Report – HW 2

1. Problem & Purpose

i. Purpose

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

ii. Problem

Problem 1.

- Strcmp함수 작성하기
- 만약 두 문자열이 같은 경우에는 0x0A를, 그렇지 않을 경우에는 0x0B를 4000번지에저장

Problem 2.

- 배열을 정렬하여 4000번지부터 저장하시오
- Array[10]= {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}
  - 정렬 알고리즘을 사용하지 않아도 되며, 올바르게 정렬된 값(1, 2, 3, 4, 5, 6, 7, 8, 9 ,10)이 저장되기만 하면 됨

Problem 3.

- 11+13+15+…27+29를 계산하여 결과값을 4000번지에 저장
  - Loop을 이용한 방법
    - 11을 MOV 연산으로 넣는 것이 아니라 shift 연산을 이용해야 함
      - 숫자는 #1만 사용
  - 일반화된 식인 n(n+10)을 이용한 방법
    - 10을 위해 Shift 연산을 이용해야 함!
  - Unrolling을 이용한 방법

■ 위의 방법으로 각각 구현해 보고 각 방법의 성능을 비교해 보자

Branch와 conditional execution의 차이점과 성능 차이

2. Used Instruction

MOV, MOVNE, MOVEQ // END // CMP // LDR // STR // LSL // B, BEQ // DCD // ADD

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

   A. MOVEQ Rd, operand 2 : CPSR에 있는 Z flag 가 1일 때 실행한다. Comparison 명령어가 비교한 결과가 0일 때 실행한다.

   B. MOVNE RD, oprand 2 : CPSR에 있는 Z flag가 0일 때 실행한다.

ii. END : Assembly code가 끝났음을 의미하는 Instruction

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

iv. LDR Rd, addressing : addressing에 있는 값을 Rd에 저장한다.

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

vi. LSL #number : Number 만큼 왼쪽으로 bit stream을 shift

vii. B label : 작성한 label의 첫 번째 instruction으로 이동

   A. BEQ label : CPSR에 있는 Z flag가 1일 때 작성한 label의 첫 번째 instruction으로 이동한다.

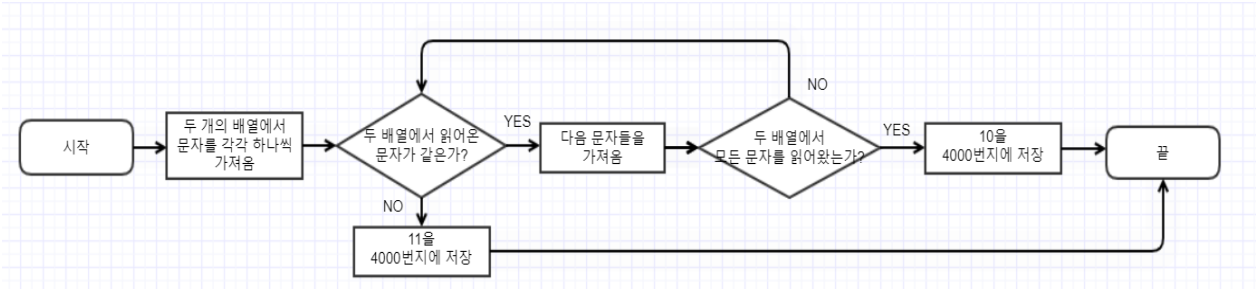
viii. {label} DCD exer{,expr} or {label} & expr{,expr} : 4 byte 단위로 메모리를 할당 및 해당 값으로 초기화{

ix. ADD Rd, Rn, N : Rd에 Rn+N 값을 저장한다

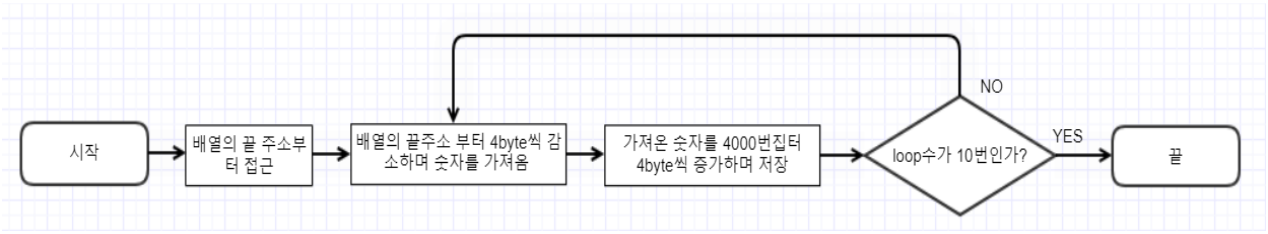
3. Design(Flow chart)

