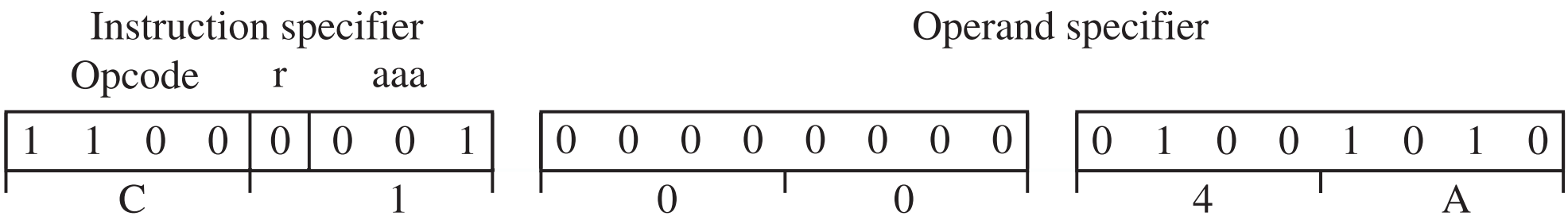
- Instruction specifier: 0000 0000
- Causes the computer to stop

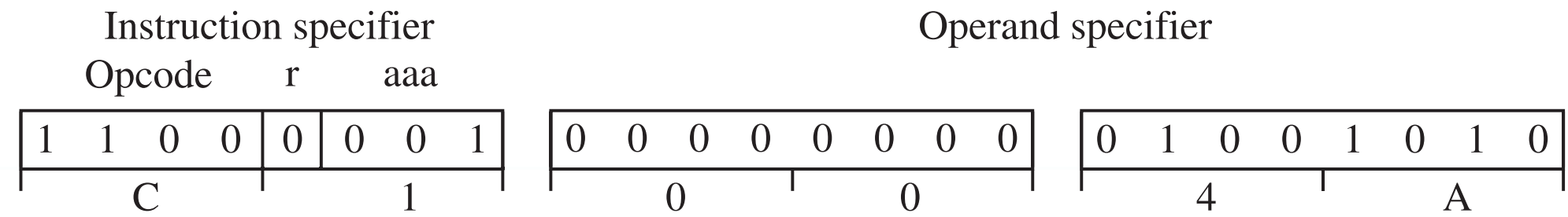
The load word instruction

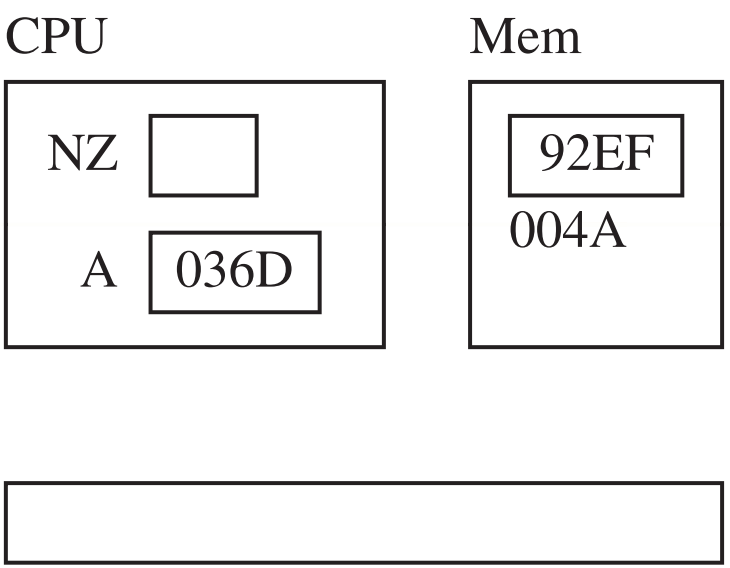
- Instruction specifier: 1100 raaa
- Loads one word (two bytes) from memory to register r

$$r \leftarrow Oprnd\ ;\ N \leftarrow r < 0\ ,\ Z \leftarrow r = 0$$

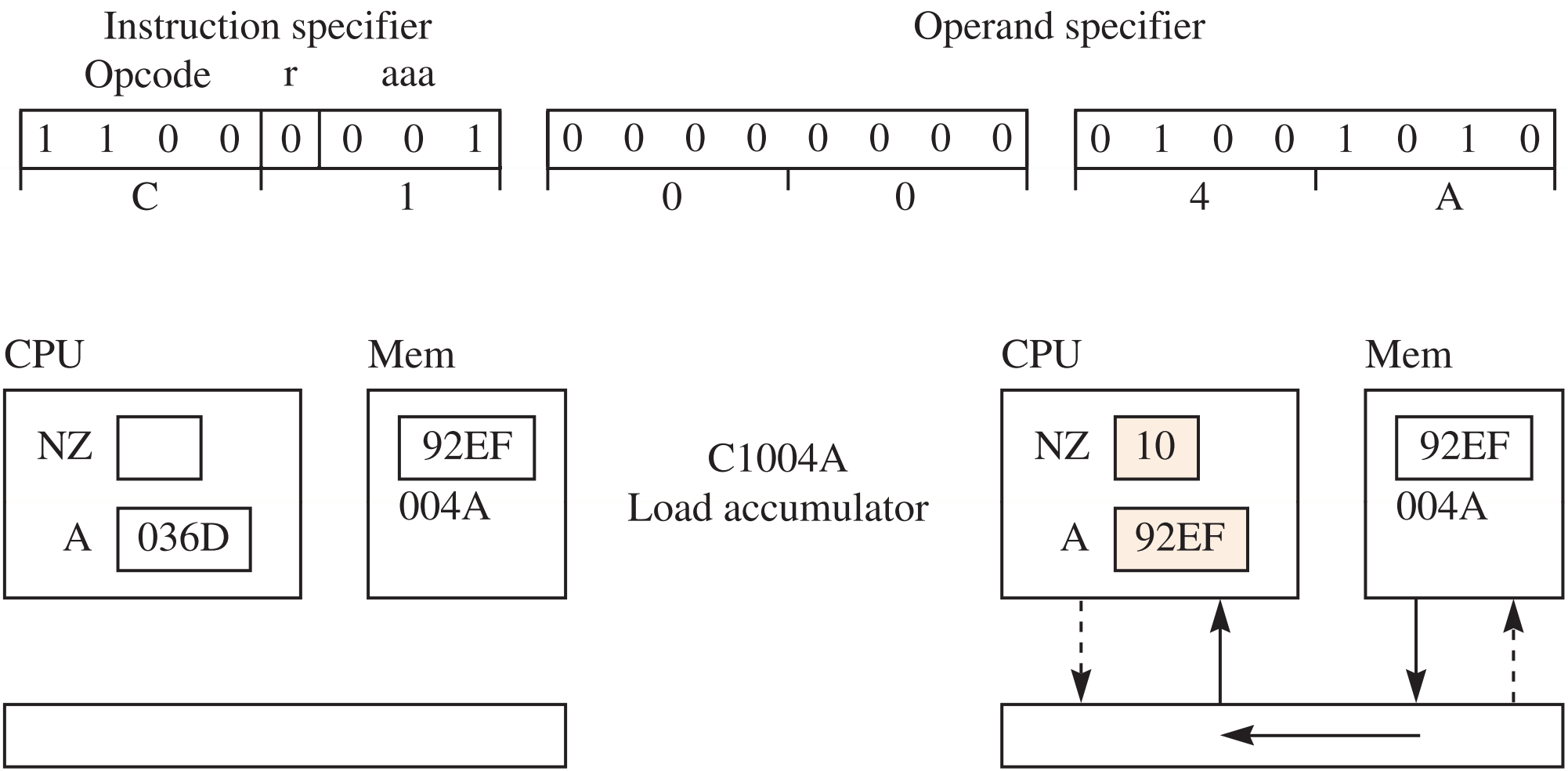
Figure 4.10, 4.11







(a) Before.



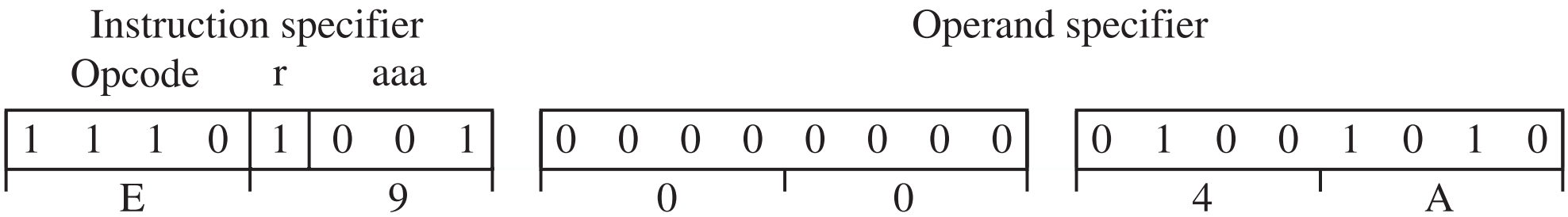
(a) Before.

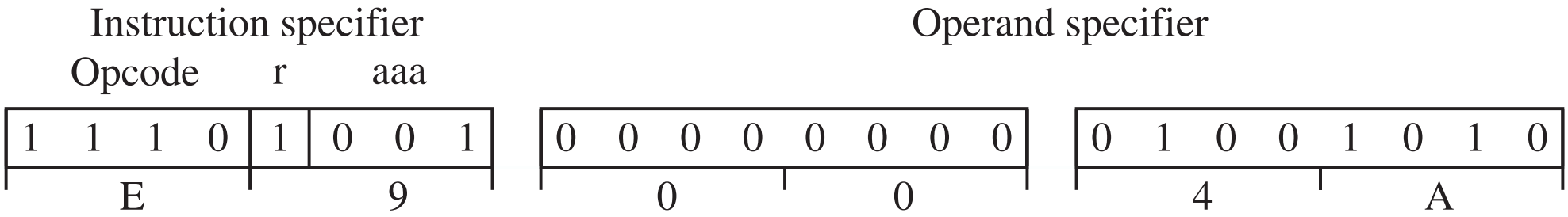
(b) After.

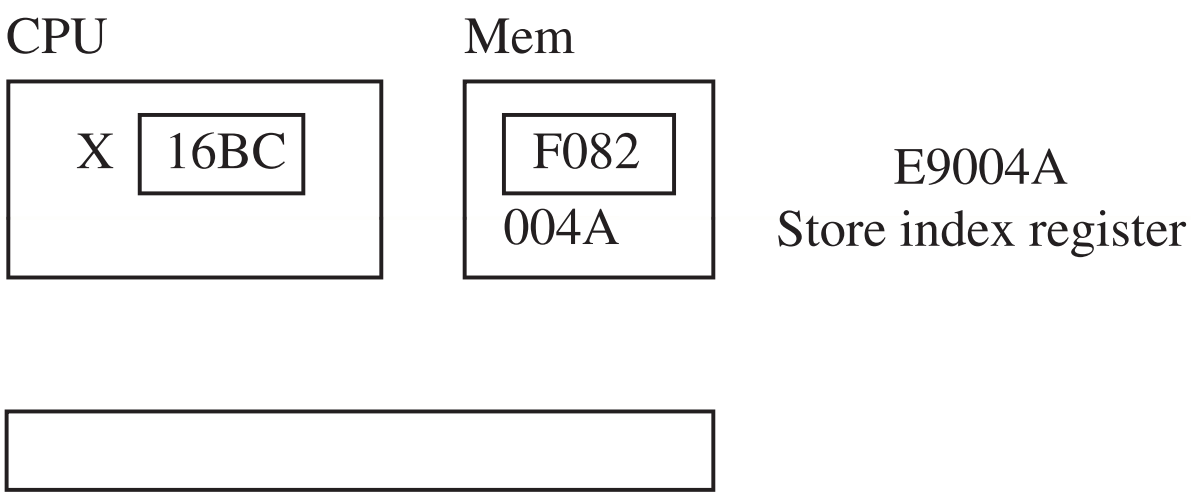
The store word instruction

- Instruction specifier: 1110 raaa
- Stores one word (two bytes) from register r to memory

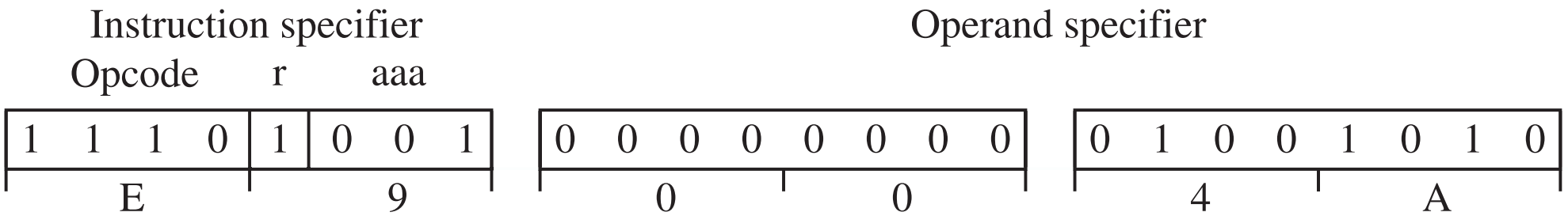
$$\text{Oprnd} \leftarrow \text{r}$$

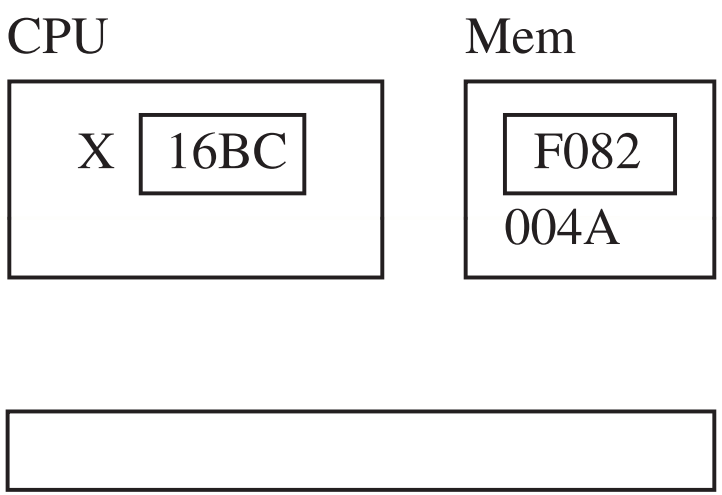






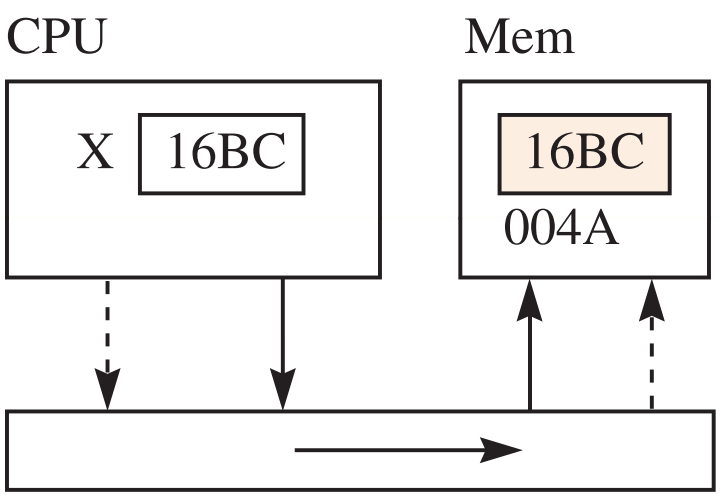
(a) Before.





(a) Before.

E9004A
Store index register



(b) After.

