SECTION V/

REFERENCE

SECTION V/

REFERENCE

This section summarizes all the features of the Disk EDTASM.

Reference A/ Editor Commands

Definition of Terms

line
A *line* number in the program. Any *lines* between 0-63999 may be used. These symbols may be used:

- # First line in the program
- * Last line in the program
- . Current line in the program

current line
The last line inserted, edited, or printed.

startline
The line where an operation will begin. In most commands *startline* is optional. If *startline* is omitted, the current line is used.

An asterisk (*) denotes a comment line when used as the first character in the line.

range
The line or lines to use in an operation. If the *range* includes more than one line, they must be specified with one of these symbols:

- : to separate the *startline* from the ending line
- , to separate the *startline* from the number of lines

increment
The *increment* to use between lines. In most commands, *increment* is optional. If the *increment* is omitted, the last specified *increment* is used. On startup, *increment* is set to 10.

filespec
A DOS disk file specification in the format:

filename/ext:drive

COMMANDS	PAGES DISCUSSED

C*startline, range, increment*
Copies range to a new location beginning with *startline* using the specified *increments*. *startline*, *range*, and *increment* must be included.

```
C500,100:150,10
```

D*range*
Deletes *range*. If *range* is omitted, *current line* is deleted.

```
D100        D100:150    D
```

E*line*
Enters a *line* for editing. If *line* is omitted, *current line* is used.

```
E100    E
```

These are the editing subcommands:

A	Cancels all changes and restarts the edit.
n*C*string	Changes *n* characters to *string*. If *n* is omitted, changes the character at the current cursor position.
***n*D**	Deletes *n* characters. If *n* is omitted, deletes character at current cursor position.
E	Ends line editing and enters all changes without displaying the rest of the line.
H	Deletes rest of line and allows insert.
I *string*	Inserts *string* starting at the current cursor position. While in the mode, ← deletes a character, and (SHIFT) ↑ (ESCAPE) ends the mode.
K	Deletes all characters from the current cursor position to the end of the line.
L	Lists current line and continues edit.
n*S*character	Searches for *n*th occurrence of *character*. If *n* is omitted, searches for the first occurrence.
X	Extends line.
(ENTER)	Ends line editing, enters all changes and displays the rest of the line.
(SHIFT) ↑	Escapes from subcommand.
n (SPACEBAR)	Moves cursor n positions to the right. If *n* is omitted, moves one position.
n ←	Moves cursor *n* positions to the left. If *n* is omitted, moves the cursor one position.

F*string*
Finds the string of characters. Search begins with the *current line* and ends each time *string* is found. If *string* is omitted, the last string defined is used.

```
FABC      F
```

H*range*
Prints *range* on the printer. If *range* is omitted, the *current line* is printed.

```
H100        H100:200    H
```

I*startline,increment*
Inserts lines up to 127 characters long beginning at *startline*, using the specified *increment*. *startline* and *increment* are optional.

```
I150,5      I200        I,10
```

K
Returns to DOS.

LCA *filename*
Loads *filename* from tape into the edit buffer. A is optional. If included, *filename* is appended to the edit buffer. If *filename* is omitted, the next tape file is loaded.

```
LC SAMPLE/EXT          LCA SAMPLE/EXT
```

LDA *filespec*
Loads the specified file from disk into the edit buffer. A is optional. If included, *filespec* is appended to the current contents of the edit buffer. If extension is omitted, /ASM is used.

```
LD SAMPLE/EXT          LDA SAMPLE/EXT
```

Mstartline, range, increment
Move command, works like copy except the original lines are deleted.

Nstartline, increment
Renumbers beginning at *startline*, using the specified *increment. startline* and *increment* are optional.

```
N100,50     N100        N
```

O
Shows the hexadecimal values of (1) the first available memory address, (2) the last available address, and (3) USRORG, the address where the assembler originates an /IM assembly with the /MO switch. Then, prompts you to change USRORG.

```
O
```

Prange
Displays *range* on the screen.

```
P100:200    P100!5    P#    P+
P (Prints 15 lines to the screen)
```

Q
Returns to BASIC.

R *startline, increment*
Allows you to replace *startline* and then insert lines using *increment. startline* and *increment* are optional.

```
R100,10     R100        R
```

S
Shows the current printer parameters and lets you change them.

Trange
Prints *range* to the printer, without line numbers.

```
T100        T100:500
```

Vfilename
Verifies *filename* (a tape file) to ensure that it is free of checksum errors. Works like BASIC's SKIPF command. If *filename* is omitted, this command verifies the next file found.

WC *filename*
Writes *filename* to tape. If *filename* is omitted, NONAME is used.

WD *filespec*
Writes *filespec* to disk. If the extension is omitted, ASM is used.

```
WD SAMPLE/EXT
```

Z
Jumps to ZBUG (EDTASM system only).

(↑)
Scrolls up in memory.

(↓)
Scrolls down in memory.

(SHIFT) (CLEAR)
Is used to create a backslash (\).

Reference B/ Assembler Commands and Switches

COMMANDS	PAGES DISCUSSED

AC *filename switch . . .*
Assembles the source program into machine code. If you specify the /IM switch, the assembly is in memory. If you specify *filename*, the assembly is saved on tape as *filename*. If you omit both *filename* and *switch*, the assembly is saved on tape as NONAME.

AD *filespec switch . . .*
Assembles the source program into machine code. Either the /IM switch or *filespec* is required: With /IM, the assembly is in memory; with *filespec*, the assembly is on disk. The D is optional.

There must be a space between *filespec* and *switch*.

The switches are:

/AO	Absolute origin.(Applies only If /IM is set.)
/IM	In-memory assembly.
/LP	Assembly listing on the printer.
/MO	Manual origin. (Applies only if /IM is set.)
/NL	No listing printed.
/NO	No object code generated.
/NS	No symbol table generated.
/SR	Single record.
/SS	Short screen.
/WE	Wait on assembly errors.
/WS	With symbols.

Examples:

```
AD SAMPLE
AD/IM/AO
AD SAMPLE /WE/SR
A SAMPLE/TST /WE
AC SAMPLE
AC
```

Reference C/ ZBUG Commands

Definition of Terms

expression
One or more numbers, symbols, or ASCII characters. If more than one is used, you may separate them with these operators:

Multiplication	*	Addition	+
Division	.DIV	Subtraction	–
Modulus	.MOD	Equals	.EQU
Shift	<	Not Equal	.NEG
Local And	.AND	Positive	+
Exclusive Or	.XOR	Negative	–
Logical Or	.OR	Complement	.NOT

address
A location in memory. This may be specified as an expression using numbers or symbols.

filename
A BASIC cassette file specification.

filespec
A DOS file specification. (The same as a BASIC specification.)

COMMANDS	PAGES DISCUSSED

C
Continues execution of the program after interruption at a breakpoint.

D
Displays all breakpoints that have been set.

E
Exits ZBUG and enters the editor. (This applies to the EDTASM ZBUG only, not to Stand-Alone ZBUG.)

Gaddress
Executes the program beginning at *address*.

K
Returns to DOS. (Applies to Stand-Alone ZBUG only.)

LC *filename address*
Loads *filename* from tape. The optional *address* offsets the file's loading address. If *filename* is omitted, the next file is loaded.

LD *filespec address*
Loads *filespec* from disk. The optional *address* offsets the file's loading address.

LDS *filespec address1 address2*
Loads *filespec* from disk with its appended symbol table. The optional *address1* offsets the file's loading address. The optional *address2* offsets the symbol table's loading address. Note that *address2* does not offset the values of the symbols. The D is optional.

PC *filename start address end address execution address*
Saves memory from *start address* to *end address* to tape. You must also specify an *execution address*, the first address to be executed when the file is loaded. *Filename* is optional; if omitted, NONAME is used.

PD *filespec start address end address execution address*
Saves memory to disk from *start address* to *end address*. You must also specify an *execution address*, the first address to be executed when the file is loaded. (The D is optional.)

PDS *filespec start address end address execution address*
Saves memory to disk from *start address* to *end address*, with the current appended symbol table. You must also specify an *execution address*, the first address to be executed when the file is loaded. (The D is optional.)

Q
Returns to BASIC. (Applies to Stand-Alone ZBUG only.)

R
Displays the contents of all the registers.

T*address1 address2*
Displays the memory locations from *address1* to *address2*, inclusive.

TH*address1 address2*
Prints the memory locations from *address1* to *address2*, inclusive.

U*source address destination address count*
Transfers the contents of memory beginning at *source address* and continuing for *count* bytes to another location in memory beginning with *destination address*.

V*filename*
Verifies date on the specified file or, if no *filename* is specified, the next file on tape.

X*address*
Sets a breakpoint at *address*. If *address* is omitted, the current location is used. Each breakpoint is assigned a number from 0 to 7. The first breakpoint set is assigned as Breakpoint 0. A maximum of eight breakpoints may be set at one time.

Y*n*
Deletes the breakpoint referenced by the *n* number. If *n* is omitted, all breakpoints are deleted.

Examination Mode Commands

A	ASCII Mode
B	Byte Mode
M	Mnemonic Mode
W	Word Mode

(The default is M)

Display Mode Commands

H	Half Symbolic
N	Numeric
S	Symbolic

(The default is S)

Numbering System Mode Commands

Obase	Output
Ibase	Input

(Base can be 8, 10, or 16. The default is 16)

Special Symbols

address/
register/

Opens *address* of *register* and displays its contents.
If *address* or *register* is omitted, the last address opened will be reopened. After the contents have been displayed, you may type:

new value	To change the contents.
(ENTER)	To close and enter any change.
(BREAK)	To close and delete any change.
(↓)	To open next *address* and enter any change.
(↑)	To open preceding *address*.
address (→)	To branch to the *address* pointed to by the instruction beginning at *address*. If *address* is omitted, the current *address* is used.
;	To force numeric display mode.
=	To force numeric and byte modes.
:	To force flags.*
"	To force ASCII mode.

address,
Executes *address*, if *address* is omitted, the next instruction is executed.

expression =
Calculates expression and displays the results.

* The colon does not actually have anything to do with the CC (status flag) register. It simply interprets the contents of the given address AS IF it contained flag bits.

Reference D/ EDTASM Error Messages

These are error messages you can get while in EDTASM or EDTASMOV:

BAD BREAKPOINT (*ZBUG*)
You are attempting to set a breakpoint (1) greater than 7, (2) in ROM, (3) at a SWI command, (4) at an address where one is already set.

BAD COMMAND (*Editor*)
An illegal command letter was used on the command line.

BAD COMMAND (*ZBUG*)
You are not using a ZBUG command.

BAD FILE DESCRIPTOR (*Disk,ZBug*)
The filespec is not in the proper DOS format. See "About This Manual" at the beginning of this manual for the proper file specification format.

BAD LABEL (*Assembler*)
The symbol you are using is (1) not a legal symbol, (2) not terminated with either a space, a tab, or a carriage return, (3) has been used with ORG or END, which do not allow labels, or (4) longer than six characters.

BAD MEMORY (*Assembler*)
You are attempting to do an in-memory assembly that would (1) overwrite system memory (an address lower than $1200) (2) overwrite the edit buffer of the symbol table, (3) go into the protected area set by USROG, or (4) go over the top of RAM.

If using the /AO switch, check to see that you've included an ORG instruction. When using /MO, check the addresses you set for BEGTEMP and USRORG. This could also be caused by the data not being stored correctly because of some code generated by an in-memory assembly. See *Chapter 7* for more information.

BAD MEMORY (*ZBUG*)
The data did not store correctly on a memory modification. This error will occur if you try to modify ROM addresses or try to store anything beyond MAXMEM.

BAD OPCODE (*Assembler*)
The op code is either not valid or is not terminated with a space, tab, or carriage return.

BAD OPERAND (*Assembler*)
There is some syntax error in the operand field. See *Section III* for the syntax of assembly language instructions.

