2020.07.18
수업 복습

임베디드스쿨1기
Lv1과정
2020. 07. 21
권 영근

C언어는 왜 배울까?

⊙ 사람들이 0, 1 두 개의 수들로만 이루어진 기계어를 보고 이해하기가 어렵습니다.
그렇기 때문에 사람들이 이해하기 쉬운 언어를 개발할 필요가 있었습니다.

그렇게 해서 C언어가 만들어지게 되었으며 사람들(특히 프로그래머들)은 이러한 C언어를 배움으로서 컴퓨터에게 그들이 원하거나 의도한 작업을 하도록 지시 혹은 명령을 할 수 있게 되었습니다.

◆ 기계어('https://blog.naver.com/ktmgame' 에서 발췌)

001001 11101 11101 1111111111111000
001000 00001 00000 0000000000001010

◆ C언어

```
#include <stdio.h>

int main(void){
    return 0;
}
```

그럼, Compiler는 뭘하는 걸까?

⊙ 사람들에게 친숙한 C언어를 컴퓨터는 정확하게 이해하지 못합니다. 마치 사람들이 0, 1 들로만 이루어진 기계어(컴퓨터가 이해할 수 있는 언어)를 보고 잘 이해하지 못하는 것처럼 말입니다.

실제 현실 속에 있는 일을 한번 보겠습니다. 한국어를 배운 한국 사람들은 (배우지 않은 이상) 영어를 말하거나 이해하지 못합니다. 이와 마찬가지로 영어를 배운 미국 사람들은 (역시 배우지 않은 이상) 한국어를 말하거나 이해하지 못합니다.

하지만 한국어와 영어 모두에 능통한 번역가가 있다면 한국 사람과 미국 사람은 서로 소통하는 것이 가능합니다. 한국 사람의 말을 번역가가 영어로 번역하여 미국 사람에게 전달하면 됩니다.

이와 비슷하게, 사람들에게 친숙한 C언어와 기계어 사이에 어떤 번역 프로그램이 있으면, 그 프로그램이 C언어를 변환하여 기계어로 바꾸어 컴퓨터에게 전달하면 됩니다. 한국 사람과 미국 사람 사이에 있는 번역가처럼 말입니다. 그렇게 하면 사람들은 컴퓨터에게 그들이 원하는 일을 할 수 있게 만듭니다.

이러한 번역 프로그램을 우리는 '컴파일러' 라고 합니다.

CPU instruction이란?

⊙ 한국 사람이 사용하는 언어에는 한국어가 있습니다. 미국 사람이 사용하는 언어에는 영어가 있습니다. 중국 사람이 사용하는 언어에는 중국어가 있으며 일본 사람이 사용하는 언어에는 일본어가 있습니다. 각 나라 사람들에게 대개 그들이 사용하는 고유한 언어들이 있습니다.

CPU의 세계도 이러한 면에서 사람들의 세계와 비슷합니다. X86 아키텍처에서 사용하는 고유 언어(기계어)가 다르고 ARM 아키텍처에서 사용하는 고유 언어가 다릅니다. MIPS 아키텍처에서 사용하는 고유 언어 역시 다르며 Xtensa 아키텍처에서 사용하는 고유 언어 또한 다릅니다.

여기에서 각 CPU들이 사용하는 고유 언어(기계어)를 우리는 CPU Instruction이라고 합니다. 그리고 다른 표현으로 ISA(Instruction Set Architecture)가 있습니다. 이 표현이 많이 쓰이는 듯 합니다.

ISA를 또 언어로 만든 게 어셈블리어?

◉ 네이버 지식 백과에서 '저급 언어' 를 검색한 후 그 내용을 참고하였습니다.

C언어만큼은 아니지만 기계어보다는 사람들이 이해하기 쉽도록 기계어에 1:1 대응이 되는 언어를 만들었는데 이를 '어셈블리어' 라고 합니다.

기계어는 CPU마다 천차만별로 다릅니다. 어셈블리어는 기계어에 1:1 대응이 되는 언어입니다. 따라서 어셈블리어 또한 CPU마다 천차만별로 다릅니다.

하드웨어에 가까운 언어일수록 하드웨어에 직접적인 접근을 하기가 수월합니다. 어셈블리어는 C언어보다 하드웨어 접근에 매우 유리한 언어이기 때문에 수행 속도가 매우 빠릅니다. 그래서 메모리 용량이 한정된 임베디드 시스템에서 프로그램 최적화를 위하여 적지 않은 빈도로 사용됩니다.

Microcontroller는 뭘까?

◉ NAVER 지식 백과에서 발췌하였습니다.

