시스템프로그래밍

Homework #2 (3)

Implement a Sorting Program using SIC / SIC/XE

< Merge Sort >

이름 : 정유진

학번 : 201120895

제출일 : 2016년 11월 23일

Merge Sort

1. Pseudocode

```
mergesort(data, first, last)
{
    if (first < last)
    {
         mid = (first + last) / 2;
         mergesort(data, first, mid);
         mergesort(data, mid+1, last);

         merge(data, first, last);
    }
```

2. 구현 코드

```
MERSOT      START       0

.CHOice MaSsaGe
..수행하고자 하는 Sorting Algorithm 을 선택한다.
CHOMSG      LDA         #95
            WD          OUTDEV
            TIX         #55
            JLT         CHOMSG
            LDA         #10
            WD          OUTDEV
            WD          OUTDEV
            WD          OUTDEV
            CLEAR       X
            LDA         #CHOLEN     .CHOLEN 에 TXT 길이를 저장한다.
            SUB         #CHOTXT
            STA         CHOLEN
CHOPRT      LDA         CHOTXT, X
            WD          OUTDEV
            TIX         CHOLEN
            JLT         CHOPRT
            LDA         #CHOSOT
            STA         TMPIND
            LDT         #8          .CHOSOT 밑으로 몇칸씩 내려가야하는지
            CLEAR       A
CHOICE      TD          #0
            JEQ         CHOICE
            RD          #0
            STA         TMPNUM
            .1, 2 만 입력을 받음.
            COMP        #50         .2 까지
            JGT         CHOICE
            COMP        #49
            JLT         CHOICE
            .1 이면 CHOSOT 으로 이동한다.
            JEQ         @TMPIND
            COMP        #50
            JEQ         EXIT

CHOSOT      LDA         #10
            WD          OUTDEV
            JSUB        MEGMSG
```

```
                CLEAR   X
                LDA     #10
                WD      OUTDEV
                WD      OUTDEV
                JSUB    RESMSG
                J       CHOMSG

.....END.....
                END     MERSOT
EXIT            J       EXITTW
EXITTW          J       EXIT
.....
```

```
.....Input Processing.....
.Input Msg 출력
INSAMP  CLEAR   X
        LDT     #5
INPMSG  LDA     #10
        WD      OUTDEV
        TIX     #2
        JLT     INPMSG
        CLEAR   X
        LDA     #IMSLEN
        SUB     #IMSTXT
        STA     IMSLEN
IMSPRT  LDA     IMSTXT, X
        WD      OUTDEV
        TIX     IMSLEN
        JLT     IMSPRT
        CLEAR   X
.#STAADD ~ #FIGURE CLEAR
        LDA     #STAADD
        STA     IMSLEN
CLENUM  CLEAR   A
        STA     @IMSLEN
        LDA     #3
        ADD     IMSLEN
        STA     IMSLEN
        COMP    #STRLEN
        JLT     CLENUM
        CLEAR   A       .Input 제대로 들어가게 하려고 clear 안하면 junk 처리가 된다.

.INput LOOP
..Input 을 받아옵니다.
INLOOP  TD      #0
        JEQ     INLOOP
        RD      #0
        COMP    #69     .'E' EOF 체크를 위해서 비교
        JEQ     EOFCHK
        COMP    #32     .공백을 나타내는 아스키값
        JEQ     ENDFIL
        COMP    #0
        JEQ     INLOOP
        COMP    #48     .'0' 보다 작으면 JUNK 처리
        JLT     JUNK
        COMP    #57     .'9' 보다 크면 JUNK 처리
        JGT     JUNK
        SUB     #48     .아스키 값을 10 진수로 변경
        CLEAR   S
        ADDR    A, S    .S 에 A 값(읽어온 값)을 옮겨놓음
        CLEAR   A       .A 초기화

.STRTMP
..A 에 TMPNUM 예전꺼 집어넣고 10 곱해줘서 자릿수 맞추어줍니다.
...6 자리 넘어갈 경우 JUNK 처리하고 버립니다.
....그리고 S 에 있던 이번에 읽어온 값을 1 의 자리에 넣어주는 형식
STRTMP  LDA     TMPNUM
        MUL     #10
        ADDR    S, A
        TIXR    T       .6 자리 이상이면 JUNK 처리
        JGT     INLOOP
        STA     TMPNUM
        CLEAR   A
        STA     JUKCHK
        J       INLOOP
```

