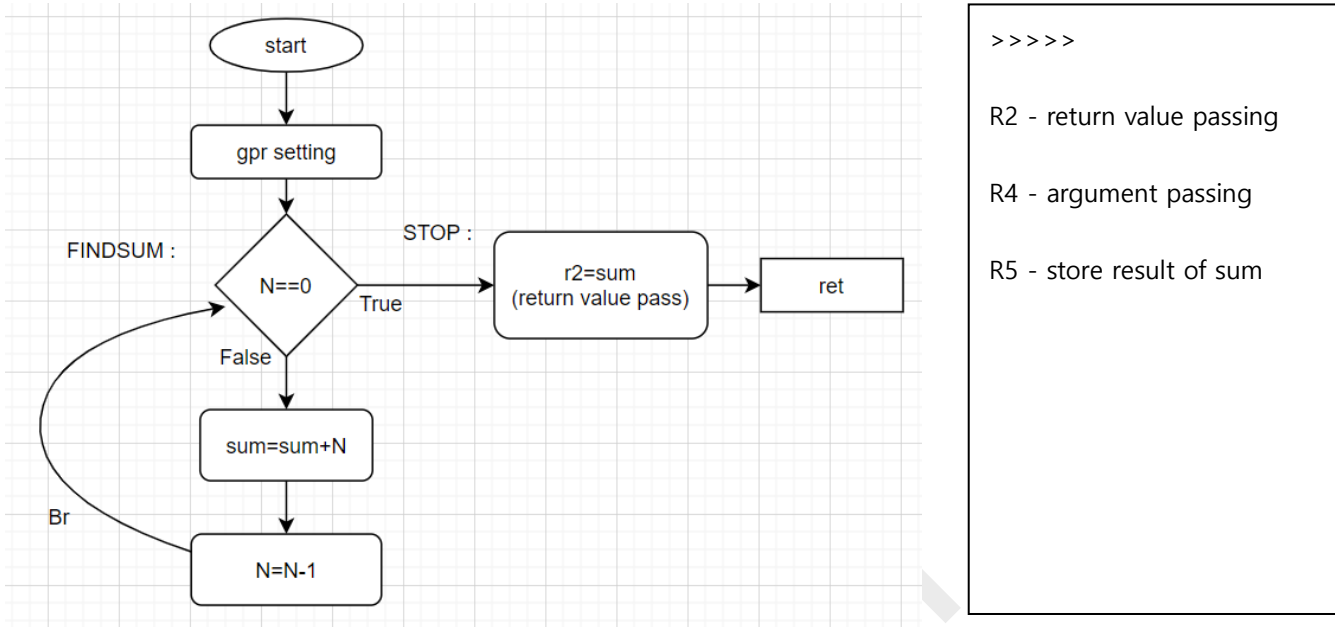


# 전자 HW 설계 - 실습 보고서

이름: 박관영(2016124099)

## ✓ Part I

### 동작 원리



1부터 N까지의 정수합을 iterative하게 구하는 프로그램이다.

#### <GPR setting>

R2로 return value passing을 한다. R4로 N의 주소를 가져온 후 그 주소의 value를 저장한다 (이 프로그램에서는 9). R5로 FINDSUM subroutine 내부에서 sum값을 저장한다.

#### <FINDSUM>

Beq instruction을 통해 N이 0인지 판별하고, 0이 아니면 r5에 N을 더하여 sum값을 갱신한다. 이후 N=N-1을 통해 다음 수를 더하게 한다. 이 때 N이 0이 된다면 N부터 1까지의 정수합이 r5에 저장되었다는 것이므로 subroutine을 빠져나오게 된다.

#### <STOP>

Mov를 통해 r5에 저장된 sum을 r2에 저장한다. R2를 통해 return value passing을 하기 위함이다. 이후 ret를 통해 subroutine을 종료하게 된다.

