

**Sp2009: CSE7000 Computer Science Entreprises Series 7000
Machine Description**

This term we will be using an imaginary digital computer called the CSE7000. The system has been designed and is ready for manufacturing. Before manufacturing the product manager insists that a assembler, linker and simulator be developed to validate the design and to verify that the software is in place and operational before production.

Memory: 4,096 words of 32 bits each. We will limit our implementation to 500 words.

Registers and Index Registers: 9 User Accessible Registers
 • AC - 32-bit Accumulator
 • MQ - 32-bit Multiplier-Quotient
 • XR - 16-bit Index Registers (seven)

'AC' (Accumulator) Register: Used primarily for addition, subtraction, shifting, masking, etc.

'MQ' (Multiplier-Quotient) Register: Used primarily with the AC for multiplication and division. In multiplication, the multiplier was placed in the MQ register and the results were developed in the MQ and the rest in the AC. In division, the MQ would contain the quotient and the AC would retain the calculated remainder.

'XR' (7 Index Registers): The index registers are used for address modification and loop counters. Index registers had four specific associated instruction types: Load an index register (LDXR); store the contents of an index register (STXR); increase or decrease the contents of an index register (DRXR,INCXR); and test the contents of an index register (TSXR, TXRL, TXRP). Index registers could be used for loop counting either by counting down or by counting up.

Non-Addressable Registers

IC 'Instruction Counter' (16-Bits): Holds the address of the next instruction to be executed.

AR 'Address Register' (16-Bits): Holds address of the data or instruction to be read from memory.

IR 'Instruction Register' (32-Bits): Holds current instruction being executed

Arithmetic Unit: Operands are 32 bits in length. Arithmetic is 2's complement, fixed point. Multiplication retains the low 32 bits only of the product. Division yields a quotient only; the remainder is stored in MQ. Attempt to divide by zero LEAVES EVERYTHING UNCHANGED. Overflow LEAVES EVERYTHING UNCHANGED.

Integers: Each 32 bit word, when interpreted as a decimal number is in the range -8,388,608 to 8,388,607. The numbers -2^{31} to $2^{31} - 1$.

Characters: The ASCII (8 bit) codes are used for character information. Each 32 bit word can contain four characters.

Memory Addressing: The machine supports two addressing modes, relative and indirect. If the a-field bit is off the memory address is relative to the start of this program. If the a-field bit is on the memory address is indirect.

Instruction Formats:

label	opc	address_reference
label	opc	address_reference,XR-Reference
Label	opc	=32
Label	opc	none
label	opc	IOWD address_reference,XR,Device

Location or label identifier	Maximum length 24 characters A-Z, a-z, 0-9. The label must start in column 1. All local labels (defined in this program) are relocatable.
Operation field	3 to 7 in length
Variable field	Can consist of 0 to 2 operands
Comment field	Begins with a ':'

Instructions are entered one per line. Major fields are separated by one or more blanks or tabs **or both**. See Appendix A for a full list of instructions.

op code 8 bits	XR 3 bits	Debug 1 bit	Unused 2 bits	Addr-flgs 2 bits	Address portion (s-fld) 16 bits	Total 32 bits
-------------------	--------------	----------------	------------------	---------------------	------------------------------------	------------------

Notation:

Operation Field (op-code) is an 8 bit field corresponding to the alpha instruction. This field normally contains an alphabetic code representing a machine operation or directive. Anything appearing in this field that does not match the entries in the machine and directive operation tables is considered an illegal operation. When this occurs for our CSE560 class we will replace it with a NOP (no-operation) instruction.

XR: indicates which if any of the Index Registers are in use.

Addr-flgs: These flags indicate how the address portion is to be used.

00	relative memory reference	10	immediate literal value
01	indirect memory reference (%)	11	Parameter list reference (#)

Throughout the description of the machine instructions and directives/pseudo-ops the following notation will be used:

C(?) Indicates that the contents of a register or memory location is to be used.

