Assignment 4

1. Problem & Purpose

i. strcpy함수를 구현한다.

disassembly 화면을 캡쳐하여 pusedo instruction이 어떻게 변경되고 변경된 instruction이 어떤 원리로 프로그램에서 동작하는지에 대한 설명을 첨부한다.

2. Used Instruction

I. 6: LDR // LDRB // STRB // CMP // BNE // MOV // END

i. LDR Rd, operand1 : operand1의 메모리 위치의 값을 word 크기만큼 Rd에 불러온다.

ii. LDRB Rd, [operand1], #num : operand1의 위치에서 데이터를 가져와 Rd에 저장한다. 이때 operand1의 주소는 num만큼 더하여 저장한다.

iii. STR Rd, [R0, offset] : R0으로부터 offset만큼 이동한 위치에 R0의 값을 word 크기만큼 저장한다.

iv. STRB Rd, [operand1], #num : operand1의 위치에 Rd의 데이터를 저장한다. 이때 operand1의 주소는 num만큼 더하여 저장한다.

v. MOV Rd operand1 : operand1에 있는 값을 Rd에 저장한다.

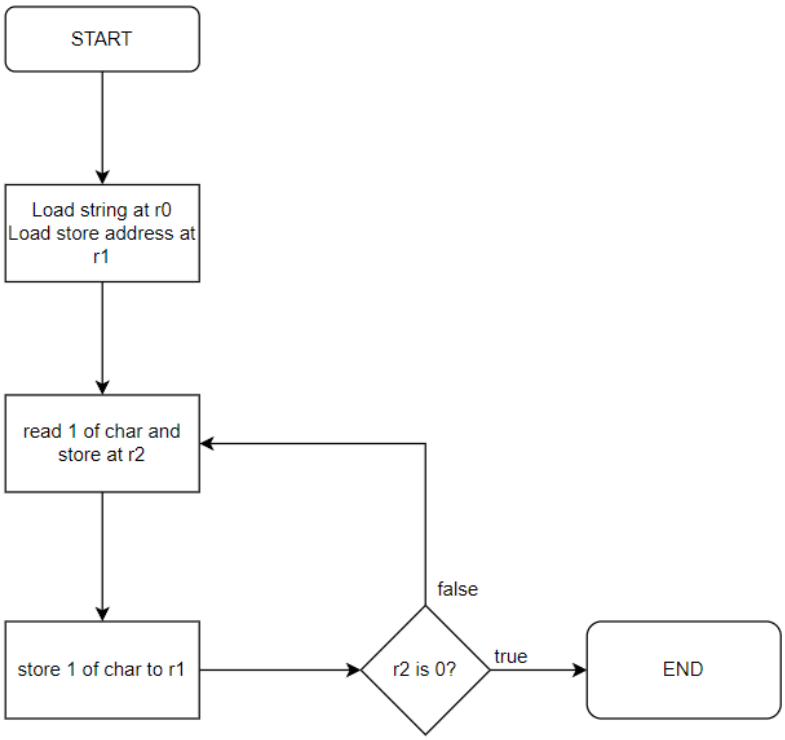
vi. CMP Rd, operand1 : Rd – operand1을 한 state를 cpsr에 업데이트한다.

vii. BNE operand : CMP로 비교하였을 때 NE일 경우 operand의 위치로 pc를 이동하여 작업을 수행한다.

