Computer Architecture

Figure 4.1

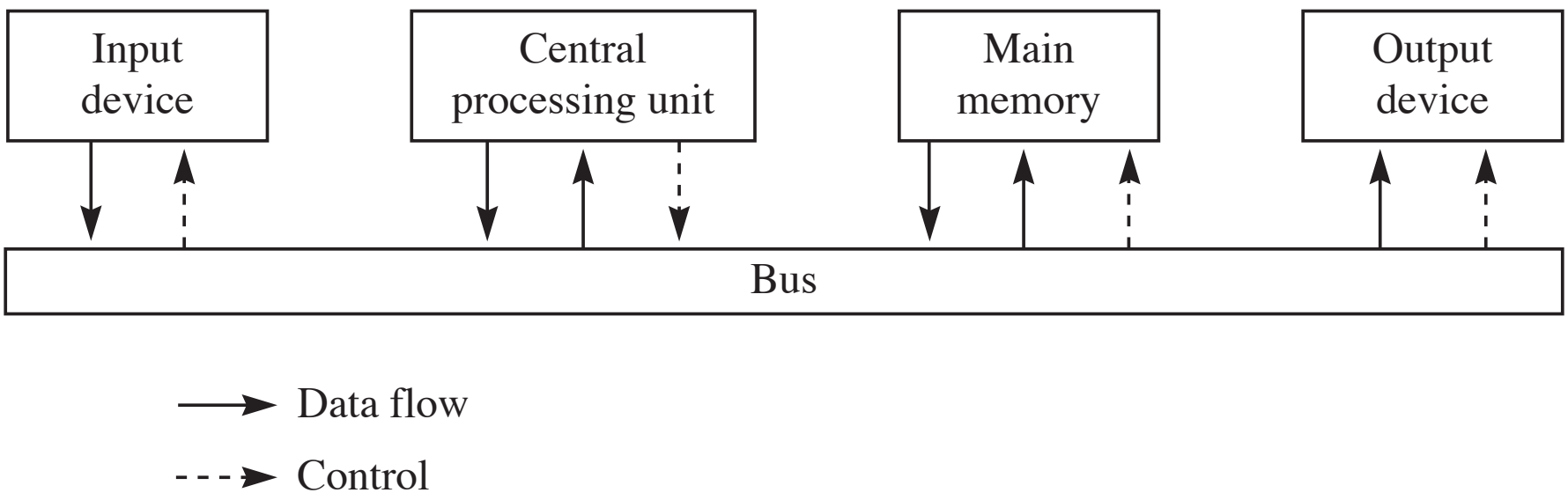


Figure 4.2

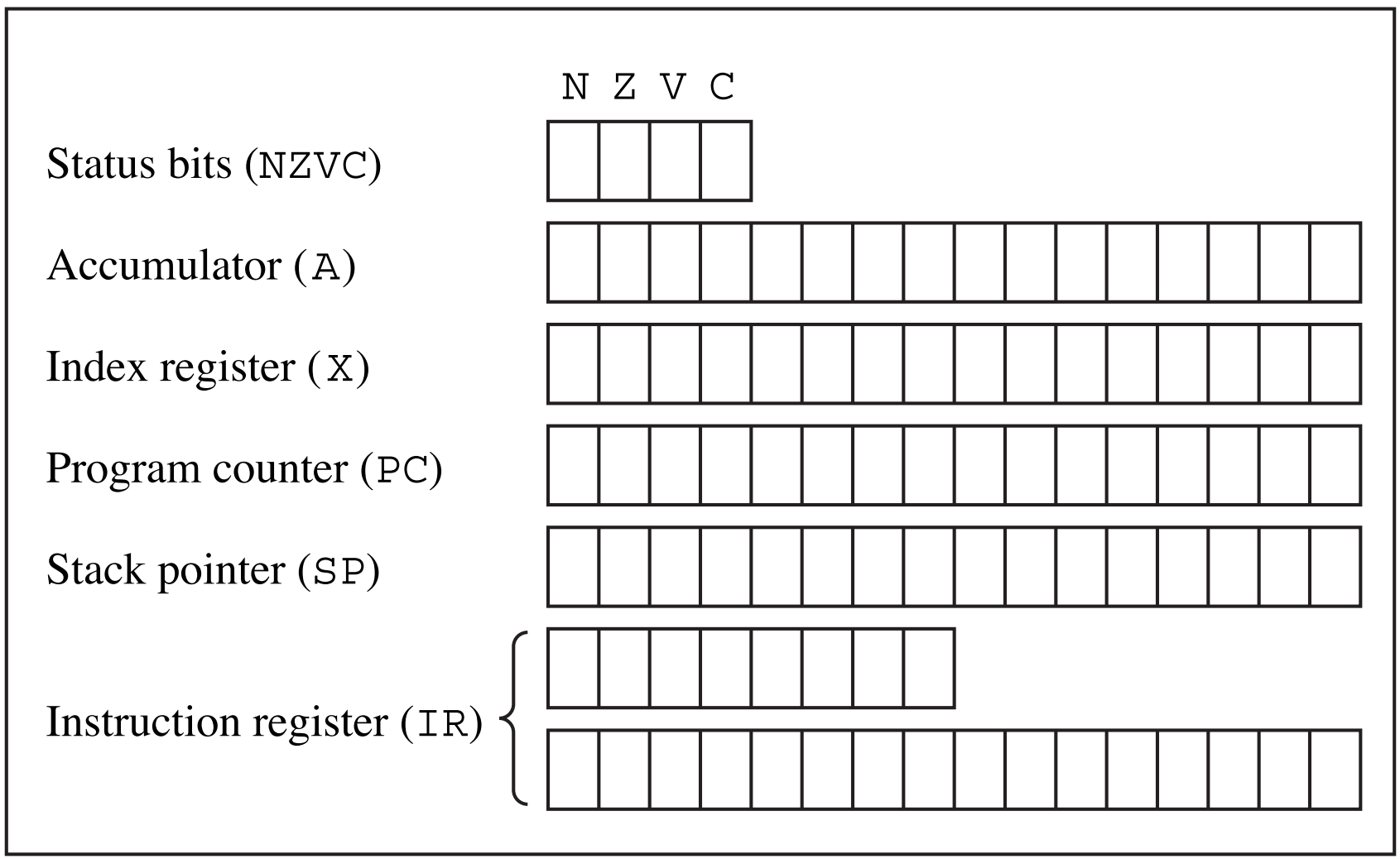


Figure 4.3

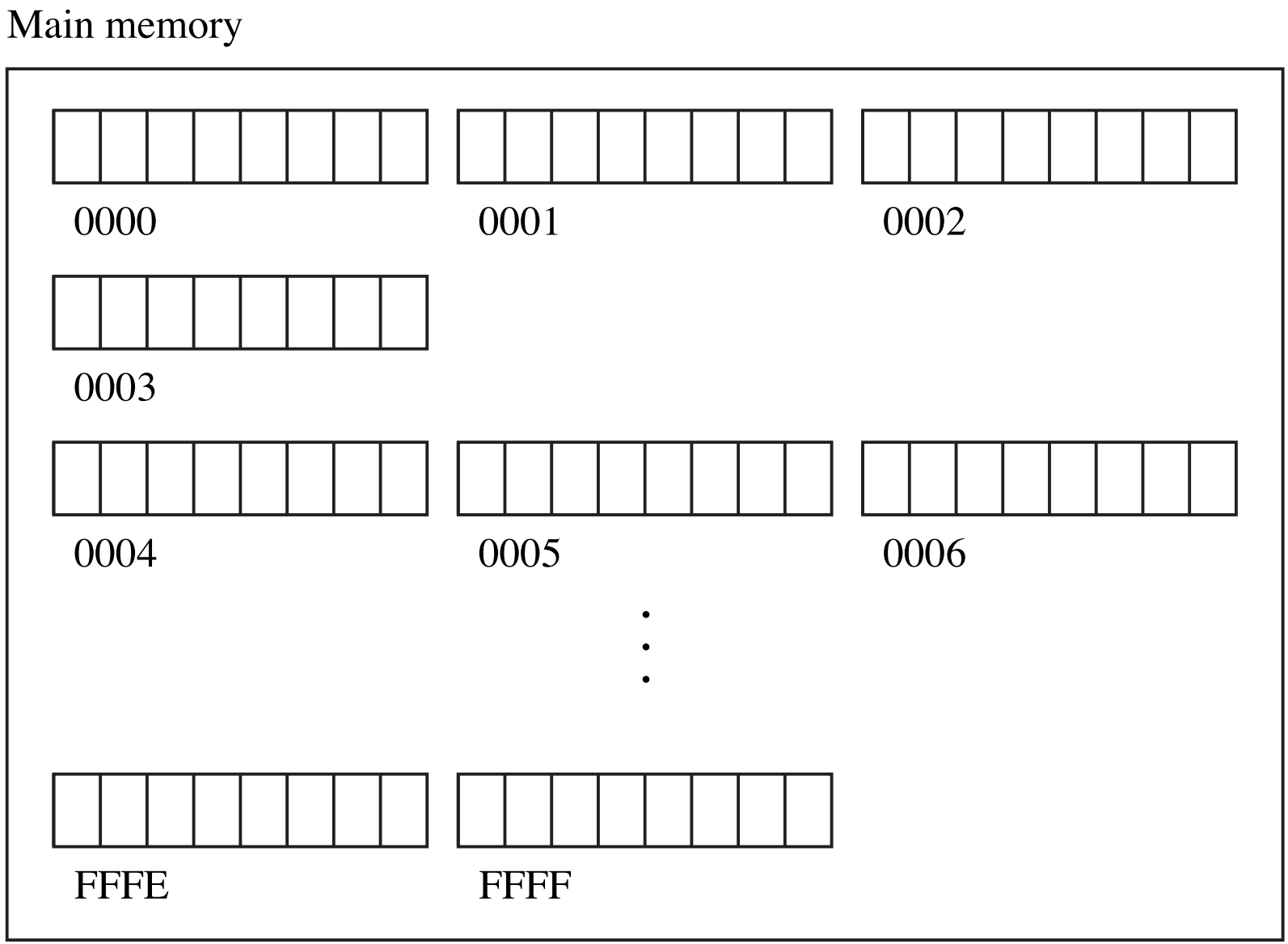


Figure 4.4

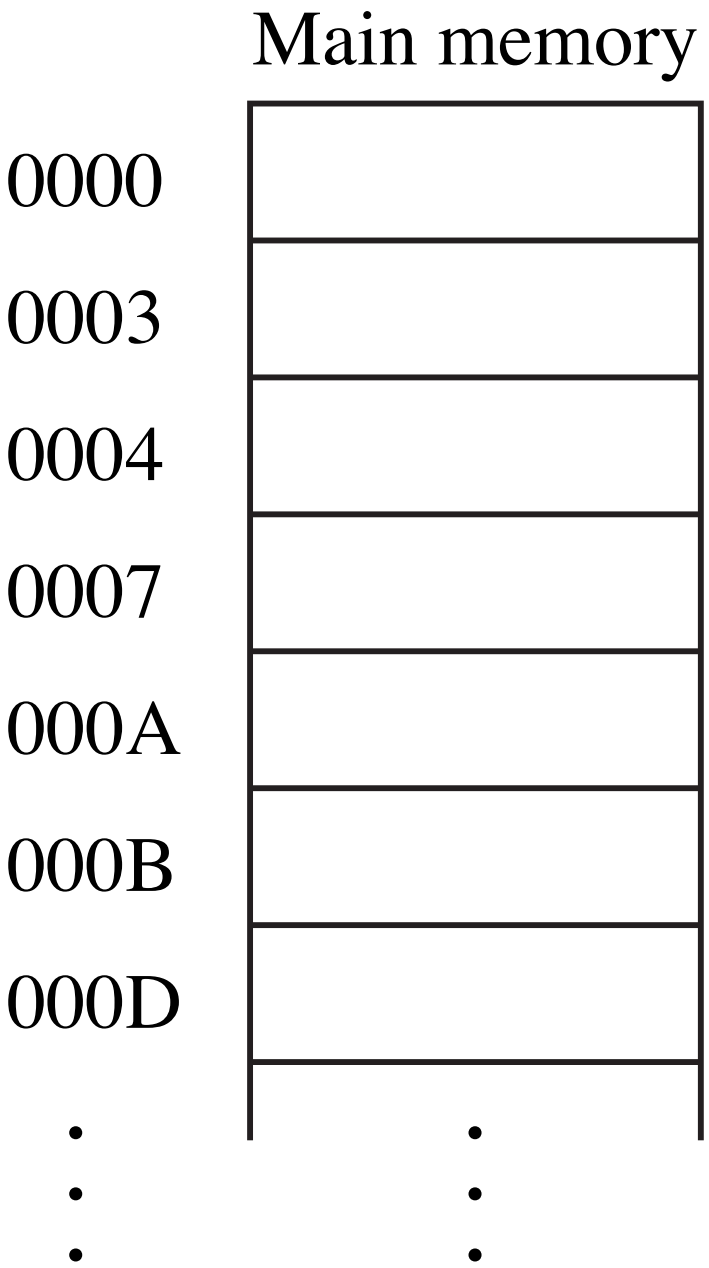
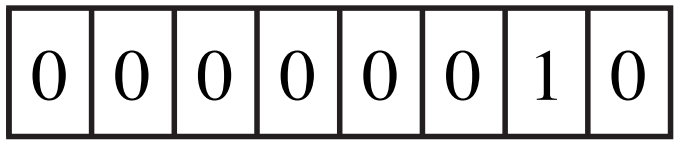
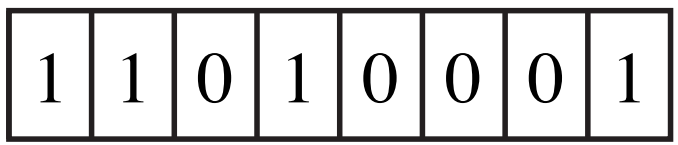
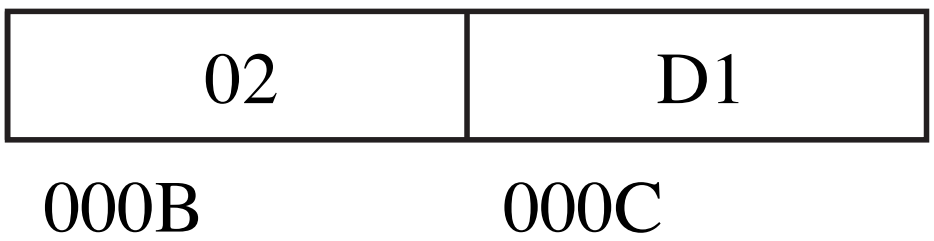


Figure 4.5





(a) The content in binary.



(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 trap
0000 0010	Move stack pointer (SP) to accumulator (A)
0000 0011	Move NZVC flags to accumulator (A)
0000 010a	Branch unconditional
0000 011a	Branch if less than or equal to
0000 100a	Branch if less than
0000 101a	Branch if equal to
0000 110a	Branch if not equal to
0000 111a	Branch if greater than or equal to
0001 000a	Branch if greater than
0001 001a	Branch if V
0001 010a	Branch if C
0001 011a	Call subroutine

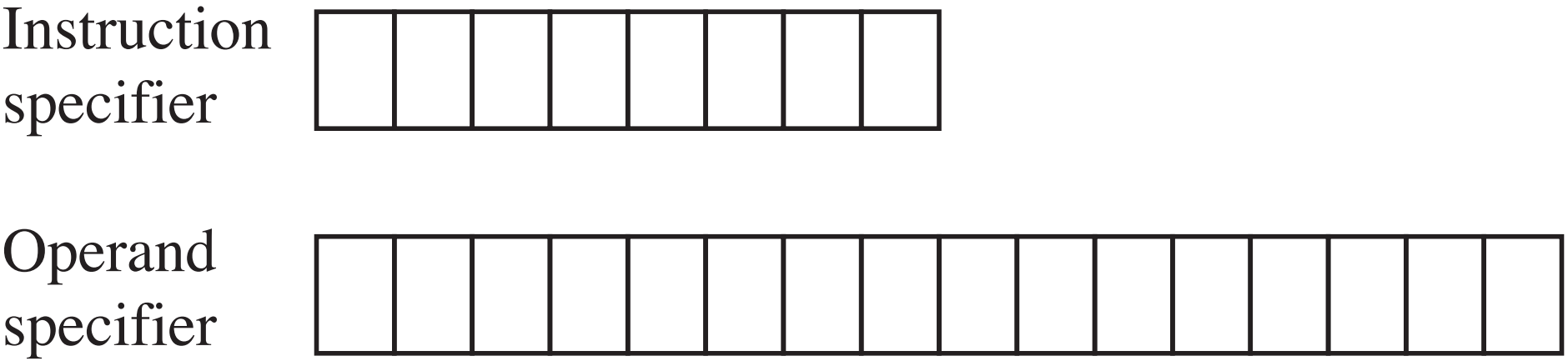
