1. Getting Started

SASM.EXE is an assembler for the Saturn processor family. As a historical note, the Saturn processor first appeared in the HP 71B, and has since been used in various fashions for other HP calculator models, including the HP 28 and the HP 48.

The file SASM.EXE should be installed in a directory such as BIN, with the PATH variable set to include that directory. An environment variable SASM_LIB should be set to point to the file SASM.OPC. For instance, if the opcode file is in \LIB, the command "SET SASM_LIB=C:\LIB" should be added to the autoexec.bat file.

A file naming convention extends throughout the Saturn development environment. Sasm accepts input from files with a ".a" extension, and produces listing files with a ".l" extension and code files with a ".o" extension.

Several options are available:

Option	Description
A	Write listing to stdout
a lstfile	Write listing to "lstfile"
c column	Fields starting after "column" are considered comments
D sym=val	Defines symbol "sym" to have value "val" (default=1)
d dbgfile	Write debug information to "dbgfile"
E	Write C-like error messages to stderr
e	Write error messages to stderr
f flglist	Sets the flags indicated by "flglist" (comma separated)
H	Write object file as raw code (no object header or symbols)
h	Write object file as hexadecimal characters (no object header or symbols)
N	Suppress listing entirely
o objfile	Write object file to "objfile"
P plevel	Sets the processor level to "plevel" (0, 1, 2, or 3)
p patelen	Do a page break each "pagelen" lines
t opcfile	Read opcodes from file "opcfile"
w width	Set output page width to "width" columns (default 80)

Here is an example to illustrate Saturn coding and show what a typical Saturn source file looks like. The name of the source file is "example.a". The Saturn instructions are explained later in this manual.

```
        TITLE   Example Saturn Assembly file
*
* External entry point: =ENTRY1
*
* This routine shifts the C register right 6 digits
* without altering the sticky bit value
*
=ENTRY1 P=      5
        C=0     WP
ENTRY2  CSR     W
        P=P-1
        GONC    ENTRY2
        P=      0
        RTNCC
        END
```

Assembling this file with the command "sasm example.a" generates a Saturn object file (example.o), and a list file (example.l) which is shown on the next page.

```
   1                    TITLE   Example Saturn Assembly file
   2             *
   3             * External entry point: =ENTRY1
   4             *
   5             * This routine shifts the C register right 6 digits
   6             * without altering the sticky bit value
   7             *
   8 00000 25    =ENTRY1 P=      5
   9 00002 A92           C=0     WP
  10 00005 BF6   ENTRY2  CSR     W
  11 00008 0D            P=P-1
  12 0000A 5AF           GONC    ENTRY2
  13 0000D 20            P=      0
  14 0000F 03            RTNCC
  15 00011               END
```

[page break]


```
=ENTRY1        Rel       0 #00000000 -     8
 ENTRY2        Rel       5 #00000005 -    10    12
```

[page break]

Input Parameters

  Source file name is example.a

  Listing file name is example.l

  Object file name is example.o

  Flags set on command line
    None

Errors

  None

2. Saturn CPU Overview

The Saturn CPU is a proprietary CPU optimized for high-accuracy BCD math and low power consumption. The data path is 4 bits wide. Memory is accessed in 4-bit quantities called ``nibbles'' or ``nibs''. Addresses are 20 bits, yielding a physical address space of 512K bytes.

2.1 Registers

There are four working 64-bit registers, five scratch 64-bit registers, two 20-bit data pointer registers, one 4-bit pointer register, a 20-bit program counter, a 16-bit input register, and a 12-bit output register. Return addresses are stored on an eight-level hardware return stack that accepts 20-bit addresses. In addition, there are 4 Hardware Status bits, a Carry bit, and 16 Program Status bits. The lower 12 Program Status bits can be manipulated as a 12-bit register.

2.2 Working and Scratch Registers

The working registers A, B, C, and D are used for data manipulation. Working registers A and C are also used for memory access. The scratch registers R0, R1, R2, R3, and R4 are used to temporarily hold the contents of working registers.

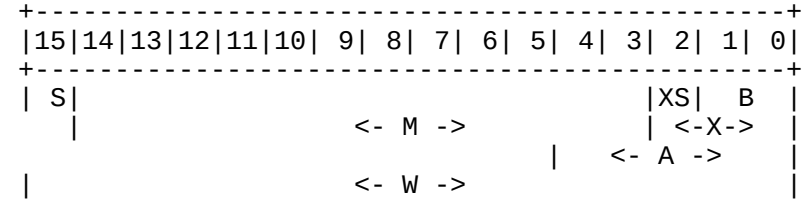
2.3 Field Selection

Subfields of the working registers A, B, C, and D may be accessed by the use of field selection. The possible field selections range from the entire register to any single nibble of the register. Certain subfields are designed for use in BCD calculations. Others are used for data access or general data manipulation.

Field Selection Description

P	Nibble indicated by P register
WP	Nibbles from nibble P through nibble 0, inclusive
XS	Nibble 2;- Exponent sign
X	Nibbles 2-0;- Exponent including exponent sign
S	Nibble 15;- Mantissa sign
M	Nibbles 14-3;- Mantissa
B	Nibbles 1-0;- Byte field
A	Nibbles 4-0;- Address field (20 bits)
W	Nibbles 15-0;- Word (entire 64-bit register)

Register Nibbles



2.4 Pointer Registers

The 20 bit Data Pointer registers D0 and D1 are used to contain addresses during memory access, and are used in conjunction with the working registers.

The 4 bit Pointer register P is used in Field Selection operations with the working registers.

2.5 Input, Output, and Program Counter Registers

The input/output registers are used to communicate with the system bus. The program counter points to the next instruction to be executed by the CPU. The input register IN is 16 bits, the output register OUT is 12 bits, and the PC register is 20 bits.

2.6 Carry, Program Status, and Hardware Status Bits

The Carry bit is adjusted when a calculation or logical test is performed. During a calculation, such as incrementing or decrementing a register, it is set if the calculation overflows or borrows; otherwise it is cleared. During a logical test, such as comparing two registers for equality, it is set if the test is true; otherwise it is cleared.

The upper 4 Program Status bits are typically used to indicate the state of the operating system. The remaining 12 Program Status bits are generally available to applications software, and may be manipulated collectively as the ST register.

The four Hardware Status bits are set (but not cleared) by hardware-related events, and must therefore be cleared beforehand in order to detect a particular occurrence. They are individually accessible by name. The Module Pulled bit (MP) is set whenever the *NINTX CPU input is pulled low (regardless of whether an interrupt is actually executed). The Sticky Bit (SB) is set when a non-zero bit shifts off the right end of a working register as the result of a shift instruction. The Service Request (SR) bit is set as a result of the SREQ? instruction if any hardware service request is pending. The external Module Missing bit is set by execution of a ``00'' opcode (RTNSXM instruction).

Hardware Status: 4 bits

Bit	Symbol	Name
3	MP	Module Pulled (*NINTX pulled low)
2	SR	Service Request
1	SB	Sticky Bit
0	XM	External Module Missing

2.7 Arithmetic Mode

The arithmetic mode is set by the SETHEX and SETDEC instructions. When SETHEX is executed, the arithmetic mode is set so that all register arithmetic is performed in hexadecimal mode. When SETDEC is executed, the arithmetic mode is set so that most register arithmetic is performed in decimal mode. The following instructions are *always* performed in hexadecimal mode, regardless of the arithmetic mode setting:

```
P=P+1
P=P-1
C+P+1
D0=D0+     n   D0=D0-     n
D1=D1+     n   D1=D1-     n
A=A+CON fs,n   A=A-CON fs,n
B=B+CON fs,n   B=B-CON fs,n
C=C+CON fs,n   C=C-CON fs,n
D=D+CON fs,n   D=D-CON fs,n
```

The arithmetic mode is not ``readable'', but can be inferred by doing an appropriate operation, followed by a test. For example:

```
LCHEX   9
C=C+1   P
```

sets the carry if, and only if, the arithmetic mode is decimal.

2.8 Loading Data from Memory

When data is read from memory into a register, the CPU places the lowest addressed nibble in the least significant nibble of the register.

For example, if the data shown below in memory is read into the C register using the C=DAT1 4 instruction, the data in the register will be arranged as shown.

```
 Memory
Location  Value            C Register
--------  -----     +-----------------------+
  1000      6       |  | ... | 9 | 8 | 7 | 6 |
  1001      7       +-----------------------+
  1002      8        15  ...   3   2   1   0
  1003      9
```

This principle applies also to loading constants into a CPU register such as C, D0, or D1, since the CPU must read the constant from the instruction opcode in memory. For example, the instruction LCHEX 9876 produces the opcode

336789 and the C register is loaded as shown above (assuming P= 0).

2.9 Storing Data in Memory

When data is written from a register to memory, the CPU places the least significant nibble of the register in the lowest nibble of the addressed memory location. For example, if the data shown above in the C register is written to memory using the DAT1=C 4 instruction, the data will be written to memory as shown.

2.10 Interrupt System

All Saturn CPU interrupts cause a subroutine jump to address #0000F. Determining the cause of the interrupt is up to the interrupt service routine at that address.

3. Conditional Assembly

The assembler supports conditional assembly tests to allow different code to be generated based on various conditions. Conditions which can be tested include assembly flags set on the command line or modified with the SETFLAG or CLRFLAG instructions, the value of an assemble-time expression compared to zero, the presence or absence of a symbol definition, the (guaranteed) carry state (for example, after GOYES the carry is clear), the current assembler pass (useful for messages), and the presence of an opcode or macro definition.

The structure of a conditional assembly block is:

```
label   IF
          <<code if condition is true>>
label   ELSE
          <<code if condition is false>>
label   ENDIF
```

The label and ELSE sections are optional.

Conditional assembly blocks can be nested up to a maximum of 20 levels if a unique label is present for each nesting. If the label is omitted or not unique, the code assembled may not nest as expected for the false block. For example:

```
 1              IF      0
 2              IF      1
 3       * Flags 0 and 1 are set
 4              ELSE
 5       * Flag 0 is set, flag 1 is clear
 6              ENDIF
 7              ELSE
 8              IF      1
 9       * Flag 0 is clear, flag 1 is set
10              ELSE
11       * Flags 0 and 1 are clear
12              ENDIF
13              ENDIF
```

If flag 0 is clear, the ELSE on line 4 is found as the ELSE matching the IF 0. The ELSE on line 7 and the ENDIF on line 13 are flagged as errors (ELSE without matching IF, ENDIF without matching IF). Lines 5 and 11 are both assembled. If the nested IF statements had unique labels, they would work as expected.

For a list of the conditional assembly opcodes, see ``Conditional Assembly'' in the ``Pseudo-Op Instructions'' section of the ``Saturn Assembler Mnemonics" appendix.

4. Using Macros

This chapter explains how to create and use macros in your source files. The macro directives are MACRO, EXITM, and ENDM.

A macro is a named block of source statements. When a macro name is used as a statement, it is automatically replaced by the block of source statements it represents.

Macro definitions cannot be nested; the MACRO statement is illegal within a macro call.

Macro lines are not listed by default; to enable listing of the macro expansion, use either the LISTM or SETLIST MACRO statement.

4.1 Defining a Macro

A macro definition consists of a MACRO statement, followed by the source statements to make up the macro, followed by an ENDM statement. The label on the ENDM statement, if any, must match the name of the macro (the label on the MACRO statement). Text which follows the MACRO statement on the same line is ignored (comment only). It is suggested that the comments indicate the parameters which are expected when the macro is called. Up to nine parameters can be passed to a macro when it is called. The statements between the MACRO statement and the ENDM statement are not assembled until the macro is called. Any assembler pseudo-ops within a macro definition are executed when the macro is called, not when it is defined. An exclamation mark (!) in the first column is removed from the line when the macro is called. The rest of the line is included in the expansion.

Blank lines and comment lines are not normally included in the macro expansion. To include a blank line or a comment line in a macro expansion, add an exclamation mark in the first column.

The dollar sign ($) is used in macros as a text substitution character. The character which follows the dollar sign indicates the replacement:

Sequence	Replacement Text
$$	$ (one dollar sign)
$0	Line number on which the macro was called
$<	Current source file name
$n	Parameter reference (1_n_9)
$(n)	Indirect parameter reference (1_n_9)
$(nf)	Formatted indirect parameter reference (1_n_9)

Parameter references are replaced by the corresponding parameter text.

Indirect parameter references are replaced by the value of the corresponding parameter when interpreted as an expression. Formatted indirect parameter references include a format string f which follows the parameter number. The format string has the form [:] [length] [format char]. Length indicates the minimum number of characters to use. The value is zero-filled if it requires fewer than length digits. The default length is one. Format char controls the radix and case of the value. The table shows valid format char values:

Character	Description	Digit Characters
H,X	Hexadecimal	0123456789ABCDEF
h,x	Hexadecimal	0123456789abcdef
D,d	Signed decimal	0123456789 (default)
U,u	Unsigned decimal	0123456789
O,o	Octal	01234567

The maximum combined nesting depth for macros and include files is 20 levels.

4.2 Calling a Macro

To call a macro, specify the name of the macro as an instruction. The macro body will be included at this point in the source file. Parameters to be passed to the macro follow the macro name on the line. Up to nine parameters may be passed to a macro.

4.3 Parameter Assignment Rules

The parameter text on the macro call line is assigned to parameters in the following way:

1. Skip all leading blanks and tabs and set the current parameter number to one.

2. If the first character is a <, all text up to a > is assigned to the current parameter number. If the first character is not a < and there is a comma (,) in the remaining text, all text up to the comma is assigned to the current parameter number.

   If the first character is a <, but there is not a > in the remaining text, the < is considered to be a normal text character.

3. If the first character is not a < and there is no comma in the remaining text, all text up to the first blank or tab is assigned to the current parameter number.

4. Increment the parameter number. If the parameter number is less than nine, go back to step 2.

NOTE: The comment column value is ignored for macro call lines; only characters which follow the first blank or tab following the last parameter are ignored.

4.4 Macro Example

The (simple) example below is a macro that increments the A register by the amount passed as a parameter.

```
ADDTOA  MACRO
        LC(5)   $1
        A=A+C   A
ADDTOA  ENDM

        ...
        DAT0=A  M
        ADDTOA  312
        D0=D0+  12
```

5. File Access Statements

There are three statements which access data in other files. RDSYMB reads the symbols from a Saturn object file, INCLUDE reads Saturn source statements from a file, and CHARMAP simplifies the problem of working with a non-ASCII character set often found in calculators.

The file name for each of the statements can be specified several ways. The name can be specified by itself or surrounded by quotes ("filename"), apostrophes ('filename'), or brackets (<filename>). If the file name contains blanks or tabs, it must be quoted by one of these methods.

Each statement uses an environment variable to determine where to search for the file (the default is the current directory and a system directory). See the "Environment Variables" appendix for more information about environment variable names and defaults.

5.1 RDSYMB Statement

The RDSYMB statement reads the symbol table from a Saturn object file. All external symbols which are defined in the object file and are not relocatable are available for use in the file being assembled. Symbols defined by a RDSYMB statement are not included in the symbol table listing unless they are actually used in the assembly.

5.2 INCLUDE Statement

The INCLUDE statement tells the assembler to read source statements from the specified file. The assembler reads from the file until an END statement is processed or the end of the file is reached.

The lines read from include files are not normally listed. Only pseudo-op statements and lines containing errors are listed by default. To enable full include file listing, use either the SETLIST INCLUDE statement or the LISTM statement. Included lines have a - after the line number in the listing. The line number shown in the listing is the line within the include file.

5.3 CHARMAP Statement

The LAASC, LCASC, and NIBASC statements are of limited usefulness when the character set used in a product is not ASCII. The CHARMAP statement allows the ASCII characters specified in those statements to be automatically converted

to a different character set. For example, if a particular product is only capable of displaying letters and digits, the character set mapping might be A ... Z = #00 ... #19, a ... z = #1A ... #35, and 0 ... 9 = #36 ... #40. Using CHARMAP with a file which contains all these pairings allows the assembler to automatically convert all references to ASCII characters to the corresponding character set character. This permits source files to be independent of the actual character mapping by including a CHARMAP statement at the start of the file.

The mapping becomes effective when the CHARMAP statement is executed; it does not affect ASCII characters preceding the CHARMAP statement. The effect of multiple CHARMAP statements is cumulative.

5.3.1 Charmap_File_Format The file indicated by the CHARMAP statement contains a list of pairings, one pair per line. The first character position is the ASCII character which is used in the source file, and the second character position is the value which is used for the generated code. Each character position can be either an ASCII character or an escape sequence similar to those found in the C language. The following table summarizes the escape sequences recognized in a CHARMAP file:

Sequence	ASCII code	Description
ASCII char	Same	The specified ASCII character
\a	7	BEL (alert character)
\b	8	BS (backspace)
\t	9	HT (tab)
\n	10	LF (linefeed)
\v	11	VT (vertical tab)
\f	12	FF (formfeed)
\r	13	CR (carriage return)
\\	92	Backslash
\xhh	Hex hh	Character with hex value hh (\xh ok if not ambiguous)
\ddd	Octal ddd	Character with oct value ddd (\dd or \d ok if not ambiguous)

6. Saturn Assembler Format and Mnemonics

This chapter describes the Saturn assembler instruction set. The Saturn CPU has three variations used in several products. The 1LF2 was used in the first versions of the HP-71B. The 1LK7 is a variation of the 1LF2 used in later versions of the HP-71B, the HP-18C, and the HP-28C. The 1LR2 is an integrated CPU/ROM/RAM/Display Driver IC. Each new version of the Saturn CPU added new instructions to the Saturn instruction set. Instructions available in all Saturn CPUs are referred to as "level 0" instructions. Instructions available in the 1LK7 and 1LR2 but not the 1LF2 are referred to as "level 1" instructions. Instructions available only in the 1LR2 are referred to as "level 2" instructions. In this section, "level 1" instructions are marked with an asterisk (*), "level 2" instructions are marked with two asterisks (**). Instructions with no mark are "level 0" instructions.

6.1 Instruction Syntax

The assembler is "free format" and a space or tab is required to delimit the different fields. A label, if present, must start in column one or two. The format below is a recommended column alignment:

```
1       9       17              33       ...   80
-------------------------------------------------
Label   Opcode  Modifier        Comments ...
-------------------------------------------------
```

6.1.1 Comments

A comment line begins with an asterisk (*) in column one, and may occur anywhere in the file. An in-line comment may begin with any non-blank character and must follow the modifier field of an instruction (or the opcode if no modifier is required).

6.1.2 Symbols_and_Labels

A symbol is a name for a numeric value. A symbol acquires its value by appearing in the label field of certain statements. The word "symbol" is a general term for a label, and the two are used interchangeably.

Symbols consist of one to twelve alphanumeric characters with the following restrictions: the characters comma (,), space ( ), and right parenthesis are prohibited, and special care must be used if the first character is an equal sign (=), colon (:), sharp (#), left parenthesis, or a digit (0 through 9).

A symbol may be immediately preceded by an equal sign (=) which declares the symbol to be an external symbol. An external symbol defined in one module may be referenced as an external symbol by another module. Such references are resolved when the modules are linked together. Certain Saturn assemblers, such as the HP-71 FORTH/Assembler ROM, have no associated linker and therefore do not support external symbols. In this case, any leading equal sign is ignored.

A symbol may instead be immediately preceded by a colon (:) which simply declares what follows to be a (local) symbol. Either an equal sign or a colon must be used with any symbol whose first character is in the special care category (=, :, #, (, 0-9).

When a symbol is used as part of an expression, parentheses are required to delineate it. That is, AD1-10 is a symbol but (AD1)-10 is a computed expression.

6.1.3 Expressions

Wherever an expression may appear in the modifier field of an instruction, it is represented by the symbol "expr" in the instruction descriptions below. Expressions are evaluated using 32-bit signed integer math. If a value does not fit within 32 bits, the most significant bits are lost (only the low 32 bits are saved).

