**시스템 소프트웨어와 실습**

**실습과제 #1**

**보고서**

**이름: 정여준**

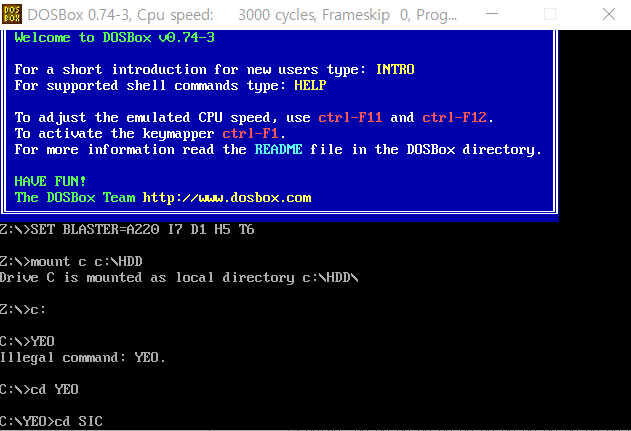
**학번: 2017112138  
학과: 컴퓨터공학과  
교수님: 문봉교 교수님**

**강의명: 시스템소프트웨어와 실습**

**실습#1**

**1번. 콘솔 기반의 simulator를 이용해서 주어진 샘플 실행해 본다.**

일단 SIC파일이 있는 C의 HDD폴더로 마운트 해주고 SIC폴더까지 cd명령어를 통해 이동한다.



srcfile

test start 1000

first lda five

sta xx

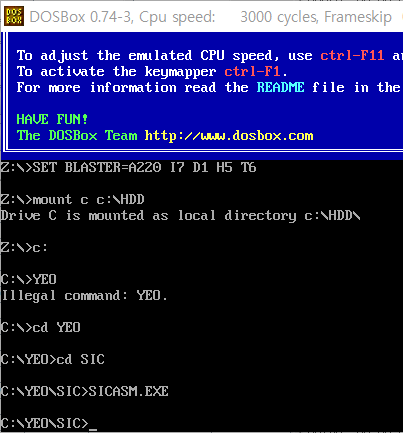
.

five word 5

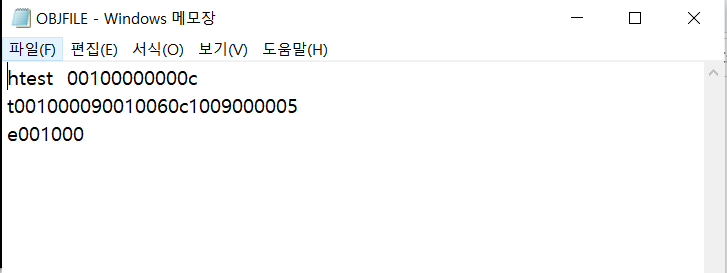
xx resw 1

end first

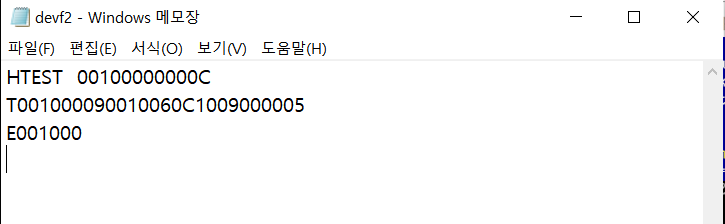
srcfile을 다음과 같이 작성하고 SICASM.EXE를 실행한다.



SICASM.EXE를 실행하면 다음과 같은 OBJFILE이 생성될 것이다.



OBJFILE 이름을 devf2로 변경 후 L2U를 실행해서 대문자로 변환해준다.

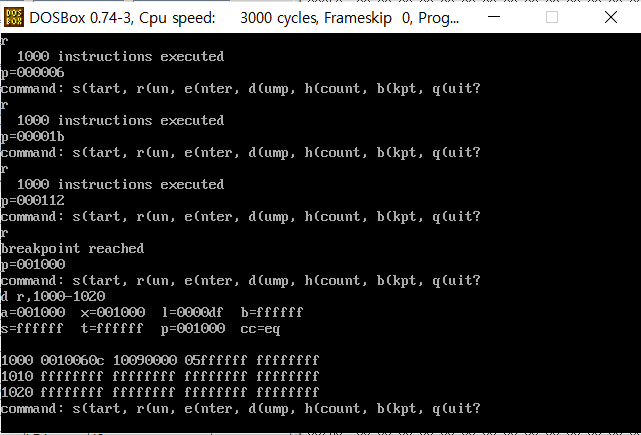
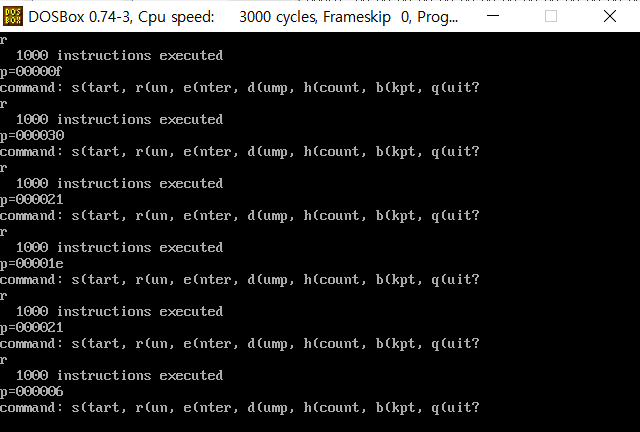


그 이후 SICSIM.EXE를 실행시킨다.

그 이후 s를 입력한다. (bootstrab file이 로드된다.) 그리고 b <1000(시작주소)>를 입력해서 break point를 프로그램 시작 주소로 한다. 그 이후 break point에 올때까지 r을 계속 입력한다.

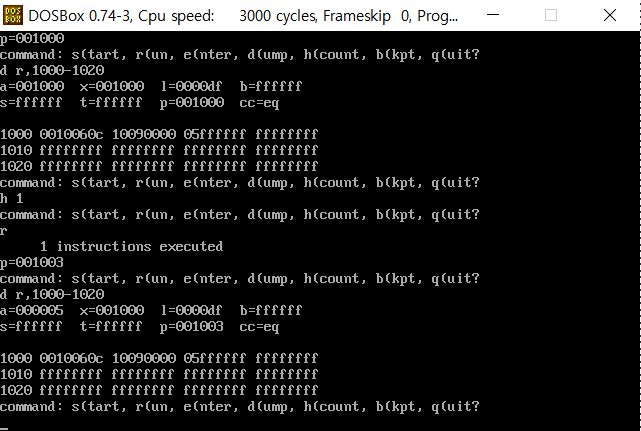
Break point에 도달하면 h 1을 입력하고 시뮬레이션을 시작한다.





여기서 break point에 도달하였다.

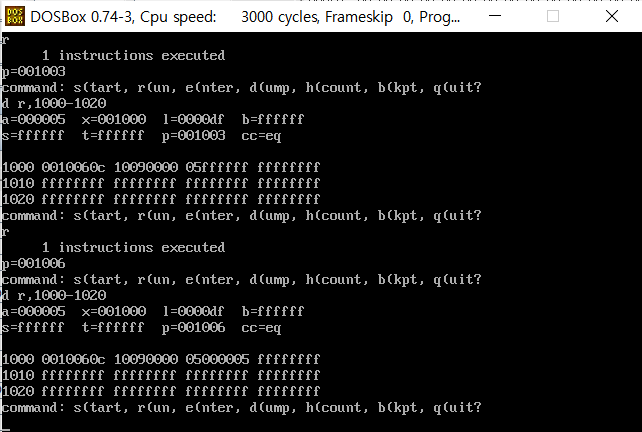
처음에 PC는 001000에 위치하게 된다.



명령어 한 개를 실행하면 PC는 1003에 위치하게 된다.

LDA의 opcode 00이고 five가 1006 주소지에 위치하고 있으므로 object code로 001006이 된다.

그리고 register a에 5가 load되므로 a=000005로 바뀌게 된다.



명령어 1개를 다시 실행하면 PC는 1006에 위치하게 된다.

STA의 opcode가 0C이고 xx는 1009 주소지에 위치하고 있으므로 object code는 0C1009이 된다.

그리고 데이터 값에 000005가 들어가게 된다.

**2번. GUI기반의 <SIC/XE 시뮬레이터>를 사용해 주어진 동일한 샘플을 실행해본다.**

주어진 Sample.asm은 다음과 같다.

COPY START 0

FIRST STL RETADR

LDB #LENGTH

BASE LENGTH

CLOOP +JSUB RDREC

LDA LENGTH

COMP #0

JEQ ENDFIL

+JSUB WRREC

J CLOOP

ENDFIL LDA EOF

STA BUFFER

LDA #3

STA LENGTH

+JSUB WRREC

J @RETADR

EOF BYTE C'EOF'

RETADR RESW 1

LENGTH RESW 1

BUFFER RESB 4096

.

. Subroutine to read record into buffer

.

RDREC CLEAR X

CLEAR A

CLEAR S

+LDT #4096

RLOOP TD INPUT

JEQ RLOOP

RD INPUT

COMPR A,S

JEQ EXIT

STCH BUFFER,X

TIXR T

JLT RLOOP

EXIT STX LENGTH

RSUB

INPUT BYTE X'F1'

.

. Subroutine to write record from buffer

.

WRREC CLEAR X

LDT LENGTH

WLOOP TD OUTPUT

JEQ WLOOP

LDCH BUFFER,X

WD OUTPUT

TIXR T

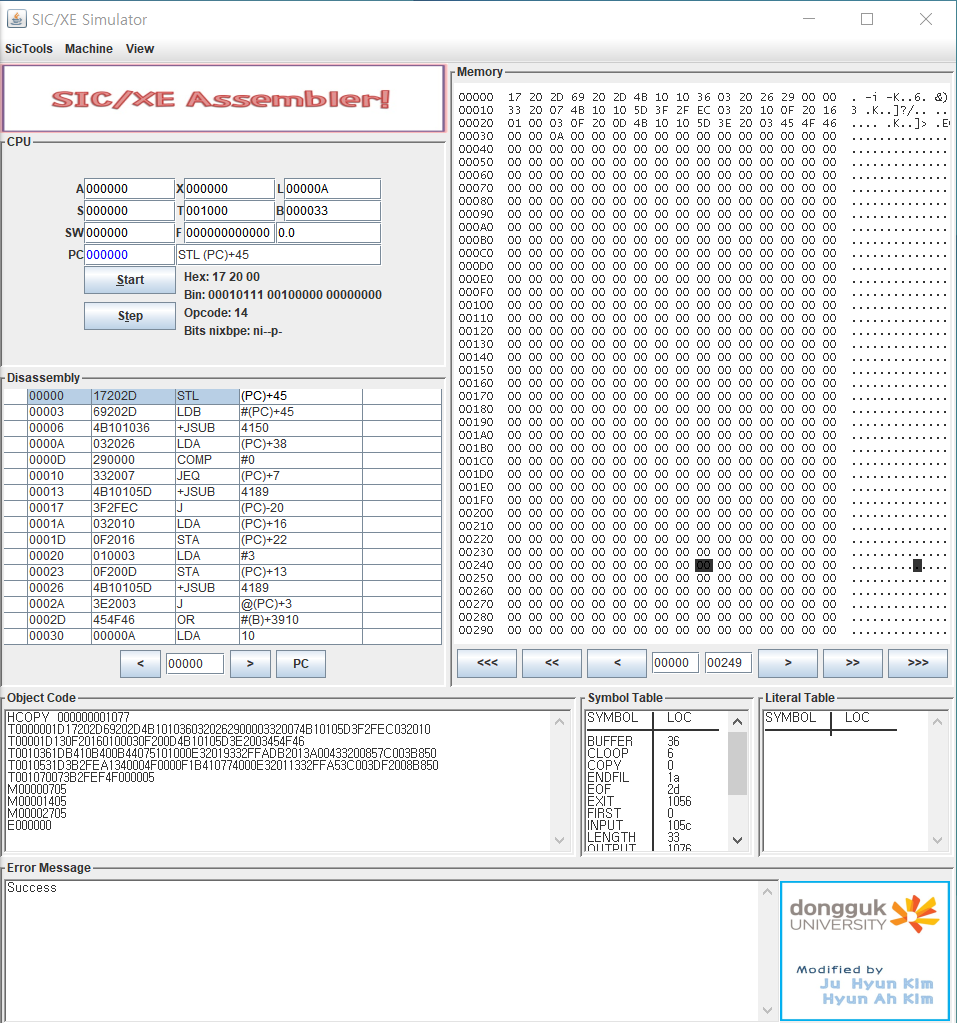
JLT WLOOP

RSUB

OUTPUT BYTE X'05'

END FIRST

이 소스코드를 SIC/XE Simulator로 Load한 결과이다.



Object code와 Memory상에 코드가 들어간다.

00000은 nixbpe=110010이고 opcode는 14이므로 STL을 (PC)부터+02D(45)를 해준 값이 된다.

00003은 nixbpe=010010이고 opcode는 68이므로 LDB를 immediate(#)으로 PC로부터+02D(45)를 해준 값이 된다.

00006은 nixbpe=110001이므로 extended하고 opcode는 48이므로 +JSUB이다. 그리고 주소는 01036이므로 십진수로 표현하면 4150이 된다.

0000A는 nixbpe=110010이므로 pc-relative이고 opcode는 00이므로 LDA이다. 주소는 PC+026(38)가 된다.

0000D는 nixbpe=01000이므로 immediate이고 opcodesms 28이므로 COMP이다. 주소는 #0이다.

00010는 nixbpe=110010이므로 pc-relative이고 opcode는 30이므로 JEQ이다. 주소는 PC+007(7)이다.

00013은 nixbpe=110001이므로 extended이고 opcode는 48이므로 +JSUB이다. 그리고 주소는 0105D이므로 십진수로 표현하면 4189가 된다.

00017는 nixbpe=110010이므로 pc-relative이고 opcode는 3C이므로 J이다. 주소는 PC+FEC(=-014=-20)이 된다. 즉 PC-20이 된다.

0001A는 nixbpe=110010이므로 pc-relative이고 opcode는 00이므로 LDA이다. 주소는 PC+010(16)이다.

0001D는 nixbpe=110010이므로 pc-relative이고 opcode는 0C이므로 STA이다. 주소는 PC+016(22)이다.

00020은 nixbpe=010000이므로 immediate하다. Opcode는 00이므로 LDA이고 주소는 #3이다.

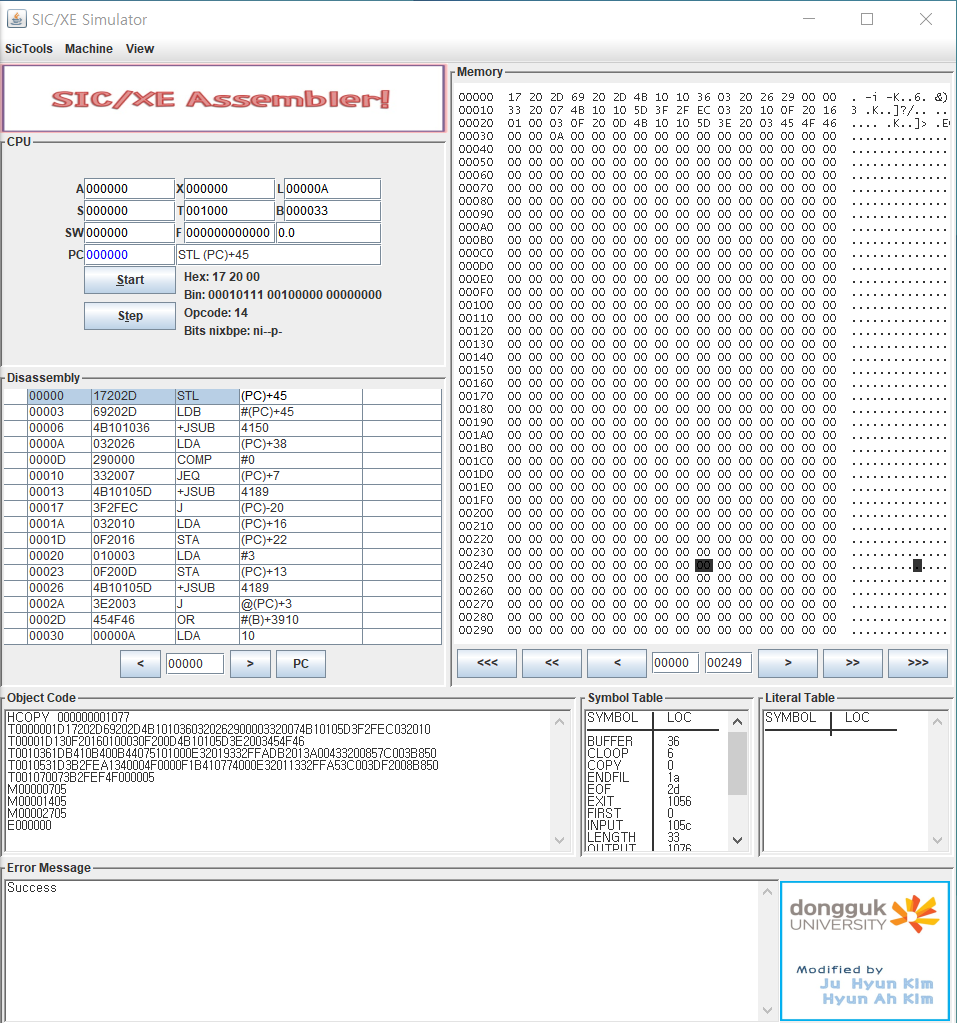
00023은 nixbpe=110010이므로 pc-relative이고 opcode 0C이므로 STA이다. 주소는 PC+00D(13)이다.

00026은 nixbpe=110001이므로 extended이고 opcode는 48이므로 +JSUB이다. 주소는 015D이므로 십진수로 표현하면 4189가 된다.

