어셈블리 프로그래밍 설계 및 실습

보고서

실험제목:

실험일자: 2018년 11월 02일 (목)

제출일자: 2018년 11월 06일 (화)

학 과: 컴퓨터공학과

담당교수: 이형근 교수님

실습 분반: 화 6,7 , 수 5

학 번: 2017202010

성 명: 박유림

1. 제목 및 목적

A. 제목

Pseudo Instructions

B. 목적

Pseudo instruction이 assembler에 의해 어떻게 실제 instruction으로 변환되는지 직접 확인하고 이해한다.

Disassembly를 해석하는 방법을 이해한다. (32bit 명령어들의 구조 이해)

Label의 이름이 실제 메모리 주소임을 이해하고 이를 이용해 어셈블리어 프로그래밍을 진행하는 능력을 습득한다.

2. 설계 (Design)

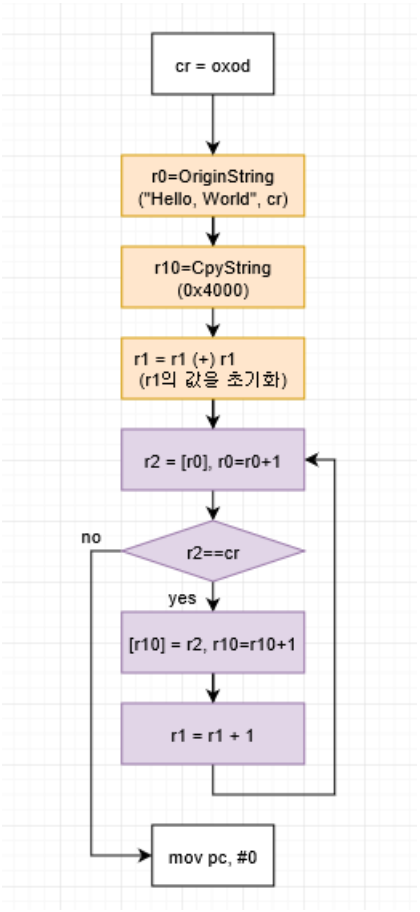
A. Pseudo code

1. Problem1

```
Cr = 0x0d

R0 = OriginString = (“Hello, World”, cr)
R10 = CpyString (0x4000)
R1 = R1 ⊕ R1 // R1의 초기화 과정

R2 = [R0] && R0 = R0 + 1

While(true){
  If(R2 == cr){
        [R10] = R2;
          R10 = R10 + 1;
          R1 = R1 + 1;
  }
  Else break;
}

Pc = 0;
```

B. Flow chart 작성

1. Problem1



문자열의 끝을 알려주는 문자인 0x0d를 cr이라고 이름을 붙인다. 후에 Main Label로 들어오게 된다.

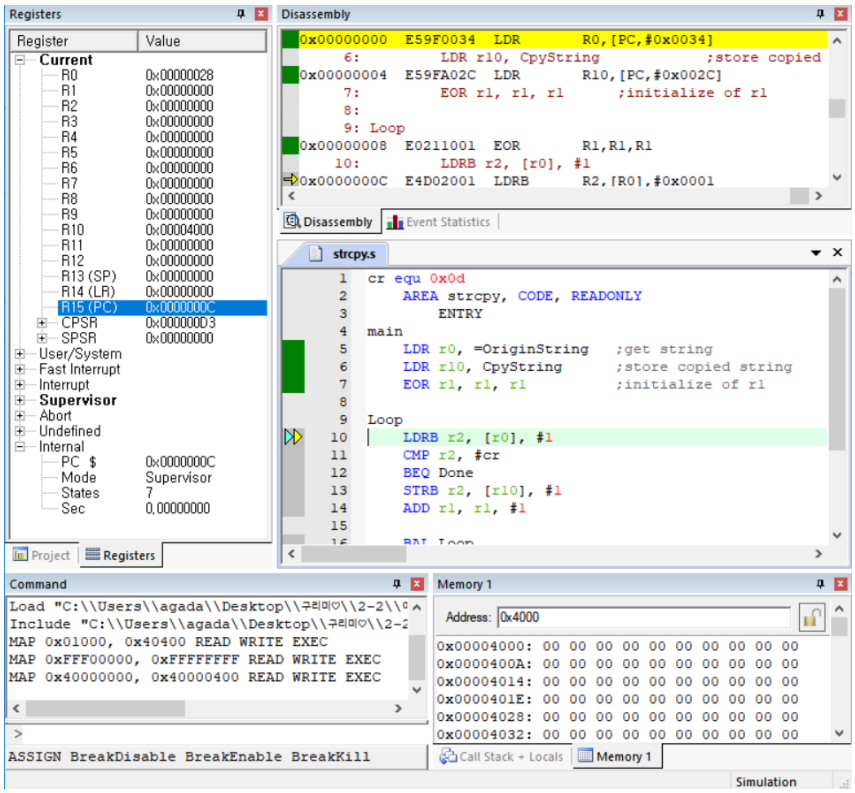
r0에 OriginString을 넣어준다. 이때, OriginString이란 복사할 문자열+cr이고 이 문자열은 memory에 저장이 되어있다. 다음으로 r10에는 CpyString을 대입해준다. CpyString은 복사한 문자열을 담을 memory의 주소이다. 이 주소값은 0x4000으로 할당한다. r1을 초기화 하기 위해 exclusive or 을 거치게 한다.

