Dalhousie University
Department of Electrical and Computer Engineering

ECED 3403 – Computer Architecture

Assignment 2: Designing, implementing, and testing of an XM3 loader and debugger for XM3's memory

1 Objective

Before we can design and implement the XM3 ISA emulator, it is necessary to have memory for the machine, a way of getting programs into the memory, and a way of checking the contents of memory.

In this assignment, we are to design, implement, and test the loader and debugger on the emulated memory.

2 Memory

XM3 has 64 KiB of memory. This is to be emulated using an array.

2.1 The Bus

XM3's 64 KiB of memory will be accessed through a bus() function that implements the memory address decoder. The entry point has four arguments:

```
void bus(unsigned short mar, unsigned short *mbr, enum ACTION rw, enum SIZE bw)
```

The function's parameters are:

mar: The memory address being accessed. The address refers to a byte location (it can be an odd or even address). The parameter **bw** indicates whether the memory location accessed is either a byte or a word. A word address is **mar** >> 1.

mbr: The bidirectional memory buffer. The **mbr** is a pointer and refers to the address of the data being written to the specified memory location (i.e., the pointer is to be used as an *rvalue*) or the address of the variable to be assigned the contents of the specified (i.e., the pointer is to be used as an *lvalue*). The parameter **rw** indicates whether the **mbr** is to be used for reading or writing.

rw: The read-write indicator. Action should be either READ or WRITE.

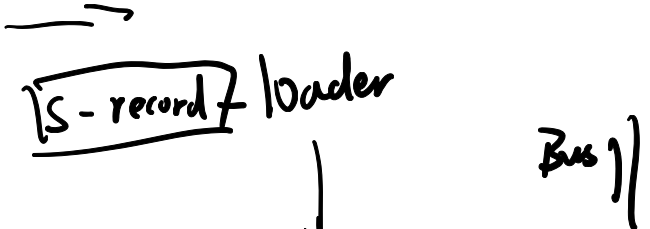
bw: The byte-word indicator. The size should be either BYTE or WORD.

3 The Loader

The loader is to read a file containing S-Records. If the loader detects an invalid record, it is to issue one of these warnings:

- Missing file
- File does not contain S-records
- Incorrect checksum.

If an error other than the above is detected, the warning should be "Warning, unknown error".



The loader is to write to memory using the bus() function.

3.1 The S-Record

The XM3 assembler support three different S-Record types; the format and contents of each are:

S0: The source file name.

S1: The data values or instructions, or both to be stored in contiguous memory locations.

S9: The initial entry point (i.e., the starting address).

Each record is prefixed with a header (Sx), its length (one-byte), a two-byte address (0000 to FFFF), the contents of the record (from 0 to 62 hex characters, with two characters representing one byte), and a one-byte checksum.

A complete description of the S-Record format can be found here.

3.2 Command line

The loader's command line is to have an optional command-line argument for use when running in the VM or a command-line interpreter.[1] The format is:

```
myxm3 -d fname.dbg
```

where:

`fname.dbg`: If the command line contains '`-d`', a file containing a list of test commands to be executed (i.e., a script file) is supplied. For this assignment, the commands are 'L', 'D', and 'X'. Invalid commands can be included as well to test your debugger.

When your debugger prompts for input, rather than reading a character from the user's keyboard, the program is to read from `fname.dbg`. The character read by the input function should be returned and processed like any other command.

4 The Debugger

The debugger is to allow the user to interact with the XM3 emulator; at this point, the emulator is only memory and the bus() function. In this assignment, it is to be an interactive tool that supports three commands (the commands can be in upper or lower case):

L <fname.ext>: Load the specified file. Only files containing S-Records are accepted. The loader function to be used is described above.

D <start> <stop>: Dump memory from memory address *start* to address *stop*. The addresses refer to a byte location and are hex values (i.e., 0 to FFFF). The format of the output should be the address (4 hex characters), the contents of 16 memory locations starting at the specified address (shown as bytes), and the 16 memory locations displayed as ASCII characters.

[1] To run the Windows' command-line interpreter, type ⊞+'R'. At the prompt, type "cmd".

X: Exit (terminate) the emulator.

The debugger is to access memory using the bus() function.

An example of the debugger follows.

