Starter의 기능 설명 및 테스트 계획서

지난 주 AbsoluteLoader를 통해 Starter를 받아와 새로운 Boot파일을 만드는 실습을 하였습니다.

*** 아래는 어셈블리 후 생성된 Loader의 리스트입니다.**

```
Execute% Assemble 3:cassette/myTest.lmc
                 0000:0000   1 ABSOLUTE_LOADER START   0
                 0000:0000   2 KEYBOARD_DEC    EQU     10
                 0000:0000   3 SCREEN_DEC      EQU     12
                 0000:0000   4 SCREEN_TXT      EQU     SCREEN_DEC+1
                 0000:0000   5 BOOT_DEV_ADDR   EQU     30
                             6
                 0000:0000   7 PROGNAME        RESBOX  10
                 0000:0010   8 I_VAL           RESDBOX 1
                 0000:0012   9 LENGTH          RESDBOX 1
                 0000:0014  10 EXEC_START      RESDBOX 1
                 0000:0016  11 PTR_START       RESDBOX 1
                            12
6700 0022 0000   0000:0018  13                 CALL    MAIN
0700             0000:0021  14                 COB
                            15
                            16
9630             0000:0022  17 MAIN            TIO     BOOT_DEV_ADDR
0530             0000:0023  18                 IN      BOOT_DEV_ADDR
1605 0072 0000   0000:0024  19                 SUB     A #72
5910 0031 0000   0000:0027  20                 JZ      HEAD_IS_FINE
9710             0000:0030  21                 INT     10
                            22
                            23 //#begin
                            23 //% HEAD_IS_FINE MOV C #10
4325 0010 0000   0000:0031  23 HEAD_IS_FINE    LD  C #10
                            23 //#end
                            24
                            25 //#begin
                            25 //% MOV B #0
4315 0000 0000   0000:0034  25                 LD  B #0
                            25 //#end
                            26
0530             0000:0037  27 LOAD_NAME       IN      BOOT_DEV_ADDR
                            28 //#begin
                            28 //% MOV %PROGNAME A
5205 0000        0000:0038  28                 ST  A %PROGNAME
                            28 //#end
                            29
1315 0001 0000   0000:0040  30                 ADD     B #1
5990 0037 0000   0000:0043  31                 JLOOP   LOAD_NAME
                            32
0530             0000:0046  33                 IN      BOOT_DEV_ADDR
5300 0016 0000   0000:0047  34                 ST      A PTR_START
0530             0000:0050  35                 IN      BOOT_DEV_ADDR
5300 0017 0000   0000:0051  36                 ST      A PTR_START+1
                            37
0530             0000:0054  38                 IN      BOOT_DEV_ADDR
5300 0012 0000   0000:0055  39                 ST      A LENGTH
0530             0000:0058  40                 IN      BOOT_DEV_ADDR
5300 0013 0000   0000:0059  41                 ST      A LENGTH+1
```

```
                              42
4320 0012 0000    0000:0062   43                  LD      C LENGTH
                              44 //#begin
                              44 //% MOV X #0
4345 0000 0000    0000:0065   44                  LD  X #0
                              44 //#end
0530              0000:0068   45 LOAD_TAPE        IN      BOOT_DEV_ADDR
5303 0016 0000    0000:0069   46                  ST      A *@PTR_START
                              47
1345 0001 0000    0000:0072   48                  ADD     X #1
5990 0068 0000    0000:0075   49                  JLOOP   LOAD_TAPE
                              50
0530              0000:0078   51                  IN      BOOT_DEV_ADDR
                              52 //#begin
                              52 //% MOV B #69
4315 0069 0000    0000:0079   52                  LD  B #69
                              52 //#end
1401              0000:0082   53                  SUB     A B
5910 0087 0000    0000:0083   54                  JZ      END_IS_FINE
9710              0000:0086   55                  INT     10
                              56
0530              0000:0087   57 END_IS_FINE     IN      BOOT_DEV_ADDR
5300 0014 0000    0000:0088   58                  ST      A EXEC_START
0530              0000:0091   59                  IN      BOOT_DEV_ADDR
5300 0015 0000    0000:0092   60                  ST      A EXEC_START+1
5901 0014 0000    0000:0095   61                  JUMP    *EXEC_START
                              62
                  0000:0098   63 NULBOX           RESBOX  1
9999              0000:0099   64                  RET
                              65 END
```