다음은 Loop Label이다. 이 Loop Label에서는 문자열을 한 글자 씩 읽고 해당 글자를 순서대로 memory에 저장함으로써 strcpy의 본격 실행문이다. r2에 r0의 주소에 저장되어 있는 값을 읽어온다. 후에 r0의 주소를 1늘린다. 만약 r2가 cr이라면, 문자열이 끝난다는 소리이므로 Done문으로 보내준다. 그렇지 않다면, 문자열이 끝나지 않았다는 뜻이다. 따라서 r10의 주소에 r2에 저장된 값을 할당한다. 다음, r10의 주소를 1 늘려준 후, 문자열

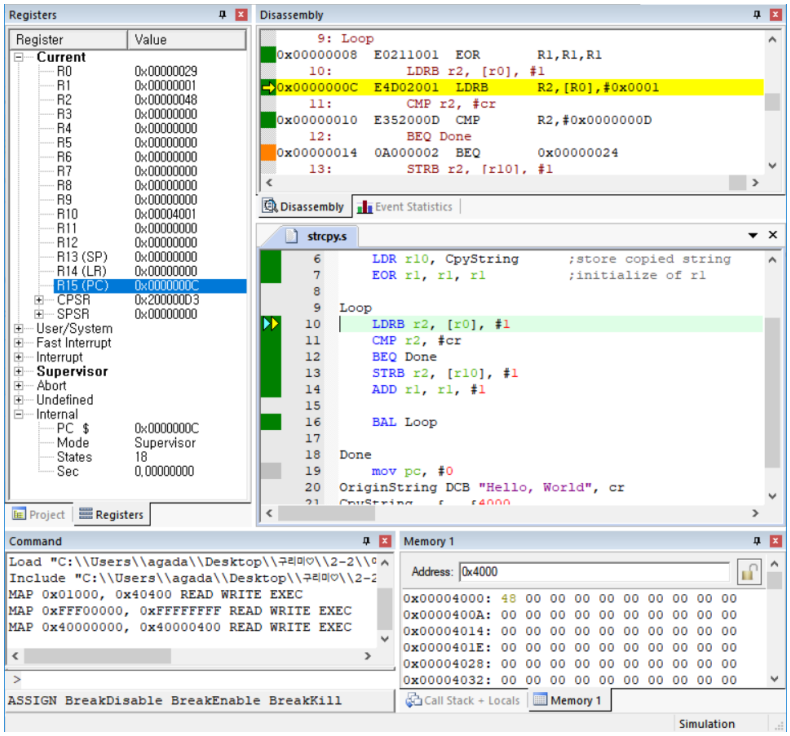
의 길이를 파악하는 r1을 1 높여준다. 후에 Loop문을 다시 돌게 한다.