.JUNK
..JUNK 는 Junk Check 변수에 값을 1 로 해주고 junk 임을 표시해 준다.
...그리고 해당하는 곳의 값을 다 제거해줘야되니까 띄어쓰기가 나올 때까지 Loop 를 돌린다.

```
JUNK        LDA         #1
            STA         JUKCHK
            J           INLOOP
```

.ENDFIL
..공백을 만나게 되면 ENDFIL 로 이동된다.
...Junk 일 경우 Clear Ready 로 이동하여 TempNum 을 비워준다.
....1 줄씩 옮겨가려면 3byte 를 이동해야해서 3 에 input 숫자를 곱해준다.(0 부터 시작; 0 = 1 개, 1 = 2 개)
.....STArt ADDress 에 첫번째 변수의 주소를 immediate addressing 을 통해 더해준다.

```
.ENDFIL     CLEAR       A
ENDFIL      LDA         JUKCHK      .Junk 인지 check
            COMP        #1
            JEQ         CLERDY      .Junk 면 다 폐기
            CLEAR       A
            LDA         #3          .시작주소 계산을 위해서
            MUL         INPNUM      .INPut NUMber 는 0 부터 시작하며 Input 의 개수를 나타낸다.
            STA         STAADD
            CLEAR       X
```

.STRSAM
..SToRe SAMple 은 Temp Number 에 임시로 저장해 두었던 값을 배열?에 차례대로 정리해줍니다.
...끝나고 INPut NUMber 를 1 증가 시켜준다.
....최대로 받을 수 있는 숫자의 개수는 15 개로 한정하고 15 개가 넘어가면 값을 받아오는 것을 종료한다.
.....그렇지 않으면 CLEar ReaDY 로 이동해서 Temp Number 를 초기화한다.

```
STRSAM      LDA         TMPNUM
            ..그냥 Index 로 해도 될 듯
            ...처음에는 Indirect 로 하려고 했으나 그렇게까지 할 필요가..?
            LDX         STAADD
            STA         STR1, X
            LDA         INPNUM
            ADD         #1          .Input Number 1 증가
            STA         INPNUM
            .COMP       #15         .Sample 의 숫자가 15 개가 넘어가면 종료한다.
            JGT         ENDINP
            CLEAR       S
            J           CLERDY
```

.EOFCHK
..EOF CHecK 는 byte 단위로 READ 를 진행 중 'E'가 발견되었을 시에 이동되어 EOF 를 체크한다.

...EOF 가 아닐 경우 공백을 만날 때까지 빼주어야하는 데 그건 좀 생각해보자.

```
EOFCHK      RD          #0
            COMP        #79
            JEQ         EOFCHK
            COMP        #70
            CLEAR       A
            CLEAR       X
            JEQ         ENDINP
```

.CLERDY
..CLEar ReaDY 는 Temp Number 를 초기화하기 전 준비 단계이다.

```
CLERDY      CLEAR       A
            CLEAR       X
            CLEAR       S
```

