어셈블리프로그램설계및실습 보고서

과제 주차: 7주차

학 과: 컴퓨터정보공학부

담당교수: 이형근 교수님

실습분반: 화 67 목 5

학 번: 2019202021

성 명: 정성엽

제 출 일: 2022.11.11(금)

1. Problem Statement

Pseudo instruction이 assembler에 의해 어떻게 실제 instruction으로 변환되는지 확인하고 이해한다. 그리고 32bit 명령어들의 구조를 이해하여 Disassembly를 해석하는 방법을 이해하고 Label의 이름이 실제 주소임을 이해하여 어셈블리 프로그래밍에 적용해 본다.

2. Design

A. Pseudo instruction

실제 기계어로 1대1 치환 되는 명령어는 아니지만 Assembler 단에서 변환되어 동작하는 명령어이다. 이때 여러 개의 명령어 묶음으로 변환되어 실행되는 것으로 Disassembly에서 Pseudo 명령어가 어떻게 해석되는지 확인 가능하다. 예를 들어 ADR 명령어는 ADD와 SUB 명령어의 묶음으로 해당 주소를 PC에서 더하거나 빼서 구하고 PC의 값을 Rn으로 저장한다.

B. Assembler

명령어를 기계어와 1대1로 치환하고 CPU가 직접 읽을 수 있는 값으로 변환한다. ARM CPU는 32bit 명령어를 set으로 사용하므로 모든 명령어는 32bit의 길이의 기계어로 변환된다.

C. Disassembly

위의 Assembler를 통해서 기계어로 변환된 명령어와 명령어의 묶음으로 이루어진 Peseudo 명령어가 어떻게 32bit 기계어로 변환되는지 Disassembly를 통해서 해당 명령어의 변환을 확인할 수 있다.

D. 과제 수행

먼저 cr은 0x0d로 정의한 후 STR1에 복사할 문자 값을 저장하고 STR2에는 후에 저장할 주소를 저장한뒤 ADR 명령어를 통해 STR1의 주소를 가져와 저장하고 LDR 명령어를 통해 STR2의 주소를 불러온다.

Loop label로 이동해서 가져온 문자를 하나씩 가져와 cr과 비교하고 같은 경우는 문자가 더이상 없는 것이므로 loop를 탈출하고 아닌 경우 loop를 계속 돌아 문자 하나씩 옮겨 저장한다.

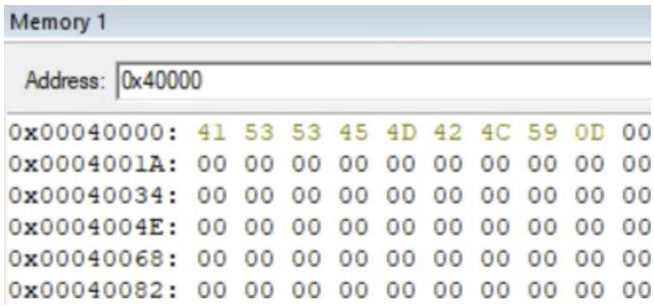
마지막에는 cr(0x0d)를 저장함으로써 문자의 끝을 나타내도록 저장하고 STR2 주소에 저장하고 끝낸다.

3. Conclusion

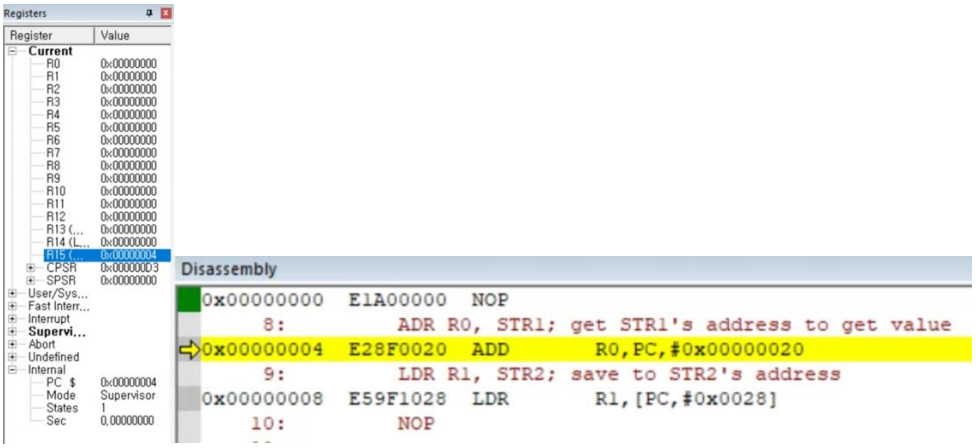
결론

```
STR1
    ALIGN
    DCB "ASSEMBLY",cr ;  source in cpy
STR2 & &00040000; save copy data in address
    END
```

STR1에 "ASSEMBLY"라는 문자열을 저장해두었고 STR2에는 0x00040000의 주소를 불러놓았다. 모든 명령어를 진행하였을 때 0x00040000주소부터 1byte 씩 문자가 복사되었음을 확인할 수 있다.