BAD PARAMETERS (*Editor,ZBug*)
Usually this means your command line has a syntax error.

BAD PARAMETERS (*ZBUG*)
You have specified a filename that has more than eight characters.

BAD RADIX (*ZBUG*)
You have specified a numbering system other than 10, 8 or 16.

BUFFER EMPTY (*Editor*)
The specified command requires that there be some text in the Edit Buffer, and there isn't any.

BUFFER FULL (*Editor*)
There is not enough room in the edit buffer for another line of text.

BYTE OVERFLOW (*Assembler*)
There is a field overflow in an 8-bit data quantity in an immediate operand, an offset, a short branch, or an FCB pseudo op.

DIRECTORY FULL (*Disk*)
The directory does not have enough room for another entry. Use another diskette or delete a file (using the BASIC KILL command).

DISK FULL (*Disk*)
The diskette does not have enough room for another file. Use another diskette or delete a file (using the BASIC KILL command).

DISK WRITE PROTECTED (*Disk*)
You are attempting to write to a diskette that has the write-protect notch covered. Remove the write-protect label or use another diskette.

DOS ERROR (*Disk*)
This indicates an internal DOS error. It usually means either the DOS or the Editor/Assembler has been modified by the user program with harmful results.

DP ERROR (*Assembler*)
Direct Page error. The high order byte of an operand where direct addressing has been forced (,) does not match the value set by the most recent SETDP pseudo op.

DRIVE NOT READY (*Disk*)
The drive is not connected, powered up, working properly, or loaded properly.

END OF FILE (*Disk*)
Your program is attempting to access a record past the end of the file.

ENDC WITHOUT COND (*Assembler*)
The pseudo op ENDC was found without a matching COND having previously been encountered.

ENDM WITHOUT MACRO (*Assembler*)
The pseudo op ENDM was found without a matching MACRO having previously been encountered.

EXPRESSION ERROR (*Assembler and ZBUG*)
Either the syntax for the expression is incorrect (check *Chapter 9*) or the expression is dividing by zero.

FILE NOT FOUND (*Disk*)
The file is not on the disk's directory.

FM ERROR (*Editor, ZBUG and Disk*)
File Mode Error. The file you are attempting to load is not a TEXT file (if in the Editor) or a CODE file (if in ZBUG).

ILLEGAL NESTING (*Assembler*)
Illegal nesting conditions include the following:

1. Nested macro definitions.
2. Nested macro expansions.
3. Nested INCLUDE pseudo ops.
4. INCLUDE nested within a macro definition.

I/O ERROR (*Editor, ZBUG and Disk*)
Input/Output error. A checksum error was encountered while loading a file from a cassette tape. The tape may be bad, or the volume setting may be wrong. Try a higher volume.

MACRO FORWARD REFERENCE (*Assembler*)
A reference to the macro, which is defined on the current line, occurs previous to the macro definition.

MACRO TABLE FULL (*Assembler*)
The macro table is full, any additional entries will overwrite the symbol table. This happens when all memory allocated for the edit buffer, macro table, and symbol table has been used. Adjust USRORG using the Origin (O) command. (See the *Chapter 7*.)

MISSING END (*Assembler*)
Every assembly language program must have END as its last command.

MISSING INFORMATION (*Assembler*)
(1) There is a missing delimiter in an FCC pseudo op or (2) there is no label on a SET or EQU pseudo op.

MISSING OPERAND (*Assembler,ZBug*)
The command requires one or more operands.

MULTIPLY DEFINED SYMBOL (*Assembler*)
Your program has defined the same symbol with different values. If the error occurs in a macro expansion, use the /.1 notation to name the symbols. See *Chapter 12*.

NO ROOM BETWEEN LINES (*Editor*)
There is not enough room between lines to use the increment specified. Specify a smaller increment or renumber (N) the text using a larger increment. Remember that the last increment you used is kept until you specify a new one.

NO SUCH LINES (*Editor*)
The specified line or lines do not exist.

REGISTER ERROR (*Assembler*)
(1) No registers have been specified with a PSH/PUL instruction, (2) a register has been specified more than once in a PSH/PUL instruction, or (3) there is a register mismatch with an EXG/TFR instruction.

SEARCH FAILS (*Editor*)
The string specified in the Find (F) command could not be found in the edit buffer beginning with the line specified. If no line is specified the current line is used.

SYMBOL TABLE OVERFLOW (*Assembler*)
The symbol table is extending past USRORG into the protected area of user memory. Adjust USRORG using the O command. See *Chapter 7*.

SYNTAX ERROR (*Assembler*)
There is a syntax error in a macro dummy argument.

UNDEFINED SYMBOL (*Assembler,ZBug*)
Your program has not defined the symbol being used.

Reference E/ Assembler Pseudo Ops

Definition of Terms

symbol
Any *string* from one to six characters long, typed in the symbol field.

expression
Any *expression* typed in the operand field. See *Reference C*, "ZBUG commands," for a definition of valid expressions.

COMMANDS	PAGES DISCUSSED

COND *expression*
Assembles the instructions between COND and ENDC only if *expression* is true (a non-zero value).

```
                COND        SYMBOL
SYMBOL          FCB         10
VALUE           FCB          5
                COND        SYMBOL-VALUE
```

Valid operators for a conditional expression are +, –, /, *. If the expression equals zero, it is false; if non-zero, it is true.

END *expression*
Ends the assembly. The optional *expression* specifies the start address of the program.

ENDC
Ends a conditional assembly.

ENDM
Ends a macro definition.

symbol* EQU *expression
Equates *symbol* to an *expression*.

```
SYMBOL          EQU         $5000
```

symbol* FCB *expression, . . .
Stores a 1-byte *expression* beginning at the current address.

```
DATA2       FCB         $33+COUNT
```

symbol* FCC *delimiter string delimiter
Stores *string* in memory beginning with the current address. The *delimiter* can be any character.

```
TABLE       FCC         /THIS IS A STRING/
```

symbol* FDB *expression
Stores a 2-byte expression in memory begining at the current address.

```
DATA        FDB         $3322
```

INCLUDE *source filespec*
Includes *source filespec* in the current position of the source program.

```
INCLUDE     SAMPLE/ASM
```

***symbol* MACRO**
Defines the instructions between MACRO and ENDM as a macro named *symbol.*

```
DIVIDE      MACRO
```

OPT *switch, . . .*
Uses *switch* to control the listing of macros when assembling the program. The switches are:

MC	List macro calls (default)
NOMC	Do not list macro calls
MD	List macro definitions (default)
NOMD	Do not list macro definitions
MEX	List macro expansionns
NOMEX	Do not list macro expansions (default)
L	Turn on the listing (default)
NOL	Turn off the listing

ORG *expression*
Originates the program at *expression* address.

```
ORG         $3F00
```

PAGE
Ejects the assembly listing to the next page.

RMB *expression*
Reserves *expression* bytes of memory for data.

```
DATA        RMB         $06
```

symbol* SET *expression
Sets or resets *symbol* to *expression.*

```
SYMBOL      SET         $3500
```

SETDP *expression*
Sets the direct page to *expression.*

```
SETDP       $20
```

TITLE *string*
Prints *string* as the title of each page of the assembly listing. *String* can be up to 32 characters.

```
TITLE       Program 1
```

Reference F/ Rom Routines

This reference lists the indirect addresses where the Color Computer's ROM routines are stored. It also shows the entry and exit conditions for each routine.

The name of the routine is for documentation only. To jump to the routine, you must use its indirect address (the address contained in the brackets).

COMMANDS	PAGES DISCUSSED

BLKIN = [$A006]
Reads a block from a cassette.
Entry Conditions:
Cassette must be on and in bit sync (see CSRDON).
CBUFAD contains the buffer address.
Exit Conditions:
BLKTYP, located at $7C, contains the block type:
0 = file header
1 = data
FF = end of file
BLKLEN, located at $7D, contains the number of data bytes in the block (0-255):
Bit Z in the Register CC, Register A, and CSRERR, located at Address $81, contains the error:
Z = 1, A = CSRERR = 0 (if no errors)
Z = 0, A = CSRERR = 1 (if a checksum error occurs)
Z = 0, A = CSRERR = 2 (if a memory error occurs)

BLKOUT = [$A008]
Writes a block to cassette.
Entry Conditions:
If this is the first block write after turning the motor on, the tape should be up to speed and a $55s should be written first.
CBUFAD, located at $7E, contains the buffer address.
BLKTYP, located at $7C, contains the block type.
BLKLEN, located at $7D, contains the number of bytes.
Exit Conditions:
Interrupts are masked.
X = CBUFAD + BLKLEN.
All registers are modified.

CHROUT = [A002]

Outputs a character to a device.

Entry Conditions:

Register A = character to be output
Address 6F (DEVNUM) = the device (–2 = printer; 0 = screen)

Exit Conditions:

Register CC is changed; all others are preserved.

CSRDON = [$A004]

Starts the cassette and gets into bit sync for reading.

Entry Conditions:

None

Exit Conditions:

FIRQ and IRQ are masked.
Registers U and Y are preserved. All others are modified.

JOYIN = [$A00A]

Samples the four joystick pots and stores their values in POTVAL through POTVAL+3.

Left Joystick:	
Up/Down	15A
Right/Left	15B
Right Joystick:	
Up/Down	15C
Right/Left	15D

For Up/Down, the minimum value equals Up.
For Right/Left, the minimum value equals Left.

POLCAT = [A000]

Polls the keyboard for a character.

Entry Conditions:

None

Exit Conditions:

If no key is seen — Flag Z = 1, Register A = 0
If a key is seen — Flag Z = 0, Register A = key code
Registers B and X are preserved.
All other registers are modified.

Reference G/
DOS Disk Data Control Block (DCB)

DOS uses a 49-byte DCB to access a disk file. This reference shows the contents of each of the bytes (Bytes 0-48) in the DCB.

Bytes 0-31

The first 32 bytes of the DCB correspond to the disk file's 32-byte directory entry. When creating a file, DOS writes the DCB's first 32 bytes to the directory.

When opening an existing file, DOS searches each directory entry for the filename and extension you have set in the DCB. If it finds a match, it overwrites the first 32 bytes of the DCB with the 32-byte directory entry.

When you close the file, DOS overwrites the directory entry with the first 32 bytes of the DCB.

Filename (DCBFNM) **Bytes 0-7**
Contains the name of the file you want to access. You must set this value.

Extension (DCBFNM) **Bytes 8-10**
Contains the extension of the file you want to access. You must set this value.

File Type (DCBFTY) **Byte 11**
Contains the type of file you want to access. DOS ignores this, but BASIC uses it. You need to set this value when creating the file if you want the file compatible with BASIC.

ASCII Flag (DCBASC) **Byte 12**
Contains a flag if the file is in ASCII format. DOS ignores this, but BASIC uses it. You need to set this value when creating the file if you want the file compatible with BASIC.

First Cluster (DCBFCL) **Byte 13**
Contains the number of the first cluster in the file. (When you first create a file, this contains $FF.) DOS sets this value. Do not change it.

First Sector Bytes (DCBNLS) **Bytes 14-15**
Contains the number of bytes used in the first sector of the file. DOS ignores this. However, to be compatible with BASIC files, you should set this value before closing an output file.

File Mode (DCBCFS) **Byte 16**
Contains the mode you specified with Register A in the OPEN, WRITE, or READ routine. DOS sets this value.

Record Size (DCBRSZ) **Bytes 17-18**
Contains the size of each record. Use this with fixed-length records only. You set this value before reading from or writing to a direct access file.

Record Terminator (DCBTRM) **Byte 19**
Contains the character that DOS uses to terminate each record. You supply this value when reading from or writing to a sequential access file.

Undefined (DCBUSR) **Bytes 20-31**
Contains nothing at present. In future releases, DOS may use part of this.

Bytes 32 – 48

Bytes 32-48 are primarily set by DOS. However, you may use the contents of these bytes as data in your program.