◆ 마이크로프로세서

연산을 미리 확립된 순서에 의해 체계적으로 실행할 뿐만 아니라 컴퓨터의 각 장치에 제어 신호를 제공하는 제어장치를 1개의 작은 실리콘 칩에 집적시킨 초대규모 집적회로로 이루어진 처리장치이다. 내부는 산술논리연산기, 레지스터, 프로그램 카운터, 명령 디코더, 제어회로 등으로 구성되어 있다. 마이크로프로세서는 주기억장치에 저장되어 있는 명령어를 인출하여 해독하고, 해독된 명령어를 실행하며 실행 결과를 다시 주기억장치에 저장할 수 있는 기능 등을 자동으로 수행함과 동시에 입출력 장치들과도 데이터 교환을 수행한다.

[네이버 지식백과] 마이크로프로세서 [microprocessor] (두산백과)

한국어 위키백과에서 발췌하였습니다.

◆ 마이크로컨트롤러

마이크로컨트롤러(Microcontroller) 또는 MCU(Micro Controller Unit)는
마이크로프로세서와 입출력 모듈을 하나의 칩으로 만들어 정해진 기능을 수행하는
컴퓨터를 말한다.

CPU 코어, 메모리 그리고 프로그램 가능한 입/출력을 가지고 있다. NOR 플래시 메모리, EPROM 그리고 OTP ROM등의 메모리를 가지고 있어 정해진 기능을 수행하도록 프로그래밍 코딩하고 이 기계어 코드를 써 넣는다. 기계어 코드가 실행되기 위한 변수나 데이터 저장을 위해 적은 용량의 SRAM을 가지고 있다. 기타 칩에 따라 EEPROM을 내장하기도 한다.

마이크로프로세서는 산술 논리 장치(Arithmetic Logic Unit, ALU)와 레지스터 그리고 제어 회로(Control Logic) 등으로 이루어졌고 미리 설계한 어떤 알고리즘에 따라서 순차적으로 작업을 하는 처리 장치입니다.

그리고 마이크로컨트롤러는 마이크로프로세서와 각종 기억 장치(ROM, RAM, 플래시 메모리, ...), 페리페럴(UART, I2C, SPI, ADC, ...) 등으로 이루어진 장치입니다.

변수는 왜 배워야 할까?

⊙ 변수는 프로그래머들이 프로그램을 쉽게 조작할 수 있도록 만든 메모리 영역의 지정된 이름입니다. 즉 메모리를 대표합니다.

특히 C언어에서는 각 변수들마다 특정 유형이 있는데 이를 Data type이라고 합니다. 이 Data type은 변수 메모리 크기를 결정을 합니다.

◆ char

1 바이트로 정수 type입니다.

◆ int

4 바이트로 역시 정수 type입니다.

◆ float

32 비트 표시형 부동 소수점 값을 가집니다.

◆ double

64 비트 표시형 부동 소수점 값을 가집니다.

◆ void

어느 type에도 속하지 않는 type입니다.

변수의 이름은 문자, 숫자, 밑줄로 구성될 수 있습니다. 다만 시작하는 부분에서는 문자나 밑줄이 있어야 합니다.

C언어에서는 대문자와 소문자를 구별합니다. 이를 염두에 두고 변수의 이름을 작성하면 됩니다.

Single / Double Precision

◉ 다음과 같은 float형 변수 값이 있다고 가정해보겠습니다.

00111110001000000000000000000000

float의 크기가 32 비트이기 때문에 32개의 비트로 이루어졌고 각 비트 안에는 0 혹은 1이 적혀져 있습니다.

이 float 변수는 다음과 같은 구조로 이루어졌습니다. 먼저 맨 왼쪽의 첫 번째 비트는 실수의 부호를 나타냅니다. 0이면 양수, 1이면 음수입니다.

그 다음 왼쪽에서 2번째 비트부터 왼쪽에서 9번째 비트까지는 지수 부분을 나타냅니다.

마지막으로 왼쪽에서 10번째 비트부터 왼쪽에서 마지막 비트까지는 가수 부분을 나타냅니다.

‘/’ 기호를 사용하여 구분을 해보겠습니다.

0/01111100/01000000000000000000000

이제 부호 부분과 지수 부분, 가수 부분들을 활용하여 float 변수 안의 값을 구해보겠습니다

부호가 양수이므로 일단 float 변수 안의 값에는 '-' 가 붙지 않습니다.

예시에서 지수는 '01111100' 인데 이는 2진수입니다. 이 2진수의 지수를 10진수로 변환을 합니다. 그러면 '124' 라는 숫자를 얻게 됩니다. 이 124에서 127을 뺍니다. 그러면 -3이 되는데 이 -3을 지수로 삼아서 2의 거듭제곱 값을 구하십시오. 그러면 1 / 8, 즉 0.125가 됩니다.

