시스템소프트웨어와 실습

LAB 2.

제출일	2023년 9월 20일	학과	컴퓨터공학과
과목	시소프 1분반	학번	2022111967
담당교수	정준호 교수님	이름	조현우

1. 주어진 코드 flag 값 바꿔 실행해보기.

(코드 분석하기)

```
test1    start   1000
first    lda     flag
         comp    two
         jeq     add2
         lda     alpha
         add     one
         sta     beta
         j       next
add2     lda     alpha
         add     two
         sta     beta
next     lda     gamm
         sub     one
         sta     delta
one      word    1
two      word    2
flag     word    1
alpha    word    5
gamm     word    10
incr     word    3
beta     resw    1
delta    resw    1
         end     first
```

(first)

Flag 값을 a 레지스터에 load한 후 two(2)와 비교 후 같다면 add2 프로세스로 점프, 다르다면 alpha를 load 하고 one을 add 한다. 그 값을 beta에 저장한 후 next로 점프한다.

(add2)

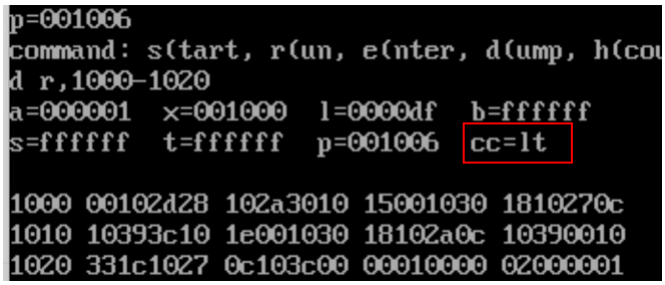
alpha를 a 레지스터에 load하고 two를 더한다, 그 값을 beta에 저장한다

(next)

gamm를 load하고 one를 뺀 후, 그 값을 delta에 저장한다.

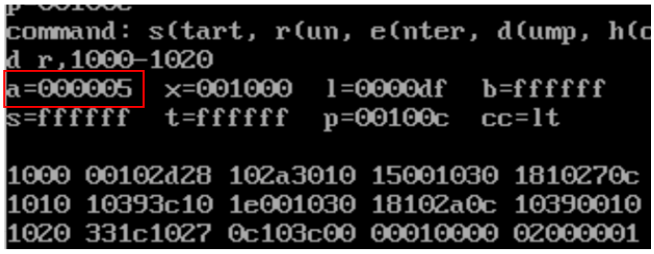
결론: flag값이 2일 때와 2가 아닐 때로 나뉘게 되며, 2일땐 beta는 7, delta는 9를 가지게 될 것이며, 2가 아닐 땐 beta는 6, delta는 9를 가지게 될 것이다.)

1-1) Flag = 1 로 설정



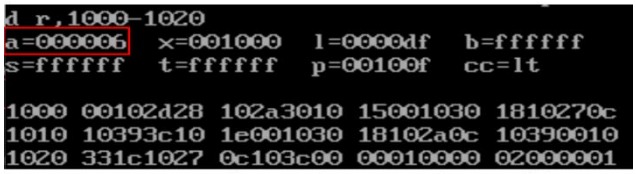
(lda flag, comp two, jeq add2)

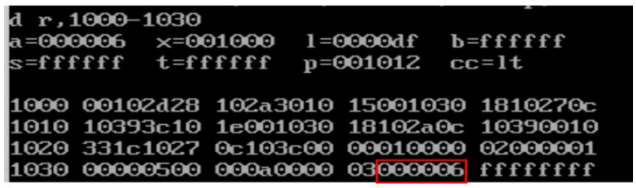
flag가 1이므로 2와 같지 않아 lt로 변한 것을 확인할 수 있다.

