TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/3/14
수업일수	15 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

1. queue 구현
2. 모의고사 2 회차
    -난수를 이용한 stack 구현
    -14 번 풀이
3. 기계어 모르는 부분 정리
    - 모의고사 1 회차 16 번
    - 모의고사 2 회차 1 번

1. queue 구현

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

typedef struct __queue{
	int data;
	struct __queue *link;
}queue;

queue *get_node(void)
{
	queue *tmp;
	tmp=(queue *)malloc(sizeof(queue));
	tmp->link=NULL;
	return tmp;
}

void print_queue(queue *top)
{
	queue *tmp;
	tmp=top;
	if(tmp->link != NULL)
	{
		printf("print : %d₩n",tmp ->data);
		print_queue(tmp->link);
	}
	else if(tmp-> link ==NULL)
	{
		printf("print : %d₩n",tmp->data);
		return ;
```

```
	}
}

void enqueue(queue **top,int data)
{

	queue *tmp=NULL;

	if(*top==NULL)
	{
		*top=get_node();
		(*top)->data=data;
		printf("enqueue : %d₩n",(*top)->data);
		return ;
	}
	else if((*top)->link!=NULL)
	{
		enqueue(&(*top)->link,data);
	}
	else if((*top)->link==NULL)
	{
		tmp=get_node();
		(*top)->link=tmp;
		tmp->data=data;
		printf("enqueue : %d₩n",tmp->data);

	}		강사님의 코드와 비교했을 때 주황색 부분을 지워 코드를 더 간단하게 짤 수 있다는 것을 확인했다…ㅠ

}
void dequeue(queue **top,int data)
{
	queue *tmp;
```

```
        tmp=*top;
        if((*top)->data==data)
        {
                *top=tmp->link;
                free(tmp);
                return ;
        }
        else
        {
                dequeue(&(*top)->link,data);
        }

}

int main(void)
{
        queue *top=NULL;
        queue *head=NULL;
        int i;

        for(i=1;i<5;i++)
        {
                enqueue(&top,10*i);
        }
                print_queue(top);
                printf("dequeue₩n");
                dequeue(&top,20);
                print_queue(top);
        return 0;
}
```

2. 모의 2 회차

21. 난수를 활용해서 Stack 을 구성한다.
(같은 숫자가 들어가지 않게 하고 20 개 이상 넣는다
이때 들어가는 숫자는 1 ~ 50 사이의 숫자로 넣는다)

```
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>
#include<time.h>
typedef struct __stack{
	int data;
	struct __stack *link;
}Stack;

Stack *get_node(void)
{
	Stack *tmp;
	tmp=(Stack*)malloc(sizeof(Stack));
	tmp->link=NULL;
	return tmp;
}

int check_rand(int arr[20],int i)
{

	int num,count;
	num=rand()%51;
	for(count=0;count<i;count++)
	{
		if(arr[count]==num)
			num=check_rand(arr,i);
```

```
        }
        return num;
}


void push(Stack **top,int data)
{
        Stack *tmp;
        tmp=*top;
        *top=get_node();
        (*top)->data=data;
        (*top)->link=tmp;
        printf("%d₩n",(*top)->data);
}

int pop(Stack **top)
{
        int n;
        Stack *tmp;
        tmp=*top;
        if(*top==NULL)
        {
                printf("Stack is Empty!");
                return 0;
        }
        *top=tmp->link;
        n=tmp->data;
        free(tmp);
        return n;
}
int main(void)
{
```

```
	Stack *top=NULL;
	int i;
	int arr[20];
	int bt[]={0};
	int avl[]={0};
	srand(time(NULL));
	printf("난수 생성₩n");
	for(i=0;i<20;i++)
	{
		arr[i]=check_rand(arr,i);
		printf("%d₩n",arr[i]);
	}

	printf("입력₩n");
	for(i=0;i<20;i++)
	{
		push(&top,arr[i]);
	}
	for(i=0;i<20;i++)
	{
		printf("data : %d₩n",pop(&top));
	}

	return 0;
}
```

14. 7 명의 직원에 대한 급여를 입력받는다.
이들이 받는 급여의 평균을 출력하고 평균 이상을 받는 사람들의 이름을 출력하라

```
#include<stdio.h>

typedef struct __info{
	int pay;
	char name[20];
}info;

int main(void)
{
	info sev[7]={0};
	int i;
	float sum=0;
	for(i=0;i<7;i++)
	{
		printf("pay, name [%d]₩n",i);
		scanf("%d,%s",&sev[i].pay, sev[i].name);
		printf("__₩n");
		sum+=sev[i].pay;
	}
	printf("%f₩n",sum);
	for(i=0;i<7;i++)
	{
		if((sum/7) <= sev[i].pay)
			printf("%s₩n",sev[i].name);
	}
	return 0;
}
```

3. 기계어 모르는 부분 정리

모의고사 1 회 16 번

16. 다음 어셈블리어를 보고 함수의 main 함수부터 my_function 함수까지
stack 의 동작 방식을 그림과 함께 기술하라.