C.Result

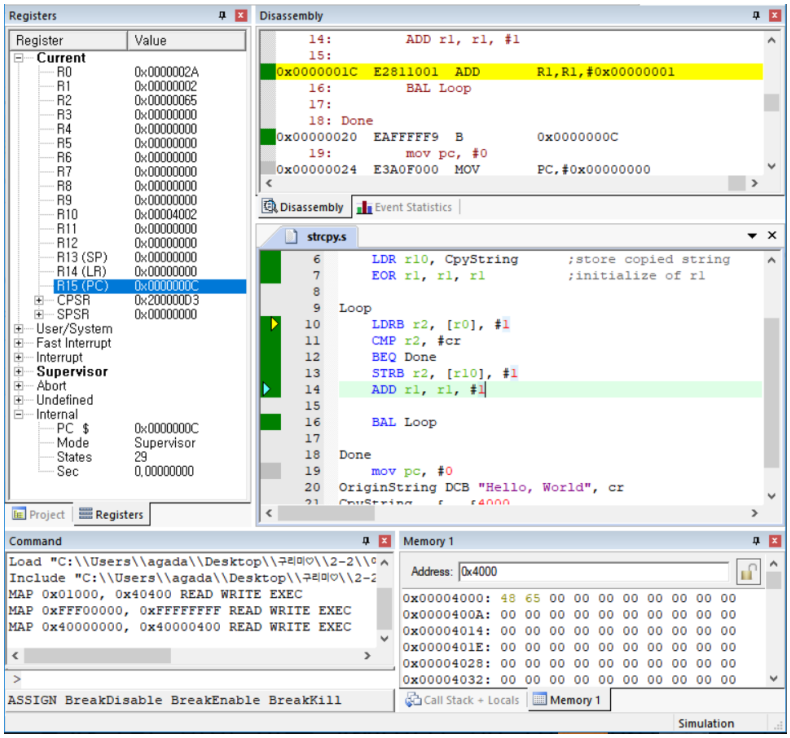
1. Problem1



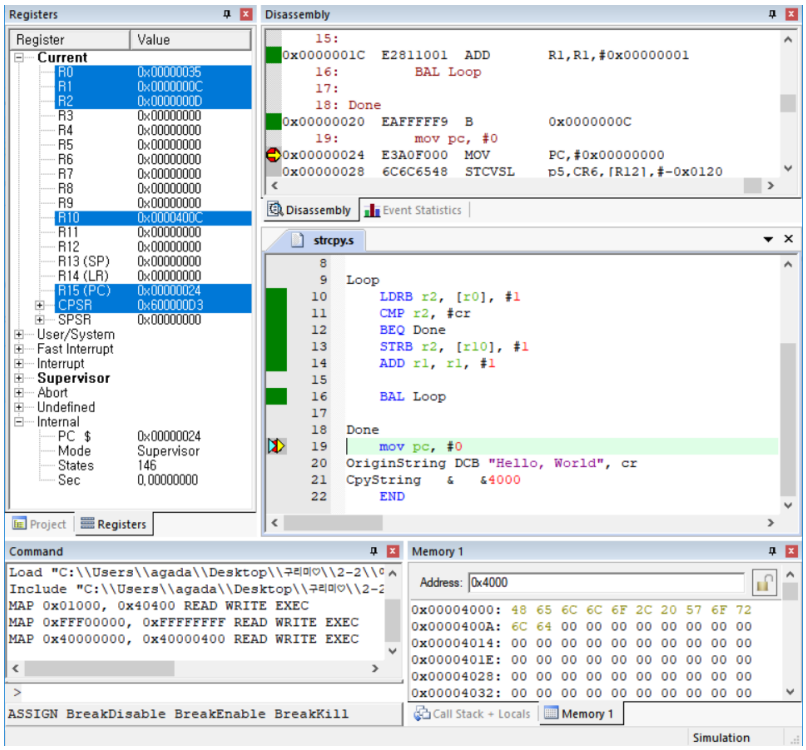
Main Label을 실행했다. 이는 r0와 r10, r1을 초기화 하는 단계이다. 초기화 정보에 따라서 r0에는 "Hello, World"가 담긴 memory의 주소값인 0x28이, r1에는 0이, r10에는 복사할 위치를 가리키는 주소값인 0x4000번지가 저장되어 있는 것을 확인 할 수 있다.



Loop를 처음 돌았을 때를 나타낸다. r2에는 'H'를 나타내는 값이 저장되고 r0의 값이 1 늘어난 것을 확인 할 수 있다. 다음으로 'H' != cr이므로 계속 r2의 값이 0x4000번지에 저장이 되어있으며 r0에 0x4001번지가 저장된다. 문자열의 길이를 나타내는 r1이 하나 증가하고 Loop를 계속 돌게 한다.



Loop를 두번 돌았을 때를 나타낸다. r2에는 'e'를 나타내는 값이 저장되고 r0의 값이 1 늘어난 것을 확인 할 수 있다. 다음으로 'e' != cr이므로 계속 r2의 값이 0x4001번지에 저장이 되어있으며 r0에 0x4002번지가 저장된다. 문자열의 길이를 나타내는 r1이 하나 증가하고 Loop를 계속 돌게 한다.



