Computer Systems

Sixth edition

Chapter 4

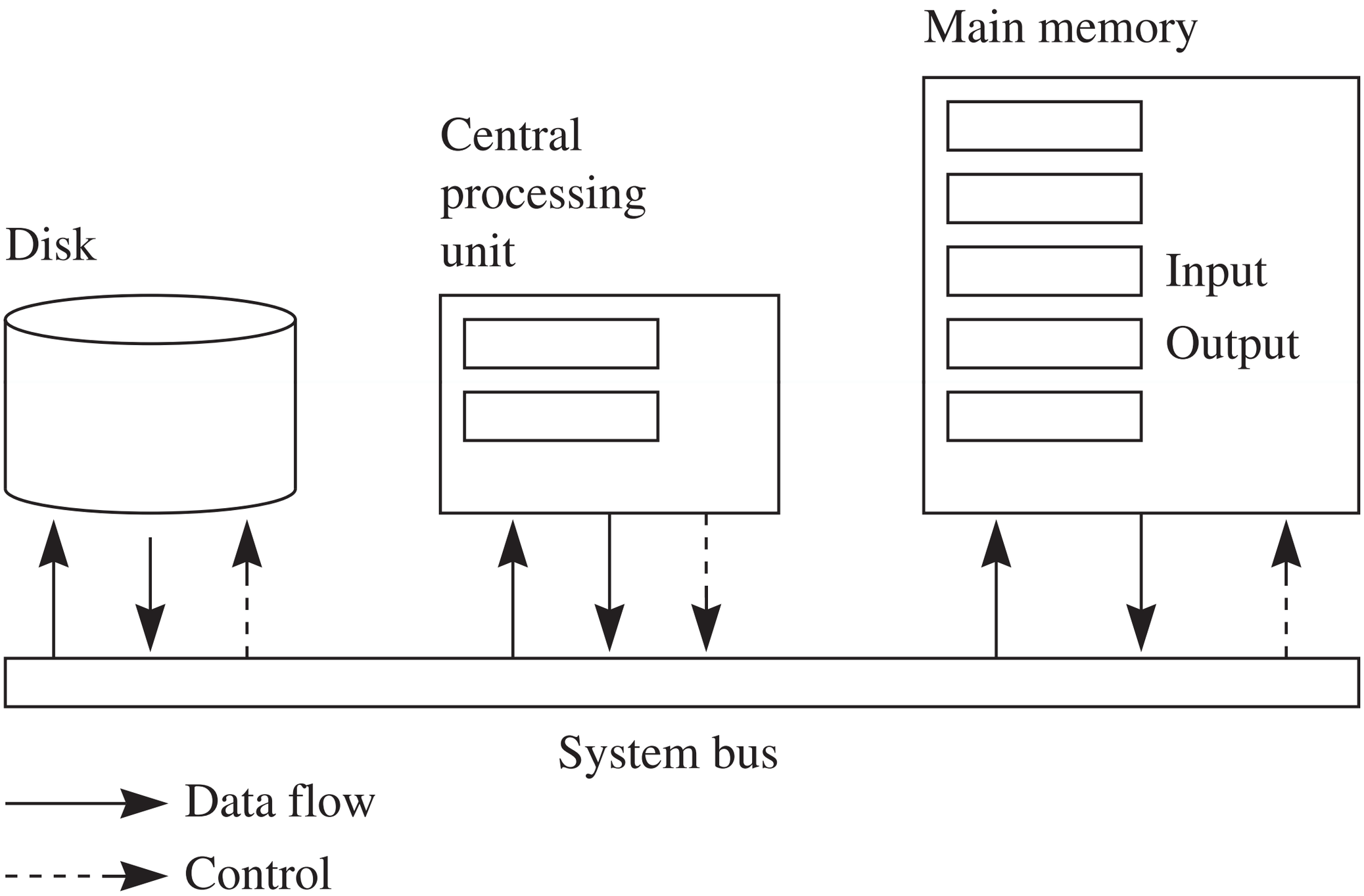
Computer Architecture

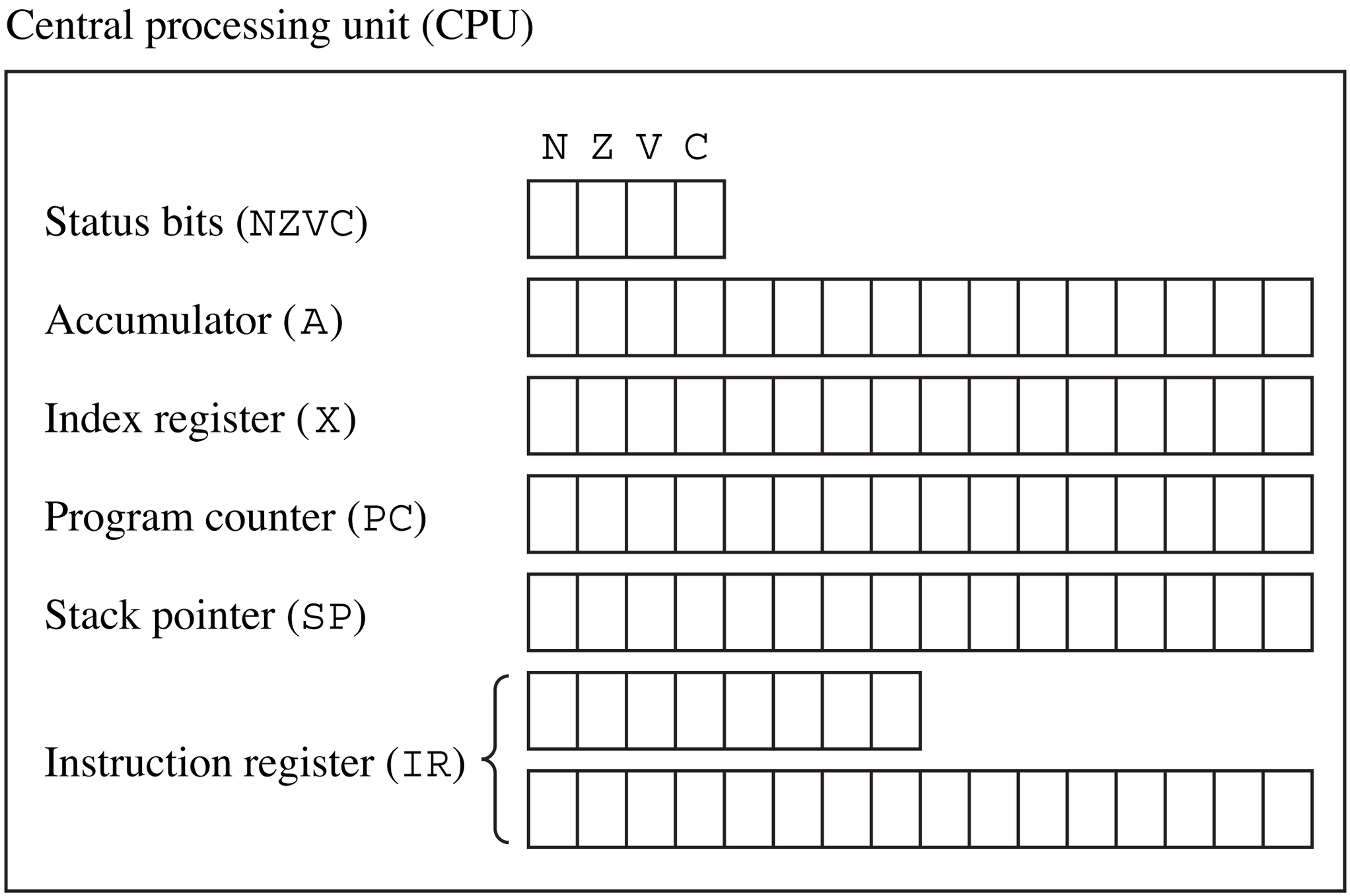
Pep/10 virtual machine

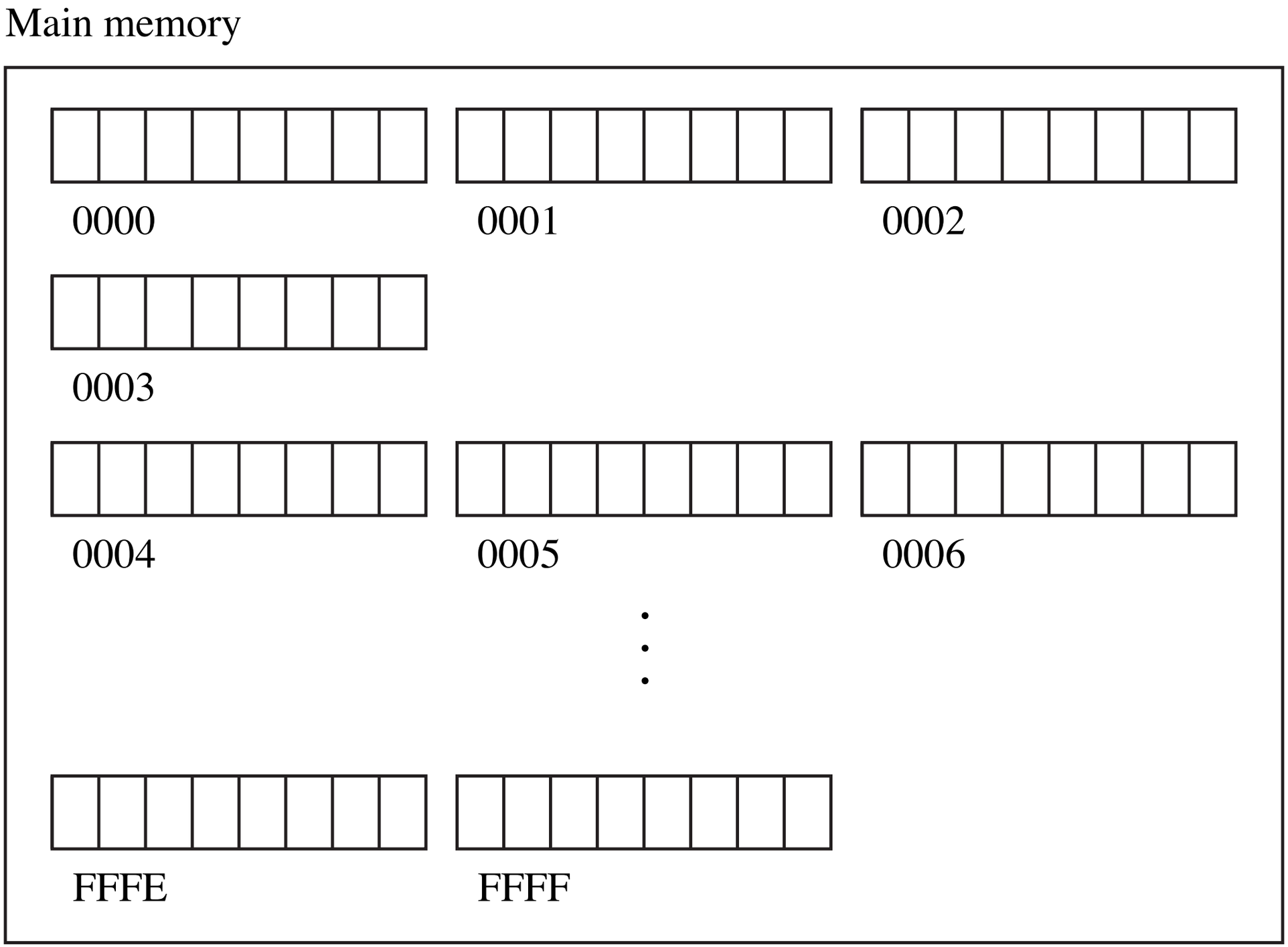
- Advantages
- Disadvantages

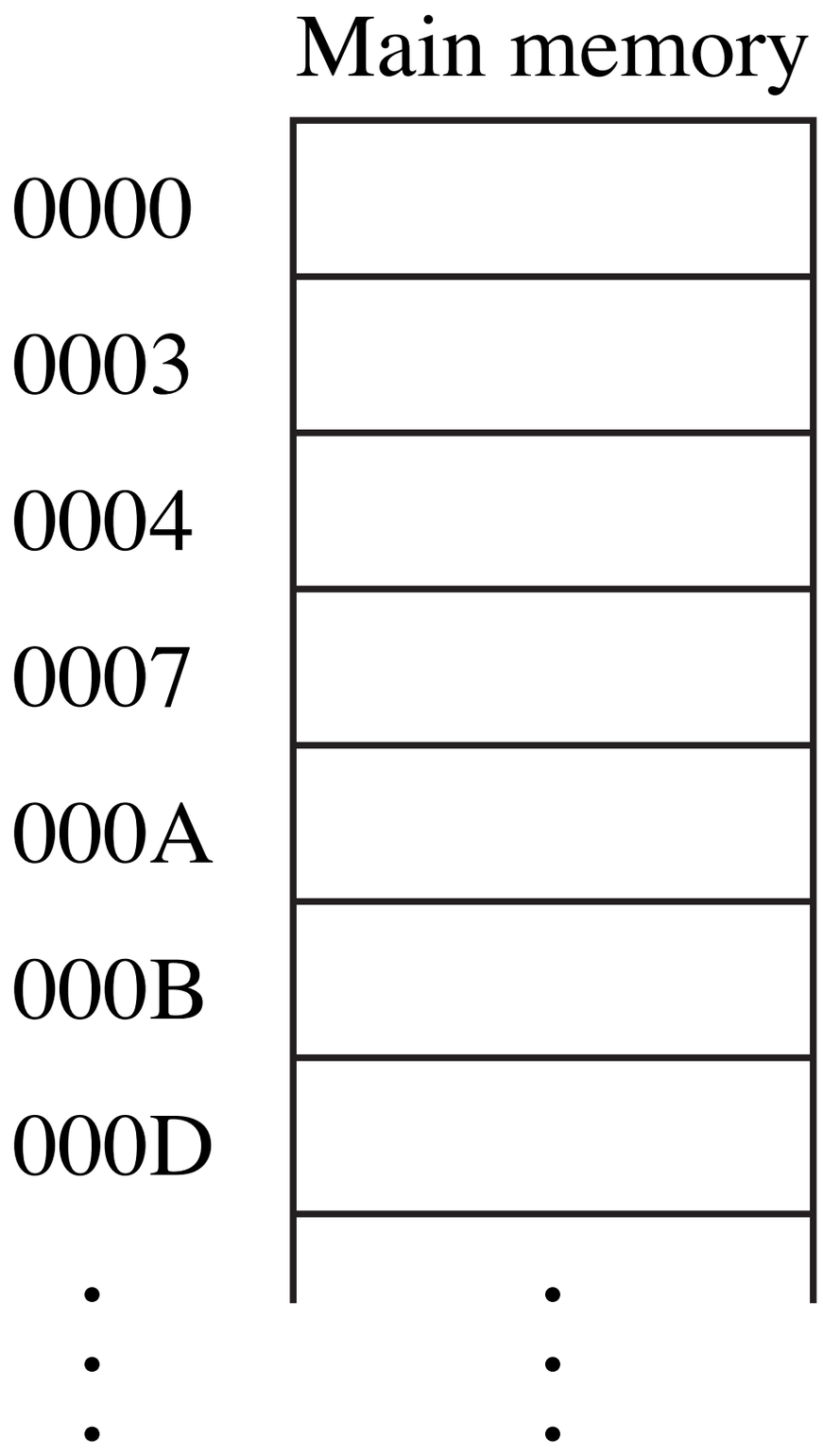
Pep/10 virtual machine

- Disk
- Central processing unit (CPU)
- Main memory
- System bus


Main memory
Central processing unit
Disk
Input
Output
System bus
Data flow
Control


Central processing unit (CPU)
N Z V C
Status bits (NZVC)
Accumulator (A)
Index register (X)
Program counter (PC)
Stack pointer (SP)
Instruction register (IR)


Main memory
0000
0001
0002
0003
0004
0005
0006
FFFE
FFFF


Main memory
0000
0003
0004
0007
000A
000B
000D

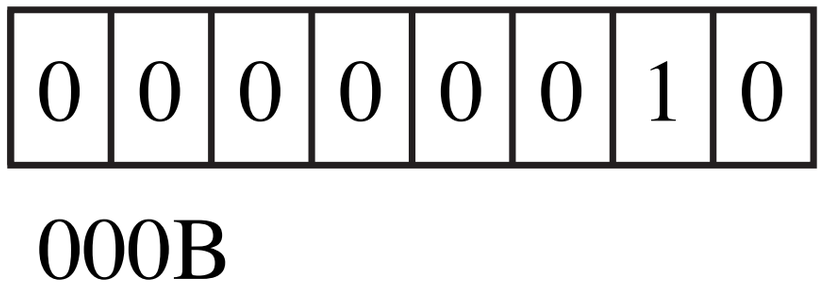
0	0	0	0	0	0	1	0

000B

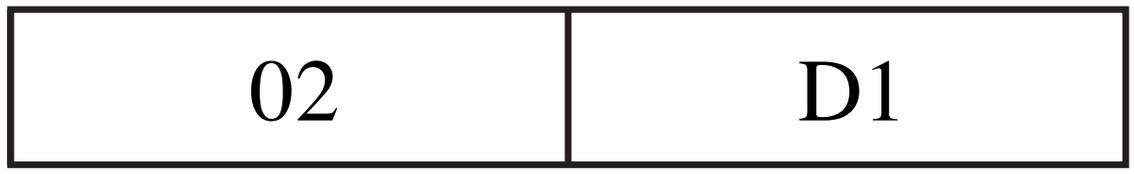
1	1	0	1	0	0	0	1

000C

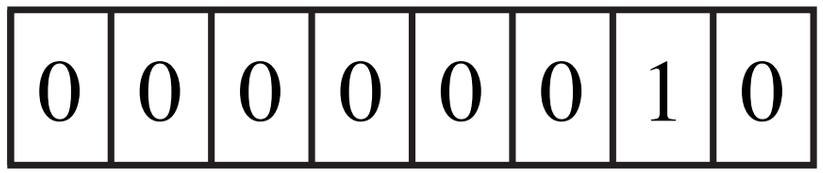
(a) The content in binary.



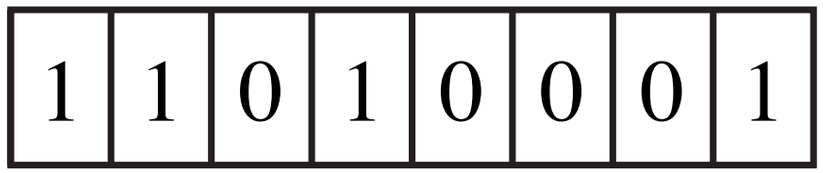
(a) The content in binary.



(b) The content in hexadecimal.

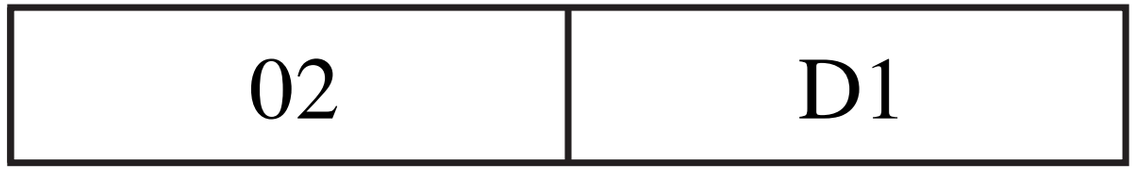


000B



000C

(a) The content in binary.

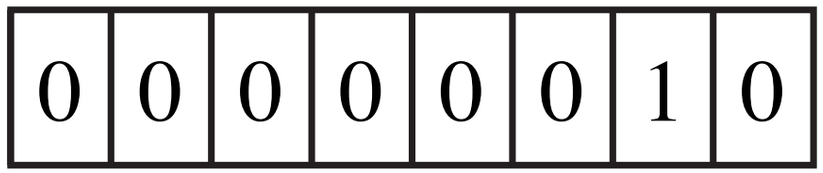


000B 000C

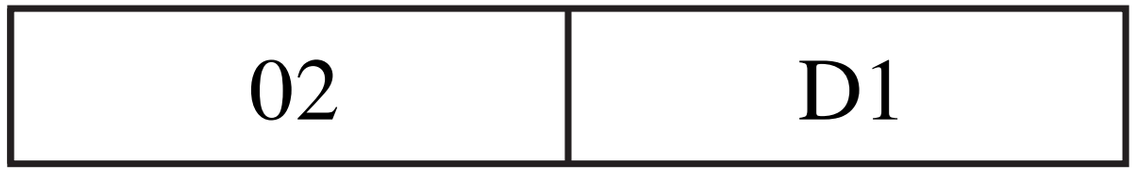
(b) The content in hexadecimal.

000B 02D1

(c) The content in a machine language listing.



(a) The content in binary.

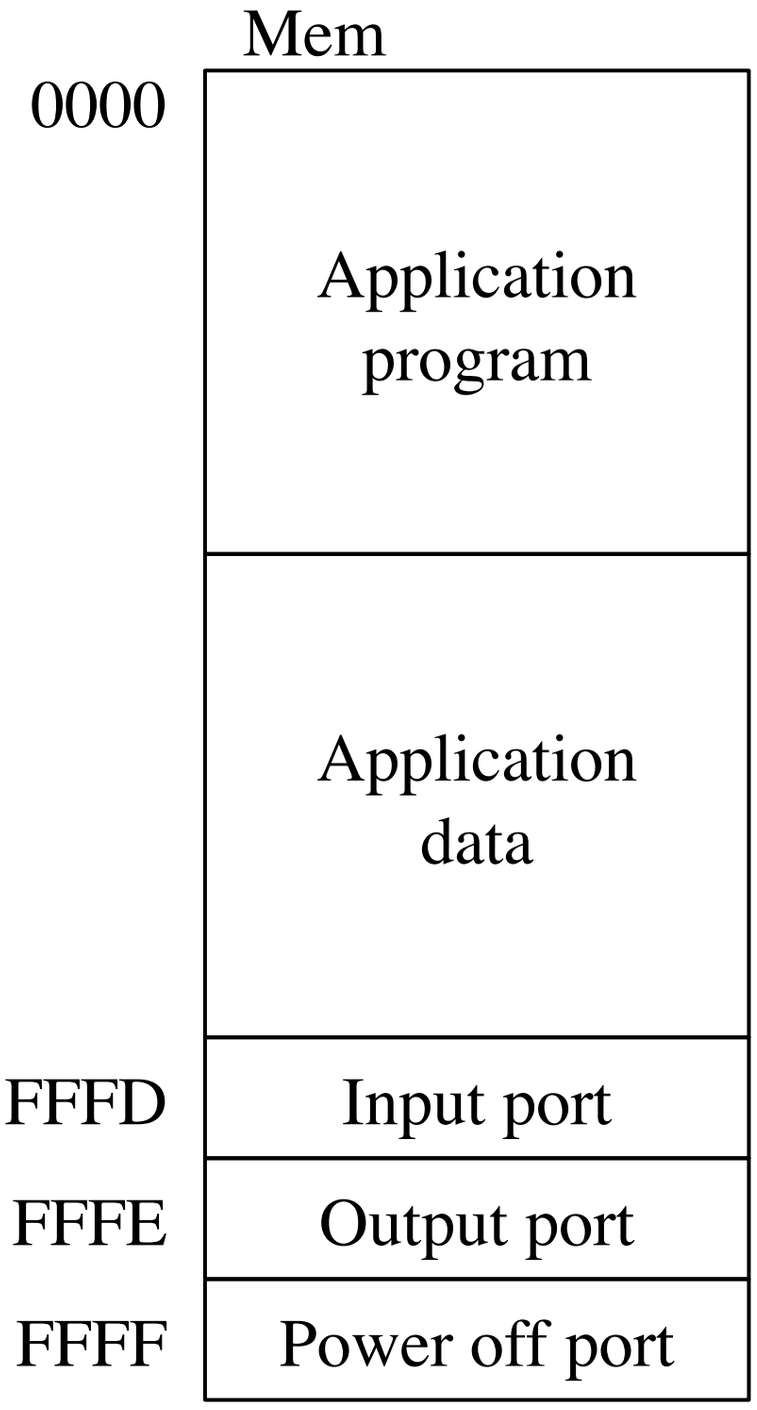


(b) The content in hexadecimal.

