Report – HW 6

1. Problem & Purpose

i. Purpose

예제 코드를 수행하여 기본적인 동작에 대해 이해하고, register 값을 확인하는 법을 숙지하기 위해 해당 과제를 진행한다.

ii. Problem

strcpy함수 구현

- 단, 이번 보고서에는 반드시 disassembly 화면을 캡쳐하여 다음 언급이 반드 시 포함돼야 함.
  - pseudo instruction이 어떻게 변경되고, 변경된 instruction이 어떤 원리로 프로그 램에서 동작하는지에 대한 설명
  - 또한, 사용된 모든 명령어의 disassembly 화면을 캡쳐하여 32bit의 코드가 어떤 방식으로 이뤄져 있는지에 대한 해석

2. Used Instruction

MOV // END // LDR // LDRB // STR // B, BEQ // DCB // CMP

i. MOV Rd, operand2 : operand2에 있는 값을 Rd에 저장한다.

ii. END : Assembly code가 끝났음을 의미하는 Instruction

iii. LDR Rd, <address> : addressing에 있는 값을 Rd에 저장한다.

iv. LDRB Rd, <address> : address에 있는 값을 Byte단위로 읽어서 Rd에 저장한다.

v. STR Rd <address> : Rd에 있는 값을 address에 저장한다.

vi. B label : 작성한 label의 첫 번째 instruction으로 이동

A. BEQ label : CPSR에 있는 Z flag 가 1일 때 실행한다.

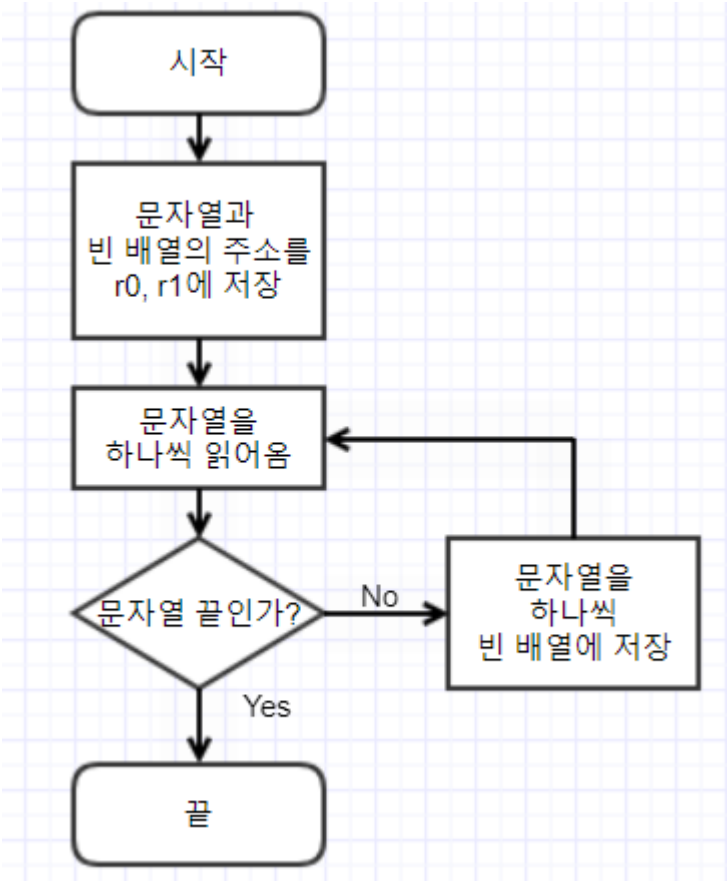
vii. DCB : 1 byte 단위로 메모리를 할당 및 해당 값으로 초기화{

viii. CMP Rn operand2 : Rn와 operand2에 있는 값을 빼서 그 결과에 따라 CPSR에 있는 N, Z, C, V flag를 설정한다.

