어셈블리 프로그래밍 설계 및 실습

실험제목: control flow& data processing

실험일자: 2020년 09월 22일 (화)

제출일자: 2020년 09월 28일 (월)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

학 번: 2019202052

성 명: 김 호 성

1. 제목및목적
   1. 제목

1. problem 1

2. problem 2

3. problem 3-1

problem 3-2

problem 3-3

* 1. 목적

- Branch와 Branch with link에 대해 이해한다.

- Loop와 일반화된 식 unrolling 총 3가지 방법의 성능을 비교한다.

1. 설계 (Design)

Problem 1

* 1. Pseudo code

Main

LDR R[0] 🡨 Address; 0x00040000

CMP R[9] ,#0, BEQ first ;if (r9 == 0) goto first

CMP R[9] ,#1, BEQ second ;if (r9 == 1) goto second

CMP R[9] ,#2, BEQ third ;if (r9 == 1) goto second

CMP R[9] ,#3, BCS program\_exit ;else goto program\_exit

skip

MOV R[7] 🡨 #0x0A

MOV R[8] 🡨 #0x0B

Loop1

LDRB R[4] 🡨 R[1], #1 ;load r1 in r4, R[1] = R[1] + 1

LDRB R[5] 🡨 R[2], #1 ;load r2 in r5, R[2] = R[2] + 1

CMP R[4], R[5], BNE Loop2 ;if (r4 != r5) goto loop2

CMPEQ R[4], #0, BNE loop1 ;if (r4! = end) goto Loop1

BNE Endline ;else goto Endline(string end)

Loop2

STR R[0] 🡨 R[8]

ADD R[9] 🡨 R[9], #1 ;r9 = r9 + 1

MOV PC 🡨 LR ;goto main

Endline

STR R[0] 🡨 R[7]