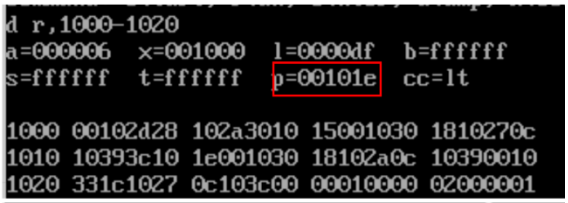


(lda alpha, add one, sta beta)

a레지스터에 5라는 alpha값을 load하고 1를 더한 후 beta에 6를 저장한 것을 확인할 수 있다.

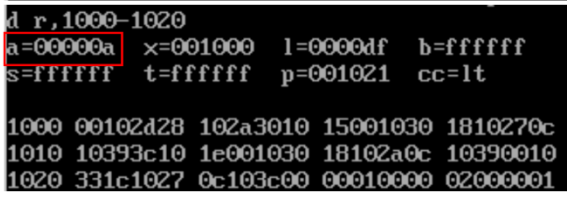






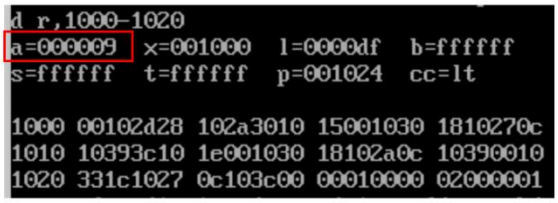
(j next)

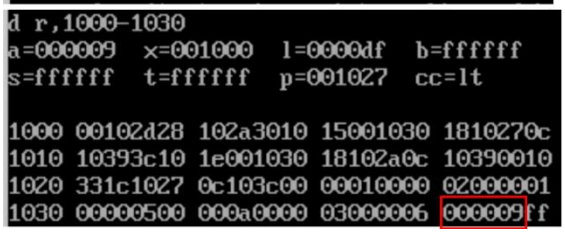
Next로 점프하여 다음 프로그램 값이 next 시작 주소로 설정되었다.



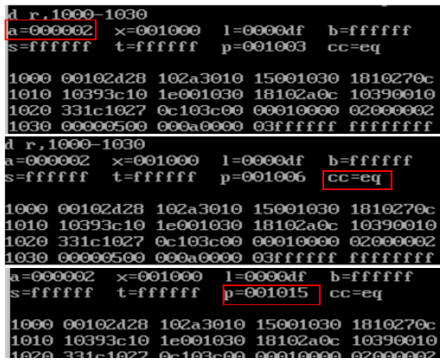
(lda gamm, sub one, sta delta)

gamm값 10(a)가 a레지스터에 load 되었으며, 1를 빼고 delta에 그 값인 9를 저장한 것을 확인할 수 있다.



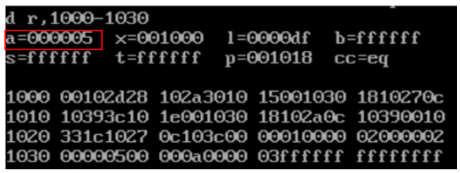


1-2) Flag = 2 로 설정



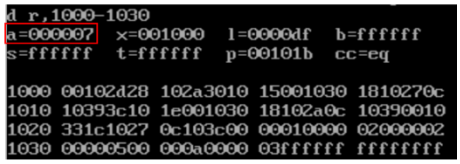
(first 실행 내용)

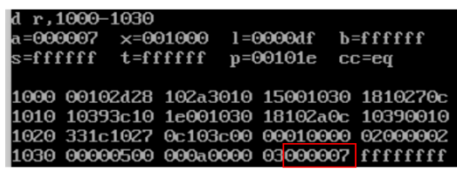
a레지스터에 flag값을 load하였으며, two와 비교하였을 때 같기 때문에, eq로 유지되는 것을 확인할 수 있다. 또한 jeq에 걸려 add2로 점프해야 하기 때문에 다음 프로그램 값 또한 0105로 수정된 것을 확인할 수 있다.

