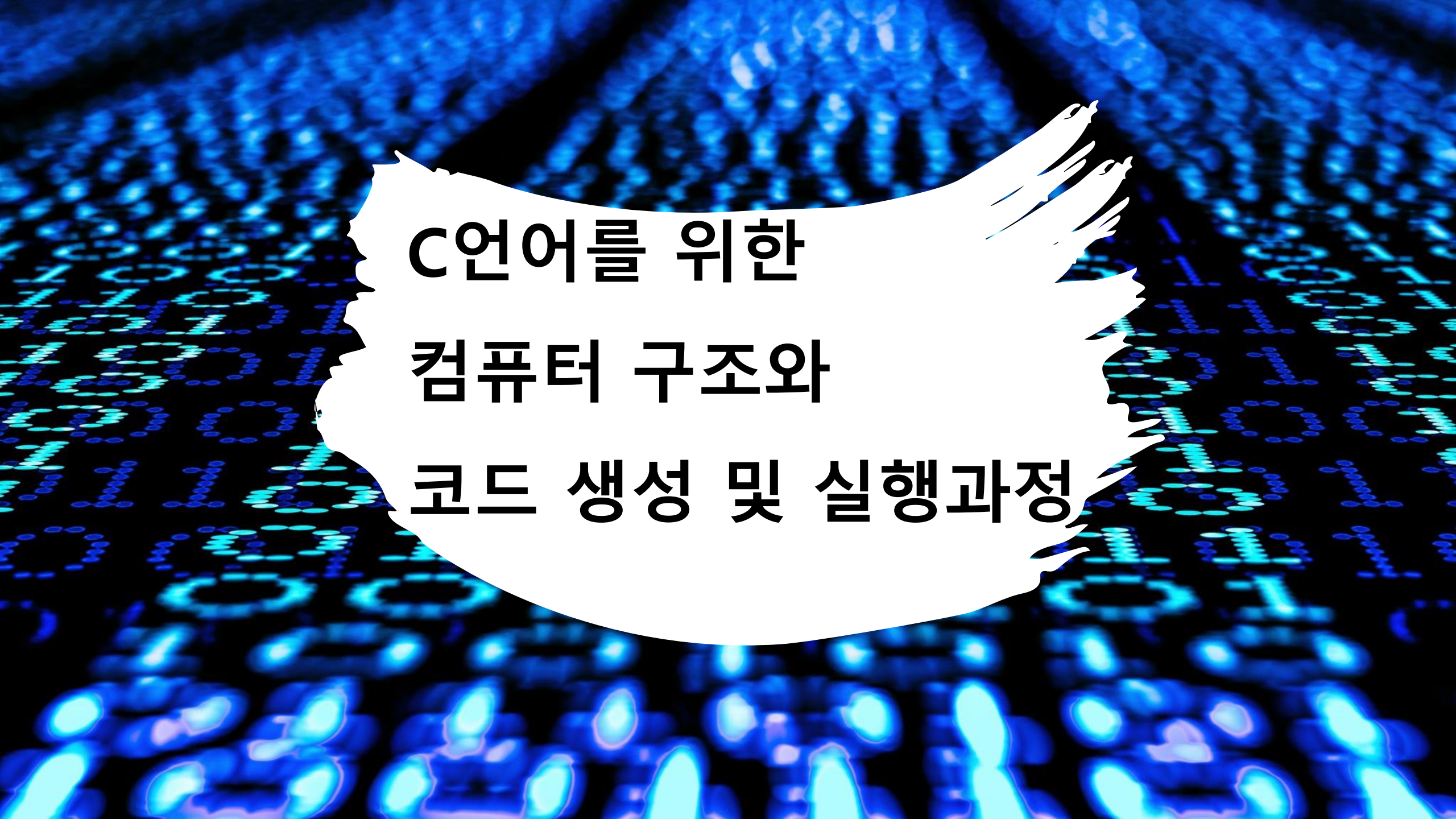
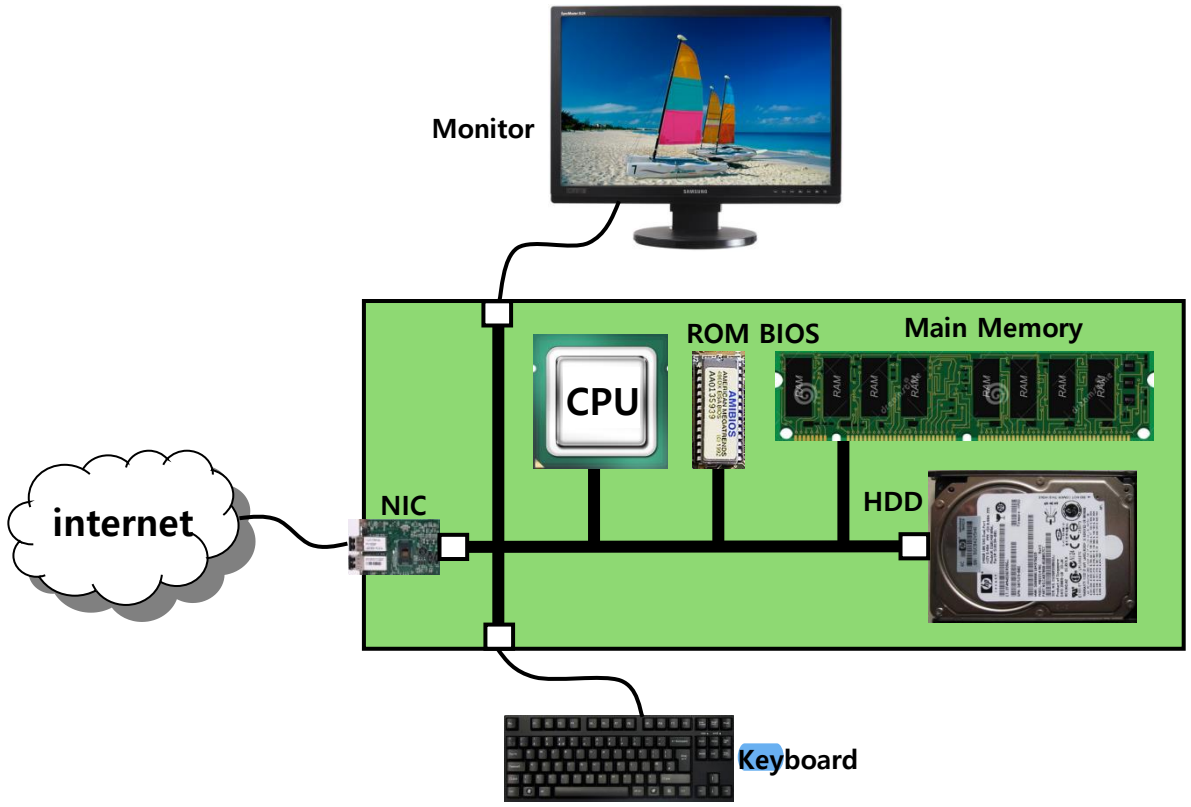
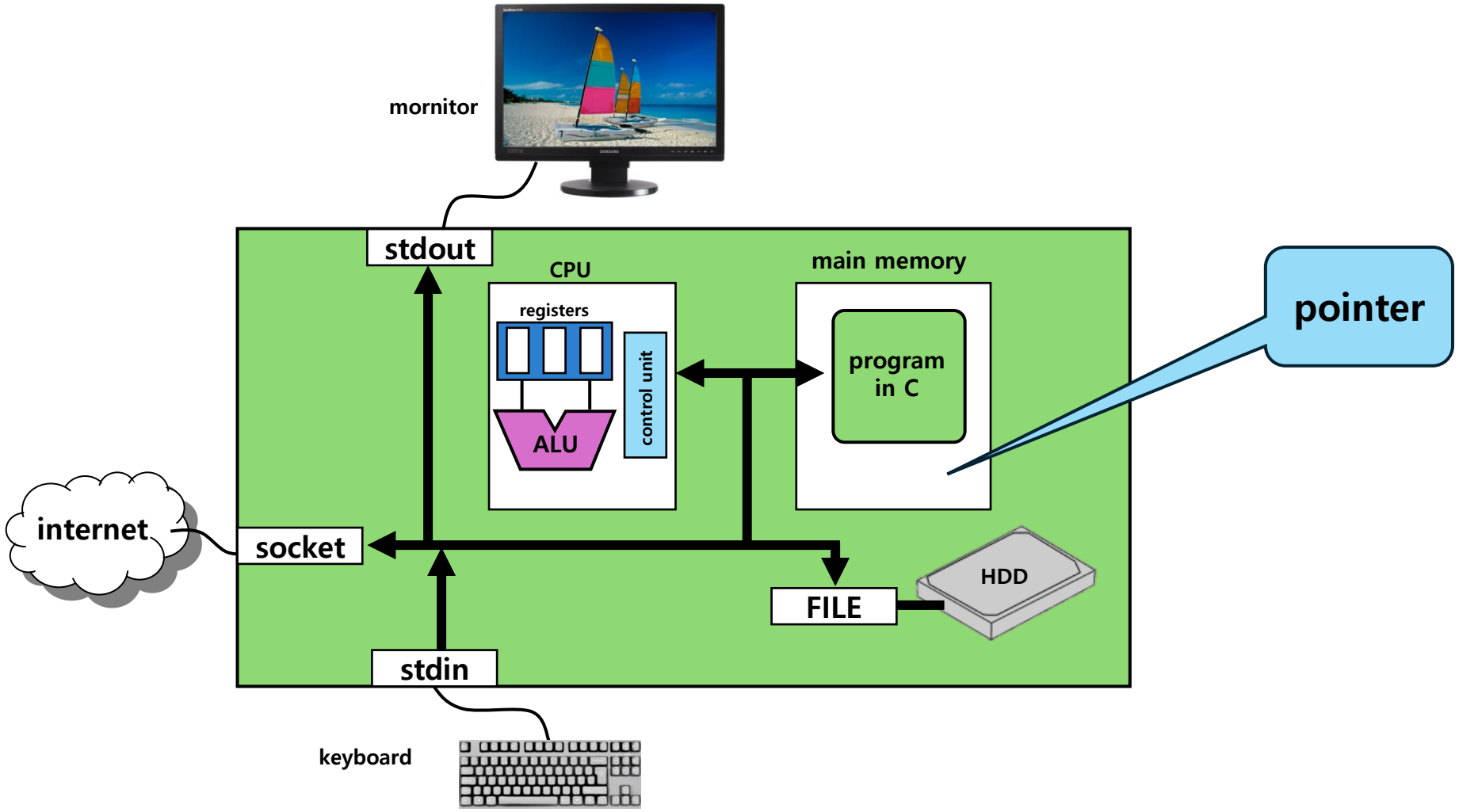

