1.
헤더파일

2.

3.

4.
goto 를 사용하지 않는 경우, if 와 break 등을 여러번 이용하여 구현하여야 한다. goto 를 사용하게 되면 jmp 를 한 번 사용하지만 후자의 경우에는 jmp 를 여러 번 사용하게 된다. jmp 를 사용하는 횟수만큼 파이프라인이 깨지게 되고 그 만큼 CPU 에 부담을 주므로, goto 를 사용하여 최소한의 부담을 주려고 사용한다.

5.
포인터의 크기는 64bit 운영체제의 경우 8byte, 32bit 의 경우 4byte, 16bit 의 경우 2byte, 8bit 의 경우 1byte 이다. 즉 운영체제의 크기만큼을 pointer 의 크기로 잡는다. 이러한 이유는 포인터는 원하는 주소를 가르킬 수 있는 작업을 하기때문에 빠른 속도를 위해서 운영체제의 크기를 전부 사용하게 된다.

6.

7.

```
#include <stdio.h>


int multiply(int (*arr1)[3], int (*arr2)[3])
{
    int i, j, k;
    int multiply[3][3] = {0};


    printf("\n");


    for(i = 0; i < 3; i++)
    {
      for(j = 0; j < 3; j++)
      {
         for(k = 0; k < 3; k++)
         {
            multiply[i][j] += arr1[i][k] * arr2[k][j];


         }
```

```
            printf("%d ", multiply[i][j]);
        }
        printf("\n");
    }
  return 0;
}

int main(void)
{
  int i, j, k, l;
  int arr_1[3][3] = {{10, 20, 30}, {40, 50, 60}, {70, 80, 90}};
  int arr_2[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

  for(i = 0; i < 3; i++)
  {
     for(j = 0; j < 3; j++)
     {
          arr_1[i][j];

          printf("%d ", arr_1[i][j]);
     }
     printf("\n");
   }

   printf("\n");

   for(k = 0; k < 3; k++)
   {
      for(l = 0; l < 3; l++)
      {
           arr_2[k][l];
```

```
            printf("%d ", arr_2[k][l]);
        }
        printf("\n");
    }

    multiply(arr_1, arr_2);

    return 0;
}
```

8.

return void(*)int

함수 signal

인자 int signum, void(* handler)(int)

9.

10.

11.

12.

```
#include <stdio.h>

int multiply(int (*arr1)[3], int (*arr2)[3])
{
    int i, j, k;
    int multiply[3][3] = {0};

    printf("\n");

    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 3; j++)
```

```
        {
            for(k = 0; k < 3; k++)
            {
                multiply[i][j] += arr1[i][k] * arr2[k][j];

            }
            printf("%d ", multiply[i][j]);
        }
        printf("\n");
    }
  return 0;
}

int main(void)
{
  int i, j, k, l;
  int arr_1[3][3] = {{10, 20, 30}, {40, 50, 60}, {70, 80, 90}};
  int arr_2[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

  for(i = 0; i < 3; i++)
  {
      for(j = 0; j < 3; j++)
      {
          arr_1[i][j];

          printf("%d ", arr_1[i][j]);
      }
      printf("\n");
    }

    printf("\n");
```

```
    for(k = 0; k < 3; k++)
    {
        for(l = 0; l < 3; l++)
        {
            arr_2[k][l];

            printf("%d ", arr_2[k][l]);
         }
         printf("\n");
     }

    multiply(arr_1, arr_2);

    return 0;
}
```

13.

```
#include <stdio.h>

int fib(int num)
{
     if(num == 1 || num == 2)
          return 1;
     else
          return fib(num - 1) + fib(num - 2);
}

int main(void)
{
    int num, res, i, sum_e, sum_o;
```

```
    sum_e = 0;
    sum_o = 0;

    for(i = 1; i < 28; i++)
    {
      res = fib(i);



      if(fib(i)%2 == 0)
          sum_e += res;
      else
          sum_o += res;
     }
        printf("SUM only even = %d\n", sum_e);
        printf("SUM only odd = %d\n", sum_o);

        printf("result = %d\n", sum_e - sum_o);
   return 0;
}
```

14.

```
#include <stdio.h>

int fib(int num)
{
    if(num == 1)
        return 1;
    else if(num == 2)
        return 4;
    else
        return fib(num -1) + fib(num -2);
}
```

```
int main(void)
{
   int num, res;
   num = 23;

   res = fib(num);

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



    return 0;
}
```

15.

16.

17.

18.

19.

다르다.