*** 아래는 어셈블리 후 생성된 Starter의 리스트입니다.**

```
Execute% Assemble 3:cassette/myStarter.lmc
                               1 // System Monitor Program Entry
                  0000:2000    2 ROM_BEGIN         EQU 91000000U
                  0000:2000    3 BIOS              EQU 90011000U
                  0000:2000    4 M_PROMPT          EQU 30
                  0000:2000    5 M_QUIT            EQU 31
                  0000:2000    6 M_LOADDEV         EQU 32
                  0000:2000    7 M_LOADBL          EQU 33
                  0000:2000    8 M_LOADRL          EQU 34
                  0000:2000    9 M_LOADLL          EQU 35
                  0000:2000   10 M_LOAD            EQU 36
                  0000:2000   11 M_MOVE            EQU 37
                  0000:2000   12 M_RUN             EQU 38
                  0000:2000   13 M_DUMP_CMOS       EQU 39
                  0000:2000   14 M_DUMP            EQU 40
                  0000:2000   15 M_ASM             EQU 41
                  0000:2000   16 M_DASM            EQU 42
                  0000:2000   17 M_INTERPRET       EQU 43
                  0000:2000   18 M_HELP            EQU 44
                  0000:2000   19 M_CC              EQU 45
                  0000:2000   20 M_CAT             EQU 46
                  0000:2000   21 M_MKBOOT          EQU 47
                  0000:2000   22 M_VIEWBL          EQU 48
                  0000:2000   23 M_VIEWRL          EQU 48
                  0000:2000   24 M_VIEWLL          EQU 48
                              25                  // Monitor
                  0000:2000   26 ROM3              START  2000 // ROM3
9993              0000:2000   27 MONITOR_ENTRY     INTON
                              28
                              29
```

해당하는 명령어로 이동하기 위해, 명령어에 해당하는 값을 지정해 둡니다.

프롬프트에서 입력을 받으면 이 값을 참고하여 해당 명령어로 이동 후 실행합니다.

```
                              30
4250 0022          0000:2001  31                LD   SP NEAR =4900U
4200 0022          0000:2003  32 PROMPT          LD   A NEAR =4700U // BUFFER 주소
4345 2027 0000     0000:2005  33                LD   X #=C'20103390' // PROMPT 주소
9730               0000:2008  34                INT  M_PROMPT
4140               0000:2009  35                LD   X A // Return value X에 저장
1640 2036 0000     0000:2010  36                SUB  X =31 // PROMPT는 INT값을 반환
                              37                          // 31부터시작함으로 31을 뺀다.
                              38                          // M_PROMPT가 30임으로 31부터...
1144               0000:2013  39                ADD  X X // 2X, DBOX Indexing
4200 0011          0000:2014  40                LD   A NEAR =4700U // BUFFER 주소
9880               0000:2016  41                PUSH A
6802 0023          0000:2017  42                CALL NEAR @MONITOR_TABLE
1250 0019          0000:2019  43                ADD  SP NEAR =2
5800 9982          0000:2021  44                JUMP NEAR PROMPT
                              45
                              46                LTORG
4900 0000          0000:2023  46 $             =4900
4700 0000          0000:2025  46 $             =4700
0050               0000:2027  46 $             ='2'
0048               0000:2028  46                '0'
0049               0000:2029  46                '1'
0048               0000:2030  46                '0'
0051               0000:2031  46                '3'
0051               0000:2032  46                '3'
0057               0000:2033  46                '9'
0048               0000:2034  46                '0'
0000               0000:2035  46                '\0'
0031 0000          0000:2036  46 $             =31
0002 0000          0000:2038  46 $             =2
                              47
9731               0000:2040  48 MONITOR_TABLE   INT M_QUIT
9999               0000:2041  49                RET
9732               0000:2042  50                INT M_LOADDEV
9999               0000:2043  51                RET
9733               0000:2044  52                INT M_LOADBL
9999               0000:2045  53                RET
9734               0000:2046  54                INT M_LOADRL
9999               0000:2047  55                RET
9735               0000:2048  56                INT M_LOADLL
9999               0000:2049  57                RET
9736               0000:2050  58                INT M_LOAD
9999               0000:2051  59                RET
9737               0000:2052  60                INT M_MOVE
9999               0000:2053  61                RET
9738               0000:2054  62                INT M_RUN
9999               0000:2055  63                RET
9739               0000:2056  64                INT M_DUMP_CMOS
9999               0000:2057  65                RET
9740               0000:2058  66                INT M_DUMP
9999               0000:2059  67                RET
9741               0000:2060  68                INT M_ASM
9999               0000:2061  69                RET
9742               0000:2062  70                INT M_DASM
9999               0000:2063  71                RET
9743               0000:2064  72                INT M_INTERPRET
9999               0000:2065  73                RET
9744               0000:2066  74                INT M_HELP
9999               0000:2067  75                RET
9745               0000:2068  76                INT M_CC
9999               0000:2069  77                RET
9746               0000:2070  78                INT M_CAT
9999               0000:2071  79                RET
9747               0000:2072  80                INT M_MKBOOT
9999               0000:2073  81                RET
```

프롬포트를 실행하여 명령어와 주소값을 입력받게 됩니다.