Figure 4.6
(Continued)

0001 100r	Bitwise invert register r
0001 101r	Negate register r
0001 110r	Arithmetic shift left register r
0001 111r	Arithmetic shift right register r
0010 000r	Rotate left register r
0010 001r	Rotate right register r
0010 01nn	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	Character input
0101 0aaa	Character output
0101 1nnn	Return from call with n local bytes

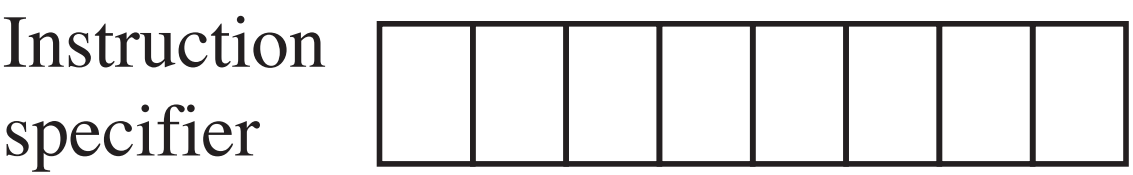
Figure 4.6
(Continued)

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

Figure 4.7



(a) The two parts of a nonunary instruction



(b) A unary instruction

Figure 4.8

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

(a) The addressing-aaa field.

Figure 4.8
(Continued)

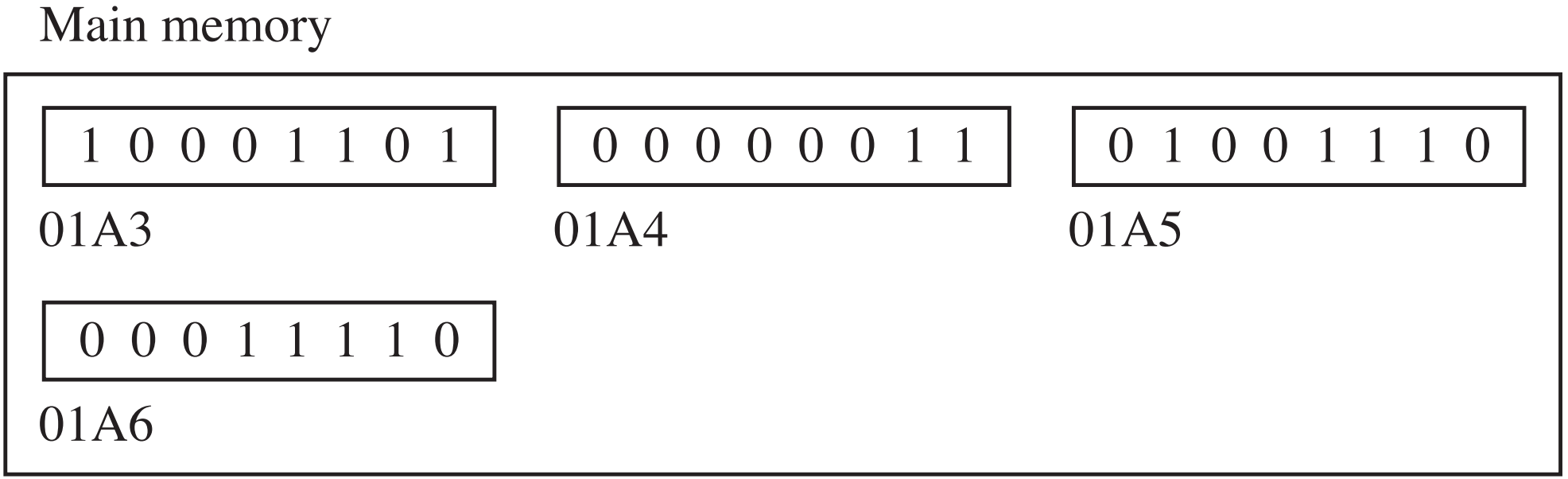
a	Addressing mode
0	Immediate
1	Indexed

(b) The addressing-a field.

r	Register
0	Accumulator, A
1	Index register, X

(c) The register-r field.