The exceptions to this are the bytes for the drive number, physical buffer address, and logical buffer address. You must set the contents of these bytes before opening a file.

Operation Code (DCBOPC) **Byte 32**
Contains the last physical I/O operation performed on the file. See your Disk System Manual for details. DOS sets this value.

Drive Number (DCBDRV) **Byte 33**
Contains the drive number (0-3 or $FF). $FF tells DOS to use the first available drive and then insert the drive number in this segment. You must set this value before opening a file.

Track Number (DCBTRK) **Byte 34**
Contains the number of the last track DOS accessed while doing I/O for this file. DOS sets this value.

Sector Number (DCBSEC) **Byte 35**
Contains the number of the last sector DOS accessed while doing I/O for this file. DOS sets this value.

Physical Buffer Address (DCBBUF) **Bytes 36-37**
Contains the start address of a 256-byte physical buffer. The physical buffer is for storing data before or after disk I/O. You must set this value before opening a file.

Error Code (DCBOK) **Byte 38**
Contains the same value that the DOS routine returns in Register A: a zero if the last DOS routine was successful; the error number if there was an error. DOS sets this value.

Logical Buffer Address (DCBLRN) **Bytes 39-40**
Contains the start address of a logical buffer. The logical buffer is for storing a logical record before or after it goes through the physical buffer. You must set this value before opening a file, unless you have specified the "share" file mode. (See OPEN.)

Physical Record Number (DCBPRN). **Bytes 41-42**
Contains the number of the physical record currently in the physical buffer. DOS uses this to determine whether another physical read or write is required. This contains $FFFF when the file is opened. It also contains $FFFF after every read or write when the buffer is "shared." DOS sets this value.

Relative Byte Address (DCBRBA) **Bytes 43-45**
Contains an address which points to the record you want to read or write (zero when the file is first opened). With sequential access, this address always points to the next record. With direct access, this address is the product of DCBRSZ times DCBPRN. DOS sets and updates this value.

Logical Record Number (DCBLRN). **Bytes 46-47**
Contains the number of the next record to be accessed (zero when the file is first opened). Unless you set this value, DOS increments it after accessing each record.

Modified Data Tag (DCBMDT) **Byte 48**
Contains a tag ("1") if the contents of the physical buffer need to be written to disk. DOS sets this tag each time it writes to the logical buffer. The contents of the physical buffer are written to disk only when DOS must access a different sector (because the 256-byte buffer is full) or close the file. If the physical buffer is "shared," the physical buffer is written to disk after each logical write. DOS sets and updates this value.

Reference H/ DOS Routines

This reference lists all the DOS routines that Radio Shack will continue to provide in future releases. Please note that Radio Shack will support only the OPEN, CLOSE, READ, and WRITE routines. The other routines listed in this reference will be provided, but not necessarily supported.

Definition of Terms

root program
The portion of the program that is not an overlay. If you are not using overlays, this is the entire program.

overlay
A portion of the program that DOS loads into memory only when called. This can be your own overlay (called with DOUSR, GOUSR, or LOUSR) or a DOS overlay (called with DO, GO, or LOAD).

DOS programming convention
A convention, which any program using DOS routines must follow:

- The execution address must be the first instruction in the program.
- The first three bytes of the program must contain a JMP or LBR to any part of the root program. (JMP and LBR are both 3-byte instructions.) Example:

```
START       JMP         BEGIN
```

- The next two bytes must contain the length of the root program. If you are not using overlays, this is the entire program. Example:

```
            FDB         DONE-START
```

- If you are using overlays, this is the root program. Example:

```
            FDB         DONE-OVY1
```

DOS overlay conventions
A convention, which any of your own overlays must follow:

- The first two bytes must contain the size of the overlay. Example:

```
OVY1        FDB         OVY2-OVY1
```

- The next three bytes must contain a JMP or LBRA to any part of the overlay. Example:

```
            JSR         PROV1
```

- The last instruction should be an RTS, GO, or GOUSR.
- You must assign the overlay a number that is sequential. For example, assign your first overlay the overlay number of 1:

```
OVY         EQU         1
```

- The overlay must be written with relocatable (rather than absolute) addresses. When DOS loads the overlay, it sets Register X equal to the overlay's base address. Therefore, you can refer to all the local variables as an offset to Register X.

COMMANDS	PAGES DISCUSSED

CLOSE =[$602]

Closes access to a disk file.

Entry Conditions:

Register U = the address of the DCB that was previously opened.
Program must follow DOS programming convention.

Exit Conditions

Register A = status code

Technical Function of CLOSE:

- Checks the drive specified by DCBDRV for a directory entry matching DCBFNM and DCBFEX. When the entry is found, checks to see if the file was previously open by seeing if DCBCFS contains a non-zero value.
- Checks DCBMDT for a modification tag. If found, writes the contents of the physical buffer to the disk.
- Sets DCBCFS to zero.
- Rewrites the directory entry with the first 32 bytes of the DCB. Any changes in the first 32 bytes of the DCB after OPEN and before CLOSE are recorded in the directory.
- Rewrites the diskette's FAT.

DO = [$60A]

Calls a DOS overlay.

Entry Conditions:

Register A = DOS overlay number

Exit Conditions:

Register A = status code

DOUSR = [$0610]

Calls one of your own overlays.

Entry Conditions:

Register A = overlay number (the number you have assigned to the overlay)

Exit Conditions:

Register A = status code

GO = [$60C]

Calls one DOS overlay from another DOS overlay.

Entry Conditions:

Register A = DOS overlay number

Exit Conditions:

Register A = status code

GOUSR = [$612]

Calls one overlay from another overlay. For example, OVY1 calls OVY2.

Entry Conditions:

Register A = overlay number (the number you have assigned to the overlay)

Exit Conditions:

Register A = "0" if no error; error code if error

LOAD = [$60E]

Loads a DOS overlay but does not execute it.

Entry Conditions:

Register A = DOS overlay number

Exit Conditions:

Register A = "0" if no error; error code if error

LODUSR = [$614]

Loads one of your overlays but does not execute it.

Entry Conditions:

Register A = overlay number (the number you have assigned to the overlay)

Exit Conditions:

Register A = "0" if no error; error code if error

OPEN = [$600]

Opens access to a disk file using the specified file mode.

Entry Conditions:

Register A = file mode

The file modes are:

Bit 0 set — allows reads
Bit 1 set — allows writes
Bit 2 set — allows file creation
Bit 3 set — allows extension past end of file
Bit 4 set — deletes the file when closed (work file)
Bit 5 set — rewrites the directory's file allocation table (FAT) only when the file is closed. (Otherwise, rewrites FAT after each READ; see the *Disk System Manual* for information on the FAT.)
Bit 6 set — shares the physical and logical buffer
Bit 7 set — undefined

Register U = the address where the DCB is stored.

The DCB must contain values for DCBFNM, DCBFEX, DCBDRV, and DCBBUF

Program must follow DOS programming conventions.

Exit Conditions:

Register A = 0 if no error; error code if error

Technical Function of OPEN:

- Checks the drive specified by DCBDRV for a directory entry matching DCBFNM and DCBFEX.
- If a match is found:
  - Uses the directory entry to overwrite the first 32 bytes of the DCB
  - Checks DCBCFS. It indicates a write, create, or extend, the file is opened and Status Code L is returned.
  - Inserts the file mode (contained in Register A) in DCBCFS.
  - Overwrites the directory entry with the first 32 bytes of the DCB.
- If a match is not found and the file mode is "create," creates a directory entry using the first 32 bytes of the DCB

- Sets DCBPRN to $FFFF
- Clears DCBLRN, DCBMDT, and DCBRBA.

READ = [$604]

Reads a record from a disk file.

Entry Conditions

Register A = read option

The read options are:

Bit 0 clear — direct access (read by record number; fixed length records)

Bit 0 set — sequential access (read by terminator character; variable length records)

Bit 1 clear — exit READ pointing to next record

Bit 1 set — exit READ leaving DCBLRN and DCBRBA the same (not pointing to next record)

The other bits can contain any value.

Register U = address pointing to the DCB

Program must follow DOS programming convention

Exit Conditions:

Register A = 0 if no error; error number if error logical buffer (pointed to by DCBLRB) contains the record

Technical Function of READ:

- Checks DCBCFS to see if the file was opened for "read."
- Checks DCBRBA for the record you want to access. (If Bit 0 in Register A is clear, READ calculates DCBRBA as the product of DCBLRN times DCBRSZ).
- Checks to see if the record is in the physical buffer (by comparing the high two bytes of DCBRBA with the contents of DCBPRN).
  If the record is not in the physical buffer, READ reads the record into the physical buffer then transfers it to the logical buffer.
- Checks to see if Register A's Bit 1 is set. If so, restore DCBLRN and DCBRBA to their original values.

RELSE = [$608]

Frees a physical buffer so that you can use it with another file.

Entry Conditions:

Register U = address where the DCB is stored of the file currently using the physical buffer.

Register A = 0 if no error; error code if error.

Technical Function of RELSE:

- Check DCBMDT. If the tag is set, the contents of the physical buffer are written to disk and DCBMDT is cleared.
- Sets DCBPRN to $FFFF.

WRITE = [$606]

Writes a logical record to disk.

Entry Conditions:

Register A = read/write option

The read/write options are:

Bit 0 clear — direct access (write by record number; fixed length records)

Bit 0 set — sequential access (write by terminator character; variable length records)

Bit 1 clear — exit READ pointing to next record
Bit 1 set — exit READ leaving DCBLRN and DCBRBA the same (not pointing to next record)
The other bits can contain any value.

Register U = address pointing to the DCB logical buffer (pointed to by DCBLRB) contains the record you want to write

Program must follow DOS programming conventions.

Exit Conditions:

Register A = 0 if no error; status code if error

Technical Function of WRITE:

- Checks DCBCFS to see if the file was opened for "write."
- Checks DCBRBA for the record you want to access. (If Bit 0 in Register A is off, WRITE calculates DCBRBA as the product of DCBLRN times DCBRSZ).
- Transfers the contents of the logical buffer to the physical buffer. If all 256 bytes of the physical buffer are full, writes the contents of the physical buffer to disk. If there is still more contents in the logical buffer, WRITE transfer these contents to the physical buffer and sets DCBMDT to 1.
- If the file mode is "share," writes the complete contents of the physical buffer to disk regardless of whether it completely fills the sector. Then, sets DCBPRN to $FFFF.