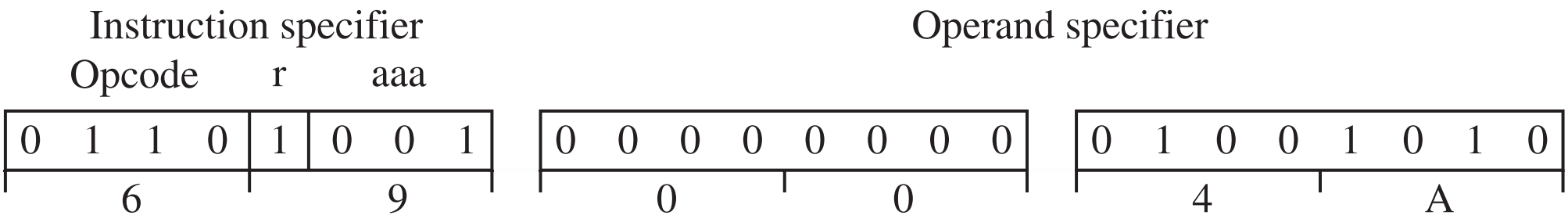
The add instruction

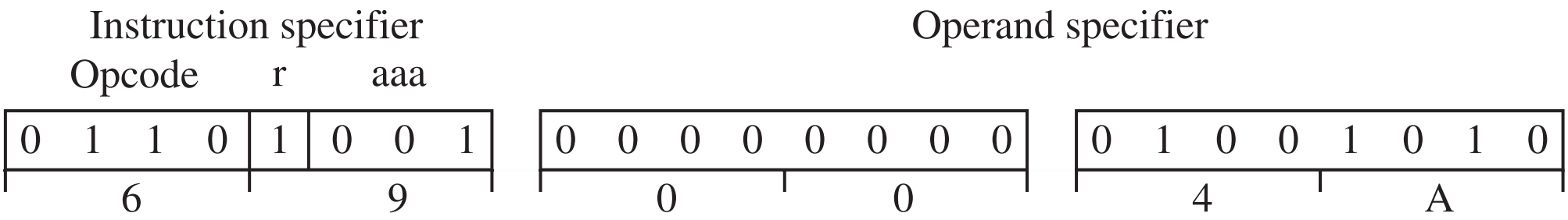
- Instruction specifier: 0110 raaa
- Adds one word (two bytes) from memory to register r

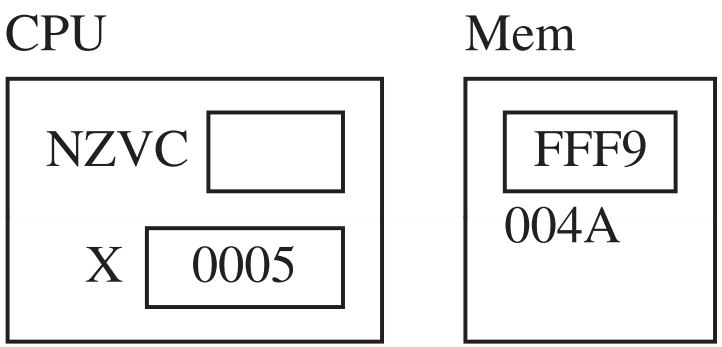
$$r \leftarrow r + Oprnd\ ;\ N \leftarrow r < 0\ ,\ Z \leftarrow r = 0\ ,$$

$$V \leftarrow \{overflow\}\ ,\ C \leftarrow \{carry\}$$

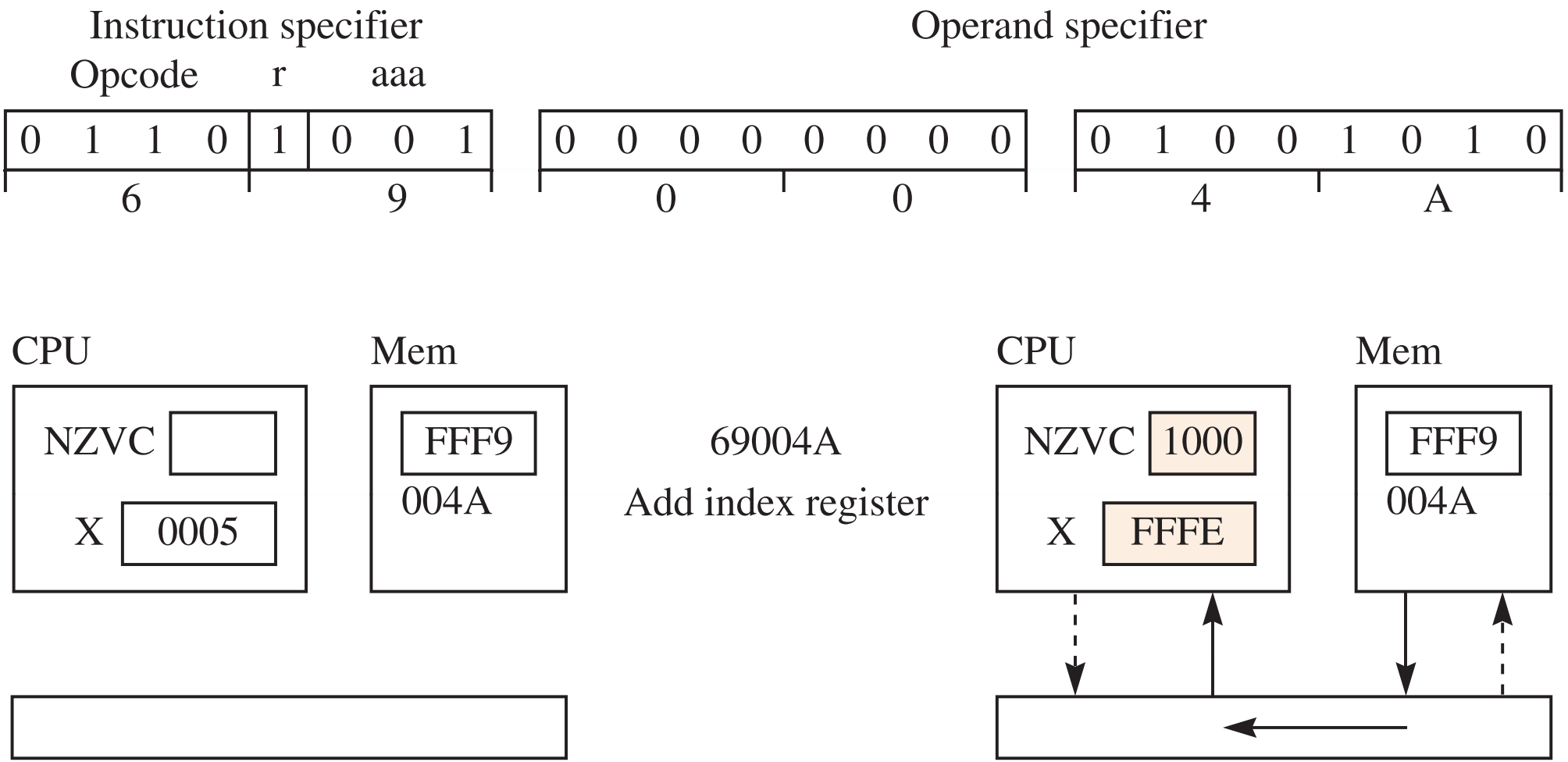
Figure 4.14, 4.15







(a) Before.



(a) Before.

(b) After.

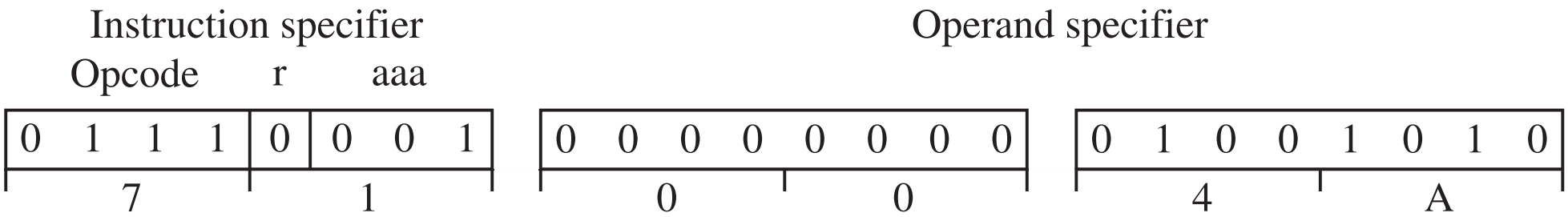
The subtract instruction

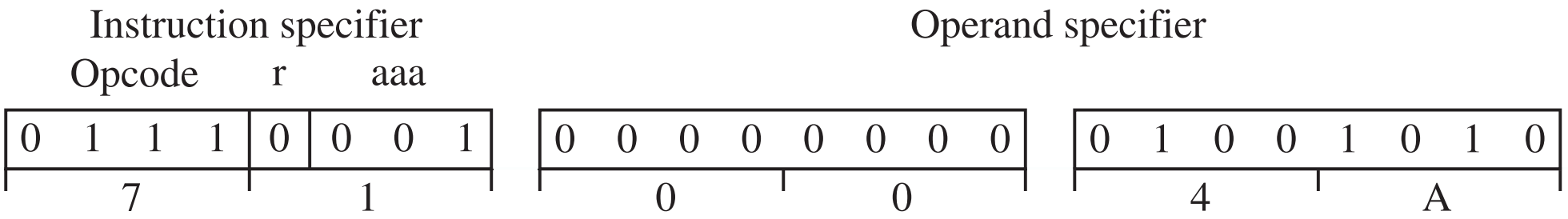
- Instruction specifier: 0111 raaa
- Subtracts one word (two bytes) from memory from register r

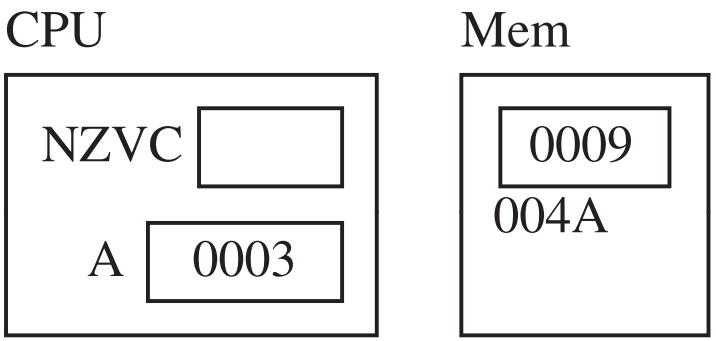
$$r \leftarrow r - Oprnd\ ;\ N \leftarrow r < 0\ ,\ Z \leftarrow r = 0\ ,$$

$$V \leftarrow \{overflow\}\ ,\ C \leftarrow \{carry\}$$

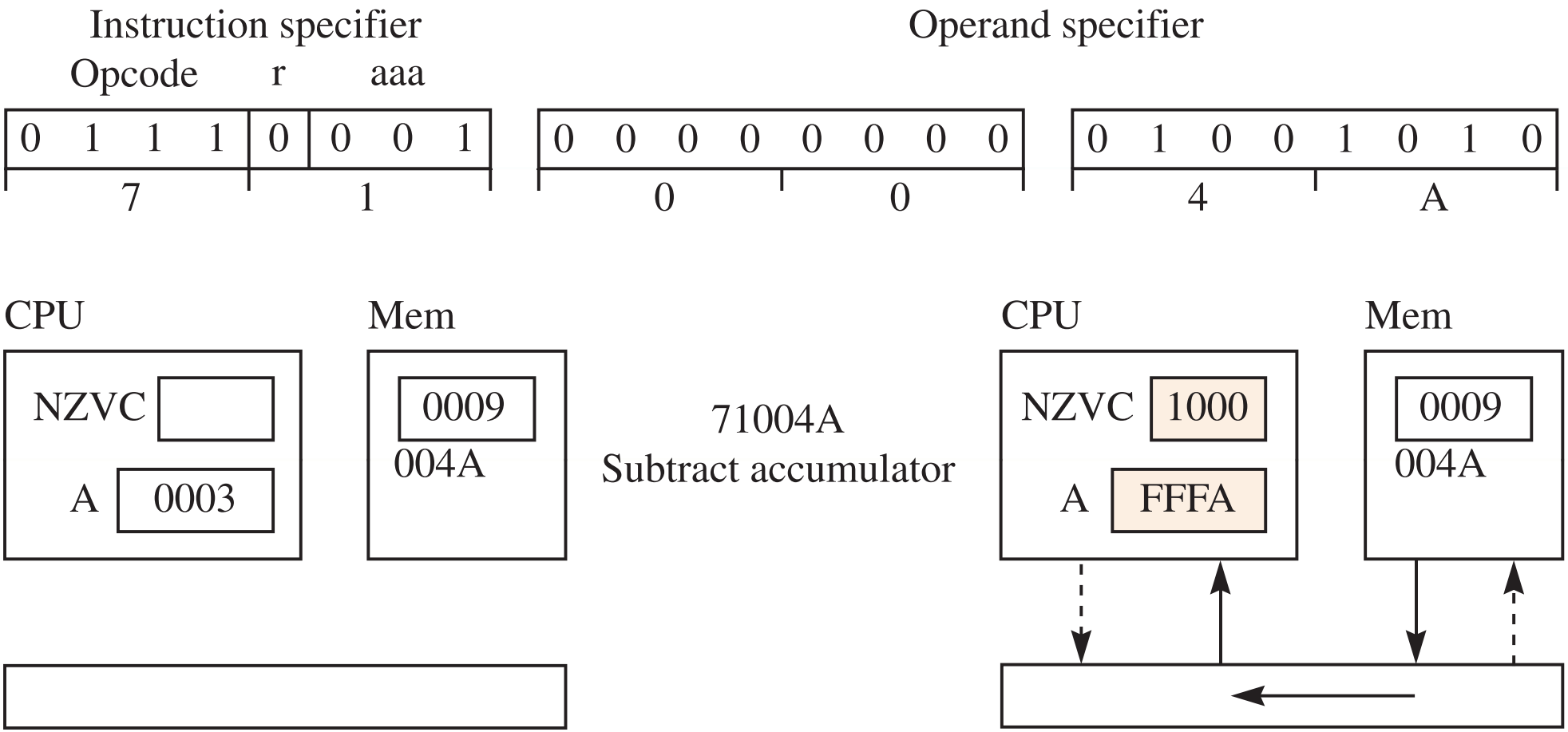
Figure 4.16, 4.17







(a) Before.



(a) Before.

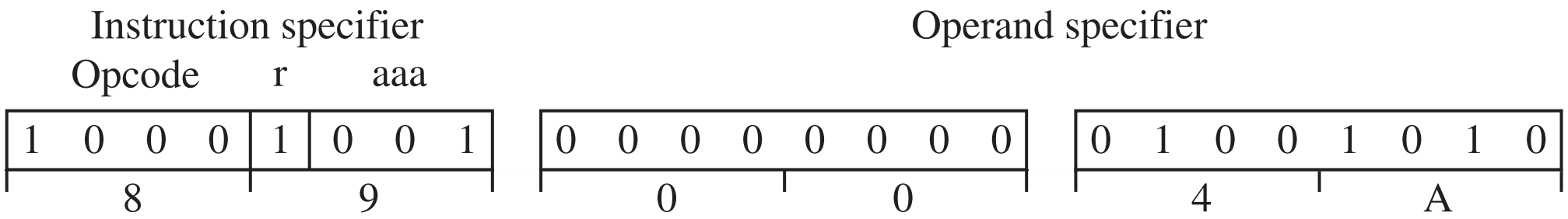
(b) After.

