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*

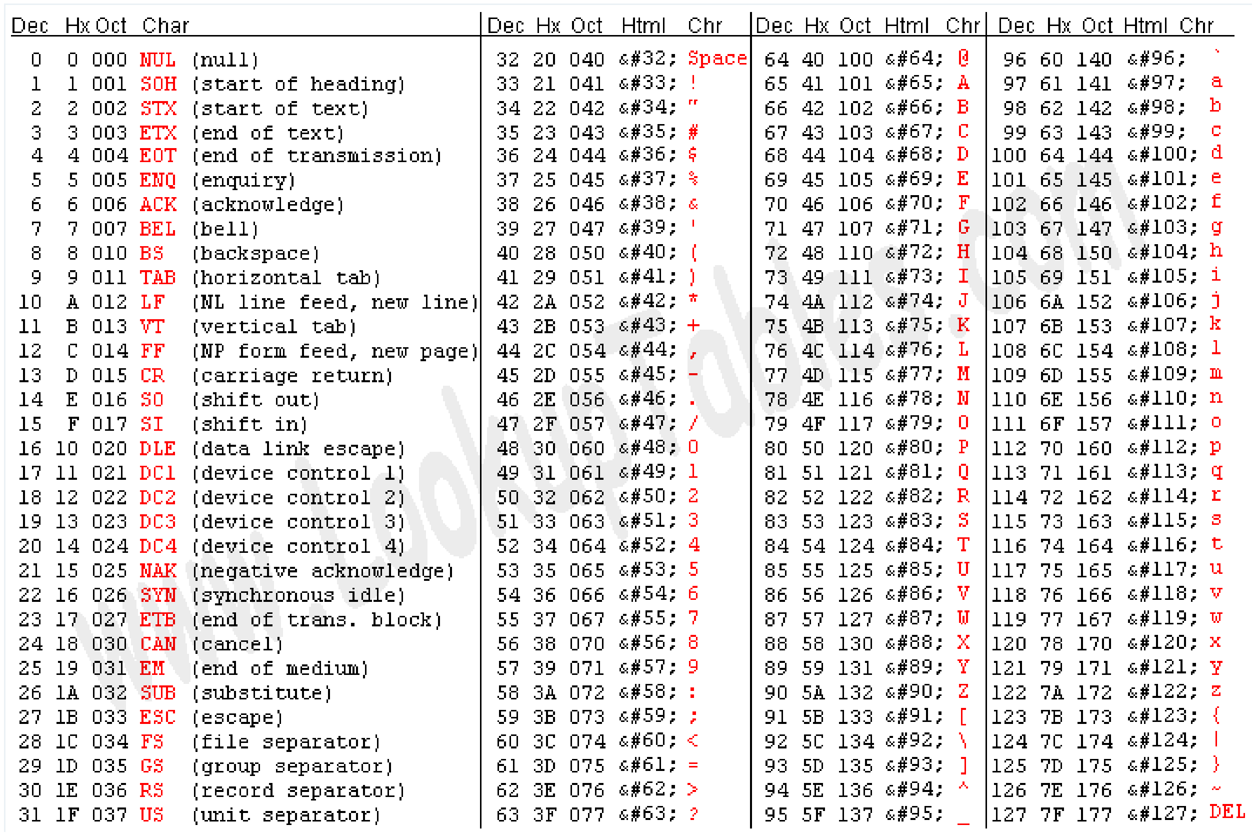
RW = *Read-write indicator*

BW = *Byte-word indicator*

Byte = *8 Bit characters*

Word = *16 Bit characters*

ASCII Table =



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

***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 | _]