CHAPTER 7
CC-40 ASSEMBLY-LANGUAGE INSTRUCTION SET

INSTRUCTION LINE FORMAT

A line containing an assembly-language instruction consists of four fields. The fields are separated from each other by one or more spaces ("blanks") as follows.

LABEL   OPCODE   OPERAND   COMMENT

Each field in an instruction line has a particular purpose.

LABELS

Labels may be used in any instruction line. A label is a "word" which identifies or names a line so that it may be referenced by name in other lines in a program. More specifically, the ALDS assembler identifies the word in the label field with the memory address of any object-code command produced by the line. If no object code is produced by the line, the label is identified with the first object-code producing line following the label. Other instructions, such as those requiring a jump to the labeled instruction, can therefore refer to the instruction by name rather than by memory address.

Labels consist of either an initial uppercase or lowercase alphabetic character or the symbol "$" and any number of subsequent alphabetic or numeric characters. Only the first eight characters of a label are significant (the assembler considers a label "SPECIFIC1" to be the same as "SPECIFIC2").

The assembler interprets any character which is placed the first column of the line ~~to be~~ the first character of a label.

Therefore, if no label is used in an instruction, the first character position in the instruction line must be a space.

OP CODES AND OPERANDS

Op codes (operation codes)--sometimes called "assembly-language commands"--are required on each assembly-language instruction line which, through assembly, is to produce machine code. Op codes are standardized mnemonic abbreviations for the names of the command codes and commands executed by the 7000-family processor. They consist of from two to five alphabetic characters.

Operands, in general, provide value or addressing information particular to each operation. Some operations are completely described by an opcode. For example, the RETS (return from subroutine), TSTA (set status flags on value in A), and LDSP (load stack pointer) instructions have value and addressing information "implied" in them. Other instructions require as many as three operands. The BTJO (bit test and jump if ONE) operation, for example, requires three operands to specify a bit-test mask byte, a register to be tested, and a destination address for the jump.

Operands follow opcodes on instruction lines, and they must be preceded by one or more spaces. When multiple operands are required, they are separated by commas. Single- or multiple-operand must not contain spaces.

COMMENTS

Comments may be written in an optional field following the operand field (or the opcode field in instructions in which no operand is required). At least one space must precede a comment. Any printable (ASCII) characters may be used in a comment, and the length of a comment is limited only by the maximum instruction-line length.

Comments, additionally, may use the entire length of an instruction line if they are preceded by an asterisk (*) in the first column of the line. Such comment lines are ignored during assembly: they produce no machine code.

INSTRUCTION DATA SHEETS

Detailed information about particular instructions, operations, opcodes, and operands is provided in the instruction data sheets contained in this chapter. The instruction data sheets are arranged in alphabetical order by opcode. Each sheet contains the following blocks of information.

DESCRIPTION:	A description of the operation performed.
TABLE:	A table listing opcodes, operands, address modes, machine codes, and times (cycles) required for command execution.
PROCESS:	A symbolic operation description of the process which occurs during command execution.
FLAGS:	A description of the effect an operation has upon processor status flags.

Figure 1-1 shows a typical instruction data sheet and identifies the content of each part of the sheet.

ADC

BINARY ADD WITH CARRY

DESCRIPTION: Add the value from either the source register or the immediate operand to the value from the destination register.

If the Carry flag is set to ONE by execution of a previous instruction, the sum is incremented by one. The sum is stored in the destination register.

The Zero and Negative flags are set according to the value stored in the destination register. The Carry flag is set to ONE if the addition has incremented Bit 7 from ONE to ZERO.

WHAT THE INSTRUCTION DOES

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
ADC B,A	Implied	ADC B,A	>69	5
ADC Rn,A	Register file	ADC R2,A	>19 >02	8
ADC Rn,B	Register file	ADC R15,B	>39 >0F	8
ADC Rn,Rn	Register file	ADC R66,R17	>49 >42 >11	10
ADC %N,A	Immediate	ADC %>35,A	>29 >35	7
ADC %N,B	Immediate	ADC %255,B	>59 >FF	7
ADC %N,Rn	Immediate	ADC %127,R96	>79 >7F >60	9

TABLE OF OP CODES, OPERANDS, MACHINE LANGUAGE, AND TIMES

```
           B        A                  A
PROCESS:   ( Rn ) + ( B  ) + Carry --> ( B  )
           %N       Rn                 Rn
```

DESCRIPTION OF PROCESS IN 7000-FAMILY EXECUTION OF COMMAND

```
FLAGS:     Zero:     Set to ONE if the sum is zero
           Negative: Set to ONE if Bit 7 of the sum is ONE
           Carry:    Set to ONE if Bit 7 has been incremented
                        past zero
```

DESCRIPTION OF COMMAND EFFECT ON FLAGS

Figure 1-2. Data Sheet Contents and Uses

INSTRUCTION DESCRIPTION

The description of each instruction explains the execution of the instruction by the processor.

INSTRUCTION TABLE ENTRIES

The instruction table lists all of the specific information specific to each instruction.

The "FORM" column lists the opcode with all possible operand combinations.

The "ADR MODE" column lists the most prominent addressing mode used with each combination of opcode and operands. If both the A and B registers are used as operands, the addressing mode is "implied." Instructions with operands "implied" in them, such as CLRC, EINT, and TSTB) also use implied addressing. If a register other than A or B is specified in the operand, the addressing mode is "register file" or "peripheral file." If an immediate value is specified, the addressing mode is immediate. Operands for memory-addressing instructions (such as BR, CALL, and LDA) yield direct, indirect, and indexed addressing modes.

The "SRC EXAMPLE" column lists examples of each opcode and operand combination for every addressing mode.

The "OBJ CODE" column lists the machine-language code resulting from assembly of each instruction in the "SRC EXAMPLE" column. The first (and perhaps only) byte of each object-code expression is the single-byte machine-language command which corresponds to the instruction. Second, third, and fourth bytes provide address information or data required by the processor for command execution.

The "CYCLES" column records the number of system clock (machine) cycles taken to execute an instruction. The actual time consumed by each machine cycle is dependent on the area of memory from which a command or data byte is being fetched or a data byte is being written to. During RAM accesses, the time used during one cycle is 800 nanoseconds. During ROM accesses, cycle time can be as slow as 7.2 microseconds.

If a jump instruction has two execution possibilities (i.e., if it is a conditional jump instruction), two quantities are listed under "CYCLES" to indicate (1) the number of machine cycles taken during execution when the jump is not taken and (2) the number of cycles taken during execution when the jump is taken.

PROCESS DESCRIPTION ENTRIES

The "PROCESS" block contains symbolic description of the operation resulting from each instruction. Processing activity during instruction execution is diagrammed.

FLAG-DESCRIPTION ENTRIES

The "FLAGS" block describes in detail the setting or resetting of the carry, negative, and zero flags in the status byte during the execution of each instruction.

CONVENTIONS AND SYMBOLS USED IN DATA SHEETS

Number conventions and symbols used in the following instruction data sheets are the same as those used in previous sections of this manual. In summary, they are as follows.

NUMBER REPRESENTATION CONVENTIONS

Numbers express decimal values unless they are preceded by a ">" (greater than) symbol or a "?" (question mark) symbol. A ">"

symbol indicates hexadecimal value. A "?" symbol indicates binary value.

All numbers used for immediate-value, or literal operands must be preceded by a "%" (percent) symbol. All numbers used for direct- or immediate-address operands are preceded by an "@" (at) sign.

The length of numeric expressions described in the instruction data sheets is indicated by the number of numeric characters present in the "SRC EXAMPLE" column of the instruction table. If a value greater than decimal 255 or >FF is present in the column, the instruction being exemplified evaluates the corresponding operand to a two-byte value ranging from 0 to 65,535. If the value in the column is less than 256 or >100, the instruction evaluates the operand to a single-byte value (from 0 to 255).

GENERAL-PURPOSE REGISTER SYMBOLS

The 128 memory-resident, general-purpose registers in the CC-40 are identified either by a combination of alphabetic or graphic symbol and number or (in two cases) by single alphabetic letter.

Alphabetic or graphic symbols used in combination with numbers to designate registers are illustrated in the following examples.

R0	@1
R80	@>7F
R>3F	@127

Only the two registers at the low-address boundary of the register file are represented by single alphabetic letters. The register with the lowest address (designated also by R0 and @0) is represented by the letter "A"; similarly, the letter "B" represents the second register of the file (R1 or @1).

PERIPHERAL-FILE REGISTER SYMBOLS

The 255 registers in the CC-40 peripheral address space are designated by register numbers prefixed with "P" as shown in the following list.

```
P0
P255
P>FF
```

SPECIAL SYMBOLS

The "$" (dollar sign) symbol used alone or in a mathematical expression represents the current machine-language program counter. The instruction " JMP $-2" provides, for example, a two-byte infinitely executing program loop.

The symbol "ST" represents the status (flag) register. Bit 7 of the status register is the the CARRY flag, bit 6 the NEGATIVE flag, bit 5 the ZERO flag, and bit 4 the INTERRUPT flag. The lower four bits are insignificant.

ADC

BINARY ADD WITH CARRY

DESCRIPTION: Add the value from either the source register or the immediate operand to the value from the destination register.

If the Carry flag is set to ONE by execution of a previous instruction, the sum is incremented by one. The sum is stored in the destination register.

The Zero and Negative flags are set according to the value stored in the destination register. The Carry flag is set to ONE if the addition has incremented Bit 7 from ONE to ZERO.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
ADC B,A	Implied	ADC B,A	>69	5
ADC Rn,A	Register file	ADC R2,A	>19 >02	8
ADC Rn,B	Register file	ADC R15,B	>39 >0F	8
ADC Rn,Rn	Register file	ADC R66,R17	>49 >42 >11	10
ADC %N,A	Immediate	ADC %>35,A	>29 >35	7
ADC %N,B	Immediate	ADC %255,B	>59 >FF	7
ADC %N,Rn	Immediate	ADC %127,R96	>79 >7F >60	9

PROCESS:

```
  B        A                  A
( Rn ) + ( B  ) + Carry --> ( B  )
  %N       Rn                 Rn
```

FLAGS:

Zero:	Set to ONE if the sum is zero
Negative:	Set to ONE if Bit 7 of the sum is ONE
Carry:	Set to ONE if Bit 7 has been incremented past ONE

ADD

BINARY ADD

DESCRIPTION: Add the value from either the source register or the immediate operand to the value from the destination register.

The sum is stored in the destination register.