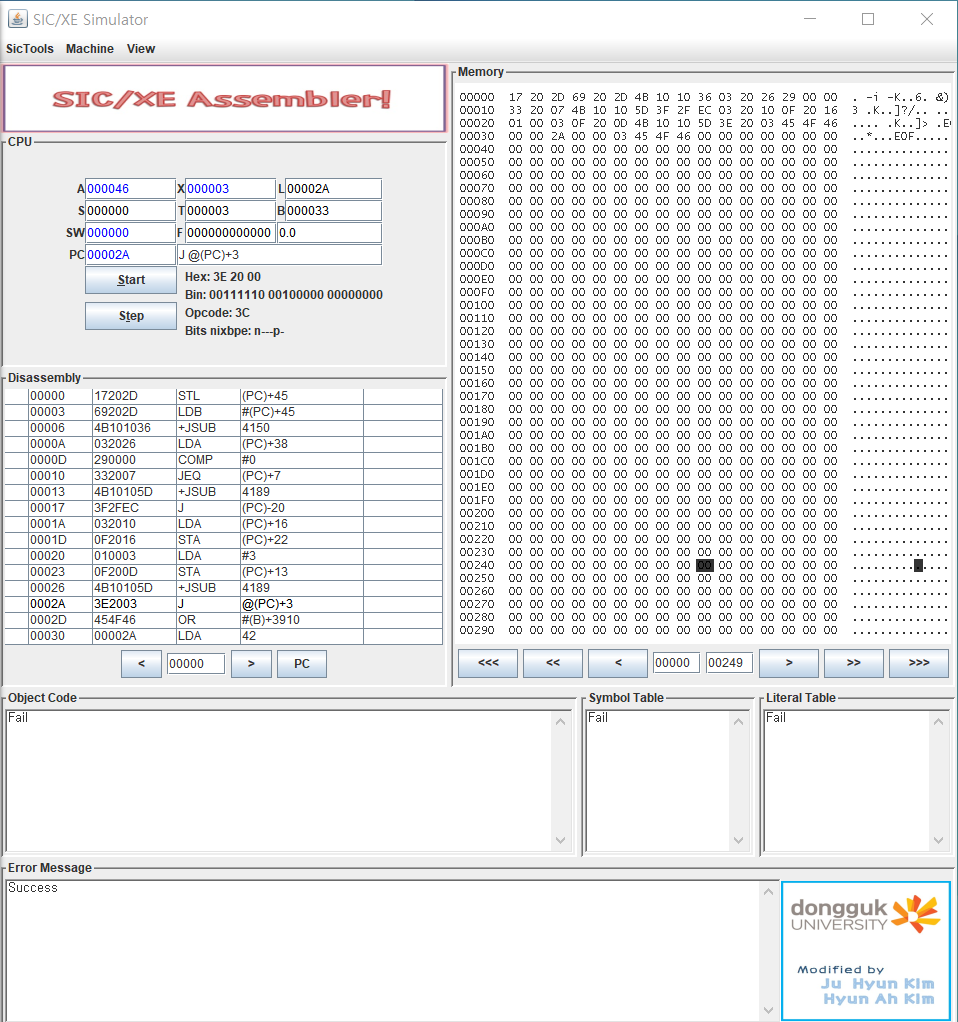
0002A은 nixbpe=100010이므로 indirect이고 pc-relative이다. opcode는 3C이므로 J이다. 주소는 @PC+3이다.

0002D는 nixbpe=010100이므로 immediate하고 base-relative이다. Opcode는 44이므로 OR이다. 주소는 #B+F46(3910)이다.

<START>



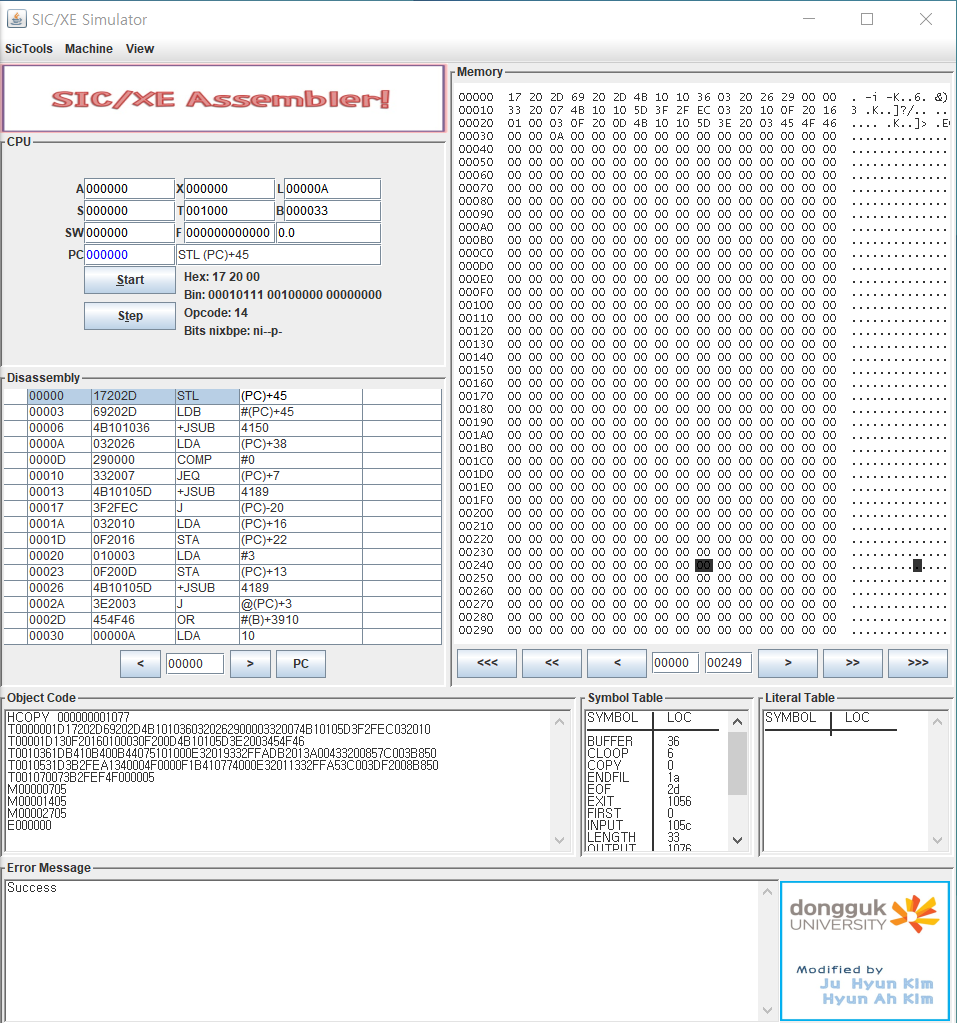
샘플코드를 Load하면 위와 같은 화면이 된다. 여기서 CPU란에 START를 실행해보자.



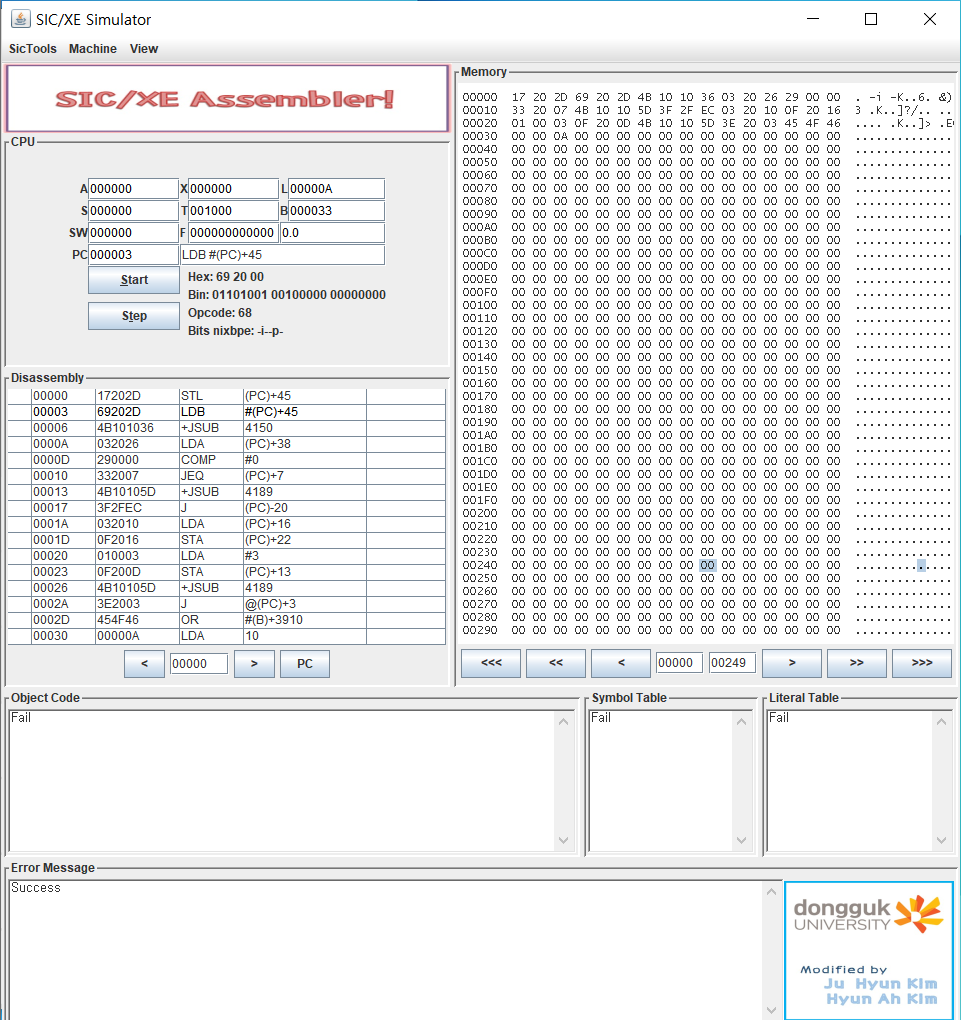
실행을 한 화면은 위와 같다. 레지스터 A에 000046이 들어가고 X에 000003, L에 00002A, T에000003가 들어가게 된다.

MEMORY상에 00032부분에 2A가 들어가 있는 것도 확인할 수 있다.

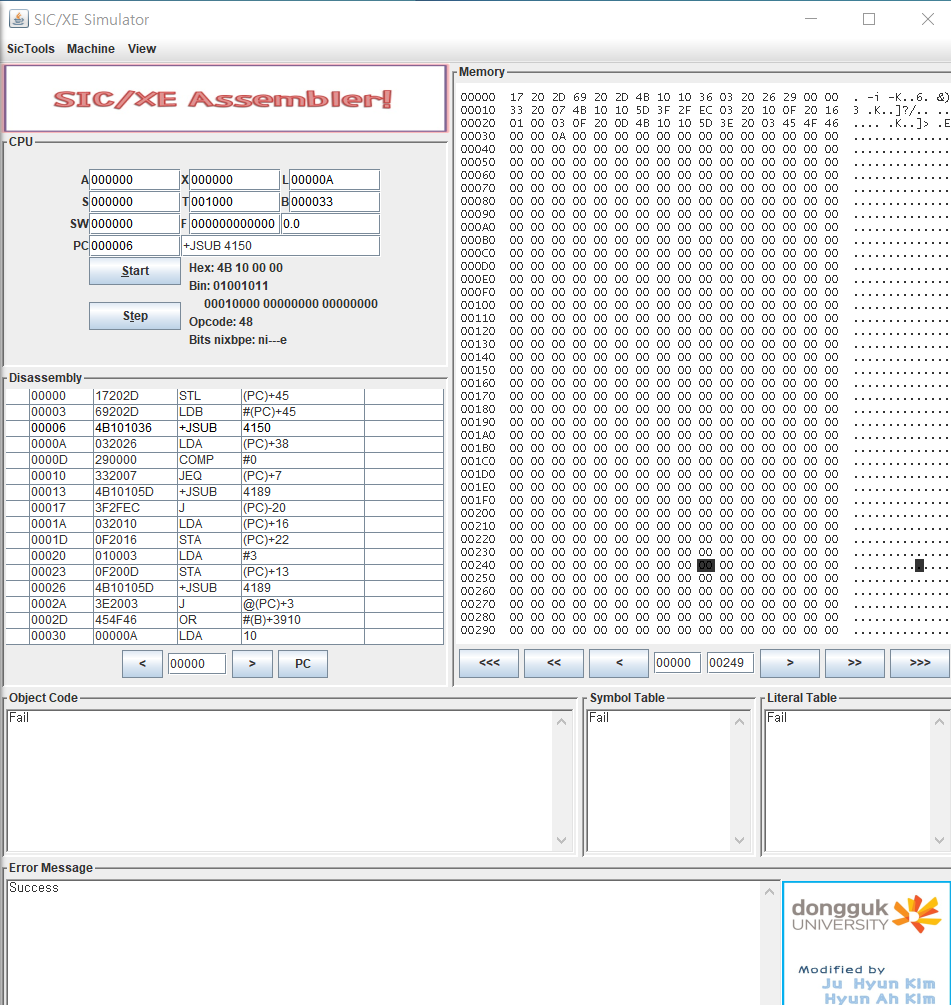
<STEP하기>



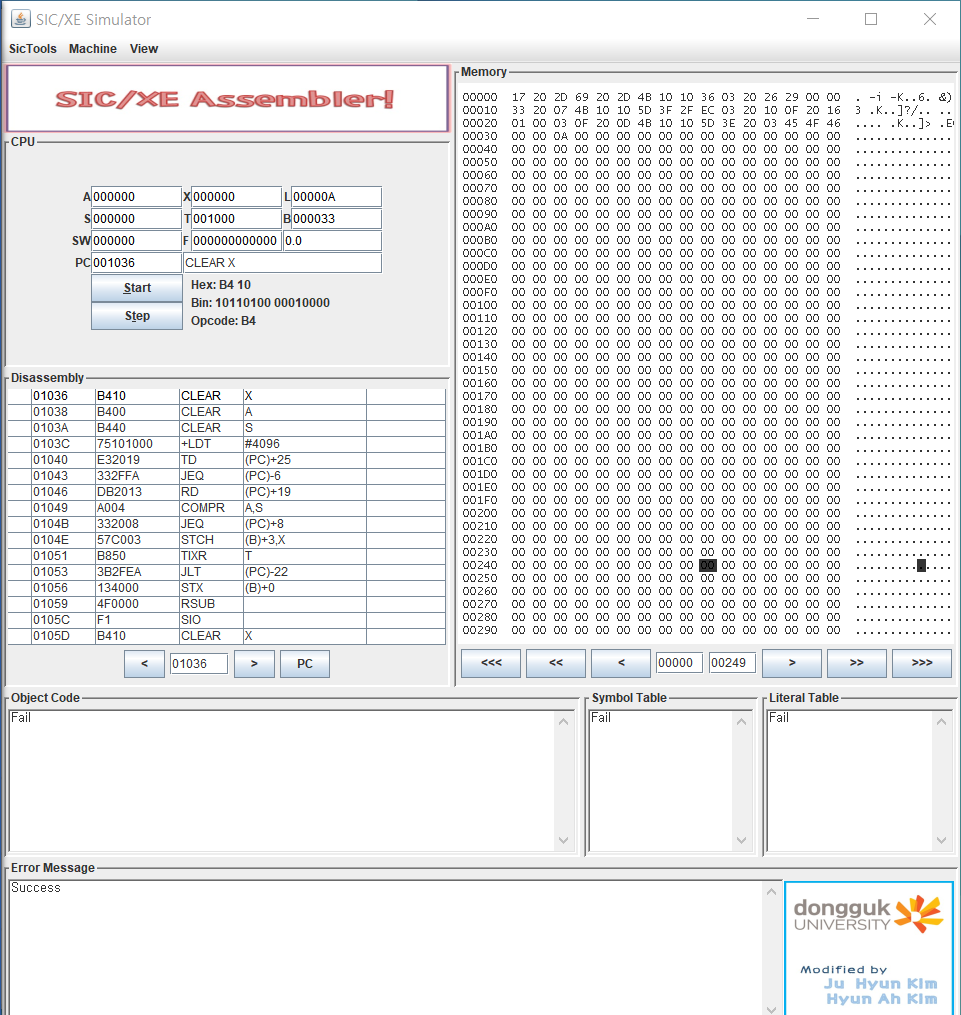
PC가 000000에 위치하게 되어 STL (PC)+45가 실행된다.



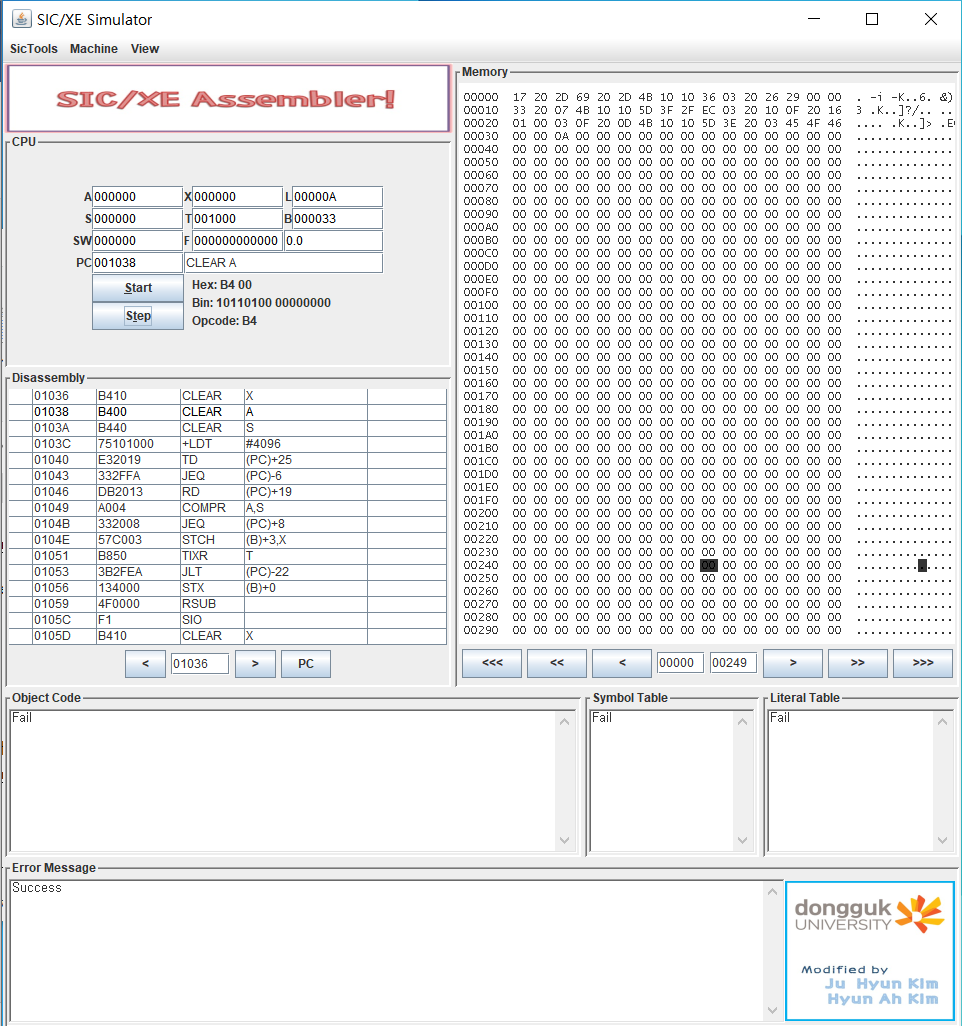
Step을 하면 PC가 000003으로 이동하고 그 위치에 있는 명령어 LDB #(PC)+45를 실행한다.