Reference I/ DOS Error Codes

Error Code	Hex Code	Character Displayed	Error
00	40	@	No errors
01	41	A	I/O error (drive not ready)
02	42	B	I/O error (write-protected diskette)
03	43	C	I/O error (write fault)
04	44	D	I/O error (seek error or record not found)
05	45	E	I/O error (CER error)
06	46	F	I/O error (lost data)
07	47	G	I/O error (undefined Bit 1)
08	48	H	I/O error (undefined Bit 0)
09	49	I	Register argument is invalid
0A	4A	J	File directory entry not found
0B	4B	K	Full directory
0C	4C	L	File was created by the OPEN function
0D	4D	M	File not closed after changes
0E	4E	N	Attempt to access an opened file
0F	4F	O	Attempt to read a read-protected file
10	50	P	RBA overflow (exceeds 3 bytes -16,777,216)
11	51	Q	Access beyond EOF or extension not allowed
12	52	R	FAT rewrite error
13	53	S	Attempt to close an unopened file
14	54	T	Can't access directly (record size is 0)
15	55	U	Attempt to write on write-protected diskette
16	56	V	Can't extend file (disk capacity exceeded)
17	57	W	Error while loading overlay
18	58	X	Insufficient print space allocated
19	59	Y	I/O error during BASIC line read
1A	5A	Z	Program's load address is too low
1B	5B	[	First byte of program file is not equal to zero
1C	5C	\	Not enough space for buffered keyboard
1D	5D	]	Not enough memory
1E	5E	^	Output file already exists
1F	5F	–	Wrong diskette

Reference J/
Memory Map

$0 - $69	Direct page RAM
$70-$FF	System direct page RAM
$100-$111	Interrupt vectors
$112-$119	System RAM
$11A	Keyboard alpha lock flag
$11B-$159	System RAM
$15A-$15D	Joystick pot values
$15E-$3FF	System RAM
$400-$5FF	Video memory
$600-$11FF	DOS
$1200-$3FFF	16K user memory
$1200-$7FFF	32K user memory
$8000-$9FFF	Extended BASIC
$A000-$BFFF	BASIC
$C000-$DFFF	Disk BASIC
$E000-$FEFF	ROM expansion
$FF00-$FFEE	Hardware address
$FFF0-$FFFF	Interrupt vectors

Reference K/ ASCII Codes

Video Control Codes

Dec	Hex	PRINT CHR$ *(code)*
8	08	Backspaces and erases current character.
13	0D	Line feed with carriage return.
32	20	Space

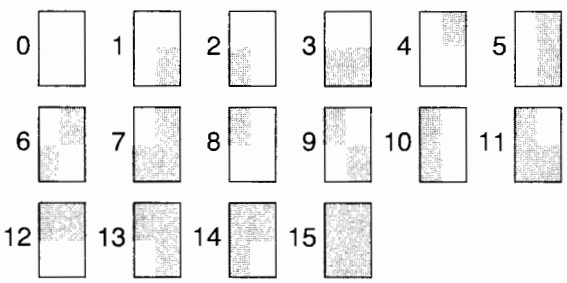
Color Codes

CODE	COLOR
0	Black
1	Green
2	Yellow
3	Blue
4	Red
5	Buff
6	Cyan
7	Magenta
8	Orange

Graphic Character Codes

Given the *color* (1-8) and the *pattern* (0-15), this formula will generate the correct code:

code = 128 + 16 * (*color* − 1) + *pattern*



For example, to print *pattern* 9 in blue (*code* 3), type:

```
C = 128 + 16 * (3-1) + 9
? CHR$ (C)
```

Alphanumeric Character Codes

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
SPACEBAR	32	20
!	33	21
"	34	22
#	35	23
$	36	24
%	37	25
&	38	26
'	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F
@	64	40
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D
N	78	4E
O	79	4F
P	80	50
Q	81	51
R	82	52
S	83	53

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A
↑ *	94	5E
↓ *	10	0A
← *	8	08
→ *	9	09
BREAK	03	03
CLEAR	12	0C
ENTER	13	0D

*If shifted, the code for these characters are as follows: CLEAR is 92 (hex 5C); ↑ is 95 (hex 5F); ↓ is 91 (hex 5B); ← is 21 (hex 15); and → is 93 (hex 5D).

These are the ASCII codes for lowercase letters. You can produce these characters by pressing SHIFT 0 simultaneously to get into an upper-lowercase mode. The lowercase letters will appear on your screen in reversed colors (green with a black background).

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
a	97	61
b	98	62
c	99	63
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A

Reference L/ 6809 Mnemonics

Definition of Terms

Source Forms:

This shows all the possible variations you can use with the instruction. *Table 4* gives the meaning of all the notations we use. The notations in italics represent values you can supply.

For example, the BEQ instruction has two source forms. BEQ *dd* allows you to use these instructions:

```
BEQ $08     BEQ $FF     BEQ $A0
```

Whereas LBEQ DDDD allows you these:

```
LBEQ $C000     LBEQ $FFFF
```

Operation:

This uses shorthand notation to show exactly what the instruction does, step by step. The meaning of all the codes are also in *Table 4.*

For example, the BEQ operation does this:

> *"If, (but only if), the zero flag is set, branch to the location indicated by the program counter plus the value of the 8-bit offset."*

Condition Codes:

This shows which of the flags in the CC register are affected by the instruction, if any. As you'll note, BEQ does not set or clear any of the flags.

Description:

This is an overall description, in English, of what the instruction does.

Addressing Mode:

This tells you which addressing modes you may use with the instruction. BEQ allows only the Relative addressing mode.

ABBREVIATION	MEANING
ACCA or A	Accumulator A.
ACCB or B	Accumulator B.
ACCA:ACCB or D	Accumulator D.
ACCX	Either accumulator A or accumulator B.
CCR or CC	Condition code register.
DPR or DP	Direct page register.
EA	Effective address.
IFF	If and only if.
IX or X	Index register X.
IY or Y	Index register Y.
LSN	Least significant nibble.
M	Memory location.
MI	Memory immediate.
MSN	Most significant nibble.
PC	Program counter.
R	A register before the operation.
R'	A register after the operation.
TEMP	A temporary storage location.
xxH	Most significant byte of any location.
xxL	Least significant byte of any location.
Sp or S	Hardware stack pointer.

ABBREVIATION	MEANING
Us or U	User stack pointer.
P	A memory location with immediate, direct, extended, and indexed addressing modes.
Q	A read-write-modify argument with direct, extended and indexed addressing modes.
( )	The data pointed to by the enclosed (16 bit address).
dd	8-bit branch offset.
DDDD	16-bit offset.
#	Immediate value follows.
$	Hexadecimal value follows.
[ ]	Indirection.
,	Indicates indexed addressing.
←	Is transferred to.
/	Boolean AND.
V	Boolean OR.
O	Boolean Exclusive OR (XOR).
—	Boolean NOT.
:	Concatination.
+	Arithmetic plus.
−	Arithmetic minus.
×	Arithmetic multiply.

Table 4. Notations and Codes

Add Accumulator B into Index Register X

ABX

Source Form: ABX
Operation: IX'←IX + ACCB

Condition Codes: Not affected.
Description: Add the 8-bit unsigned value in accumulator B into index register X.
Addressing Mode: Inherent.

Add with Carry into Register

ADC

Source Forms: ADCA *P*; ADCB *P*
Operation: R'←R + M + C
Condition Codes:
H — Set if a half-carry is generated; cleared otherwise.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.
Description: Adds the contents of the C (carry) bit and the memory byte into an 8-bit accumulator.
Addressing Modes: Immediate; Extended; Direct; Indexed.

Add Memory into Register

ADD (8-Bit)

Source Forms: ADDA *P*; ADDB *P*
Operation: R'←R + M
Condition Codes:
H — Set if a half-carry is generated; cleared otherwise.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.
Description: Adds the memory byte into an 8-bit accumulator.
Addressing Modes: Immediate; Extended; Direct; Indexed.

Add Memory into Register

ADD (16-Bit)

Source Form: ADDD *P*
Operation: R'←R + M:M + 1
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.
Description: Adds the 16-bit memory value into the 16-bit accumulator.
Addressing Modes: Immediate; Extended; Direct; Indexed.

Logical AND Memory into Register

AND

Source Forms: ANDA *P*; ANDB *P*
Operation: R'←R ∧ M
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.
Description: Performs the logical AND operation between the contents of an accumulator and the contents of memory location M and the result is stored in the accumulator.
Addressing Modes: Immediate; Extended; Direct; Indexed.

Logical AND Immediate Memory into Condition Code Register

AND

Source Form: ANDCC *#xx*
Operation: R'←R ∧ MI
Condition Codes: Affected according to the operation.

Description: Performs a logical AND between the condition code register and the immediate byte specified in the instruction and places the result in the condition code register.
Addressing Mode: Immediate.

Arithmetic Shift Left

ASL

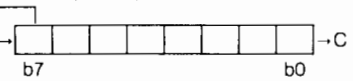
Source Forms: ASL *Q*; ASLA; ASLB
Operation: C←[b7 ← b0]←0
Condition Codes:
H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Loaded with the result of the exclusive OR of bits six and seven of the original operand.
C — Loaded with bit seven of the original operand.
Description: Shifts all bits of the operand one place to the left. Bit zero is loaded with a zero. Bit seven is shifted into the C (carry) bit.
Addressing Modes: Inherent; Extended; Direct; Indexed.

ASR

Arithmetic Shift Right

Source Forms: ASR *Q*; ASRA; ASRB

Operation:



Condition Codes:
H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Not affected.
C — Loaded with bit zero of the original operand.

Description: Shifts all bits of the operand one place to the right. Bit seven is held constant. Bit zero is shifted into the C (carry) bit.
Addressing Modes: Inherent; Extended; Direct; Indexed.

BCC

Branch on Carry Clear

Source Forms: BCC *dd*; LBCC *DDDD*
Operation:
TEMP←MI
IFF C = 0 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Tests the state of the C (carry) bit and causes a branch if it is clear.
Addressing Mode: Relative.
Comments: Equivalent to BHS dd; LBHS DDDD.

BCS

Branch on Carry Set

Source Forms: BCS *dd*; LBCS *DDDD*
Operation:
TEMP←MI
IFF C = 1 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Tests the state of the C (carry) bit and causes a branch if it is set.
Addressing Mode: Relative.
Comments: Equivalent to BLO dd; LBLO DDDD.

BEQ

Branch on Equal

Source Forms: BEQ *dd*; LBEQ *DDDD*
Operation:
TEMP←MI
IFF Z = 1 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Tests the state of the Z (zero) bit and causes a branch if it is set. When used after a subtract or compare operation, this instruction will branch if the compared values, signed or unsigned, were exactly the same.
Addressing Mode: Relative.

BGE

Branch on Greater than or Equal to Zero

Source Forms: BGE *dd*; LBGE *DDDD*
Operation:
TEMP←MI
IFF [N ⊕ V] = 0 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Causes a branch if the N (negative) bit and V (overflow) bit are either both set or both clear. That is, branch if the sign of a valid twos complement result is, or would be, positive. When used after a subtract or compare operation on twos complement values, this instruction will branch if the register was greater than or equal to the memory operand.
Addressing Mode: Relative.

BGT

Branch on Greater

Source Forms: BGT *dd*; LBGT *DDDD*
Operation:
TEMP←MI
IFF Z ∧ [N ⊕ V] = 0 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Causes a branch if the N (negative) bit and V (overflow) bit are either both set or both clear and the Z (zero) bit is clear. In other words, branch if the sign of a valid twos complement result is, or would be, positive and not zero. When used after a subtract or compare operation on twos complement values, this instruction will branch if the register was greater than the memory operand.
Addressing Mode: Relative.

BHI

Branch if Higher

Source Forms: BHI *dd*; LBHI *DDDD*
Operation:
TEMP←MI
IFF [C v Z] = 0 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Causes a branch if the previous operation caused neither a carry nor a zero result. When used after a subtract or compare operation on unsigned binary values, this instruction will branch if the register was higher than the memory operand.
Addressing Mode: Relative.
Comments: Generally not useful after INC/DEC, LD/TST, and TST/CLR/COM instructions.

Branch if Higher or Same

BHS

Source Forms: BHS *dd*; LBHS *DDDD*
Operation:
TEMP←MI
IFF C = 0 then PC'←PC + MI
Condition Codes: Not affected.
Description: Tests the state of the C (carry) bit and causes a branch if it is clear. When used after a subtract or compare on unsigned binary values, this instruction will branch if the register was higher than or the same as the memory operand.
Addressing Mode: Relative.
Comments: This is a duplicate assembly-language mnemonic for the single machine instruction BCC. Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.

Bit Test

BIT

Source Form: BIT *P*
Operation: TEMP←R ∧ M
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.
Description: Performs the logical AND of the contents of accumulator A or B and the contents of memory location M and modifies the condition codes accordingly. The contents of accumulator A or B and memory location M are not affected.
Addressing Modes: Immediate; Extended; Direct; Indexed.

Branch on Less than or Equal to Zero

BLE

Source Forms: BLE *dd*; LBLE *DDDD*
Operation:
TEMP←MI
IFF Z v [N ⊕ V] = 1 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Causes a branch if the exclusive OR of the N (negative) and V (overflow) bits is 1 or if the Z (zero) bit is set. That is, branch if the sign of a valid twos complement result is, or would be, negative. When used after a subtract or compare operation on twos complement values, this instruction will branch if the register was less than or equal to the memory operand.
Addressing Mode: Relative.

