System Programming

(Chapter 1: Background)

김지환교수

Office: AS관 713

Tel: 705-8924

Email: kimjihwan@sogang.ac.kr

System Software vs. Application Software (1)

System Software - a variety of programs that support the operation of a computer system. This makes it possible for the user to focus on an application or other problem to be solved without needing to know the details of how the machine works internally.

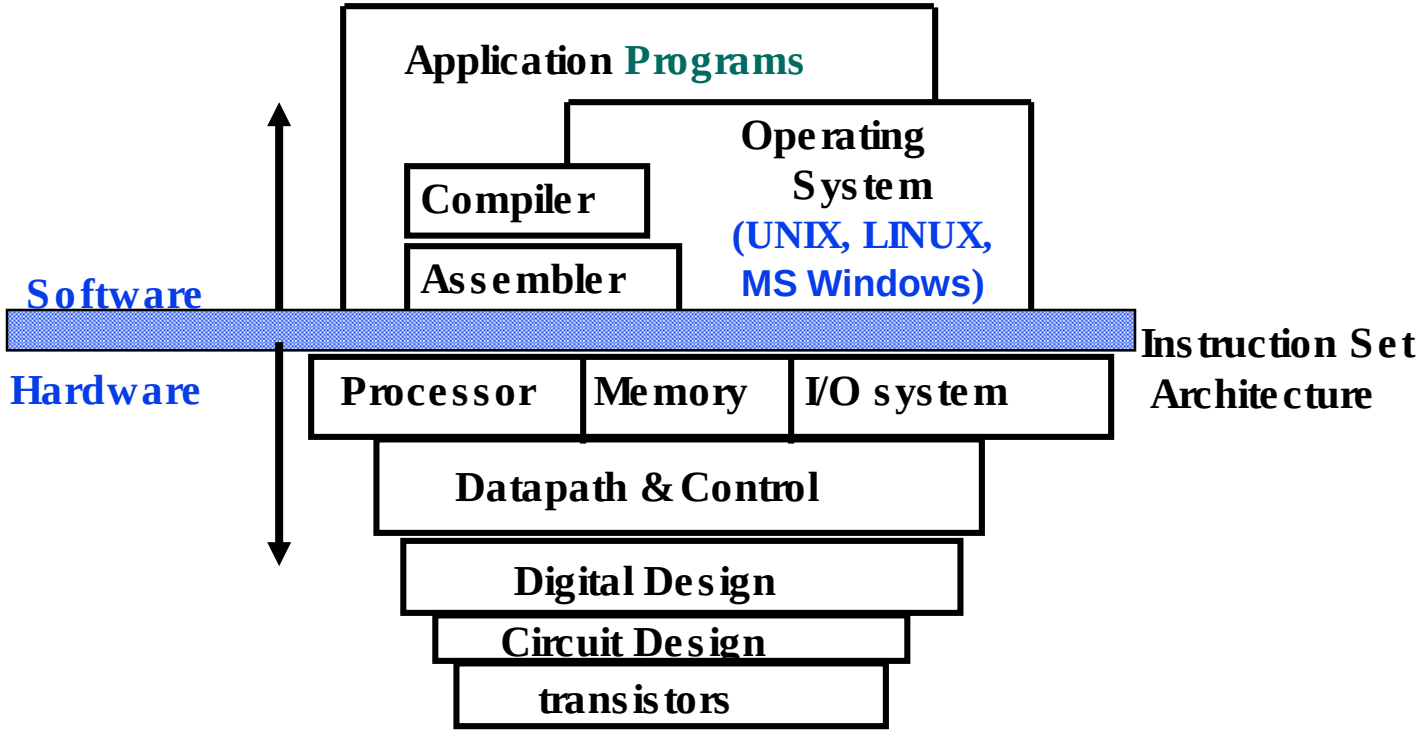
Examples of System Software

- **Assembler : Translate assembly language into machine code.**
- **Compiler : Translate high-level language into assembly code.**
- **Text Editor : Help users to create files.**
- **Linker : Combine object files into a single executable image.**
- **Loader : Place object file on the memory.**
- **Operating System : Manage all the hardware and software.**

System Software vs. Application Software (2)

Characteristics of a System Software - machine dependency.