.CLETMP
..CLEar TeMP 는 Temp Number 를 초기화 해서 재사용하게 만든다.

```
CLETMP      STA         TMPNUM, X
            TIX         #3
            JLT         CLETMP
            CLEAR       X
            JEQ         INLOOP
```

```
.END INPut
..END INPut 는 현재 숫자 받아오는 것이 끝난 상태이며 임시로 저장해놓은 이름이고 바뀔 수도 있다.
...RSUB 추가 예정 ver1.1
...RSUB 추가 ver1.3
...Print 로 이름 바꿀 수도 ver1.4
ENDINP      LDA         #100
            STA         FIGURE
            LDA         #0
            STA         TMPNUM
            STA         FIGURE      .FIGURE, TMPNUM 초기화
            LDA         #100        .자릿수 조절을 위해 FIGURE 에 100 추가 최대 3 자리.
            STA         FIGURE
            LDA         #STR1
            STA         NUMADD      .STR1 주소 불러와서 NUMADD 에 저장
            ..X 는 input 개수 chk 해야되니까 indirect 로
            LDA         #3
            MULR        X, A        .X 는 몇번째 Sample 인지 나타내고 3 을 곱해주어서 해당 주소로 이동하게 한다.
            ADD         NUMADD
            STA         NUMADD
            LDA         @NUMADD     .해당 되는 주소의 값을 불러온다.
            STA         TMPNUM      .TMPNUM 에 해당 값을 저장한 뒤 S reg 를 초기화한다.
            CLEAR       S
            LDA         #32         .띄어쓰기를 한번 한다. ver1.4 이거 수정할 수도 첫번째에 띄어쓰기 되어서;
            WD          OUTDEV      .쓰기
.           LDA         NUMADD
.           COMP        PIVIND
.           JEQ         PIVMAK
            J           CALFIG
..pivot 을 표시하기 위해 사용되어진다.
.PIVMAK     LDA         #124
.           WD          OUTDEV
.           LDA         #32
.           WD          OUTDEV
.           LDA         #0
.           STA         INPLEN
.CALFIG
..CALculate FIGure 는 자릿수를 계산해서 저장한 뒤 출력해준다.
CALFIG      CLEAR       T
            LDA         TMPNUM      .TMPNUM 을 불러와서 현재 자릿수(Figure)로 나누어준다.
            DIV         FIGURE
            ADDR        A, T        .T reg 에 A reg 값을 할당한다.(현재 A reg 값은 해당 자릿수의 값이다)
            LDA         INPLEN
            COMP        #0
            JGT         PRTNUM
            CLEAR       A
            ADDR        T, A
            COMP        #0          .만약 0 이면 해당 자릿수에 값이 없는 것이니 JZERO 로 넘어간다.
            JEQ         JZERO
            STA         INPLEN
PRTNUM      CLEAR       S
            CLEAR       A
            ADDR        T, A
            MUL         FIGURE      .해당 자릿수의 값을 다시 곱해주어 200, 10, 40 이런 식으로 나타나게 한다.
            ADDR        A, S        .해당 값을 S reg 에 할당한 뒤 TMPNUM 값에서 빼주어 해당 자릿수를 없앤다.
            LDA         TMPNUM
            SUBR        S, A
            STA         TMPNUM
            CLEAR       A           .A reg 를 초기화 한뒤 T reg 에 할당해 놓은 자릿수의 계수 값을 받아온다.
            ADDR        T, A
            ADD         #48         .48 을 더해주어 ASCII 값으로 표현해 출력한다.
            WD          OUTDEV
.JZERO
..Jump Zero? 는.. 자릿수가 없으면 점프해 오는 곳이고 본 목적은 자릿수를 한자리 감소시키는 역할을 한다.
..감소 시킨 뒤 자릿수가 남았으면 CALFIG 로 Jump 하고
..안남았으면 X 를 1 증가 시키고 Input Number 보다 작으면 ENDINP 로 Jump 하고 아니면 루틴을 빠져나온다.
```

```
JZERO   LDA     FIGURE
        DIV     #10
        STA     FIGURE
        COMP    #0
        JGT     CALFIG
        TIX     INPNUM
        CLEAR   A
        STA     INPLEN
        JLT     ENDINP
        RSUB

.....Merge Sort Ready.....

.MErGe MaSsGe
..Merge Sort 문자열을 출력해준다.
MEGMSG  CLEAR   A
        ADDR    L, A
        STA     RETADD
        LDA     INPNUM
        ADD     #1
        STA     INPNUM
        LDA     #10
        WD      OUTDEV
        CLEAR   X
        JSUB    INSAMP
        LDL     RETADD
        LDA     #10
        WD      OUTDEV
        WD      OUTDEV
