A.S.P.

[A]ssembly [S]ource [P]reprocessor

For CUBIX 6809 Assembler

Revised: 30-Jul-90

Dunfield Development Systems

----------------------------

High quality tools for

Embedded Development

at low prices.

http://www.dunfield.com

Copyright 1983-2005 Dave Dunfield

All rights Reserved

A.S.P. Users Guide

TABLE OF CONTENTS

Page

1. INTRODUCTION 1

1.1 ASP Control Statements 2

1.2 Assignments 4

1.3 Expressions 4

1.4 Labels 4

1.5 Comments 4

2. QUALIFIERS 5

2.1 /COMMENT 5

2.2 /QUIET 5

2.3 /SOURCE 5

3. SAMPLE PROGRAM 6

A.S.P. Users Guide Page: 1

1. INTRODUCTION

ASP is an assembly language source preprocessor, which allows the

programmer to uses expressions, and control structures from within an

assembler program.

ASP reads as it's input an ASP/Assembler source file,

(Type='.ASP'), and produces as its output a standard assembler file

(Type='.ASM') which can be assembled using the system assembler.

ASP will produce extra code in the output assembler file, which

will perform the expressions and control structures which were

detected in the input file.

The code generated by ASP uses ONLY the A and B accumulator

registers in the 6809, and will never affect the index registers,

allowing them to be accessed completely by the program code. The A

and B accumultors may be modified by ASP when performing certain

functions, but will never be expected to contain values after an

assembler portion of the program is executed.

For the execution of the 'IF / ELSE / END' and 'WHILE / END'

constructs, ASP will place labels in the output file. The internal

labels generated by ASP are always preceded by the '\_' character, to

avoid conflicts with assembler code labels. Variable names, and

statement labels are placed in the output file exactly as they are

spelled in the program, allowing the assembler code to access them.

ASP is intended to simplify the control structures within

assembler source programs, and as such, does not make great attempts

to produce efficent code. While entire programs can be written using

ASP statements, such an approach is bound to produce less efficent

code then when writing in proper assembler.

A.S.P. Users Guide Page: 2

1.1 ASP Control Statements

1.1.1 ASM

The ASM statement causes ASP to assume all following lines

up to an 'END' statement are assembler source statements, and

will copy them into the output file qithout change.

1.1.2 BYTE <sym> [,<sym> ...]

or

BYTE CONSTANT <sym>=<val> [,<sym>=<val> ...]

BYTE defines 1 or more 8 bit storage location, which can be

refered to by the symbolic names specified. Each symbol will be

assigned one byte of storage, unless it is followed by a number

in braces EG: 'BYTE ABC(10 )' which causes the specified number

of bytes to be reserved. (The space before the closing brace is

nessary due to DOS call used to evaluate number).

If the CONSTANT keyword is found, the sumbols must have a

value assigned to them, and will be EQUated to that value, and

not assigned a location in memory.

1.1.3 CALL <label>

Calls the subroutine indicated by the <label> parameter.

1.1.4 CHAR <sym> [,<sym> ...]

or

CHAR CONSTANT <sym>=<val> [,<sym>=val ...]

CHAR is another name for BYTE, for readability purposes.

1.1.5 CODE <address>

Causes all subsequent code to be generated following the

indicated address. <address> may be any valid operand for the

assembler 'ORG' statement.

1.1.6 DATA <address>

Causes ASP to reserve data storage at the indicated address.

If the program does not contain a DATA statement, ASP reserves

data storage at the address following the last byte of code

produced.

1.1.7 END

End terminates one of 'ASM', 'IF/ELSE', 'WHILE', or

'SUBROUTINE'.

1.1.8 EXIT <expression>

Exits to the DOS, with the return code value indicated by

<expression>.

A.S.P. Users Guide Page: 3

1.1.9 GOTO

Produces a branch to the indicated label.

1.1.10 IF <expr1>, <condition>, <expr2>

Evaluates the expressions, and executes the following

statements, only if the condition is satisfied. <condition> may

be one of:

'EQ' - <expr1> is equal to <expr2>.

'NE' - <expr1> not equal to <expr2>.

'LT' - <expr1> less than <expr2>. (Signed)

'GT' - <expr1> greated than <expr2>. (Signed)

'LE' - <expr1> less than or equal to <expr2>. (Signed)

'GE' - <expr1> greated than or equal to <expr2> (Signed)

'LO' - <expr1> lower than <expr2>. (Unsigned)

'HI' - <expr1> higher than <expr2>. (Unsigned)

'LS' - <expr1> Lower or same as <expr2>. (Unsigned)

'HS' - <expr1> Higher or same as <expr2>. (Unsigned)

An 'ELSE' statement may follow an 'IF' statement, indicating

a section of code to be executed only if the comparison fails.