An application program is primarily concerned with the solution of some problem, while system programs are intended to support the operation and use of the computer itself.



Road Maps of This Course (전반부)

Teach how to build system software based on the hypothetical computer, SIC (Simplified Instructional Computer).

Clearly separate the fundamental concepts of a piece of software from the implementation details associated with a particular computer.

Will cover assembler, linker and loader, compiler, operating system issues based on the SIC.

Two versions

- SIC : Standard version.
- SIC/XE : SIC with Extra Equipment.

SIC is upward compatible to SIC/XE.

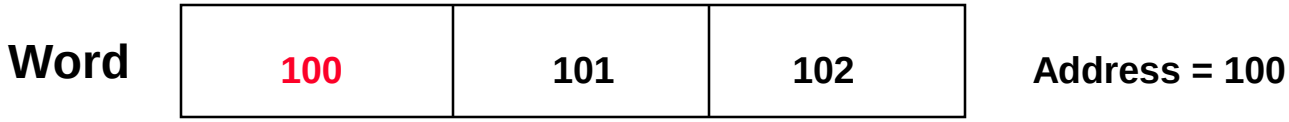
SIC Machine Structure (1)

Memory

1 byte : 8 bits, 1 word : 3 bytes (24 bits).

All addresses are byte addresses.

Address of a word : address of its lowest numbered byte.



Total of 32,768 (2^{15}) bytes in the memory.

Registers

5 registers.

Each register is 24 bits in length.

SIC Machine Structure (2)

Mnemonic	Number	Special Use
A	0	Accumulator; used for arithmetic operations
X	1	Index registers; used for addressing
L	2	Linkage register; the Jump to Subroutine (JSUB) instruction stores the return address in this register
PC	8	Program counter
SW	9	Status word; contains a variety of information; including a Condition Code (CC)

Data Format

Integer : 24 bit binary numbers

Character : 8 bit ASCII code. No floating point on the SIC.

SIC Machine Structure (3)

Instruction Format

Addressing Mode

2 addressing mode using x-bit.

Direct : x = 0, TA (Target Address) = address

Indexed : x = 1, TA = address + (X)

Instruction Sets

Data Movement : LDA, LDX, STA, STX, ….

Arithmetic : ADD, SUB, MUL, DIV, ….

Comparison : COMP, …

Conditional Jump : JEQ, JLT, JGT, ….

SIC Machine Structure (4)

Instruction Sets (continued)

Jump to Subroutine : JSUB (L register contains the return address)

Return from Subroutine : RSUB

Input and Output

Each I/O device is identified by an unique 8-bit code.

Transfer 1 byte from the rightmost 8 bits of register A.

3 I/O instructions

TD (Test Device) : Check if the I/O device is ready.

Condition code : “<“ means ready, “=“ means device busy.

RD (Read Data), WD (Write Data) : Read/Write from/to the I/O device.

SIC/XE Machine Structure (1)

Memory

1 byte : 8 bits, 1 word : 3 bytes (24 bits)

Total 1 megabyte (2^{20} bytes) can be addressed.

Registers (additional registers)

Mnemonic	Number	Special Use
B	3	Base register; used for addressing
S	4	General working register
T	5	General working register
F	6	Floating point accumulator (48 bits)

Data Format

Same data format as the standard version.

1 additional format for the 48-bit floating point data type.

SIC/XE Machine Structure (2)

0 =< Fraction =< 1

0 =< Exponent <= 2047

$(-1)^{s} frac * 2^{(exp-1024)}$

S = 1 -> Negative

S = 0 -> Positive

Instruction Format

Consists of 4 instruction formats.