000B 02D1

Address of D1 Is 000C

(c) The content in a machine language listing.



Pep/10 memory map

- Bare metal mode
- No operating system

Instruction set

Instruction Specifier	Instruction	Type
0000 0000	Illegal instruction	
0000 0001	Return from call	Monadic
0000 0010	Return from system call	Monadic
0000 0011	Move SP to A	Monadic
0000 0100	Move A to SP	Monadic
0000 0101	Move NZVC flags to A[12 : 15]	Monadic
0000 0110	Move A[12 : 15] to NZVC flags	Monadic
0000 0111	No operation	Monadic
0001 100r	Negate r	Monadic
0001 101r	Arithmetic shift left r	Monadic
0001 110r	Arithmetic shift right r	Monadic
0001 111r	Bitwise Not r	Monadic
0010 000r	Rotate left r	Monadic
0010 001r	Rotate right r	Monadic
0010 010a	Branch unconditional	Dyadic
0010 011a	Branch if less than or equal to	Dyadic
0010 100a	Branch if less than	Dyadic
0010 101a	Branch if equal to	Dyadic
0010 110a	Branch if not equal to	Dyadic
0010 111a	Branch if greater than or equal to	Dyadic
0011 000a	Branch if greater than	Dyadic
0011 001a	Branch if V	Dyadic
0011 010a	Branch if C	Dyadic
0011 011a	Call subroutine	Dyadic
0011 1aaa	System call	Dyadic
0100 0aaa	Add to SP	Dyadic
0100 1aaa	Subtract from SP	Dyadic
0101 raaa	Add to r	Dyadic
0110 raaa	Subtract from r	Dyadic
0111 raaa	Bitwise And to r	Dyadic
1000 raaa	Bitwise Or to r	Dyadic
1001 raaa	Bitwise Exclusive Or to r	Dyadic
1010 raaa	Compare word to r	Dyadic
1011 raaa	Compare byte to r[8 : 15]	Dyadic
1100 raaa	Load word r from memory	Dyadic
1101 raaa	Load byte r[8 : 15] from memory	Dyadic
1110 raaa	Store word r to memory	Dyadic
1111 raaa	Store byte r[8 : 15] to memory	Dyadic

Instruction set

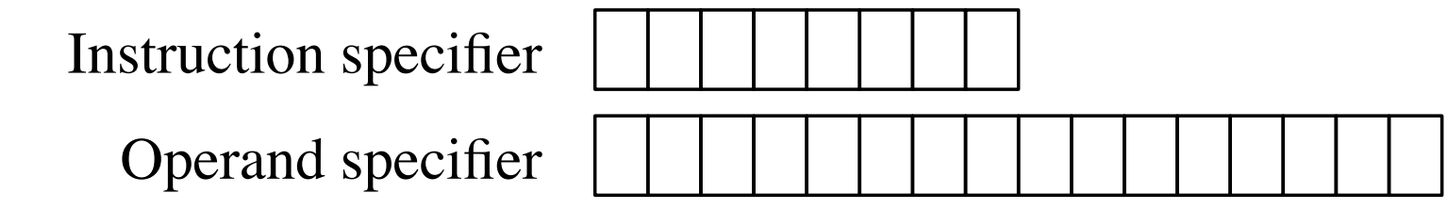
Instruction Specifier	Instruction	Type
0000 0000	Illegal instruction	
0000 0001	Return from call	Monadic
0000 0010	Return from system call	Monadic
0000 0011	Move SP to A	Monadic
0000 0100	Move A to SP	Monadic
0000 0101	Move NZVC flags to A[12 : 15]	Monadic
0000 0110	Move A[12 : 15] to NZVC flags	Monadic
0000 0111	No operation	Monadic
0001 100r	Negate r	Monadic
0001 101r	Arithmetic shift left r	Monadic
0001 110r	Arithmetic shift right r	Monadic
0001 111r	Bitwise Not r	Monadic
0010 000r	Rotate left r	Monadic
0010 001r	Rotate right r	Monadic
0010 010a	Branch unconditional	Dyadic
0010 011a	Branch if less than or equal to	Dyadic
0010 100a	Branch if less than	Dyadic
0010 101a	Branch if equal to	Dyadic
0010 110a	Branch if not equal to	Dyadic
0010 111a	Branch if greater than or equal to	Dyadic
0011 000a	Branch if greater than	Dyadic
0011 001a	Branch if V	Dyadic
0011 010a	Branch if C	Dyadic
0011 011a	Call subroutine	Dyadic
0011 1aaa	System call	Dyadic
0100 0aaa	Add to SP	Dyadic
0100 1aaa	Subtract from SP	Dyadic
0101 raaa	Add to r	Dyadic
0110 raaa	Subtract from r	Dyadic
0111 raaa	Bitwise And to r	Dyadic
1000 raaa	Bitwise Or to r	Dyadic
1001 raaa	Bitwise Exclusive Or to r	Dyadic
1010 raaa	Compare word to r	Dyadic
1011 raaa	Compare byte to r[8 : 15]	Dyadic
1100 raaa	Load word r from memory	Dyadic
1101 raaa	Load byte r[8 : 15] from memory	Dyadic
1110 raaa	Store word r to memory	Dyadic
1111 raaa	Store byte r[8 : 15] to memory	Dyadic

(a) A monadic instruction.

Instruction set

Instruction Specifier	Instruction	Type
0000 0000	Illegal instruction	
0000 0001	Return from call	Monadic
0000 0010	Return from system call	Monadic
0000 0011	Move SP to A	Monadic
0000 0100	Move A to SP	Monadic
0000 0101	Move NZVC flags to A[12 : 15]	Monadic
0000 0110	Move A[12 : 15] to NZVC flags	Monadic
0000 0111	No operation	Monadic
0001 100r	Negate r	Monadic
0001 101r	Arithmetic shift left r	Monadic
0001 110r	Arithmetic shift right r	Monadic
0001 111r	Bitwise Not r	Monadic
0010 000r	Rotate left r	Monadic
0010 001r	Rotate right r	Monadic
0010 010a	Branch unconditional	Dyadic
0010 011a	Branch if less than or equal to	Dyadic
0010 100a	Branch if less than	Dyadic
0010 101a	Branch if equal to	Dyadic
0010 110a	Branch if not equal to	Dyadic
0010 111a	Branch if greater than or equal to	Dyadic
0011 000a	Branch if greater than	Dyadic
0011 001a	Branch if V	Dyadic
0011 010a	Branch if C	Dyadic
0011 011a	Call subroutine	Dyadic
0011 1aaa	System call	Dyadic
0100 0aaa	Add to SP	Dyadic
0100 1aaa	Subtract from SP	Dyadic
0101 raaa	Add to r	Dyadic
0110 raaa	Subtract from r	Dyadic
0111 raaa	Bitwise And to r	Dyadic
1000 raaa	Bitwise Or to r	Dyadic
1001 raaa	Bitwise Exclusive Or to r	Dyadic
1010 raaa	Compare word to r	Dyadic
1011 raaa	Compare byte to r[8 : 15]	Dyadic
1100 raaa	Load word r from memory	Dyadic
1101 raaa	Load byte r[8 : 15] from memory	Dyadic
1110 raaa	Store word r to memory	Dyadic
1111 raaa	Store byte r[8 : 15] to memory	Dyadic

(a) A monadic instruction.



(b) The two parts of a dyadic instruction.

Instruction Specifier	Instruction	Type
0000 0000	Illegal instruction	
0000 0001	Return from call	Monadic
0000 0010	Return from system call	Monadic
0000 0011	Move SP to A	Monadic
0000 0100	Move A to SP	Monadic
0000 0101	Move NZVC flags to A[12 : 15]	Monadic
0000 0110	Move A[12 : 15] to NZVC flags	Monadic
0000 0111	No operation	Monadic
0001 100r	Negate r	Monadic
0001 101r	Arithmetic shift left r	Monadic
0001 110r	Arithmetic shift right r	Monadic
0001 111r	Bitwise Not r	Monadic
0010 000r	Rotate left r	Monadic
0010 001r	Rotate right r	Monadic

Instruction Specifier	Instruction	Type
0000 0000	Illegal instruction	
0000 0001	Return from call	Monadic
0000 0010	Return from system call	Monadic
0000 0011	Move SP to A	Monadic
0000 0100	Move A to SP	Monadic
0000 0101	Move NZVC flags to A[12 : 15]	Monadic
0000 0110	Move A[12 : 15] to NZVC flags	Monadic
0000 0111	No operation	Monadic
0001 100r	Negate r	Monadic
0001 101r	Arithmetic shift left r	Monadic
0001 110r	Arithmetic shift right r	Monadic
0001 111r	Bitwise Not r	Monadic
0010 000r	Rotate left r	Monadic
0010 001r	Rotate right r	Monadic

opcode

Instruction Specifier	Instruction	Type
0000 0000	Illegal instruction	
0000 0001	Return from call	Monadic
0000 0010	Return from system call	Monadic
0000 0011	Move SP to A	Monadic
0000 0100	Move A to SP	Monadic
0000 0101	Move NZVC flags to A[12 : 15]	Monadic
0000 0110	Move A[12 : 15] to NZVC flags	Monadic
0000 0111	No operation	Monadic
0001 100r	Negate r	Monadic
0001 101r	Arithmetic shift left r	Monadic
0001 110r	Arithmetic shift right r	Monadic
0001 111r	Bitwise Not r	Monadic
0010 000r	Rotate left r	Monadic
0010 001r	Rotate right r	Monadic

opcode register-r field

Instruction Specifier	Instruction	Type
0000 0000	Illegal instruction	
0000 0001	Return from call	Monadic
0000 0010	Return from system call	Monadic
0000 0011	Move SP to A	Monadic
0000 0100	Move A to SP	Monadic
0000 0101	Move NZVC flags to A[12 : 15]	Monadic
0000 0110	Move A[12 : 15] to NZVC flags	Monadic
0000 0111	No operation	Monadic
0001 100r	Negate r	Monadic
0001 101r	Arithmetic shift left r	Monadic
0001 110r	Arithmetic shift right r	Monadic
0001 111r	Bitwise Not r	Monadic
0010 000r	Rotate left r	Monadic
0010 001r	Rotate right r	Monadic

opcode register-r field

r	Register
0	Accumulator (A)
1	Index register (X)

(c) The register-r field.

Instruction Specifier	Instruction	Type
0000 0000	Illegal instruction	
0000 0001	Return from call	Monadic
0000 0010	Return from system call	Monadic
0000 0011	Move SP to A	Monadic
0000 0100	Move A to SP	Monadic
0000 0101	Move NZVC flags to A[12 : 15]	Monadic
0000 0110	Move A[12 : 15] to NZVC flags	Monadic
0000 0111	No operation	Monadic
0001 100r	Negate r	Monadic
0001 101r	Arithmetic shift left r	Monadic
0001 110r	Arithmetic shift right r	Monadic
0001 111r	Bitwise Not r	Monadic
0010 000r	Rotate left r	Monadic
0010 001r	Rotate right r	Monadic

opcode register-r field

r	Register
0	Accumulator (A)
1	Index register (X)

(c) The register-r field.

0010 0010 Rotate right accumulator
0010 0011 Rotate right index register

0010 010a	Branch unconditional	Dyadic
0010 011a	Branch if less than or equal to	Dyadic
0010 100a	Branch if less than	Dyadic
0010 101a	Branch if equal to	Dyadic
0010 110a	Branch if not equal to	Dyadic
0010 111a	Branch if greater than or equal to	Dyadic
0011 000a	Branch if greater than	Dyadic
0011 001a	Branch if V	Dyadic
0011 010a	Branch if C	Dyadic
0011 011a	Call subroutine	Dyadic
0011 1aaa	System call	Dyadic
0100 0aaa	Add to SP	Dyadic
0100 1aaa	Subtract from SP	Dyadic

0010 010a	Branch unconditional	Dyadic
0010 011a	Branch if less than or equal to	Dyadic
0010 100a	Branch if less than	Dyadic
0010 101a	Branch if equal to	Dyadic
0010 110a	Branch if not equal to	Dyadic
0010 111a	Branch if greater than or equal to	Dyadic
0011 000a	Branch if greater than	Dyadic
0011 001a	Branch if V	Dyadic
0011 010a	Branch if C	Dyadic
0011 011a	Call subroutine	Dyadic
0011 1aaa	System call	Dyadic
0100 0aaa	Add to SP	Dyadic
0100 1aaa	Subtract from SP	Dyadic

opcode

0010 010a	Branch unconditional	Dyadic
0010 011a	Branch if less than or equal to	Dyadic
0010 100a	Branch if less than	Dyadic
0010 101a	Branch if equal to	Dyadic
0010 110a	Branch if not equal to	Dyadic
0010 111a	Branch if greater than or equal to	Dyadic
0011 000a	Branch if greater than	Dyadic
0011 001a	Branch if V	Dyadic
0011 010a	Branch if C	Dyadic
0011 011a	Call subroutine	Dyadic
0011 1aaa	System call	Dyadic
0100 0aaa	Add to SP	Dyadic
0100 1aaa	Subtract from SP	Dyadic

opcode

addressing-a field

0010 010a	Branch unconditional	Dyadic
0010 011a	Branch if less than or equal to	Dyadic
0010 100a	Branch if less than	Dyadic
0010 101a	Branch if equal to	Dyadic
0010 110a	Branch if not equal to	Dyadic
0010 111a	Branch if greater than or equal to	Dyadic
0011 000a	Branch if greater than	Dyadic
0011 001a	Branch if V	Dyadic
0011 010a	Branch if C	Dyadic
0011 011a	Call subroutine	Dyadic
0011 1aaa	System call	Dyadic
0100 0aaa	Add to SP	Dyadic
0100 1aaa	Subtract from SP	Dyadic

opcode

addressing-a field

a	Addressing Mode
0	Immediate
1	Indexed

(b) The addressing-a field.

0010 010a	Branch unconditional	Dyadic
0010 011a	Branch if less than or equal to	Dyadic
0010 100a	Branch if less than	Dyadic
0010 101a	Branch if equal to	Dyadic
0010 110a	Branch if not equal to	Dyadic
0010 111a	Branch if greater than or equal to	Dyadic
0011 000a	Branch if greater than	Dyadic
0011 001a	Branch if V	Dyadic
0011 010a	Branch if C	Dyadic
0011 011a	Call subroutine	Dyadic
0011 1aaa	System call	Dyadic
0100 0aaa	Add to SP	Dyadic
0100 1aaa	Subtract from SP	Dyadic

opcode

addressing-a field

a	Addressing Mode
0	Immediate
1	Indexed

(b) The addressing-a field.

0011 1110 Call subroutine with immediate addressing
0010 0011 Call subroutine with indexed addressing

	0010 010a	Branch unconditional	Dyadic
	0010 011a	Branch if less than or equal to	Dyadic
	0010 100a	Branch if less than	Dyadic
	0010 101a	Branch if equal to	Dyadic
	0010 110a	Branch if not equal to	Dyadic
	0010 111a	Branch if greater than or equal to	Dyadic
	0011 000a	Branch if greater than	Dyadic
	0011 001a	Branch if V	Dyadic
	0011 010a	Branch if C	Dyadic
	0011 011a	Call subroutine	Dyadic
	0011 1aaa	System call	Dyadic
opcode	0100 0aaa	Add to SP	Dyadic
	0100 1aaa	Subtract from SP	Dyadic

	0010 010a	Branch unconditional	Dyadic
	0010 011a	Branch if less than or equal to	Dyadic
	0010 100a	Branch if less than	Dyadic
	0010 101a	Branch if equal to	Dyadic
	0010 110a	Branch if not equal to	Dyadic
	0010 111a	Branch if greater than or equal to	Dyadic
	0011 000a	Branch if greater than	Dyadic
	0011 001a	Branch if V	Dyadic
	0011 010a	Branch if C	Dyadic
	0011 011a	Call subroutine	Dyadic
	0011 1aaa	System call	Dyadic
opcode	0100 0aaa	Add to SP	Dyadic
	0100 1aaa	Subtract from SP	Dyadic

addressing-aaa field

0010 010a	Branch unconditional	Dyadic
0010 011a	Branch if less than or equal to	Dyadic
0010 100a	Branch if less than	Dyadic
0010 101a	Branch if equal to	Dyadic
0010 110a	Branch if not equal to	Dyadic
0010 111a	Branch if greater than or equal to	Dyadic
0011 000a	Branch if greater than	Dyadic
0011 001a	Branch if V	Dyadic
0011 010a	Branch if C	Dyadic
0011 011a	Call subroutine	Dyadic
0011 1aaa	System call	Dyadic
0100 0aaa	Add to SP	Dyadic
0100 1aaa	Subtract from SP	Dyadic

opcode

addressing-aaa field

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.

0010 010a	Branch unconditional	Dyadic
0010 011a	Branch if less than or equal to	Dyadic
0010 100a	Branch if less than	Dyadic
0010 101a	Branch if equal to	Dyadic
0010 110a	Branch if not equal to	Dyadic
0010 111a	Branch if greater than or equal to	Dyadic
0011 000a	Branch if greater than	Dyadic
0011 001a	Branch if V	Dyadic
0011 010a	Branch if C	Dyadic
0011 011a	Call subroutine	Dyadic
0011 1aaa	System call	Dyadic
0100 0aaa	Add to SP	Dyadic
0100 1aaa	Subtract from SP	Dyadic

opcode

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.

addressing-aaa field

0100 0000 Add to SP with immediate addressing
0100 0001 Add to SP with direct addressing
0100 0010 Add to SP with indirect addressing
0100 0011 Add to SP with stack-relative addressing
…

0101 raaa	Add to r	Dyadic
0110 raaa	Subtract from r	Dyadic
0111 raaa	Bitwise And to r	Dyadic
1000 raaa	Bitwise Or to r	Dyadic
1001 raaa	Bitwise Exclusive Or to r	Dyadic
1010 raaa	Compare word to r	Dyadic
1011 raaa	Compare byte to r[8 : 15]	Dyadic
1100 raaa	Load word r from memory	Dyadic
1101 raaa	Load byte r[8 : 15] from memory	Dyadic
1110 raaa	Store word r to memory	Dyadic
1111 raaa	Store byte r[8 : 15] to memory	Dyadic

0101 raaa	Add to r	Dyadic
0110 raaa	Subtract from r	Dyadic
0111 raaa	Bitwise And to r	Dyadic
1000 raaa	Bitwise Or to r	Dyadic
1001 raaa	Bitwise Exclusive Or to r	Dyadic
1010 raaa	Compare word to r	Dyadic
1011 raaa	Compare byte to r[8 : 15]	Dyadic
1100 raaa	Load word r from memory	Dyadic
1101 raaa	Load byte r[8 : 15] from memory	Dyadic
1110 raaa	Store word r to memory	Dyadic
1111 raaa	Store byte r[8 : 15] to memory	Dyadic

opcode

0101 raaa	Add to r	Dyadic
0110 raaa	Subtract from r	Dyadic
0111 raaa	Bitwise And to r	Dyadic
1000 raaa	Bitwise Or to r	Dyadic
1001 raaa	Bitwise Exclusive Or to r	Dyadic
1010 raaa	Compare word to r	Dyadic
1011 raaa	Compare byte to r[8 : 15]	Dyadic
1100 raaa	Load word r from memory	Dyadic
1101 raaa	Load byte r[8 : 15] from memory	Dyadic
1110 raaa	Store word r to memory	Dyadic
1111 raaa	Store byte r[8 : 15] to memory	Dyadic

opcode

register-r field

0101 raaa	Add to r	Dyadic
0110 raaa	Subtract from r	Dyadic
0111 raaa	Bitwise And to r	Dyadic
1000 raaa	Bitwise Or to r	Dyadic
1001 raaa	Bitwise Exclusive Or to r	Dyadic
1010 raaa	Compare word to r	Dyadic
1011 raaa	Compare byte to r[8 : 15]	Dyadic
1100 raaa	Load word r from memory	Dyadic
1101 raaa	Load byte r[8 : 15] from memory	Dyadic
1110 raaa	Store word r to memory	Dyadic
1111 raaa	Store byte r[8 : 15] to memory	Dyadic

opcode

register-r field

addressing-aaa field

0101 raaa	Add to r	Dyadic
0110 raaa	Subtract from r	Dyadic
0111 raaa	Bitwise And to r	Dyadic
1000 raaa	Bitwise Or to r	Dyadic
1001 raaa	Bitwise Exclusive Or to r	Dyadic
1010 raaa	Compare word to r	Dyadic
1011 raaa	Compare byte to r[8 : 15]	Dyadic
1100 raaa	Load word r from memory	Dyadic
1101 raaa	Load byte r[8 : 15] from memory	Dyadic
1110 raaa	Store word r to memory	Dyadic
1111 raaa	Store byte r[8 : 15] to memory	Dyadic

opcode

register-r field

addressing-aaa field

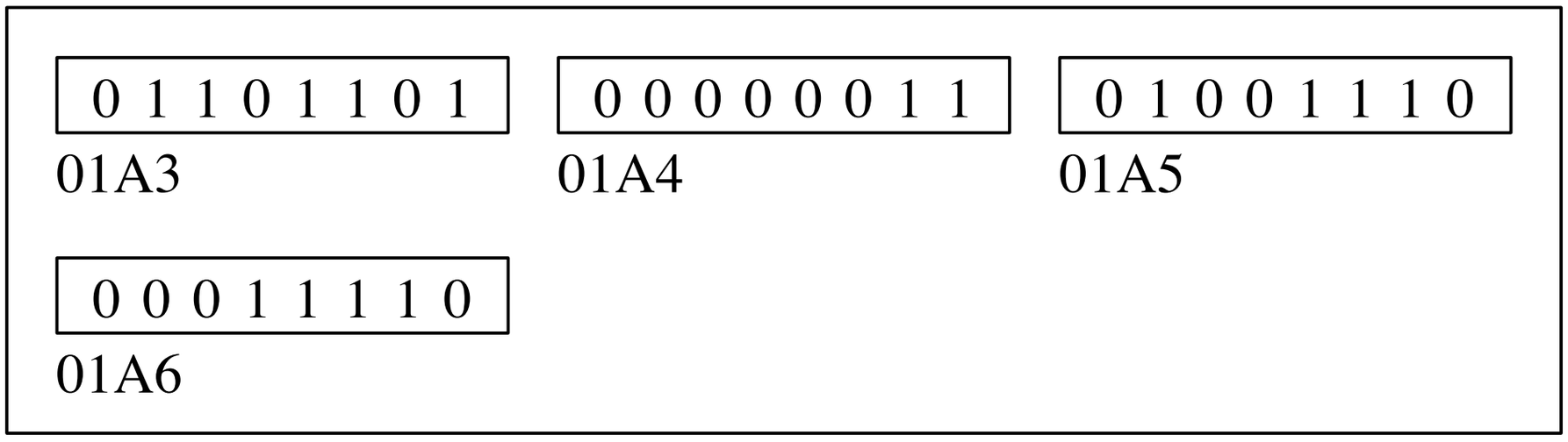
1111 0100 Store byte accumulator to memory with stack-relative deferred addressing
1111 1100 Store byte index register to memory with stack-relative deferred addressing