C언어를 위한
컴퓨터 구조와
코드 생성 및 실행과정

컴퓨터 구조

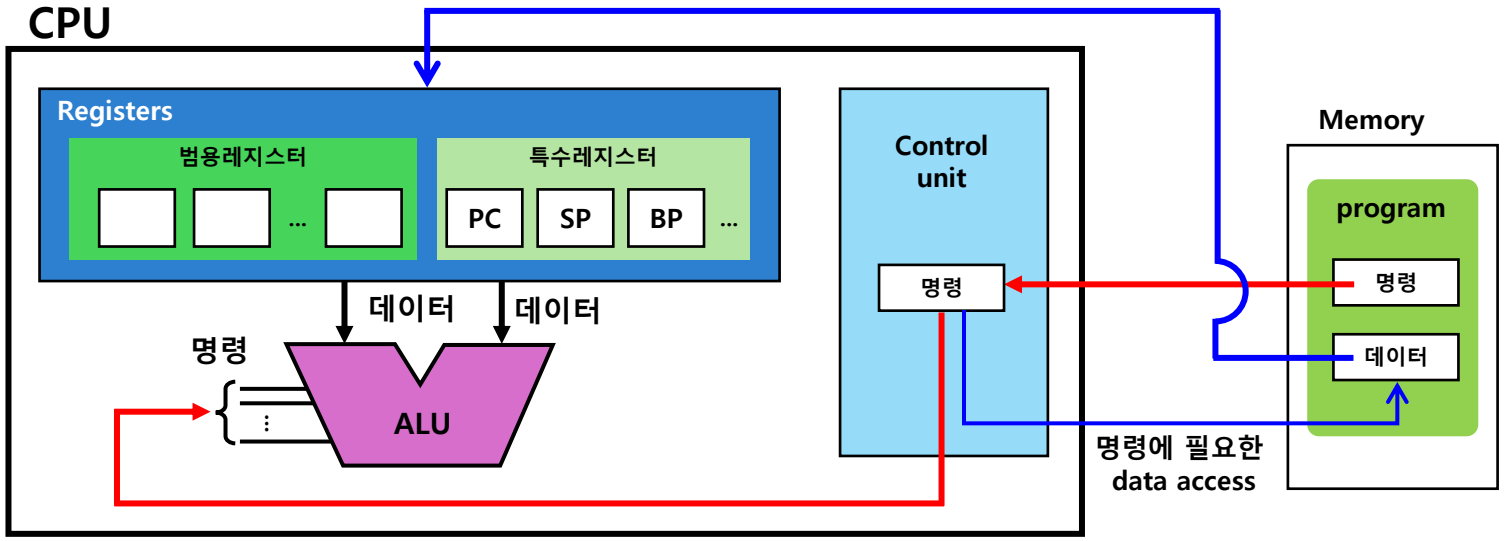
컴퓨터 구조



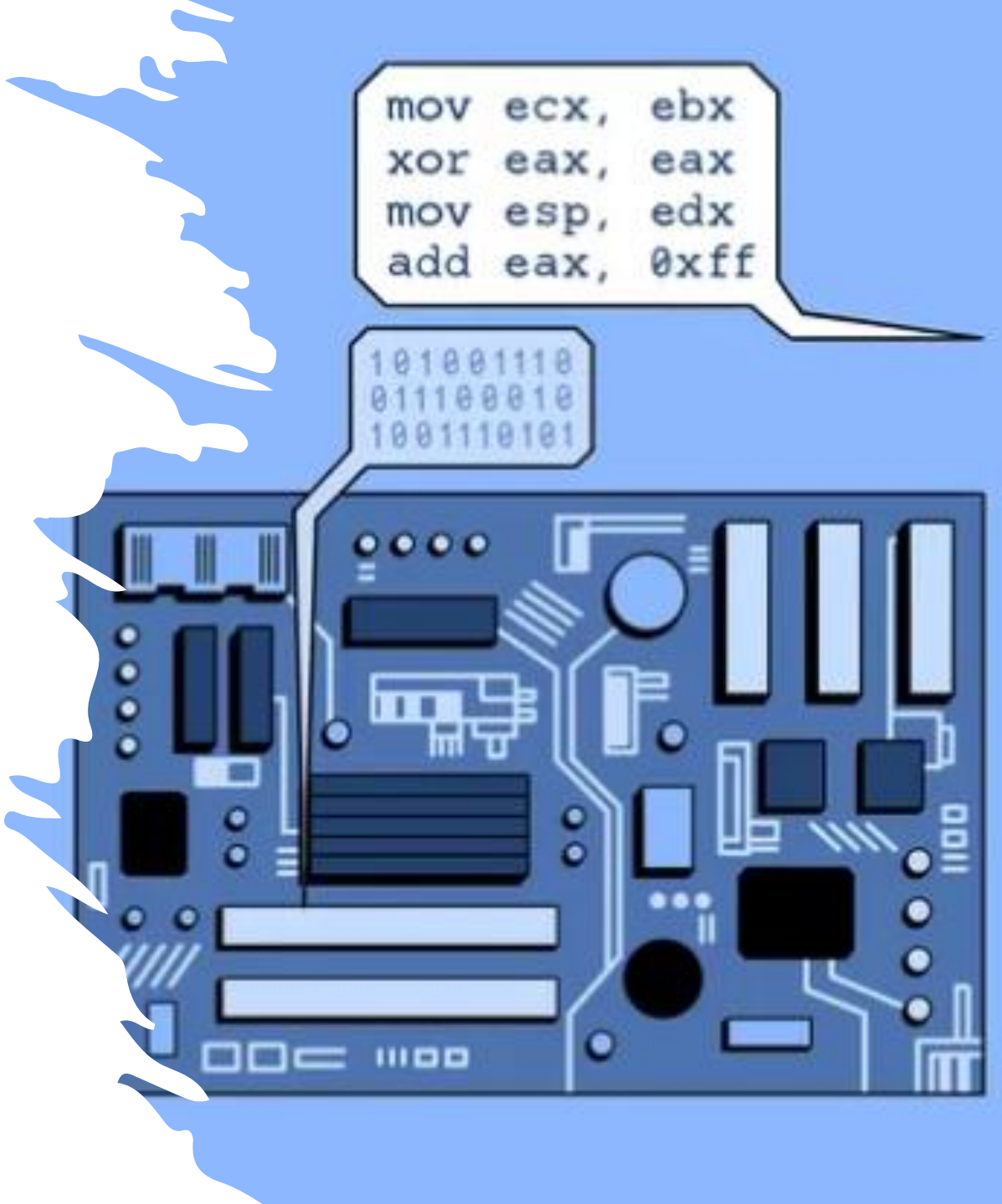
C언어에서 바라본 컴퓨터 구조



CPU 구조


CPU
Registers
범용레지스터
...
특수레지스터
PC
SP
BP
...
데이터
데이터
명령
ALU
Control unit
명령
Memory
program
명령
데이터
명령에 필요한 data access

