COSMAC Level I Assembly Language

# Lines and Comments[[1]](#footnote-1)

Each line or record in the source file is distinguished by an ending carriage return character. A line may consist only of a comment or of one or more statements optionally followed by a comment.

A comment is any series of characters beginning with two periods. It extends to the end of the line. Thus, the occurrence of two periods at any point in a line causes CRA to ignore the remainder of the line. Statements within a line are normally separated by semicolons (with the last statement optionally terminated by a semicolon). Within each statement, spaces (blanks) maybe used freely (except within symbolics and mnemonics) in order to improve readability. CRA will ignore them.

In all the examples which follow, a pair of square brackets will be used to enclose an optional entity – one which may or may not be included. Examples of valid lines are then:

1) .. COMMENT

2) STATEMENT1 [ ;STATEMENT2 ;···;

STATEMENTn ] [;] [ .. COMMENT]

# Symbol Definitions

## (Statement Labels and Equate Statements)

Any statement may optionally begin with a symbol (called a "statement label") immediately followed by a colon. Under these conditions, the symbol is entered into the symbol table and assigned the present location counter value. A statement thus has the form:

[SYMBOL:] STATEMENT BODY

(For example, LOOP: INC R4)

A symbol is also defined when it appears as the left-hand part of an EQUATE statement, which has the form:

SYMBOL=EXPRESSION

(For example, READER=6)

In this case, the expression is evaluated and the resulting two-byte value is assigned to the symbol in the symbol table. (Acceptable forms for symbols and expressions have already been explained.)

Thus, a symbol definition is indicated to CRA by the occurrence of ":" or "=" immediately after a leading sequence of alphanumeric characters in a statement.

When equating a symbol to a register number, only a decimal or a hexadecimal number should appear on the right side of the equation. For example:

COUNTR= 7 is correct

COUNTR= #07 is correct

COUNTR= R7 is incorrect

DELAY = COUNTR is also correct

Explicit Constants

At numerous points in the source program, the

programmer desires to directly ~pecify explicit

constants to CRA. Most often (but not always) the

hexadecimal equivalent of an explicit constant is

inserted directly into the output code stream at the

point where it appears in the source program. (For

example, initial data values and immediate

arguments may be explicitly defined this way.) CRA

allows the programmer the ability to specify absolute

constants in binary, hexadecimal, decimal, and

alphanumeric forms. The possible explicit constants

are summarized below.

Hexadecimal constants: A hex constant is specified

with either of the following forms:

Example

1) #hh ... hh #3EOF

2) X'hh .. hh' X'3EOF'

where each h is a hex digit (0 to F). eRA requires

that an even number of hex digits be specified.

There are further restrictions on hex constant lengths

under certain conditions.

Decimal constants: A decimal constant is specified

with either of the following forms:

Example

1) dd ... dd 635

2) F'dd ... dd' D'635'

where each d is a decimal digit (0 to 9). Each such

constant is converted into hex, producing one or two

bytes, depending on the space required to represent

it. Decimal values greater than 65535 are converted to

hex. but then truncated to two bytes (upper bytes

removed).

Expression constants: An expression constant may

be either form of the hex constant or the first form of

the decimal constant. Because an expression tran- Ii

slates to two bytes, a hex expression constant should "

normally be restricted to two or four digits in length.

Binary constants: A binary constant is specified in

the fonn:

B'bb ... bb'

(For example, B'OllOl')

where· each b is 0 or 1. Up to eight bits may be

specified. Each such constant is converted to one

byte', with leading O's assumed.

'{ext constants: A text constant is specified using the

·fOrm:

T'cc ... cc'

(For example, T'THIS IS TEXT')

where each c is any printable character, including

space. Each character is converted to its ASCII code

equivalent (see Appendix F) and is represented in one

byte. Characters that have no graphic associated with

them (i.e., ETX, DC-3, CR, LF, etc) should not be

used within a text constant. Entering an apostrophe

within a text constant is treated specially, however.

See "Additional Notes" below. Refer to Example 4

under "Examples of UT20 Read and Type Usage" to

see how CR, LF is handled.

Address Constants

The programmer finds it useful to specify not

only explicit or absolute constants, but also derived

constants whose values are assigned or "computed"

by the assembler. Because the fundamental function

of the assembler is to assign address values, such

constants are nonnally called address constants. For

CRA, an address constant has one of the following

forms:

Example:

1) A(expression) A(GEORGE + 2)

2) A. 1 (expression) A.1(LOOP)

3) A.O(expression) A.O(\*-X'10')

where the permissible forms for an expression have

already been defined. For all cases, the resulting

constant is derived by first evaluating the expression.

In the first case, the two-byte result is the constant. In

the second case, only the upper byte is used; for the

third case, only the lower byte. For all cases, the

resulting one- or two-byte value is assembled directly

into the code output.