S(X) Indicates that you should use the value of the bit string in the address portion (S-fld) of the instruction + the contents of the index register (pointed to by X).

Note: The instruction name tells us which register is being used. If the instruction contains a Q or MQ the instruction impacts the MQ register, if there is an X it impacts one of the index registers.

Effective Address:

The effective address implied by S(X) is formed by using address portion of the word for the S value and then adding the contents of the X register ($S(X) = C(X) + S$). The result of this calculation may exceed 4096. Since there are only 4096 words of memory, we must perform memory wrap-around. Therefore using modulo arithmetic we can bring the result within 0 to 4095 with memory wrap-around. Index registers work additively: S(X) means that $S+C(X)$ is the address used with the low order 12 bits actually determining the effective address, EFFADDR.

Star Addressing:

Use * as the current instruction address during assembly e.g. ADD * would mean to place the current location counter in the last byte of the instruction.

ADD *+ll would mean to place the result of the arithmetic operation in the last byte of the instruction.

Literals:

The language supports the use of literals to provide a short-cut for the programmer. The literals are imbedded in the instruction in order to reduce latency and enhance performance. There are 4 types of literals: character contained in single quotes, and numeric (bases 2, 10, and 16). Character strings are limited to 2 characters.

Examples:

LD =3H would be 00000003₁₆
 LD =1100B would be 0000000C₁₆
 LD =43 would be 00000029₁₆
 LD ='AB' would be 00024142₁₆

Syntax Checking:

All assemblers must verify that the input it receives is valid. The input must meet all established syntax rules. Some items to consider include: invalid label, invalid op-code, invalid operand, invalid register, undefined variable, multiple definition of symbol or label etc....

Comments:

A line of comment begins with a : in column 1

Directives /pseudo-ops:

The language supports assembler level directives/pseudo-ops to provide information to the assembler about how to define variables, where to start/end the assembly process, and special features. See Appendix B for a complete list.

Expressions: The language supports expression in the operand fields of the EQU and ADR directives. Expressions allow the end user to ask the assembler to performance some basic calculations (+/-) with absolute values, EQU'ed labels, or addresses. In the CSE7000 we will support up to 3 levels of nesting. For example:
X1 EQU 5+2-1 or X2 EQU AB+CD+EF (can not use external references)
X3 adrc mud+oak-dirt (the operand fields can be local, external or a constant)

Address Types: The assembler has the role to share information with the linker so that the linked program does what the programmer had planned. Information is shared via the object file and the associated fields. One of the fields A/R/C defines how the loader should adjust the address portion of the instruction. The codes:
A Absolute tells the loader that the address portion of the instruction does NOT require adjustment.
R Indicates that a simple relocation adjustment is needed by simply adding the address where this program was loaded.
C Indicates that the address needs to be adjusted by the address location that the external symbol has been assigned.
There can be up to 3 entries for each address field. The only time you would use more than one is with an expression operand on a an directive through an expression.

Instruction Syntax: A typical symbolic instruction consists of four divisions: location field or label identifier, operation field, variable fields, and comments. The location field contains a name by which other instructions may refer to the instruction named. The operation field contains the name of the machine operation or directive. The variable field normally contains the location of the operand. The comments field exists so that the programmer can further clarify the intent of the instruction.

For example
CASEB CLA TMFX : Example comment
The label (location reference) is CASEB The machine operation is "Clear and Add."
The memory reference is pointed to by the label TMFX.

Relocation: Most modules are assigned a starting address of zero. Since a program is typically begins at zero this seems correct. Since a job to be executed may contain several modules, it is obvious that they may not all be loaded into memory location zero. In reality no program is loaded at zero. There must be space set aside for the operating system in most cases other independent programs. The actually starting point is determined by the operating system. Based on this all programs need to be relocated. Most of the time all we need to do is add the new starting address to all address fields. There are exceptions due to literals, and absolute values.