A machine language listing of two instructions in main memory

```
01A3    6D034E
01A6    1E
```

A machine language listing of two instructions in main memory

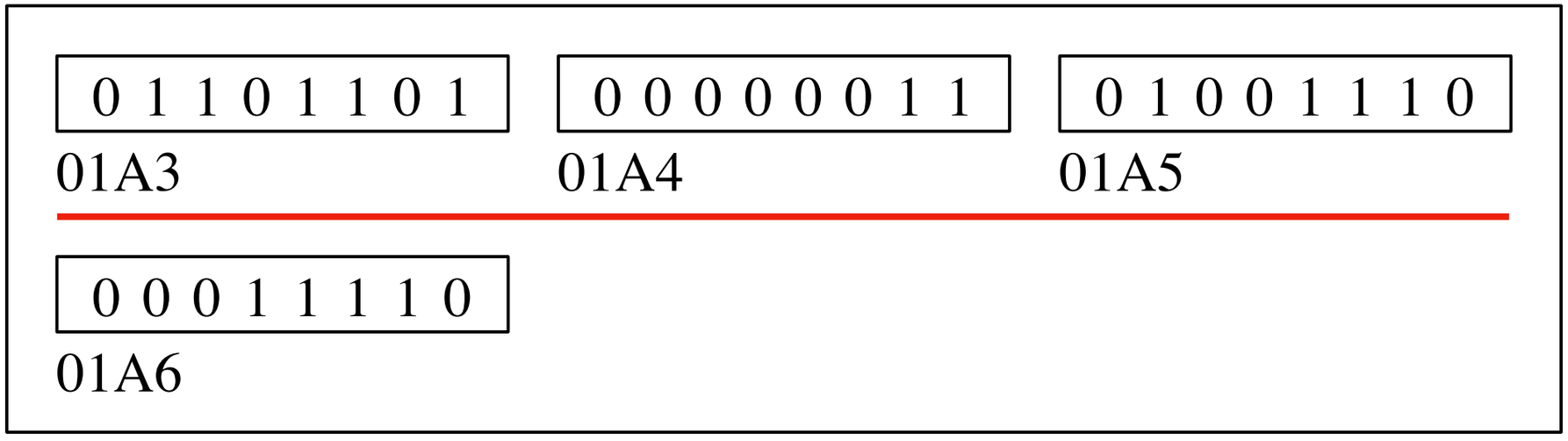
01A3 6D034E
01A6 1E



A machine language listing of two instructions in main memory

01A3	6D034E
01A6	1E

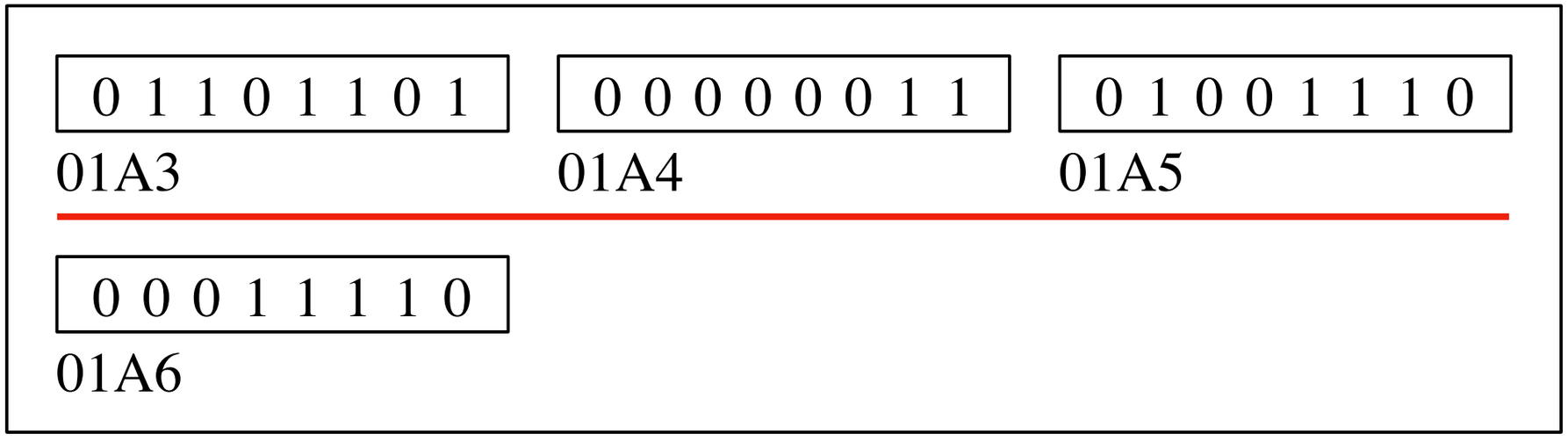
Main memory



A machine language listing of two instructions in main memory

01A3	6D034E
01A6	1E

Main memory

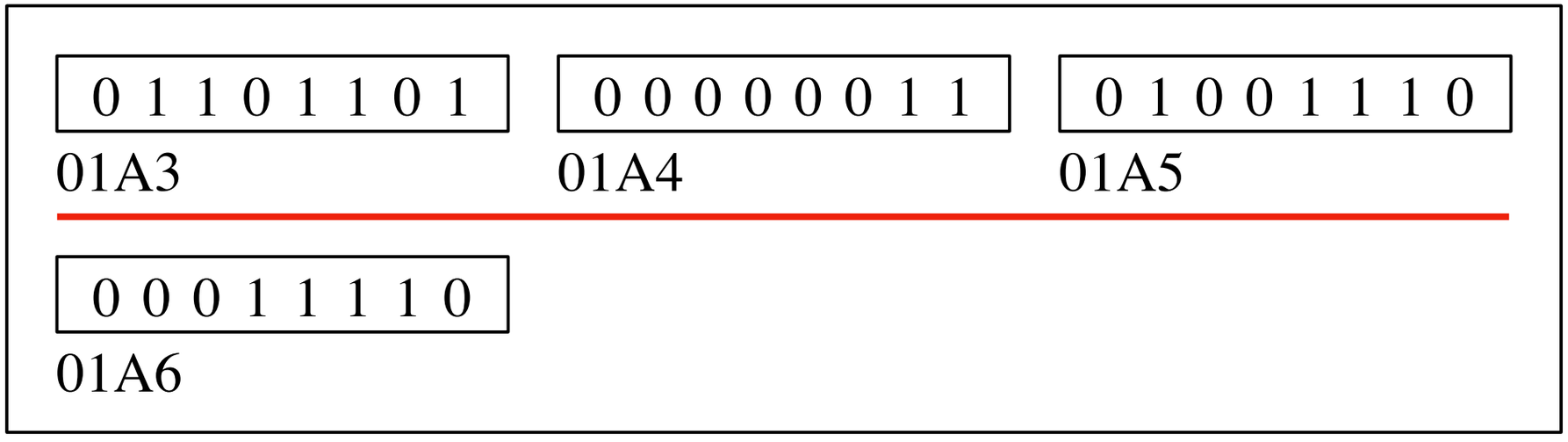


Opcode:
Register-r field:
Addressing-aaa field:
Operand specifier:

A machine language listing of two instructions in main memory

01A3	6D034E
01A6	1E

Main memory

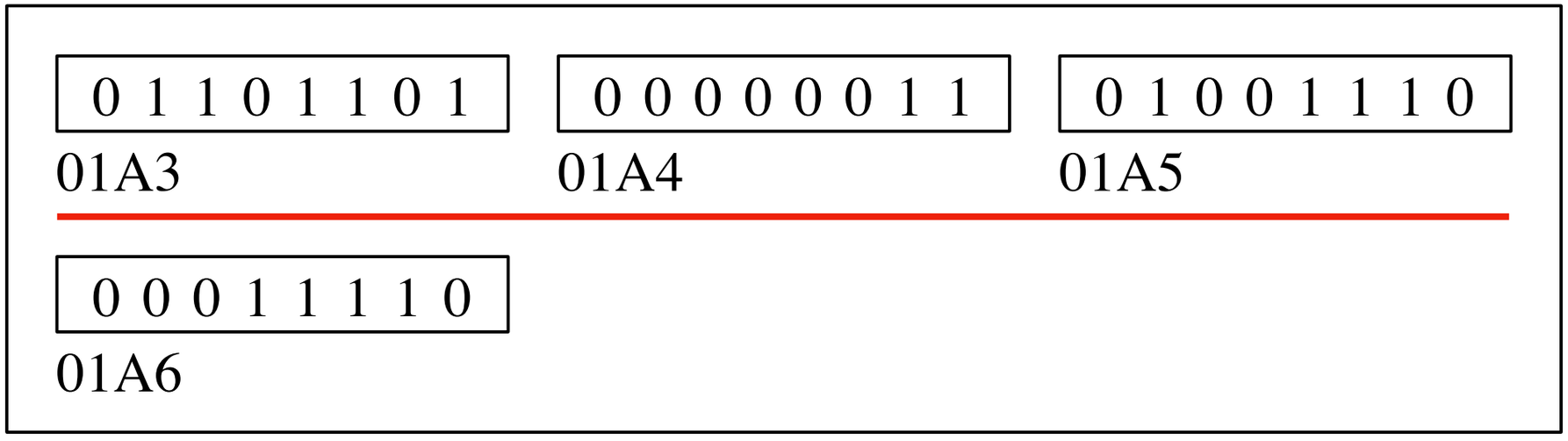


Opcode: 0110
Register-r field:
Addressing-aaa field:
Operand specifier:

A machine language listing of two instructions in main memory

01A3	6D034E
01A6	1E

Main memory

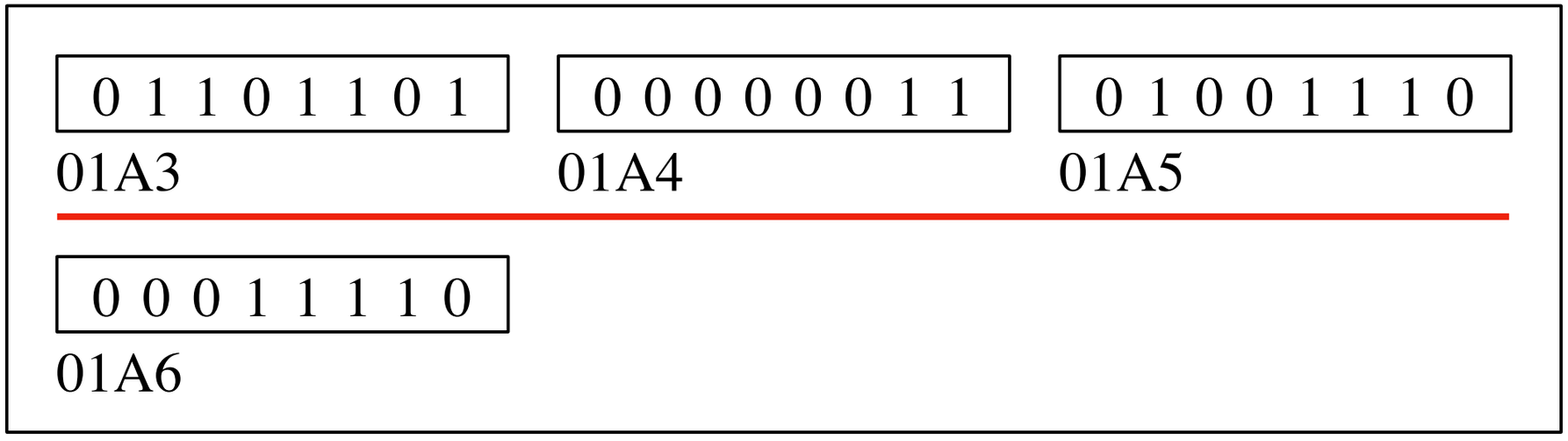


Opcode: 0110
Register-r field: 1
Addressing-aaa field:
Operand specifier:

A machine language listing of two instructions in main memory

01A3	6D034E
01A6	1E

Main memory

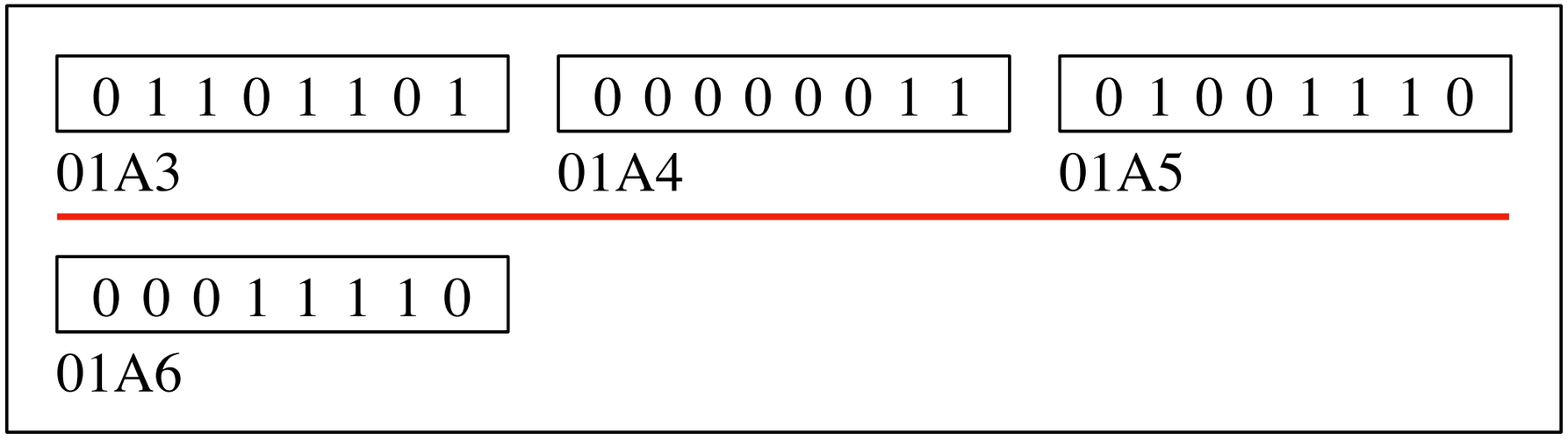


Opcode: 0110
Register-r field: 1
Addressing-aaa field: 101
Operand specifier:

A machine language listing of two instructions in main memory

01A3	6D034E
01A6	1E

Main memory

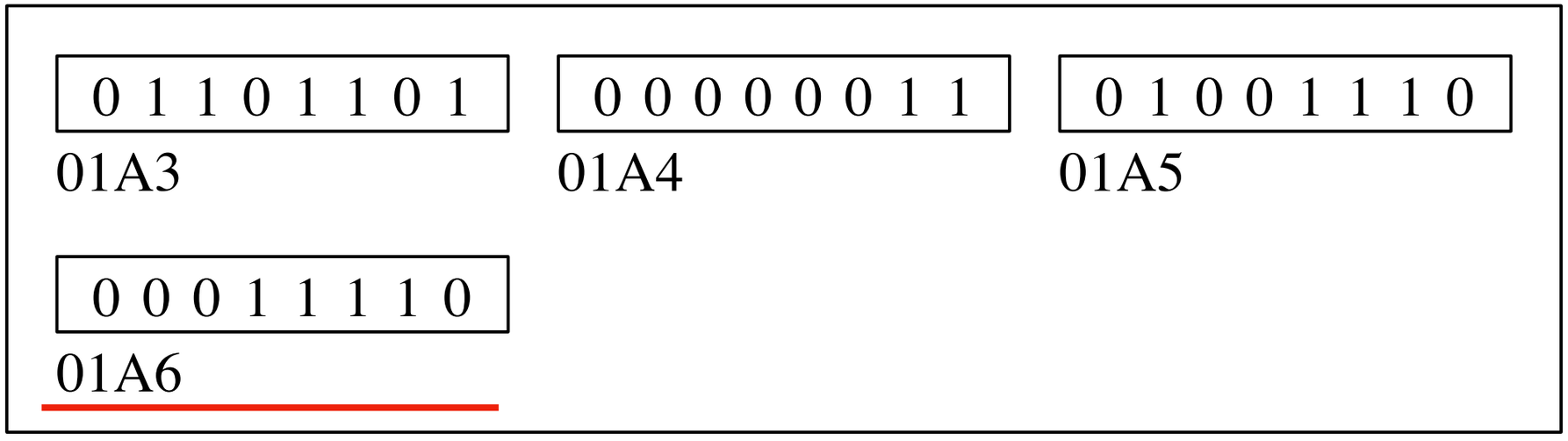


Opcode: 0110
Register-r field: 1
Addressing-aaa field: 101
Operand specifier: 0000 0011 0100 1110

A machine language listing of two instructions in main memory

01A3 6D034E
01A6 1E

Main memory

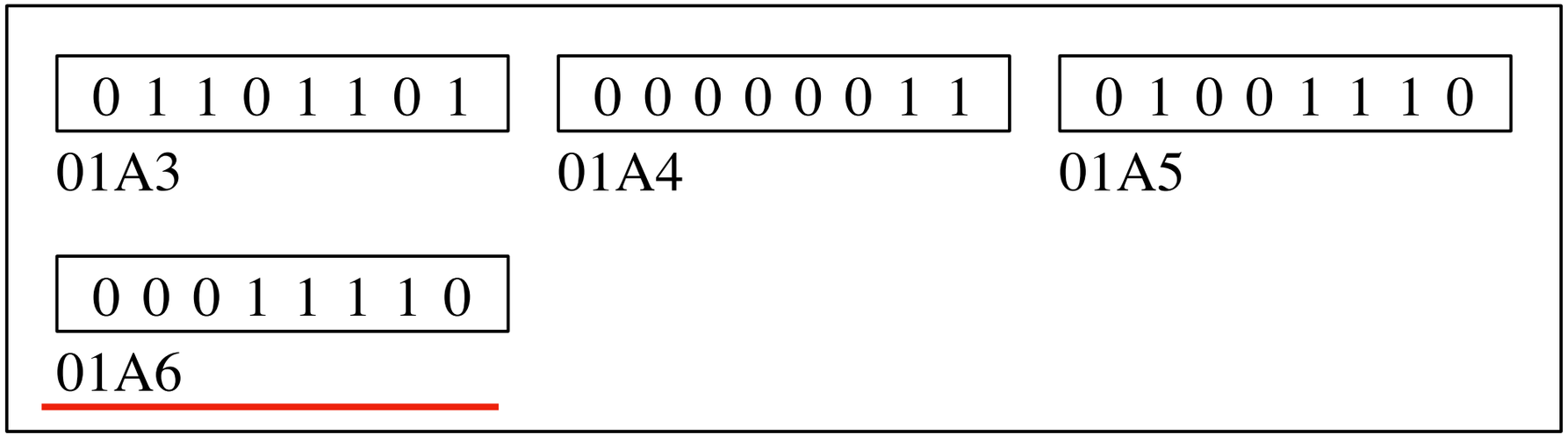


Opcode: 0110
Register-r field: 1
Addressing-aaa field: 101
Operand specifier: 0000 0011 0100 1110

A machine language listing of two instructions in main memory

01A3 6D034E
01A6 1E

Main memory



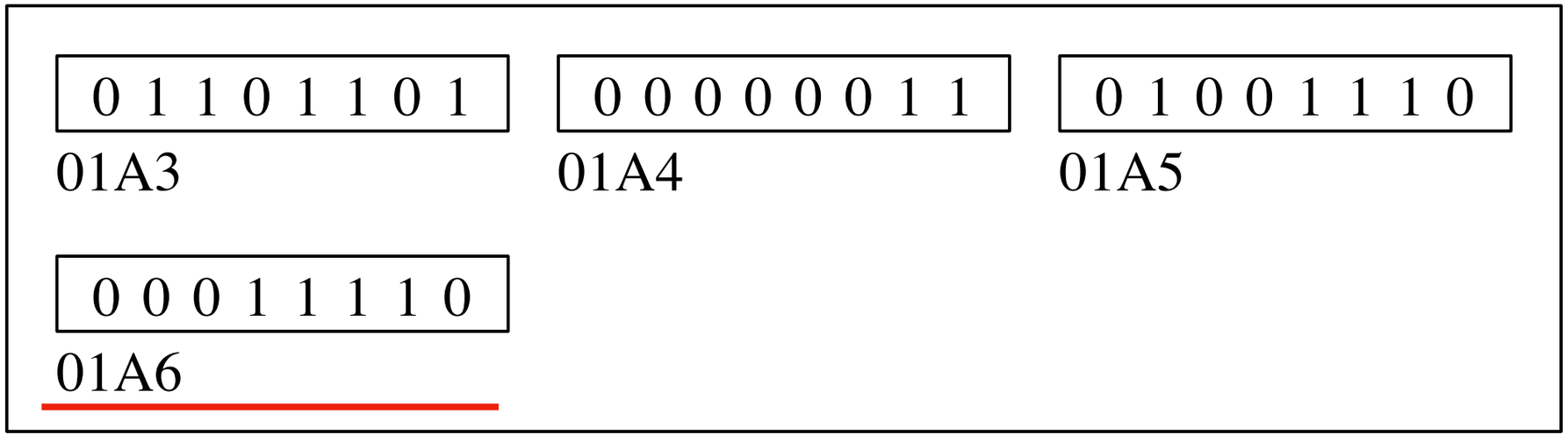
Opcode: 0110
Register-r field: 1
Addressing-aaa field: 101
Operand specifier: 0000 0011 0100 1110