Operation Mnemonics

CRA uses special two-, three-, and fourcharacter

mnemonics to represent the various instructions

in the COSMAC instruction set. These

mn'emonics are listed in Appendix E. When CRA

determines that an operation is being specified; it

looks it up in a table to detennine the code equivalent

of the mnemonic. (Note that this table is not the

symbol table, which contains only programmerdefined

symbols. ) Thus, use of an operation

mnemonic effectively defines an explicit hex coc;le

value to be inserted into the object stream. '

Instructions and Operands

There are two types of output code-producing

statements: instructions and data lists. An instruction

begins with an instruction operation

mnemonic. In some cases (such as IDL, RET, LDX,

etc.) this mnemonic is all that needs to be specified.

In most cases, however, the operation mnemonic

must be followed by an operand. The form of the

operand (i.e., the additional information which the

programmer needs to supply to fully define the instruction)

depends on the type of instruction. The

four operand forms follow.

Register operands: Many instructions (e.g., INC,

LDA, etc.) include a hex digit identifying one of the

scratchpad registers. The operand field in such a

statement may include either a single hex digit, or a

symbol. For the last case, CRA uses the least

significant hex digit of the symbol's value in the

symbol table as the register identifying field.

Examples:

DEC9

LDARF

PLOSAVE

("SAVE =#OF" could have previously defined

SAVE.)

I/O device operands: The instructions OUT and

INP require a device-identifying field. The operand

in such a statement may be a single digit in the range

1 to 7, or a symbol. Again, for the latter case, a

symbol table lookup occurs, using the least significant

hex digit of the symbol's value (checking also that it is

within the appropriate range).

Examples:

OUT 4

INPREADER

Branch address Every branch in truction

requires an operand speciJying th branch addre . II

th mnemonic i a short branch a one-byte operand i

g nerated. A two-byte operand i g Derated if the

mn moni ' i a long branch. Whenever CRA ee a

branch operation mnemonic, it expects to nextfind an

operand in the fOlm of an expression . The acceptable

(orins for expression have already been defined. In

en o-{ a short branch CRA valuates the expression

. by g tling a two-byte addre s che k that thi addre

s is within the current 256-byte page by

examin in g the upper byte, and uses the lower byte a

.:" the ' cond byt in the in truction. For a long branch,

the upper byte represents the page number and the

lower byte is the addre within that page.

Examples:

If A(LABEL) is #6789

BZLABEL

generates #3289

and LBZ LABEL

generates #C26789 -

Immediate pcrands: Several in tl'llctions include a

s ond byte a an immediate argument. The operand

field in uch a stat ment may be anyone-byte constant

(i.e., an ab oint or explicit constant or an

addres con tantl or a ymbol. For the latter case

CRA uses the lea t significant byte of th • ymbol s

assigned value.

Examples:

XRI X'FF'

ADIINCREM

LDI A.O(\*)

Note: When an immediate argument j pecified it is

the programmer' re ponsibility to make sure that it is

a one-byte con tanto If it is longer. CRA will not

generate an error message, but will merely insert the

entire constant into the output stream, possibly

causing an error during program execution.

Data Lists

The typical program normally includes memory

areas which ontain data values. Statement wh ich

d · fine inItial data values are also code-pr dueing

statements (although th code generated is normally

not "executable' J. The data list i a special statement

provided for the e purposes. It begin with either a

comma or the special mnemonic "DC" (which stands

for "Define Constant") and is followed by a sequence

of one or more constants separated by commas. Each

constant may be an absolute, explicit constant

(hexadecimal, binary, decimal, or text) or an address

constant or a symbol. For the last case, to be consistent

with the treatment of symbols as immediate

data, CRA substitutes the lower byte of the symbol's

assigned value. Thus, a constant in a dat~ list is

similar to an immediate operand, but now a length

greater than one byte is entirely justifiab~e.

Examples:

DC X'ABCD' ,355

,#ABCDEF,T'TEXT' ,B'On'

(Note: Any statement may be directly followed by a

data list without the intervening semicolon. For

example LDA 9,#3001.)

eRA Directives

The EQUATE DIRECTIVE (of the form

SYMBOL=EXPRESSION) has already been

discussed. Three other directive statements are also

recognized by CRA:

ORG Statement: This statement is written "ORG"

followed by an expression. CRA executes this

directive by setting the location counter equal to the

value of the expression.

Example: ORG \*+20 .. Reserve 2010 bytes of space

PAGE statement: The PAGE directive, simply

written "PAGE", increases the value of the location

counter to that of the beginning of the next 256-byte

page; i.e., the upper byte of the location counter is

incremented and the lower byte is set equal to zero.

END statement: The END directive, written

"END", informs CRA to terminate the assembly. It

should appear only once, as the last statement in the

source program. The END directive is normally

followed by a DC3 character. The DC3 is produced

by the EDITOR to signify an end of file.

Thu , in addition to recognizing all the instruction

operation mnemonic listed in Appendix G, CRA also

recognizes the special mnemonics "DC", "ORG",

"PAGE", and "END '.

Additional Notes

1) As noted earlier, a space is not permitted

within a syntactic entity (symbol, mnemonic, constant,

etc.). A space is not permitted between a

symbol being defined and the following colon or

equals sign. Note, however, that a space within a text

constant is permitted. It is translated into its ASCII

equivalent code. There is a case where a space is

required as a punctuation character. In order to

distinguish an operation mnemonic (including ORG)

from its following operand (if present), CRA expects

to find at least one space.