Branch on Lower

BLO

Source Forms: BLO *dd*; LBLO *DDDD*
Operation:
TEMP←MI
IFF C = 1 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Tests the state of the C (carry) bit and causes a branch if it is set. When used after a subtract or compare on unsigned binary values, this instruction will branch if the register was lower than the memory operand.
Addressing Mode: Relative.
Comments: This is a duplicate assembly-language mnemonic for the single machine instruction BCS. Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.

Branch on Lower or Same

BLS

Source Forms: BLS *dd*; LBLS *DDDD*
Operation:
TEMP←MI
IFF (C v Z) = 1 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Causes a branch if the previous operation caused either a carry or a zero result. When used after a subtract or compare operation on unsigned binary values, this instruction will branch if the register was lower than or the same as the memory operand.
Addressing Mode: Relative.
Comments: Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.

Branch on Less than Zero

BLT

Source Forms: BLT *dd*; LBLT *DDDD*
Operation:
TEMP←MI
IFF [N ⊕ V] = 1 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Causes a branch if either, but not both, of the N (negative) or V (overflow) bits is set. That is, branch if the sign of a valid twos complement result is, or would be, negative. When used after a subtract or compare operation on twos complement binary values, this instruction will branch if the register was less than the memory operand.
Addressing Mode: Relative.

Branch on Minus

BMI

Source Forms: BMI *dd*; LBMI *DDDD*
Operation:
TEMP←MI
IFF N = 1 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Tests the state of the N (negative) bit and causes a branch if set. That is, branch if the sign of the twos complement result is negative.
Addressing Mode: Relative.
Comments: When used after an operation on signed binary values, this instruction will branch if the result is minus. It is generally preferred to use the LBLT instruction after signed operations.

BNE

Branch Not Equal

Source Forms: BNE *dd*; LBNE *DDDD*
Operation:
TEMP←MI
IFF Z = 0 then PC'←PC + TEMP
Condition Codes: Not affected.

Description: Tests the state of the Z (zero) bit and causes a branch if it is clear. When used after a subtract or compare operation on any binary values, this instruction will branch if the register is, or would be, not equal to the memory operand.
Addressing Mode: Relative.

BPL

Branch on Plus

Source Forms: BPL *dd*; LBPL *DDDD*
Operation:
TEMP←MI
IFF N = 0 then PC'←PC + TEMP
Condition Codes: Not affected.
Description: Tests the state of the N (negative) bit and causes a branch if it is clear. That is, branch if the sign of the twos complement result is positive.
Addressing Mode: Relative.
Comments: When used after an operation on signed binary values, this instruction will branch if the result (possibly invalid) is positive. It is generally preferred to use the BGE instruction after signed operations.

BRA

Branch Always

Source Forms: BRA *dd*; LBRA *DDDD*
Operation:
TEMP←MI
PC'←PC + TEMP

Condition Codes: Not affected.
Description: Causes an unconditional branch.
Addressing Mode: Relative.

BRN

Branch Never

Source Forms: BRN *dd*; LBRN *DDDD*
Operation: TEMP←MI
Condition Codes: Not affected.

Description: Does not cause a branch. This instruction is essentially a no operation, but has a bit pattern logically related to branch always.
Addressing Mode: Relative.

BSR

Branch to Subroutine

Source Forms: BSR *dd*; LBSR *DDDD*
Operation:
TEMP←MI
SP'←SP − 1, (SP)←PCL
SP'←SP − 1, (SP)←PCH
PC'←PC + TEMP

Condition Codes: Not affected.
Description: The program counter is pushed onto the stack. The program counter is then loaded with the sum of the program counter and the offset.
Addressing Mode: Relative.
Comments: A return from subroutine (RTS) instruction is used to reverse this process and must be the last instruction executed in a subroutine.

BVC

Branch on Overflow Clear

Source Forms: BVC *dd*; LBVC *DDDD*
Operation:
TEMP←MI
IFF V = 0 then PC'←PC + TEMP
Condition Codes: Not affected.

Description: Tests the state of the V (overflow) bit and causes a branch if it is clear. That is, branch if the twos complement result was valid. When used after an operation on twos complement binary values, this instruction will branch if there was no overflow.
Addressing Mode: Relative.

BVS

BVS Branch on Overflow set

Source Forms: BVS dd; LBVS DDDD
Operation: Temp←MI IFF V = 1 then PC'←PC + TEMP
Condition Codes: Not affected.

Description: Tests the state of V (overflow) bit and causes a branch if it is set. That is, branch if twos complement result was invalid. When used after an operation on twos complement binary values, this instruction will branch if there was an overflow.
Addressing Mode: Relative.

CLR

CLR Clear

Source Forms: CLR Q
Operation: TEMP←M M←00 (base 16)
Condition codes:
H — Not affected.
N — Always cleared.
Z — Always set.
V — Always cleared.
C — Always cleared.
Description: Accumulator A or B or memory location M is loaded with 00000000. Note that the EA is read during this operation.
Addressing Modes: Inherent, Extended, Direct, Indexed.

Compare Memory from Register

CMP (8-Bit)

Source Forms: CMPA *P*; CMPB *P*
Operation: TEMP←R − M
Condition Codes:
H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.
Description: Compares the contents of memory location to the contents of the specified register and sets the appropriate condition codes. Neither memory location M nor the specified register is modified. The carry flag represents a borrow and is set to the inverse of the resulting binary carry.
Addressing Modes: Immediate; Extended; Direct; Indexed.

Compare Memory from Register

CMP (16-Bit)

Source Forms: CMPD *P*; CMPX *P*; CMPY *P*; CMPU *P*; CMPS *P*
Operation: TEMP←R − M:M + 1
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.
Description: Compares the 16-bit contents of the concatenated memory locations M:M + 1 to the contents of the specified register and sets the appropriate condition codes. Neither the memory locations nor the specified register is modified unless autoincrement or autodecrement are used. The carry flag represents a borrow and is set to the inverse of the resulting binary carry.
Addressing Modes: Immediate; Extended; Direct; Indexed.

Complement

COM

Source Forms: COM *Q*; COMA; COMB
Operation: M′←O + $\overline{M}$
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Always set.
Description: Replaces the contents of memory location M or accumulator A or B with its logical complement. When operating on unsigned values, only BEQ and BNE branches can be expected to behave properly following a COM instruction. When operating on twos complement values, all signed branches are available.
Addressing Modes: Inherent; Extended; Direct; Indexed.

Clear CC bits and Wait for Interrupt

CWAI

Source Form: CWAI #$*XX*

E	F	H	I	N	Z	V	C

Operation:
CCR←CCR .\ MI (Possibly clear masks)
Set E (entire state saved)
SP′←SP − 1, (SP)←PCL
SP′←SP − 1, (SP)←PCH
SP′←SP − 1, (SP)←USL
SP′←SP − 1, (SP)←USH
SP′←SP − 1, (SP)←IYL
SP′←SP − 1, (SP)←IYH
SP′←SP − 1, (SP)←IXL
SP′←SP − 1, (SP)←IXH
SP′←SP − 1, (SP)←DPR
SP′←SP − 1, (SP)←ACCB
SP′←SP − 1, (SP)←ACCA
SP′←SP − 1, (SP)←CCR
Condition Codes: Affected according to the operation.
Description: This instruction ANDs an immediate byte with the condition code register which may clear the interrupt mask bits I and F, stacks the entire machine state on the hardware stack and then looks for an interrupt. When a non-masked interrupt occurs, no further machine state information need be saved before vectoring to the interrupt handling routine. This instruction replaced the MC6800 CLI WAI sequence, but does not place the buses in a high-impedance state. A $\overline{FIRQ}$ (fast interrupt request) may enter its interrupt handler with its entire machine state saved. The RTI (return from interrupt) instruction will automatically return the entire machine state after testing the E (entire) bit of the recovered condition code register.
Addressing Mode: Immediate.
Comments: The following immediate values will have the following results:
FF = enable neither
EF = enable $\overline{IRQ}$
BF = enable $\overline{FIRQ}$
AF = enable both

Decimal Addition Adjust

DAA

Source Form: DAA
Operation: ACCA′←ACCA + CF (MSN):CF(LSN)
where CF is a Correction Factor, as follows: the CF for each nibble (BCD) digit is determined separately, and is either 6 or 0.
Least Significant Nibble
CF(LSN) = 6 IFF 1) C = 1
or 2) LSN>9
Most Significant Nibble
CF(MSN) = 6 IFF 1) C = 1
or 2) MSN>9
or 3) MSN>8 *and* LSN>9
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Undefined.
C — Set if a carry is generated or if the carry bit was set before the operation; cleared otherwise.
Description: The sequence of a single-byte add instruction on accumulator A (either ADDA or ADCA) and a following decimal addition adjust instruction results in a BCD addition with an appropriate carry bit. Both values to be added must be in proper BCD form (each nibble such that: 0≤nibble≤9). Multiple-precision addition must add the carry generated by this decimal addition adust into the next higher digit during the add operation (ADCA) immediately prior to the next decimal addition adjust.
Addressing Mode: Inherent.

DEC

Decrement

Source Forms: DEC *Q*; DECA; DECB
Operation: M′←M − 1
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the original operand was 10000000; cleared otherwise.
C — Not affected.
Description: Subtract one from the operand. The carry bit is not affected, thus allowing this instruction to be used as a loop counter in multiple-precision computations. When operating on unsigned values, only BEQ and BNE branches can be expected to behave consistently. When operating on twos complement values, all signed branches are available.
Addressing Modes: Inherent; Extended; Direct; Indexed.

EOR

Exclusive OR

Source Forms: EORA *P*; EORB *P*
Operation: R′←R ⊕ M
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.
Description: The contents of memory location M is exclusive ORed into an 8-bit register.
Addressing Modes: Immediate; Extended; Direct; Indexed.

EXG

Exchange Registers

Source Form: EXG *R1,R2*
Operation: R1↔R2
Condition Codes: Not affected (unless one of the registers is the condition code register).
Description: Exchanges data between two designated registers. Bits 3-0 of the postbyte define one register, while bits 7-4 define the other, as follows:

0000 = A:B	1000 = A
0001 = X	1001 = B
0010 = Y	1010 = CCR
0011 = US	1011 = DPR
0100 = SP	1100 = Undefined
0101 = PC	1101 = Undefined
0110 = Undefined	1110 = Undefined
0111 = Undefined	1111 = Undefined

Only like size registers may be exchanged. (8-bit with 8-bit or 16-bit with 16-bit.)
Addressing Mode: Immediate.

INC

Increment

Source Forms: INC *Q*; INCA; INCB
Operation: M′←M + 1
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the original operand was 01111111; cleared otherwise.
C — Not affected.
Description: Adds to the operand. The carry bit is not affected, thus allowing this instruction to be used as a loop counter in multiple-precision computations. When operating on unsigned values, only the BEQ and BNE branches can be expected to behave consistently. When operating on twos complement values, all signed branches are correctly available.
Addressing Modes: Inherent; Extended; Direct; Indexed.

JMP

Jump

Source Form: JMP *EA*
Operation: PC′←EA
Condition Codes: Not affected.
Description: Program control is transferred to the effective address.
Addressing Modes: Extended; Direct; Indexed.

JSR

Jump to Subroutine

Source Form: JSR *EA*
Operation:
SP′←SP − 1, (SP)←PCL
SP′←SP − 1, (SP)←PCH
PC′←EA
Condition Codes: Not affected.
Description: Program control is transferred to the effective address after storing the return address on the hardware stack. A RTS instruction should be the last executed instruction of the subroutine.
Addressing Modes: Extended; Direct; Indexed.