근거

int *p[3] 이 뜻하는 바는 data type 이 int *인 3 개짜리의 배열을 의미

int (*p)[3] 이 뜻하는 바는 3 개짜리의 배열의 주소를 받는 다는 의미

20.

21.

22.

자료를 저장하는 공간

23.

주소를 저장하는 공간

24.

함수의 주소를 저장하는 포인터(함수가 시작하는 부분의 주소)

25.

파이프라인이 깨지는 시점은 call 처럼 jmp 를 이용하여 함수에 들어갈 경우에 깨지게된다.

26.

메모리를

속도가 빠른 순서대로 적게되면 register – cache – memory – disk 순이다. (용량의 크기는 반대)

27.

메모리 내에는 크게 kernel 영역과 user 영역이 존재한다.

다시 user 영역에는 stack, heap, data, text 4 가지 영역으로 나뉘어진다. stack 은 지역변수가 저장되는 공간, heap 은 동적할당된 것들이 존재하는 공간, data 는 전역변수, static 변수가 저장되는 공간, text 는 기계어가 존재하는 공간이다.

28.

가짜주소이다. 현재 우리가 보고있는 주소들은 가상메모리에 존재하는 가짜주소이고, 실제주소는 물리적메모리에 존재한다.

29.
gcc -g -o 파일이름(새로 만드려는 파일이름) 파일이름.c(기존에 있던 파일이름)

30.

최적화 : gcc -g -O0 -o 파일이름(새로 만드려는 파일이름) 파일이름.c(기존에 있던 파일이름)

최적화 x : gcc -g -o 파일이름 파일이름.c

31.

디버깅을 하는 이유에는 크게 2 가지의 이유가 존재한다. 첫번째로는 문법적인 오류(컴파일자체가 불가)를 찾아내기 위함이고, 두번째로는 논리적인 오류(컴파일은 가능하나, 원하는 값이 나오지 않음)를 찾아내기 위함이다.

32.

push, pop + rsp 를 직접 건드릴 경우(sub, add 의 경우)

33.

34.

35.

36.

37.

38.

39.

40.

40.1 한 달간 수업을 들으며 느낀 점

한 달간 수업을 들으며 느낀 점은 1 가지밖에 없었습니다. 그 점을 한 마디로 표현하자면 굉장히 즐겁고 설레였습니다. 지금까지 한 번도 경험하지 못한 분야였으면서도, 언젠가 한 번은 해보고 싶었던 것이 C 였습니다. C 가 무엇인지도 몰랐지만, 한 번쯤 알아두면 좋을 것 같은 막연한 생각만 가지고 지냈습니다. 그랬던 C 와 자료구조를 약 한 달간 배우고 익히는 동안에 새로운 분야에 대한 배움이 주는 만족감이 굉장히 컸습니다. 물론 수업내용자체가 어려워 버겁기도 하였지만, 그것을 이해할 때는 어려운만큼 뿌듯함도 컸습니다. 또한 남은 3 개월동안 전혀 모르는 새로운 것을 더 배울 수 있다는 점이 설레입니다. 근래들어 스스로가 성장하는 정도가 굉장히 더디다고 느꼈는데, 이 곳에 와서 빠른 속도로 성장을 할 수 있게되어 만족스럽니다.