        CLEAR   X
        LDA     #MEGLEN
        SUB     #MEGTXT
        STA     MEGLEN
MEGPRT  LDA     MEGTXT, X
        WD      OUTDEV
        TIX     MEGLEN
        JLT     MEGPRT

.....Merge Sort Processing.....

MEGRDY  CLEAR   A
        CLEAR   X
        LDA     INPNUM
        STA     MEGNUM  .처음에 인풋 전체 길이를 받아온다.
        LDA     #STR1
        STA     STAADD  .Input 이 저장되어 있는 배열 첫번째 값의 주소를 STAADD 에 저장한다.
        LDA     #3
        MUL     INPNUM
        ADD     STAADD
        STA     ENDADD  .End Address 를 구해 저장한다.
        .MErGe Index
        ..Stack 형식으로 재귀함수를 돌리기 위해 재귀로 들어갈 때마다 해당 함수의 Return Address/Start Address/Length 를 저장한다.
        ..밑에서 이루어지는 과정은 MEGIND 에 다음 주소값인 첫번째 Stack 의 주소값을 저장하는 과정이다.
        LDA     #MEGIND
        ADD     #3
        STA     MEGIND

MEGINI  LDA     STAADD
        SUB     #48     .16 칸 뒤
        STA     MSTIND
        .MSTIND 는 Merge String Index 를 나타낸다.
        ..해당 공간은  Merge Sorting 를 위해 임시로 값들이 저장되어지는 공간이다.
```

```
          .밑의 프로세스는 Input 이 저장되어져있는 공간에서 임시로 저장하는 공간인 MEGST1~에 저장하는 과정이다.
          LDA       @STAADD
          STA       @MSTIND
          LDA       #3
          ADD       STAADD
          STA       STAADD
          COMP      ENDADD
          JLT       MEGINI
          .복사가 끝나면 다시 초기화를 해준다.
          LDA       #STR1
          STA       STAADD
          LDA       #MEGST1
          STA       MSTIND

MEGCHK    LDA       MEGNUM      .덩어리의 크기가 1 이면 Index 를 뒤로 물리고 return 한다.
          COMP      #1
          JGT       MERGE       .CALMEG 로 이동 (3 보다 작으면 1,2 개 있다는 거니까 바로 계산해서 출력가능)
          .해당 LOOP 의 정보를 스택에 저장
          LDA       MEGIND
          SUB       #9
          STA       MEGIND
          RSUB

MERGE     CLEAR     A           .466
          CLEAR     T
          .ADDR     L, T
          .SHIFTL   T, 12
          .LDA      STAADD
          .ADDR     T, A
          .STA      @MEGIND
          .LDA      #3
          .ADD      MEGIND
          .STA      MEGIND
          .LDA      MEGNUM
          .STA      @MEGIND
          .LDA      #3
          .ADD      MEGIND
          .STA      MEGIND
          ..밑의 프로세스는 Stack 에 해당 LOOP 의 Return Address/Start Address/Length 를 저장하는 과정이다.
          ADDR      L, A
          STA       @MEGIND     .Return Address
          LDA       #3
          ADD       MEGIND
          STA       MEGIND
          LDA       STAADD
          STA       @MEGIND     .Start Address
          LDA       #3
          ADD       MEGIND
          STA       MEGIND
          LDA       MEGNUM
          STA       @MEGIND     .Length
          LDA       #3
          ADD       MEGIND
          STA       MEGIND
          .LDA      #12
          .ADD      #MEGIND
          .COMP     MEGIND
          .JLT      ONEDIV
          .CLEAR    T
          .LDA      #RESMSG
          ..SHIFTL  A, 12
          .STA      @MEGIND
          .LDA      #9
          .ADD      MEGIND
          .STA      MEGIND
```

```
..이제 나누기 시작
. 첫번째 덩어리
..첫번째 덩어리는 Start Address 가 같다.
ONEDIV      LDA         MEGNUM      .길이를 반으로 쪼개고 재귀함수를 돌린다.
            DIV         #2
            STA         MEGNUM
            JSUB        MEGCHK      .4A7
..두번째 덩어리
TWODIV      LDA         #3
            ADD         MEGIND
            STA         MEGIND
            LDA         @MEGIND
            STA         STAADD      .첫번째 덩어리가 끝나고 돌아온거라 Stack 에서 해당 전체 loop 의 Start Address 를 꺼내온다.

            LDA         #3
            ADD         MEGIND
