

# 전자 HW 설계 – 실습 보고서

이름: 박관영 (2016124099)

## ✓ Part III

### 동작 원리

#### <Program 목적>

List의 최댓값을 찾아내는 코드를 subroutine을 사용하여 구현한다.

#### <GPR setting>

R4, r8에 미리 지정된 결과값 저장주소와 list 원소의 개수(N)을 저장한다. R5에는 list의 시작주소를 저장한다. 이후 r5를 통해 list원소를 하나하나 읽고, r2에 list 원소를 저장하여 최댓값인지를 비교하게 된다.

#### <LARGE>

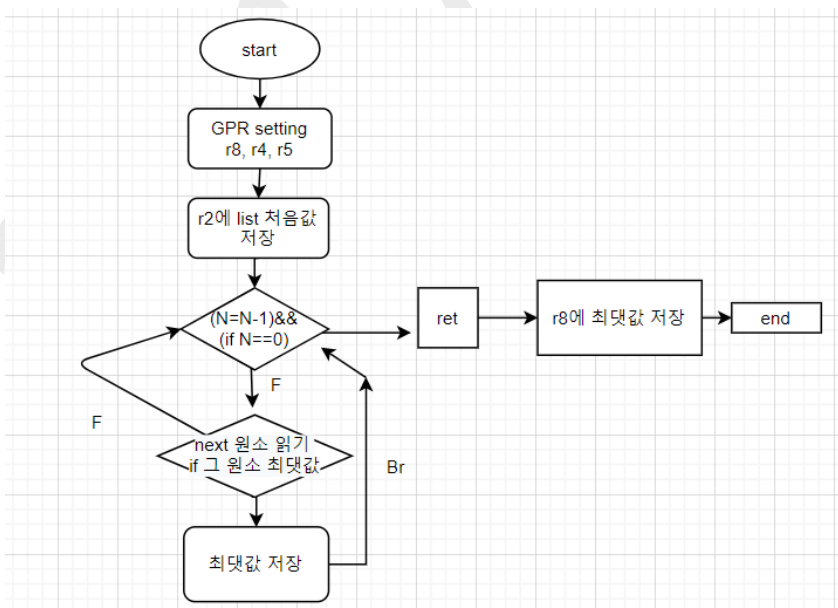
Subroutine 이외에 문제에 제시되어 있는 코드를 건드리지 않기 위해 작성된 구문이다. R5를 ldw하여 list의 첫 원소를 r2에 저장하게 된다.

#### <LARGE2>

$R4 = R4 - 1$ 을 반복적으로 실행하면서 그 값이 0이 되는지를 비교한다. 즉  $N = N - 1$ , (if  $N = 0$ )을 통해 list 원소의 개수(N)만큼 반복문을 실행하게 된다. 또한 반복문이 실행되는 동안  $r5 = r5 + 4$ 를 통해 다음 list 원소의 주소를 읽을 수 있게 된다. 그 값을 r6에 저장 후 r2와 비교하여 최댓값인지 아닌지를 비교한 후 다시 반복문의 처음으로 돌아간다. 만약 최댓값이라면 저장 후 돌아간다.

#### <end>

처음에 설정하였던 저장위치인 r8에 subroutine을 통해 나온 최댓값을 저장한 뒤 무한 반복문으로 시스템을 종료하게 된다.



<Flowchart>

# 전자 HW 설계 – 실습 보고서

## 구현 코드 설명

```
.text
.global _start      # 외부에서 시작위치 참조할 수 있도록 함.
_start:
    movia r8, RESULT # 저장할 주소 지정
    ldw   r4, 4(r8)   # list원소 개수(N) 저장.
    addi  r5, r8, 8   # list 시작주소 불러옴
    call  LARGE       # subroutine call
    mov   r8, r2      # 지정된 저장위치에 결과값 저장.

STOP: br STOP

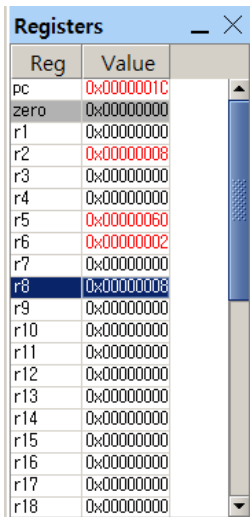
LARGE: ldw   r2, (r5)   # 최댓값을 첫 원소로 초기화
        br   LARGE2
LARGE2: subi r4, r4, 1
        beq  r4, r0, DONE # 반복문 N회 반복
        addi r5, r5, 4     # 다음 원소를 읽는다. (4bytes=1word)
        ldw  r6, (r5)
        bge  r2, r6, LARGE2
        mov  r2, r6        # 최댓값이라면 r2에 update
        br   LARGE2

DONE: ret              # subroutine 종료

RESULT: .skip 4
N: .word 7
NUMBERS: .word 4,5,6,7
          .word 1,8,2

.end
```