프롬포트 실행 후, 파라미터로 넘어온 값을 통해 해당하는 명령문으로 넘어갑니다.

개인화한 프롬포트 출력문이 저장되어 있습니다.

해당하는 명령어로 점프하여 실행하게 됩니다

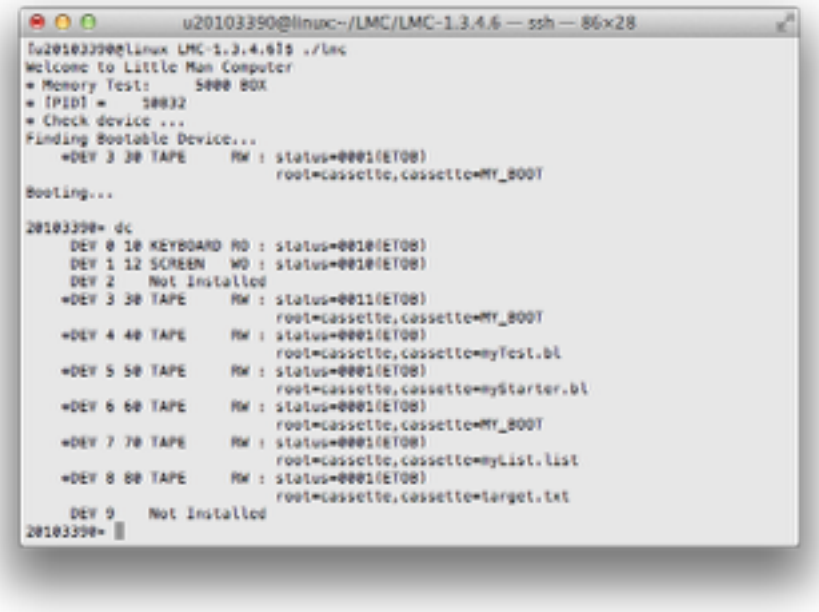
.

INT 를 통해 이미 정의되어 있는 명령어를 실행하게 됩니다.

```
9744            0000:2074  82                INT M_HELP   // VIEW BL이어야 하지만 구현이 안
되어있음.
9999            0000:2075  83                RET
9744            0000:2076  84                INT M_HELP   // VIEW RL이어야 하지만 구현이 안
되어있음.
9999            0000:2077  85                RET
9744            0000:2078  86                INT M_HELP   // VIEW LL이어야 하지만 구현이 안
되어있음.
9999            0000:2079  87                RET
                           88                END
```

*** 아래는 mkboot 를 통해 생성된 MY_BOOT의 목적코드 입니다.**

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000006700002200000700963005301605007200005910003100009710432500100000431500000000530520500001315000100005990003700000530530000160000053053000017000005305300001200000530530000130000432000120000434500000000053053030016000013450001000059900068000005304315006900001401591000870000971005305300014000005305300001500005901001400000000999900720082007900770051000000000000000000000000020000000080000099934250002242000022434520270000973041401640203600001144420000119880680200231250001958009982490000004700000000500048004900480051005100570048000000310000000200009731999997329999973399999734999997359999973699999737999997389999973999999740999997419999974299999974399999744999997459999974699999747999997449999974499999744999997449999006920000000
```



AbsoluteLoader를 통해 3번 장치에 장착되어 있는 Starter의 Tape값을 2000번지로 옮겨 적어(Loading) 그 위치로 Jump하는 과정을 진행합니다.

이를 mkboot하여 새로운 BOOT파일을 생성하여 작동시켰고, 이로써 새로 생성된 STARTER가 정상 작동함을 확인하였습니다.

* myTest.bl 은
AbsloluteLoader에 해당합니다.

* myStarter 는
Starter에 해당합니다.

위 기존의 Starter를 보면, 기본적인 프롬프트 명령어들이 정의가 되어 있습니다. 하지만 82번부터 87까지 VIEW BL/RL/LL 이 구현이 되어있지 않음을 확인 할 수 있습니다. 또한 프롬프트를 통해 입력받은 문자열을 따로 저장하지 않아, 다양한 기능을 구현함에 있어 제한이 있습니다.

앞으로 'lmcc' 확장자로 작성되는 Starter를 통해, STRCMP, STRTOK, STRLEN 명령어를 이용하여 프롬프트에서 입력받은 문자열을 저장할 것 입니다. 그리고 이를 이용하여 VIEW BL/RL/LL 및 LOAD RL 등의 기본적인 프롬프트 명령어를 구현(1-필수)하고 더 나아가 저장된 문자열을 이용하여 기존에 없는, 예를 들어 간단한 도형출력 등의 기능을 추가(2-옵션)하여 볼 것입니다.