Star notation: ALPHA TRA ALPHA+2 is equivalent to *+2

Indirect Addressing Where indirect addressing was allowed, it was specified in the symbolic coding by placing an percent symbol just after the operation code mnemonic (e.g., ADD%). In absolute octal coding, indirect addressing was specified by setting bits 12 and 13 of the instruction word to binary ones. If index register modification was also used, then the indexing occurred first followed by the indirection.

DIRECT ADDRESSING:	ADD DATA1	The memory location is locally defined and relative to the LC starting address of the BEGIN directive.
INDIRECT ADDRESSING:	ADD %DATA1	The memory location contains the address to the address of the real memory value.
INDIRECT ADDRESSING WITH INDEXING:	ADD %DATA1, 2	The contents of the index register are added to the address in the address field. Then this result is used to locate the address of the real memory value.

Subroutine Linkage Linkage (calling functions with arguments and returning with values) depended on the 'TSX' (Transfer and Set Index) instruction. 'TSX' placed the address of the instruction itself into a specified index register and then transferred to the instruction specified by the mem(EFFADD) address. Once inside the called subroutine, incoming variables -- pointers to which were usually strung after the 'TSX', could be easily accessed via the index register and indirect addressing. Returned results could be placed into similarly strung out variables. The return from the called subroutine was easily accomplished with a 'TRA' (Transfer) instruction that specified an offset and the index register.

The following simple example, that calculates $X ** 2 + Y$ and places the result into ANS, illustrates typical subroutine linkage:

Main Procedure		Subroutine
CALL	SUBRT, 4	SUBRT LD #1 :Get X Value by Indirect
RENTRY ...	(Press On)	MPY #1 :Multiply by Mud to Get Mud ** 2 by Same Route
		ADD #2 :Add in Dirt to Get X ** 2
		STO #3 :Store Result in ANS thro
		TRA #4 :Return
TheLIST prmlst	1,4	
Prmid	Mud	
Prmid	Dirt	
Prmid	ANS	
Prmid	reentry	

The steps are as follows:

1. The 'CALL' places the address of the call in XR4 of the parameter list
2. Inside SUBRT, the 'LD' uses XR1 as a pointer to the pointer in the main procedure to X. The pointer is created by the prmlst directive PRMLST XR,n.
3. The 'MPY' fetches the value of X again and multiplies to get $X ** 2$.
4. The 'ADD' fetches the value of Y (the pointer to Y in the main procedure is offset from the address in XR4 by 2 words) and multiplies to get $X ** 2 + Y$.
5. The 'STO' stores the result in ANS. Indirection is not needed as ANS, in the main procedure, is not a pointer but is rather the actual variable. The location of ANS is offset from the address in XR4 by 3 words.
6. The 'TRA' returns back to the main procedure by transferring to the address offset by 4 from the address contained in XR4 (labeled RENTRY).

Predefined variable Names:

MEM	representing memory 0 to 4095
EFFADDR	representing the value of S(X), which equals $S+C(XR(?))$
Addr-field	representing the value of S-field
XR	representing the which index register is in use.
Statusword	representing the status word

Sample Program

SAMPLE	BEGIN	0	:Start Location Counter at zero
	CAL	DEC1	:Load Number
	ADD	DEC2	:Add second number
	STO	SUM	:Store SUM
	STO	SUM+1	
OUT	NOP		: Enter code for I/O
	HALT		:Stop execution
SUM	DEC	0	
	DEC	0	
	FINSH		

SP1 & SP2: Lab Problems

The manufacturer of the CSE7000 plans to furnish with each unit a set of standard software support items. Among them is an assembler, CSE7000-ASM. It is your responsibility to write this assembler for the machine. Since the software is being developed concurrently with the hardware you will not have a prototype CSE7000 to work with. Therefore, you may write CSE7000-ASM on any machine and in any language you prefer (the grader and all your team members must have access).

