

## <임베디드 어플리케이션 분석>

### 1. 이것이 없으면 사실상 C 언어를 사용할 수 없다.

셀에 부트로드

### 2. 배열에 아래와 같은 정보들이 들어있다.

### 3.

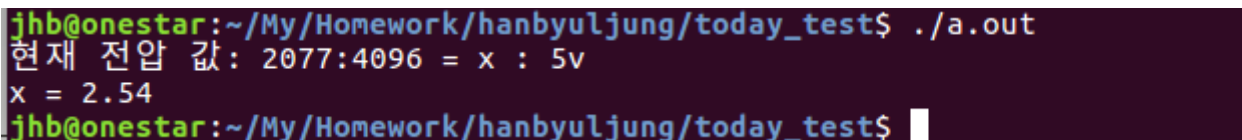
```
#include<stdio.h>
float fuc(int ADC, int voltage_max , int mcu_bit){

    return (float)ADC*(float)voltage_max/(float)mcu_bit;
}

int main(void)
{
    int ADC = 2077;
    int voltage_max = 5;
    int mcu_bit = 4096;
    float res;

    printf("현재 전압 값: 2077:4096 = x : 5v\n");
    res = fuc(ADC,voltage_max,mcu_bit);
    printf("x = %.2f\n", res);

    return 0;
}
```



```
jhb@onestar:~/My/Homework/hanbyuljung/today_test$ ./a.out
현재 전압 값: 2077:4096 = x : 5v
x = 2.54
jhb@onestar:~/My/Homework/hanbyuljung/today_test$
```

### 4. goto 문을 사용하는 이유는

goto 로 표현 가능할 수 있는 if 문이나 반복문 등 조건, 분기, 반복문 등의 경우가 있는데 goto 외의 것들은 어셈블리어 관점에서 볼 때 더 많은 cpu 클럭을 잡아먹게 된다. if 문 같은 경우에는 mov cmp jmp 등의 3 가지 클럭을 잡아 먹는 반면 goto 는 jmp 를 한번한다. 그렇기 때문에 속도와 효율의 관점에서 사용하게 되며 파이프라인 관점에서 볼 때 반복 조건 분기 등은 call 등의 작업이 들어가기 때문에 파이프라인을 깨트린다는 측면에서 좋지 않다.

### 5. 포인터 크기는 운영체제에 따라 바뀐다.

(16bit → 2byte 포인터, 32bit → 4byte 포인터, 64bit → 8byte 포인터)

### 6. |

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
```

```

int main(void)
{
    char a[]="TI Cortex-R5F Safety MCU is very good to Real-Time System.";

    printf("같은 주소 값은 %p\n",strchr(a,"Safety MCU"));

    return 0 ;
}

```

## 7.

```
#include<stdio.h>
```

```

void print(int (*arr)[3])
{
    int i, j;
    printf("\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++){
            printf("%4d",arr[i][j]);

        }
        printf("\n");
    }
}

```

```

int main(void)
{

    int a[3][3]={0};
    int b[3][3]={0};
    int res[3]={0};
    int i, j,num;
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            printf("3*3 행렬의 %d*%d 의 값을 입력하시오. \n",i,j);
            scanf("%d",&a[i][j]);

        }
    }
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            printf("3*3 행렬의 %d*%d 의 값을 입력하시오. \n",i,j);
            scanf("%d",&b[i][j]);

        }
    }

    print(a);
    print(b);

    for(i = 0;i<3;i++)
    {
        for(j=0;j<3;j++)
            res[i]+=a[i][j]*b[j][i];
        printf("\n");
    }
}

```

```

    }
    for(i=0; i<3;i++)
        printf("%3d, ", res[i]);
    printf("\n");
}

```

**8.**

**void (\*)(int) signal(int signum , void (\* handler)(int))**

**리턴: void(\*) (int)**

**함수명: signal**

**인자: int signum 과 void (\* handler)(int)**

**9. |** void (\*) (\*\*) p(int, int) → void (\*( \*p(int , int ) (\*\*)) )

**10.** 명령의 지연을 시켜줌으로써 파이프라인이 쓸 클럭수를 초기화해 속도적 측면의 효율을 다시 향상시킬 수 있다.

**11. |**