### 구현 코드 설명

```
.text
.global _start
_start:
    movia r4, N(r0) # absolute addressing을 통해 N의 addr 저장
    ldw r4, (r4) # r4 - N value argument passing
```

## 전자 HW 설계 – 실습 보고서

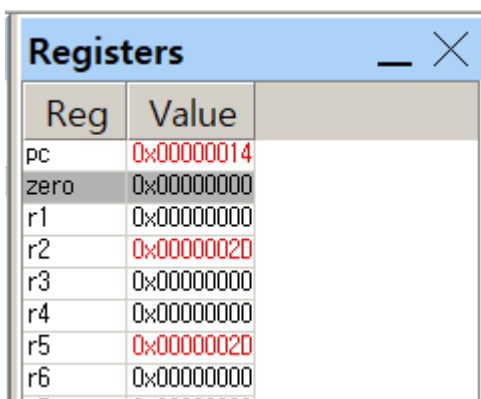
```
    mov r5, r0      # r5 - store result of sum
    call FINDSUM
END: br END

FINDSUM: beq r4, r0, STOP # N==0 이면 종료
        add r5, r5, r4    # sum=sum+N
        subi r4, r4, 1    # N=N-1
        br FINDSUM

STOP: mov r2, r5      # r2 - return value passing
      ret

N: .word 9
   .end
```

결과 및 토의



Reg	Value
pc	0x00000014
zero	0x00000000
r1	0x00000000
r2	0x0000002D
r3	0x00000000
r4	0x00000000
r5	0x0000002D
r6	0x00000000

R2와 r5에 0x2D가 저장되어 있는 것을 통해 1~9의 정수합이 제대로 계산되었다는 것을 알 수 있다. PC counter가 END 무한반복루프를 가리키는 것을 통해 프로그램이 잘 종료되었다는 것을 확인할 수 있다. R4는 subroutine 내에서 subi를 통해서 0으로 되므로 0값을 저장하고 있는 것을 통해 subroutine이 끝까지 잘 실행되었다는 것을 확인할 수 있다.