Loop를 0xC번 돌고 나면, cr을 읽게 된다. 따라서 Done문으로 넘어가게 되는 것을 확인할 수 있다. 후에는 pc가 0이되서 처음부터 다시 실행하게 되는 것을 확인할 수 있었다.,

3. 고찰 및 결론

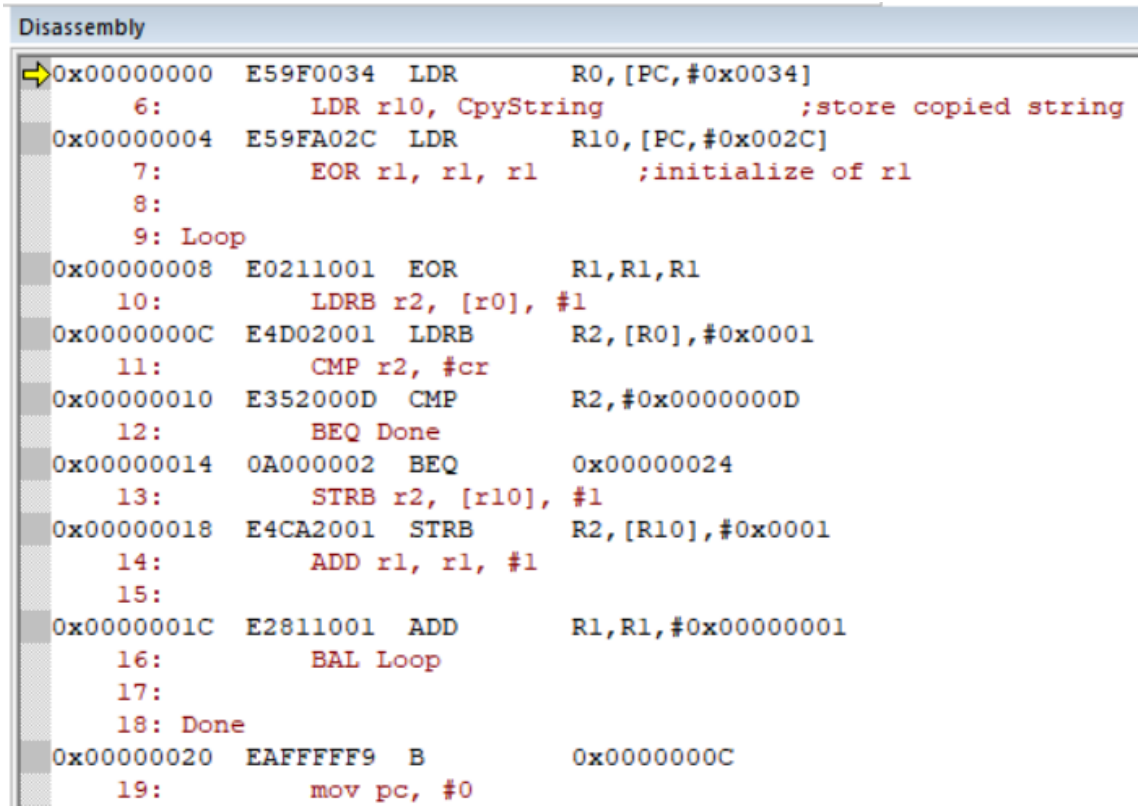
A. 고찰

저번에 다뤘었던 문자열을 이용하여 문자열 복사 함수인 strcpy를 구현하는 문제였다. 프로그래밍 후에 디스어셈블리로 해석하는 것이 이번 과제였다.

비교적으로 프로그래밍을 하는 과정은 단순하고 쉬웠지만 문제는 디스어셈블리어를 해석하는 것이었다. 디스어셈블리는 어셈블리언어의 32-bits 명령어를 hexadecimal로 표현한 것을 의미한다. 이때 이 명령어의 set format이 정해져 있기 때문에 이를 보며 해석해야했다. 하지만 이 format에는 각종 case에 관해 다 다른

명령어 해석법을 가지고 있었으며 P, U, W, L, Rn, Rd등 이해할 수 없는 용어들 투성이었기 때문에 처음에는 너무나 어렵게 다가왔다. 이에 강의자료를 처음부터 끝까지 살펴보니 명령어의 case마다 명령어에 필요한 요소들이 잘 설명이 되있는 것을 알 수 있었다. 이에 디스어셈블리언어의 한 줄씩 해설해갈 수 있었다.