ADD R[9] 🡨 R[9], #1 ;r9 = r9 + 1

MOV PC 🡨 LR

first

LDR R[1] 🡨 C\_string1

LDR R[2] 🡨 C\_string2

B skip

second

LDR R[1] 🡨 C\_string3  
 LDR R[2] 🡨 C\_string4

B skip

third

LDR R[1] 🡨 C\_string5

LDR R[2] 🡨 C\_string6

B skip

C\_string1 DCB “Hi”,0

C\_string2 DCB “Hi”,0

C\_string3 DCB “Hi”,0

C\_string4 DCB “Di”,0

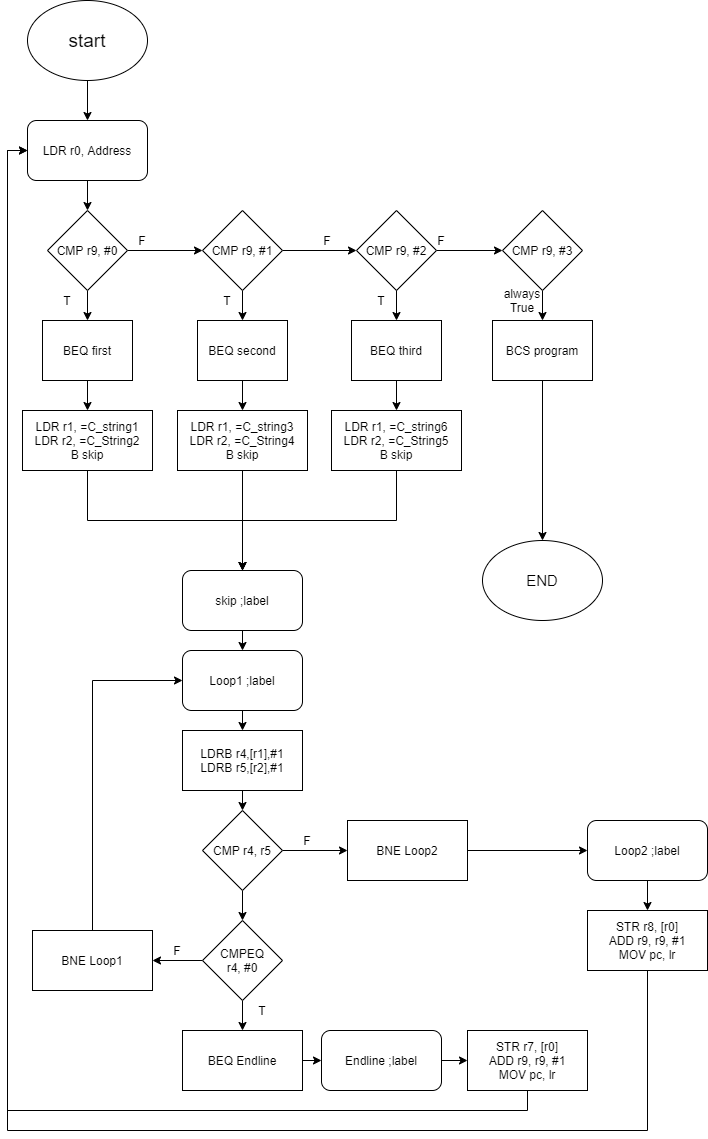
C\_string5 DCB “De”,0

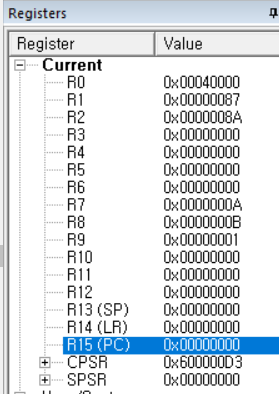
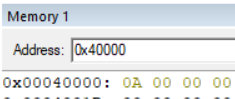
C\_string6 DCB “Se”,0

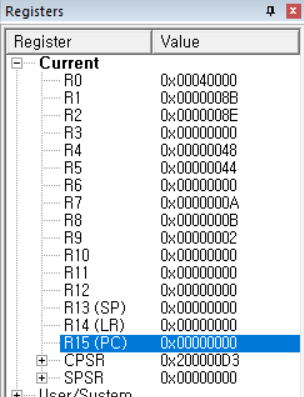
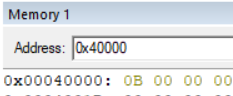
Address DCD &00040000

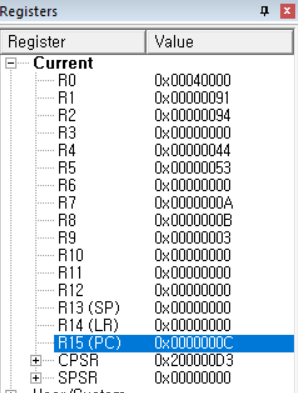
program\_exit

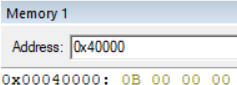
END

* 1. Flow chart 작성
  2. Result

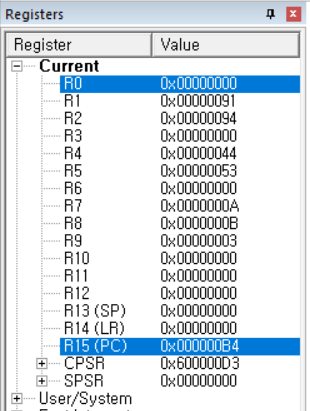
Case 1완료

Case 2 완료

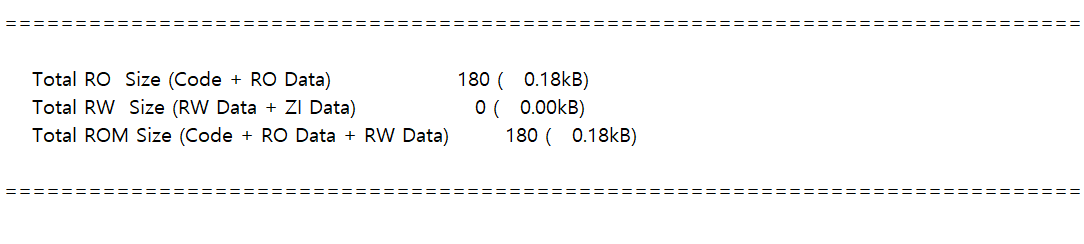
Case 3 완료



Else case(CMP r9, #3 => END)