✓ Part II

```

graph TD
    Start([start]) --> GPR[gpr setting  
(sp=0x20000)]
    GPR --> FINDSUM_N[FINDSUM(N) :  
N==0]
    FINDSUM_N -- T --> R2_R0_T[r2=r0  
(return value 0 pass)]
    R2_R0_T --> RET_T[ret]
    FINDSUM_N -- F --> RECURSE_N[RECURSE :  
save N & ra]
    RECURSE_N --> N_N_1[N=N-1  
call Findsum(N-1)]
    N_N_1 --> FINDSUM_N_1[FINDSUM(N-1) :  
N==0]
    FINDSUM_N_1 -- T --> R2_R0_T_1[r2=r0  
(return value 0 pass)]
    R2_R0_T_1 --> RET_T_1[ret]
    FINDSUM_N_1 -- F --> RECURSE_N_1[RECURSE :  
save N-1 & ra]
    RECURSE_N_1 --> N_N_1_1[N=N-1  
call Findsum(N-2)]
    N_N_1_1 --> FINDSUM_N_2[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_2 -- T --> R2_R0_T_2[r2=r0  
(return value 0 pass)]
    R2_R0_T_2 --> RET_T_2[ret]
    FINDSUM_N_2 -- F --> RECURSE_N_2[RECURSE :  
save N-1 & ra]
    RECURSE_N_2 --> N_N_1_2[N=N-1  
call Findsum(N-2)]
    N_N_1_2 --> FINDSUM_N_3[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_3 -- T --> R2_R0_T_3[r2=r0  
(return value 0 pass)]
    R2_R0_T_3 --> RET_T_3[ret]
    FINDSUM_N_3 -- F --> RECURSE_N_3[RECURSE :  
save N-1 & ra]
    RECURSE_N_3 --> N_N_1_3[N=N-1  
call Findsum(N-2)]
    N_N_1_3 --> FINDSUM_N_4[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_4 -- T --> R2_R0_T_4[r2=r0  
(return value 0 pass)]
    R2_R0_T_4 --> RET_T_4[ret]
    FINDSUM_N_4 -- F --> RECURSE_N_4[RECURSE :  
save N-1 & ra]
    RECURSE_N_4 --> N_N_1_4[N=N-1  
call Findsum(N-2)]
    N_N_1_4 --> FINDSUM_N_5[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_5 -- T --> R2_R0_T_5[r2=r0  
(return value 0 pass)]
    R2_R0_T_5 --> RET_T_5[ret]
    FINDSUM_N_5 -- F --> RECURSE_N_5[RECURSE :  
save N-1 & ra]
    RECURSE_N_5 --> N_N_1_5[N=N-1  
call Findsum(N-2)]
    N_N_1_5 --> FINDSUM_N_6[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_6 -- T --> R2_R0_T_6[r2=r0  
(return value 0 pass)]
    R2_R0_T_6 --> RET_T_6[ret]
    FINDSUM_N_6 -- F --> RECURSE_N_6[RECURSE :  
save N-1 & ra]
    RECURSE_N_6 --> N_N_1_6[N=N-1  
call Findsum(N-2)]
    N_N_1_6 --> FINDSUM_N_7[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_7 -- T --> R2_R0_T_7[r2=r0  
(return value 0 pass)]
    R2_R0_T_7 --> RET_T_7[ret]
    FINDSUM_N_7 -- F --> RECURSE_N_7[RECURSE :  
save N-1 & ra]
    RECURSE_N_7 --> N_N_1_7[N=N-1  
call Findsum(N-2)]
    N_N_1_7 --> FINDSUM_N_8[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_8 -- T --> R2_R0_T_8[r2=r0  
(return value 0 pass)]
    R2_R0_T_8 --> RET_T_8[ret]
    FINDSUM_N_8 -- F --> RECURSE_N_8[RECURSE :  
save N-1 & ra]
    RECURSE_N_8 --> N_N_1_8[N=N-1  
call Findsum(N-2)]
    N_N_1_8 --> FINDSUM_N_9[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_9 -- T --> R2_R0_T_9[r2=r0  
(return value 0 pass)]
    R2_R0_T_9 --> RET_T_9[ret]
    FINDSUM_N_9 -- F --> RECURSE_N_9[RECURSE :  
save N-1 & ra]
    RECURSE_N_9 --> N_N_1_9[N=N-1  
call Findsum(N-2)]
    N_N_1_9 --> FINDSUM_N_10[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_10 -- T --> R2_R0_T_10[r2=r0  
(return value 0 pass)]
    R2_R0_T_10 --> RET_T_10[ret]
    FINDSUM_N_10 -- F --> RECURSE_N_10[RECURSE :  
save N-1 & ra]
    RECURSE_N_10 --> N_N_1_10[N=N-1  
call Findsum(N-2)]
    N_N_1_10 --> FINDSUM_N_11[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_11 -- T --> R2_R0_T_11[r2=r0  
(return value 0 pass)]
    R2_R0_T_11 --> RET_T_11[ret]
    FINDSUM_N_11 -- F --> RECURSE_N_11[RECURSE :  
save N-1 & ra]
    RECURSE_N_11 --> N_N_1_11[N=N-1  
call Findsum(N-2)]
    N_N_1_11 --> FINDSUM_N_12[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_12 -- T --> R2_R0_T_12[r2=r0  
(return value 0 pass)]
    R2_R0_T_12 --> RET_T_12[ret]
    FINDSUM_N_12 -- F --> RECURSE_N_12[RECURSE :  