The Zero and Negative flags are set according to the value stored in the destination register. The Carry flag is set to ONE if the addition has incremented Bit 7 from ONE to ZERO.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
ADD B,A	Implied	ADD B,A	>68	5
ADD Rn,A	Register file	ADD R2,A	>18 >02	8
ADD Rn,B	Register file	ADD R15,B	>38 >0F	8
ADD Rn,Rn	Register file	ADD R66,R17	>48 >42 >11	10
ADD %N,A	Immediate	ADD %>33,A	>28 >33	7
ADD %N,B	Immediate	ADD %255,B	>58 >FF	7
ADD %N,Rn	Immediate	ADD %127,R96	>78 >7F >60	9

PROCESS:

```
  B        A         A
{ Rn } + ( B  ) --> ( B  }
  %N       Rn        Rn
```

FLAGS:

Zero:	Set to ONE if the sum is zero
Negative:	Set to ONE if Bit 7 of the sum is ONE
Carry:	Set to ONE if Bit 7 has been incremented past ONE

AND

AND WITH GENERAL-PURPOSE REGISTER

DESCRIPTION: Logically "and" each bit from either the source- operand or immediate-operand byte with the corresponding bit from the destination-operand byte.

When the bits of each pair are "anded," the resulting bit is ONE if and only if both of them have a value of ONE.

The resulting logical product is stored in the destination-operand byte. The Zero flag is set if the resulting byte is 0, and the Negative flag is set to ONE if Bit 7 of the result is ONE.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
AND B,A	Implied	AND B,A	>63	5
AND Rn,A	Register file	AND R2,A	>13 >02	8
AND Rn,B	Register file	AND R15,B	>33 >0F	8
AND Rn,Rn	Register file	AND R66,R17	>43 >42 >11	10
AND %N,A	Immediate	AND %>35,A	>23 >35	7
AND %N,B	Immediate	AND %255,B	>53 >FF	7
AND %N,Rn	Immediate	AND %127,R96	>73 >7F >60	9

```
PROCESS:   B         A        A
         ( Rn ) AND ( B  ) --> ( B  )
           %N        Rn       Rn
```

FLAGS: Zero: Set to ONE if the result is zero
Negative: Set to ONE if Bit 7 of the result is ONE
Carry: Reset to ZERO

ANDP

AND WITH PERIPHERAL-FILE REGISTER

DESCRIPTION: Logically "and" each bit from register A, register B, or an immediate value specified in the source operand with the corresponding bit in a peripheral-file register. Store the result in the peripheral-file register.

When the bits of each pair are anded, the resulting bit is ONE if and only if both of them have a value of ONE.

The resulting logical product is stored in the peripheral-file register. The Zero flag is set if the resulting byte is 0, and the Negative flag is set to ONE if Bit 7 of the result is ONE.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
ANDP A,Pn	Peripheral file	ANDP A,P2	>83 >02	10
ANDP B,Pn	Peripheral file	ANDP B,P35	>93 >23	9
ANDP %N,Pn	Peripheral file	ANDP %15,P64	>A3 >0F >40	11

PROCESS:

```
  A
( B  ) AND Pn --> Pn
  %N
```

FLAGS:

Zero:	Set to ONE if the result is zero
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Reset to ZERO

BR

BRANCH TO ADDRESS

DESCRIPTION: Load the program counter with a value for use as the address of the next instruction to be executed.

The two-byte operand value may address any location in CC-40 memory. The source of the address is determined by the address mode of the branch instruction.

In direct address mode, the program counter is loaded with a two-byte immediate value specified by the expression in the operand.

In indirect mode, the least-significant byte of the program counter is loaded from a register specified by the operand. The most-significant byte of the program counter is loaded from the next-lower numbered register.

In indexed mode, the program counter is loaded with the sum of the two-byte immediate value specified by the expression in the operand and the value (from 0 to 255) in the B register.

Where immediate hexadecimal values are shown in the table below, symbolic values assigned in equate instructions and labels can also be used.

FORM	ADDRESSING	SRC EXAMPLE	OBJ EXAMPLE	CYCLES
BR @EXPR	Direct	BR @>8000	>8C >80 >00	10
BR *Rn	Indirect	BR *R30	>9C >1E	9
BR @EXPR(B)	Indexed	BR @>4500(B)	>AC >45 >00	12

PROCESS: Direct Mode: Operand --> PC
Indirect Mode: *Rn --> PC
Indexed Mode: Operand + B --> PC

FLAGS: Unchanged

BTJO

BIT TEST AND JUMP IF ONE

DESCRIPTION: Test for a value of ONE in specified bit positions of a register. If any bit tested is ONE, perform a relative jump.

Bit positions for the test are identified by ONE bits in the source-operand byte. The register in which bits are to be tested is specified by the destination operand. The source-operand and destination-operand bytes are "anded" to set or reset status-register flags. No other register contents are affected by the test.

If the Zero flag is reset to ZERO, the program counter is loaded with the the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following the BTJO. The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the BTJO, and FWDTEN is a label which occurs ten bytes after the same instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
BTJO B,A,JOPRND	Implied	BTJO B,A,BCKTEN	>66 >F6	7,9
BTJO Rn,A,JOPRND	Register file	BTJO R2,A,FWDTEN	>16 >02 >0A	10,12
BTJO Rn,B,JOPRND	Register file	BTJO R15,B,BCKTEN	>36 >0F >F6	10,12
BTJO Rn,Rn,JOPRND	Register file	BTJO R66,R17,FWDTEN	>46 >42 >11 >0A	12,14
BTJO %N,A,JOPRND	Immediate	BTJO %>33,A,BCKTEN	>26 >33 >F6	9,11
BTJO %N,B,JOPRND	Immediate	BTJO %255,B,FWDTEN	>56 >FF >0A	9,11
BTJO %N,Rn,JOPRND	Immediate	BTJO %127,R96,BCKTEN	>76 >7F >60 >F6	11,13

```
                B          A
PROCESS:    If ( Rn ) AND ( B  ) <> 0 then PC + DISPLACEMENT --> PC
                %N         Rn
```

FLAGS:
Zero: Set to ONE if the result is zero (no jump)
Negative: Set to ONE if Bit 7 of the result is ONE
Carry: Reset to ZERO

BTJOP

BIT TEST AND JUMP IF ONE--PERIPHERAL

DESCRIPTION: Test for a value of ONE in specified bit positions of a peripheral-file byte. If any bit tested is ONE, perform a relative jump.

Bit positions for the test are identified by ONE bits in the source-operand byte. The register in which bits are to be tested is specified in the destination operand. The source-operand and destination-operand bytes are "anded" to set or reset status-register flags. No register contents are affected by the test.

If the Zero flag is reset to ZERO, the program counter is loaded with the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following the BTJOP. The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the BTJOP, and FWDTEN is a label which occurs ten bytes after the same instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
BTJOP A,Pn,JOPRND	Implied	BTJOP A,P2,BCKTEN	>86 >02 >F6	11,13
BTJOP B,Pn,JOPRND	Implied	BTJOP B,P51,FWDTEN	>96 >33 >0A	10,12
BTJOP %N,Pn,JOPRND	Immediate	BTJOP %15,P10,BCKTEN	>A6 >0F >0A >F6	12,14

```
            A
PROCESS:    If ( B  ) AND Pn <> 0 then PC + DISPLACEMENT --> PC
            %N
```

FLAGS:

Zero:	Set to ONE if the result is zero (no jump)
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Reset to ZERO

BTJZ

BIT TEST AND JUMP IF ZERO

DESCRIPTION: Test for a value of ZERO in specified bit positions of a byte. If any bit tested is ZERO, perform a program-counter relative jump.

Bit positions for the test are identified by ONE bits in the source-operand byte. The register in which bits are to be tested is specified by the destination operand. The source-operand byte and the complement of the destination-operand byte are "anded" to set or reset status-register flags. No other register contents are affected by the test.

If the "and" operation sets the Zero flag to ONE, the program counter is loaded with the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following the BTJZ. The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the BTJZ, and FWDTEN is a label which occurs ten bytes after the same instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
BTJZ B,A,JOPRND	Implied	BTJZ B,A,BCKTEN	>67 >F6	7,9
BTJZ Rn,A,JOPRND	Register file	BTJZ Rn,A,FWDTEN	>17 >02 >0A	10,12
BTJZ Rn,B,JOPRND	Register file	BTJZ R15,B,BCKTEN	>37 >0F >F6	10,12
BTJZ Rn,Rn,JOPRND	Register file	BTJZ R66,R17,FWDTEN	>47 >42 >11 >0A	12,14
BTJZ %N,A,JOPRND	Immediate	BTJZ %>35,A,BCKTEN	>27 >35 >F6	9,11
BTJZ %N,B,JOPRND	Immediate	BTJZ %255,B,FWDTEN	>57 >FF >0A	9,11
BTJZ %N,Rn,JOPRND	Immediate	BTJZ %127,R96,BCKTEN	>77 >7F >60 >F6	11,13

PROCESS:

```
     B             A
If ( Rn ) AND NOT ( B  ) <> 0 then PC + DISPLACEMENT --> PC
     %N            Rn
```

FLAGS:

Zero:	Set to ONE if the result is zero (jump)
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Reset to ZERO

BTJZP

BIT TEST AND JUMP IF ZERO--PERIPHERAL

DESCRIPTION: Test for a value of ZERO in specified bit positions of a peripheral-file byte. If any bit tested is ZERO, perform a relative jump.

Bit positions for the test are identified by ONE bits in the source-operand byte. The register in which bits are to be tested is specified in the destination operand. The source-operand byte and the complement of the destination-operand byte are "anded" to set or reset status-register flags. No register contents are affected by the test.

If the Zero flag is set to ONE, the program counter is loaded with the the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following the BTJZP. The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the BTJZP, and FWDTEN is a label which occurs ten bytes after the same instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
BTJZP A,Pn,JOPRND	Implied	BTJZP A,P2,BCKTEN	>87 >02 >F6	11,13
BTJZP B,Pn,JOPRND	Implied	BTJZP B,P51,FWDTEN	>97 >33 >0A	10,12
BTJZP %N,Pn,JOPRND	Immediate	BTJZP %15,P10,BCKTEN	>A7 >0F >0A >F6	12,14

PROCESS:

```
      A
If (  B  ) AND NOT Pn <> 0 then PC + DISPLACEMENT --> PC
      %N
```

FLAGS:

Zero:	Set to ONE if the result is zero (jump)
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Reset to ZERO

CALL

SUBROUTINE CALL

DESCRIPTION: Increment the stack pointer and save the next-instruction address, most-significant byte first, on the stack as the subroutine-return address. Then load the subroutine-address value from the expression in the operand into the the program counter, thus transferring program control to the subroutine.