```
#include<stdio.h>
```

```
double a=4.0,b=5.0,e=1.0;
```

```
int c=2, d=1, f=0;
```

```
double res1,res2,res3;
```

```
void cal()
```

```
{
    double res;
    c++;
    d++;
    f++;

    res1 = a/(double)(c);
    res2 = b/(double)(d);
    res3 = e/(double)(f);
}
```

```
int main(void)
```

```
{

    printf("4x^2 +5x +1 정적분 구하기 ");

    cal();
    printf("4x^2 +5x +1 -->> %.2lfx^%d +%.2lfx^%d + %.2lfx^%d", res1,c,res2,d,res3,f);

    return 0;
}
```

## 12. |

```
int main(void)
{
    int a[3][3]={0};
    int b[3][3]={0};
    int res[3]={0};
    int i, j,num;
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            printf("3*3 행렬의 %d*%d 의 값을 입력하시오. \n",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            printf("3*3 행렬의 %d*%d 의 값을 입력하시오. \n",i,j);
            scanf("%d",&b[i][j]);
        }
    }

    print(a);
    print(b);

    for(i = 0;i<3;i++)
    {
        for(j=0;j<3;j++)
            res[i]+=a[i][j]*b[j][i];
        printf("\n");
    }
    for(i=0; i<3;i++)
        printf("%3d, ", res[i]);
    printf("\n");
}
```

## 13. |

## 14. |

## 15. |

intel 의 경우 함수의 입력을 스택으로 전달함

arm 의 경우 함수 4 개 까지 가능 넘어가면 stack 에 넣는다. 함수는 4 개 이하로 하는게 좋다.

모든 프로세서는 레지스터에서 레지스터로 연산이 가능하다.

x86 intel 은 메모리->메모리로 가능하다.

하지만 arm 은 메모리에서 메모리로 바로 불가능 하여 memory 에서 레지스터 레지스터에서 memory 로 연산한다.

## 16. 네이버의 쓰레기같은 사다리 게임을 우리끼리 즐길 수 있는 것으로 대체하는 프로그램을 만들어보자.

## 17. | 아래와 같은 행렬을 생각해보자!

2 4 6

2 4 6

sapply(arr, func) 으로 위의 행렬을 아래와 같이 바꿔보자!

2 4 6

4 16 36

sapply 함수를 위와 같이 구현하라는 의미다.

(R 이나 python 같은 언어에서 지원되는 기법중 하나에 해당한다)

모든 문제는 기능별로 함수를 분리해야함

(통 메인, 통 함수 전부 감점임)

## 18. char \*str = "WTF, Where is my Pointer ? Where is it ?" 라는 문자열이 있다.

여기에 소문자가 총 몇 개 사용되었는지 세는 프로그램을 만들어보자!

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main(void)
{
    char *str = "WTF, Where is my Important Pointer?";

    int i,len,res=0;
    len= strlen(str);
    printf("글자 길이 : %d\n", len);
    for(i=0;i<len;i++){
        if(str[i] >= 'a'){
```

```

        res++;

        printf("%c", str[i]);
    }

}

printf("\nWTF, Where is my Importatn Pointer?\n");

printf("소문자의 개 수: %d\n", res);


return 0;

}

```

## 19. int \*p[3] 와 int (\*p)[3] 는 같은 것일까 ? 다른 것일까 ? 이유를 함께 기술하도록 한다.

int \*p[3] 은 int 형타입의 포인터 3 개의 배열이 있다.  
 int(\*p)[3] 은 int 형 3 개 짜리에 대한 포인터 p

## 20. 임의의 값 x 가 있는데, 이를 134217728 단위로 정렬하고 싶다면 어떻게 해야할까 ?

```
#include<stdio.h>
```

```

int main(void)
{
    int a;
    printf("임의의 값을 입력하시오.:");
    scanf("%d", &a);
    printf("%u\n", a&-134217728);

    return 0;
}