* 1. Performance

Performance = Code Size \* States \* States =

= 180 \* 156 \* 156

= 4,380,480

설계 (Design)

Problem 2

* 1. Pseudo code

Main

LDR R[0] 🡨 =Number ;address of array

LDR R[1] 🡨 Address ;address value

MOV R[4] 🡨 #10

ADD R[0] 🡨 R[0], #36 ;R[0] = R[0] + 36

;To load in reverse order

Loop

LDR R[2] 🡨 R[0], #-4 ;load the values of the array

STR R[1] 🡨 R[2], #4 ;store the r2 values to r1

ADD R[3] 🡨 R[3], #1 ;r3 = r3 + 1

CMP R[3], R[4], BEQ Endline ;if (r3 == r4) goto Endline

, B Loop ;else goto Loop

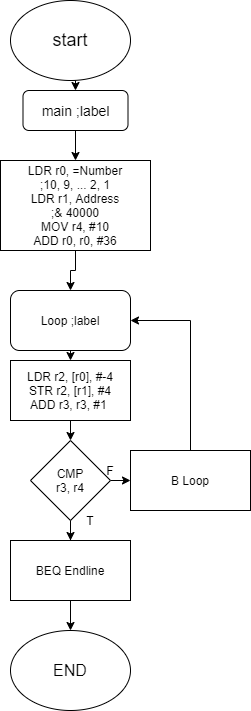
Number DCD 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

Address DCD &00040000

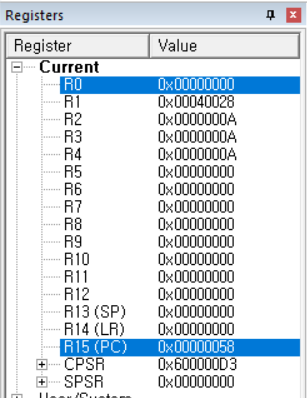
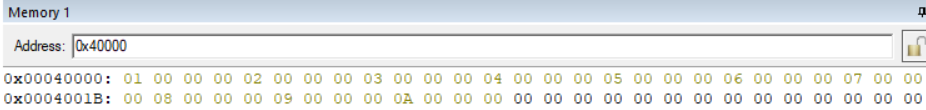
Endline

End

* 1. Flow chart 작성

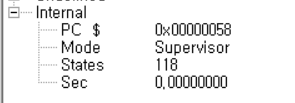
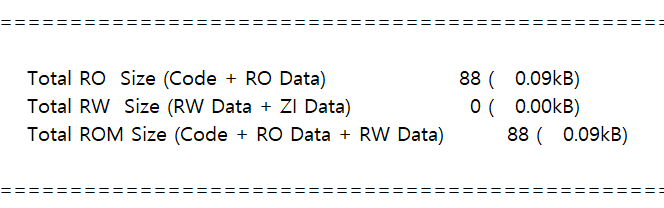


* 1. Result

Register & memory

* 1. Performance

Performance = Code size \* States \* States =

88 \* 118 \* 118 = 1,225,312

설계 (Design)