The source of the subroutine address value is determined by the address mode of the CALL instruction.

In direct mode, the program counter is loaded with a two-byte immediate value specified by the expression in the operand.

In indirect mode, the program counter is loaded with the contents of the register specified by the operand (least-significant byte) and the register one address below it (most-significant byte).

In indexed mode, the program counter is loaded with the sum of the two-byte immediate value specified by the expression in the operand and the value in the B register.

Where immediate hexadecimal values are shown in the table below, symbolic values assigned in equate instructions and labels can also be used.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
CALL @EXPR	Direct	CALL @>8000	>8E >80 >00	14
CALL *Rn	Indirect	CALL *R30	>9E >1E	13
CALL @EXPR(B)	Indexed	CALL @>E015(B)	>AE >E0 >15	16

PROCESS:

```
Direct Mode:   PC MSB,LSB  --> STACK; Operand --> PC
Indirect Mode: PC MSB,LSB --> STACK; *Rn --> PC
Indexed Mode:  PC MSB,LSB --> STACK; Operand + B --> PC
```

FLAGS: Unchanged

CLR

CLEAR REGISTER

DESCRIPTION: Reset all bits in the register specified by the operand to ZERO.

The Carry and Negative flags are reset to ZERO, and the Zero flag is set to ONE.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
CLR A	Implied	CLR A	>B5	5
CLR B	Implied	CLR B	>C5	5
CLR Rn	Register file	CLR R6	>D5 >06	7

PROCESS:

```
        A
0 --> ( B  ) ; Set or reset flags.
        Rn
```

FLAGS:

Zero:	Set to ONE
Negative:	Reset to ZERO
Carry:	Reset to ZERO

CLRC

CLEAR CARRY FLAG

DESCRIPTION: Reset the Carry flag to ZERO and set or reset the Zero and Negative flags according to the value in the A register.

This instruction produces the same object code as the TSTA instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
CLRC	Implied	CLRC	>B0	6

PROCESS: Set or reset flags.

FLAGS:

Zero:	Set to ONE if A contains zero
Negative:	Set to ONE if BIT 7 of A is ONE
Carry:	Reset to ZERO

CMP

COMPARE REGISTER

DESCRIPTION: Subtract the value in the source register or the value of the immediate operand from contents of the destination register. Discard the difference.

The Zero and Negative flags are set or reset according to the difference. The Carry flag is reset to ZERO if a borrow has occurred (i.e., if bit 7 has been decremented past ZERO), when the destination register or immediate operand is logically less than or equal to the destination register.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
CMP B,A	Register file	CMP B,A	>6D	5
CMP Rn,A	Register file	CMP R2,A	>1D >02	8
CMP Rn,B	Register file	CMP R15,B	>3D >0F	8
CMP Rn,Rn	Register file	CMP R66,R17	>4D >42 >11	10
CMP %N,A	Immediate	CMP %>35,A	>2D >35	7
CMP %N,B	Immediate	CMP %255,B	>5D >FF	7
CMP %N,Rn	Immediate	CMP %127,R96	>7D >7F >60	9

PROCESS:

```
  A        B
( B  ) - ( Rn ) sets flags
  Rn       %N
```

FLAGS:

Zero:	Set to ONE if difference is zero (equal)
Negative:	Set to ONE if Bit 7 of the difference is ONE
Carry:	Reset to ZERO if Bit 7 has been decremented past ZERO

CMPA

COMPARE WITH A EXTENDED

DESCRIPTION: Subtract the contents of a memory location from the A register. Discard the difference.

The Zero and Negative flags are set or reset according to the difference. The Carry flag is reset to ZERO if a borrow has occurred (i.e., if bit 7 has been decremented past ZERO). The memory location used in the comparison may be specified by direct, indirect, or indexed addressing mode.

In direct mode, the location is specified by the expression in the operand.

In indirect mode, the location is specified by the contents of the named register (least significant byte) and the register one address below it (most significant byte).

In indexed mode, the location is specified by the sum of the two-byte immediate value specified by the expression in the operand and the value (from 0 through 255) in the B register.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
CMPA @EXPR	Direct	CMPA @>8000	>8D >80 >00	12
CMPA *Rn	Indirect	CMPA *R30	>9D >1E	11
CMPA @EXPR(B)	Indexed	CMPA @415(B)	>AD >01 >9F	14

PROCESS: The operand value subtracted from A sets flags

FLAGS:

Zero:	Set to ONE if the difference is zero (equal)
Negative:	Set to ONE if Bit 7 of the difference is ONE
Carry:	Reset to ZERO if Bit 7 has been decremented past ZERO

DAC

DECIMAL ADD WITH CARRY

DESCRIPTION: Add two binary-coded decimal (BCD) nibbles (four-bit numbers) in the source register or in the value of the immediate operand to two BCD nibbles in the destination register. Results of this operation are undefined if the nibbles do not contain valid BCD values (0 through 9).

If the Carry flag has been set to ONE by execution of a previous instruction, the sum is incremented by one. A decimal adjust operation is performed on each nibble of the sum to correct it for BCD representation. The sum is then stored in the destination register. The Zero and Negative flags are set or reset according to the sum. The Carry flag is set to ONE if a carry has occurred (that is, when the value in the destination register exceeds >99 BCD).

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
DAC B,A	Implied	DAC B,A	>6E	7
DAC Rn,A	Register file	DAC R2,A	>1E >02	10
DAC Rn,B	Register file	DAC R15,B	>3E >0F	10
DAC Rn,Rn	Register file	DAC R66,R17	>4E >42 >11	12
DAC %N,A	Immediate	DAC %>33,A	>2E >33	9
DAC %N,B	Immediate	DAC %>62,B	>5E >62	9
DAC %N,Rn	Immediate	DAC %>99,R96	>7E >99 >60	11

```
           B        A                     A
PROCESS:  ( Rn ) + ( B  ) + Carry --> ( B  )   (MODULO 10)
           %N       Rn                    Rn
```

FLAGS:
Zero: Set to ONE if sum is zero
Negative: Set to ONE if Bit 7 of the sum is ONE
Carry: Set to ONE if Bit 7 of the sum has been incremented past ONE

DEC

DECREMENT REGISTER

DESCRIPTION: Decrement by one the contents of the register specified in the operand.

The Zero and Negative flags are set or reset according to the result. The Carry flag is reset to ZERO if the decrement has been from >00 to >FF. Otherwise the Carry flag is set to ONE.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
DEC A	Implied	DEC A	>B2	5
DEC B	Implied	DEC B	>C2	5
DEC Rn	Register file	DEC R6	>D2 06	7

PROCESS:

```
  A           A
( B ) - 1 --> ( B )
  Rn          Rn
```

FLAGS:

Zero:	Set to ONE if result is zero
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Reset to ZERO if the decrement has been from >00 to >FF

DECD

DECREMENT DOUBLE REGISTER

DESCRIPTION: Decrement the two-byte value stored in a register pair.

The value in the register specified in the operand and the value in the register in the next lower memory location form a 16-bit. two-byte value which is decremented by one. The register at the lower memory location holds the most-significant byte of the pair.

The Zero and Negative flags are set according to the decremented contents of the most significant byte. The Carry flag is reset to ZERO if the decrement of the most significant byte has been from >00 to >FF. Otherwise the Carry flag is set to ONE.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
DECD A	Implied	DECD A	>BB	9
DECD B	Implied	DECD B	>CB	9
DECD Rn	Register file	DECD R6	>DB >06	11

PROCESS: RPn - 1 --> RPn

FLAGS:

Zero:	Set to ONE if the most significant byte of the result is zero
Negative:	Set to ONE if Bit 7 of the most significant byte of the result is ONE
Carry:	Reset to ZERO if the most significant byte has been decremented from >00 to >FF

DINT

DISABLE INTERRUPTS

DESCRIPTION: Reset all status flags, including the Interrupt flag, to ZERO.

All interrupts are disabled until an Enable Interrupt instruction is executed.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
DINT	Implied	DINT	>06	5

PROCESS: Reset all flags to ZERO

FLAGS:

Zero:	Set to ZERO
Negative:	Set to ZERO
Carry:	Set to ZERO
Interrupt:	Set to ZERO

DJNZ

DECREMENT REGISTER AND JUMP IF NON-ZERO

DESCRIPTION: Decrement the register named in the operand and perform a jump relative to the program-counter if the resulting byte does not have a value of 0.

When the register is not decremented to 0, the program counter is loaded with the the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following the DJNZ. The assembler computes a one-byte offset to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the DJNZ, and FWDTEN is a label which occurs ten bytes after the same instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
DJNZ A,JOPRND	Implied, PC relative	DJNZ,A,BCKTEN	>BA >F6	2,7
DJNZ B,JOPRND	Implied, PC relative	DJNZ B,BCKTEN	>CA >F6	2,7
DJNZ Rn,JOPRND	Register file, PC relative	DJNZ R15,FWDTEN	>DA >0F >0A	3,9

```
            A
PROCESS:    If ( B  ) - 1 <> 0 then PC + DISPLACEMENT --> PC   (<>=not equals)
            Rn
```

FLAGS:

Zero:	Set to ONE if the result is not zero
Negative:	Set to ONE if Bit 7 of the decrement result is ONE
Carry:	Reset to ZERO

DSB

DECIMAL SUBTRACT WITH BORROW

DESCRIPTION: Subtract two binary-coded decimal (BCD) nibbles (four-bit numbers) in the source register or in the value of the immediate operand from two BCD nibbles in the destination register. If the Carry flag has been set to ZERO by execution of a previous instruction, decrement the difference by one. Results of this operation are undefined if the nibbles do not contain valid BCD values (0 through 9).

A decimal adjust operation is performed on each nibble of the difference to correct it for BCD representation. The corrected difference is then stored in the destination register. The Zero and Negative flags are set or reset according to the difference. The Carry flag is set to ZERO if a borrow has occurred (i.e., if Bit 7 has been decremented past ZERO).