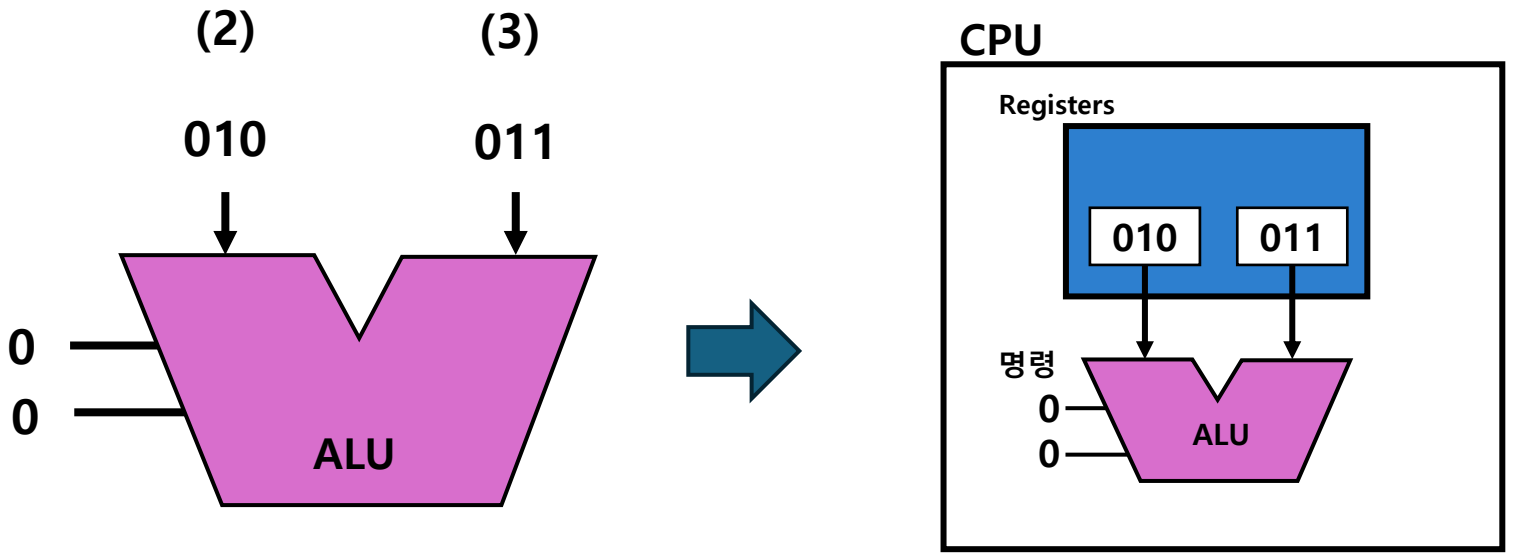
기계어,
어셈블리어
그리고 고급언어



기계어 코드

CPU Instruction Set

연산	명령코드
더하기	00
빼기	01
곱하기	10
나누기	11

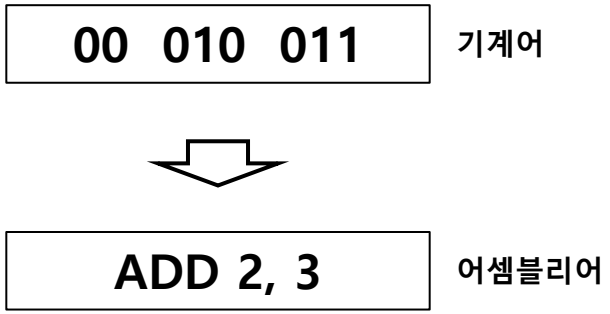


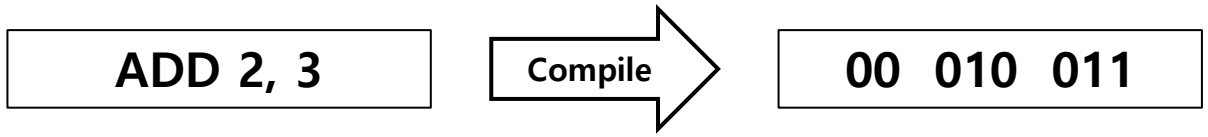
00010011
11100010
01101010
10011010

어셈블리어 코드

Instruction Set

연산	명령코드	니모닉
더하기	00	ADD
빼기	01	SUB
곱하기	10	MUL
나누기	11	DIV





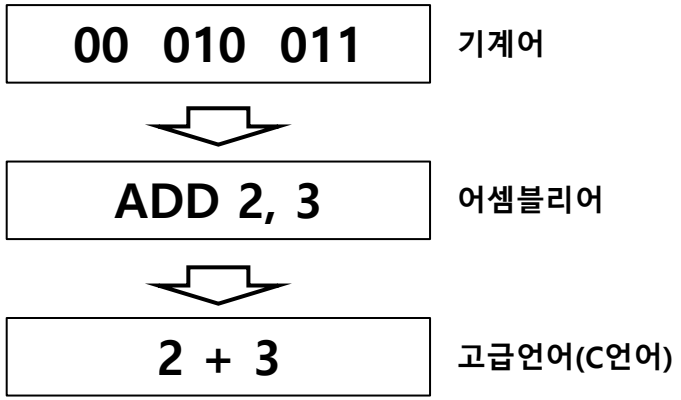
어셈블리어 코드

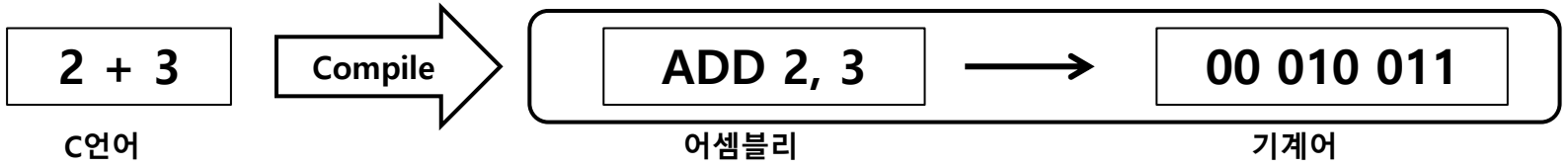
기계어어 코드

고급언어 (C언어) 코드

Instruction Set

연산	명령코드	니모닉	연산자
더하기	00	ADD	+
빼기	01	SUB	-
곱하기	10	MUL	*
나누기	11	DIV	/





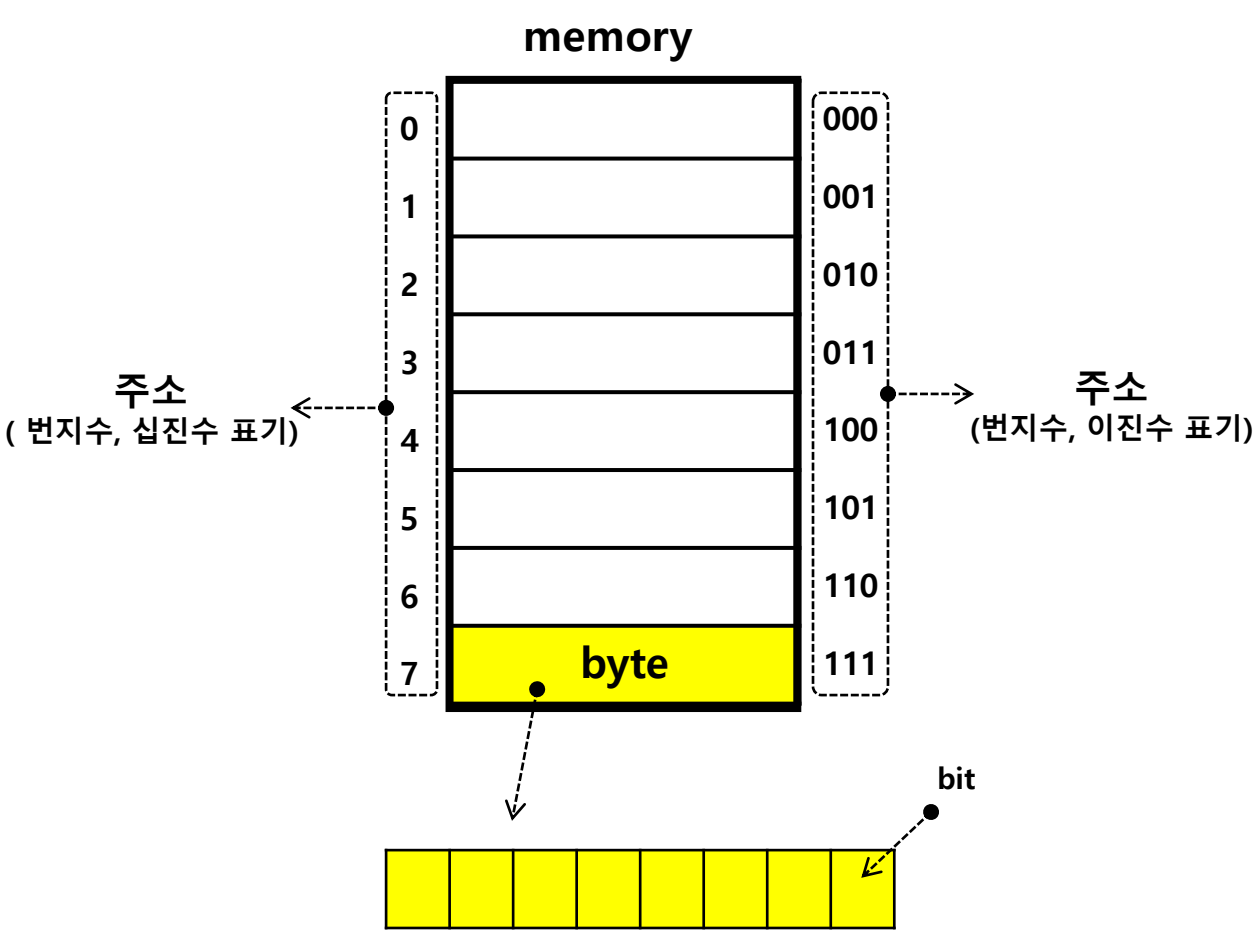

2 + 3
4 / 2
5 – 2
3 * 2
Compile
ADD 2, 3
DIV 4, 2
SUB 5, 2
MUL 3, 2
00010011
11100010
01101010
10011010
C언어 프로그램
어셈블리어
기계어