가수 부분의 맨 왼쪽 비트 부분부터 시작을 하여 오른쪽으로 나아가 보겠습니다. 맨 왼쪽 비트 값에 1 / 2를 곱합니다. 그리고 다음 비트 값에 2의 제곱의 역수인 1 / 4를 곱합니다. 그리고 그 다음 비트 값에 2의 세제곱의 역수인 1 / 8을 곱합니다. 이러한 규칙으로 마지막 비트까지 2의 거듭제곱 값의 역수를 곱합니다.

예시의 경우 가수 부분에서 맨 왼쪽에서 2번째 비트 부분만 1이고 나머지는 0입니다. 따라서 위 구절에서 말한 규칙대로 계산을 하면 1 /4의 값을 얻는데 이는 0.25입니다. 그리고 이 0.25에 1을 더합니다. 그러면 1.25가 됩니다.

그리고 지수 부분에서 얻은 값 0.125와 가수 부분에서 얻은 값 1.25를 서로 곱합니다. 그 결과 양수 0.15625를 얻게 됩니다. 이 값이 예시 속의 float 변수 안에 있는 값을 10진수로 나타낸 값입니다.

64 비트의 double 변수의 값을 10진수로 바꾸는 방법은 32비트의 float 변수의 값을 10진수로 바꾸는 방법과 같은 방법입니다. 다만 비트 수가 달라졌기 때문에 지수 부분과 가수 부분이 차지하는 비트 수가 float 변수의 그것보다 많아졌을 뿐입니다.

참고로 double 변수 값에서 부호를 결정하는 것은 맨 왼쪽의 첫 번째 비트입니다. 이는 float 변수에서 부호를 결정하는 비트의 형식과 크기가 같습니다.

지수 부분은 부호 비트 다음의 비트부터 시작을 해서 맨 왼쪽으로부터 12번째 비트까지입니다.

가수 부분은 맨 왼쪽으로부터 13번째 비트부터 맨 오른쪽 비트까지입니다.

printf & scanf

⊙ 순차적인 C언어에서 출력과 입력을 담당하는 함수들이 있는데, 각각 ‘printf’ 와 ‘scanf’ 입니다. 이 2개의 함수들을 사용하기 위해서는 헤더 파일 ‘stdio.h’ 가 필요합니다.

◆ printf 포맷

1. %d는 char, int 등에 대응하는 정수를 출력합니다.

2. %ld는 long 변수형의 값을 출력합니다.

3. %lld는 long long 변수형의 값을 출력합니다.

4. %f는 float, double 등의 변수형의 값을 출력합니다.

5. %lf는 long double 변수형의 값을 출력합니다.

6. %c는 char, short, int에 대응하는 문자를 출력합니다.

7. %s는 문자열을 출력합니다.

8. %p는 포인터의 주소 값을 출력합니다.

◆ scanf 포맷

1. %d는 char, int 변수를 입력받습니다.

2. %ld는 long 변수를 입력받습니다.

3. %lld는 long long 변수를 입력받습니다.

4. %f는 float 변수를 입력받습니다.

5. %lf는 double 변수를 입력받습니다.

Datatype

◉ Datatype은 변수에 따라서 메모리에서 차지하는 공간의 양을 결정합니다.

◆ 변수의 크기와 범위

1. char 변수의 크기는 1 바이트이고 -128~127 혹은 0~255의 값을 가지게 됩니다.

2. unsigned char 변수의 크기는 1바이트이고 범위는 0~255입니다.

3. signed chat 변수의 크기는 1 바이트이고 범위는 -128~127입니다.

4. int 변수의 크기는 2 혹은 4바이트(CPU마다 다릅니다.)이고 범위는 -32768~32767 혹은 -2147483648~2147483647입니다.

5. unsigned int 변수의 크기는 2 혹은 4 바이트(CPU마다 다릅니다.)이고 범위는 0~65535 혹은 0~4294967295입니다.

6. short 변수의 크기는 2바이트이고 범위는 -32768~32767입니다. Long 변수의 크기는 8 바이트이고 범위는 -9223372036854775808~9223372036854775807입니다.

7. float 변수의 크기는 4 바이트이고 범위는 1.2E-38~3.4E+38입니다. Double 변수의 크기는 8 바이트이고 범위는 2.3E-308~1.7E+308입니다. 즉 이 두 변수들의 범위에는 엄청나게 큰 실수들이 있습니다.

Type Casting

◉ 기존 Data Type을 다른 Data Type으로 바꾸는 것입니다.

키보드 키 'A' 를 누른다고 가정해보겠습니다. 이 A가 담겨져 있는 변수의 형태는 'char' 인데 정수 값을 저장하는 변수입니다. 그런데 ASCII 코드를 통하여 A에 해당되는 정수 값이 문자로 변환이 됩니다. 정수 형태의 값이 문자 형태의 값으로 바뀌져 A가 출력되었습니다. 이는 형 변환의 한 사례입니다.