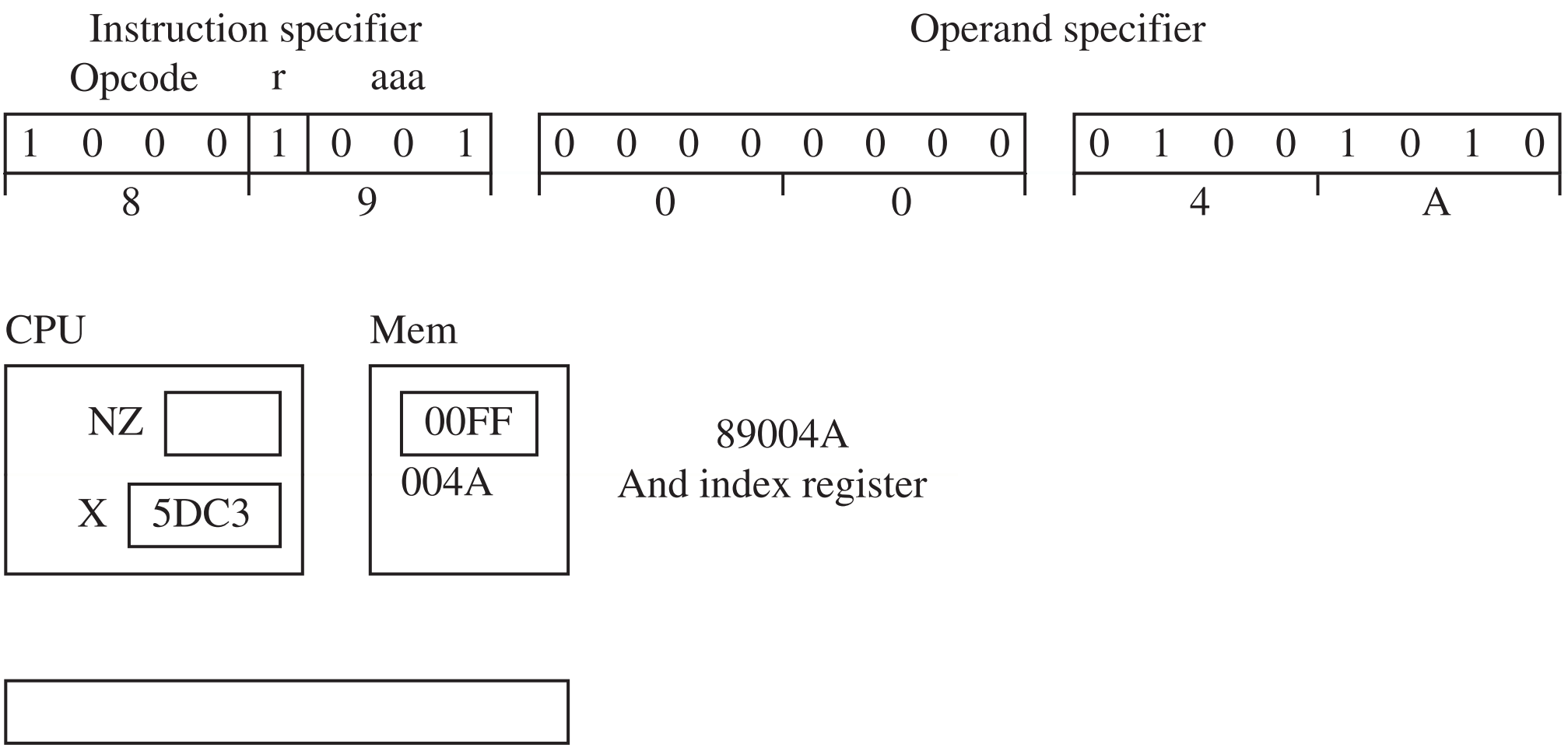
The and instruction

- Instruction specifier: 1000 raaa
- ANDs one word (two bytes) from memory to register r

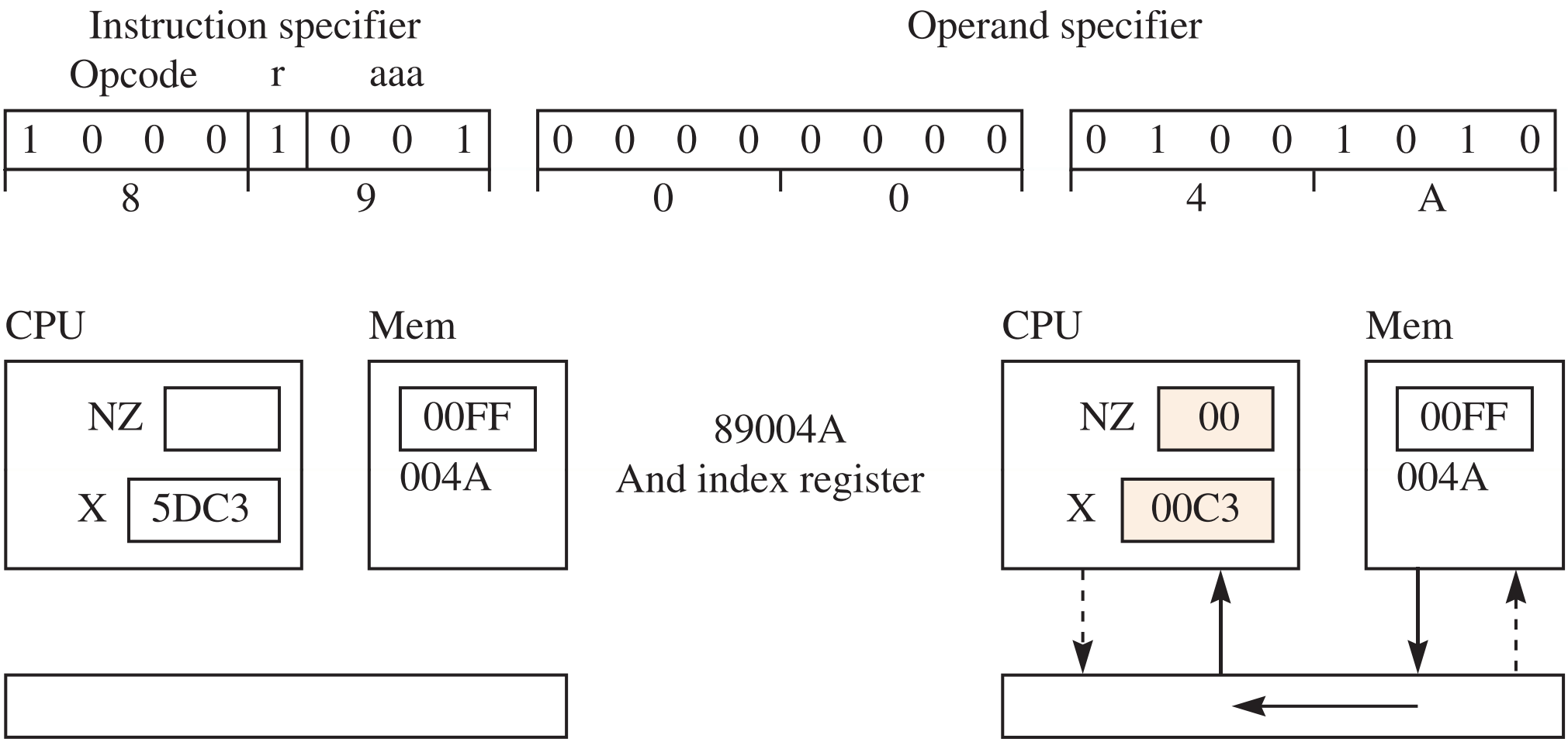
$$r \leftarrow r \wedge Oprnd \; ; \; N \leftarrow r < 0 \; , \; Z \leftarrow r = 0$$

Figure 4.18, 4.19





(a) Before.



(a) Before.

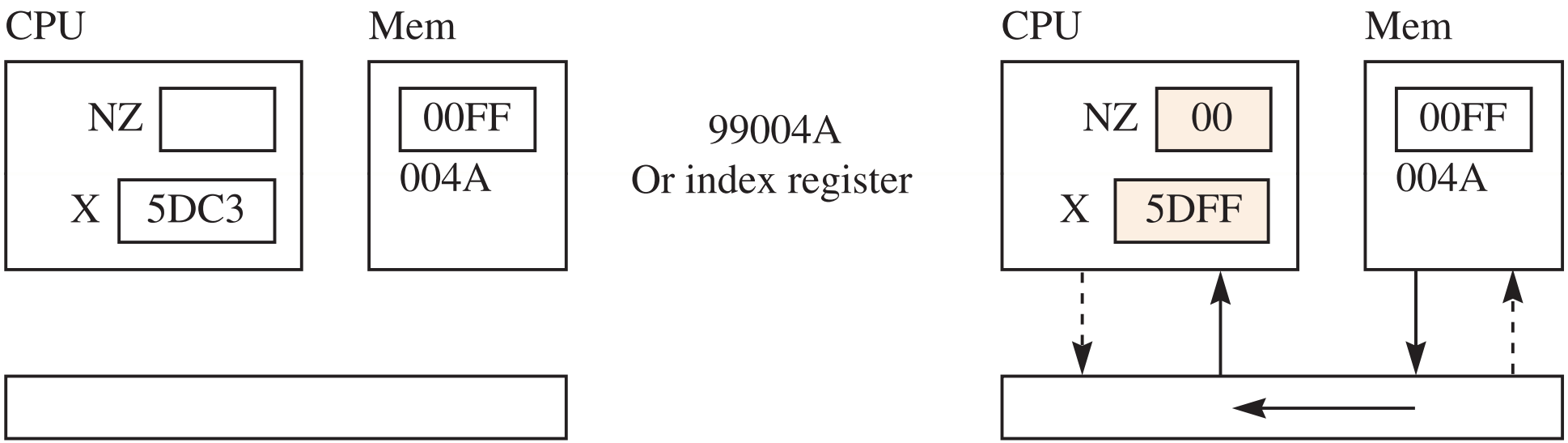
(b) After.

The or instruction

- Instruction specifier: 1001 raaa
- ORs one word (two bytes) from memory to register r

$$r \leftarrow r \vee Oprnd \; ; \; N \leftarrow r < 0 \; , \; Z \leftarrow r = 0$$

Figure 4.20



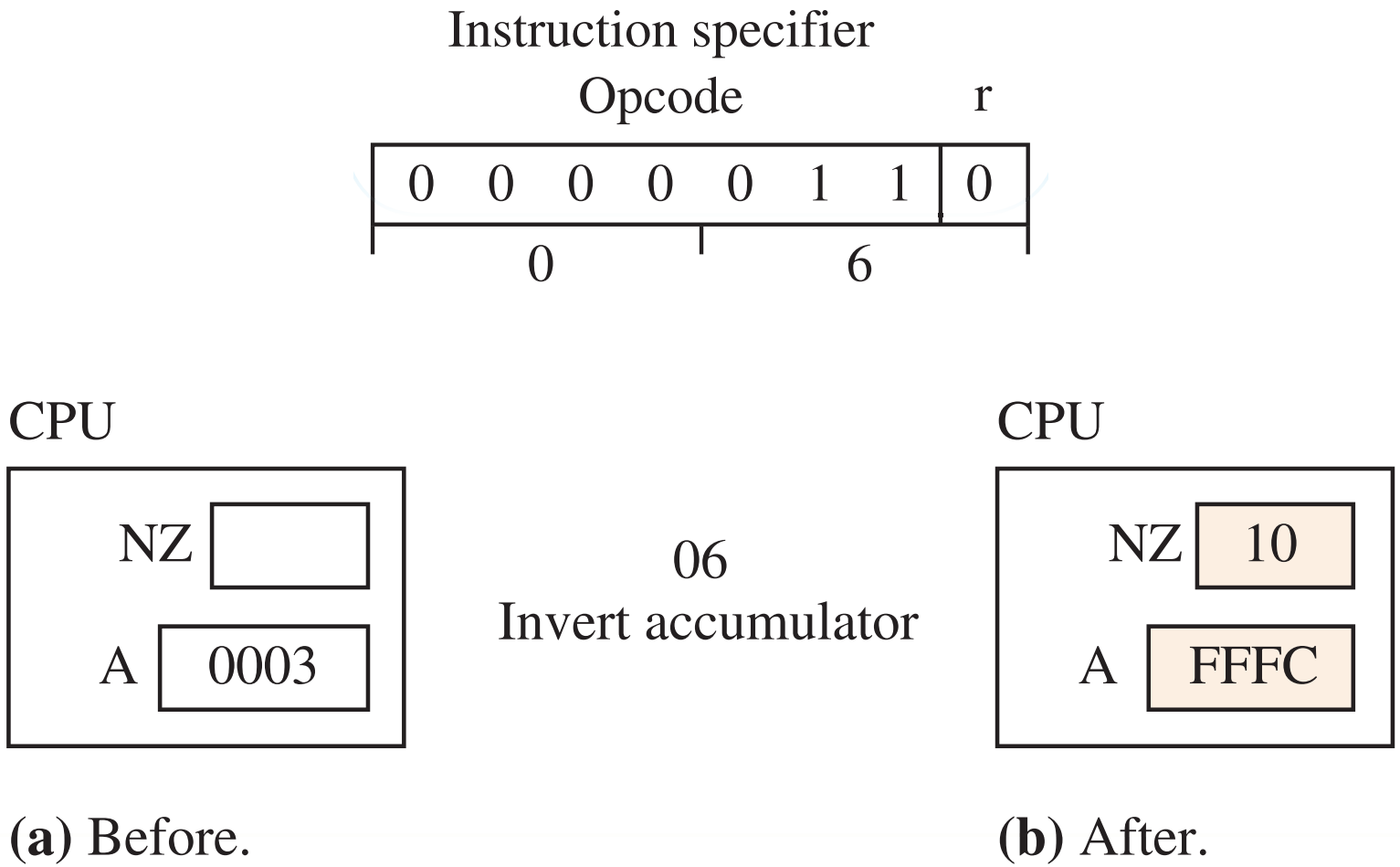
(a) Before.

(b) After.

The invert instruction

- Instruction specifier: 0000 011r
- Bit-wise NOT operation on register r
- Each 0 changed to 1, each 1 changed to 0

$$r \leftarrow \neg r \; ; \; N \leftarrow r < 0 \; , \; Z \leftarrow r = 0$$



(a) Before.

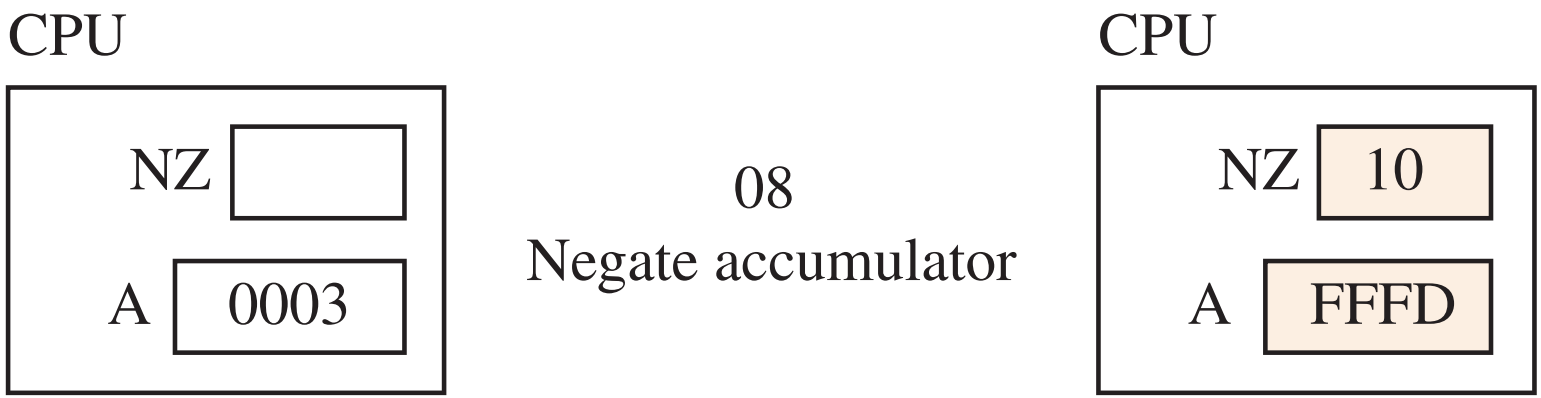
(b) After.

The negate instruction

- Instruction specifier: 0000 100r
- Negate (take two's complement of) register r

$$r \leftarrow -r\ ;\ N \leftarrow r < 0\ ,\ Z \leftarrow r = 0$$

Figure 4.23



(a) Before.