Figure 4.9



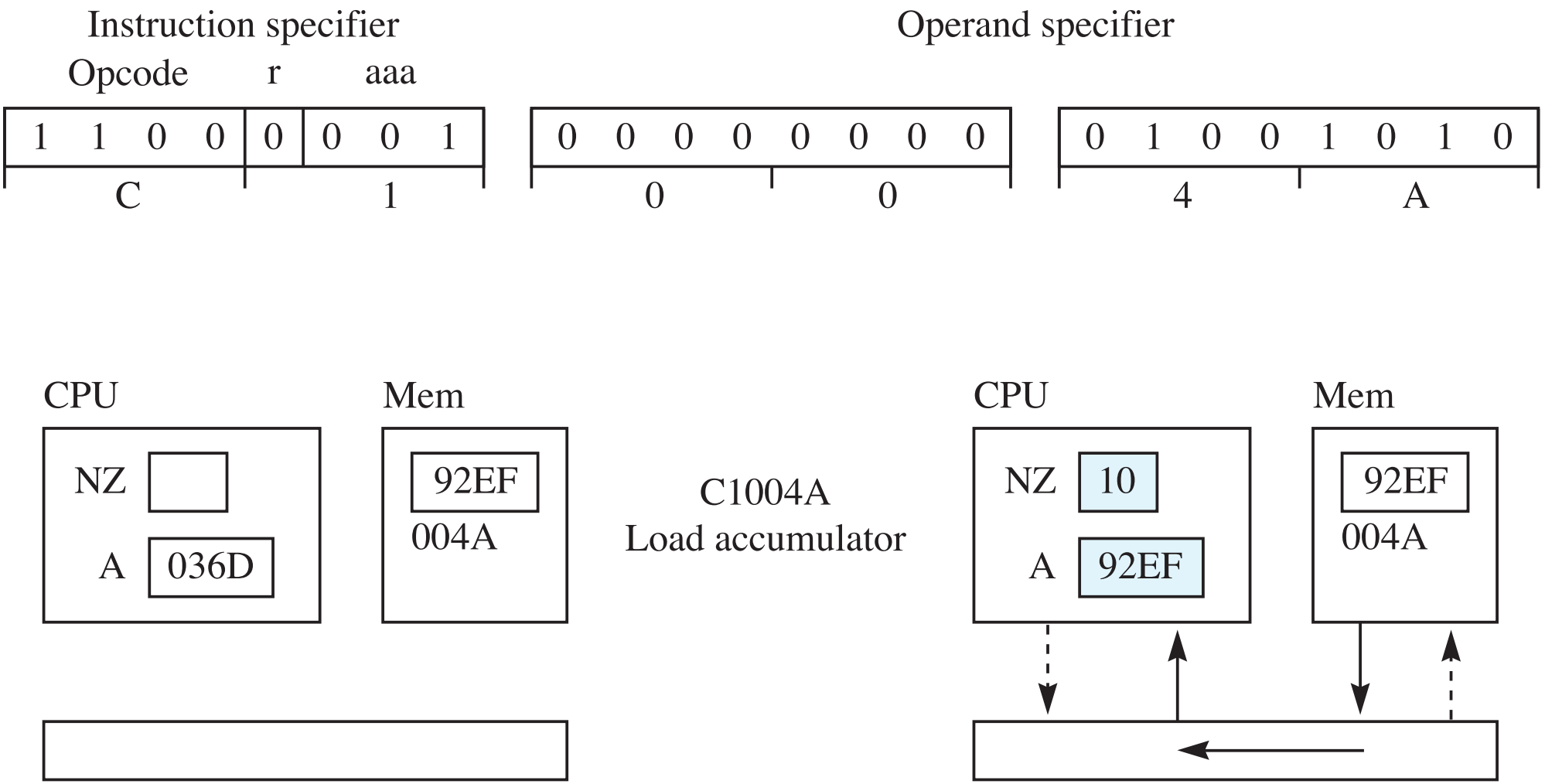
The stop instruction

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

The load instruction

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

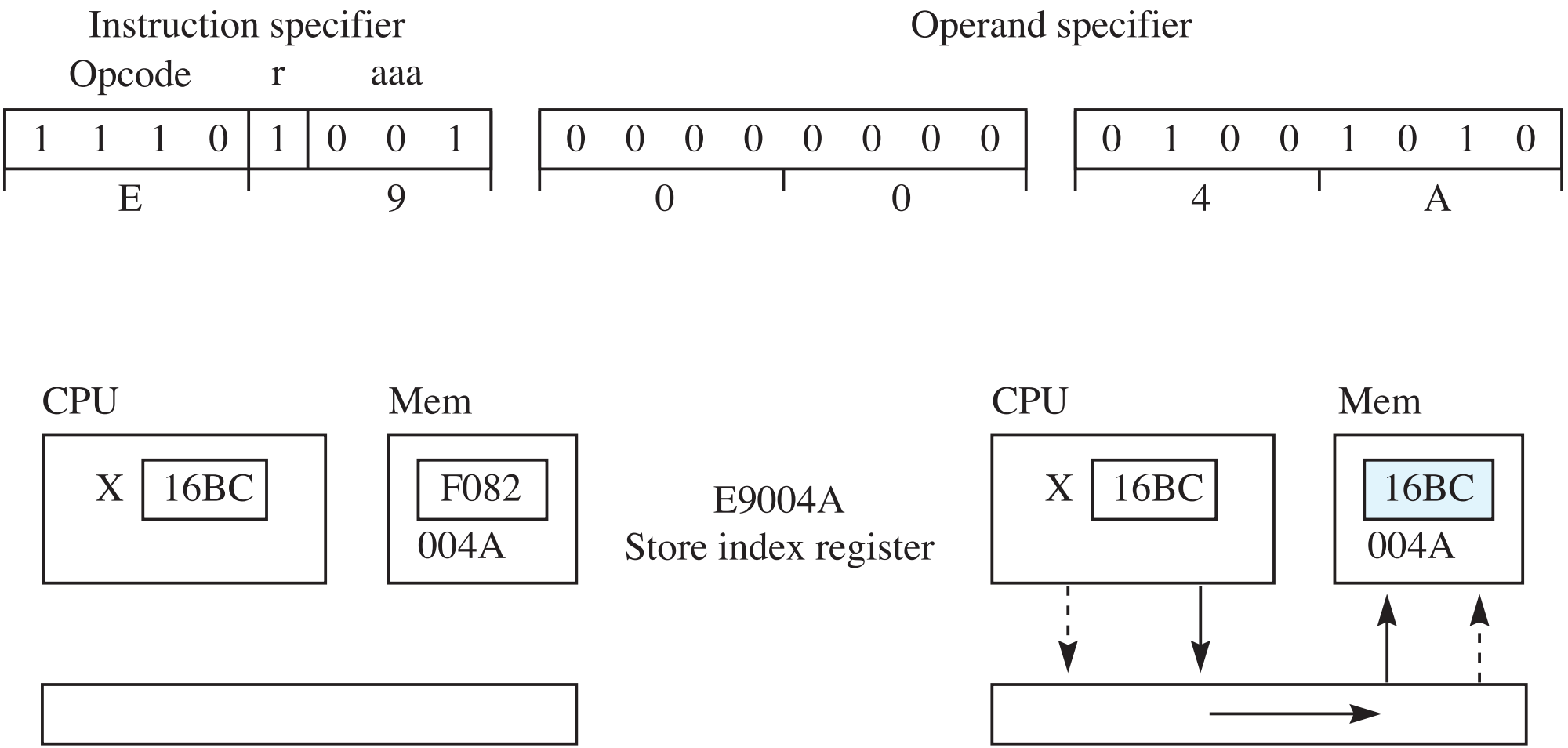
$$r \leftarrow \text{Oprnd} ; N \leftarrow r < 0 , Z \leftarrow r = 0$$