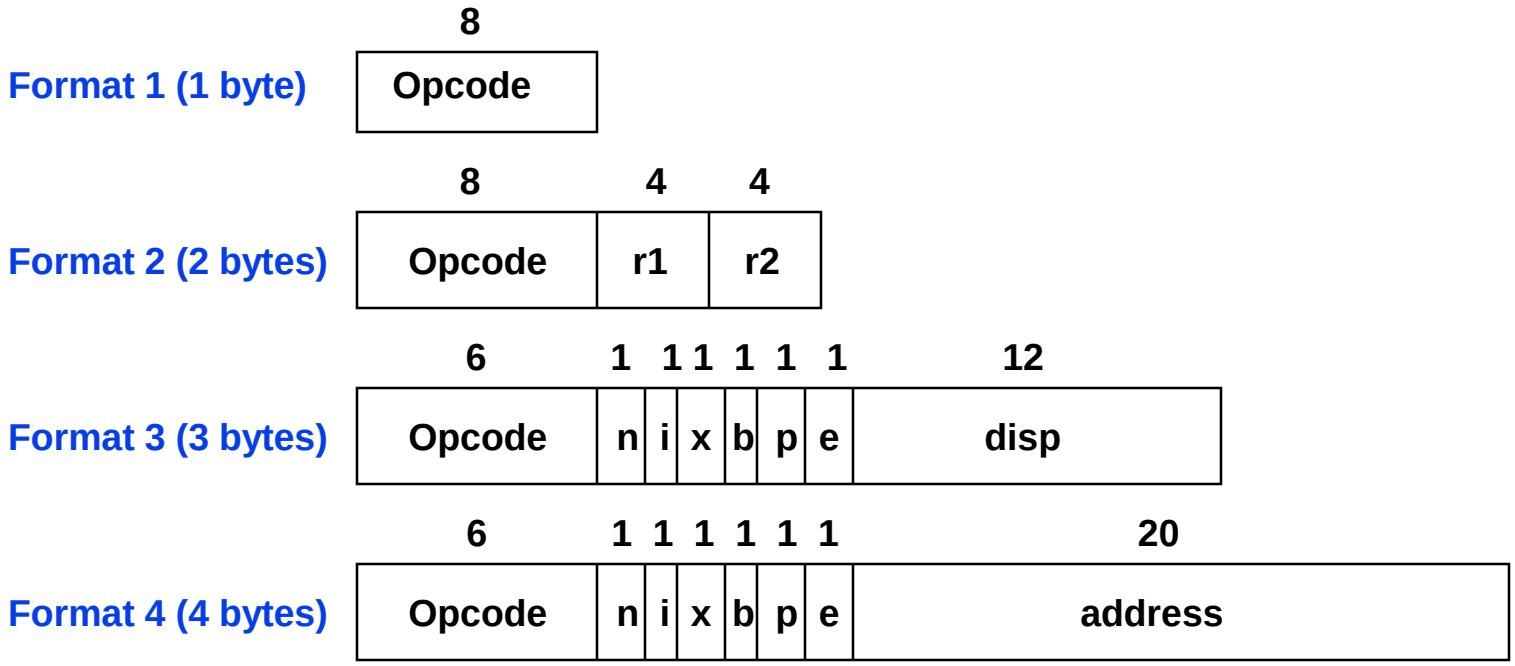
The larger memory available on SIC/XE means that an address will no longer fit into a 15-bit field (SIC case).

SIC/XE includes two solutions.

- Relative addressing mode (format 3 - check next slide).
- Extend the address field to 20 bits (format 4 - check next slide).

SIC/XE Machine Structure (3)

Instruction Format (continued)



Format 1 and 2 are used for instructions that do not reference memory at all.

If bit e = 0, format 3, else (bit e = 1) format 4. Other bits will be covered in the addressing mode section (next slide).

SIC/XE Machine Structure (4)

Addressing Mode

Base relative : b = 1, p = 0, TA = (B) + disp (0 =< disp =< 4095)

disp part is interpreted as a 12-bit unsigned integer.

PC relative : b = 0, p = 1, TA = (PC) + disp (-2048 =< disp =< 2047)

disp is interpreted as a 12-bit signed integer.

If b = 0 and p = 0, disp field is the target address (*direct addressing*).

Any of the addressing modes can be combined with *indexed addressing* mode (i.e., if x bit is set to 1).

