

어셈블리 프로그래밍 설계 및 실습

실험제목: Block Data Transfer & Stack

실험일자: 2020 년 10 월 06 일 (화)

제출일자: 2020 년 10 월 12 일 (화)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

학 번: 2019202052

성 명: 김 호 성

1. 제목및목적

A. 제목

Block Data Transfer & Stack

B. 목적

- 레지스터와 메모리간 블록 단위의 데이터 저장/가져오기를 이해한다.
- 어셈블리어의 서브 루틴(함수) 사용 방법에 대해 익힌다.

2. 설계 (Design)

Problem1

A. Pseudo code

Main

MOV R[0] \leftarrow #1

LDR sp \leftarrow TEMPADDR1; 0x00040000

MOV R[3] \leftarrow #1

LOOP

CMP R[0], #1

MULGT R[3] \leftarrow R[0] - R[3]

SUBGT R[0] \leftarrow R[0] - #1

BGT LOOP

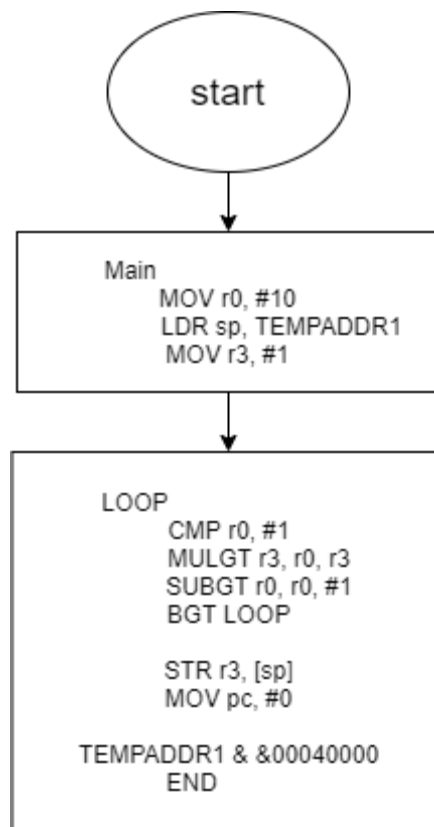
STR R[3] \rightarrow [sp]

MOV pc \leftarrow #0

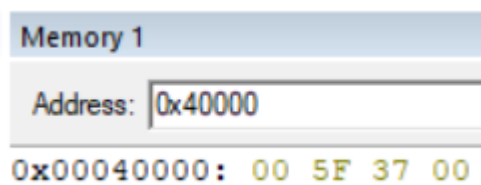
TEMPADDR1 & &0x40000

END

B. Flow chart 작성



C. Result



Little endian 기준이므로 375F00 = 3628800(10 진수)로 올바른 값이 저장됨.

D. Performance

$$\begin{aligned} \text{Performance} &= \text{Code Size} * \text{States} \\ &= 74 * 40 \\ &= 2960 \end{aligned}$$

Problem2

A. Pseudo code

Main

MOV R[0] \leftarrow #2

MOV R[1] \leftarrow #7

MOV R[2] \leftarrow #1

MOV R[3] \leftarrow #3

MOV R[4] \leftarrow #8

MOV R[5] \leftarrow #4

MOV R[6] \leftarrow #5

MOV R[7] \leftarrow #6

LDR sp \leftarrow TEMPADDR1; 0x40000

STMEA sp! \leftarrow R[2]

STMEA sp! \leftarrow R[0], R[3]

STMEA sp! \leftarrow R[5], R[6], R[7]

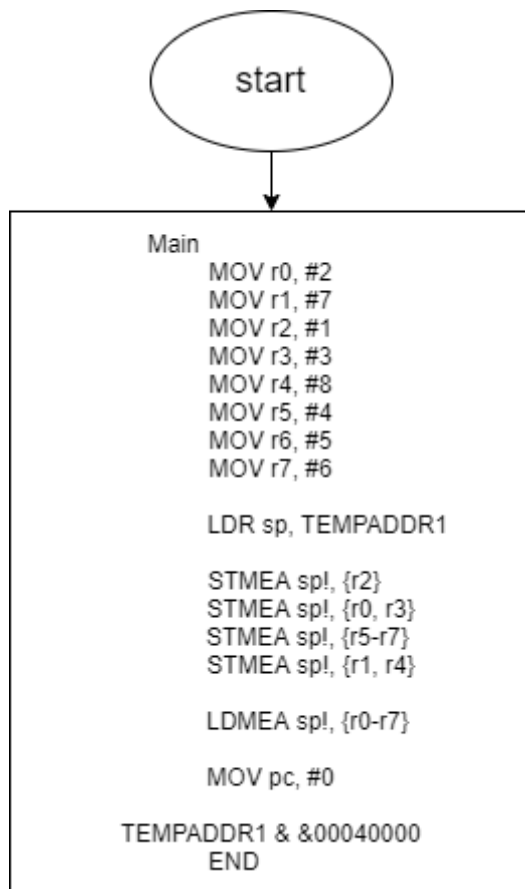
STMEA sp! \leftarrow R[1], R[4]