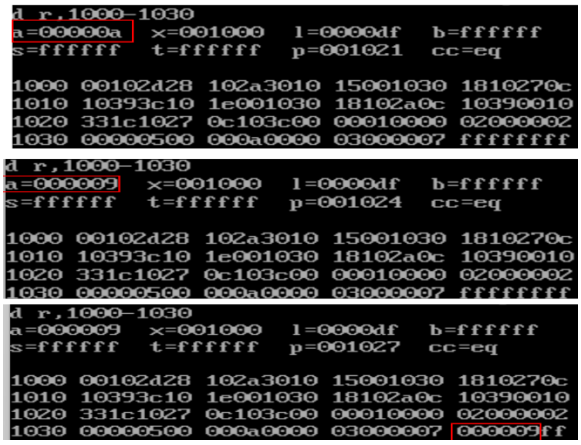


(add2 실행 내용)

Add2로 점프하여, alpha값인 5를 load하고, 2를 더한 후 beta에 7을 저장한 것을 확인할 수 있다.







(next 실행 내용)

gamm 값인 10을 load하고 1를 뺀 후 delta에 정상적으로 저장한 것을 확인할 수 있다.

2. 문자열 복사 sic프로그램 실행해보기

(코드 분석하기)

```
prog2    start   1000
sec      ldx     zero
loop     ldch    str1,x
         stch    str2,x
         tix     count
         jlt     loop
str1     byte    c'sting data!'
str2     resb    11
zero     word    0
count    word    11
         end     sec
```

X 레지스터에 zero (0)을 로드하고 str1의 x번째 글자를 a레지스터에 load한다. a레지스터값(str1의 x번째 글자)을 str2의 x번째로 저장한다.

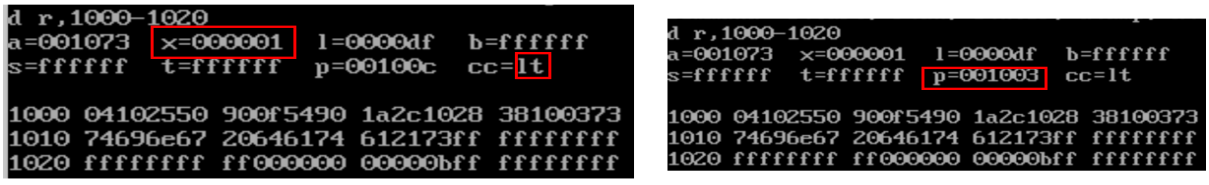
X를 1올린후 count와 비교한 후 작으면 loop로 다시 점프한다.

- **실행 결과**

(첫번째 loop 결과)

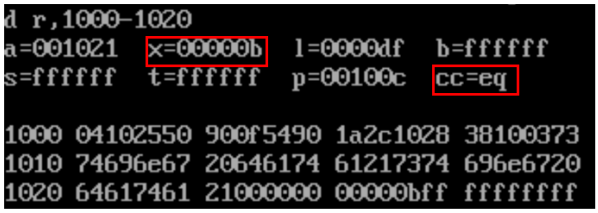
X = 0를 load 하였으며, str1의 첫번째 글자 s의 아스키 코드 73을 무사히 load하였으며, str2의 첫 글자에 저장한 것을 확인할 수 있다.

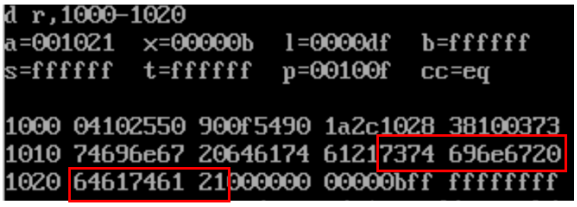
(tix count, jlt loop)



1로 count보다 작아 lt가 되었으며, loop로 jump해야하기 때문에 1003의 주소로 변경되었다.

(위 과정을 11번 반복 한 후 x = 11(b)되었을 때의 실행 모습)





x = 11(b)가 되었을 때 eq로 바뀐 것을 확인할 수 있으며, str2에 str1의 문자열이 모두 복사된 것을 확인할 수 있다.