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
DSB B,A	Implied	DSB B,A	>6F	7
DSB Rn,A	Register file, Implied	DSB R2,A	>1F >02	10
DSB Rn,B	Register file, Implied	DSB R15,B	>3F >0F	10
DSB Rn,Rn	Register file	DSB R66,R17	>4F >42 >11	12
DSB %N,A	Immediate, Implied	DSB %>33,A	>2F >33	9
DSB %N,B	Immediate, Implied	DSB %>11,B	>5F >11	9
DSB %N,Rn	Immediate, Register file	DSB %>79,R95	>7F >79 >5F	11

PROCESS:

```
  A       B                      A
( B  ) - ( Rn ) - NOT CARRY --> ( B  )  (MODULO 10)
  Rn      %N                     Rn
```

FLAGS:

Zero:	Set to ONE if the difference is zero
Negative:	Set to ONE if Bit 7 of the difference is ONE
Carry:	Reset to ZERO if Bit 7 of the difference has been decremented past ZERO

EINT

ENABLE INTERRUPTS

DESCRIPTION: Set all status flags, including the Interrupt flag, to ONE.

All interrupts set in the IOCR are enabled until a Disable Interrupt instruction is executed or an interrupt is received.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
EINT	Implied	EINT	>05	5

PROCESS: Set all flags to ONE

FLAGS:

Zero:	Set to ONE
Negative:	Set to ONE
Carry:	Set to ONE
Interrupt:	Set to ONE

IDLE

IDLE UNTIL INTERRUPT

DESCRIPTION: Halt processing until an interrupt or a reset occurs.

For an interrupt to cause processing to resume, the Interrupt flag and any interrupt-enable bits in the I/O control registe- must be set prior to execution of the IDLE instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
IDLE	Implied	IDLE	>01	6+

PROCESS: Halt processing

FLAGS SET: Unchanged

INC

INCREMENT REGISTER

DESCRIPTION: Increment the byte in the register designated by the operand.

The Zero and Negative flags are set according to the incremented contents. The Carry flag is set to ONE if the increment has been from >FF to 0. Otherwise the Carry flag is reset to ZERO.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
INC A	Implied	INC A	>B3	5
INC B	Implied	INC B	>C3	5
INC Rn	Register file	INC R6	>D3 >06	7

PROCESS: $\left(\begin{array}{c}A\\B\\Rn\end{array}\right) + 1 \rightarrow \left(\begin{array}{c}A\\B\\Rn\end{array}\right)$

FLAGS:

Zero:	Set to ONE if the result is zero
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Set to ONE if the increment has been from >FF to 0

INV

INVERT REGISTER

DESCRIPTION: Invert (complement) all bits in the register designated by the operand .

The Negative flag and the Zero flag are set according to the resulting value. The Carry flag is reset to ZERO.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
INV A	Implied	INV A	>B4	3
INV B	Implied	INV B	>C4	5
INV Rn	Register file	INV R6	>D4 >06	7

PROCESS: $\text{NOT } \{ \overset{A}{\underset{Rn}{B}} \} \dashrightarrow \{ \overset{A}{\underset{Rn}{B}} \}$

FLAGS:
Zero: Set to ONE if the result is zero
Negative: Set to ONE if Bit 7 of the result is ONE
Carry: Reset to ZERO

JC

JUMP IF CARRY FLAG SET

DESCRIPTION: If the Carry flag is set to ONE, load the program counter with the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following this instruction.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the JC instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JC JOPRND	PC relative	JC BCKTEN	>E3 >F6	5,7

PROCESS: If Carry flag = ONE then PC + DISPLACEMENT --> PC

FLAGS: Unchanged

JEQ

JUMP IF EQUAL (ZERO FLAG SET)

DESCRIPTION: If the Zero flag is set to ONE, load the program counter with the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following this instruction.

Although this instruction produces the same machine-language command that is produced by JZ, the JEQ mnemonic is particularly useful after a comparison to test whether two values are equal.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCXTEN is a label which occurs ten bytes prior to the instruction following the JEQ instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JEQ JOPRND	PC relative	JEQ BCXTEN	>E2 >F6	5,7

PROCESS: If Zero flag = ONE then PC + DISPLACEMENT --> PC

FLAGS: Unchanged

JHS

JUMP IF HIGHER OR THE SAME (CARRY FLAG SET)

DESCRIPTION: If the Carry flag is set to ONE, load the program counter with the the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following this instruction.

Although this instruction produces the same machine-language command that is produced by JC, the JHS mnemonic is particularly useful after a comparison to test whether the value reprepresented by the second operand is higher than or the same as the value represented by the first operand.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the JHS instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JHS JOPRND	PC relative	JHS BCKTEN	>E3 >F6	5,7

PROCESS: If Carry flag = ONE then PC + DISPLACEMENT --> PC

FLAGS: Unchanged

JL

JUMP IF LOWER (CARRY FLAG RESET)

DESCRIPTION: If the Carry flag is reset to ZERO, load the program counter with the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following this instruction.

Although this instruction produces the same machine-language command that is produced by JNC, the JL mnemonic is particularly useful after a comparison to test whether the value repreprresented by the second operand is lower than the value represented by the first operand.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the JL instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JL JOPRND	PC relative	JL BCKTEN	>E7 >F6	5,7

PROCESS: If Carry flag = ZERO then PC + DISPLACEMENT --> PC

FLAGS: Unchanged

JMP

JUMP UNCONDITIONALLY

DESCRIPTION: Load the program counter with the the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following this instruction.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the JMP instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JMP JOPRND	PC relative	JMP BCKTEN	>E0 >F6	7

PROCESS: PC + DISPLACEMENT --> PC

FLAGS: Unchanged

JN

JUMP IF NEGATIVE (NEGATIVE FLAG SET)

DESCRIPTION: If the Negative flag is set to ONE, load the the program counter with the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following this instruction.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the JN instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JN JOPRND	PC relative	JN BCKTEN	>E1 >F6	5,7

PROCESS: If Negative flag = ONE then PC + DISPLACEMENT --> PC

FLAGS: Unchanged

JNC

JUMP IF NO CARRY (CARRY FLAG RESET)

DESCRIPTION: If the Carry flag is reset to ZERO, load the the program counter with the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following the this instruction.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the JNC instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JNC JOPRND	PC relative	JNC BCKTEN	>E7 >F6	5,7

PROCESS: If Carry flag = ZERO then PC + DISPLACEMENT --> PC

FLAGS: Unchanged

JNE

JUMP IF NOT EQUAL (ZERO FLAG RESET)

DESCRIPTION: If the Zero flag is reset to ZERO, load the the program counte with the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following the this instruction.

Although this instruction produces the same machine-language command that is produced by JNZ, the JNE mnemonic is particularly useful after a comparison to test whether the value reprepresented by the operands are unequal.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCXTEN is a label which occurs ten bytes prior to the instruction following the JNE instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JNE JOPRND	PC relative	JNE BCXTEN	>E6 >F6	5,7

PROCESS: If Zero flag = ZERO then PC + DISPLACEMENT --> PC

FLAGS: Unchanged

JNZ

JUMP IF NOT ZERO (ZERO FLAG RESET)

DESCRIPTION: If the Zero flag is reset to ZERO, load the the program counter with the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following the this instruction.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the JNZ instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JNZ JOPRND	PC relative	JNZ BCKTEN	>E6 >F6	5,7

PROCESS: If Zero flag = ZERO then PC + DISPLACEMENT --> PC

FLAGS: Unchanged

JP

JUMP IF POSITIVE (NEGATIVE AND ZERO FLAGS RESET)

DESCRIPTION: If both the Negative flag and the Zero flag are reset to ZERO, load the program counter with the the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following the this instruction.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the JP instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JP JOPRND	PC relative	JP BCKTEN	>E4 >F6	5,7

PROCESS: If both Negative flag and Zero flag = ZERO
then PC + DISPLACEMENT --> PC

FLAGS: Unchanged

JPZ

JUMP IF POSITIVE OR ZERO (NEGATIVE FLAG RESET)

DESCRIPTION: If the Negative flag is reset to ZERO. load the the program counter with the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following the this instruction.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCKTEN is a label which occurs ten bytes prior to the instruction following the JPZ instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JPZ JOPRND	PC relative	JPZ BCKTEN	>E5 >F6	5,7

PROCESS: If Negative flag = ZERO then PC + DISPLACEMENT --> PC

FLAGS: Unchanged

JZ

JUMP IF ZERO (ZERO FLAG SET)

DESCRIPTION: If the Zero flag is set to ONE, load the the program counter with the value of the jump operand (abbreviated "JOPRND" in the table below) relative to the program-counter value at the instruction following the this instruction.

The assembler computes a one-byte displacement to the jump destination. If the value computed for this byte lies outside the range >00 through >7F for positive values of 0 through 127 or >80 through >FF for negative values of -128 through -1, the assembler outputs a "DISPLACEMENT TOO BIG" error message.

In the following table BCXTEN is a label which occurs ten bytes prior to the instruction following the JZ instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
JZ JOPRND	PC relative	JZ BCXTEN	>E2 >F6	7,9

PROCESS: If Zero flag = ONE then PC + DISPLACEMENT --> PC

FLAGS: Unchanged

LDA

LOAD THE A REGISTER FROM MEMORY

DESCRIPTION: Load the A register with the contents of a memory location. Set the flags on the byte loaded.

The memory location is specified by direct, indirect, or indexed addressing. With direct addressing, the location is specified by an expression in the operand. With indirect addressing, it is specified by the contents of the named register (least significant byte) and the register one address below it (most significant byte). In indexed mode, it is specified by the sum of the value represented in the operand and the contents of the B register.

Where immediate hexadecimal values are shown in the table below, symbolic values assigned in equate instructions and labels can also be used.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
LDA @EXPR	Direct	LDA @>B000H	>8A >B0 >00	11
LDA *Rn	Indirect	LDA *R30	>9A >1E	10
LDA @EXPR(B)	Indexed	LDA @B15(B)	>AA >03 >2F	13

PROCESS: MEMORY --> A REGISTER

FLAGS:

Zero:	Set to ONE if the byte loaded is zero
Negative:	Set to ONE if Bit 7 of the byte loaded is ONE
Carry:	Reset to ZERO

LDSP

LOAD STACK POINTER

DESCRIPTION: Load the value in the B register into the stack pointer.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
LDSP	Implied	LDSP	>0D	5

PROCESS: B --> SP

FLAGS: Unchanged

MOV

MOVE TO REGISTER

DESCRIPTION: Transfer a one-byte value to a register. Set the flags on the byte moved.

