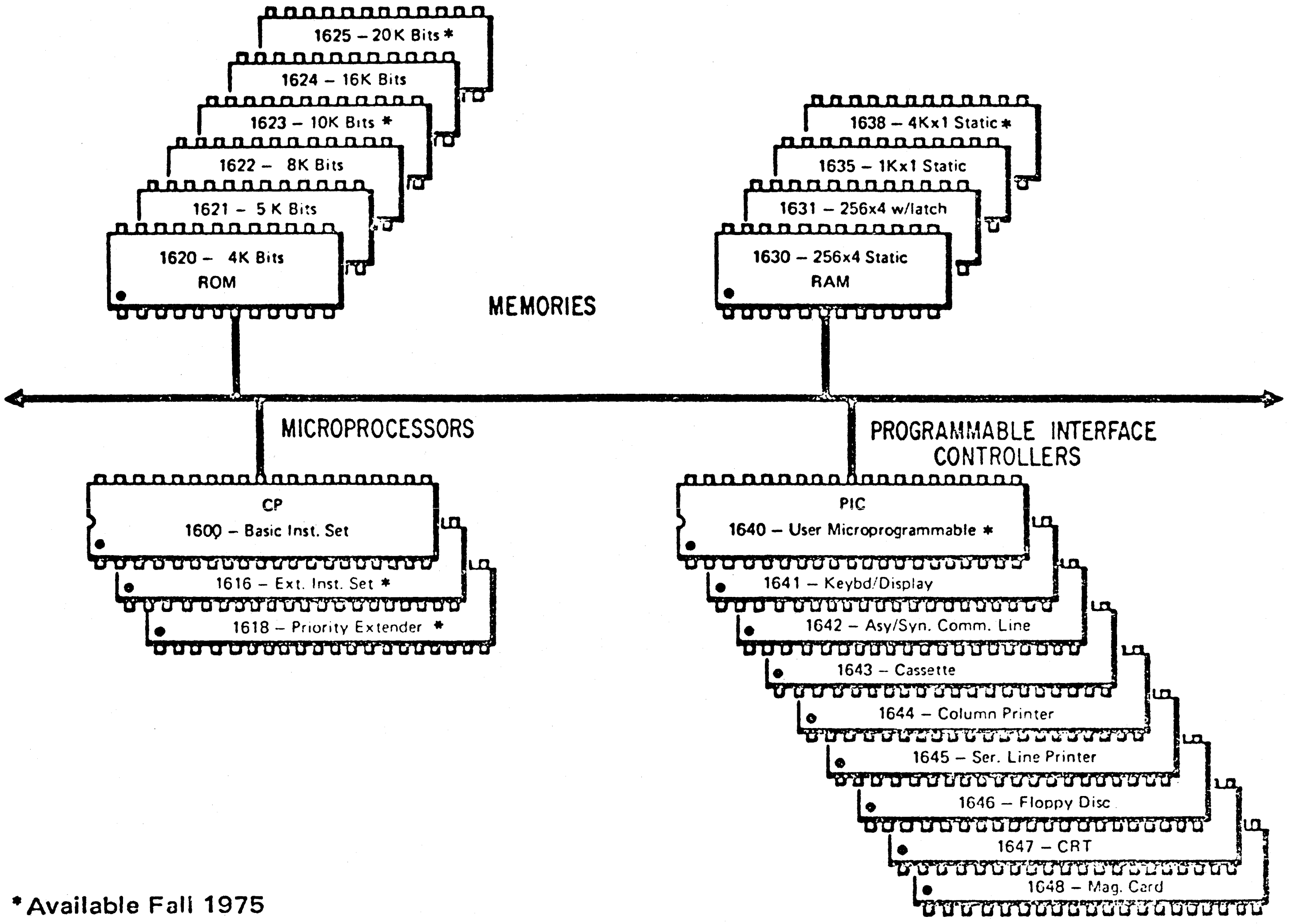
PRICE $8.00

# SERIES 1600
## MICROPROCESSOR SYSTEM

GENERAL INSTRUMENT
MICROELECTRONICS

## SOFTWARE DOCUMENTATION

## THE SERIES 1600 SEMICONDUCTOR LINEUP



*Available Fall 1975

# SERIES 1600 MICROPROCESSOR SYSTEM
# SOFTWARE DOCUMENTATION

CROSS ASSEMBLER/SIMULATOR

USERS MANUAL

This manual contains a detailed specification of the Series 1600 Symbolic Assembly Language and information pertaining to the operating commands, input/output options; and utility functions provided by the Series 1600 Cross Software Package.

GENERAL INSTRUMENT CORPORATION ■ MICROELECTRONICS DIVISION ■ 600 W. John St., Hicksville, N.Y. 11802 ■ 516-733-3107

## SUMMARY

| | |
|---|---|
| Title: | Series 1600 Cross Assembler/Simulator Users Manual |
| Document No.: | S16DOC-XALSIM-02, May 1975 |
| Revision Level: | Supersedes S16DOC-XALSIM-01, November 1974 |
| Subject Programs: | S16XSFT Series 1600 Cross Software Package |

| | | |
|---|---|---|
| a. | S16XAL-1 | Series 1600 Symbolic Cross Assembler |
| b. | S16SIM-1 | Series 1600 Simulator |
| c. | S16LNK-1 | Series 1600 Object Module Linker |
| d. | S16XRF-1 | Series 1600 Concordance (XREF) Generator |
| e. | S16BPT-1 | Series 1600 Binary Paper Tape Generator |
| f. | S16RTG-1 | Series 1600 ROM Tape Generator |

Scope: This manual describes the Cross Assembler (S16XAL-1) and Simulator (S16SIM-1) programs which support the General Instrument Corporation Series 1600 Microprocessor System. In addition, four companion programs, the Object Module Linker (S16LNK-1), Concordance Generator (S16XRF-1), Binary Paper Tape Generator (S16BPT-1), and ROM Tape Generator (S16RTG-1) are described.

Operating Environment: The Series 1600 Cross Software Package is written in F level Fortran IV and is specifically designed to operate in a 16-bit minicomputer environment. Versions are available for the following systems and require 12 to 16K words of memory, a disc operating system, and a terminal device:

| | | |
|---|---|---|
| a. | DEC-PDP11 | S16XSFT-PDP11 |
| b. | DGC-NOVA | S16XSFT-NOVA |

The total S16XSFT Series 1600 Cross Software Package is also available for use on the General Electric Time-Share Computer Network.

Support Programs: Host Computer System Editing Facilities

Reference Documents:

| | |
|---|---|
| S16DOC-SXFT11-00 | Series 1600 Cross Software Operators Guide - PDP11 |
| S16DOC-XSFTNV-00 | Series 1600 Cross Software Operators Guide - NOVA |
| S16DOC-XSFTGE-00 | Series 1600 Cross Software Operators Guide - G.E. Time-Share |

TABLE OF CONTENTS

# CHAPTER 1

# S16XAL SYMBOLIC CROSS ASSEMBLER

## 1.1 INTRODUCTION

The Series 1600 Symbolic Cross Assembler (S16XAL) is a program preparation aid which supports General Instrument's family of 16-bit microprocessors. It translates ASCII coded alphanumeric source programs into several different types of binary coded object modules. The S16XAL Symbolic Cross Assembler is written in F level Fortran IV and is designed for operation in a 16-bit data word environment making it compatible with all minicomputers as well as larger computer systems.

## 1.2 FEATURES

The S16XAL Symbolic Cross Assembler provides the following major features:

- Symbolic language representation of all instructions
- Literal representations in four formats; Octal, Decimal, Hexadecimal, Character
- Arithmetic evaluation of operand expressions
- Assembly directives for
  - Controlling memory allocation
  - Defining character strings
  - Specifying input/output options
  - Establishing conditional assemblies
  - Declaring global and external gymbols
- Assembly in three forms
  - Absolute
  - Relocatable
  - Relocatable/Linking
- Program listings
- Error detection

## 1.3 OPERATION

The S16XAL Symbolic Cross Assembler converts symbolic source programs into machine code format in a two pass process. During the first pass through the source files, all user specified symbols are placed in a symbol table containing the symbol, its value, and several other attributes. During the second pass through the source file, symbolic instruction mnemonics are translated, symbol references resolved, errors diagnosed, a machine code file generated, and an optional program listing produced.

The machine code file produced by the S16XAL Cross Assembler can be of several forms. If the source program specified an absolute assembly, the binary file will be an absolute load module. An absolute load module can be directly loaded and executed by the Series 1600 Simulator (S16SIM) or punched on paper tape for subsequent loading in a microprocessor system by the resident loader (S16LDR). If the source program contains global symbol definitions and/or external symbol definitions and/or external symbol references, the binary file will be a relocatable object module. Relocatable object modules must be linked together by the Series 1600 Object Module Linker (S16LNK) to form one relocatable load module for input to the Series 1600 Simulator (S16SIM). Alternatively, relocatable object modules or load modules may be punched on paper tape for subsequent loading in a microprocessor system by the resident relocatable/linking loader (S16LDR). If the source program does not contain global or external symbol references, the binary file will be a relocatable load module which can be directly loaded and executed by the Series 1600 Simulator (S16SIM) or punched on paper tape for loading on a microprocessor system by the resident loader (S16LDR). These options are shown diagramatically in Fig. 1.3.1.

## 1.4 SOURCE PROGRAM FORMAT

A S16XAL source program is composed of a sequence of statements with each statement contained on a single line. A statement is terminated by a carriage return character or is punched on one computer card. A statement may contain up to four fields which are identified by their order of appearance from left to right. The general format of a S16XAL statement is: Label, Operator, Operand, Comment. The label and comment are optional, while the operator is always required. The presence and nature of the operand depends upon individual operators. It is recommended that statements be limited to approximately 50 characters so that assembled programs can be printed on teletype or CRT terminals.

### 1.4.1 Label

A label is a user defined character string, used to symbolically reference a specific location within a program. If a statement contains a label, the label must begin in the first position of the statement. Labels may contain up to six characters, the first of which must be a letter (A-Z), a currency symbol ($), a question mark (?), or an ampersand (&). The remaining five optional characters may be any valid character (EBCIDIC or ASCII) except a blank space, since this character is the label terminator. Labels containing more than six characters cause a diagnostic to be issued and are truncated after the sixth character. Labels must be unique in the first six characters, i.e., a specific character string cannot be used in the label field of a statement more than once in a program. Multiple use of a label causes a diagnostic to be issued and the subsequent definitions of the label to be ignored.

### 1.4.2 Operator

An operator follows the label field in a statement. A statement operator contains up to four characters and may be an instruction mnemonic or an assembly directive. Instruction mnemonics are symbolic character strings which represent the various Series 1600 instructions. Assembly directives are also symbolic character strings but are used to represent

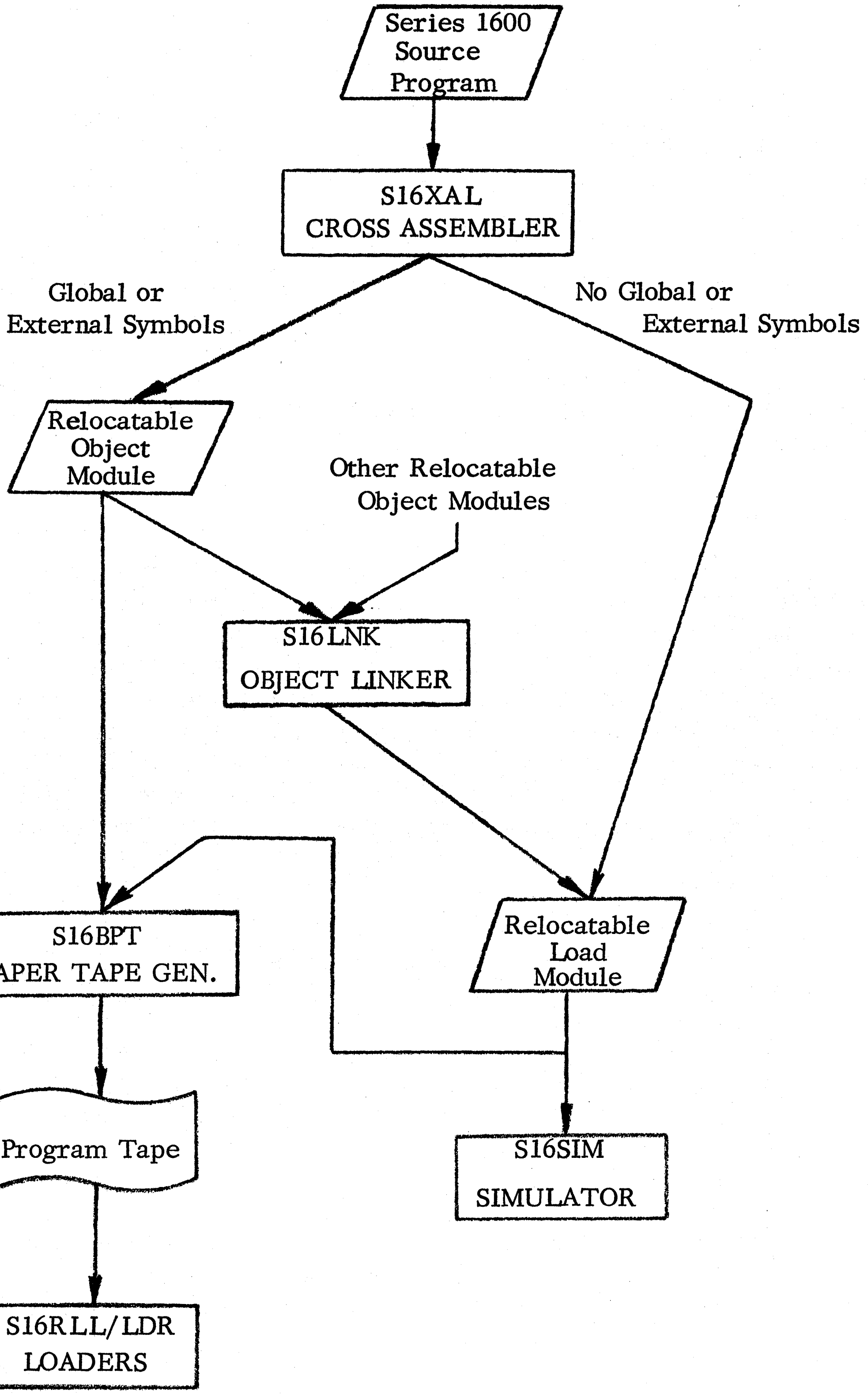

Series 1600 Source Program
S16XAL CROSS ASSEMBLER
Global or External Symbols
No Global or External Symbols
Relocatable Object Module
Other Relocatable Object Modules
S16LNK OBJECT LINKER
S16BPT PAPER TAPE GEN.
Relocatable Load Module
Program Tape
S16SIM SIMULATOR
S16RLL/LDR LOADERS

Figure 1.3.1

certain functions or actions performed by the assembler during the assembly process. If a statement does not contain a label, the operator must be preceded by at least one blank space. If the operator is the last field in a statement, it is followed by a carriage return, otherwise it is followed by a blank space.