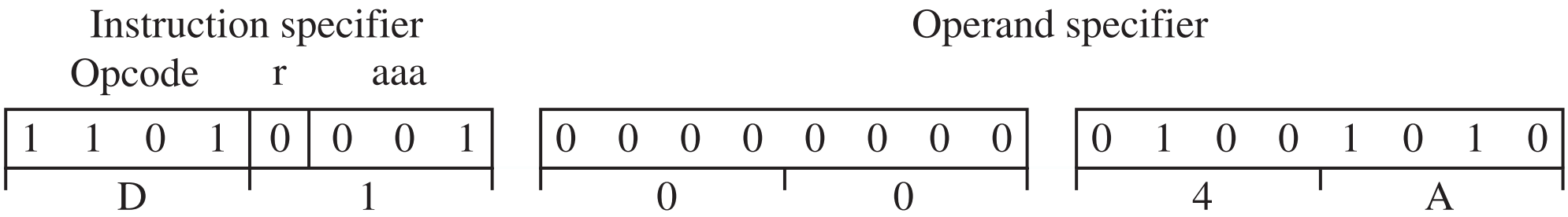
(b) After.

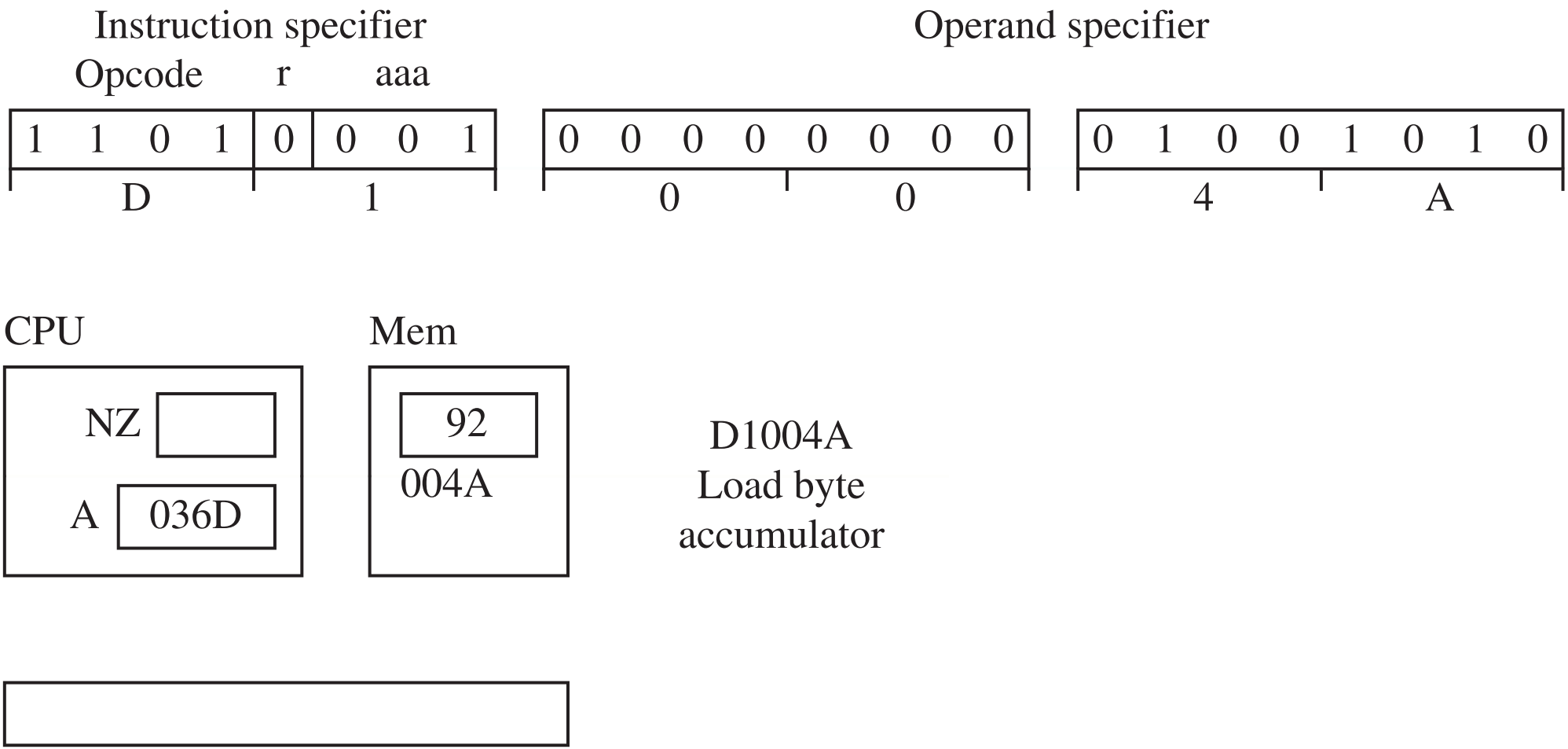
The load byte instruction

- Instruction specifier: 1101 raaa
- Loads one byte from memory to the right half of register r

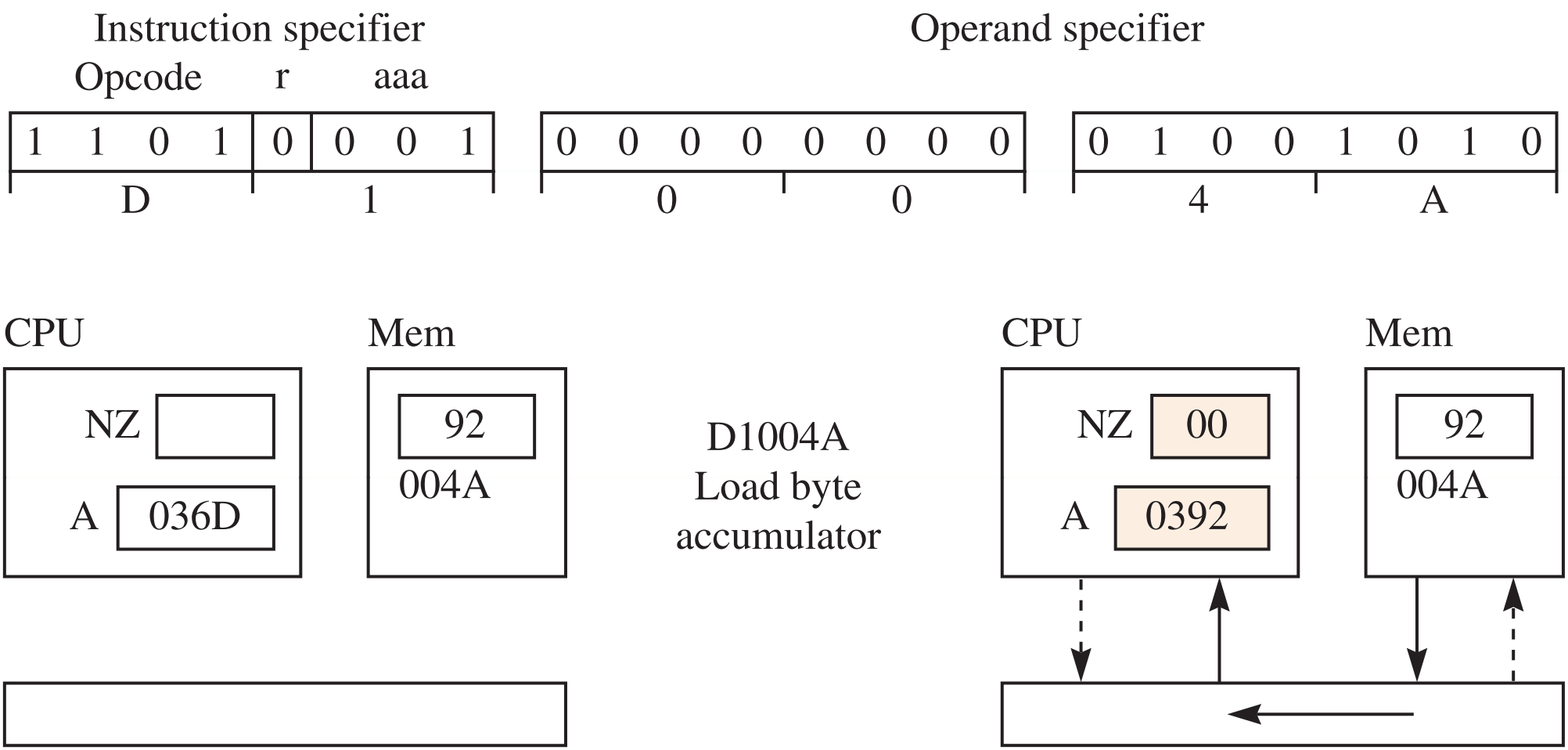
$$r\langle 8..15\rangle \leftarrow \text{byte Oprnd ; N} \leftarrow 0 \text{ , Z} \leftarrow r\langle 8..15\rangle = 0$$

Figure 4.24, 4.25





(a) Before.



(a) Before.

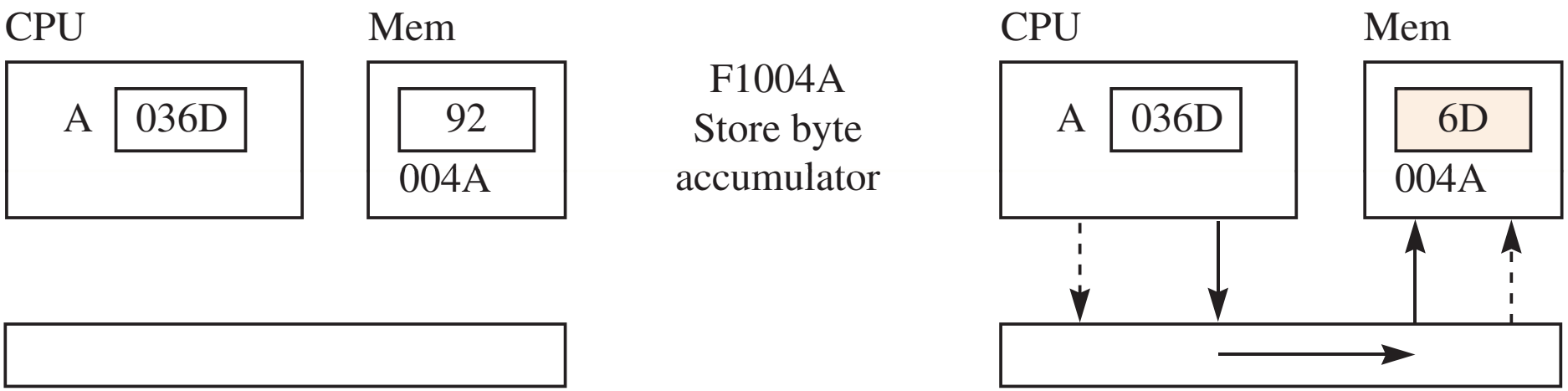
(b) After.

The store byte instruction

- Instruction specifier: 1111 raaa
- Stores one byte from the right half of register r to memory

$$\text{byte Oprnd} \leftarrow r\langle 8..15 \rangle$$

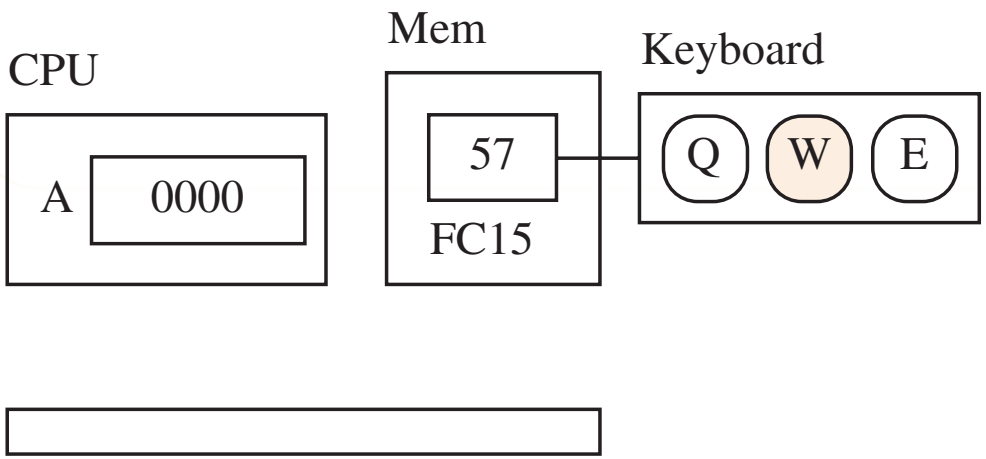
Figure 4.26



(a) Before.

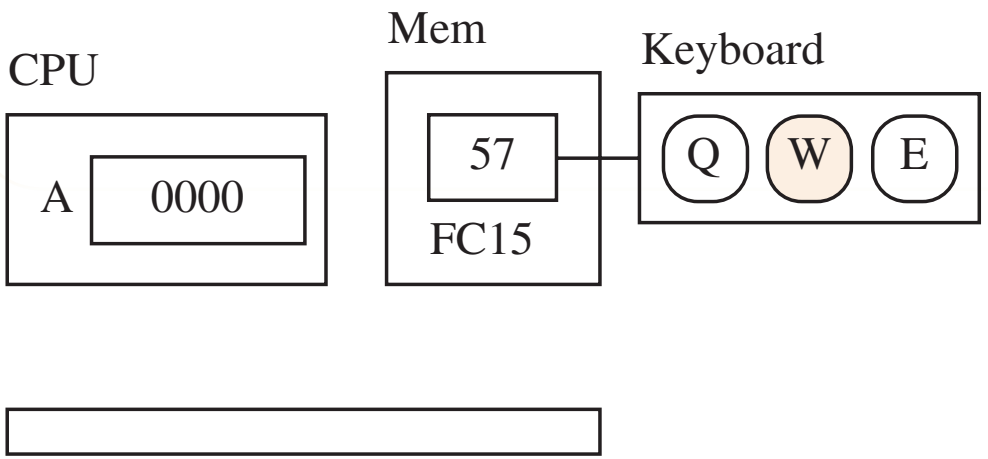
(b) After.

Load byte from input device

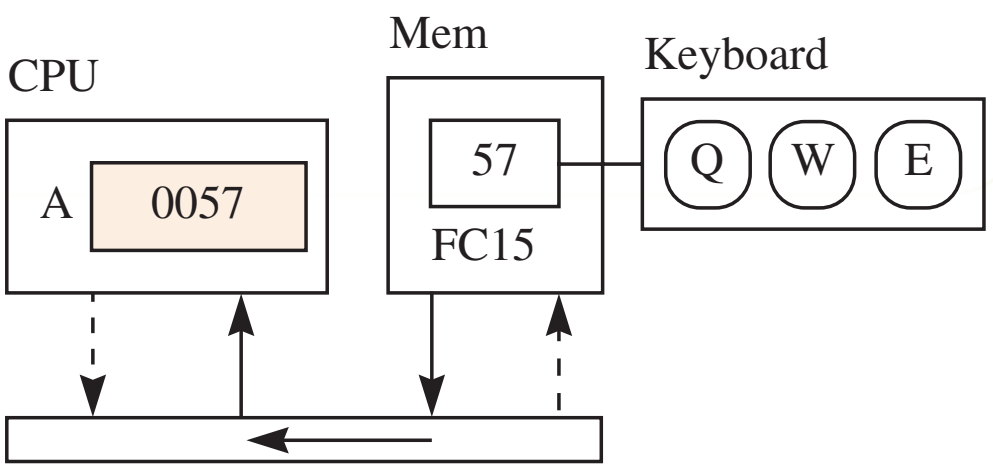


(a) Before.