The store instruction

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

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

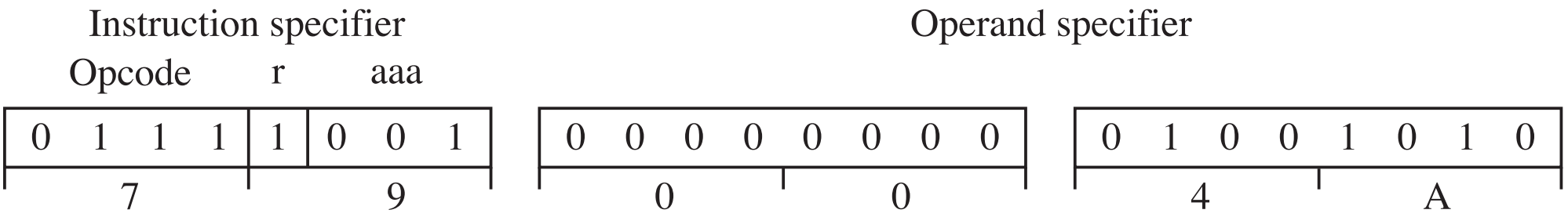


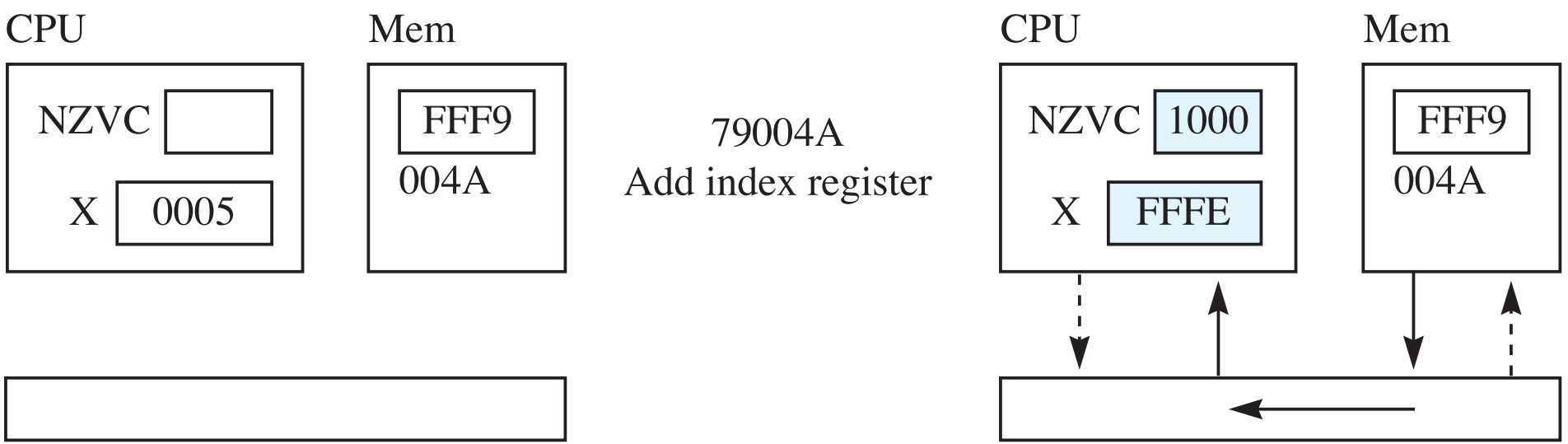
The add instruction

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

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

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





(a) Before

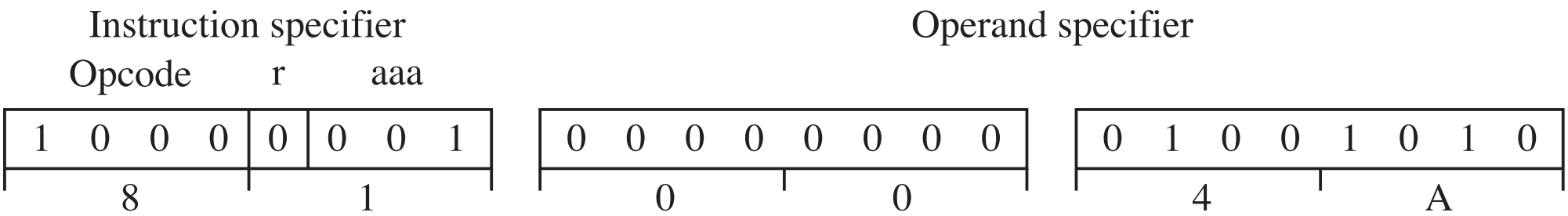
(b) After

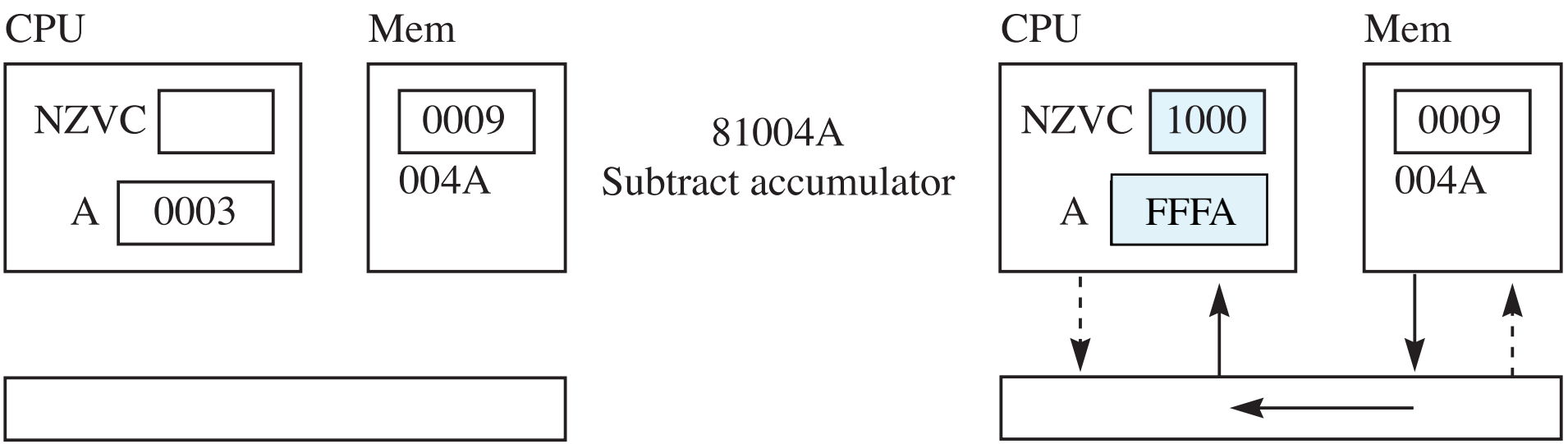
The subtract instruction

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

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

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





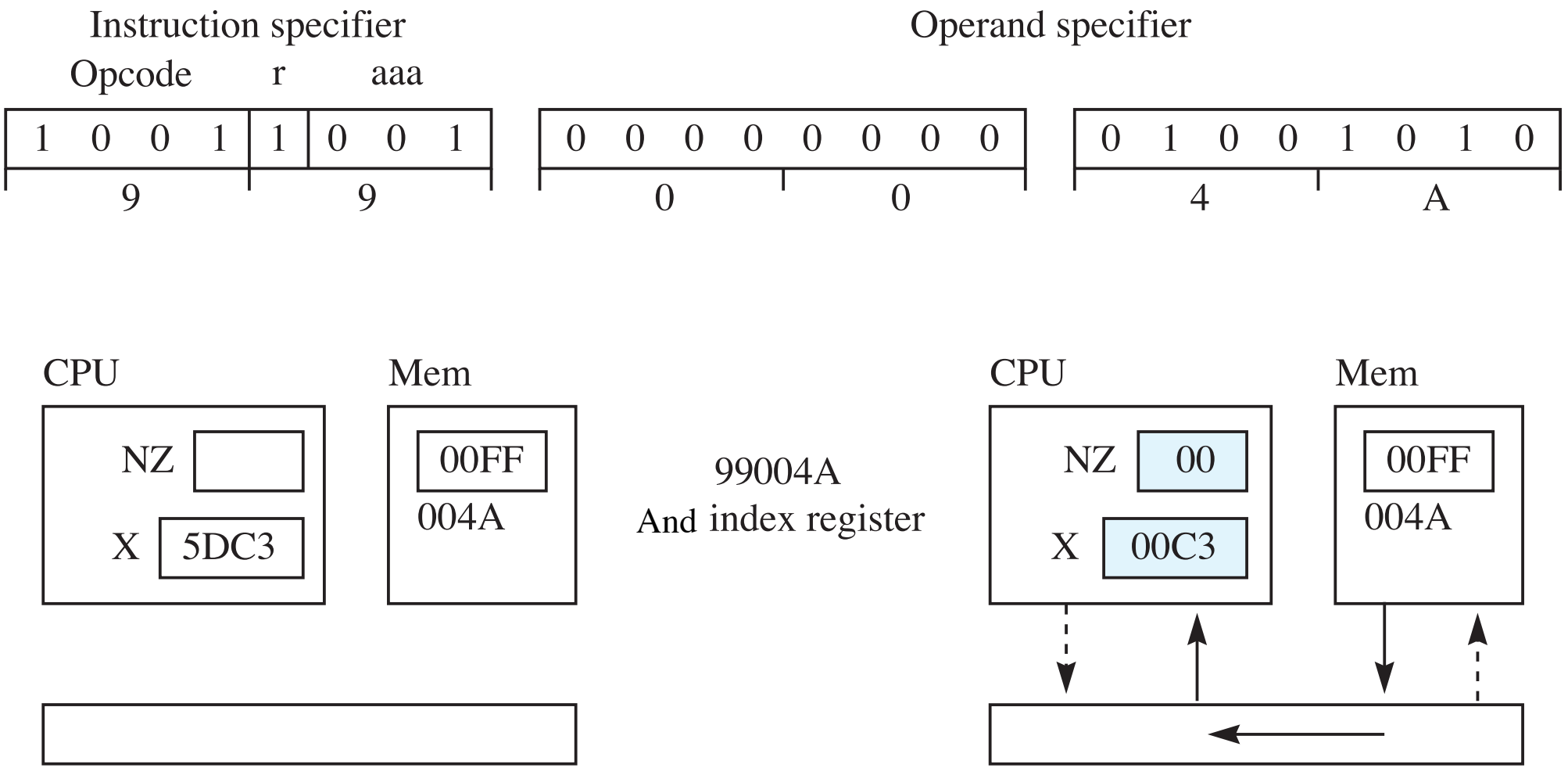
(a) Before

(b) After

The and instruction

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

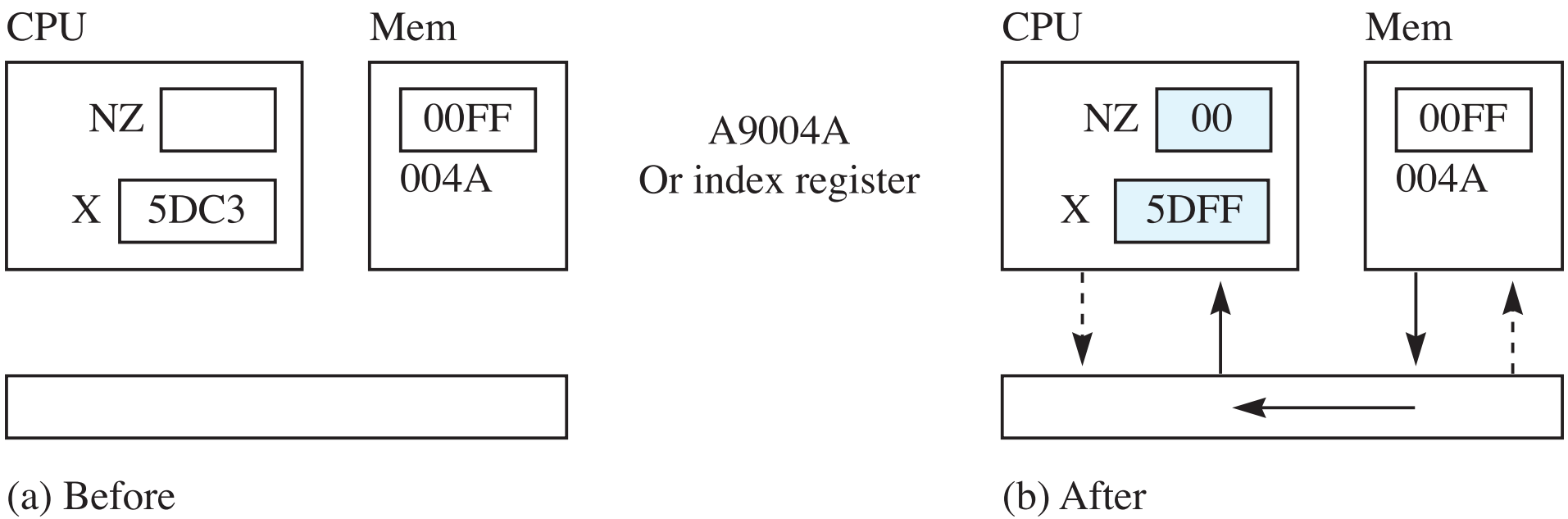
$$r \leftarrow r \wedge Oprnd \; ; \; N \leftarrow r < 0 \; , \; Z \leftarrow r = 0$$



The or instruction

- Instruction specifier: 1010 raaa
- ORs one word (two bytes) from memory to register r

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



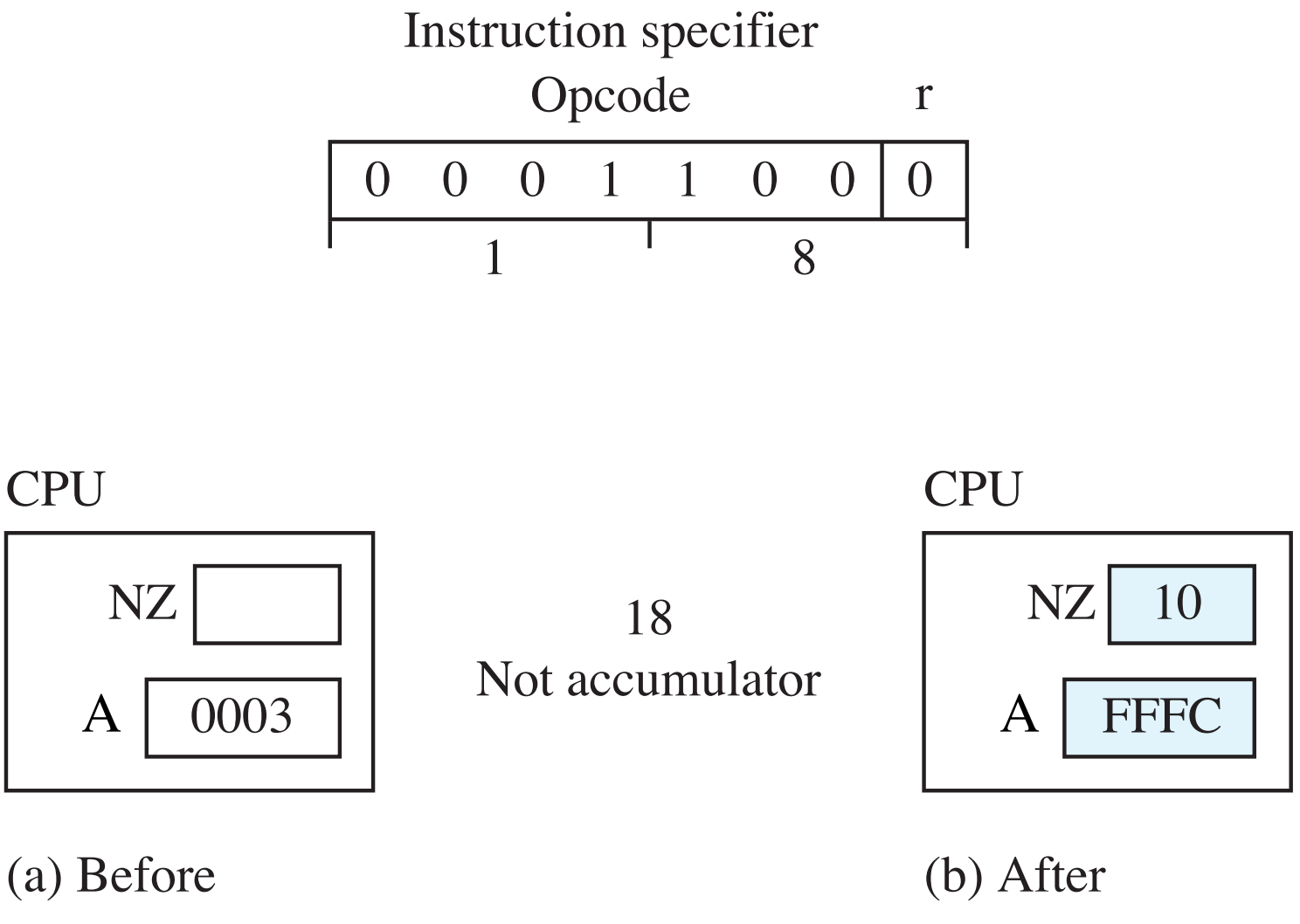
(a) Before

(b) After

The not instruction

- Instruction specifier: 0001 100r
- 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

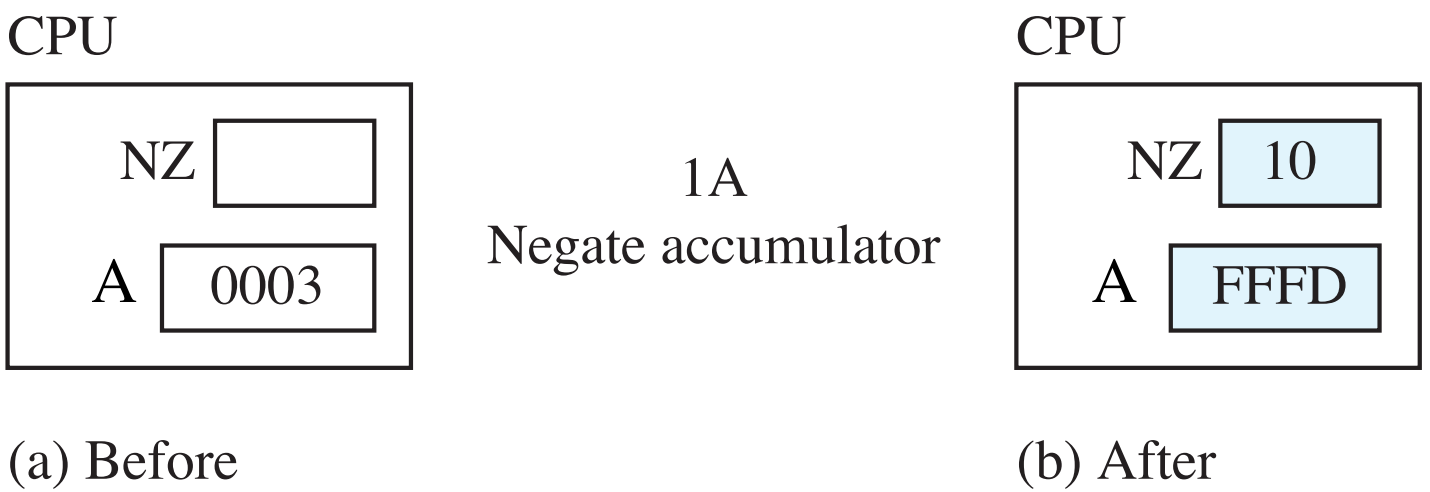
(b) After

The negate instruction

- Instruction specifier: 0001 101r
- Negate (take two's complement of) register r

