▣ C언어 기초1

1. Compiler
   1) 기계어와 인간이 이해할 수 있는 C언어를 서로 변환해주는 것.
   2) 기계어는 10101001010...과 같이 사람이 직관적으로 이해하기 힘듦.
   3) 컴파일러의 종류 다양하다.

   (SOURCE CODE가 CPU INSTRUCTION을 제어하는 과정)
   4) CPU INSTRUCTION란 각 CPU가 사용하는 기계어를 이야기한다.
      CPU INSTRUCTION는 제조사별로 다르다.
      하지만 어느정도는 ISA라는 공통규격을 갖는다. (Instruction Set Architecture)
   5) 어셈블리어란 이 ISA 를 인간이 이해가능한 언어로 만든것이다.
      어셈블리어는 제조사별로 다르기 때문에 모두 배우기 힘들다.

2. Microprocessor VS Microcontroller
   1) 마이크로 프로세서는 범용적이며 (PC 등) 마이크로 컨트롤러는 기능이 정해져있다.
      (세탁기,비데,청소기 등)
   2) 마이크로 프로세서는 RAM ROM등 주변장치를 사용자가 선정할 수 있다.

3. 아래 소스코드 결과가 126이 아닌이유?

```
#include <stdio.h>

int main(void)
{
    char c= 125;
    c = c+10;

    printf("%d₩n",c)
    return 0;
}
```

   위와같은 코드의 결과값은 126이 아니다.
   이유는 char 의 value range 가 -128~127까지로 정해져 있기 때문이다.

4. 변수(Variable)
   컴퓨터가 데이터를 기억하는 방법
   RAM이라는 공간에 이를 기록한다. (RAM 메모리)

   한 칸 한칸은 1byte
   0x12345678 0x1234567B

한칸은 1byte 즉 -128~127까지 수 데이터가 저장이 가능함.
이때, 기록한 값을 확인하기 위해 저장된 주소를 일일히 조회하려면 힘듦.
따라서 C언어는 변수를 활용해서 임의의 공간에 지정된 크기를 할당해줌.

만약 변수라는것이 없다면 0x12345678 부터 0x1234567B까지 다 불러와서 값을 확인후 8을 더 한후 다시 집어넣어야함.

int count = init_value;
count = init_value + 8;
이렇게 간단히 해결 할 수 있음.

(변수 DATA TYPE)

Keyword	***Variable Type***	***Range***
`char`	Character (or string)	-128 to 127
`int`	Integer	-32,768 to 32,767
`short` `short int`	Short integer	-32,768 to 32,767
`long`	Long integer	-2,147,483,648 to 2,147,483,647
`unsigned char`	Unsigned character	0 to 255
`unsigned int`	Unsigned integer	0 to 65,535
`unsigned short`	Unsigned short integer	0 to 65,535
`unsigned long`	Unsigned long integer	0 to 4,294,967,295
`float`	Single-precision floating point (accurate to 7 digits)	$\pm 3.4 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$
`double`	Double-precision floating point (accurate to 15 digits)	$\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$

5. Type Casting (형변환)
   1) 기존의 데이터 타입을 다른 데이터 타입으로 바꾸는 것을 이야기함.
   2) int 형 변수와 char형 변수를 덧셈 연산할 경우 그 결과는 int형 변수임.
      → 암묵적 형변환 Compiler가 효율을 위해 알아서 형변환 하는 경우.
   3) int a = 1.54;
      float b = (float)a ;
      b의 값은 1.54로 정상 출력됨. 위와같이 변수 옆에 괄호로 원하는
      Data type으로 감싸줌

▣ 폰 노이만 구조 VS 하버드 구조

2020.07.21 KKS

1. 무책임한 설명

1)폰 노이만 구조

① 명령어 메모리와, 데이터 메모리를 공통으로 사용
② 데이터 메모리와 프로그램 메모리 한 버스 사용
③ 데이터에는 고유 의미가 없다.
④ 명령어를 읽을때 데이터에 접근 불가능하다. (병목현상)

2)하버드 구조

① 프로그램과 데이터를 물리적으로 구분하여 각각 다른 메모리에 저장
② 명령어를 읽을때 데이터를 읽거나 쓸 수 있어 성능이 우수함.
③ 버스 시스템이 복잡해 설계가 복잡
④ 디코딩이 간단