5 Example

As an example of the debugger, the loader, and memory, consider the following XM3 assembly file named A2.asm:

```
;
; Example of S-Record output
; A simple looping program (expensive way to add 8 to a register)
; ECED 3403
; 23 May 2020
;
        org     #80             ; Data section
Data    word    #0
;
        org     #1000           ; Code section
Start   movlz   Data,R0         ; R0 <- address of Data (#0080)
        ld      R0,R1           ; R1 <- mem[R0]
;
Loop    cmp     #8,R1           ; Check R0 = 8
        cex     ne,#2,#0; Do 2 instructions if NE; skip 2 if EQ
;
        add     #1,R1           ; R1 <- R1 + 1
        bra     Loop            ; Repeat
;
        st      R1,R0           ; mem[R0] <- R1
Done                            ; Use as breakpoint
;
        end     Start           ; S9 record get address of Start (#1000)
```

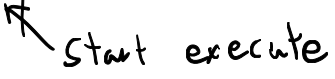
when run through the assembler, two files are produced, A2.lis and A2.xme.

The listing file, A2.lis, lists each record; for those records that produce data or instructions for the machine, the address and value of the data or instruction are listed. For example:

```
X-Makina Assembler - Version 3.01 (23 May 2020)
Input file name: A2.asm
Time of assembly: Sat 23 May 2020 12:12:23
 1                          ;
 2                          ; Example of S-Record output
 3                          ; A simple looping program (expensive way to add 8 to a register)
 4                          ; ECED 3403
 5                          ; 23 May 2020
 6                          ;
 7                                  org     #80             ; Data section
 8       0080    0000       Data     word    #0
 9                          ;
10                                  org     #1000           ; Code section
11       1000    6C00       Start    movlz   Data,R0         ; R0 <- address of Data (#0080)
12       1002    5801               ld      R0,R1           ; R1 <- mem[R0]
13                          ;
14       1004    45A1       Loop     cmp     #8,R1           ; Check R0 = 8
15       1006    2450               cex     ne,#2,#0; Do 2 instructions if NE; skip 2 if EQ
16                          ;
17       1008    4089               add     #1,R1           ; R1 <- R1 + 1
18       100A    23FC               bra     Loop            ; Repeat
19                          ;
20       100C    5C08               st      R1,R0           ; mem[R0] <- R1
21                          Done                            ; Use as breakpoint
22                          ;
23                                  end     Start           ; S9 record get address of Start (#1000)

Successful completion of assembly

** Symbol table **
Name                       Type    Value   Decimal
Done                       LBL     100E    4110
Loop                       LBL     1004    4100
Start                      LBL     1000    4096
Data                       LBL     0080    128
R7                         REG     0007    7
R6                         REG     0006    6
R5                         REG     0005    5
R4                         REG     0004    4
R3                         REG     0003    3
R2                         REG     0002    2
R1                         REG     0001    1
R0                         REG     0000    0

.XME file: C:\Users\larry\OneDrive\Courses\ECED 3403 - 2020\XM3\XM3 - Test files\Test files 4\A2.xme
```

The load module or executable file, A2.xme, contains the data, instructions, and directives for the loader in S-Records:

```
S00900004322E61736D14
S105008000007A
S1111000006C0158A14550248940FC23085C73
S9031000EC
```



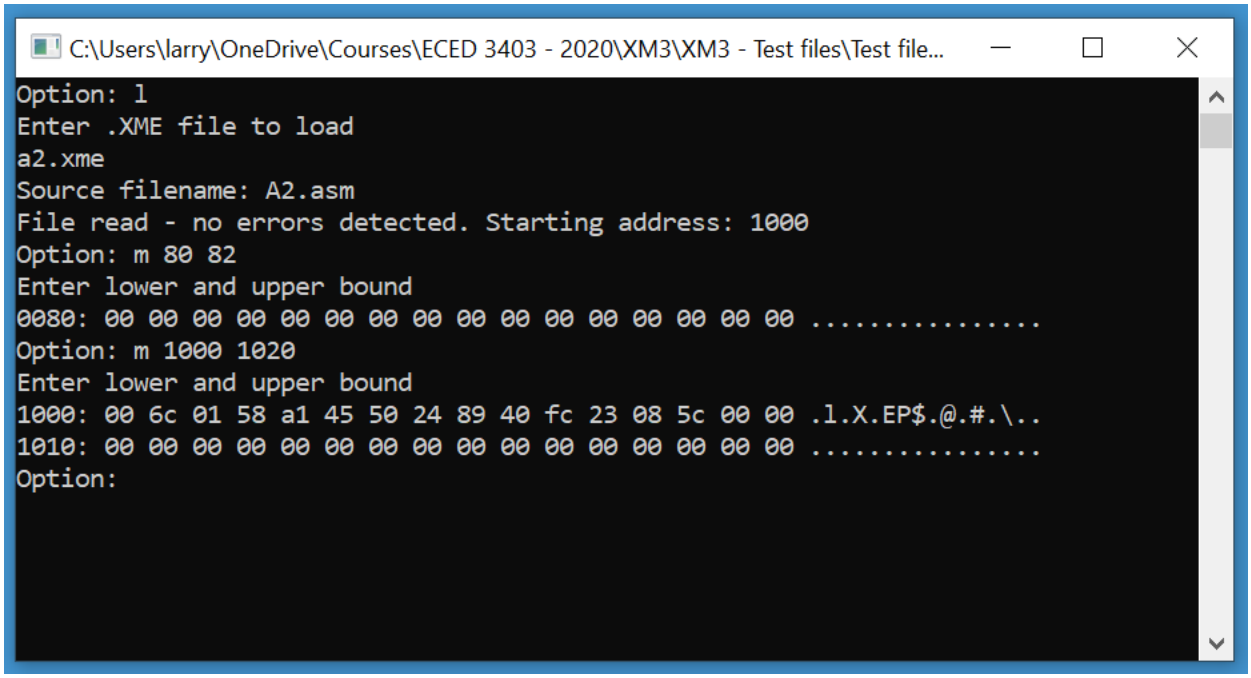
The starting address of the second S1 record is 1000; this is followed by the first two content bytes which contain the values 00 and 6C and are to be stored in locations 1000 (00) and 1001 (6C). However, if we look at the assembler output for location 1000 we see:

```
11      1000    6C00    Start   movlz   Data,R0         ; R0 <- address of Data (#0080)
```