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

Figure 4.23



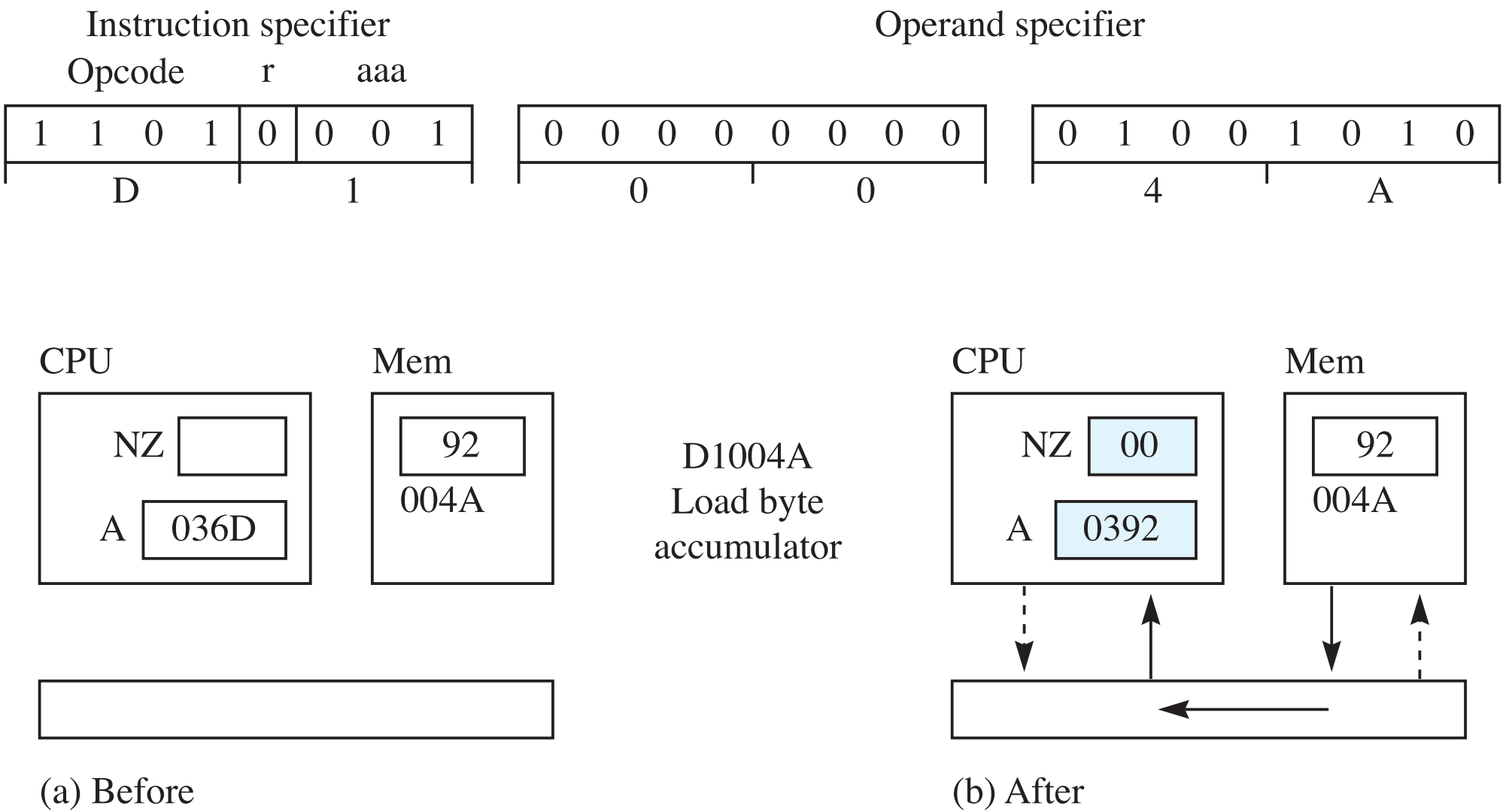
(a) Before

(b) After

The load byte instruction

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

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

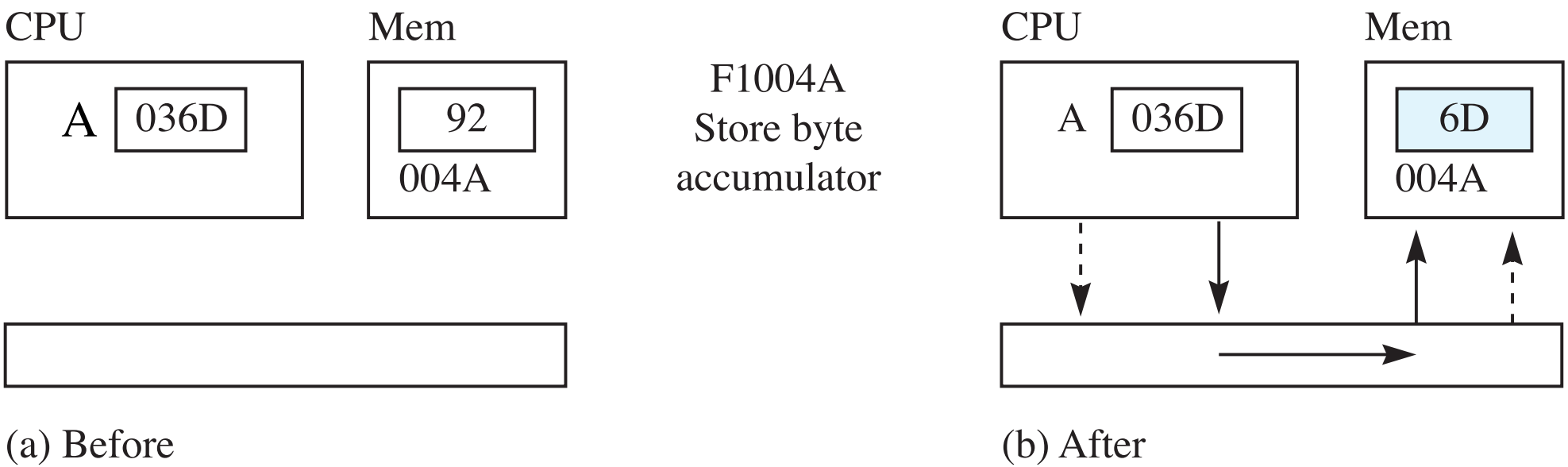


The store byte instruction

- 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



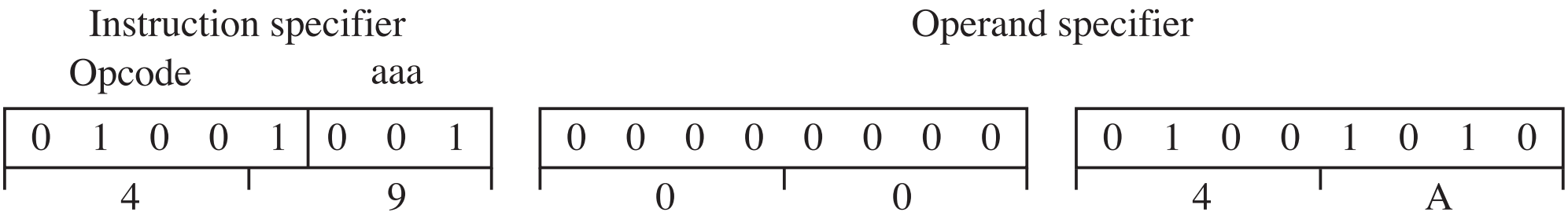
(a) Before

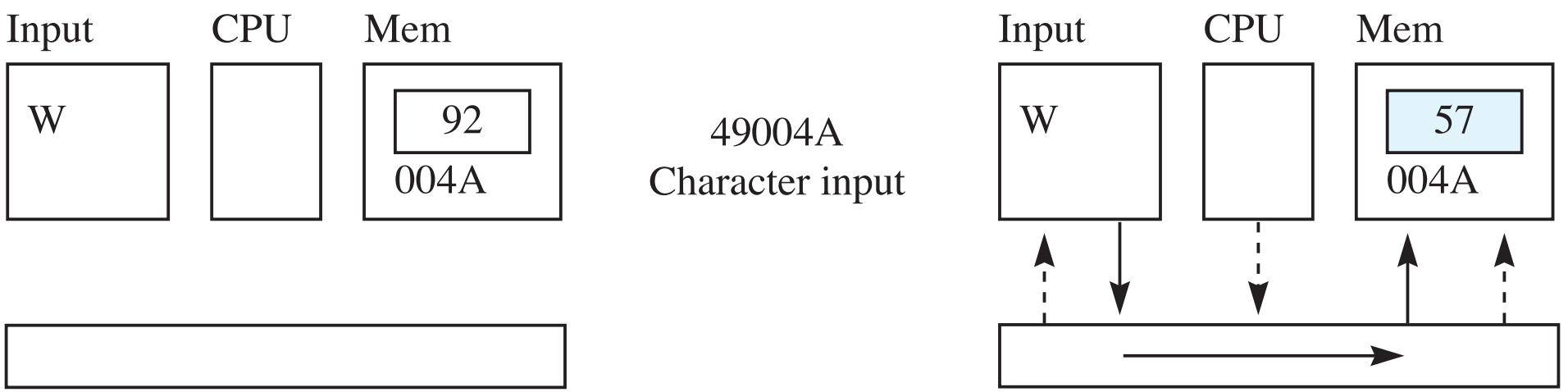
(b) After

The character input instruction

- Instruction specifier: 0100 1aaa
- Stores one ASCII byte from the input device to memory

$$\text{byte Oprnd} \leftarrow \{\textit{character input}\}$$





(a) Before

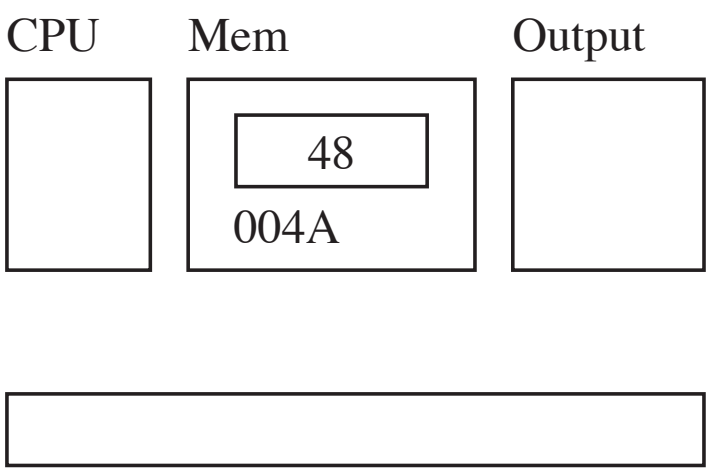
(b) After

The character output instruction

- Instruction specifier: 0101 0aaa
- Sends one ASCII byte from memory to the output device