The source value may be immediate or the contents of another register.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
MOV A,B	Implied	MOV A,B	>C0	6
MOV B,A	Implied	MOV B,A	>62	5
MOV A,Rn	Register file	MOV A,R20	>D0 >14	8
MOV B,Rn	Register file	MOV B,R100	>D1 >64	7
MOV Rn,A	Register file	MOV R2,A	>12 >02	8
MOV Rn,B	Register file	MOV R15,B	>32 >0F	8
MOV Rn,Rn	Register file	MOV R66,R17	>42 >42 >11	10
MOV %N,A	Immediate	MOV %>35,A	>22 >35	7
MOV %N,B	Immediate	MOV %255,B	>52 >FF	7
MOV %N,Rn	Immediate	MOV %127,R96	>72 >7F >60	9

PROCESS:

```
          A
          B          A
        (    ) --> ( B )
          Rn         Rn
          %N
```

FLAGS:

Zero:	Set to ONE if the byte moved is zero
Negative:	Set to ONE if Bit 7 of the byte moved is ONE
Carry:	Reset to ZERO

MOVD

MOVE DOUBLE BYTE

DESCRIPTION: Transfer a two-byte value to a register pair. Set the flags on the most significant byte moved.

The source of the value may be immediate, the contents of another register pair, or the sum of an immediate value and the value in the B register.

The destination register specified by the operand is the higher numbered register of the pair (which contains the least-significant byte of the two-byte value).

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
MOVD %NN,Rn	Immediate	MOVD %>8000,R5	>88 >80 >00 >05	15
MOVD Rn,Rn	Register file	MOVD R7,R5	>98 >07 >05	14
MOVD %NN(B),Rn	Indexed	MOVD %>7FFF(B),R1	>A8 >7F >FF >01	17

```
           Immediate
PROCESS:  ( Reg Pair  ) --> Register pair
           Immed + B
```

FLAGS:

Zero:	Set to ONE if the most significant byte moved is zero	
Negative:	Set to ONE if Bit 7 of the most significant byte moved is ONE	
Carry:	Reset to ZERO	

MOVP

MOVE TO OR FROM PERIPHERAL FILE REGISTER

DESCRIPTION: Transfer a byte from the register designated by the source operand to the register designated by the destination operand. Set the flags on the byte moved.

If the source operand designates a peripheral file register, the instruction inputs a byte from the peripheral file register. If the source operand designates register A or B or an immediate value, the instruction outputs a byte to the peripheral file register.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
MOVP A,Pn	Peripheral file	MOVP A,P99	>62 >63	10
MOVP B,Pn	Peripheral file	MOVP B,P100	>92 >64	9
MOVP %N,Pn	Peripheral file	MOVP %32,P6	>A2 >20 >06	11
MOVP Pn,A	Peripheral file	MOVP P1,A	>80 >01	9
MOVP Pn,B	Peripheral file	MOVP P3,B	>91 >03	8

PROCESS:

```
   A                   A
 { B  } --> Pn  ;  Pn --> {   }
   %N                  B
```

FLAGS:
Zero: Set to ONE if byte moved is zero
Negative: Set to ONE if Bit 7 of byte moved is ONE
Carry: Reset to ZERO

MPY

MULTIPLY

DESCRIPTION: Multiply the value in the source register or the value of the immediate operand by the value in the destination register. The most significant byte of the product is stored in the A register, and the least significant byte is stored in the B register.

The Zero and Negative flags are set according to the value of the most significant byte of the product. The Carry flag is reset to ZERO.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
MPY B,A	Implied	MPY B,A	>6C	43
MPY Rn,A	Register file	MPY R2,A	>1C >02	46.
MPY Rn,B	Register file	MPY R15,B	>3C >0F	46
MPY Rn,Rn	Register file	MPY R66,R17	>4C >42 >11	48
MPY %N,A	Immediate	MPY %>33,A	>2C >33	45
MPY %N,B	Immediate	MPY %255,B	>5C >FF	45
MPY %N,Rn	Immediate	MPY %127,R99	>7C >7F >63	47

PROCESS:

```
  B      A
( Rn ) * ( B  ) --> A,B
  %N     Rn
```

FLAGS:

Zero:	Set to ONE if the most significant byte of product (in A) is zero
Negative:	Set to ONE if Bit 7 of the most significant byte of product (in A) is ONE
Carry:	Reset to ZERO

NOP

NO OPERATION

DESCRIPTION: Perform an instruction with no processing results.

The NOP instruction has no effect on processing except to increment the program counter by 1. It is useful for "padding" object programs for future patching or for replacing unwanted object code during program debugging.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
NOP	Implied	NOP	>00	4

PROCESS: None

FLAGS: Unchanged

OR

OR GENERAL-PURPOSE REGISTER

DESCRIPTION: Logically "or" each bit of the byte represented by the source operand with the corresponding bit of the byte represented by the destination operand. Then store the resulting value in the byte represented by the destination operand and set the Zero and Negative flags according to the value.

When two bits are ored, the resulting bit is ONE if either or both of them have a value of ONE.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
OR B,A	Implied	OR B,A	>64	5
OR Rn,A	Register file	OR R2,A	>14 >02	8
OR Rn,B	Register file	OR R15,B	>34 >0F	8
OR Rn,Rn	Register file	OR R66,R17	>44 >42 >11	10
OR %N,A	Immediate	OR %>35,A	>24 >35	7
OR %N,B	Immediate	OR %255,B	>54 >FF	7
OR %N,Rn	Immediate	OR %127,R96	>74 >7F >60	9

PROCESS:

```
           B         A          A
PROCESS:  ( Rn ) OR ( B  ) --> ( B  )
           %N        Rn         Rn
```

FLAGS:

Zero:	Set to ONE if the result is zero
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Reset to ZERO

ORP

OR PERIPHERAL FILE REGISTER

DESCRIPTION: Logically "or" each bit of the byte from register A or B or each bit from an immediate value from the source operand with each corresponding bit of the peripheral-file register designated by the destination operand. Place the result in the peripheral file register, and set the Zero and Negative flag according to the result.

When two bits are "ored," the resulting bit is ONE if either or both of them have a value of ONE.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
ORP A,Pn	Peripheral file	ORP A,P48	>84 >30	10
ORP B,Pn	Peripheral file	ORP B,P89	>94 >59	9
ORP %N,Pn	Peripheral file	ORP %16,P8	>A4 >10 >08	11

PROCESS: Pn OR $\left\{ \begin{array}{c} A \\ B \\ \%N \end{array} \right\}$ --> Pn

FLAGS:

Zero:	Set to ONE if result is zero
Negative:	Set to ONE if BIT 7 of result is ONE
Carry:	Reset to ZERO

POP

POP REGISTER FROM STACK

DESCRIPTION: Move the value at the top of the stack to either a specified general-purpose register or the status register. Then decrement the stack pointer.

The Zero and Negative flags are set on the value popped into a register. When a value is popped into the status register, flags are set in accordance with the bits that make up the value.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
POP A	Implied	POP A	>B9	6
POP B	Implied	POP B	>C9	6
POP Rn	Register file	POP R2	>D9 >02	8
POP ST	Implied	POP ST	>08	6

PROCESS:

```
                          A
                          B
Top of stack --> {          }
                          Rn
                          ST
```

FLAGS:

Zero:	Set to ONE if the register popped contains zero
Negative:	Set to ONE if Bit 7 of the register popped is ONE
Carry:	Reset to ZERO

PUSH

PUSH REGISTER

DESCRIPTION: Increment the stack pointer by one and move the value in either a specified general-purpose register or the status register to the location pointed to.

The Zero and Negative flags are set on the value pushed, except when the status register is pushed. Flags remain the same when the status register is pushed.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
PUSH A	Implied	PUSH A	>B8	6
PUSH B	Implied	PUSH B	>C8	6
PUSH Rn	Register file	PUSH R9	>D8 >09	8
PUSH ST	Implied	PUSH ST	>0E	6

```
           A
           B
PROCESS:  {   } --> Top of stack
           Rn
           ST
```

FLAGS:

Zero:	Set to ONE if the register pushed contains zero
Negative:	Set to ONE if Bit 7 of the register pushed is ONE
Carry:	Reset to ZERO

RETI

RETURN FROM INTERRUPT

DESCRIPTION: Load the program counter with the topmost two bytes of the stack, load the status register from the next byte of the stack, and decrement the stack pointer three times.

The program counter and status register are restored with the values they contained upon interrupt.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
RETI	Implied	RETI	>0B	9

PROCESS: Restore PC (LSB, MSB) and Status register

FLAGS: Set according to the third stack byte (ST)

RETS

RETURN FROM SUBROUTINE

DESCRIPTION: Load the program counter with the topmost two bytes of the stack and decrement the stack pointer twice.

The program counter is restored with the value it contained upon executing the last subroutine call instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
RETS	Implied	RETS	>0A	7

PROCESS: Restore PC (LSB, MSB)

FLAGS: Unchanged

RL

ROTATE LEFT

DESCRIPTION: Shift all bits in the designated register one position to the left.

Bit 7 (the highest-order bit) is shifted into both bit 0 (the lowest-order bit) and into the Carry flag. The Zero and Negative flags are set according to the resulting value of the byte.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
RL A	Implied	RL A	>BE	5
RL B	Implied	RL B	>CE	5
RL Rn	Register file	RL R6	>DE >06	7

PROCESS: Bit n --> Bit n+1; Bit 7 --> Bit 0; Bit 7 --> Carry

FLAGS:

Zero:	Set to ONE if the byte is zero
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Set or reset by Bit 7 of original byte

RLC

ROTATE LEFT THROUGH CARRY

DESCRIPTION: Shift all bits in the designated register one position to the left through the Carry flag.

The Carry flag is shifted into bit 0 (the lowest-order bit) of the register, and Bit 7 (the highest-order bit) is shifted into the Carry flag. The Zero and Negative flags are set according to the resulting value of the byte.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
RLC A	Implied	RLC A	>BF	5
RLC B	Implied	RLC B	>CF	5
RLC Rn	Register file	RLC R6	>DF >06	7

PROCESS: Bit n --> Bit n+1; Carry --> Bit 0; Bit 7 --> Carry

FLAGS:

Zero:	Set to ONE if the byte is zero
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Set or reset by Bit 7 of original byte

RR

ROTATE RIGHT

DESCRIPTION: Shift all bits in the named register one position to the right.

Bit 0 (the lowest-order bit) is shifted into both Bit 7 (the highest-order bit) and into the Carry flag. The Zero and Negative flags are set according to the value of the resulting byte.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
RR A	Implied	RR A	>8C	5
RR B	Implied	RR B	>CC	5
RR Rn	Register file	RR R6	>DC >06	7

PROCESS: Bit n --> Bit n-1; Bit 0 --> Carry; Bit 0 --> Bit 7

FLAGS:

Zero:	Set to ONE if the byte is zero
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Set or reset by Bit 0 of original byte

RRC

ROTATE RIGHT THROUGH CARRY

DESCRIPTION: Shift all bits in the designated register one position to the right through the carry.