save N-1 & ra]
    RECURSE_N_12 --> N_N_1_12[N=N-1  
call Findsum(N-2)]
    N_N_1_12 --> FINDSUM_N_13[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_13 -- T --> R2_R0_T_13[r2=r0  
(return value 0 pass)]
    R2_R0_T_13 --> RET_T_13[ret]
    FINDSUM_N_13 -- F --> RECURSE_N_13[RECURSE :  
save N-1 & ra]
    RECURSE_N_13 --> N_N_1_13[N=N-1  
call Findsum(N-2)]
    N_N_1_13 --> FINDSUM_N_14[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_14 -- T --> R2_R0_T_14[r2=r0  
(return value 0 pass)]
    R2_R0_T_14 --> RET_T_14[ret]
    FINDSUM_N_14 -- F --> RECURSE_N_14[RECURSE :  
save N-1 & ra]
    RECURSE_N_14 --> N_N_1_14[N=N-1  
call Findsum(N-2)]
    N_N_1_14 --> FINDSUM_N_15[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_15 -- T --> R2_R0_T_15[r2=r0  
(return value 0 pass)]
    R2_R0_T_15 --> RET_T_15[ret]
    FINDSUM_N_15 -- F --> RECURSE_N_15[RECURSE :  
save N-1 & ra]
    RECURSE_N_15 --> N_N_1_15[N=N-1  
call Findsum(N-2)]
    N_N_1_15 --> FINDSUM_N_16[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_16 -- T --> R2_R0_T_16[r2=r0  
(return value 0 pass)]
    R2_R0_T_16 --> RET_T_16[ret]
    FINDSUM_N_16 -- F --> RECURSE_N_16[RECURSE :  
save N-1 & ra]
    RECURSE_N_16 --> N_N_1_16[N=N-1  
call Findsum(N-2)]
    N_N_1_16 --> FINDSUM_N_17[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_17 -- T --> R2_R0_T_17[r2=r0  
(return value 0 pass)]
    R2_R0_T_17 --> RET_T_17[ret]
    FINDSUM_N_17 -- F --> RECURSE_N_17[RECURSE :  
save N-1 & ra]
    RECURSE_N_17 --> N_N_1_17[N=N-1  
call Findsum(N-2)]
    N_N_1_17 --> FINDSUM_N_18[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_18 -- T --> R2_R0_T_18[r2=r0  
(return value 0 pass)]
    R2_R0_T_18 --> RET_T_18[ret]
    FINDSUM_N_18 -- F --> RECURSE_N_18[RECURSE :  
save N-1 & ra]
    RECURSE_N_18 --> N_N_1_18[N=N-1  
call Findsum(N-2)]
    N_N_1_18 --> FINDSUM_N_19[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_19 -- T --> R2_R0_T_19[r2=r0  
(return value 0 pass)]
    R2_R0_T_19 --> RET_T_19[ret]
    FINDSUM_N_19 -- F --> RECURSE_N_19[RECURSE :  
save N-1 & ra]
    RECURSE_N_19 --> N_N_1_19[N=N-1  
call Findsum(N-2)]
    N_N_1_19 --> FINDSUM_N_20[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_20 -- T --> R2_R0_T_20[r2=r0  
(return value 0 pass)]
    R2_R0_T_20 --> RET_T_20[ret]
    FINDSUM_N_20 -- F --> RECURSE_N_20[RECURSE :  
save N-1 & ra]
    RECURSE_N_20 --> N_N_1_20[N=N-1  
call Findsum(N-2)]
    N_N_1_20 --> FINDSUM_N_21[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_21 -- T --> R2_R0_T_21[r2=r0  
(return value 0 pass)]
    R2_R0_T_21 --> RET_T_21[ret]
    FINDSUM_N_21 -- F --> RECURSE_N_21[RECURSE :  
save N-1 & ra]
    RECURSE_N_21 --> N_N_1_21[N=N-1  
call Findsum(N-2)]
    N_N_1_21 --> FINDSUM_N_22[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_22 -- T --> R2_R0_T_22[r2=r0  
(return value 0 pass)]
    R2_R0_T_22 --> RET_T_22[ret]
    FINDSUM_N_22 -- F --> RECURSE_N_22[RECURSE :  
save N-1 & ra]
    RECURSE_N_22 --> N_N_1_22[N=N-1  
call Findsum(N-2)]
    N_N_1_22 --> FINDSUM_N_23[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_23 -- T --> R2_R0_T_23[r2=r0  
(return value 0 pass)]
    R2_R0_T_23 --> RET_T_23[ret]
    FINDSUM_N_23 -- F --> RECURSE_N_23[RECURSE :  
save N-1 & ra]
    RECURSE_N_23 --> N_N_1_23[N=N-1  
call Findsum(N-2)]
    N_N_1_23 --> FINDSUM_N_24[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_24 -- T --> R2_R0_T_24[r2=r0  
(return value 0 pass)]
    R2_R0_T_24 --> RET_T_24[ret]
    FINDSUM_N_24 -- F --> RECURSE_N_24[RECURSE :  
save N-1 & ra]
    RECURSE_N_24 --> N_N_1_24[N=N-1  
call Findsum(N-2)]
    N_N_1_24 --> FINDSUM_N_25[FINDSUM(N-2) :  
N==0]
    FINDSUM_N_25
```