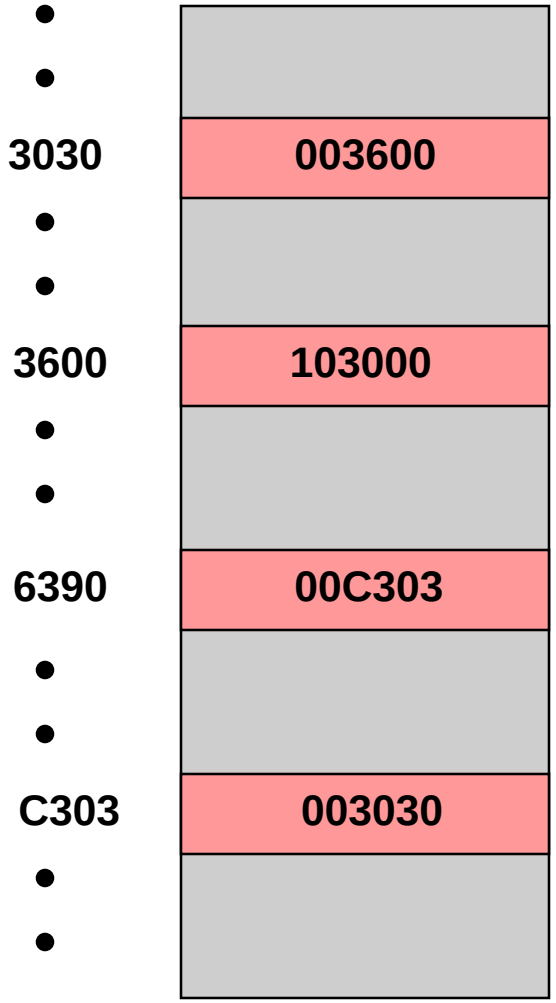
If i = 1 and n = 0, *immediate addressing* (no memory reference).

If i = 0 and n = 1, *indirect addressing*.

If i = 1 and n = 1, *simple addressing* (target address is considered as the location of the operand).

If i = 0 and n = 0, considered as standard SIC instruction. Bits b, p, e are considered to be part of the address field of the instruction.

Examples of SIC/XE Addressing Mode



(B) = 006000

(PC) = 003000

(X) = 000090

← Machine Instruction (Binary) →

OP	n	i	x	b	p	e	disp/address	Target Ad	Val (A)
000000	1	1	0	0	1	0	0110 0000 0000	3600	103000
000000	1	1	1	1	0	0	0011 0000 0000	6390	00C303
000000	1	0	0	0	1	0	0000 0011 0000	3030	103000
000000	0	1	0	0	0	0	0000 0011 0000	30	000030
000000	0	0	0	0	1	1	0110 0000 0000	3600	103000
000000	1	1	0	0	0	1	0000 1100 0011 0000 0011	C303	003030

SIC/XE Machine Structure (5)

Instruction Sets

Provide all the instructions that are available on the SIC.

Additional data movement : LDB, STB, ….

New floating point arithmetic : ADDF, SUBF, MULTF, DIVF, ….

Register-to-register : ADDR, SUBR, MULR, DIVR, ….

Supervisor call (SVC) for communicating with OS (interrupt).

Input and Output

Supports all I/O instructions that are available on the SIC.

Additional I/O channel instructions : SIO (Start I/O), TIO (Test I/O), HIO (Halt I/O).

I/O channel instructions are used to perform input and output while the CPU is executing other instructions (overlapping).

SIC Programming Examples (1)

Data Movement Operations

```
        LDA     FIVE    A <= (FIVE)
        STA     ALPHA   (ALPHA) <= A
        LDCH    CHARZ   A <= (CHARZ)
        STCH    C1      (C1) <= A
        .
        .
        .
ALPHA   RESW    1       Word Variable
FIVE    WORD    5       Word '5'
CHARZ   BYTE    C'Z'    Byte 'Z'
C1      RESB    1       Byte Variable
```

(a) SIC

```
        LDA     #5      A <= 5
        STA     ALPHA   (ALPHA) <= A
        LDA     #90     A <= 90
        STCH    C1      (C1) <= A
        .
        .
        .
ALPHA   RESW    1       Word Variable
C1      RESB    1       Byte Variable
```

(b) SIC/XE

SIC Programming Examples (2)

Arithmetic Operations

```
        LDA     ALPHA   A <= (ALPHA)
        ADD     INCR    A <= A + (INCR)
        SUB     ONE     A <= A - (ONE)
        STA     BETA    (BETA) <= A
        LDA     GAMMA   A <= (GAMMA)
        ADD     INCR    A <= A + (INCR)
        SUB     ONE     A <= A - (ONE)
        STA     DELTA   (DELTA) <= A
        .
ONE     WORD    1       Word '1'
ALPHA   RESW    1       Word Variable
BETA    RESW    1       Word Variable
GAMMA   RESW    1       Word Variable
DELTA   RESW    1       Word Variable
INCR    RESW    1       Word Variable
```