Opcode:
Register-r field:

A machine language listing of two instructions in main memory

01A3 6D034E
01A6 1E

Main memory



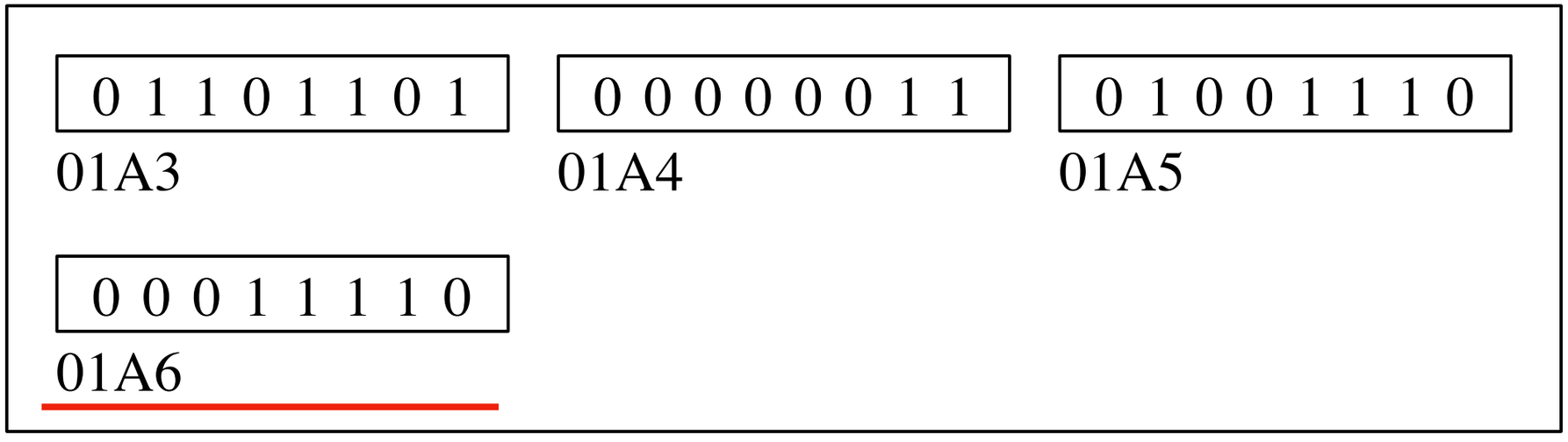
Opcode: 0110
Register-r field: 1
Addressing-aaa field: 101
Operand specifier: 0000 0011 0100 1110

Opcode: 0001 111
Register-r field:

A machine language listing of two instructions in main memory

01A3 6D034E
01A6 1E

Main memory



Opcode: 0110
Register-r field: 1
Addressing-aaa field: 101
Operand specifier: 0000 0011 0100 1110

Opcode: 0001 111
Register-r field: 0

Direct addressing

Direct addressing

- Oprnd = Mem[OprndSpec]

Direct addressing

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

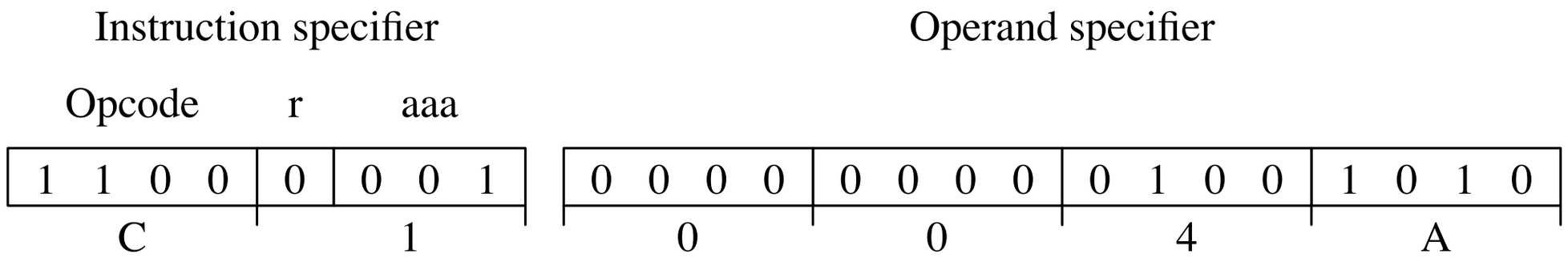
The load/store instructions

- Load word instruction
- Store word instruction
- Load byte instruction
- Store byte instruction

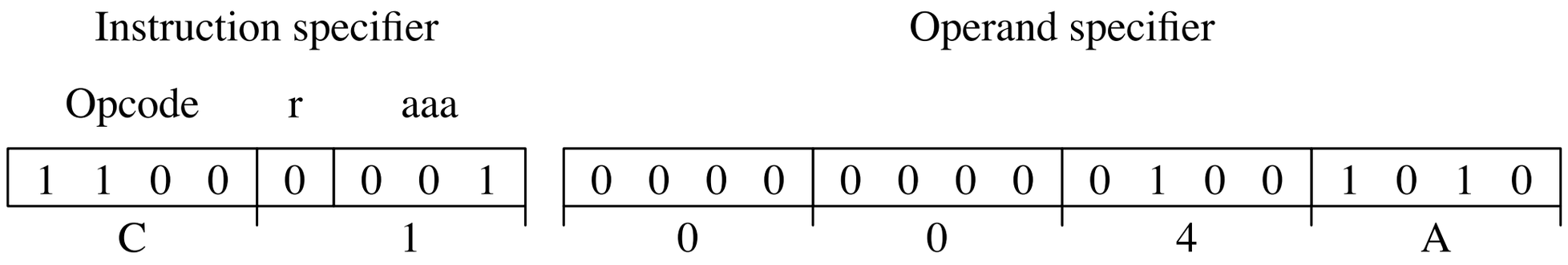
The load word instruction

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

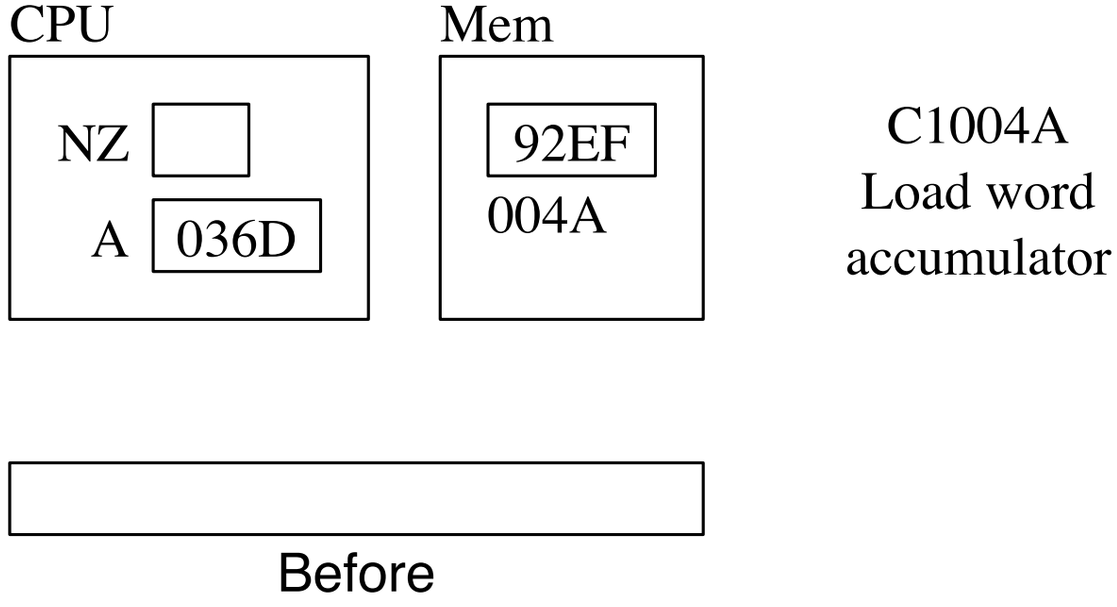
$$r \leftarrow \text{Oprnd} \; ; \; N \leftarrow r < 0 \; , \; Z \leftarrow r = 0$$



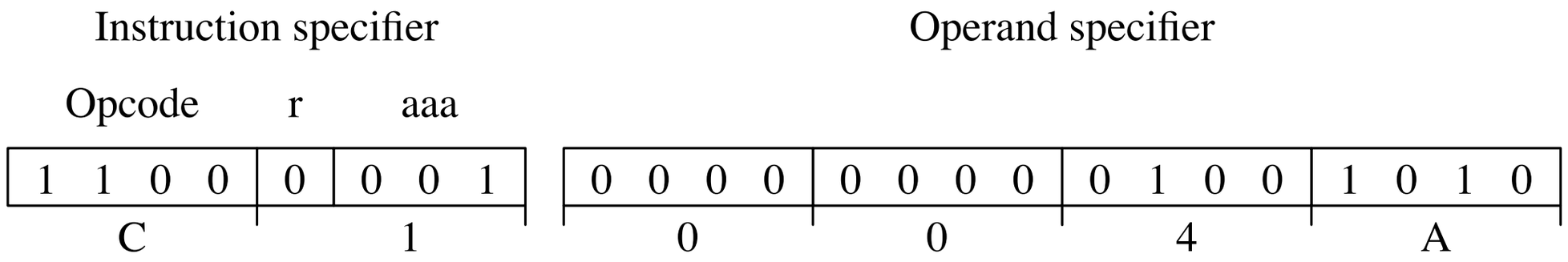
(a) Instruction format.



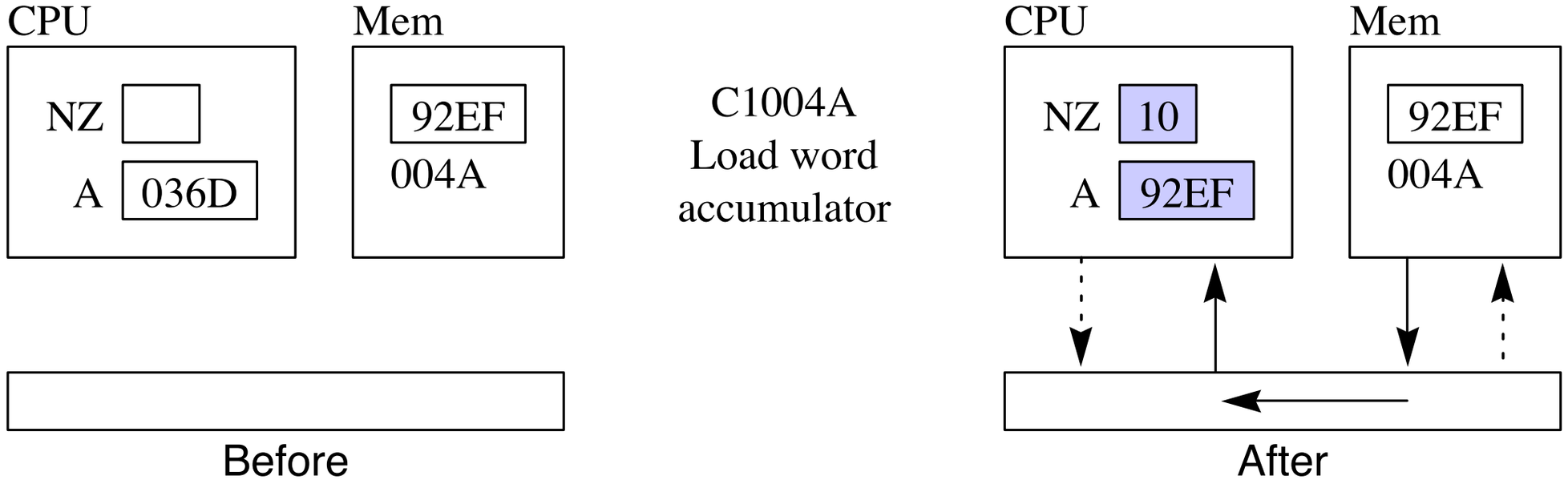
(a) Instruction format.



(b) Execution



(a) Instruction format.

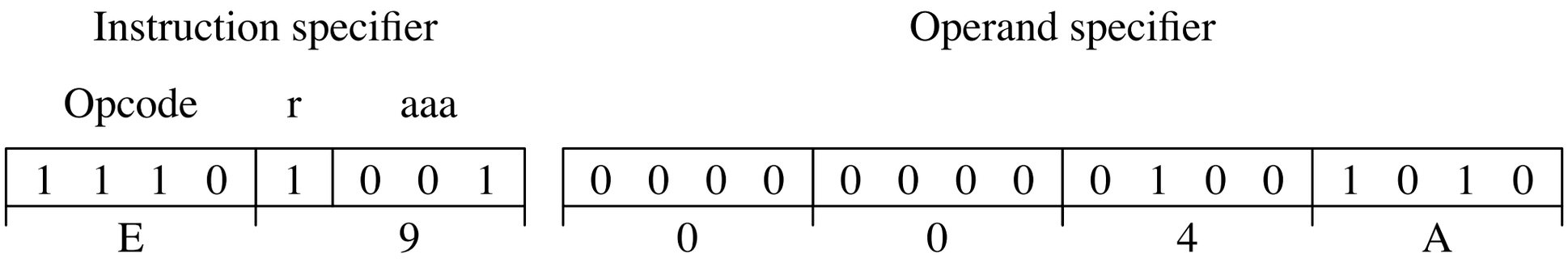


(b) Execution

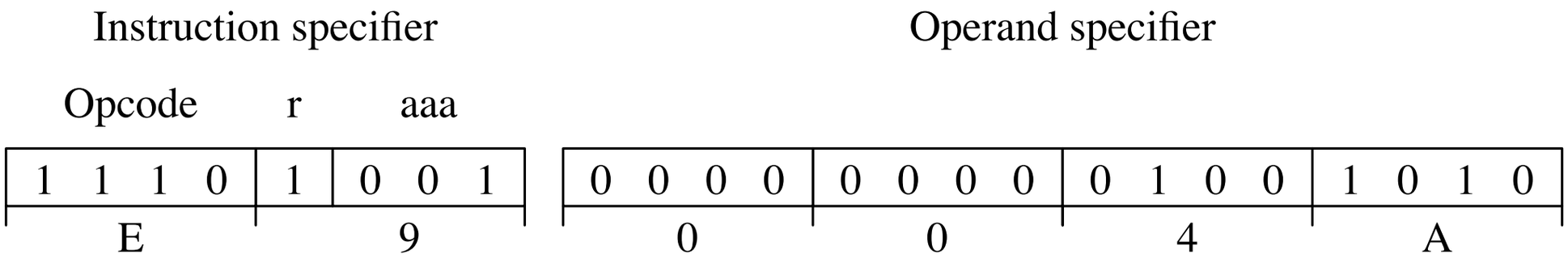
The store word instruction

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

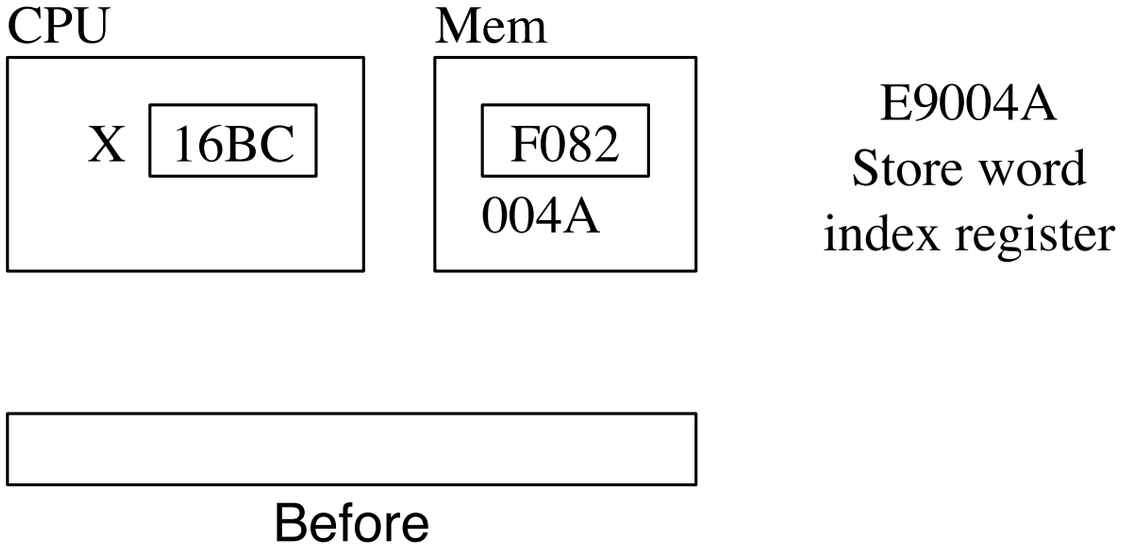
$$\text{Oprnd} \leftarrow \text{r}$$



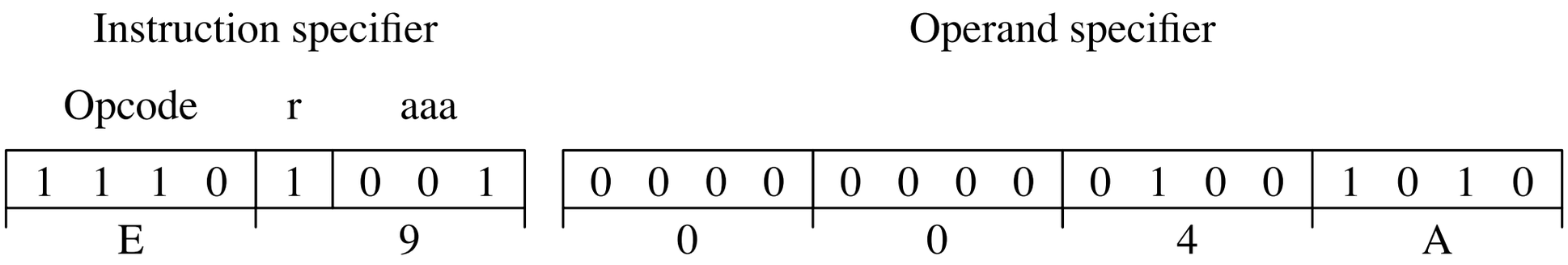
(a) Instruction format.



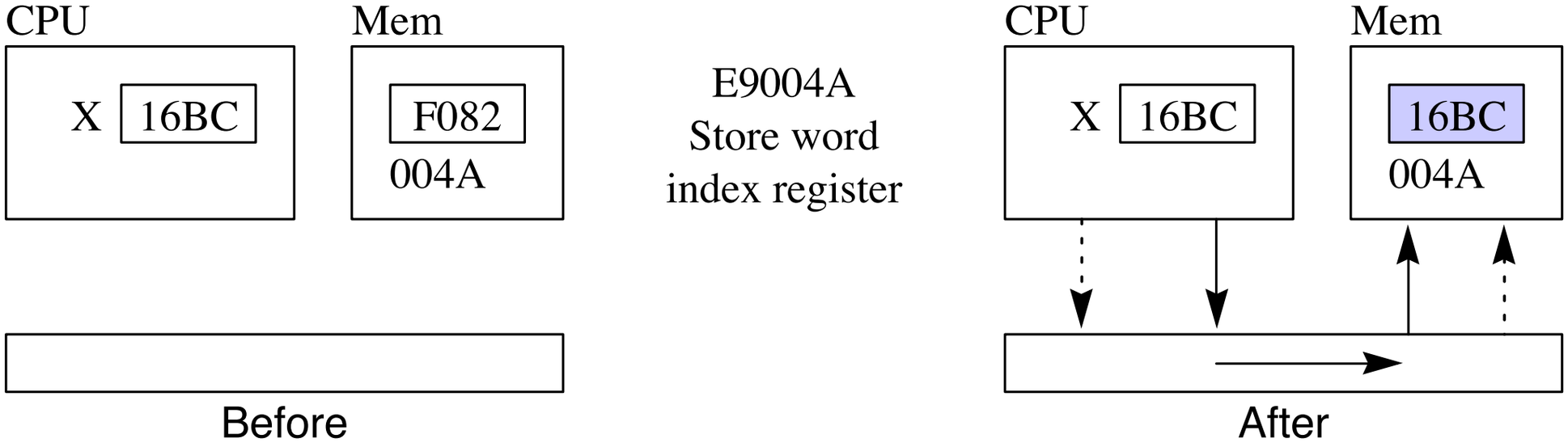
(a) Instruction format.



(b) Execution



(a) Instruction format.