Input: All input is expected to be in decimal with addresses in either absolute or symbolic form. All labels and variable names are two characters long with the first alphabetic and the second alphabetic or numeric. Blanks or tabs can separate the fields. Source code is entered in the following fixed format:

Fields	Label (optional)
	Operation code (OP-code)
	Variables or operand field
	Comments field

Assembler Output: There are three outputs from the assembler: Assembler source listing, a sorted symbol table, and an object file.

1. **The source listing (see appendix E)**
2. **Object file:** See Appendix C for format specifications.
3. **Sorted Symbol Table (see appendix E)**

Miscellaneous: To verify that all the features of the assembler work you must also run your own test programs. These programs should test: every instruction, all error conditions and any special features.

SP-1 Things to do:

Turn in final documentation (see documentation grading tool)

Turn in ALL test plan runs using the exact same source code

(Note: You have several sample programs in the course documents, but these should not serve as the only programs in your test plan. They need to be included, but your team need to develop quite a few test programs of your own)

Each run must:

- **Print a FORMATTED intermediate file**
- **Print a FORMATTED symbol table**

Email graders and instructor

SP-2 Things to do:

Turn in final documentation

Turn in all test plan runs using the exact same source code

Each run must:

- **Print an assembly listing**
- **Print an Object File**
- **Print the symbol table**
- **Print error messages immediately below the instruction that caused the error**
- **Test plan must also include the program on the next page.**
- **Generation of all error messages**

ASSEMBLER TEST PROGRAM

To test the operation of your CSE7000-ASM use the following program (not checked out yet). The program is supposed to read four pairs of variables XX and YY, and compute the sum of their corresponding quotients, XX/YY.

```
Altest1   BEGIN      0
RD        EQU        0           :EQUATES FOR EASY I/O REFERENCE
RC        EQU        1
PD        EQU        2
PC        EQU        3
          EXTRN      DQ
P1        SARZ                :Clear all registers
LoopStart IOWD      XX,1,RD     :READ NEXT XX
          IOWD      XX,1,PD     :ECHO XX
          IOWD      YY,1,RD     :READ IN NEXT YY
          IOWD      YY,1,PD     :ECHO YY
          LAC       XX,1        :Load AC with XX
          DVH       YY,1        :FORM XX/YY, IF ZERO HALT
          ADQ       QQ,1
          STO       QQ,1        :STORE RESULT
          INCX      1           :INCREMENT INDEX REGISTER
          DRCX      2           :TEST IF 4 TIME THROUGH THE LOOP
          TXL       2,AllDone   :IF NOT DONE, LOOP BACK
          TRA       LoopStart   :IF DONE, STORE AND PRINT RESULT
AllDone   IOWD      M1,0,PC
          IOWD      Q1,PD
          TRA       E1          :THEN GO TO THE EXIT ROUTINE
E1        IOWD      M2,0,PC
          HALT                :HALT
: CONSTANTS AND TEMPORARIES
A1        ADRC      DQ          :ADDRESS OF DQ.
A2        ADRC      XX          :ADDRESS of XX.
C1        DEC       1
C4        DEC       4
M1        ACHR      '  '       :MESSAGE IS:           Q=
          ACHR      '  '
          ACHR      '  '
          ACHR      'Q='
M2        ACHR      'END '
          ACHR      'PROG'
QQ        BSS       4
Q1        BSS       1
XX        BSS       4
YY        BSS       4
          FINSH      Altest1
```

SP3: Lab Problem 3

As programming and checkout nears completion for our Assembler, a loader is the next piece of software to be developed. You are to write a loader (CSE7000-LINKER), which will read in object files produced by the assembler, link these files together, handle relocation, load the resulting executable module into memory and transfer control. CSE7000-LINKER is also to print a load map showing the names and memory location of all the modules and the ENTRY points for each module.

In order to prove that everything works together you must do the following:

1. Write a meaningful CSE7000 Assembler program consisting of at least two modules. Each module must use a label contained in the other module.
2. Run your assembler using these programs and produce an object file.
3. Have your loader read in the object modules and produce a load module to be read in and processed by the simulator.
4. Call your Simulator and have it execute the program.