viii. END : Assembly code가 끝났음을 의미하는 Instruction

3. Design(Flow chart)

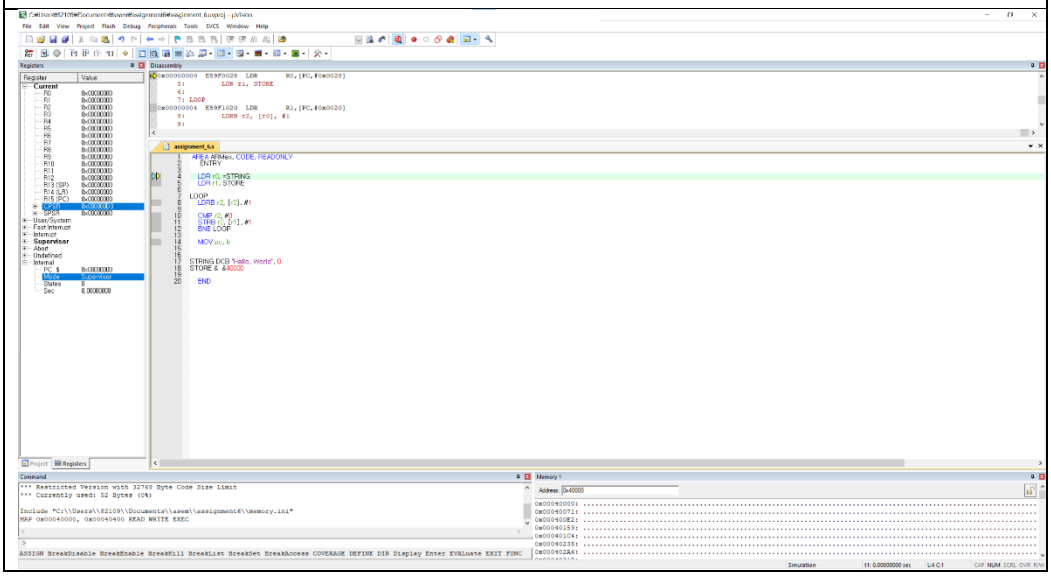
i. 6 flow chart

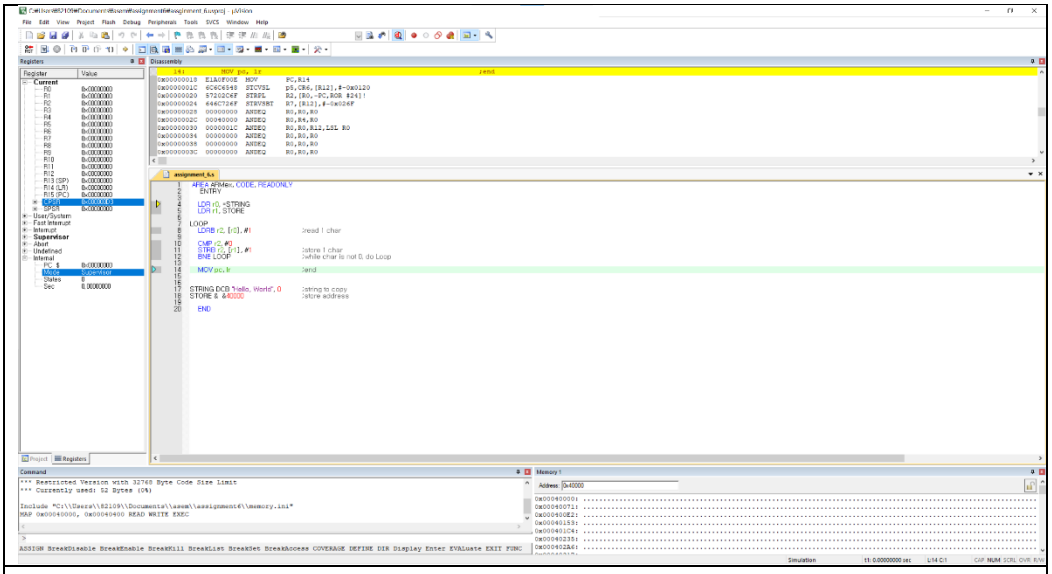


4. Conclusion

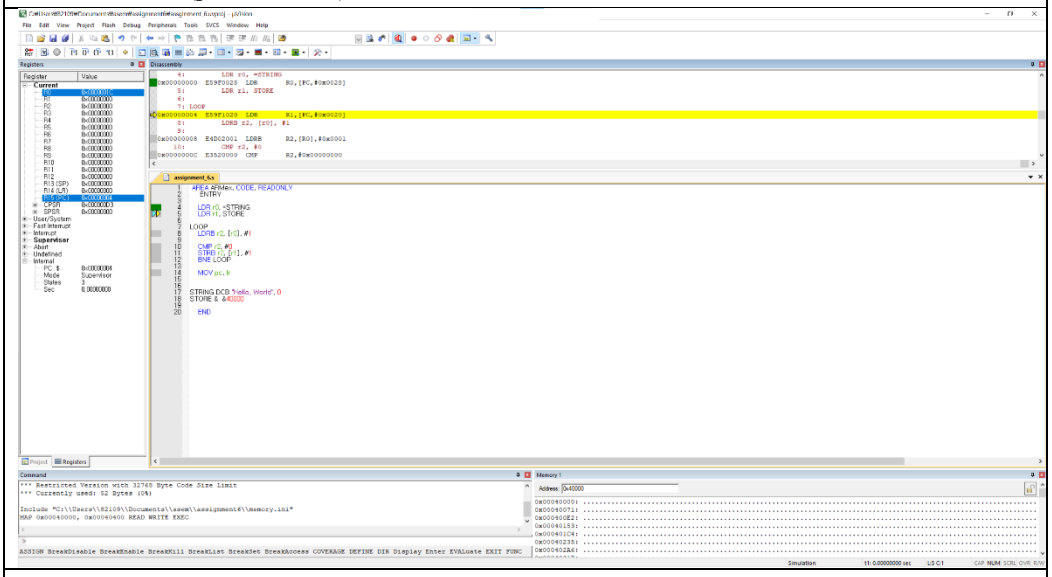
i. 6 result

LDR을 할 때 어셈블리는 PC위치로부터 0x28 위치로 이동하여 해당 위치에 있는 데이터를 읽어오고 돌아온다.