그리고 다른 범위를 가지는 서로 다른 변수들을 합하는 과정에서도 형 변환이 일어납니다. 예를 들어서 int 변수 'x' 와 long 변수 'y' 를 더한다고 가정을 해보겠습니다. 서로 다른 자료형들끼리 연산을 할 수 없기 때문에 8바이트의 long 변수보다 크기가 작은 int 변수 x 는 long 변수로 형 변환이 이루어지고 y와 합해지게 됩니다. 서로 다른 변수의 덧셈 과정에서 크기가 작은 변수가 크기가 큰 변수로 형 변환이 이루워집니다.

좀 더 정리를 해보자면, int → unsigned int → long → unsigned long → long long → unsigned long long → float → double → long double의 순서로 형 변환이 이루어지게 됩니다. 오른쪽으로 갈수록 더 큰 변수입니다.

<Stdint.h>

◉ ‘stdint.h’ 라는 헤더 파일에는 여러 가지 사용자 정의의 변수들이 있습니다. 그것들 중에서 기억해두면 도움이 되는 변수들이 있는데 이에 대해 간략히 작성해보겠습니다.

1. typedef unsigned char uint8_t

2. typedef unsigned short uint16_t

3. typedef unsigned int uint32_t

4. typedef unsigned long long uint64_t

사용자 정의 변수들이 간결하게 표시되고 한 눈에 이해하기 쉬운 속성을 가지고 있습니다.

HW0-
폰노이만 구조와
하버드 구조

임베디드스쿨1기
Lv1과정
2020. 07. 22
권 영근

전형적인 임베디드 시스템 구조

◉ 인터넷 사이트 'https://blog.naver.com/mmwook94/221507034985' 를 참고하였습니다.

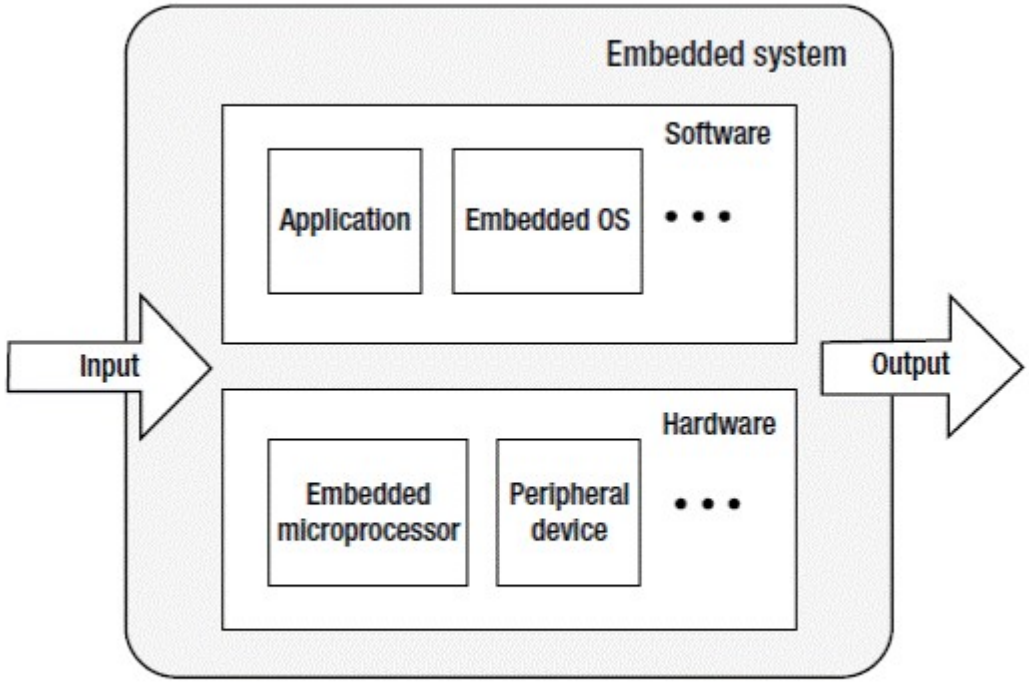


Figure 1-2. Basic architecture of an embedded system

(임베디드 하드웨어는) 주로 프로세서, 메모리, 버스, 페리페럴 장치들, 입출력 포트들 그리고 다양한 컨트롤러들을 포함합니다. 임베디드 소프트웨어는 대개 임베디드 운영체제와 다양한 어플리케이션들을 포함합니다.

전형적인 하드웨어 구조

◉ 기본적인 컴퓨터 시스템 성분들-마이크로프로세서, 메모리 그리고 입출력 모듈들-은 모든 부분들이 통신을 하고 프로그램을 실행하기 위해서 서로 연결되어 있습니다(사진 1-3을 보세요.).