Expression Components

Component	Examples
decimal constant	23434
hexadecimal constant	#1FF0 (less than #100000000)
ASCII constant	\AB\, 'AB' (4 or less characters)
operator	+ addition - subtraction synonym for * 256 + * multiplication / integer division % modulo (remainder) ^ integer exponentiation & bitwise AND ! bitwise OR
*	Current assembly program counter
symbol	Symbol defined within this file
(expression)	Parenthesized expression

Two classes of instructions require a modifier field which contains a constant of a specific type that does not conform to the above rules. These are:

1. Instructions with a string constant which can exceed 4 characters:

```
LAASC   \ASCII\       ( 8 characters maximum) **
LCASC   \ASCII\       ( 8 characters maximum)
NIBASC  \ASCII...\    (40 characters maximum)
STRING  \ASCII...\    (40 characters maximum)
```

2. Instructions with a required hexadecimal constant:

```
LAHEX   0123456789ABCDEF    (16 digits maximum) **
LCHEX   048C3               (16 digits maximum)
NIBHEX  0123456789ABCDEF    (80 digits maximum)
```

6.2 Explanation of Symbols

In the descriptions of the Saturn assembler mnemonics, these symbols are defined as follows:

a — The hex nibble used to encode the field selection in the assembled opcode of an instruction. See the Field Select Table in the next section for details.

b — The hex nibble used to encode the field selection in the assembled opcode of an instruction. See the Field Select Table in the next section for details.

d — The number of nibbles represented by a field selection field. Used in calculating the execution cycle time of some instructions. See the Field Select Table in the next section for details. When used in an extended field selection fsd, represents an expression which indicates the number of nibbles of the register that will be affected by the instruction, proceeding from the low-order nibble to higher-order nibbles.

expr — An expression that evaluates to an absolute or relocatable value, usually less than or equal to 5 nibbles in length.

fs — Field selection symbol. See the Field Select Table in the next section for details.

fsd — Extended field selection symbol. Represents either a normal field selection symbol fs, or an expression that gives the number of nibbles d of the register that will be affected by the instruction, proceeding from the low-order nibble to higher-order nibbles.

hh Two-digit hex constant, such as 08 or F2. Within an opcode represents the hex digits used to store the value of the expression in the opcode in reverse order (see "Loading Data From Memory").

hhhh Four-digit hex constant, such as 38FE. Within an opcode, represents the hex digits used to store the value of the expression in the opcode in reverse order (see "Loading Data From Memory").

hhhhh Five-digit hex constant, such as 308FE. Within an opcode, represents the hex digits used to store the value of the expression in the opcode in reverse order (see "Loading Data From Memory").

label A symbol defined in the label field of an instruction.

m A one-digit decimal integer constant.

n Represents an expression that evaluates to a 1-nibble value, unless specified otherwise. Within an opcode, represents the hex digit used to store the assembled value of the expression in the opcode.

nn Represents an expression that evaluates to a 2-nibble value, unless specified otherwise. Within an opcode, represents the hex digits used to store the assembled value of the expression in the opcode.

nnnn Represents an expression that evaluates to a 4-nibble value, unless specified otherwise. Within an opcode, represents the hex digits used to store the assembled value of the expression in the opcode.

nnnnn Represents an expression that evaluates to a 5-nibble value, unless specified otherwise. Within an opcode, represents the hex digits used to store the assembled value of the expression in the opcode.

r,s Represents a register (r alone) or a register pair (r and s used together). The valid combinations are (r,s) = { (A,B), (B,C), (C,A), (D,C) }.

ss Represents a scratch register name (R0, R1, R2, R3, or R4).

dp Represents a data pointer name (D0 or D1).

6.3 Field Select Table

The following symbols are used in the instruction descriptions to denote field selections.

There are two ways in which field selection is encoded in the opcode of an instruction. These two patterns are shown in the table below, and are designated by the letters a and b.

Field Select Table

Field	Name and Description	Opcode Representation (a)	(b)	Number of Nibs (d)
P	Pointer Field. Nibble specified by P pointer register.	0	8	1
WP	Word-Through-Pointer Field. Nibbles P through 0.	1	9	P+1
XS	Exponent Sign Field. Nib 2.	2	A	1
X	Exponent Field. Nibs 2-0.	3	B	3
S	Sign Field. Nibble 15.	4	C	1
M	Mantissa Field. Nibs 14-3.	5	D	12
B	Byte Field. Nibs 1-0.	6	E	2
W	Word Field. Nibs 15-0.	7	F	16
A	Address Field. Nibs 4-0.	F	-	5

Some instructions have an entirely different opcode representation for the A field.

6.4 Instruction Set Overview

This is a summary of the Saturn instruction set, grouped by functional category.

Fields of a register are indicated using the convention that a register name followed by a field in parentheses means that field of the register. For example, C(A) means the A field of register C, and A(3:0) means nibbles 3 through 0 of register A.

6.5 Jump Instructions

GOTO	label	Unconditional relative jump; range -2047, +2048 nibs.
GOC	label	Relative jump if Carry is set; range -127, +128 nibs.
GONC	label	Relative jump if Carry is clear; range -127, +128 nibs.
GOSHORT	label	Generate a short jump to label. If the carry state cannot be determined at assembly time, a GOTO is generated. If the carry is known to be set, a GOC is generated. If the carry is known to be clear, a GONC is generated.
JUMP	label	Alias for GOSHORT.
GOLONG	label	Unconditional long relative jump; range -32766, +32769 nibs.
GOVLNG	label	Absolute jump; range unrestricted.
PC=(A)		* Indirect jump; A(A) is the address of the destination address.
PC=(C)		** Indirect jump; C(A) is the address of the destination address.
PC=A		** Direct jump; A(A) is the destination address.
PC=C		** Direct jump; C(A) is the destination address.
APCEX		** Direct jump and save PC in A(A); A(A) is the destination address.
CPCEX		** Direct jump and save PC in C(A); C(A) is the destination address.
GOYES	label	Relative jump if test is true (second half of test instruction); range -125, +130 nibs from test.

6.6 Subroutine Call Instructions

GOSUB	label	Relative jump to subroutine; range -2044, +2051 nibs.
GOSUBL	label	Long relative jump to subroutine; range -32762, +32773 nibs.
GOSBVL	label	Absolute jump to subroutine.

6.7 Subroutine Return Instructions

RTN	Return from subroutine.
RTNSC	Return from subroutine and set Carry.
RTNCC	Return from subroutine and clear Carry.
RTNSXM	Return from subroutine and set hardware status bit XM.
RTI	Return from subroutine and enable interrupt handling.
RTNC	Return from subroutine if Carry is set.
RTNNC	Return from subroutine if Carry is clear.
RTNYES	Return from subroutine if test is true (second half of test instruction).

6.8 Test Instructions

All test instructions must be followed with a GOYES or a RTNYES instruction. The test instruction and the following GOYES or RTNYES instruction together form a single 5-nibble opcode. The Carry is set when the test is true and cleared when the test is false. All register comparisons are unsigned (#FFFFF is greater than #7FFFF). The test is performed only on the selected field.

6.8.1 Register_Tests

?r=s	fs	True if r(fs) and s(fs) are equal.
?r#s	fs	True if r(fs) and s(fs) are not equal.
?r=0	fs	True if r(fs) is zero.
?r#0	fs	True if r(fs) is non-zero.
?r>s	fs	True if r(fs) is greater than s(fs).
?s>r	fs	True if s(fs) is greater than r(fs).
?r<s	fs	True if r(fs) is less than s(fs).
?s<r	fs	True if s(fs) is less than r(fs).
?r>=s	fs	True if r(fs) is greater than or equal to s(fs).
?s>=r	fs	True if s(fs) is greater than or equal to r(fs).
?r<=s	fs	True if r(fs) is less than or equal to s(fs).
?s<=r	fs	True if s(fs) is less than or equal to r(fs).

6.8.2 Pointer_Tests

?P=	n	True if P is equal to n.
?P#	n	True if P is not equal to n.

6.8.3 Program_Status_Bit_Tests

?ST=0	n	True if status bit n is clear.
?ST=1	n	True if status bit n is set.
?ST#1	n	Alias for ?ST=0 n.
?ST#0	n	Alias for ?ST=1 n.

6.8.4 Hardware_Status_Bit_Tests

?XM=0		True if XM bit (external module missing) is clear.
?SB=0		True if SB bit (sticky bit) is clear.
?SR=0		True if SR bit (service request) is clear.
?MP=0		True if MP bit (module pulled) is clear.
?HS=0	n	True if all bits corresponding to n are clear.

6.8.5 Register_Bit_Tests

?ABIT=0	n	** True if bit n of register A is clear.
?ABIT=1	n	** True if bit n of register A is set.
?CBIT=0	n	** True if bit n of register C is clear.
?CBIT=1	n	** True if bit n of register C is set.
?ABIT#1	n	** Alias for ?ABIT=0 n.
?CBIT#1	n	** Alias for ?CBIT=0 n.
?ABIT#0	n	** Alias for ?ABIT=1 n.
?CBIT#0	n	** Alias for ?CBIT=1 n.

6.9 Pointer Instructions

All arithmetic calculations on the pointer are performed in HEX mode.

P=	n	Set register P to n.
P=P+1		Increment P register; affects Carry.
P=P-1		Decrement P register; affects Carry.
C+P+1		Add P plus one to A field of C; affects Carry.
C=C+P+1		Alias for C+P+1.
CPEX	n	Exchange P register and nibble n of C register.
P=C	n	Copy nibble n of C register to P register.
C=P	n	Copy P register to nibble n of C register.

6.10 Bit Manipulation Instructions

ABIT=0	n	** Clear bit n of register A.
ABIT=1	n	** Set bit n of register A.
CBIT=0	n	** Clear bit n of register C.
CBIT=1	n	** Set bit n of register C.

6.11 Status Instructions

6.11.1 Program_Status

ST=0	n	Set status bit n.
ST=1	n	Clear status bit n.
CSTEX		Exchange status bits 11-0 with C(X).
C=ST		Copy status bits 11-0 to C(X).
ST=C		Copy C(X) to status bits 11-0.
CLRST		Clear status bits 11-0.

6.11.2 Hardware_Status

XM=0		Clear XM bit (external module missing).
SB=0		Clear SB bit (sticky bit).
SR=0		Clear SR bit (service request).
MP=0		Clear MP bit (module pulled).
HS=0	n	Clear all bits corresponding to n.
CLRHST		Clear all Hardware Status bits (XM, SB, SR, and MP).

6.11.3 System_State_Instructions

SETHEX	Set arithmetic mode to hexadecimal.
SETDEC	Set arithmetic mode to decimal.
SREQ?	Set C(0) to service request response from bus. Set SR bit if service is requested.
C=RSTK	Pop subroutine return stack into C(A).
RSTK=C	Push C(A) onto subroutine return stack.
A=PC	** Copy current PC into A(A).
C=PC	** Copy current PC into C(A).
CONFIG	Configure a device to the address in C(A).
UNCNFG	Unconfigure a device at the address in C(A).
RESET	Send Reset command to system bus.
BUSCB	** Issue bus command B on the system bus.
BUSCC	Issue bus command C on the system bus.
BUSCD	** Issue bus command D on the system bus.
SHUTDN	Stop CPU here, stay in low-power state until wake-up requested.
C=ID	Copy chip ID from system bus to C(A).
INTOFF	Disable maskable interrupts.
INTON	Enable maskable interrupts.
RSI	* Reset interrupt detect circuitry.

6.11.4 Keyscan_Instructions

```
OUT=C           Copy C(X) to OUT register.
OUT=CS          Copy C(0) to low 4 bits of OUT register.
A=IN            Copy IN register to A(3:0).
C=IN            Copy IN register to C(3:0).
```

6.11.5 Scratch_Register_Instructions

```
A=ss            Copy ss to A.
C=ss            Copy ss to C.
ss=A            Copy A to ss.
ss=C            Copy C to ss.
AssEX           Exchange A with ss.
CssEX           Exchange C with ss.
A=ss.F  fs      ** Copy ss(fs) to A(fs).
C=ss.F  fs      ** Copy ss(fs) to C(fs).
ss=A.F  fs      ** Copy A(fs) to ss(fs).
ss=C.F  fs      ** Copy C(fs) to ss(fs).
AssEX.F fs      ** Exchange A(fs) with ss(fs).
CssEX.F fs      ** Exchange C(fs) with ss(fs).
```

6.11.6 Data_Pointer_Instructions

```
dp=A            Copy A(A) to dp.
dp=C            Copy C(A) to dp.
AdpEX           Exchange A(A) with dp.
CdpEX           Exchange C(A) with dp.
dp=AS           Copy A(3:0) to (dp3:0).
dp=CS           Copy C(3:0) to (dp3:0).
AdpXS           Exchange A(3:0) with dp(3:0).
CdpXS           Exchange C(3:0) with dp(3:0).
dp=dp+  n       Increment register dp by n; alters
                Carry.
dp=dp-  n       Decrement register dp by n; alters
                Carry.
dp=HEX  hh      Load hh into dp(1:0).
dp=HEX  hhhh    Load hhhh into dp(3:0).
dp=HEX  hhhhh   Load hhhhh into dp.
dp=(2)  expr    Load expr into dp(1:0); use low 2
                nibbles of expr if too big.
dp=(4)  expr    Load expr into dp(3:0); use low 4
                nibbles of expr if too big.
dp=(5)  expr    Load expr into dp; use low 5 nibbles
                of expr if too big.
```

6.11.7 Data_Transfer_Instructions

If fsd is an expression, the value of the expression is the number of nibbles to transfer. For example, if fsd is an expression whose value is 7, nibbles 6 through 0 will be transferred.

```
    A=DAT0  fsd     Read data pointed to by D0 into A(fsd).
    A=DAT1  fsd     Read data pointed to by D1 into A(fsd).
    C=DAT0  fsd     Read data pointed to by D0 into C(fsd).
    C=DAT1  fsd     Read data pointed to by D1 into C(fsd).
    DAT0=A  fsd     Write A(fsd) to location indicated by D0.
    DAT1=A  fsd     Write A(fsd) to location indicated by D1.
    DAT0=C  fsd     Write C(fsd) to location indicated by D0.
    DAT1=C  fsd     Write C(fsd) to location indicated by D1.
```

6.11.8 Load_Constant_Instructions

All constants are loaded into the target register least significant nibble first, with the least significant nibble loaded into r(P) and subsequent nibbles loaded at r(P+1), r(P+2), etc. until all nibbles have been loaded. A constant can wrap around from r(15) to r(0).

```
    LAHEX   hhhhhhhh    ** Load hex constant hhhhhhhh into A.
    LCHEX   hhhhhhhh       Load hex constant hhhhhhhh into C.
    LAASC   \ASCII\     ** Load ASCII constant ASCII into A.
    LCASC   \ASCII\        Load ASCII constant ASCII into C.
    LA(m)   expr        ** Load an m-nibble constant into A;
                           use low m nibbles of expression if
                           too big.
    LC(m)   expr           Load an m-nibble constant into C;
                           use low m nibbles of expression if
                           too big.
    LA(N)   expr        ** Start an expr-nibble Load Constant
                           into register A.  This is useful
                           for Load Constants which are too
                           large for LA(m) or which involve
                           multiple external references.
    LC(N)   expr           Start an expr-nibble Load Constant
                           into register C.  This is useful
                           for Load Constants which are too
                           large for LC(m) or which involve
                           multiple external references.
```

6.11.9 Shift_Instructions

The term circular shift means that the nibble shifted out gets shifted in at the other end of the selected field. NOTE: Right shift instructions set the Sticky Bit if any non-zero bits are shifted out.

```
    rSRB              Shift reg r right one bit.
    rSRB.F  fs     ** Shift reg r(fs) right one bit.
    rSLC              Shift reg r left circular one nibble.
    rSRC              Shift reg r right circular one nibble.
    rSL     fs        Shift reg r(fs) left one nibble.
    rSR     fs        Shift reg r(fs) right one nibble.
```

6.11.10 Arithmetic_Instructions

NOTE: There is no s=r-s fs instruction. This means these instructions are not available on the Saturn CPU: B=A-B, C=B-C, A=C-A, and C=D-C.

```
    r=0     fs         Set r(fs) to zero.
    r=r-1   fs         Decrement r(fs); alters Carry.
    r=r+1   fs         Increment r(fs); alters Carry.
    r=s     fs         Copy s(fs) to r(fs).
    s=r     fs         Copy r(fs) to s(fs).
    rsEX    fs         Exchange r(fs) and s(fs).
    srEX    fs         Alias for rsEX    fs.
    r=r+CON fs,d    ** Add d to r(fs); alters Carry.
    r=r+r   fs         Add r(fs) to itself; alters Carry.
    r=r+s   fs         Add s(fs) to r(fs); alters Carry.
    r=s+r   fs         Alias for r=r+s   fs.
    s=r+s   fs         Add r(fs) to s(fs); alters Carry.
    s=s+r   fs         Alias for s=r+s   fs.
    r=r-CON fs,d    ** Subtract d to r(fs); alters Carry.
    r=r-s   fs         Subtract s(fs) from r(fs); alters Carry.
    s=s-r   fs         Subtract r(fs) from s(fs); alters Carry.
    r=s-r   fs         Subtract r(fs) from s(fs), put result in
                       r(fs); alters Carry.
    r=-r    fs         2's or 10's complement of r(fs); clear
                       Carry if r(fs) was zero, else set Carry.
    r=-r-1  fs         1's or 9's complement of r(fs);
                       unconditionally clear Carry.
```

6.11.11 Logical_Operation_Instructions

```
    r=r&s   fs         Bit-wise AND of register r(fs) with
                       register s(fs).
    r=r!s   fs         Bit-wise OR of register r(fs) with
                       register s(fs).
```

6.11.12 No-Operation_Instructions

```
    NOP3               Three nibble No-op.
    NOP4               Four nibble No-op.
    NOP5               Five nibble No-op.
```

6.12 Pseudo-Op Instructions

NOTE: The label field is ignored by some of the pseudo-op instructions. These instructions ignore the label field: CHARMAP, CLRCARRY, CLRFLAG, CLRLIST, EJECT, INCLUDE, LISTALL, LISTM, LIST, MESSAGE, NOTREACHED, RDSYMB, SETCARRY, SETFLAG, SETLIST, STITLE, TITLE, and UNLIST.

6.12.1 Data_Storage_Allocation

BSS	nnnnn	Allocate nnnnn zero nibbles here.
CON(m)	expr	Generate m-nibble constant. The constant is stored with the least significant nibble at the lowest address. [1 _ m _ 8]
REL(m)	label	Generate m-nibble relative offset. The offset is stored with the least significant nibble at the lowest address. [1 _ m _ 8]
NIBASC	\ASCII\	Generate ASCII characters. Each character is stored with the least significant nibble at the lowest address. The first character is placed at the lowest address. [40 characters maximum]
STRING	\ASCII\	Generate ASCII characters, set the high bit on the last character. Each character is stored with the least significant nibble at the lowest address. The first character is placed at the lowest address. [40 characters maximum]
NIBHEX	hhhhhhh	Generate hexadecimal nibbles. The first nibble is placed at the lowest address. [80 nibbles max]
NIBFS	fs	Generate the field selection nibble for field fs. The opcode representation used is from column a in the Field Select Table.
LINK	label	Generate five nibble relative offset to the next LINK reference to label. The value of the offset is filled in by the linker.
SLINK	label	Generate five nibble relative offset to the first LINK reference to label. The value of the offset is filled in by the linker.
INC(m)	label	Generate an m-nibble reference to label which is passed to the linker. The label must be an external symbol. The linker fills in the position of the INC(m) reference to label. For example, if a file contains three INC(3) =label references, the first INC(3) will be filled in as 000, the second INC(3) will be filled in as 100, and the third INC(3) will be filled in as 200 (least significant nibble of the position at the lowest address).

6.13 Conditional Assembly

Conditional assembly pseudo-ops allow alternate versions of assembly code to be assembled dependent on some specific conditions. An optional label on the conditional assembly statements allows nesting.

Conditions which can be tested include assembly flags set on invocation, the value of an expression compared to zero, the relationship between two strings, the presence or absence of a symbol, the current carry state (for example, after GOYES or RTNYES, the carry is clear), and whether a specific mnemonic is available in this assembly (dependent on the processor level selected).

label	IF	expr	Assemble code only if flag expr is set.
label	IFEQ	expr	Assemble code only if expr is zero.
label	IFNE	expr	Assemble code only if expr is non-zero.
label	IFLT	expr	Assemble code only if expr is less than zero.
label	IFLE	expr	Assemble code only if expr is less than or equal to zero.
label	IFGT	expr	Assemble code only if expr is greater than zero.
label	IFGE	expr	Assemble code only if expr is greater than or equal to zero.
label	IFZER	expr	Alias for IFEQ.
label	IFNZ	expr	Alias for IFNE.
label	IFNEG	expr	Alias for IFLT.
label	IFPOS	expr	Alias for IFGT.
label	IFDEF	symbol	Assemble code only if symbol is defined now.
label	IFNDEF	symbol	Assemble code only if symbol is not defined now.
label	IFOPC	symbol	Assemble code only if symbol is a valid opcode mnemonic.
label	IFNOPC	symbol	Assemble code only if symbol is not a valid opcode mnemonic.
label	IFPASS1		Assemble code only if this is the first pass of the assembler. This is most useful in conjunction with the MESSAGE pseudo-op.
label	IFPASS2		Assemble code only if this is the second pass of the assembler. This is most useful in conjunction with the MESSAGE pseudo-op.
label	IFANYCARRY		Assemble code only if the carry can't be determined at assembly time.
label	IFCARRYCLR		Assemble code only if the carry can be determined at assembly time and the carry is clear.

label	IFCARRYSET	Assemble code only if the carry can be determined at assembly time and the carry is set.
label	IFREACHED	Assemble code only if the current statement can be reached.
label	ELSE	Reverse the sense of the IF test with label "label".
label	ENDIF	End conditional assembly started by IF with label "label".

6.14 Listing Control

When no listing file is being generated (-N option), these pseudo-ops have no effect on the assembly.

TITLE	text	Set title to text (at most one TITLE instruction is permitted per file).
STITLE	text	Set subtitle to text and force a new page in the assembly listing.
EJECT		Force a new page in the assembly listing.
UNLIST		Turn off assembly listing except for some pseudo-ops.
LIST		Turn on assembly listing.
LISTM		Turn on assembly listing for macro expansion and include files.
LISTALL	expr	Unconditionally list the next expr lines. LISTALL is independent of LIST and UNLIST. If expr is less than or equal to zero or is not a legal expression, disable LISTALL mode.
CLRLIST	type	Turn off assembly listing of type, where type is one or more of { CODE, MACRO, INCLUDE, PSEUDO, ALL }. If the type includes NOLIST, the CLRLIST line is not listed.
SETLIST	type	Turn on assembly listing of type, where type is one or more of { CODE, MACRO, INCLUDE, PSEUDO, ALL }. If the type includes NOLIST, the SETLIST line is not listed.

LIST is an alias for SETLIST CODE. LISTM is an alias for SETLIST MACRO,INCLUDE. UNLIST is an alias for CLRLIST CODE,MACRO,INCLUDE.

6.15 Symbol Definition

symbol	EQU	expr	Assigns the value expr to symbol. If symbol is already defined, EQU generates an error.
symbol	=	expr	Assigns the value expr to symbol. If symbol is already defined, it is given

the new value expr.

