어셈블리프로그램설계및실습 보고서

과제 주차: 3주차

학 과: 컴퓨터정보공학부

담당교수: 이형근 교수님

실습분반: 화 67,목요일 5

학 번: 2019202021

성 명: 정성엽

제 출 일: 2022.10.09(일)

1. Problem Statement

Memory로부터 값을 받아 이를 이용해 원하는 연산을 수행해 보고 Assembly code를 작성할 때 perfomance를 생각하며 작성해 본다. Branch 명령어와 conditional execution의 차이를 알아본다. 마지막으로 Memory를 할당하여 data를 저장하여 본다.

2. Design

A. DCD, DCB 명령어를 통한 배열 또는 문자열

DCD는 4바이트 DCB는 1바이트로 연속된 숫자, 문자를 특정 메모리에 저장하는 것으로 저장된 숫자 또는 문자들은 임의의 메모리에 저장된 후 label을 통하여 레지스터에 특정되어 해당 주소를 불러와 데이터에 접근할 수 있다. 또한 문자열을 저장할 때는 마지막에 ",0"을 추가하여 이 문자열이 끝남을 표현해줘야 한다. 일반적인 숫자는 10진수 "2_"가 붙으면 이진수 "0x"가 붙으면 16진수가 저장된다.

B. Label과 branch 명령어

Assembly 언어 코드를 구현할 때 label을 두어 'B' 명령어를 통해 원하는 label로 이동할 수 있다. 이때 'B' 명령어는 이전에 다루었던 conditional field를 통해 조건부 실행할 수 있으며 이를 이용해 반복문을 구현할 수 있다.

C. MUL 명령어

MUL R1, R2, R3는 곱연산을 실행하는 명령어로 R1 = R2*R3를 실행한다.

D. Shift 명령어

자릿수를 옮기는 명령어이다. 해당 문제에서는 LSL 명령어를 사용했으며 왼쪽으로 bitstream을 옮기는 명령어이다. 1번 shift 할 때, 이진수의 특성으로 값이 2배씩 증가한다.

E. Problem1

Strcmp 함수를 작성하는 문제로 두 문자열을 비교하여 같은 값인 경우 0x0A를, 다른 경우 0x0B의 값을 4000번지 주소에 저장해야 한다.

i. 값을 저장할 주소를 세팅하고 비교할 문자열 두개를 DCB 명령어를 통해 세팅한다.

ii. 문자열 2개에 대한 주소를 R0와 R1 레지스터에 불러오고 결과값을 저장할 주소를 R12에 할당하여 불러온다. 비교하기 전임으로 후에 주소로 넘길 값인 R11에 0x0B(#11)을 미리 저장해 초기화 한다.

iii. R2와 R3에 문자열이 저장된 주소에서 첫번째 문자의 값을 불러온다. 후에 CMP

명령어를 통해 이 두개를 비교하고 다른 경우 R11에 11을 저장하고 Finish 라벨로 이동한다. 이동하지 않고 만약 둘 중 하나가 0의 값을 가지고 있다면 Finish 라벨로 이동한다. 위 두개가 이뤄지지 않고 비교했을 때 같다면 R11에 10을 저장하고 Loop로 다시 이동하여 반복한다.

iv. Finish label로 이동한 경우 R11에 저장된 값을 R12의 주소에 저장한다.

F. Problem2

배열에 저장된 숫자를 오름차순으로 정리하여 4000번지에 저장한다.

i. 후에 배열을 저장할 주소를 세팅하고 DCB 명령어를 이용해 배열 숫자를 세팅한다.

ii. R0에 배열의 주소를 가져오고 R1에는 후에 배열을 저장할 주소를 저장한다. R12에는 9를 저장하여 몇 번 반복했는지 카운트한다.

iii. R2에 R0의 주소부터 10개의 위치를 불러오고 1자리씩 접근하여 값을 R3에 불러오도록 한다. 이 작업이 수행한 후에는 R12의 카운트를 1씩 줄인다. 후에 R12 카운트가 0이 되면 Finish label로 이동하여 끝낸다.