```

## 21.

```
#include<stdio.h>
#include<stdlib.h>
```

```

void change(char a)
{
    if(a>='a')
        printf("소문자%c 를 바꾼 대문자 %c\n",a, a-32);
    else
        printf("대문자 %c 를 바꾼 소문자 %c\n",a, a+32);
}

```

```

}

int main(void)
{
    char a;
    printf("알파벳을 입력하시오\n");
    scanf("%c", &a);

    change(a);
    return 0;
}

```

## 22. 변수의 정의를 기술하시오.

메모리에 존재하는 어느 한 공간, 정보를 저장 할 수 있는 한 공간

## 23. 포인터의 정의를 기술하시오.

어느 주소를 가리킨다.

## 24. 함수 포인터

함수 포인터는 메모리 내에서 실행 가능한 코드를 가리킨다.

## 25. |

파이프라인은 분기,반복 문등이 명령어로 들어올 때 깨진다. jmp , call 등의 것에서 깨진다.

## 26. |

(레지스터 > 캐시 > 메모리 > 디스크 )\_속도순 (레지스터 < 캐시 < 메모리 < 디스크 )\_용량순

## 27. |

stack : 스택만 아래로 자란다, 값이 쌓이면 스택은 -의 주소를 가지게 된다. 스택은 빠르지만 용량이 적고 비싸다 근데 비슷하게 쓸 수 있는 적당한 녀석은 메모리이다. 지역변수가 들어간다.

Heap: 동적할당;

Data: 전역변수와 stack 변수가 선언되면 위치하는 공간.

Text: 머신코드 위치.

## 28. |

우리가 보는 주소는 진짜 주소가 아니다. 이주소는 엄밀히 가상메모리에 해당하고 운영체제의 paging 메커니즘을 통해서 실제 메모리로 변환된다.

## 29. | 이름과 급여를 저장하도록 만든다.

30. |

프로그램 컴파일 방법은 gcc -g c파일 이름 최적화를 하지 않으려면 -O0 옵션을 추가 한다. 이름을 바꾸려면 -o 옵션을 추가한다. (예 : gcc -g -O0 -o 바꿀이름 c파일 )

31. |

esc 를 눌러 입력 모드를 해제 후 내가 원하는 부분으로 이동하여 ‘v’를 눌러 그부분들 싸그리 드래그해준다. 마지막으로 ‘=’ 을 누르면 정렬이 된다.

32. |

gcc 컴파일 할 때 -O0 영어 대문자 O 와 숫자 0 을 사용하면 최적화를 하지 않고 O1,O2,O3 등을 사용하면 최적화를 하여 컴파일을 할 수 있다.

33. |

프로그램을 하다가 논리적으로 오류가 발생할 때나 segment fault 가 날때 디버그를 하기 위해서 사용한다. 한편 기계어 수준에서 동작과정을 살펴볼 수 있어 프로그램의 주소나 cpu 레지스터의 값들이 변경되는 시점을 알아보기 위한 용으로도 사용이 가능하다.