Problem 3-1

* 1. Pseudo code

Main

MOV R[0] 🡨 #1

MOV R[1] 🡨 R[0], LSL #1

; Store r0 in r1 as much as 1 bit operation

MOV R[2] 🡨 R[0], LSL #3

; Store r0 in r1 as much as 3 bit operation

LDR R[7] 🡨 Address

ADD R[6] 🡨 R[0], R[1] ;r6 = r0 + r1

ADD R[6] 🡨 R[6], R[2] ;r6 = r6 + r2

SUB R[5] 🡨 R[6], R[0] ;r5 = r6 – r0

Loop

ADD R[3] 🡨 R[3], R[6] ;r3 = r3 + r6

ADD R[6] 🡨 R[6], R[1] ;r6 = r6 + r1

ADD R[4] 🡨 R[4], R[0] ;r4 = r4 + r0

CMP R[5], R[4], BEQ Endline ;if(r5 = r4) goto Endline

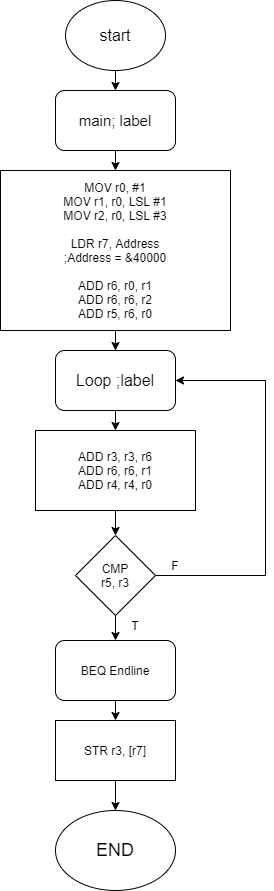
, B Loop ;else goto Loop

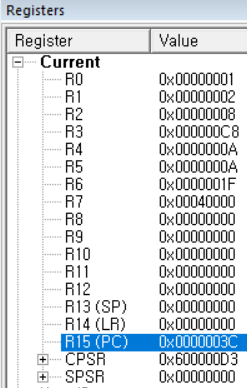
Address DCD &00040000

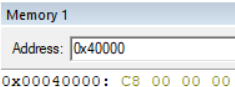
Endline

STR R[7] 🡨 R[3]

END

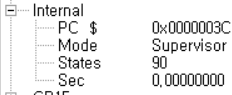
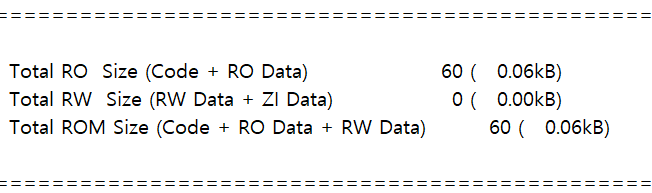
* 1. Flow chart 작성
  2. Result

Register

Memory

* 1. Performance

Performance = Code size \* states \* States =

60 \* 90 \* 90 = 486,000

설계 (Design)

Problem 3-2

* 1. Pseudo code

Main

LDR R[0] 🡨 Address

MOV R[1] 🡨 #1

MOV R[2] 🡨 R[1], LSL #1

MOV R[3] 🡨 R[2], LSL #2

ADD R[4] 🡨 R[2], R[3] ;r4 = r2 + r3

Loop

ADD R[5] 🡨 R[6], R[4]

MUL R[5] 🡨 R[6], R[5]

CMP R[6] ,R[4], BEQ Endline ;if(r6 == r4) goto Endline

, B Loop ;else goto Loop

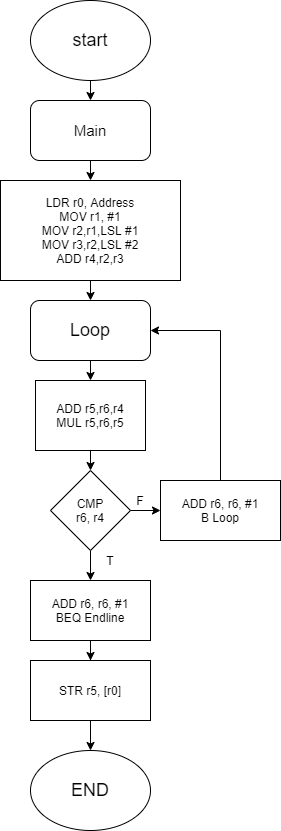
Endline

STR R[0] 🡨 R[5]