B. 결론



Cond	0	0	I	Opcode				S	Rn	Rd	Operand2											
Cond	0	0	0	0	0	0	A	S	Rd	Rn	Rs				1	0	0	1	Rm			
Cond	0	0	0	0	1	U	A	S	RdHi	RdLo	Rs				1	0	0	1	Rm			
Cond	0	0	0	1	0	B	0	0	Rn	Rd	0	0	0	0	1	0	0	1	Rm			
Cond	0	1	I	P	U	B	W	L	Rn	Rd	Offset											
Cond	1	0	0	P	U	S	W	L	Rn	Register list												
Cond	0	0	0	P	U	1	W	L	Rn	Rd	Offset1				1	S	H	1	Offset2			
Cond	0	0	0	P	U	0	2	L	Rn	Rd	0	0	0	0	1	S	H	1	Rm			
Cond	1	0	1	L	Offset																	
Cond	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn
Cond	1	1	0	P	U	N	W	L	Rn	CRd	CPNum				Offset							
Cond	1	1	1	0	Op1				CRn	CRd	CPNum				Op2			0	CRm			
Cond	1	1	1	0	Op1			L	CRn	Rd	CPNum				Op2			1	CRm			
Cond	1	1	1	1	SWI Number																	

1) E59F0034 (LDR r0, =OriginString) /1110010110011111 0000 000000110100

   1110 -> condition (AL -> always)

   01 -> Load/ Store Word signal

   0 -> offset을 12-bit immediate로 지정

   1 -> pre-index

   1 -> up

   0 -> unsigned word

   0 -> write back X

   1 -> Load

   1111 -> base register

   0000 -> source/ destination register (r0)

   000000110100 -> 0x34에 r0의 값 저장

2) E59FA02C (LDR r10, CpyString) / 11100101100111111010000000101100

   1110 -> condition (AL -> always)

   01 -> Load/ Store Word signal

   0 -> offset을 12-bit immediate로 지정

   1 -> pre-index

   1 -> up

   0 -> unsigned word

0 -> write back X

1 -> Load

1111 -> base register

1010 -> source/ destination register (r10)

000000101100 -> 0x2C에 r10의 값 저장

3) E0211001 (EOR r1, r1, r1) / 11100000001000010001000000000001

1110 -> condition

00 -> Data processing / PSR Transfer signal

0 -> I (offset을 12-bit immediate로 지정)

0001 -> opcode (EOR)

0 -> unsigned

0001 -> base register

0001 -> source/ destination register (r1)

000000000001 -> operand2

4) E4D02001 (LDRB r2, [r0], #1) / 11100100110100000010000000000001

1110 -> condition (AL -> always)

01 -> Load/ Store Word signal

0 -> offset을 12-bit immediate로 지정

0 -> post-index

1 -> up

1 -> unsigned byte

0 -> write-back X

1 -> Load

0000 -> base register -> r0

0010 -> source/ destination register -> r2

000000000001 -> offset (후위 증가이므로 +1)

5) E352000D (CMP r2, #cr) / 11100011010100100000000000001101

1110 -> condition (AL -> always)

00 -> Data processing / PSR Transfer signal

1 -> I (offset의 형태를 지정)

1010 -> opcode (CMP)

1 -> set condition codes

0010 -> base register -> r2

0000 -> source/ destination register

0000 -> immediate alignment

00001101 ->offset register (0x0d)

6) 0A000002 (BEQ Done) / 00001010000000000000000000000010

0000 -> condition

101 -> Branch signal

0 -> Load

00000000000000000000000010 -> offset

7) E4CA2001 (STRB r2, [r10], #1) / 11100100110010100010000000001

1110 -> condition (AL -> always)

01 -> Load/ Store Word signal

0 -> offset을 12-bit immediate로 지정

0 -> post-index

1 -> up

1 -> unsigned byte

0 -> write-back X

0 -> Store

1010 -> base register -> r10

0010 -> destination/ source register -> r2

00000001 -> offset (후위 증가이므로 +1)

8) E2811001 (ADD r1, r1, #1) / 11100010100000010001000000000001

1110 -> condition (AL -> always)

00 -> Data processing / PSR Transfer signal

1 -> I (offset의 형태를 지정)

0100 -> opcode (ADD)

0 ->set condtion codes

0001 -> base register -> r1

0001 -> source/ destination register -> r1

000000000001 -> operand2

9) EAFFFFF9 (BAL Loop) / 11101010111111111111111111111001

1110 -> condition (AL -> always)

101 -> Branch Signal

0 -> Load

111111111111111111111001 -> offset

10) E3A0F000 (MOV pc, #0) / 11100011101000001111000000000000

1110 -> condition (AL -> always)

00 -> Data processing / PSR Transfer signal

1 -> I (offset의 형태를 지정)

1101 -> opcode (CMP)

0 -> set condition codes

0000 -> base register

1111 -> source/ destination register

0000 -> immediate alignment

00000000 ->offset register (0x0d)

4.Performance

5. 참고문헌

이형근/ 어셈블리프로그래밍/ 광운대학교/ 2018