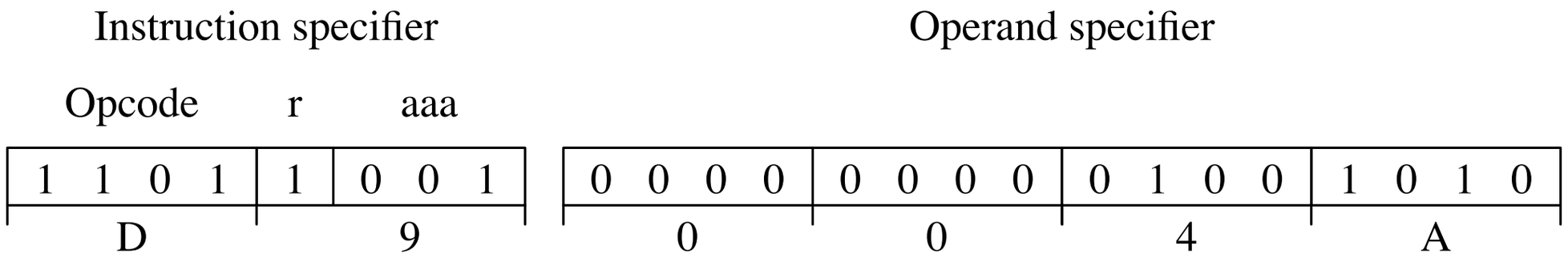
(b) Execution

The load byte instruction

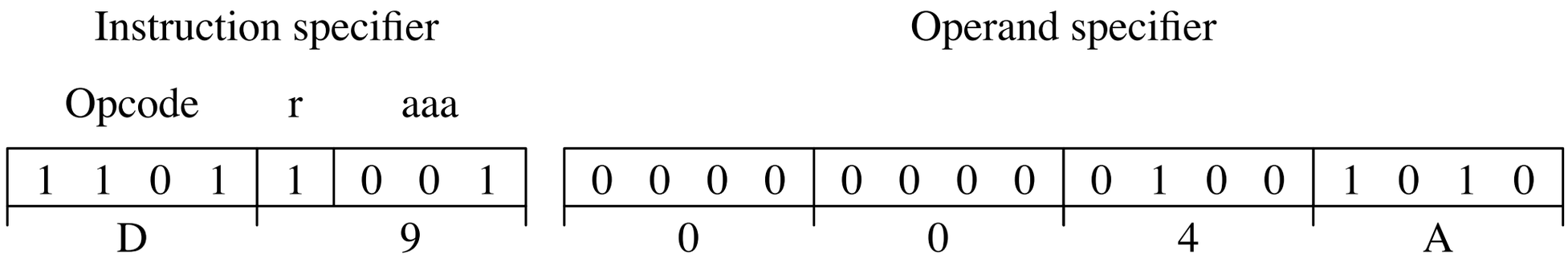
- Instruction specifier: 1101 raaa
- Loads one byte from memory to register r

$$r[0:7] \leftarrow 0\ ,\ r[8:15] \leftarrow \text{byte Oprnd}\ ;$$

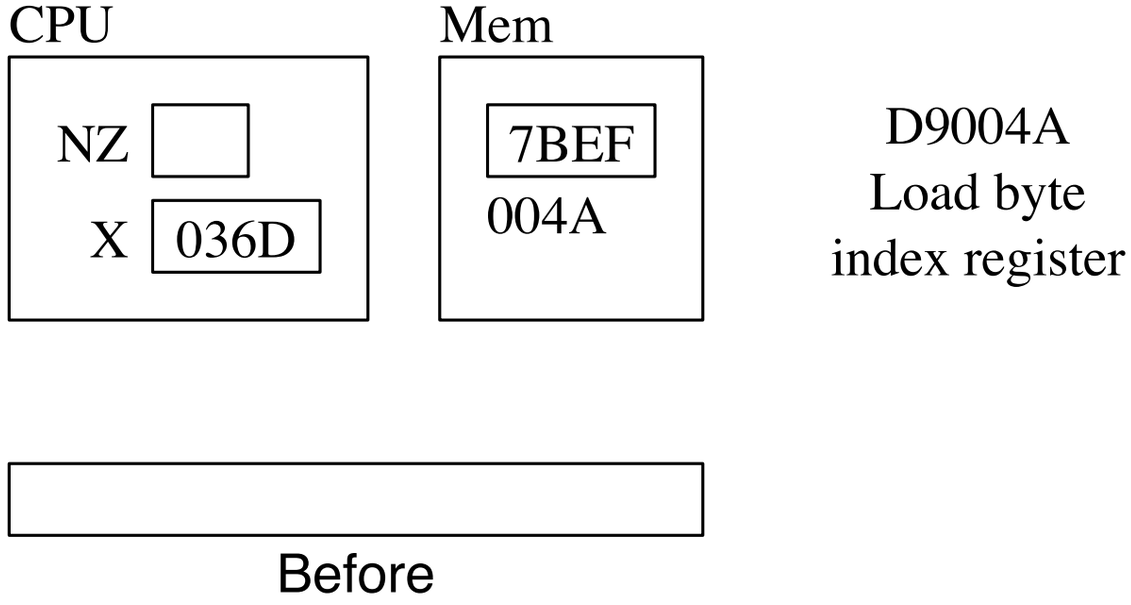
$$N \leftarrow 0\ ,\ Z \leftarrow r[8:15] = 0$$



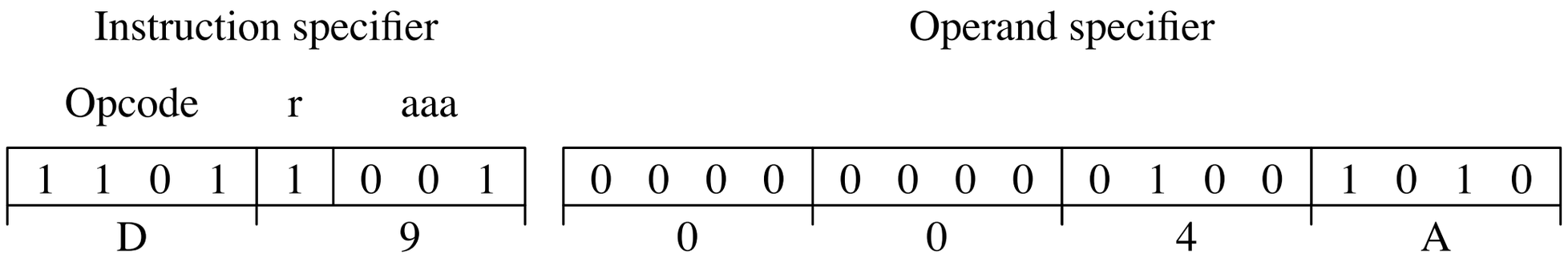
(a) Instruction format.



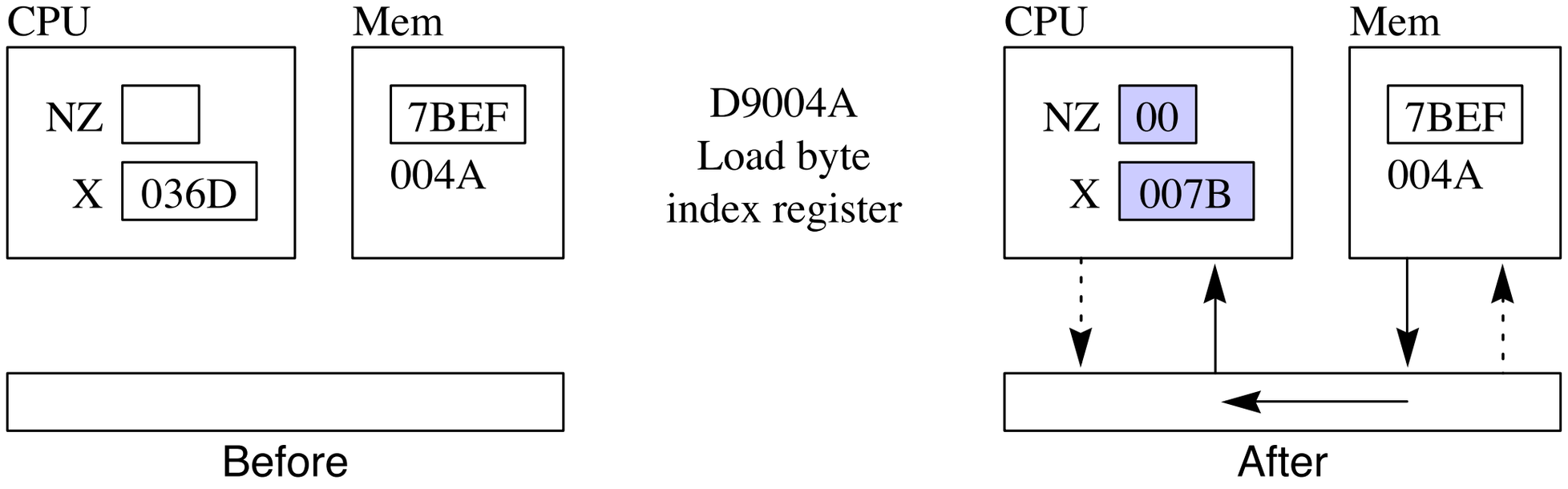
(a) Instruction format.



(b) Execution



(a) Instruction format.

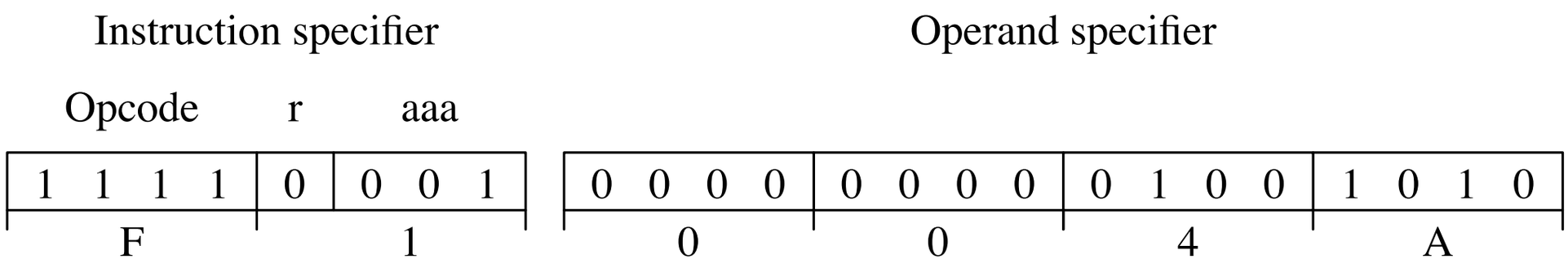


(b) Execution

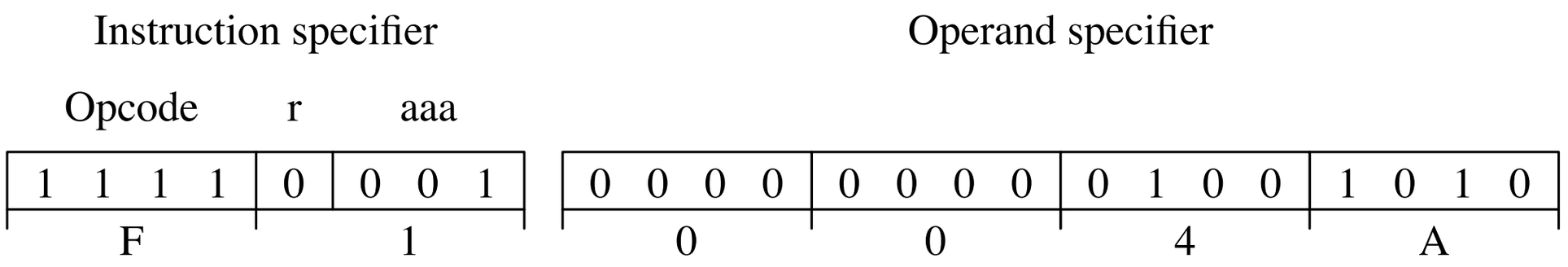
The store byte instruction

- Instruction specifier: 1111 raaa
- Stores one byte from register r to memory

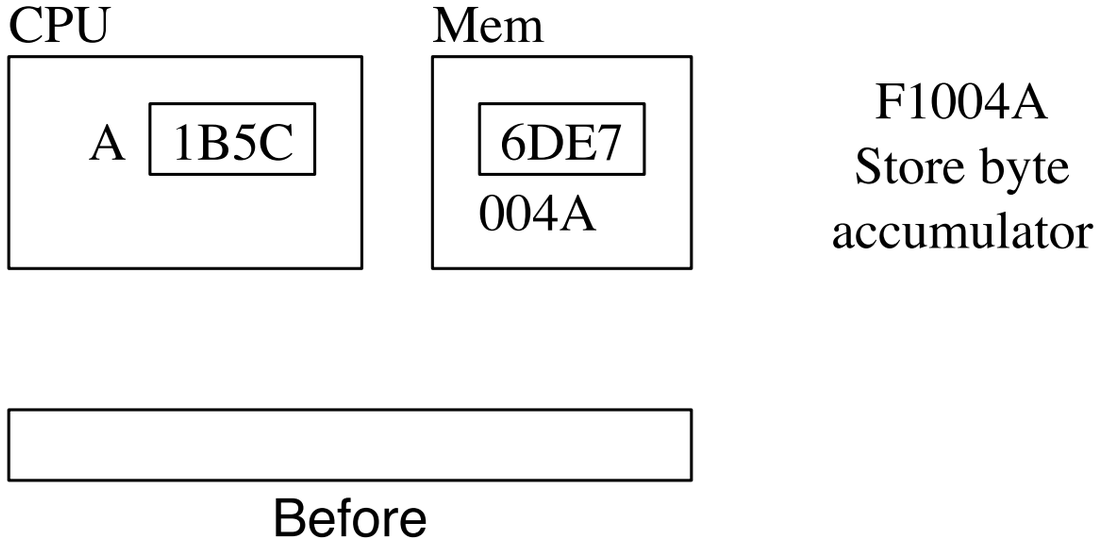
$$\text{byte Oprnd} \leftarrow \text{r}[8 : 15]$$



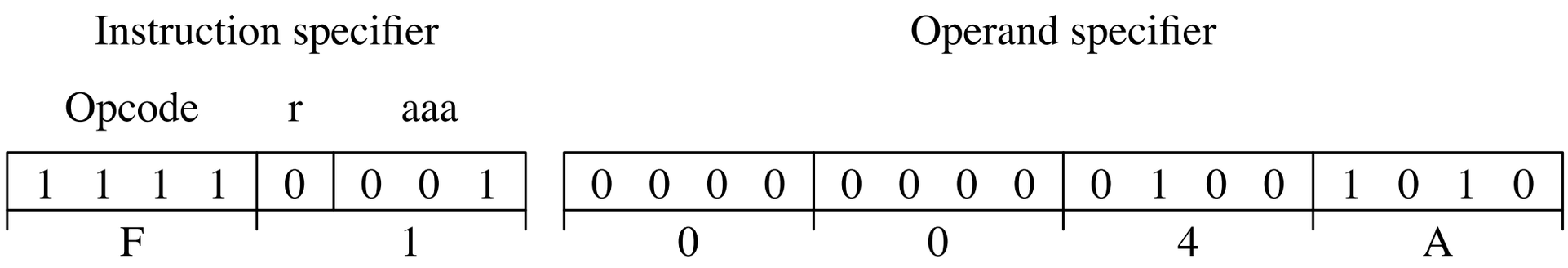
(a) Instruction format.



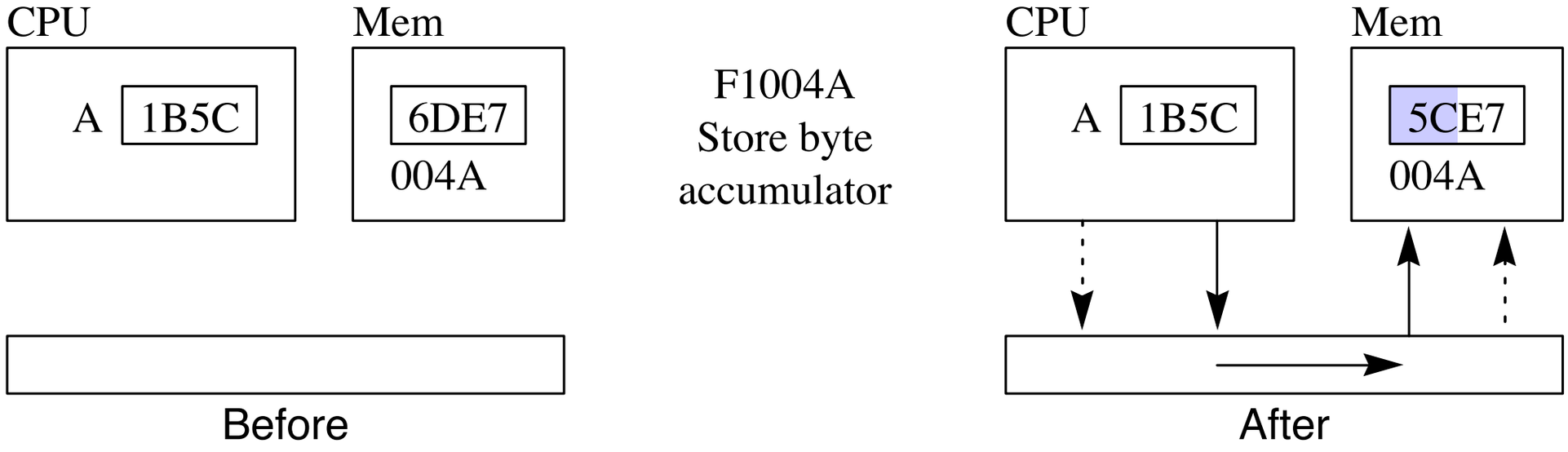
(a) Instruction format.



(b) Execution



(a) Instruction format.



(b) Execution

Load/Store summary

- Load is memory to CPU
- Store is CPU to memory

The arithmetic instructions

- Dyadic arithmetic instructions
  - The add instruction
  - The subtract instruction

The arithmetic instructions

- Dyadic arithmetic instructions
  - The add instruction
  - The subtract instruction
- Monadic arithmetic instructions
  - The negate instruction
  - The arithmetic shift left instruction
  - The arithmetic shift right instruction

Register transfer language

Add instruction RTL

$$\mathrm{r} \leftarrow \mathrm{r} + \mathrm{Oprnd}\ ;\ \mathrm{N} \leftarrow \mathrm{r} < 0\ ,\ \mathrm{Z} \leftarrow \mathrm{r} = 0\ ,\ \mathrm{V} \leftarrow \{overflow\}\ ,\ \mathrm{C} \leftarrow \{carry\}$$

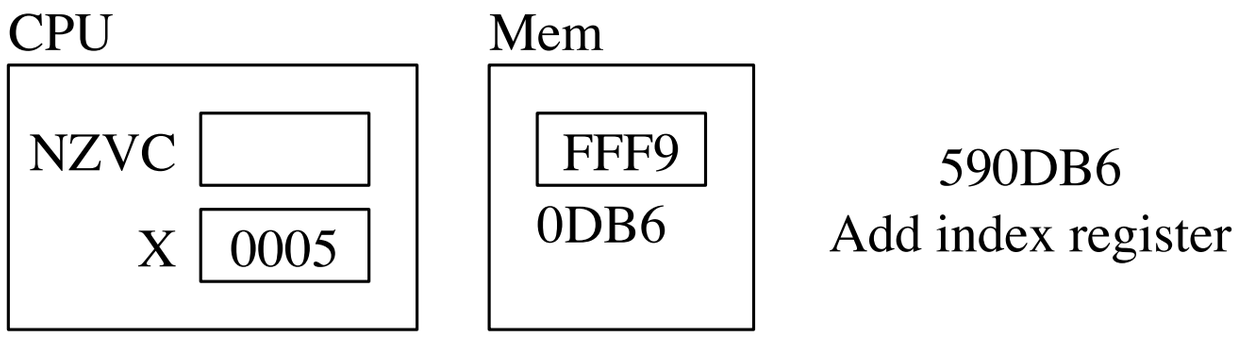
Register transfer language

Add instruction RTL

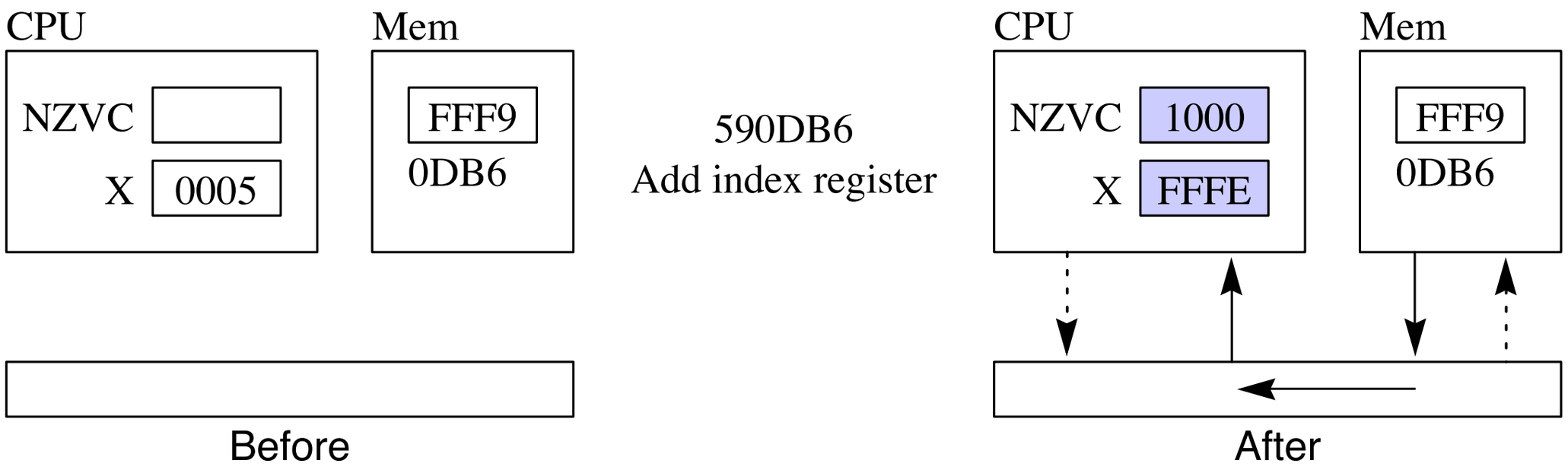
$$\mathrm{r} \leftarrow \mathrm{r} + \mathrm{Oprnd}\ ;\ \mathrm{N} \leftarrow \mathrm{r} < 0\ ,\ \mathrm{Z} \leftarrow \mathrm{r} = 0\ ,\ \mathrm{V} \leftarrow \{\mathit{overflow}\}\ ,\ \mathrm{C} \leftarrow \{\mathit{carry}\}$$

Subtract instruction RTL

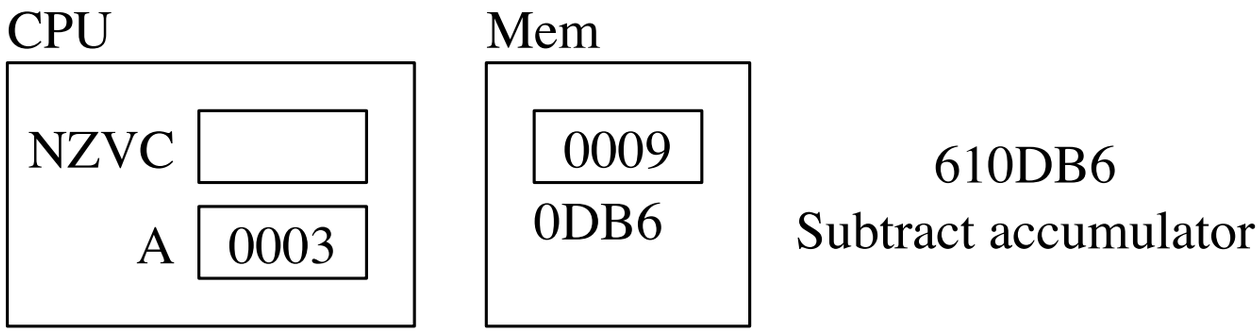
$$\mathrm{r} \leftarrow \mathrm{r} - \mathrm{Oprnd}\ ;\ \mathrm{N} \leftarrow \mathrm{r} < 0\ ,\ \mathrm{Z} \leftarrow \mathrm{r} = 0\ ,\ \mathrm{V} \leftarrow \{\mathit{overflow}\}\ ,\ \mathrm{C} \leftarrow \{\mathit{carry}\}$$



(a) The Add index register instruction.