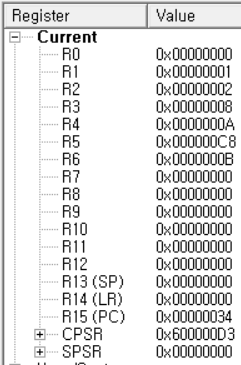
Address DCD &00040000

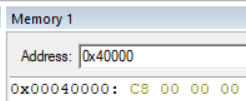
END

* 1. Flow chart 작성



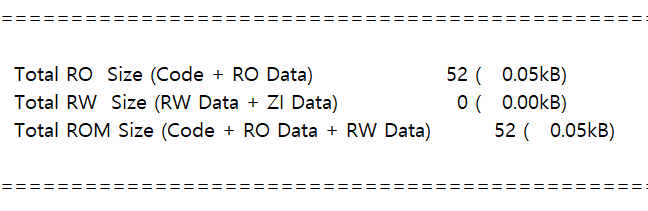
* 1. Result

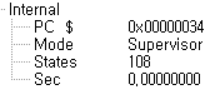
Register

Memory

* 1. Performance

Performance = Code size \* states \* States =

52 \* 108 \* 108 = 606,528



설계 (Design)

Problem 3-3

* 1. Pseudo code

LDR R[0] 🡨 Address

MOV R[1] 🡨 #1 ;r1 = 1

MOV R[2] 🡨 R[1], LSL #1 ;r2 = 2

MOV R[3] 🡨 R[1], LSL #3 ;r3 = 8

ADD R[3] 🡨 R[3], R[1] ;r3 = 9

ADD R[3] 🡨 R[3], R[2] ;r3 = 9 + 2

ADD R[4] 🡨 R[3], R[2] ;r4 = 11 + 2

ADD R[5] 🡨 R[4], R[2] ;r5 = 13 + 2

ADD R[6] 🡨 R[5], R[2] ;r6 = 15 + 2

ADD R[7] 🡨 R[6], R[2] ;r7 = 17 + 2

ADD R[8] 🡨 R[7], R[2] ;r8 = 19 + 2

ADD R[9] 🡨 R[8], R[2] ;r9 = 21 + 2

ADD R[10] 🡨 R[9], R[2] ;r10 = 23 + 2

ADD R[11] 🡨 R[10], R[2] ;r11 = 25 + 2

ADD R[12] 🡨 R[11], R[2] ;r12 = 27 + 2

ADD R[3] 🡨 R[3], R[4] ;r3 = 11 + 13

ADD R[4] 🡨 R[5], R[6] ;r4 = 15 + 17

ADD R[5] 🡨 R[7], R[8] ;r5 = 19 + 21

ADD R[6] 🡨 R[9], R[10] ;r6 = 23 + 25

ADD R[7] 🡨 R[11], R[12] ;r7 = 27 + 29

ADD R[3] 🡨 R[3], R[4] ;r3 = 24 + 32

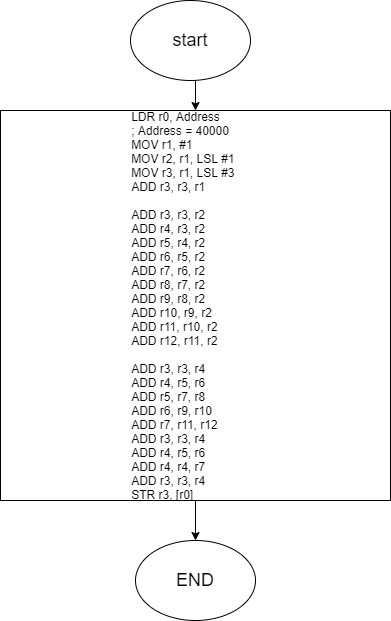
ADD R[4] 🡨 R[5], R[6] ;r4 = 40 + 48

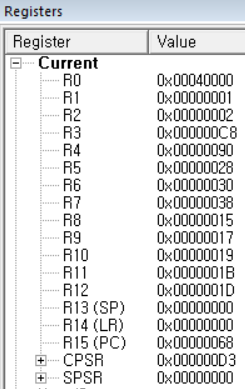
ADD R[4] 🡨 R[4], R[7] ;r4 = 88 + 56

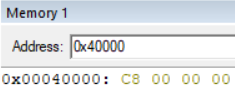
ADD R[3] 🡨 R[3]. R[4] ;r3 = 56 + 144 = 200