```
// MyStarter Program
M_PROMPT          EQU 30
M_QUIT            EQU 31
M_LOADDEV         EQU 32
M_LOADBL          EQU 33
M_LOADRL          EQU 34
M_LOADLL          EQU 35
M_LOAD            EQU 36
M_MOVE            EQU 37
M_RUN             EQU 38
M_DUMP_CMOS       EQU 39
M_DUMP            EQU 40
M_ASM             EQU 41
M_DASM            EQU 42
M_INTERPRET       EQU 43
M_HELP            EQU 44
M_CC              EQU 45
M_CAT             EQU 46
M_MKBOOT          EQU 47
VIEWBL            EQU 48
VIEWRL            EQU 49
VIEWLL            EQU 50

                  // MyStart 시작
MYSTARTER         START  2000
                  LD   SP #STACK_BTM
PROMPT            LD   A #BUF // BUFFER 주소
                  LD   X #=C'U20103390' // PROMPT 문자열 주소.
                  INT  M_PROMPT
                  LD   X A // Return value X에 저장
                  SUB  X #31

                  LD   C X
                  ADD  X C
                  ADD  X C // 3X, 3BOX Indexing
                  LD   A #BUF
                  PUSH A
                  CALL NEAR @MONITOR_TABLE
                  ADD  SP #2
                  JUMP NEAR PROMPT

MONITOR_TABLE     INT M_QUIT
                  RET
                  NOP
                  INT M_LOADDEV
                  RET
                  NOP
                  INT M_LOADBL
                  RET
                  NOP
                  INT M_LOADRL
                  RET
                  NOP
                  INT M_LOADLL
                  RET
                  NOP
                  INT M_LOAD
                  RET
                  NOP
                  INT M_MOVE
                  RET
                  NOP
                  INT M_RUN
                  RET
                  NOP
```

해당하는 명령어로 이동하기 위해, 명령어에 해당하는 값을 지정해 둡니다.

프롬프트에서 입력을 받으면 이 값을 참고하여 해당 명령어로 이동 후 실행합니다.

또한 새로운 값을 추가하여 기존에 없는 명령어를 추가 할 수 있습니다. 이때,
'memory/roms/monitor.h'
'memory/roms/monitor.c'
파일에도 새로운 값을 추가해야 합니다.

프롬프트 출력문을 개인화 할 수 있습니다.

QUIT를 입력받기 전까지는 STARTER 프로그램을 반복하여 프롬프트를 실행 합니다.

```
                INT M_DUMP_CMOS
                RET
                NOP
                INT M_DUMP
                RET
                NOP
                INT M_ASM
                RET
                NOP
                INT M_DASM
                RET
                NOP
                INT M_INTERPRET
                RET
                NOP
                INT M_HELP
                RET
                NOP
                INT M_CC
                RET
                NOP
                INT M_CAT
                RET
                NOP
                INT M_MKBOOT
                RET
                NOP
                JUMP NEAR VIEWBL /* VIEW BL 아래 의사코드 구현 */
                NOP
                INT M_HELP   // VIEW RL이어야 하지만 구현이 안되어있음.
                RET
                NOP
                INT M_HELP   // VIEW LL이어야 하지만 구현이 안되어있음.
                RET
                NOP

VIEWBL          #pragma C {

                // 입력이 "VIEW BL 50" 일경우
                // BUF에도 C'VIEW BL 50' 이 저장되어 있다.
                char* cmd  = strtok(BUF);  // "VIEW"
                char* arg1 = strtok(BUF);  // "BL"
                char* arg2 = strtok(BUF); // "50";
                int ioaddr = asm
                  {
                    LD.I.CW A arg2
                    ST A ioaddr
                  } // atoi(arg2);
                while( feof(ioaddr) ) // TIO 50
                                      // SKZ3 ...
                  {
                    ch = fgetc(ioaddr); // IN ioaddr
                    if( ch == 'H' ) {
                    } else ... {
                    }
                  }

                } // end #pragma C
                RET

                LTORG
BUF             RESBOX  100
STACK           RESDBOX 200
STACK_BTM       EQU     $
                END
```

해당하는 명령어로 점프하여 실행하게 됩니다. INT 를 통해 이미 정의되어 있는 명령어를 실행하게 됩니다.

새로 추가되는 명령어는 Starter의 아래부분에 작성되며, 이를 실행하기 위해 해당하는 명령어가 작성된 주소로 JUMP 하여 실행하게 됩니다.

추가하게 될 VIEWBL이 의사코드로 작성되어 있습니다. 이와 같은 식으로 구현되지 않거나 혹은 새로운 명령어를 추가하게 될 것입니다.

프롬프트에서 입력받은 문자열이 저장되어 있습니다.