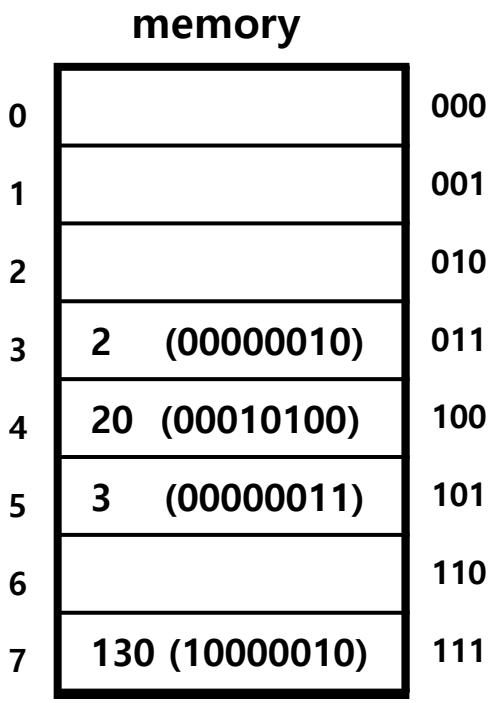
메모리 구조

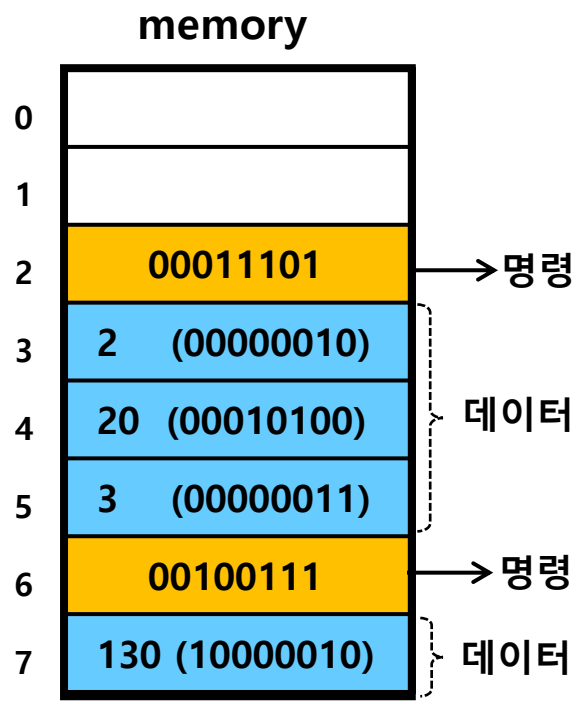

hynix 251TA
HY5DU561622CT - 33
KOREA 7J30010BA
hynix 251TA
HY5DU561622CT - 33
KOREA 7J30010BA

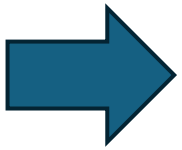
메모리 구조

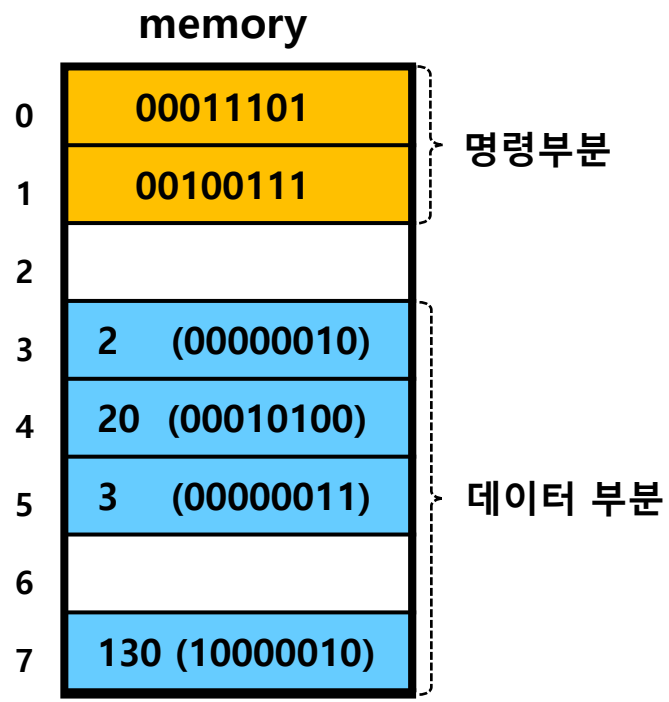
8 바이트 메모리









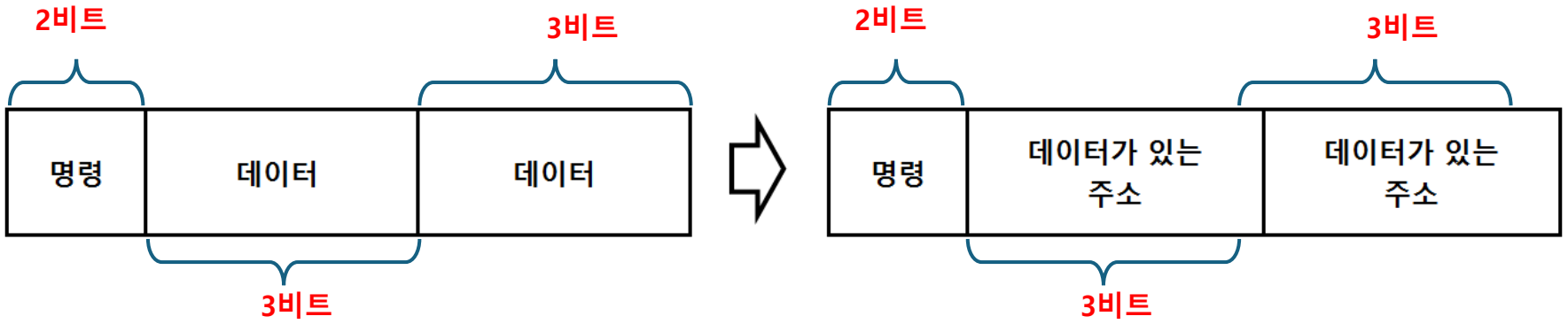


명령과 데이터 혼재
➔ 명령이 불규칙적으로 배치
➔ CPU가 차례차례로 명령을 가져와 실행할 수 없다!!!

명령과 데이터를 분리
➔ 메모리를 명령이 들어가는 부분과 데이터가 들어가는 부분으로 분리
➔ CPU가 차례차례로 명령을 가져와 실행할 수 있다!!!