3. Design(Flow chart)

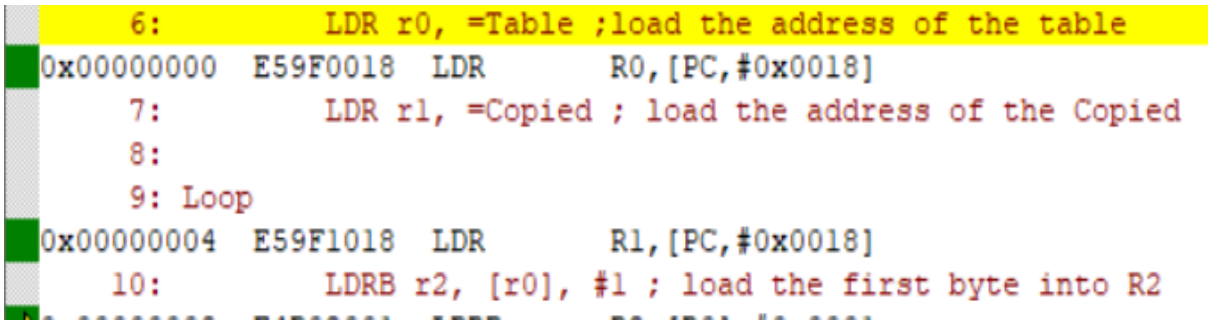
i. 설계한 내용의 Flow chart

A. strcpy



4. Conclusion

[pseudo instruction이 어떻게 변경되고, 변경된 instruction이 어떤 원리로 프로그램에서 동작하는지에 대한 설명]



Main

LDR r0, =Table ;load the address of the table

LDR r1, =Copied ; load the address of the Copied

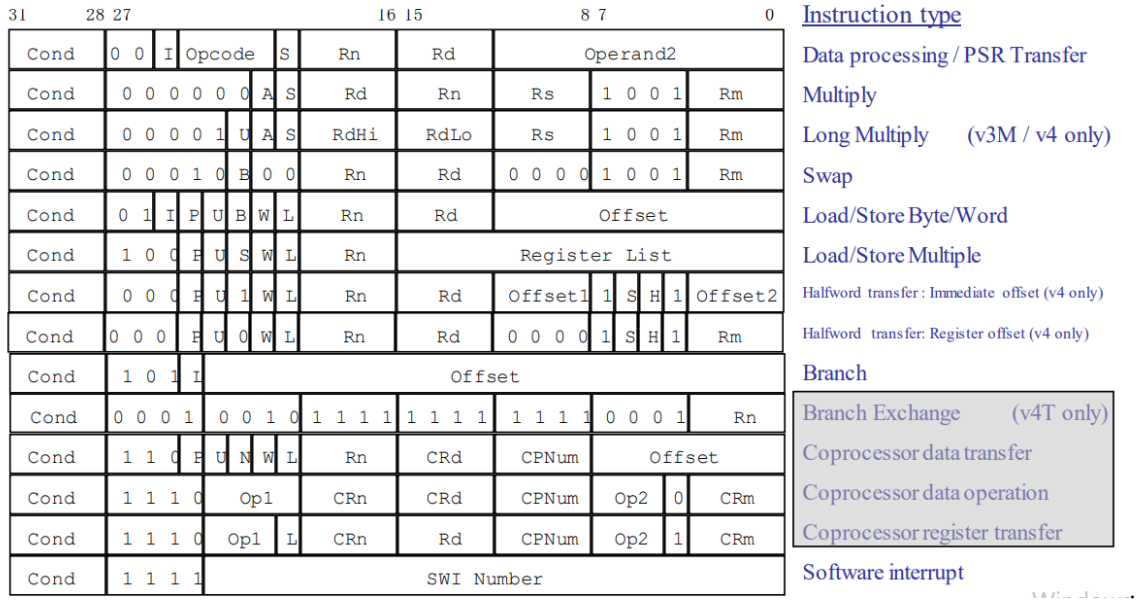
위 코드 두 줄이 LDR r0, [PC, #0x0018] LDR r1, [PC,#0x0018]로 각각 바뀌었다. 원래는 expression 자리에 Label값이 왔지만, 어셈블러가 [PC, #0x0018]로 변경시켰다. 코딩할 때는 사람이 이해하기 쉽게 =Table같은 label 값을 넣었지만, 실제로는 [PC + #0x0018]에 있는 Table 주소 값을 r0에 넣어야 한다.