FINDSUM(0)가 종료되었을 때 스택의 상태

## 전자 HW 설계 – 실습 보고서

Part1의 프로그램과 동일한 역할을 수행하는 프로그램이다. Part1이 1부터 N까지의 정수합을 구하는 프로그램을 iterative하게 구현하였다면, part2에서는 recursive하게 구현하였다. 즉, subroutine내에서 subroutine을 연속적으로 다시 call하는 과정을 통해 같은 작업을 N번 반복하게 한 것이다. 여기서는 ( $N=N-1$  과  $sum=sum+N$ )을 N번 반복하게 된다. Part1과 다른점이 또 있다면 part2에서는 stack을 사용하기 때문에 N부터 1까지 stack에 저장한 후 역순으로 값을 restore하여 더하게 되므로 1부터 N까지 순차적으로 더하게 되고, 그에 따라 프로그램이 종료되었을 때 r4에 N의 값이 남아있게 된다. (part1에서는 0이 저장 되어있음.)

같은 subroutine call을 N번 하면 되기 때문에 Part1에 비해 구현하는 과정이 간단하다. 그러나 stack overflow가 발생할 수 있고, stack이라는 메모리에 access하는 것이기 때문에 더 복잡한 과정을 수행할 때 speed면에서 단점이 있을 수 있다.

<GPR setting>

이 프로그램에서는 stack을 사용하므로 따로 subroutine 내에서 sum을 저장하는 register를 사용하지 않았다. 바로 r2에 값을 저장한 후 return하였다. 이 과목에서 사용하는 DE1-soC board의 stack pointer는 0x20000으로 설정되어 있다. 그러므로, sp를 0x20000으로 초기화한 후 stack을 사용할 수 있다. 또한, R4를 통해 N값을 저장하였다.

<FINDSUM>

N이 0이라면 바로 0을 return하고, 그렇지 않으면 recurse로 branch한다.

<RECURSE>

$Sp=sp-8$ 을 통해 stack에 r4와 ra가 저장될 공간을 마련한다. 다음 subroutine call에서 N값이 갱신되므로, stack에 N값을 저장한다. 또한 subroutine내에서 subroutine call을 하게 되면 return address도 갱신되므로 stack에 ra를 저장한다. 그 후 다음 N에 대해 계산을 수행하기 위해  $N=N-1$ 을 한 후 FINDSUM을 call한다. 이 과정을 계속 진행하면 언젠가  $N=0$ 인 순간이 오게 된다. 즉 FINDSUM(0)을 수행하는 순간이 오게 되는데, 이 때는 r2를 통해 0을 passing하게 된다. 즉, r2가 0으로 초기화 되는 것이다. 이 후  $sp=sp+8$ 을 통해 차례로 stack에 저장되었던 N과 ra를 restore하면서  $sum=sum+N$ 을 반복하고, N번째 FINDSUM의 return address로 return 하게 된다. 결국 마지막  $sum=sum+9$ 까지 수행하고 subroutine은 종료된다. 따라서 r4에는 9가 저장 되어있음을 예상할 수 있다.