메모리를 효율적으로 사용하기 위한 명령어 구조

명령어의 길이가 1 바이트(8 비트) 라고 하면



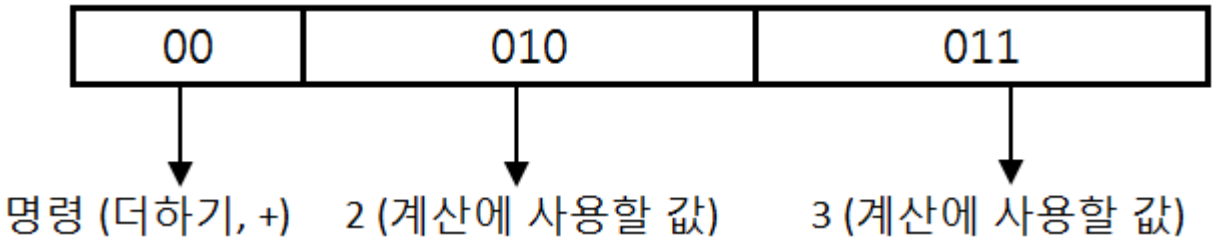
3 비트로 표현할 수 있는 데이터 값은 0부터 7까지

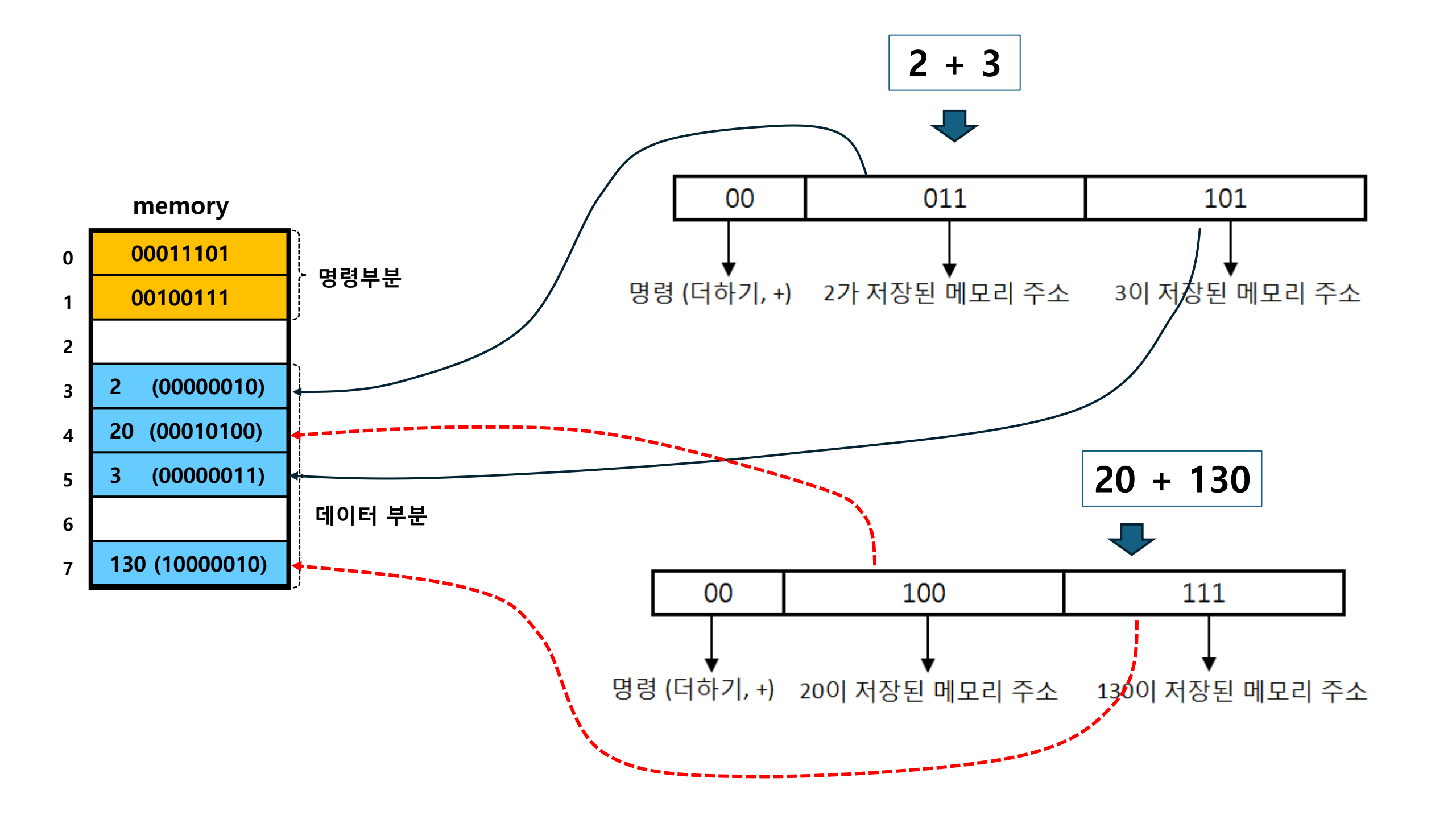
➔사용할 수 있는 수의 범위에 제약이 생김

➔ 2 + 10을 기계어로 표현할 수 없음

3 비트로 표현할 수 있는 주소은 0부터 7번지까지

➔ 한 번지는 8비트이므로 0~ 255까지 데이터 값을 사용 가능

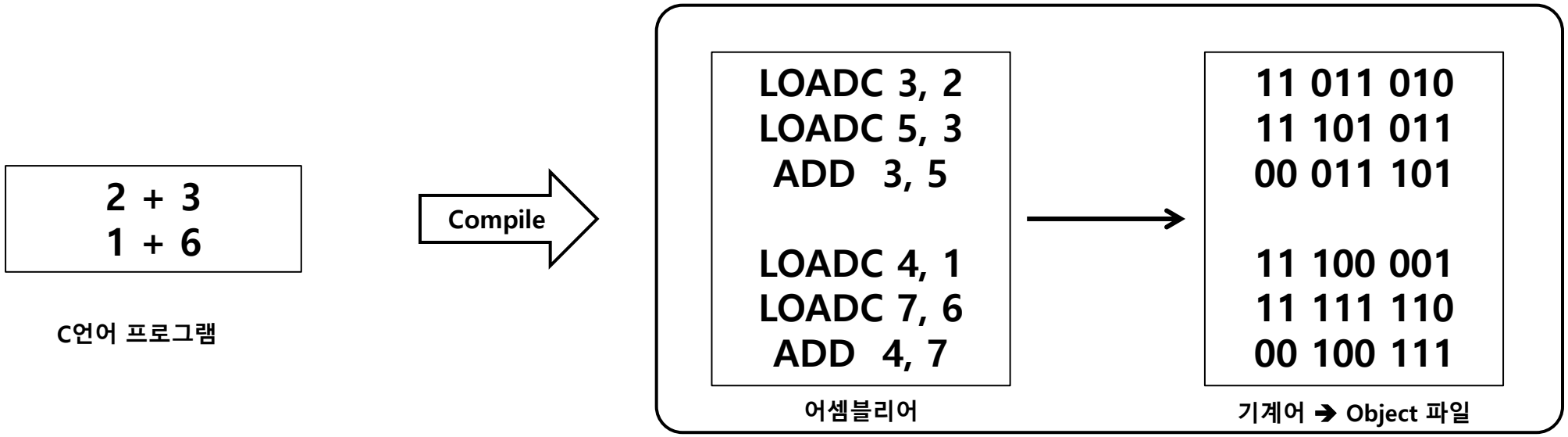



2 + 3
00
011
101
명령 (더하기, +)
2가 저장된 메모리 주소
3이 저장된 메모리 주소
memory
0
1
2
3
4
5
6
7
00011101
00100111
2 (00000010)
20 (00010100)
3 (00000011)
130 (10000010)
명령부분
데이터 부분
20 + 130
00
100
111
명령 (더하기, +)
20이 저장된 메모리 주소
130이 저장된 메모리 주소

```
        printf("Mafia can choice who wanna kill : ");
        scanf_s("%d",&i);
        printf("Player %d is die.\n",i);
        if (i==X)
        {
            printf("This Player was Mafia\n");

        }
        else if (i!=X)
        {
            printf("This Player was citizen\n");

        }
    }

    else if (N%2 != 0)
    {
        printf("--Daytime--\n");
        printf("Candidate can choice who are(is) mafi
        scanf_s("%d",&i);
        printf("Player %d is die.\n",i);
```