Format of Load File is in Appendix C.

Things to do:

Turn in all documentation

Turn in all test plan runs using the exact same source code

Each run must:

- **Echo print the hex code as it is read in**
- **Print a Load Map--Loader symbol table**
- **Print a Load File--suited for lab sp4 input**
- **Dump of memory**
- **Print error messages immediately below the instruction that caused the error**
- **Test plan must also include the program on the next page.**
- **Generation of all error messages**

Load Map--Loader symbol table Format

Symbol	Type	Assembler Assigned Location	Program/Module Length	Linker Assigned Location	Adjustment Value
Mud	Pgm_Name	0000	50	0120	120
dirt	Entry_Sym	0020	n/a internal symbol	0140	n/a
Oak	Pgm_Name	0000	40	0170	170
elm	Entry_Sym	0035	n/a internal symbol	01A5	n/a

SP4:Lab Problem 4

You are to write a program that will simulate the operation of the CSE7000 hardware. The simulation should be done at a fairly high level. You do not have to simulate 2's compliment binary arithmetic at the bit level. To do an add you may simply:

$$\text{MEM}(R)=\text{MEM}(R)+\text{MEM}(\text{EFFADDR})$$

The simulator should include error checks and produce appropriate messages. In our simulator no error will halt execution - we must proceed until the end of the program. Exception handling of error's in a proper fashion is our goal, your SP4 program should continue running no matter the error and should handle all errors appropriately. Of course there are always fatal errors (e.g. transferring outside of the range of the program). Some tests to consider are overflow, division by zero, a shift > 32 bits etc. You cannot simply rely on the compiler/loader to catch errors. You must write your own code to identify, report, and take corrective action before the system message would have been generated.

INPUT record is defined in Appendix D.

The simulator should accept the values for the array MEM by reading them in rather than using assignment statements. You must print each line of code read to a file. The simulator should include some error checks and produce appropriate messages.

On the next page is one sample program. This program MUST be included in your test plan and turned in with the lab report. Certainly this test program does not test all possible instructions or all possible error conditions. You must write your own test program (or programs) that will test all commands I didn't and all (ALL) the error conditions. Therefore to complete this assignment you must turn in many runs of the SIMULATOR. (if you can do it all in one run then go ahead). The test must be included in your test plan. If you chose to use it in another part of your documentation as well, that's fine, but it still must appear in the test plan.

Things to do:

Turn in all documentation

Turn in all test plan runs using the exact same source code

Each run must:

Echo print the hex code as it is read in

Dump the array mem before simulation begins

Dump the array mem after simulation is completed

At the start & finish of each instruction being simulated dump

All register contents, op-code, All User Accessible registers, S, EFFADD, LC, and binary version of the instruction (32 bits)

Development of a test program.

Start with assembler symbolic instructions; convert to binary and then to hex. You will have to take responsibility for calculating the location counter by hand. Each instruction takes one location (word). Assume the program is named CSE7000, the length is 38₁₀ and there were no external references.