6.16 Macro Definition

label	MACRO	Start definition of macro label.
label	ENDM	End definition of macro label.
	EXITM	If reached while interpreting a macro, terminate that macro interpretation immediately. EXITM has no effect during the definition of a macro.

6.17 Assembly Mode

symbol	ABS	nnnnn	Specify an absolute assembly starting at address nnnnn.
symbol	REL	nnnnn	Specify a relocatable assembly starting at address nnnnn.
	END		Terminate assembly with this line. Any lines following the END instruction are ignored.

6.18 File Access

These pseudo-ops allow access to other files. This allows commonly-used symbols, macros, etc. to be defined in a file shared by several assembly files.

RDSYMB	file	Read the symbol table from the Saturn file named file. Each symbol which is defined, external, and not relocatable is made available for the duration of this assembly.
INCLUDE	file	Read assembly source statements from file until either an END instruction is read or an End-of-File condition occurs.
CHARMAP	file	Read a set of character mappings from file. Each line in the file consists of an ASCII character, followed by the character which should be assembled when the character is used in an ASCII string. (See "Charmap File Format").

6.19 Assembly Flag Modification

CLRFLAG	expr	Clear assembly flag expr.
SETFLAG	expr	Set assembly flag expr.

6.20 Carry State Modification

CLRCARRY	Indicate to the assembler that the carry is always clear at this point.
SETCARRY	Indicate to the assembler that the carry is always set at this point.
NOTREACHED	Indicate to the assembler that this point is never reached by assembly code.

6.21 Miscellaneous

MESSAGE text	Write text to the standard error location. The message is written once for each assembler pass. This is most useful when tracking down symbols which change between passes.

7. Saturn Assembly Tips

This chapter summarizes some general advice accumulated over many many man-years of Saturn programming in Corvallis. While no doubt incomplete, these should spare some agony.

7.1 Three Warnings

The following three "gotcha's" are lessons that have been learned repeatedly by every Saturn programmer. You have been warned.

7.1.1 Return_Levels

If you have experience with standard processors, be aware that this one has a fixed number of return stack levels (8). As the interrupt system uses two of these whenever an interrupt occurs (which generally can be any time), HP 48 programmers are limited to a maximum of 6 levels. If code you write is called, you will be further limited. The symptoms of violation will include a Warmstart, but may include more severe effects. ...So watch those levels.

7.1.2 Mode

One of the big features of the Saturn processor, is also the programmers bane. The processor supports both DEC and HEX modes. Code designed to run in HEX mode can behave very badly if invoked from a DEC mode state and vice versa. ...So watch the mode.

7.1.3 Remember_P=0!

Many routines require P=0 as an entry condition (this includes the RPL inner loop by the way). Quite frequently other routines don't care about the value of P on entry, but use it as a resource and exit with it in various states (eg; most of the floating point math routines). This could be generalized to simply "watch entry and exit conditions", but this one seems to happen frequently. ...So watch P=0.

7.2 Code Packing Tips

7.2.1 A-Field_Operations

Frequently, use of the A field for register operations that require only the P,B,X or XS fields is a code saving. Eg; Replace "A=C X" with "A=C A" field to save a nibble if you don't care about nibbles 3 and 4 of A.

7.2.2 Loading_Constants

For loading small constants into a larger field, it is frequently cheaper to clear the field and generate only the "digits" required. For example if kfactor < 256, than you will save a nibble of ROM and get the same effect with

```
        C=0     A
        LC(2)   kfactor
```

instead of

```
        LC(5)   kfactor
```

7.2.3 The_3-Branches

There are 3 varieties of "GOTO" and "GOSUB" that require 4,6, and 7 nibbles of code. Two of these are "relative" branches, and the long one is "absolute". The assembler takes care of all the details, informing you if a branch is out of range, so there is really no drawback to using the shorter versions when appropriate (they also execute faster by the way). As a general rule, references to external routines (routines in the HP 48) should use the long version, and references to routines in your application should use relative branches. The neumonics for the three varieties are:

GOTO	GOSUB	(4 nibs)
GOLONG	GOSUBL	(6 nibs)
GOVLNG	GOSBVL	(7 nibs)

Numerous references to an external routine may be shortened by means of a "jump table". Eg; Replace all "GOSBVL =GETPTR" by calls to the local version ("GOSUB getptr") below.

```
getptr  GOVLNG   =GETPTR
```

7.2.4 GOSUB/RTN

Code that might naturally end with something like

```
        ...
        ...
        GOSUB    dotask8
        RTN
```

will run faster, save 2 nibs, and may make your routine take less stack levels by replacing that combination with

```
        ...
        ...
        GOTO     dotask8
```

7.2.5 Use_Expressions

Use the Saturn Assembler to evaluate certain expressions instead of at run time.

Example

```
        LC(5)    (=TBLADRS)+5*t2
```

instead of

```
        LC(5)    =TBLADRS
        A=C      A
        LC(5)    5*t2
        C=C+A    A
```

7.2.6 Count_Up

Frequently P is used as the control variable for loops that require no more than 16 passes. If the loop is structured so that P is decremented until a carry test causes an exit, the value of P on exit will be 15 (generally not a very useful value). Often the code which follows will reset P to 0. Optionally you can avoid the need to reset P to 0 (save 2 nibs) by counting up. In this way, when the carry occurs, P will have become 0.

Example

```
        P=       16-5       Compute x-5*y;  A:x, C:y
arglp   A=A-C    A
        P=P+1
        GONC     arglp
```

7.2.7 Before_you_leap

One of the advantages of assembly language programming is the plethora of methods available to the innovator. Often the first solution you think of will not be the most code efficient, time efficient, resource efficient, reliable, or easiest to implement.

Speaking of time efficient - this document tells you about execution time. Instruction execution time varies depending on instruction type and the fields that it operates on.

In addition, when executing code out of standard 8-bit wide devices (ala HP 48 256 KByte ROM), instruction timing will vary depending on whether the instruction occurs on an even or odd address. If one measures instruction timing on some sort of consistent scale, you find that instruction times will vary from 2 to 33 units of time. The most time expensive instructions are those which access data in memory. Also expensive are full word (16-nibble) operations. The least expensive are operations on P.

7.3 Some Common Operations

7.3.1 A_nibble_from_here_to_there

This type of operation is usually accomplished by one or more of the following 3 types of CPU instructions. The "Thru P" variety can only be used in the C register.

Register Transfer:
==========
Example - Transfer nibble from C[P] to A[P]

```
        A=C     P
```

Nibble Shifts:
==========
Example - Shift B field of A into nibs 1 and 2 of A

```
        ASL.F   X
```

Thru P:
==========
Example - Copy Sign field of C into nib 4 of C

```
        P=C     15
        C=P     4
```

7.3.2 Testing_a_Bit

This type of operation may be accomplished in a variety of ways depending on where the bit to be tested is located, the state of the CPU, and what CPU resources may be used.

Direct Bit Test:
==========
Generally this is the best choice when it's available (only in nibbles 0-3 of A and C registers), as this is destructive only to the CARRY and depends only on the thing being tested. It also has the advantage of working with symbolic arguments (which makes it easy when the location of the bit is changed).

Example:

```
bEDIT   EQU     6
        ...
        ...
        ?ABIT=1 bEDIT
        GOYES   doEDIT
```

Left (Arithmetic) bit shifts in a field:
==
Left bit shifts require HEX Mode, and are often used in cases where the bit lives in a location other than A[0-3] or C[0-3] and it is too expensive or otherwise undesirable to copy it there. The shift is done arithmetically via HEX mode arithmetic, and is destructive to the field in which the operation is performed. Your code is also dependent on the bit# being tested (not symbolic).

Example - Test bit 2 in A[S]

```
        SETHEX
        A=A+A    S         Shift it to msb
        A=A+A    S         CS iff bit 2 was originally set.
        GOC      bit2_on
```

Right bit shifts in a field:
============================
Right arithmetic shifts are accomplished by provided CPU shift instructions. When a non-zero bit is shifted out the right side of a field, a bit in the CPU known as the "sticky bit" (SB) is set. This bit is "sticky" and must be explicitly cleared before it is used for a bit test. The same example above could be done in either HEX or DEC mode by:

```
        ASRB.F   S         Move bit to position 1
        ASRB.F   S                     position 0
        SB=0               Prepare for test
        ASRB.F   S         SB=1 iff bit 2 originally set.
        ?SB=0
        GOYES    bit2_off
```

The placement of the "SB=0" instruction is important. There is no instruction to test SB=1, hence the test sense reversal.

Mask it out:
============

```
        P=       15
        LCHEX    4         C[S]: 0100  (Mask)
        A=A&C    S         Mask out all bits of non-interest.
        ?A#0     S         bit2 set?
        GOYES    bit2_on   Yes.
```

7.3.3 Saving/Testing_a_State

Frequently it is advantage to record a condition that may later be tested. The 12 CPU (Local) Status Bits (S0-S11) are frequently used for this purpose. The (Global) Status bits (S12-S15) are reserved for recording operating system status. There is a functional difference also. The Global Status bits cannot be swapped in and out of the C-Register like the others. Be careful to document your usage of the status bits, as failure to exercise care here can result in contention for the same status bit and a "Gotcha". As far as usage goes it's quite simple. Use a symbolic name for the status bit (The symbol should be global if the status bit will be referenced in other files).

Example:

```
=sDMY   EQU     8               Day-Month-Year Date Format if Set.
        ...
        ...
        ?ST=1   sDMY            DD.MMYYYY Date Format?
        GOYES   date10          Yes.
        CDEX    B               No.  - Swap DD, MM
date10
```

7.3.4 Memory_Access

The Saturn Processor Reads (Writes) from (to) low order memory nibble wise into (from) the low order nibs of either the A or C registers. You will stay out of trouble if you remember low order of the register to low order memory. Thus an ASCII "A" stored in the B-field of CPU register A will appear as A[B]: 41. If this same value is written out to address #82000, you will see

```
#82000   1
#82001   4
```

If memory is displayed from left to right in increasing addresses (as it is on our development systems), the data will appear as

```
82000:14...
```

making it appear backwards. But all is well. If the 2 nibbles starting at address #82000 are read back into A[B] you will get what you expect (A[B]: 41). The transfers take place by using one of the "Data Pointers" D0, D1 to specify the address.

Example: Write ASCII "A" to address #82000:

```
P=      0
LC(2)    A
D0=(5)  #82000
DAT0=C  B               #82000: "A"
```

7.4 Some Other Tips

7.4.1 Labels

Nothing prohibits you from using global labels on all routines. Don't. Use global labels only on routines that are referenced externally (ie; those that require them) or which might reasonably be called externally in the future. Use "global references" only where required. This permits a code reviewer to know whether or not a routine that is being called is in the same file or not w/o referring to the symbol table. (ie; if I see "GOSUBL =PADDER", I assume that PADDER is in another file - otherwise, why the "=" ?).

7.4.2 Status_Bits

All status bit usage should be symbolic. That is; never write something like "ST=0 5". Instead, this can be "ST=0 sLOAN" where sLOAN has been equated to 5 elsewhere. In addition to the obvious advantage of self documenting status bit usage, this permits relativelty safe changes to the actual status bit in use at a later time. The symbol table identifies all references to sLOAN for the purposes of making changes, avoiding status bit clash, etc.

7.4.3 Entry_Points

I find it is best to not have more than one Global entry in a given routine. The main reason is that such a routine is frequently difficult to maintain. Invariably the entry points will have different entry conditions (else why the separate entry). This tenet may be violated in the interest of code conservation, but always balance it against the maintenence cost. If you do it, and it's not completely obvious from a casual glance what the entry conditions need to be, insert a "mini-header" consisting of a few comment lines explaining what the entry conditions are. Something like

```
;*****************************************************************
;** Here:   A: Last Arg Count                                    *
;**         C: Loop Counter                                      *
;**         P: 14                                                *
;*****************************************************************
=ARG50
```

7.4.4 Exits

Good "style" suggests that routines should not have more than one normal (non-error) exit. This is something that gets violated with impunity in the interest of code saving. It is not unusual to find cases where use of a common exit actually saves code. It's best to let code cost be the judge. When it is a "push", use a common exit.

7.5 Documentation

7.5.1 Comments_on_Comments

Any line beginning with an "*" is a comment line. Comments are also inserted on the same line as code (no "*" required). Comments can be grouped into one of 3 types:

Module level:
==========
These comments generally describe the nature of the code in the file, perhaps providing a list of the major routines, general conventions and notation that will be used in subsequent documentation, etc. This should appear at the front of the file (before code).

Depending on your style, you may choose to place symbolic equates (eg; status bits, various constants, etc;) at the front of a file (These generate no code, but effect the code that references the symbols). This makes it easy for a reviewer to find their values.

Routine level:
==========
These comments describe the general nature of a routine, it's inputs and outputs, and CPU and RAM resources altered by invocation. These should appear in a header at the front of the routine. Begin the coding process by inserting a standard blank header. Fill in certain fields immediately (eg; name, abstract, implementation date) and others as soon as the code is reasonably stable.

Line level:
==========
Line by line comments will help the reviewer (which often will be you) wade thru the code later. These should appear on the same line as the code or on separate lines near the code being documented.

Guidelines:
==========

1) How Much? Generally, the more the better. However not every line needs a comment, and some short, simple, local routines don't require a header. If a routine has an external entry, it deserves a header.

2) Avoid no content comments (A=C A Copy C[A] to A[A])

3) Avoid comments that document unimportant register contents. This seems to help my focus. For example, reviewers usually prefer #1 over #2:

```
                          #1                    #2
C=D     A                                  C[A]: Last Arg Ptr
A=C     A          A[A]: Last Arg Ptr      A[A]: Last Arg Ptr
P=      0
LC(5)   PROC5      C[A]: Exec Adrs         C[A]: Exec Adrs
```

4) Put content into your comments. They should be easily decipherable, but grammar is unimportant. Don't permit comment lines to be longer than what will be printed on the list (.l) files.

5) The Saturn Assembler is free format allowing you to be quite flexible in your placement of code. DON'T OVERUSE THIS FEATURE. Employ standard fields. The header on the next page has reasonable fields marked (1,9,17,28). The important thing is consistency.

7.5.2 A_Standard_Assembly_Language_Header

Some version of this header is in use by all of the software people generating Saturn Assembly Language Code. There is no law saying you have to use this one, or use one at all. This one works, and has passed the test of time.

```
        EJECT
*****************************************************************
*****************************************************************
** Name: XXXXXXXXXXXX - .......
**
** Category: .....
**
** Abstract: .....
**
** Entry:    .....
**
** Exit:     .....
**
** Error Exits: .....
**
** Alters: CPU - .....
**         RAM - .....
**
** Calls:    .....
**
** Stack Levels: .....
**
** Notes: ......
**
**     Date     Prog             Modification
**   --------   ----   ----------------------------------------
**   04/../91   XX     Implemented.
**===============================================================
=XXXXXX XXXXX   XXXXX      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
12345678901234567890123456789012345678901234567890123456789012345
```

Here is how we use the various fields:

NAME: Routine Name and a 1-Line description of the routine.

CATEGORY: General grouping (eg; MATH, CLKUTL, MEM, etc;)

ABSTRACT: Purpose of the routine, with perhaps minimal details regarding how it does it.

ENTRY: Exactly what CPU entry conditions must be satisfied. Sometimes worthwhile mentioning other required conditions as well (eg; RAM values, Timer State, interrupts disabled, etc.).

EXIT: What conditions can be depended upon for a normal (non error) exit (Eg; where are the results? Does the routine always exit CC? Is there a particular value in P? What about HEX/DEC Mode?)

ERROR EXITS: Same as EXIT, but for the error exits.

ALTERS: What does the routine change (WORST CASE!) for a non-error exit. If no RAM is altered, eliminate that line and just list all CPU registers altered. This includes CARRY, P, MODE, SB, etc. The caller generally doesn't care what resources you use, just what you may have altered. So, do not list resources used, but always restored to their entry state before exit.

CALLS: Names of the routines called before exit back to the caller. It is often useful to indicate the stack levels of the called routine here as well. This assists in filling out the Stack Levels field.

STACK LEVELS: The number of stack levels used by this routine. If this routine has no GOSUB's, and does not employ C=RSTK or RSTK=C, then this will be 0.

NOTES: The place for additional documentation, perhaps an algorithm description, or certain CAVEATS.

HISTORY: Indicate date, programmers initials, and reason for changes. Don't start really using this until you feel the code is stable.

```
*****************************************************************
*****************************************************************
** Name(S): PKDATE   - Pack Date Components  (YYYYMMDD order)
**
** Category: DATEUTL
**
** Purpose:  Facilitates storage of date components in a single
**    CPU register.  Also useful for comparing two dates to
**    determine which occurrs first in time.  Entry conditions
**    are designed so that this routine may easily be called
**    after CKDATE.
**
** Entry:  A[A]: 0YYYY
**         B[B]: MM
**         D[B]: DD
**
** Exit:   C: 00000000YYYYMMDD;  CC;  P=7
**
** Alters:   C;  P;  CARRY
**
** Calls:    None
**
** Stack Levels: 0
**
** Notes:
**
**     Date     Prog             Modification
**   --------   ----   ----------------------------------------
**   08/23/85   SB     Implemented.
*****************************************************************
*****************************************************************
=PKDATE C=0    W              Initialize Result
        C=A    A              Copy YYYY and shift left twice
        CSL    A
        P=     7
        CSL    WP
        C=B    B              Copy MM and shift left twice
        CSL    WP
        CSL    WP
        C=D    B              C: 00000000YYYYMMDD
        RTNCC
*****************************************************************
```

```
        EJECT
*****************************************************************
*****************************************************************
** Name(S): LEAPYR?  - Determine if specified year is leap yr.
**
** Category: DATEUTL
**
** Purpose:  Determine if specified year is a leap year.
**
** Entry:    A[A]: YYYY;   DEC Mode.
**
** Exit:     CS - YYYY is a Leap Year
**           CC - YYYY is not a Leap Year
**
** Alters:   C[A];  SB;  Carry
**
** Calls:    None
**
** Stack Levels: 1
**
** Notes:  Y is a leap year iff both of the following:
**                    1)  Y MOD 4 = 0
**          -AND-     2)  (Y MOD 100 # 0) or (Y MOD 400 =0)
**
**     Date     Prog             Modification
**   --------   ----   ----------------------------------------
**   07/12/85   SB     Implemented.
*****************************************************************
*****************************************************************
=LEAPYR?
        C=A    A              Init X=Y
        RSTK=C                Save Y
          ?A#0   B            Y divisible by 100 ?
          GOYES  LEAP10       No.  Leap yr iff X div by 4
          ASR    A            Yes. Leap yr iff Y div by 400
          ASR    A
          C=A    A            C[A]=A[A]: X=Y/100

*  Reduced to testing whether X divisible by 4.
LEAP10    SB=0                Init SB=0
          C=C+C  A
          C=C+C  A
          C=C+A  A            5*X
          CSR    A            C[A]: X/2;   Sets SB if X odd.
          CSRB                Set SB if X/2 odd
        C=RSTK
        A=C    A              Restore Y
        ?SB=0                 Leap Year?
        RTNYES                Yes.  CS
        RTN                   No.   CC
*****************************************************************
```

```
        EJECT
*****************************************************************
*****************************************************************
**
*R Name: Ticks>DOW - Make Day of Week from Time in Ticks
**
** Category:   TIMESYS
**
** Abstract:   From Time in Ticks (since 0), return a real number
**             integer (1-7) that identifies the day of week.
**
** Stack:  hxs  --> % (Day of Week)
**
** Error Exits:    Insufficient Memory
**
**     Date     Prog             Modification
**   --------   ----   ----------------------------------------
**   03/18/88   SB     Implemented.
**=============================================================
=Ticks>DOW
        CON(5) (*)+5

* Pop Time(ticks) from stack
        A=DAT1 A
        AD1EX               Save D1* in A[A];  D1:->hxs
        D1=D1+ 10           Skip over prologue and length
        C=0    W
        C=DAT1 13
        R0=C                R0: Time (Ticks)
        D1=A                Restore D1
        D1=D1+ 5
        D=D+1  A            Pop hxs and save new pointers
        GOSBVL =SAVPTR

* Convert to Day of Week and put in float form
        C=R0
        GOSBVL =dowutil     A: Day of Week Index (1-7)
        C=A    A
        P=C    0
        C=0    W
        C=P    14
        A=C    W            A: % (DOW: 1-7; 1=SUN)

*  Push % on stack and loop.
        GOTO   push%lp
*****************************************************************
```

8. Mnemonic Dictionary

This section contains a description of each Saturn assembler instruction or pseudo-op. The description shows the binary opcode generated by the mnemonic, if any, as well as the execution cycle time required if the mnemonic is an executable instruction.