i. 설계한 내용의 Flow chart

   A. Problem 1

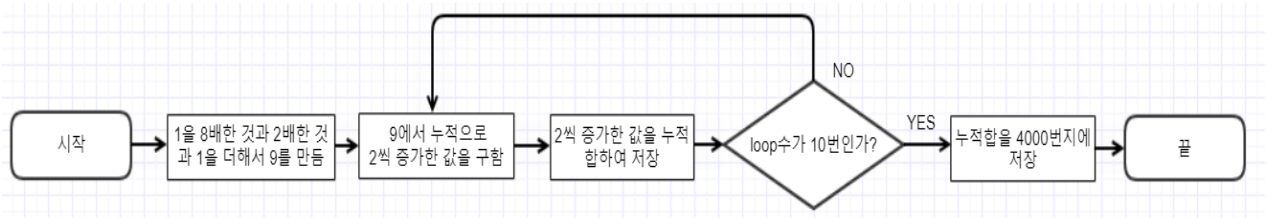


B. Problem 2

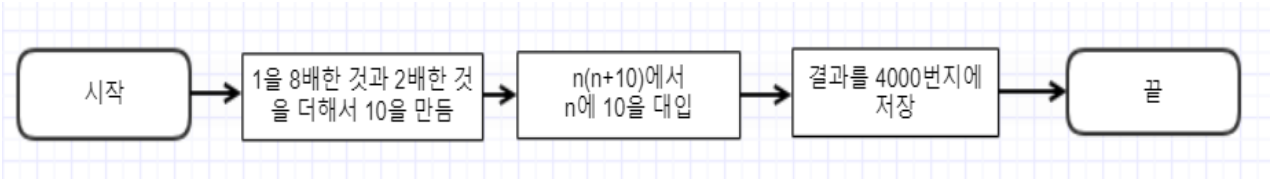


C. Problem 3

- Loop



- 일반식



- unrolling

1을 비트연산을 통해 11, 19, 21...29 값으로 만들고 전부 다 더한다.

4. Conclusion

[Branch와 conditional execution의 차이점과 성능 차이]

교안에 있는 Exercise로 Branch와 conditional execution을 비교해보았다.

Branch 방법 - code size : 120 states : 153

Conditional execution 방법 - code size 116, states : 160

code size는 당연히 branch가 더 길 줄은 알았는데 생각보다 차이가 많이 나지 않았다. States도 conditional execution이 더 작을 줄 알았는데 branch가 더 작았다. 생각을 잘못했나 보다. 막연하게 무조건 conditional execution방법이 훨씬 더 성능이 좋을 줄 알았는데 아니었다. 적절하게 두 가지 방법 중 성능이 더 좋을 만한 것을 골라 사용해야할 것 같다. Branch를 사용했을 때는 코드가 더욱 보기 쉬웠다. 반면 conditional execution을 사용할 때는 코드가 나열될 뿐이어서 주석을 다는 절차가 꼭 필요하다. Branch보다는 conditional execution이 좀 더 직관적이어서 머릿속에 있는 알고리즘을 구현하기 쉬웠다. Branch는 아무래도 conditional execution보다는 좀 더 고차원적인 생각이 필요했다. 코드의 재사용성을 생각했을 때는 Branch 기법이 훨씬 효율적이다. 코드의 재사용 여부를 하려면 프로그램의 전체적인 그림을 보아야한다. 그렇기 때문에 짧은 시간안에 괜찮은 Branch를 만들기는 쉽지 않아 보였다. 아직은 간단한 코드만 짜서 conditional execution이 편해 사용빈도가 높았지만, 조금 더 복잡한 코드를 짠다면 branch도 많이 사용될 것으로 보인다.

Problem 1번을 풀면서 문자열의 끝에 도달하면 어떻게 해결하지? 를 많이 고민했다. 그러던 중 C언어에서 배열이나 문자열에 끝난다는 표시로 맨 뒤에 0을 넣는다는 게 생각이 났다. 그래서 이 아이디어를 가지고 strcmp를 구현했다. 뜻깊은 시간이었다.

Problem 3번을 loop, 일반식, unrolling 방식으로 해결해보았다.

Loop로 짰을 땐, code size가 52였고, register를 5개를 썼다. Loop를 쓰면 아무래도 반복횟수에 대한 변수도 중요하기 때문에 register를 많이 활용해야 했다.

일반식을 이용했을 땐 code size가 32였다. 가장 짧았고 가장 구현하기 쉬웠다. 수학적인 부분은 수식으로 해결하는 게 최선의 방법인걸 느꼈다.

unrolling기법을 이용했을 땐 code size 104였다. 가장 길었다. 그리고 진짜 너무 쉬웠지만 그만큼 코드의 길이가 길었다. 그렇지만 융통성이 없는 코드라서 이 방법을 많이 쓸 것 같지는 않다.

3가지 방법으로 같은 문제를 접근하니 각자의 장단점을 알 수 있어서 좋았다.

5. Reference

DCD 참조 : http://heart4u.co.kr/tblog/328

https://k1rha.tistory.com/entry/ARMAssambly-%EA%B3%B5%EB%B6%80-%EC%8B%9C%EC%9E%91

LSL 참조 :https://controlbit.tistory.com/3

Unrolling 참조 :https://hoororyn.tistory.com/10