### 1.4.3 Operand

An operand follows the statement operator separated by at least one blank space. The operand represents an item or items to be operated upon by the statement operator. Operands may be symbols, literals or expressions. When multiple operands are used, they are separated by commas. If an operand is the last field in a statement it is followed by a carriage return, otherwise it is followed by the comment field.

### 1.4.4 Comment

The comment field is optional in all statements and must be preceded by a semicolon (;). The contents of the comment field are printed on the program listing but have no effect on the assembled program. Entire lines may serve as comments if the first non blank character is a semicolon. Blank lines are printed on the program listing but otherwise ignored so that statements may be separated in order to enhance program readability. The liberal use of commentary is strongly recommended so that the function and operation of programs is evident from the program listing.

## 1.5 SYMBOLS

A symbol is a character string which appears in an operand and represents the value assigned to the symbol. A symbol is given a value by direct assignment via an assembly directive or by appearing in the label field of a statement. Labels are assigned the value of the assembly location counter for the instruction on which they appear. The assembler recognizes the exclamation (!) sign as a special symbol for the current value of the program counter.

## 1.6 LITERALS

Literals are character strings which serve as sources of data, i.e., cannot be changed and are interpreted by the assembler as constants. The assembler accepts literals expressed as octal, decimal, hexadecimal and character. Numeric literals may be preceded by a plus or minus to signify sign. Plus is assumed unless a minus is present.

### 1.6.1 Octal (Default Radix)

| | | |
|---|---|---|
| sooooooo | - | s = optional + or −, + assumed |
| | | o = 0 - 7 |
| | | oooooo = 0 to 177777 |

### 1.6.2 Decimal

| | | |
|---|---|---|
| sdddd | - | s = optional + or −, + assumed |
| | | d = 0 - 9 |
| | | ddddd = − 32768 to 32767 |

### 1.6.3 Hexadecimal

| | | |
|---|---|---|
| sX'hhhh' | - | s = optional + or −, + assumed |
| | | h = 0-9, A - F |
| | | hhhh = 0 to FFFF |

### 1.6.4 ASCII Character

| | | |
|---|---|---|
| "cc" or 'c' | - | "or' = delimiter |
| | | c = any ASCII character |

One or two characters may be packed into each 16-bit word. If one character is specified ("c" or 'c') it is placed in the low order byte of the word with zeros in the high order byte. If two characters are specified ("ab" or 'ab') the first (a) is placed in the low order byte and the second (b) is placed in the high order byte.

## 1.7 EXPRESSIONS

Arithmetic operators (+ and −) may be used to form operand expressions. An element of an expression may be: a user defined symbol, the current assembly location counter symbol (!), or a literal. Expressions may contain up to six elements separated by either + or − operators. The total expression may be terminated by a comma, a carriage return or a semicolon. Expressions are always evaluated from left to right with no parenthetical groupings allowed.

## 1.8 ASSEMBLY DIRECTIVES

Assembly directives are used to control the assembly process and in some cases cause data to be generated. In the following assembly directive descriptions optional elements are enclosed in [ ] . Comments may be used with all assembly directives.

| | LABEL | OP | OPRND | ACTION |
|---|---|---|---|---|
| 1.8.1 | | PAGE | | Advance program listing to the top of the next page. Sixty lines are normally printed on each page. |
| 1.8.2 | | HEAD | 'ccc...c' | Use the character string specified as the operand as a page heading for the next page. The first character in the string is used as the string terminator. |
| 1.8.3 | | REL | [name] | Generate a relocatable assembly and use the six character name as the object module identifier. If no name is specified, an unnamed relocatable object module is generated. The module name is used by the Series 1600 Object Module Linker (S16LNK) to identify object modules on its load module map. The REL directive must be encountered by the S16XAL assembler before any data generating operators are processed. If this is not the case, an informative diagnostic is issued and an unnamed relocatable object module generated. If no REL or ABS directive is specified, an unnamed relocatable object module is generated; i.e., REL is the default assembly mode. |
| 1.8.4 | | ABS | | Generate an absolute assembly; the binary file generated by the S16XAL assembler will be an absolute load module. It cannot be relocated or linked when loaded. The ABS directive must be encountered by the S16XAL assembler before any data generating operators are processed. If this is not the case, an informative diagnostic is issued and an unnamed relocatable object module is generated. |
| 1.8.5 | | ENTR | V | Establish the program entry point; i.e., the point at which execution is to begin. The operand may be either a symbol or a literal. |

| | LABEL | OP | OPRND | ACTION |
|---|---|---|---|---|
| 1.8.6 | | GLOB | S[ , S, . . . , S] | Declare the symbol(s) as global. Global Global symbols must be defined as labels in the current program unit but can be referenced from other program units. |
| 1.8.7 | | EXT | S[, S, . . . , S] | Declare the symbol(s) as external. External symbols reference global symbols in other program units. Both external and global symbol references are resolved by the Series 1600 Object Module Linker (S16LNK) and the relocatable/linking loader (S16LDR) resident in the microprocessor system. |
| 1.8.8 | | ORG | expr | Set the assembly location counter to the value of expr, default is zero. |
| 1.8.9 | SYMBOL | EQU | V | Assign the value of the operand to the symbol. The operand may be a symbol, a literal or the assembly location counter symbol (!). If ! is specified it may be followed by + or – and a literal. |
| 1.8.10 | [LABEL] | RES | expr | Reserve a block of storage whose length is specified by expr. The contents of individual storage locations is undefined. If a label is specified, it is assigned a value equal to the address of the first word in the block. |
| 1.8.11 | [LABEL] | ZERO | expr | Zero a block of storage whose length is specified by expr. If a label is specified, it is assigned a value equal to the address of the first word in the block. |
| 1.8.12 | | BITS | expr | Specify the number of bits in a memory word as the value of expr. The word size is used by the assembler to check generated data for magnitude exceeding word size. The default is 16 bits. |
| 1.8.13 | | MEML | expr1[, expr2] | Specify lower and upper memory address limits as the values of expr1 and expr2. If only expr1 is specified, its value will be used as the upper memory address limit and the lower limit will be set to zero. These limits are used by the assembler to check the validity of addresses assigned to generated code. The defaults are 0 and 17777. |

|  | LABEL | OP | OPRND | ACTION |
|---|---|---|---|---|
| 1.8.14 | [LABEL] | WORD | expr [, expr, . . . , expr] | Generate a data word for each operand expression. The contents of each word is set equal to the value of the respective expr. If a label is specified it is assigned a value equal to the address of the first word. |
| 1.8.15 | [LABEL] | BYTE | expr [, expr, . . . , expr] | Generate two data bytes for each operand expression. The operation is the same as with but 8 bit data is generated for use with double byte addressing in 10 bit memory. |
| 1.8.16 | [LABEL] | TEXT | 'cc. . . .cc' | Generate a word or words of data which contain the seven bit ASCII code for each character. Two characters are packed in each word, low byte to high byte. Incomplete words contain a blank in the high byte. If a label is specified, it is assigned a value equal to the address of the first word generated. |
| 1.8.17 |  | END |  | End of the program, the assembly is terminated on the previous statement |
| 1.8.18 |  | EOT |  | End of tape indicator, used to separate a source program into several paper tapes. This directive is ignored by the file oriented cross assembler. |
| 1.8.19 |  | NLST |  | Disable the program listing. The assembly proceeds normally but with the listing suppressed. This directive is used, for example, to avoid printing a length ZERO block. |
| 1.8.20 |  | LST |  | Enable the program listing. This directive is used to cause a listing to again be produced after a NLST directive. |
| 1.8.21 |  | IFEQ | expr | Start conditional assembly. The statements that follow will be assembled if expr is equal to zero. If expr is not equal to zero, the statements will be listed but not assembled. Conditional assemblies are useful when a program has statements which are to be assembled only under certain conditions. For example, statements which are to be assembled only during debugging of the program. |

| | LABEL | OP | OPERAND | ACTION |
|---|---|---|---|---|
| 1.8.22 | | IFNE | expr | Start conditional assembly. The following statements will be assembled if expr is not equal to zero, the following statements will be listed but not assembled. |
| 1.8.23 | | ENDC | | End conditional assembly, i.e., resume normal assembly. |

## 1.9 PROGRAM LISTING

The S16XAL Symbolic Cross Assembler produces a listing of the assembled program containing the following fields: line number; six octal digits of address; six octal digits of contents; the statement label, operator, operand and comments. The operator, operand and comments are tabulated to enhance program readability. If the assembled word is subject to modification when the program is loaded at an address different than that of assembly; i.e., relocated, the contents are followed by the letter "R". If the assembled word references an external symbol, the contents are followed by the letter "X".

Each page of listing contains sixty lines and begins with a one or two line heading. The first heading line contains the module name, the version of the assembler in use, the time and date of the assembly and the page number. If the user has specified a heading via the HEAD directive, it follows on the next line. The program listing follows, separated from the page heading by a blank line.

At the end of the program listing, all user defined symbols are summarized followed by the number of diagnostics issued. The symbol summary contains each symbol in alphabetical order, its octal value, and its attributes. The following table lists the codes used for symbol attributes:

| | | |
|---|---|---|
| U | - | symbol is undefined |
| A | - | symbol is absolute |
| R | - | symbol is relocatable |
| X | - | symbol is external |
| IN | - | symbols is an instruction label |
| EQ | - | symbol is defined by an EQU statement |
| RS | - | symbol is a RES or ZERO statement label |
| DT | - | symbol is a WORD BYTE, or TEXT statement label |
| | | |
| G | - | symbol is global |
| E | - | symbol is entry point |
| DD | - | symbol is doubly defined |
| UR | - | symbol is unreferenced |

## 1.10 DIAGNOSTICS

The S16XAL Symbolic Cross Assembler performs extensive error checking during program assembly issuing both error and informative diagnostics. Each diagnostic is printed on the program listing on a line immediately preceeding the offending statement. Diagnostics and the associated statement are always listed even though the program listing is surpressed or no listing was requested by the user.

The S16XAL assembly error codes are listed in the following table:

| ERROR | CODE | | MEANING |
|---|---|---|---|
| L (E) | LABEL | - | label illegal or missing |
| D (E) | DBL DEF | - | label is doubly defined |
| U (E) | UNDF SYM | - | reference to undefined symbol |
| M (E) | MDEF REF | - | reference to multi defined symbol |
| O (E) | OP UNREC | - | operator is unrecognized |
| S (E) | SYNTAX | - | syntax illegal |
| R (E) | REGISTER | - | register designator illegal or use of reg illegal |
| C (E) | CHR ILL | - | character is illegal string terminator |
| B (E) | DBL BYTE | - | double byte data sequence illegal |
| P (E) | PHASE | - | label's value differs in phase 2 |
| X (E) | EXT NUM | - | more than one external symbol in expression |
| V (E) | VAL OPRN | - | value of operand illegal |
| N (E) | NUMBER | - | value of literal illegal |
| Q (I) | ? SYNTAX | - | questionable syntax |
| W (I) | WRD SIZE | - | word size exceeded |
| A (I) | ADR/DEST | - | address out of range or destination questionable |
| T (I) | TRUNCATN | - | possible statement truncation |
| E (I) | END ? | - | END directive missing |
| ? (I) | ? USE | - | questional use of directive |
| L (I) | MEM LIMIT | - | memory limits exceeded |

## 1.11 INTERACTIVE DIALOGUE

The S16XAL Assembler is available for use on several interactive computer systems. In order to assemble programs, the user must first estalish communications with the computer and log in with a valid account number or user code (see appropriate System Operation Manuals). After the log in has been completed, the computer operating system responds with a prompt character ($ or * used on many systems). The user responds by entering a command which requests the operating system to load and begin execution of the S16XAL assembler. The assembler first identifies the version in use by displaying S16XAL-VXX, then displays the message: SOURCE FILE, ACCNT?. The user must enter an appropriate source file name followed by a comma and the account in which the file exists. On some systems, if the source file exists in the current account the comma and account may be omitted. Next the assembly listing option is requested by the message: LISTING? (Y/N OR F = NAME). If N is entered, no listing except for diagnostics will be produced, if Y is entered the program listing will be output on the interactive terminal. If F=name is entered the listing will be output to the named file. Finally the object file name is requested by the message: OBJECT FILE?. At the end of the assembly if no terminal listing was requested the number of diagnostics issued is summarized. This concludes the current assembly and another source file is requested for the next assembly. If only a carriage return is entered, the assembler returns control to the operating system.

# CHAPTER 2

# S16LNK OBJECT MODULE LINKER

## 2.1 INTRODUCTION

The Series 1600 Object Module Linker (S16LNK) is a support program to the Series 1600 Symbolic Cross Assembler (S16XAL). It combines or links together two or more relocatable object modules produced by the S16XAL Assembler to form the single relocatable load module necessary for input to the Series 1600 Simulator (S16SIM). The S16LNK Object Module Linker is written in F level Fortran IV and is designed for operation in a 16-bit data word environment making it compatible with all minicomputers as well as larger computer systems.

## 2.2 FEATURES

The S16LNK Object Module Linker performs the following functions on object modules generated by the S16XAL Cross Assembler:

- Resolves global symbol declarations
- Satisfies external symbol references
- Relocates and links multiple object modules into one load module
- Produces a load module memory map
- Generates a load module file for subsequent execution by the S16SIM Simulator

## 2.3 OPERATION

The S16LNK Object Module Linker performs the linkage function by making two passes through the specified object modules. During the first pass, a table of global symbols and their assigned addresses is constructed and a load module memory map is generated. The map lists the object modules in order of linkage, the relocated base address of each object module, all global symbols and their associated addresses in each module, and the size of each module. At the end of the first pass, a linkage summary is produced which indicates the initial address, the final address, and the entry address of the relocatable load module. During the second pass, each object module is again read and the load module file is constructed.

## 2.4 INTERACTIVE DIALOGUE

The S16LNK Object Module Linker operates interactively with the user via any appropriate terminal. Upon program initiation the current version is use is identified and the user is requested to name the resultant linked load module file by the message: LOAD MODULE ?

If a load module file currently exists with the same name, it is deleted and a new file of the same name created. Next, the user is requested to select a load map option by the message: MAP ? (Y/N or F = NAME)

If the user enters "N", no map will be produced; if a "Y" is entered the map will be output on the terminal. If, however, F = NAME is entered the map will be output on a file with the specified name. Finally, the user is requested to enter the names of the object modules to be linked by the message: OBJECT MODULES ?