The Carry flag is shifted into bit 7 (the highest-order bit). Bit 0 (the lowest-order bit) is shifted into the Carry flag. The Zero and Negative flags are set according to the resulting value of the byte.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
RRC A	Implied	RRC A	>BD	5
RRC B	Implied	RRC B	>CD	5
RRC Rn	Register file	RRC R6	>DD >06	7

PROCESS: Bit n --> Bit n-1; Carry --> Bit 7; Bit 0 --> Carry

FLAGS:	Zero:	Set to ONE if the byte is zero
	Negative:	Set to ONE if Bit 7 of the result is ONE
	Carry:	Set or reset by Bit 0 of original byte

SBB

BINARY SUBTRACT WITH BORROW

DESCRIPTION: Subtract the value in the source register or the value of the immediate operand by two's complement addition from the destination register. If the Carry flag has been reset to ZERO by execution of a previous instruction, decrement the difference by one. Store the difference in the destination register.

The Zero and Negative flags are set or reset according to the difference. The Carry flag is reset to ZERO if a borrow has occurred (i.e., if bit Bit 7 has been decremented past ZERO).

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
SBB B,A	Implied	SBB B,A	>6B	5
SBB Rn,A	Register file, Implied	SBB R2,A	>1B >02	8
SBB Rn,B	Register file, Implied	SBB R15,B	>3B >0F	8
SBB Rn,Rn	Register file	SBB R66,R17	>4B >42 >11	10
SBB %N,A	Immediate, Implied	SBB %>35,A	>2B >35	7
SBB %N,B	Immediate, Implied	SBB %100,B	>5B >64	7
SBB %N,Rn	Immediate, Register file	SBB %126,R96	>7B >7E >60	9

PROCESS:

```
  B        A                  A
( Rn ) - ( B  ) - NOT Carry --> ( B  )
  %N       Rn                 Rn
```

FLAGS:

Zero: Set to ONE if the difference is zero
Negative: Set to ONE if Bit 7 of the difference is ONE
Carry: Reset to ZERO if Bit 7 of the difference has been decremented past ZERO

SETC

SET CARRY FLAG

DESCRIPTION: Set the Carry and Zero flags to ONE. Reset the Negative flag to ZERO.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
SETC	Implied	SETC	>07	5

PROCESS: Set or reset flags

FLAGS:

Zero:	Set to ONE.
Negative:	Reset to ZERO
Carry:	Set to ONE

STA

STORE THE A REGISTER IN MEMORY

DESCRIPTION: Store the value in the A register into a memory location. The memory location is specified by immediate, relative, or indexed addressing. Set the Zero and Negative flags according to the value of the byte stored.

In direct mode, the location is specified in the instruction operand. In indirect mode, it is specified by the contents of the named register (least significant byte) and the register one address below it (most significant byte). In indexed mode, it is specified by the sum of the expression in the operand and the contents of the B register.

Where immediate hexadecimal values are shown in the table below, symbolic values assigned in equate instructions and labels can also be used.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
STA @EXPR	Direct	STA @>8000	>8B >80 >00	11
STA *RN	Indirect	STA *R30	>9B >1E	10
STA @EXPR(B)	Indexed	STA @300(B)	>AB >01 >2C	13

PROCESS: A --> Memory

FLAGS:

Zero:	Set to ONE if the byte stored is zero
Negative:	Set to ONE if Bit 7 of byte stored is ONE
Carry:	Reset to ZERO

STSP

STORE STACK POINTER

DESCRIPTION: Move the value in the stack pointer into the B register.

FORM.	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
STSP	Implied	STSP	>09	6

PROCESS: SP --> B

FLAGS: Unchanged

SUB

BINARY SUBTRACT

DESCRIPTION: Subtract the contents of the source register or the value of the immediate operand by two's complement addition from the destination register. Store the difference in the destination register.

The Zero and Negative flags are set or reset according to the difference. The Carry flag is reset to ZERO if a borrow has occurred (i.e., if Bit 7 has been decremented past ZERO).

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
SUB B,A	Implied	SUB B,A	>6A	5
SUB Rn,A	Register file, Implied	SUB R2,A	>1A >02	8
SUB Rn,B	Register file, Implied	SUB R15,B	>3A >0F	8
SUB Rn,Rn	Register file	SUB R66,R17	>4A >42 >11	10
SUB %N,A	Immediate, Implied	SUB %>33,A	>2A >33	7
SUB %N,B	Immediate, Implied	SUB %100,B	>5A >64	7
SUB %N,Rn	Immediate, Register file	SUB %126,R96	>7A >7E >60	9

PROCESS:

```
  B        A          A
( Rn ) - ( B  ) --> ( B  )
  %N       Rn         Rn
```

FLAGS:

Zero:	Set to ONE if the difference is zero
Negative:	Set to ONE if Bit 7 of the difference is ONE
Carry:	Reset to ZERO if Bit 7 of the difference has been decremented past ZERO

TRAP N

TRAP TO SUBROUTINE-CALL VECTOR N

DESCRIPTION: Save the address of the next-instruction program counter on the stack and load the program counter with a value from a 24-address table at the top of CC-40 memory.

The 24 two-byte values stored in memory from >FD00 through >FFFF are inversely indexed on the vector number. TRAP 0 uses the value in >FFFE->FFFF, TRAP 13 uses the value in >FFE0->FFE1, and TRAP 23 uses the value at >FD00->FD01.

Four of the table vectors are dedicated to system-hardware use. The TRAP 0 vector points to system-reset routines. TRAP 1, TRAP 2, and TRAP 3 point to interrupt-handling routines for interrupts 1, 2, and 3 respectively.

Single-byte machine-language command codes for TRAP 0 to TRAP 23 are listed in the table of COMMANDS below.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
TRAP N	Implied	TRAP 0	>FF	14

COMMANDS:

N	CMD CD	N	CMD CD	N	CMD CD
0	>FF	8	>F7	16	>EF
1	>FE	9	>F6	17	>EE
2	>FD	10	>F5	18	>ED
3	>FC	11	>F4	19	>EC
4	>FB	12	>F3	20	>EB
5	>FA	13	>F2	21	>EA
6	>F9	14	>F1	22	>E9
7	>F8	15	>F0	23	>E8

PROCESS: Save PC (MSB, LSB) on the stack and branch to address in the TRAP table

FLAGS: Unchanged

SWAP

SWAP NIBBLES IN REGISTER

DESCRIPTION: Exchange the least-significant nibble (four-bit value) in the specified general-purpose register with the most-significant nibble. Set the Zero and Negative flags according to the resulting byte, and set the Carry flag if bit 0 of the resuting byte contains a ONE.

The Carry flag is set on bit 0 of the resulting byte. (The SWAP instruction produces the same results as the execution of four consecutive Rotate Left instructions.)

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
SWAP A	Implied	SWAP A	>B7	8
SWAP B	Implied	SWAP B	>C7	8
SWAP Rn	Register file	SWAP R8	>D7 >08	10

PROCESS: Bits 7-4 <--> Bits 3-0

FLAGS:

Zero:	Set to ONE if the register contains zero
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Set or reset according to Bit 0 of the result

TSTA

TEST THE A REGISTER

DESCRIPTION: Set or reset the Zero and Negative status flags based on the value in the A register. Reset the Carry bit to ZERO.

This instruction produces the same object code as the CLRC instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
TSTA	Implied	TSTA	>B0	6

PROCESS: Set or reset flags.

FLAGS:

Zero Flag:	Set to ONE if A contains zero
Negative Flag:	Set to ONE if Bit 7 of A is ONE
Carry Flag:	Reset to ZERO

TSTB

TEST THE B REGISTER

DESCRIPTION: Set or reset the Zero and Negative status flags based on the value in the B register. Reset the Carry bit to ZERO.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
TSTB	Implied	TSTB	>C1	6

PROCESS: Set or reset flags.

FLAGS:

Zero Flag:	Set to ONE if B contains zero
Negative Flag:	Set to ONE if Bit 7 is ONE
Carry Flag:	Reset to ZERO

XCHB

EXCHANGE WITH B REGISTER

DESCRIPTION: Exchange the value in the specified general-purpose register with the value in the B register.

The Zero and Negative flags are set according to the original contents of B. The XCHB B instruction produces the same results as the TSTB instruction.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
XCHB A	Implied	XCHB A	>B6	6
XCHB B	Implied	XCHB B	>C6	6
XCHB Rn	Register file	XCHB R80	>D6 >50	8

PROCESS:

```
 A
(B ) <--> B ; Set flags
 Rn
```

FLAGS:

Zero:	Set to ONE if the B register originally contains zero
Negative:	Set to ONE if Bit 7 of the value originally in B is ONE
Carry:	Reset to ZERO

XOR

EXCLUSIVE OR GENERAL-PURPOSE REGISTER

DESCRIPTION: Perform an exclusive-or operation on each bit of the byte represented by the source operand and each corresponding bit of the byte represented by the destination operand. Store the result in the byte represented by the destination operand, and set the Zero and Negative flags according to the result.

When two bits are "exclusive ored," the resulting bit is ONE if one, but not both, of them have a value of ONE.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
XOR B,A	Implied	XOR B,A	>65	5
XOR Rn,A	Register file	XOR R2,A	>15 >02	8
XOR Rn,B	Register file	XOR R15,B	>35 >0F	8
XOR Rn,Rn	Register file	XOR R66,R17	>45 >42 >11	10
XOR %N,A	Immediate	XOR %>33,A	>25 >33	7
XOR %N,B	Immediate	XOR %255,B	>55 >FF	7
XOR %N,Rn	Immediate	XOR %127,R96	>75 >7F >60	9

PROCESS:

```
  B          A          A
( Rn ) XOR ( B  ) --> ( B  )
  %N         Rn         Rn
```

FLAGS:

Zero:	Set to ONE if result is zero
Negative:	Set to ONE if Bit 7 of the result is ONE
Carry:	Reset to ZERO

XORP

EXCLUSIVE OR PERIPHERAL FILE REGISTER

DESCRIPTION: Perform an exclusive-or operation on each bit of the byte from register A or B or from the immediate value in the source operand with each corresponding bit of the byte in the peripheral-file register designated by the destination operand. Place the result in the peripheral file register, and set the Zero and Negative flags according to the result.

The result of an exclusive-or operation on two bits is ONE if one of the bits--but not both--are ONE.