Load byte from input device

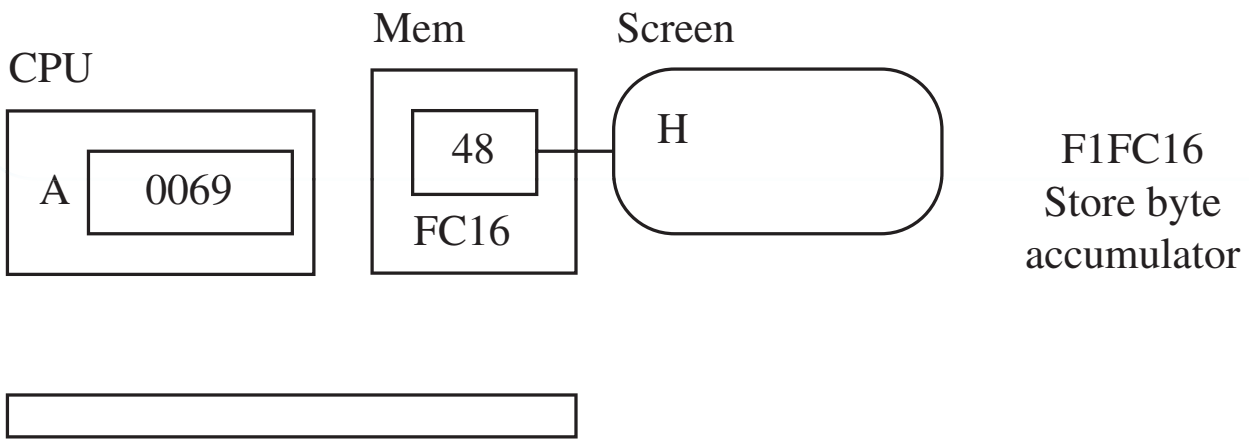


(a) Before.



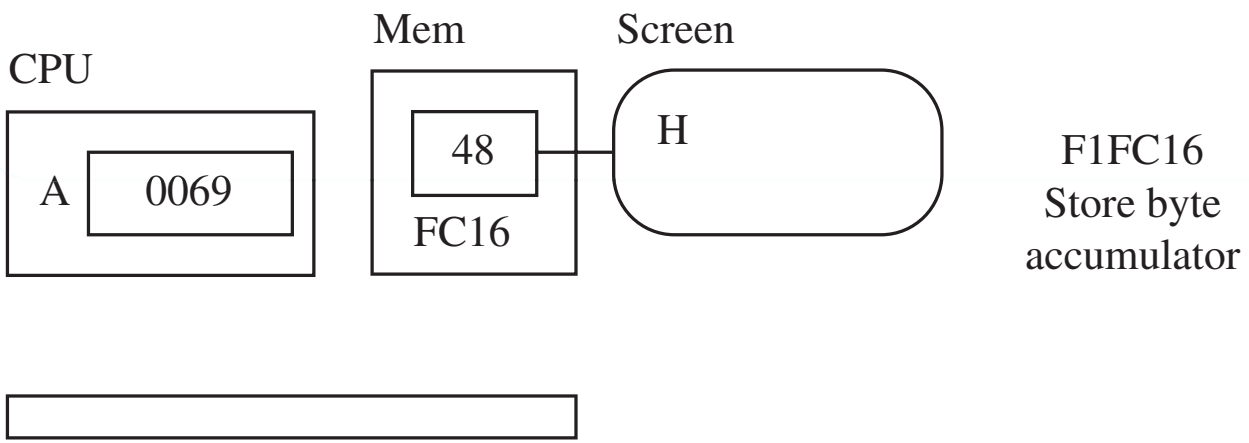
(b) After.

Store byte to output device

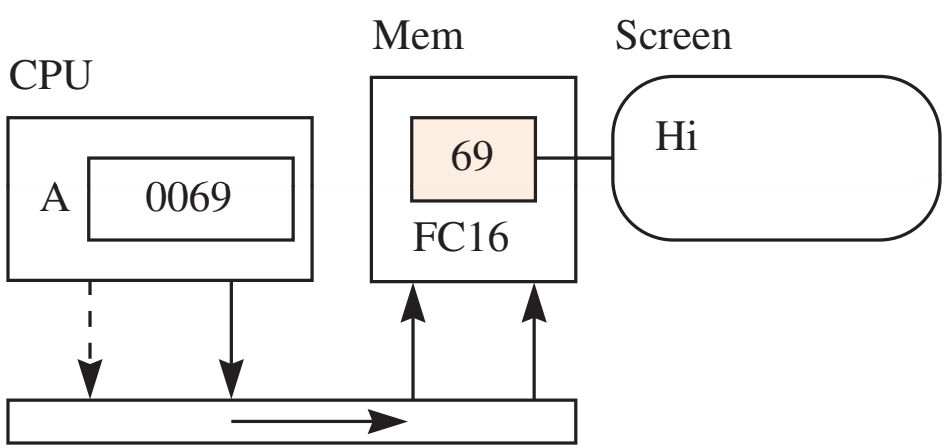


(a) Before.

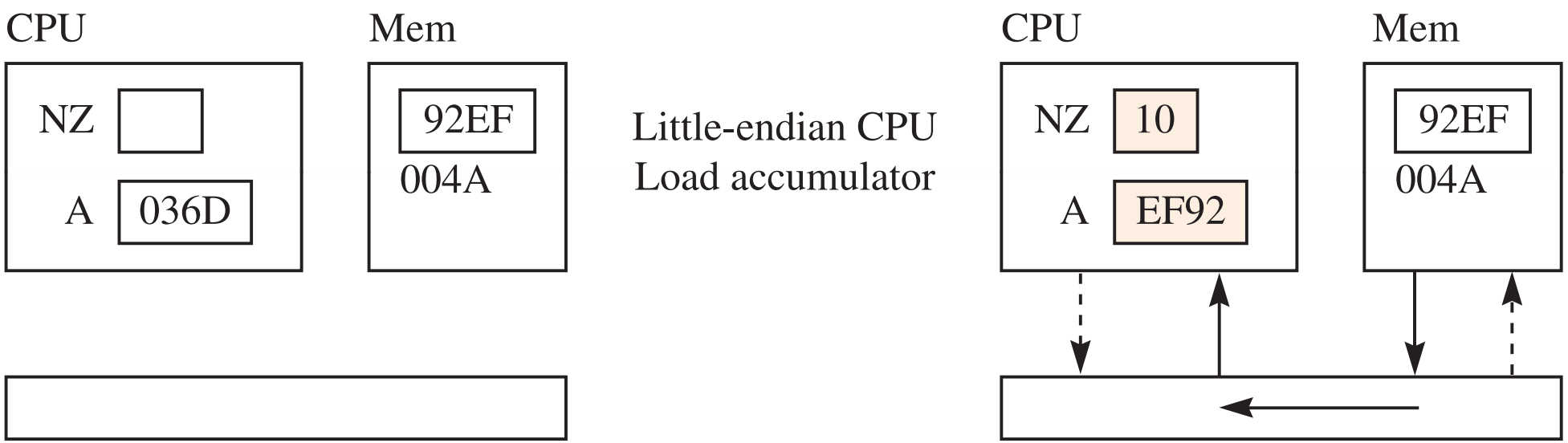
Store byte to output device



(a) Before.



A little-endian CPU



(a) Before.

(b) After.

A 32-bit register load

	Initial State	Big Endian Final State	Little Endian Final State
Mem[019E]	89	89	89
Mem[019F]	AB	AB	AB
Mem[01A0]	CD	CD	CD
Mem[01A1]	EF	EF	EF
Accumulator		89 AB CD EF	EF CD AB 89

The von Neumann execution cycle

- Fetch instruction at Mem[PC].
- Decode instruction fetched.
- Increment PC.
- Execute the instruction fetched.
- Repeat.

Figure 4.31

Load the machine language program
Initialize PC *and* SP
do {
    Fetch the next instruction
    Decode the instruction specifier
    Increment PC
    Execute the instruction fetched
}
while (*the stop instruction does not execute***)**

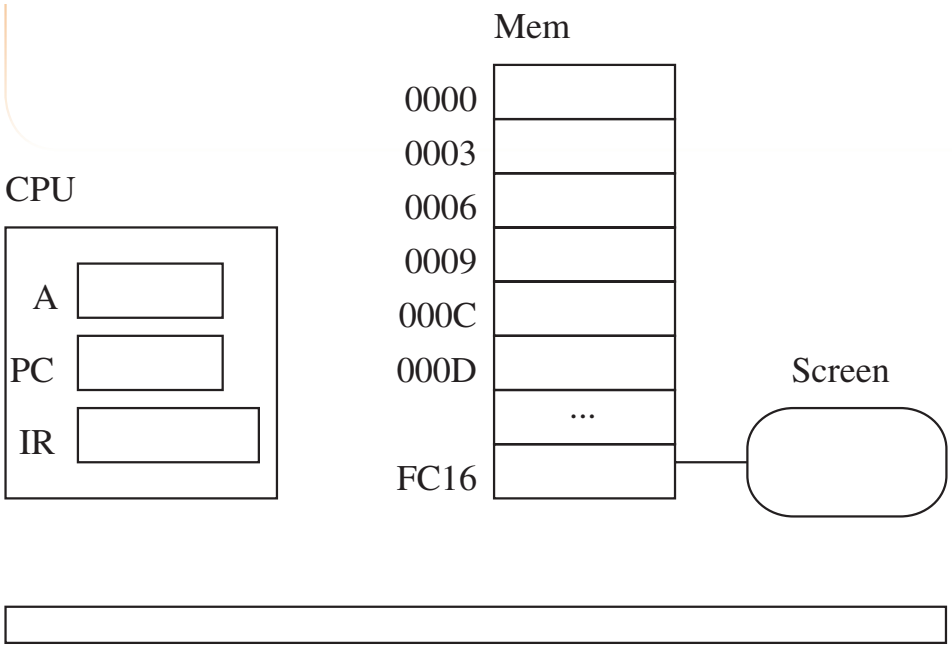
Load the machine language program into memory starting at address 0000
PC ← 0000
SP ← Mem[FFF4]
do {
    Fetch the instruction specifier at address in PC
    PC ← PC + 1
    Decode the instruction specifier
    if (*the instruction is not unary***) {**
        Fetch the operand specifier at address in PC
        PC ← PC + 2
    }
    Execute the instruction fetched
}
while ((*the stop instruction does not execute***) &&**
        (*the instruction is legal***))**

Address	Machine Language (bin)
0000	1101 0001 0000 0000 0000 1101
0003	1111 0001 1111 1100 0001 0110
0006	1101 0001 0000 0000 0000 1110
0009	1111 0001 1111 1100 0001 0110
000C	0000 0000
000D	0100 1000 0110 1001
000F	

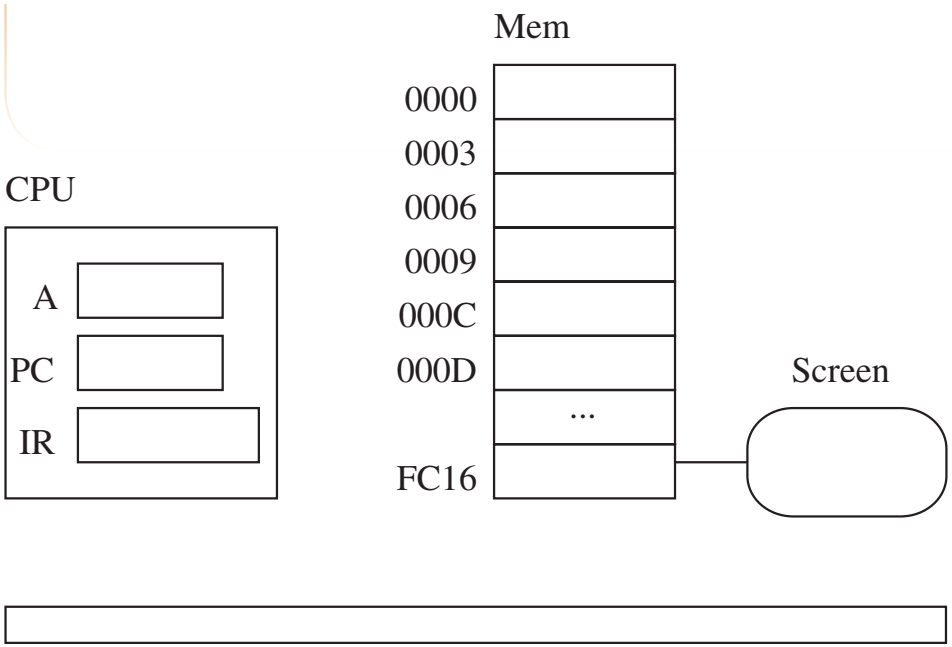
Address	Machine Language (hex)	
0000	D1000D	;Load byte accumulator 'H'
0003	F1FC16	;Store byte accumulator output device
0006	D1000E	;Load byte accumulator 'i'
0009	F1FC16	;Store byte accumulator output device
000C	00	;Stop
000D	4869	;ASCII "Hi" characters

Output

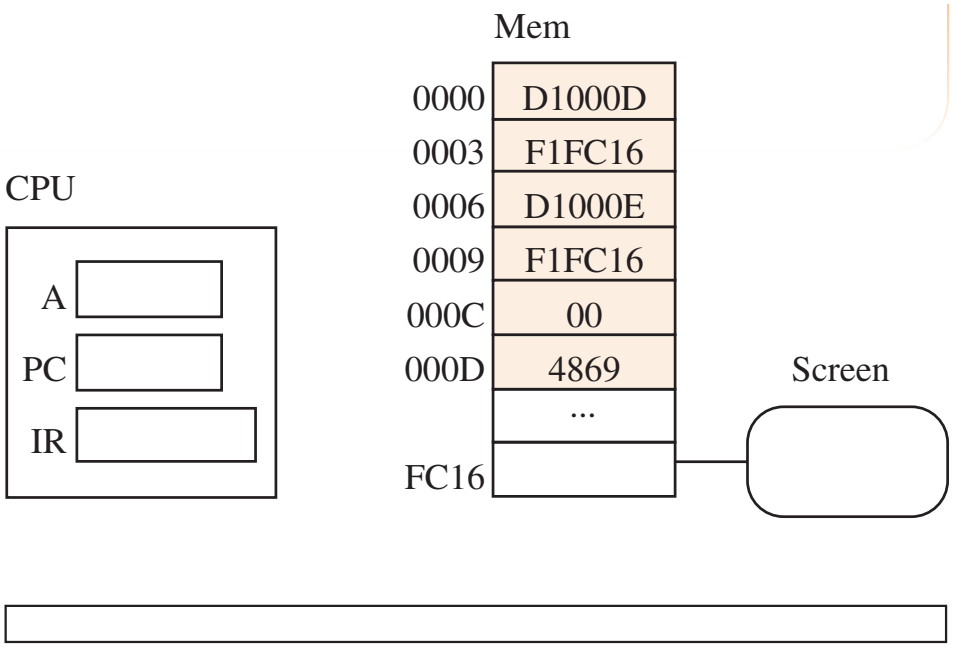
```
Hi
```



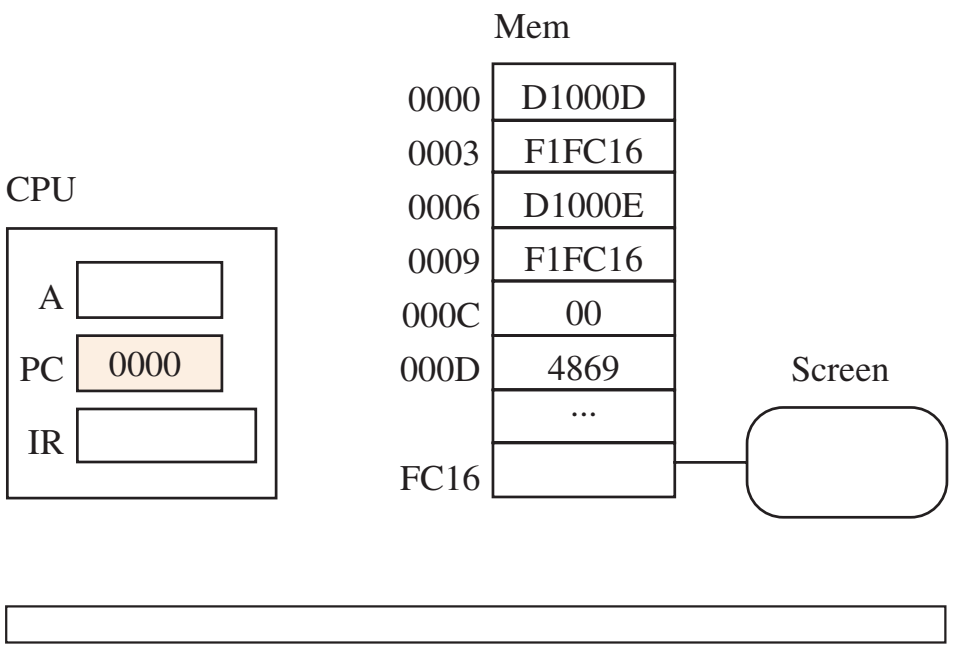
(a) Initial state before loading.