```
int my_function(int num1, int num2, int num3)
{
    return num2 * num3 - num1;
}

int main(void)
{
    int result = 0;
    int num1 = 2, num2 = 3, num3 = 4;
    result = my_function(num1, num2, num3);
    printf("result = %d₩n", result);
    return 0;
}
```

```
Dump of assembler code for function main:
   0x000000000040053f <+0>:    push   %rbp
   0x0000000000400540 <+1>:    mov    %rsp,%rbp
   0x0000000000400543 <+4>:    sub    $0x10,%rsp          rsp 에서 16 을 빼는 이유
=> 0x0000000000400547 <+8>:    movl   $0x0,-0x10(%rbp)        //-0x10(%rbp) : 0        result=0
   0x000000000040054e <+15>:   movl   $0x2,-0xc(%rbp)         //-0xc(%rbp) : 2         num2=4
   0x0000000000400555 <+22>:   movl   $0x3,-0x8(%rbp)         //-0x8(%rbp) : 3         num2=3
   0x000000000040055c <+29>:   movl   $0x4,-0x4(%rbp)         //,-0x4(%rbp) : 4        num1=2
   0x0000000000400563 <+36>:   mov    -0x4(%rbp),%edx         //edx : 4
   0x0000000000400566 <+39>:   mov    -0x8(%rbp),%ecx         //ecx : 3
```

```
   0x0000000000400569 <+42>:    mov    -0xc(%rbp),%eax          //eax : 2
   0x000000000040056c <+45>:    mov    %ecx,%esi                //esi : 3
   0x000000000040056e <+47>:    mov    %eax,%edi                //edi   2
   0x0000000000400570 <+49>:    callq  0x400526 <my_function>           //함수호출
   0x0000000000400575 <+54>:    mov    %eax,-0x10(%rbp)                 //-0x10(%rbp) : 0 -> 10
   0x0000000000400578 <+57>:    mov    -0x10(%rbp),%eax                 //eax : 10          result=10          eax 가 각각 다른 eax?
   0x000000000040057b <+60>:    mov    %eax,%esi                        //esi : 10
   0x000000000040057d <+62>:    mov    $0x400624,%edi
   0x0000000000400582 <+67>:    mov    $0x0,%eax
   0x0000000000400587 <+72>:    callq  0x400400 <printf@plt>
   0x000000000040058c <+77>:    mov    $0x0,%eax
   0x0000000000400591 <+82>:    leaveq
   0x0000000000400592 <+83>:    retq

Dump of assembler code for function my_function:
   0x0000000000400526 <+0>:    push   %rbp
   0x0000000000400527 <+1>:    mov    %rsp,%rbp
   0x000000000040052a <+4>:    mov    %edi,-0x4(%rbp)          //-0x4(%rbp) : 2
   0x000000000040052d <+7>:    mov    %esi,-0x8(%rbp)          //-0x8(%rbp) : 3
   0x0000000000400530 <+10>:    mov    %edx,-0xc(%rbp)         //-0xc(%rbp) : 4
=> 0x0000000000400533 <+13>:    mov    -0x8(%rbp),%eax         //%eax : 3
   0x0000000000400536 <+16>:    imul   -0xc(%rbp),%eax         //%eax : 3*4=12
   0x000000000040053a <+20>:    sub    -0x4(%rbp),%eax         //%eax : 12-2=10
   0x000000000040053d <+23>:    pop    %rbp              %rbp pop 의 의미: 저장된 rbp 의 값으로 되돌아간다?
   0x000000000040053e <+24>:    retq
```

모의고사 2 회 1 번

1. 아래 Code 를 작성하고 이 Code 의 기계어에 대한 그림을 그리고 분석하시오.

```
void swap(int *a, int *b)
{
	int tmp;

	tmp = *a;
	*a = *b;
	*b = tmp;
}

int main(void)
{
	int num1 = 3, num2 = 7;
	swap(&num1, &num2);
	return 0;
}
```

```
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          %fs 가 뭐임? 왜 30 이라는 값이 저장되어있음?
   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        lea 가 포인터와 관련 있는 것?
   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
   0x00000000004005c5 <+82>:    retq

 (gdb) s
swap (a=0x7fffffffdd50, b=0x7fffffffdd54) at 2_1.c:6
6     tmp = *a;
(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                  괄호의 의미 (%rax)
   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
```