3. 2번 실습문제에서 문자열에 ':' 추가하기

```
prog3    start   1000
sec      ldx     zero
loop     stx     temp
         lda     temp
         comp    five
         jeq     101e
         ldch    str1,x
         stch    str2,x
         tix     count
         jlt     loop
         j       102a
         ldch    colon
         stch    str2,x
         tix     count
         jlt     loop
str1     byte    c'sting data!'
colon    byte    c':'
str2     resb    11
temp     resw    0
zero     word    0
five     word    5
count    word    11
         end     sec
```

(코드 설명)

이 문제는 문제2번에서 str1의 " " 공백을 ":" 콜론으로 바꾸는 문제이다. 즉 str1의 index 5번의 공백을 콜론으로 바꾸면 되는 문제이기에, ':' 값을 가질 colon, x 레지스터 값을 저장할 temp를 선언한 후, loop문을 수정하면 된다.

또한 temp는 값을 계속 바꿔야하기 때문에 resw로 선언하고, colon은 byte로 선언한다.

(stx temp, lda temp, comp five, jeq 101e)

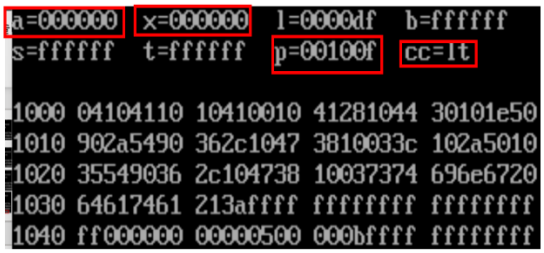
4개의 코드를 통해 x레지스터값이 5일 때 colon을 저장하기 시작하는 주소인 101e 주소로 점프하게 하였다.

(ldch colon, stch str2,x, tix count, jlt loop)

4개의 코드를 사용해 str2의 x번째 인덱스에 colon 값인 ":"을 저장하며, x가 count보다 작으면 loop문으로 점프한다.

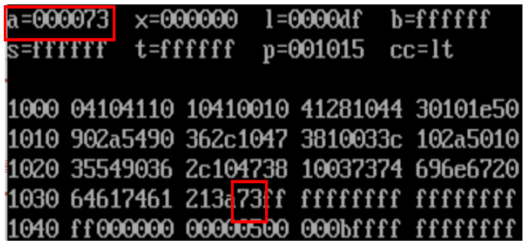
즉, x가 5가 아니면 str1을 복사하는 코드를 실행하며, x가 5가 되면 colon을 복사한다. 또한 x값이 count보다 높아지게 되면 102a 주소 즉 끝나는 주소로 점프하여 프로그램이 종료되게 하였다.

(다음은 x = 0 일 때, 즉 str1의 문자열을 복사할 때의 모습이다.)



(stx temp, lda temp, comp five, jeq 101e)

위 코드의 실행된 결과이며, five와 달라 lt가 나왔고 같지 않아 101e로 점프하지 않았다.

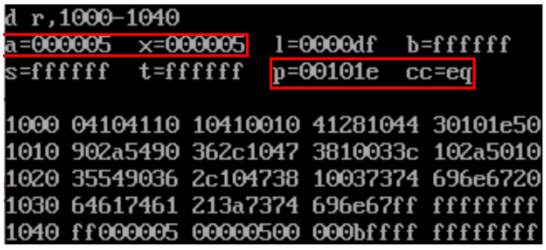


(ldch str1,x , stch str2,x)

정상적으로 실행되며 첫 글자인 's'의 아스키 코드 73이 load되고 store 되었다.

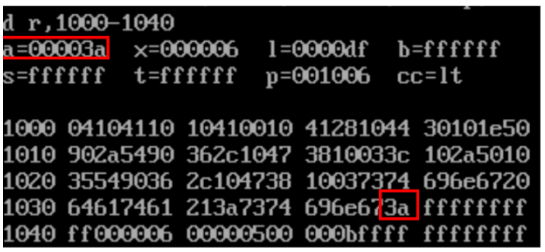
또한 다시 loop로 점프한다.

