임베디드스쿨_1기_김인겸_C_1주차_HW (2020.7.23)

강의 내용 정리

1. What is C Programming

1) c언어를 배우는 이유
컴퓨터는 10101과 같은 이진수만을 정보로 받아들이고 처리한다. 이런 방법을 기계어라고 한다. 하지만 우리가 직접 0과 1로만 코딩을 하기에는 너무 힘들기 때문에 컴파일러라는 것을 개발해 인간과 기계 사이의 언어 사이의 장벽을 줄였다. 컴파일러는 통역가로 생각하면 이해하기 쉽다. 사람이 보기 쉬운 방법으로 코딩을 하면 컴파일러가 이를 컴퓨터가 해석 가능한 0과1의 정보로 바꾸어 준다. 이로써 우리는 더 쉬운 방법으로 프로그래밍을 할 수 있게 되었다.

2) CPU Instruction(ISA)이란?
우리가 작정한 소스코드를 컴파일러가 기계어로 바꾸어 줄 때 그냥 무작위의 0과1로 바꿔놓는 것이 아니라 각각의 CPU에 맞는 포맷으로 바꾸어 준다. Intel CPU와 ARM CPU가 기계어를 처리하는 ISA방식이 다르다. ARM CPU(32비트)를 예를들면, 각각의 비트 자리마다 차지하는 역할이 있다(Cond, Opcode, Operand 등) CPU내부의 제어장치,레지스터,ALU에 포함된 이진수 정보들을 전부 ISA 포맷으로 불러들여와서 각각의 역할에 맞는 비트자리에 정보들을 입력해서 컴퓨터가 순차적으로 0과 1의 정보들을 처리함으로써 컴퓨터가 동작한다. 이처럼 ISA는 컴파일러가 통역해준 기계어를 실제로 처리하는 역할을 한다.

3) 어셈블리어
기계어와 1:1로 대응되는 언어이다. ISA와 더 밀접한 관련이 있다.

4)마이크로프로세서와 마이크로컨트롤러
- 마이크로프로세서 : 마이크로프로세서는 혼자서 동작하지 않고 다양한 장치들과 결합되어 사용된다. 컴퓨터에서는 CPU라고 부르며 마이크로컨트롤러보다 더 다양한 기능을 수행할 수 있다
- 마이크로컨트롤러 : 마이크로컨트롤러는 혼자서 자체적으로 동작한다. 그리고 특정한 목적을 위해서 사용되는 장치이다. 예를들어, 세탁기, 전자레인지, 냉장고 내부에 포함된 마이크로컨트롤러는 각각 세탁기를 구동시키고, 음식을 데우고, 냉장고의 온도를 조절하는 등 특정한 목적을 위해 사용된다. 전원만 있으면 단순한 동작들을 마이크로컨트롤러만으로 구현할 수 있다. 마이크로컨트롤러는 마이크로프로세서, ROM, RAM, 입출력장치로 구성되어 있으며 이때의 마이크로프로세서를 코어라고 부르고 일반적인 컴퓨터에 사용되는 것보다 성능이 많이 떨어진다.

2. C Programing Variable

1) 변수는 왜 배워야 할까?
변수 선언은 메모리에 공간을 부여하는 방식이고 변수는 메모리에 들어갈 값을 표현하는 메모리 주소이다. Data type에 따라 변수가 들어갈 메모리 공간의 범위가 달라진다.

2) 부동소수점수
-32bit Sigle Precision(float형)
맨앞비트는 부호를 표현
지수부 8비트와 가수부 23비트의 곱으로 십진수를 이진수로 표현(소수점 6자리까지 유효)
-64bit Double Precision(duoble형)
맨앞비트는 부호를 표현
지수부 11비트와 가수부 52비트의 곱으로 표현(소수점15자리까지 유효)

3. C Programing Datatype / type Castng

1) int, char, float, duoble 등의 데이터타입이 표현할 수 있는 비트수와 max값, min값에 대해 알아보았다.

2) 형변환
int로 선언된 i와 char로 선언된 c를 더하는 연산을 할 때 컴파일러가 내부적으로 c를 int형으로 형변환을 해준다. 이처럼 컴파일러가 상황에 맞게 알아서 형변환을 해줄 수도 있고 사용자가 직접 형변환을 해줄 수도 있다.
형변환이 사용되는 예로는 키보드를 들 수 있다. 우리가 입력하는 키보드 문자가 아스키코드에 의해 이진수로 형변환 되고 다시 출력할 때는 문자로 형변환이 된다.