            STA         MEGIND
            LDA         @MEGIND     .Length
            STA         MEGNUM
            DIV         #2
            MUL         #3
            ADD         STAADD
            STA         STAADD      .두번째 덩어리 시작주소

            LDA         #3
            ADD         MEGIND
            STA         MEGIND
            .Length 를 구하기 위해서(해당 Loop 의 전체 Length - (전체 Length/2)를 해주면 남은 Length 가 나온다)
            LDA         MEGNUM
            DIV         #2
            STA         TMPNUM
            LDA         MEGNUM
            SUB         TMPNUM
            STA         MEGNUM
            COMP        #2
            JSUB        MEGCHK      .재귀

..변수 만들어주기(각 덩어리 처음 값 마지막 값, 길이?)
...Merge Index One|  |  |Merge End One   ; 첫번째 덩어리
...Merge Index Two|  |  |Merge End Two   ; 두번째 덩어리
..두번째 덩어리가 끝나고 온 상태라 두번째 덩어리부터 채워준다.
MAKEVA      LDA         STAADD      .두번째 덩어리 Start Address
            STA         MINDT
            LDA         MEGNUM
            MUL         #3
            ADD         STAADD
            STA         MENDT       .두번째 덩어리 End Address

            LDL         @MEGIND     .Return Address
            LDA         #3
            ADD         MEGIND
            STA         MEGIND
            LDA         @MEGIND     .Start Address
            STA         MINDO       .첫번째 덩어리 Start Address
            STA         STAADD      .해당 값을 전체 loop Start Address 변수에 저장.
            LDA         MEGIND
            ADD         #3
            STA         MEGIND
            LDA         @MEGIND
            DIV         #2
            SUB         #1
            MUL         #3
            ADD         MINDO
            STA         MENDO       .두번째 덩어리 Start Address
            LDA         MEGIND
```

```
            ADD     #3
            STA     MEGIND
            LDA     STAADD
            SUB     #48
            STA     MSTIND
            LDA     MENDT
            SUB     #48
            STA     ENDADD
            .MSTLNE 에 시작주소 넣어야하는데..
            .계산 끝나고 4 뒤로 밀것

.덩어리의 맨 앞의 값부터 비교 후 순서대로 저장.
            CLEAR   S
            CLEAR   T
CALMEG      CLEAR   A
            COMPR   A, S        .첫번째 덩어리가 끝났는지 chk
            JLT     MEGTWO
            COMPR   A, T        .두번째 덩어리가 끝났는지 chk
            JLT     MEGONE
            LDA     @MINDO      .첫번째 덩어리의 값과 두번째 덩어리의 값을 차례대로 비교 후 임시 저장소에 순서대로 저장.
            COMP    @MINDT
            JGT     MEGTWO

            .one 이 더 작을 때
MEGONE      LDA     @MINDO      .573
            STA     @MSTIND
            LDA     #3
            ADD     MSTIND
            STA     MSTIND
            COMP    ENDADD
            JEQ     ENDMEG
            JGT     ENDMEG
            LDA     #3
            ADD     MINDO
            STA     MINDO
            COMP    MENDO
            JGT     ONEEND
            J       CALMEG

            .Two 가 더 작을 때
MEGTWO      LDA     @MINDT
            STA     @MSTIND
            LDA     #3
            ADD     MSTIND
            STA     MSTIND
            COMP    ENDADD
            JGT     ENDMEG
            LDA     #3
            ADD     MINDT
            STA     MINDT
            COMP    MENDT
            JEQ     TWOEND
            JGT     TWOEND
            J       CALMEG

.각 덩어리가 끝났으면 다른 덩어리에 있는 값을 계속 저장.
ONEEND      LDS     #1
            J       CALMEG
TWOEND      LDT     #1
            J       CALMEG

.MSTIND ~ MENDT 까지 복사
ENDMEG      LDA     STAADD
            SUB     #48
            STA     MSTIND
```