Address DCD &00040000

END

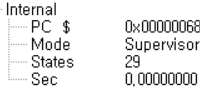
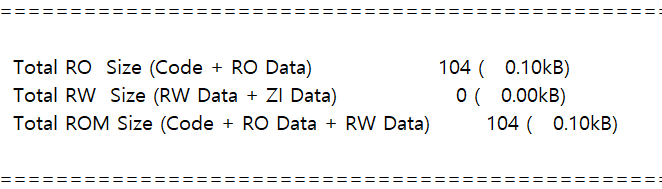
* 1. Flow chart 작성
  2. Result

Register

Memory

* 1. Performance

Performance = Code size \* state \* States =

=104 \* 29 \* 29 =87,464

1. 고찰및결론
   1. 고찰

이번 실습의 경우 Unrolling을 이용하는 문제가 있었다. Unrolling의 경우 반복문에서 반복의 횟수를 줄이는 방법이다. 이를 조건을 계산해서 하는 방법도 있지만, 그렇다면 반복문을 전혀 쓰지 않는 방법으로 코딩한다면 어떨까? 하는 의문과 함께 3-3번의 문제를 풀게 되었다. 결과적으로 performance의 값은 다른 방법들에 비해 현저히 낮은 값을 얻을 수는 있었으나, 만약 원소들의 개수가 무수히 많았을 때의 경우에는 사용할 수 없다는 점을 느꼈다. Branch와 Conditional execution을 모두 사용해보았을 때, Conditional execution은 조건에 따라 반복하여 코드가 길고 보기에 불편함을 느꼈다. 하지만 Branch는 경우에 따라 실행할 코드를 따로 적어 코드의 내용이 한 눈에 들어오는 장점이 있었다.

그러나 Branch를 사용한 이후 pipline이 비기 전까지 다른 instruction을 실행할 수 없는 큰 단점을 가지고 있다. (다른 insturction과 달리 3번의 clock으로 끝나기 때문에 clock에 delay를 2정도 넣어줘야 한다.)

* 1. 결론

3번의 경우 shift연산, 일반화된 연산, 극단적으로 unrolling한 연산을 통해 같은 문제를 다른 방법으로 풀어냈고, 아래는 각각의 performance들의 값이다.

1번째 방법: 486,000

2번째 방법: 606,528

3번째 방법: 87,464

3번째 경우 loop를 아예 넣지 않았으며 연산의 반복으로 풀어내어 값이 적게 나왔으나 만약, 계산해야 하는 값이 지나치게 많다면, 비효율적인 방법이 될 수 있다. (코드 수가 길어져 가독성이 떨어지는 것은 물론이고, 사람이 코딩할 수 있는 수준이 아닐 수도 있음) 또한, 이론상 2번째 방법이 1번째 방법보다 state 값이 적게 나와야 맞으나, 현재는 그렇지 않음을 확인할 수 있다. 만약 일반화 된 식을 알고 개수를 아니까, MUL r5, r6, r5; r6 = 10, r5 = 10을 비트연산을 통해 구한 후 대입한다면 가장 작은 performance값을 얻었을 것이다.

1. 참고문헌

이준환교수님/어셈블리설계및실습/광운대학교(컴퓨터정보공학부)/2020