[사용된 모든 명령어의 disassembly 화면을 캡쳐하여 32bit의 코드가 어떤 방식으로 이뤄져 있는지에 대한 해석]

```
     6:          LDR r0, =Table ;load the address of the table
0x00000000  E59F0018  LDR        R0,[PC,#0x0018]
     7:          LDR r1, =Copied ; load the address of the Copied
     8:
     9: Loop
0x00000004  E59F1018  LDR        R1,[PC,#0x0018]
    10:          LDRB r2, [r0], #1 ; load the first byte into R2
0x00000008  E4D02001  LDRB       R2,[R0],#0x0001
    11:          CMP r2, #0 ; 문자열 끝인가??
0x0000000C  E3520000  CMP        R2,#0x00000000
    12:          BEQ Out
0x00000010  0A000001  BEQ        0x0000001C
    13:          STR r2, [r1], #1
0x00000014  E4812001  STR        R2,[R1],#0x0001
    14:          B Loop
    15:
    16: Out
0x00000018  EAFFFFFA  B          0x00000008
    17:          MOV pc, #0
0x0000001C  E3A0F000  MOV        PC,#0x00000000
0x00000020  00000028  ANDEQ      R0,R0,R8,LSR #32
0x00000024  00000038  ANDEQ      R0,R0,R8,LSR R0
0x00000028  6C6C6548  STCVSL     p5,CR6,[R12],#-0x0120
0x0000002C  57202C6F  STRPL      R2,[R0,-PC,ROR #24]!
0x00000030  646C726F  STRVSBT    R7,[R12],#-0x026F
0x00000034  00000000  ANDEQ      R0,R0,R0
0x00000038  0000006C  ANDEQ      R0,R0,R12,RRX
0x0000003C  00000057  ANDEQ      R0,R0,R7,ASR R0
0x00000040  00000064  ANDEQ      R0,R0,R4,RRX
```

디스어셈블리 화면을 보면 맨 왼쪽에는 0x00000000 16진수 숫자가 있는데, 이것은 명령어의 주소값이다. 주소값 다음엔 E59F0018 같은 16진수 값이 있다. 이것은 명령어 format에 따라 이진수 값을 16진수로 바꿔 놓은 것이다

ARM Instruction Set Format



Load/Store 명령어를 설명해보겠다.

Load/Stroe 명령어의 상위 31~28bit 까지는 Condition 값이 온다. 31bit는 N flag 값, 30bit는 Z flag 값, 29bit는 C flag 값, 28 bit는 V flag 값이다.

25번 bit는 I라고 적혀있는데, 이것은 Immediate Value인가, 아니면 Register-based offset인가를 의미한다. 0이면 Immediate Value, 1이면 Register-based offset이다.

24번 bit는 P라고 적혀있는데, 이것은 Pre index인가, Post index인가를 의미한다. 0이면 Post-index, 1이면 Pre-index이다.

23번 bit는 U라고 적혀있는데, 이것은 Pointer direction이 증가하는가 감소하는가를 의미한다. U는 UP/DOWN의 UP에서 따온 것이다. 0이면 감소하고, 1이면 증가한다.

22번 bit는 B라고 적혀있는데, 이것은 Byte단위로 access 할 것인가, Word access할 것인가를 의미한다. 0이면 Word 단위로 읽고, 1이면 Byte 단위로 읽는다

21번 bit는 W라고 적혀있는데, 이것은 Write-Back를 의미한다. 0이면 Write-back을 하지 않고, 1이면 Write back을 수행한다.

20번 bit는 L이라고 적혀있는데, 이것은 Load할 것인가, Store할 것인가를 의미한다. 0이면 Store이고, 1이면 Load한다.

19~16bit는 Base Register를 의미한다.

15~12bit는 destination register를 의미한다.

11~0bit는 Operand2를 의미한다. 위에서 설명한 25번 bit와 관련이 있다. 25번 bit가 0이면 Immediate Value를 의미하는데, 이렇게 되면 11bit~0bit는 12bit Immediate Value가 올 수 있다. 반면 25번 bit가 1이면 Register-based offset을 의미하는데, 이렇게 되면 11bit~0bit가 또 쪼개진다. 11bit~7bit는 shift legth를 의미하고, 6bit~ 5bit는 Type을 의미하고, 3bit~0bit는 Register를 의미한다.

그렇다면 코드에 있는 LDR과 LDRB를 비교해보자.

LDR r0, =Table ;load the address of the table

➔ Disassemble 결과 : LDR R0, [PC, #0x0018]

16진수 : E59F0018

2진수 : 11100101100111110000000000011000

LDRB r2, [r0], #1 ; load the first byte into R2

➔ Disassemble 결과 : LDRB [R2], #0x0001

16진수: E4D02001

2진수 : 11100100110100000010000000000001

두 명령어 중에 비교해볼 것은 **24번 bit(P)**와 22번 bit(B)다.

(1) LDR r0, =Table : 1110 01 0 1 1 0 0 1 1111 0000 0000 0001 1000

(2) LDRB r2, [r0], #1 : 1110 01 0 0 1 1 0 1 0000 0010 0000 0000 0001

하늘색 표시로 된 부분은 24번째 bit로 pre index냐 post index냐를 가른다. 명령어 (1)번은 disassemble 결과 pre index로 나타났으며 24번째 bit가 set 되었다. 명령어 (2)번 명령어는 post index므로 24번째 bit가 set 되지 않아 0이었다.