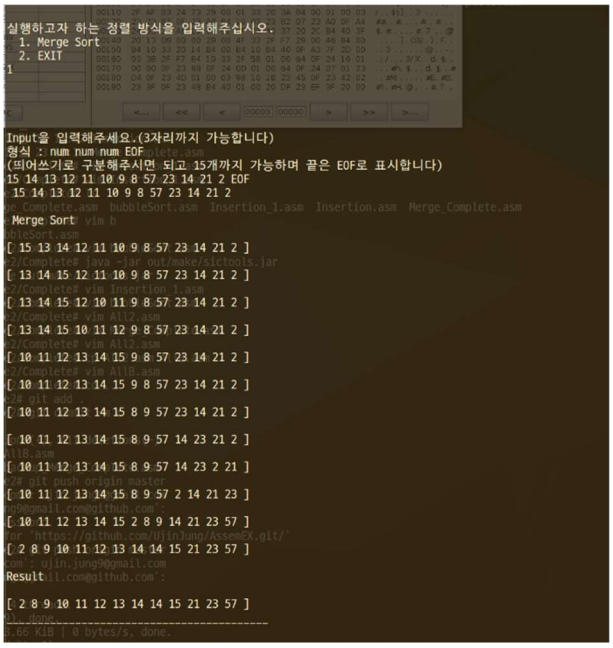
```
COPYST      LDA         @MSTIND
            STA         @STAADD
            LDA         #3
            ADD         STAADD
            STA         STAADD
            SUB         #48
            STA         MSTIND
            COMP        ENDADD
            JLT         COPYST
            JEQ         COPYST
            LDA         #10
            WD          OUTDEV
            WD          OUTDEV

            .계산된 값 출력
            LDA         #91
            WD          OUTDEV
            CLEAR       X
            JSUB        ENDINP
            LDA         #32
            WD          OUTDEV
            LDA         #93
            WD          OUTDEV
            LDA         MEGIND
            SUB         #9
            STA         MEGIND
            LDL         @MEGIND


            CLEAR       A
            STA         @MEGIND
            LDA         #3
            ADD         MEGIND
            STA         MEGIND
            LDA         @MEGIND
            STA         STAADD
            CLEAR       A
            STA         @MEGIND
            LDA         #3
            ADD         MEGIND
            STA         MEGIND
            LDA         @MEGIND
            STA         MEGNUM
            CLEAR       A
            STA         @MEGIND
.두개 읽어오고 해당 stack 을 삭제해야하기 때문에
..(2+3)*3 인 15 를 빼준다.
            LDA         MEGIND
            SUB         #15
            STA         MEGIND

            RSUB
```

```
.----------------Result----------------------

RESMSG      CLEAR       X
            CLEAR       A
            .STA        PIVIND
            LDA         #RESLEN
            SUB         #RESTXT
            STA         RESLEN
RESPRT      LDA         RESTXT, X
            WD          OUTDEV
            TIX         RESLEN
            JLT         RESPRT
            CLEAR       A
            ADDR        L, A
            STA         RETADD
            LDA         #10
            WD          OUTDEV
            LDA         #91
            WD          OUTDEV
            CLEAR       X
            JSUB        ENDINP
            LDA         #32
            WD          OUTDEV
            LDA         #93
            WD          OUTDEV
            LDA         #10
            WD          OUTDEV
            LDL         RETADD
            RSUB
.
.교환 후에 TMPNXT 3 증가 후 PIVIND 와 비교
.PIVIND 보다 크지 않으면 BUBBLE 로 이동해서 시작
.TMPNXT > PIVIND 이면 PIVIND 3 감소
..PIVIND 3 감소 후 STR1 과 비교 후 같으면 끝

STAADD      RESW        1                   .각 SAMPLE Input 시작 주소 Index 값
NUMADD      RESW        1                   .각 SAMPLE Input 숫자 시작 주소 Index 값
                                            .ver1.3 출력용 indirect 주소저장소로 사
ENDADD      RESW        1
INPLEN      RESW        1                   .각 SAMPLE 숫자 길이
INPNUM      WORD        0                   .각 SAMPLE Input 개수
TMPNUM      RESW        1                   .숫자를 임시로 저장해둔다.
TMPIND      RESW        1
TMPNXT      RESW        1

.Merge Sort
MINDO       RESW        1
MENDO       RESW        1
MINDT       RESW        1
MENDT       RESW        1
MEGNUM      RESW        1

.STACK?
MEGIND      RESW        1           .밑의 Stack 의 주소값을 가리키는 공간
            RESW        3           .ReturnAddress(1Word) | StartAddress(1Word) | Length(1Word)
            RESW        3
            RESW        3
            RESW        3
            RESW        3

.Merge 를 위해 임시로 저장되는 공간
MEGST1      RESW        1
MEGST2      RESW        1
            RESW        1
            RESW        1
```