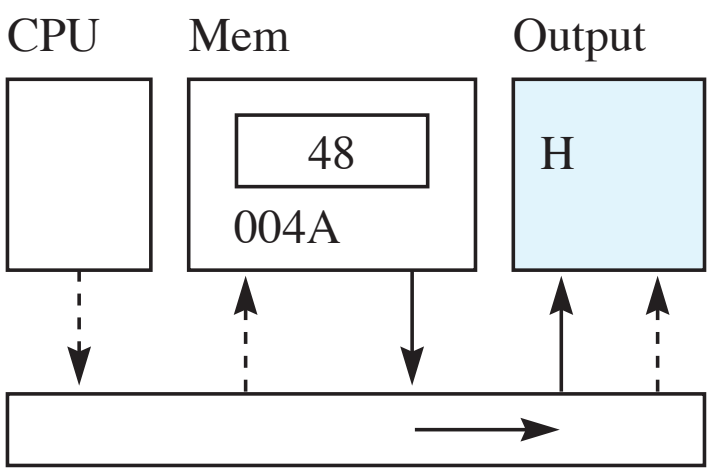
$$\{character\ output\} \leftarrow \text{byte Oprnd}$$

Figure 4.29



(a) Before

51004A
Character output



(b) After

The von Neumann execution cycle

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

Figure 4.30

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***)**

Figure 4.31

Load the machine language program into memory starting at address 0000
PC ← 0000
SP ← Mem[FFF8]
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***))**

Figure 4.32

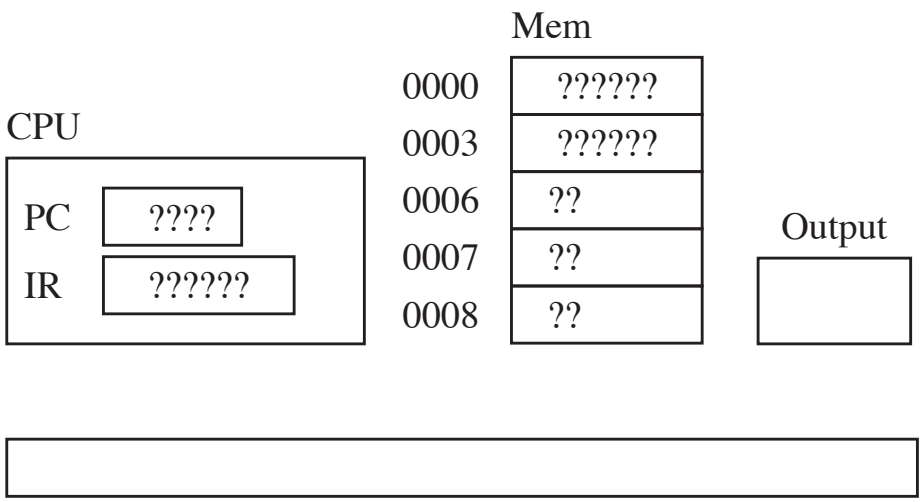
Address	Machine Language (bin)
0000	0101 0001 0000 0000 0000 0111
0003	0101 0001 0000 0000 0000 1000
0006	0000 0000
0007	0100 1000
0008	0110 1001

Address	Machine Language (hex)	
0000	510007	;Character output
0003	510008	;Character output
0006	00	;Stop
0007	48	;ASCII H character
0008	69	;ASCII i character

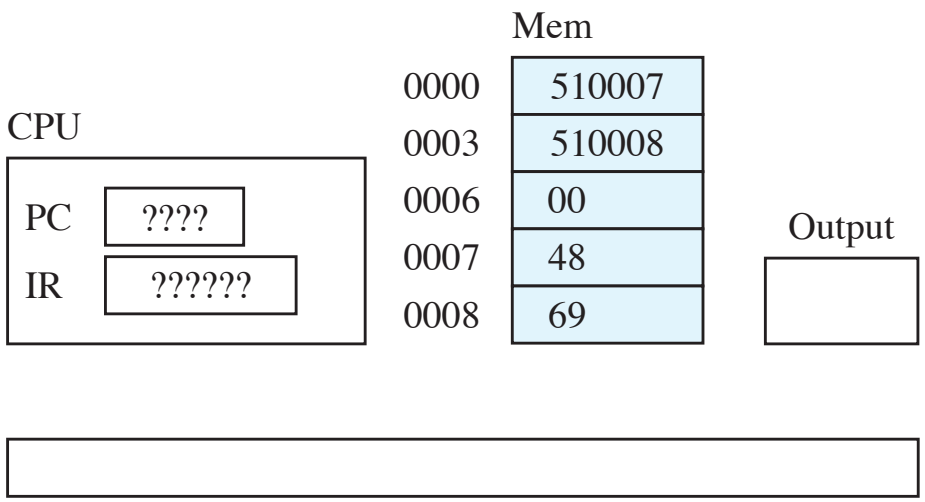
Output

```
Hi
```

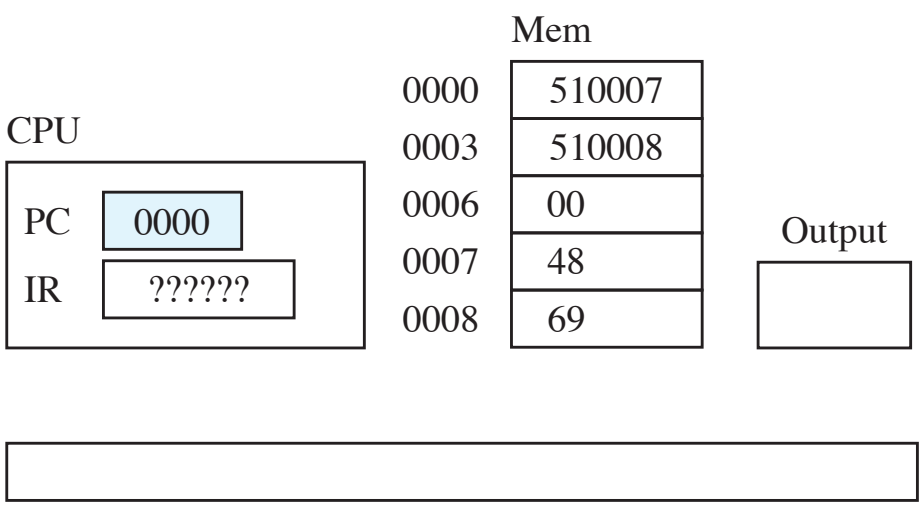
Figure 4.33



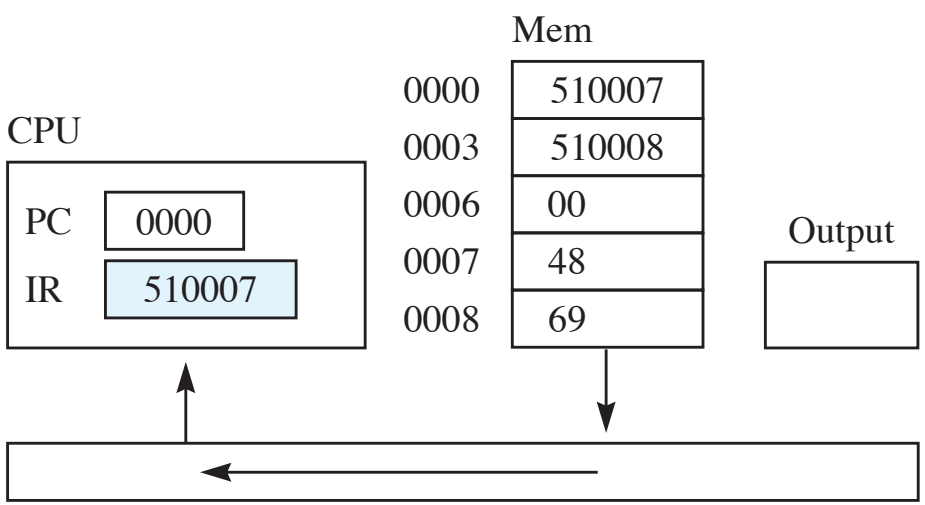
(a) Initial state before loading.



(b) Program loaded into main memory.

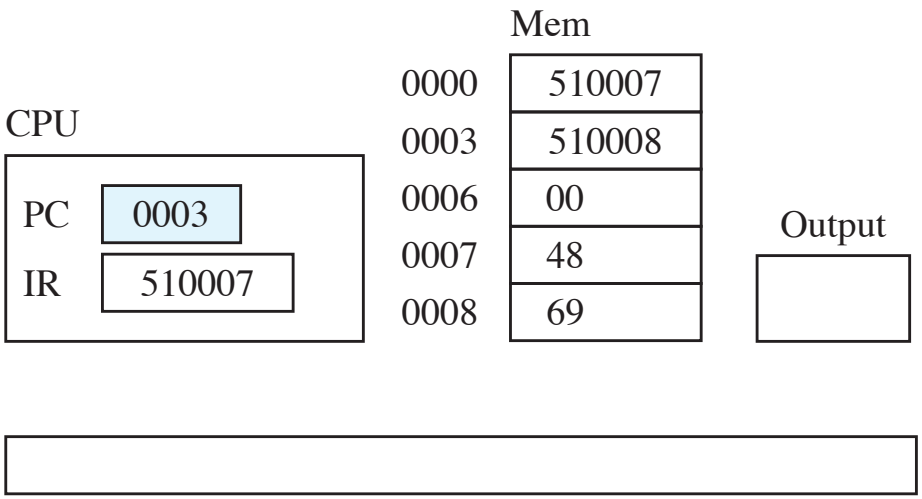


(c) PC ← 0000 (hex)

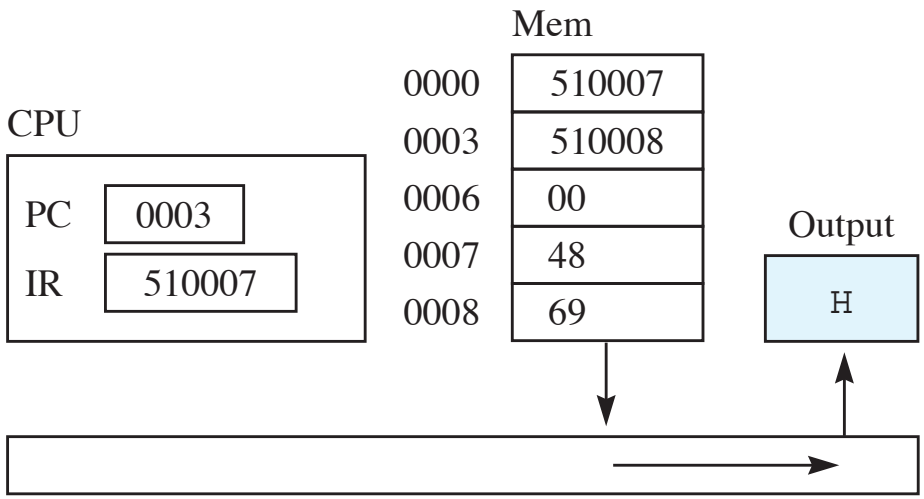


(d) Fetch

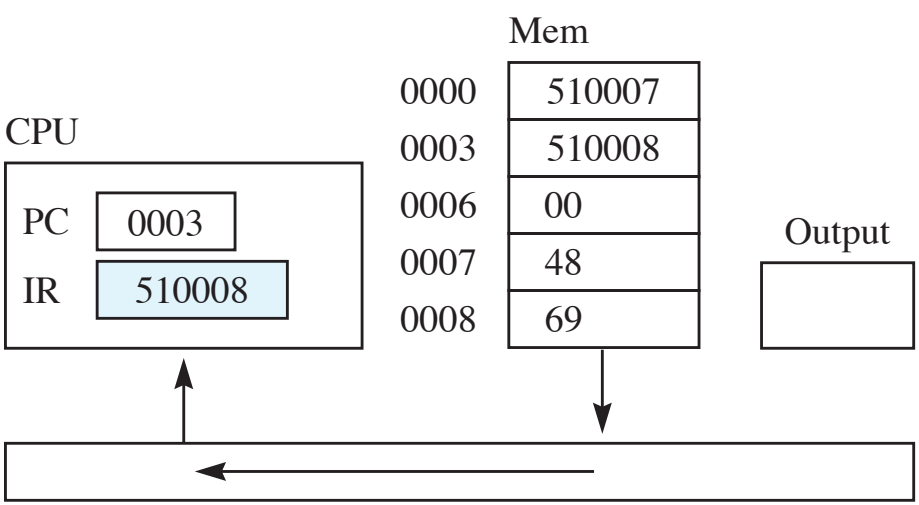
Figure 4.33
(Continued)