The user enters each object module file name in response to a prompt ":". If an object module exists in an account which is different than the current account, the account code is entered on the same line immediately after the file name separated by a comma ", ". The end of the object module identification sequence is indicated by entering a null line, i.e., only a carriage return. The S16LNK Object Module Linker then performs the required linkage and responds with the load map if the user so requested, followed by the linkage summary.

## 2.5 ERROR DIAGNOSIS

During the linkage process, S16LNK detects several error conditions and issues the following diagnostic messages:

- FILE DOES NOT EXIST !
  The last object module file specified does not exist, enter another file name.
- OBJECT MODE : x x x x x x NO LONGER EXISTS !
  The indicated object module file has been deleted during linkage, link aborted.
- MODULE : x x x x x x NOT REL, CANNOT LINK !
  The indicated file is not relocatable, i.e., it was declared absolute via the ABS assembly directive, link aborted.
- ! ! MULTIPLE ENTRY DEFINITION : x x x x x x IGNORED
  More than one entry point was specified, link continues.
- CANNOT ACCOMMODATE ANY MORE GLOBALS ! !
  The global symbol table has overflowed, link aborted.
- ! ! MULTIPLE GLOBAL DEFINITION : x x x x x x IGNORED
  A global symbol has been defined more than once, link continues.

- UNSATISFIED EXTERNALS
  SSSSSS NNNNNN

  The indicated external symbol references could not be resolved because corresponding global symbols are not known. NNNNNN is the address of the first word of the instruction containing the external reference. The symbol is assigned value zero and the link continues.

- ILLEGAL LINKAGE CODE

  A non valid linkage code was detected, link aborted.

## 2.6 LIMITATIONS

Since S16LNK is not a program loader, the linked relocatable load module size is limited only by available file space, not by memory size of the host computer.

# CHAPTER 3

# S16SIM SIMULATOR

## 3.1 INTRODUCTION

The Series 1600 Simulator (S16SIM) is a program debugging aid for the Series 1600 Microprocessor System. Its input is either a relocatable or absolute load module derived from the Series 1600 Symbolic Cross Assembler (S16XAL) or from the Series 1600 Object Module Linker (S16LNK). The S16SIM Simulator executes each instruction of a Series 1600 program in a simulated microprocessor environment allowing detailed examination of program flow and dynamic conditions in the simulated system. The S16SIM Simulator is written in F level Fortran IV and is designed for operation in a 16-bit data word environment making it compatible with all minicomputers as well as larger computer systems.

## 3.2 FEATURES

The S16SIM Simulator provides a comprehensive simulation facility for debugging and testing Series 1600 programs before they are executed on an actual actual microprocessor. It provides the following major features:

- Simulation of all Series 1600 instructions
- Simulation of full 65K memory
- Simulation of I/O via data files
- Simulation of external interrupts
- Simulation of external branch conditions
- Simulation of TTY I/O via interactive terminal device
- Execution in run or step mode
- Access to all registers and memory locations
- Trap or breakpoint on register or memory activity
- Trace or monitor registers or memory activity
- Simulate varying memory configurations and speeds
- Display actual program execution time
- Determine actual stack depth used

## 3.3 OPERATION

The user communicates interactively with the S16SIM Simulator using a vocabulary of commands which control the simulation environment and program execution. The user can inspect, change, or monitor microprocessor registers and bus addresses; begin and suspend program execution; execute a program one instruction at a time; simulate real time input/output operations, external interrupts and external conditions and determine actual program execution time for various combinations of ROM/RAM/CPU clock rates.

## 3.4 INITIAL SIMULATOR CONDITIONS

After the S16SIM Simulator is initially loaded and before any user program is loaded or executed the following conditions exist:

> Register 0-7 are set to zero.
> Status bits S, Z, C and OV are set to zero.
> Interrupts are disabled.
> Privileged instruction mode prevails.
> Double byte data is disabled.
> CPU clock rate is 400 nanoseconds.
> External bus accesses are defined at 700 nanoseconds.
> RAM memory is assigned to bus addresses 0-167767 and is initialized at zero.
> TTY simulation is assigned to bus addresses 167775-167777.
> All tables are initialized.

## 3.5 SYSTEM LOADING

The S16SIM Simulator provides simulation of the full 65K address capability of the Series 1600 microprocessor by virtual memory techniques. The user may relocate and load any number of load modules produced by the Series 1600 Symbolic Cross Assembler (S16XAL) or Object Module Linker (S16LNK).

Prior to program execution, the user may redefine memory blocks as Read Only (ROM), modify access times, and change word size. Up to 8 sets of parameters are accommodated. In addition, the upper and lower limits of the memory stack area can be defined. The S16SIM Simulator issues various error messages based on these memory definition values.

During program execution, the S16SIM Simulator generates a pseudo real time execution clock which is used to schedule simulated input/output and interrupt activities. The microprocessor cycle time can be varied from its 400 nsec default value to simulate different system speeds. The memory access times and the microprocessor cycle time determine if delays are required while waiting for memory accesses. If the microprocessor cycle time is 400 nsec, then memory access times from 1-700 nsec produce no wait, from 701-1100 nsec produce 1 wait cycle, 1101-1500 nsec produce 2 wait cycles, etc.

## 3.6 CONTROL COMMAND SEQUENCE

After program load and environment definition, the program can be executed or single-stepped. Appropriate commands may be entered to display registers or memory locations; search memory for bit patterns; or modify registers or memory. A table is kept of the last 10 program execution addresses in order to allow the user to retrace program flow. When the simulator is ready for input commands the message "ENTER COMMAND" is displayed; after the programmer has completed his input he may continue simulation with a Continue (C), Step (S), or Execute (E). The C will remain in either the Step or Execute mode while the S and E will reset the mode.

When in the execute mode the simulation may be interrupted by depressing the "Break" key on the console.

## 3.7 TRAPS AND TRACES

The S16SIM Simulator provides program monitoring capability on CPU registers 0-7 and on any bus address. A trace provides a dynamic display of activities when specified conditions are met. A trap provides a suspension of program execution, i.e., a breakpoint, when specified conditions are met. Since register 7 serves as the CPU program counter, a trap on register 7 serves as a program trap or breakpoint and a trace on register 7 serves as a program flow trace. A memory address trap on a specific address may also serve as a breakpoint when a memory address is accessed during execution. Both traps and traces are inspected after the instruction is executed.

## 3.8 INPUT/OUTPUT AND INTERRUPT SIMULATION

S16SIM providesfor simulation of up to eight input/output devices and up to eight external interrupts. These activities are controlled by tables which specify when such events are to occur relative to real execution time. During program simulation, execution time is accumulated and used to schedule I/O operations and interrupts. This scheduling is relative to zero time, i.e., the start of program execution.

Interrupts are scheduled by defining the start time and the repitition rate. Input/output simulation is provided for through use of data register and status register tables. If a polled or sensed device is to be simulated, it is defined by corresponding entries in both data and status register tables. The status register table defines when and how a device becomes ready, while the data register table defines the data source or destination. If an interrupt driven device is to be simulated it is defined by entries in the interrupt table and the data register table. The eight simulated I/O devices correspond to eight data files named: S16SIMOn, where n is 0 to 7. If input is to be simulated, the corresponding data file must be created by the user beforehand. Output files are created and extended during simulation. When output occurs on a file which already exists, it is deleted and a new file is created with the same name.

## 3.9 EXTERNAL CONDITION SIMULATION

S16SIM provides for simulation of 16 external sensed conditions. In a real system configuration four lines derived from the four lower bits of the BEXT instruction are available at CPU output pins. These signals may be decoded externally to obtain up to 16 test points with the selected one being returned to the CPU for the branch decision. In order to simulate these external sensing activities, a 16 position table is used to control when each of the possible 16 decodes isto be true or false. In a manner similar to I/O and interrupt simulation, the external condition activities are scheduled relative to program execution time accumulated since the start of execution. The external conditions are scheduled by defining a start time and with what repetition rate they are to occur. In addition, the logical condition of individual external conditions when active or ready may be specified as true (high) or false (low). An external condition not defined by a table entry always results in a false evaluation.

## 3.10 TELETYPE SIMULATION

The input/output of ASCII characters from and to an ASR-33 teletype machine or equivalent is simulated without timing via bus addresses 167774-167777. The TTY is simulated during program execution by the user terminal which functions in a character by character mode. All other terminal activities are record, i.e., line oriented.

Bus addresses 167774 and 167776 simulate the TTY keyboard and printer status registers respectively. Input from either of these addresses always returns 000001, indicating device ready. Bus address 167775 simulates the TTY keyboard data buffer register while bus address 167777 simulates the TTY printer data buffer register. Data input from the TTY is always 7 bit ASCII and data output to the TTY must always be 7 bit ASCII. Note that character literal data generated by the S16XAL assembler is 7 bit ASCII compatible with TTY data.

## 3.11 COMMAND STRING FORMAT

S16SIM interactive command strings must comform to a format which always contains a keyword or verb and seven elements which may be optional depending on the nature of the command verb. The general command string format is:

$$\text{VERB, n ; } e_1 \text{ , } e_2 \text{ , } e_3 \text{ , } e_4 \text{ , } e_5 \swarrow$$

The indicated element separators, i.e., commas and semicolon, must be used as shown. Command strings are limited in length to 60 characters and must be complete on one line.

S16SIM recognizes a set of control commands which are entered on the terminal keyboard in response to the "ENTER COMMAND" display. If a command is entered which cannot be recognized, a question mark (?) is displayed under the questionable area of the command and "ENTER COMMAND" is again displayed.

In the following descriptions, required command string elements are underlined, optional elements are not underlined and $\swarrow$ represents a carriage return. Command elements to the right of the semicolon (;) may be symbols from a S16XAL assembly, literals or expressions composed of symbols and literals.

Expression operators may be addition (+) and subtraction (−). Command elements indicated by n which are to the left of the semicolon (;) must be literals. Literals may be octal, decimal, hexadecimal, or ASCII and are expressed as follows:

OCTAL - sooooo (Default Radix)

s = optional + or −, + assumed
d = 0-7
oooooo = 0 to 177777

DECIMAL - s.ddddd

. = decimal indicator
ddddd = -32768 to +37767

HEXADECIMAL - sX'hhhh'

s = optional + or −, + assumed
X' ' = hexadecimal indicator
hhhh = 0 to FFFF

ASCII - 'cc' or "cc"

'or" = delimiters
c - any ASCII character
One or two characters are packed into each 16 bit word.
If one character is specified ("c" or 'c') it is placed in the low order byte of the word with zeros in the high order byte.
If two characters are specified ("ab" or 'ab') the first (a) is placed in the low order byte and the second (b) is placed in the high order byte.

In the following descriptions "ENTER COMMAND" and ":" serve as user prompts, i.e., the user must respond with a keyboard entry. After each command action is completed by S16SIM, the user is again prompted.

## 3.12 LOADING AND SYSTEM PARAMETER DEFINITIONS

### 3.12.1 Load Binary Program

| | |
|---|---|
| ENTER COMMAND<br>: LOAD , n ↓<br>ENTER BINARY FILE NAME,<br>ACCOUNT<br>: name, account ↓ | Load indicated S16XAL generated binary program file at address n, if n is not specified the program will be loaded at the assembly base address. If the file exists in the current account, the account may be omitted. Any number of binary program files may be loaded. |

### 3.12.2 Set Data Display Radix

| | |
|---|---|
| ENTER COMMAND<br>:RADX, n ↓ | Octal radix = 8, decimal radix = 10, ASCII = 0, hexadecimal radix = 16. If no radix is specified, octal is used. |

## 3.12.3 Set Time Limit

| | |
|---|---|
| :TLIM, t↵ | Set program execution time limit of t microseconds. Program execution will be suspended after t microseconds have been accumulated. If t is not specified, the time limit is removed. |

## 3.12.4 Display / Modify CPU Clock Rate

```
ENTER COMMAND
:CLK↵
CLK = nnnn Nsec
:nnnn↵
```

If nnn is entered the Clock rate will be changed. If only a carriage return is entered the clock rate will not be changed.

## 3.12.5 Define Memory

```
ENTER COMMAND
:RAM, n ; addl, addh, t, b, f↵
```

Define a Read/Write Memory Block.

| | |
|---|---|
| n | Position in the memory table for this entry. If not specified, the memory definition is added to the end of the table until eight (0-7) definitions have been accumulated. If more than eight memory definitions are specified, the most recent is replaced. |
| addl | Low bus address assigned to memory segment n. |
| addh | High bus address assigned to memory segment n. |
| t | Memory access time in nanoseconds. If not specified, the current CPU clock rate is used. |
| f | Fill mode, if word size is less than 16 bits. f specifies how data is read out of memory. If f is not specified, mode 0 is assumed. |

0 - right justified, 0's left fill.

1 - right justified, 1's left fill.

2 - right justified, sign bit extension.

3 - left justified, o's right fill.

4 - left justified, 1's right fill.

### 3.12.6 Remove Memory Definition

```
ENTER COMMAND
: RAM, n↙
    or
ENTER COMMAND
: ROM, n↙
```

In the above command strings if n is specified, the corresponding memory definition table entry will be removed. If n is not specified all entries will be removed.

### 3.12.7 Display Memory Blocks

```
ENTER COMMAND
: DMB↙
```

The memory block table will be displayed with all associated definitions.

### 3.12.8 Set Stack Limits

```
ENTER COMMAND
: SLIM ; addl, addh ↙
```

Set the lower stack limit to address add 1 and the upper stack limit to address addh. Program execution is suspended if the stack pointer (R6) exceeds these limits.

### 3.12.9 Remove Stack Limits

```
ENTER COMMAND
: SLIM ↙
```

Remove stack limits.

### 3.12.10 Display Maximum Stack Used

```
ENTER COMMAND
: DMXS↙
MAX STACK = x x x x x x
```

x x x x x x is the maximum value that the stack pointer reached during execution.

## 3.13 BASIC CONTROL COMMANDS

### 3.13.1

```
ENTER COMMAND
: E ; addri, addrf↙
```

Execute a program beginning at location addri and suspend execution when addrf is reached. If no addri is specified, execution begins using the current value of the program counter (register 7). If no addrf is specified, execution continues until: a HALT instruction is executed, a "BREAK" interrupt is issued by the user via the terminal keyboard, the execution time limit is exceeded, or a trap condition is reached.

### 3.13.2 Execute and Display Registers

```
ENTER COMMAND
: EDR ; addri, addrf ↙
```

Same as above, except the registers are displayed after each instruction is executed.

### 3.13.3 Step

```
ENTER COMMAND
: S ; addr ↙
```

Step (one instruction at a time) a program beginning at location addr. If no addr is specified, the program is stepped using the current value of the program counter (register 7).