FORM	ADDRESSING	SRC EXAMPLE	OBJ CODE	CYCLES
XORP A,Pn	Peripheral file	XORP A,P13	>B5 >0D	10
XORP B,Pn	Peripheral file	XORP B,P64	>95 >40	9
XORP %N,Pn	Peripheral file	XORP %1,P6	>A5 >01 >06	11

PROCESS:

```
          A
Pn XOR (  B  ) --> Pn
          %N
```

FLAGS:

Zero:	Set to ONE if result is zero
Negative:	Set to ONE if Bit 7 of result is ONE
Carry:	Reset to ZERO

APPENDIX A
VALUES IN BINARY, DECIMAL, AND HEX

There is correspondence between numbers entered by keyboard or produced by an ALDS program such as the editor, assembler, or loader and the state of bits in a byte is in accordance with the following equation.

NUM VAL = b7*128 + b6*64 + b5*32 + b4*16 +b3*8 +b2*4 + b1*2 + b0*1

where,

b7 = most significant bit in the byte
b0 = least significant bit in the byte, and so on

128 = 2 raised to the 7th power
1 = 2 raised to the 0th power, and so on

Bit 7 of a byte (the most-significant bit) is valued at 128, Bit 7 at 64, .., and Bit 0 at 1. Determining the value of a byte requires summing up the bit-position values (128, and 1) for which a bit is ONE and ignoring the values for which a bit is ZERO.

The following table shows the values (expressed as standard decimal numbers) of six typical bit combinations in a byte.

BITS IN BYTE	DECIMAL VALUE
00000001	1
10000000	128
10001001	137

BITS IN BYTE	DECIMAL VALUE
01010101	85
10101011	171
11111111	255

"Values" in the table above are expressed in decimal numbers. Decimal numbers, however, are clumsy for expressing values which must also be expressed in binary numbers. More practically, values entered through a keyboard for expression in bytes are usually entered in "hexadecimal" or "hex" numbers rather than decimal numbers because of the correspondence of hex numbers to bit positions in a byte. (The DEBUG Monitor and the assembler will accept decimal numbers from keyboard entry and convert them internally to hex and binary, but values which are output by the debug monitor and assembler are always expressed in hex.

In hex numbers, numerals include "0" through "9" and "A" through "F" Therefore single-digit numbers can take on values between 0 and 15 (with "A"=10, "B"=11, "C"=12, "D"=13, "E"=14, and "F"=15. up to 15.. Expressed in hex numbers, the value equation for a binary number is, as shown below, somewhat simpler.

```
NUM VAL (HEX) = b7*>80 + b6*>40 + b5*>20 + b4*>10 + b3*>8 +b2*>4 + b1*>2+b0*>1

          where,

               b7  =  most significant bit in the byte
               b0  =  least significant bit in the byte, and so on

               >80 =  8*>10^1 (hex, ie., >10 hex = 16 decimal)
               >1  =  1*>10^0, and so on
```

The practicality of hexadecimal notation is twofold. First, the bit-position values follow a regular progression of increasing hexadecimal value (i.e., 1,2,4,8,10,20,40,80). Second, all possible values of a byte can be expressed in two digits (e.g., FF = 255) each of which are directly representative of a one of the two groups of four bits (nibbles) in the byte.

The inconvenience of hexadecimal numbering is its possible confusion with decimal notation. To minimize this inconvenience, the CC-40 assembler uses a ">" (greater-than) sign to uniquely identify hexadecimal numbers. Thus >10 has a value of 16, but 10 has a value of 10. The correspondence of 4-bit binary numbers to hexadecimal numbers and decimal numbers is shown in the following table. Note that the CC-40 assembler also uniquely prefixes binary values with a "?" (question mark).

BINARY	HEX	DECIMAL
?0000	>0	0
?0001	>1	1
?0010	>2	2
?0011	>3	3
?0100	>4	4
?0101	>5	5
?0110	>6	6
?0111	>7	7
?1000	>8	8
?1001	>9	9
?1010	>A	10
?1011	>B	11
?1100	>C	12
?1101	>D	13
?1110	>E	14
?1111	>F	15

Hex numbers represent a byte with two hex digits which correspond directly to the bit composition of the byte. The first hex digit represents the most-significant four bits, and the second represents the least-significant.

In the byte represented by >81, for example, the highest-order bit (b7) of the higher four-bit grouping is "high" or "ON," and so is the lowest-order bit (b1) of the lower four-bits grouping. The remaining bits "low" or "OFF". Similarly, the byte represented by >A5, has the most significant and third-most significant bits of the higher order grouping (b7 and b5) on and the second-most and least significant of the lower grouping (b1 and b3) on. These four-bit groupings within a byte can play such an important role in programming that they have a special (though humorous) name: they are called nibbles.

Hex numbers follow the same rules of arithmetic that decimal or binary numbers follow. In addition operations, a carry out of a digit position occurs when the sum is incremented past the highest possible digit (9 in decimal, >F in hex). In a subtraction operation, a borrow transfers the highest-possible digit plus 1 to the digit-position to the right in the minuend (9+1 in decimal, >F+>1 in hex) and decrements the digit in the position "borrowed from" by 1. Some examples of hexadecimal arithmetic are as follows.

```
.------------------------------------------------------------------------.
!                                                                        !
!     6        7        B        D      }                                !
!   + 4      + 8      - 3      - 3      } NO CARRIES, NO BORROWS         !
!   ---      ---      ---      ---      }                                !
!     A        F        8        A      }                                !
!                                                                        !
!                                                                        !
!     9       FF       10       01      }                                !
!   + E      + 3      - 4      - 3      } CARRIES AND BORROWS            !
!   ---      ---      ---      ---      }                                !
!    17    CY+02        C    BW+FE      }                                !
!                                                                        !
.------------------------------------------------------------------------.
```

APPENDIX B
SCREEN CONTROL CODES FOR SELECTED TERMINALS

The following table lists the codes used to control screen operation on five CRT terminals.

	LEER SIEGLER ADM3A	LEER SIEGLER ADM31	HAZELTINE 1420	SOROC 10120	RADIO SHACK MODEL II
MOVE CUSOR	27 61	27 61	27 61	27 61	27 61
ROW FIRST	1	1	0	1	1
OFFSET	32	32	0	32	32
CLEAR SCREEN	27 58	26	27 28	27 42	26
CURSOR UP	11	11	27 12	11	11
LINE FEED (CURSOR DOWN)	10	10	10	10	10
CURSOR RIGHT	12	12	16	12	12
CURSOR LEFT	8	8	8	8	8
BEEP	7	7	7	7	
ERASE TO END OF SCREEN	27 12 1		27 24	27 89	16
ERASE TO END OF LINE	27 84		27 15	27 84	17
INSERT LINE	27 69		27 26		

APPENDIX C
EDITOR DEFAULT KEYBOARD DEFINITIONS

FUNCTIONS OF KEYS	KEYS USED	
	EXTERNAL KEYBOARD (Televideo 920)	INTERNAL KEYBOARD (CC-40)
Copy	[ESC][C]	[FN][C]
Delete	[ESC][D]	[FN][D]
Find	[ESC][F]	[FN][F]
Format	[ESC][N]	[FN][N]
Help	[ESC][H]	[FN][H]
Jump	[ESC][J]	[FN][J]
List	[ESC][L]	[FN][L]
Move	[ESC][M]	[FN][M]
Quit	[ESC][Q]	[FN][Q]
Replace	[ESC][R]	[FN][R]
Save	[ESC][S]	[FN][S]
Tab define	[ESC][T]	[FN][T]
Undo	[ESC][U]	[FN][U]
Verify	[ESC][V]	[FN][V]
Auto	[CTL][A]	[CTL][A]
Back tab	[CTL][U]	[CTL][<-]
Clear	[CTL][C]	[CLR]
Command exit	[CTL][X]	[CTL][X]
Delete char	[DEL] (>7F)	[SHIFT][<-]
Delete line	[CTL][D]	[CTL][D]
Down	[!] ([CTL][J])	[!]
Enter	[ENTER] ([CTL][M])	[ENTER]
Erase field	[CTL][E]	[CTL][!]
Home	[HOME] (>1E)	[CTL][^]
Insert char	[CTL][Q]	[SHIFT][->]
Insert line	[CTL][N]	[CTL][N]
Left	[<-] ([CTL][H])	[<-]
Line display	[CTL][T]	[CTL][T]
Page back	[ESC][^]	[CTL][-]
Page forward	[ESC][!]	[CTL][+]
Right	[->] ([CTL][L])	[->]
Tab	[TAB] ([CTL][I]	[CTL][->]
Up	[^] ([CTL][K]	[^]
View	[CTL][V]	[CTL][V]

APPENDIX D
USING THE HEX-BUS-tm VIDEO INTERFACE

The HEX-BUS video interface provides multiple-line display for the screen editor. Entry of editor commands and text is performed by the CC-40 internal keyboard. The video interface displays 24 lines of 40 characters on video monitor.

The characteristics required by the VIDEO INTERFACE for standard 24 line by 40 character operation during an OPEN command are "40.R=N".

The VIDEO INTERFACE is compatable with the default screen parameter table for the internal keyboard. The defaults for display codes, display size, and optional codes are also the same also.

The screen editor supports the 80-column, but line numbers cannot be used. The 80-column option creates a virtual screen 80 columns by 24 lines displayed through a 40 column by 24 line window.

Use of the 80-column option requires the screen size to be redefined to be 80,24.

With th 80-column option, line numbers can be scrolled off the screen and cannot be brought back into view. Movement of the cursor controls the shifting of the window over the virtual screen. Because the screen editor does not allow the cursor to backspace over line numbers, it is impossible to bring line nümbers back into view when they have been scrolled.

The option string required by the VIDEO INTERFACE in this virtual mode of operation is: "40.R=N.80".

APPENDIX E
TAGGED OBJECT CODE

The format of the machine-language program output by the assembler is tagged object format.

Tagged object code consists of a series of variable-length fields. Each field begins with a single alphabetic label or tag. The label or tag uniquely identifies the format, the contents, and the type of data following it in the field.

The following table lists the elements of fields containing either absolute or relocatabe binary data. As the table enries show, the alphabetic tag which identifies whether the following data is a byte (8 bits) or word (16 bits) and is absolute or relocatable is followed immediately by the appropriate one or two bytes of data.

TABLE 6-1
TAGGED-OBJECT FIELDS CONTAINING BINARY DATA

TAG	ELEMENTS OF FIELD	DESCRIPTION
H	Byte	Absolute byte of data
I	Word	Absolute data
J	Word	Relocatable data

The fields tagged with H, I, and J contain the code which is rewritten as executable-object code after processing by the linker. Opcodes and bytes of data defined for assembly with BYTE instructions occur in H fields. Words of data which are either not related to addresses or defined by absolute addressing (following AORG in assembly) occur in I fields. Address-related (relocatable) words of data which must be recalculated or biased for execution occur in J fields.