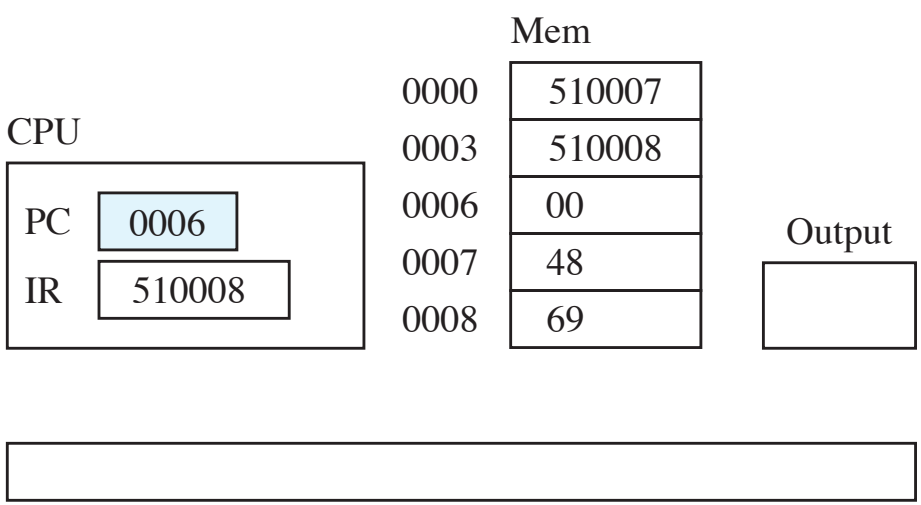
(e) Increment PC.



(f) Execute. Character H sent to output device.

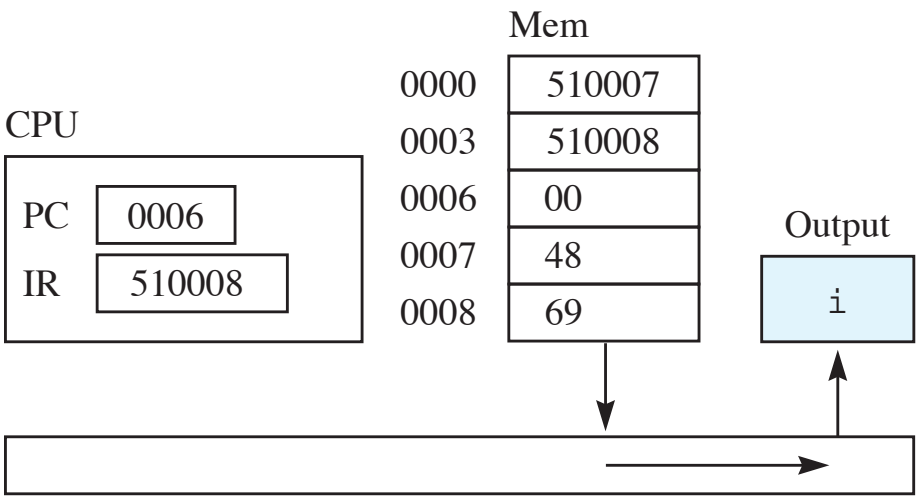


(g) Fetch.

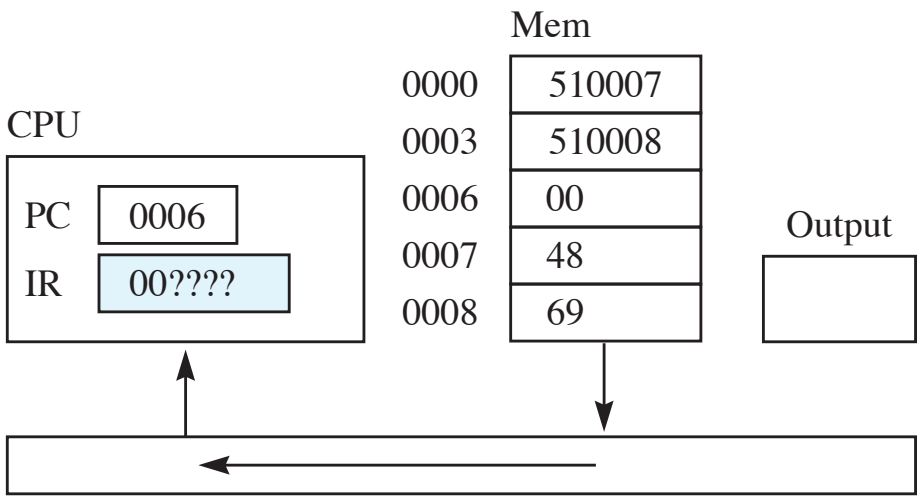


(h) Increment PC.

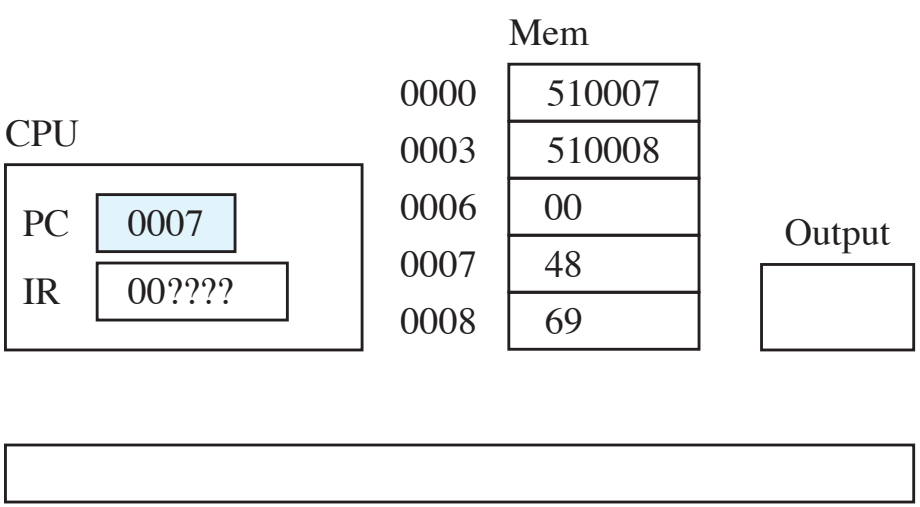
Figure 4.33
(Continued)



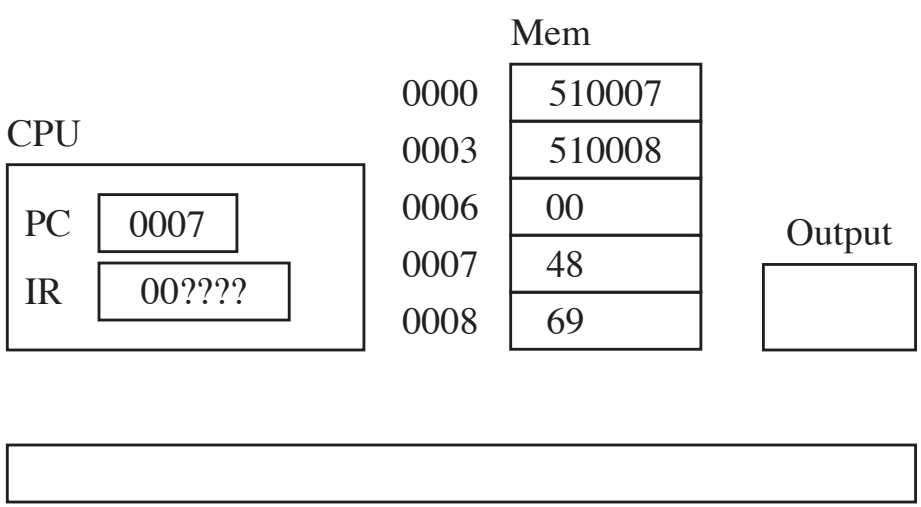
(i) Execute. Character `i` sent to output device.



(j) Fetch



(k) Increment PC.



(l) Execute. The computer halts.

Figure 4.34

Address	Machine Language (bin)
0000	0100 1001 0000 0000 0000 1101
0003	0100 1001 0000 0000 0000 1110
0006	0101 0001 0000 0000 0000 1110
0009	0101 0001 0000 0000 0000 1101
000C	0000 0000

Address	Machine Language (hex)	
0000	49000D	;Character input
0003	49000E	;Character input
0006	51000E	;Character output
0009	51000D	;Character output
000C	00	;Stop

Input

```
up
```

Output

```
pu
```

Figure 4.35

Address	Machine Language (bin)
0000	1100 0001 0000 0000 0001 0001
0003	0111 0001 0000 0000 0001 0011
0006	1010 0001 0000 0000 0001 0101
0009	1111 0001 0000 0000 0001 0000
000C	0101 0001 0000 0000 0001 0000
000F	0000 0000
0010	0000 0000
0011	0000 0000 0000 0101
0013	0000 0000 0000 0011
0015	0000 0000 0011 0000

Address	Machine Language (hex)	
0000	C10011	;A := first number
0003	710013	;Add the two numbers
0006	A10015	;Convert sum to character
0009	F10010	;Store the character
000C	510010	;Output the character
000F	00	;Stop
0010	00	;Character to output
0011	0005	;Decimal 5
0013	0003	;Decimal 3
0015	0030	;Mask for ASCII char

Output

8

Figure 4.36

Address	Machine Language (bin)
0000	1101 0001 0000 0000 0001 1101
0003	1111 0001 0000 0000 0000 1001
0006	1100 0001 0000 0000 0001 0111
0009	0111 0001 0000 0000 0001 1001
000C	1010 0001 0000 0000 0001 1011
000F	1111 0001 0000 0000 0001 0110
0012	0101 0001 0000 0000 0001 0110
0015	0000 0000
0016	0000 0000
0017	0000 0000 0000 0101
0019	0000 0000 0000 0011
001B	0000 0000 0011 0000
001D	1000 0001