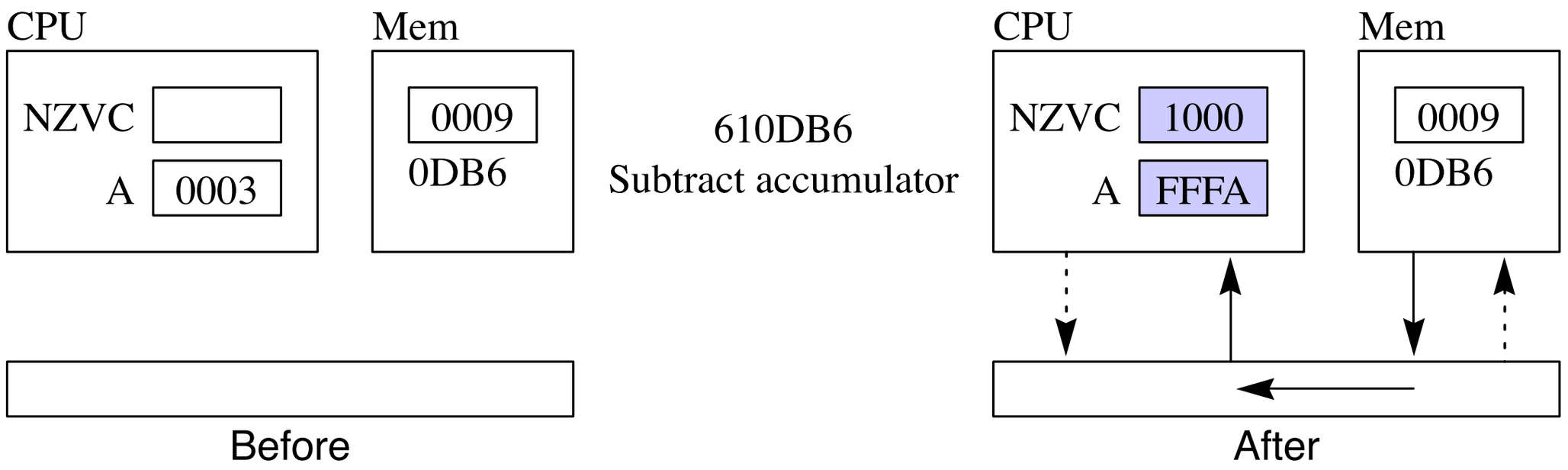


(a) The Add index register instruction.



Before

(b) The Subtract accumulator instruction.



(b) The Subtract accumulator instruction.

Register transfer language

Negate instruction RTL

$$\mathrm{r} \leftarrow -\mathrm{r}\ ;\ \mathrm{N} \leftarrow \mathrm{r} < 0\ ,\ \mathrm{Z} \leftarrow \mathrm{r} = 0\ ,\ \mathrm{V} \leftarrow \{overflow\}\ ,\ \mathrm{C} \leftarrow \{carry\}$$

Register transfer language

Negate instruction RTL

$$\mathrm{r} \leftarrow -\mathrm{r}\ ;\ \mathrm{N} \leftarrow \mathrm{r} < 0\ ,\ \mathrm{Z} \leftarrow \mathrm{r} = 0\ ,\ \mathrm{V} \leftarrow \{\mathit{overflow}\}\ ,\ \mathrm{C} \leftarrow \{\mathit{carry}\}$$

Arithmetic shift left instruction RTL

$$\mathrm{C} \leftarrow \mathrm{r}[0]\ ,\ \mathrm{r}[0:14] \leftarrow \mathrm{r}[1:15]\ ,\ \mathrm{r}[15] \leftarrow 0\ ;\ \mathrm{N} \leftarrow \mathrm{r} < 0\ ,\ \mathrm{Z} \leftarrow \mathrm{r} = 0\ ,$$
$$\mathrm{V} \leftarrow \{\mathit{overflow}\}$$

Register transfer language

Negate instruction RTL

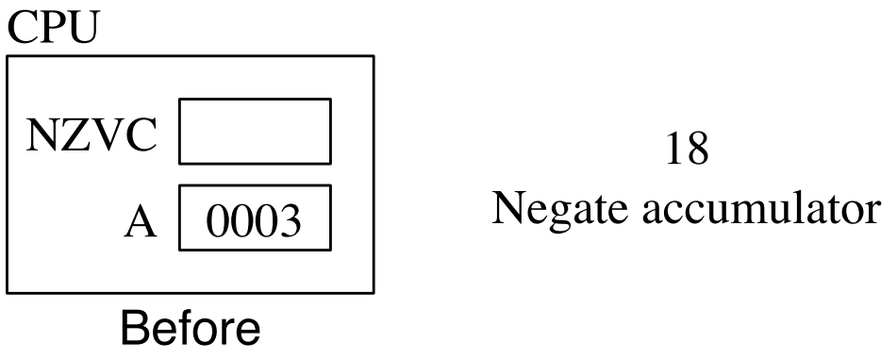
$$\mathrm{r} \leftarrow -\mathrm{r}\ ;\ \mathrm{N} \leftarrow \mathrm{r} < 0\ ,\ \mathrm{Z} \leftarrow \mathrm{r} = 0\ ,\ \mathrm{V} \leftarrow \{\mathit{overflow}\}\ ,\ \mathrm{C} \leftarrow \{\mathit{carry}\}$$

Arithmetic shift left instruction RTL

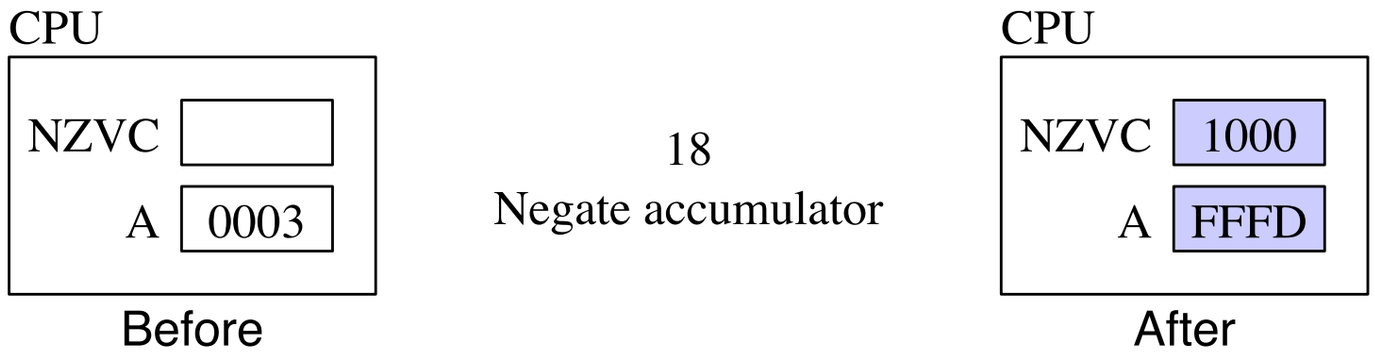
$$\mathrm{C} \leftarrow \mathrm{r}[0]\ ,\ \mathrm{r}[0:14] \leftarrow \mathrm{r}[1:15]\ ,\ \mathrm{r}[15] \leftarrow 0\ ;\ \mathrm{N} \leftarrow \mathrm{r} < 0\ ,\ \mathrm{Z} \leftarrow \mathrm{r} = 0\ ,\ \mathrm{V} \leftarrow \{\mathit{overflow}\}$$

Arithmetic shift right instruction RTL

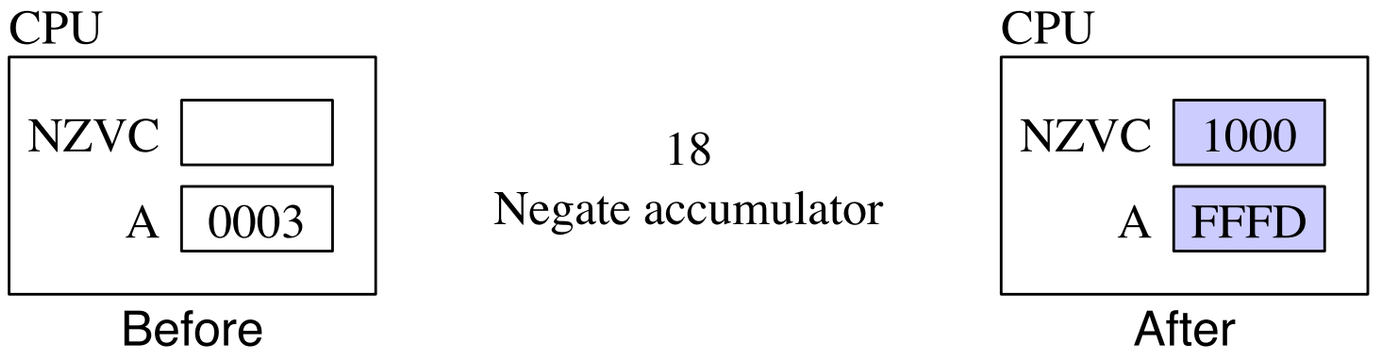
$$\mathrm{C} \leftarrow \mathrm{r}[15]\ ,\ \mathrm{r}[1:15] \leftarrow \mathrm{r}[0:14]\ ;\ \mathrm{N} \leftarrow \mathrm{r} < 0\ ,\ \mathrm{Z} \leftarrow \mathrm{r} = 0\ ,\ \mathrm{V} \leftarrow 0$$



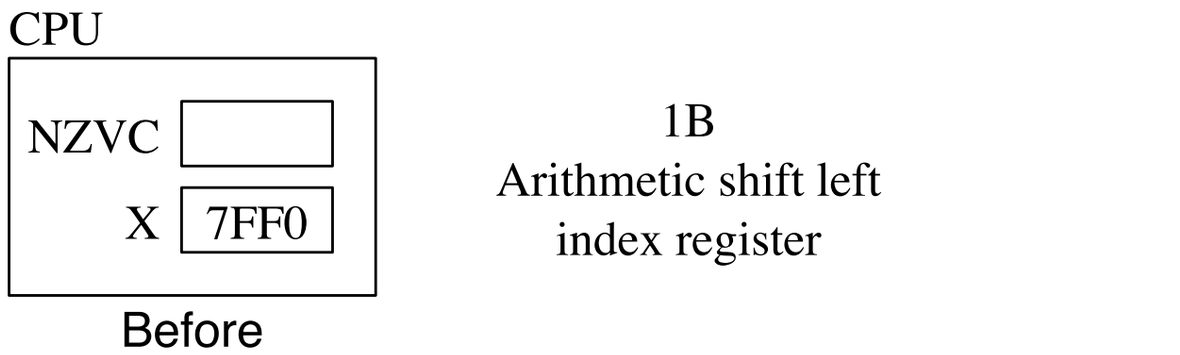
(a) The Negate accumulator instruction.



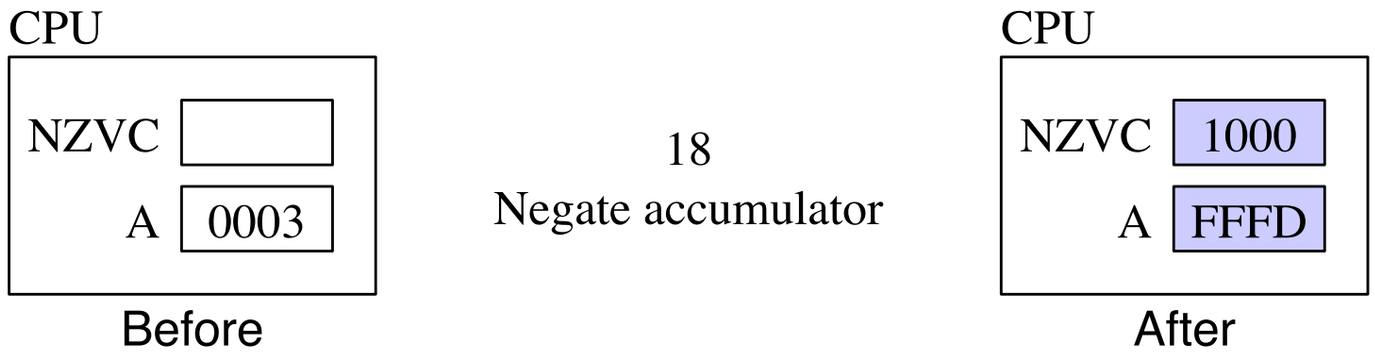
(a) The Negate accumulator instruction.



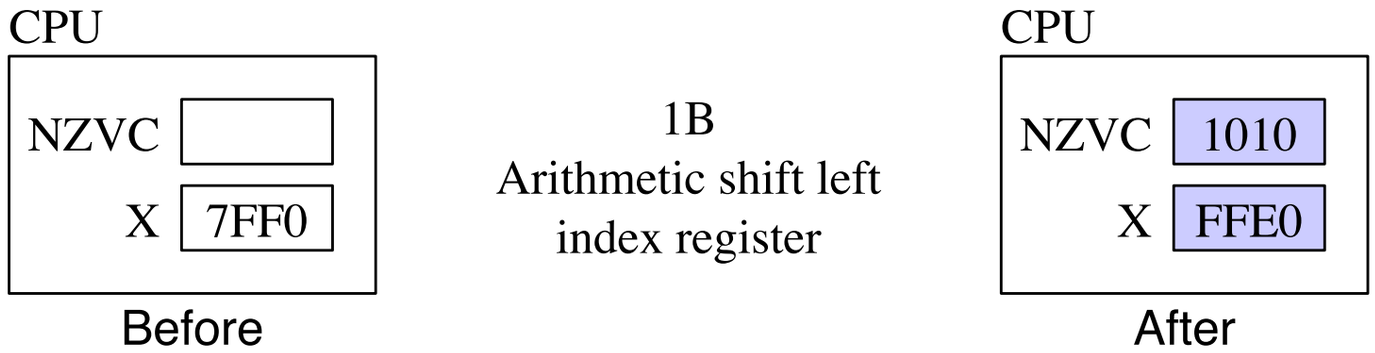
(a) The Negate accumulator instruction.



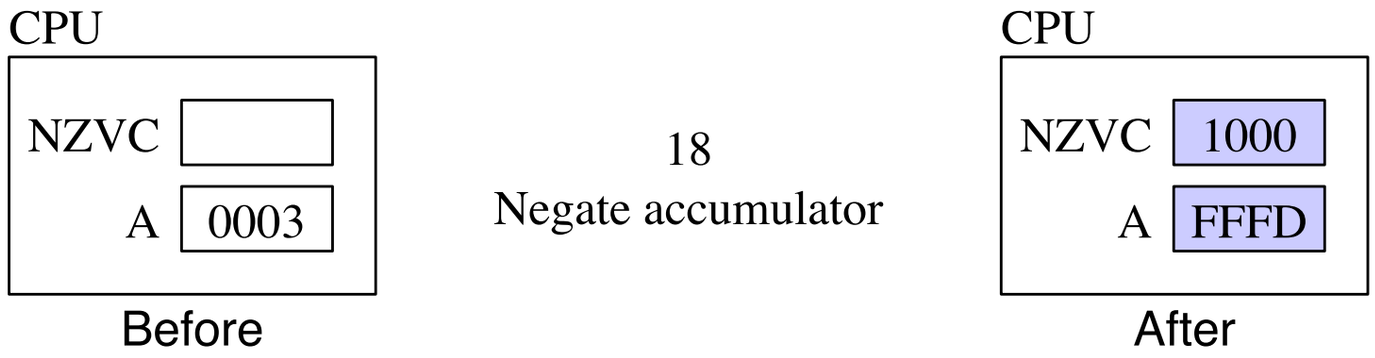
(b) The Arithmetic shift left index register instruction.



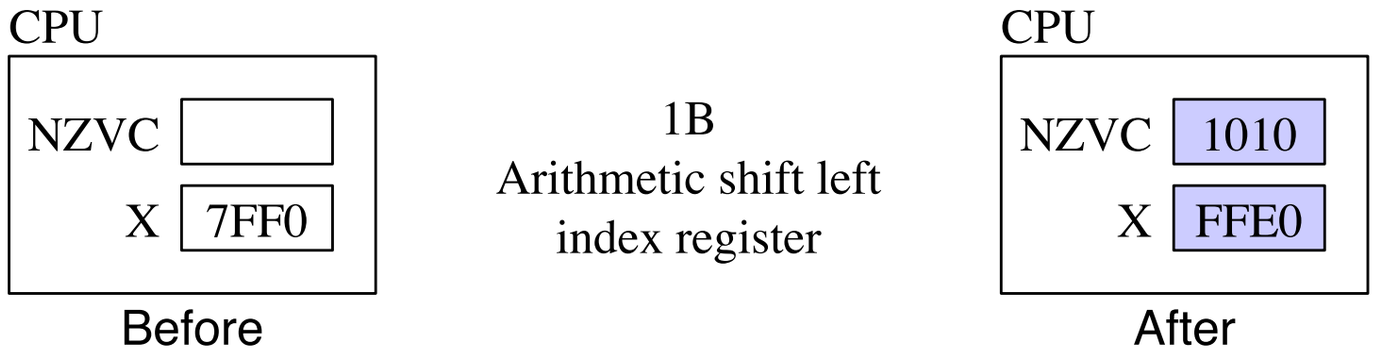
(a) The Negate accumulator instruction.



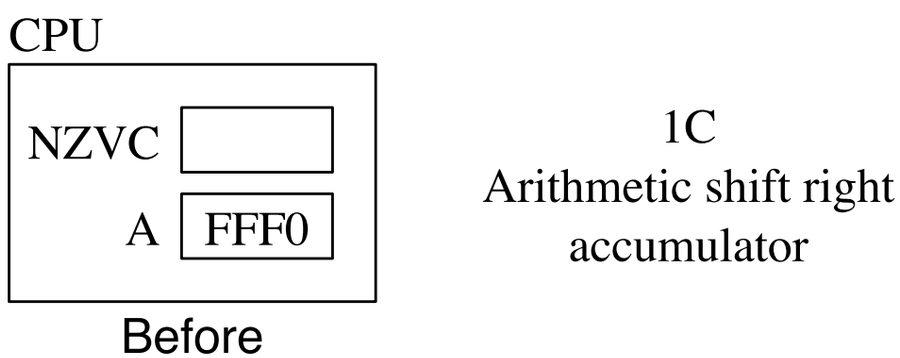
(b) The Arithmetic shift left index register instruction.



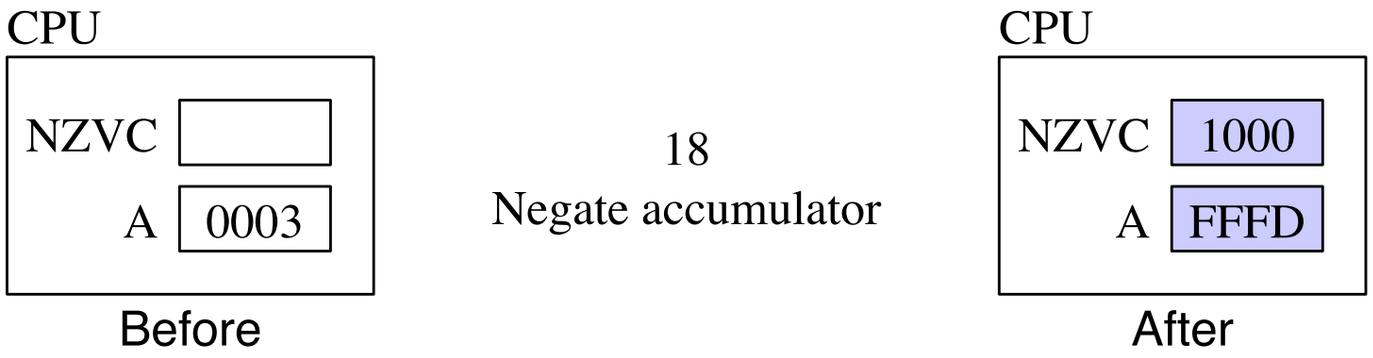
(a) The Negate accumulator instruction.



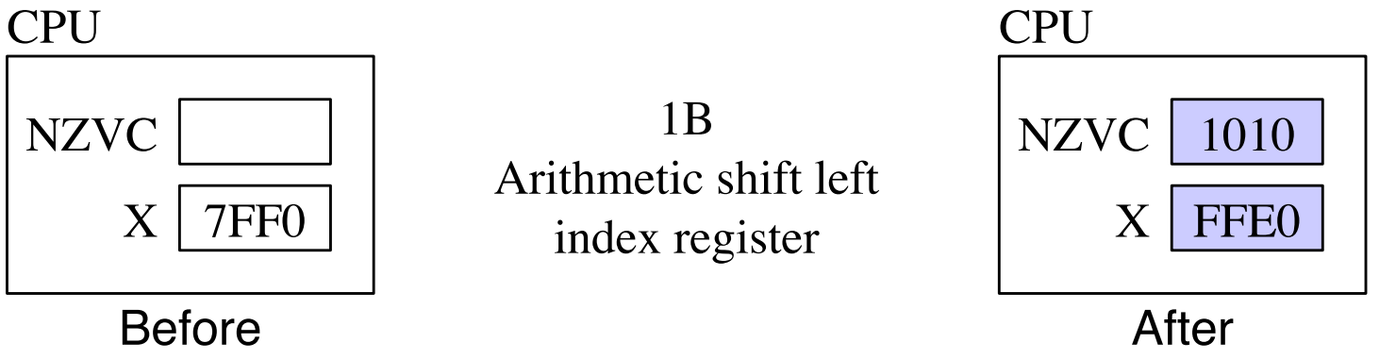
(b) The Arithmetic shift left index register instruction.