C의 결과값은? 그리고 왜 128이 아닌 걸까?

```c
#include <stdio.h>

int main()
{
    char c = 127;
    c = c + 1;

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

```
ingyeomk
-128
```

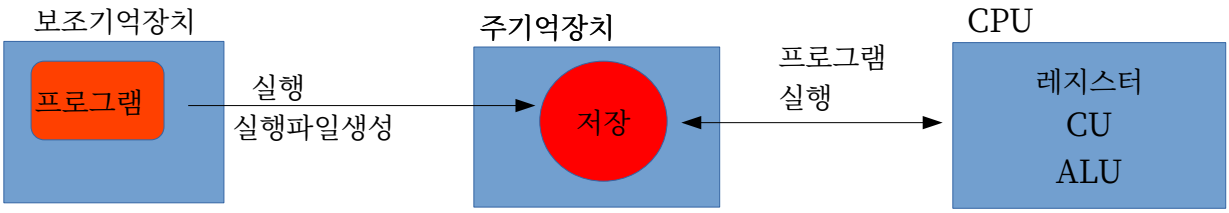
char형은 8비트로 표현되는데 표현할 수 있는 값의 범위가 -128부터 127까지다.
8비트에서 127+1을 이진수로 해보자
01111111 + 00000001 = 10000000. 맨 앞 비트가 1(음수)를 의미하므로 2의보수를 이용해 해석하면 -128이다.

이와 같은 현상이 발생하는 이유는 char형이 다룰 수 있는 비트를 넘어선 숫자가 사용돼 오버플로우를 일으켰기 때문이다.

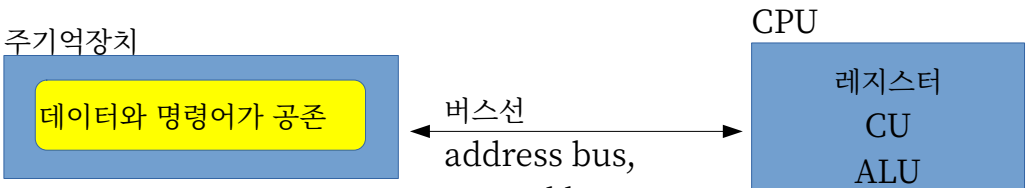
폰노이만 구조와 하버드 구조

1. 프로그램이 시작되는 과정(과거)
옛날에는 프로그램을 시작하려면 펀치카드에 직접 0과1의 정보를 새겨넣고 수백장의 펀치카드를 컴퓨터에 넣어줌으로써 정보를 처리하여 프로그램을 실행시켰다.

2. 프로그램이 시작되는 과정(현대)
현대 컴퓨터는 보조기억장치(HDD,SSD)에 프로그램을 저장한 뒤 실행을 시키면 전처리기>컴파일러>어셈블러>링커 순서로 실행파일이 생성된다. 그리고 이 실행파일을 구성하는 명령어들을 주기억장치인 RAM으로 불러들여온다. RAM으로 불러들인 정보를 CPU가 처리하여 프로그램이 실행된다.



3. 폰노이만 구조
폰노이만 구조는 위에 설명했던 것처럼 프로그램을 실행할 때 더이상 외부적인 요소에 의존하지 않고 컴퓨터 내부에 기억장치를 두어 프로그램을 실행할 수 있게 하는 방식이다. 이를 내장메모리 순차처리 방식이라고 한다.



데이터 : 이진수로 된 숫자정보
명령어 : LOAD, ADD, SOTRE 등

-CPU가 일하는 방법
CPU는 기억장치로부터 명령어를 CU로 불러들이고(Fetch)
CU는 명령어를 해석하고(Decode)
ALU가 데이터를 계산하고(Execute)
결과를 메모리에 저장한다(Store).

이렇게 프로그램이 메모리에 내장되어 있으므로 '내장메모리'인 것이고 순차적으로 정보를 처리하기 때문에 '순차처리' 방식인 것이다.

하지만 폰노이만 구조에는 문제점이 있다. 하나의 메모리에 명령어와 데이터를 저장하고 한번에 순차적으로 처리하기 때문에 CPU가 명령어를 처리하는 속도를 메모리가 따라가지 못하는 것이다. 이렇게

메모리를 읽고 쓰는 동안 CPU는 아무런 일도 하지 않게 되어 효율이 떨어지는데 이를 병목현상이라고 한다.

4)하버드 구조
하버드 구조는 폰노이만 구조의 문제점을 해결하기 위해 고안된 방식이다.

주기억장치 안에 명령어 메모리와 데이터 메모리를 구분해서 저장하고 각각의 구분된 버스선을 이용해 동작하는 방식이다. CPU가 명령어를 처리하는 동안 데이터를 읽고 쓰는 것이 가능하므로 효율이 크게 향상된다. 다만 회로가 복잡하고 비싸다는 단점이 있다.

+) 나중에 배울 AVR은 하버드구조를 가지는 고성능 8비트 마이크로컨트롤러이다.