컴퓨터구조

MIPS Assembly Programming

학과: 컴퓨터공학과

학번: 18011549

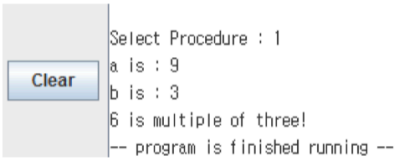
이름: 박태정

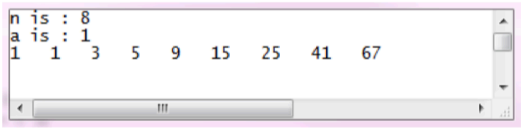
1. Challenge 과정 선택했습니다.

- 1을 입력 받으면 a, b 두 수를 입력 받아, a-b가 3의 배수인지를 출력 후 종료 단 (a-b 〉 0)
- 2를 입력 받으면 a와 n의 두 수를 입력 받은 후 n까지의 점화식을 계산하여 출력하는 MIPS assembly code를 작성하고, MARS로 수행하여라.
- (단, Procedure 형식으로 두 기능을 구현해야만 한다.)

$T(0)=1, T(1)=1, \quad T(n)=\{T(n-1)+T(n-2)\}+a$ 단, $(n \geq 2)$

- 결과 화면





<코드전문>

```
.globl main
.data

select: .asciiz "Select Procedure : "
a_is: .asciiz "a is : "
b_is: .asciiz "b is : "
n_is: .asciiz "n is : "

notmulti: .asciiz " is not multiple of three! "
multi: .asciiz " is multiple of three! "

enter : .asciiz "\n"
space : .asciiz " "

.text
main:
        la $a0 select
        li $v0 4
        syscall
        # printf "Input number : Select Procedure"

        li $v0 5
        syscall
        #input number

        beq $v0,1,isthismulti
        #multi of three

        beq $v0,2,fivo

isthismulti:

        la $a0,a_is
        li $v0,4
```

```
        syscall

        li  $v0  5
        syscall
        #input  number  :  a  is  ~

        li  $t1,0
        add  $t1,$v0,$zero
        #save  a  in  t1

        la  $a0,b_is
        li  $v0,4
        syscall

        li  $v0  5
        syscall
        #input  number  :  a  is  ~

        li  $t2,0
        add  $t2,$v0,$zero
        #save  a  in  t2

        sub  $t1,$t1,$t2
        #save  a-b  into  t1

        j  multiofthree

multiofthree:

        li  $a0,0
        add  $a0,$a0,$t1
        li  $v0,1
        syscall
        li  $t3,3


        div  $t1,$t3
        mfhi  $s1

        beq  $s1,1,multi_exit

        la  $a0,multi
        li  $v0  4
        syscall

        j  exit


multi_exit:
        la  $a0,notmulti
        li  $v0,4
        syscall

        j  exit

fivo:
        la  $a0,n_is
```

```
li  $v0,4
syscall

li  $v0,5
syscall

li  $t1,0
add  $t1,$t1,$v0
#save  n  into  t1  register

la  $a0,a_is
li  $v0,4
syscall

li  $v0,5
syscall

li  $t2,0
add  $t2,$t2,$v0
#save  a  into  t2  register

li  $s0,1
li  $s1,1
#if  number  is  0,1  ->  return  1

la  $a0,enter
li  $v0,4
syscall
#enter

li  $s2,0
li  $t3,0

li  $a0,1
li  $v0,1
syscall
#print  0'th  fivo  num

la  $a0,space
li  $v0,4
syscall
#  space

li  $a0,1
li  $v0,1
syscall
#print  1'th  fivo  num


la  $a0,space
li  $v0,4
syscall
#  space

li  $t3,0
sub  $t1,$t1,1
j  fivo_plus
```

```
fivo_plus:
        # t1 = n , t2 = a
        #s0=1, s1=1 ,s2 = T(n)
        #t3=0 -> count 0~8
        # print 0,1 count  1, 1

        #Loop
        # s2 = s1+s0
        # print s2
        # s0 <- s1
        # s1 <- s2
        # t3 ++ if t3=n -> exit.

        beq $t3,$t1,exit

        add $s2,$s1,$s0 # T(n) = T(n-1)+T(n-2)
        add $s2,$s2,$t2 # T(n) = T(n) + a
        la $a0,($s2)
        li $v0,1
        syscall
        #print n'th fivo num

        la $a0,space
        li $v0,4
        syscall
        # space

        li $t4,0
        add $t4,$zero,$s1
        # tmp = s1
        li $s1,0
        add $s1,$zero,$s2
        #s1 = s2

        li $s0,0
        add $s0,$zero,$t4
        #s0 = tmp

        addi $t3,$t3,1

        j fivo_plus

exit:
        li $v0,10
syscall
```

2. Run I/O 화면

Procedure 1.

```
Select Procedure : 1
a is : 9
b is : 3
6 is multiple of three!
-- program is finished running --
```

```
Select Procedure : 1
a is : 9
b is : 2
7 is not multiple of three!
```

```
Select Procedure : 1
a is : 10
b is : 1
9 is multiple of three!
-- program is finished running --
```

Procedure2.

```
Select Procedure : 2
n is : 8
a is : 1

1 1 3 5 9 15 25 41 67
```

```
Select Procedure : 2
n is : 10
a is : 1

1 1 3 5 9 15 25 41 67 109 177
-- program is finished running --
```

```
Select Procedure : 2
n is : 8
a is : 2

1 1 4 7 13 22 37 61 100
-- program is finished running --
```

3. 느낀점.

우리가 보통 사용하는 지역변수와 전역변수라는 개념이 어셈블리에는 존재하지 않고 변수가 아닌 레지스터에 직접 값을 저장하고 컨트롤 한다는 것이 신선하게 다가왔다.

또한 평소 bit shift 에 대해서 논리연산(AND OR 등)이 있는데 bit shift 가 왜 필요한 것인지 의구심을 가졌었는데, 이번 3단계 Mandatory 과정을 하면서 bit shift 의 필요성을 느꼈다. Challenge 과정에서도 bit shift 를 이용한 배수 판정을 하려 했었다.

```
andi $t2,$t5,1
beq $t2,1,exit

srl $t2,$t1,1
andi $t3,$t2,1
beq $t3,1,exit

srl $t2,$t1,2
andi $t3,$t2,1
beq $t3,1,exit

srl $t2,$t1,3
andi $t3,$t2,1
beq $t3,1,multi
```

<Mandantory 에 사용한 bit shift 를 이용한 배수 판정 코드 >

하지만 8에 비해 3은 규칙을 찾기가 힘들었다. 그렇기에 어쩔 수 없이 div 명령어를 사용했다.
그와 별개로 '하드웨어를 구성하는 기본 원리 중 간단한것이 가장 빠르다. 라는 공감하지만 그래도 필요한 명령어들에 대해서 좀 구현 해줬으면 ~' 하고 생각했었다. 하지만 좀 더 생각해본 결과 이것 또한 추상화의 역할이라고 생각을 했다. 하위단으로 갈 수록 좀 더 간단하고 어쩌면 멍청하다고 생각 될 수 있는 작업들을 반복해야 한다는 것 이다. 그것이 우리가 assembly 만을 사용하지 않고 Python C JAVA 를 사용하는 이유라고 생각했다.

예상한 것 보다 시간이 오래 걸렸지만 그럼에도 유익한 과제였다고 생각한다.