# 펭귄 스터디 내용10

작성자 : 류성수
작성일자 : 24.07.02

url : https://www.youtube.com/watch?v=nRR0ymmICBo&list=PLz--ENLG_8TMdMJIwyqDIpcEOysvNoonf&index=11

## ● 데이터 버스 폭

항상 컴퓨터로 어떤 어플을 받을 때, 32bit 컴퓨터용과 64bit 컴퓨터용이 따로 있다.
여기서 bit는 무엇을 의미하는가?
여러 가지요소로 구분되지만, 가장 범용적으로 구분되는 요소는 RAM과 CPU와의 통신 데이터버스의 폭이 32bit 혹은 64bit 라는 것이다.

이전에 전공수업을 들으면서, 32bit 컴퓨터에선 4GB이상의 RAM을 꽂아도 컴퓨터는 오직 4GB의 공간만 사용할 수 있다는 점을 들었었다.
2^32 ~ 4GB 이기에, CPU가 1clk로 한번에 긁어올 수 있는 주소값이 4GB까지임을 알 수 있다. 여기서 4GB 이상이면, 주소표현을 할 수가 없기에, 컴퓨터 내부적으로 4GB까지만 사용하는 것이다.
그렇기 때문에, 32bit 시스템에선 포인터 주소값의 크기가 4byte로 항상 고정된다.
64bit에선 한번에 긁어올 수 있는 크기가 64bit = 8byte 이기에, 그 이상의 RAM도 표현할 수 있는 것이다.

## ● 포인터와 배열

포인터와 배열은 어셈블리어 단에서 매우 유사한 구조를 가진다.
배열을 포인터로 사용할 수 있고, 그 역으로도 가능하다. 이를 알아보기 위해, 아래와 같은 소스코드를 컴파일하여 어셈블리어 단에서 살펴보았다.

<table>
<tr>
<td>

```
1       void main()
2       {
3               int Arr[5];
4
5               Arr[0] = 0x00;
6               Arr[1] = 0x11;
7               Arr[2] = 0x22;
8               Arr[3] = 0x33;
9               Arr[4] = 0x44;
10
(gdb) list
11              *(Arr+0) = 0x55;
12              *(Arr+1) = 0x66;
13              *(Arr+2) = 0x77;
14              *(Arr+3) = 0x88;
15              *(Arr+4) = 0x99;
16      }
17
```

</td>
<td>

```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000001149 <+0>:     endbr64
   0x000000000000114d <+4>:     push   %rbp
   0x000000000000114e <+5>:     mov    %rsp,%rbp
   0x0000000000001151 <+8>:     sub    $0x20,%rsp
   0x0000000000001155 <+12>:    mov    %fs:0x28,%rax
   0x000000000000115e <+21>:    mov    %rax,-0x8(%rbp)
   0x0000000000001162 <+25>:    xor    %eax,%eax
   0x0000000000001164 <+27>:    movl   $0x0,-0x20(%rbp)
   0x000000000000116b <+34>:    movl   $0x11,-0x1c(%rbp)
   0x0000000000001172 <+41>:    movl   $0x22,-0x18(%rbp)
   0x0000000000001179 <+48>:    movl   $0x33,-0x14(%rbp)
   0x0000000000001180 <+55>:    movl   $0x44,-0x10(%rbp)
   0x0000000000001187 <+62>:    movl   $0x55,-0x20(%rbp)
   0x000000000000118e <+69>:    movl   $0x66,-0x1c(%rbp)
   0x0000000000001195 <+76>:    movl   $0x77,-0x18(%rbp)
   0x000000000000119c <+83>:    movl   $0x88,-0x14(%rbp)
   0x00000000000011a3 <+90>:    movl   $0x99,-0x10(%rbp)
   0x00000000000011aa <+97>:    nop
   0x00000000000011ab <+98>:    mov    -0x8(%rbp),%rax
   0x00000000000011af <+102>:   sub    %fs:0x28,%rax
   0x00000000000011b8 <+111>:   je     0x11bf <main+118>
   0x00000000000011ba <+113>:   call   0x1050 <__stack_chk_
   0x00000000000011bf <+118>:   leave
   0x00000000000011c0 <+119>:   ret
End of assembler dump.
```

</td>
</tr>
<tr>
<td>소스코드단</td>
<td>어셈블리단</td>
</tr>
</table>

두 네모의 값을 보면, movl 명령어로 0xαβ 의 값을 rbp에서 -연산하여 계산한 저장공간의 위치로 옮기는 것(대입)을 확인할 수 있다.
서로 동일한 위치로 옮기고, 똑같은 명령으로 구성된 것을 확인할 수 잊다.
그래서 C에서 포인터 주소연산을 하든 배열 인덱스 계산을 하든 결국 똑같은 결과가 나온다는 사실을 직접 실습하며 확인할 수 있었다.