?A#0 fs - Test for A not equal to 0

fs = A	opcode:	8ACyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9aCyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of A is not equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A#B fs - Test for A not equal to B

fs = A	opcode:	8A4yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9a4yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of A is not equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A#C fs - Test for A not equal to C

---

fs = A opcode: 8A6yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9a6yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of A is not equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A<=B fs - Test for A less than or equal to B

---

fs = A opcode: 8BCyy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9bCyy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of A is less than or equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A<B fs - Test for A less than B

---

fs = A opcode: 8B4yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9b4yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of A is less than the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A=0 fs - Test for A equal to 0

---

fs = A — opcode: 8A8yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) — opcode: 9a8yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of A is equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A=B fs - Test for A equal to B

---

fs = A — opcode: 8A0yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) — opcode: 9a0yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of A is equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A=C fs - Test for A equal to C

---

fs = A — opcode: 8A2yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) — opcode: 9a2yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of A is equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A>=B fs - Test for A greater than or equal to B

---

fs = A	opcode:	8B8yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9b8yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of A is greater than or equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A>B fs - Test for A greater than B

---

fs = A	opcode:	8B0yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9b0yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of A is greater than the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B#0 fs - Test for B not equal to 0

---

fs = A	opcode:	8ADyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9aDyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of B is not equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B#A fs - Test for B not equal to A

---

fs = A opcode: 8A4yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9a4yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of B is not equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B#C fs - Test for B not equal to C

---

fs = A opcode: 8A5yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9a5yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of B is not equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B<=C fs - Test for B less than or equal to C

---

fs = A	opcode:	8BDyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9bDyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of B is less than or equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B<C   fs - Test for B less than C

---

fs = A	opcode:	8B5yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9b5yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of B is less than the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B=0   fs - Test for B equal to 0

---

fs = A	opcode:	8A9yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9a9yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of B is equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B=A fs - Test for B equal to A
---

fs = A	opcode:	8A0yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9a0yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of B is equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B=C fs - Test for B equal to C
---

fs = A	opcode:	8A1yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9a1yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of B is equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B>=C fs - Test for B greater than or equal to C
---

fs = A	opcode:	8B9yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9b9yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of B is greater than or equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B>C fs - Test for B greater than C

```
fs = A                          opcode:  8B1yy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)        opcode:  9b1yy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)
```

Test whether the fs field of B is greater than the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C#0 fs - Test for C not equal to 0

```
fs = A                          opcode:  8AEyy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)        opcode:  9aEyy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)
```

Test whether the fs field of C is not equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C#A fs - Test for C not equal to A

```
fs = A                          opcode:  8A6yy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)        opcode:  9a6yy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)
```

Test whether the fs field of C is not equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C#B fs - Test for C not equal to B
---

fs = A	opcode:	8A5yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9a5yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of C is not equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C#D fs - Test for C not equal to D
---

fs = A	opcode:	8A7yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9a7yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of C is not equal to the fs field of D. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C<=A fs - Test for C less than or equal to A
---

fs = A	opcode:	8BEyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9bEyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of C is less than or equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C<A fs - Test for C less than A
---

fs = A opcode: 8B6yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9b6yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of C is less than the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C=0 fs - Test for C equal to 0
---

fs = A opcode: 8AAyy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9aAyy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of C is equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C=A fs - Test for C equal to A
---

fs = A opcode: 8A2yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9a2yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of C is equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C=B fs - Test for C equal to B
---

fs = A	opcode:	8A1yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9a1yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of C is equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C=D fs - Test for C equal to D
---

fs = A	opcode:	8A3yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9a3yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of C is equal to the fs field of D. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C>=A fs - Test for C greater than or equal to A
---

fs = A	opcode:	8BAyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9bAyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of C is greater than or equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C>A fs - Test for C greater than A
---

fs = A	opcode:	8B2yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9b2yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of C is greater than the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D#0 fs - Test for D not equal to 0
---

fs = A	opcode:	8AFyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9aFyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of D is not equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D#C fs - Test for D not equal to C
---

fs = A	opcode:	8A7yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9a7yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of D is not equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D<=C fs - Test for D less than or equal to C
---

fs = A	opcode:	8BFyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9bFyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of D is less than or equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D<C   fs - Test for D less than to C
---

fs = A	opcode:	8B7yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9b7yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of D is less than the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D=0   fs - Test for D equal to 0
---

fs = A	opcode:	8AByy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9aByy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of D is equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D=C fs - Test for D equal to C
---

fs = A	opcode:	8A3yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9a3yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of D is equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D>=C fs - Test for D greater than or equal to C
---

fs = A	opcode:	8BByy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9bByy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of D is greater than or equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D>C fs - Test for D greater than C
---

fs = A	opcode:	8B3yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9b3yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of D is greater than the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?MP=0 - Test Module Pulled bit (MP)
---

opcode: 838yy
cycles: 13 (GO/RTNYES)
6 (NO)

Test whether the Module Pulled bit (MP) is zero. This hardware status bit is set whenever a module-pulled interrupt occurs (the *INT line of the CPU is pulled high), and must be explictly cleared by the MP=0 mnemonic. See the "HP-71 Hardware Specification" for more information. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?P# n - Test if P pointer not equal to n
---

opcode: 88nyy
cycles: 13 (GO/RTNYES)
6 (NO)

Test whether the P pointer is not equal to n. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?P= n - Test if P pointer is equal to n
---

opcode: 89nyy
cycles: 13 (GO/RTNYES)
6 (NO)

Test whether the P pointer is equal to n. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?SB=0 - Test Sticky Bit (SB)
---

opcode: 832yy
cycles: 13 (GO/RTNYES)
6 (NO)

Test whether the Sticky Bit (SB) is zero. This hardware status bit is set on right shifts when a non-zero nibble or bit is shifted off the end of the field. The Sticky Bit must be cleared explicitly. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?SR=0 - Test Service Request bit (SR) for zero
---

opcode: 834yy
cycles: 13 (GO/RTNYES)
6 (NO)

Test whether the Service Request bit (SR) is zero. This hardware status bit is set by the SREQ? mnemonic, and must be cleared explicitly by the SR=0 instruciton. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?ST#0 n - Test status bit n not equal to 0
---

opcode: 87nyy
cycles: 14 (GO/RTNYES)
7 (NO)

Test whether Program Status bit n is set. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?ST#1 n - Test status bit n not equal to 1
---

opcode: 86nyy
cycles: 14 (GO/RTNYES)
7 (NO)

Test whether Program Status bit n is clear. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?ST=0 n - Test status bit n equal to 0
---

opcode: 86nyy
cycles: 14 (GO/RTNYES)
7 (NO)

Test whether Program Status bit n is clear. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?ST=1 n   - Test status bit n equal to 1

```
                              opcode:  87nyy
                              cycles:   14 (GO/RTNYES)
                                         7 (NO)
```

Test whether Program Status bit n is set. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?XM=0   - Test External Module Missing bit (XM)

```
                              opcode:  831yy
                              cycles:   13 (GO/RTNYES)
                                         6 (NO)
```

Test the whether the External Module Missing bit (XM) is zero. This hardware status bit is set by the RTNSXM mnemonic, and must be explicitly cleared by the XM=0 mnemonic. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

A=-A   fs - Two's complement of A into A

```
fs = A                        opcode:  F8
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Bb8
                              cycles:    3 + d
```

Complement the specified fs field of A. Complement is two's complement if in HEX mode, ten's complement if in DEC mode. Carry is set if the field is not zero, else Carry is cleared.

A=-A-1 fs - One's complement of A into A

```
fs = A                        opcode:  FC
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  BbC
                              cycles:    3 + d
```

Perform a one's complement on the specified fs field of A. Carry is always cleared.

```
A=0    fs  -  Set A equal to 0
---------
fs = A                        opcode:  D0
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ab0
                              cycles:    3 + d
```

Set the specified fs field of A to zero. Carry is not affected.

```
A=A!B  fs  -  A OR B into A
---------
fs = A                        opcode:  0EF8
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0Ea8
                              cycles:    4 + d
```

Set the fs field of register A to its logical OR with the corresponding field of register B. Carry is not affected.

```
A=A!C  fs  -  A OR C into A
---------
fs = A                        opcode:  0EFE
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0EaE
                              cycles:    4 + d
```

Set the fs field of register A to its logical OR with the corresponding field of register C. Carry is not affected.

```
A=A&B  fs  -  A AND B into A
---------
fs = A                        opcode:  0EF0
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0Ea0
                              cycles:    4 + d
```

Set the fs field of register A to its logical AND with the corresponding field of register B. Carry is not affected.

A=A&C fs - A AND C into A
---

fs = A	opcode: 0EF6 cycles: 4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode: 0Ea6 cycles: 4 + d

Set the fs field of register A to its logical AND with the corresponding field of register C. Carry is not affected.

A=A+1 fs - Increment A
---

fs = A	opcode: E4 cycles: 7
fs = (P,WP,XS,X,S,M,B,W)	opcode: Ba4 cycles: 3 + d

Increment the specified fs field of register A by one. Adjusts Carry.

A=A+A fs - Sum of A and A into A
---

fs = A	opcode: C4 cycles: 7
fs = (P,WP,XS,X,S,M,B,W)	opcode: Aa4 cycles: 3 + d

Double the specified fs field of register A. Adjusts Carry.

A=A+B fs - Sum of A and B into A
---

fs = A	opcode: C0 cycles: 7
fs = (P,WP,XS,X,S,M,B,W)	opcode: Aa0 cycles: 3 + d

Set the specified fs field of register A to the sum of itself and the corresponding field of register B. Adjusts Carry.

A=A+C fs - Sum of A and C into A
---

fs = A	opcode:	CA
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AaA
	cycles:	3 + d

Set the specified fs field of register A to the sum of itself and the corresponding field of register C. Adjusts Carry.

A=A-1 fs - Decrement A
---

fs = A	opcode:	CC
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AaC
	cycles:	3 + d

Decrement the specified fs field of register A by one. Adjusts Carry.

A=A-B fs - A minus B into A
---

fs = A	opcode:	E0
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Ba0
	cycles:	3 + d

Set the specified fs field of register A to the difference between itself and the corresponding field of register B. Adjusts Carry.

A=A-C fs - A minus C into A
---

fs = A	opcode:	EA
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	BaA
	cycles:	3 + d

Set the specified fs field of register A to the difference between itself and the corresponding field of register C. Adjusts Carry.

A=B fs - Copy B to A

```
fs = A                          opcode:  D4
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  Ab4
                                cycles:    3 + d
```

Copy the fs field of register B into the corresponding field of register A. Carry is not affected.

A=B-A fs - B minus A into A

```
fs = A                          opcode:  EC
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  BaC
                                cycles:    3 + d
```

Set the specified fs field of register A to the inverse difference between itself and the corresponding field of register B. Adjusts Carry.

A=C fs - Copy C to A

```
fs = A                          opcode:  DA
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  AbA
                                cycles:    3 + d
```

Copy the fs field of register C into the corresponding field of register A. Carry is not affected.

A=DAT0 fsd - Load A from memory

fs = A	opcode:	142
	cycles:	18
fs = B	opcode:	14A
	cycles:	15
fs = (P,WP,XS,X,S,M,W)	opcode:	152a
	cycles:	17 + d
fs = d	opcode:	15Ax (x=d-1)
	cycles:	16 + d

The amount of data (d nibbles) specified by fsd will be transferred from the memory address pointed to by D0 into the specified field of register A. The lowest-addressed nibble will be transferred into the lowest-order nibble of the register field, proceeding toward the higher-order nibbles. If fs = d, d nibbles are transferred into the register starting at nibble 0. See the section on "Loading Data From Memory".

A=DAT1 fsd - Load A from memory

fs = A	opcode:	143
	cycles:	18
fs = B	opcode:	14B
	cycles:	15
fs = (P,WP,XS,X,S,M,W)	opcode:	153a
	cycles:	17 + d
fs = d	opcode:	15Bx (x=d-1)
	cycles:	16 + d

The amount of data (d nibbles) specified by fsd will be transferred from the memory address pointed to by D1 into the specified field of register A. The lowest-addressed nibble will be transferred into the lowest-order nibble of the register field, proceeding toward the higher-order nibbles. If fs = d, d nibbles are transferred into the register starting at nibble 0. See the section on "Loading Data From Memory".

A=IN - Load A with IN
---

opcode: 802
cycles: 7

Load the low-order 4 nibbles of the A register with the contents of the Input register.

A=R0 - Copy R0 to A
---

opcode: 110
cycles: 19

The contents of the scratch register R0 is copied to the working register A.

A=R1 - Copy R1 to A
---

opcode: 111
cycles: 19

The contents of the scratch register R1 is copied to the working register A.

A=R2 - Copy R2 to A
---

opcode: 112
cycles: 19

The contents of the scratch register R2 is copied to the working register A.

A=R3 - Copy R3 to A
---

opcode: 113
cycles: 19

The contents of the scratch register R3 is copied to the working register A.

A=R4 - Copy R4 to A

opcode: 114
cycles: 19

The contents of the scratch register R4 is copied to the working register A.

ABEX fs - Exchange Registers A and B

fs = A — opcode: DC, cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: AbC, cycles: 3 + d

Exchange the fs fields of registers of A and B. Carry is not affected.

ACEX fs - Exchange Registers A and C

fs = A — opcode: DE, cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: AbE, cycles: 3 + d

Exchange the fs fields of registers of A and C. Carry is not affected.

AD0EX - Exchange A and D0 (nibs 0-4)

opcode: 132
cycles: 8

Exchange the A field of register A with Data pointer D0. Carry is not affected.

AD0XS - Exchange A and D0 short (nibs 0-3)
---

opcode: 13A
cycles: 7

Exchange the lower 4 nibbles of A with the lower 4 nibbles of Data pointer D0. Carry is not affected.

AD1EX - Exchange A and D1 (nibs 0-4)
---

opcode: 133
cycles: 8

Exchange the A field of register A with Data pointer D1. Carry is not affected.

AD1XS - Exchange A and D1 short (nibs 0-3)
---

opcode: 13B
cycles: 7

Exchange the lower 4 nibbles of A with the lower 4 nibbles of Data pointer D1. Carry is not affected.

AR0EX - Exchange A and R0
---

opcode: 120
cycles: 19

Exchange the contents of the working register A and the scratch register R0.

AR1EX - Exchange A and R1
---

opcode: 121
cycles: 19

Exchange the contents of the working register A and the scratch register R1.

AR2EX - Exchange A and R2

opcode: 122
cycles: 19

Exchange the contents of the working register A and the scratch register R2.

AR3EX - Exchange A and R3

opcode: 123
cycles: 19

Exchange the contents of the working register A and the scratch register R3.

AR4EX - Exchange A and R4

opcode: 124
cycles: 19

Exchange the contents of the working register A and the scratch register R4.

ASL fs - A Shift Left

fs = A — opcode: F0, cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: Bb0, cycles: 3 + d

Shift the contents of the specified fs field of register A left one nibble, without affecting the rest of the register. The nibble shifted off the left end of the field is lost. The new low-order nibble of the field is zero. The Sticky Bit (SB) is not affected.

ASLC - A Shift Left Circular

opcode: 810
cycles: 21

Circular shift register A left one nibble. Operates on all 16 digits. The Sticky Bit (SB) is not affected.

ASR fs - A Shift Right

fs = A opcode: F4
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: Bb4
cycles: 3 + d

Shift the contents of the specified fs field of register A right one nibble, without affecting the rest of the register. The nibble shifted off the right end of the field is lost, but the Sticky Bit (SB) is set if the nibble was non-zero. The new high-order nibble of the field is zero.

ASRB - A Shift Right Bit

opcode: 81C
cycles: 20

Shift register A right one bit. Operates on all 16 digits. The bit shifted off the end is lost, but the Sticky Bit (SB) is set if it was non-zero. The new high-order bit of the register is zero.

ASRC - A Shift Right Circular

opcode: 814
cycles: 21

Circular shift register A right one nibble. Operates on all 16 digits. The Sticky Bit (SB) is set if the nibble shifted from low-order around to high-order position was non-zero.

B=-B fs - Two's complement of B into B

fs = A — opcode: F9, cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: Bb9, cycles: 3 + d

Complement the specified fs field of B. Complement is two's complement if in HEX mode, ten's complement if in DEC mode. Carry is set if the field is not zero, else Carry is cleared.

B=-B-1 fs - One's complement of B into B

fs = A — opcode: FD, cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: BbD, cycles: 3 + d

Perform a one's complement on the specified fs field of B. Carry is always cleared.

B=0 fs - Set B equal to 0

fs = A — opcode: D1, cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: Ab1, cycles: 3 + d

Set the specified fs field of B to zero. Carry is not affected.

B=A fs - Copy A to B

fs = A — opcode: D8, cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: Ab8, cycles: 3 + d

Copy the fs field of register A into the corresponding field of register B. Carry is not affected.

B=B!A fs - B OR A into B
---

fs = A	opcode: 0EFC cycles: 4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode: 0EaC cycles: 4 + d

Set the fs field of register B to its logical OR with the corresponding field of register A. Carry is not affected.

B=B!C fs - B OR C into B
---

fs = A	opcode: 0EF9 cycles: 4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode: 0Ea9 cycles: 4 + d

Set the fs field of register B to its logical OR with the corresponding field of register C. Carry is not affected.

B=B&A fs - B AND A into B
---

fs = A	opcode: 0EF4 cycles: 4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode: 0Ea4 cycles: 4 + d

Set the fs field of register B to its logical AND with the corresponding field of register A. Carry is not affected.

B=B&C fs - B AND C into B
---

fs = A	opcode: 0EF1 cycles: 4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode: 0Ea1 cycles: 4 + d

Set the fs field of register B to its logical AND with the corresponding field of register C. Carry is not affected.

B=B+1 fs - Increment B

fs = A — opcode: E5; cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: Ba5; cycles: 3 + d

Increment the specified fs field of register B by one. Adjusts Carry.

B=B+A fs - Sum of B and A into B

fs = A — opcode: C8; cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: Aa8; cycles: 3 + d

Set the specified fs field of register B to the sum of itself and the corresponding field of register A. Adjusts Carry.

B=B+B fs - Sum of B and B into B

fs = A — opcode: C5; cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: Aa5; cycles: 3 + d

Double the specified fs field of register B. Adjusts Carry.

B=B+C fs - Sum of B and C into B

fs = A — opcode: C1; cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: Aa1; cycles: 3 + d

Set the specified fs field of register B to the sum of itself and the corresponding field of register C. Adjusts Carry.

B=B-1 fs - Decrement B
---

fs = A	opcode: CD cycles: 7
fs = (P,WP,XS,X,S,M,B,W)	opcode: AaD cycles: 3 + d

Decrement the specified fs field of register B by one.
Adjusts Carry.

B=B-A fs - B minus A into B
---

fs = A	opcode: E8 cycles: 7
fs = (P,WP,XS,X,S,M,B,W)	opcode: Ba8 cycles: 3 + d

Set the specified fs field of register B to the difference
between itself and the corresponding field of register A.
Adjusts Carry.

B=B-C fs - B minus C into B
---

fs = A	opcode: E1 cycles: 7
fs = (P,WP,XS,X,S,M,B,W)	opcode: Ba1 cycles: 3 + d

Set the specified fs field of register B to the difference
between itself and the corresponding field of register C.
Adjusts Carry.

B=C fs - Copy C to B
---

fs = A	opcode: D5 cycles: 7
fs = (P,WP,XS,X,S,M,B,W)	opcode: Ab5 cycles: 3 + d

Copy the fs field of register C into the corresponding field
of register B. Carry is not affected.

B=C-B fs - C minus B into B
---

fs = A	opcode:	ED
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	BaD
	cycles:	3 + d

Set the specified fs field of register B to the inverse difference between itself and the corresponding field of register C. Adjusts Carry.

BAEX fs - Exchange Registers B and A
---

fs = A	opcode:	DC
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AbC
	cycles:	3 + d

Exchange the fs fields of registers of B and A. Carry is not affected.

BCEX fs - Exchange Registers B and C
---

fs = A	opcode:	DD
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AbD
	cycles:	3 + d

Exchange the fs fields of registers of B and C. Carry is not affected.

BSL fs - B Shift Left
---

fs = A	opcode:	F1
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Bb1
	cycles:	3 + d

Shift the contents of the specified fs field of register B left one nibble, without affecting the rest of the register. The nibble shifted off the left end of the field is lost. The new low-order nibble of the field is zero. The Sticky Bit (SB) is not affected.

BSLC - B Shift Left Circular
---

opcode: 811
cycles: 21

Circular shift register B left one nibble. Operates on all 16 digits. The Sticky Bit (SB) is not affected.

BSR fs - B Shift Right
---

fs = A opcode: F5
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: Bb5
cycles: 3 + d

Shift the contents of the specified fs field of register B right one nibble, without affecting the rest of the register. The nibble shifted off the right end of the field is lost, but the Sticky Bit (SB) is set if the nibble was non-zero. The new high-order nibble of the field is zero.

BSRB - B Shift Right Bit
---

opcode: 81D
cycles: 20

Shift register B right one bit. Operates on all 16 digits. The bit shifted off the end is lost, but the Sticky Bit (SB) is set if it was non-zero. The new high-order bit of the register is zero.

BSRC - B Shift Right Circular
---

opcode: 815
cycles: 21

Circular shift register B right one nibble. Operates on all 16 digits. The Sticky Bit (SB) is set if the nibble shifted from low-order around to high-order position was non-zero.

BUSCC - Bus Command "C"
---

opcode: 80B
cycles: 6

Enters the Saturn bus command "C" onto the system bus (this command is reserved for later use). No other operation is performed. See the "HP-71 Hardware Specification" for more information.

C+P+1 - Increment C by One Plus P Pointer
---

opcode: 809
cycles: 8

The A field of the C register is incremented by one plus the value of the P pointer. This instruction is always executed in HEX mode. Adjusts Carry.

C=-C fs - Two's complement of C into C
---

fs = A opcode: FA
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: BbA
cycles: 3 + d

Complement the specified fs field of C. Complement is two's complement if in HEX mode, ten's complement if in DEC mode. Carry is set if the field is not zero, else Carry is cleared.

C=-C-1 fs - One's complement of C into C
---

fs = A opcode: FE
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: BbE
cycles: 3 + d

Perform a one's complement on the specified fs field of C. Carry is always cleared.