The whole IF/ELSE structure must be ended with an 'END'

statement.

1.1.11 INTEGER <sym> [,<sym> ...]

or

INTEGER CONSTANT <sym>=<val> [, ... ]

Defines one or more 16 bit variables or constants, the

format is the same for the 'BYTE' statement. If the optional

size is given for a symbol, that indicates the number of 16 bit

words of storage to reserve.

1.1.12 SUBROUTINE

Indicates the start of a subroutine, a label should be

placed on the same line as the 'SUBROUTINE' statement. The

subroutine must be terminated by an 'END' statement, at which

time a 'RTS' instruction will be generated.

1.1.13 WHILE <expr1> , <condition> , <expr2>

Performs the next block of statements (Up to an 'END'

statement), until the specified condition fails. The condition

may be any condition which is valid for the 'IF' statement. The

condition is tested at the beginning of each loop.

A.S.P. Users Guide Page: 4

1.2 Assignments

Whenever a statement is encountered which is not one of the

preceding ASP control statements, ASP attempts to evaluate it as

an expression, and to perform an assignment. The statement must be

of the form:

<symbol> = <expression>

1.3 Expressions

Whenever ASP evaluates an expression, it parses the expression

from left to right, and generates code to perform the indicated

function. There is no operator precedence, ASP generates the

expression code as soon as it parses. In particular, monadic

operators ('>', '<', '++', '--', '-:') etc, operate on the next

immediate element in the expression only. precedence of operations

may be forced using braces '()'.

EG: -:A+B Complements A, then adds B.

-:(A+B) Adds A and B, then Complements.

The stack is used to hold temporary results, when multi-level

or nested expressions are encountered.

Expressions are evaluated as integers, unless a byte value is

present as an element of the expression, in which code to evaluate

byte operands is generated. ASP will access the lower eight bits

of an integer is it is used within a BYTE oriented expression.

The following operators are recognized by ASP:

+ - Addition.

- - Subtraction.

! - Bitwise Logical OR.

& - Bitwise Logical AND.

!! - Bitwise Exclusive OR.

++ - Increment. (One operand)

-- - Decrement. (One operand)

-: - Complement. (One operand)

> - Shift right one. (One operand)

< - Shift left one. (One operand)

1.4 Labels

Labels may be placed on any ASP source statement line, and must

be followed by a colon (':').

1.5 Comments

Any line beginning with a '\*' in column number one, is assumed

to be a comment, and is ignored.

A.S.P. Users Guide Page: 5

2. QUALIFIERS

The following qualifiers are recognized by the ASP command.

2.1 /COMMENT

Causes ASP to include comments from the ASP source in the

assembler output file. By default, comments are NOT passed

through.

2.2 /QUIET

Prevents ASP from displaying its startup and progress messages

as it runs.

2.3 /SOURCE

Causes ASP to include the ASP source lines in the assembler

output as comments.

A.S.P. Users Guide Page: 6

3. SAMPLE PROGRAM

The following is a sample ASP program, It is written almost

entirely using ASP statements. Although for efficency this is not

normally a good practice, the purpose of this program is to

demonstrate the capabilities of ASP.

\*

CODE $D000

\*

INTEGER CONSTANT MAXCOUNT=256

BYTE CONSTANT LINESIZE=16

INTEGER OUTDIG

BYTE TEMP, COUNT

\*

MAIN: OUTDIG = 0

COUNT = 0

WHILE OUTDIG, LO, MAXCOUNT

CALL HEXOUT

OUTDIG = ++OUTDIG

COUNT = ++COUNT

IF COUNT, LO, LINESIZE

CALL SPACE

ELSE

COUNT = 0

CALL LFCR

END

END

EXIT 0

\* SUBROUTINE TO DISPLAY A NUMBER IN HEX.

HEXOUT: SUBROUTINE

TEMP = >>>>OUTDIG & $0F

CALL HEX1

TEMP = OUTDIG & $F

HEX1: TEMP = TEMP + $30

IF TEMP, HI, $39

TEMP = TEMP + 7

END

ASM

LDA TEMP

SSR 33

END

END

\* SUBROUTINE TO DISPLAY A LINE-FEED CARRIAGE RETURN.

LFCR: SUBROUTINE

ASM

SSR 22

END

END

\* SUBROUTINE TO DISPLAY A SPACE.

SPACE: SUBROUTINE

ASM

SSR 21

END

END