**Part I: Purpose**

X-Makina is a 16-bit load-and-store RISC emulator. To support this emulator, a set of operations on machine memory is required. Including putting program into memory and checking contents of specific memory.

To achieve that, XM3’s 64 KiB of memory, its loader and debugger need to be built. Memory can be reached by bus() function, the loader will load the s-record and put corresponding data into the memory, if there is any error while reading in the file, it has the ability to handle the error and issue the warning. Debugger is to allow user to interact with XM3 emulator, for example: load file (L), access memory (M) and exit the emulator (X).

**Part II: Description of the Algorithms**

**\*MAIN\***

CREATE 64 KiB Memory Array

WHILE DO  
 GET Input Command from Console

IF Input Command is “L” + File name

CALL **LOADXMEFILE** (PASS File name)

CALL **LOADDATA** (PASS File)

ELSE IF Input Command is “M” + Start Address + End Address

CALL **MEMDUMP** (PASS Start AND END Address)

ELSE IF Input Command is “X”

BREAK

ELSE IF Input Command is “project -d fname.dbg”

OPEN fname.dbg File

LOAD file content as Input Commands to Console

CONTINUE

ELSE

ISSUE Warning: Invalid Command

END WHILE

**\*LOADXMEFILE\***

OPEN S-Records file

IF file do not exist

ISSUE Warning: Missing file

ELSE IF CHECK Records are not started with Record Type

ISSUE Warning: File does not contain S-record

ELSE IF Record contains Non Hex characters after Record type

/OR Record length is more than number of bytes after it

ISSUE Warning: Warning, unknown error

ELSE IF CALL **CHKSUM** (PASS the Records)

ISSUE Warning: Incorrect checksum

ELSE

CONTINUE

ENDIF

EXIT

\***CHKSUM**\*

READ last two character as Checksum byte

SUM Hex data in Content AND COMPARE WITH Checksum byte

IF MATCH the Checksum

RETURN TRUE

ELSE  
 RETURN FALSE

ENDIF

EXIT

\***LOADDATA**\*

READ first s-record

WHILE (Not reach the end of file) DO

IF Record Type is S0

GET Source ASM File name

ELSE IF Record Type is S1

LOCATE Start Memory

CALL **MEMBUS** SAVE Content data into Memory Stack in byte

ELSE IF Record Type is S9

GET Starting Address

ENDIF

ENDWHILE

PRINT ASM File name and Starting Address

EXIT

\***MEMBUS**\*

CHECK parameter MAR MBR\*Bidirectional\* RW BW

IF RW equals READ

GET data from memory address MAR

LOAD it into MBR buffer in Byte or Work as BW indicates

ELSE IF RW equals WRITE

GET data from the variable that MBR pointing to

SAVE it into memory stack as MAR address in Byte or Word as BW indicates

ENDIF  
EXIT

\***MEMDUMP**\*

CALL **MEMBUS** PASS Memory Address AND Read Mode

CHECKING ASCII Table

CONVERT Hex Chars\*In Byte\* to ASCII corresponding char

PRINT Address AND Content Data AND corresponding char

**Part III: Major Data Structure**

**\*S-Record\***

XME File = S0 Record + (S1) Record + S9 Record

S-Record = S-Record Type + Record Length + Address + (Content) + Checksum

S-Record Type = [S0 | S1 | S9]

Record Length = One Byte \*Type\* + One Byte \*length\* + Two Bytes \*Address\*+0{hex char}62 \*Content\* + One Byte \*Checksum\*

Address = \*Starting to execute address in memory\*

Content = [\*Data stored in memory stack\* | \*File name\*]

Checksum = \*Check sum all current s-record\*

**\*General Concept: \***

Input Commands = [‘L’ | ‘M’ | ‘X’]

Warning = [“Missing file” | “File does not contain S-records” | “Incorrect checksum” | “Warning, unknown error” | “Invalid Input”]

Memory Stack = 0X0000 {Hex Value} 0XFFFF \*64 KiB memory stack in machine\*

Array = \*Container object that holds a fixed number of values of a single type\*

Program Counter = \*Pointer to current address in memory\*

Register = [R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7]

BUS Parameter = MAR + MBR + RW + BW

MAR = \*Memory address being accessed\*

MBR = \*Bidirectional pointer refers to the address of data\*

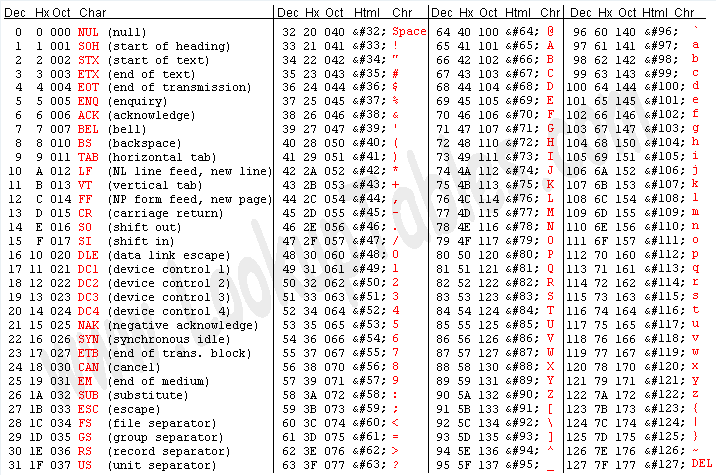
RW = \*Read-write indicator\*

BW = \*Byte-word indicator\*

Byte = \*8 Bit characters\*

Word = \*16 Bit characters\*

ASCII Table =



**\*Data Type**: **\***

Numeric = [“$” + [Unsigned | Signed] | “’” + Char | “#” + Hex]

Unsigned = [0 .. 65535]

Signed = [-32768 .. +0 .. +65535]

Char = [Alphanumeric | Escaped] + “’”

Hex = 1{0 .. 9 | A .. F | a .. f} \* Hex values range from #0 to #FFFF \*

Escaped = “\” + Alphanumeric

Alphabetic = [A..Z | a..z | \_ ]

Alphanumeric = [A..Z | a..z | 0..9 | \_ ]