노란색 표시로 된 부분은 22번째 bit로 Byte 단위로 access하냐 Word 단위로 access 하냐를 가른다. 명령어(1)번은 Word 단위로 읽기 때문에 0이었고, 명령어 (2)번은 Byte 단위로 읽기 때문에 1이었다.

다음은 Branch 명령어를 설명하겠다.

Branch 명령어의 상위 31~28bit 까지는 Condition 값이 온다. 31bit는 N flag 값, 30bit는 Z flag 값, 29bit는 C flag 값, 28 bit는 V flag 값이다.

24번 bit는 L이라고 적혀있는데, 이것은 Link bit를 의미한다. 0이면 Branch, 1이면 Branch with Link이다.

23~ 0bit는 Offset 값이 온다. Label의 주소를 가리킨다.

코드에 있는 브랜치 명령어를 살펴보겠다.

BEQ Out ;r2==0이면 loop 아웃

➔ Disassembly 결과 : BEQ, #0x0000001C

16진수 : 0A000001

2진수 : 00001010000000000000000000000001

B Loop

➔ Disassembly 결과 : B, #0x00000008

16진수 : EAFFFFFA

2진수 : 11101010111111111111111111111010

(1) BEQ Out : 0000 101 0 00000000000000000000001

(2) B Loop : 1110 101 0 11111111111111111111010

하늘색 표시가 된 부분은 Link bit다. 명령어 (1)번과 명령어 (2)번 모두 branch with link를 하지 않기 때문에 Link bit가 0으로 설정되어 있다. 노란색 표시가 된 부분은 label의 주소를 의미하는데 pc에서 얼마만큼 떨어진 곳으로 Jump할 건지 어셈블러에서 계산한다.

CMP 명령어, Mov 명령어를 설명하겠다

CMP 명령어, MOV 명령어는 Data processing 명령어 타입에 속한다.

31~28bit는 Condition 값이 온다. 31bit는 N flag 값, 30bit는 Z flag 값, 29bit는 C flag 값, 28 bit는 V flag 값이다

25번 bit는 I라고 적혀있는데, 이것은 Operand2가 Immediate Value인지, Register인지 의미한다. 0이면 register고, 1이면 Immediate Value다.

24~21bit는 Opcode가 온다. Opcode는 Operation Code로 이 값을 통해 ADD나 SUB, CMP 등 연산을 구별한다.

20번 bit는 S라고 적혀있는데, 이것은 condition code를 저장할지 말지를 결정한다. 0이면 CPSR의 값을 바꾸지 않고, 1이면 CPSR의 값을 바꾼다.

19~16 bit는 1st operand register를 의미한다

15~12 bit는 Destination register를 의미한다.

11~0bit는 2nd operand를 의미한다.

코드에 있는 data processing 명령어를 살펴보겠다.

CMP r2, #0 ; 문자열 끝인가??

➔ Disassembly 결과 : CMP R2, #0x00000000

16진수 : E3520000

2진수 : 11100011010100100000000000000000

MOV pc, #0 ;return

➔ Disassembly 결과 : MOV PC, #0x00000000

16진수 : E3A0F000

2진수 : 11100011101000001111000000000000

(1) CMP r2, #0 : 1110 00 1 1010 1 0010 0000 000000000000

(2) MOV pc, #0 : 1110 00 1 1101 0 0000 1111 000000000000

두 명령어 중에 비교해볼 것은 24~21bit(Opcode)와 20번 bit(S)다.

하늘색 표시가 된 부분은 Opcode다. 명령어 (1)번은 CMP명령어로, 1010이 왔다. 명령어 (2)번은 MOV 명령어로 1101가 왔다. 노란색 표시가 된 부분은 condition flag를 변경할지를 결정한다. CMP 명령어는 항상 CPSR을 변경하기 때문에 20번 bit가 1이다. 반면 MOV 명령어에는 S옵션을 주지 않았기 때문에 0으로 되어있다.

5. Reference

- 강의교안 example(6 page) 참고
- http://www.riscos.com/support/developers/asm/instrset.html - OPCODE 참조
- https://www.cs.uregina.ca/Links/class-info/301/ARM-addressing/lecture.html
- http://www.hipenpal.com/tool/binary-octal-decimal-hexadecimal-number-converter-in-korean.php 진수 변환