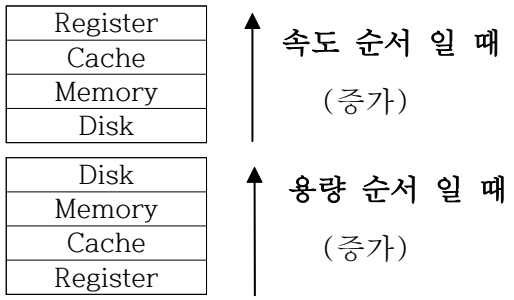
사물인터넷(IoT/ICT)환경에서의
임베디드 응용SW 개발자 양성과정

2st Class

* 우분투 한글 패치. (techlog.gurucat.net/288) 설명을 참조 한다.

* **Memory hierarchy**
  (메모리 계층구조)



*** ALU는 범용 레지스터를 가진다.**
- ax :16bit 운영체제
- eax :32bit 운영체제 (e 가 32bit를 의미 즉 4byte, **4byte 포인터**)
- rax :64bit 운영체제 (r 가 32bit를 의미 즉 8byte, **8byte 포인터**)

*** Stack은 아래로 자란다. 즉 -방향으로 자란다. 원래주소가 0x000044면 32bit운영체제에서 0x000040이 된다는 뜻이다.(-4byte 니까)**

- 값이 쌓이면 스택은 -의 주소를 가지게 됨 반대로 빠지면 +된 주소 값을 가진다.
- stack을 제외하고 나머지 Heap, Data, Text는 정상적으로 쌓인다. (위로 자람)

*** Stack은 빠르지만 용량이 적고 비싸다. 근데 비슷하게 쓸 수 있는 적당한 녀석이 메모리 공간이다.**

-x86의 경우, 함수의 입력을 스택으로 전달함, ARM의 경우 함수4개까지 가능하다, 넘어가면 스택에 넣는다. *< 함수는 4개 이하로 하자 >*

+추가정보 _
모든 프로세서는 레지스터에서 레지스터로 연산이 가능하다. x86은 메모리->메모리로의 연산이 가능하다. 하지만 ARM 프로세서는 불가능.
Arm은 load/sore 방식 이다. (반도체_웨이퍼의 다이가 작아 연산에 도움을 주는 고성능 기능을 취급하지 못한다.)
메모리->레지스터 __레지스터->메모리 방식으로 연산을 한다.

---> gcc -g -O0 -o 생성이름(디버깅파일) 디버깅할c파일

_-g 옵션: 디버깅을 분석 할 수 있도록 미리 설정하는 옵션.
_-O0 옵션: 영어 대문자 O ,숫자 0을 한 것으로 디버깅이 자동최적화가 되지 않게 컴파일 한다.
_-o 옵션: c언어프로그램 컴파일 시, 이름을 지정하기 위한 옵션.

<포인터의 크기?>
2byte - 16bit운영체제)
4byte - 32bit운영체제) ----> ALU 비트 (ax bx sp bp di ip)
8byte - 64bit운영체제)

***운영체제 크기에 따라 포인터의 크기가 결정이 된다.**

< 16진수 , 2진수 는 컴퓨터 용어로 사용 된다>

1bit -> 0,1을 표현한 2진수의 단위 형식
8bit = 1byte를 나타낸다.
64bit는 16개의 16진수로 구성되어 있다.(2진수를 4개씩 끊어 읽으면 16진수 니까)

포인터의 크기

8bit의 경우 1byte
16bit의 경우 2byte
32bit의 경우 4byte
64bit의 경우 8byte

왜 그럴까?

컴퓨터 **연산 -> ALU에 의존적.**
ALU -> 범용 레지스터에 종속적.
*따라서 컴퓨터 64bit는 이들이 64bit라는 뜻.

<(16진수 ,2진수) 변환 정리>

*** 16진수나 2진수를 분해해서 생각한다.**

예를 들면) 144 를 10 진수 개념에서 분해해보자!

1 x 10^2 + 4 x 10^1 + 4 x 10^0

가만 보면 10 진수 시스템에서는
곱하는 부분에 10 이 계속 곱해지고 있음을 볼 수 있다.
그리고 승수로 붙는 곳에 자릿수 - 1 이
배치되고 있음을 볼 수 있다.

그렇다면 2 진수도 비슷하게 생각해보면 되지 않을까 ?
먼저 2 진수의 자릿수를 쭉 적는다.