(a) Initial state before loading.

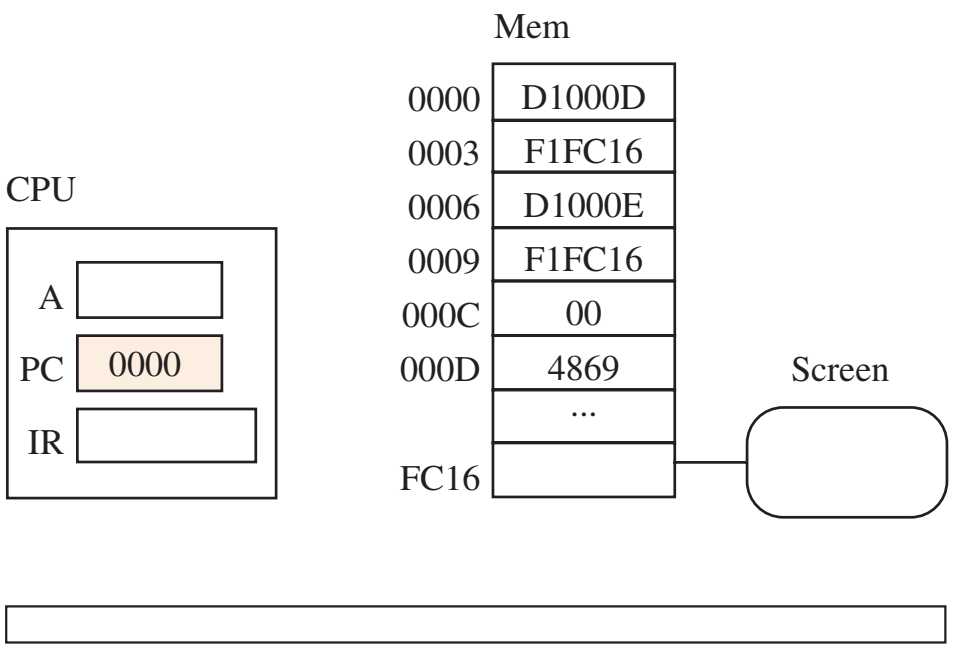


(b) Program loaded into main memory.

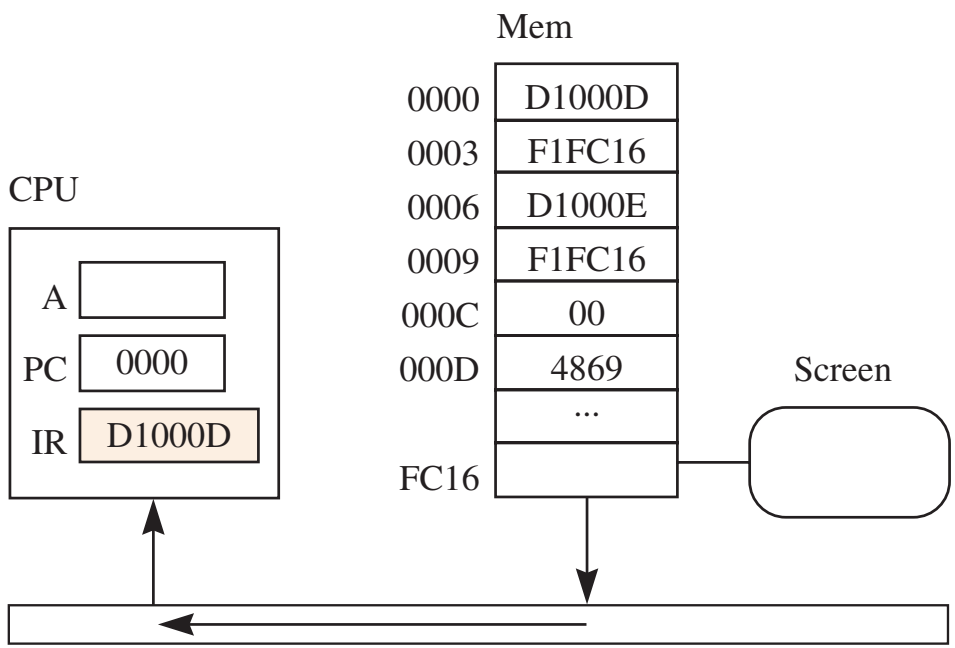


(c) PC ← 0000 (hex).

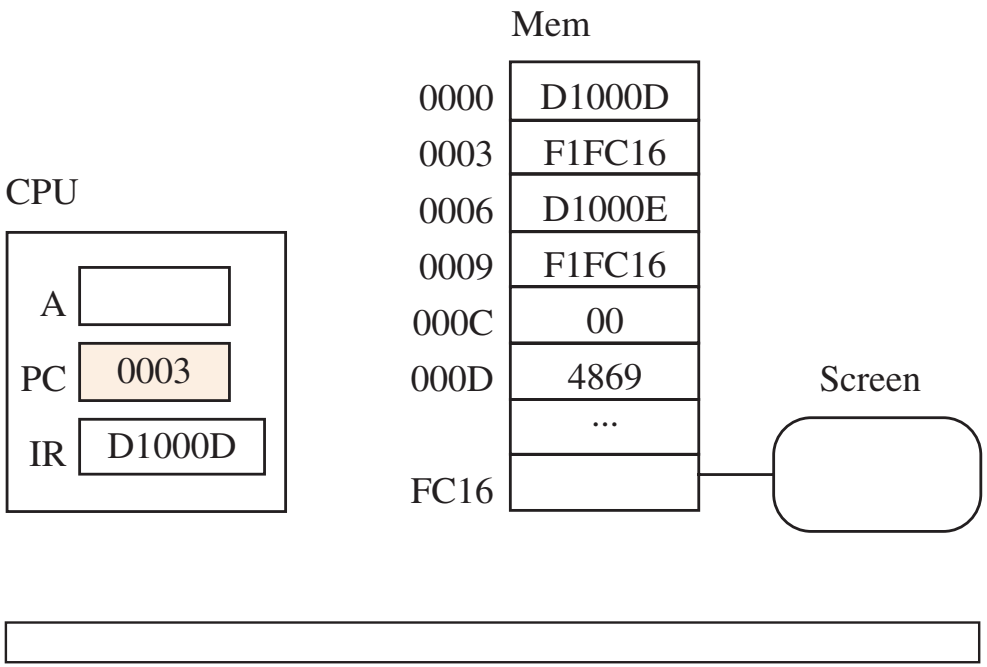
Figure 4.34 (continued)



(c) PC ← 0000 (hex).

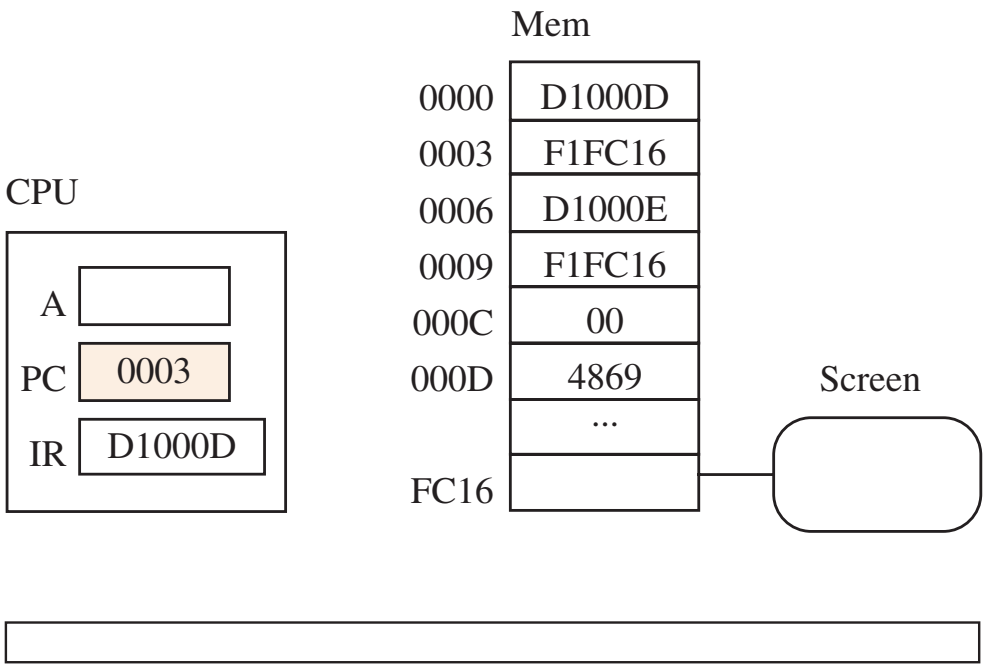


(d) Fetch instruction at Mem[PC].

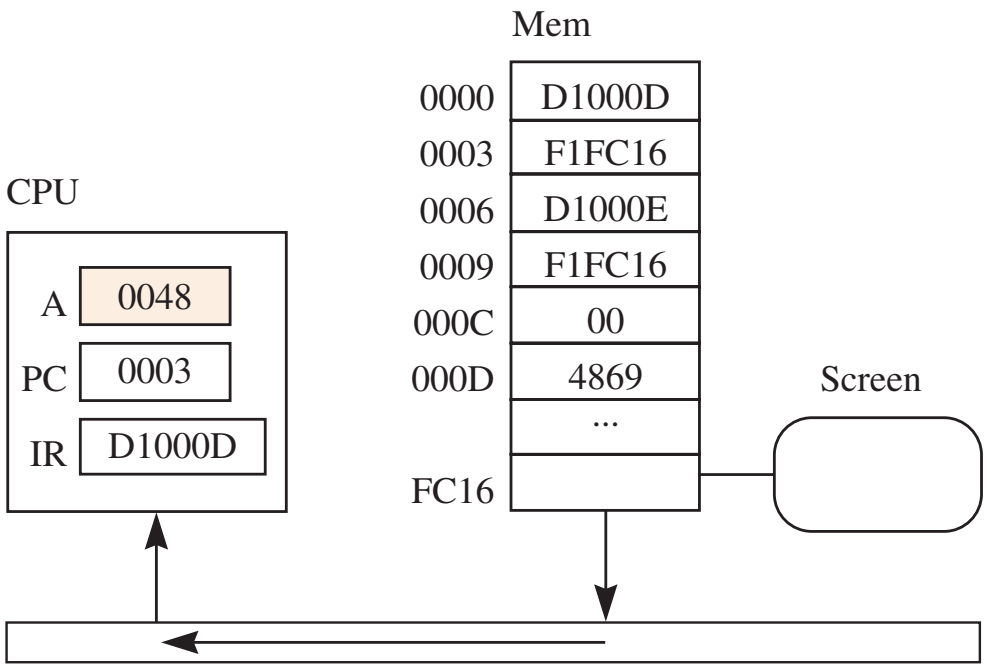


(e) Increment PC.

Figure 4.34 (continued)

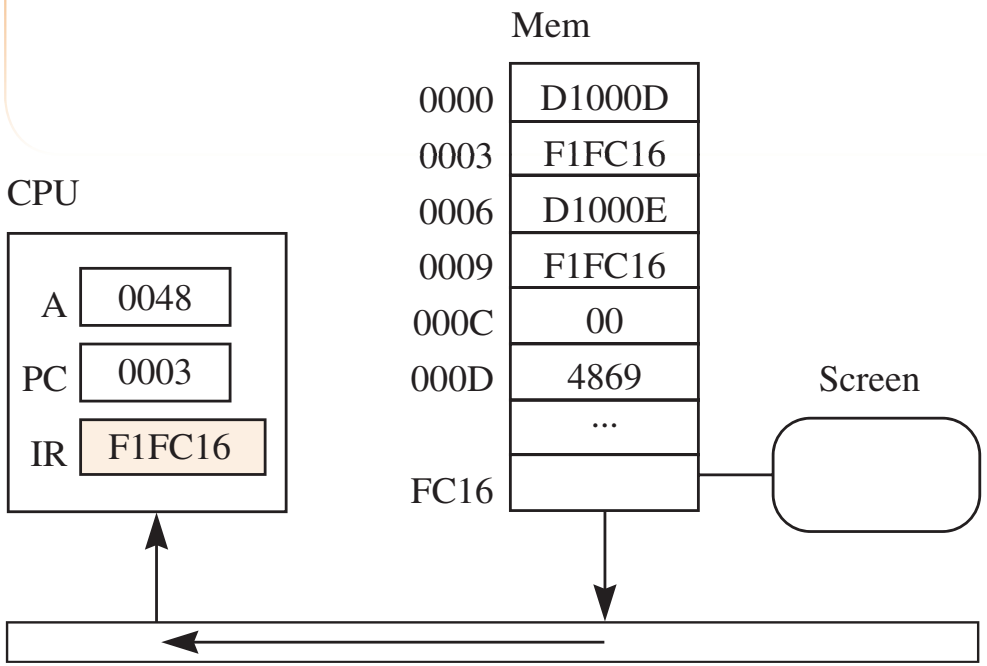


(e) Increment PC.



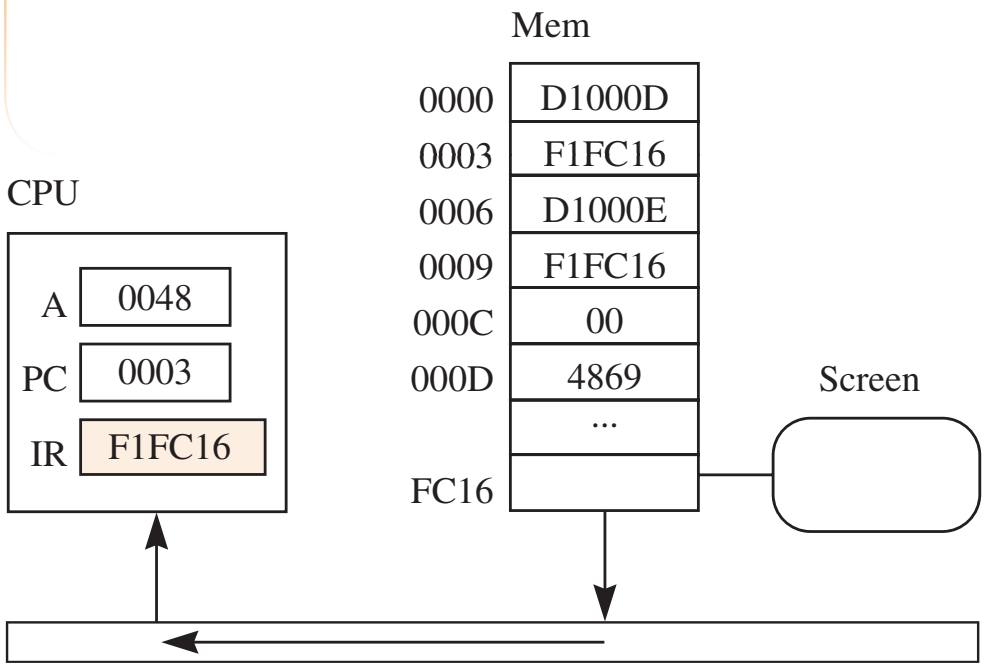
(f) Execute. Load byte for H to accumulator.