C=0 fs - Set C equal to 0

```
fs = A                          opcode:  D2
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  Ab2
                                cycles:    3 + d
```

Set the specified fs field of C to zero. Carry is not affected.

C=A fs - Copy A to C

```
fs = A                          opcode:  D6
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  Ab6
                                cycles:    3 + d
```

Copy the fs field of register A into the corresponding field of register C. Carry is not affected.

C=A-C fs - A minus C into C

```
fs = A                          opcode:  EE
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  BaE
                                cycles:    3 + d
```

Set the specified fs field of register C to the inverse difference between itself and the corresponding field of register A. Adjusts Carry.

C=B fs - Copy B to C

```
fs = A                          opcode:  D9
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  Ab9
                                cycles:    3 + d
```

Copy the fs field of register B into the corresponding field of register C. Carry is not affected.

C=C!A fs - C OR A into C
---

fs = A	opcode: 0EFA cycles: 4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode: 0EaA cycles: 4 + d

Set the fs field of register C to its logical OR with the corresponding field of register A. Carry is not affected.

C=C!B fs - C OR B into C
---

fs = A	opcode: 0EFD cycles: 4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode: 0EaD cycles: 4 + d

Set the fs field of register C to its logical OR with the corresponding field of register B. Carry is not affected.

C=C!D fs - C OR D into C
---

fs = A	opcode: 0EFF cycles: 4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode: 0EaF cycles: 4 + d

Set the fs field of register C to its logical OR with the corresponding field of register D. Carry is not affected.

C=C&A fs - C AND A into A
---

fs = A	opcode: 0EF2 cycles: 4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode: 0Ea2 cycles: 4 + d

Set the fs field of register C to its logical AND with the corresponding field of register A. Carry is not affected.

C=C&B fs - C AND B into C
---

fs = A	opcode:	0EF5
	cycles:	4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode:	0Ea5
	cycles:	4 + d

Set the fs field of register C to its logical AND with the corresponding field of register B. Carry is not affected.

C=C&D fs - C AND D into C
---

fs = A	opcode:	0EF7
	cycles:	4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode:	0Ea7
	cycles:	4 + d

Set the fs field of register C to its logical AND with the corresponding field of register D. Carry is not affected.

C=C+1 fs - Increment C
---

fs = A	opcode:	E6
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Ba6
	cycles:	3 + d

Increment the specified fs field of register C by one. Adjusts Carry.

C=C+A fs - Sum of C and A into C
---

fs = A	opcode:	C2
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Aa2
	cycles:	3 + d

Set the specified fs field of register C to the sum of itself and the corresponding field of register A. Adjusts Carry.

C=C+B fs - Sum of C and B into C
---

fs = A	opcode:	C9
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Aa9
	cycles:	3 + d

Set the specified fs field of register C to the sum of itself and the corresponding field of register B. Adjusts Carry.

C=C+C fs - Sum of C and C into C
---

fs = A	opcode:	C6
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Aa6
	cycles:	3 + d

Double the specified fs field of register C. Adjusts Carry.

C=C+D fs - Sum of C and D into C
---

fs = A	opcode:	CB
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AaB
	cycles:	3 + d

Set the specified fs field of register C to the sum of itself and the corresponding field of register D. Adjusts Carry.

C=C-1 fs - Decrement C
---

fs = A	opcode:	CE
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AaE
	cycles:	3 + d

Decrement the specified fs field of register C by one. Adjusts Carry.

C=C-A fs - C minus A into C
---
fs = A opcode: E2
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: Ba2
cycles: 3 + d

Set the specified fs field of register C to the difference between itself and the corresponding field of register A. Adjusts Carry.

C=C-B fs - C minus B into C
---
fs = A opcode: E9
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: Ba9
cycles: 3 + d

Set the specified fs field of register C to the difference between itself and the corresponding field of register B. Adjusts Carry.

C=C-D fs - C minus D into C
---
fs = A opcode: EB
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: BaB
cycles: 3 + d

Set the specified fs field of register C to the difference between itself and the corresponding field of register D. Adjusts Carry.

C=D fs - Copy D to C
---
fs = A opcode: DB
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: AbB
cycles: 3 + d

Copy the fs field of register D into the corresponding field of register C. Carry is not affected.

C=DAT0 fsd - Load C from memory

fs = A	opcode:	146
	cycles:	18
fs = B	opcode:	14E
	cycles:	15
fs = (P,WP,XS,X,S,M,W)	opcode:	156a
	cycles:	17 + d
fs = d	opcode:	15Ex (x=d-1)
	cycles:	16 + d

The amount of data (d nibbles) specified by fsd will be transferred from the memory address pointed to by D0 into the specified field of register C. The lowest-addressed nibble will be transferred into the lowest-order nibble of the register field, proceeding toward the higher-order nibbles. If fs = d, d nibbles are transferred into the register starting at nibble 0. See the section on "Loading Data From Memory".

C=DAT1 fsd - Load C from memory

fs = A	opcode:	147
	cycles:	18
fs = B	opcode:	14F
	cycles:	15
fs = (P,WP,XS,X,S,M,W)	opcode:	157a
	cycles:	17 + d
fs = d	opcode:	15Fx (x=d-1)
	cycles:	16 + d

The amount of data (d nibbles) specified by fsd will be transferred from the memory address pointed to by D1 into the specified field of register C. The lowest-addressed nibble will be transferred into the lowest-order nibble of the register field, proceeding toward the higher-order nibbles. If fs = d, d nibbles are transferred into the register starting at nibble 0. See the section on "Loading Data From Memory".

C=ID - Request chip ID
---

opcode: 806
cycles: 11

The chip which has its DAISY-IN line high and its configuration flag low will send its 5 nibble ID register to the system bus which will be loaded into the low-order 5 nibbles (A field) of the C register.

C=IN - Load C with IN
---

opcode: 803
cycles: 7

Load the low-order 4 nibbles of the C register with the contents of the Input register.

C=P n - Copy P Pointer into Nibble n of C
---

opcode: 80Cn
cycles: 6

Copy P pointer into C register at digit position specified by n.

C=R0 - Copy R0 to C
---

opcode: 118
cycles: 19

The contents of the scratch register R0 is copied to the working register C.

C=R1 - Copy R1 to C
---

opcode: 119
cycles: 19

The contents of the scratch register R1 is copied to the working register C.

C=R2 - Copy R2 to C

opcode: 11A
cycles: 19

The contents of the scratch register R2 is copied to the working register C.

C=R3 - Copy R3 to C

opcode: 11B
cycles: 19

The contents of the scratch register R3 is copied to the working register C.

C=R4 - Copy R4 to C

opcode: 11C
cycles: 19

The contents of the scratch register R4 is copied to the working register C.

C=RSTK - Pop stack to C

opcode: 07
cycles: 8

Pop the top-most address off of the hardware return stack, placing the address in the lower 5 nibbles (A field) of register C. The high-order nibbles of C are unchanged. As the address is popped from the return stack, a zero address is inserted at the bottom of the stack. Compare with the RTN mnemonic.

C=ST - Status to C

opcode: 09
cycles: 6

Copy the low-order 12 bits of the status register into the low-order 12 bits (X field) of the C register.

CAEX fs - Exchange Registers C and A

fs = A — opcode: DE, cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: AbE, cycles: 3 + d

Exchange the fs fields of registers of C and A. Carry is not affected.

CBEX fs - Exchange Registers C and B

fs = A — opcode: DD, cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: AbD, cycles: 3 + d

Exchange the fs fields of registers of C and B. Carry is not affected.

CD0EX - Exchange C and D0 (nibs 0-4)

opcode: 136
cycles: 8

Exchange the A field of register C with Data pointer D0. Carry is not affected.

CD0XS - Exchange C and D0 short (nibs 0-3)
---

opcode: 13E
cycles: 7

Exchange the lower 4 nibbles of C with the lower 4 nibbles of Data pointer D0. Carry is not affected.

CD1EX - Exchange C and D1 (nibs 0-4)
---

opcode: 137
cycles: 8

Exchange the A field of register C with Data pointer D1. Carry is not affected.

CD1XS - Exchange C and D1 short (nibs 0-3)
---

opcode: 13F
cycles: 7

Exchange the lower 4 nibbles of C with the lower 4 nibbles of Data pointer D1. Carry is not affected.

CDEX fs - Exchange Registers C and D
---

fs = A
opcode: DF
cycles: 7

fs = (P,WP,XS,X,S,M,B,W)
opcode: AbF
cycles: 3 + d

Exchange the fs fields of registers of C and D. Carry is not affected.

CLRHST - Clear Hardware Status bits
---

opcode: 82F
cycles: 3

Clears the 4 Hardware Status bits XM, SB, SR and MP. Note that the opcode is actually 82x, where x is merely a mask for which Hardware Status bits to clear, as follows:

bit 0 - External Module Missing bit (see XM=0 mnemonic)
bit 1 - Sticky Bit (see SB=0 mnemonic)
bit 2 - Service Request bit (see SR=0 mnemonic)
bit 3 - Module Pulled bit (see MP=0 mnemonic)

For example opcode 829 clears XM and MP. Although there is no mnemonic for this, the opcode can be inserted into the code by using, for example, NIBHEX 829.

CLRST - Clear Program Status
---

opcode: 08
cycles: 6

Clear the low-order 12 bits (S0 through S11) of the Program Status register ST.

CONFIG - Configure
---

opcode: 805
cycles: 11

Copy the low-order 5 nibbles (A field) of the C register into the Configuration register of the chip which has its DAISY-IN line high and its configuration flag low. See the "HP-71 Hardware Specification" for information.

CPEX n - Exchange Nibble n of C With P Pointer
---

opcode: 80Fn
cycles: 6

Exchange the P pointer with digit n of the C register.

CR0EX - Exchange C and R0

opcode: 128
cycles: 19

Exchange the contents of the working register C and the scratch register R0.

CR1EX - Exchange C and R1

opcode: 129
cycles: 19

Exchange the contents of the working register C and the scratch register R1.

CR2EX - Exchange C and R2

opcode: 12A
cycles: 19

Exchange the contents of the working register C and the scratch register R2.

CR3EX - Exchange C and R3

opcode: 12B
cycles: 19

Exchange the contents of the working register C and the scratch register R3.

CR4EX - Exchange C and R4

opcode: 12C
cycles: 19

Exchange the contents of the working register C and the scratch register R4.

CSL fs - C Shift Left

fs = A — opcode: F2, cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: Bb2, cycles: 3 + d

Shift the contents of the specified fs field of register C left one nibble, without affecting the rest of the register. The nibble shifted off the left end of the field is lost. The new low-order nibble of the field is zero. The Sticky Bit (SB) is not affected.

CSLC - C Shift Left Circular

opcode: 812
cycles: 21

Circular shift register C left one nibble. Operates on all 16 digits. The Sticky Bit (SB) is not affected.

CSR fs - C Shift Right

fs = A — opcode: F6, cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: Bb6, cycles: 3 + d

Shift the contents of the specified fs field of register C right one nibble, without affecting the rest of the register. The nibble shifted off the right end of the field is lost, but the Sticky Bit (SB) is set if the nibble was non-zero. The new high-order nibble of the field is zero.

CSRB - C Shift Right Bit

opcode: 81E, cycles: 20

Shift register C right one bit. Operates on all 16 digits. The bit shifted off the end is lost, but the Sticky Bit (SB) is set if it was non-zero. The new high-order bit of the register is zero.

CSRC - C Shift Right Circular

opcode: 816, cycles: 21

Circular shift register C right one nibble. Operates on all 16 digits. The Sticky Bit (SB) is set if the nibble shifted from low-order around to high-order position was non-zero.

CSTEX - Exchange Status

opcode: 0B, cycles: 6

Exchange the low-order 12 bits (S0 through S11) of the Program Status register ST with the low-order 12 bits of the C register.

D0=(2) nn - Load 2 Nibbles Into D0

opcode: 19nn
cycles: 4

Load the low-order two nibbles of D0 with nn. The upper nibbles of D0 remain unchanged. Any overflow is ignored by the assembler. The assembled digits of nn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D0=(4) nnnn - Load 4 Nibbles Into D0

opcode: 1Annnn
cycles: 6

Load the low-order four nibbles of D0 with nnnn. The upper nibble of D0 remains unchanged. Any overflow is ignored by the assembler. The assembled digits of nnnn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D0=(5) nnnnn - Load 5 Nibbles Into D0

opcode: 1Bnnnnn
cycles: 7

Load all five nibbles of D0 with nnnnn. Any overflow is ignored by the assembler. The assembled digits of nnnnn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D0=A - Copy A to D0 (nibs 0-4)
---

opcode: 130
cycles: 8

The A field of register A is copied into Data pointer register D0. Carry is not affected.

D0=AS - Copy A to D0 short (nibs 0-3)
---

opcode: 138
cycles: 7

The lower 4 nibbles of A are copied into the lower 4 nibbles of Data pointer register D0. Carry is not affected.

D0=C - Copy C to D0 (nibs 0-4)
---

opcode: 134
cycles: 8

The A field of register C is copied into Data pointer register D0. Carry is not affected.

D0=CS - Copy C to D0 short (nibs 0-3)
---

opcode: 13C
cycles: 7

The lower 4 nibbles of C are copied into the lower 4 nibbles of Data pointer register D0. Carry is not affected.

D0=D0+ n - Add n to D0 (1<=n<=16)
---

opcode: 16x (x=n-1)
cycles: 7

Increment D0 by n. This instruction is always executed in HEX mode. Adjusts Carry.

D0=D0- n - Subtract n from D0 (1<=n<=16) ---

opcode: 18x (x=n-1)
cycles: 7

Decrement D0 by n. This instruction is always executed in HEX mode. Adjusts Carry.

D0=HEX hh - Load D0 with hex constant hh
---

opcode: 19hh
cycles: 4

Load the low-order two nibbles of D0 with the hex constant hh. The upper nibbles of D0 remain unchanged. The digits of hh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D0=HEX hhhh - Load D0 with hex constant hhhh
---

opcode: 1Ahhhh
cycles: 6

Load the low-order four nibbles of D0 with the hex constant hhhh. The upper nibble of D0 remains unchanged. The digits of hhhh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D0=HEX hhhhh - Load D0 with hex constant hhhhh
---

opcode: 1Bhhhhh
cycles: 7

Load all five nibbles of D0 with the hex constant hhhhh. The digits of hhhhh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D1=(2) nn   - Load 2 Nibbles Into D1
---

opcode: 1Dnn
cycles: 4

Load the low-order two nibbles of D1 with nn. The upper nibbles of D1 remain unchanged. Any overflow is ignored by the assembler. The assembled digits of nn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D1=(4) nnnn - Load 4 Nibbles Into D1
---

opcode: 1Ennnn
cycles: 6

Load the low-order four nibbles of D1 with nnnn. The upper nibble of D1 remains unchanged. Any overflow is ignored by the assembler. The assembled digits of nnnn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D1=(5) nnnnn - Load 5 Nibbles Into D1
---

opcode: 1Fnnnnn
cycles: 7

Load all five nibbles of D1 with nnnnn. Any overflow is ignored by the assembler. The assembled digits of nnnnn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D1=A   - Copy A to D1 (nibs 0-4)
---

opcode: 131
cycles: 8

The A field of register A is copied into Data pointer register D1. Carry is not affected.

D1=AS   - Copy A to D1 short (nibs 0-3)
---

opcode: 139
cycles: 7

The lower 4 nibbles of A are copied into the lower 4 nibbles of Data pointer register D1. Carry is not affected.

D1=C   - Copy C to D1 (nibs 0-4)
---

opcode: 135
cycles: 8

The A field of register C is copied into Data pointer register D1. Carry is not affected.

D1=CS - Copy C to D1 short (nibs 0-3)
---

opcode: 13D
cycles: 7

The lower 4 nibbles of C are copied into the lower 4 nibbles of Data pointer register D1. Carry is not affected.

D1=D1+ n - Add n to D1 (1<=n<=16)
---

opcode: 17x (x=n-1)
cycles: 7

Increment D1 by n. This instruction is always executed in HEX mode. Adjusts Carry.

D1=D1- n - Subtract n from D1 (1<=n<=16)
---

opcode: 1Cx (x=n-1)
cycles: 7

Decrement D1 by n. This instruction is always executed in HEX mode. Adjusts Carry.

D1=HEX hh - Load D1 with hex constant hh
---

opcode: 1Dhh
cycles: 4

Load the low-order two nibbles of D1 with the hex constant hh. The upper nibbles of D1 remain unchanged. The digits of hh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D1=HEX hhhh - Load D1 with hex constant hhhh

```
-----------
                              opcode:  1Ehhhh
                              cycles:     6
```

Load the low-order four nibbles of D1 with the hex constant hhhh. The upper nibble of D1 remains unchanged. The digits of hhhh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D1=HEX hhhhh - Load D1 with hex constant hhhhh

```
------------
                              opcode:  1Fhhhhh
                              cycles:     7
```

Load all five nibbles of D1 with the hex constant hhhhh. The digits of hhhhh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

D=-D   fs - Two's complement of D into D

```
---------
fs = A                        opcode:  FB
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  BbB
                              cycles:    3 + d
```

Complement the specified fs field of D. Complement is two's complement if in HEX mode, ten's complement if in DEC mode. Carry is set if the field is not zero, else Carry is cleared.

D=-D-1 fs - One's complement of D into D
---

fs = A	opcode:	FF
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	BbF
	cycles:	3 + d

Perform a one's complement on the specified fs field of D.
Carry is always cleared.

D=0 fs - Set D equal to 0
---

fs = A	opcode:	D3
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Ab3
	cycles:	3 + d

Set the specified fs field of D to zero. Carry is not
affected.

D=C fs - Copy C to D
---

fs = A	opcode:	D7
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Ab7
	cycles:	3 + d

Copy the fs field of register C into the corresponding field
of register D. Carry is not affected.

D=C-D fs - C minus D into D
---

fs = A	opcode: ED cycles: 7
fs = (P,WP,XS,X,S,M,B,W)	opcode: BaD cycles: 3 + d

Set the specified fs field of register D to the inverse difference between itself and the corresponding field of register C. Adjusts Carry.

D=D!C fs - D OR C into D
---

fs = A	opcode: 0EFB cycles: 4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode: 0EaB cycles: 4 + d

Set the fs field of register D to its logical OR with the corresponding field of register C. Carry is not affected.

D=D&C fs - D AND C into D
---

fs = A	opcode: 0EF3 cycles: 4 + d
fs = (P,WP,XS,X,S,M,B,W)	opcode: 0Ea3 cycles: 4 + d

Set the fs field of register D to its logical AND with the corresponding field of register C. Carry is not affected.

D=D+1 fs - Increment D

---

fs = A	opcode:	E7
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Ba7
	cycles:	3 + d

Increment the specified fs field of register D by one. Adjusts Carry.

D=D+C fs - Sum of D and C into D

---

fs = A	opcode:	C3
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Aa3
	cycles:	3 + d

Set the specified fs field of register D to the sum of itself and the corresponding field of register C. Adjusts Carry.

D=D+D fs - Sum of D and D into D

---

fs = A	opcode:	C7
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Aa7
	cycles:	3 + d

Double the specified fs field of register D. Adjusts Carry.

D=D-1 fs - Decrement D
---

fs = A	opcode:	CF
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AaF
	cycles:	3 + d

Decrement the specified fs field of register D by one. Adjusts Carry.

D=D-C fs - D minus C into D
---

fs = A	opcode:	E3
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Ba3
	cycles:	3 + d

Set the specified fs field of register D to the difference between itself and the corresponding field of register C. Adjusts Carry.

DAT0=A fsd - Load memory from A
---

fs = A	opcode:	140
	cycles:	17
fs = B	opcode:	148
	cycles:	14
fs = (P,WP,XS,X,S,M,W)	opcode:	150a
	cycles:	16 + d
fs = d	opcode:	158x (x=d-1)
	cycles:	15 + d

The amount of data (d nibbles) specified by fsd will be written to the memory address pointed to by D0 from the specified field of register A. The lowest-order nibble of the register field will be written to the lowest-addressed nibble of memory, proceeding toward the higher-order nibbles. If fs = d, d nibbles are written to memory starting from nibble 0 of the register. See the section on "Storing Data Into Memory".

DAT0=C fsd - Store into memory from C
---

fs = A	opcode: 144 cycles: 17
fs = B	opcode: 14C cycles: 14
fs = (P,WP,XS,X,S,M,W)	opcode: 154a cycles: 16 + d
fs = d	opcode: 15Cx (x=d-1) cycles: 15 + d

The amount of data (d nibbles) specified by fsd will be written to the memory address pointed to by D0 from the specified field of register C. The lowest-order nibble of the register field will be written to the lowest-addressed nibble of memory, proceeding toward the higher-order nibbles. If fs = d, d nibbles are written to memory starting from nibble 0 of the register. See the section on "Storing Data Into Memory".

DAT1=A fs - Store into memory from A
---

fs = A	opcode: 141 cycles: 17
fs = B	opcode: 149 cycles: 14
fs = (P,WP,XS,X,S,M,W)	opcode: 151a cycles: 16 + d
fs = d	opcode: 159x (x=d-1) cycles: 15 + d

The amount of data (d nibbles) specified by fsd will be written to the memory address pointed to by D1 from the specified field of register A. The lowest-order nibble of the register field will be written to the lowest-addressed nibble of memory, proceeding toward the higher-order nibbles. If fs = d, d nibbles are written to memory starting from nibble 0 of the register. See the section on "Storing Data Into Memory".

