The CUBIX

6809 Assembler

Users Guide

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

6809 ASSEMBLER

TABLE OF CONTENTS

Page

1. INTRODUCTION 1

2. ASSEMBLER SOURCE INPUT 1

2.1 Source File Format 1

2.2 Expressions 2

2.3 Addressing modes 3

2.4 Special Characters 4

3. ASSEMBLER DIRECTIVES 5

4. ERROR MESSAGES 7

5. QUALIFIERS 8

6. IMAGE BUILDING UTILITY 9

7. USING THE ASSEMBLER 9

7.1 Null Filename 10

7.2 Performance 10

6809 ASSEMBLER Page: 1

1. INTRODUCTION

The assembler, is a multi-pass assembler, which produces code for

the Motorola 6809 microprocessor, and runs under the CUBIX Disk

Operating System on the 6809. It accepts a input file containing

assembly source statements, and generates both an object file,

containing machine instructions, and a listing file, containing a

printed list of the code generated.

2. ASSEMBLER SOURCE INPUT

2.1 Source File Format

The assembler accepts a source file, containing assembly

statements, one per line. Lines beginning with an asterisk ('\*'),

are treated as comments, and are ignored by the assembler. All

other lines have the following format:

<label field> <opcode field> <operand field> <comment field>

A description of the individual fields follows:

<label field>

This field must start at column one, and contains the label for

this line. The label will be entered in the assemblers SYMBOL

TABLE, such that the address of the instruction on this line can

be referenced by this label. If the character in column one is a

blank or a tab, then the assembler will ignore the label field,

and no symbol table entry will be made for this address.

<opcode field>

This field occurs immediately after the label field, and

contains the symbolic 6809 opcode or assembler directive that is

to be processed for this line. The opcode field is separated from

the label field by a number of spaces or tabs.

<operand field>

This field occurs immediately after the opcode field, and

contains the operands to the 6809 opcode or assembler directive

which was specified in the opcode field. If no operands were

required, then the opcode field is ignored by the assembler, and

is treated as part of the comment field. The operand field is

separated from the opcode field by a number of spaces or tabs.

<comment field>

This field occurs immediately after the operand field, and is

ignored by the assembler, but reproduced in the listing file. It

extends to the end of the source line, and is separated from the

operand field by tabs or spaces.

6809 ASSEMBLER Page: 2

2.2 Expressions

When an 8 or 16 bit value is required as an operand to an

assembler directive or an instruction, either a simple value, or

an expression consisting of simple values and operators may be

used. All expressions are evaluated using 16 bit values. When used

for an eight bit result, the lower eight bits are used.

Expressions are evaluated from left to right, as each operator is

encountered, with no precedence. Spaces or tabs are not allowed

within an expression, unless they are contained within a character

string.

The following operators may be used in an expression:

2.2.1 Unary (one operand) operators:

- - Negation, returns the negative of the next value.

~ - Complement, returns one's complement of the value.

= - Swaps the high and low bytes of the next value.

2.2.2 Binary (two operand) operators:

+ - Addition.

- - Subtraction.

\* - Multiplication.

/ - Division.

\ - Modulus, returns remainder after division.

& - Bitwise logical AND.

| - Bitwise logical OR.

^ - Bitwise exclusive OR.

2.2.3 Values in expressions

The following forms of simple values may be used.

nnn - Decimal number, eg: 21

$hhh - Hexidecimal number, eg: $15

%bbb - Binary number, eg: %10101

@ooo - Octal number, eg: @177

'cc' - ASCII character data.

<label> - Value of a label from symbol table.

\* - Value of current program counter.

6809 ASSEMBLER Page: 3

2.3 Addressing modes

The assembler supports all of the 6809's addressing modes, and

will determine from the operands to an instruction which

addressing mode is to be used. If register offset addressing is

used, then the assembler will determine the most efficient offset

size (5 bit, 8 bit or 16 bit).

If the operand is preceded by a pound sign ('#'), then the

assembler will use immediate addressing mode, and code the value

of the expression as the immediate data. Note that if an

instruction only requires eight bits of immediate data, (eg. CMPA)

then only the LOWER eight bits of the expression value will be

used. The high eight bits can be accessed by preceding the value

with '=' (Swapping the high and low bytes).

If the operand is preceded by a left angle bracket ('<'), then

direct page addressing will be used. Note that only the lower

eight bits if the expression value will be used, and it is the

responsibility of the programmer to insure that the direct page is

set correctly.

If the operand is preceded by a right angle bracket ('>'), or

if no other addressing mode applies, then extended addressing will

be used.