## 결과 및 토의



Reg	Value
pc	0x0000001C
zero	0x00000000
r1	0x00000000
r2	0x00000008
r3	0x00000000
r4	0x00000000
r5	0x00000060
r6	0x00000002
r7	0x00000000
r8	0x00000008
r9	0x00000000
r10	0x00000000
r11	0x00000000
r12	0x00000000
r13	0x00000000
r14	0x00000000
r15	0x00000000
r16	0x00000000
r17	0x00000000
r18	0x00000000

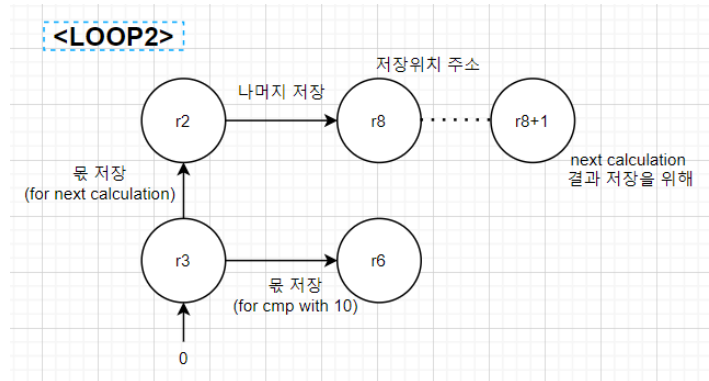
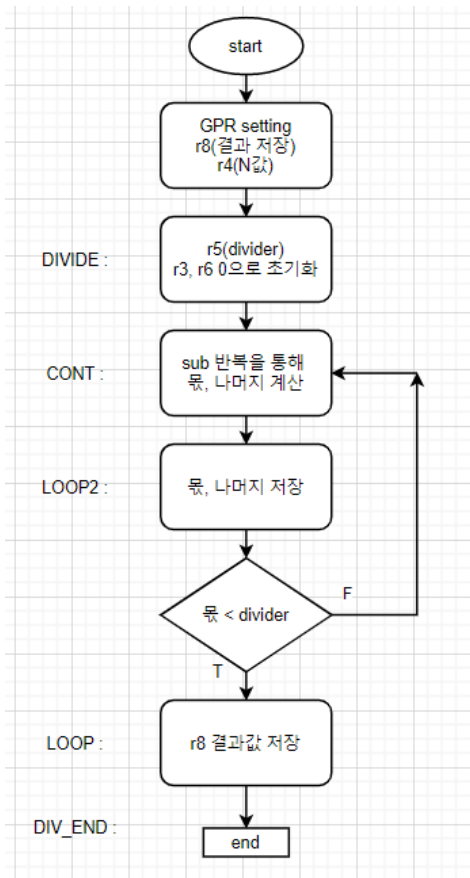
예상한 바와 같이 r8에 8이 저장된 것을 확인할 수 있다. NIOS II는 32-bit processor이므로 pc가 운터는 1word씩 읽기 위해 4씩 증가하는 것을 확인할 수 있었다.

비교를 위해 list의 원소 값을 저장하였던 r6에 2가 저장되어 있는 것을 통해 모든 list의 원소를 비교한 것을 알 수 있다. 또한 r5가 가리키고 있는 주소가 list의 마지막 원소의 주소이다.

# 전자 HW 설계 – 실습 보고서

## ✓ Part IV

### 동작 원리



#### <Program 목적>

n자리의 10진수를 받아 각 자리 숫자를 reg에 저장하는 것이다. 나눗셈을 뿔셈의 반복문을 통해 구현하였고 reg은 16진수로 숫자를 표현하기 때문에 주소를 옮겨서 저장하여야 한다.

#### <GPR setting>

R4에 n자리숫자(N)을 저장하고, r8에 최종 결과값을 저장할 위치의 주소를 저장한다.

#### <DIVIDE>

연산에서 필요한 gpr들(r2, r3, r5, r6)을 초기화하고, 10진수이므로 divider인 r5를 10으로 초기화한다. R2는 나머지, r3는 몫을 저장하게 되고, r6는 후에 몫이 10보다 큰지를 비교할 수 있도록 하는 매개 register 역할을 하게 된다.