### 3.13.4 Continue

```
ENTER COMMAND
: C ↙
```

Continue program execution from current location. C is used when in STEP mode, after a TRAP or after a "HLT" instruction.

### 3.13.5 Exit

```
ENTER COMMAND
:X ↙
```

Exit from S16SIM to the operating system Monitor.

### 3.13.6 Keyboard Interrupt

"BREAK"

One depression of the "BREAK" Key generates an interrupt to the S16SIM executive. This interrupt can be used during simulation to suspend program execution or to suspend extended terminal outputs requested by DA or SA commands. If the interrupt is issued during program execution, "BREAK" will be displayed followed by the ENTER COMMAND request. If the interrupt is issued during a non execution activity, the activity will be suspended and the ENTER COMMAND request will be displayed.

### 3.13.7 Display Elapsed Time

```
ENTER COMMAND
: DET ↙
```

Display program execution time in microseconds.

## 3.13.8 Display Registers

ENTER COMMAND
: DR ↙

Display the current contents of: Registers 0-7; status register bits S, Z, C, OV; INTFF and NONINT.

## 3.13.9 Display Addresses

ENTER COMMAND
: DA ; addr1, addrh ↙

Display the current contents of bus locations addr1 through addrh inclusive. If addr1 and addrh are not specified all bus locations are displayed.

## 3.13.10 Search Addresses

ENTER COMMAND
: SA ; addr1, addr2, value, mask ↙

Search bus locations addr1 through addr2 inclusive for value using mask to "and" out corresponding bits. If value and/or mask are not specified, 0 and 177777 are used.

## 3.13.11 Initialize Addresses

ENTER COMMAND
: IA, addr1, addr2, value, mask ↙

Initialize bus locations addr1 through addr2 inclusive using value and mask to set bits. If value and/or mask are not specified 0 and 177777 are used.

## 3.13.12 Display/Modify Addresses

ENTER COMMAND
: A ; addr ↙
nnnnnn = nnnnnn
:nnnnnn ↙

Open bus address addr for display and modification.
If no ; addr is specified, address 0 is opened. if the contents of the currently open address is to be changed, the new value is entered followed by optional "x" and a carriage return. If no change is to be made, only optional "x" and a carriage return is entered. The character "x" represents either / or + and −. These characters are used to indicate if the next address (/ or +) or the previous address (−) is to be opened after the current location is closed. If no "x" is specified the current location is closed.

## 3.13.13 Display/Modify Registers

```
ENTER COMMAND
: R , n↵
Rn = nnnnnn
: nnnnn↵
```

Open registers n (0-7) for display and modification. If no n is specified, R0 is opened. If the contents of the currently open register is to be changed, the new value is entered followed by optional "x" and a carriage return. If no change is to be made, only optional "x" and a carriage return is entered. The character "x" represents either / or + and –. These characters are used to indicate if the next register (/ or +) or the previous register (–) is to be opened after the current register is closed.

## 3.13.14 Display/Modify Status Register

```
ENTER COMMAND
: SR↵
S = N
: n
Z = N
: n
C = N
: n
C = N
: n
OV = N
: n
```

Open status register S, Z, C, OV bits for display and modification.
After the current value of each bit is displayed (N), a new value (n = 0 or 1) may be entered. If no change is to be made, only a carriage return is entered.

## 3.13.15 Branch Destination Modification

```
ENTER COMMAND
: MB ; addr1, addr2↵
```

Modify branch instruction at addr1 and addr1 + 1 for destination addr2.

## 3.13.16 Jump Destination Modification

```
ENTER COMMAND
: MJ ; addr1, addr2↵
```

Modify jump instruction at addr1, addr + 1 and addr1 + 2 for destination addr2.

## 3.13.17 Display Symbols

```
ENTER COMMAND
: DSYM↵
```

Display the current symbol table.

### 3.13.18 Display Previous Addresses

ENTER COMMAND
: DPA ↙

Display the 10 most recent program counter (R7) values.

### 3.13.19 Clear Previous Addresses

ENTER COMMAND
:CPA ↙

Clears the 10 most recent program counter (R7) values.

### 3.13.20 Set Software Interrupt Vector

ENTER COMMAND
: SIN ; addr ↙

Specify SIN (software interrupt) vector address. If a SIN is executed and no vector address is specified, the instruction is executed as a NOP. The current vector can be deactivated by entering only the command.

## 3.14 TRAP/TRACE COMMANDS

### 3.14.1 Memory Trap or Trace

ENTER COMMAND
: MT , n ; addr , mode, f, v, m ↙

Set memory address trap or trace.

| | |
|---|---|
| n = 0-7 | MT number; if not specified each MT will be accumulated until MT 7 has been specified. Subsequent MT entries will then be accumulated by replacing the current MT 7 entry. |
| addr | address at which a trap (breakpoint) is to be set or which is to be traced. |
| mode = 0 | trap when addr is referenced. (Default condition) |
| 1 | trace when addr is referenced. |
| 2 | trap when contents of addr are changed. |
| 3 | trace when contents of addr are changed. |
| f | frequency of occurence, i.e., the number of times the specified condition will be satisfied before the trap or trace is activated. Default value 1. |
| v | value of contents of addr which will activate trap or trace. Default, unconditional trap or trace. |
| m | Mask used to extract a value to be compared with V. Default value 177777. |

### 3.14.2 Register Trap or Trace

ENTER COMMAND

| | |
|---|---|
| : RT, n ; mode, f, v, m↙ | Set register trap or trace. |
| n = 0-7 | Register number. |
| mode = 0 | trap when register n is referenced. (Default condition) |
| 1 | trace when register n is referenced. |
| 2 | trap when contents of register n are changed. |
| 3 | trace when contents of register n are changed. |
| f | frequency of occurrence, i.e., the number of times the specified condition will be satisfied before the trap or trace is activated. Default value 1. |
| v | value of register n which will activate trap or trace. Default, unconditional trap or trace. |
| m | mask used to extract a quantity to be compared with v. Default, 0177777. |

### 3.14.3 Remove Register Trap or Trace

ENTER COMMAND

| | |
|---|---|
| : RT , n↙ | Remove Register n trap or trace. If n is not specified, all register traps or traces are removed. |

### 3.14.4 Remove Memory Trap or Trace

ENTER COMMAND

| | |
|---|---|
| : MT , n↙ | Remove memory address trap or trace n. If n is not specified, all memory address traps or traces are removed. |

### 3.14.5 Display Register Traps or Traces

ENTER COMMAND

| | |
|---|---|
| :DRT, n↙ | Display register trap or trace table entry n. If n is not specified all entries in the table are displayed. |

### 3.14.6 Display Memory Traps or Traces

ENTER COMMAND

| | |
|---|---|
| :DMT, n↙ | Display memory address trap or trace table entry n. If n is not specified all entries in the table are displayed. |

## 3.15 INPUT/OUTPUT COMMANDS

### 3.15.1 Define I/O Buffer Register

| | |
|---|---|
| ENTER DOMMAND | |
| :IOBR, n;addr, d↙ | Define an I/O buffer register. |
| n | Entry in buffer register table. If not specified the buffer register is added to the end of the table until eight (0-7) definitions have been accumulated. If more than eight buffer registers are defined, the most recent is replaced. |
| addr | Buffer register bus address. |
| d | The direction of data transfer, 0 = output, 1 = input. If not specified output is assumed. |

### 3.15.2 Remove I/O Buffer Register Definition

| | |
|---|---|
| ENTER COMMAND | |
| :IOBR, n↙ | Remove buffer register definition n. If n is not specified, all buffer register definitions are removed. |

### 3.15.3 Display I/O Buffer Register Table

| | |
|---|---|
| ENTER COMMAND | |
| :DIOB↙ | Display the I/O Buffer Register Table. |

### 3.15.4 Define I/O Status Register

| | |
|---|---|
| ENTER COMMAND | |
| :IOSR, n;addr, r,time, rate↙ | Define an I/O status register. |
| n | Entry in status register table. If not specified the status register is added to the end of the table until eight (0-7) definitions have been accumulated. If more than eight status registers are defined the most recent is replaced. |
| addr | Status register bus address. |
| r | The contents of the status register when ready. No ready is the complement of r. If r is not specified, zero is used. |
| time | Time of the initial ready state in microseconds. If not specified the status register will become ready after the first instruction is executed. |
| rate | Rate at which the status register subsequently becomes ready in microseconds after (time). If not specified the status register will ready after the next instruction. If rate is equal to zero the status register will ready only once (time). |

## 3.15.5 Removal I/O Status Register Definition

| | |
|---|---|
| ENTER COMMAND<br>:IOSR, n↙ | Remove status register definition n. If n is not specified, all buffer register definitions are removed. |

## 3.15.6 Display I/O Status Register Table

| | |
|---|---|
| ENTER COMMAND<br>:DIOS↙ | Display I/O Status Register Table. |

## 3.15.7 Define Interrupt

| | |
|---|---|
| ENTER COMMAND<br>: INT, n;addr, time, rate↙ | Define an interrupt. |
| n | Entry in interrupt table. If not specified, the interrupt definition is added to the end of the table until eight (0-7) definitions have been accumulated. If more than eight interrupt definitions are specified, the most recent is replaced. The position in the table represents the relative interrupt priority, i.e., 0-7 with 7 having the highest priority. |
| addr | Interrupt vector address. |
| time | Time of the initial occurrence of interrupt in microseconds. If not specified the interrupt will occur immediately after the execution of the first instruction which enables interrupts. |
| rate | Time of interrupt re-occurrence in microseconds. If not specified the interrupt will occur at the maximum rate possible, i.e., after every instruction which enables interrupt. If rate is equal to zero the interrupt will occur only once at (time). |

## 3.15.8 Display Interrupt Table

| | |
|---|---|
| ENTER COMMAND<br>: DINT↙ | Display Interrupt Table. |

## 3.15.9 Remove Interrupt Definition

| | |
|---|---|
| ENTER COMMAND<br>: INT , n↙ | Remove interrupt definition n. If n is not specified, all interrupt definitions are removed. |

## 3.16 EXTERNAL CONDITION COMMANDS

### 3.16.1 Define External Condition

| | |
|---|---|
| ENTER COMMAND<br>:EXT, n;ext, time, rate↙ | Define an external condition. |
| n | External condition (0-15) which corresponds with an appropriate BEXT instruction. If n is not specified, each EXT definition is accumulated starting with EXTO until EXT 15 has been defined. Additional definitions will then replace EXT 15. |
| ext | The logical condition of the external condition when ready, 0 = false, 1 = true. When not ready the external condition is in the complement condition. |
| time | Time of the initial ready state in microseconds. If not specified, the external condition will become ready after the first instruction is executed. |
| rate | Rate at which the external condition subsequently becomes ready in microseconds after (time). If not specified the external condition will remain ready. If rate is equal to zero, the external condition will ready only once at (time). |

### 3.16.2 Remove External Condition Definition

| | |
|---|---|
| ENTER COMMAND<br>:EXT, n↙ | Remove external condition definition n. If n is not specified, all external condition definitions are removed. |

### 3.16.3 Display External Conditions Table

| | |
|---|---|
| ENTER COMMAND<br>:DEXT↙ | Display the External Condition Table. |

## 3.17 INFORMATIVE MESSAGES

During program debugging and simulation using S16SIM, certain conditions may occur which cause one or more of the following messages to be displayed on the terminal:

UNDEF BUS ADDR = x x x x x x x PC = YYYYYY

A reference to an undefined bus address has been detected. Program execution is suspended.

ILL ROM WRITE BUS ADDR = x x x x x x BUS DATA = YYYYYY PC = ZZZZZZ

An attempt to write into a bus address defined as read only memory has been detected. Program execution is suspended.

INVALID ADDRESS, LOAD ABORTED

A program load resulted in reference to an undefined memory address. The load is terminated at that address.

"HLT" AT x x x x x x

A halt instruction has been executed, program execution is suspended.

"BREAK"

The "BREAK" Key on the user terminal was depressed during program execution. Program execution is suspended.

REGn TRAP CONTENTS = x x x x x x PC = YYYYYY

REGn TRAC CONTENTS = x x x x x x PC = YYYYYY

A reference to a CPU register which has been specified in the register trace and trap table has been detected. A trace allows program execution to continue, while a trap suspends program execution.

MEM ADDR TRAP n AT x x x x x x DATA = YYYYYY PC = ZZZZZZ

MEM ADDR TRAC n AT x x x x x x DATA = YYYYYY PC = ZZZZZZ

A reference to a memory address which has been specified in the memory address trap and trace table has been detected. A trace allows program execution to continue, while a trap suspends program execution.

"SDBD" PRECEDES NON DBL BYTE DATA INSTR, PC = x x x x x x

The double byte data flip flop was set during execution of a non double byte data instruction. The flip flop is cleared and program execution continues.

STAK LIMIT VIOLATION, PC = x x x x x x x SP = x x x x x x

The stack pointer, R6, has exceeded the currently specified stack limits. Program execution is suspended.

"TIME LIMIT" AFTER x x x x x x x USEC

The user specified simulated execution time limit has been exceeded. Program execution is suspended.

EIS & DIS SET ON JSR OR J INSTR AT x x x x x x IGNORED

Both interrupt enable and disable bits were detected set during execution of a JSR or J instruction. Interrupts are unchanged and program execution continues.

CANNOT RELOCATE ABS BINARY FILE, LOAD ABORTED

The user has requested load relocation on a binary file generated by an absolute assembly.

DEVICE DATA EXHAUSTED AFTER n INPUTS BUS = x x x x x x PC = YYYYYY

More data has been requested from a simulated input device than the device data file contains. Program execution is suspended.

BUS ADDRx x x x x x CONFLICT

An attempt to redefine a bus address which is already defined. The current bus address definition is not affected.

MEMORY BLOCK FROM x x x x x x TO YYYYYY CONFLICTS WITH PREVIOUS DEFINITION

An attempt to redefine a memory block which is already defined. The current memory block definition is not changed.

DATA WORD SIZE, ADDR = x x x x x x, DATA = YYYYYY, PC = ZZZZZZ

An attempt to write into memory with data which is greater in magnitude than the defined memory word size. Program execution is suspended.

OUTPUT TO TTY OF NON ASCII CHAR x x x x x x PC = YYYYYY

An output to the simulated teletype at bus address 177777 of a non ASCII character has been detected. The character is replaced by an ASCII blank and program execution continues.

INPUT FROM TTY OF NON ASCII CHAR x x x x x x PC = YYYYYY

A non ASCII character has been received from the simulated teletype at bus address 167775. The character is replaced by an ASCII blank and program execution continues.

OUTPUT TO INPUT BUFFER REG x x x x x x BUS DATA = YYYYYY PC = ZZZZZZ

An output to a buss address defined as a simulated device input buffer register has been detected. Program execution is suspended.