(a) SIC

```
        LDS     INCR    S <= (INCR)
        LDA     ALPHA   A <= (ALPHA)
        ADDR    S, A    A <= A + S
        SUB     #1      A <= A - 1
        STA     BETA    (BETA) <= A
        LDA     GAMMA   A <= (GAMMA)
        ADDR    S, A    A <= A + S
        SUB     #1      A <= A - 1
        STA     DELTA   (DELTA) <= A
        .
ALPHA   RESW    1       Word Variable
BETA    RESW    1       Word Variable
GAMMA   RESW    1       Word Variable
DELTA   RESW    1       Word Variable
INCR    RESW    1       Word Variable
```

(b) SIC/XE

SIC Programming Examples (3)

Looping and Indexing Operations (Check Figure 1.5)

```
        LDX     ZERO      X <= (ZERO)
MOVECH  LDCH    STR1, X   A <= (STR1+X)
        STCH    STR2, X   (STR2+X) <= A
        TIX     ELEVEN    Compare X+1
                          and (ELEVEN)
        JLT     MOVECH    Loop if X is less
                          than 11
        .
STR1    BYTE    C'TEST STRING'
STR2    RESB    11        11 Byte Variable
        .
ZERO    WORD    0         Word '0'
ELEVEN  WORD    11        Word '11'
```

(a) SIC

```
        LDT     #11       T <= 11
        LDX     #0        X <= 0
MOVECH  LDCH    STR1, X   A <= (STR1+X)
        STCH    STR2, X   (STR2+X) <= A
        TIXR    T         Compare X+1
                          and T
        JLT     MOVECH    Loop if X is less
                          than 11
        .
STR1    BYTE    C'TEST STRING'
STR2    RESB    11        11 Byte Variable
```

(b) SIC/XE

SIC Programming Examples (4)

Input and Output Operations

```
INLOOP   TD      INDEV    Test Input Dev
         JEQ     INLOOP   = : Not ready
                          < : Ready
         RD      INDEV    A <= Read Byte
         STCH    DATA     (DATA) <= A
         .
         .
OUTLP    TD      OUTDEV   Test Output Dev
         JEQ     OUTLP    = : Not Ready
                          < : Ready
         LDCH    DATA     A <= (DATA)
         WD      OUTDEV   Device <= A
```

```
INDEV    BYTE    X'F1'    Byte 'F1'
OUTDEV   BYTE    X'05'    Byte '05'
DATA     RESB    1        Byte Variable
```