(5번을 반복하여 x가 5가 되었을 때의 코드 실행 모습, 즉 colon을 복사할 때의 모습)



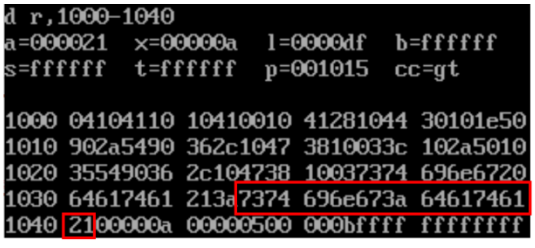
(stx temp, lda temp, comp five, jeq 101e)

다음 코드가 실행된 결과이며, x = 5인 것을 확인하여 eq로 반환된 것을 확인할 수 있고, eq가 나와 다음 프로그램의 시작 주소를 101e의 주소로 설정된 것을 확인할 수 있다.



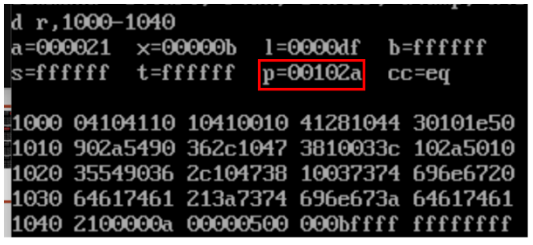
또한 ':"의 아스키 코드인 3a가 정상적으로 load, store된 것을 확인할 수 있다.

(모든 문자열을 복사한 후 정상적으로 빠져나오는 모습)



"sting:data!" 를 모두 아스키 코드로 바꾼

73 74 69 6e 67 3a 64 61 74 61 21 이 정상적으로 str2에 저장된 것을 확인할 수 있으며,



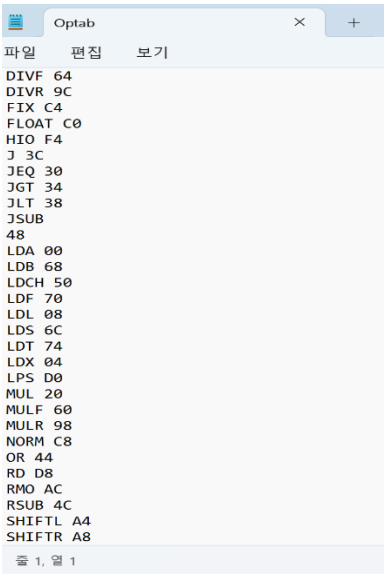
x레지스터 값이 count와 같아져 jlt 조건에 만족하지 않기 때문에 프로그램이 종료되는 것을 확인 할 수 있었다.