LD (8-Bit)

Load Register from Memory

Source Forms: LDA *P*; LDB *P*
Operation: R′←M
Condition Codes:
H — Not affected.
N — Set if the loaded data is negative; cleared otherwise.
Z — Set if the loaded data is zero; cleared otherwise.
V — Always cleared.
C — Not affected.
Description: Loads the contents of memory location M into the designated register.
Addressing Modes: Immediate; Extended; Direct; Indexed.

LD (16-Bit)

Load Register from Memory

Source Forms: LDD *P*; LDX *P*; LDY *P*; LDS *P*; LDU *P*
Operation: R'←M:M+1
Condition Codes:
H — Not affected.
N — Set if the loaded data is negative; cleared otherwise.
Z — Set if the loaded data is zero; cleared otherwise.
V — Always cleared.
C — Not affected.
Description: Load the contents of the memory location M:M+1 into the designated 16-bit register.
Addressing Modes: Immediate; Extended; Direct; Indexed.

LEA

Load Effective Address

Source Forms: LEAX, LEAY, LEAS, LEAU
Operation: R'←EA
Condition Codes:
H — Not affected.
N — Not affected.
Z — LEAX, LEAY: Set if the result is zero; cleared otherwise. LEAS, LEAU: Not affected.
V — Not affected.
C — Not affected.
Description: Calculates the effective address from the index addressing mode and places the address in an indexable register.

LEAX and LEAY affect the Z (zero) bit to allow use of these registers as counters and for MC6800 INX/DEX compatibility.

LEAU and LEAS do not affect the Z bit to allow cleaning up the stack while returning the Z bit as a parameter to a calling routine, and also for MC6800 INS/DES compatibility.
Addressing Mode: Indexed.
Comments: Due to the order in which effective addresses are calculated internally, the LEAX, X+ + and LEAX,X+ do not add 2 and 1 (respectively) to the X register; but instead leave the X register unchanged. This also applies to the Y, U, and S registers. For the expected results, use the faster instruction LEAX 2, X and LEAX 1, X.

Some examples of LEA instruction uses are given in the following table.

Instruction	Operation	Comment
LEAX 10, X	X+10→X	Adds 5-bit constant 10 to X.
LEAX 500, X	X+500→X	Adds 16-bit constant 500 to X.
LEAY A, Y	Y+A→Y	Adds 8-bit accumulator to Y.
LEAY D, Y	Y+D→Y	Adds 16-bit D accumulator to Y.
LEAU -10, U	U-10→U	Subtracts 10 from U.
LEAS -10, S	S-10→S	Used to reserve area on stack.
LEAS 10, S	S+10→S	Used to 'clean up' stack.
LEAX 5, S	S+5→X	Transfers as well as adds.

LSL

Logical Shift Left

Source Forms: LSL *Q*; LSLA; LSLB
Operation: C← [ ][ ][ ][ ][ ][ ][ ][ ] ←0
b7 b0
Condition Codes:
H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Loaded with the result of the exclusive OR of bits six and seven of the original operand.
C — Loaded with bit seven of the original operand.
Description: Shifts all bits of accumulator A or B or memory location M one place to the left. Bit zero is loaded with a zero. Bit seven of accumulator A or B or memory location M is shifted into the C (carry) bit.
Addressing Modes: Inherent; Extended; Direct; Indexed.
Comments: This is a duplicate assembly-language mnemonic for the single machine instruction ASL.

LSR

Logical Shift Right

Source Forms: LSR *Q*; LSRA; LSRB
Operation: 0→ [ ][ ][ ][ ][ ][ ][ ][ ] →C
b7 b0
Condition Codes:
H — Not affected.
N — Always cleared.
Z — Set if the result is zero; cleared otherwise.
V — Not affected.
C — Loaded with bit zero of the original operand.
Description: Performs a logical shift right on the operand. Shifts a zero into bit seven and bit zero into the C (carry) bit.
Addressing Modes: Inherent; Extended; Direct; Indexed.

MUL

Multiply

Source Form: MUL
Operation: ACCA':ACCB'←ACCA × ACCB
Condition Codes:
H — Not affected.
N — Not affected.
Z — Set if the result is zero; cleared otherwise.
V — Not affected.
C — Set if ACCB bit 7 of result is set; cleared otherwise.
Description: Multiply the unsigned binary numbers in the accumulators and place the result in both accumulators (ACCA contains the most-significant byte of the result). Unsigned multiply allows multiple-precision operations.
Addressing Mode: Inherent.
Comments: The C (carry) bit allows rounding the most-significant byte through the sequence: MUL, ADCA #0.

NEG

Negate

Source Forms: NEG *Q*; NEGA; NEGB
Operation: M′←0−M
Condition Codes:
H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the original operand was 10000000.
C — Set if a borrow is generated; cleared otherwise.
Description: Replaces the operand with its twos complement. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry. Note that 80_{16} is replaced by itself and only in this case is the V (overflow) bit set. The value 00_{16} is also replaced by itself, and only in this case is the C (carry) bit cleared.
Addressing Modes: Inherent; Extended; Direct.

NOP

No Operation

Source Form: NOP
Operation: Not affected.
Condition Codes: This instruction causes only the program counter to be incremented. No other registers or memory locations are affected.
Addressing Mode: Inherent.

OR

Inclusive OR Memory into Register

Source Forms: ORA *P*; ORB *P*
Operation: R′←R v M
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.
Description: Performs an inclusive OR operation between the contents of accumulator A or B and the contents of memory location M and the result is stored in accumulator A or B.
Addressing Modes: Immediate; Extended; Direct; Indexed.

OR

Inclusive OR Memory Immediate into Condition Code Register

Source Form: ORCC *#XX*
Operation: R′←R v MI
Condition Codes: Affected according to the operation.
Description: Performs an inclusive OR operation between the contents of the condition code registers and the immediate value, and the result is placed in the condition code register. This instruction may be used to set interrupt masks (disable interrupts) or any other bit(s).
Addressing Mode: Immediate.

PSHS

Push Registers on the Hardware Stack

Source Form:
PSHS *register list*
PSHS *#LABEL*
Postbyte:

b7	b6	b5	b4	b3	b2	b1	b0
PC	U	Y	X	DP	B	A	CC

push order →

Operation:
IFF b7 of postbyte set, then: SP′←SP−1, (SP)←PCL
SP′←SP−1, (SP)←PCH
IFF b6 of postbyte set, then: SP′←SP−1, (SP)←USL
SP′←SP−1, (SP)←USH
IFF b5 of postbyte set, then: SP′←SP−1, (SP)←IYL
SP′←SP−1, (SP)←IYH
IFF b4 of postbyte set, then: SP′←SP−1, (SP)←IXL
SP′←SP−1, (SP)←IXH
IFF b3 of postbyte set, then: SP′←SP−1, (SP)←DPR
IFF b2 of postbyte set, then: SP′←SP−1, (SP)←ACCB
IFF b1 of postbyte set, then: SP′←SP−1, (SP)←ACCA
IFF b0 of postbyte set, then: SP′←SP−1, (SP)←CCR
Condition Codes: Not affected.
Description: All, some, or none of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself).
Addressing Mode: Immediate.
Comments: A single register may be placed on the stack with the condition codes set by doing an autodecrement store onto the stack (example: STX, − −S).

PSHU

Push Registers on the User Stack

Source Form:
PSHU *register list*
PSHU *#LABEL*
Postbyte:

b7	b6	b5	b4	b3	b2	b1	b0
PC	U	Y	X	DP	B	A	CC

push order →

Operation:
IFF b7 of postbyte set, then: US′←US−1, (US)←PCL
US′←US−1, (US)←PCH
IFF b6 of postbyte set, then: US′←US−1, (US)←SPL
US′←US−1, (US)←SPH
IFF b5 of postbyte set, then: US′←US−1, (US)←IYL
US′←US−1, (US)←IYH
IFF b4 of postbyte set, then: US′←US−1, (US)←IXL
US′←US−1, (US)←IXH
IFF b3 of postbyte set, then: US′←US−1, (US)←DPR
IFF b2 of postbyte set, then: US′←US−1, (US)←ACCB
IFF b1 of postbyte set, then: US′←US−1, (US)←ACCA
IFF b0 of postbyte set, then: US′←US−1, (US)←CCR
Condition Codes: Not affected.
Description: All, some, or none of the processor registers are pushed onto the user stack (with the exception of the user stack pointer itself).
Addressing Mode: Immediate.
Comments: A single register may be placed on the stack with the condition codes set by doing an autodecrement store onto the stack (example: STX, − −U).

Pull Registers from the Hardware Stack

PULS

Source Form:
PULS *register list*
PULS *#LABEL*
Postbyte:

b7	b6	b5	b4	b3	b2	b1	b0
PC	U	Y	X	DP	B	A	CC

← pull order

Operation:
IFF b0 of postbyte set, then: CCR′ ←(SP), SP′←SP + 1
IFF b1 of postbyte set, then: ACCA′←(SP), SP′←SP + 1
IFF b2 of postbyte set, then: ACCB′←(SP), SP′←SP + 1
IFF b3 of postbyte set, then: DPR′ ←(SP), SP′←SP + 1
IFF b4 of postbyte set, then: IXH′ ←(SP), SP′←SP + 1
IXL′ ←(SP), SP′←SP + 1
IFF b5 of postbyte set, then: IYH′ ←(SP), SP′←SP + 1
IYL′ ←(SP), SP′←SP + 1
IFF b6 of postbyte set, then: USH′ ←(SP), SP′←SP + 1
USL′ ←(SP), SP′←SP + 1
IFF b7 of postbyte set, then: PCH′ ←(SP), SP′←SP + 1
PCL′ ←(SP), SP′←SP + 1

Condition Codes: May be pulled from stack; not affected otherwise.
Description: All, some, or none of the processor registers are pulled from the hardware stack (with the exception of the hardware stack pointer itself).
Addressing Mode: Immediate.
Comments: A single register may be pulled from the stack with condition codes set by doing an autoincrement load from the stack (example; LDX,S + +).

Pull Registers from the User Stack

PULU

Source Form:
PULU *register list*
PULU *#LABEL*
Postbyte:

b7	b6	b5	b4	b3	b2	b1	b0
PC	U	Y	X	DP	B	A	CC

← pull order

Operation:
IFF b0 of postbyte set, then: CCR′ ←(US), US′←US + 1
IFF b1 of postbyte set, then: ACCA′←(US), US′←US + 1
IFF b2 of postbyte set, then: ACCB′←(US), US′←US + 1
IFF b3 of postbyte set, then: DPR′ ←(US), US′←US + 1
IFF b4 of postbyte set, then: IXH′ ←(US), US′←US + 1
IXL′ ←(US), US′←US + 1
IFF b5 of postbyte set, then: IYH′ ←(US), US′←US + 1
IYL′ ←(US), US′←US + 1
IFF b6 of postbyte set, then: SPH′ ←(US), US′←US + 1
SPL′ ←(US), US′←US + 1
IFF b7 of postbyte set, then: PCH ←(US), US′←US + 1
PCL′ ←(US), US′←US + 1

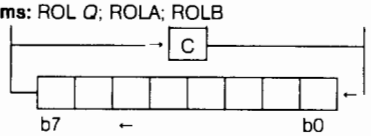
Condition Codes: May be pulled from stack; not affected otherwise.
Description: All, some, or none of the processor registers are pulled from the user stack (with the exception of the user stack pointer itself).
Addressing Mode: Immediate.
Comments: A single register may be pulled from the stack with condition codes set by doing an autoincrement load from the stack (example: LDX,U + +).

Rotate Left

ROL

Source Forms: ROL *Q*; ROLA; ROLB

Operation:



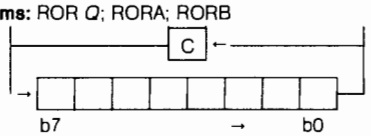
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Loaded with the result of the exclusive OR of bits six and seven of the original operand.
C — Loaded with bit seven of the original operand.
Description: Rotates all bits of the operand one place left through the C (carry) bit. This is a 9-bit rotation.
Addressing Mode: Inherent; Extended; Direct; Indexed.