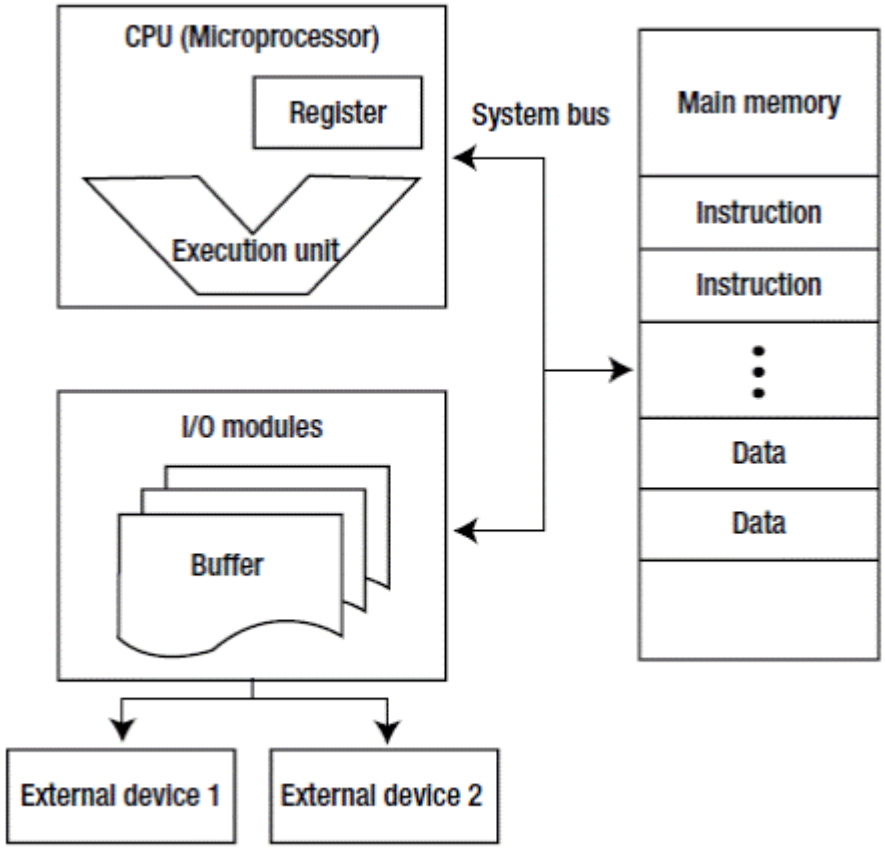


Figure 1-3. Computer architecture

임베디드 시스템의 마이크로프로세서는 범용 컴퓨터에서의 CPU에 대응합니다. 그리고 두 장치들 모두 컴퓨터 작동을 제어하고 명령어를 실행하고 데이터를 처리합니다.

메모리는 명령어와 데이터를 저장을 하는데 사용이 됩니다.

입출력 모듈들은 프로세서, 메모리 그리고 외부 장치들 사이에서 데이터 교환을 책임집니다.

외부 장치들은 부차적인 저장 장치들(플래시 메모리와 하드 디스크같은 것들), 통신 장비 그리고 터미널 장비들을 포함합니다.

시스템 버스는 프로세서, 메모리 그리고 입출력 모듈들에게 데이터와 제어 신호 통신 및 송신을 제공합니다.

임베디드 시스템에 적용이 되는 기본적인 두 종류의 구조들이 있습니다: 폰노이만 구조와 하버드 구조들입니다.

폰노이만 구조

◉ 폰노이만 구조(또한 프린스턴 구조로도 알려져 있습니다.)는 John von Neuman에 의해 처음으로 제안이 되었습니다. 이 구조의 가장 중요한 특징은 소프트웨어와 데이터가 같은 메모리를 사용한다는 것입니다: 즉 “프로그램이 데이터이고 데이터가 프로그램.” 입니다 (사진 1-4에서 볼 수 있습니다.).