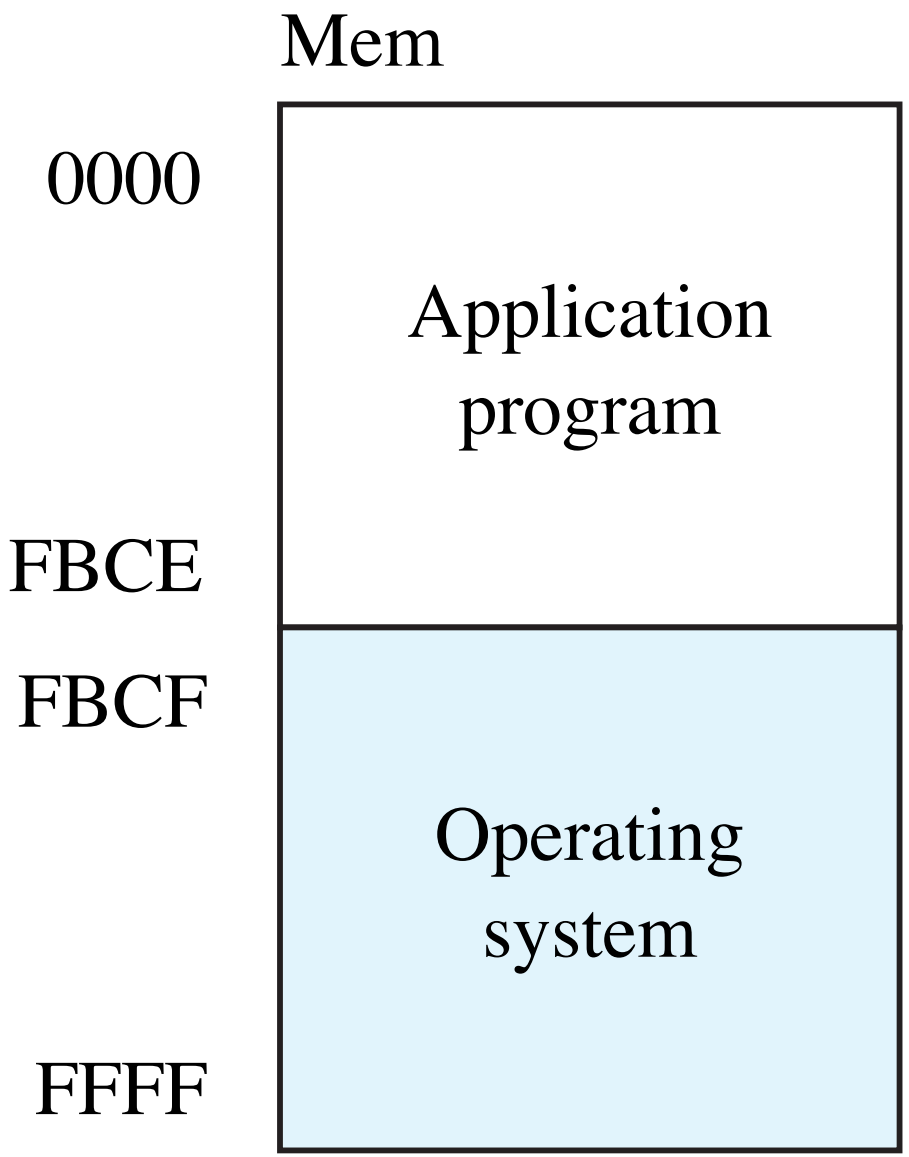
Figure 4.36
(Continued)

Address	Machine Language (hex)	
0000	D1001D	;Load byte accumulator
0003	F10009	;Store byte accumulator
0006	C10017	;A := first number
0009	710019	;Add the two numbers
000C	A1001B	;Convert sum to character
000F	F10016	;Store the character
0012	510016	;Output the character
0015	00	;Stop
0016	00	;Character to output
0017	0005	;Decimal 5
0019	0003	;Decimal 3
001B	0030	;Mask for ASCII char
001D	81	;Byte to modify instruction

Output

2

Figure 4.37



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 4.38

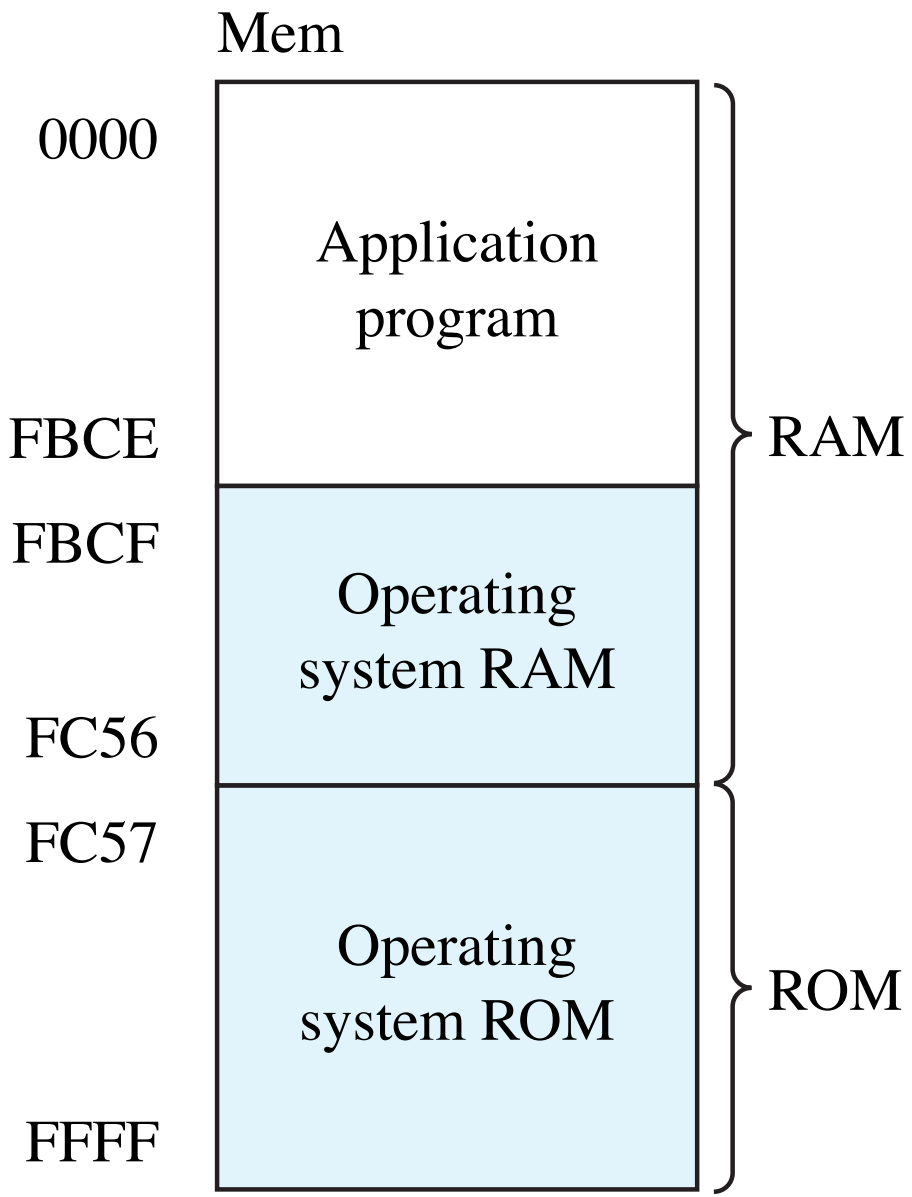
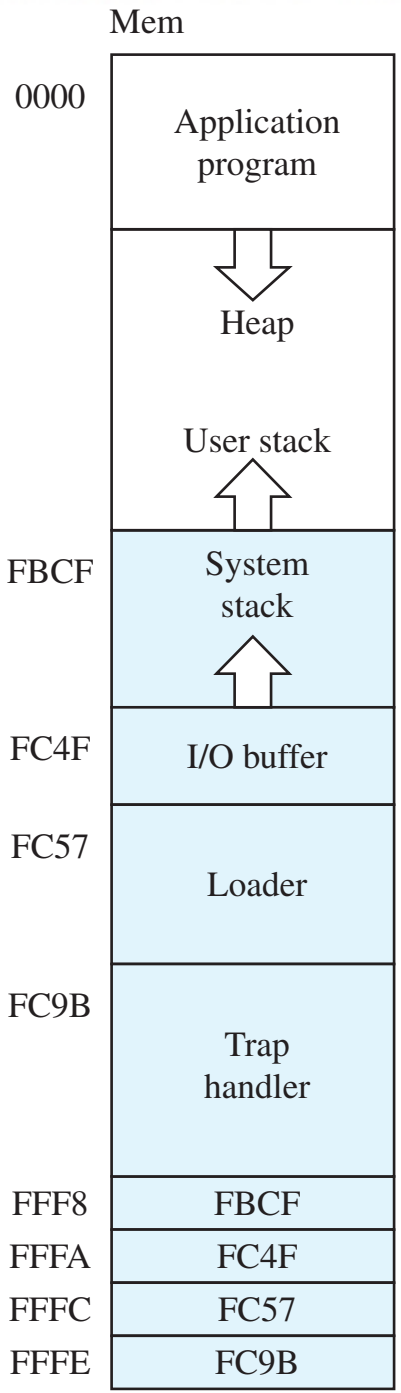
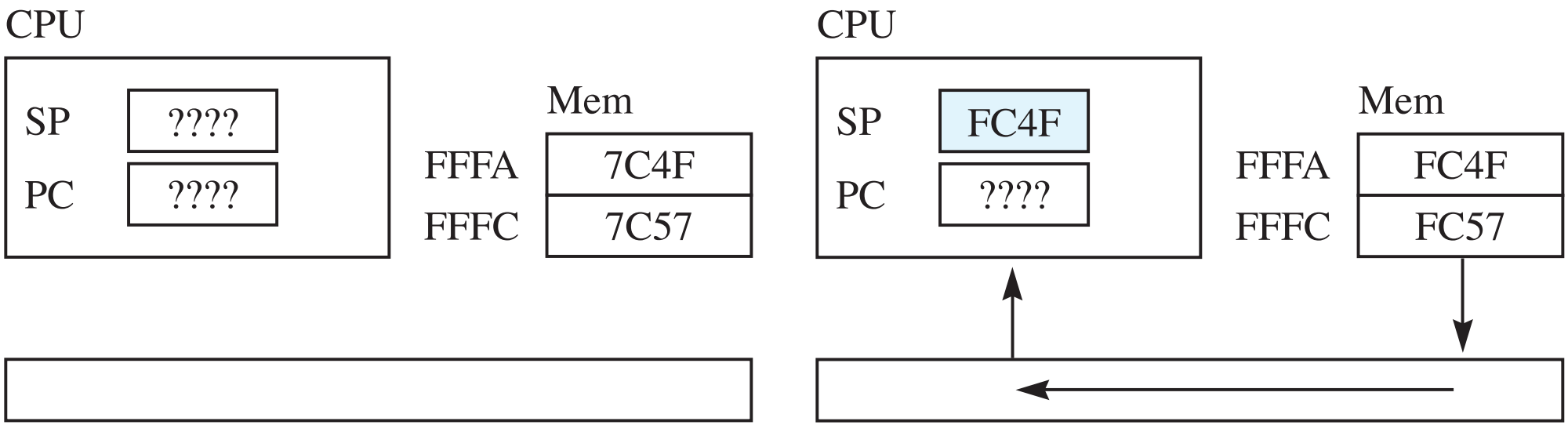


Figure 4.39



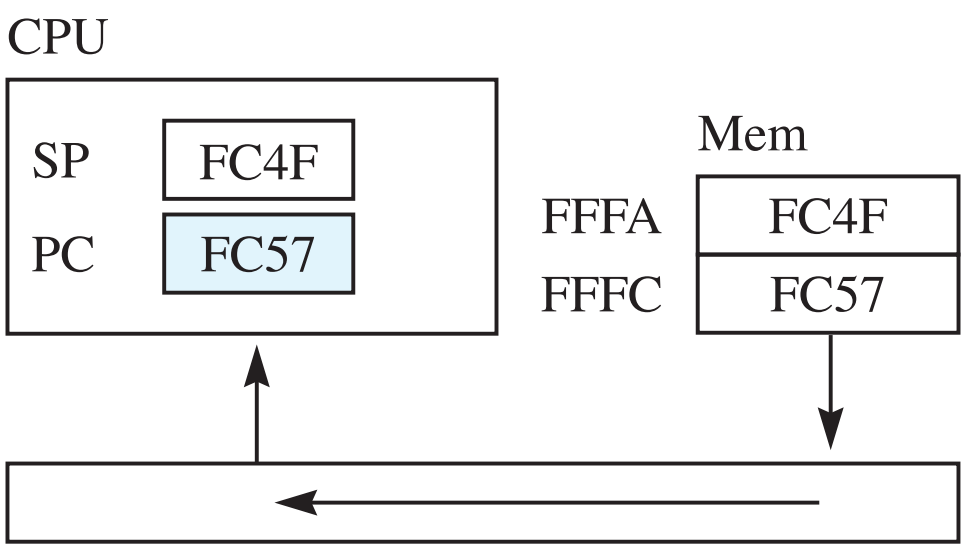
The load option

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



(a) Initial state.

(b) SP ← Mem[FFFA]



(c) PC ← Mem[FFFC]

The execute option

- SP ← Mem[FFF8]
- PC ← 0000
- Start the von Neumann cycle