DAT1=C fsd - Store into memory from C
---

fs = A	opcode:	145
	cycles:	17
fs = B	opcode:	14D
	cycles:	14
fs = (P,WP,XS,X,S,M,W)	opcode:	155a
	cycles:	16 + d
fs = d	opcode:	15Dx (x=d-1)
	cycles:	15 + d

The amount of data (d nibbles) specified by fsd will be written to the memory address pointed to by D1 from the specified field of register C. The lowest-order nibble of the register field will be written to the lowest-addressed nibble of memory, proceeding toward the higher-order nibbles. If fs = d, d nibbles are written to memory starting from nibble 0 of the register. See the section on "Storing Data Into Memory".

DCEX fs - Exchange Registers D and C
---

fs = A	opcode:	DF
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AbF
	cycles:	3 + d

Exchange the fs fields of registers of D and C. Carry is not affected.

DSL fs - D Shift Left
---

fs = A	opcode:	F3
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Bb3
	cycles:	3 + d

Shift the contents of the specified fs field of register D left one nibble, without affecting the rest of the register. The nibble shifted off the left end of the field is lost. The new low-order nibble of the field is zero. The Sticky Bit (SB) is not affected.

DSLC - D Shift Left Circular

opcode: 813
cycles: 21

Circular shift register D left one nibble. Operates on all 16 digits. The Sticky Bit (SB) is not affected.

DSR fs - D Shift Right

fs = A opcode: F7
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: Bb7
cycles: 3 + d

Shift the contents of the specified fs field of register D right one nibble, without affecting the rest of the register. The nibble shifted off the right end of the field is lost, but the Sticky Bit (SB) is set if the nibble was non-zero. The new high-order nibble of the field is zero.

DSRB - D Shift Right Bit

opcode: 81F
cycles: 20

Shift register D right one bit. Operates on all 16 digits. The bit shifted off the end is lost, but the Sticky Bit (SB) is set if it was non-zero. The new high-order bit of the register is zero.

DSRC - D Shift Right Circular

opcode: 817
cycles: 21

Circular shift register D right one nibble. Operates on all 16 digits. The Sticky Bit (SB) is set if the nibble shifted from low-order around to high-order position was non-zero.

GOC label - Go relative on carry

```
                              opcode:  4aa (Carry=0)
                              cycles:   10 (GO)
                                         3 (NO)
```

Short relative jump to label if Carry is set. label must be in the range:

addr - 128 <= label <= addr + 127

where addr is the address of the second nibble of the opcode. The address offset aa is in two's complement form and is relative to addr.

GOLONG label - Go Long

```
                              opcode:  8Caaaa
                              cycles:   14
```

Long relative jump to label unconditionally. label must be in the range:

addr - 32768 <= label <= addr + 32767

where addr is the address of the third nibble of the opcode. The address offset aaaa is in two's complement form and is relative to addr.

GONC label - Go relative on no carry

```
                              opcode:  5aa (Carry=1)
                              cycles:   10 (GO)
                                         3 (NO)
```

Short relative jump to label if Carry is clear. label must be in the range:

addr - 128 <= label <= addr + 127

where addr is the address of the second nibble of the opcode. The address offset aa is in two's complement form and is relative to addr.

GOSBVL label - Gosub very long to label
---

opcode: 8Faaaaa
cycles: 15

Absolute subroutine jump to aaaaa, which is the absolute address of label. See the GOSUB mnemonic.

GOSUB label - Gosub to label
---

opcode: 7aaa
cycles: 12

Relative subroutine jump to label. label must be in the range:

addr - 2048 <= label <= addr + 2047

where addr is the starting address of the next instruction. The address offset aaa is in two's complement form and is relative to addr.

As with all subroutine jumps, the address (addr) of the instruction following the gosub opcode is pushed onto the hardware return stack, so that when a corresponding return is executed, control resumes with the instruction at address addr.

As the return address is pushed onto the return stack, the bottom-most address on the stack is discarded. Therefore, the return stack always contains 8 addresses, and if pushes exceed pops by 8 levels, the bottom-most return addresses are lost. Since the interrupt system requires one level to process interrupts, only 7 levels of the return stack can be used by code which must execute when interrupts are enabled. See the RTN mnemonic for further information.

GOSUBL label - Gosub long to label
---

opcode: 8Eaaaa
cycles: 15

Long relative subroutine jump to label. label must be in the range:

addr - 32768 <= label <= addr + 32767

where addr is the starting address of the next instruction. The address offset aaaa is in two's complement form and is relative to addr. See the GOSUB mnemonic.

GOTO label - Jump relative
---

opcode: 6aaa
cycles: 11

Relative jump to label unconditionally. label must be in the range:

addr - 2048 <= label <= addr + 2047

where addr is the address of the second nibble of the opcode. The address offset aaa is in two's complement form and is relative to addr.

GOVLNG label - Jump very long
---

opcode: 8Daaaaa
cycles: 14

Unconditional jump to aaaaa, which is the absolute address of label.

GOYES label - Jump if Test is True
---

opcode: yy
cycles: included in the accompaning Test mnemonic cycle time.

GOYES is a mnemonic to specify part of a CPU test opcode. GOYES must always follow a test mnemonic. If the condition of the test is met, a jump is performed to label with Carry set. label must be in the range

addr - 128 <= label <= addr + 127

where addr is the starting address of the jump offset yy. If the test condition is not met, Carry is cleared and control passes to the next instruction. Compare with RTNYES.

INTOFF - Interrupt Off
---

opcode: 808F
cycles: 5

Disable the keyboard interrupt system.

INTON - Interrupt On
---

opcode: 8080
cycles: 5

Enable the keyboard interrupt system. See the "Hp-71 Hardware Specification" for more information.

LC(m) n..n - Load C with constant (1<=m<=6)
---

opcode: 3xn..n (x=m-1)
cycles: 3+m

Load m digits of the expression n..n to the C register beginning at the P pointer position, and proceeding toward higher-order nibbles, with the ability to wrap around the register. See the section on "Loading Data From Memory".

LCASC A..A - Load C with ASCII constant
---

opcode: 3mc..c
(m = 2*(# of chars)-1;
c..c = ASCII codes)
cycles: 3+2*(# of chars)

Load up to 8 ASCII characters to the C register beginning at the P pointer position, and proceeding toward higher-order nibbles, with the ability to wrap around the register. Each A represents an ASCII character. The ASCII characters are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

LCHEX h..h - Load C with hex constant
---

opcode: 3nh..h (n=# of digits-1)
cycles: 4+n

Load up to 16 hex digits into the C register beginning at the P pointer position, and proceeding toward higher-order nibbles, with the ability to wrap around the register. The hex digits are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory".

MP=0   - Clear Module Pulled bit (MP)
---

opcode: 828
cycles: 3

Clears the Module Pulled bit (MP) and pulls the Module Pulled Interrupt line low. See CLRHST mnemonic.

NOP3   - Three nibble No-op
---

opcode: 420
cycles: 10 (GO/RTNYES)
          3 (NO)

This mnemonic generates a GOC to the next instruction, effectively skiping three nibbles.

NOP4 - Four nibble No-op
----

opcode: 6300
cycles: 11

This mnemonic generates a GOTO to the next instruction, efectively skiping four nibbles.

NOP5 - Five nibble No-op
----

opcode: 64000
cycles: 11

This mnemonic generates a relative GOTO to +4 nibbles. The fifth nibble in the opcode is a place holder and is jumped over. The mnemonic effectively skips five nibbles.

OUT=C - Load 3 nibbles of OR -----

opcode: 801
cycles: 6

All nibbles of the Output register are loaded with the low-order three nibbles of C (X field).

OUT=CS - Load 1 nibble of OR
------

opcode: 800
cycles: 4

The least significant nibble of the Output register is loaded with the least significant nibble of the C register.

PC=(A) - Jump (Set PC) indirectly thru A field of A register
---

opcode: 808C
cycles: 23

This instruction causes the CPU to jump to the address pointed to by memory at the address specified by the A field of the A register. In symbolic form, the operation is PC=mem(A[A]). This opcode is not available on the 1LF2 version of the Saturn CPU.

P=C n - Copy P pointer into C at Nibble n
---

opcode: 80Dn
cycles: 6

Copy nibble n of register C into the P pointer.

P=P+1 - Increment P Pointer
---

opcode: 0C
cycles: 3

Increment the P pointer. If P is incremented past F it will automatically wrap around to 0. This instruction is always executed in HEX mode. Adjusts carry.

P=P-1 - Decrement P Pointer
---

opcode: 0D
cycles: 3

Decrement the P pointer. If P is decremented past 0 it automatically wraps around to F. This instruction is always executed in HEX mode. Adjusts Carry.

P= n - Set P Pointer to n
---

opcode: 2n
cycles: 2

Set the P pointer to n.

R0=A - Copy A to register R0
---

opcode: 100
cycles: 19

The contents of the working register A is copied to the scratch register R0.

R0=C - Copy C to register R0
---

opcode: 108
cycles: 19

The contents of the working register C is copied to the scratch register R0.

R1=A - Copy A to register R1
---

opcode: 101
cycles: 19

The contents of the working register A is copied to the scratch register R1.

R1=C - Copy C to register R1

---

opcode: 109
cycles: 19

The contents of the working register C is copied to the scratch register R1.

R2=A - Copy A to register R2

---

opcode: 102
cycles: 19

The contents of the working register A is copied to the scratch register R2.

R2=C - Copy C to register R2

---

opcode: 10A
cycles: 19

The contents of the working register C is copied to the scratch register R2.

R3=A - Copy A to register R3

---

opcode: 103
cycles: 19

The contents of the working register A is copied to the scratch register R3.

R3=C - Copy C to register R3

opcode: 10B
cycles: 19

The contents of the working register C is copied to the scratch register R3.

R4=A - Copy A to register R4

opcode: 104
cycles: 19

The contents of the working register A is copied to the scratch register R4.

R4=C - Copy C to register R4

opcode: 10C
cycles: 19

The contents of the working register C is copied to the scratch register R4.

RESET - System reset

opcode: 80A
cycles: 6

The System Reset Bus Command is issued with all chips performing a local reset. The reset function will vary according to the chip type.

RSI - Reset Interrupt System

opcode: 80810
cycles: 6

This instruction causes CPU to consider any input line (ie input register bits) presently high as a new interrupt. If the CPU is presently in the interrupt routine it will wait for an RTI before vectoring, otherwise the CPU will vector immediately following the RSI instruction. For a complete discussion on the interrupt system see the CPU Hardware Specification (A-1LK7-9005-1). This instruction is not available on the 1LF2 version of the Saturn CPU.

RSTK=C - Push C to Return Stack

opcode: 06
cycles: 8

Push the low-order 5 nibbles (A field) of the C register onto the Return Stack. See the GOSUB mnemonic.

RTI - Return from interrupt

opcode: 0F
cycles: 9

Return and re-enable the interrupt system. See the RTN mnemonic.

RTN - Return

opcode: 01
cycles: 9

Return control to the top address on the hardware return stack. The top address on the hardware return stack is popped off and placed in the program counter PC. As the address is popped off the stack, a zero address is inserted at the bottom of the stack.

Therefore the the hardware return stack always contains 8 addresses, and if more pops (returns) than pushes (gosubs) are performed, zeros will be read off the stack. Such an attempt to "return" to address 0 results in a memory reset, since the memory reset code of the operating system resides at address 0.

RTNC -Return on carry

```
                              opcode:  400
                              cycles:   10 (RTN)
                                         3 (NO)
```

Return if Carry is set. See RTN mnemonic.

RTNCC - Return, clear carry

```
                              opcode:  03
                              cycles:    9
```

Return and set Carry. See RTN mnemonic.

RTNNC - Return on no carry

```
                              opcode:  500 (Carry=1)
                              cycles:   10 (RTN)
                                         3 (NO)
```

Return if Carry is not set. See RTN mnemonic.

RTNSC - Return, set carry
---

opcode: 02
cycles: 9

Return and set Carry. See RTN mnemonic.

RTNSXM - Return, set External Module Missing bit (XM)
---

opcode: 00
cycles: 9

Return and set the External Module Missing bit (XM). Since the opcode is zero, this mnemonic is executed on a jump to a non-existent memory device. See the "HP-71 Hardware Specification" for more information. See also the RTN mnemonic.

RTNYES - Return if Test is True
---

opcode: 00
cycles: included in the accompaning mnemonic cycle time.

RTNYES is a mnemonic to specify part of a CPU test opcode. RTNYES must always follow a test mnemonic. If the test condition is met, Carry is set and a return is executed. If the test condition is not met, control passes to the instruction following the RTNYES. Compare with the RTN and GOYES mnemonics.

SB=0 - Clear Sticky Bit (SB)
---

opcode: 822
cycles: 3

Clear the Sticky Bit (SB). See CLRHST mnemonic.

SETDEC - Set decimal
---

opcode: 05
cycles: 3

Set CPU arithmetic mode to decimal.

SETHEX - Set hexadecimal mode
---

opcode: 04
cycles: 3

Set CPU arithmetic mode to hexadecimal.

SHUTDN - System Shutdown
---

opcode: 807
cycles: 5

When this mnemonic is executed the CPU sends out the Shutdown Bus Command and stops its clock. Issuing the SHUTDN command with the output register=000 (see OUT=C or OUT=CS) will cause the PC to be set to zero, causing an automatic cold-start in some systems, or a system halt in the HP 48.

SR=0 - Clear Service Request bit (SR)

opcode: 824
cycles: 3

Clear the Service Request bit (SR). See the CLRHST mnemonic.

SREQ? - Service Request

opcode: 80E
cycles: 7

This mnemonic sets the Service Request bit (SR) if any chip on the system bus requests service. When it is executed, a Service Request Bus Command is issued on the system bus to poll all chips for a Service Request. If any chip requests service, a bus line will be pulled high during the next strobe following the Service Request Bus Command. This value of the bus will be latched into the least significant nibble of the C register. The bus line pulled high determines the device type according to the following table.

Bit	Device
3	Unused
2	Card Reader
1	HP-IL Mailbox
0	Display Driver (timer)

If any bus line is high, the Service Request bit (SR) will be set. See the "HP-71 Hardware Specification" for more information. See also the ?SREQ and SR=0 mnemonics.

ST=0 n - Clear Program Status bit n
---

opcode: 84n
cycles: 4

Clear the Program Status bit selected by n.

ST=1 n - Set Program Status bit n
---

opcode: 85n
cycles: 4

Set the Program Status bit selected by n.

ST=C - C to Status
---

opcode: 0A
cycles: 6

Copy the low-order 12 bits of the Status register (X field) into the low-order 12 bits of the C register.

UNCNFG - Unconfigure
---

opcode: 804
cycles: 12

Load the low-order 5 nibbles (A field) of the C register into each Data pointer with the device addressed by the Data pointer unconfiguring. See the "HP-71 Hardware Specification" for more information.

XM=0 - Clear External Module Missing bit (XM)
---

opcode: 821
cycles: 3

Clear the External Module Missing bit (XM). This hardware status bit is set by the RTNSXM mnemonic. See the CLRHST mnemonic.

9. Alphabetic Mnemonic List

This chapter contains all the code-generating instructions which are recognized by SASM.EXE. Instructions found in the HP 28 and HP 48 but not the HP 71 are marked with **.

Symbols used in modifier field:

fs	Field Select character.
rfs	Restricted Field Selection (S,P,WP, and XS not allowed).
d	Single-nibble field.
expr	Expression.
hh	Two-digit hexadecimal value.
hhhh	Four-digit hexadecimal value.
hhhhh	Five-digit hexadecimal value.
label	Label destination.
ASCII	ASCII character string.

Symbols used in code field:

tt	Code included in test instruction.
a	Field select in the range 0-7.
b	Field select in the range 8-F.
c	Single-nibble length field (Load Constant).
f	Field select including A field.
h...h	Hexadecimal value.
n	Nibble whose value is d.
n...n	Nibbles whose value is expr.
m	Nibble whose value is d - 1.
x...x	Nibbles corresponding to ASCII characters.
y	Reference to a symbol (relative or absolute).

Instruction		Code	Comments
?A#0	A	8ACyy	Requires GOYES or RTNYES; Affects Carry
?A#0	fs	9aCyy	Requires GOYES or RTNYES; Affects Carry
?A#B	A	8A4yy	Requires GOYES or RTNYES; Affects Carry
?A#B	fs	9a4yy	Requires GOYES or RTNYES; Affects Carry
?A#C	A	8A6yy	Requires GOYES or RTNYES; Affects Carry
?A#C	fs	9a6yy	Requires GOYES or RTNYES; Affects Carry
?A<=B	A	8BCyy	Requires GOYES or RTNYES; Affects Carry
?A<=B	fs	9bCyy	Requires GOYES or RTNYES; Affects Carry
?A<=C	A	8BAyy	Requires GOYES or RTNYES; Affects Carry
?A<=C	fs	9bAyy	Requires GOYES or RTNYES; Affects Carry
?A<B	A	8B4yy	Requires GOYES or RTNYES; Affects Carry
?A<B	fs	9b4yy	Requires GOYES or RTNYES; Affects Carry
?A<C	A	8B2yy	Requires GOYES or RTNYES; Affects Carry
?A<C	fs	9b2yy	Requires GOYES or RTNYES; Affects Carry
?A=0	A	8A8yy	Requires GOYES or RTNYES; Affects Carry
?A=0	fs	9a8yy	Requires GOYES or RTNYES; Affects Carry
?A=B	A	8A0yy	Requires GOYES or RTNYES; Affects Carry
?A=B	fs	9a0yy	Requires GOYES or RTNYES; Affects Carry
?A=C	A	8A2yy	Requires GOYES or RTNYES; Affects Carry
?A=C	fs	9a2yy	Requires GOYES or RTNYES; Affects Carry
?A>=B	A	8B8yy	Requires GOYES or RTNYES; Affects Carry
?A>=B	fs	9b8yy	Requires GOYES or RTNYES; Affects Carry

```
?A>=C   A       8BEyy   Requires GOYES or RTNYES; Affects Carry
?A>=C   fs      9bEyy   Requires GOYES or RTNYES; Affects Carry
?A>B    A       8B0yy   Requires GOYES or RTNYES; Affects Carry
?A>B    fs      9b0yy   Requires GOYES or RTNYES; Affects Carry
?A>C    A       8B6yy   Requires GOYES or RTNYES; Affects Carry
?A>C    fs      9b6yy   Requires GOYES or RTNYES; Affects Carry
?ABIT=0 d       8086nyy ** Requires GOYES or RTNYES; Affects Carry
?ABIT=1 d       8087nyy ** Requires GOYES or RTNYES; Affects Carry
?B#0    A       8ADyy   Requires GOYES or RTNYES; Affects Carry
?B#0    fs      9aDyy   Requires GOYES or RTNYES; Affects Carry
?B#A    A       8A4yy   Requires GOYES or RTNYES; Affects Carry
?B#A    fs      9a4yy   Requires GOYES or RTNYES; Affects Carry
?B#C    A       8A5yy   Requires GOYES or RTNYES; Affects Carry
?B#C    fs      9a5yy   Requires GOYES or RTNYES; Affects Carry
?B<=A   A       8B8yy   Requires GOYES or RTNYES; Affects Carry
?B<=A   fs      9b8yy   Requires GOYES or RTNYES; Affects Carry
?B<=C   A       8BDyy   Requires GOYES or RTNYES; Affects Carry
?B<=C   fs      9bDyy   Requires GOYES or RTNYES; Affects Carry
?B<A    A       8B0yy   Requires GOYES or RTNYES; Affects Carry
?B<A    fs      9b0yy   Requires GOYES or RTNYES; Affects Carry
?B<C    A       8B5yy   Requires GOYES or RTNYES; Affects Carry
?B<C    fs      9b5yy   Requires GOYES or RTNYES; Affects Carry
?B=0    A       8A9yy   Requires GOYES or RTNYES; Affects Carry
?B=0    fs      9a9yy   Requires GOYES or RTNYES; Affects Carry
?B=A    A       8A0yy   Requires GOYES or RTNYES; Affects Carry
?B=A    fs      9a0yy   Requires GOYES or RTNYES; Affects Carry
?B=C    A       8A1yy   Requires GOYES or RTNYES; Affects Carry
?B=C    fs      9a1yy   Requires GOYES or RTNYES; Affects Carry
?B>=A   A       8BCyy   Requires GOYES or RTNYES; Affects Carry
?B>=A   fs      9bCyy   Requires GOYES or RTNYES; Affects Carry
?B>=C   A       8B9yy   Requires GOYES or RTNYES; Affects Carry
?B>=C   fs      9b9yy   Requires GOYES or RTNYES; Affects Carry
?B>A    A       8B4yy   Requires GOYES or RTNYES; Affects Carry
?B>A    fs      9b4yy   Requires GOYES or RTNYES; Affects Carry
?B>C    A       8B1yy   Requires GOYES or RTNYES; Affects Carry
?B>C    fs      9b1yy   Requires GOYES or RTNYES; Affects Carry
?C#0    A       8AEyy   Requires GOYES or RTNYES; Affects Carry
?C#0    fs      9aEyy   Requires GOYES or RTNYES; Affects Carry
?C#A    A       8A6yy   Requires GOYES or RTNYES; Affects Carry
?C#A    fs      9a6yy   Requires GOYES or RTNYES; Affects Carry
?C#B    A       8A5yy   Requires GOYES or RTNYES; Affects Carry
?C#B    fs      9a5yy   Requires GOYES or RTNYES; Affects Carry
?C#D    A       8A7yy   Requires GOYES or RTNYES; Affects Carry
?C#D    fs      9a7yy   Requires GOYES or RTNYES; Affects Carry
?C<=A   A       8BEyy   Requires GOYES or RTNYES; Affects Carry
?C<=A   fs      9bEyy   Requires GOYES or RTNYES; Affects Carry
?C<=B   A       8B9yy   Requires GOYES or RTNYES; Affects Carry
?C<=B   fs      9b9yy   Requires GOYES or RTNYES; Affects Carry
?C<=D   A       8BByy   Requires GOYES or RTNYES; Affects Carry
?C<=D   fs      9bByy   Requires GOYES or RTNYES; Affects Carry
?C<A    A       8B6yy   Requires GOYES or RTNYES; Affects Carry
?C<A    fs      9b6yy   Requires GOYES or RTNYES; Affects Carry
?C<B    A       8B1yy   Requires GOYES or RTNYES; Affects Carry
?C<B    fs      9b1yy   Requires GOYES or RTNYES; Affects Carry
?C<D    A       8B3yy   Requires GOYES or RTNYES; Affects Carry
?C<D    fs      9b3yy   Requires GOYES or RTNYES; Affects Carry
```