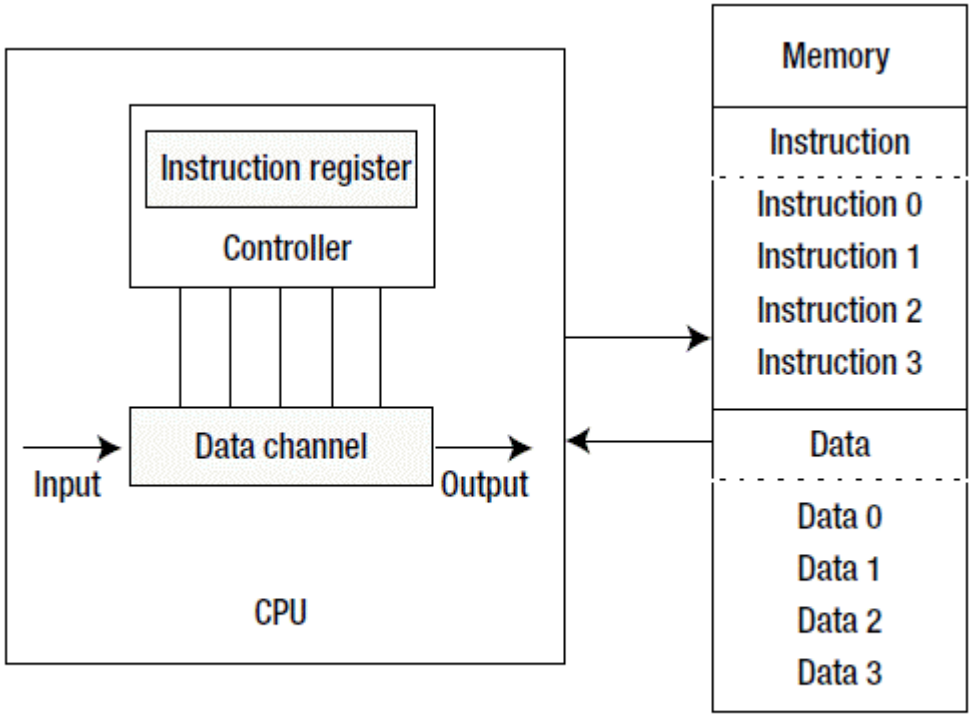


Figure 1-4. Von Neumann architecture

폰노이만 구조에서 명령어와 데이터는 같은 버스를 공유합니다. 이 구조에서 정보 송신은 컴퓨터 실행 중 병목 현상에 부딛히게 되고 데이터 처리 속도는 영향을 받게 됩니다; 그래서 이것은 가끔 ‘폰노이만 병목 현상’ 이라고 불리어 집니다. 실제로는 캐시 메모리와 분기 예측 기술로 이 문제를 효과적으로 해결할 수 있습니다.

하버드 구조

◉ 하버드 구조는 처음으로 Harvard Mark I 컴퓨터의 이름을 따왔습니다. 폰노이만 구조와 비교를 해서 하버드 구조 프로세서는 2개의 두드러진 특징들을 가집니다. 첫 번째, 명령어들과 데이터가 각각 2개의 분리된 메모리 모듈들에 저장이 됩니다; 명령어들과 데이터는 같은 모듈에서 공존하지 않습니다. 두 번째, CPU와 메모리 사이에 2개의 독립적인 버스들이 전용 통신 통로들처럼 사용이 됩니다; 두 버스들 사이에 연결점이 없습니다. 사진 1-5에서 하버드 구조를 볼 수 있습니다.

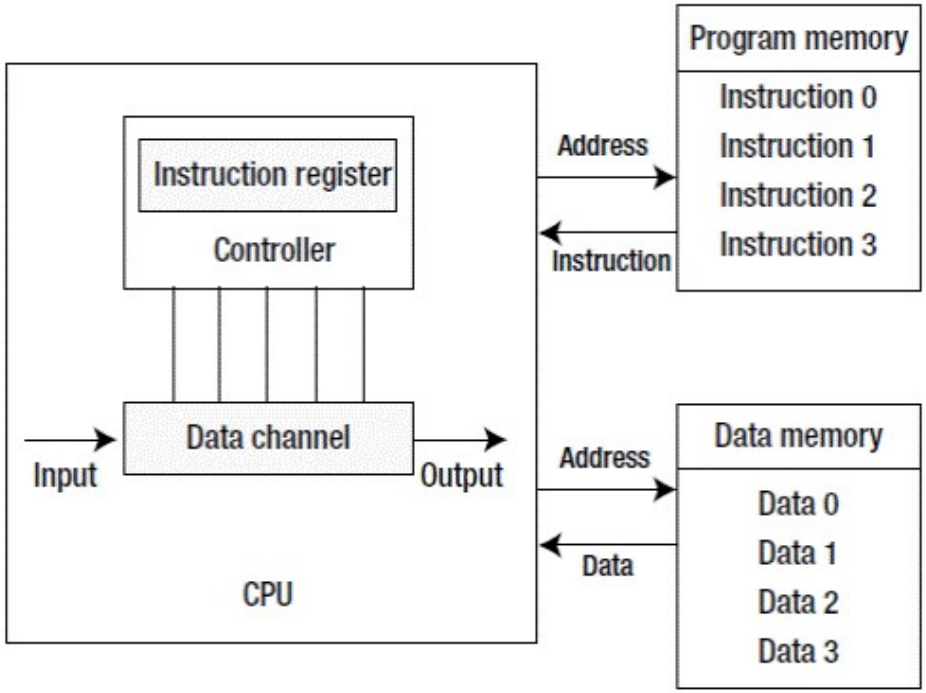


Figure 1-5. Harvard architecture

하버드 구조가 분리되어 있는 프로그램 메모리와 데이터 메모리를 가지고 있기 때문에 그것은 더 큰 데이터-메모리 대역폭을 제공할 수 있으며 이는 디지털 신호 처리에 있어서 이상적인 선택지가 될 수 있습니다. 디지털 신호 처리(DSP)용으로 설계된 대부분의 시스템들은 하버드 구조를 채택합니다. 폰노이만 구조는 단순한 하드웨어 설계와 유연한 프로그램 및 데이터 저장소라는 특징들을 가지며 대개는 범용 그리고 대부분의 임베디드 시스템에서 선택이 됩니다.