LOCATION		INSTRUCTION		Instruction						Symbolic Lab 2	
DEC	Hex	Hex		Binary						Text	
04	04	00000011	0000	0000	0000	0000	0000	0000	0001	0001	DEC 17
05	05	00000009	0000	0000	0000	0000	0000	0000	0000	1001	DEC 9
16	10	00000004	0000	0000	0000	0000	0000	0000	0000	0100	LD 4
17	11	48900000	0100	1000	1001	0000	0000	0000	0000	0000	CHS
18	12	40000005	0100	0000	0000	0000	0000	0000	0000	0101	ADD 5
19	13	2000000A	0010	0000	0000	0000	0000	0000	0000	1010	STO 10
20	14	00000004	0000	0000	0000	0000	0000	0000	0000	0100	LD 4
21	15	44000005	0100	0100	0000	0000	0000	0000	0000	0101	DVH 5
22	16	2000000B	0010	0000	0000	0000	0000	0000	0000	1011	STO 11
23	17	00000004	0000	0000	0000	0000	0000	0000	0000	0100	LD 4
24	18	31000005	0011	0001	0000	0000	0000	0000	0000	0101	ANS 5
25	19	2000000C	0010	0000	0000	0000	0000	0000	0000	1100	STO 12
26	1A	00000004	0000	0000	0000	0000	0000	0000	0000	0100	LD 4
27	1B	33000005	0011	0011	0001	0000	0000	0000	0000	0101	LOR 5
28	1C	2000000D	0010	0000	0000	0000	0000	0000	0000	1101	STO 13
29	1D	00000004	0000	0000	0000	0000	0000	0000	0000	0100	LD 4
30	1E	41000005	0100	0001	0000	0000	0000	0000	0000	0101	SUB 1,5
31	1F	2000000E	0010	0000	0000	0000	0000	0000	0000	1110	STO 14
32	20	00000004	0000	0000	0000	0000	0000	0000	0000	0100	LD 4
33	21	39200000	0011	1001	0010	0000	0000	0000	0000	0000	LGSL 0,2
34	22	2000000F	0010	0000	0000	0000	0000	0000	0000	1111	STO 15
35	23	70000024	0111	0000	0000	0000	0000	0000	0010	0101S	TRA 37
36	24	40000005	0100	0000	0000	0000	0000	0000	0000	0101	ADD 5
37	25	FF000000	1111	1111	0000	0000	0000	0000	0000	0000	HALT

Sample Object file

```

H|CES7000|2009:060|12:24:32|0022|0004|0000|0018|0010|CSE7000|0001|0000
T|0004|00000011|R|||CSE7000
T|0005|00000009|R|||CSE7000
T|0010|00000004|R|||CSE7000
T|0011|48900000|R|||CSE7000
T|0012|40000005|R|||CSE7000
T|0013|2000000A|R|||CSE7000
T|0014|00000004|R|||CSE7000
T|0015|44000005|R|||CSE7000
T|0016|2000000B|R|||CSE7000
T|0017|00000004|R|||CSE7000
T|0018|31000005|R|||CSE7000
T|0019|2000000C|R|||CSE7000
T|001A|00000004|R|||CSE7000
T|001B|33000005|R|||CSE7000
T|001C|2000000D|R|||CSE7000
T|001D|00000004|R|||CSE7000
T|001E|41000005|R|||CSE7000
T|001F|2000000E|R|||CSE7000
T|0020|00000004|R|||CSE7000
T|0021|39200000|R|||CSE7000
T|0022|2000000F|R|||CSE7000
T|0023|70000024|R|||CSE7000
T|0024|40000005|R|||CSE7000
T|0025|FF000000|R|||CSE7000
E|001A|CSE7000

```

Appendix A: Instructions

See www.cse.ohio-state.edu/~al Choose Instructions

Appendix B: Directives

	Directive	operand	Purpose	Memory impact
ol	ACHR	'cccc'	ASCII Character	YES
ol	ADRC	Address constant OR EXPRESSION	Address Constant	YES
none	ALT	Label or local address	Updates the execution start address in object file	YES
none	AVERT	Symbol_name	Local label to be shared with other programs	NO
none	BEGIN	?????	Define Beginning of Program Source	NO
ol	BSS	Equated symbol or number in range 0 to 2 ¹⁵ -1	Reserve Block of Storage	YES
ol	CALL	External label located in another program	Link to Externally Defined Subroutine	YES
none	DEBUG	ON/OFF or on/off	Sets debug bit	NO
ol	DEC	-2 ³² to 2 ³² -1	Generate Decimal Data	YES
ol	DUMP	option	1 dump all registers 2 dump active memory 3 dump both	YES
rl	EQU	-2 ³² to 2 ³² -1, previously equated symbol OR EXPRESSION	Define Symbols	YES
none	EXTRN	Symbol_name	External label located in another program	NO
none	FINSH	none	Define End of Source	NO
ol	HEX	'xxxxxxxx'	Hex representation	YES
ol	OCT	'oooooooo'	Generate Octal Data	YES
none	PRMID	Symbol_name	Builds parameter list for call to external routine	YES
rl	PRMLST	XR,n	Establish parameter list	NO