40.2 앞으로의 포부

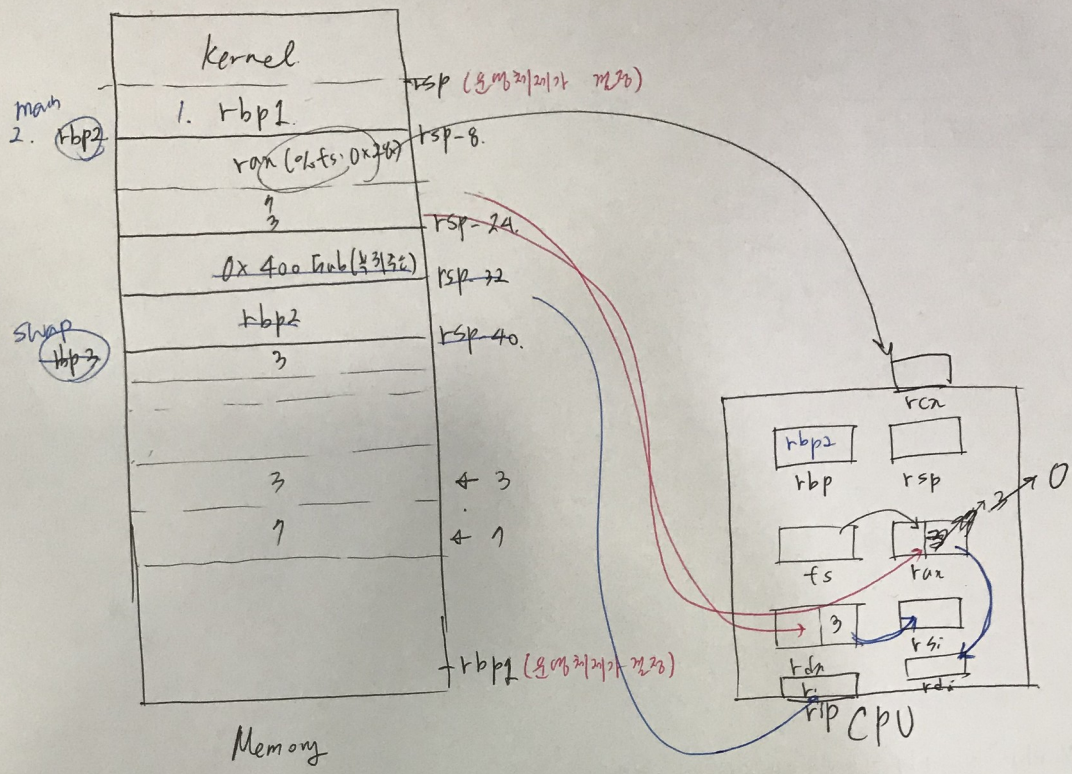
앞으로의 포부는 단순히 열심히 하는 것이 마음가짐입니다. 남들보다 늦은 만큼, 모자란 만큼 수업시간에 좀 더 일찍와서 공부를 하고, 학원에 조금 더 남아 공부를 하려고합니다. 꾀부리지 않고, 꾸준히 노력하면 성장할 수 있음을 굳게 믿기때문에 요령따위를 부리기 보다는 정직하게 많은 시간을 투자해 엉덩이로 공부하려 합니다. 어떤 말로 포부를 표현하기 보다는 직접 행동으로 나타내려 합니다.

40.3 앞으로 하고 싶은 일

앞으로 하고 싶은 일에 대하여 많은 생각을 해보았지만, 지금까지 정하지는 못하였습니다. 현재 공부하는 마음가짐도 취업을 위해서 한다기보다는 하루하루 하고싶은 분야를 공부할 수 있다는 기쁨으로 하고 있습니다. 평소에도 어느 분야를들어가더라도 항상 즐거웠기 때문에 진로를 고르는 것이 아직은 많이 난해하고 어렵습니다. 분야의 관점을 벗어나 인생의 관점에서 보게 되면 앞으로 되고 싶은 모습은 존재합니다. 이제부터 많은 기술을 배우고 많은 노하우를 익혀 어디를 가더라도 항상 필요한 인력이 되어 슈퍼을의 개발자가 되는 것이 목표입니다. 최종적으로는 해외로 진출하여 저 스스로가 회사를 선택해서 갈 수 있을 정도의 사람이 되고 싶습니다. 만약 8 개월이 지난 후 당장이라도 헤외에 나갈 수 있는 기회가 있다면 바로 나가서 실무능력을 쌓을 마음이 있으므로 강의기간동안 스스로 많은 정보를 알아보려합니다.

남은 3 개월의 강의기간동안에 수업내용을 통하여 어떤 분야가 적성에 맞는지 찾아보도록 노력하고 프로젝트때 검증할 수 있는 과정을 거칠 수 있도록 노력하겠습니다.

41.



1. push %rbp : rbp값을 stack 최근값에 저장
2. mov %rsp, %rbp : rbp에 rsp값을 복사
3. sub $0x10, %rsp : rsp에서 16byte를 늘려준다.
4. mov %fs:0x28, %rax : rax에 %fs:0x28을 복사
5. mov %rax, -0x8(%rbp) : rbp기준 8byte 아래 공간에 rax값을 넣음
6. xor %eax, %eax : 값 초기화 (eax가 새로 값받으므로)
7. movl $0x3, -0x10(%rbp) : rbp기준 16byte 아래공간에 3을 넣음
8. movl $0x1, -0xc(%rbp) : rbp기준 12byte 아래공간에 1을 넣음
9. lea -0xc(%rbp), %rdx : rdx에 rbp기준 12byte 아래공간 주소를 복사
10. lea -0x10(%rbp), %rax : rax에 rbp기준 16byte 아래공간 주소를 복사
11. mov %rdx, %rsi : rdx값을 rsi에 복사
12. mov %rax, %rdi : rax값을 rdi에 복사
13. callq : push & jmp