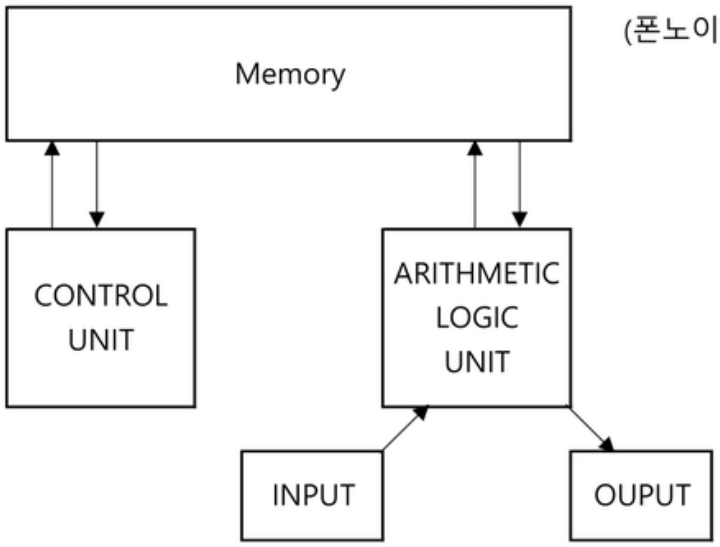
→ 위에 글만 읽었을때 폰노이만과 하버드를 왜 굳이 비교 하였는지
어떤 의미에서 차이점이 중요한지 이해가 가는가? 또 왜 저런차이가 발생하는지
쉽게 이해되는가? NO! 원리를 알아야 한다.

2. Stored Program Concept

1) How computer do 9 * 7 = 63?

우선 컴퓨터가 9 * 7 = 63 이라는 연산을 하는 과정을 알아보자

①곱하기 명령어를 Control Unit이 Meomory로부터 가져온다(Fetch)
②Control Unit이곱하기 명령어가 무엇인지 해석한다. (Decode)
③ALU가곱하기를 실행한다. (Execute)
④값을 메모리에 저장한다.(Store)



(폰노이만 구조)

위 연산을 매우 빠르게 하려면 어떻게 할까?
단순히 속도를 올려도 된다. 하지만 반도체 공학적인 제약, 생산과 관련된 문제점 등 의 이유로 불가능하다. 그렇다면 어떻게 해야할것인가?

2. Pipelining Concept

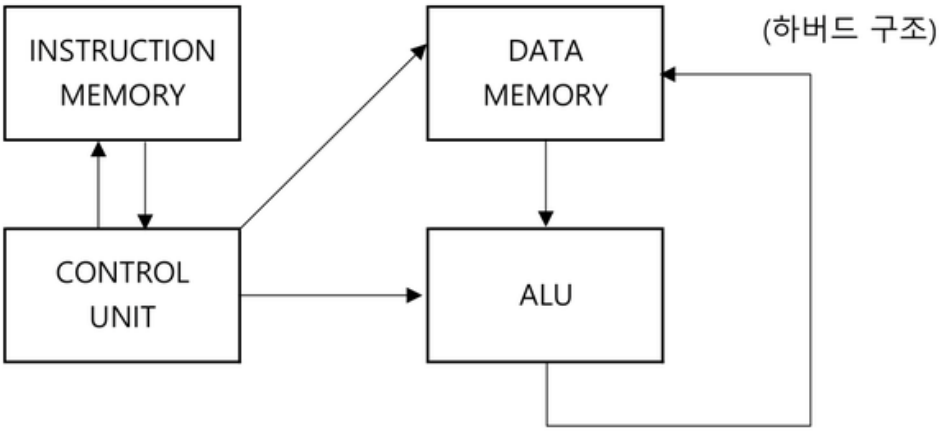
FETCH	DECODE	EXCUTE	STORED	-	-	-
-	FETCH	DECODE	EXCUTE	STORED	-	-
-	-	FETCH	DECODE	EXCUTE	STORED	-
-	-	-	FETCH	DECODE	EXCUTE	STORED

위 그림을 봤을때 어떤 생각이 드는가?
나는 제품 생산 공장 라인이 생각난다
누군가는 볼트만을 계속해서 조립하고 누군가는 포장만 하고 누군가는
제품을 쌓기만 하는 그런 공장 말이다.

Fetch 단계에서 Memory 에서 CU로 명령어를 전달할때 ALU는 논다.
이때 Pipelining구조를 사용하면 놀고있는 ALU를 쥐어 짤 수 있다.

문제점은 폰노이만 구조에서는 Memory가 decode빼고 다 관여한다는점이다.
따라서 Pipelinijng구조를 사용하기 위해 Memory를 물리적으로 분리한다.
① Instruction Memory ← Fetch 전용
② Data Memory ← Store/Excute전용
왜 Store하고 Excute는 동시에 사용가능한가? 메모리 특성상
쓰는것을 어렵고 읽는것은 간단하다.
메모리 주소만 겹치지 않는다면 동시 수행가능하다.

2. Harvard Architecutre



위와 같은 하버드 구조를 사용하면 Pipelining 구현이 가능하며 속도를
증가시킬 수 있다.

Q.1 C코드는 위에서부터 한줄씩 내려오는 순차언어인데
이때 하버드 구조가 제 역할을 할 수 있는것인가?