과정



레지스터에 있는 PC와 Disasembly에 있는 왼쪽 값이 같음을 볼 수 있고, 해당 줄의 E28F0020의 문자열 같은 경우 해석하면 , 1110-001-0100-0-1111-0000-000000100000 을. AL- 001- ADD -S-R15-R0-0x020으로 Data Processing Instructions의 구조를 가진다. ADR R0, STR1 명령어는 Disassembly에서 다른 명령어의 묶음으로 변환되므로 R0에 PC+0x00000020 만큼의 값을 더하여 넣는 명령어로 ADD가 실행되어 해당 주소에 STR1 라벨의 주소인 것이다.

```
0x00000008  E59F1028  LDR       R1,[PC,#0x0028]
    10:           NOP
```

E59F1028는 1110-01-01-1001-1111-0001-000000101000으로 나타낼 수 있고 AL-01-0(Immediate Offset) 1(Pre/post) – 1(Destination reg)0(word quantity transfer)0(write

back)1(Load)-R15-R1-0x028로 표현 가능하다 이는 STR2의 주소값을 R1에 저장하는 동작을 한다.

```
    13:         LDRB R2, [R0], #1;get R0's 1byte to R2
0x00000010  E4D02001  LDRB      R2,[R0],#0x0001
```

E4D02001은 1110-01-00-1101-0000-0010-000000000001으로 나타낼 수 있고 AL-0(Immediate offset)1(Pre/post)-1(up/down)1(byte/word)0(write-back)1(load/store)-R0-R2-0x0001(offset reg)로 해석이 가능하다 따라서 1byte 씩 R1에 저장된 주소의 메모리값을 읽고 R2에 저장한 후에 R1에 저장된 주소를 #1만큼 이동시킨다.

```
    14:         CMP R2, #cr; compare R2 and cr(0x0d), if NULL
0x00000014  E352000D  CMP       R2,#0x0000000D
```

E352000D는 1110-001-1010-1-0010-0000-000000001101으로 나타낼 수 있고 AL-001-SUB-1(set condition)-R2-0-0x00D로 해석가능하고 R2-0x00D를 하고 condition을 set 하는 동작을 수행한다.

```
    15:         BEQ Finish;goto Finish
    16:
0x00000018  0A000001  BEQ       0x00000024
```

0A000001은 0000-101-0-00000000000000000000000001이고 EQ-101-0(L)-0x000001(offset으로 점프할 구간의 labal value)이고 Branch instruction 구조이다. 따라서 z flag가 1로 set 되면 해당 라벨의 주소로 pc를 이동한다.

```
    17:         STRB R2, [R1], #1; //store R2's 1byte to R1 address
0x0000001C  E4C12001  STRB      R2,[R1],#0x0001
```

E4C12001은 1110-01-00-1100-0001-0010-000000000001이고 AL-0(Immediate offset)1(Pre/post)-1(up/down)1(byte/word)0(write-back)0(load/store)-R1-R2-0x001(offset register)로 해석 가능하다. 이 코드는 1byte 씩 R1에 저장된 주소의 메모리에 R2의 값을 저장하고 주소를 #1만큼 이동시키는 명령어이다.

```
    18:         B LOOP;goto Loop
    19:
    20:
    21: Finish
0x00000020  EAFFFFFA  B         0x00000010
```

EAFFFFFA는 1110-101-0-11111111111111111111010이고 AL-101-0(L)-0xFFFFFA(offset으로 점프할 구간의 label value)이고 BEQ와 마찬가지로 Branch Instruction 구조이다. 따라서 Label Loop으로 pc를 이동한다.

```
    22:         MOV R2, #cr; ; R2 = cr(0x0d)
0x00000024  E3A0200D  MOV       R2,#0x0000000D
```

E3A0200D는 1110-001-1101-0-0000-0010-000000001101이고 AL-001-MOV-0(S)-0(Rn X)-R2-0x00D로 해석할 수 있고 Processing Instructions 구조이며 R2에 0x000D 값을 저장한다.

```
    23:         STRB R2,[R1],#1 ; store cr 0x0d R3
0x00000028  E4C12001  STRB      R2,[R1],#0x0001
```

E4C12001은 1110-01-00-1100-0001-0010-000000000001이고 AL-0(Immediate offset)1(pre/post)-1(up/down)1(byte/word)0(write-back)0(load/store)-R1-R2-0x0001(offset register)로 해석할 수 있고 1byte 씩 R1에 저장된 주소의 메모리에 R2의 값을 저장하고 R1의 주소를 #1만큼 이동시키는 동작을 수행한다.

4. Consideration

이번 과제를 통해 Pseudo instruction이 무엇인지 그리고 왜 사용하는지를 알 수 있었고 다른 명령어들이 기계어와 1대1 매칭 되는 것과 달리 assembler가 여러 개의 명령어를 조합하여 명령어를 수행한다는 것을 알았다. 그러므로 disassembly를 통해 Pseudo instruction의 코드를 보면 다른 기계어 묶음으로 변환되어 있고 앞서 설명한 ADR 명령어는 ADD 또는 SUB 명령어로 해당 주소를 더하거나 빼서 pc 값을 Rn으로 저장한 것을 확인할 수 있었다.

Assembly 명령어들은 Thumb 명령어가 아니라면 32bit로 이루어져 있기 때문에 32bit 명령어 set을 분석하면 명령어의 수행을 알 수 있으며 이번 과제를 통해 instruction set format을 분석하여 명령어 구조를 익혔다.

5. Reference

1) 이형근 교수님/ 어셈블리프로그램설계및실습/광운대학교 컴퓨터정보공학부/ 2022