G. 11+13+15+ … + 27+29를 계산하여 결과값을 4000번지에 저장한다.

i. Loop를 이용해 연산할 경우 숫자는 #1만 사용 가능하고 shift 연산을 이용해야 한다. shift연산을 할 경우가 한 칸당 2배가 늘어남을 생각한다.

- 저장된 주소를 세팅하고 R0에 불러온다. R1에 #1을 저장하고 LSL을 이용하여 2,4,8을 R2~R3까지 저장하고 각각 조합하여 더하여 R5에는 3 R6에는 11을 저장한다. R7은 1부터 카운트하여 10번 반복 시 끝나도록 R8에는 10이란 숫자를 할당한다.
- 빈공간인 R9에 R6의 값을 저장하고 다음에 2를 더한 값을 더 더해야 하므로 R6에 2를 더한다. 만약 R7과 R8이 같다면 Finish라벨로 이동하고 다르다면 R7에 1을 더하고 Loop로 이동한다.
- Finish라벨로 이동했다면 R9값을 R0에 저장된 주소에 저장한다.

ii. 일반화된 식인 n(n+10)을 이용한 방법도 shift 연산을 이용해 10을 만든다.

- 위와 같이 LSL 명령어를 이용하여 1,2,8을 구현하고 ADD 명령어를 이용하여 10을 구현하여 R4에 저장한다. 또한 이를 두 배를 하여 R5에 저장하여 20을 구현하고 MUL 어셈블리 명령어를 이용해 두 값을 곱하여 결과값을 R6에 저장하고 R0에 저장된 주소 값으로 저장한다.

iii. Unrolling을 이용하여 구하는 경우 위와 같이 MOV와 shift 연산을 이용하여 11을 저장하여 진행하되 처음과 다르게 전체적 반복은 2번만 하여 내부에서 5번의 연산을 실행한다.

- 저장할 주소를 세팅하고 R0에 이를 저장한다. 그리고 R1에 1을 저장하고

LSL을 이용해 2,8을 구현하고 ADD 명령어를 이용해 R4에는 3, R5에는 11 또한 구현해 저장한다.

- 빈 공간인 R7에 R5 값을 더하고 R5에 2를 더하는 연산을 5번 반복하고 이후에 이를 카운트하는 R6에 1씩 더하여 2와 같은 경우 Finish 라벨로 이동하도록 한다.

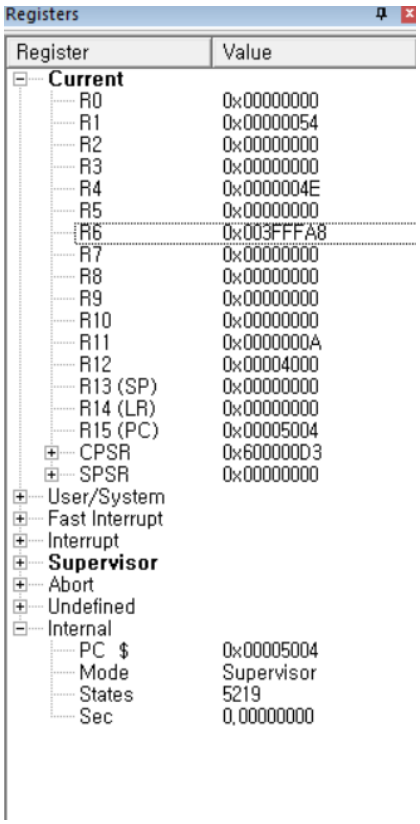
3. Conclusion

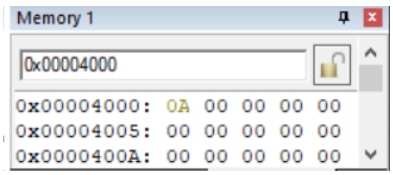
A. Problem1

```
Build Output
linking...
Program Size: Code=92 RO-data=0 RW-data=0 ZI-data=0
".\Objects\Problem1.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:00
```