Figure 4.34 (continued)

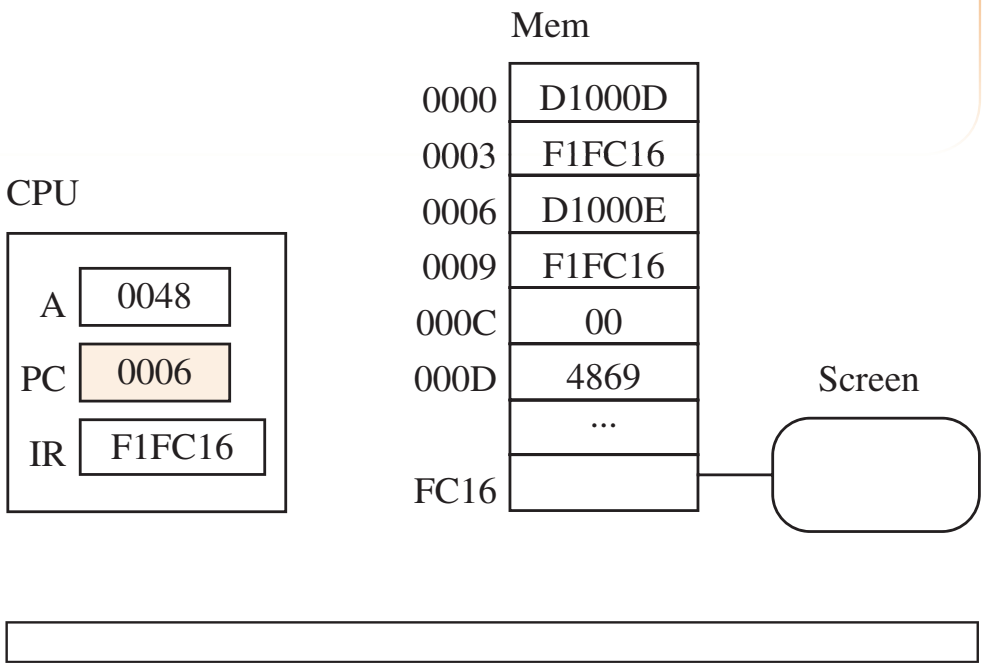


(g) Fetch instruction at Mem[PC].

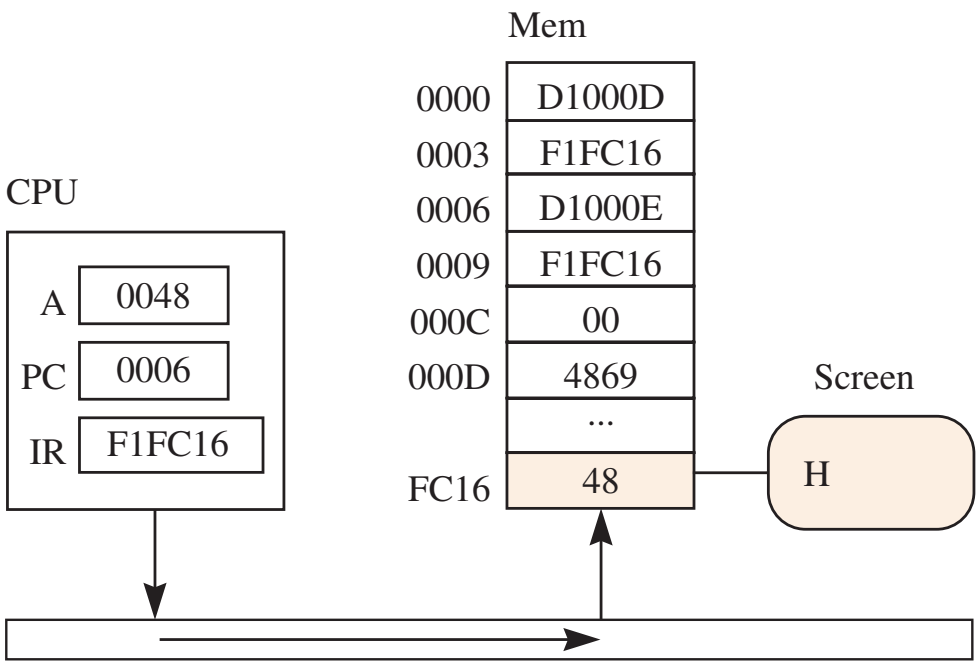
Figure 4.34 (continued)



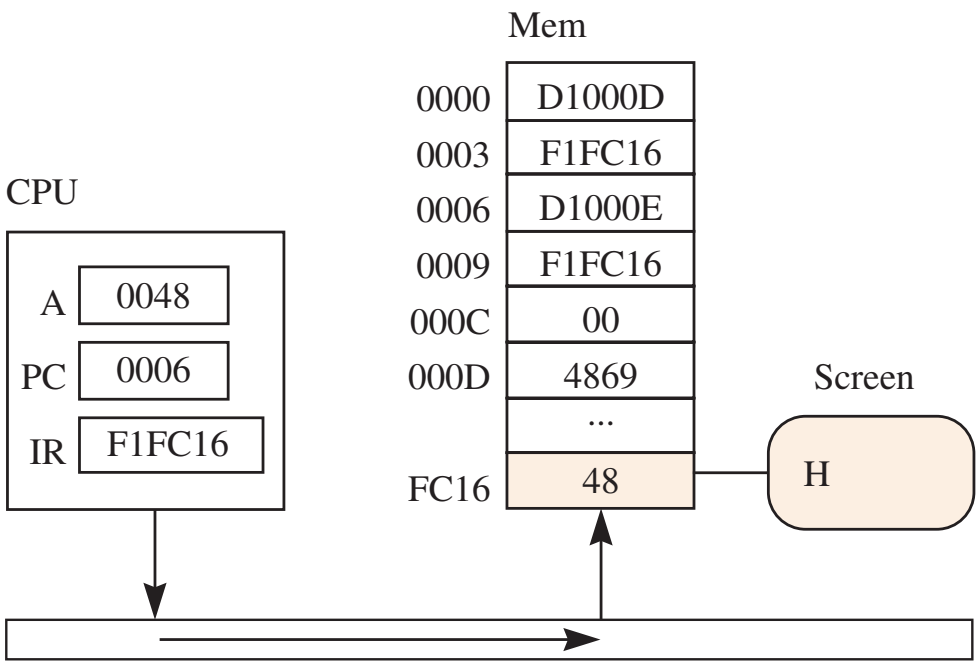
(g) Fetch instruction at Mem[PC].



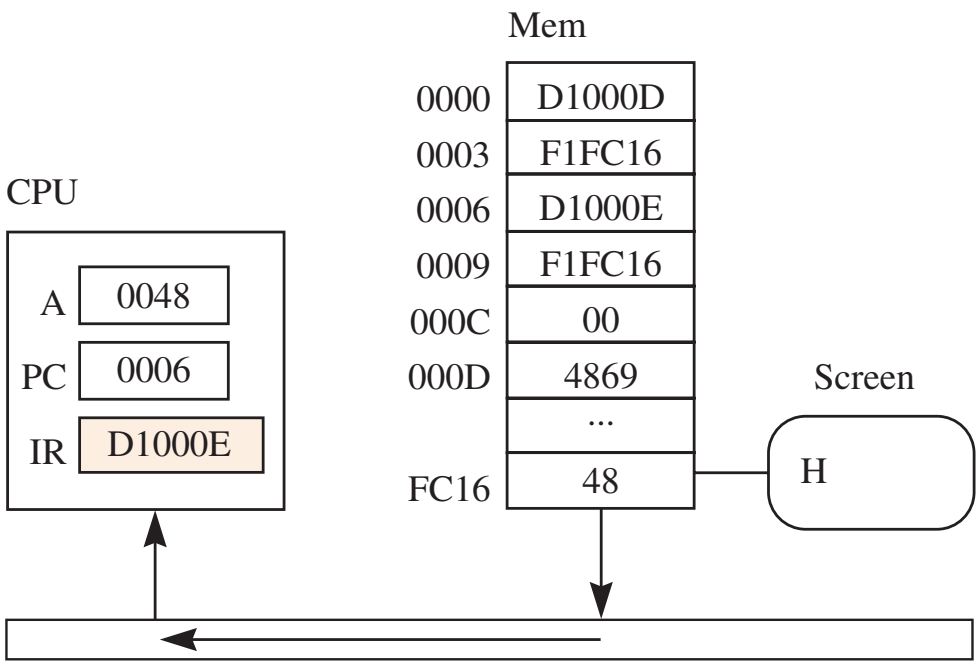
(h) Increment PC.



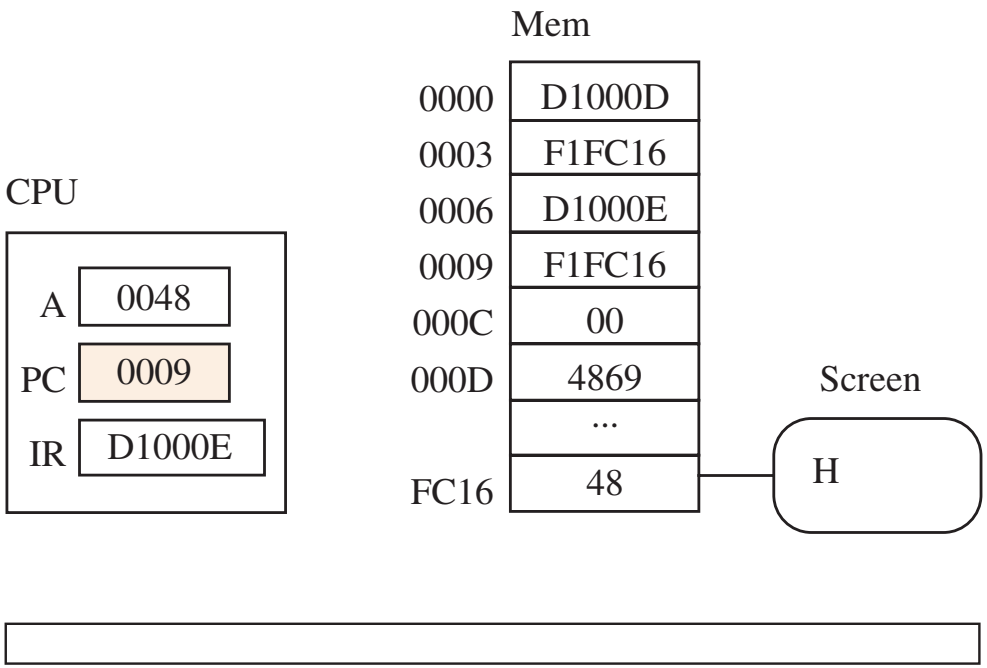
(i) Execute. Store byte from accumulator to output device.



(i) Execute. Store byte from accumulator to output device.

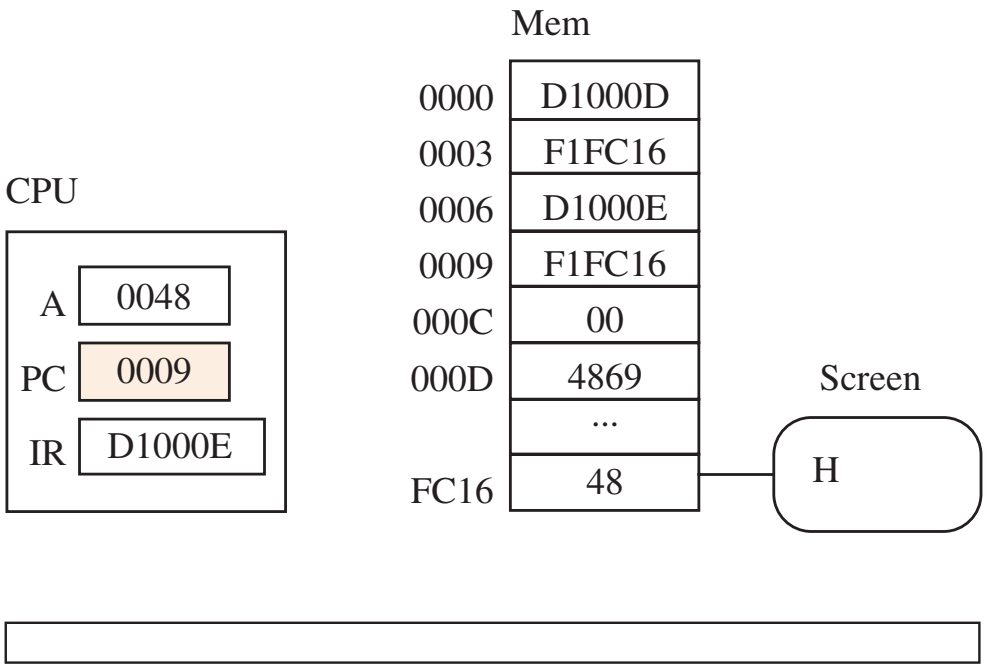


(j) Fetch instruction at Mem[PC].

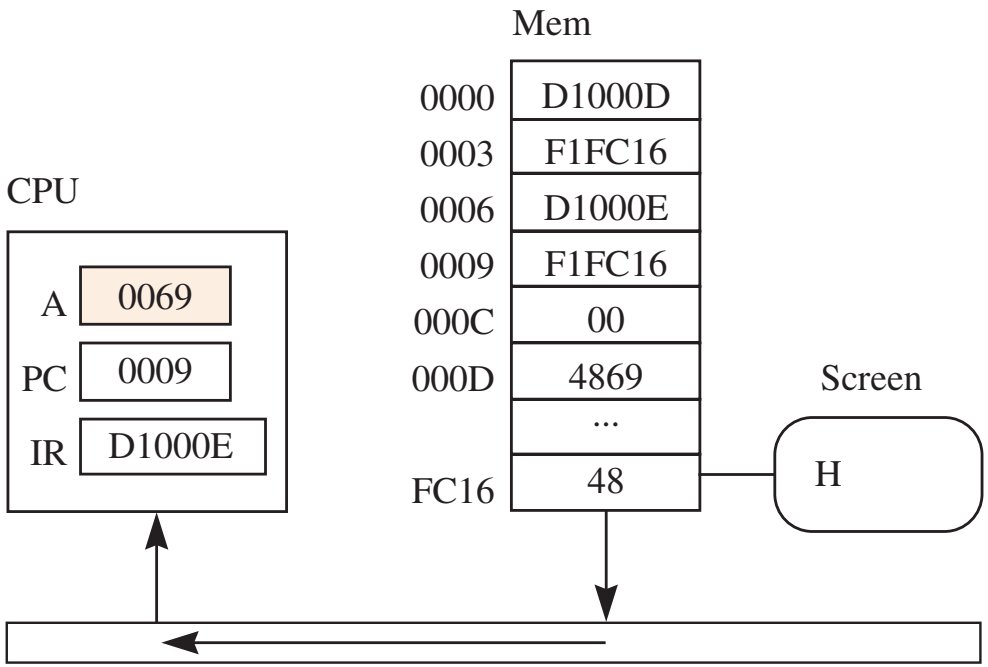


(k) Increment PC.

Figure 4.34 (continued)



(k) Increment PC.



(l) Execute. Load byte for i to accumulator.