If the operand in enclosed in square braces ('[' & ']'), then

the indirect form of that particular addressing mode will be used.

6809 ASSEMBLER Page: 4

2.4 Special Characters

Below is a summary of the special characters recognised by the

assembler:

Character(s) Meaning

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

<Blank> - Separates assembler source fields.

<Tab> - Separates assembler source fields.

0-9 - DECIMAL numbers.

$ - Specifies a HEXIDECIMAL value.

% - Specifies a BINARY value.

@ - Specifies an OCTAL value.

' - Specifies an ASCII LITERAL value.

+ - Performs ADDITION in expressions.

- - Performs SUBTRACTION in expressions, indicates

negative values.

~ - Performs complement of a value.

| - Performs a LOGICAL OR in expressions.

& - Performs a LOGICAL AND in expressions.

^ - Performs EXCLUSIVE OR in expressions.

= - Swaps the high and low bytes in an expression.

Can be used to access the high byte of a 16 bit

value as eight bits.

, - Separates multiple operands in operand field.

# - Specifies IMMEDIATE addressing mode.

> - Specifies EXTENDED addressing mode.

< - Specifies DIRECT PAGE addressing mode.

[ ] - Specifies INDIRECT addresing mode.

\* - When found in column one, specified that this is a

COMMENT line. When used as a value element in an

expression, specifies the value of the PROGRAM

COUNTER. When used as an operator in an expression,

performs multiplication.

/ - Performs DIVISION in expressions.

\ - Performs modular arithmitic in expressions.

<Cr> - End of line, Statement terminator.

6809 ASSEMBLER Page: 5

3. ASSEMBLER DIRECTIVES

The following assembler directives are implemented:

ORG <expression>

Sets the current assembly program counter to the value of the

operand expression. This causes all subsequent code (up to next ORG)

to be generated to run at that address.

<label> EQU <expression>

Creates the symbol <label>, and gives it the value of the

expression.

FCB <expression>[,<expression,...]

Generates eight bit data bytes containing the values of the

expressions.

FDB <expression>[,<expression>,...]

Generates sixteen bit data words containing the values of the

expressions.

RMB <expression>

Reserves a number of bytes of memory equal to the value of the

expression. The contents of the reserved bytes will be undefined.

FCC "<string>"

Generates an ASCII string, the delimiter characters may be any

character which is not part of the string.

FCCZ "<string>"

Operates the same as 'FCC' except that the string will be

terminated with a zero byte.

SSR <expression>

Generates an SYSTEM SERVICE REQUEST to the operating system.

PAGE

Forces a page skip in the listing.

TITLE <text>

Sets the page title to the text contained in the remainder of the

line.

6809 ASSEMBLER Page: 6

SETDP <expression>

Sets the assemblers default direct page register to the eight bit

value of <expression>. Whenever a reference is made to a memory

location by address, without a specific addressing mode specified,

the assembler will use extended addressing, unless the high byte of

the address matches the value of the default direct page register. If

this occurs, the assembler will use direct page addressing. It is the

responsibility of the programmer to insure that the 6809 direct page

register will contain the proper value during memory references

subsequent to a 'SETDP' directive. If the value of <expression> is

greater than 255, or less than zero, the default direct page register

will be disabled, and all unspecified memory references will use

extended addressing. This is the default mode at the beginning of an

assembly.

Note: The upper eight bits of a label's value can be accessed as an

eight bit value by dividing the lable by 256.

EG: ' SETDP LABLE/256 '

6809 ASSEMBLER Page: 7

4. ERROR MESSAGES

When the assembler detects an error in the assembly, it will

generate a message in the listing indicating what type of error has

occured. This will occur on the line following the error, or at the

top of the listing if the error was generated in the first assembly

pass. The error message also contains an error number which may be

used to reference the following summary of error messages.

Message# Message and description.

--------+--------------------------------------------------------------

0 | Duplicate symbol: <symbol name>

| The indicated symbol is defined more that once within this

| assembly.

--------+--------------------------------------------------------------

1 | Unknown opcode or directive

| The instruction field on the indicated line does not

| contain a valid 6809 instruction or assembler directive.

--------+--------------------------------------------------------------

2 | Address out of range

| A short branch instruction on the indicated line has an

| operand indicating a address which is more than +127 or -128

| bytes away from the current program counter address.

--------+--------------------------------------------------------------

3 | Invalid addressing mode

| The addressing mode indicated by the operand field of the

! indicated line does not apply to the instruction on that line.