This appears to suggest that 1000 should contain 6C and 1001, 00. This is an example of little-endian format. The value 6C00 has an MSbyte of 6C and an LSbyte of 00. In little-endian, 00 is stored in the low-order memory (1000), while 6C is in the high-order memory (1001). This means that the S1 record reflects the correct byte ordering for a little-endian machine.

The S0 record contains the name of the file (41322E61736D or A2.asm), the two S1 records contain directives to the loader instructing it where to store the data or instructions; and the S9 record indicating where the program is to start execution (location 1000).

An example of what your emulator (containing the debugger, loader, and memory) should appear as is:



A prompt ("Option" in this example) indicates that the emulator is ready for input. The user types '1' and is then asked for the filename; "a2.xme" is entered. The emulator responds with the name of the file (from the XME file) and the starting address:

```
Source filename: A2.asm
File read - no errors detected. Starting address: 1000
```

The user then requests to see the data section (in memory location #80) and the emulator shows 16-bytes in hex (00) and the ASCII contents (unprintable characters are shown as '.'):

```
Option: m 80 82
Enter lower and upper bound
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Then the code section is dumped, starting at #1000 for 32 locations (there is one space between the last memory location displayed in hex and the first memory location displayed as a character:

```
Option: m 1000 1020
Enter lower and upper bound
1000: 00 6c 01 58 a1 45 50 24 89 40 fc 23 08 5c 00 00 .l.X.EP$.@.#.\..
1010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

The output shows the contents of memory after the load; for example, 1000 (00), 1001 (6c), and 1002 (01). These values are taken from the second S1 record which contained 006C01 in the first three bytes of content.

The byte values (not the instructions) are listed to the right. If the value happens to have a printable value, its ASCII equivalent is shown, otherwise it appears as a '.'.

The prompt is shown again, allowing the user to enter one of the supported commands; for example, X:

```
Option: X
```

6 Marking

The assignment will be graded out of 20 using the following marking scheme:

Design Document
The design document is to include an introduction as to the purpose of your software (i.e., your understanding of the problem, essentially an elevator pitch), a description of the algorithms (using tools such as state diagrams or structured-English), and a description of the major data structures required to solve the problem (in a data dictionary).

Total points: 6.

Software
A fully commented, indented, magic-numberless, tidy piece of software that meets the requirements described above and follows the design description. The software must work with the VM described in the labs.

Total points: 10.

System Testing
A set of system tests demonstrating that the software operates according to the design description. The submission must include the name of the test, its purpose or objective, the test configuration, and the test results. The designer of the software is responsible for defining and supplying the tests; sample test files will be supplied.[2]

Total points: 4.

Each part of the assignment must be submitted through Brightspace.

[2] For more information on testing in general, see Welcome to Software Testing Fundamentals and on system testing in particular, see System Testing.

7 Important Dates

Available: 3 June 2020 (00h ADT)

Design document submission: 12 June 2020 (24h ADT)

Software (source only) and testing submission 22 June 2020 (24h ADT)

Late submissions for this assignment will be penalized 1 point-per-day.

8 Miscellaneous

This assignment is to be completed individually.

Do *not* discard this work when completed, as it can be used with the remaining assignments or potentially in future courses.

This assignment is worth 10% of your overall course grade.

It will be necessary to use Revision 3.01 of the assembler to complete this assignment has it has the correct S0 record format. Copies of the assembler are available here.

If you are having *any* difficulty with this assignment or the course, *please* contact Dr. Hughes or one of the TAs, Gary Hubley or Zach Thorne, as soon as possible.