7	6	5	4	3	2	1	0
128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

위와 같은 형식을 가제 한다면 아래와 같이 적으면 된다.
144 = 128 + 16 이다.

위 색인에서 7 번째에 1 을 셋팅하고
4 번째에 1 을 셋팅하면 아래와 같이 된다.

1001 0000

이것이 144 의 2 진수 변환에 해당한다.
그렇다면 정말로 이게 10 진수 144 가
맞는지 확인할 필요가 있다.
10 진수에 적용한대로 동일한 계산을 적용한다.

1 x 2^7 + 1 x 2^4 = 128 + 16 = 144

***16진수에서 2진수로, 2진수에서 16진수로 빠르게 변환하려면?**
1. 2진수에서 16진수 : 2진수를 4개씩 끊어서 보면 2진수 4자리당 16진수가 된다.
2. 16진수에서 2진수 : 위와 같은 방법으로 16진수를 4자리의 2진수로 넓혀 생각하면 된다.

ex) 10 진수 33 을 2 진수 및 16 진수로 표기해보자.

33 = 32 + 1

32	16	8	4	2	1
1	0	0	0	0	1

10 0001

8421	8421
0010	0001
0x2	1

0x21 => 2 x 16^1 + 1 x 16^0 = 33

ex) 10 진수 2568 을 2 진수 및 16 진수로 표기해보자.