Rotate Right

ROR

Source Forms: ROR *Q*; RORA; RORB

Operation:



Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Not affected.
C — Loaded with bit zero of the previous operand.
Description: Rotates all bits of the operand one place right through the C (carry) bit. This is a 9-bit rotation.
Addressing Modes: Inherent; Extended; Direct; Indexed.

RTI

Return from Interrupt

Source Form: RTI
Operation: CCR′←(SP), SP′←SP + 1, then
IFF CCR bit E is set, then: ACCA′←(SP), SP′←SP + 1
ACCB′←(SP), SP′←SP + 1
DPR′ ←(SP), SP′←SP + 1
IXH′ ←(SP), SP′←SP + 1
IXL′ ←(SP), SP′←SP + 1
IYH′ ←(SP), SP′←SP + 1
IYL′ ←(SP), SP′←SP + 1
USH′ ←(SP), SP′←SP + 1
USL′ ←(SP), SP′←SP + 1
PCH′ ←(SP), SP′←SP + 1
PCL′ ←(SP), SP′←SP + 1
IFF CCR bit E is clear, then: PCH′ ←(SP), SP′←SP + 1
PCL′ ←(SP), SP′←SP + 1
Condition Codes: Recovered from the stack.
Description: The saved machine state is recovered from the hardware stack and control is returned to the interrupted program. If the recovered E (entire) bit is clear, it indicates that only a subset of the machine state was saved (return address and condition codes) and only that subset is recovered.
Addressing Mode: Inherent.

RTS

Return from Subroutine

Source Form: RTS
Operation:
PCH′←(SP), SP′←SP + 1
PCL′←(SP), SP′←SP + 1
Condition Codes: Not affected.
Description: Program control is returned from the subroutine to the calling program. The return address is pulled from the stack.
Addressing Mode: Inherent.

SBC

Subtract with Borrow

Source Forms: SBCA *P*; SBCB *P*
Operation: R′←R − M − C
Condition Codes:
H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.
Description: Subtracts the contents of memory location M and the borrow (in the C (carry) bit) from the contents of the designated 8-bit register, and places the result in that register. The C bit represents a borrow and is set to the inverse of the resulting binary carry.
Addressing Modes: Immediate; Extended; Direct; Indexed.

SEX

Sign Extended

Source Form: SEX
Operation:
If bit seven of ACCB is set then ACCA′←FF_{16}
else ACCA′←00_{16}
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero, cleared otherwise.
V — Not affected.
C — Not affected.
Description: This instruction transforms a twos complement 8-bit value in accumulator B into a twos complement 16-bit value in the D accumulator.
Addressing Mode: Inherent.

ST (8-Bit)

Store Register into Memory

Source Forms: STA *P*; STB *P*
Operation: M′←R
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.
Description: Writes the contents of an 8-bit register into a memory location.
Addressing Modes: Extended; Direct; Indexed.

ST (16-Bit)

Store Register into Memory

Source Forms: STD *P*; STX *P*; STY *P*; STS *P*; STU *P*
Operation: M′:M + 1′←R
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.
Description: Writes the contents of a 16-bit register into two consecutive memory locations.
Addressing Modes: Extended; Direct; Indexed.

SUB (8-Bit)

Subtract Memory from Register

Source Forms: SUBA *P*; SUBB *P*
Operation: R′←R − M
Condition Codes:
H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.
Description: Subtracts the value in memory location M from the contents of a designated 8-bit register. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.
Addressing Modes: Immediate; Extended; Direct; Indexed.

SUB (16-Bit)

Subtract Memory from Register

Source Forms: SUBD *P*
Operation: R'←R − M:M + 1
Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if the overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.
Description: Subtracts the value in memory location M:M + 1 from the contents of a designated 16-bit register. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.
Addressing Modes: Immediate; Extended; Direct; Indexed.

SWI

Software Interrupt

Source Form: SWI
Operation:
Set E (entire state will be saved)
SP'←SP − 1, (SP)←PCL
SP'←SP − 1, (SP)←PCH
SP'←SP − 1, (SP)←USL
SP'←SP − 1, (SP)←USH
SP'←SP − 1, (SP)←IYL
SP'←SP − 1, (SP)←IYH
SP'←SP − 1, (SP)←IXL
SP'←SP − 1, (SP)←IXH
SP'←SP − 1, (SP)←DPR
SP'←SP − 1, (SP)←ACCB
SP'←SP − 1, (SP)←ACCA
SP'←SP − 1, (SP)←CCR
Set I, F (mask interrupts)
PC'←(FFFA):(FFFB)
Condition Codes: Not affected.
Description: All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt vector. Both the normal and fast interrupts are masked (disabled).
Addressing Mode: Inherent.

SWI2

Software Interrupt 2

Source Form: SWI2
Operation:
Set E (entire state saved)
SP'←SP − 1, (SP)←PCL
SP'←SP − 1, (SP)←PCH
SP'←SP − 1, (SP)←USL
SP'←SP − 1, (SP)←USH
SP'←SP − 1, (SP)←IYL
SP'←SP − 1, (SP)←IYH
SP'←SP − 1, (SP)←IXL
SP'←SP − 1, (SP)←IXH
SP'←SP − 1, (SP)←DPR
SP'←SP − 1, (SP)←ACCB
SP'←SP − 1, (SP)←ACCA
SP'←SP − 1, (SP)←CCR
PC'←(FFF4):(FFF5)
Condition Codes: Not affected.
Description: All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt 2 vector. This interrupt is available to the end user and must not be used in packaged software. This interrupt does not mask (disable) the normal and fast interrupts.
Addressing Mode: Inherent.

SWI3

Software Interrupt 3

Source Form: SWI3
Operation:
Set E (entire state will be saved)
SP'←SP − 1, (SP)←PCL
SP'←SP − 1, (SP)←PCH
SP'←SP − 1, (SP)←USL
SP'←SP − 1, (SP)←USH
SP'←SP − 1, (SP)←IYL
SP'←SP − 1, (SP)←IYH
SP'←SP − 1, (SP)←IXL
SP'←SP − 1, (SP)←IXH
SP'←SP − 1, (SP)←DPR
SP'←SP − 1, (SP)←ACCB
SP'←SP − 1, (SP)←ACCA
SP'←SP − 1, (SP)←CCR
PC'←(FFF2):(FFF3)
Condition Codes: Not affected.
Description: All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt 3 vector. This interrupt does not mask (disable) the normal and fast interrupts.
Addressing Mode: Inherent.

SYNC

Synchronize to External Event

Source Form: SYNC
Operation: Stop processing instructions.
Condition Codes: Not affected.
Description: When a SYNC instruction is executed, the processor enters a synchronizing state, stops processing instructions, and waits for an interrupt. When an interrupt occurs, the synchronizing state is cleared and processing continues. If the interrupt is enabled, and it last three cycles or more, the processor will perform the interrupt routine. If the interrupt is masked or is shorter than three cycles, the processor simply continues to the next instruction. While in the synchronizing state, the address and data buses are in the high-impedance state.

This instruction provides software synchronization with a hardware process. Consider the following example for high-speed acquisition of data:

```
FAST    SYNC            WAIT FOR DATA
        Interrupt!
        LDA     DISC    DATA FROM DISC AND
                          CLEAR INTERRUPT
        STA     ,X+     PUT IN BUFFER
        DECB            COUNT IT, DONE?
        BNE     FAST    GO AGAIN IF NOT.
```

The synchronizing state is cleared by any interrupt. Of course, enabled interrupts at this point may destroy the data transfer and, as such, should represent only emergency conditions.

The same connection used for interrupt-driven I/O service may also be used for high-speed data transfers by setting the interrupt mask and using the SYNC instruction as the above example demonstrates.
Addressing Mode: Inherent.

TFR

Transfer Register to Register

Source Form: TFR *R1, R2*
Operation: R1→R2
Condition Code: Not affected unless R2 is the condition code register.
Description: Transfers data between two designated registers. Bits 7-4 of the postbyte define the source register, while bits 3-0 define the destination register, as follows:

0000 = A:B	1000 = A
0001 = X	1001 = B
0010 = Y	1010 = CCR
0011 = US	1011 = DPR
0100 = SP	1100 = Undefined
0101 = PC	1101 = Undefined
0110 = Undefined	1110 = Undefined
0111 = Undefined	1111 = Undefined

Only like size registers may be transferred. (8-bit to 8-bit, or 16-bit to 16-bit.)
Addressing Mode: Immediate.

TST

Test

Source Forms: TST *Q*; TSTA; TSTB
Operation: TEMP←M−0
Condition Codes:
H —Not affected.
N —Set if the result is negative; cleared otherwise.
Z —Set if the result is zero; cleared otherwise.
V —Always cleared.
C —Not affected.

Description: Set the N (negative) and Z (zero) bits according to the contents of memory location M, and clear the V (overflow) bit. The TST instruction provides only minimum information when testing unsigned values; since no unsigned value is less than zero, BLO and BLS have no utility. While BHI could be used after TST, it provides exactly the same control as BNE, which is preferred. The signed branches are available.
Addressing Modes: Inherent; Extended; Direct; Indexed.
Comments: The MC6800 processor clears the C (carry) bit.

$\overline{\text{FIRQ}}$

Fast Interrupt Request (Hardware Interrupt)

Operation:
IFF F bit clear, then: SP′←SP−1, (SP)←PCL
SP′←SP−1, (SP)←PCH
Clear E (subset state is saved)
SP′←SP−1, (SP)←CCR
Set F, I (mask further interrupts)
PC′←(FFF6):(FFF7)
Condition Codes: Not affected.
Description: A $\overline{\text{FIRQ}}$ (fast interrupt request) with the F (fast interrupt request mask) bit clear causes this interrupt sequence to occur at the end of the current instruction. The program counter and condition code register are pushed onto the hardware stack. Program control is transferred through the fast interrupt request vector. An RTI (return from interrupt) instruction returns the processor to the original task. It is possible to enter the fast interrupt request routine with the entire machine state saved if the fast interrupt request occurs after a clear and wait for interrupt instruction. A normal interrupt request has lower priority than the fast interrupt request and is prevented from interrupting the fast interrupt request routine by automatic setting of the I (interrupt request mask) bit. This mask bit could then be reset during the interrupt routine if priority was not desired. The fast interrupt request allows operations on memory, TST, INC, DEC, etc. instructions without the overhead of saving the entire machine state on the stack.
Addressing Mode: Inherent.

Interrupt Request (Hardware Interrupt)

$\overline{IRQ}$

Operation:

IFF I bit clear, then: SP'←SP – 1, (SP)←PCL
SP'←SP – 1, (SP)←PCH
SP'←SP – 1, (SP)←USL
SP'←SP – 1, (SP)←USH
SP'←SP – 1, (SP)←IYL
SP'←SP – 1, (SP)←IYH
SP'←SP – 1, (SP)←IXL
SP'←SP – 1, (SP)←IXH
SP'←SP – 1, (SP)←DPR
SP'←SP – 1, (SP)←ACCB
SP'←SP – 1, (SP)←ACCA
Set E (entire state saved)
SP'←SP – 1, (SP)←CCR
Set I (mask further $\overline{IRQ}$ interrupts)
PC'←(FFF8):(FFF9)

Condition Codes: Not affected.

Description: If the I (interrupt request mask) bit is clear, a low level on the $\overline{IRQ}$ input causes this interrupt sequence to occur at the end of the current instruction. Control is returned to the interrupted program using a RTI (return from interrupt) instruction. A $\overline{FIRQ}$ (fast interrupt request) may interrupt a normal $\overline{IRQ}$ (interupt request) routine and be recognized anytime after the interrupt vector is taken.

Addressing Mode: Inherent.