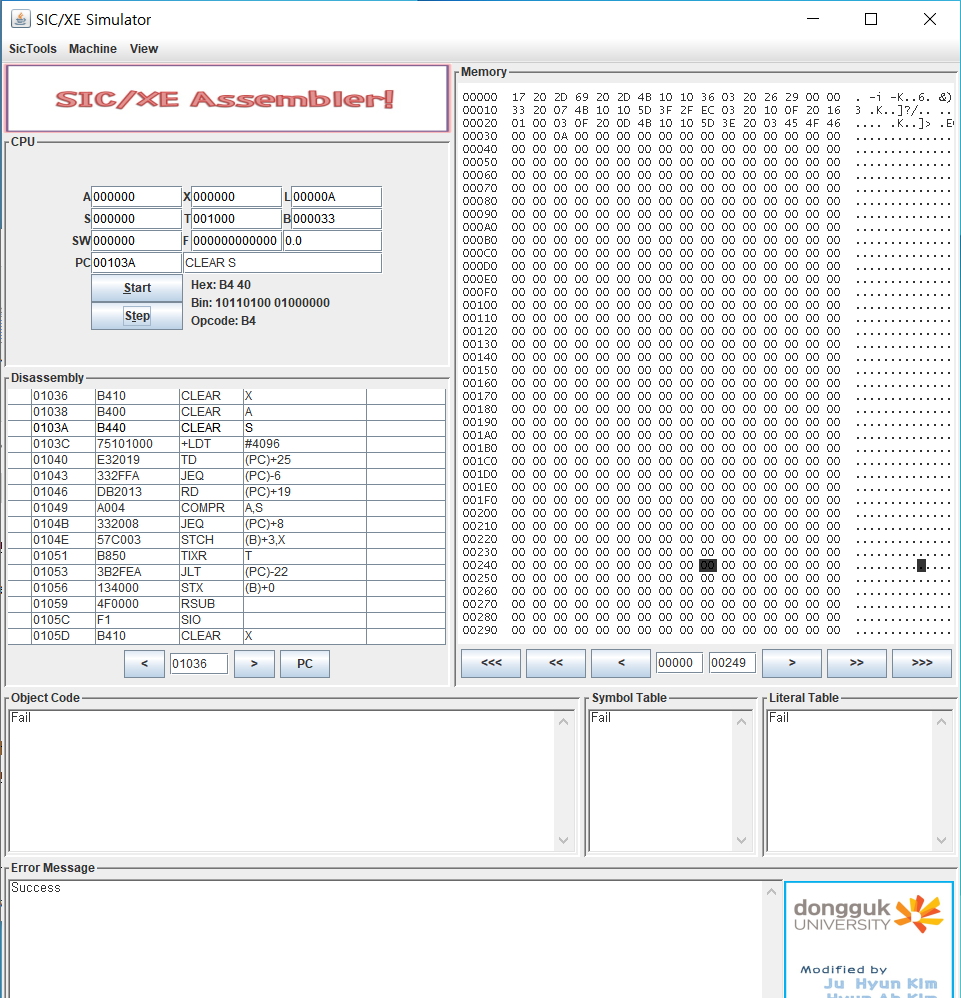
Step을 하면 PC가 000006에 위치하게 되고 그 위치에 명령어 +JSUB+4150을 실행한다.



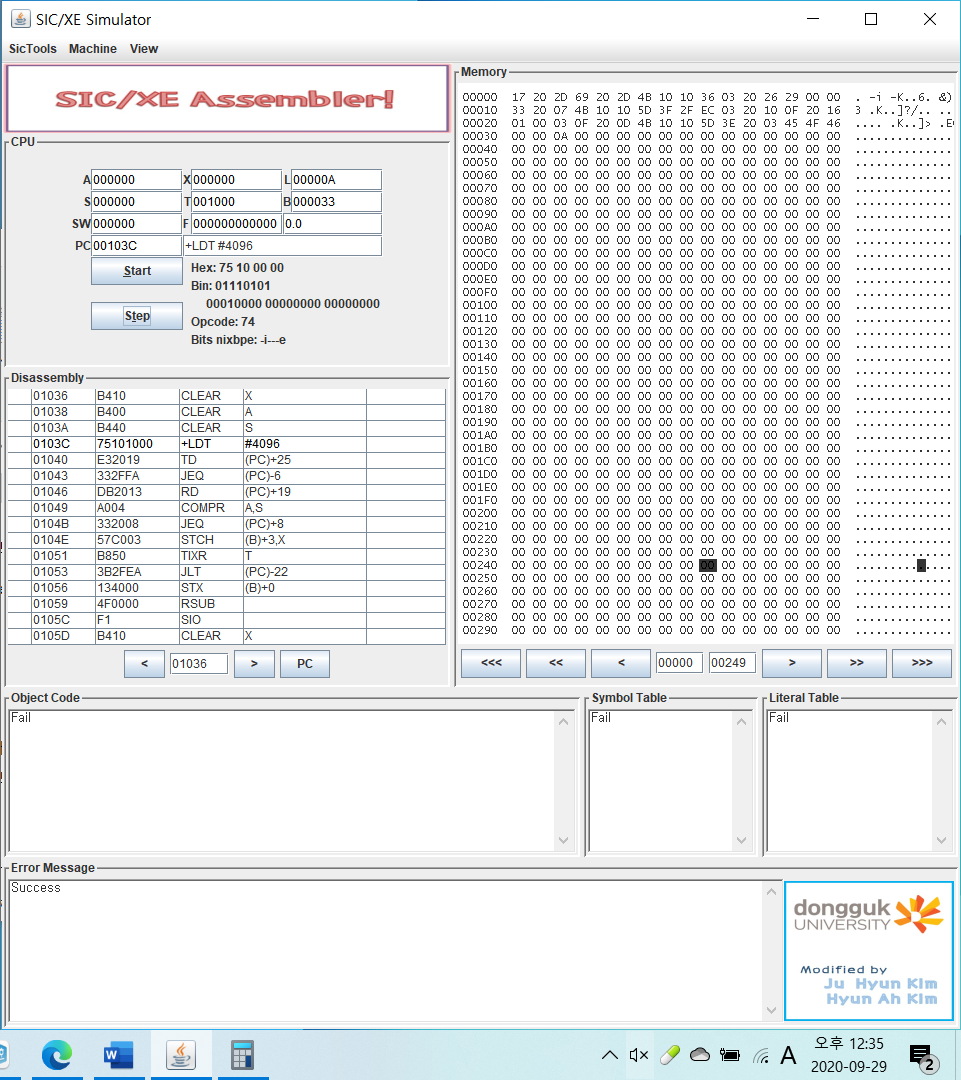
Step을 하면 PC가 1036에 위치하게 되는데 +JSUB+4150이 실행됐기 때문이다. 4150을 16진수로 표현하면 1036이기 때문이다. 1036번지에 존재하는 CLEAR X가 실행된다.



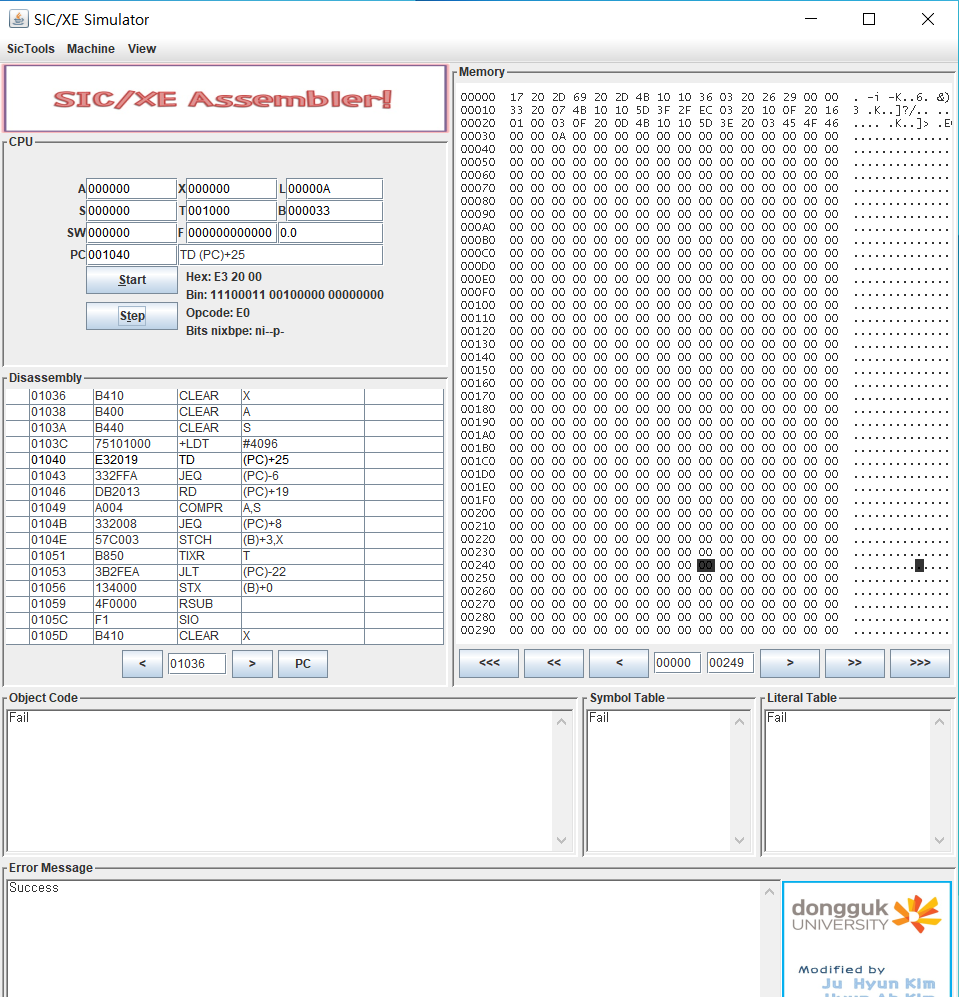
Step을 하면 PC가 1038에 위치하게 되고 그 위치에 있는 명령어 CLEAR A가 실행된다.



Step을 PC가 00103A에 위치하게 되고 그 주소에 있는 명령어 CLEAR S가 실행된다.

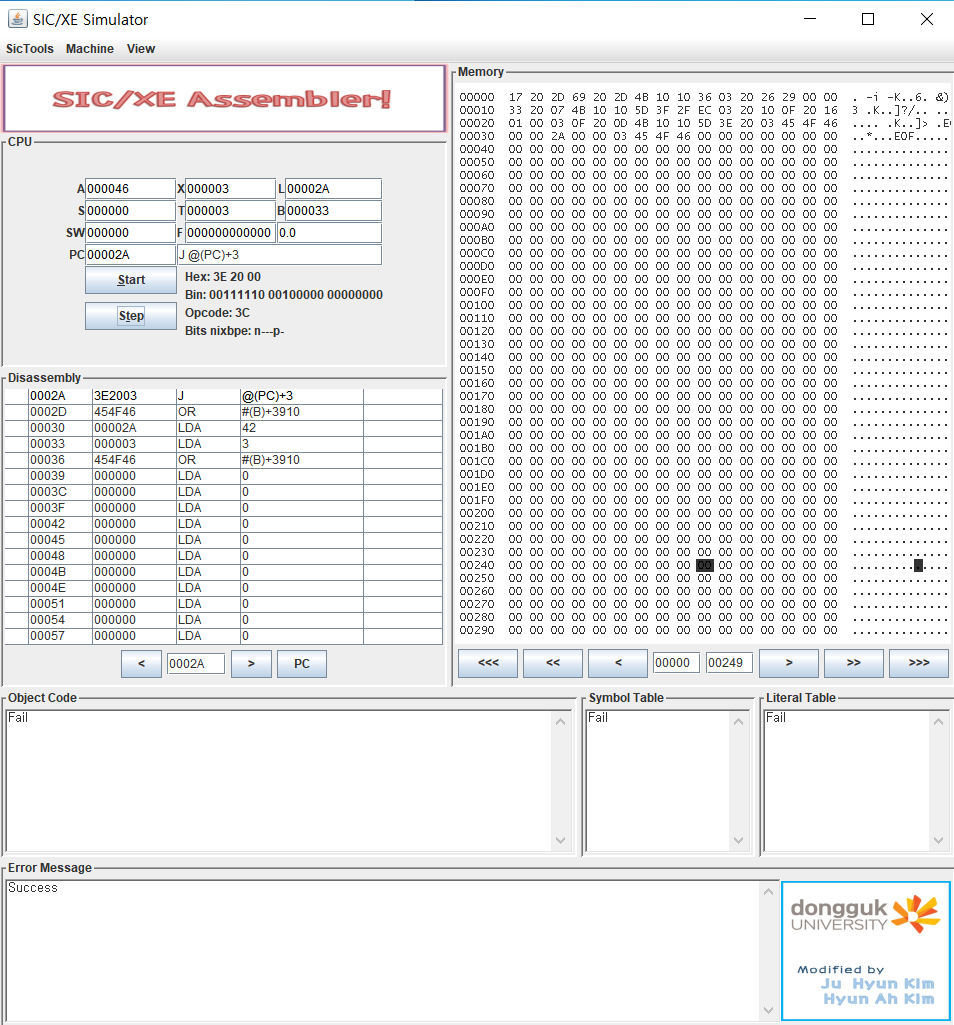


Step을 하면 PC가 00103C에 위치하게 되고 그 주소지에 있는 명령어 +LDT #4096이 실행된다.



Step을 하면 PC가 001040에 위치하게 되고 그 주소에 있는 명령어 TD (PC)+25가 실행된다.

이렇게 계속 step을 해서 끝날 때까지 반복하면 다음과 같은 화면이 된다.



이는 Start를 실행한 것과 결과화면이 동일한 것을 확인할 수 있다.

**3번 1번과 2번에서 SIC/XE GUI버전과 및 콘솔버전 sicsim을 사용해서 실행해 본 결과 (캡처된 내용)를 바탕으로 향후 실습보고서에 개인적인 생각과 느낌을 에세이 형식으로 정리하여 추가합니다.**

콘솔버전 sicsim을 사용해본 결과 코드를 쓰는 형식이 엄격하고 지켜야했다. 그리고 실행하는 방법도 쉽지 않았다. 하지만 도스박스를 이용해서 사용해보면서 도스박스 같은 리눅스에서의 명령어를 배울 수 있어서 좋았고 LISFILE을 통해서 각 줄마다 opcode를 볼 수 있는 것도 좋았다. SICSIM.EXE를 실행하는 법도 배울 수 있어서 좋았다. 명령어도 처음 생각했던 것보다는 어렵지 않고 이해하기 쉬웠다.

GUI버전 SIC/XE를 사용해본 결과는 사용법이 굉장히 간단하고 object code와 메모리 그리고 레지스터 데이터 PC의 위치를 보기 쉽게 되어있어서 편하게 사용할 수 있었다.

**실습#2**

**1번. 교재의 2장 Figure 2.1의 SIC assembly program을 시뮬레이터로 실행시켜 결과를 얻는다. 이때 이 결과를 Figure 2.2의 프로그램 및 Figure2.3의 object프로그램과 비교하여 제대로 결과가 생성되는지 비교하라.**

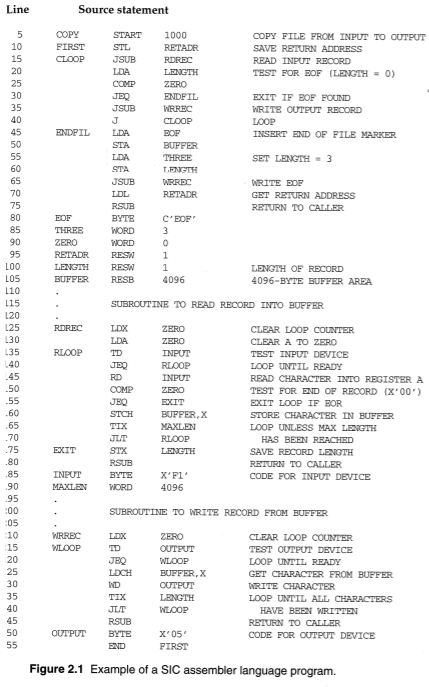
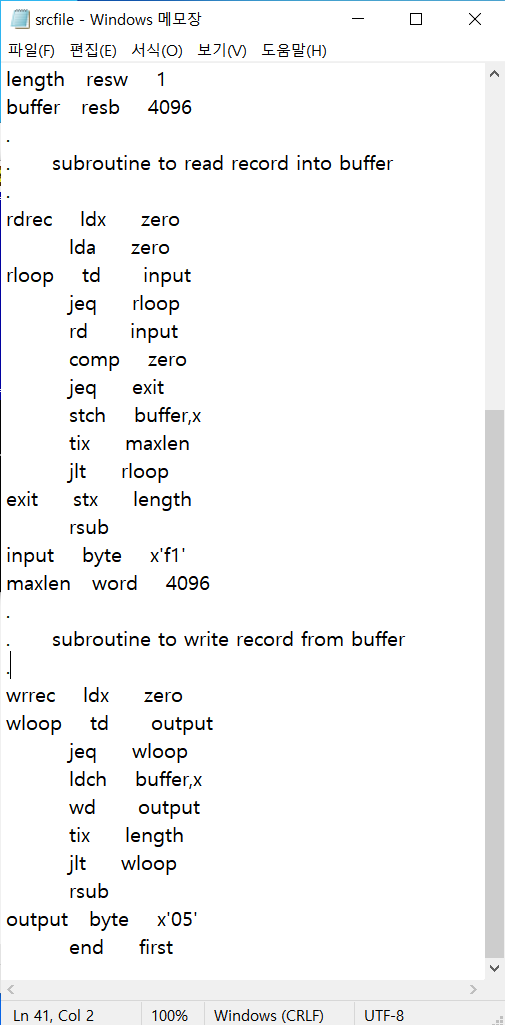
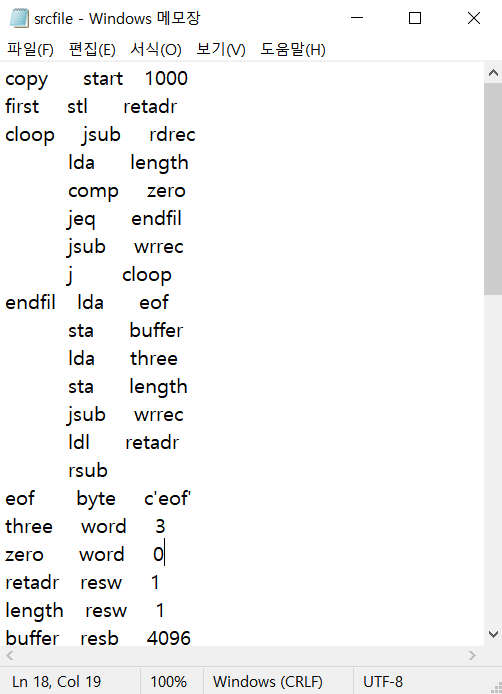
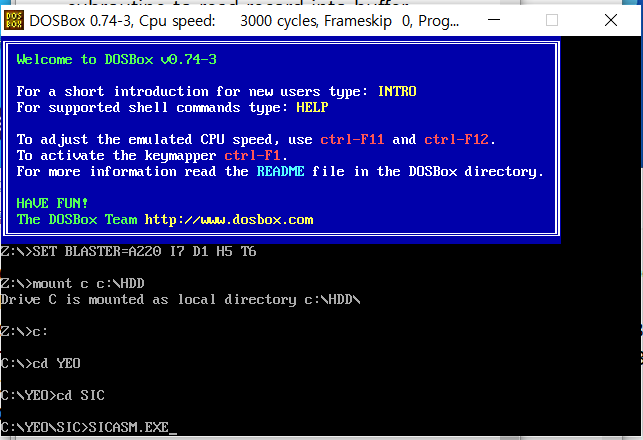


Figure 2.1의 코드를 srcfile에 적어서 저장한다.