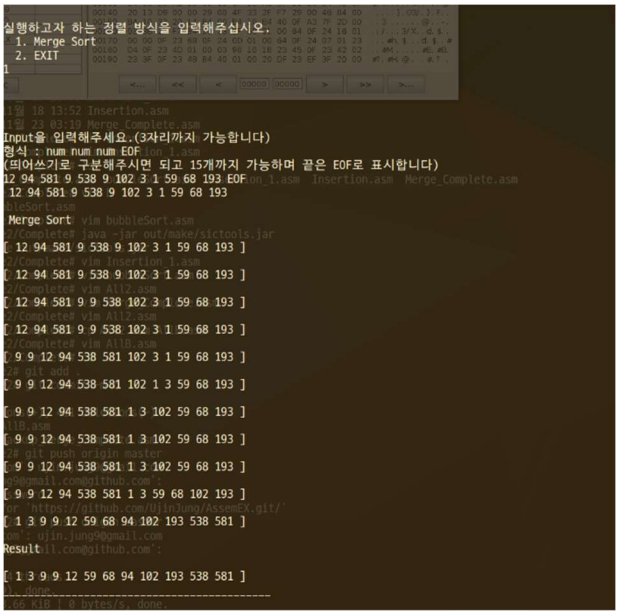
```
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
MSTIND      RESW    1

.Input 이 저장되는 공간
STR1        RESW    1                       .배열 시작
STR2        RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
            RESW    1
STRLEN      RESW    1

FIGURE      WORD    100
JUKCHK      WORD    0           .Junk 인지 chk 하기 위한 변수 (0 = not junk, 1 = junk)
INDEV       BYTE    X'00'
OUTDEV      BYTE    X'01'
RETADD      RESW    1           .Main 으로 Return Address
PREADD      RESW    1           .Print 하고 Sort 로 Return Address
CHOTXT      BYTE    C'  실행하고자 하는 정렬 방식을 입력해주십시오.'
            BYTE    10
            BYTE    C'  1. Merge Sort'
            BYTE    10
            BYTE    C'  2. EXIT'
            BYTE    10
CHOLEN      RESW    1
IMSTXT      BYTE    C'  Input 을 입력해주세요.(3 자리까지 가능합니다)'          .Input MaSsage Text
            BYTE    10
            BYTE    C'형식 : num num num EOF'
            BYTE    10
            BYTE    C'(띄어쓰기로 구분해주시면 되고 15 개까지 가능하며 끝은 EOF 로 표시합니다)'
            BYTE    10
IMSLEN      RESW    1
RESTXT      BYTE    C'  Result'
            BYTE    10
RESLEN      RESW    1
MEGTXT      BYTE    10
            BYTE    C'  Merge Sort'

MEGLEN      RESW    1
```