2^10 = 1024
2^11 = 2048

2048	1024	512	256	128	64	32	16	8	4	2	1
1	0	1	0	0	0	0	0	1	0	0	0

2568 - 2048 = 520
520 - 512 = 8
8 - 8 = 0

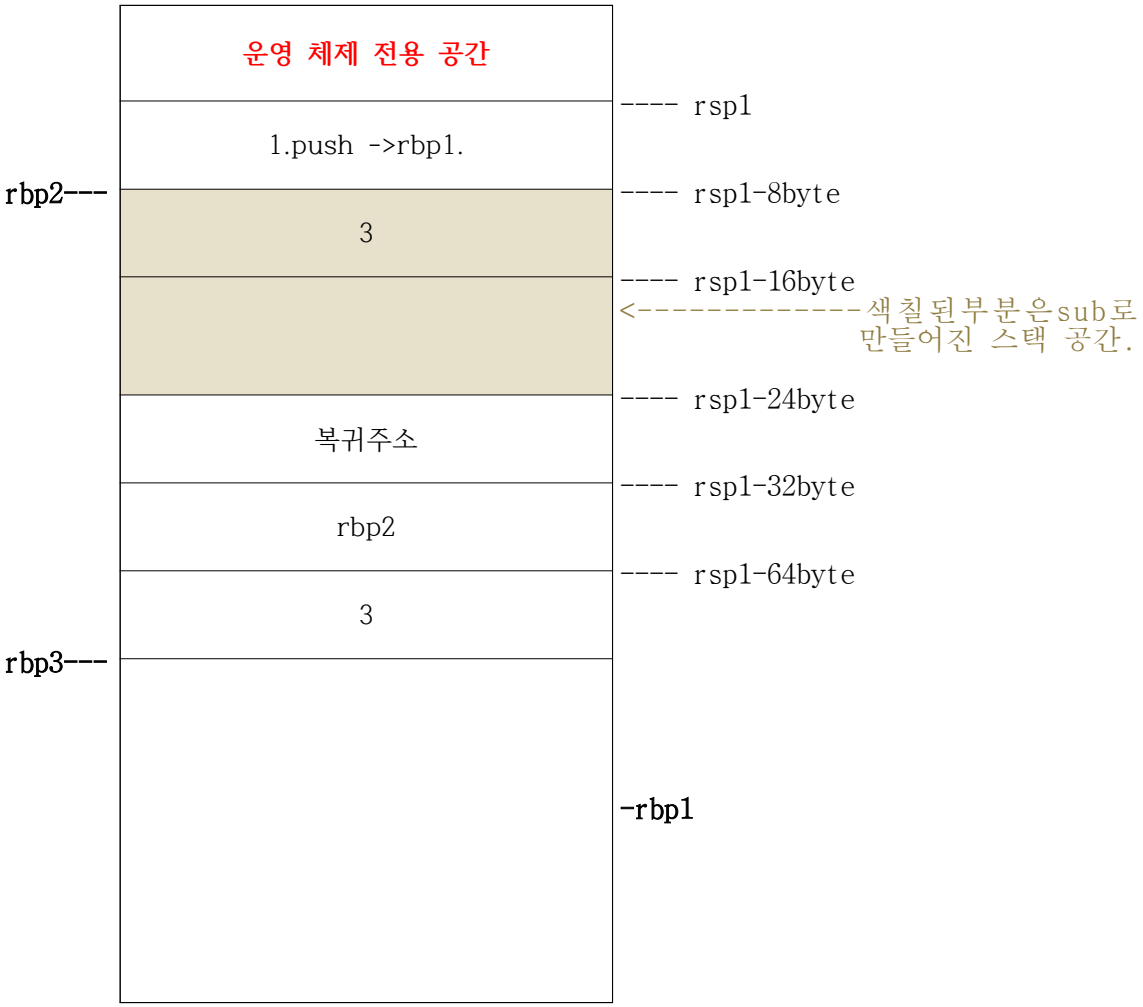
1010 0000 1000

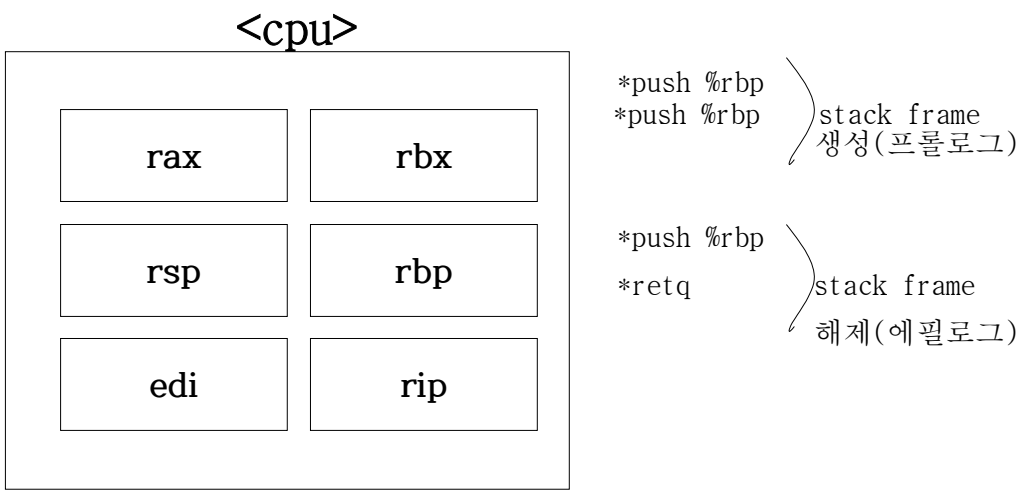
	8421	8421	8421
	1010	0000	1000
0x	a	0	8

0xA08 => A x 16^2 + 8 x 16^0 = 256 x 10 + 1 x 8 = 2568

<어셈블리어 정리>

```
push 1          : 현재위치에 "1"의 값을 저장한다.
pop             : 현재sp에서 값을 빼서 위쪽으로 올려준다. push의 반대.
mov 1 2         : "1" 에서 "2" 로 값을 이동 혹은 복사.
movl 1 2        : "1" 에서 "2" 로 값을 이동 혹은 복사(long type).
add 1 2         : "1"과 "2"값을 더하고 2에 저장.
sub 1 2         : "2" - "1" 의 값을 빼고 "2" 에 저장.
call 1 <함수2>  : push + jmp 의 동작의 복합 명령어.(push:복귀주소를 스택저장,
                  jmp는 call뒤에 오는 주소 값으로 이동)
retq %rbp       : pop + rip (pop: 현재 rbp값을 버리고 복귀주소에 담겨있는
                  rbp주소로 돌아 감. 그리고 rip가 현재 시작할 위치를 나타냄.)
```


운영 체제 전용 공간
---- rsp1
1.push ->rbp1.
rbp2---
---- rsp1-8byte
3
---- rsp1-16byte
<----------색칠된부분은sub로
만들어진 스택 공간.
---- rsp1-24byte
복귀주소
---- rsp1-32byte
rbp2
---- rsp1-64byte
3
rbp3---
-rbp1


<cpu>
rax
rbx
rsp
rbp
edi
rip
*push %rbp
*push %rbp
stack frame
생성(프롤로그)
*push %rbp
*retq
stack frame
해제(에필로그)