INPUT FROM OUTPUT BUFFER REG x x x x x x PC = YYYYYY

An input from a buss address defined as a simulated device output buffer register has been detected. Program execution is suspended.

# CHAPTER 4

# S16XRF CONCORDANCE (XREF) GENERATOR

## 4.1 INTRODUCTION

The Series 1600 Concordance Generator program (S16XRF) provides the user with a concordance or cross reference listing of Series 1600 assembly language programs. A cross reference listing consists of program statement symbols and all references to each symbol. Cross reference listings are useful when debugging and modifying large programs.

## 4.2 OPERATION

S16XRF produces a symbol cross reference listing of Series 1600 assembly language source program by reading the source file and noting all symbols and references. A listing is then produced which indicates each symbol, the line number on which each symbol is defined and all references to each symbol by ascending line number. The symbols are listed in alphabetical order down the page while the line numbers of all references are listed across the page.

## 4.3 INTERACTIVE DIALOGUE

Upon initial startup, S16XRF identifies the version in use and then requests identification of the assembly language source file for which the cross reference listing is to be produced by displaying: SOURCE FILE, ACCNT? The user must then enter the name of an appropriate file followed by a comma and the file account or user code. If the file exists in the current account, the comma and account may be omitted. Next, the user is requested to specify a listing file by the message: LIST FILE? If the user enters a file name, the symbol cross reference listing will be output to the named file. If no file is named, i.e., only a carriage return is entered, the listing will be output on the user's terminal.

## 4.4 ERROR MESSAGES

If the named source file does not exist as specified, the message "FILE DOES NOT EXIST!" is displayed and another source file is requested.

If the named source file contains more symbols than can be assommodated by the version of S16XRF in use, the message "TOO MANY SYMBOLS AT LINE # nnnn!" is displayed. In this case the cross reference listing will not contain symbol references on or after the indicated line number.

## 4.5 LISTING FORMAT

The symbol cross reference listing which is produced by S16XRF contains symbols in alphabetical order and references to each symbol in numerical order as follows:

| SYMBOL | LINE # | REFERENCES | | | | |
|---|---|---|---|---|---|---|
| MAX | 25 | 35 | 89 | 602 | 702 | 703 |
| NUMBER | 30 | 208 | | | | |
| QTY | 51 | 2 | 3 | 201 | | |
| Z | 98 | 95 | 205 | 208 | 352 | |

## 4.5 LIMITATIONS

S16XRF-1 (Version 1) can accommodate 300 symbols and 1200 symbol references. These limitations are related to the size of the memory available on the particular host computer in use.

# CHAPTER 5

# S16BPT BINARY PAPER TAPE GENERATOR

## 5.1 INTRODUCTION

The Series 1600 Binary Paper Tape Generator program (S16BPT) is used to punch on paper tape an object module file produced by the Series 1600 Symbolic Cross Assembler (S16XAL) or a load module file produced by the Series 1600 Object Module Linker (S16LNK). A paper tape is required when a program which has been assembled and/or linked on a host computer system is to be loaded and executed on a Series 1600 microprocessor system. The format of the tape produced is compatible with the Series 1600 Relocating Linking Loader.

## 5.2 OPERATION

Upon initial startup (via appropriate host system Run or Execute command) S16BPT identifies the current version in use and requests object file identification by the message: "ENTER BINARY FILE NAME, ACCNT". The user must enter the name of an appropriate Series 1600 binary file followed by a comma and the file's account identification. On many systems if the file exists in the current account, the comma and account may be omitted. Next, the user is requested to ready the paper tape punch and acknowledge when ready. When the paper tape punching is complete, the user is requested to enter another file to be punched; if only a carriage return is entered, control is returned to the host computer operating system monitor.

## 5.3 ERROR MESSAGES

Several error conditions are detected by S16BPT and are reported to the user by the following messages:

FILE DOES NOT EXIST!! - the named object file does not exist, enter another file name.

INVALID OBJECT CODE SEQUENCE, TAPE ABORTED!! - the named object file contains erroneous or invalid object data, tape aborted.

# CHAPTER 6

# S16RTG ROM TAPE GENERATOR

## 6.1 INTRODUCTION

The Series 1600 ROM Tape Generator program (S16RIG) produces data from which Read Only Memories for the CP-1600 microprocessor are fabricated. A load module produced by the Series 1600 Cross Assembler (S16XAL) or the Series 1600 Object Module Linker (S16LNK) is converted to data patterns which are input to General Instrument Corporation's automated ROM manufacturing facility for processing.

## 6.2 OPERATION

Upon initial startup (via appropriate host system RUN or EXECUTE command), S16RTG identifies the current version in use and requests load module identification by displaying "LOAD MODULE NAME, ACCT?". The user must enter the file name of the load module which is to be placed on a ROM followed by a comma and the file's account identification. On many host systems, if the file exists in the current account, the comma and account identification may be omitted. Next, the user is requested to enter a program relocation address by the message: "RELOCATION ADDRESS?". S16RTG includes a program relocation facility so that the load module need not be assembled at the actual ROM addresses. The user is next requested to specify a name for the ROM data pattern file which is to be produced by the display: "ROM PATTERN FILE NAME?". The user must enter a suitable file name of one to eight characters, with the first alphabetic (A-Z) and the rest alphanumeric (A-Z, 0-9). S16RTG produces a EBCDIC or ASCII sequential ROM data pattern file of 129 eighty character records which may be transferred to punched cards, magnetic tape or paper tape for transmittal to General Instrument Corporation for ROM processing. Next, the user is requested to enter a ROM pattern number by the message: "ROM PATTERN NUMBER?". This three digit number is used to identify the ROM patterns during processing and must be obtained from General Instrument Corporation prior to producing the ROM pattern file. Finally, the user is requested to specify the particular ROM base address by the message: "ROM BASE ADDRESS?". This address is the one at which S16RTG will commence producing data patterns and must be equal to or greater than the relocation address and have the three least significant octal digits equal to 000. The 512x10 bit ROMs are assigned addresses xxx000 - xxx777. When file generation is complete, another load module file name is requested. If a name is entered the process is repeated, if, however only a carriage return is entered, control is returned to the host computer operating system monitor.

## 6.3 ERROR MESSAGES

Several error conditions are detected by S16RTG and are reported to the user by the following messages:

STRING ERROR!! - The load module file name, account specification contains an error.

FILE DOES NOT EXIST!! - The specified file does not exist.

FILE NAME CONFLICT!! - The specified ROM pattern file name is the same as the specified load module file name.

NON SEQUENTIAL ADDRESSING IN ROM, ZEROS USED IN UNDEFINED ADDRESSES - The load module contains a RES or an ORG assembly directive, resulting in unspecified contents for ROM locations. It is recommended that the user use the ZERO assembly directive for specifying unused ROM locations.

LOAD MODULE CONTAINS LINKAGE INFORMATION, ROM ABORTED!! - The specified load module was not a load module, but probably an object module.

ABS LOAD MODULE, CANNOT RELOCATE, ROM ABORTED!! - The user has requested relocation of an ABS assembly load module.

# APPENDIX A

## SAMPLE S16XAL ASSEMBLY

The listing shows a sample assembly performed by the S16XAL Symbolic Cross Assembler. The program is a generalized code conversion utility routine. A number of intentional errors have been incorporated into the source program to show the error diagnostics of the S16XAL Assembler.

```
                ;;;;;;;
                ;
                ;  SOURCE FILE = INASCSR
                ;  OBJECT FILE = INASCOB
                ;
                ;;;;;;;
                        REL    INASC
                        HEAD   '.. ASCII TO BINARY CONVERSION ..'
                        PAGE
```

```
                      ;;;;;;
                      ;
                      ;  ASCII TØ BINARY CØNVERSIØN RØUTINE
                      ;
                      ;   HEXBIN - HEXADECIMAL ASCII TØ BINARY
                      ;   INTBIN - INTEGER ASCII TØ BINARY
                      ;   ØCTBIN - ØCTAL ASCII TØ BINARY
                      ;   BINBIN - BINARY ASCII TØ BINARY
                      ;
                      ;  CALLING SEQUENCE:
                      ;
                      ;   R1 = INPUT FIELD BASE ADDRESS
                      ;   R2 = # CHARACTERS TØ BE CØNVERTED
                      ;   JSR R5,NAME
                      ;
                      ;   R0 = CØNVERTED BINARY VALUE
                      ;   R1 = PØINTER TØ END ØF CØNVERSIØN
                      ;   R2 - R4 DESTRØYED
                      ;
                      ;  CØNVERSIØN TERMINATES ØN FIRST NØN NUMERIC
                      ;  CHARARACTER ENCØUNTERED.  LEADING SPACES ARE
                      ;   IGNØRED.  LEADING + ØR - ARE HANDLED.
                      ;
                      ;;;;;;
** L  (E)  LABEL      !
                              EQU    0
                      R1      EQU    1
                      R2      EQU    2
                      R3      EQU    3
                      R4      EQU    4
                      R5      EQU    5
** S  (E)  SYNTAX                    !
                      SP      EQU
                      PC      EQU    7
                              GLØB   HEXBIN,INTBIN,ØCTBIN,BINBIN
                              PAGE
```

```
.. ASCII TO BINARY CONVERSION ..

  44   000000 001274   HEXBIN  MVII   16,R4       ;RADIX 16
       000001 000020
** M   (E)  MDEF SYM                      !
  45   000002 001000           B      ASC1
       000003 000012
** Q   (I)  ? SYNTAX                   !
** S   (E)  SYNTAX                                !
  46   000004 001270   INTBIN  MVII   10R4        ;RADIX 10
       000005 000012
** M   (E)  MDEF SYM                      !
  47   000006 001000           B      ASC1
       000007 000006
  48   000010 001274   OCTBIN  MVII   8,R4        ;RADIX 8
       000011 000010
** M   (E)  MDEF SYM                      !
  49   000012 001000           B      ASC1
       000013 000002
** Q   (I)  ? SYNTAX                   !
  50   000014 001274   BINBIN  MVII   2 R4        ;RADIX 2
       000015 000002
** D   (E)  DBL DEF        !
  51   000016 000223   ASC1    MOVR   R2,R3
** D   (E)  DBL DEF        !
  52   000017 000223   ASC1    MOVR   R2,R3       ;# CHRS
  53   000020 001165           PSHR   R5          ;SAVE RETURN
** U   (E)  UNDF SYM                   !
  54   000021 000700           CLRR   R0          ;INIT BIN ACCUM
  55   000022 000755           CLRR   R5          ;INIT STR STRT FLG
  56   000023 001212   ASC2    MVI@   R1,R2       ;PICK UP CHR
  57   000024 000255           TSTR   R5          ;STR STRT YET ?
  58   000025 001014           BNZE   ASC4        ;YES, NO LDNG CHRS
       000026 000020
  59   000027 001572           CMPI   ' ',R2      ;SPC ?
       000030 000040
  60   000031 001004           BEQ    ASC7        ;YES, BYPASS
       000032 000063
  61   000033 001572           CMPI   '-',R2      ;MINUS ?
       000034 000055
  62   000035 001014           BNEQ   ASC3        ;NO, CHK FOR PLUS
       000036 000003
  63   000037 000025           DECR   R5          ;YES, SET MINUS FLG
  64   000040 001000           B      ASC7        ;BYPASS MINUS
       000041 000054
** R   (E)  REGISTER                   !
  65   000042 000010   ASC3    INCR   55          ;SET PLUS FLG
  66   000043 001572           CMPI   '+',R2      ;PLUS ?
       000044 000053
  67   000045 001004           BEQ    ASC7        ;YES, BYPASS IT
       000046 000047
** S   (E)  SYNTAX                                !
  68   000047 001470   ASC4    SUBI   060         ;STRIP ASCII MASK
       000050 000060
  69   000051 001013           BMI    ASCFIN      ;NON DIG, TRMN CNVRT
       000052 000047
  70   000053 001502           CMP    021,R2      ;CHK FOR A-F
       000054 000021
  71   000055 001005           BLT    ASC5        ;NOT
```

```
,. ASCII TØ BINARY CØNVERSIØN ..

      000056 000006
** S  (E)  SYNTAX                       !
** Q  (I)  ? SYNTAX                     !
  72  000057 001570          CMPI  026,,R2
      000060 000026
** Q  (I)  ? SYNTAX                         !
  73  000061 001016          BGT   ASCFIN  NØN DIG, TRMN CNVRT
      000062 000037
  74  000063 001472          SUBI  07,R2       ;ADJ A-F => 10-15
      000064 000007
  75  000065 000542  ASC5    CMPR  R4,R2       ;CMPR DIG & # BASE
  76  000066 001015          BGE   ASCFIN      ;NØN DIG, TRMN CNVRT
      000067 000032
  77  000070 001574          CMPI  10,R4       ;CHK FØR DEC CNVRT
      000071 000012
  78  000072 001004          BEQ   ASC10
      000073 000033
** U  (E)  UNDF SYM                   !
  79  000074 000130          SLLC  R0          ;MULT ACCUM BY 2
  80  000075 001001          BC    ASCFIN
      000076 000023
  81  000077 001574          CMPI  2,R4        ;CHK FØR BASE 2
      000100 000002
  82  000101 001004          BEQ   ASC6
      000102 000012
** U  (E)  UNDF SYM                   !
  83  000103 000134          SLLC  R0,2        ;MULT ACCUM BY 8
  84  000104 001001          BC    ASCFIN
      000105 000014
  85  000106 001574          CMPI  8,R4        ;CHK FØR BASE 8
      000107 000010
  86  000110 001004          BEQ   ASC6
      000111 000003
** U  (E)  UNDF SYM                   !
  87  000112 000130          SLLC  R0          ;MULT ACCUM BY 16
  88  000113 001001          BC    ASCFIN
      000114 000005
** U  (E)  UNDF SYM                     !
  89  000115 000320  ASC6    ADDR  R2,R0       ;INSRT CURRNT DIG
  90  000116 000011  ASC7    INCR  R1          ;INCR CHR STR PTR
  91  000117 000023          DECR  R3          ;CHK FØR ALL CHRS CNVRT
  92  000120 001054          BNZE  ASC2        ;NØT, GET NXT CHR
      000121 000076
** Q  (I)  ? SYNTAX        !
  93  000122 000255  ASCFINNNTSTR  R5          ;CHK SIGN FLG
  94  000123 001003          BPL   ASCXIT      ;PLUS
      000124 000001
** U  (E)  UNDF SYM                   !
  95  000125 000040          NEGR  R0          ;MINUS
  96  000126 001267  ASCXIT  PULR  PC          ;EXIT
  97  000127 001162  ASC10   PSHR  R2          ;SAVE CURR DIG
** U  (E)  UNDF SYM                   !
  98  000130 000202          MØVR  R0,R2       ;MULT ACCUM BY 10
** U  (E)  UNDF SYM                   !
  99  000131 000134          SLLC  R0,2
 100  000132 001041          BC    ASCFIN
      000133 000011
```

```
** U  (E)  UNDF SYM                              !
 101  000134 000420          SUBR   R2,R0
** U  (E)  UNDF SYM                          !
 102  000135 000130          SLLC   R0
 103  000136 001041          BC     ASCFIN
      000137 000015
** U  (E)  UNDF SYM                              !
 104  000140 000420          SUBR   R2,R0
** U  (E)  UNDF SYM                          !
 105  000141 000130          SLLC   R0
 106  000142 001041          BC     ASCFIN
      000143 000021
 107  000144 001262          PULR   R2          ;GET CURR DIG
 108  000145 001040          B      ASC6        ;INSRT DIG
      000146 000031
 109  000146                 END
```

```
,. ASCII TO BINARY CONVERSION ..

ASCFIN 000122  R IN                      ASCXIT 000126  R IN
ASC1   000016  R IN          DD          ASC10  000127  R IN
ASC2   000023  R IN                      ASC3   000042  R IN
ASC4   000047  R IN                      ASC5   000065  R IN
ASC6   000115  R IN                      ASC7   000116  R IN
BINBIN 000014  R IN G    UR              HEXBIN 000000  R IN G    UR
INTBIN 000004  R IN G    UR              OCTBIN 000010  R IN G    UR
PC     000007  A EQ                      R1     000001  A EQ
R2     000002  A EQ                      R3     000003  A EQ
R4     000004  A EQ                      R5     000005  A EQ
SP     000000  U EQ      UR


  21 SYMBOLS
  28 DIAGNOSTIC(S)  23 ERROR(S)   5 INFORMATIVE(S)

SOURCE FILE:SAMPLE    BINARY FILE:S0
```