소스코드 작성과 실행과정

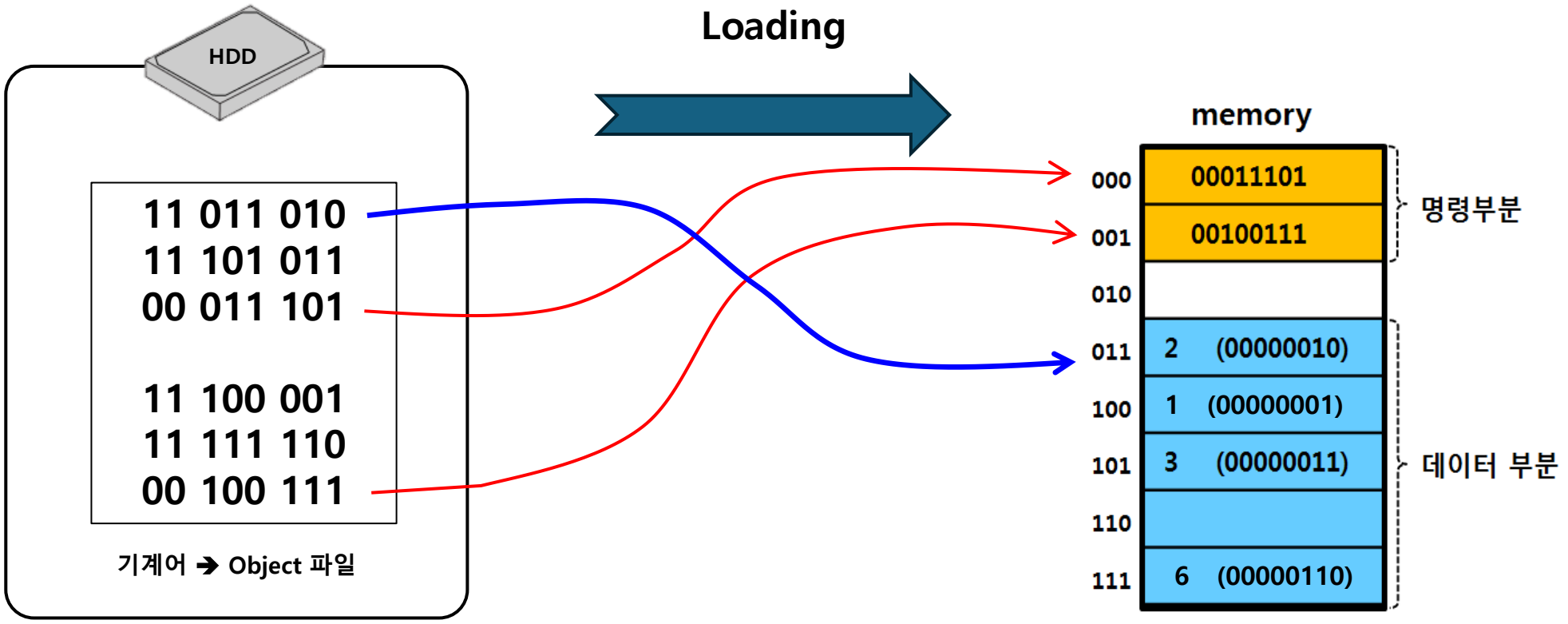
소스코드 작성과 컴파일

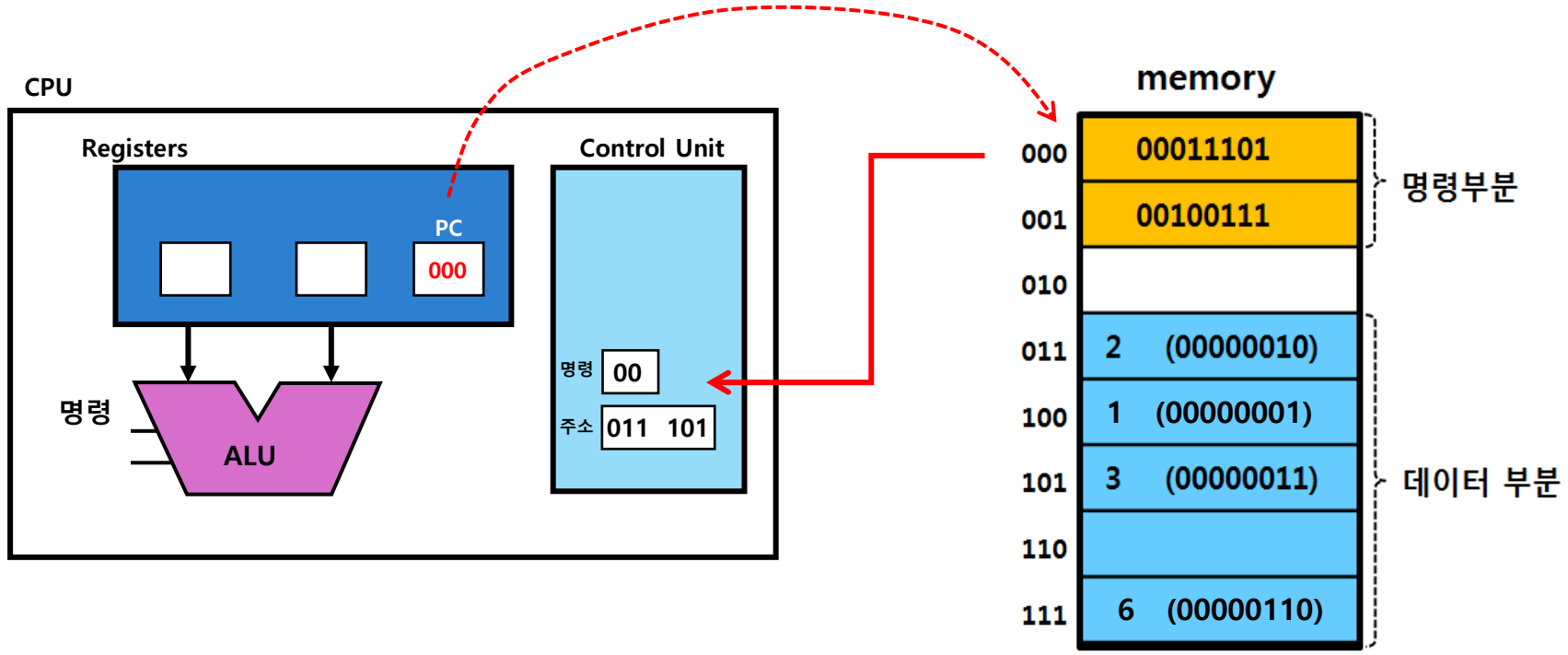


- LOADC addr, value ➔ addr 주소(번지)에 value 저장
  LOADC의 기계어 코드 ➔ 11 이라고 가정

- ADD addr1, addr2 ➔ addr1 번지와 addr2 번지에 있는 값을 더하기
  ADD의 기계어 코드 ➔ 00 이라고 가정

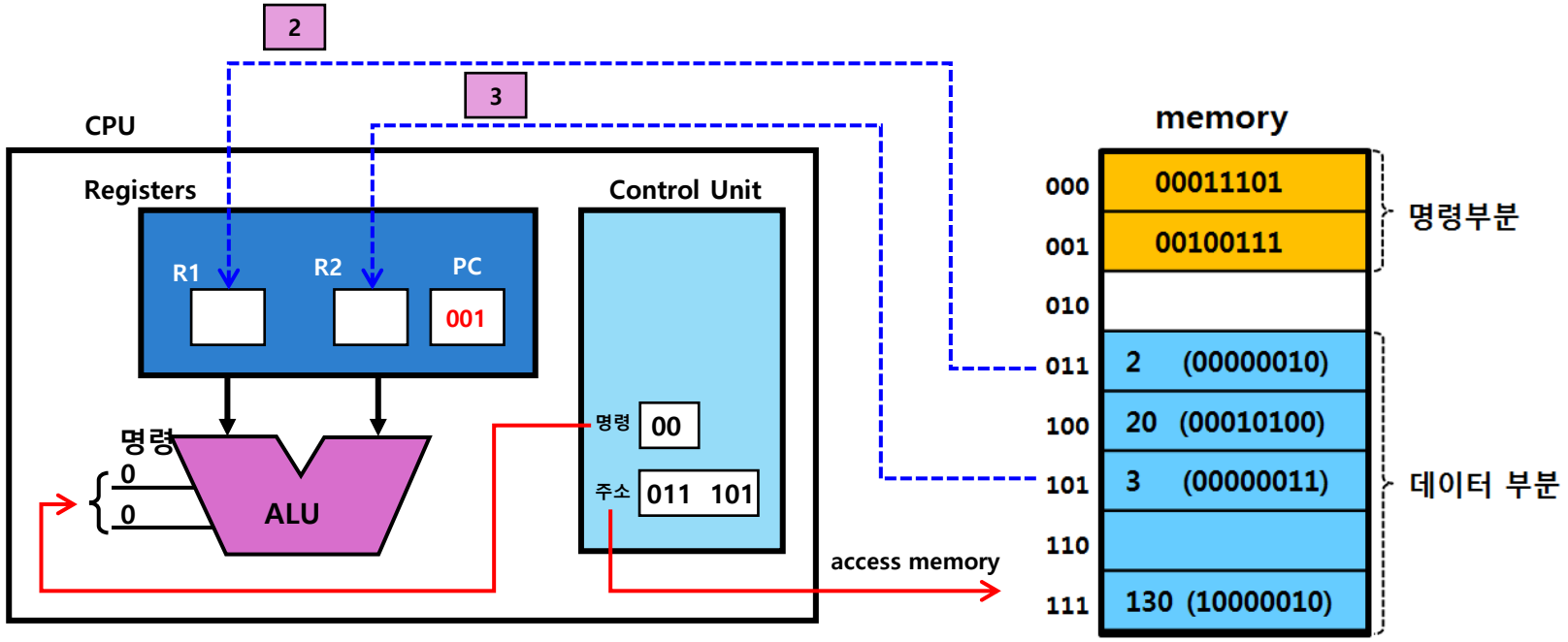
프로그램 실행


HDD
11 011 010
11 101 011
00 011 101
11 100 001
11 111 110
00 100 111
기계어 ➔ Object 파일
Loading
memory
000
001
010
011
100
101
110
111
00011101
00100111
2 (00000010)
1 (00000001)
3 (00000011)
6 (00000110)
명령부분
데이터 부분



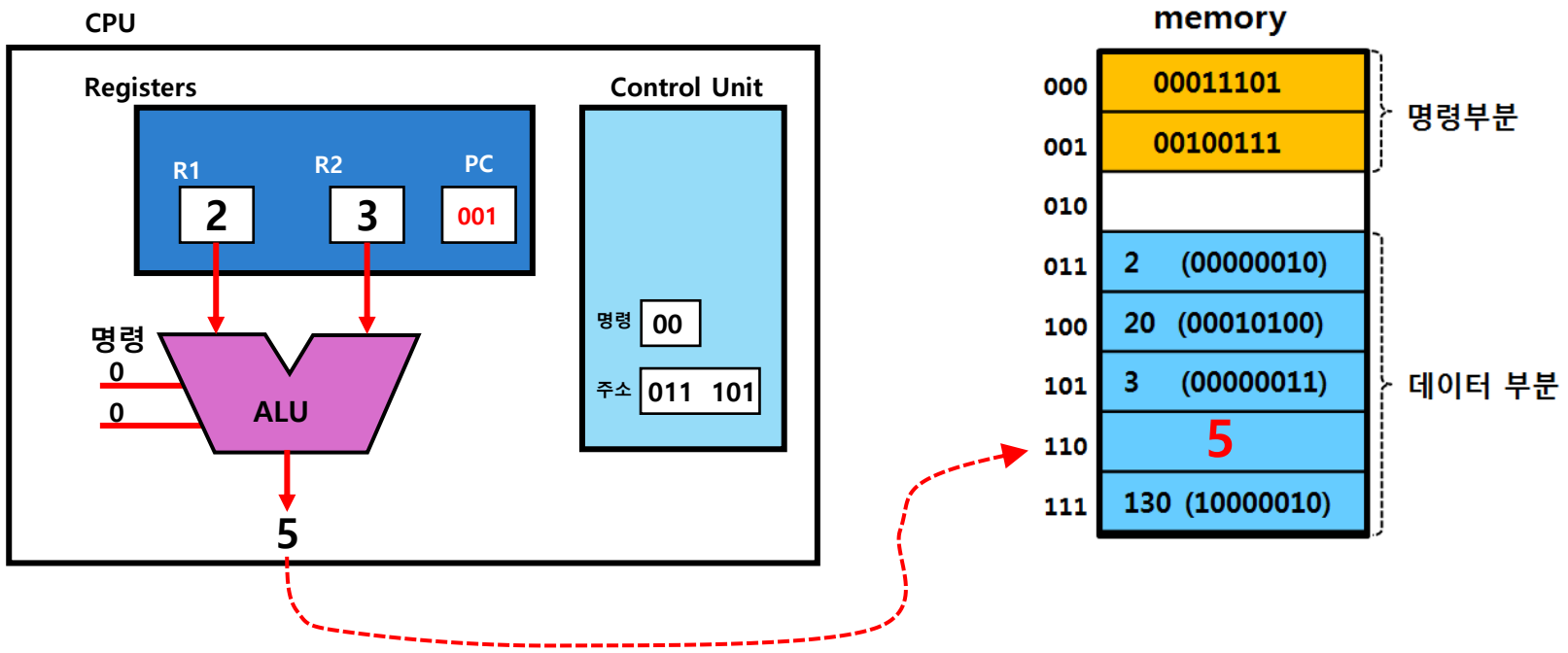
1. PC 레지스터에 있는 주소에서 명령가져오기

2. PC 값 증가 (다음 명령을 가져오기 위함)



1. CU이 명령해석 ➔ ALU에 명령어 셋팅

2. 메모리에서 데이터 가져와 범용레지스터 R1과 R2에 저장



1. 명령실행
2. 계산 결과를 메모리에 저장

실행파일 생성 과정
(라이브러리와 링킹)

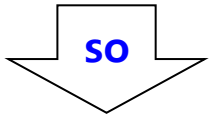
앞의 예의 2+3의 계산 결과 5를 모니터 화면에 출력하려면 어떻게 해야할까?