?C=0	A	8AAyy	Requires GOYES or RTNYES; Affects Carry
?C=0	fs	9aAyy	Requires GOYES or RTNYES; Affects Carry
?C=A	A	8A2yy	Requires GOYES or RTNYES; Affects Carry
?C=A	fs	9a2yy	Requires GOYES or RTNYES; Affects Carry
?C=B	A	8A1yy	Requires GOYES or RTNYES; Affects Carry
?C=B	fs	9a1yy	Requires GOYES or RTNYES; Affects Carry
?C=D	A	8A3yy	Requires GOYES or RTNYES; Affects Carry
?C=D	fs	9a3yy	Requires GOYES or RTNYES; Affects Carry
?C>=A	A	8BAyy	Requires GOYES or RTNYES; Affects Carry
?C>=A	fs	9bAyy	Requires GOYES or RTNYES; Affects Carry
?C>=B	A	8BDyy	Requires GOYES or RTNYES; Affects Carry
?C>=B	fs	9bDyy	Requires GOYES or RTNYES; Affects Carry
?C>=D	A	8BFyy	Requires GOYES or RTNYES; Affects Carry
?C>=D	fs	9bFyy	Requires GOYES or RTNYES; Affects Carry
?C>A	A	8B2yy	Requires GOYES or RTNYES; Affects Carry
?C>A	fs	9b2yy	Requires GOYES or RTNYES; Affects Carry
?C>B	A	8B5yy	Requires GOYES or RTNYES; Affects Carry
?C>B	fs	9b5yy	Requires GOYES or RTNYES; Affects Carry
?C>D	A	8B7yy	Requires GOYES or RTNYES; Affects Carry
?C>D	fs	9b7yy	Requires GOYES or RTNYES; Affects Carry
?CBIT=0	d	808Anyy	** Requires GOYES or RTNYES; Affects Carry
?CBIT=1	d	808Bnyy	** Requires GOYES or RTNYES; Affects Carry
?D#0	A	8AFyy	Requires GOYES or RTNYES; Affects Carry
?D#0	fs	9aFyy	Requires GOYES or RTNYES; Affects Carry
?D#C	A	8A7yy	Requires GOYES or RTNYES; Affects Carry
?D#C	fs	9a7yy	Requires GOYES or RTNYES; Affects Carry
?D<=C	A	8BFyy	Requires GOYES or RTNYES; Affects Carry
?D<=C	fs	9bFyy	Requires GOYES or RTNYES; Affects Carry
?D<C	A	8B7yy	Requires GOYES or RTNYES; Affects Carry
?D<C	fs	9b7yy	Requires GOYES or RTNYES; Affects Carry
?D=0	A	8AByy	Requires GOYES or RTNYES; Affects Carry
?D=0	fs	9aByy	Requires GOYES or RTNYES; Affects Carry
?D=C	A	8A3yy	Requires GOYES or RTNYES; Affects Carry
?D=C	fs	9a3yy	Requires GOYES or RTNYES; Affects Carry
?D>=C	A	8BByy	Requires GOYES or RTNYES; Affects Carry
?D>=C	fs	9bByy	Requires GOYES or RTNYES; Affects Carry
?D>C	A	8B3yy	Requires GOYES or RTNYES; Affects Carry
?D>C	fs	9b3yy	Requires GOYES or RTNYES; Affects Carry
?HS=0	d	83nyy	Requires GOYES or RTNYES; Affects Carry
?MP=0		838yy	Requires GOYES or RTNYES; Affects Carry
?P#	d	88nyy	Requires GOYES or RTNYES; Affects Carry
?P=	d	89nyy	Requires GOYES or RTNYES; Affects Carry
?SB=0		832yy	Requires GOYES or RTNYES; Affects Carry
?SR=0		834yy	Requires GOYES or RTNYES; Affects Carry
?ST=0	d	86nyy	Requires GOYES or RTNYES; Affects Carry
?ST=1	d	87nyy	Requires GOYES or RTNYES; Affects Carry
?XM=0		831yy	Requires GOYES or RTNYES; Affects Carry
A=-A	A	F8	Affects Carry
A=-A	fs	Bb8	Affects Carry
A=-A-1	A	FC	Clears Carry
A=-A-1	fs	BbC	Clears Carry
A=0	A	D0	
A=0	fs	Ab0	
A=A!B	fs	0Ef8	
A=A!C	fs	0EfE	
A=A&B	fs	0Ef0	
A=A&C	fs	0Ef6	
A=A+1	A	E4	Affects Carry

```
A=A+1   fs      Ba4     Affects Carry
A=A+A   A       C4      Affects Carry
A=A+A   fs      Aa4     Affects Carry
A=A+B   A       C0      Affects Carry
A=A+B   fs      Aa0     Affects Carry
A=A+C   A       CA      Affects Carry
A=A+C   fs      AaA     Affects Carry
A=A+CON rfs,d   818f0m  ** Affects Carry
A=A-1   A       CC      Affects Carry
A=A-1   fs      AaC     Affects Carry
A=A-B   A       E0      Affects Carry
A=A-B   fs      Ba0     Affects Carry
A=A-C   A       EA      Affects Carry
A=A-C   fs      BaA     Affects Carry
A=A-CON rfs,d   818f8m  ** Affects Carry
A=B     A       D4
A=B     fs      Ab4
A=B!A   fs      0Ef8
A=B&A   fs      0Ef0
A=B+A   A       C0      Affects Carry
A=B+A   fs      Aa0     Affects Carry
A=B-A   A       EC      Affects Carry
A=B-A   fs      BaC     Affects Carry
A=C     A       DA
A=C     fs      AbA
A=C!A   fs      0EfE
A=C&A   fs      0Ef6
A=C+A   A       CA      Affects Carry
A=C+A   fs      AaA     Affects Carry
A=DAT0  A       142
A=DAT0  B       14A
A=DAT0  fs      152a
A=DAT0  d       15Am
A=DAT1  A       143
A=DAT1  B       14B
A=DAT1  fs      153a
A=DAT1  d       15Bm
A=IN            802
A=PC            81B4    **
A=R0            110
A=R0.F  W       110     **
A=R0.F  fs      81Af10  **
A=R1            111
A=R1.F  W       111     **
A=R1.F  fs      81Af11  **
A=R2            112
A=R2.F  W       112     **
A=R2.F  fs      81Af12  **
A=R3            113
A=R3.F  W       113     **
A=R3.F  fs      81Af13  **
A=R4            114
A=R4.F  W       114     **
A=R4.F  fs      81Af14  **
ABEX    A       DC
ABEX    fs      AbC
ABIT=0  d       8084n   ** Clears bit d of A register
ABIT=1  d       8085n   ** Sets bit d of A register
```

```
ACEX    A       DE
ACEX    fs      AbE
AD0EX           132
AD0XS           13A
AD1EX           133
AD1XS           13B
APCEX           81B6    ** Continues elsewhere
AR0EX           120
AR0EX.F fs      81Af20  **
AR1EX           121
AR1EX.F fs      81Af21  **
AR2EX           122
AR2EX.F fs      81Af22  **
AR3EX           123
AR3EX.F fs      81Af23  **
AR4EX           124
AR4EX.F fs      81Af24  **
ASL     A       F0
ASL     fs      Bb0
ASLC            810
ASR     A       F4
ASR     fs      Bb4
ASRB            81C
ASRB.F  fs      819f0   **
ASRC            814
B=-B    A       F9      Affects Carry
B=-B    fs      Bb9     Affects Carry
B=-B-1  A       FD      Clears Carry
B=-B-1  fs      BbD     Clears Carry
B=0     A       D1
B=0     fs      Ab1
B=A     A       D8
B=A     fs      Ab8
B=A!B   fs      0EfC
B=A&B   fs      0Ef4
B=A+B   A       C8      Affects Carry
B=A+B   fs      Aa8     Affects Carry
B=B!A   fs      0EfC
B=B!C   fs      0Ef9
B=B&A   fs      0Ef4
B=B&C   fs      0Ef1
B=B+1   A       E5      Affects Carry
B=B+1   fs      Ba5     Affects Carry
B=B+A   A       C8      Affects Carry
B=B+A   fs      Aa8     Affects Carry
B=B+B   A       C5      Affects Carry
B=B+B   fs      Aa5     Affects Carry
B=B+C   A       C1      Affects Carry
B=B+C   fs      Aa1     Affects Carry
B=B+CON rfs,d   818f1m  ** Affects Carry
B=B-1   A       CD      Affects Carry
B=B-1   fs      AaD     Affects Carry
B=B-A   A       E8      Affects Carry
B=B-A   fs      Ba8     Affects Carry
B=B-C   A       E1      Affects Carry
B=B-C   fs      Ba1     Affects Carry
B=B-CON rfs,d   818f9m  ** Affects Carry
B=C     A       D5
```

B=C	fs	Ab5	
B=C!B	fs	0Ef9	
B=C&B	fs	0Ef1	
B=C+B	A	C1	Affects Carry
B=C+B	fs	Aa1	Affects Carry
B=C-B	A	ED	Affects Carry
B=C-B	fs	BaD	Affects Carry
BAEX	A	DC	
BAEX	fs	AbC	
BCEX	A	DD	
BCEX	fs	AbD	
BSL	A	F1	
BSL	fs	Bb1	
BSLC		811	
BSR	A	F5	
BSR	fs	Bb5	
BSRB		81D	
BSRB.F	W	81D	**
BSRB.F	fs	819f1	**
BSRC		815	
BUSCB		8083	**
BUSCC		80B	
BUSCD		808D	**
C+P+1		809	Affects Carry
C=-C	A	FA	Affects Carry
C=-C	fs	BbA	Affects Carry
C=-C-1	A	FE	Clears Carry
C=-C-1	fs	BbE	Clears Carry
C=0	A	D2	
C=0	fs	Ab2	
C=A	A	D6	
C=A	fs	Ab6	
C=A!C	fs	0EfA	
C=A&C	fs	0Ef2	
C=A+C	A	C2	Affects Carry
C=A+C	fs	Aa2	Affects Carry
C=A-C	A	EE	Affects Carry
C=A-C	fs	BaE	Affects Carry
C=B	A	D9	
C=B	fs	Ab9	
C=B!C	fs	0EfD	
C=B&C	fs	0Ef5	
C=B+C	A	C9	Affects Carry
C=B+C	fs	Aa9	Affects Carry
C=C!A	fs	0EfA	
C=C!B	fs	0EfD	
C=C!D	fs	0EfF	
C=C&A	fs	0Ef2	
C=C&B	fs	0Ef5	
C=C&D	fs	0Ef7	
C=C+1	A	E6	Affects Carry
C=C+1	fs	Ba6	Affects Carry
C=C+A	A	C2	Affects Carry
C=C+A	fs	Aa2	Affects Carry
C=C+B	A	C9	Affects Carry
C=C+B	fs	Aa9	Affects Carry
C=C+C	A	C6	Affects Carry
C=C+C	fs	Aa6	Affects Carry

```
C=C+CON rfs,d   818f2m  ** Affects Carry
C=C+D   A       CB      Affects Carry
C=C+D   fs      AaB     Affects Carry
C=C+P+1         809     Affects Carry
C=C-1   A       CE      Affects Carry
C=C-1   fs      AaE     Affects Carry
C=C-A   A       E2      Affects Carry
C=C-A   fs      Ba2     Affects Carry
C=C-B   A       E9      Affects Carry
C=C-B   fs      Ba9     Affects Carry
C=C-CON rfs,d   818fAm  ** Affects Carry
C=C-D   A       EB      Affects Carry
C=C-D   fs      BaB     Affects Carry
C=D     A       DB
C=D     fs      AbB
C=D!C   fs      0EfF
C=D&C   fs      0Ef7
C=D+C   A       CB      Affects Carry
C=D+C   fs      AaB     Affects Carry
C=DAT0  A       146
C=DAT0  B       14E
C=DAT0  fs      156a
C=DAT0  d       15Em
C=DAT1  A       147
C=DAT1  B       14F
C=DAT1  fs      157a
C=DAT1  d       15Fm
C=ID            806
C=IN            803
C=P     d       80Cn
C=PC            81B5    **
C=R0            118
C=R0.F  fs      81Af18  **
C=R1            119
C=R1.F  fs      81Af19  **
C=R2            11A
C=R2.F  fs      81Af1A  **
C=R3            11B
C=R3.F  fs      81Af1B  **
C=R4            11C
C=R4.F  fs      81Af1C  **
C=RSTK          07
C=ST            09
CAEX    A       DE
CAEX    fs      AbE
CBEX    A       DD
CBEX    fs      AbD
CBIT=0  d       8088n   ** Clears bit d of C register
CBIT=1  d       8089n   ** Sets bit d of C register
CD0EX           136
CD0XS           13E
CD1EX           137
CD1XS           13F
CDEX    A       DF
CDEX    fs      AbF
CLRHST          82F
CLRST           08
CON(1)  expr    n
```

```
CON(2)  expr    nn
CON(3)  expr    nnn
CON(4)  expr    nnnn
CON(5)  expr    nnnnn
CON(6)  expr    nnnnnn
CON(7)  expr    nnnnnnn
CON(8)  expr    nnnnnnnn
CONFIG          805
CPCEX           81B7    ** Continues elsewhere
CPEX    d       80Fn
CR0EX           128
CR0EX.F fs      81Af28  **
CR1EX           129
CR1EX.F fs      81Af29  **
CR2EX           12A
CR2EX.F fs      81Af2A  **
CR3EX           12B
CR3EX.F fs      81Af2B  **
CR4EX           12C
CR4EX.F fs      81Af2C  **
CSL     A       F2
CSL     fs      Bb2
CSLC            812
CSR     A       F6
CSR     fs      Bb6
CSRB            81E
CSRB.F  fs      819f2   **
CSRC            816
CSTEX           0B
D0=(2)  expr    19nn
D0=(4)  expr    1Annnn
D0=(5)  expr    1Bnnnnn
D0=A            130
D0=AS           138
D0=C            134
D0=CS           13C
D0=D0+  d       16m     Affects Carry
D0=D0-  d       18m     Affects Carry
D0=HEX  hh      19hh
D0=HEX  hhhh    1Ahhhh
D0=HEX  hhhhh   1Bhhhhh
D1=(2)  expr    1Dnn
D1=(4)  expr    1Ennnn
D1=(5)  expr    1Fnnnnn
D1=A            131
D1=AS           139
D1=C            135
D1=CS           13D
D1=D1+  d       17m     Affects Carry
D1=D1-  d       1Cm     Affects Carry
D1=HEX  hh      1Dhh
D1=HEX  hhhh    1Ehhhh
D1=HEX  hhhhh   1Fhhhhh
D=-D    A       FB      Affects Carry
D=-D    fs      BbB     Affects Carry
D=-D-1  A       FF      Clears Carry
D=-D-1  fs      BbF     Clears Carry
D=0     A       D3
```

```
D=0     fs      Ab3
D=C     A       D7
D=C     fs      Ab7
D=C!D   fs      0EfB
D=C&D   fs      0Ef3
D=C+D   A       C3      Affects Carry
D=C+D   fs      Aa3     Affects Carry
D=C-D   A       EF      Affects Carry
D=C-D   fs      BaF     Affects Carry
D=D!C   fs      0EfB
D=D&C   fs      0Ef3
D=D+1   A       E7      Affects Carry
D=D+1   fs      Ba7     Affects Carry
D=D+C   A       C3      Affects Carry
D=D+C   fs      Aa3     Affects Carry
D=D+CON rfs,d   818f3m  ** Affects Carry
D=D+D   A       C7      Affects Carry
D=D+D   fs      Aa7     Affects Carry
D=D-1   A       CF      Affects Carry
D=D-1   fs      AaF     Affects Carry
D=D-C   A       E3      Affects Carry
D=D-C   fs      Ba3     Affects Carry
D=D-CON rfs,d   818fBm  ** Affects Carry
DAT0=A  A       140
DAT0=A  B       148
DAT0=A  fs      150a
DAT0=A  d       158m
DAT0=C  A       144
DAT0=C  B       14C
DAT0=C  fs      154a
DAT0=C  d       15Cm
DAT1=A  A       141
DAT1=A  B       149
DAT1=A  fs      151a
DAT1=A  d       159m
DAT1=C  A       145
DAT1=C  B       14D
DAT1=C  fs      155a
DAT1=C  d       15Dm
DCEX    A       DF
DCEX    fs      AbF
DSL     A       F3
DSL     fs      Bb3
DSLC            813
DSR     A       F7
DSR     fs      Bb7
DSRB            81F
DSRB.F  fs      819f3   **
DSRC            817
GOC     label   4yy     Continues elsewhere if Carry set
GOLONG  label   8Cyyyy  Continues elsewhere
GONC    label   5yy     Continues elsewhere if Carry clear
GOSBVL  label   8Fyyyyy Affects Carry
GOSUB   label   7yyy    Affects Carry
GOSUBL  label   8Eyyyy  Affects Carry
GOTO    label   6yyy    Continues elsewhere
GOVLNG  label   8Dyyyyy Continues elsewhere
GOYES   label   tt      Must follow a test instruction
```

```
HS=0    d       82n     Clears (MP SR SB XM) according to d
                        Example: 823 Clears SB and XM
INTOFF          808F
INTON           8080
LA(1)   expr    80820n          ** Load A register
LA(2)   expr    80821nn         ** Load A register
LA(3)   expr    80822nnn        ** Load A register
LA(4)   expr    80823nnnn       ** Load A register
LA(5)   expr    80824nnnnn      ** Load A register
LA(6)   expr    80825nnnnnn     ** Load A register
LA(7)   expr    80826nnnnnnn    ** Load A register
LA(8)   expr    80827nnnnnnnn   ** Load A register
LC(1)   expr    30n             Load C register
LC(2)   expr    31nn            Load C register
LC(3)   expr    32nnn           Load C register
LC(4)   expr    33nnnn          Load C register
LC(5)   expr    34nnnnn         Load C register
LC(6)   expr    35nnnnnn        Load C register
LC(7)   expr    36nnnnnnn       Load C register
LC(8)   expr    37nnnnnnnn      Load C register
MP=0            828
NIBASC  ASCII x...x
NIBFS   fs      f
NIBHEX  hhhhhhh hhhhhhh Affects Carry
NOP3            820
NOP4            6300
NOP5            64000
OUT=C           801
OUT=CS          800
P=      d       2n
P=C     d       80Dn
P=P+1           0C      Affects Carry
P=P-1           0D      Affects Carry
PC=(A)          808C    ** Continues elsewhere
PC=(C)          808E    ** Continues elsewhere
PC=A            81B2    ** Continues elsewhere
PC=C            81B3    ** Continues elsewhere
R0=A            100
R0=A.F  fs      81Af00  **
R0=C            108
R0=C.F  fs      81Af08  **
R1=A            101
R1=A.F  fs      81Af01  **
R1=C            109
R1=C.F  fs      81Af09  **
R2=A            102
R2=A.F  fs      81Af02  **
R2=C            10A
R2=C.F  fs      81Af0A  **
R3=A            103
R3=A.F  fs      81Af03  **
R3=C            10B
R3=C.F  fs      81Af0B  **
R4=A            104
R4=A.F  fs      81Af04  **
R4=C            10C
R4=C.F  fs      81Af0C  **
REL(1)  label   y
```

```
REL(2)  label   yy
REL(3)  label   yyy
REL(4)  label   yyyy
REL(5)  label   yyyyy
REL(6)  label   yyyyyy
REL(7)  label   yyyyyyy
REL(8)  label   yyyyyyyy
RESET           80A
RSI             80810   **
RSTK=C          06
RTI             0F      Continues elsewhere
RTN             01      Continues elsewhere
RTNC            400     Continues elsewhere if Carry set
RTNCC           03      Continues elsewhere clearing Carry
RTNNC           500     Continues elsewhere if Carry clear
RTNSC           02      Continues elsewhere setting Carry
RTNSXM          00      Continues elsewhere setting XM bit
RTNYES          **      Must follow a test instruction
SB=0            822
SETDEC          05
SETHEX          04
SHUTDN          807
SLINK   label   yyyyy
SR=0            824
SREQ?           80E
ST=0    d       84n
ST=1    d       85n
ST=C            0A
STRING  ASCII x...x
UNCNFG          804
XM=0            821
```

10. Error Messages

This is a list of all errors which are generated by the assembler and explanations of what causes that error message to be printed.

Non-fatal error messages are lines of the form *** ERROR: message *** which are followed by the source line containing the error. These messages indicate errors in the code, but the assembly continues to subsequent lines. For a few error messages, the message follows the line containing the error. Those error messages are noted in the list.

Fatal error messages are lines of the form SASM.EXE: message (details). These messages indicate errors encountered during the assembly process which are not code-related and which terminate the assembly. The list and object files for the assembly, if any, may not be complete.