2) An apostrophe may be included within a text

constant by preceding it with a "dummy

apostrophe". Thus, the string IT's is written as a text

'constant as

,T'IT"S'

3) Special control characters (non-printing

characters, such as carriage return, line feed, etc.)

should not be placed within the quotes of a text

constant. Rather, they should be defined by splitting

the text constant into two successive text constants,

with the intervening control character represented

with a hex constant (using its ASCII code). For

example:

,T'LINEl',#ODOA,T'LINE\_2'

4) Several COSMAC instructions execute by

automatically advancing the pointer to an operand

byte after processing it. If the pointer to the operand

byte is the same as the current program counter (for

example, if X=P or if N=P), then the operand byte

may be considered an immediate operand (provided

an auto-increment occurs). A statement for such an

instruction (under the conditions specified) is most

conveniently followed by a comma followed by the

one-byte immediate constant. This sequence is

permissible because any statement may be immediately

followed by a data list - omitting the intervening

semicolon.

For example, assuming P=O, the sequence SEX

0; OUT 5 ,X'52' outputs the immediate hex constant,

#52, to output port and continues.

5) In general, any symbol may be referenced

before it is defined in a program (termed a "forward

reference"). Only one restriction exists: A symbol on

the right-hand side of an EQUATE statement (i.e., in

the expression) must have been previously defined.

6) CRA uses the location counter value before a

statement is processed as the value for any "\*,, occurring

within the statement. Thus, for example, for

BR \*+3, the value used for the \* is the location

where the branch byte (hex 30) will be placed, not one

byte past that. Thus, BNl \* will cause a program

loop until flag 1 goes true.

Code Examples and Review

Fig. 24 is a hypothetical program designed not to

do anything meaningful, but rather to present

examples of various acceptable CRA statements. It

contains a listing of the program and the

corresponding output code generated. Fig. 25 contains

the symbol table for the program. Both were

generated by a typical CRA assembly run.

In Fig 24, the left-hand column gives the location

counter value before the line was processeq. The next

column give the output hex code generated at that

location by the line. (Terminating semicolons in this

column should be ignored. They are present to format

the output file properly for subsequent loading of the

object program. See later operating instructions.) The

next column gives a source program line number for

reference purposes, and Jinally the source code is

reproduced. The running comments in the source

program refer to the statement examples where they

appear.

By reading the source program in detail (paying

special attention to the running comments), one can

quickly review much of what has been said concerning

COSMAC Level I Assembly Language.

Output code values may be verified by referring to

Appendices E and F. In particular the reader should

verify the values assigned to the symbols in Fig. 25.

Error Messages

Whenever CRA detects a violation of its syntax

rules, it generates an error message. There are,

however, some possible program errors which will not

be detected by CR A because they do not result in

syntax rule violations. For example, R3=8; INC R3.

R3 is now a symbol, the value of which is 8, so

register 8 gets incremented.

When there is a syntactical error, CRA indicates it

first, by printing the line in violation using its

standard listing format (location counter, output

code, line sequence number, source line); second, by

inserting a "?" at the detection point in the source

line; and third, by printing an error code on the next

line. If the error is detected at the end of the line, the

"?" may be omitted. In most cases, by looking up the

error code meaning in the listing which follows and by

noting the position of the inserted "?", the user can

easily determine the nature of the error.

It should be emphasized, however, that it is

possible that an error at one point in a source line may

be interpreted by CRA as an error at a different

point. For example, in T'TEXT ... COMMENT, a

single quote is missing after TEXT. It will not be

detected until the end of the line. (In fact, if the

comment happens to end in a single quote, the error

will go undetected.) Further, and more important, it

is possible for the error code to indicate one type of

error when another actually occurred. For example,

the statement SAM INC 3 is missing a colon after the

label SAM. The' primary meaning of the error code

which will return in this case is: "unrecognized

mnemonic". This response is understandable because

if CRA does not detect a colon or an equals sign, it

assUmes that the statement does not begin with a

symbol. If therefore expects a mnemonic or a comma

an,d does not find either~

Whenever an error exists, the output code is

questionable. However, as best it can, CRA increments

its location counter past this code and

continues to process the source program, possibly

detecting further errors which it flags similarly.

Detection of an error does not stop assembly of ~

program. CRA continues in its. attempt to find all

syntax errors.

.. ' An error on one line may cause several lines to be

}lagged. This response typically occurs when a line

containing a label is flagged because of a missing

colon and all subsequent references to that label are

also noted as "undefined. "

The possible error codes and their meanings . are

given in Table VII, and a summary of error messages

is given in Table VIII. If in the process of generating

a listingCRA creates a line that exceeds the standard

length (typically 78 characters), the line is broken by

a (CR) (LF) sequence. The rest of the line is continued

on the next line but is preceded by a' continuation

mark - a period.

1. NOTE: All discussion regarding special CRA punctuation characters (such as semicolon, colon, period, asterisk, parenthesis, equals sign, number sign, apostrophe, etc.) refer to those which do not appear within text constants (defined later). Any character within a text constant has no special punctuation significance to CRA. [↑](#footnote-ref-1)