➔메모리의 내용을 비디오 카드로 보내서 모니터 화면에 출력하도록 프로그램을 작성해야한다.

➔이 작업은 실제 우리가 사용하는 컴퓨터 하드웨어를 모두 알아야 작성할 수 있다.

➔우리는 하드웨어에 대해서 잘 모르니까 화면에 결과를 출력할 수 없다.

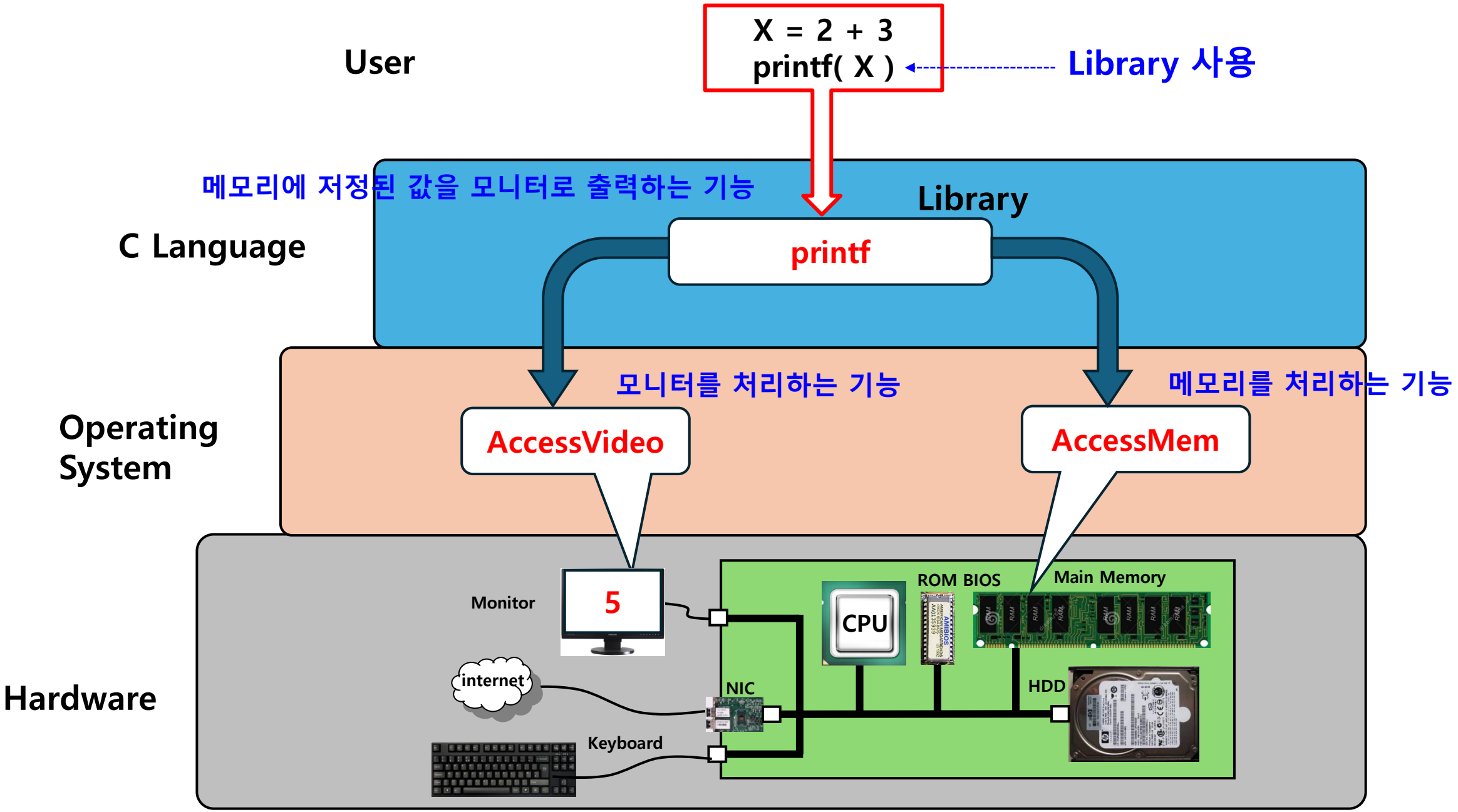
그럼 어떻게 해야하는가?

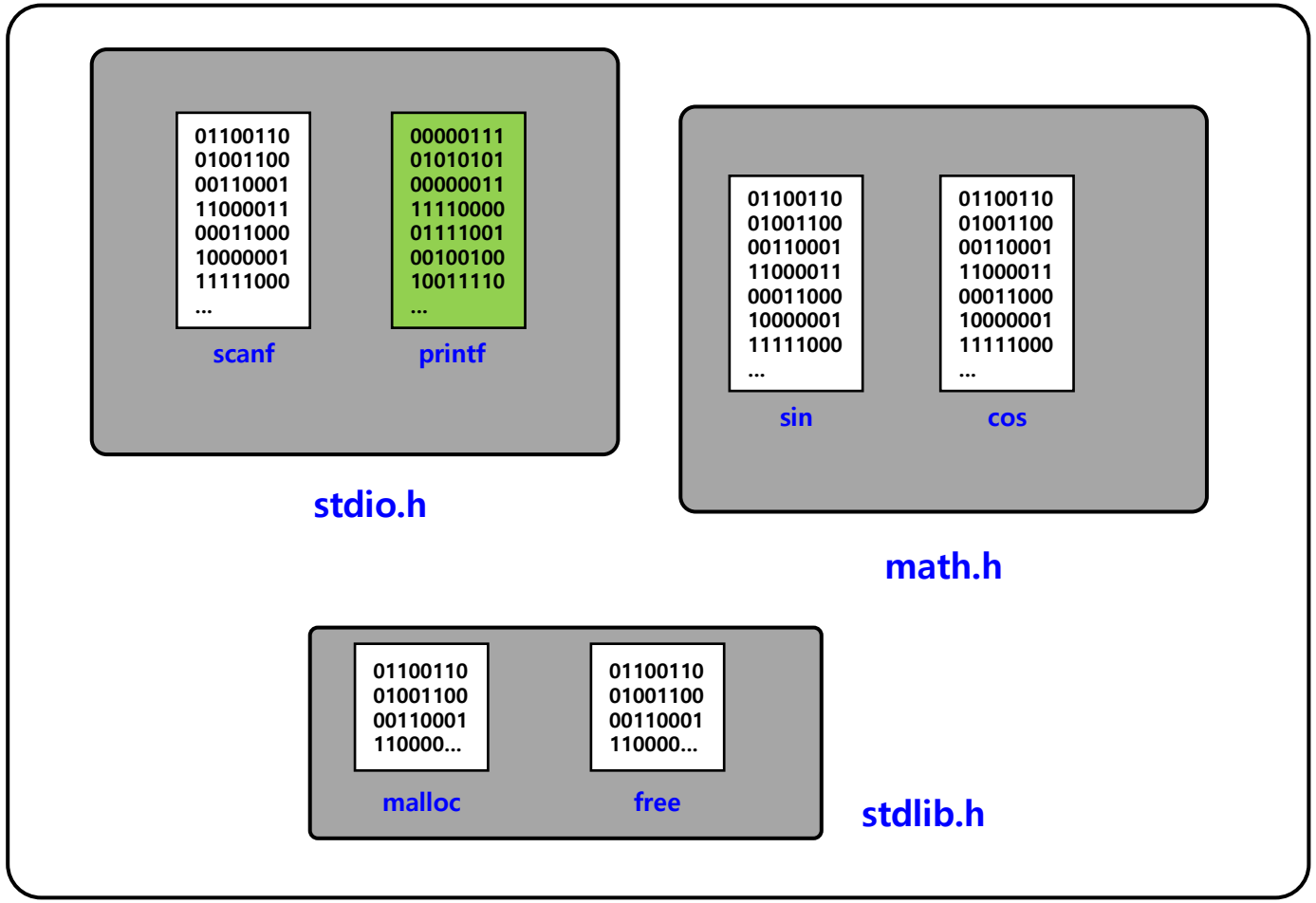


메모리에 저장된 내용을 화면에 출력하는 다른 프로그램을 **연결(link)**해서 이용할 수 있다.

➔ 이럴 때 사용하는 다른 사람이 작성해 놓은 프로그램을 **라이브러리(library)**라고 한다.

➔ 실생활에서 도서관에 가면 다른 사람이 써놓은 책이 있고 그것을 우리가 읽고 지식을 얻는 것과 같은 원리이다.