APPENDIX B

## INSTRUCTION SET

REGISTER - REGISTER

| MNEMONIC | OPERAND | CYCLES | INSTRUCTION | | | DESCRIPTION | STATUS CHANGE |
|---|---|---|---|---|---|---|---|
| MOVR | SSS, DDD | 6 * | 0010 | SSS | DDD | MOVe contents of Register SSS to register DDD. *If DDD is 6 or 7 add 1 to Cycles. | S, Z |
| TSTR | SSS | 6 * | 0010 | SSS | SSS | TeST contents of Register SSS. *If SSS is 6 or 7 add 1 to Cycles. | S, Z |
| JR | SSS | 7 | 0010 | SSS | 111 | Jump to address in Register SSS. (Move address to Register 7). | S, Z |
| ADDR | SSS, DDD | 6 | 0011 | SSS | DDD | ADD contents of Register SSS to contents of register DDD. Results to DDD | S, Z, C, OV |
| SUBR | SSS, DDD | 6 | 0100 | SSS | DDD | SUBtract of Register SSS from contents of register DDD. Results to DDD | S, Z, C, OV |
| CMPR | SSS, DDD | 6 | 0101 | SSS | DDD | CoMPare Register SSS with register DDD by subtraction. Results not stored. | S, Z, C, OV |
| ANDR | SSS, DDD | 6 | 0110 | SSS | DDD | logical AND contents of Register SSS with contents of register DDD.Results to DDD | S, Z |
| XORR | SSS, DDD | 6 | 0111 | SSS | DDD | eXclusive OR contents of Register SSS with contents of register DDD.Results to DDD | S, Z |
| CLRR | DDD | 6 | 0111 | DDD | DDD | CLeaR Register to zero. | S, Z |
| INCR | DDD | 6 | 0000 | 001 | DDD | INCrement contents of Register DDD. Results to DDD | S, Z |
| DECR | DDD | 6 | 0000 | 010 | DDD | DECrement contents of Register DDD. Results to DDD | S, Z |
| COMR | DDD | 6 | 0000 | 011 | DDD | one's COMplement contents of Register DDD. Results to DDD | S, Z |
| NEGR | DDD | 6 | 0000 | 100 | DDD | Two's complement contents of Register DDD. Results to DDD | S, Z, C, OV |
| ADCR | DDD | 6 | 0000 | 101 | DDD | ADd Carry bit to contents of Register DDD. Results to DDD | S, Z, C, OV |

REGISTER SHIFT Executable only with Register 0, 1, 2, 3.
Shift Right instructions set the S flip-flop with Bit 7 of the result after the instruction.
Add 2 cycles if shift is 2 bits or two bytes.
Shifts are not interruptable.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| SWAP | RR<n> | 6 | 0001 | 000 | NRR | N = 0, SWAP bytes of register RR. S equals Bit 7 of results of SWAP. | S, Z |
| | | 8 | | | | N = 1, SWAP bytes of register RR, then swap them back to original form. | S, Z |
| SLL | RR<n> | 6 | 0001 | 001 | NRR | N = 0. Shift Logical Left one bit, zero to low bit. | S, Z |
| | | 8 | | | | N = 1, Shift Logical Left two bits, zero to low 2 bits. | S, Z |
| RLC | RR<n> | 6 | 0001 | 010 | NRR | N = 0, Rotate Left one bit using Carry bit as bit 16. | S, Z, C |
| | | 8 | | | | N = 1, Rotate Left two bits using C as bit 17 and OV as bit 16. | S, Z, C, OV |
| SLLC | RR<n> | 6 | 0001 | 011 | NRR | N = 0, Shift Logical Left one bit using C as bit 16, zero to low bit. | S, Z, C |
| | | 8 | | | | N = 1, Shift Logical Left two bits using C as bit 17, OV as bit 16, zero to low 2 bits. | S, Z, C, OV |
| SLR | RR<n> | 6 | 0001 | 100 | NRR | N = 0, Shift Logical Right one bit, zero to high bit. | S, Z |
| | | 8 | | | | N = 1, Shift Logical Right two bits, zero to high two bits. | S, Z |
| SAR | RR<n> | 6 | 0001 | 101 | NRR | N = 0, Shift Arithmetic Right one bit, sign bit copied to high bit. | S, Z |
| | | 8 | | | | N = 1, Shift Arithmetic Right two bits, sign bit copied to high bits. | S, Z |
| RRC | RR<n> | 6 | 0001 | 110 | NRR | N = 0, Rotate Right one bit using Carry as bit 16. | S, Z, C |
| | | 8 | | | | N = 1, Rotate Right two bits using C as bit 16, OV as bit 17. | S, Z, C, OV |
| SARC | RR<n> | 6 | 0001 | 111 | NRR | N = 0, Shift Arithmetic Right one bit, thru Carry, sign bit copied to high bit. | S, Z, C |
| | | 8 | | | | N = 1, Shift Arithmetic Right two bits, thru Carry and OV, sign bit copied to high 2 bits | S, Z, C, OV |

## APPENDIX B

## INSTRUCTION SET (continued)

BRANCHES The Branch instructions are Program Counter Relative, i.e., the Effective Address = PC+Displacement. PPPPPPPPPP is the Displacement and S is 0 for +, 1 for −. If Memory is greater than 10 bits then the appropriate number of lead bits pppppp will be a part of the Displacement. For a forward branch an addition is performed; for a backward branch a ones complement subtraction is performed. Computation performed on PC+2.

| MNEMONIC | OPERAND | CYCLES | INSTRUCTION | DESCRIPTION | STATUS CHANGE |
|---|---|---|---|---|---|
| B | DA | 7/9 | 1000 S0 0000 | Branch unconditional, Program Counter Relative (+1025 to -1024) | |
| | | | pppppp PPPP PP PPPP | | |
| NOPP | | 7 | 1000 S0 1000 | No OPeration, two words | |
| | | | pppppp PPPP PP PPPP | | |
| BC | DA | 7/9 | 1000 S0 0001 | Branch on Carry. C = 1 | |
| BLGT | DA | | pppppp PPPP PP PPPP | Branch if Logical Greater Than. C = 1 | |
| BNC | DA | 7/9 | 1000 S0 1001 | Branch on No Carry. C = 0 | |
| BLLT | DA | | pppppp PPPP PP PPPP | Branch if Logical Less Than. C = 0 | |
| BOV | DA | 7/9 | 1000 S0 0010 | Branch on OVerflow. OV = 1 | |
| | | | pppppp PPPP PP PPPP | | |
| BNOV | DA | 7/9 | 1000 S0 1010 | Branch on No OVerflow. OV = 0 | |
| | | | pppppp PPPP PP PPPP | | |
| BPL | DA | 7/9 | 1000 S0 0011 | Branch on PLus. S = 0 | |
| | | | pppppp PPPP PP PPPP | | |
| BMI | DA | 7/9 | 1000 S0 1011 | Branch on MInus. S - 1 | |
| | | | pppppp PPPP PP PPPP | | |
| BZE | DA | 7/9 | 1000 S0 0100 | Branch on ZEro. Z = 1 | |
| BEQ | DA | | pppppp PPPP PP PPPP | Branch if EQual. Z - 1 | |
| BNZE | DA | 7/9 | 1000 S0 1100 | Branch on No ZEro. Z = 0 | |
| BNEQ | DA | | pppppp PPPP PP PPPP | Branch if Not EQual. Z = 0 | |
| BLT | DA | 7/9 | 1000 S0 0101 | Branch if Less Than. $S \veebar OV = 1$ | |
| | | | pppppp PPPP PP PPPP | | |
| BGE | DA | 7/9 | 1000 S0 1101 | Branch if Greater than or Equal. $S \veebar OV = 0$ | |
| | | | pppppp PPPP PP PPPP | | |
| BLE | DA | 7/9 | 1000 S0 0110 | Branch if Less than or Equal. $Z \vee (S \veebar OV) = 1$ | |
| | | | pppppp PPPP PP PPPP | | |
| BGT | DA | 7/9 | 1000 S0 1110 | Branch if Greater Than. $Z \vee (S \veebar OV) = 0$ | |
| | | | pppppp PPPP PP PPPP | | |
| BUSC | DA | 7/9 | 1000 S0 0111 | Branch if Unequal Sign and Carry $C \veebar S = 1$ | |
| | | | pppppp PPPP PP PPPP | | |
| BESC | DA | 7/9 | 1000 S0 1111 | Branch if Equal Sign and Carry $C \veebar S = 0$ | |
| | | | pppppp PPPP PP PPPP | | |
| BEXT | DA, E | 7/9 | 1000 S1 EEEE | Branch if EXternal condition is True. Field E is externally decoded to select 1 of 16 conditions. Response is tested for true condition. | |
| | | | pppppp PPPP PP PPPP | | |

APPENDIX B

INSTRUCTION SET (continued)

CONTROL

| MNEMONIC | OPERAND | CYCLES | INSTRUCTION | DESCRIPTION | STATUS CHANGE |
|---|---|---|---|---|---|
| GSWD | DD | 6 | 0000 110 0DD | Get Status WorD in register DD. Bits 0-3, 8-11 set to 0. Bits 4, 12 = C; 5, 13 = OV; 6, 14 = Z; 7, 15 = S. | |
| NOP | <n> | 6 | 0000 110 10N | No operation. | |
| SIN | <n> | 6 | 0000 110 11N | Software Interrupt; pulse to PCIT * pin | |
| RSWD | SSS | 6 | 0000 111 SSS | Restore Status Word from register SSS; Bit 4 to C. Bit 5 to OV, Bit 6 to Z, Bit 7 to S. | S, Z, C, OV |
| HLT | | 4 | 0000 000 000 | HaLT after next instruction is executed. Resume on control start. | |
| EIS | | 4 | 0000 000 010 | Enable Interrupt System. Not Interruptable. | |
| DIS | | 4 | 0000 000 011 | Disable Interrupt System. Not Interruptable. | |
| TCI | | 4 | 0000 000 101 | Terminate Current Interrupt. Not Interruptable. | |
| CLRC | | 4 | 0000 000 110 | CLeaR Carry to zero. Not Interruptable. | C |
| SETC | | 4 | 0000 000 111 | SET Carry to one. Not Interruptable. | C |

JUMP

| | | | | | |
|---|---|---|---|---|---|
| J | DA | 12 | 0000 000 100<br>11AA AAA A00<br>AAAA AAA AAA | Jump to address. Program counter is set to 16 bits of A's. | |
| JE | DA | 12 | 0000 000 100<br>11AA AAA A01<br>AAAA AAA AAA | Jump to address. Enable interrupt system. Program counter is set to 16 bits of A's. | |
| JD | DA | 12 | 0000 000 100<br>11AA AAA A10<br>AAAA AAA AAA | Jump to address. Disable interrupt system. Program counter is set to 16 bits of A's. | |
| JSR | BB, DA | 12 | 0000 000 100<br>BBAA AAA A00<br>AAAA AAA AAA | Jump and Save Return address (PC+3) in register designated by 1BB. Program counter is set to 16 bits of A's. BB≠11 | |
| JSRE | BB, DA | 12 | 0000 000 100<br>BBAA AAA A01<br>AAAA AAA AAA | Jump and Save Return and Enable interrupt system. Return (PC+3) is saved in register 1BB. Program counter is set to 16 bits of A's. BB≠11 | |
| JSRD | BB, DA | 12 | 0000 000 100<br>BBAA AAA A10<br>AAAA AAA AAA | Jump and Save Return and Disable interrupt system. Return (PC+3) is saved in register 1BB. Program counter is set to 16 bits of A's. BB≠11 | |

DIRECT ADDRESSED DATA - MEMORY

Field aaa aaa is dependent on the width of memory. 16 bits is maximum for aaa aaa AAAAAAAAAA.

| MNEMONIC | OPERAND | CYCLES | INSTRUCTION | DESCRIPTION | STATUS CHANGE |
|---|---|---|---|---|---|
| MVO | SSS, A | 11 | 1001 000 SSS<br>aaa aaa AAAA AAA AAA | MoV Out data from register SSS to address A - A. | |
| MVI | A, DDD | 10 | 1010 000 DDD<br>aaa aaa AAAA AAA AAA | MoVe In data from address A - A to register DDD. | |
| ADD | A, DDD | 10 | 1011 000 DDD<br>aaa aaa AAAA AAA AAA | ADD data from address A - A to register DDD. Results to DDD. | S, Z, C, OV |
| SUB | A, DDD | 10 | 1100 000 DDD<br>aaa aaa AAAA AAA AAA | SUBtract data from address A - A from register DDD. Results to DDD. | S, Z, C, OV |
| CMP | A, SSS | 10 | 1101 000 SSS<br>aaa aaa AAAA AAA AAA | CoMPare data from address A - A with register SSS by subtraction. Results not stored. | S, Z, C, OV |
| AND | A, DDD | 10 | 1110 000 DDD<br>aaa aaa AAAA AAA AAA | logical AND data from address A - A with register DDD. Results to DDD. | S, Z |
| XOR | A, DDD | 10 | 1111 000 DDD<br>aaa aaa AAAA AAA AAA | eXclusive OR data from address A - A with register DDD. Results to DDD. | S, Z |

INDIRECT ADDRESSED DATA - REGISTER

MMM Source data is located at the address contained in Register.
MMM = 4, 5 post increment R4 or R5.
MMM = 6 - MVO instruction - post increment R6. PUSH data from Register SSS to the Stack.
Other instructions - pre-decrement R6. PULL data from the Stack to be used as the first operand.

| | | | | | |
|---|---|---|---|---|---|
| MVO@ | SSS, MMM | 9 | 1001 MMM SSS | MoVe Out data from register SSS to the address in register MMM<br>Note: SSS = MMM = 4, 5, 6 or 7 not supported. | |
| PSHR | SSS | 9 | 1001 110 SSS | PuSH data from Register SSS to the stack. | |
| MVI@ | MMM, DDD | 8 * | 1010 MMM DDD | MoVe In data to register DDD from address in register MMM. | |
| PULR | DDD | 11 | 1010 110 DDD | PULl data from the stack to Register DDD. | |
| ADD@ | MMM, DDD | 8 * | 1011 MMM DDD | ADD data located at address in register MMM to the contents of register DDD. Results to DDD. | S, Z, C, OV |
| SUB@ | MMM, DDD | 8 * | 1100 MMM DDD | SUBtract data located at address in Register MMM from contents of register DDD. Results to DDD. | S, Z, C, OV |
| CMP@ | MMM, DDD | 8 * | 1101 MMM SSS | CoMPare data located at address in Register MMM with contents of register SSS, by subtraction. Results not stored. | S, Z, C, OV |
| AND@ | MMM, DDD | 8 * | 1110 MMM DDD | logical AND contents of register DDD with data located at address in register MMM. Results to DDD. | S, Z |
| XOR@ | MMM, DDD | 8 * | 1111 MMM DDD | eXclusive OR contents of register DDD with data located at address in register MMM. Results to DDD. | S, Z |

*Add 3 to number of cycles if MMM=6.