LDMEA sp! \rightarrow R[0] ~ R[7]

TEMPADDR1 & & 00040000

END

B. Flow chart 작성



C. Result

메모리에 저장된 값

Memory 1	
Address:	0x40000
0x00040000:	01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 06 00 00 00 07 00 00
0x0004001B:	00 08 00

Register 에 저장된 값(정렬 전)

Register	Value
Current	
R0	0x00000002
R1	0x00000007
R2	0x00000001
R3	0x00000003
R4	0x00000008
R5	0x00000004
R6	0x00000005
R7	0x00000006
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00040020
R14 (LR)	0x00000000
R15 (PC)	0x00000034
CPSR	0x00000003
SPSR	0x00000000

Register 에 저장된 값(정렬 후)₩

D. Performance

$$\begin{aligned}\text{Performance} &= \text{Code Size} * \text{States} = \\ &= 64 * 33 \\ &= 2112\end{aligned}$$

Problem 3

A. Pseudo code

```
MOV R[0] ← #10
MOV R[1] ← #11
MOV R[2] ← #12
MOV R[3] ← #13
MOV R[4] ← #14
MOV R[5] ← #15
MOV R[6] ← #16
MOV R[7] ← #17
MOV R[9] ← #160
```

```
LDR sp ← TEMPADDR1
```

doRegister

```
STMPA sp! ← R[0] ~ R[7]
```

```
ADD R[0] = R[0] + #0
ADD R[1] = R[1] + #1
ADD R[2] = R[2] + #2
ADD R[3] = R[3] + #3
ADD R[4] = R[4] + #4
ADD R[5] = R[5] + #5
```

ADD R[6] = R[6] + #6

ADD R[7] = R[7] + #7

ADD R[8] = R[0] + R[1]

ADD R[8] = R[8] + R[2]

ADD R[8] = R[8] + R[3]

ADD R[8] = R[8] + R[4]

ADD R[8] = R[8] + R[5]

ADD R[8] = R[8] + R[6]

ADD R[8] = R[8] + R[7]

LDMDA sp! → {R[0] ~ R[7]}

STMFA sp! ← {R[8], R[9]}

doGCD

CMP R[8], R[9]

SUBGT R[8] = R[8] - R[9]

SUBLE R[9] = R[9] - R[8]

BEQ Endline

BNE doGCD

Endline

ADD R[12] = R[8] + R[4]