User
X = 2 + 3
printf(X)
Library 사용
메모리에 저정된 값을 모니터로 출력하는 기능
C Language
Library
printf
모니터를 처리하는 기능
메모리를 처리하는 기능
Operating System
AccessVideo
AccessMem
Hardware
Monitor
5
ROM BIOS
Main Memory
CPU
internet
NIC
HDD
Keyboard

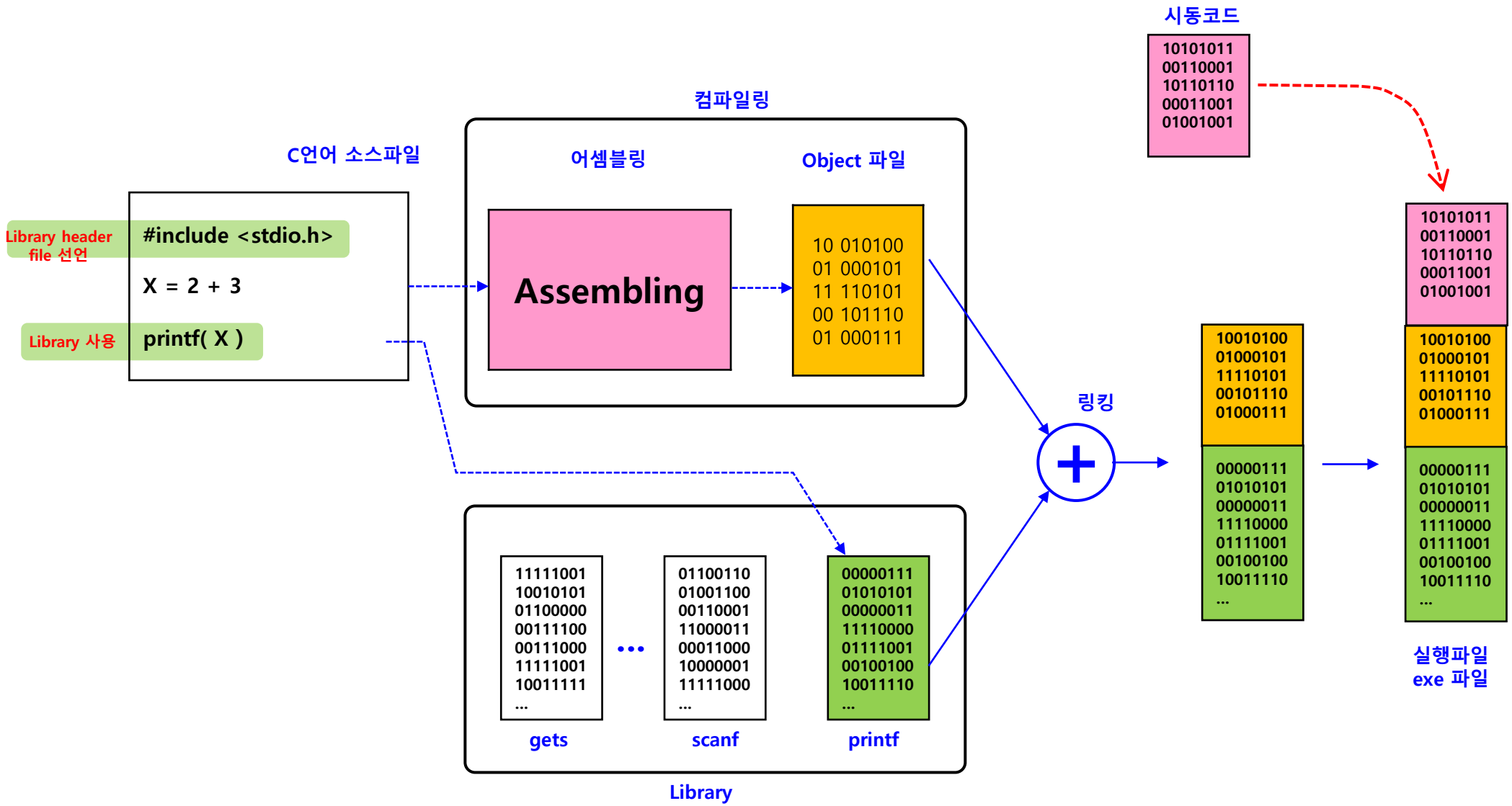


Source Code

Library Header File 선언

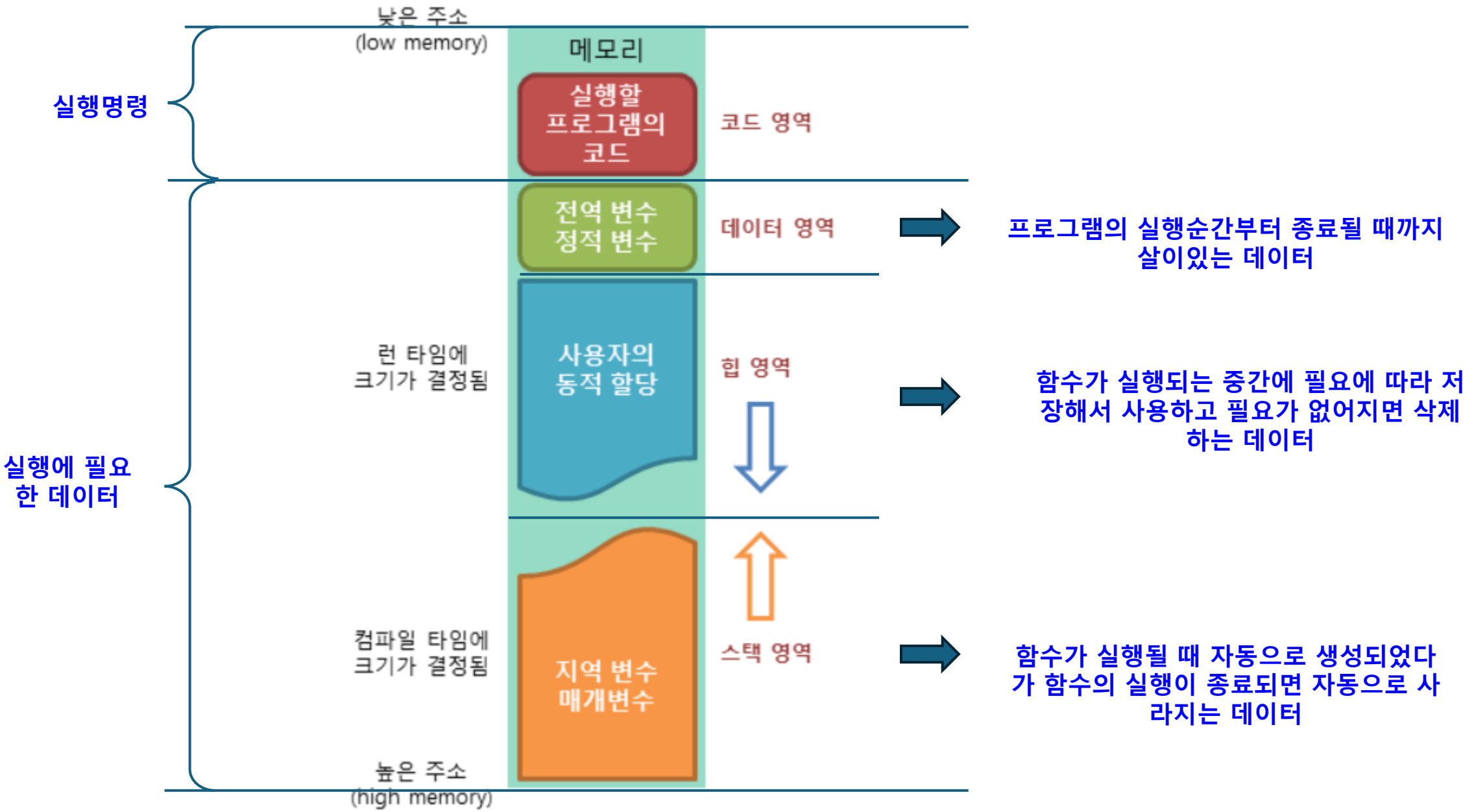
```
#include <stdio.h>

X = 2 + 3

printf( X )
```

Library 함수 사용


시동코드
10101011
00110001
10110110
00011001
01001001
컴파일링
C언어 소스파일
어셈블링
Object 파일
Library header file 선언
#include <stdio.h>
X = 2 + 3
Library 사용
printf(X)
Assembling
10 010100
01 000101
11 110101
00 101110
01 000111
링킹
10010100
01000101
11110101
00101110
01000111
00000111
01010101
00000011
11110000
01111001
00100100
10011110
...
11111001
10010101
01100000
00111100
00111000
11111001
10011111
...
01100110
01001100
00110001
11000011
00011000
10000001
11111000
...
gets
scanf
printf
Library
실행파일
exe 파일

C언어 메모리 모델

(C언어가 메모리를 사용하는 방법)


낮은 주소
(low memory)
메모리
실행명령
실행할
프로그램의
코드
코드 영역
전역 변수
정적 변수
데이터 영역
프로그램의 실행순간부터 종료될 때까지
살이있는 데이터
런 타임에
크기가 결정됨
사용자의
동적 할당
힙 영역
함수가 실행되는 중간에 필요에 따라 저
장해서 사용하고 필요가 없어지면 삭제
하는 데이터
실행에 필요
한 데이터
컴파일 타임에
크기가 결정됨
지역 변수
매개변수
스택 영역
함수가 실행될 때 자동으로 생성되었다
가 함수의 실행이 종료되면 자동으로 사
라지는 데이터
높은 주소
(high memory)