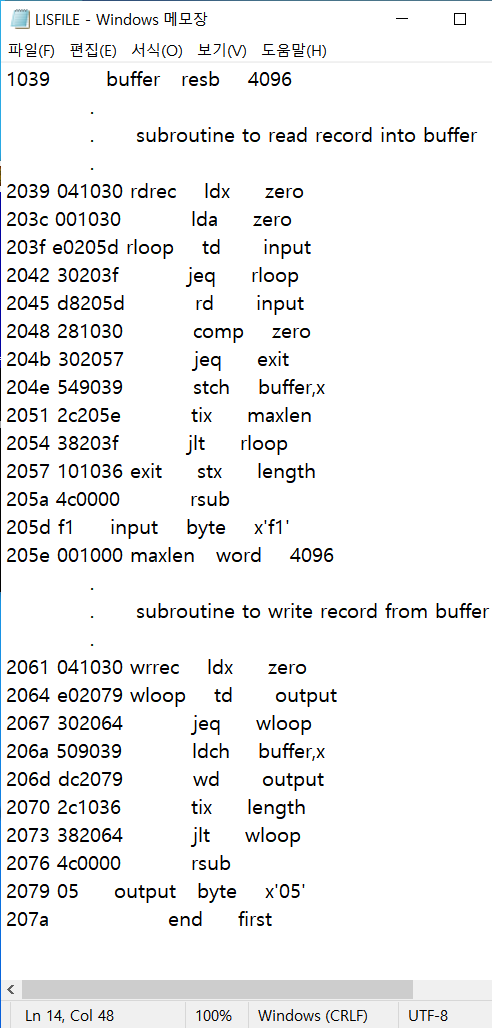
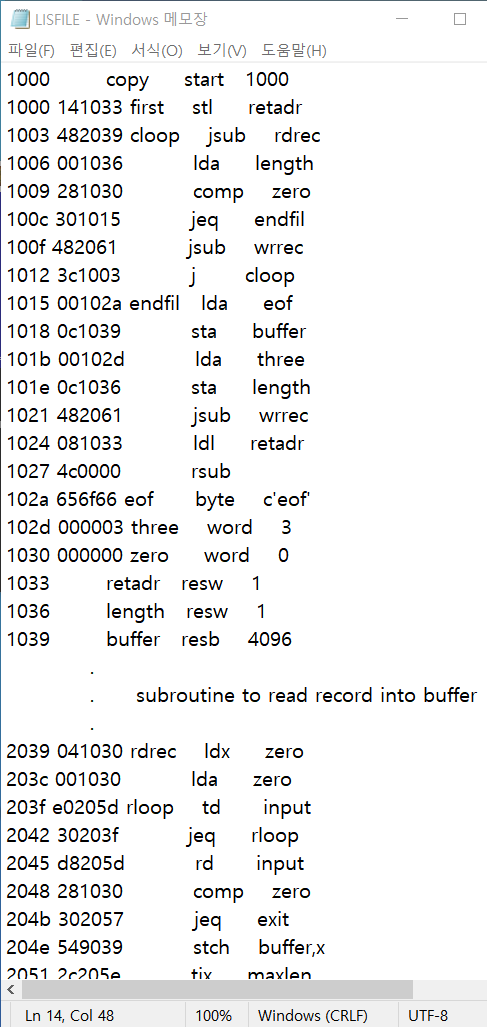


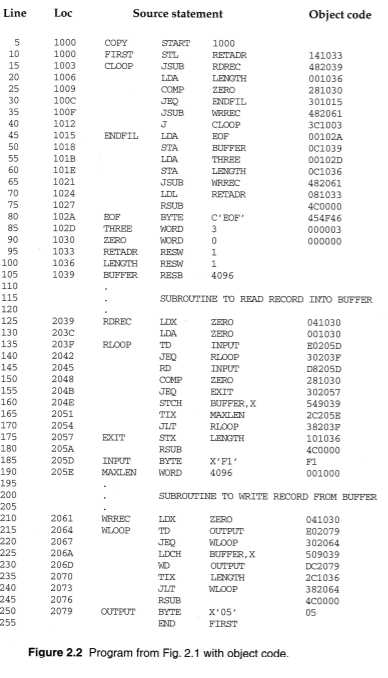
위와 같이 파일에 입력하여 저장한다.

그 이후에 SICASM.EXE를 실행시킨다.



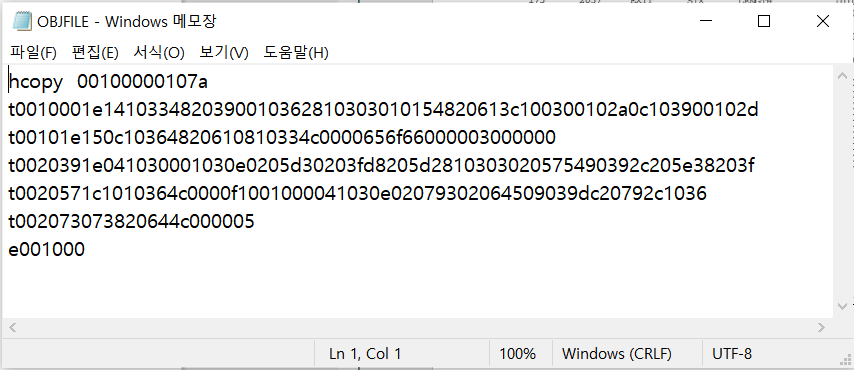
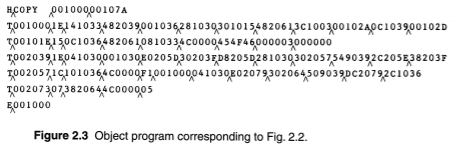
LISFILE을 열어서 Figure 2.2의 프로그램과 비교해본다.





Object code가 동일하게 나오는 것을 확인했다.

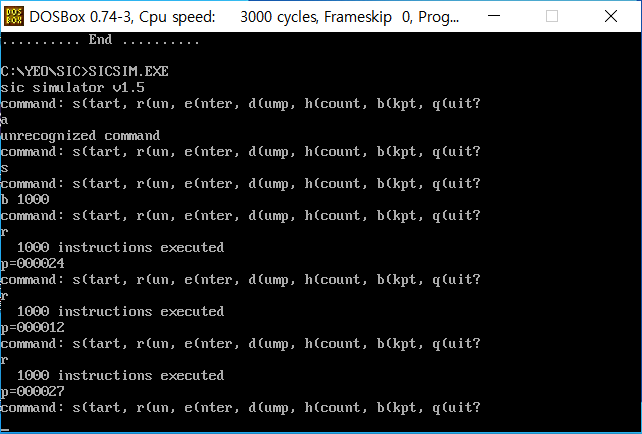
그 이후 생성된 OBJFILE을 열어서 Figure3과 비교한다.

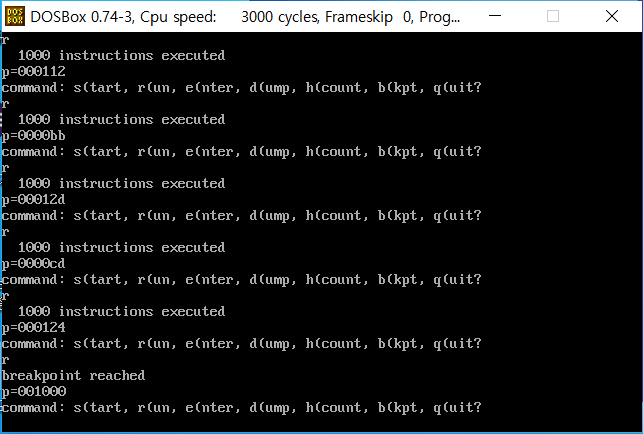
 

확인한 결과 동일하게 나온 것을 확인했다.

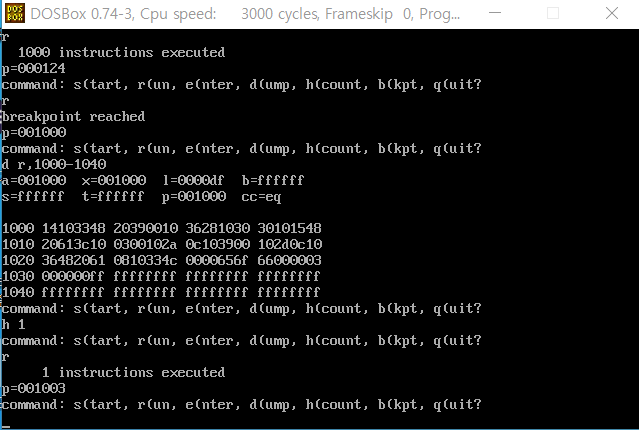
**2번. 이 과정에서 시뮬레이터를 통해 실행환경의 전체 메모리를 단계별로 dump하여 실제 메모리의 내용을 확인해본다. 또한, 이 과정에서 확인한 내용을 단계별로 모두 화면 캡처하여 보고서를 작성한다.**

OBJFILE을 DEVF2로 이름변경을 해주고 L2U.EXE를 통해 DEVF2에 있는 코드를 모두 대문자로 변경해주고 SICISM.EXE를 실행한다.

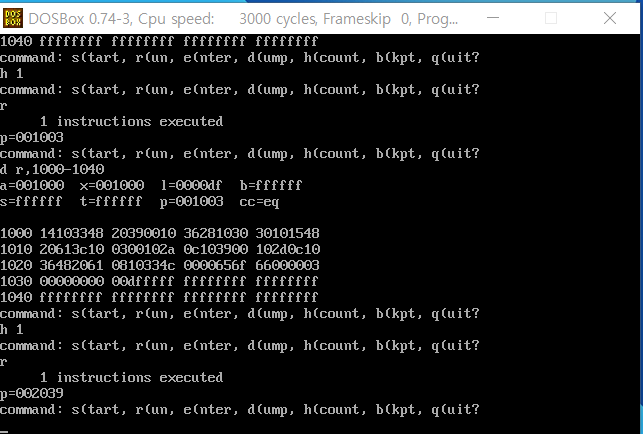
s를 입력하여 프로그램을 실행하고 b 1000을 통해 시작주소를 입력해주고 r을 break point에 도달할 때까지 입력해준다.



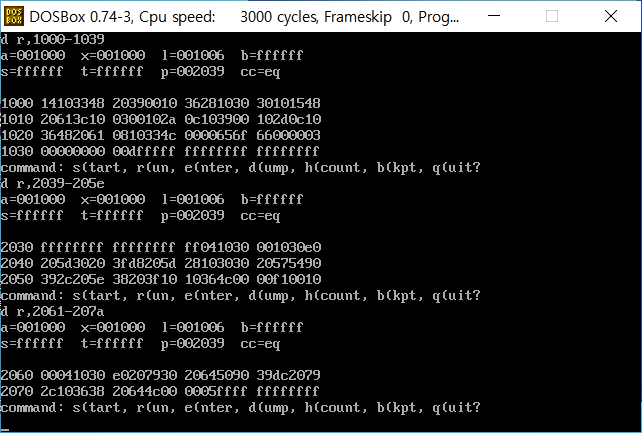
Break point에 도달한 모습이다.



d r, 1000-1040을 통해 시작주소부터 1040번지까지의 명령어와 현재 상황을 볼 수 있다. 여기서 PC는 1000번지 즉 시작주소에 위치하고 있는 것을 볼 수 있다. 이후에 h 1을 입력하고 r을 입력해서 실행하면 pc가 1003번지에 위치하는 것을 볼 수 있다.

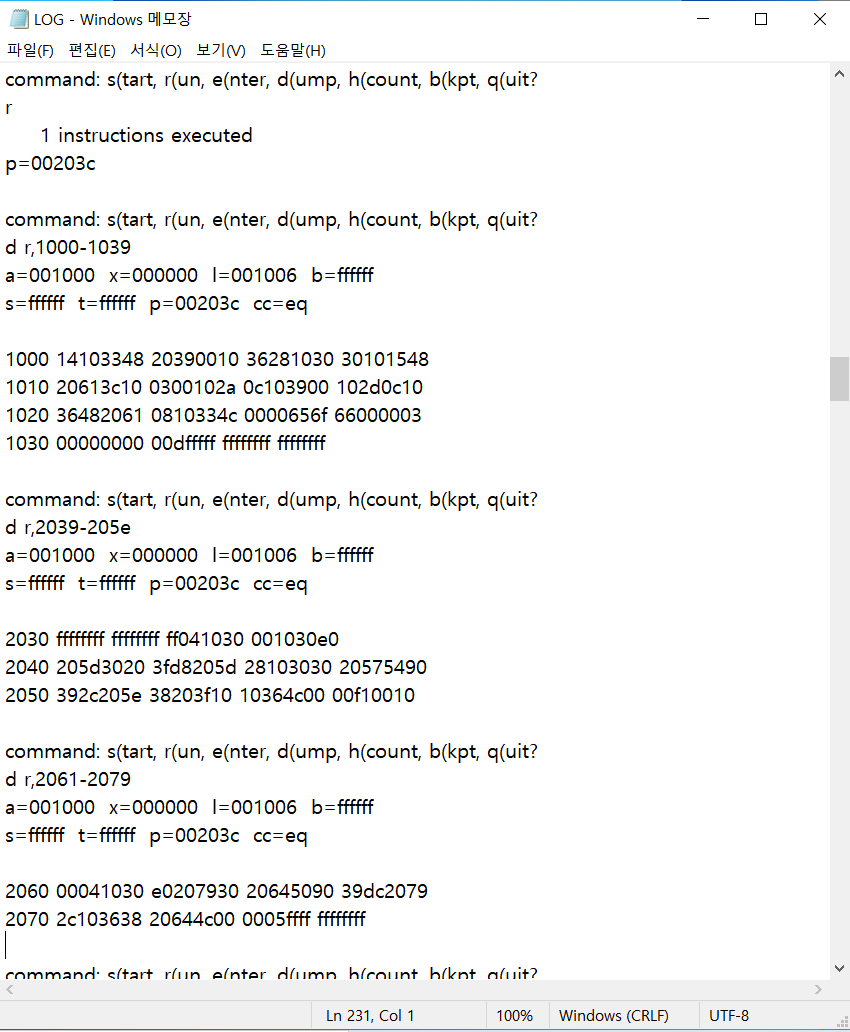
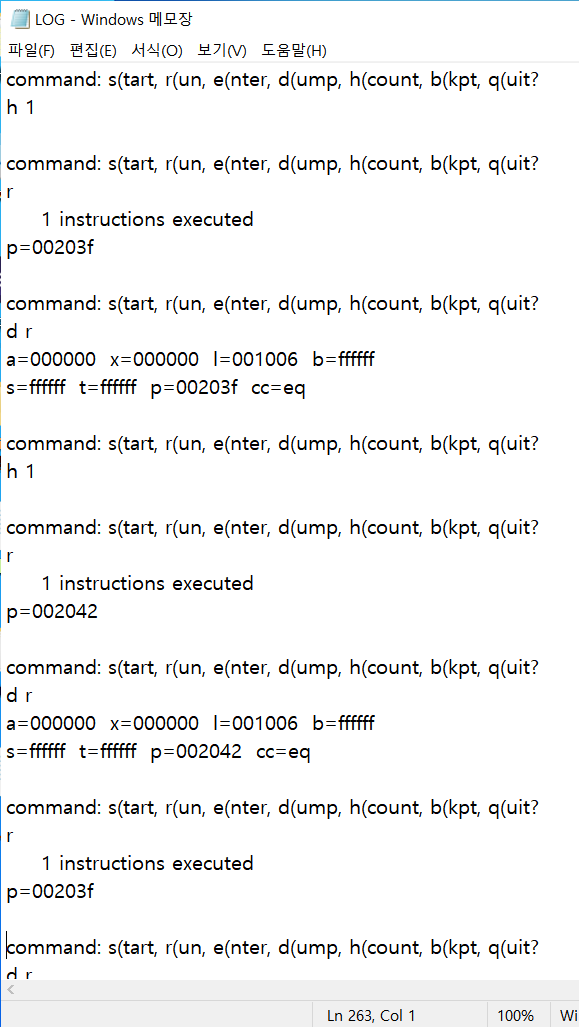