Where:

rl required label
ol optional label
ccc 4 characters requiring 32 bits
adr a valid label or an address in the range of the program
nnnn indicates a integer number (0-4095)
minmax Integer number (-2³¹ to 2³¹-1)
ignored Anything in the field is ignored
none the field is required to be blank
expression An expression using integer constants, equated symbols, local and external labels

Appendix C: Format of Object File

Header Record (1 per assembly)

H		Module name		Date yyyy:ddd		Time hh:mm:ss		program length in HEX (n of wds) hhhh		assembler assigned program load address in HEX hhhh
----------	--	-------------	--	------------------	--	------------------	--	---	--	---

Header Record continued

Number of linking records in H hhhh		Number of Text Records in HEX hhhh		Execution start address for this module hhhh		CSE7000		Version #		Revision #
--	--	---	--	---	--	---------	--	-----------	--	------------

Linking Record (?? per assembly - from Begin, ENTRY directives)

L		Entry Name		Entry Address in HEX		Module Name
----------	--	---------------	--	-------------------------	--	----------------

Text Record (?? per assembly)

T		Address in hex hhhh		Code/data Word hhhh		Relocation Type for S1 A - absolute R - relocatable C - external c		+ OR -		S1 External Reference Symbol Name Used only if reloc type is C		Relocation Type for S2 c
----------	--	---------------------------	--	---------------------------	--	---	--	---------------	--	---	--	--------------------------------

Text Record - continued

+ OR -		S2 External Reference Symbol Name Used only if reloc type is C		Relocation Type for S3 c		+ OR -		S2 External Reference Symbol Name Used only if reloc type is C		Module Name
---------------	--	---	--	--------------------------------	--	---------------	--	---	--	----------------

END Record (1 per assembly)

E		Total Number of Records H+L+T+E in hex hhhh		Module Name
----------	--	--	--	----------------

Appendix D: Load Module Format (Loader output and therefore Simulator Input)

Record LH Linker Header Record

LH		module name from first module		execution start address padr-hhhh		Length of the entire module in HEX hhhh		Initial Program Load Address in HEX padr - hhhh		Date yyyy:ddd
-----------	--	-------------------------------------	--	---	--	--	--	--	--	------------------

Record LH Linker Header Record continued

Time hh:mm:ss		CSE7000 -LINK		Version #		Revision #
------------------	--	---------------	--	-----------	--	------------

Record LT Linker Text Record

LT		load address for the instruction/data in HEX padr - hhhh		code to be loaded in HEX hhhhhhhh		module name from source module for this line of code
-----------	--	---	--	---	--	--

Record LE Linker End Record

LE		Total number of records in this load file
-----------	--	---

Appendix E: Assembly Listing and Sorted Symbol Table

```
LOC  OBJ CODE  A/R/E  STMT  SOURCE STATEMENT
(HEX) (HEX)          (DEC)
.      .
.      .
18     hhhhhhhh  R    25    PT  MPY AA,2      FORM NEXT VALUE.
19     -----  R    26    MUD SUB BB,,,
      error:Warning: too many commas in the operand field
20     -----  R    27    Oak DVH CC
21     -----  -    28    R7  EQU 7
21     -----  R    29    ZZ  HALT
22     -----  A    30    AA  DEC 12
23     -----  A    31    BB  DEC 24
Notice the error message appears below the line with the error.
```

Sorted Symbol Table

Label	address	substitution	usage
AA	22		data
BB	23		data
MUD	19	none	label
Oak	20	none	label
PT	18	none	label
R7	--	7	equ
ZZ	21	none	label