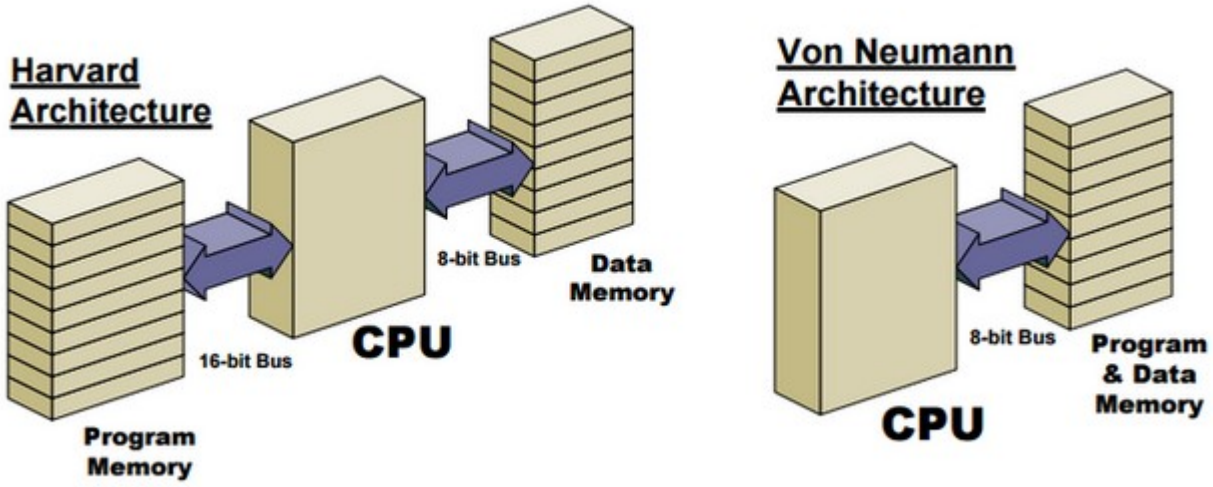
효과적인 메모리 읽기/쓰기를 실행하기 위해서 프로세서는 주 메모리에 직접적으로 연결되지 않고 캐시 메모리에 연결이 됩니다. 흔히 하버드 구조와 폰노이만 구조 사이의 유일한 차이점은 단일 혹은 듀얼 L1 캐시 메모리입니다. 하버드 구조에서 L1 캐시 메모리는 종종 명령어 캐시 메모리(I 캐시 메모리)와 데이터 캐시 메모리(D 캐시 메모리)들로 나뉘어지지만 폰노이만 구조에서는 단일한 캐시 메모리를 가집니다.

폰노이만 구조 VS 하버드 구조

◉ ‘https://m.blog.naver.com/brickbot/220415631821’ 에서 참고를 하였습니다.

◆ 폰노이만 구조는 CPU와 메모리 사이에 1개의 버스만이 있기 때문에 명령어 실행을 함에 있어서 명령어, 데이터 2번의 인출 사이클이 필요하기 때문에 처리 속도가 상대적으로 저속입니다.

◆ 하버드 구조는 명령어 메모리와 데이터 메모리가 분리되어 있어서 동시에 각각의 메모리들에 접근이 가능합니다. 그러므로 처리 속도가 상대적으로 고속입니다.



HW1

임베디드스쿨1기
Lv1과정
2020. 07. 23
권 영근

C의 결과 값은?

◉ 다음과 같은 source code를 작성하였습니다.

```c
#include <stdio.h>

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

    printf("%d\n", c);

    return -1;
}
```

결과 값은 다음과 같았습니다.

```
bolthakziy@bolthakziy-340XAA-350XAA-550XAA:~/C$ vim var.c
bolthakziy@bolthakziy-340XAA-350XAA-550XAA:~/C$ gcc -o main var.c
bolthakziy@bolthakziy-340XAA-350XAA-550XAA:~/C$ ./main
-128
```

128이 아닌 -128이 나왔습니다.

왜 128이 아닌 걸까?

◉ 왜냐하면 '오버플로우' 가 발생했기 때문입니다.

변수 char의 범위는 -128~127입니다. 즉 char가 담을 수 있는 최대 값은 127입니다. 128이라는 값을 담기에는 char의 그릇이 작습니다.

조금 더 자세히 이야기를 해보겠습니다. char는 1바이트의 크기를 가졌습니다. char에 저장되는 127을 8개의 비트들로 표현을 해보겠습니다.