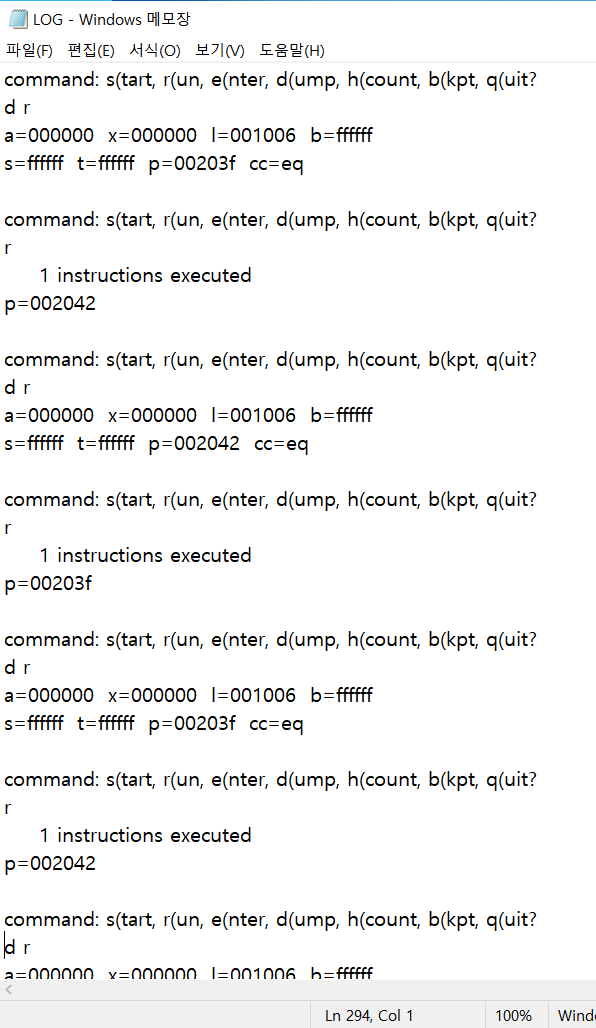
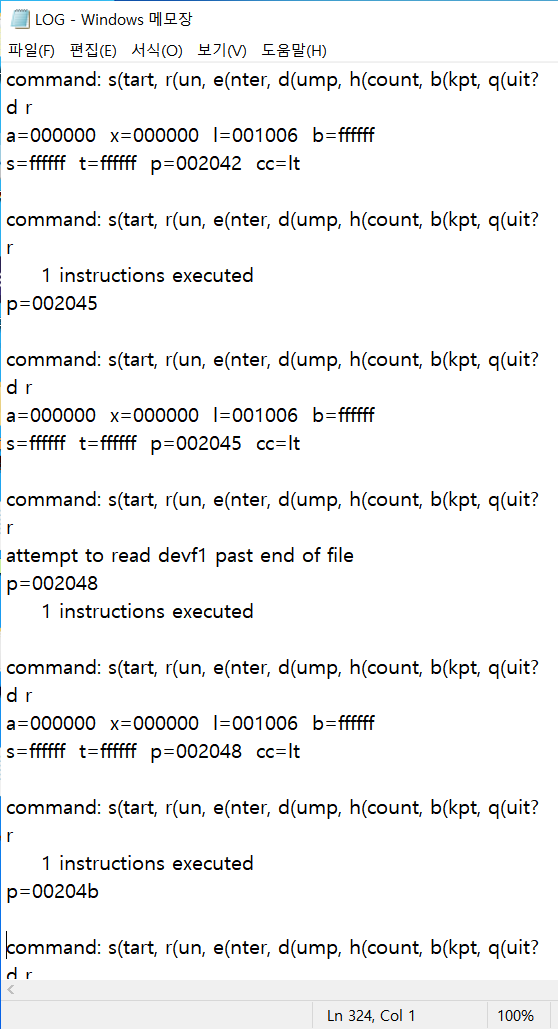
Pc가 1003번지에 위치하는 것을 볼 수 있다. 그 이후 다시 명령어 1개를 실행하기 위해 h 1을 입력하고 r을 입력하여 실행하면 RDREC의 주소 2039로 점프하여 pc가 2039에 위치하는 것을 볼 수 있다.

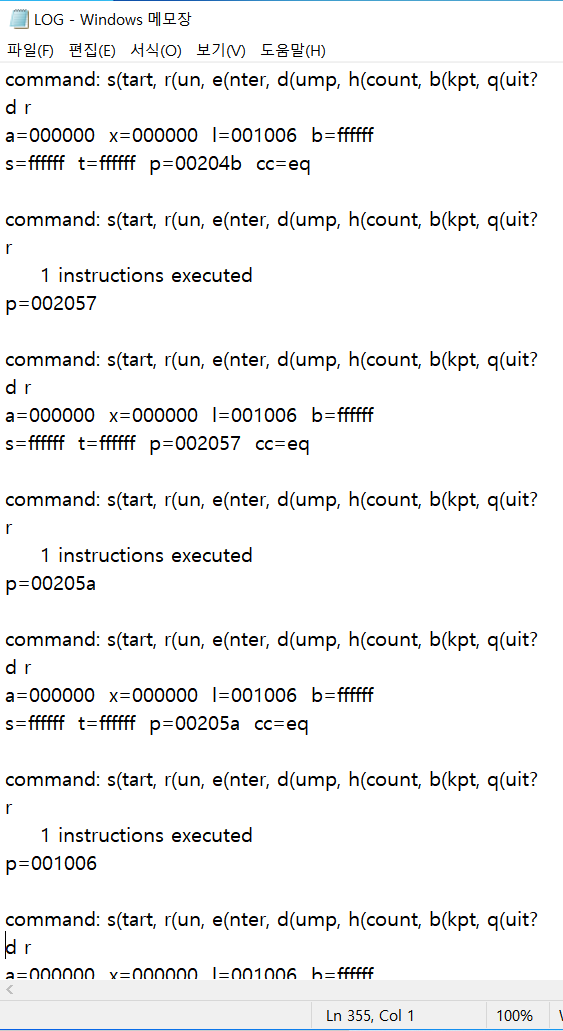
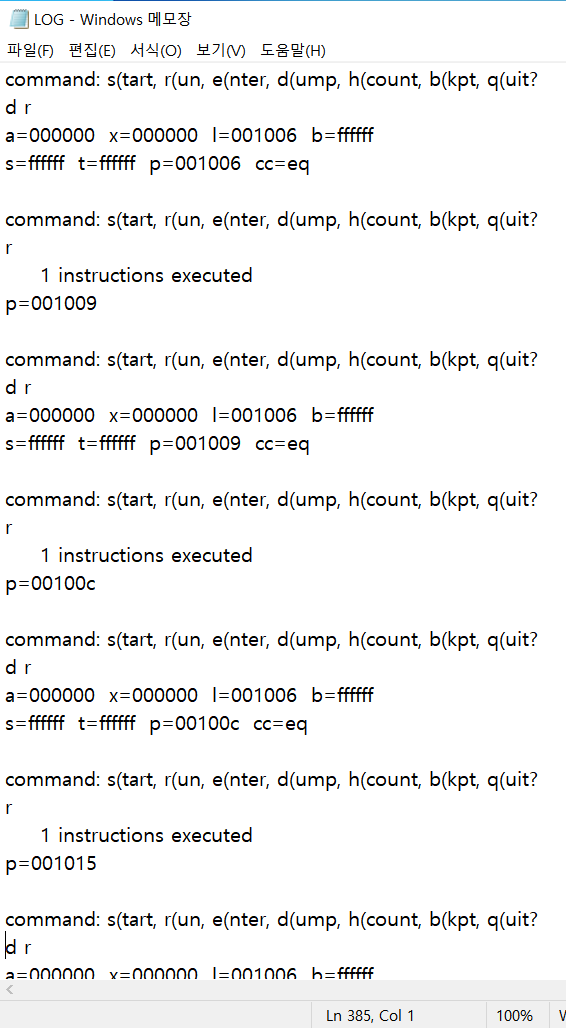
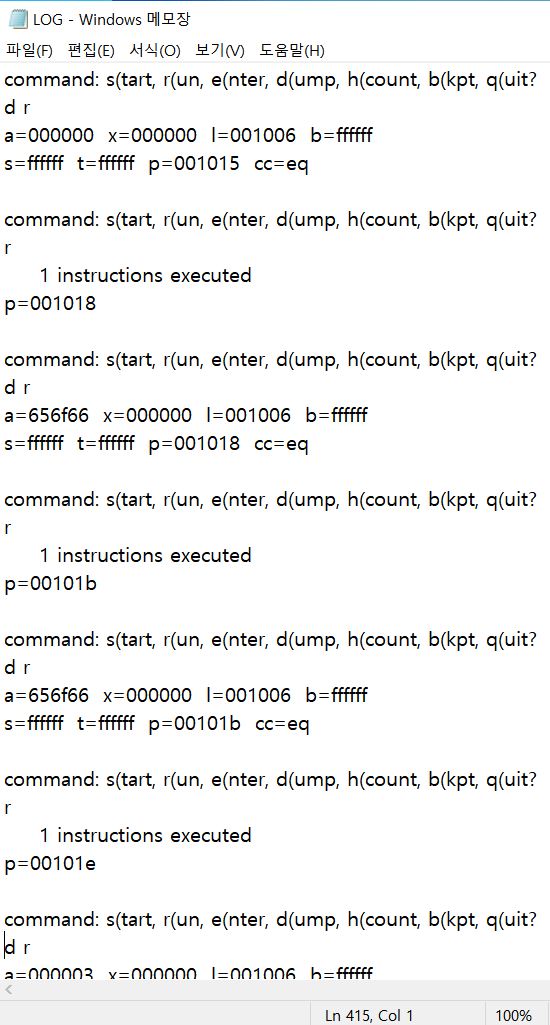
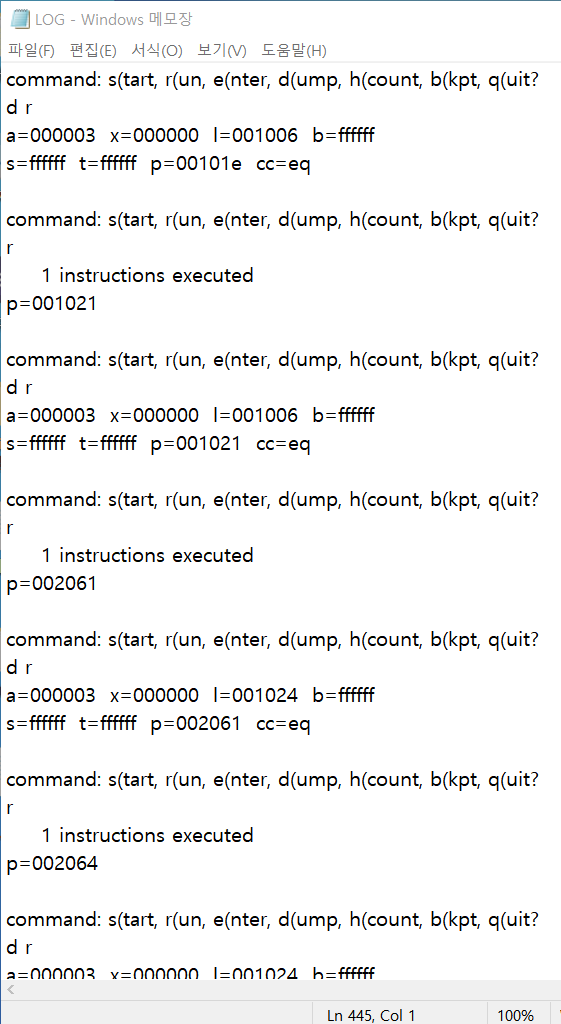


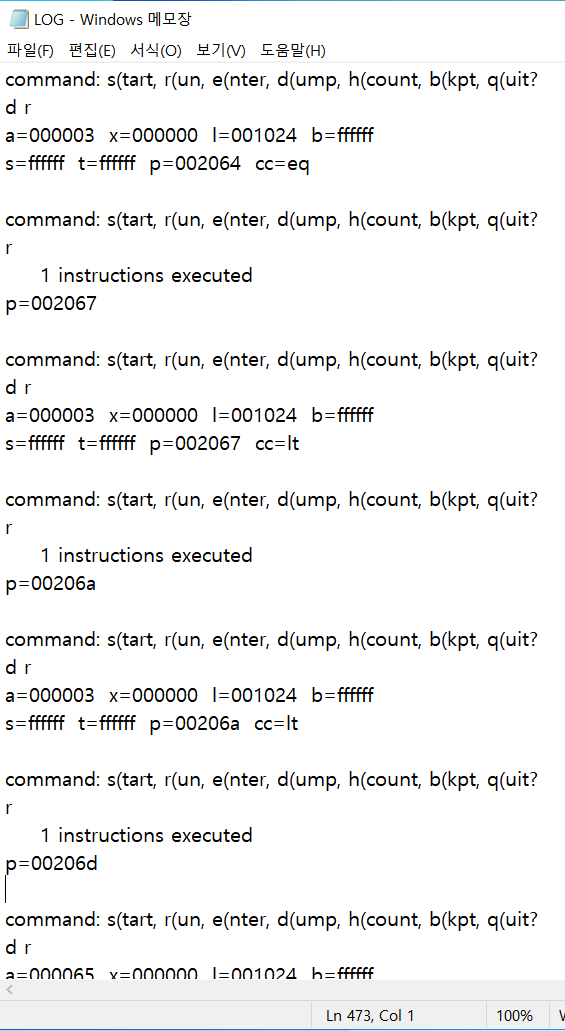
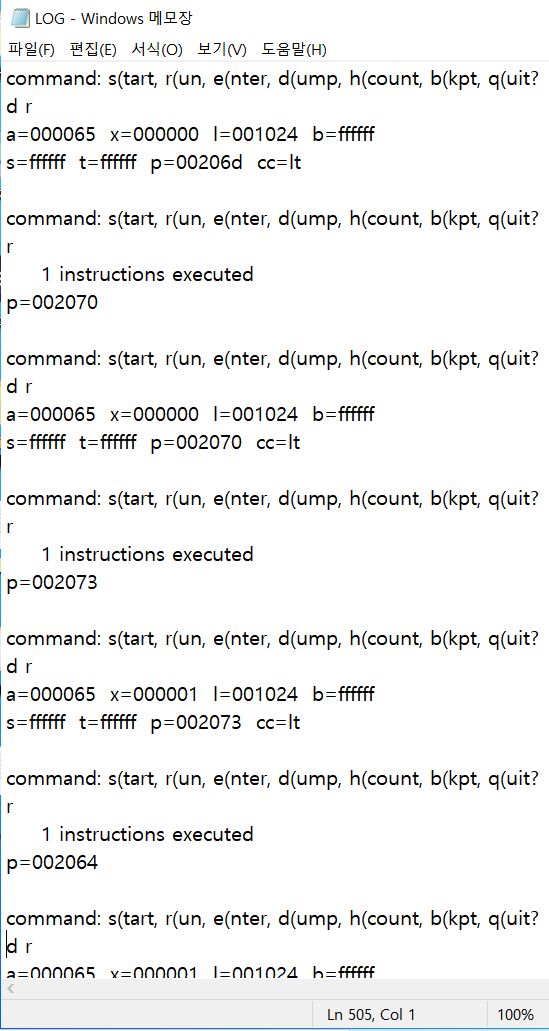
레지스터 L에 001006이 저장된 것을 확인했고 pc가 2039번지에 위치하고 있는 것을 확인했다.

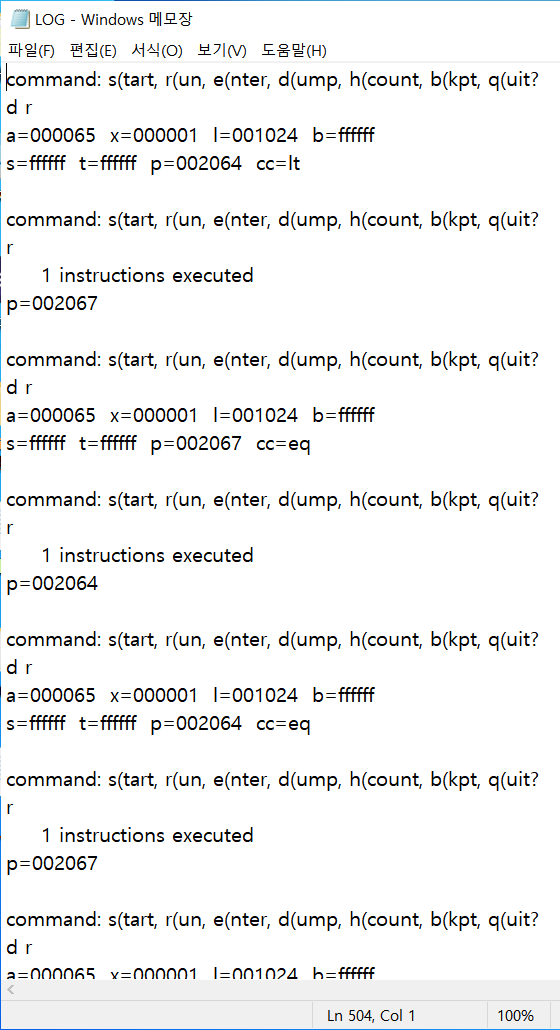
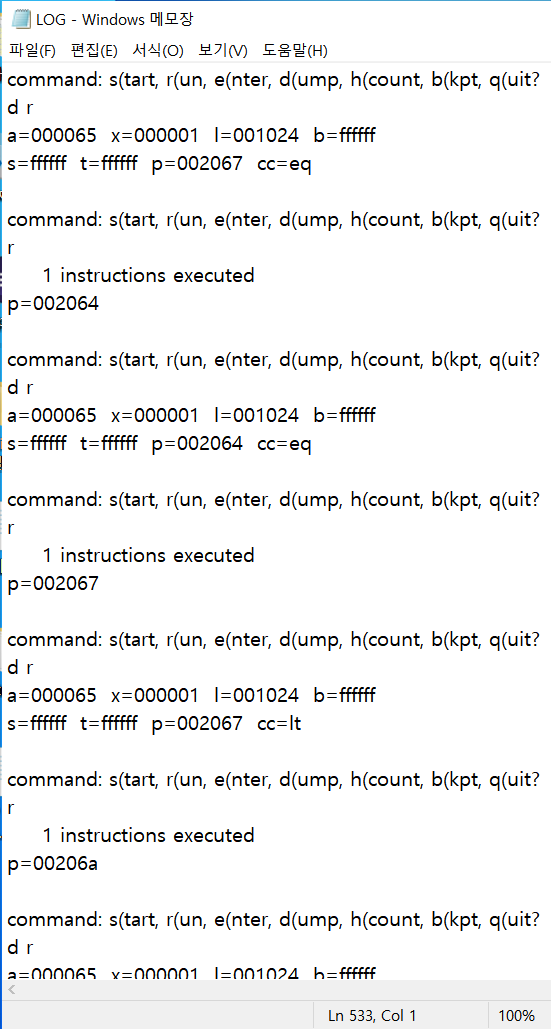
이것을 반복해서 pc가 다시 1000번지 즉 END에 도달할 때까지 반복해 주고 LOG파일을 통해 확인한다. 아래는 위의 실행화면 뒤를 이어서 실행한 LOG파일을 캡처한 사진들이다.