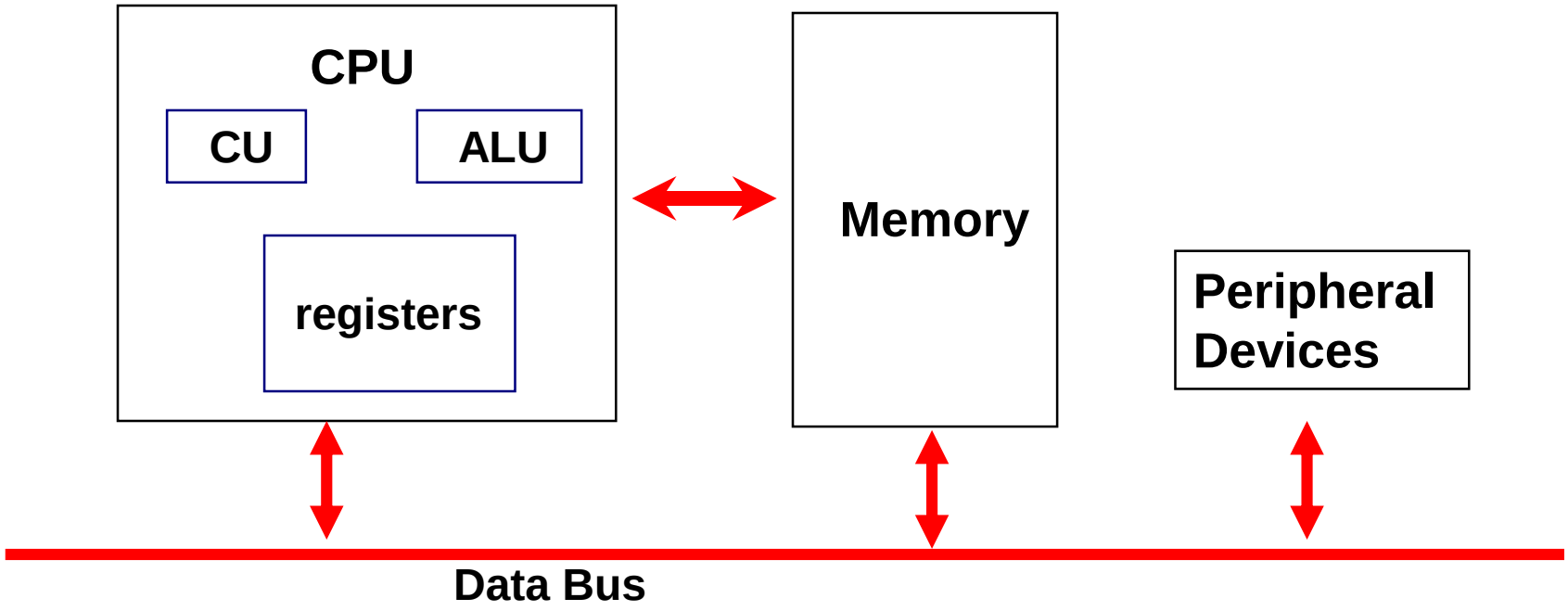
SIC Programming Examples (5)

Subroutine Call and Record Input Operations

```
        JSUB    READ    Jump to READ
        .
READ    LDX     ZERO    X <= (ZERO)
RLOOP   TD      INDEV   Test if Ready
        JEQ     RLOOP   Repeat if Busy
        RD      INDEV   A <= Read Byte
        STCH    RECORD, X (RECORD+X)
                          <= A
        TIX     K100    X+1 & Compare
        JLT     RLOOP   Repeat if less
        RSUB            Return
        .
INDEV   BYTE    X'F1'   Byte 'F1'
RECORD  RESB    100     100 Bytes
ZERO    WORD    0       Word '0'
K100    WORD    100     Word '100'
```

```
        JSUB    READ    Jump to READ
        .
READ    LDX     #0      X <= 0
        LDT     #100    T <= 100
RLOOP   TD      INDEV   Test if Ready
        JEQ     RLOOP   Repeat if Busy
        RD      INDEV   A <= Read Byte
        STCH    RECORD, X (RECORD+X)
                          <= A
        TIXR    T       X+1 & Compare
        JLT     RLOOP   Repeat if less
        RSUB            Return
        .
INDEV   BYTE    X'F1'   Byte 'F1'
RECORD  RESB    100     100 Bytes
```

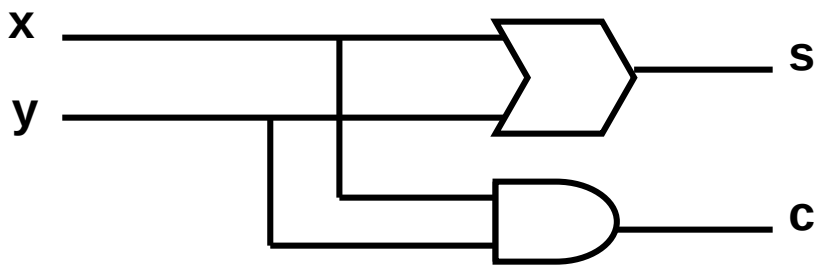
참고자료: 컴퓨터의 기본 구조 및 작동원리(review)