INSTRUCTION SET (continued)

IMMEDIATE DATA - REGISTER The number of iiiii bits depends on the memory width, 16 bits is maximum.

| MNEMONIC | OPERAND | CYCLES | INSTRUCTION | DESCRIPTION | STATUS CHANGE |
|---|---|---|---|---|---|
| MVOI | SSS,I | 9 | 1001 111 SSS<br>iiiiii IIII III III | MoVe Out Immediate data from register SSS to PC+1 (field ) | |
| MVII | I,DDD | 8 | 1010 111 DDD<br>iiiiii IIII III III | MoVe In Immediate data to register DDD from PC+1 (field ). | |
| ADDI | I,DDD | 8 | 1011 111 DDD<br>iiiiii IIII III III | ADD Immediate data to contents of register DDD. Results to DDD. | S, Z, C, OV |
| SUBI | I,DDD | 8 | 1100 111 DDD<br>iiiiii IIII III III | SUBtract Immediate data from contents of register DDD. Results to DDD. | S, Z. C. OV |
| CMPI | I,SSS | 8 | 1101 111 SSS<br>iiiiii IIII III III | CoMPare Immediate data from contents of register SSS by subtraction. Results not stored. | S, Z, C, OV |
| ANDI | I,DDD | 8 | 1110 111 DDD<br>iiiiii IIII III III | logical AND Immediate data with contents of register DDD. Results to DDD. | S, Z |
| XORI | I,DDD | 8 | 1111 111 DDD<br>iiiiii IIII III III | eXclusive OR Immediate data with contents of register DDD. Results to DDD. | S, Z |

| | | | | | |
|---|---|---|---|---|---|
| SDBD | | 4 | 0000 000 001<br>This instruction is normally supplied by the assembler as required to properly generate machine code. | Set Double Byte Data for the next instruction which must be an external reference instruction. The effective address of the external reference instruction will address the low order data byte; the address of the high order data byte will be EA+1 if register 4, 5 or 7 is used. If register 1-3 is used the EA will access the same byte twice resulting in both bytes of data being the same. Use of modes 0 and 6 are not supported by this instruction. | |

INDIRECT ADDRESSED DOUBLE BYTE DATA - REGISTER

| | | | | | |
|---|---|---|---|---|---|
| SDBD<br>MVI@ | <br>MMM, DDD | 4<br>10 | 0000 000 001<br>1010 MMM DDD | MoVe In double byte data from the address in register MMM to register DDD. | |
| SDBD<br>ADD@ | <br>MMM, DDD | 4<br>10 | 000 000 001<br>1011 MMM DDD | ADD double byte data from the address in register MMM to the content of register DDD. Results to DDD. | S, Z, C, OV |
| SDBD<br>SUB@ | <br>MMM, DDD | 4<br>10 | 0000 000 001<br>1100 MMM DDD | SUBtract double byte data located at address MMM from the content of register DDD. Results to DDD. | S, Z, C, OV |
| SDBD<br>CMP@ | <br>MMM, DDD | 4<br>10 | 0000 000 001<br>1101 MMM SSS | CoMPare double byte data located at address in register MMM with the content of register SSS by subtraction. Results is not stored. | S, Z, C, OV |
| SDBD<br>AND@ | <br>MMM, DDD | 4<br>10 | 0000 000 001<br>1110 MMM DDD | logical AND double byte data located at address in register MMM with the content of register DDD. Results to DDD. | S, Z |
| SDBD<br>XOR@ | <br>MMM, DDD | 4<br>10 | 0000 000 001<br>111 MMM DDD | eXclusive OR double byte data located at address in register MMM with the content of register DDD. Results to DDD. | S, Z |

## INSTRUCTION SET (continued)

IMMEDIATE DOUBLE BYTE DATA - REGISTER

Note: The SDBD command is provided by the assembler when the immediate data is greater than the memory width and requires two bytes.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| MVII | I,DDD | 14 | | 0000<br>1010<br>XXLL<br>XXUU | 000<br>111<br>LLL<br>UUU | 001<br>DDD<br>LLL<br>UUU | MoVe In Immediate double byte data to register DDD. L's will be low byte and U's upper byte.<br>XX = don't care. | |
| ADDI | I,DDD | 14 | | 0000<br>1011<br>XXLL<br>XXUU | 000<br>111<br>LLL<br>UUU | 001<br>DDD<br>LLL<br>UUU | ADD Immediate double byte data to contents of register DDD. Results to DDD. L's indicate low byte of literal, U's upper byte. | S, Z, C, OV |
| SUBI | I,DDD | 14 | | 0000<br>1100<br>XXLL<br>XXUU | 000<br>111<br>LLL<br>UUU | 001<br>DDD<br>LLL<br>UUU | SUBtract Immediate double byte data from contents of register DDD. Results to DDD. L's indicate low byte of literal, U's upper byte. | S, Z, C, OV |
| CMPI | I,SSS | 14 | | 0000<br>1101<br>XXLL<br>XXUU | 000<br>111<br>LLL<br>UUU | 001<br>SSS<br>LLL<br>UUU | CoMPare Immediate double byte data with contents of register SSS by subtraction. Results not stored. L's indicate low byte of literal, U's upper byte. | S, Z, C, OV |
| ANDI | I,DDD | 14 | | 0000<br>1110<br>XXLL<br>XXUU | 000<br>111<br>LLL<br>UUU | 001<br>DDD<br>LLL<br>UUU | logical AND Immediate double byte data with the contents of Register DDD. Results to register DDD. L's indicate low byte of literal, U's upper byte. | S, Z |
| XORI | I,DDD | 14 | | 0000<br>1111<br>XXLL<br>XXUU | 000<br>111<br>LLL<br>UUU | 001<br>DDD<br>LLL<br>UUU | eXclusive OR Immediate double byte data with the contents of register DDD. Results to Register DDD. L's indicate low byte of literal, U's upper byte. | S, Z |

GLOSSARY OF TERMS

| | |
|---|---|
| SSS - | Source Register |
| DDD - | Destination Register |
| n - | Number of Shifts |
| RR - | Register to Shift (only 0-3 allowed) |
| AAAAAA<br>AAAAAAAAAA | Memory address for Jump.<br>(new Program Counter) |
| BB - | Register to save old PC in for Jump. (Reg = 1BB, 4, 5, or 6) |
| S - | Sign of address displacement for Branch (PC relative). |
| pppppp PPPPPPPPPP - | Address displacement for Branch<br>pppppp is dependent on the memory word size. |
| aaaaaa AAAAAAAAAA - | Direct address of data word.<br>aaaaaa is dependent on the memory word size. |
| iiiiii IIIIIIIIII - | Immediate data word. iiiiii is dependent on memory word size. |
| LLLLLLLL | Lower 8 bits of double byte data. |
| UUUUUUUU | Upper 8 bits of double byte data. |

| | |
|---|---|
| MMM - | Address Mode |
| 000 - | direct address in location following instruction. |
| 001 - | indirect address for Register 1 |
| 010 - | indirect address for Register 2 |
| 011 - | indirect address for Register 3 |
| 100 - | indirect address for Register 4, post increment |
| 101 - | indirect address for Register 5, post increment |
| 110 - | indirect address for Register 6, post increment for MVO only<br>indirect address for Register 6, pre decrement for all instructions except MVO. |
| 111 - | indirect address for Register 7, post increment.<br>(Immediate data in location following instruction.) |

# APPENDIX C

## S16LNK OBJECT MODULE LINKER

## SAMPLE DIALOGUE

```
 S16LNK VER. 01A

 LOAD MODULE ?
:IOCONVRT
 MAP ? (Y/N OR F=NAME)
:Y
 OBJECT MODULES
:IOCNVROB
:INASCOB
:OUTASCOB
:TTYINOB
:TTYOUTOB
:

 GI S16LNK VER. 01A  10:18 JAN 30,'75
 LOAD MODULE:IOCONVRT

 ****
    <BASE 000000>
 MODULE:CNVRT
   GLOBALS
    IOCNVR 000000
    <SIZE 000375>
 ****
    <BASE 000375>
 MODULE:INASC
   GLOBALS
    HEXBIN 000375
    INTBIN 000401
    OCTBIN 000405
    BINBIN 000411
    <SIZE 000146>
 ****
    <BASE 000543>
 MODULE:OUTASC
   GLOBALS
    HEXASC 000543
    INTASC 000547
    OCTASC 000555
    BINASC 000561
    <SIZE 000246>
 ****
    <BASE 001011>
 MODULE:TTYIN
   GLOBALS
    TTYIN  001011
    <SIZE 000223>
 ****
    <BASE 001234>
 MODULE:TTYOUT
   GLOBALS
    TTYOUT 001234
    TYPCHR 001257
    TYPR2  001270
    <SIZE 000053>


 LINKAGE SUMMARY:
  INITIAL ADDRESS 000000
  FINAL   ADDRESS 001306
  ENTRY   ADDRESS 000000
```

## ASCII CHARACTER CODES

| Char | 7 Bit Octal Code | Char | 7 Bit Octal Code |
|---|---|---|---|
| Space | 040 | @ | 100 |
| ! | 041 | A | 101 |
| " | 042 | B | 102 |
| # | 043 | C | 103 |
| $ | 044 | D | 104 |
| % | 045 | E | 105 |
| & | 046 | F | 106 |
| ' | 047 | G | 107 |
| ( | 050 | H | 110 |
| ) | 051 | I | 111 |
| * | 052 | J | 112 |
| + | 053 | K | 113 |
| , | 054 | L | 114 |
| - | 055 | M | 115 |
| . | 056 | N | 116 |
| / | 057 | O | 117 |
| 0 | 060 | P. | '20 |
| 1 | 061 | Q | 121 |
| 2 | 062 | R | 122 |
| 3 | 063 | S | 123 |
| 4 | 064 | T | 124 |
| 5 | 065 | U | 125 |
| 6 | 066 | V | 126 |
| 7 | 067 | W | 127 |
| 8 | 070 | X | 130 |
| 9 | 071 | Y | 131 |
| : | 072 | Z | 132 |
| ; | 073 | [ | 133 |
|  | 074 | \ | 134 |
| = | 075 | ] | 135 |
|  | 076 | ↑ | 136 |
| ? | 077 | ← | 137 |

# APPENDIX E

## GENERAL INSTRUMENT CORPORATION S16SIM-1 SIMULATOR COMMANDS

| | |
|---|---|
| RT | SET REGISTER TRACE OR TRAP |
| MT | SET MEMORY TRACE OR TRAP |
| RAM | DEFINE RANDOM ACCESS MEMORY BLOCK |
| ROM | DEFINE READ ONLY MEMORY BLOCK |
| IOSR | DEFINE I/O DEVICE STATUS REGISTER |
| IODR | DEFINE I/O DEVICE DATA REGISTER |
| INT | DEFINE INTERRUPT |
| EXT | DEFINE EXTERNAL CONDITION |
| LOAD | LOAD PROGRAM |
| E | EXECUTE PROGRAM |
| EDR | EXECUTE AND DISPLAY REGISTERS |
| S | STEP PROGRAM |
| C | CONTINUE PROGRAM |
| X | EXIT SIMULATOR |
| IA | INITIALIZE ADDRESSES |
| SA | SEARCH ADDRESSES |
| CPA | CLEAR PREVIOUS ADDRESSES |
| TLIM | SET EXECUTION TIME LIMIT |
| RADX | SET DISPLAY RADIX |
| SLIM | SET STACK LIMITS |
| MB | MODIFY BRANCH INSTRUCTION |
| MJ | MODIFY JUMP INSTRUCTION |
| DRT | DISPLAY REGISTER TRAPS AND TRACES |
| DMT | DISPLAY MEMORY TRAPS AND TRACES |
| DMB | DISPLAY MEMORY BLOCKS |
| DIOS | DISPLAY I/O DEVICE STATUS REGISTERS |
| DIOD | DISPLAY I/O DEVICE DATA REGISTERS |
| DINT | DISPLAY INTERRUPTS |
| DEXT | DISPLAY EXTERNAL CONDITIONS |
| R | DISPLAY/MODIFY REGISTER |
| A | DISPLAY/MODIFY ADDRESS |
| SR | DISPLAY/MODIFY CPU STATUS REGISTER |
| CLK | DISPLAY/MODIFY CPU CLOCK RATE |
| INFF | DISPLAY/MODIFY INTRPT FF |
| DR | DISPLAY REGISTERS |
| DA | DISPLAY ADDRESSES |
| DET | DISPLAY EXECUTION TIME |
| DSYM | DISPLAY SYMBOL VALUES |
| DPA | DISPLAY PREVIOUS ADDRESSES, IE, PC VALUES |
| DMXS | DISPLAY MAXIMUM STACK USED |