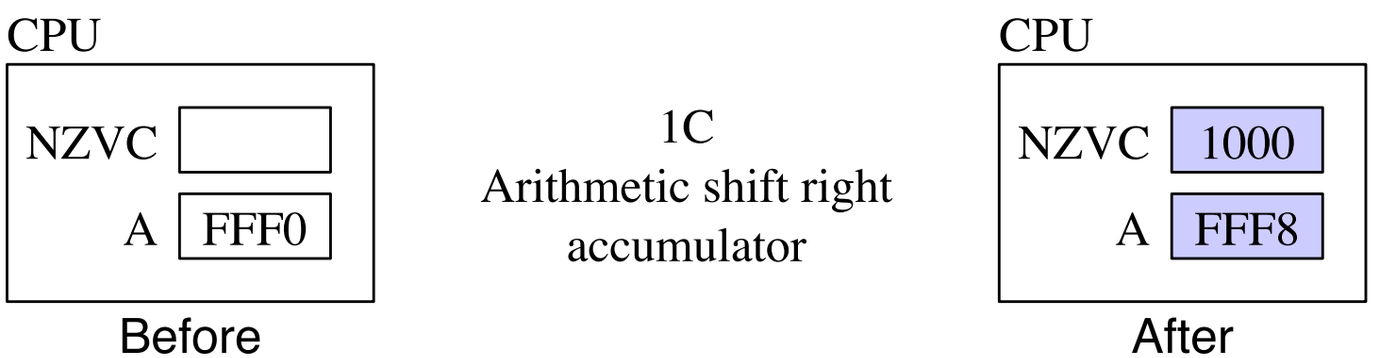
(c) The Arithmetic shift right index register instruction.



(a) The Negate accumulator instruction.



(b) The Arithmetic shift left index register instruction.



(c) The Arithmetic shift right index register instruction.

The logic instructions

- Dyadic logic instructions
  - The And instruction
  - The Or instruction
  - The Exclusive Or instruction

The logic instructions

- Dyadic logic instructions
  - The And instruction
  - The Or instruction
  - The Exclusive Or instruction
- Monadic logic instructions
  - The Bitwise Not instruction
  - The rotate left instruction
  - The rotate right instruction

Register transfer language

Bitwise And instruction RTL

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

Register transfer language

Bitwise And instruction RTL

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

Bitwise Or instruction RTL

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

Register transfer language

Bitwise And instruction RTL

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

Bitwise Or instruction RTL

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

Bitwise Exclusive Or instruction RTL

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

Before	Instruction	After
A: 9DC3 Mem[0DB2]: FF00	710DB2 Bitwise And accumulator	

Before	Instruction	After
A: 9DC3 Mem[0DB2]: FF00	710DB2 Bitwise And accumulator	A: 9D00 NZ: 10 Mem[0DB2]: FF00

Before	Instruction	After
A: 9DC3 Mem[0DB2]: FF00	710DB2 Bitwise And accumulator	A: 9D00 NZ: 10 Mem[0DB2]: FF00
X: A56B Mem[0DB2]: 00FF	890DB2 Bitwise Or index register	

Before	Instruction	After
A: 9DC3 Mem[0DB2]: FF00	710DB2 Bitwise And accumulator	A: 9D00 NZ: 10 Mem[0DB2]: FF00
X: A56B Mem[0DB2]: 00FF	890DB2 Bitwise Or index register	X: A5FF NZ: 10 Mem[0DB2]: 00FF

Before	Instruction	After
A: 9DC3 Mem[0DB2]: FF00	710DB2 Bitwise And accumulator	A: 9D00 NZ: 10 Mem[0DB2]: FF00
X: A56B Mem[0DB2]: 00FF	890DB2 Bitwise Or index register	X: A5FF NZ: 10 Mem[0DB2]: 00FF
A: 83C9 Mem[0DB2]: F00F	910DB2 Bitwise Exclusive Or accumulator	

Before	Instruction	After
A: 9DC3 Mem[0DB2]: FF00	710DB2 Bitwise And accumulator	A: 9D00 NZ: 10 Mem[0DB2]: FF00
X: A56B Mem[0DB2]: 00FF	890DB2 Bitwise Or index register	X: A5FF NZ: 10 Mem[0DB2]: 00FF
A: 83C9 Mem[0DB2]: F00F	910DB2 Bitwise Exclusive Or accumulator	A: 73C6 NZ: 00 Mem[0DB2]: F00F

Register transfer language

Bitwise Not instruction RTL

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

Register transfer language

Bitwise Not instruction RTL

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

Rotate left instruction RTL

$$C \leftarrow r[0] \; , \; r[0:14] \leftarrow r[1:15] \; , \; r[15] \leftarrow C \; ; \; N \leftarrow r < 0 \; , \; Z \leftarrow r = 0$$

Register transfer language

Bitwise Not instruction RTL

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

Rotate left instruction RTL

$$C \leftarrow r[0] \; , \; r[0:14] \leftarrow r[1:15] \; , \; r[15] \leftarrow C \; ; \; N \leftarrow r < 0 \; , \; Z \leftarrow r = 0$$

Rotate right instruction RTL

$$C \leftarrow r[15] \; , \; r[1:15] \leftarrow r[0:14] \; , \; r[0] \leftarrow C \; ; \; N \leftarrow r < 0 \; , \; Z \leftarrow r = 0$$

Before	Instruction	After
X: 103F	1F Bitwise Not index register	

Before	Instruction	After
X: 103F	1F Bitwise Not index register	X: EFC0 NZ: 10

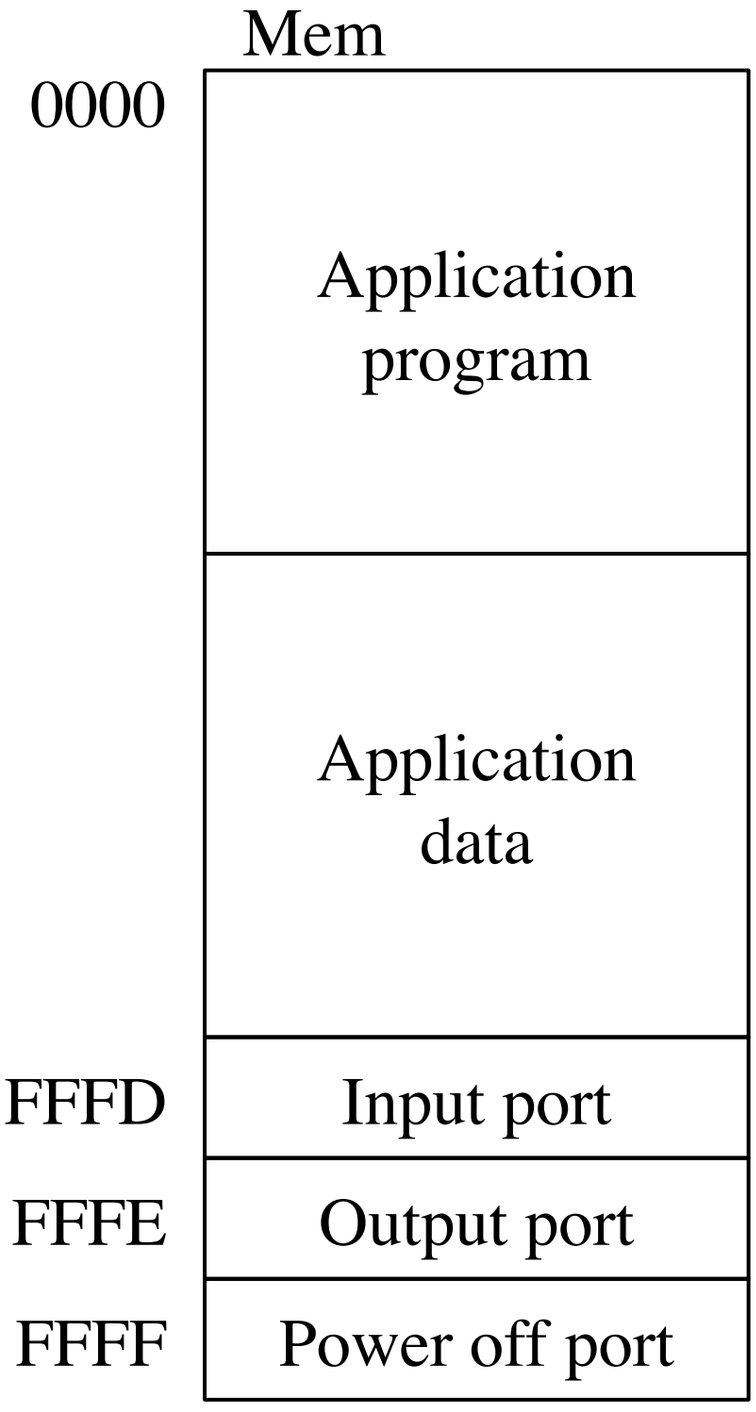
Before	Instruction	After
X: 103F	1F Bitwise Not index register	X: EFC0 NZ: 10
A: 7F69 C:1	20 Rotate left accumulator	

Before	Instruction	After
X: 103F	1F Bitwise Not index register	X: EFC0 NZ: 10
A: 7F69 C:1	20 Rotate left accumulator	A: FED3 NZC: 100

Before	Instruction	After
X: 103F	1F Bitwise Not index register	X: EFC0 NZ: 10
A: 7F69 C:1	20 Rotate left accumulator	A: FED3 NZC: 100
A: 7F69 C:1	22 Rotate right accumulator	

Before	Instruction	After
X: 103F	1F Bitwise Not index register	X: EFC0 NZ: 10
A: 7F69 C:1	20 Rotate left accumulator	A: FED3 NZC: 100
A: 7F69 C:1	22 Rotate right accumulator	A: BFB4 NZC: 001

Accessing the memory-mapped ports



Accessing the memory-mapped ports

Before	Instruction	After
A: FFFF Input: `Hello World!`	D1FFFD Load byte accumulator	

Accessing the memory-mapped ports

Before	Instruction	After
A: FFFF Input: `Hello World!`	D1FFFD Load byte accumulator	A: 0048 NZ: 00 Input: `ello World!`

Accessing the memory-mapped ports

Before	Instruction	After
A: FFFF Input: `Hello World!`	D1FFFD Load byte accumulator	A: 0048 NZ: 00 Input: `ello World!`
A: 0021 Output: `Hello World`	F1FFFE Store byte accumulator	

Accessing the memory-mapped ports

Before	Instruction	After
A: FFFF Input: `Hello World!`	D1FFFD Load byte accumulator	A: 0048 NZ: 00 Input: `ello World!`
A: 0021 Output: `Hello World`	F1FFFE Store byte accumulator	A: 0021 Output: `Hello World!`

Accessing the memory-mapped ports

Before	Instruction	After
A: FFFF Input: `Hello World!`	D1FFFD Load byte accumulator	A: 0048 NZ: 00 Input: `ello World!`
A: 0021 Output: `Hello World`	F1FFFE Store byte accumulator	A: 0021 Output: `Hello World!`
A: 0000	F1FFFF Store byte accumulator	

Accessing the memory-mapped ports

Before	Instruction	After
A: FFFF Input: `Hello World!`	D1FFFD Load byte accumulator	A: 0048 NZ: 00 Input: `ello World!`
A: 0021 Output: `Hello World`	F1FFFE Store byte accumulator	A: 0021 Output: `Hello World!`
A: 0000	F1FFFF Store byte accumulator	Shut down

Address Machine Language (bin)

```
0000    1101 0001 0000 0000 0000 1111
0003    1111 0001 1111 1111 1111 1110
0006    1101 0001 0000 0000 0001 0000
0009    1111 0001 1111 1111 1111 1110
000C    1111 0001 1111 1111 1111 1111
000F    0100 1000 0110 1001
```

Address Machine Language (hex)

```
0000    D1000F ;Load byte accumulator 'H'
0003    F1FFFE ;Store byte accumulator output port
0006    D10010 ;Load byte accumulator 'i'
0009    F1FFFE ;Store byte accumulator output port
000C    F1FFFF ;Store byte accumulator power off port
000F    4869   ;ASCII "Hi" characters
```

Output

```
Hi
```

Address	Machine Language (hex)
0000	D1000F ;Load byte accumulator 'H'
0003	F1FFFE ;Store byte accumulator output port
0006	D10010 ;Load byte accumulator 'i'
0009	F1FFFE ;Store byte accumulator output port
000C	F1FFFF ;Store byte accumulator power off port
000F	4869 ;ASCII "Hi" characters

Cycle	Instruction	State	Output
0	Initial state	A:	
1	Mem[0000]: D1000F		
2	Mem[0003]: F1FFFE		
3	Mem[0006]: D10010		
4	Mem[0009]: F1FFFE		
5	Mem[000C]: F1FFFF		

Address	Machine Language (hex)
0000	D1000F ;Load byte accumulator 'H'
0003	F1FFFE ;Store byte accumulator output port
0006	D10010 ;Load byte accumulator 'i'
0009	F1FFFE ;Store byte accumulator output port
000C	F1FFFF ;Store byte accumulator power off port
000F	4869 ;ASCII "Hi" characters

Cycle	Instruction	State	Output
0	Initial state	A:	
1	Mem[0000]: D1000F	A: 0048	
2	Mem[0003]: F1FFFE		
3	Mem[0006]: D10010		
4	Mem[0009]: F1FFFE		
5	Mem[000C]: F1FFFF		

Address	Machine Language (hex)
0000	D1000F ;Load byte accumulator 'H'
0003	F1FFFE ;Store byte accumulator output port
0006	D10010 ;Load byte accumulator 'i'
0009	F1FFFE ;Store byte accumulator output port
000C	F1FFFF ;Store byte accumulator power off port
000F	4869 ;ASCII "Hi" characters

Cycle	Instruction	State	Output
0	Initial state	A:	
1	Mem[0000]: D1000F	A: 0048	
2	Mem[0003]: F1FFFE	A: 0048	H
3	Mem[0006]: D10010		
4	Mem[0009]: F1FFFE		
5	Mem[000C]: F1FFFF		

```
Address     Machine Language (hex)
0000        D1000F ;Load byte accumulator 'H'
0003        F1FFFE ;Store byte accumulator output port
0006        D10010 ;Load byte accumulator 'i'
0009        F1FFFE ;Store byte accumulator output port
000C        F1FFFF ;Store byte accumulator power off port
000F        4869   ;ASCII "Hi" characters
```

Cycle	Instruction	State	Output
0	Initial state	A:	
1	Mem[0000]: D1000F	A: 0048	
2	Mem[0003]: F1FFFE	A: 0048	H
3	Mem[0006]: D10010	A: 0069	H
4	Mem[0009]: F1FFFE		
5	Mem[000C]: F1FFFF		

Address	Machine Language (hex)
0000	D1000F ;Load byte accumulator 'H'
0003	F1FFFE ;Store byte accumulator output port
0006	D10010 ;Load byte accumulator 'i'
0009	F1FFFE ;Store byte accumulator output port
000C	F1FFFF ;Store byte accumulator power off port
000F	4869 ;ASCII "Hi" characters

Cycle	Instruction	State	Output
0	Initial state	A:	
1	Mem[0000]: D1000F	A: 0048	
2	Mem[0003]: F1FFFE	A: 0048	H
3	Mem[0006]: D10010	A: 0069	H
4	Mem[0009]: F1FFFE	A: 0069	Hi
5	Mem[000C]: F1FFFF		

Address	Machine Language (hex)
0000	D1000F ;Load byte accumulator 'H'
0003	F1FFFE ;Store byte accumulator output port
0006	D10010 ;Load byte accumulator 'i'
0009	F1FFFE ;Store byte accumulator output port
000C	F1FFFF ;Store byte accumulator power off port
000F	4869 ;ASCII "Hi" characters

Cycle	Instruction	State	Output
0	Initial state	A:	
1	Mem[0000]: D1000F	A: 0048	
2	Mem[0003]: F1FFFE	A: 0048	H
3	Mem[0006]: D10010	A: 0069	H
4	Mem[0009]: F1FFFE	A: 0069	Hi
5	Mem[000C]: F1FFFF	Shut down	

Demo Pepp IDE

The program execution process

```
Load program into memory at Mem[0000]
PC ← 0000
do
        Fetch:  IR[0:7] ← Mem[PC]
        Decode:  Decode instruction specifier IR[0:7]
        Increment:  PC ← PC + 1
        if  IR[0:7] is a dyadic instruction
                IR[8:23] ← Mem[PC]
                PC ← PC + 2
        Execute:  Execute the instruction in the IR
while  not shut down  &&  instruction in IR is legal
```

The program execution process

Initialization

Load program into memory at Mem[0000]
PC ← 0000
`do`
    Fetch: IR[0:7] ← Mem[PC]
    Decode: Decode instruction specifier IR[0:7]
    Increment: PC ← PC + 1
    `if` IR[0:7] is a dyadic instruction
        IR[8:23] ← Mem[PC]
        PC ← PC + 2
    Execute: Execute the instruction in the IR
`while` not shut down `&&` instruction in IR is legal

The program execution process

Initialization

Load program into memory at Mem[0000]

PC ← 0000

von Neumann execution cycle

```
do
    Fetch:  IR[0:7] ← Mem[PC]
    Decode:  Decode instruction specifier IR[0:7]
    Increment:  PC ← PC + 1
    if  IR[0:7] is a dyadic instruction
        IR[8:23] ← Mem[PC]
        PC ← PC + 2
    Execute:  Execute the instruction in the IR
while not shut down  &&  instruction in IR is legal
```

Simplified summary of the von Neumann execution cycle

- Fetch the instruction at Mem[PC] into the IR.
- Decode the instruction in the IR.
- Increment the PC.
- Execute the instruction in the IR.
- Repeat the cycle.

Cycle	State	Output
0	A: PC: 0000 IR:	
1-fetch		
1-increment		
1-execute		
2-fetch		
2-increment		
2-execute		
3-fetch		
3-increment		
3-execute		
4-fetch		
4-increment		
4-execute		
5-fetch		
5-increment		
5-execute		

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment				
1-execute				
2-fetch				
2-increment				
2-execute				
3-fetch				
3-increment				
3-execute				
4-fetch				
4-increment				
4-execute				
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute				
2-fetch				
2-increment				
2-execute				
3-fetch				
3-increment				
3-execute				
4-fetch				
4-increment				
4-execute				
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch				
2-increment				
2-execute				
3-fetch				
3-increment				
3-execute				
4-fetch				
4-increment				
4-execute				
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment				
2-execute				
3-fetch				
3-increment				
3-execute				
4-fetch				
4-increment				
4-execute				
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment	A: 0048	PC: 0006	IR: F1FFFE	
2-execute				
3-fetch				
3-increment				
3-execute				
4-fetch				
4-increment				
4-execute				
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment	A: 0048	PC: 0006	IR: F1FFFE	
2-execute	A: 0048	PC: 0006	IR: F1FFFE	H
3-fetch				
3-increment				
3-execute				
4-fetch				
4-increment				
4-execute				
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment	A: 0048	PC: 0006	IR: F1FFFE	
2-execute	A: 0048	PC: 0006	IR: F1FFFE	H
3-fetch	A: 0048	PC: 0006	IR: D10010	H
3-increment				
3-execute				
4-fetch				
4-increment				
4-execute				
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment	A: 0048	PC: 0006	IR: F1FFFE	
2-execute	A: 0048	PC: 0006	IR: F1FFFE	`H`
3-fetch	A: 0048	PC: 0006	IR: D10010	`H`
3-increment	A: 0048	PC: 0009	IR: D10010	`H`
3-execute				
4-fetch				
4-increment				
4-execute				
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment	A: 0048	PC: 0006	IR: F1FFFE	
2-execute	A: 0048	PC: 0006	IR: F1FFFE	`H`
3-fetch	A: 0048	PC: 0006	IR: D10010	`H`
3-increment	A: 0048	PC: 0009	IR: D10010	`H`
3-execute	A: 0069	PC: 0009	IR: D10010	`H`
4-fetch				
4-increment				
4-execute				
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment	A: 0048	PC: 0006	IR: F1FFFE	
2-execute	A: 0048	PC: 0006	IR: F1FFFE	`H`
3-fetch	A: 0048	PC: 0006	IR: D10010	`H`
3-increment	A: 0048	PC: 0009	IR: D10010	`H`
3-execute	A: 0069	PC: 0009	IR: D10010	`H`
4-fetch	A: 0069	PC: 0009	IR: F1FFFE	`H`
4-increment				
4-execute				
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment	A: 0048	PC: 0006	IR: F1FFFE	
2-execute	A: 0048	PC: 0006	IR: F1FFFE	H
3-fetch	A: 0048	PC: 0006	IR: D10010	H
3-increment	A: 0048	PC: 0009	IR: D10010	H
3-execute	A: 0069	PC: 0009	IR: D10010	H
4-fetch	A: 0069	PC: 0009	IR: F1FFFE	H
4-increment	A: 0069	PC: 000C	IR: F1FFFE	H
4-execute				
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment	A: 0048	PC: 0006	IR: F1FFFE	
2-execute	A: 0048	PC: 0006	IR: F1FFFE	`H`
3-fetch	A: 0048	PC: 0006	IR: D10010	`H`
3-increment	A: 0048	PC: 0009	IR: D10010	`H`
3-execute	A: 0069	PC: 0009	IR: D10010	`H`
4-fetch	A: 0069	PC: 0009	IR: F1FFFE	`H`
4-increment	A: 0069	PC: 000C	IR: F1FFFE	`H`
4-execute	A: 0069	PC: 000C	IR: F1FFFE	`Hi`
5-fetch				
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment	A: 0048	PC: 0006	IR: F1FFFE	
2-execute	A: 0048	PC: 0006	IR: F1FFFE	`H`
3-fetch	A: 0048	PC: 0006	IR: D10010	`H`
3-increment	A: 0048	PC: 0009	IR: D10010	`H`
3-execute	A: 0069	PC: 0009	IR: D10010	`H`
4-fetch	A: 0069	PC: 0009	IR: F1FFFE	`H`
4-increment	A: 0069	PC: 000C	IR: F1FFFE	`H`
4-execute	A: 0069	PC: 000C	IR: F1FFFE	`Hi`
5-fetch	A: 0069	PC: 000C	IR: F1FFFF	`Hi`
5-increment				
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment	A: 0048	PC: 0006	IR: F1FFFE	
2-execute	A: 0048	PC: 0006	IR: F1FFFE	H
3-fetch	A: 0048	PC: 0006	IR: D10010	H
3-increment	A: 0048	PC: 0009	IR: D10010	H
3-execute	A: 0069	PC: 0009	IR: D10010	H
4-fetch	A: 0069	PC: 0009	IR: F1FFFE	H
4-increment	A: 0069	PC: 000C	IR: F1FFFE	H
4-execute	A: 0069	PC: 000C	IR: F1FFFE	Hi
5-fetch	A: 0069	PC: 000C	IR: F1FFFF	Hi
5-increment	A: 0069	PC: 000F	IR: F1FFFF	Hi
5-execute				