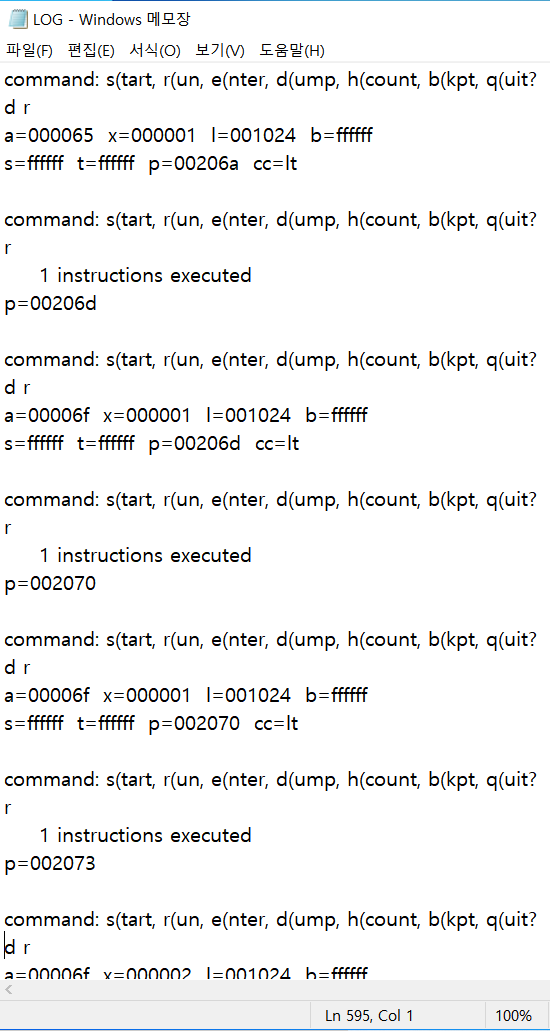
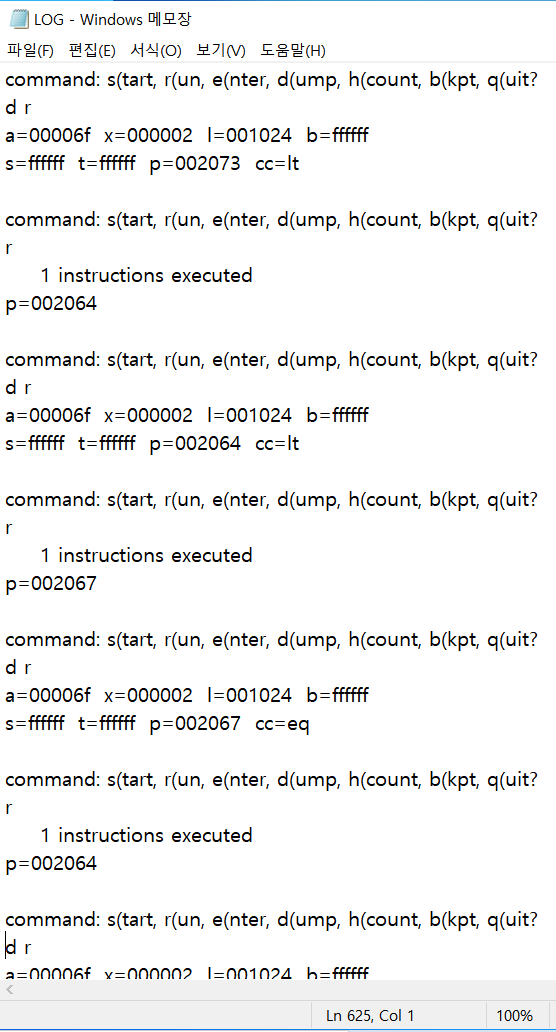
 

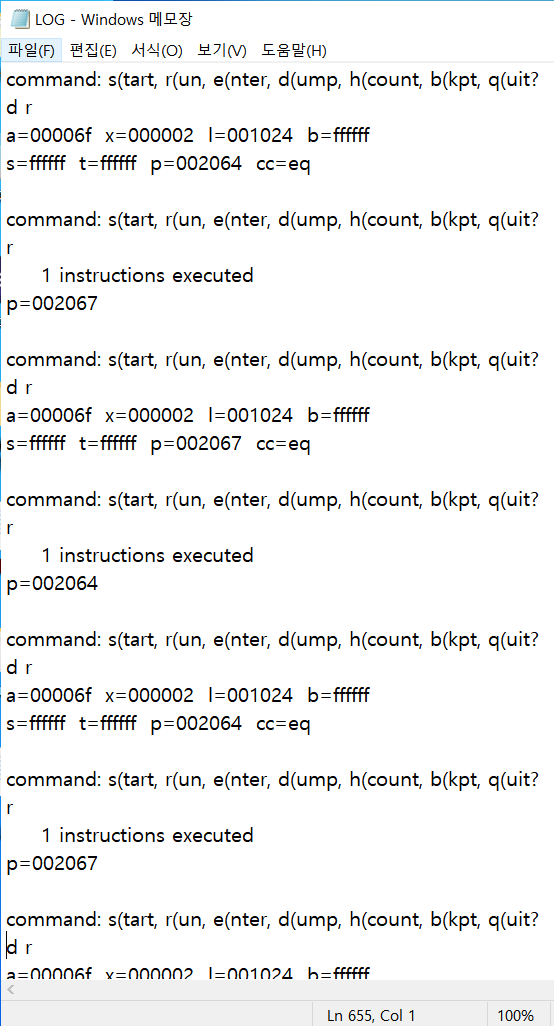
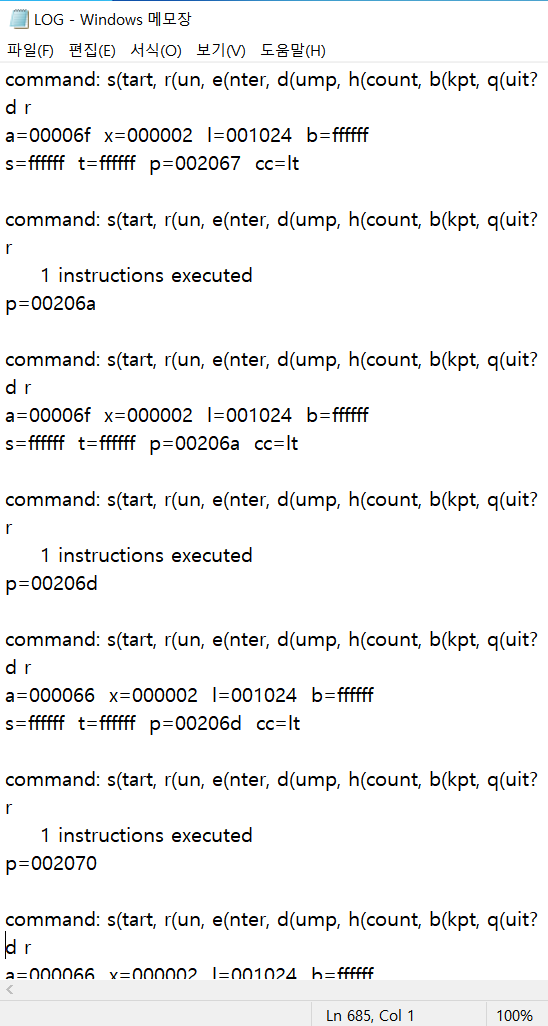
 

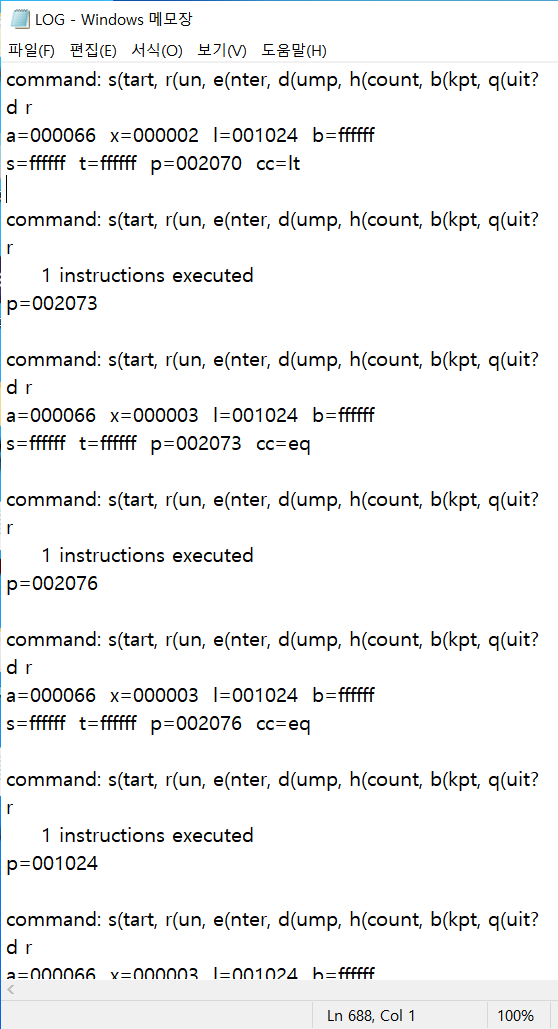
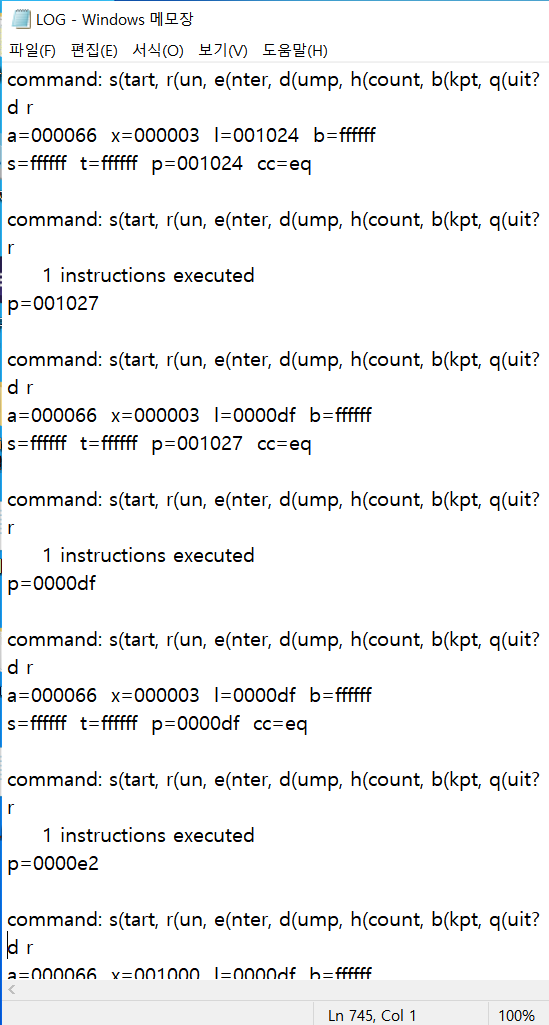
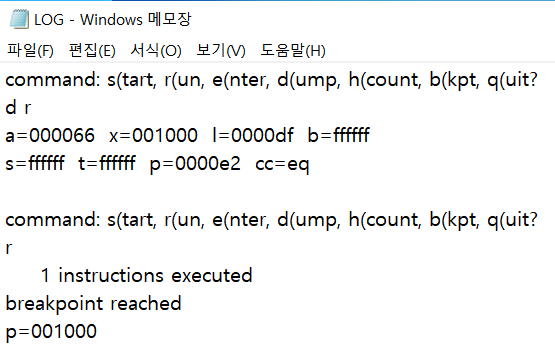
   

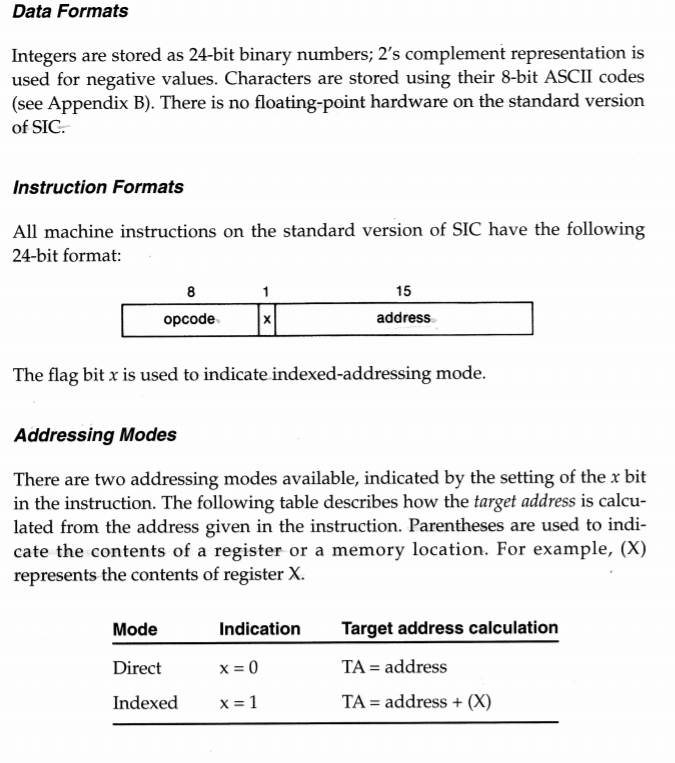
 

프로그램을 1개의 명령어씩 실행시켰을 때의 모습이다. Pc가 다시 1000번지(END)를 가리키면서 break point에 도달했다는 문구가 나오게 된다. 각각 실행했을 때마다 명령어가 하나씩 실행되고 레지스터에 있는 값이 변동도 되는 것을 확인할 수 있다.

**3번. 한편, 1장의 1.3.2절에서 언급된 Data Formats 및 Instruction Formats 그리고 Memory Addressing Modes들을 관련시켜 Figure 2.1의 소스 프로그램의 코드를 각 라인 별로 분석하여 설명하라.**



1. FIRST STL RETADR 141033

STL의 OPCODE(8 bit)는 14이고 1033을 2진수로 표현하면 0001 0000 0011 0011이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1033이 된다.

2. CLOOP JSUB RDREC 482039

JSUB의 OPCODE(8 bit)는 48이고 2039를 2진수로 표현하면 0010 0000 0011 1001이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 2039가 된다.

3. LDA LENGTH 001036

LDA의 OPCODE(8 bit)는 00이고 1036을 2진수로 표현하면 0001 0000 0011 0110이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1036이 된다.

4. COMP ZERO 281030

COMP의 OPCODE(8 bit)는 28이고 1030을 2진수로 표현하면 0001 0000 0011 0000이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1030이 된다.

5. JEQ ENDFIL 301015

JEQ의 OPCODE(8 bit)는 30이고 1015를 2진수로 표현하면 0001 0000 0001 0101이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1015가 된다.

6. JSUB WRREC 482061

JSUB의 OPCODE(8 bit)는 48이고 2061을 2진수로 표현하면 0010 0000 0110 0001이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 2061이 된다.

7. J CLOOP 3C1003

J의 OPCODE(8 bit)는 3C이고 1003을 2진수로 표현하면 0001 0000 0000 0011이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1003이 된다.

8. ENDFIL LDA EOF 00102A

5번째 줄에서 ENDFIL의 주소가 1015인 것을 알았으니 ENDFIL은 1015에 위치한다. LDA의 OPCODE(8 bit)는 00이고 102A를 2진수로 표현하면 0001 0000 0010 1010이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 102A가 된다.

9. STA BUFFER 0C1039

STA의 OPCODE(8 bit)는 0C이고 1039를 2진수로 표현하면 0001 0000 0011 1001이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1039가 된다.

10. LDA THREE 00102D

LDA의 OPCODE(8 bit)는 00이고 102D를 2진수로 표현하면 0001 0000 0010 1101이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 102D가 된다.

11. STA LENGTH 0C1036

STA의 OPCODE(8 bit)는 0C이고 1036을 2진수로 표현하면 0001 0000 0011 0110이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1036이 된다.

12. JSUB WRREC 482061

JSUB의 OPCODE(8 bit)는 48이고 2061을 2진수로 표현하면 0010 0000 0110 0001이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 2061이 된다.

13. LDL RETADR 081033

LDL의 OPCODE(8 bit)는 08이고 1033을 2진수로 표현하면 0001 0000 0011 0011이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1033이 된다.

14. RSUB 4C0000

RSUB의 OPCODE(8 bit)는 4C이다.

15. EOF BYTE C’EOF’ 454F46

8번째 줄에서 EOF의 주소가 102A인 것을 알았으니 EOF는 102A에 위치한다. 그리고 E는 45, O는 4F, F는 46이므로 object code는 454F46이 된다.

16. THREE WORD 3 000003

10번째 줄에서 THREE의 주소가 102D인 것을 알았으니 THREE는 102D에 위치한다. 그리고 상수 3이므로 object code는 000003이 된다.

17. ZERO WORD 0 000000

4번째 줄에서 ZERO의 주소가 1030인 것을 알았으니 ZERO는 1030에 위치한다. 그리고 상수 0이므로 object code는 000000이 된다.

18. RETADR RESW 1

1번째 줄에서 RETADR 의 주소가 1033인 것을 알았으니 RETADR은 1033에 위치한다. 1만큼의 워드를 예약한다.

19. LENGTH RESW 1

3번째 줄에서 LENGTH의 주소가 1036인 것을 알았으니 LENGTH는 1036에 위치한다. 1만큼의 워드를 예약한다.

20. BUFFER RESB 4096

9번째 줄에서 BUFFER의 주소가 1039인 것을 알았으니 BUFFER는 1039에 위치하게 된다. 4096만큼의 바이트를 예약한다.

21. RDREC LDX ZERO 041030

2번째 줄에서 RDREC의 주소가 2039인 것을 알았으니 RDREC은 2039에 위치한다. LDX의 OPCODE(8 bit)는 04이고 1030을 2진수로 표현하면 0001 0000 0011 0000이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1030이 된다. ZERO에 있는 VALUE가 X register에 load된다.

22. LDA ZERO 001030

LDA의 OPCODE(8 bit)는 00이고 1030을 2진수로 표현하면 0001 0000 0011 0000이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1030이 된다.