34. |

push, pop, call,retq

35. a 좌표(3, 6), b 좌표(4, 4) 가 주어졌다.원점으로부터 이 두 좌표가 만  
들어내는 삼각형의 크기

36. |

37. |

38. |

39. |

40. |

약 한달 동안 c 언어의 자료구조에 대해 배웠습니다. 처음에는 아무 생각도 없이 그저 내 스펙을 향상하기 위해 의미없이 c 언어의 기초를 공부하는 구나 했는데... 막상 와서 배우면서 느낀 것은 내가 기초가 없다는 생각 뿐이었습니다. 제대로 배워 본적이 없었다는 것을 깨달았고 앞으로 알아가야 하는 것이 더욱 많다는 것을 알았으며 꾸준히 노력해야 할 부분도 많다는 것을 느꼈습니다. 원래 잘 하는 것도 아니었지만 저의 지식은 그저 어설픈 초보자 였던 것 같습니다. 좋은 강사님을 만나게 되어서 정말 이번 기회가 감사하고 앞으로가 기대가 됩니다. 강사님의 실력을 너무나도 인정하고 있고 수업의 만족도는 10 점 만점에 20 점이라고 해도 될 정도로 항상 그 이상이라고 생각합니다. 앞으로도 발전이 있을 저에게 그리고 열심히 가르쳐 주시는 강사님께 기대가 됩니다. 감사합니다.



41.

(gdb) disas

Dump of assembler code for function main:

```
0x0000000000400573 <+0>:    push    %rbp
0x0000000000400574 <+1>:    mov     %rsp,%rbp
0x0000000000400577 <+4>:    sub     $0x10,%rsp
0x000000000040057b <+8>:    mov     %fs:0x28,%rax
0x0000000000400584 <+17>:   mov     %rax,-0x8(%rbp)
0x0000000000400588 <+21>:   xor     %eax,%eax
0x000000000040058a <+23>:   movl    $0x3,-0x10(%rbp)
0x0000000000400591 <+30>:   movl    $0x7,-0xc(%rbp)
0x0000000000400598 <+37>:   lea     -0xc(%rbp),%rdx
0x000000000040059c <+41>:   lea     -0x10(%rbp),%rax
0x00000000004005a0 <+45>:   mov     %rdx,%rsi
0x00000000004005a3 <+48>:   mov     %rax,%rdi
=> 0x00000000004005a6 <+51>:   callq   0x400546 <swap>
0x00000000004005ab <+56>:   mov     $0x0,%eax
0x00000000004005b0 <+61>:   mov     -0x8(%rbp),%rcx
0x00000000004005b4 <+65>:   xor     %fs:0x28,%rcx
0x00000000004005bd <+74>:   je      0x4005c4 <main+81>
0x00000000004005bf <+76>:   callq   0x400420 <__stack_chk_fail@plt>
0x00000000004005c4 <+81>:   leaveq  0
0x00000000004005c5 <+82>:   retq
```

End of assembler dump.

(gdb) si

swap (a=0x40061d <\_\_libc\_csu\_init+77>, b=0x1) at last.c:1

```
1      void swap(int *a, int *b){
```

(gdb) disas

Dump of assembler code for function swap:

```
=> 0x0000000000400546 <+0>:    push    %rbp
0x0000000000400547 <+1>:    mov     %rsp,%rbp
0x000000000040054a <+4>:    mov     %rdi,-0x18(%rbp)
0x000000000040054e <+8>:    mov     %rsi,-0x20(%rbp)
0x0000000000400552 <+12>:   mov     -0x18(%rbp),%rax
0x0000000000400556 <+16>:   mov     (%rax),%eax
0x0000000000400558 <+18>:   mov     %eax,-0x4(%rbp)
0x000000000040055b <+21>:   mov     -0x20(%rbp),%rax
0x000000000040055f <+25>:   mov     (%rax),%edx
0x0000000000400561 <+27>:   mov     -0x18(%rbp),%rax
0x0000000000400565 <+31>:   mov     %edx,(%rax)
0x0000000000400567 <+33>:   mov     -0x20(%rbp),%rax
0x000000000040056b <+37>:   mov     -0x4(%rbp),%edx
0x000000000040056e <+40>:   mov     %edx,(%rax)
0x0000000000400570 <+42>:   nop
0x0000000000400571 <+43>:   pop     %rbp
0x0000000000400572 <+44>:   retq
```

End of assembler dump.

(gdb) ni

```
0x0000000000400547      1      void swap(int *a, int *b){
```