LDMDA sp! → {R[8], R[9]}

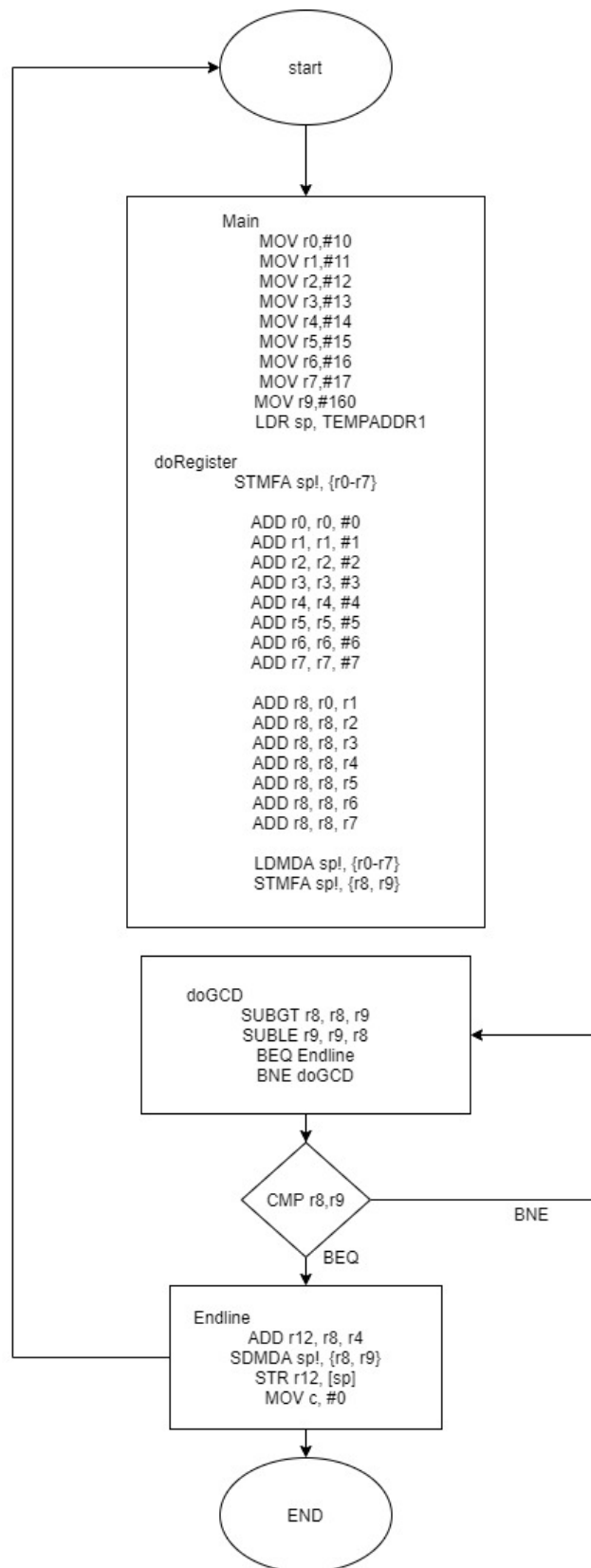
STR R[12] → sp

MOV pc ← #0

TEMPADDR1 & &0x40000

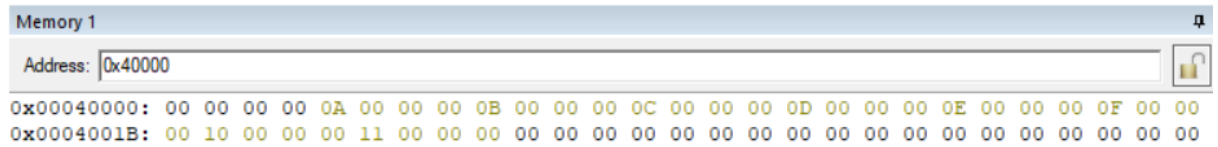
END

B. Flow chart 작성



C. Result

초기 값을 저장한 메모리



각각의 레지스터에 index 값 만큼 더한 후 R8 에 R0~R7 의 값을 더함.

Registers	
Register	Value
Current	
R0	0x0000000A
R1	0x0000000C
R2	0x0000000E
R3	0x00000010
R4	0x00000012
R5	0x00000014
R6	0x00000016
R7	0x00000018
R8	0x00000088

doRegister 함수를 종료하고 다시 복구함.

Registers	
Register	Value
Current	
R0	0x0000000A
R1	0x0000000B
R2	0x0000000C
R3	0x0000000D
R4	0x0000000E
R5	0x0000000F
R6	0x00000010
R7	0x00000011
R8	0x00000088

doGCD 함수를 통해 최대공약수를 구함

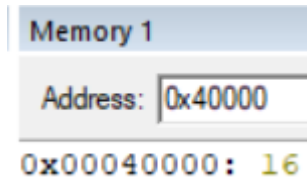
R7	0x00000011
R8	0x00000008
R9	0x00000000
R10	0x00000000

함수를 종료하고 r8, r9 값을 원상태로 복구

R7	0x00000011
R8	0x00000088
R9	0x000000A0

메모리에 최대공약수와 index 한 r4 값을 더한 값을 저장

(18+8 = 26 =0x16)



D. Performance

$$\begin{aligned}\text{Performance} &= \text{Code Size} * \text{States} = \\ &= 152 * 118 = 17936\end{aligned}$$

3. 고찰및결론

A. 고찰

3번문제를 연산할 때 C에서 나눈 값의 나머지를 나타내는 %처럼 그런 instruction을 찾다가 유클리드 호제법을 발견하게 되었다. 유클리드 호제법의 경우 임의의 정수 a, b에 대해서

1. b가 a보다 크면 b와 a를 바꾼다.
 2. $a = a - b$
 3. a가 0이면 b가 최대공약수 아니라면 재귀함수
- 이 3단계를 반복해 최대공약수를 구하는 방법입니다.
그 방법을 응용해서 3번의 GCD를 풀었다.

또 3-1에서 doRegister 함수에서 값에 index를 하건 GCD의 최대공약수를 구하건 그 이후에 값을 다시 돌려야 하는 조건이 있었다. 그러므로 값을 연산하기 전(register에 저장된 값이 바뀌기 전) 미리 memory에 값을 저장하고 연산이 끝난 후 값을 되돌려주었다.

B. 결론

- stack mode의 경우 올바르게 값이 전달되는데 정해진 짝이 존재한다.
- 메모리에 값을 저장하고 불러오는 방식을 사용해 적은 register를 보다 효율적으로 사용할 수 있다.

4. 참고문헌

이준환교수님/디지털논리회로 2/광운대학교(컴퓨터정보공학부)/2020

공영호교수님/디지털논리회로 2/광운대학교(컴퓨터정보공학부)/2020