#### <CONT>

$R2 = R2 - 10$ 을 R2가 10보다 작아질 때까지 수행한다. 이 때 한번 수행할 때마다 r3의 값을 1씩 더한다. 그 결과 r3에는  $R2/10$ 이 저장되고, R2에는 그 나머지가 저장된다.

#### <LOOP2>

n번째 자리의 숫자(연산의 나머지)를 계산하였으니, r8에 저장한다. 이후  $r8 = r8 + 1$ 을 수행하여 다음 자리의 숫자를 저장할 수 있도록 하는데, 이 때 1은 32bit 주소 값이므로 16진수로 표현되는 reg값에서는 2칸이 이동한 것으로 보인다.

또한, 다음 연산을 위해서 몫을 r2, r6 두 reg에 저장하게 된다. R2에 저장하는 이유는 다음 연산이 R2를 기준으로 이루어지기 때문이고, r6에 저장하는 이유는 몫을 divider와 비교하여 연산을 끝내야 할지의 여부를 알기 위해서이다. 만약 끝났다면, <LOOP> subroutine을 통해 마지막 자리를 저장하면서 연산은 끝나고 그렇지 않다면 다음 연산을 위해 반복문을 수행하게 될 것이다.

# 전자 HW 설계 – 실습 보고서

<DIV\_END>

LOOP subroutine에서 결과 저장이 끝났다면, pc counter에 의해 자동적으로 DIV\_END를 수행하게 된다. Ret를 수행하게 되어 있으므로 subroutine을 종료하게 된다.

## 구현 코드 설명

```
.text
.global _start
_start:
    movia r4, N
    addi r8,r4,4
    ldw r4,(r4)
    call DIVIDE

END: br END

DIVIDE: mov r2, r4    #r2를 사용해 연산 수행
        movi r5, 10   #divider
        movi r3, 0     # r3 초기화
        movi r6, 0     # r6 초기화

CONT: blt r2, r5, LOOP2  # divider보다 작을 때 까지 뺄셈 수행
        sub r2,r2,r5     # 나머지 계산
        addi r3,r3,1     # 몫 계산
        br CONT

LOOP2: stb r2, (r8)      # 계산된 나머지(n번째 자리의 숫자) r8에 저장
        addi r8, r8,1    # 다음자리 저장을 위해 주소+1 (32bit processor이므로 2자리가 건너뛴 것으로 나타남)
        mov r2, r3       # 다음 계산을 위해 몫 넘겨줌
        mov r6, r3       # divider보다 큰지 비교하기 위해 r6에도 몫 저장
        movi r3, 0       # 다음 몫 계산을 위해 0으로 초기화
        blt r6, r5, LOOP  #divider보다 크다면 연산 다시 수행
        br CONT

LOOP: stb r6, (r8)       #마지막 연산의 결과 저장
        ldw r8,(r8)      # r8 주소에 있는 값을 ldw하여 r8에 숫자 자체가 나타나도록 함.

DIV_END: ret    # subroutine 종료

N: .word 1243
Digits: .space 4

.end
```

# 전자 HW 설계 – 실습 보고서

## 결과 및 토의

Reg	Value
pc	0x00000014
zero	0x00000000
r1	0x00000000
r2	0x00000001
r3	0x00000000
r4	0x0000040B
r5	0x0000000A
r6	0x00000001
r7	0x00000000
r8	0x01020403
r9	0x00000000
r10	0x00000000
r11	0x00000000
r12	0x00000000
r13	0x00000000
r14	0x00000000
r15	0x00000000
r16	0x00000000
r17	0x00000000
r18	0x00000000

N에 1243을 저장하였으므로, 마지막 연산의 결과가 저장된 r2에 1이 저장되어 있는 것을 통해 연산이 마지막까지 잘 수행되었음을 알 수 있다. R4에는 N값이 16진수 형태로 저장되어 있는 것을 볼 수 있다.

R2, r3, r6를 통해 연산을 수행하였으므로 r2, r6에는 마지막 연산의 결과인 1이 저장되어 있고, r3는 매번 0으로 초기화 되기 때문에 0이 저장된 것을 볼 수 있다.

이 program의 목적이 N자리수의 각 자리 숫자를 32bit reg에 저장하는 것이라고 하였는데, 지금의 code로는 N자리가 아닌 4자리까지만 수행할 수 있다. 한 reg의 크기가 32bit이고, 주소를 1자리씩 옮길 때 32bit 주소 값을 증가시키게 되므로, 4자리까지만 표현할 수 없다. 따라서 더 많은 자리수의 숫자에 대해서 연산을 수행하려면 r8과 같은 결과저장 reg 수를 증가시켜야 한다.