3. 실행

1. Input : 15 14 13 12 11 10 9 8 57 23 14 21 2 EOF



2. Input : 12 94 581 9 538 9 102 3 1 59 68 193 EOF



주요 기능

```
MEGINI      LDA         STAADD
            SUB         #48
            STA         MSTIND
            LDA         @STAADD
            STA         @MSTIND
            LDA         #3
            ADD         STAADD
            STA         STAADD
            COMP        ENDADD
            JLT         MEGINI
.복사가 끝나면 다시 초기화를 해준다.
            LDA         #STR1
            STA         STAADD
            LDA         #MEGST1
            STA         MSTIND
```

STR1 ~ 에 받아온 Input 을 MEGST1 ~ 에 저장하는 과정이다.

```
MEGCHK      LDA         MEGNUM
            COMP        #1
            JGT         MERGE
.CALMEG로 이동
.해당 LOOP의 정보를 스택에 저장
            LDA         MEGIND
            SUB         #9
            STA         MEGIND
            RSUB
```

덩어리의 크기가 1 이면 Index 를 뒤로 물리고 return 한다.

아니라면 Merge 로 이동

```
MERGE       CLEAR       A           .466
            CLEAR       T
            ADDR        L, A
            STA         @MEGIND     .Return Address
            LDA         #3
            ADD         MEGIND
            STA         MEGIND
            LDA         STAADD
            STA         @MEGIND     .Start Address
            LDA         #3
            ADD         MEGIND
            STA         MEGIND
            LDA         MEGNUM
            STA         @MEGIND     .Length
            LDA         #3
            ADD         MEGIND
            STA         MEGIND
```

Stack 에 해당 LOOP 의 Return Address/Start Address/Length 를 저장하는 과정이다.

```
..이제 나누기 시작
. 첫번째 덩어리
..첫번째 덩어리는 Start Address가 같다.
ONEDIV      LDA         MEGNUM      .길이를 반으로 쪼개고 재귀함수를 돌린다.
            DIV         #2
            STA         MEGNUM
            JSUB        MEGCHK      .4A7
..두번째 덩어리
TWODIV      LDA         #3
            ADD         MEGIND
            STA         MEGIND
            LDA         @MEGIND
            STA         STAADD
.첫번째 덩어리가 끝나고 와서 Stack에서 해당 전체 loop의 Start Address를 꺼내온다.
            LDA         #3
            ADD         MEGIND
            STA         MEGIND
            LDA         @MEGIND     .Length
            STA         MEGNUM
            DIV         #2
            MUL         #3
            ADD         STAADD
            STA         STAADD      .두번째 덩어리 시작주소

            LDA         #3
            ADD         MEGIND
            STA         MEGIND
.Length = (해당 Loop의 전체 Length - (전체 Length/2))
            LDA         MEGNUM
            DIV         #2
            STA         TMPNUM
            LDA         MEGNUM
            SUB         TMPNUM
            STA         MEGNUM
            COMP        #2
            JSUB        MEGCHK      .재귀
```

덩어리를 나누어주는 작업이다. Start Address 와 Input 의 길이를 계산하여 저장 후 재귀의 과정을 거친다.

```
.덩어리의 맨 앞의 값부터 비교 후 순서대로 저장.
            CLEAR       S
            CLEAR       T
CALMEG      CLEAR       A
            COMPR       A, S        .첫번째 덩어리가 끝났는지 chk
            JLT         MEGTWO
            COMPR       A, T        .두번째 덩어리가 끝났는지 chk
            JLT         MEGONE
            LDA         @MINDO      .첫번째 덩어리의 값과 두번째 덩어리의 값을 차례대로 비교 후 임시 저장소에 순서대로 저장.
            COMP        @MINDT
            JGT         MEGTWO

            .one이 더 작을 때
MEGONE      LDA         @MINDO      .573
            STA         @MSTIND
            LDA         #3
            ADD         MSTIND
            STA         MSTIND
            COMP        ENDADD
            JEQ         ENDMEG
            JGT         ENDMEG
            LDA         #3
            ADD         MINDO
            STA         MINDO
            COMP        MENDO
            JGT         ONEEND
            J           CALMEG

            .Two가 더 작을 때
MEGTWO      LDA         @MINDT
            STA         @MSTIND
            LDA         #3
            ADD         MSTIND
            STA         MSTIND
            COMP        ENDADD
            JGT         ENDMEG
            LDA         #3
            ADD         MINDT
            STA         MINDT
            COMP        MENDT
            JEQ         TWOEND
            JGT         TWOEND
            J           CALMEG
```

재귀가 끝난 뒤에 해당 Loop 에 있는 왼쪽 덩어리와 오른쪽 덩어리를 비교하여 작은 값부터 차례대로 저장해준다.

어려웠던 점

1. 재귀함수를 구현하는 과정이 어려웠습니다.

스택에 쌓아 두고 어느 시점에 받아오고 어느 시점에 초기화 시킬지에 대해 많은 고민을 했습니다.

2. 각 덩어리 들 사이의 시작 지점과 끝 지점 계산하는 데 오류가 많이 있었습니다.