Non-Maskable Interrupt (Hardware Interrupt)

$\overline{NMI}$

Operation:

SP'←SP – 1, (SP)←PCL
SP'←SP – 1, (SP)←PCH
SP'←SP – 1, (SP)←USL
SP'←SP – 1, (SP)←USH
SP'←SP – 1, (SP)←IYL
SP'←SP – 1, (SP)←IYH
SP'←SP – 1, (SP)←IXL
SP'←SP – 1, (SP)←IXH
SP'←SP – 1, (SP)←DPR
SP'←SP – 1, (SP)←ACCB
SP'←SP – 1, (SP)←ACCA
Set E (entire state save)
SP'←SP – 1, (SP)←CCR
Set I, F (mask interrupts)
PC'←(FFFC):(FFFD)

Condition Codes: Not affected.

Description: A negative edge on the $\overline{NMI}$ (non-maskable interrupt) input causes all of the processor's registers (except the hardware stack pointer) to be pushed onto the hardware stack, starting at the end of the current instruction. Program control is transferred through the $\overline{NMI}$ vector. Successive negative edges on the $\overline{NMI}$ input will cause successive $\overline{NMI}$ operations. Non-maskable interrupt operation can be internally blocked by a $\overline{RESET}$ operation and any non-maskable interrupt that occurs will be latched. If this happens, the non-maskable interrupt operation will occur after the first load into the stack pointer (LDS; TFR r,s; EXG r,s; etc.) after $\overline{RESET}$.

Addressing Mode: Inherent.

Restart (Hardware Interrupt)

RESTART

Operation:

CCR'←X1X1XXXX
DPR'←00_{16}
PC'←(FFFE):(FFFF)

Condition Codes: Not affected.

Description: The processor is initialized (required after power-on) to start program execution. The starting address is fetched from the restart vector.

Addressing Mode: Extended; Indirect.

Reference M/ Sample Programs

Example 1

```
10 '       This is an example of a BASIC program that calls
20 '       an assembly language program to paint the screen
30 '       yellow.
40 '
60 '       After entering the BASIC program save it on disk.
70 '
80 '       Run DOS and enter the assembly language program.
90 '       Use the WD and AD assembler commands to write the
100 '      source program to disk and to assemble it.
110 '
120 '      After returning to BASIC, load the assembled
130 '      program into memory with the LOADM command. You
140 '      must load the assembled program before the BASIC
150 '      program.
160 '
170 '      This program demonstrates how much faster
180 '      an assembly program can perform a function than a
190 '      BASIC statement.  After you run the program once,
200 '      delete lines 1030, 1040, 1050, and 1120.  Insert
210 '      this statement
220 '           1120 PAINT (1,1),2
230 '      and see how much longer it takes BASIC to paint
240 '      the entire screen yellow.
250 '
1000 'Specify the highest address BASIC can use.  This
1010 '     prevents BASIC from using the memory that contains
1020 '     your assembly language subroutine.
1030 CLEAR 200,16127
1040 PCLEAR 6          'reserve 6 pages of graphics memory
1050 DEF USR0=16128 'define the subroutine starting address
1060 ' The disk drive uses pages 0 and 1 of video memory.
1070 ' You must start at page 2, hex 1200.
1080 PMODE 3,2         'select mode 3, starting at page 2
1090 PCLS              'clear the screen
1100 SCREEN 1,0        'select graphics screen, color set 0
1110 COLOR 3,1         'set foreground color to blue
1120 A=USR(0)          'call the assembly language subroutine
1130 'draw a frame
```

```
1140 LINE (0,0)-(255,191),PSET,B
1150 LINE (12,12)-(242,178),PSET,B
1160 PAINT (2,2),4,3            'fill in the frame with red
1170 FOR X=50 TO 90 STEP 20     'draw top circles
1180 Y=30:ST=.5:EN=0            '     of big cloud
1190 GOSUB 5000
1200 Y=50:ST=0:EN=.5            'draw bottom circles
1210 GOSUB 5000:NEXT X          '     of big cloud
1220 FOR X=160 TO 180 STEP 20
1230 Y=30:ST=.5:EN=0            'draw top circles
1240 GOSUB 5000                       of little cloud
1250 Y=50:ST=0:EN=.5            'draw bottom circles
1260 GOSUB 5000:NEXT X                of little cloud
1270 Y=40:ST=.25:EN=.75         'draw left sides of clouds
1280 GOSUB 5020
1290 X=150:GOSUB 5020
1300 X=100:ST=.75:EN=.25        'draw right sides of clouds
1310 GOSUB 5020
1320 X=190:GOSUB 5020
1330 PAINT (52,30),3,3          'fill the clouds in with blue
1340 PAINT (162,30),3,3
1350 R=60:H=1:GOSUB 5040        'draw the umbrella
1360 R=37:H=1.7:GOSUB 5040      'draw the spokes of the
1370 R=15:H=4.7:GOSUB 5040      '     umbrella
1380 ST=.5:EN=0                 'draw the scalloped edges
1390 FOR X=78 TO 184 STEP 23    '     on the umbrella
1400 Y=124:GOSUB 5000
1410 NEXT X
1420 'draw umbrella handle
1430 DRAW "BM121,120;D40;R2;D2;R2;D2;R8;U2;R2;U2;R2;U3;
     L2;D2;L2;D2;L2;D2;L3;U2;L2;U2;L2;U40"
1440 PAINT (122,122),3,3        'paint umbrella handle
1450 PAINT (124,161),3
1460 PAINT (126,163),3
1470 C=8                        'set highest color number
1480 FOR X=68 TO 180 STEP 24    'paint umbrella panels
1490 PAINT (X,120),C,3
1500 C=C-1:NEXT X
1510 'play the song "Raindrops Keep Falling On My Head"
1520 GOSUB 6000:PLAY L$
1530 GOSUB 9000:PLAY L$
1540 PLAY M$:PLAY E$:PLAY N$
1550 PLAY G$:PLAY E$:PLAY O$
1560 PLAY P$:PLAY Q$:PLAY E$
1570 PLAY R$:PLAY S$:PLAY R$
1580 PLAY T$:PLAY P$:PLAY E$
1590 PLAY U$:GOSUB 9000
1600 PLAY V$:PLAY E$:PLAY E$
1610 PLAY W$:PLAY X$
1620 'Keep the image on the screen until a key is pressed.
1630 Z$=INKEY$
1640 IF Z$="" THEN 1630
1650 END
```

```
5000 CIRCLE (X,Y),13,3,.45,ST,EN
5010 RETURN
5020 CIRCLE (X,Y),16,3,.75,ST,EN
5030 RETURN
5040 CIRCLE (124,124),R,3,H,.5,0
5050 RETURN
5060 '
5070 'These lines define the notes of the song.
6000 A$="03;L4A;L8.;A;L16A;L8.;B-;L16A;L8.G;L16F;L4.;A"
6010 B$="P8;P4;P8;P16"
6020 C$="03;L16;C;04;L4C;L8.;C;L16C;L8.;D;L16C;L8.;C"
6030 D$="03;L16A;L4A;B-;G;F;04;E;P4"
6040 E$="P4"
6050 F$="04;L8.;D;L16C;03;L8.;A;L16E;04;L4.E"
6060 G$="P8"
6070 H$="04;L4.;D"
6080 I$="04;L4C;L8.;C;03;L16A;L8.;B-"
6090 J$="04;L16C;03;L8.;B-;L16A"
6100 K$="04;L4.;C;P4"
6110 L$="03;L4F;F;G"
6120 M$="03;L2;A"
6130 N$="04;L8.;C;03;L2G"
6140 O$="03;L8.;A;L4B-;L4A;L4G"
6150 P$="03;L8.;F;L4A;L4.;G"
6160 Q$="03;L4A;L8.;B-;04;L4D;L4C"
6170 R$="P8;P16"
6180 S$="03;L16A;04;L8D;L4C;L2C"
6190 T$="03;L16A;04;L8E;L4D;L2C"
6200 U$="P2;P1"
6210 V$="03;L4F;F;G;L2.;A"
6220 W$="03;L8.;F;L16F;04;L8.;D;L16C;03;L4F"
6230 X$="03;L8A;G;L4F;L2.;F"
9000 PLAY A$:PLAY B$:PLAY C$
9010 PLAY D$:PLAY E$:PLAY F$
9020 PLAY G$:PLAY H$:PLAY G$
9030 PLAY I$:PLAY J$
9040 PLAY I$:PLAY K$
9050 RETURN
```

```
00100 * Use EDTASM or EDTASMOV to enter this program.  Save
00110 *       the program on disk with WD command and
00120 *       assemble the program with AD command.  Do not
00130 *       use the SR switch because this program is
00140 *       called from BASIC, not executed from DOS.
00150 *
00160 * Use the LOADM command to load the assembled code
00170 *       into memory before you load the BASIC program.
00180 *       The ORG statement tells BASIC where in memory
00190 *       to load the program.
00200 *
00210         ORG   $3F00
00220 *
```

```
00230 * Put the hex code for a yellow point (55H) in
00240 *       register A and the address of the first byte
00250 *       of video memory (1200) in register X.
00260 *       The first byte of video memory is 1200 hex
00270 *       because the disk drive uses memory up to that
00280 *       address.
00290 *
00300 START   LDA   #$55
00310         LDX   #$1200
00320 *
00330 * Store the yellow dot at the current video memory
00340 *       address and increment X to the next video
00350 *       memory address.
00360 *
00370 SCREEN  STA   ,X+
00380         CMPX  #$2FFF     Is it the end of video memory?
00390         BNE   SCREEN     If no, continue to store dots
00400         RTS              If yes, exit subprogram and
00410 *                        and return to BASIC
00420 DONE    EQU   *
00430         END   START
```

Example 2

```
20 ' After entering the BASIC program save it on disk.
30 '
40 ' Run DOS and enter the assembly language program.  Use
50 '      the WD and AD assembler commands to write the
60 '      source program to disk and to assemble it.
70 '
80 ' After returning to BASIC, load the assembled
90 '      program into memory with the LOADM command. You
100 '     must load the assembled program before the BASIC
110 '     program.
120 '
130 ' Specify the highest address BASIC can use.  This
140 '     prevents BASIC from using the memory that contains
150 '     your assembly language subroutine.
160 CLEAR 200, 16127
170 DEF USR0=16128              'define address of subroutine
180 CLS                         'clear the screen
190 ' Print a prompting message and wait for a response.
200 INPUT "Press [ENTER] when ready"; A$
210 A=USR(0)                    'call subroutine
220 'Print another prompting message and wait for a response
230 INPUT "Want to do it again"; A$
240 'If operator types yes, start over.  Otherwise end.
250 IF A$="YES" THEN 20 ELSE END
```

```
00100 * Use EDTASM or EDTASMOV to enter this program.  Save
00110 *       the program on disk with WD command and
00120 *       assemble the program with AD command.  Do not
00130 *       use the SR switch because this program is
00140 *       called from BASIC, not executed from DOS.
00150 *
00160 * Use the LOADM command to load the assembled code
00170 *       into memory before you load the BASIC program.
00180 *       The ORG statement tells BASIC where in memory
00190 *       to load the program.
00200 *
00210         ORG   $3F00
00220 *
00230 * Put the hex code for a red checkerboard in
00240 *       register A and the address of the first byte
00250 *       of video memory (400) in register X.
00260 *
00270 START   LDA   #$0F9
00310         LDX   #$400
00320 *
00330 * Store the red checkerboard at the current video
00340 *       memory address and increment X to the next
00350 *       video memory address.
00360 *
00370 SCREEN  STA   ,X+
00380         CMPX  #$600     Is it the end of video memory?
00390         BNE   SCREEN    If no, continue to store red
00400 *                       checkerboards
00410         RTS             If yes, exit subprogram and
00420 *                       and return to BASIC
00430 DONE    EQU   *
00440         END   START
```