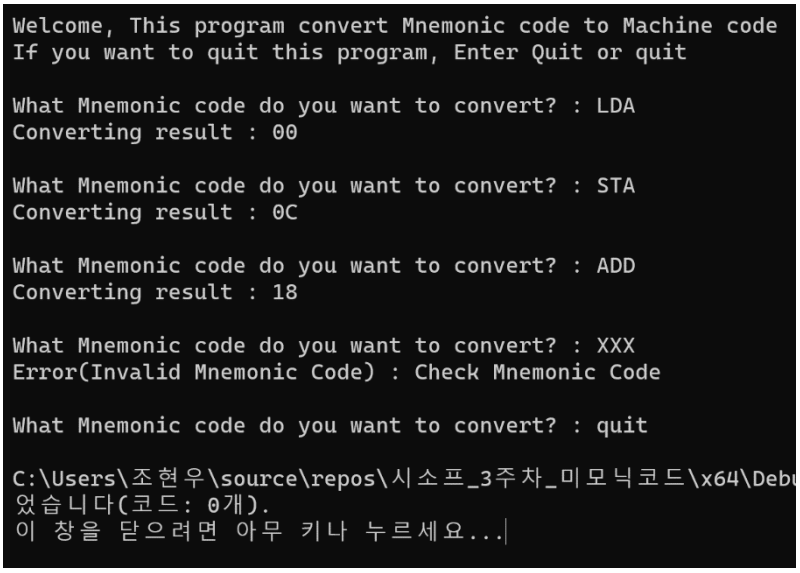
4. Mnemonic code가 Machine code로 반환되는 프로그램 작성하기

```cpp
//2022111967 조현우
#include<unordered_map> // 해쉬테이블
#include<fstream>       // 파일 입출력
#include<iostream>
#include<string>
using namespace std;

unordered_map<string, string> Optable;  // 미모닉코드와 머신코드를 묶어줄 자료구조 사용

// 파일내용으로 자료구조 구축하는 함수
void Make_Optable()
{
    ifstream file("Optab.txt");   // 파일 입출력

    // 예외 처리
    if (!file)
    {
        cout << "Error : Not Found File" << "₩n";
        exit(1);
    }

    string MnemonicCode = "";
    string MachineCode = "";
    while (file)   // 파일 모든 데이터 확인
    {
        file >> MnemonicCode;  // 미모닉 코드 첫번째 받고
        if (!file)  // 미모닉코드만 받았는데 파일 데이터가 끝나면 머신코드가 없어 저장 X
            break;
        file >> MachineCode;   // 머신 코드를 받음
        Optable[MnemonicCode] = MachineCode;  // 해쉬테이블 저장
    }
    file.close();
}
```

```cpp
// 사용자에게 입력을 받아 출력하는 함수
void Convert()
{
    string cmd = "";
    cout << "Welcome, This program convert Mnemonic code to Machine code " << "₩n";
    cout << "If you want to quit this program, Enter Quit or quit" << "₩n₩n";
    while (true)
    {
        cout << "What Mnemonic code do you want to convert? : ";
        cin >> cmd;
        if (cmd == "Quit" || cmd == "quit")  // quit 입력하면 프로그램 종료
            break;

        if (Optable[cmd] == "")  // 비어있으면 오류반환
            cout << "Error(Invalid Mnemonic Code) : Check Mnemonic Code " << "₩n₩n";
        else
            cout << "Converting result : " << Optable[cmd] << "₩n₩n";
    }
}

int main()
{
    // 빠른 입출력
    cin.tie(NULL);
    cout.tie(NULL);
    ios::sync_with_stdio(false);

    // 실행
    Make_Optable();
    Convert();

    return 0;
}
```

해당 Mnemonic code에 해당하는 machine code를 반환하면 되기 때문에 해쉬테이블을 사용하였으며, txt파일에서 데이터를 가져오기 위해 fstream 헤더를 사용하여 구현하였다.





(Optab.txt의 사진) (해당 코드를 실행했을 때의 결과)

Optab은 학습자료실의 SIC/XE 명령어를 저장하였습니다.

- **코드 설명**

위 코드는 2개의 함수로 이루어져 있다.

1. **Make_Optable 함수**

파일을 찾을 수 없을 때, 프로그램을 종료한다.

.txt에서 string을 2개씩 읽으며, 해당 string을 해쉬테이블에 미모닉코드와 머신코드로 저장한다.

모두 읽어내면 파일을 닫는다.

2. **Convert 함수**

사용자에게 메뉴를 출력하며 사용자가 변환하고 싶은 미모닉코드를 입력받는다.

While 문을 사용해 사용자가 "quit"을 입력하면 프로그램을 종료하며 아니면 해쉬테이블에서 머신코드를 찾는다. 해당 미모닉코드가 저장되어 있다면 머신코드가 출력되며, 없다면 오류가 출력된다.

5. 느낀 점

첫번째 실습을 한 후 실습이어서 꽤나 익숙해져 문제없이 수월하게 진행할 수 있었다. C++ 등으로 코드를 만들 땐 메모리 주소를 직접 사용하지 않고 포인터를 사용해 그렇게 체감이 되지 않았는데 j, jeq, jlt 와 같은 점프 함수를 사용해보면서 주소에 대해 확실히 체감하게 되었다. 또한 어셈블리 언어 또한 C++과 크게 다른 점이 없는 것 같다는 느낌 또한 받게 되어 흥미로웠다.