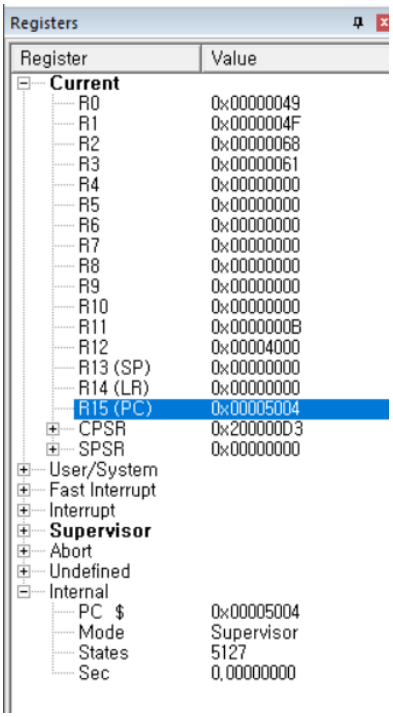
코드 사이즈는 92이다.

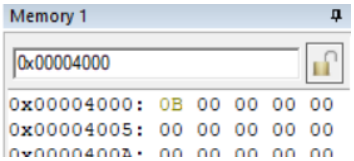
i. 같을 때



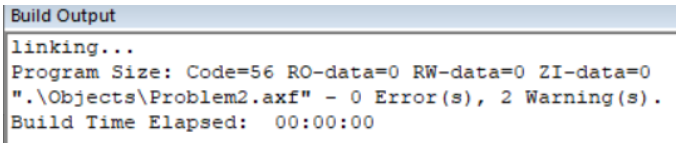


ii. 다를 때

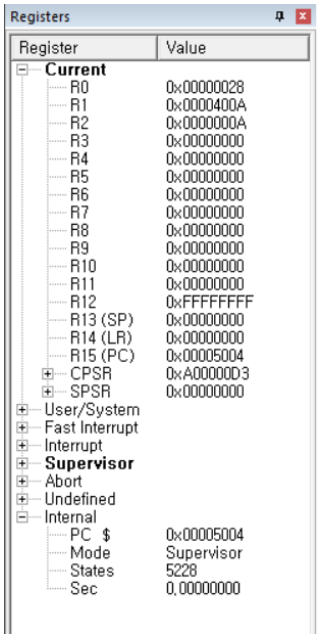


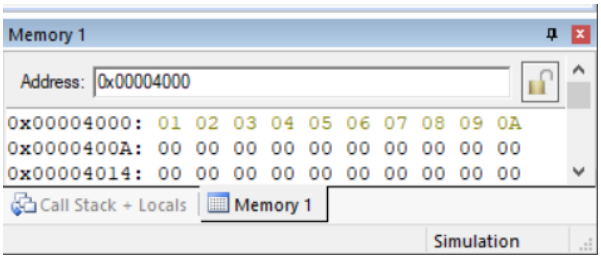


B. Problem2



코드 사이즈는 56이고 2개의 Warnings가 존재하는데 이는 반복문을 탈출하고 나서 Finish 라벨 이후 아무것도 없는데 바로 아래 주소 값을 저장할 TEMPADDR1이 존재해서 있으며 다른 하나는 숫자 배열은 마무리를 ,0으로 할 필요 없으나 해당에 대한 경고를 주는 것이다.