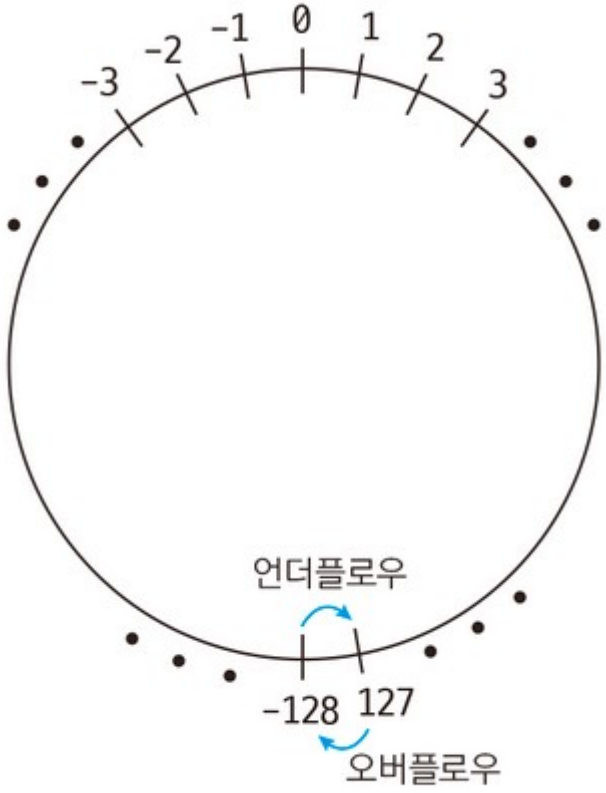
01111111

위의 127에 1을 더하면 비트들에 저장되는 값들은 아래와 같이 변하게 될 것입니다.

10000000

char 변수에서 맨 왼쪽의 비트(MSB)는 부호를 결정합니다. 그렇기 때문에 위의 비트들이 나타내는 값은 128이 아니라 -128입니다. char 변수의 크기가 9비트가 아닌 8비트, 즉 1바이트이기 때문에 128이라는 값을 담을 수 없습니다. 변수 값의 범위 부분에서 최대 값을 넘는 값을 집어넣으려고 할 때, 오버플로우가 발생합니다.

인터넷 사이트 ‘https://dojang.io/mod/page/view.php?id=32’ 에서 발췌하였습니다.

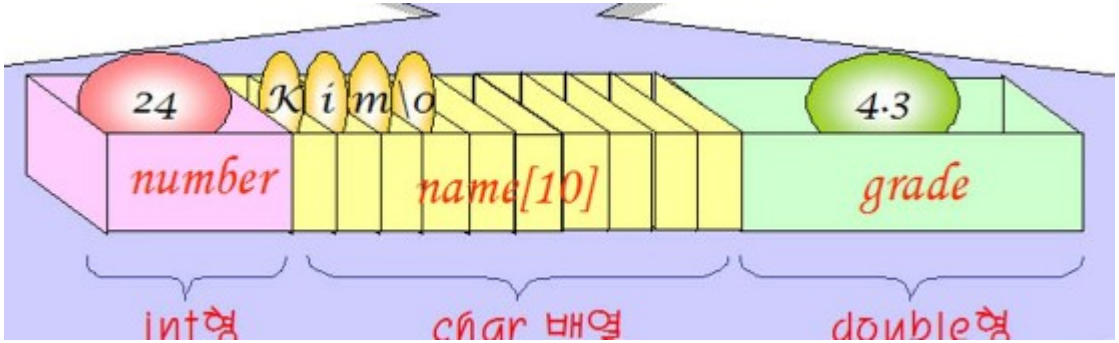


char

시계와 연관을 지어 생각하면 이해가 더 잘될 것입니다. 12시에서 한 시간을 더하면 13시가 되지 않고 1시가 되는 것과 같은 이치입니다.

폰노이만 구조의 장단점

◉ 인터넷 사이트 ‘https://web.yonsei.ac.kr/hgjung/Lectures/GEN131/’ 에서 사진을 발췌하였습니다.



만일 위 사진의 맨 왼쪽에 있는 char 변수의 값이 최대값을 넘었음에도 오버플로우 처리가 되지 않고 최대값을 표기하려고 한다면 어떻게 될까요? 이는 char 변수라는 그릇에 물이 넘쳐서 다른 곳으로 새는 현상과 비슷할 것입니다. 결국 커져버린 부분은 int 변수를 불가피하게 침범하게 될 것입니다. 이는 아주 심각한 일입니다. 외부 변수의 값 일부가 다른 변수의 값에 영향을 미칠 수 있다는 것은 정보의 변질을 의미하기 때문입니다. 정보 사회인 현대 사회에서 이러한 현상은 사람들에게 큰 손실을 끼칠 수도 있을 것입니다.