--------+--------------------------------------------------------------

4 ! Invalid register specification

! The instruction on the indicated line specifies a register

! which is not a valid 6809 register, or cannot be used in the

! context specified by the instruction.

--------+--------------------------------------------------------------

5 ! Undefined symbol

! A symbol referenced in the indicated line is not defined

! anywhere within this assembly, and has no value.

--------+--------------------------------------------------------------

6 ! Invalid expression syntax

! The expression on the indicated line contains a character

! which is not recognized as a valid operator.

--------+--------------------------------------------------------------

7 ! Invalid argument format

! The indicated line has an operand which is not in proper

! format.

--------+--------------------------------------------------------------

8 | Improperly delimited string

! A character string constant on the indicated line does not

! have a proper closing delimiter. This is normally the single

! quote character, but may be another character in conjunction

! with the FCC or FCCZ directives.

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

6809 ASSEMBLER Page: 8

5. QUALIFIERS

The following qualifiers are accepted by the 'ASM' command:

/ERROR

When this qualifier is specified, the assembler will only output

lines to the listing file if they contained an error.

/FAST

When this qualifier is specified, the assembler will not perform

optimization passes. All offsets are assumed to be 16 bits, All

non-specified memory references will be done with extended

addressing, and forward references in equate statements will not

work. The code generated will be inefficient, but the assembler will

run quite a bit faster. This qualifier is useful when you are

initialy testing and debugging a program.

NOTE: Address out or range error's occuring with this qualifier

may be a result of larger code produced, and may assemble ok without

this qualifier.

/QUIET

Prevents the assembler from displaying informational messages on

the terminal, as it performs each step of the assembly.

/SYMBOL

This qualifier causes the assembler to display the symbol table at

the end of the listing. If the /TERM qualifier is also specified, the

symbol table display will be formatted for an 80 column display

instead of 132.

/TERM

This qualifier causes the assembler to output the listing to the

terminal instead of to a file.

All of the above qualifiers may be abbreviated to a mimimum of one

character (Excluding the slash).

EG: ASM/F/E/T PROGRAM

6809 ASSEMBLER Page: 9

6. IMAGE BUILDING UTILITY

The output from the assembler is an intermediate object file,

which must be converted to a format suitable for loading. This is

done by the BUILD program. A linker is planned, which will allow

several object files to be linked together into one image, but this

is not yet implemented.

The BUILD command will accept the qualifier '/MHX' which causes it

to generate a Motorola Hex format download file (suffix '.MHX'),

instead of an executable load image (suffix '.EXE').

Unless the '/KEEP' qualifier is given to BUILD, it will delete the

intermediate (suffix '.OBJ') file produced by the assembler before it

writes the image file.

The '/QUIET' qualifier may be used to prefent BUILD from

displaying statistics on the created load image, and the name of the

output file.

7. USING THE ASSEMBLER

The assembler source input file must be entered into a file with a

type of '.ASM'. Once this is done, enter the command:

ASM <filename>

where <filename> is the name of the input file. Do not include the

'.ASM' suffix as part of the filename.

If there were any errors in the assembly, the assembler will

display a message indicating how many error occured, and return with

a return code value of 100. if there were no errors, the assembler

will return a return code of 0 (success).

The assembler will produce a <filename>.OBJ file with the

intermediate object code, and a <filename>.LST file with the listing.

When the program assembles without errors, generate a executable

image with the command:

BUILD <filename>

This will read the <filename>.OBJ file, and generate a <filename>.EXE

file which can be run with the command:

RUN <filename>

6809 ASSEMBLER Page: 10

7.1 Null Filename

If no filenames are given to the ASM and BUILD programs, they

will assume a null filename. The assembler will look for '.ASM'

and generate '.OBJ' and '.LST'. The BUILD program will read '.OBJ'

and generate '.EXE' or '.MHX'. The dos RUN command will also

assume a null filename, and will attempt to execute a file called

'.EXE' if no filename is given to it.

The above behaviour can be used to simplify the steps required

to assemble a program. This is particularly useful during the

development stage when a program may need to be assembled a number

of times.

To do this, simply copy your progam into a file called '.ASM'.

Now all you need to type to assemble it is 'ASM', to generate the

image, just type 'BUILD', and to run the program, just type 'RUN'.

7.2 Performance

Although the assembler is capable of resolving first pass

forward references, many occurences of the above can cause the

assembler to operate much slower than it normally would. Below is

a list of things to avoid when performing large assemblies:

Forward referenced symbols as offsets.

Forward referenced symbols in EQUate statements.

Direct page addressing to forward referenced symbols using

implied direct page addresing (SETDP).