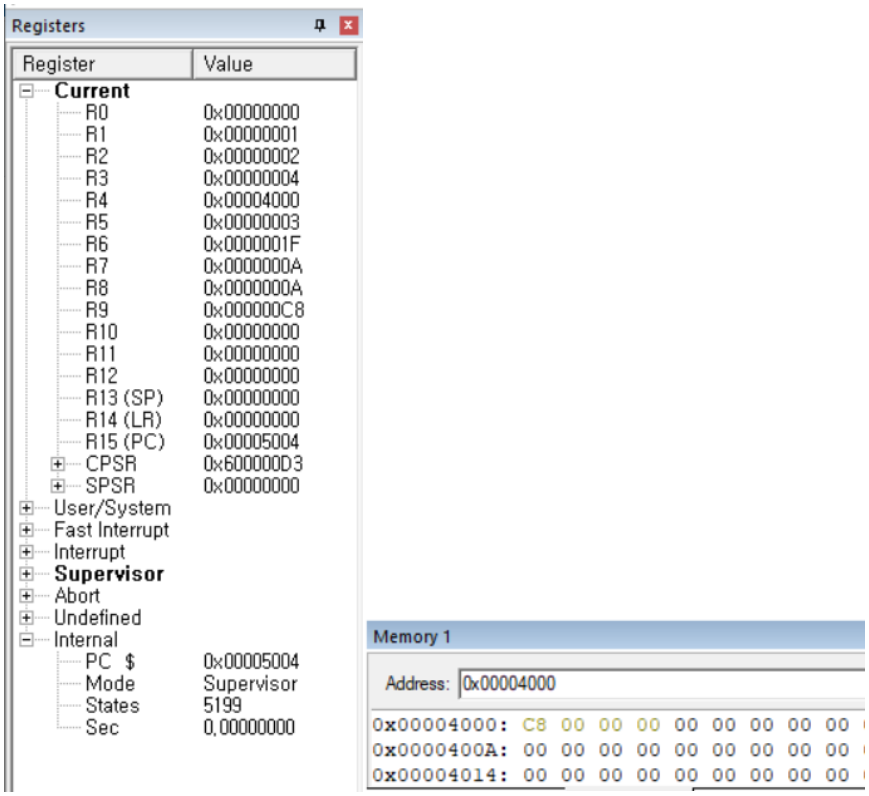
10,9,8,7,6,5,4,3,2,1 이던 배열이 1,2,3,4,5,6,7,8,9,A로 저장됨을 볼 수 있다.

C. Problem3

i. Problem3_1

```
Build Output
linking...
Program Size: Code=68 RO-data=0 RW-data=0 ZI-data=0
".\Objects\Problem3_1.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:01
```

코드 사이즈는 68이다.

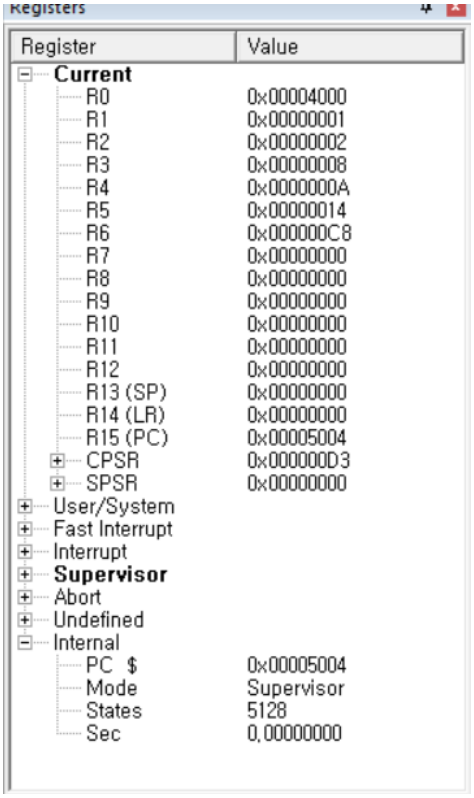


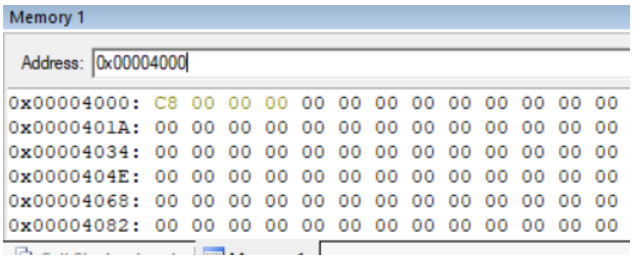
정상적으로 C8 즉 십진수로 200이 저장됨을 확인할 수 있다.