CPU

Control Unit

ALU



1-bit Adder logic

$s = x \oplus y$

$c = xy$

Adding Values Stored in Memory

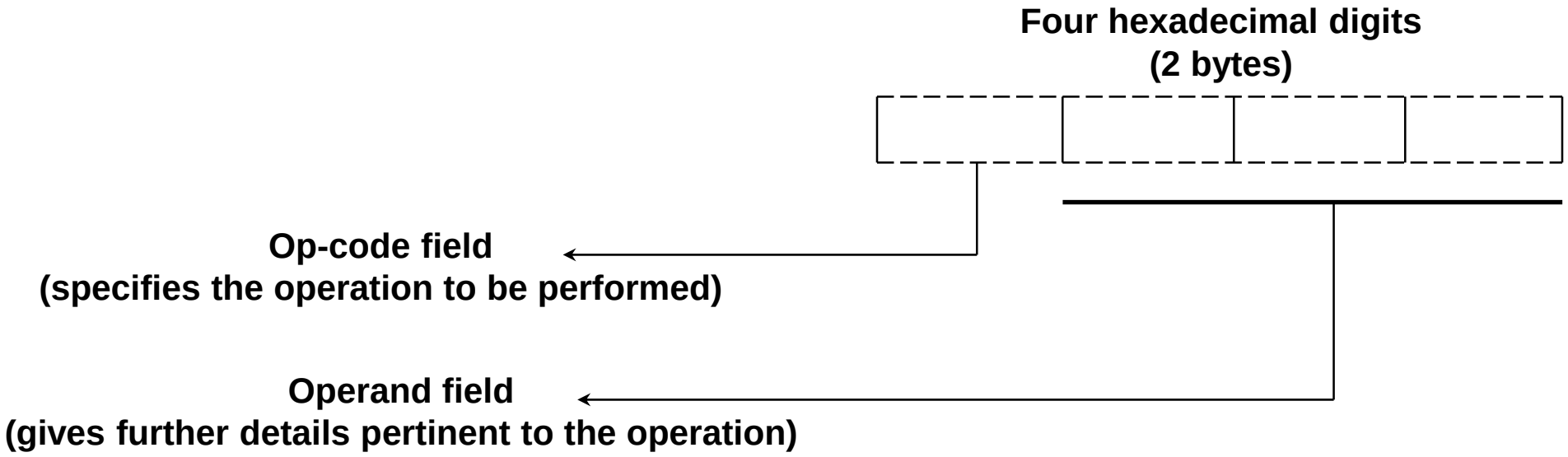
예: X = Y + Z

Step 1. Get one of the values to be added from memory and place it in a register. (Y값을 CPU로 읽어온다)
Step 2. Get the other value to be added from memory and place it in another register. (Z 값을 읽어온다)
Step 3. Activate the addition circuitry with the registers used in steps 1 and 2 as inputs and another register designated to hold the result. (더한다)
Step 4. Store the result in memory. (결과를 X 값으로 저장)
Step 5. Stop.