구현 코드 설명

```
.text
.global _start
_start:
    movia r4, N(r0)
    ldw r4, (r4)          # r4 - N argument passing
    movia sp, 0x20000     # set stack pointer at DRAM address(just at DE1-soC)
    call FINDSUM
END: br END

FINDSUM: bne r4, r0, RECURSE    # N==0이면 0 return
        mov r2, r0
```

>>>>>>>>

R2 - return value passing

R4 - N value store

Sp - stack pointer

## 전자 HW 설계 – 실습 보고서

```

ret
RECURSE: subi sp, sp, 8    # make space to save N, ra
          stw r4, 0(sp)    # save N at stack
          stw ra, 4(sp)    # save ra at stack (to ret N번째 FINDSUM)

          subi r4, r4, 1    # N=N-1
          call FINDSUM     # FINDSUM(N-1)

          ldw r4, 0(sp)    # restore N at stack
          ldw ra, 4(sp)    # restore ra at stack
          addi sp, sp, 8    # to restore next N, ra

          add r2, r4, r2    # sum = N+FINDSUM(N-1)
          ret

N: .word 9
.end

```

결과 및 토의

Registers	
Reg	Value
pc	0x00000018
zero	0x00000000
r1	0x00000000
r2	0x0000002D
r3	0x00000000
r4	0x00000009
r5	0x00000000
r6	0x00000000
r7	0x00000000

gp	0x00000000
sp	0x00020000
fp	0x00000000
ea	0x00000000
ba	0xFFFFFFFF
ra	0x00000018
status	0x00000000
estatus	0x00000000

예상한 것과 같이 r4에는 9가 저장되어 있는 것을 볼 수 있다. 또한, r2에는 part1과 같은 값인 0x2D가 저장되어 있음을 확인할 수 있다. PC counter가 END 무한반복루프를 가리키는 것을 통해 프로그램이 잘 종료되었다는 것을 확인할 수 있다. sp에는 처음에 지정한 주소 값이 저장되어 있는 것을 통해 모든 save & restore 과정이 잘 수행되었다는 것을 알 수 있다.



# 전자 HW 설계 – 실습 보고서

## 구현 코드 설명

```
.text
.global _start
_start:
    movia r4, N(r0)
    ldw r4, (r4)          # r4 - N argument passing
    movia sp, 0x20000     # set stack pointer at DRAM address(just at DE1-soC)
    call FINDFAC

END: br END

FINDFAC: bne r4, r0, RECURSE    # N==0이면 0 return
        movi r2, 1
        ret

RECURSE: subi sp, sp, 8        # make space to save N, ra
        stw r4, 0(sp)         # save N at stack
        stw ra, 4(sp)         # save ra at stack (to ret N번째 FINDFAC)

        subi r4, r4, 1        # N=N-1
        call FINDFAC          # call FINDFAC(N-1)

        ldw r4, 0(sp)         # restore N at stack
        ldw ra, 4(sp)         # restore ra at stack
        addi sp, sp, 8        # to restore next N, ra

        mul r2, r4, r2        # N! = (N-1) x FINDFAC(N-1)
        ret

N: .word 5
.end
```

## 결과 및 토의

Registers	
Reg	Value
pc	0x00000018
zero	0x00000000
r1	0x00000000
r2	0x00000078
r3	0x00000000
r4	0x00000005
r5	0x00000000
r6	0x00000000

gp	0x00000000
sp	0x00020000
fp	0x00000000
ea	0x00000000
ba	0xFFFFFFFF
ra	0x00000018

## 전자 HW 설계 – 실습 보고서

---

PC counter가 END 무한반복루프를 가리키는 것을 통해 모든 프로그램이 잘 종료되었다는 것을 확인할 수 있었다. Part2와 마찬가지로 r4에는 N값이 저장되어 있었다(여기서는 5). N부터 1까지 저장한 후 1부터 restore하기 때문에 마지막에 연산되는 숫자는 N이 된다. 또한 r2에 0x78이 저장된 것을 통해 결과값이 올바르게 나온 것을 확인할 수 있다. sp에는 처음에 지정한 주소 값이 저장되어 있는 것을 통해 모든 save & restore 과정이 잘 수행되었다는 것을 알 수 있다.