ZERO에 있는 VALUE가 A register에 load된다.

23. RLOOP TD INPUT E0205D

24번째 줄에서 RLOOP의 주소가 203F인 것을 알았으니 RLOOP은 203F에 위치한다. TD의 OPCODE(8 bit)는 E0이고 205D를 2진수로 표현하면 0010 0000 0101 1101이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 205D가 된다.

24. JEQ RLOOP 30203F

JEQ의 OPCODE(8 bit)는 30이고 203F를 2진수로 표현하면 0010 0000 0011 1111이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 203F가 된다.

25. RD INPUT D8205D

RD의 OPCODE(8 bit)는 D8이고 205D를 2진수로 표현하면 0010 0000 0101 1101이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 205D가 된다.

26. COMP ZERO 281030

COMP의 OPCODE(8 bit)는 28이고 1030을 2진수로 표현하면 0001 0000 0011 0000이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1030이 된다.

27. JEQ EXIT 302057

JEQ의 OPCODE(8 bit)는 30이고 2057을 2진수로 표현하면 0010 0000 0101 0111이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 2057이 된다.

28. STCH BUFFER,X 549039

STCH의 OPCOED(8 bit)는 54이고 9039를 2진수로 표현하면 1001 000 0011 1001이므로, 가장 앞자리 x bit가 1인 것을 확인할 수 있다. 따라서 Indexed 모드이다. 따라서 타겟 주소는 1039+(x) 이다. X에는 ZERO가 LOAD돼 있으므로 타겟 주소는 1039가 된다.

29. TIX MAXLEN 2C205E

TIX의 OPCODE(8 bit)는 2C이고 205E를 2진수로 표현하면 0010 0000 0101 1110이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 205E가 된다.

30. JLT RLOOP 38203F

JLT의 OPCODE(8 bit)는 38이고 203F를 2진수로 표현하면 0010 0000 0011 1111이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 203F가 된다.

31. EXIT STX LENGTH 101036

27번째 줄에서 EXIT의 주소가 2057인 것을 알았으니 EXIT은 2057에 위치한다. STX의 OPCODE(8 bit)는 10이고 1036을 2진수로 표현하면 0001 0000 0011 0110 이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1036이 된다. X REGISTER에 있는 VALUE를 LENGTH에 저장한다.

32. RSUB 4C0000

RSUB의 OPCODE(8 bit)는 4C이다.

33. INPUT BYTE X’F1’ F1

25번째 줄에서 INPUT의 주소가 205D인 것을 알았으니 INPUT은 205D에 위치한다. F1이라는 문자를 생성한다.

34. MAXLEN WORD 4096 001000

29번째 줄에서 MAXLEN의 주소가 205E인 것을 알았으니 MAXLEN은 205E에 위치한다. 상수 4096을 생성하고 이것을 16진수로 표현하면 001000(16^3)이 된다.

35. WRREC LDX ZERO 041030

6번째 줄에서 WRREC의 주소가 2061인 것을 알았으니 WRREC는 2061에 위치한다. LDX의 OPCODE(8 bit)는 04이고 1030을 2진수로 표현하면 0001 0000 0011 0000이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1030이 된다.

36. WLOOP TD OUTPUT E02079

37번째 줄에서 WLOOP의 주소가 2064인 것을 알았으니 WLOOP은 2064에 위치한다. TD의 OPCODE(8 bit)는 E0이고 2079를 2진수로 표현하면 0010 0000 0111 1001이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 2079가 된다.

37. JEQ WLOOP 302064

JEQ의 OPCODE(8 bit)는 30이고 2064를 2진수로 표현하면 0010 0000 0110 0100이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 2064가 된다.

38. LDCH BUFFER,X 509039

LDCH의 OPCODE(8 bit)는 50이고 9039를 2진수로 표현하면 1001 0000 0011 1001이므로, 가장 앞자리 x bit가 1인 것을 확인할 수 있다. 따라서 Indexed 모드이다. 따라서 타겟 주소는 1039+(X)가 된다.

39. WD OUTPUT DC2079

WD의 OPCODE(8 bit)는 DC이고 2079를 2진수로 표현하면 0010 0000 0111 1001이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 2079가 된다.

40. TIX LENGTH 2C1036

TIX의 OPCODE(8 bit)는 2C이고 1036의 2진수로 표현하면 0001 0000 0011 0110이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 1036이 된다.

41. JLT WLOOP 382064

JLT의 OPCODE(8 bit)는 38이고 2064를 2진수로 표현하면 0010 0000 0110 0100이므로, 가장 앞자리 x bit가 0인 것을 확인할 수 있다. 따라서 Direct 모드이다. 따라서 타겟 주소는 2064가 된다.

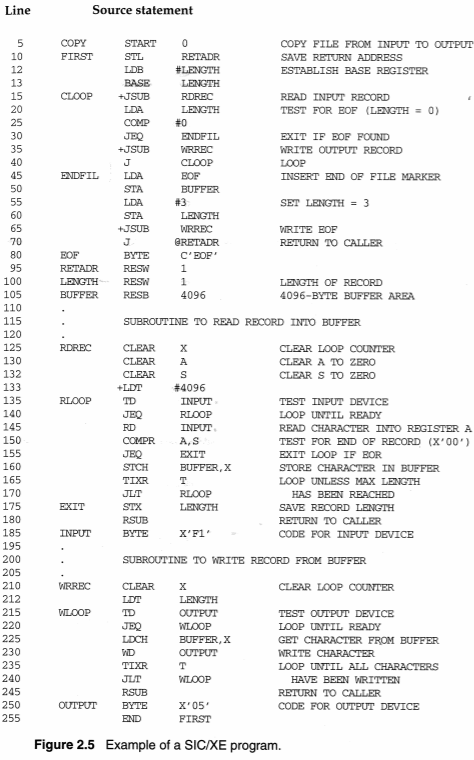
42. RSUB 4C0000

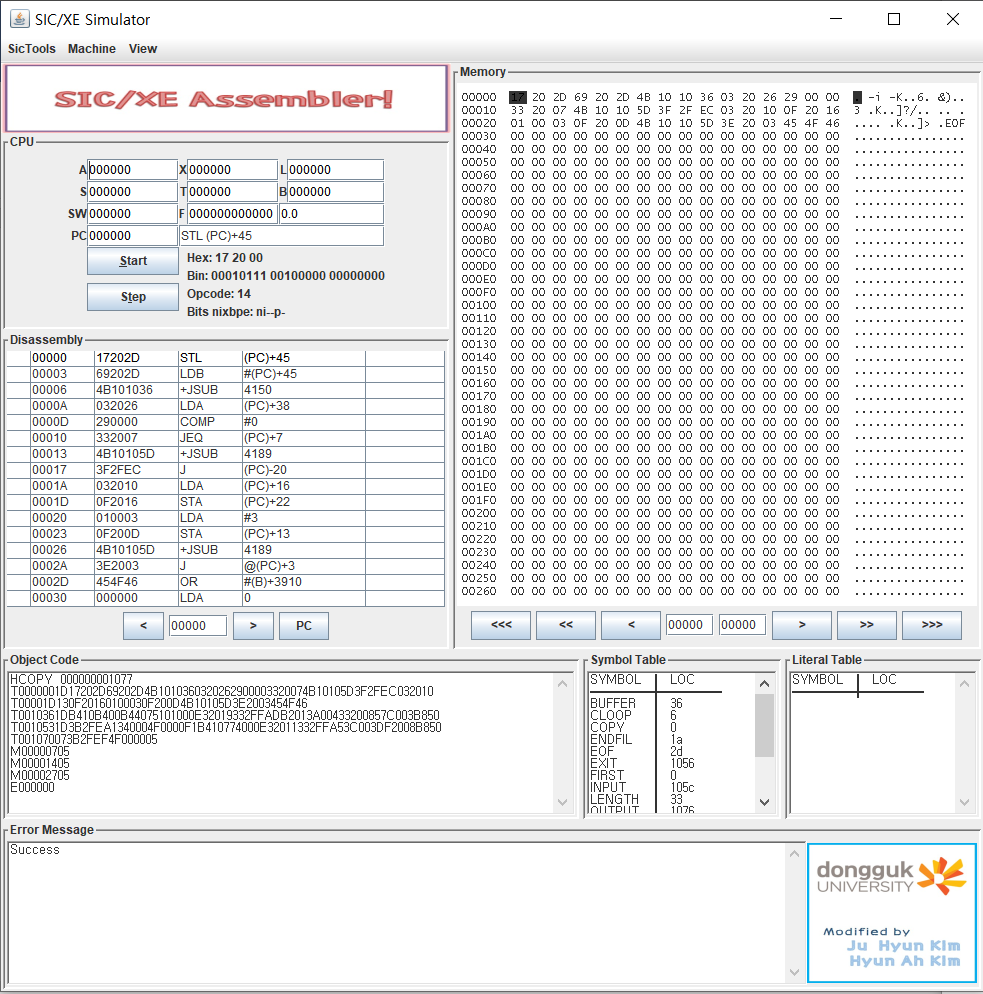
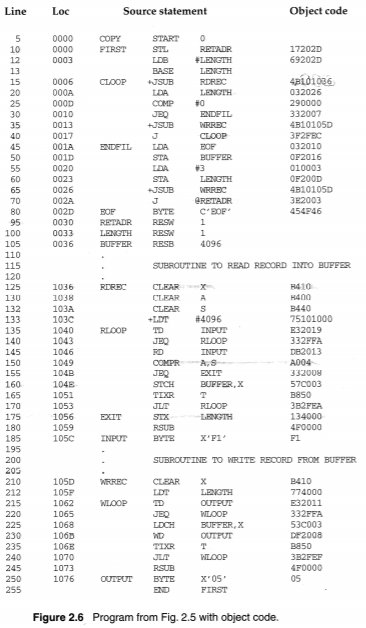
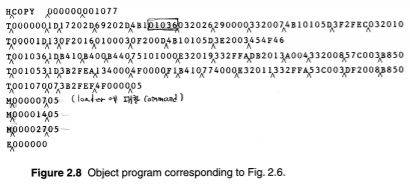
RSUB의 OPCODE(8 bit)는 4C이다.

43. OUTPUT BYTE X’05’ 05

36번째 줄에서 OUTPUT의 주소가 2079인 것을 알았으니 OUTPUT은 2079에 위치한다. 05라는 문자를 생성한다.

**4번. 교재의 2장 Figure 2.5의 코드를 SIC/XE용 어셈블러를 통해 실행시켜 얻어진 .lst 파일 및 .obj파일을 Figure 2.6과 비교하라.**



모든 부분에서 동일하게 나오는 것을 확인할 수 있었다.

**5번. 해당실습을 진행하고 보고서를 작성하는 동안 느꼈던 개인적인 생각과 느낌**

이번 실습을 진행하면서 SIC assembly program 시뮬레이터로 실행하고 코드를 하나씩 해석하면서 어떻게 object code가 나오는지 더 자세하게 알 수 있어서 좋았습니다. 그리고 시뮬레이터에서 SICSIM.EXE를 실행하고 하나씩 진행하면서 레지스터의 값이 변하는 것을 눈으로 확인하면서 명령어가 어떻게 수행되는지도 알게되었다. 또한 opcode (8 bit), x (1 bit), address (15 bit)로 object code를 해석하는 방법과 Addressing modes 에서 index모드인지 direct모드인지를 구분하는 방법도 배울 수 있어서 좋았습니다. 또한 문제 1~3번은 SIC 시뮬레이터로 진행하고 4번문제는 SIC/XE Simulator을 통해 진행하면서 SIC/XE 코드와 SIC 코드의 차이점을 알 수 있었다. SIC/XE가 많은 Addressing 하는데에 많은 모드를 지원해서 불필요한 참조가 많이 사라진 것을 알 수 있었다.

**실습#3**

**1. 교재 P.136~138(Figure 3.8 & 3.9)의 linking 및 relocation 대한 sample 프로그램의 내용에 따라 수동으로 시뮬레이션을 수행하고 그 결과를 분석한다. 또한 a)와 b)의 내용을 포함해 레포트에 반영한다.**

PROGA를 보면 LISTA, ENDA가 EXTDEF로 정의되어 있고, LISTB, ENDB, LISTC, ENDC가 EXTREF로 외부 참조되어 있는 것을 볼 수 있다.

0020라인에 REF1 LDA LISTA 03201D는 OPCODE가 00이고 nixbpe=110010으로 pc-relative이다. Pc는 0023에 위치하고 01D를 더해주면 LISTA가 0040에 위치하는 것을 알 수 있다.

0023라인에 REF2 +LDT LISTB+4 77100004는 OPCODE가 74이고 nixbpe=110001으로 extended mode이고 LISTB는 외부 참조로 주소를 모르기 때문에 00000으로 표현되고 +4이므로 00004로 나타내는 것을 볼 수 있다.

0027라인에 REF3 LDX #ENDA-LISTA 050014는 OPCODE가 04이고 nixbpe=010000으로 immediate addressing이고 ENDA의 주소는 0054이고 LISTA의 주소는 40이므로 둘의 차는 14인 것을 확인할 수 있다. 따라서 주소로 014가 오는 것을 볼 수 있다.

0040라인에 LISTA EQU \*는 LISTA의 주소가 0040인 것을 보여준다.

0054라인에 ENDA EQU \*는 ENDA의 주소가 0054인 것을 보여준다.

0054라인에 REF4 WORD ENDA-LISTA+LISTC 000014는 ENDA-LISTA는 014이고 LISTC의 주소는 모르기 때문에 000으로 표현되어 000014로 표현되는 것을 볼 수 있다.

0057라인에 REF5 WORD ENDC-LISTC-10 FFFFF6은 ENDC와 LISTC 모두 주소를 모르므로 000으로 표현되고 -10이기 때문에 FFFFF6으로 표현되는 것을 볼 수 있다.

005A라인에 REF6 WORD ENDC-LISTC+LISTA-1 000003F는 ENDC와 LISTC는 주소를 모르므로 000으로 표현되고 LISTA는 0040이고 -1을 해주면 00003F로 표현되는 것을 볼 수 있다.

005D라인에 REF7 WORD ENDA-LISTA-(ENDB-LISTB) 000014는 ENDA의 주소는 0054 LISTA의 주소는 0040이고 ENDB와 LISTB의 주소는 모르기 때문에 000으로 표현하여 000014로 표현되는 것을 볼 수 있다.

0060라인에 REF8 WORD LISTB-LISTA FFFFC0은 LISTB의 주소는 모르기 때문에 000으로 표현되고 LISTA의 주소는 040이기 때문에 000-040은 FFFFC0으로 표현되는 것을 볼 수 있다.

또한 PROGA의 길이는 0063인 것을 확인할 수 있다.

PROGB를 보면 LISTB, ENDB는 EXTDEF로 정의되어 있고, LISTA, ENDA, LISTC 그리고 ENDC가 EXTREF로 정의되어 외부 참조되어 있는 것을 볼 수 있다.

0036라인에 REF1 +LDA LISTA 03100000는 OPCODE가 00이고 nixbpe=110001으로 extended mode인 것을 볼 수 있다. 또한 LISTA의 주소는 모르기 때문에 00000으로 표현되는 것을 볼 수 있다.

003A라인에 REF2 LDT LISTB+4 772027는 OPCODE가 74이고 nixbpe=110010으로 pc-relative인 것을 알 수 있다. 현재 pc가 003D에 위치하고 있으므로 003D+027=LISTB(0060)+4인 것을 확인할 수 있다.

0060라인에 LISTB EQU \*는 LISTB가 0060에 위치하는 것을 보여준다.

0070라인에 ENDB EQU \*는 ENDB가 0070에 위치하는 것을 보여준다.

0070라인에 REF4 WORD ENDA-LISTA+LISTC 000000는 ENDA, LISTA, LISTC의 주소를 모두 모르므로 000000으로 표현되는 것을 볼 수 있다.

0073라인에 REF5 WORD ENDC-LISTC-10 FFFFF6은 ENDC와 LISTC의 주소를 모르므로 000-010인 것을 확인할 수 있고 FFFFF6으로 표현되는 것을 볼 수 있다.

0076라인에 REF6 WORD ENDC-LISTC+LISTA-1 FFFFFF는 ENDC, LISTC, LISTA의 주소를 모두 모르기 때문에 000-001인 것을 확인할 수 있고 FFFFFF로 표현되는 것을 볼 수 있다.

0079라인에 REF7 WORD ENDA-LISTA-(ENDB-LISTB) FFFFF0은 ENDA와 LISTA의 주소는 모르기 때문에 000-(ENDB-LISTB)이고 ENDB는 0070이고 LISTB는 0060이므로 -10이므로 FFFFF0으로 표현되는 것을 볼 수 있다.

007C라인에 REF8 WORD LISTB-LISTA 000060은 LISTA의 주소는 모르므로 000이고 LISTB의 주소는 060이므로 000060으로 표현되는 것을 볼 수 있다.

또한 PROGB의 길이가 7F인 것을 볼 수 있다.

PROGC를 보면 LISTC, ENDC가 EXTDEF로 정의되어 있고, LISTA, ENDA, LISTB 그리고 ENDB는 EXTREF로 정의되어 외부 참조된 것을 확인할 수 있다.

0018라인에 REF1 +LDA LISTA 03100000을 보면 OPCODE가 00이고 nixbpe=110001으로 extended mode이다. LISTA의 주소는 모르기 때문에 000으로 표현되어 00000으로 표현되는 것을 볼 수 있다.

001C라인에 REF2 +LDT LISTB+4 77100004는 OPCODE는 74이고 nixbpe=110001으로 extended mode이다. LISTB의 주소는 모르기 때문에 000으로 표현되고 LISTB+4이므로 00004로 표현되는 것을 볼 수 있다.

0020라인에 REF3 +LDX #ENDA-LISTA 05100000은 OPCODE는 04이고 nixbpe=010001으로 extended mode + immediate addressing인 것을 확인할 수 있다. ENDA와 LISTA의 주소를 모르기 때문에 00000으로 표현되는 것을 볼 수 있다.

0030라인에 LISTC EQU \*는 LISTC의 주소가 0030인 것을 보여준다.

0042라인에 ENDC EQU \*는 ENDC의 주소가 0042인 것을 보여준다.

0042라인에 REF4 WORD ENDA-LISTA+LISTC 000030은 ENDA, LISTA의 주소는 모르기 때문에 000으로 표현되고 LISTC의 주소는 030이므로 000030으로 표현되는 것을 볼 수 있다.