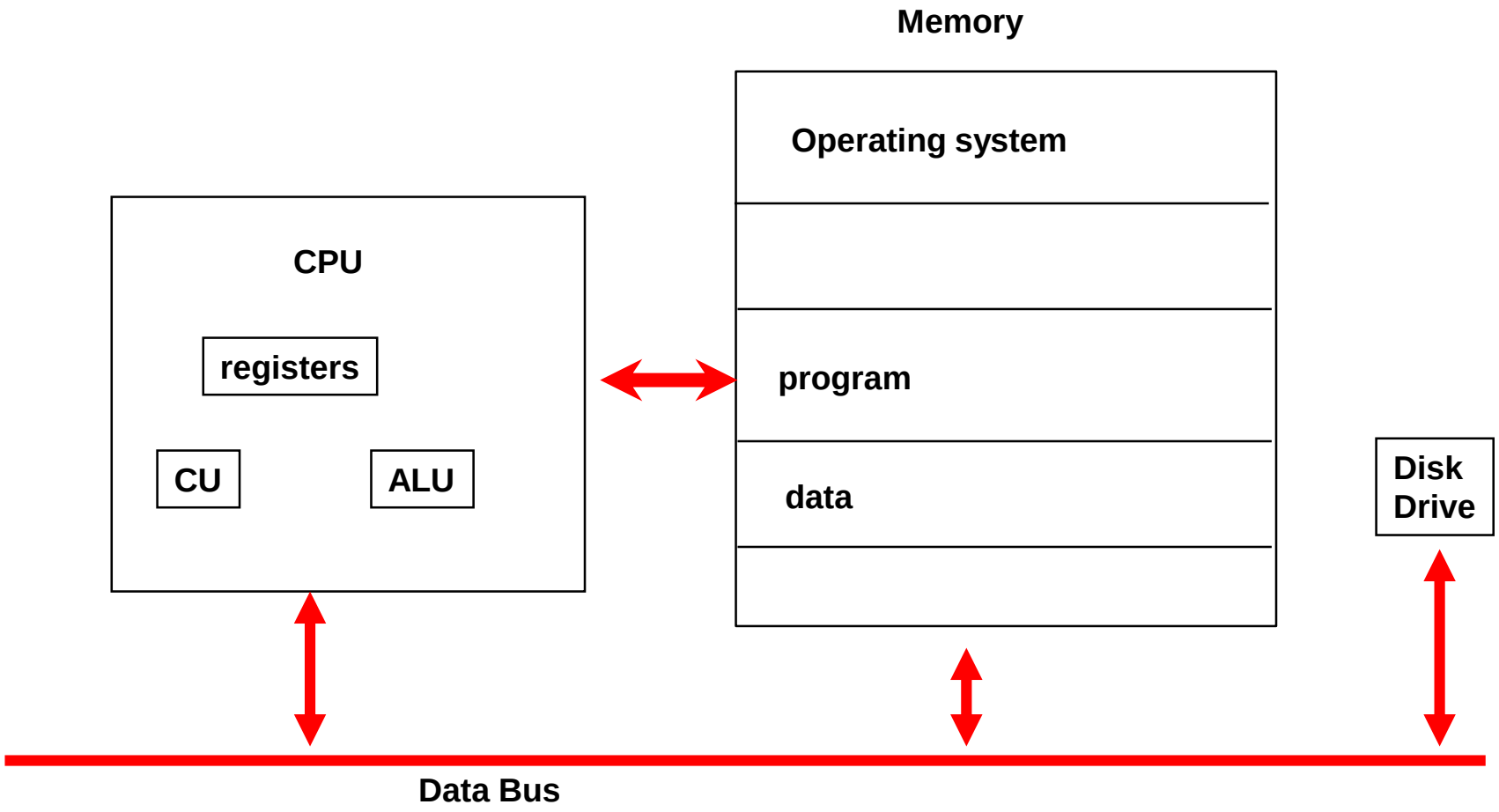
The Stored-Program Concept

- **Instructions as Bit Patterns**
  - **machine language**
  - **op-code and operand field**

The format of a machine instruction



컴퓨터의 작동 원리

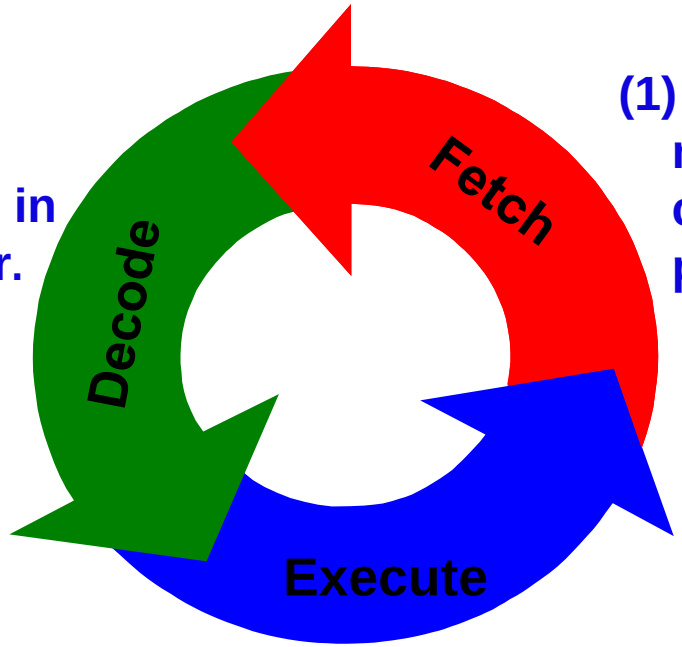

Memory
Operating system
CPU
registers
CU
ALU
program
data
Disk
Drive
Data Bus

Program Execution

The machine cycle

(1) Retrieve the next instruction from memory (as indicated be program counter) and then increment the program counter.

(2) Decode the bit pattern in the instruction register.



(4) Write-back or storage: stores the results to memory

(3) Perform the action requested by the instruction in the instruction register