Address	Machine Language (bin)
0000	1101 0001 1111 1100 0001 0101
0003	1111 0001 0000 0000 0001 0011
0006	1101 0001 1111 1100 0001 0101
0009	1111 0001 1111 1100 0001 0110
000C	1101 0001 0000 0000 0001 0011
000F	1111 0001 1111 1100 0001 0110
0012	0000 0000

Address	Machine Language (hex)	
0000	D1FC15	;Input first character
0003	F10013	;Store first character
0006	D1FC15	;Input second character
0009	F1FC16	;Output second character
000C	D10013	;Load first character
000F	F1FC16	;Output first character
0012	00	;Stop

Input

```
up
```

Output

```
pu
```

Figure 4.36

Address	Machine Language (bin)
0000	1100 0001 0000 0000 0000 1101
0003	0110 0001 0000 0000 0000 1111
0006	1001 0001 0000 0000 0001 0001
0009	1111 0001 1111 1100 0001 0110
000C	0000 0000
000D	0000 0000 0000 0101
000F	0000 0000 0000 0011
0011	0000 0000 0011 0000

Address	Machine Language (hex)	
0000	C1000D	;A <- first number
0003	61000F	;Add the two numbers
0006	910011	;Convert sum to character
0009	F1FC16	;Output the character
000C	00	;Stop
000D	0005	;Decimal 5
000F	0003	;Decimal 3
0011	0030	;Mask for ASCII char

Output

```
8
```

Figure 4.37

Address	Machine Language (bin)
0000	1101 0001 0000 0000 0001 1001
0003	1111 0001 0000 0000 0000 1001
0006	1100 0001 0000 0000 0001 0011
0009	0110 0001 0000 0000 0001 0101
000C	1001 0001 0000 0000 0001 0111
000F	1111 0001 1111 1100 0001 0110
0012	0000 0000
0013	0000 0000 0000 0101
0015	0000 0000 0000 0011
0017	0000 0000 0011 0000
0019	0111 0001

Figure 4.37 (continued)

```
Address     Machine Language (hex)
0000        D10019   ;Load byte accumulator
0003        F10009   ;Store byte accumulator
0006        C10013   ;A <- first number
0009        610015   ;Add the two numbers
000C        910017   ;Convert sum to character
000F        F1FC16   ;Output the character
0012        00       ;Stop
0013        0005     ;Decimal 5
0015        0003     ;Decimal 3
0017        0030     ;Mask for ASCII char
0019        71       ;Byte to modify instruction
```

Output

```
2
```

Intel x-86 computer architecture

Figure 4.38

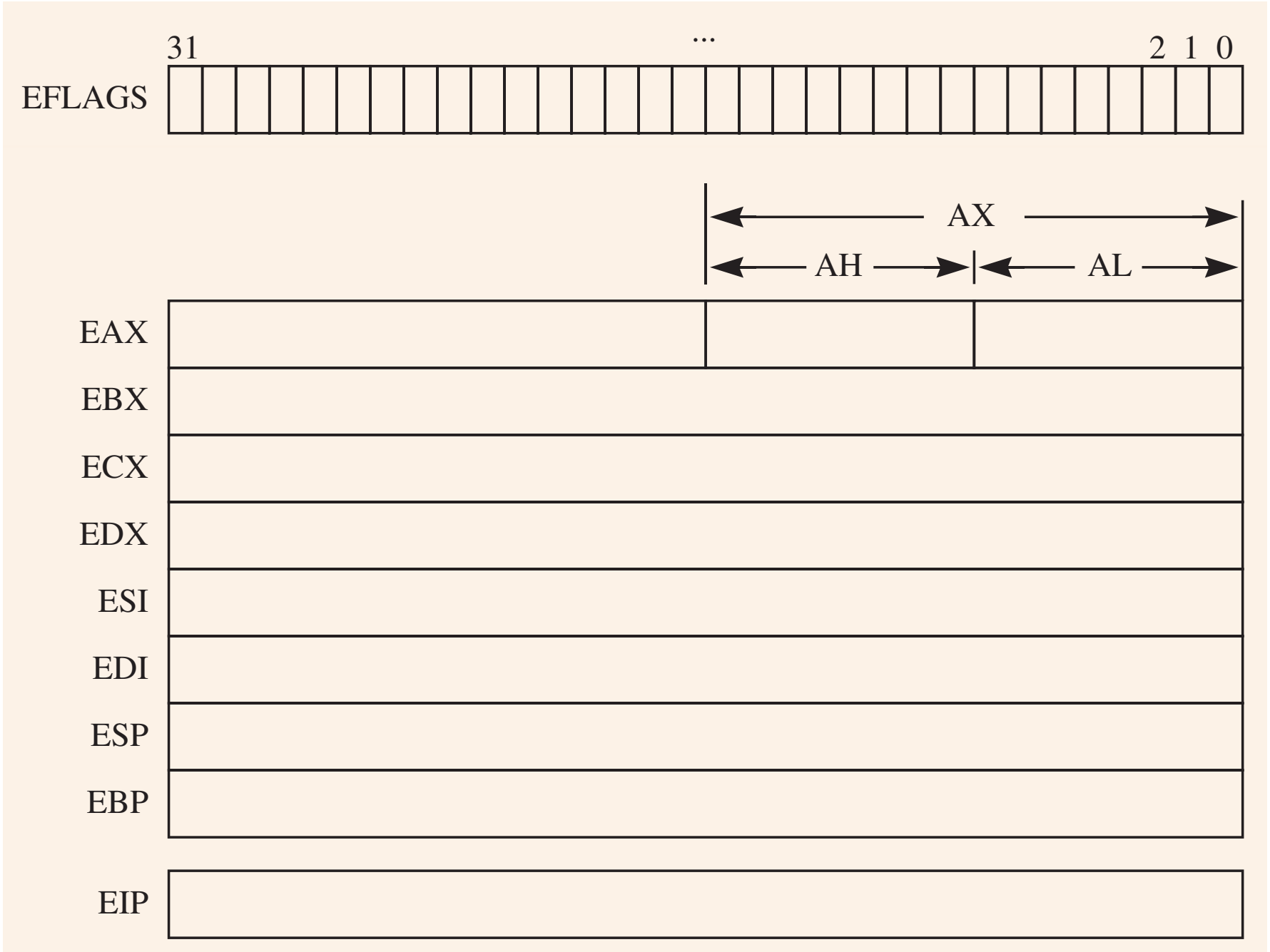
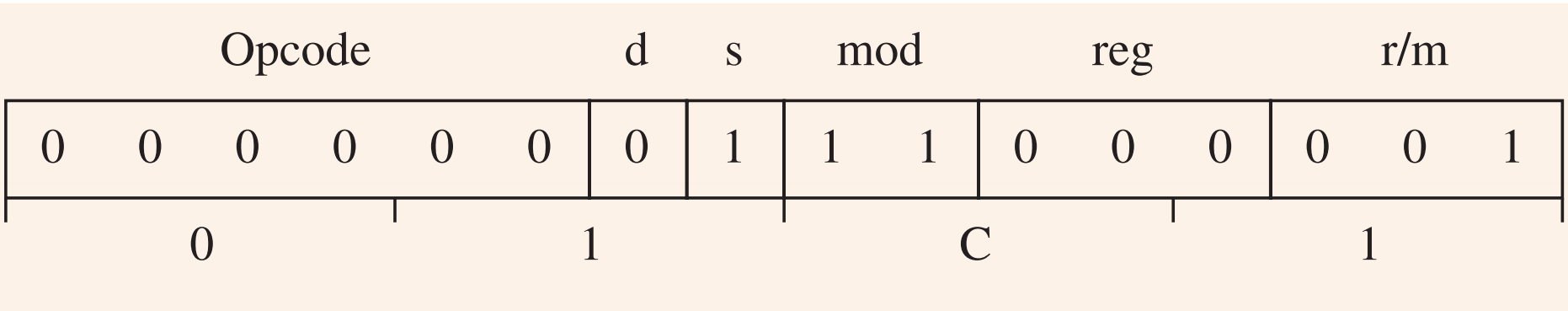


Figure 4.39

Status Bit	Intel Name	EFLAGS Position
N	SF	7
Z	ZF	6
V	OF	11
C	CF	0

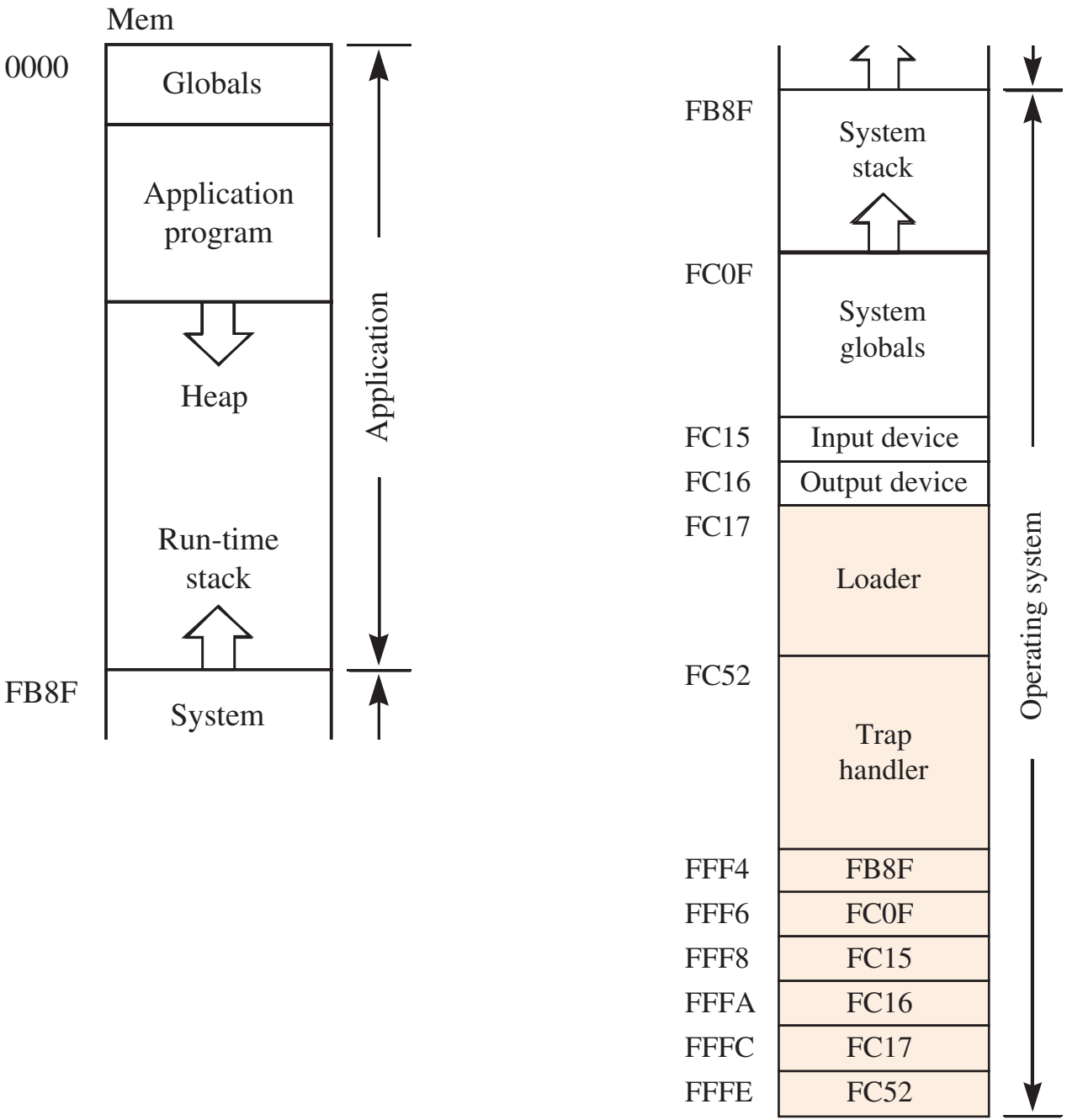
x86 add register instruction



Memory devices

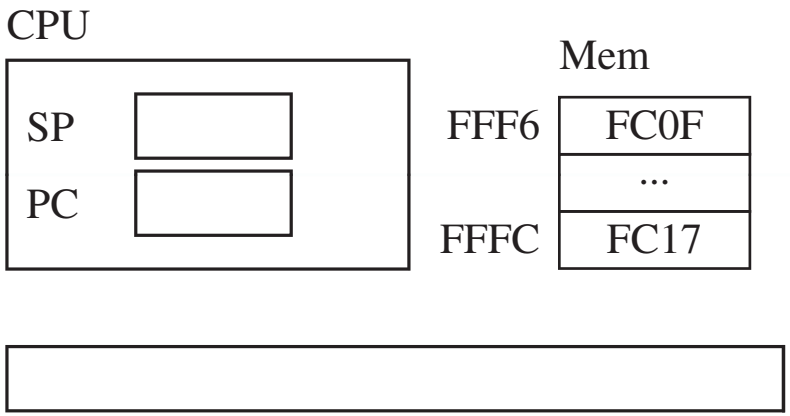
- Read/Write memory
  - Also called Random-Access Memory (RAM)
  - Can load from RAM and store to RAM
- Read-Only memory (ROM)
  - Can load from ROM
  - Can*not* store to ROM
- RAM and ROM are *both* random

Figure 4.41

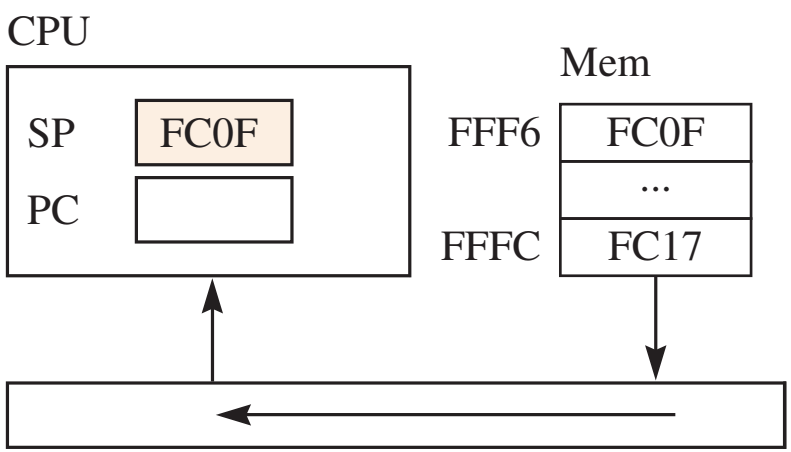


The load option

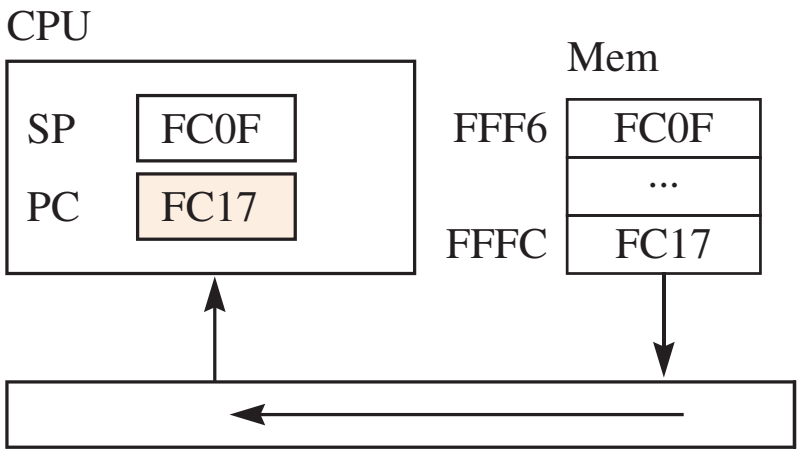
- SP ← Mem[FFF6]
- PC ← Mem[FFFC]
- Start the von Neumann cycle



(a) Initial state.



(b) SP ← Mem[FFF6]



(c) PC ← Mem[FFFC]

The execute option

- SP ← Mem[FFF4]
- PC ← 0000
- Start the von Neumann cycle

Figure 4.43

Address	Machine Language (hex)	
0000	D1000D	;Load byte accumulator 'H'
0003	F1FC16	;Store byte accumulator output device
0006	D1000E	;Load byte accumulator 'i'
0009	F1FC16	;Store byte accumulator output device
000C	00	;Stop
000D	4869	;ASCII "Hi" characters

Hex Version for the Loader

```
D1 00 0D F1 FC 16 D1 00 0E F1 FC 16 00 48 69 zz
```

Output

```
Hi
```