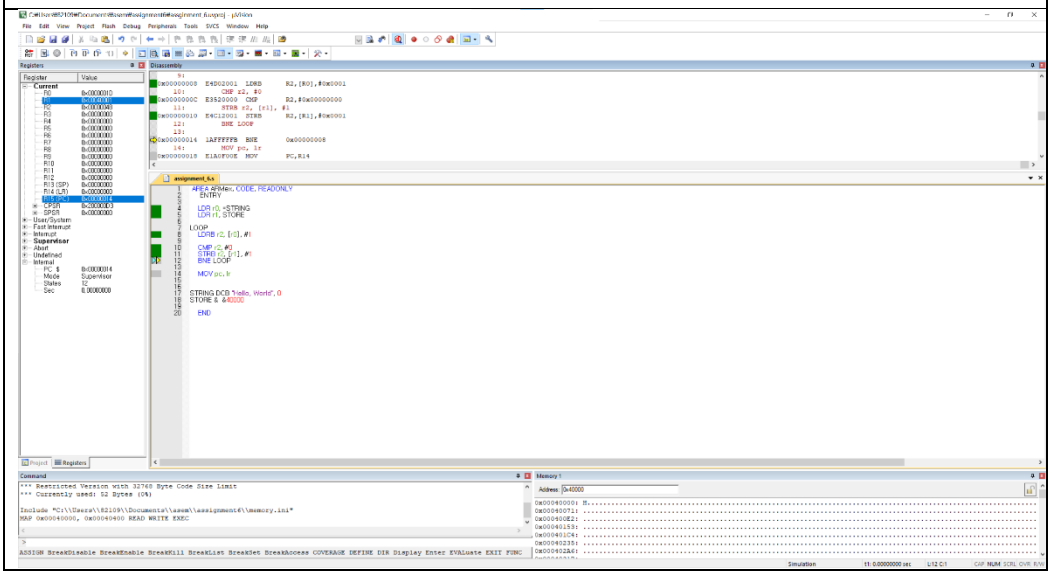




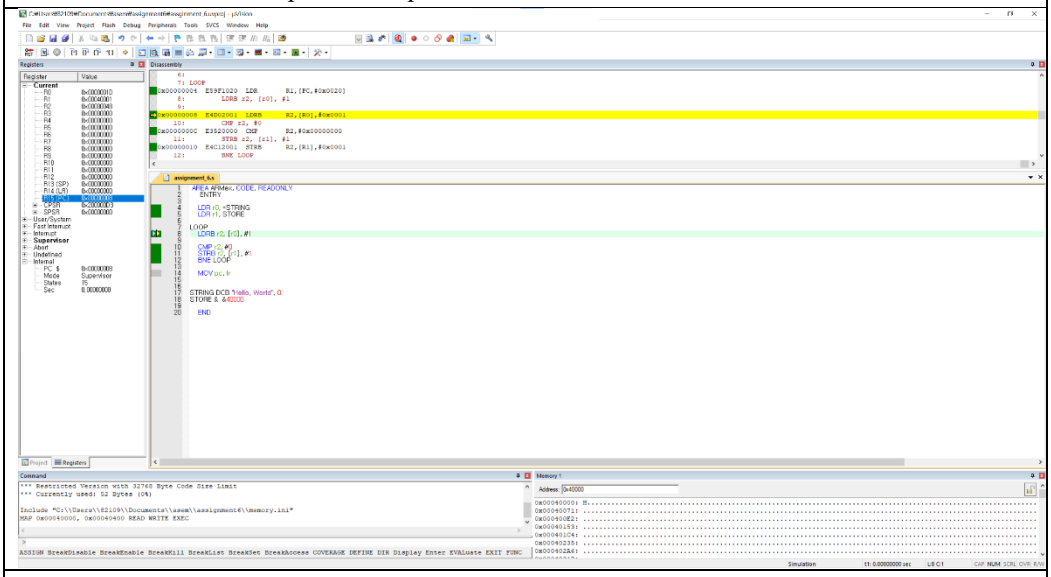
r1에 0x20번지(pc + 0x20 - 4)에 저장된 값인 주소를 읽어온다.



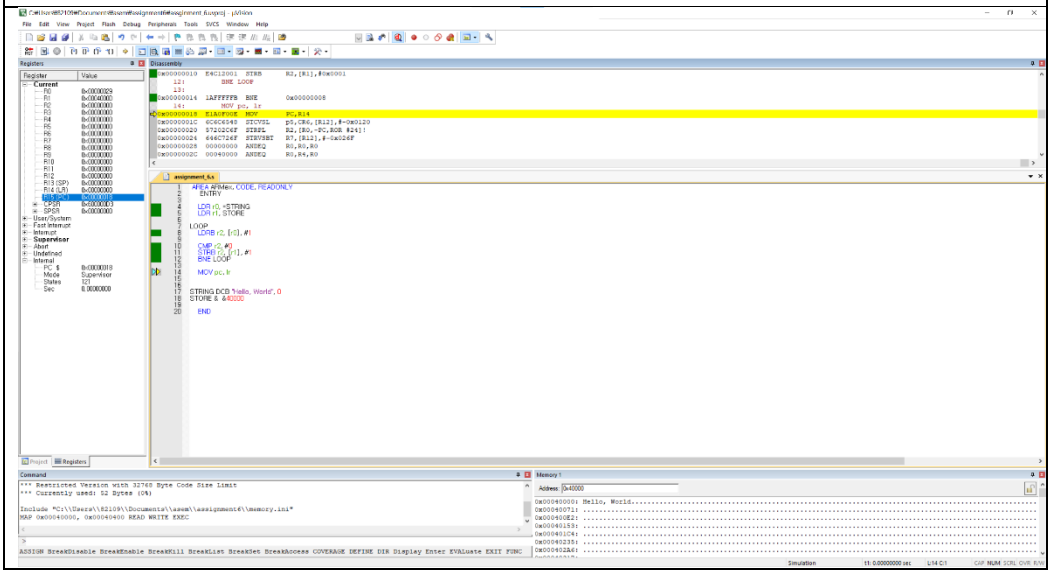
BNE의 명령의 경우에도 기존과 같이 동작하는 모습을 보인다.



BNE를 통과한 후 Loop 위치로 pc값이 옮겨진 모습이다.



모든 동작을 마치고 종료 시에 MOV pc lr도 마찬가지로 기존과 동일하게 동작한다. 데이터는 정상적으로 복사된 모습을 볼 수 있다.



5. Consideration

- 위의 해당 과제를 수행하면서 디버그 환경에 disassembly 화면이 존재한다는 것을 처음 알게 되었다. 해당 부분을 보면서 디버그를 확인할 경우 더욱 자세한 동작을 이해할 수 있다는 것 또한 알게 되었다. 마지막으로 우리가 기존에 아무렇지 않게 사용하던 LDR 용어가 사실은 기존과 다르게 동작한다는 사실을 알게 해준 과제였다.

6. Reference

i. 이준환 교수님/어셈블리프로그램설계및실습/광운대학교(컴퓨터정보공학부)/2021