Command line messages are messages related to options specified on the command line. The messages indicate problems with options. If any command line messages have been printed when all options have been processed, the assembly stops. No files have been altered (except the d option, which takes effect immediately).

Some explanations include suggestions to help track down the cause of the error.

10.1 Non-Fatal Error Messages

ERROR: ABS/REL must be at beginning

The ABS and REL statements, if present, must precede all code-generating statements.

ERROR: Branch out of range

The distance to the target label is too far for the size of this field.

ERROR: Can't add two external expressions

Both sides of + are external expressions. At most one external reference is allowed with the + operator.

ERROR: Can't define a new macro within a macro

Executed a MACRO statement while expanding a macro. A new macro cannot be defined while expanding a macro.

Suggestion: A missing ENDM at the end of a macro definition might cause this if another macro follows the one currently being expanded.

ERROR: Can't have external reference on left side of

The left side of is an external reference. An external reference is allowed only on the right side of the operator.

ERROR: Can't have external reference on right side of -

The right side of - is an external reference. An external reference is allowed only on the left side of the - operator.

ERROR: Can't redefine an existing macro

A MACRO statement was executed to define a macro which has already been defined.

ERROR: Can't redefine an existing opcode

A MACRO statement was executed to define a macro, but the macro name is the same as an opcode.

ERROR: Can't use !,&,*,/,%, or ^ with external reference

At least one of the operands is an external expression. An external reference is not allowed with these operators.

ERROR: Conditional assembly stack overflow

IF statements are nested more than 20 levels. Simplify the logic of the IF tests, or use the SETFLAG and CLRFLAG statements to indicate combinations of IF tests.

ERROR: Divide by zero (result set to 0)

The expression on the right side of the / operator is zero.

ERROR: Duplicate ELSE statement

More than one ELSE statement was executed for a given test.

ERROR: Duplicate symbol

The symbol being defined on this line has already been defined in this file.

Suggestion: Check the cross reference to find the previous definition; line 0 means the symbol was defined on the command line with -D.

ERROR: ELSE without matching IF

An ELSE statement was executed without a corresponding IF statement.

Suggestion: Check for a typing error in the label field of the IF and ELSE statements.

ERROR: ENDIF without matching IF

An ENDIF statement was executed without a corresponding IF statement.

Suggestion: Check for a typing error in the label field of the IF and ENDIF statements.

ERROR: ENDM and EXITM not permitted outside of a macro

Executed an ENDM or EXITM while not in a macro.

Suggestion: Check for an error in the preceding MACRO statement, if present.

ERROR: End of file while defining macro

The end of a file containing a macro definition was reached while defining a macro.

Suggestion: Check for an error in an ENDM statement or a missing ENDM statement.

ERROR: Error reading CHARMAP file (filespec)

A file system error occurred while reading the CHARMAP file.

ERROR: Error reading RDSYMB file (filespec)

The RDSYMB file is not a Saturn object file, or a file system error occurred while reading the header of the RDSYMB file.

Three things must be true for the file to be considered a valid Saturn object file:

1. There is a 256-byte header in the file.

2. The first six bytes of the header are ``Saturn''.

3. The offset to the start of the symbol records indicated in the header is contained in this file.

ERROR: Error reading from RDSYMB file (filespec)

The RDSYMB file is not a Saturn object file, or a file system error occurred while reading the RDSYMB file.

ERROR: Expanded macro line too long (truncated)

The line resulting from macro expansion is more than 255 characters long. Only the first 255 characters are used.

ERROR: Exponent less than zero (result set to 0)

The expression on the right side of the ^ operator is less than zero.

ERROR: Expression can not be external

The expression for the EQU or = statement contains an external reference. Only absolute or relocatable symbols are allowed for these statements.

ERROR: Expression must be absolute

The expression in the ABS, REL, BSS, or LISTALL statement contains a relocatable or external reference. Only absolute symbols are allowed for these statements.

ERROR: Expression not allowed as target of branch

The modifier is not a label reference, and the -x option was not specified on the command line.

The label is not a valid label because it contains a right parenthesis ')'

Suggestion: If this is a REL(n) statement, an expression can be used without specifying the -x option by subtracting (*) from the expression and using a CON(n) statement.

ERROR: Expression not allowed for INC(x), LINK, and SLINK

The modifier is not a label reference. INC(n), LINK, and SLINK require a label reference.

The label is not a valid label because it contains a right parenthesis ')'

ERROR: Expression out of range

A digit expression is less than zero or greater than 15 after applying the statement's adjustment factor (usually 0 or -1).

ERROR: Expression stack overflow

The expression is too complicated. There can be no more than 20 pending data items.

Suggestion: Simplify the expression by splitting it into several smaller expressions. Assign each of the smaller expressions to a symbol and combine them for the final expression.

ERROR: Expression stack underflow

An expression was not present when expected.

Suggestion: This usually indicates a missing data item following an operator. This can be caused by a blank or tab within the expression, as they indicate the end of the field.

ERROR: Field too long

A field is longer than the maximum length allowed. There should be another message following this one which gives more detailed information about the error.

ERROR: Flag value out of range

The flag expression in the IF, CLRFLAG, or SETFLAG statement is less than zero or greater than 99.

ERROR: GOYES/RTNYES without test instruction

A GOYES or RTNYES statement was executed, but the preceding instruction was not a test instruction.

ERROR: IF expression must be absolute

The flag expression in the IF, CLRFLAG, or SETFLAG statement contains a relocatable or external reference. Only absolute symbols are allowed for these statements.

ERROR: INCLUDE or MACRO nested too deeply

The number of nested INCLUDE and MACRO expansions is greater than 20.

Suggestion: Check for a macro calling itself, or an include file including itself.

ERROR: Illegal field select

The field select is not a valid field select indicator (P, WP, XS, X, S, M, B, W, or A).

ERROR: Illegal mnemonic

The instruction was not found in the opcode table, or a macro was used before it was defined.

Suggestion: Check that the processor command-line option (-P) is correct, and that the label, if any, starts in either the first or second column.

ERROR: Input line too long (extra characters ignored)

An input line is more than 255 characters long. Only the first 255 characters are used.

ERROR: Instruction not allowed with this CPU

A statement was executed which is not permitted by the current value of the -P command-line option. This message indicates a probable assembler defect, and should be reported as such.

ERROR: Invalid ASC constant

An ASCII constant in an expression is improperly formed. The terminating character must match the beginning character (\, ', or ").

This message also indicates an ASCII constant containing more than 40 characters.

ERROR: Invalid ASC constant (too large)

More than 8 characters were specified for LAASC or LCASC, or more than 4 characters were specified in an ASCII constant in an expression.

ERROR: Invalid HEX constant

A character which is not a hexadecimal digit was specified for LAHEX, LCHEX, D0=HEX, D1=HEX, or NIBHEX, or than 40 hexadecimal digits were specified for NIBHEX.

ERROR: Invalid HEX constant (not 2, 4, or 5 digits)

The hexadecimal constant for D0=HEX and D1=HEX must consist of two, four, or five digits.

ERROR: Invalid HEX constant (not HEX digit)

A character which is not a hexadecimal digit follows a # in an expression.

ERROR: Invalid HEX constant (too large)

More than 16 hexadecimal digits were specified for LAHEX or LCHEX.

ERROR: Invalid HEX constant (too many digits)

More than eight hexadecimal digits follow a # in an expression.

ERROR: Invalid expression

The expression is not a valid expression or an expression is not present. There may be another message following this one which gives more detailed information about the error.

Suggestion: Any field which starts after column 30 is considered to be a comment unless overridden with the -c option.

ERROR: Invalid field select/digit

The modifier is neither an expression nor a valid field select indicator (P, WP, XS, X, S, M, B, W, or A).

Suggestion: This message occurs whenever the expression is not valid. There may be another message printed which gives more detailed information about the error.

ERROR: Invalid file specifier

The file specifier for RDSYMB, CHARMAP, or INCLUDE does not include a terminator character, or the file specifier is not present.

Suggestion: Any field which starts after column 30 is considered to be a comment unless overridden with the -c option.

ERROR: Invalid flag expression

The flag expression is not a valid expression or it is not present. There may be another message following this one which gives more detailed information about the error.

Suggestion: Any field which starts after column 30 is considered to be a comment unless overridden with the -c option.

ERROR: Invalid format character

The indirect macro parameter reference ``$(...)'' contains an invalid character. The characters within the parentheses must be a digit, colon, or one of the characters ``HhXxDdOoUu''.

ERROR: Invalid parameter indirection

The indirect macro parameter reference ``$('' was found, but the closing right parenthesis is missing.

ERROR: Invalid symbol

No symbol name was specified with the = or EQU statement, or an equals sign (=) in columns one or two is followed by a blank or tab.

ERROR: Invalid symbol name

No symbol was specified for IFDEF, IFNDEF, IFOPC, or IFNOPC.

Suggestion: Any field which starts after column 30 is considered to be a comment unless overridden with the -c option.

ERROR: Label not allowed on GOYES/RTNYES statement

A line containing a GOYES or RTNYES statement has a label. This is not allowed because GOYES and RTNYES are the second part of a test operation.

ERROR: List type not CODE,MACRO,INCLUDE,PSEUDO,ALL, or NOLIST

The modifier(s) for CLRLIST and SETLIST must start with letters C, M, I, P, or A for Code, Macro, Include, Pseudo, or All, respectively.

ERROR: Missing expression

An expression is required, but no expression was found.

Suggestion: Check that the expression starts before the comment column, which defaults to 30 (use the c option to change the column).

ERROR: Missing separator character

The string separator character for an IFSTR?? test was not found.

Suggestion: The first non-blank character after the IFSTR?? statement is used as the separator character to delineate the two strings. The character must appear a total of three times (start of first string, separator between strings, end of second string).

ERROR: Modulo zero (result set to 0)

The expression on the right side of the % operator is equal to zero.

ERROR: More data than operators in expression

The expression is incomplete. This message can occur as the result of an incorrect operator (another error message should be printed).

Suggestion: Check for blanks within the expression, or a symbol which is not contained within parentheses.

ERROR: Name on ENDM doesn't match MACRO name

The label on the ENDM statement is not the same as the label on the most recent MACRO statement.

Suggestion: A missing ENDM statement can cause this message if there is another macro defined after the missing ENDM.

ERROR: Number of nibbles must be non-negative

The number of nibbles expression in BSS is less than zero.

ERROR: Number of nibbles too big for file

The number of nibbles requested by BSS would overflow the address space of the Saturn CPU.

ERROR: Only one TITLE statement allowed per file

A TITLE statement was executed, but the title given does not match the current title.

Suggestion: Check included files for TITLE statements.

ERROR: Operator left on stack

There is not enough data for the number of operators given. This message indicates a probable assembler defect, and should be reported as such.

ERROR: Operator stack overflow

The expression is too complicated. There can be no more than 20 pending operators, including right parentheses.

Suggestion: Simplify the expression by splitting it into several smaller expressions. Assign each of the smaller expressions to a symbol and combine them for the final expression.

ERROR: Operator stack underflow

This message indicates a probable assembler defect, and should be reported as such.

ERROR: PC changed (use old value)

The value of the label on the ABS or REL statement changed.

Suggestion: Look in the cross reference for another reference to the label which changes its value.

ERROR: PC wrapped around to 00000

The number of nibbles in the file exceeds the address space of the Saturn CPU. The current PC is decremented by #100000.

ERROR: Parameter too big (truncated)

The size of an indirect macro parameter reference field would cause the line resulting from macro expansion to be more than 255 characters long. The reference size is reduced so it fits.

ERROR: Relocatable offset not allowed here

A digit expression is relocatable. The digit expression must either be absolute or external.

ERROR: Single nibble field not allowed here

A field select which could evaluate to a single-nibble field (P, XS, S, or WP) was requested for r=r+CON or r=r-CON.

Suggestion: This restriction is due to a limitation in the 1LR2 processor which causes these instructions to fail for single-nibble fields.

ERROR: Symbol changed (use old value)

The current value of a symbol is different than the value in pass 2.

Suggestion: Check for conditional assembly which uses the IFPASS1 or IFPASS2 statements incorrectly. This can also be caused by an error in code generated before this line.

ERROR: Symbol was not defined in pass 1

A symbol which is defined in pass 2 was not defined in pass 1.

Suggestion: Check for conditional assembly which uses the IFPASS1 or IFPASS2 statements incorrectly.

ERROR: Test instruction without GOYES/RTNYES

The previous line was a test instruction, but this line is neither GOYES or RTNYES.

ERROR: Too many relocatable/external references

More than one external or relocatable reference is present. At most one external or net relocatable reference is allowed.

ERROR: Unable to open CHARMAP file (filespec)

An error occurred opening the file indicated by the file specifier provided.

Suggestion: Check that the file specifier is correct and the file is readable. Also check the value of the SASM_CHARMAP environment variable.

ERROR: Unable to open INCLUDE file (filespec)

An error occurred opening the file indicated by the file specifier provided. This message follows the line containing the error.

Suggestion: Check that the file specifier is correct and the file is readable. Also check the value of the SASM_INCLUDE environment variable.

ERROR: Unable to open RDSYMB file (filespec)

An error occurred opening the file indicated by the file specifier provided.

Suggestion: Check that the file specifier is correct and the file is readable. Also check the value of the SASM_RDSYMB environment variable.

ERROR: Undefined symbol

The symbol used is not defined within this file and the symbol is not an external symbol reference.

ERROR: Unknown PSEUDO-OP

The internal code for this instruction is not a code which is known to the assembler.

Suggestion: If the opcode file (sasm.opc) has been modified more recently than the assembler, it may contain new codes which are not recognized in the older assembler.

ERROR: Unlabeled MACRO statement (*MACRO used)

The MACRO statement does not have a label. The default name *MACRO is used to avoid attempting to assemble the macro instructions without macro expansion.

ERROR: Unmatched '(' in expression

More right parentheses (() than left parentheses ()) occurred in an expression.

ERROR: Unmatched ')' in expression

More left parentheses ()) than right parentheses (()
occurred in an expression.

ERROR: Unrecognized operator

A character which is not a valid operator character was
found when an operator character was expected.

Suggestion: This error message may be caused by other errors
in the expression. The character is ignored as if it wasn't
there, which may cause other error messages.

ERROR: Value too big

The value of the expression for ABS or REL is larger than
the maximum address for the Saturn CPU.

10.2 Fatal Error Messages

The name preceding the colon in these messages is the
current assembler name.

SASM.EXE: filename is not an opcode table file

The specified filename does not have the correct format for
an opcode table file.

SASM.EXE: can't back up objfile

The assembler attempted to write a nibble at a location in
the file which has already been used. This message
indicates a probable assembler defect, and should be
reported as such.

SASM.EXE: corrupt opcode table entry (mnemonic not in table)

The opcode table file contains an indirect reference to
mnemonic, but mnemonic is not in the opcode table. This
message indicates a defect in the opcode table.

Suggestion: If mnemonic is not a level 0 instruction, the
mnemonic which makes the indirect reference should have the
same level as mnemonic.

SASM.EXE: error creating a temporary file

Creation of a temporary file failed. Temporary files are
used for input from standard input and for macros.

SASM.EXE: error creating file description for writing

Creation of the specified file failed.

Suggestion: Verify that the directory for the file is writable, and if the file already exists, check that it is writable.

SASM.EXE: error getting memory for reason

A request for memory was rejected by the operating system.

Suggestion: Remove RAM-resident utilities to free up memory.

SASM.EXE: error opening filename for access

The specified filename could not be opened for access.

Suggestion: For writing, if the file already exists, check that it is writable.

SASM.EXE: error reading from file description

A read from the file failed. The file may be corrupt, or a system error may have occurred.

SASM.EXE: error reading location from macro file

An attempt to read the current location failed. The file may be corrupt, or a system error may have occurred.

SASM.EXE: error setting location in file description

An attempt to set the current location failed. The file may be corrupt, or a system error may have occurred.

SASM.EXE: error writing to file description

An attempt to write to the specified file failed.

Suggestion: Check for a full disc, a write-protected file, or other operating system limitations.

SASM.EXE: macro macro_name was not defined in pass 1

A macro definition was encountered in pass 2 which was not found in pass 1.

Suggestion: Check for incorrect usage of the IFPASS1 or IFPASS2 statements.

SASM.EXE: object file cannot be [stdout]

The object file name is ``-'', but neither the h nor the H option was specified on the command line.

SASM.EXE: operand type (value) is not valid

An invalid operand type was encountered.

Suggestion: Check that the opcode table file is not corrupted, and that the version of the assembler matches that of the opcode table file.

SASM.EXE: too many nibbles in output file filename

The output file already contained 1048576 nibbles when an attempt was made to add another nibble.

Suggestion: Check for erroneous BSS statements.

SASM.EXE: unable to read current time

A request to the operating system for the current time failed.

10.3 Command Line Messages

Code field width must be greater than 0 (nn invalid)

The n option has a width less than or equal to zero.

Code field width must be less than nn (nn invalid)

The n option has a width greater than the maximum allowed. The exact value of the maximum depends on the line width field, which defaults to four, and is set to five with the s option.

Comment column must be non-negative (nn)

The c option has a negative value. The value must be greater than or equal to zero. A value of zero means there is no comment column.

Flag number out of range (nn)

The f option argument contains a flag number which is not between zero and 99, inclusive.

Invalid code generation level (string)

The P option argument is not between zero and three, inclusive.

Invalid flag digit (char)

The f option argument contains a character which is neither a digit nor a separator character. Valid separator characters are ',', ' ' (blank), (;), and (:).

List and code files both [stdout] (list disabled)

The A option is specified with the o - option. The A option is ignored (with the same results as specifying the N option).

Non-numeric code field width (string)

The n option argument is not a valid integer.

Non-numeric comment column (string)

The c option argument is not a valid integer.

Non-numeric expression for -D (string)

For the D option, the expression following the equals sign (=) is not a valid integer.

Suggestion: If the symbol to be defined contains an equals sign and the default value is to be used, append another equals sign to the symbol. This forces the default value to be used.

Non-numeric page length (string)

The p option argument is not a valid integer.

Non-numeric page width (string)

The w option argument is not a valid integer.

Options -A, -a, and -o not allowed with multiple files

Multiple file names are specified in conjunction with the A, a, or o options. Multiple file names are allowed only when the default list and object file names are used.

Suggestion: This error can also be caused by trying to specify options after a file name. All options must precede the first file name on the command line.

Page length must be at least 4 (n)

The p option length is less than four. At least four lines must be printed per page (three for the header, one for program data).

Page width must be at least 44 (nn)

The w option width is less than 44. The minimum width is big enough to allow a symbol reference and one line number to fit on the line.

11. Saturn Object File Format

A Saturn object file consists of the following three components:

+ A 256-byte header

+ Zero or more 256-byte blocks containing code

+ Zero or more 256-byte blocks containing symbols.

All two-byte and four-byte quantities in the header and symbol records are stored with the most significant byte of the quantity in the first byte of the field and the least significant byte of the quantity in the last byte of the field.

11.1 Saturn Object Header Record

Object Name	Size Bytes	Description [Contents]
ID	6	File identifier [``Saturn'']
Filesize	2	Number of 256-byte blocks in this file [16-bit unsigned value]
Codesize	4	Number of nibbles of code
Symbols	2	Number of symbols [16-bit unsigned value]
Refs	2	Number of symbol references [16-bit unsigned value]
Start	4	Code start address
Absolute	1	Absolute/Relocatable [1 means ABS, 0 means REL]
Reserved	1	Reserved byte [fill to 16-bit boundary]
Date	26	Date of creation [``Day Mon DD HH:MM:SS YYYY '']
Title	40	Title, if any; default = blanks
Reserved	20	Reserved bytes [Softkeys]
Version	26	Version of program creating file
Reserved	4	Reserved bytes [ROM ID]
Reserved	118	Reserved bytes [00]

11.2 Saturn Object Code Record

The code nibbles are stored two per byte in the code record. The first nibble of code is in the most significant nibble of the first code record byte, and the second nibble is in the least significant nibble of that byte.

11.3 Saturn Object Symbol Block

Symbols are stored in alphabetical order. The most significant bit of the SymInfo field is the RESOLVED bit. If the RESOLVED bit is set, the symbol is resolved, and has value Value. If the RESOLVED bit is not set, this is an external symbol whose value is unknown.

The second most significant bit of the SymInfo field is the RELOCATABLE bit. If the RELOCATABLE bit is set, this symbol should be adjusted (relocated) if the starting address of the module is changed. If the RELOCATABLE bit is not set, this is an absolute symbol which should not be relocated.

The remaining 14 bits of the SymInfo field indicate the number of external references to the symbol. Reference records for the symbol follow the symbol record.

Object Name	Size Bytes	Description [Contents]
SymbID	4	Symbol record identifier [``Symb'']
Record	18	Symbol record or reference record (14 records per block)
Symbol Record		
Name	12	Symbol name [blank-filled]
SymInfo	2	Symbol information and reference count
Value	4	Symbol value
Reference Record		
Class	1	Fill reference class
Subclass	1	Fill reference subclass
Address	4	Fill address
Adjust	4	Adjustment to fill value [32-bit signed value]
Fillsize	2	Size of fill reference in nibbles [16-bit unsigned value]
Reserved	6	Reserved bytes [pad to 18 bytes]

11.4 Fill Reference Types

Class	Subclass	Description
0	0	Direct reference
0	1	Direct reference **
1	0	Reference relative to start of reference * **
1	1	Reference relative to nibble past end of reference * **
1	2	Reference relative to start of reference [REL(n)] *
2	0	SLINK reference
2	1	LINK reference
3	0	INC(n) reference

* Loader checks that this reference is within range
** Loader may report shortenable references for this reference

CONTENTS