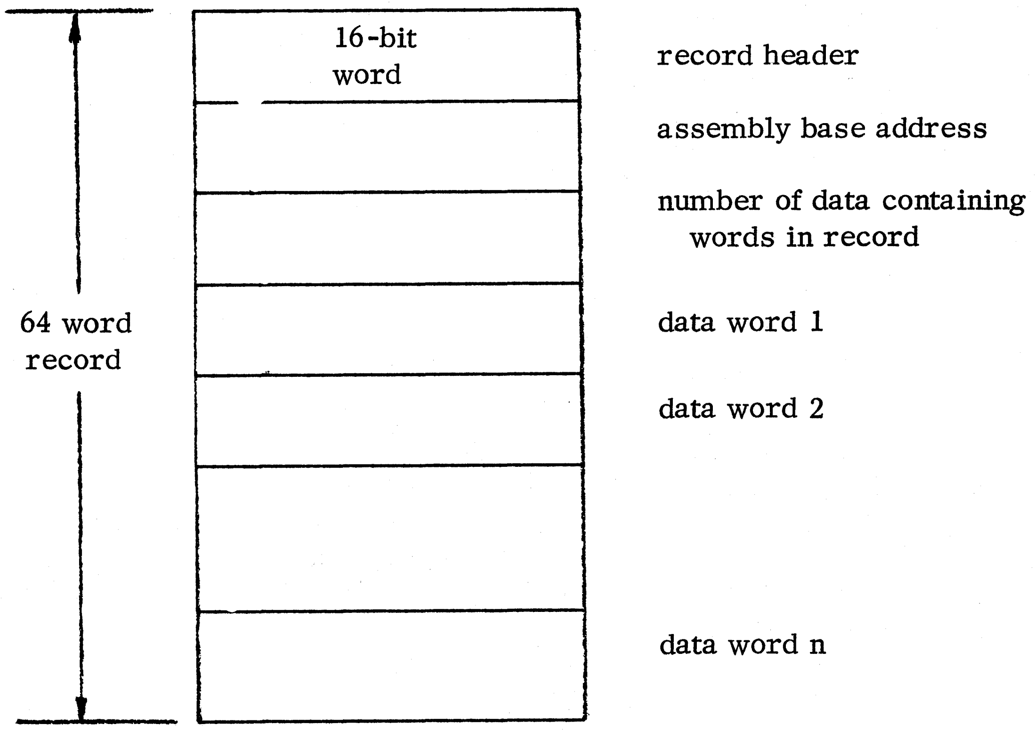
# APPENDIX F

## OBJECT FILE FORMAT

The object file produced by S16XAL contains relocatable object code generated during the assembly process. The file is composed of one or more 64 word records, each containing a three word header and up to 61 object data words.

The first word in all records is equal to either 1 or 2 (-1 or -2 if the record is the last) which indicates a relocatable or absolute module respectively. The second word in the first record contains the assembly base address or origin, in subsequent records the second word has no significance. The third word in all records contains the number of object words following in the record. The remaining significant words in each record contain object code sequences derived from the assembly of instructions or directives.

### RELOCATABLE BINARY FILE RECORD FORMAT



## RELOCATABLE OBJECT CODE SEQUENCES

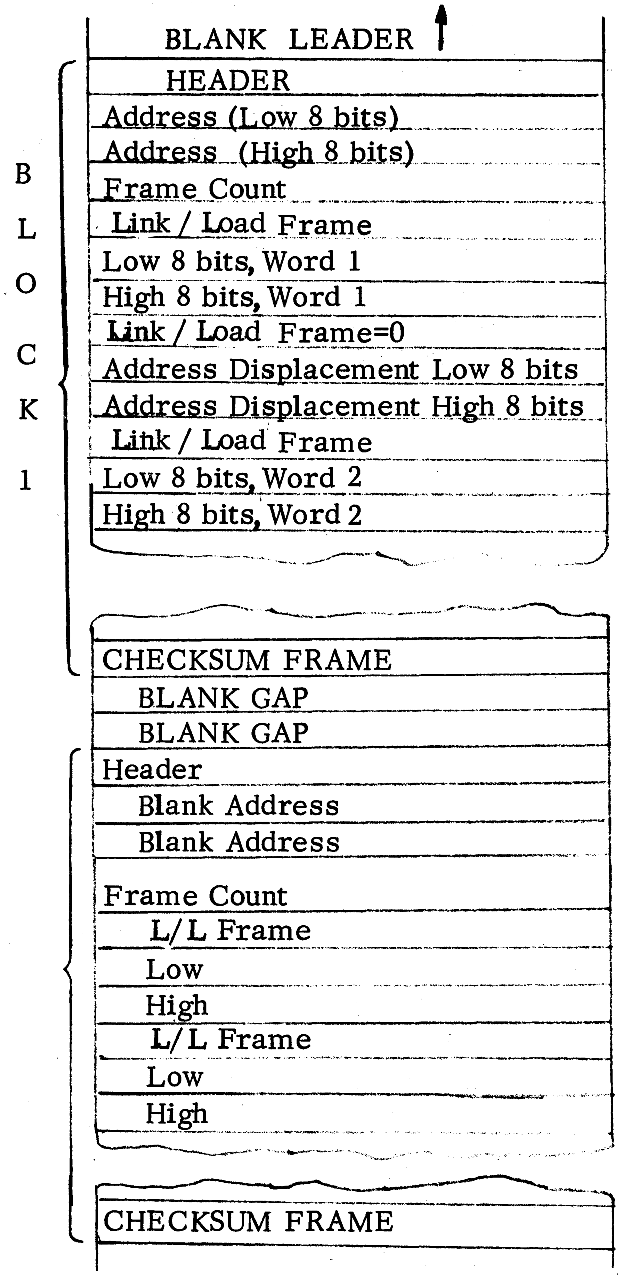
The data information in each record of a S16XAL object file is grouped into sequences of variable length. The first word in each sequence contains a link/load code which indicates the number and nature of object words following in the sequence.

| Code | No. Data Words | Object Word Significance |
|---|---|---|
| 0 | 1 | address adjustment |
| 1 | 1 | absolute word |
| 2 | 2 | absolute word |
| 3 | 3 | absolute word |
| 4 | 1 | relocatable word |
| 5 | 2 | absolute word, relocatable word |
| 6 | 3 | absolute word, 2 relocatable 8-bit bytes |
| 7 | 2 | 2 relocatable 8-bit bytes |
| 8 | 3 | absolute word, 2 relocatable 10-bit bytes |
| 9 | 1 | external reference word |
| 10 | 2 | absolute word, external reference word |
| 11 | 2 | absolute word, external reference displacement |
| 12 | 3 | absolute word, 2 external reference 8-bit bytes |
| 13 | 2 | 2 external reference 8-bit bytes |
| 14 | 3 | absolute word, 2 external reference 10-bit bytes |
| 15 | 1 | entry address word |
| 16 | 2 | module name |
| 17 | 2 | global symbol |
| 18 | 2 | external symbol |

# APPENDIX G

## BINARY PAPER TAPE FORMAT

Binary paper tapes produced by S16BPT consist of variable length records which contain a four frame header and up to 132 data frames. The first significant frame in all records indicates a relocatable or absolute tape, 001 or 002 respectively (377 or 376 in the last record). The second and third frames in the first record contain the assembly base address or origin (low byte, high byte respectively); in subsequent records these two frames have no significance. The fourth frame contains the number of object data frames in the remainder of the record. The last data frame is followed by a record checksum frame which is used during loading to verify that the record has been read correctly. Object code sequences are the same as in a relocatable binary file except that the link/load code occupies one tape frame and each object data word occupies two tape frames, low byte, high byte respectively. The first record on a tape is preceded by approximately 50 frames of blank leader, the last record is followed by blank trailer of the same length and each record is separated by two blank frames.



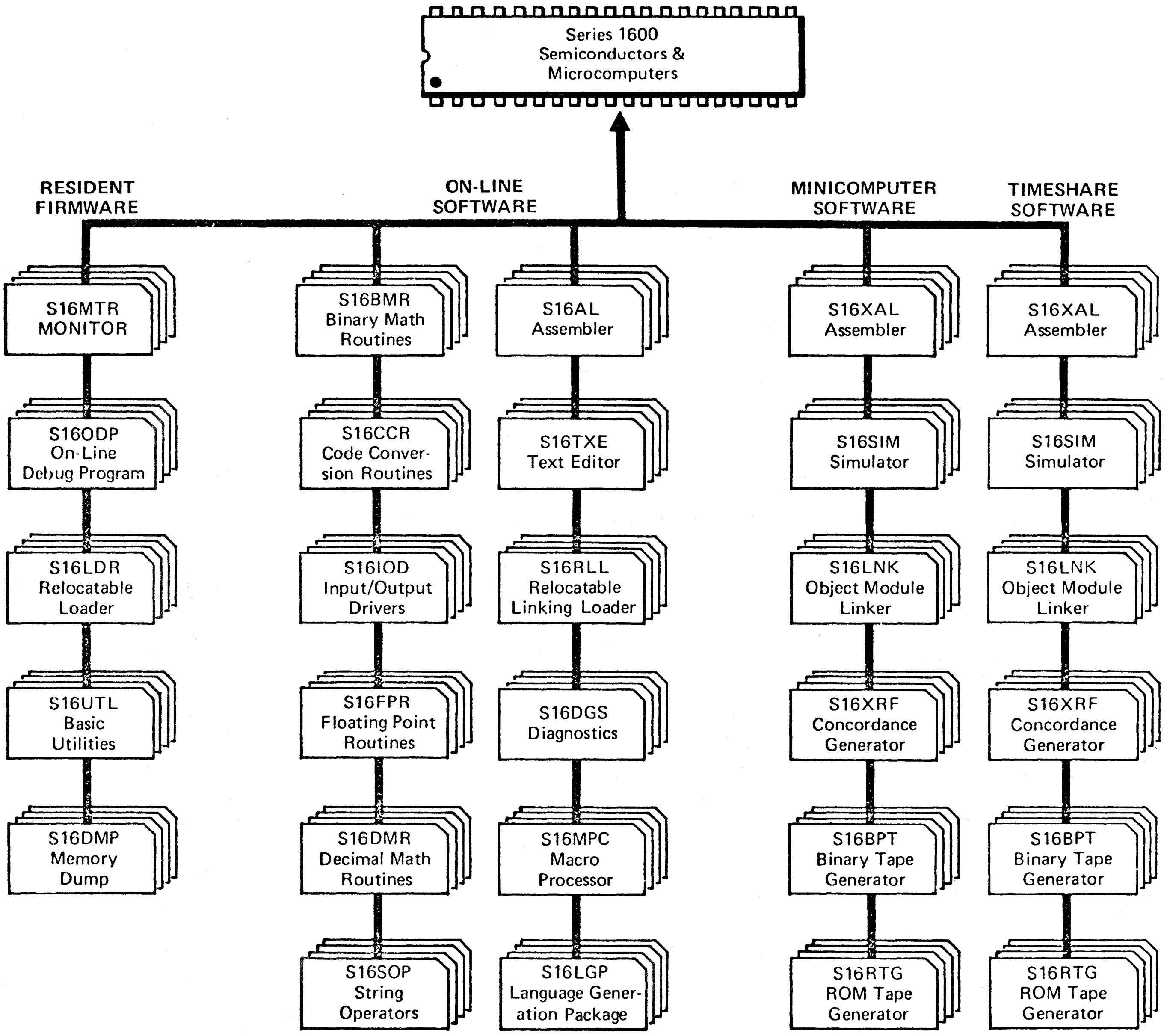
HEADER CODES

| | |
|---|---|
| 1 | Relocatable |
| 2 | Absolute |
| -1 | Last Block of Relocatable |
| -2 | Last Block of Absolute |

LINK/LOAD CODES

Same as codes for OBJECT FILE FORMAT.

# THE SERIES 1600 SOFTWARE LINE UP


Series 1600
Semiconductors &
Microcomputers
RESIDENT FIRMWARE
ON-LINE SOFTWARE
MINICOMPUTER SOFTWARE
TIMESHARE SOFTWARE
S16MTR MONITOR
S16ODP On-Line Debug Program
S16LDR Relocatable Loader
S16UTL Basic Utilities
S16DMP Memory Dump
S16BMR Binary Math Routines
S16CCR Code Conversion Routines
S16IOD Input/Output Drivers
S16FPR Floating Point Routines
S16DMR Decimal Math Routines
S16SOP String Operators
S16AL Assembler
S16TXE Text Editor
S16RLL Relocatable Linking Loader
S16DGS Diagnostics
S16MPC Macro Processor
S16LGP Language Generation Package
S16XAL Assembler
S16SIM Simulator
S16LNK Object Module Linker
S16XRF Concordance Generator
S16BPT Binary Tape Generator
S16RTG ROM Tape Generator
S16XAL Assembler
S16SIM Simulator
S16LNK Object Module Linker
S16XRF Concordance Generator
S16BPT Binary Tape Generator
S16RTG ROM Tape Generator

**GENERAL INSTRUMENT CORPORATION**
**MICROELECTRONICS**

**EASTERN AREA SALES HEADQUARTERS,** 600 W. John St., Hicksville, N.Y. 11802, (516) 733-3107
**CENTRAL AREA SALES HEADQUARTERS,** 3101 West Pratt Blvd., Chicago, Ill. 60645, (312) 338-9200
**WESTERN AREA SALES HEADQUARTERS,** 7120 Hayvenhurst Ave., Van Nuys, Calif. 91406, (213) 781-0489

---

**GENERAL INSTRUMENT CANADA LTD.,** 61 Industry St., Toronto 337, Ontario, Canada, Tel: (416) 763-4133
**GENERAL INSTRUMENT MICROELECTRONICS LTD.,** 57/61 Mortimer St., London, W1N 7TD, England, Tel: 01-636-2022
**GENERAL INSTRUMENT EUROPE S.P.A.,** Piazza Amendola 9, 20149 Milano, Italy, Tel: 469-7751
**GENERAL INSTRUMENT FRANCE SA,** 11-13 Rue Gandon, 75-Paris-13eme, France, Tel: 588-74-31
**GENERAL INSTRUMENT DEUTSCHLAND GMBH,** (Mos Produkigruppe) Nordendstrasse, 1A 8000 Munchen 40 Tel: 28-40-31
**GENERAL INSTRUMENT INTERNATIONAL CORP.,** Fukide Building, 17 Fukide-cho,, Minato-ku, Tokyo 105, Japan, Tel: (03) 437-0281-5
**GENERAL INSTRUMENT OF TAIWAN LTD.,** P.O. Box 22226. Taipei, Taiwan, Tel: 933861-3

Printed in U.S.A.