Cycle	State			Output
0	A:	PC: 0000	IR:	
1-fetch	A:	PC: 0000	IR: D1000F	
1-increment	A:	PC: 0003	IR: D1000F	
1-execute	A: 0048	PC: 0003	IR: D1000F	
2-fetch	A: 0048	PC: 0003	IR: F1FFFE	
2-increment	A: 0048	PC: 0006	IR: F1FFFE	
2-execute	A: 0048	PC: 0006	IR: F1FFFE	`H`
3-fetch	A: 0048	PC: 0006	IR: D10010	`H`
3-increment	A: 0048	PC: 0009	IR: D10010	`H`
3-execute	A: 0069	PC: 0009	IR: D10010	`H`
4-fetch	A: 0069	PC: 0009	IR: F1FFFE	`H`
4-increment	A: 0069	PC: 000C	IR: F1FFFE	`H`
4-execute	A: 0069	PC: 000C	IR: F1FFFE	`Hi`
5-fetch	A: 0069	PC: 000C	IR: F1FFFF	`Hi`
5-increment	A: 0069	PC: 000F	IR: F1FFFF	`Hi`
5-execute	Shut down			

Address	Machine Language (bin)
0000	1101 0001 1111 1111 1111 1101
0003	1111 0001 0000 0000 0001 0101
0006	1101 0001 1111 1111 1111 1101
0009	1111 0001 1111 1111 1111 1110
000C	1101 0001 0000 0000 0001 0101
000F	1111 0001 1111 1111 1111 1110
0012	1111 0001 1111 1111 1111 1111
0015	0000 0000

Address	Machine Language (hex)
0000	D1FFFD ;Load byte first char from input port
0003	F10015 ;Store byte first char to 0015
0006	D1FFFD ;Load byte from input port
0009	F1FFFE ;Store byte to output port
000C	D10015 ;Load byte first char from 0015
000F	F1FFFE ;Store byte first char to output port
0012	F1FFFF ;Store byte to power off port
0015	00 ;One byte storage for first char

Input

up

Output

pu

Address	Machine Language (hex)	
0000	D1FFFD	;Load byte first char from input port
0003	F10015	;Store byte first char to 0015
0006	D1FFFD	;Load byte from input port
0009	F1FFFE	;Store byte to output port
000C	D10015	;Load byte first char from 0015
000F	F1FFFE	;Store byte first char to output port
0012	F1FFFF	;Store byte to power off port
0015	00	;One byte storage for first char

Cycle	Instruction	State		Input	Output
0	Initial state	A:	Mem[0015]:	up	
1	Mem[0000]: D1FFFD				
2	Mem[0003]: F10015				
3	Mem[0006]: D1FFFD				
4	Mem[0009]: F1FFFE				
5	Mem[000C]: D10015				
6	Mem[000F]: F1FFFE				
7	Mem[0012]: F1FFFF				

Address	Machine Language (hex)	
0000	D1FFFD	;Load byte first char from input port
0003	F10015	;Store byte first char to 0015
0006	D1FFFD	;Load byte from input port
0009	F1FFFE	;Store byte to output port
000C	D10015	;Load byte first char from 0015
000F	F1FFFE	;Store byte first char to output port
0012	F1FFFF	;Store byte to power off port
0015	00	;One byte storage for first char

Cycle	Instruction	State		Input	Output
0	Initial state	A:	Mem[0015]:	up	
1	Mem[0000]: D1FFFD	A: 0075	Mem[0015]:	p	
2	Mem[0003]: F10015				
3	Mem[0006]: D1FFFD				
4	Mem[0009]: F1FFFE				
5	Mem[000C]: D10015				
6	Mem[000F]: F1FFFE				
7	Mem[0012]: F1FFFF				

Address	Machine Language (hex)	
0000	D1FFFD	;Load byte first char from input port
0003	F10015	;Store byte first char to 0015
0006	D1FFFD	;Load byte from input port
0009	F1FFFE	;Store byte to output port
000C	D10015	;Load byte first char from 0015
000F	F1FFFE	;Store byte first char to output port
0012	F1FFFF	;Store byte to power off port
0015	00	;One byte storage for first char

Cycle	Instruction	State		Input	Output
0	Initial state	A:	Mem[0015]:	up	
1	Mem[0000]: D1FFFD	A: 0075	Mem[0015]:	p	
2	Mem[0003]: F10015	A: 0075	Mem[0015]: 75	p	
3	Mem[0006]: D1FFFD				
4	Mem[0009]: F1FFFE				
5	Mem[000C]: D10015				
6	Mem[000F]: F1FFFE				
7	Mem[0012]: F1FFFF				

Address	Machine Language (hex)	
0000	D1FFFD	;Load byte first char from input port
0003	F10015	;Store byte first char to 0015
0006	D1FFFD	;Load byte from input port
0009	F1FFFE	;Store byte to output port
000C	D10015	;Load byte first char from 0015
000F	F1FFFE	;Store byte first char to output port
0012	F1FFFF	;Store byte to power off port
0015	00	;One byte storage for first char

Cycle	Instruction	State		Input	Output
0	Initial state	A:	Mem[0015]:	up	
1	Mem[0000]: D1FFFD	A: 0075	Mem[0015]:	p	
2	Mem[0003]: F10015	A: 0075	Mem[0015]: 75	p	
3	Mem[0006]: D1FFFD	A: 0070	Mem[0015]: 75		
4	Mem[0009]: F1FFFE				
5	Mem[000C]: D10015				
6	Mem[000F]: F1FFFE				
7	Mem[0012]: F1FFFF				

```
Address     Machine Language (hex)
0000        D1FFFD ;Load byte first char from input port
0003        F10015 ;Store byte first char to 0015
0006        D1FFFD ;Load byte from input port
0009        F1FFFE ;Store byte to output port
000C        D10015 ;Load byte first char from 0015
000F        F1FFFE ;Store byte first char to output port
0012        F1FFFF ;Store byte to power off port
0015        00     ;One byte storage for first char
```

Cycle	Instruction	State		Input	Output
0	Initial state	A:	Mem[0015]:	up	
1	Mem[0000]: D1FFFD	A: 0075	Mem[0015]:	p	
2	Mem[0003]: F10015	A: 0075	Mem[0015]: 75	p	
3	Mem[0006]: D1FFFD	A: 0070	Mem[0015]: 75		
4	Mem[0009]: F1FFFE	A: 0070	Mem[0015]: 75		p
5	Mem[000C]: D10015				
6	Mem[000F]: F1FFFE				
7	Mem[0012]: F1FFFF				

Address	Machine Language (hex)	
0000	D1FFFD	;Load byte first char from input port
0003	F10015	;Store byte first char to 0015
0006	D1FFFD	;Load byte from input port
0009	F1FFFE	;Store byte to output port
000C	D10015	;Load byte first char from 0015
000F	F1FFFE	;Store byte first char to output port
0012	F1FFFF	;Store byte to power off port
0015	00	;One byte storage for first char

Cycle	Instruction	State		Input	Output
0	Initial state	A:	Mem[0015]:	up	
1	Mem[0000]: D1FFFD	A: 0075	Mem[0015]:	p	
2	Mem[0003]: F10015	A: 0075	Mem[0015]: 75	p	
3	Mem[0006]: D1FFFD	A: 0070	Mem[0015]: 75		
4	Mem[0009]: F1FFFE	A: 0070	Mem[0015]: 75		p
5	Mem[000C]: D10015	A: 0075	Mem[0015]: 75		p
6	Mem[000F]: F1FFFE				
7	Mem[0012]: F1FFFF				

Address	Machine Language (hex)	
0000	D1FFFD	;Load byte first char from input port
0003	F10015	;Store byte first char to 0015
0006	D1FFFD	;Load byte from input port
0009	F1FFFE	;Store byte to output port
000C	D10015	;Load byte first char from 0015
000F	F1FFFE	;Store byte first char to output port
0012	F1FFFF	;Store byte to power off port
0015	00	;One byte storage for first char

Cycle	Instruction	State		Input	Output
0	Initial state	A:	Mem[0015]:	up	
1	Mem[0000]: D1FFFD	A: 0075	Mem[0015]:	p	
2	Mem[0003]: F10015	A: 0075	Mem[0015]: 75	p	
3	Mem[0006]: D1FFFD	A: 0070	Mem[0015]: 75		
4	Mem[0009]: F1FFFE	A: 0070	Mem[0015]: 75		p
5	Mem[000C]: D10015	A: 0075	Mem[0015]: 75		p
6	Mem[000F]: F1FFFE	A: 0075	Mem[0015]: 75		pu
7	Mem[0012]: F1FFFF				

```
Address     Machine Language (hex)
0000        D1FFFD ;Load byte first char from input port
0003        F10015 ;Store byte first char to 0015
0006        D1FFFD ;Load byte from input port
0009        F1FFFE ;Store byte to output port
000C        D10015 ;Load byte first char from 0015
000F        F1FFFE ;Store byte first char to output port
0012        F1FFFF ;Store byte to power off port
0015        00     ;One byte storage for first char
```

Cycle	Instruction	State		Input	Output
0	Initial state	A:	Mem[0015]:	up	
1	Mem[0000]: D1FFFD	A: 0075	Mem[0015]:	p	
2	Mem[0003]: F10015	A: 0075	Mem[0015]: 75	p	
3	Mem[0006]: D1FFFD	A: 0070	Mem[0015]: 75		
4	Mem[0009]: F1FFFE	A: 0070	Mem[0015]: 75		p
5	Mem[000C]: D10015	A: 0075	Mem[0015]: 75		p
6	Mem[000F]: F1FFFE	A: 0075	Mem[0015]: 75		pu
7	Mem[0012]: F1FFFF	Shut down			

```
Address     Machine Language (hex)
0000        C1000F ;Load word accumulator 0005 from Mem[000F]
0003        510011 ;Add accumulator 0003 from Mem[0011]
0006        810013 ;Or accumulator 0030 from Mem[0013]
0009        F1FFFE ;Store byte to output port
000C        F1FFFF ;Store byte to power off port
000F        0005   ;Decimal 5
0011        0003   ;Decimal 3
0013        0030   ;Mask for ASCII char
```

Output

8

```
Address     Machine Language (hex)
0000        C1000F ;Load word accumulator 0005 from Mem[000F]
0003        510011 ;Add accumulator 0003 from Mem[0011]
0006        810013 ;Or accumulator 0030 from Mem[0013]
0009        F1FFFE ;Store byte to output port
000C        F1FFFF ;Store byte to power off port
000F        0005   ;Decimal 5
0011        0003   ;Decimal 3
0013        0030   ;Mask for ASCII char
```

Output

```
A:  0000 0000 0000 0101
```

```
Address     Machine Language (hex)
0000        C1000F ;Load word accumulator 0005 from Mem[000F]
0003        510011 ;Add accumulator 0003 from Mem[0011]
0006        810013 ;Or accumulator 0030 from Mem[0013]
0009        F1FFFE ;Store byte to output port
000C        F1FFFF ;Store byte to power off port
000F        0005   ;Decimal 5
0011        0003   ;Decimal 3
0013        0030   ;Mask for ASCII char
```

Output

8

```
A:   0000 0000 0000 0101

     0000 0000 0000 0101
ADD  0000 0000 0000 0011
A:   0000 0000 0000 1000
```

```
Address     Machine Language (hex)
0000        C1000F ;Load word accumulator 0005 from Mem[000F]
0003        510011 ;Add accumulator 0003 from Mem[0011]
0006        810013 ;Or accumulator 0030 from Mem[0013]
0009        F1FFFE ;Store byte to output port
000C        F1FFFF ;Store byte to power off port
000F        0005   ;Decimal 5
0011        0003   ;Decimal 3
0013        0030   ;Mask for ASCII char
```

Output

8

```
A:   0000 0000 0000 0101

     0000 0000 0000 0101
ADD  0000 0000 0000 0011
A:   0000 0000 0000 1000

     0000 0000 0000 1000
OR   0000 0000 0011 0000
A:   0000 0000 0011 1000
```

```
Address     Machine Language (hex)
0000        C1000F ;Load word accumulator 0005 from Mem[000F]
0003        510011 ;Add accumulator 0003 from Mem[0011]
0006        810013 ;Or accumulator 0030 from Mem[0013]
0009        F1FFFE ;Store byte to output port
000C        F1FFFF ;Store byte to power off port
000F        0005   ;Decimal 5
0011        0003   ;Decimal 3
0013        0030   ;Mask for ASCII char
```

Output

8

```
A:   0000 0000 0000 0101

     0000 0000 0000 0101
ADD  0000 0000 0000 0011
A:   0000 0000 0000 1000

     0000 0000 0000 1000
OR   0000 0000 0011 0000
A:   0000 0000 0011 1000
```

Address	Machine Language (hex)	
0000	D1FFFD	;Load byte accumulator from input port
0003	E10018	;Store word accumulator to Mem[0018]
0006	D1FFFD	;Load byte accumulator from input port
0009	510018	;Add accumulator from Mem[0018]
000C	71001A	;And accumulator from Mem[001A]
000F	81001C	;Or accumulator from Mem[001C]
0012	F1FFFE	;Store byte to output port
0015	F1FFFF	;Store byte to power off port
0018	0000	;One-word storage for first number
001A	000F	;AND mask
001C	0030	;OR mask

Input

```
25
```

Output

```
7
```

```
Address     Machine Language (hex)
0000        D1FFFD ;Load byte accumulator from input port
0003        E10018 ;Store word accumulator to Mem[0018]
0006        D1FFFD ;Load byte accumulator from input port
0009        510018 ;Add accumulator from Mem[0018]
000C        71001A ;And accumulator from Mem[001A]
000F        81001C ;Or accumulator from Mem[001C]
0012        F1FFFE ;Store byte to output port
0015        F1FFFF ;Store byte to power off port
0018        0000   ;One-word storage for first number
001A        000F   ;AND mask
001C        0030   ;OR mask
```

Input

```
25
```

```
                    A:
Mem[0018]: 0000 0000 0000 0000
```

Output

```
7
```

```
Address     Machine Language (hex)
0000        D1FFFD ;Load byte accumulator from input port
0003        E10018 ;Store word accumulator to Mem[0018]
0006        D1FFFD ;Load byte accumulator from input port
0009        510018 ;Add accumulator from Mem[0018]
000C        71001A ;And accumulator from Mem[001A]
000F        81001C ;Or accumulator from Mem[001C]
0012        F1FFFE ;Store byte to output port
0015        F1FFFF ;Store byte to power off port
0018        0000   ;One-word storage for first number
001A        000F   ;AND mask
001C        0030   ;OR mask
```

Input

```
25
```

```
            A: 0000 0000 0011 0010
Mem[0018]: 0000 0000 0000 0000
```

Output

```
7
```

Address	Machine Language (hex)	
0000	D1FFFD	;Load byte accumulator from input port
0003	E10018	;Store word accumulator to Mem[0018]
0006	D1FFFD	;Load byte accumulator from input port
0009	510018	;Add accumulator from Mem[0018]
000C	71001A	;And accumulator from Mem[001A]
000F	81001C	;Or accumulator from Mem[001C]
0012	F1FFFE	;Store byte to output port
0015	F1FFFF	;Store byte to power off port
0018	0000	;One-word storage for first number
001A	000F	;AND mask
001C	0030	;OR mask

Input
25

Output
7

```
          A: 0000 0000 0011 0010
Mem[0018]: 0000 0000 0011 0010
```

```
Address     Machine Language (hex)
0000        D1FFFD ;Load byte accumulator from input port
0003        E10018 ;Store word accumulator to Mem[0018]
0006        D1FFFD ;Load byte accumulator from input port
0009        510018 ;Add accumulator from Mem[0018]
000C        71001A ;And accumulator from Mem[001A]
000F        81001C ;Or accumulator from Mem[001C]
0012        F1FFFE ;Store byte to output port
0015        F1FFFF ;Store byte to power off port
0018        0000   ;One-word storage for first number
001A        000F   ;AND mask
001C        0030   ;OR mask
```

Input

```
25
```

```
          A: 0000 0000 0011 0101
Mem[0018]: 0000 0000 0011 0010
```

Output

```
7
```

```
Address     Machine Language (hex)
0000        D1FFFD ;Load byte accumulator from input port
0003        E10018 ;Store word accumulator to Mem[0018]
0006        D1FFFD ;Load byte accumulator from input port
0009        510018 ;Add accumulator from Mem[0018]
000C        71001A ;And accumulator from Mem[001A]
000F        81001C ;Or accumulator from Mem[001C]
0012        F1FFFE ;Store byte to output port
0015        F1FFFF ;Store byte to power off port
0018        0000   ;One-word storage for first number
001A        000F   ;AND mask
001C        0030   ;OR mask
```

Input

```
25
```

Output

```
7
```

```
                A: 0000 0000 0110 0111
Mem[0018]:     0000 0000 0011 0010

                   0000 0000 0011 0101
              ADD  0000 0000 0011 0010
                   0000 0000 0110 0111
```

```
Address     Machine Language (hex)
0000        D1FFFD ;Load byte accumulator from input port
0003        E10018 ;Store word accumulator to Mem[0018]
0006        D1FFFD ;Load byte accumulator from input port
0009        510018 ;Add accumulator from Mem[0018]
000C        71001A ;And accumulator from Mem[001A]
000F        81001C ;Or accumulator from Mem[001C]
0012        F1FFFE ;Store byte to output port
0015        F1FFFF ;Store byte to power off port
0018        0000   ;One-word storage for first number
001A        000F   ;AND mask
001C        0030   ;OR mask
```

Input

```
25
```

Output

```
7
```

```
          A: 0000 0000 0000 0111
Mem[0018]: 0000 0000 0011 0010

           0000 0000 0110 0111
AND        0000 0000 0000 1111
           0000 0000 0000 0111
```

Address	Machine Language (hex)	
0000	D1FFFD	;Load byte accumulator from input port
0003	E10018	;Store word accumulator to Mem[0018]
0006	D1FFFD	;Load byte accumulator from input port
0009	510018	;Add accumulator from Mem[0018]
000C	71001A	;And accumulator from Mem[001A]
000F	81001C	;Or accumulator from Mem[001C]
0012	F1FFFE	;Store byte to output port
0015	F1FFFF	;Store byte to power off port
0018	0000	;One-word storage for first number
001A	000F	;AND mask
001C	0030	;OR mask

Input

25

Output

7

```
          A: 0000 0000 0011 0111
Mem[0018]: 0000 0000 0011 0010

          0000 0000 0000 0111
OR        0000 0000 0011 0000
          0000 0000 0011 0111
```

```
Address     Machine Language (hex)
0000        D1FFFD ;Load byte accumulator from input port
0003        E10018 ;Store word accumulator to Mem[0018]
0006        D1FFFD ;Load byte accumulator from input port
0009        510018 ;Add accumulator from Mem[0018]
000C        71001A ;And accumulator from Mem[001A]
000F        81001C ;Or accumulator from Mem[001C]
0012        F1FFFE ;Store byte to output port
0015        F1FFFF ;Store byte to power off port
0018        0000   ;One-word storage for first number
001A        000F   ;AND mask
001C        0030   ;OR mask
```

Input

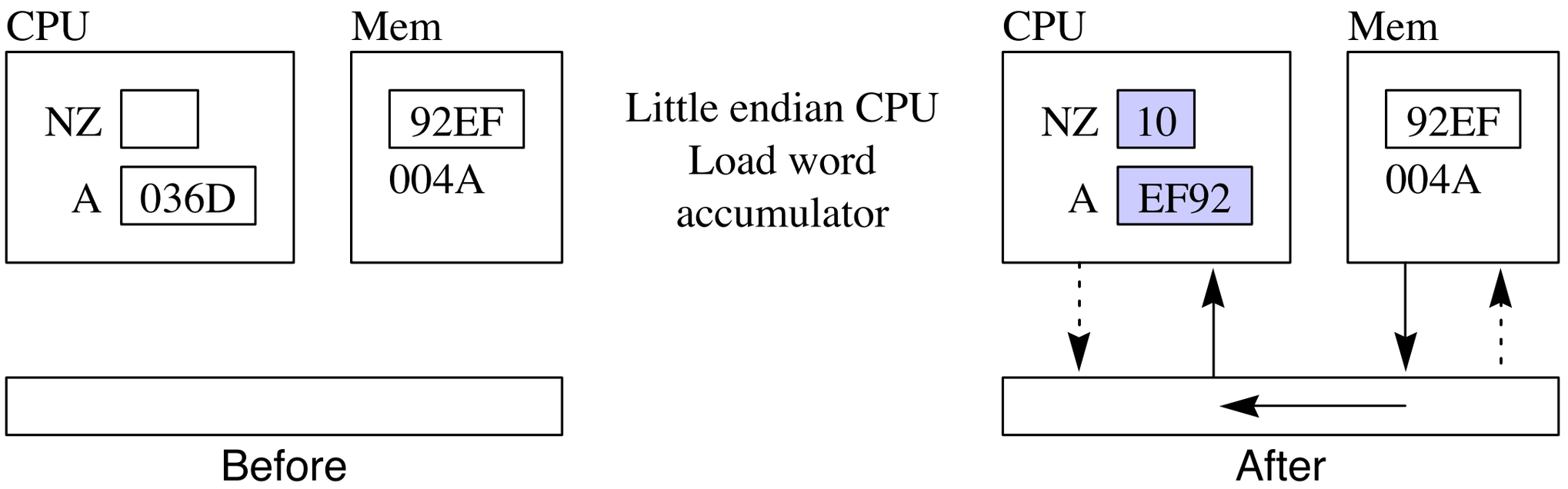
```
25
```

```
                    A: 0000 0000 0011 0111
Mem[0018]: 0000 0000 0011 0010
```

Output

```
7
```

A little-endian CPU



A 32-bit register load

Before	Instruction	After
A: 00 00 00 00 Mem[004A]: 89 AB CD EF	Big-endian CPU Load accumulator	A: 89 AB CD EF Mem[004A]: 89 AB CD EF
A: 00 00 00 00 Mem[004A]: 89 AB CD EF	Little-endian CPU Load accumulator	A: EF CD AB 89 Mem[004A]: 89 AB CD EF