0045라인에 REF5 WORD ENDC-LISTC-10 000008은 ENDC는 0042 LISTC는 0030이므로 둘의 차는0012이고 -10을 해주면 000008이 되는 것을 볼 수 있다.

0048라인에 REF6 WORD ENDC-LISTC+LISTA-1 000011은 ENDC-LISTC는 0012이고 LISTA의 주소는 모르기 때문에 000으로 표현하고 0012에서 -1을 해서 000011으로 표현되는 것을 볼 수 있다.

004B라인에 REF7 WORD ENDA-LISTA-(ENDB-LISTB) 000000은 ENDA, LISTA, ENDB, 그리고 LISTB의 주소를 모두 모르기 때문에 000000으로 표현되는 것을 볼 수 있다.

004E라인에 REF8 WORD LISTB-LISTA 000000은 LISTB, LISTA의 주소를 모르기 때문에 000000으로 표현되는 것을 볼 수 있다.

또한 PROGC의 길이가 51인 것을 확인할 수 있다.

**a) 첨부파일 (Arithmetic on operands P136-138)의 내용을 분석하여 서술식으로 자세히 설명하여라.**

H: PROGA | 000000 | 000063

프로그램이름이 PROGA이고 시작주소가 000000이고 프로그램의 길이가 0000063인 것을 알려준다.

D: LISTA | 000040 | ENDA | 000054

EXTDEF

000040에 LISTA가 위치하는 것을 알려준다.

000054에 ENDA가 위치하는 것을 알려준다.

R: LISTB | ENDB | LISTC | ENDC

EXTREF

LISTB, ENDB, LISTC, ENDC가 외부 참조된 것을 알려준다.

T: 000020 | 0A | 03201D | 77100004 | 050014 |

시작주소 REF1 REF2 REF3

T: 000054 | 0F | 000014 | FFFFF6 | 00003F | 000014 | FFFFC0

시작주소 REF4 REF5 REF6 REF7 REF8

M: 000024 | 05 | + | LISTB |

REF2 LISTB의 주소를 더해준다.

M: 000054 | 06 | + | LISTC |

REF4 LISTC의 주소를 더해준다.

M: 000057 | 06 | + | ENDC |

REF5 ENDC의 주소를 더해준다.

000057 | 06 | - | LISTC |

REF5 LISTC의 주소를 빼 준다.

M: 00005A | 06 | + | ENDC |

REF6 ENDC의 주소를 더해준다.

00005A | 06 | - | LISTC |

REF6 LISTC의 주소를 빼 준다.

00005A | 06 | + | PROGA |

REF6 LISTA-1은 (40-1) + PROGA = 3F + PROGA이므로 PROGA의 주소를 더해준다.

M: 00005D | 06 | - | ENDB |

REF7 ENDB의 주소를 빼 준다.

00005D | 06 | + | LISTB |

REF7 LISTB의 주소를 더해준다.

M: 000060 | 06 | + | LISTB |

REF8 LISTB의 주소를 더해준다.

000060 | 06 | - | PROGA |

REF8 LISTA는 40 + PROGA이므로 PROGA의 주소를 빼 준다.

H: PROGB | 000000 | 00007F |

프로그램 이름이 PROGB이고 시작주소가 000000이고 프로그램의 길이가 00007F이다.

D: LISTB | 000060 | ENDB | 000070 |

EXTDEF

000060에 LISTB가 위치하는 것을 알려준다.

000070에 ENDB가 위치하는 것을 알려준다.

R: LISTA | ENDA | LISTC | ENDC |

EXTREFE

LISTA, ENDA, LISTC, ENDC가 외부 참조되었음을 알려준다.

T: 000036 | 0B | 03100000 | 772027 | 05100000 |

시작주소 REF1 REF2 REF3

T: 000070 | 0F | 000000 | FFFFF6 | FFFFFF | FFFFF0 | 000060 |

시작주소 REF4 REF5 REF6 REF7 REF8

M: 000037 | 05 | + | LISTA |

REF1 LISTA의 주소를 더해준다.

M: 00003E | 05 | + | ENDA |

REF3 ENDA의 주소를 더해준다.

00003E | 05 | - | LISTA |

REF3 LISTA의 주소를 빼 준다.

M: 000070 | 06 | + | ENDA |

REF4 ENDA의 주소를 더해준다.

000070 | 06 | - | LISTA |

REF4 LISTA의 주소를 빼 준다.

000070 | 06 | + | LISTC |

REF4 LISTC의 주소를 더해준다.

M: 000073 | 06 | + | ENDC |

REF5 ENDC의 주소를 더해준다.

000073 | 06 | - | LISTC |

REF5 LISTC의 주소를 빼 준다.

M: 000076 | 06 | + | ENDC |

REF6 ENDC의 주소를 더해준다.

000076 | 06 | - | LISTC |

REF6 LISTC의 주소를 빼 준다.

000076 | 06 | + | LISTA |

REF6 LISTA의 주소를 더해준다.

M: 000079 | 06 | + | ENDA |

REF7 ENDA의 주소를 더해준다.

000079 | 06 | - | LISTA |

REF7 LISTA의 주소를 빼 준다.

M: 00007C | 06 | + | PROGB |

REF8 LISTB는 0060 + PROGB이기 때문에 PROGB의 주소를 더해준다.

00007C | 06 | - | LISTA |

REF8 LISTA의 주소를 빼 준다.

H: PROGC | 000000 | 000051 |

프로그램 이름이 PROGC이고 시작주소가 000000이고 프로그램의 길이가 000051이다.

D: LISTC | 000030 | ENDC | 000042

EXTDEF

000030에 LISTC가 위치하는 것을 알려준다.

000042에 ENDC가 위치하는 것을 알려준다.

R: LISTA | ENDA | LISTB | ENDB |

EXTREF

LISTA, ENDA, LISTB, ENDB가 외부 참조되었음을 알려준다.

T: 000018 | 0C | 03100000 | 77100004 | 05100000 |

시작주소 REF1 REF2 REF3

T: 000042 | 0F | 000030 | 000008 | 000011 | 000000 | 000000

시작주소 REF4 REF5 REF6 REF7 REF8

M: 000019 | 05 | + | LISTA |

REF1 LISTA의 주소를 더해준다.

M: 00001D | 05 | + | LISTB |

REF2 LISTB의 주소를 더해준다.

M: 000021 | 05 | + | ENDA |

REF3 ENDA의 주소를 더해준다.

000021 | 05 | - | LISTA |

REF3 LISTA의 주소를 빼 준다.

M: 000042 | 06 | + | ENDA |

REF4 ENDA의 주소를 더해준다.

000042 | 06 | - | LISTA |

REF4 LISTA의 주소를 빼 준다.

000042 | 06 | + | PROGC |

REF4 LISTC는 0030 + PROGC의 주소 이므로 PROGC의 주소를 더해준다.

M: 000048 | 06 | + | LISTA |

REF6 LISTA의 주소를 더해준다.

M: 00004B | 06 | + | ENDA |

REF7 ENDA의 주소를 더해준다.

00004B | 06 | - | LISTA |

REF7 LISTA의 주소를 빼 준다.

00004B | 06 | - | ENDB |

REF7 ENDB의 주소를 빼 준다.

00004B | 06 | + | LISTB |

REF7 LISTB의 주소를 더해준다.

M: 00004E | 06 | + | LISTB |

REF8 LISTB의 주소를 더해준다.

00004E | 06 | - | LISTA |

REF8 LISTA의 주소를 빼 준다.

**b) 첨부파일 (Loadmap P136-138)의 내용을 분석하여 서술식으로 자세히 설명하여라.**

LOADER MAP과 memory를 보면 프로그램 PROGA의 REF1의 위치는 0020이고 PROGA의 위치가 0100이므로 memory에 0120에 PROGA의 REF1 03201D가 들어간다. 또한 LISTA의 위치가 0040 이므로 0140에 위치하게 되고 ENDA의 위치가 0054이므로 0154에 위치하게 된다. 그리고 PROGA의 길이가 63이므로 PROGB의 위치는 0163이된다. 그리고 PROGB의 REF1의 위치가 0036이므로 REF1 0310000이 0199에 위치하게 된다. 또한 LISTB의 위치가 60이므로 LISTB는 01C3에 위치하게 되고 ENDB의 위치가 0070이므로 01D3에 위치하게 된다. PROGB의 길이가 7F이므로 PROGC의 위치는 01E2가 된다. PROGC의 REF1의 위치가 0018이므로 REF1은 01FA에 위치하게 되고 LISTC의 위치가 0030이므로 LISTC는 0212에 위치하게 되고, ENDC의 위치가 0042이므로 ENDC는 0224에 위치하게 된다.

Records as provided to the loader을 보면

PROGA에서 M | 000024 | 05 | + | LISTB |를 볼 수 있다.

여기서 LISTB의 주소가 01C3인 것과 PROGA의 주소가 0100이므로 0124에 있는 길이 5의 00004의 주소에 01C3이 더해져 001C7이 되는 것을 볼 수 있다.

M | 000054 | 06 | + | LISTC |를 보면

LISTC의 주소가 0212이고 PROGA의 주소가 0100이므로 0154에 있는 길이 6의 000014에 0212가 더해져 000226이 되는 것을 볼 수 있다.

M | 000057 | 06 | + | ENDC |

M | 000057 | 06 | - | LISTC |를 보면

LISTC의 주소가 0212이고 ENDC의 주소가 0224이다. PROGA의 주소가 0100이므로 0157에 있는 길이 6의 FFFFF6에 0224를 더하고 0212를 뺀다. 결국 0012를 더해주는 것이다. 즉 000008이 되는 것을 볼 수 있다.

M | 00005A | 06 | + | ENDC |

M | 00005A | 06 | - | LISTC |

M | 00005A | 06 | + | PROGA |을 보면

PROGA의 주소가 0100이므로 015A에 있는 길이 6 00003F에 ENDC를 더해주고 LISTC를 빼고 PROGA의 주소를 더해주면 된다. 즉 0112를 더해주면 된다. 000151이 되는 것을 확인할 수 있다.

M | 00005D | 06 | - | ENDB |

M | 00005D | 06 | + | LISTB |을 보면

PROGA의 주소가 0100이고 ENDB의 주소가 01D3이고 LISTB의 주소가 01C3이므로 015D에 있는 길이 6 000014에 ENDB의 주소를 빼고 LISTB의 주소를 더해주면 된다. 즉 000014에 0010을 빼면 된다. 즉 000004가 되는 것을 볼 수 있다.

M | 000060 | 06 | + | LISTB |

M | 000060 | 06 | - | PROGA |을 보면

PROGA의 주소가 0100이고 LISTB의 주소가 01C3이므로 0160에 있는 길이 6 FFFFC0에 01C3을 더하고 0100을 빼면 된다. 즉 FFFFC0에 00C3을 더하면 된다. 즉 000083이 되는 것을 볼 수 있다.

PROGB에서 M | 000037 | 05 | + | LISTA |을 볼 수 있다.

PROGB의 주소가 0163이고 LISTA의 주소가 0140이므로 019A에 있는 길이 5 00000에 0140을 더해주면 된다. 즉 00140이 되는 것을 볼 수 있다.

M | 00003E | 05 | + | ENDA |

M | 00003E | 05 | - | LISTA |을 보면

PROGB의 주소가 0163이고 LISTA의 주소가 0140이고, ENDA의 주소가 0154이므로 01A1에 있는 길이 5 00000에 ENDA의 주소를 더하고 LISTA의 주소를 빼면 된다. 즉 0000에 0014를 더해주면 되므로 00014가 되는 것을 볼 수 있다.

M | 000070 | 06 | + | ENDA |

M | 000070 | 06 | - | LISTA |

M | 000070 | 06 | + | LISTC |를 보면

ENDA의 주소가 0154이고 LISTA의 주소가 0140이고 LISTC의 주소가 0212이므로 PROGB의 주소에 0070을 더한 주소 01D3에 길이 6의 000000에 ENDA의 주소를 더하고 LISTA의 주소를 빼고 LISTC의 주소를 더해주면 된다. 000000 + 0154 – 0140 + 0212이므로 000226이 되는 것을 볼 수 있다.

M | 000073 | 06 | + | ENDC |

M | 000073 | 06 | - | LISTC |을 보면

ENDC의 주소가 0224이고 LISTC의 주소가 0212이다. PROGB의 주소가 0163이므로 0073을 더해준 01D6에 있는 길이 6의 FFFFF6에 0224를 더해주고 0212를 빼면 된다. 즉 FFFFF6 +0012이므로 000008인 것을 알 수 있다.

M | 000076 | 06 | + | ENDC |

M | 000076 | 06 | - | LISTC |

M | 000076 | 06 | + | LISTA |을 보면

ENDC의 주소가 0224이고 LISTC의 주소가 0212이고 LISTA의 주소가 0140이다. PROGB의 주소가 0163이므로 0163에 000076을 더한 00001D9에 있는 길이 6의 FFFFFF에 ENDC의 주소를 더해주고 LISTC의 주소를 빼고 LISTA의 주소를 더해주면 된다. 즉, FFFFFF + 000224 – 000212 + 000140이다. 따라서 000151이 되는 것을 볼 수 있다.

M | 000079 | 06 | + | ENDA |

M | 000079 | 06 | - | LISTA |을 보면

ENDA의 주소가 0154이고 LISTA의 주소가 0140이다. PROGB의 주소가 0163이므로 0163에 000079를 더한 0001DC에 있는 길이 6의 FFFFF0에 ENDA의 주소를 더해주고 LISTA의 주소를 빼면 된다. 즉, FFFFF0 + 000154 - 000140이므로 000004가 되는 것을 볼 수 있다.

M | 00007C | 06 | + | PROGB |

M | 00007C | 06 | - | LISTA |을 보면

PROGB의 주소가 0163이고 LISTA의 주소가 0140이다. PROGB의 주소가 0163이므로 0163에 00007C를 더한 0001DF에 있는 길이 6의 000060에 PROGB의 주소를 더해주고 LISTA의 주소를 빼면 된다. 즉, 000060 + 000163 – 000140이므로 000083이 되는 것을 볼 수 있다.

PROGC에서 M | 000019 | 05 | + | LISTA |을 볼 수 있다.

PROGC의 주소가 01E2이고 LISTA의 주소는 0140이다. PROGC의 주소에 000019를 더해준 0001FB에 있는 길이 5의 00000에 LISTA의 주소를 더해주면 된다. 즉, 00000 + 00140이므로 00140이 되는 것을 볼 수 있다.

M | 00001D | 05 | + | LISTB |을 보면

PROGC의 주소가 01E2이고 LISTB의 주소가 01C3이다. PROGC의 주소에 00001D를 더해준 0001FF에 있는 길이 5의 00004에 LISTB의 주소를 더해주면 된다. 즉, 00004 + 001C3이므로 001C7이 되는 것을 볼 수 있다.

M | 000021 | 05 | + | ENDA |

M | 000021 | 05 | - | LISTA |을 보면

PROGC의 주소가 01E2이고 ENDA의 주소가 0154이고 LISTA의 주소가 0140이다. PROGC의 주소에 000021을 더한 000203에 있는 길이 5의 00000에 ENDA의 주소를 더해주고 LISTA의 주소를 빼면 된다. 즉, 00000 + 00154 - 00140이므로 00014가 되는 것을 볼 수 있다.

M | 000042 | 06 | + | ENDA |

M | 000042 | 06 | - | LISTA |

M | 000042 | 06 | + | PROGC |을 보면

PROGC의 주소가 01E2이고 ENDA의 주소가 0154이고 LISTA의 주소가 0140이다. PROGC의 주소에 000042를 더한 000224에 있는 길이 6의 000030에 ENDA의 주소를 더해주고 LISTA의 주소를 빼고 PROGC의 주소를 더해주면 된다. 즉, 000030 + 000154 – 000140 + 0001E2이므로 000226이 되는 것을 볼 수 있다.

M | 000048 | 06 | + | LISTA |을 보면

PROGC의 주소가 01E2이고 LISTA의 주소가 0140이다. PROGC의 주소에 000048을 더한 00022A에 있는 길이 6의 000011에 LISTA의 주소를 더해주면 된다. 즉, 000011 + 000140이므로 000151이 되는 것을 볼 수 있다.

M | 00004B | 06 | + | ENDA |

M | 00004B | 06 | - | LISTA |

M | 00004B | 06 | - | ENDB |

M | 00004B | 06 | + | LISTB |을 보면

PROGC의 주소가 01E2이고 ENDA의 주소가 0154이고 LISTA의 주소가 0140이고 ENDB의 주소가 01D3이고 LISTB의 주소가 01C3이다. PROGC의 주소에 00004B를 더해준 00022D에 있는 길이 6의 000000에 ENDA의 주소를 더해주고 LISTA의 주소를 빼고 ENDB의 주소를 빼고 LISTB의 주소를 더해주면 된다. 즉, 000000 + 000154 – 000140 – 0001D3 + 0001C3이므로 000004가 되는 것을 볼 수 있다.

M | 00004E | 06 | + | LISTB |

M | 00004E | 06 | - | LISTA |을 보면

PROGC의 주소가 01E2이고 LISTB의 주소가 01C3이고 LISTA의 주소가 0140이다. PROGC의 주소에 00004E를 더한 000230에 있는 길이 6의 000000에 LISTB의 주소를 더해주고 LISTA의 주소를 뺀다. 즉, 000000 + 0001C3 – 000140이므로 000083이 되는 것을 볼 수 있다.

위에서 수정된 주소가 MODIFICATION을 통해 MEMORY에 적용되면서 주소가 바뀌는 것을 Result of Applying Computations를 통해 확인할 수 있다.

**2. 해당실습을 진행하고 보고서를 작성하는 동안 느꼈던 개인적인 생각과 느낌을 에세이 형식으로 정리하여 추가하여라.**

이번 실습을 진행하면서 프로그램이 실행되면서 EXTDEF가 외부의 주소를 정의하는 것과 EXTREF를 통해서 외부의 주소를 참조하는 것이 어떻게 실행되고 LOADER을 통해 어떻게 MODIFICATION이 되어 어떻게 RELOCATION되는지를 볼 수 있었다. 그리고 외부 참조가 되는 주소들이 초기에 어떤 식으로 주소가 할당되는지도 알 수 있었다. 각 프로그램의 주소와 길이에 따라서 외부 참조된 변수의 주소를 구하고 정의된 외부의 주소를 사용해서 RELOCATION을 하면서 MODIFICATION된 주소가 OBJECT CODE에 어떤 식으로 주소가 입력되는지를 확인할 수 있었다.