The remaining types of fields contain instructions for linking and subsequent loading of the binary data for execution. The following table lists these instructions.

TAG	ELEMENTS OF FIELD	DESCRIPTION
A	Word Program-ID	Program indentifier (Crunched format) Word = Highest relocatable address
B	Word Program-ID	Program indentifier (Uncrunched format) Word = Highest relocatable address
C	Word	Absolute load address
D	Word	Relocatable load address
E	Byte Symbol	REFed symbol, byte = symbol length - 1

F	Byte Word Symbol	Absolute DEFed symbol
		Word = symbol value,
		Byte = symbol length - 1

G	Byte Word Symbol	Relocatable DEFed symbol
		Word = symbol value,
		Byte = symbol length - 1

K	Word1 Word2	External reference Word1 = Reference number Word2 = Value to be added to reference
L	Word	Checksum
M	Word	Ignore checksum
N		End of record
O		End Of File

In the table shown above, a byte is an 8-bit value and a word is a 16-bit value. A symbol consists of from one to eight

ASCII characters, and the program ID consists of a symbol padded with blanks to the right, if necessary, to make up eight characters.

The tagged-object code output by the assembler and input to the linker is similar to the tagged-object code output by the TI-990/DX-10 TMS-7000 assembler (see chapter 8 for a detailed comparison).

APPENDIX F
ASSEMBLER PSEUDO OPS AND DIRECTIVES

PSEUDO OPERATIONS IN ALPHABETICAL ORDER

BYTE:	Write one or more single-byte values specified by the operand into the object file.
DATA:	Write one or more two-byte (word) values specified by the operand into the object file.
RTEXT:	Write the ASCII representations of characters in the string specified by the operand into the object file in reverse order (last character at lowest object-file address, first character at highest object-file address. This instruction is particularly useful because the TMS7000-family processor has a memory-pointer decrementing instruction (DECD) but no pointer incrementing instruction. A label preceding the RTEXT instruction is evaluated to the address of the last byte of the string (highest address).
TEXT:	Write the ASCII representations of characters in the string specified by the operand into the object file in normal order (first character at the lowest object-file address, last character at highest object-file address). A label preceding the TEXT instruction is evaluated to the address of the first byte of the string (lowest address).

DIRECTIVES IN ALPHABETICAL ORDER

AORG:	Absolute origin: Set the absolute-address origin of subsequent object code to the value specified in the operand.
BES:	Block ending with symbol: Skip past object-code locations for the number of bytes specified in the operand and assign any label for the skipped bytes to the address of the first byte after the block skipped.
BSS:	Block starting with symbol: Skip past object-code locations for the number of bytes specified in the operand and assign any label for the skipped bytes to the address of the first byte skipped.
COPY:	Copy file: Copy or include another source file named in the operand into the current file during assembly only (source files remain separate following assembly).

DEF: Define symbol: Include the symbol specified in the operand as a "public" symbol in the object file with a value made available to the linker for use in other programs being linked to the current one.

END: End source code: Terminate assembly at this line.

EQU: Equate: Assign the value in the operand to the label

IDT: Identify program: Identify the program by the string in operand both for the listing page heading and for access by the linker.

LIST: List lines: List the lines of the assembled program following this directive and set the number of lines to be printed on each page prior to automatic end-of-page eject.

OPTION: Perform actions indicated by letters in operand (separated from each other by commas); the letters are as follows.

B: Truncate byte lines: Print only the first line of object code generated by each BYTE pseudo-op.

D: Truncate data lines: Print only the first line of object code generated by each DATA pseudo-op.

F: Finish line truncation: Print full object code generated by BYTE, DATA, TEXT, and RTEXT pseudo-ops.

R: Produce a Reduced cross-reference table: Produce a cross-reference as above, except that only symbols from copied files which are referenced are listed in the table.

T: Truncate text lines: Print only the first line of object code generated by each TEXT pseudo op or the last line of each RTEXT pseudo-op.

X: Produce a Cross-reference table: Produce a detailed cross-reference table listing the value of each symbol, the source-file line on which the value definition occurs, and the source-file lines containing references to the symbol (in operands).

PAGE: Eject page: Eject a page in listing.

REF: Reference symbol: Include the symbol specified in the operand as an "external" symbol in the object file with a value to be searched for by the linker among other programs being linked to the current one.

RORG:	Relocatable origin: Set the relative-address origin of the object code to an address displaced from relative address 0 by the value specified in the operand.
TITL:	Title the following pages: Provide a page title for the next page and subsequent pages from the string in the operand.
UNL:	Unlist lines: Do not list the lines of assembled program following this directive.

OP CODE MAP AS IN
USERS GUIDE

APPENDIX H
ASCII CODE CHART

LEAST-SIGNIFICANT NIBBLE

MOST-SIGNIFICANT NIBBLE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0	1	2	3	4	5	6	BEL 7	RS 8	HT 9	LF 10	VT 11	FF 12	CR 13	SO 14	SI 15
1	16	DC1 17	DC2 18	DC3 19	DC4 20	21	22	23	24	25	26	ESC 27	28	29	30	31
2	SPACE 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
3	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
4	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
5	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	] 92	^ 93	94	95
6	96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
7	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	' 124	} 125	~ 126	127

TABLE 2-22
OP CODE MAP

	MOST-SIGNIFICANT NIBBLE															
LEAST SIGNIFICANT NIBBLE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP								MOVP Pn,A			TSTA/ CLRC	MOV A,B	MOV A,Rn	JMP J	TRAP 15
1	IDLE									MOVP Pn,B			TSTB	MOV B,Rn	JN J	TRAP 14
2		MOV Rn,A	MOV %n,A	MOV Rn,B	MOV Rn,Rn	MOV %n,B	MOV B,A	MOV %n,Rn	MOVP A,Pn	MOVP B,Pn	MOVP %n,Pn	DEC A	DEC B	DEC Rn	JZ/JEQ J	TRAP 13
3		AND Rn,A	AND %n,A	AND Rn,B	AND Rn,Rn	AND %n,B	AND B,A	AND %n,Rn	ANDP A,Pn	ANDP B,Pn	ANDP %n,Pn	INC A	INC B	INC Rn	JC/JHS J	TRAP 12
4		OR Rn,A	OR %n,A	OR Rn,B	OR Rn,Rn	OR %n,B	OR B,A	OR %n,Rn	ORP A,Pn	ORP B,Pn	ORP %n,Pn	INV A	INV B	INV Rn	JP J	TRAP 11
5	EINT	XOR Rn,A	XOR %n,A	XOR Rn,B	XOR Rn,Rn	XOR %n,B	XOR B,A	XOR %n,Rn	XORP A,Pn	XORP B,Pn	XORP %n,Pn	CLR A	CLR B	CLR Rn	JPZ J	TRAP 10
6	DINT	BTJO Rn,A,J	BTJO %n,A,J	BTJO Rn,B,J	BTJO Rn,Rn,J	BTJO %n,B,J	BTJO B,A,J	BTJO %n,Rn,J	BTJOP A,Pn,J	BTJOP B,Pn,J	BTJOP %n,Pn,J	XCHB A	XCHB B	XCHB Rn	JNZ/JNE J	TRAP 9
7	SETC	BTJZ Rn,A,J	BTJZ %n,A,J	BTJZ Rn,B,J	BTJZ Rn,Rn,J	BTJZ %n,B,J	BTJZ B,A,J	BTJZ %n,Rn,J	BTJZP A,Pn,J	BTJZP B,Pn,J	BTJZP %n,Pn,J	SWAP A	SWAP B	SWAP Rn	JNC/JL J	TRAP 8
8	POP ST	ADD Rn,A	ADD %n,A	ADD Rn,B	ADD Rn,Rn	ADD %n,B	ADD B,A	ADD %n,Rn	MOVD %nn,Rn	MOVD Rn,Rn	MOVD %nn(B),Rn	PUSH A	PUSH B	PUSH Rn	TRAP 23	TRAP 7
9	STSP	ADC Rn,A	ADC %n,A	ADC Rn,B	ADC Rn,Rn	ADC %n,B	ADC B,A	ADC %n,Rn				POP A	POP B	POP Rn	TRAP 22	TRAP 6
A	RETS	SUB Rn,A	SUB %n,A	SUB Rn,B	SUB Rn,Rn	SUB %n,B	SUB B,A	SUB %n,Rn	LDA @nn	LDA *Rn	LDA @nn(B)	DJNZ A	DJNZ B	DJNZ Rn	TRAP 21	TRAP 5
B	RETI	SBB Rn,A	SBB %n,A	SBB Rn,B	SBB Rn,Rn	SBB %n,B	SBB B,A	SBB %n,Rn	STA @nn	STA *Rn	STA @nn(B)	DECD A	DECD B	DECD Rn	TRAP 20	TRAP 4
C		MPY Rn,A	MPY %n,A	MPY Rn,B	MPY Rn,Rn	MPY %n,B	MPY B,A	MPY %n,Rn	BR @nn	BR *Rn	BR @nn(B)	RR A	RR B	RR Rn	TRAP 19	TRAP 3
D	LDSP	CMP Rn,A	CMP %n,A	CMP Rn,B	CMP Rn,Rn	CMP %n,B	CMP B,A	CMP %n,Rn	CMPA @nn	CMPA *Rn	CMPA @nn(B)	RRC A	RRC B	RRC Rn	TRAP 18	TRAP 2
E	PUSH ST	DAC Rn,A	DAC %n,A	DAC Rn,B	DAC Rn,Rn	DAC %n,B	DAC B,A	DAC %n,Rn	CALL @nn	CALL *Rn	CALL @nn(B)	RL A	RL B	RL Rn	TRAP 17	TRAP 1
F		DSB Rn,A	DSB %n,A	DSB Rn,B	DSB Rn,Rn	DSB %n,B	DSB B,A	DSB %n,Rn				RLC A	RLC B	RLC Rn	TRAP 16	TRAP 0

SYMBOLS:	Rn = R0 THROUGH R127	Pn = P0 THROUGH P255	n = ONE-BYTE VALUE	nn = TWO-BYTE (WORD) VALUE
	J = JUMP DISPLACEMENT	@ = DIRECT MEM ADR	* = REGISTER INDIRECT	(B) = DIRECT, INDEXED ON REGISTER B