ii. Problem3_2

```
Build Output
linking...
Program Size: Code=36 RO-data=0 RW-data=0 ZI-data=0
".\Objects\Problem3_2.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:00
```

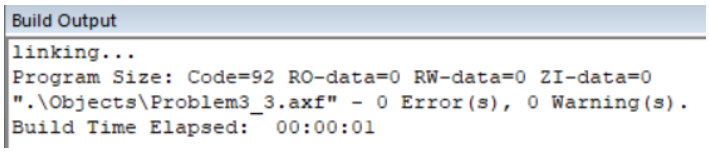
코드 사이즈는 36이다.



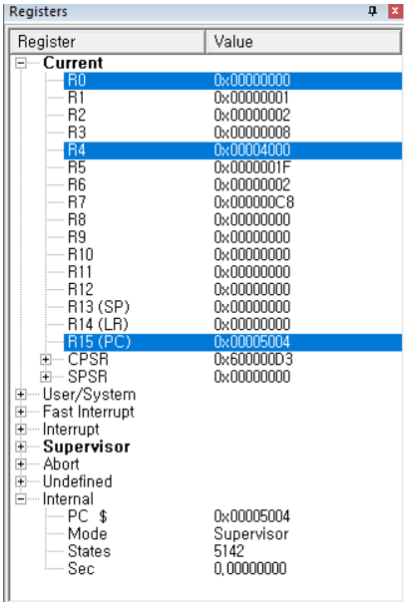


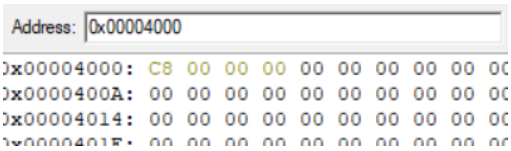
정상적으로 C8 즉 십진수로 200이 저장됨을 확인할 수 있다.

iii. Problem3_3



코드 사이즈는 92이다.





정상적으로 C8 즉 십진수로 200이 저장됨을 확인할 수 있다.

4. Consideration

이번 과제를 통해 Shift 연산인 LSL 어셈블리 명령어를 통해 왼쪽으로 자릿수를 옮기면서 2배씩 커질 수 있음을 확인하고 DCB, DCD 명령어를 통해 배열과 문자열을 표현하고 C 언어의 goto 명령어처럼 label, Branch 명령어를 사용하여 반복문 그리고 함수의 기능을 구현하였다.

또한 3번 문제에서 3가지 경우에 맞춰 각각에 맞게 코딩을 진행하였고 각 방법과 성능을 비교하였다. 코드 사이즈도 짧고 연산도 적은 3-2가 가장 빠르며 Unrolling을 통하여 레지스터 사용량은 많지만 Loop문이 2번만 사용하여 그만큼 연산횟수가 줄어 2번째로 빠르다. 마지막으로 3-1은 10번의 반복으로 가장 많은 용량과 시간이 들 것으로 예상한다.

마지막으로 Branch와 conditional execution의 차이점과 성능의 차이를 비교하면 Branch는 특정 label의 구간으로 넘어가는 연산을 진행하고 conditional execution은 특정 플래그에 따라 Conditional field를 설정하여 그에 따른 연산을 진행한다. 이를 보면 어떤 연산을 하기 위해 label을 이동하는 Branch 보다는 Flags를 이용하여 conditional execution을 사용하여 바로 값을 비교하는 것이 더 빠르고 좋을 것이라고 생각한다.

5. Reference

1) 이형근 교수님/ 어셈블리프로그램설계및실습 / 광운대학교 컴퓨터정보공학부/2022