C SC 230

Computer Architecture and Assembly Language Programming

AS11M Assembler User Guide

1. Introduction

The AS11M assembler is a cross-assembler that runs on PCs and produces code for the MC6811 microcontroller. It is available as free-ware from MSOE at the site http://www.msoe.edu/eecs/ce/ceb/barnicks/courses/cs280/cs280.htm and locally at http://gulf.uvic.ca/~csc230. AS11M is a DOS program that can be run under Windows.

2. Preparing the Assembly Language Program

The Assembly language source program should be prepared as a text file. The easiest way to create a new text document is to position the mouse where you want the document and right click. Drag down to New, slide the cursor over and then position to new Text Document and click. Left click on the name and you can then type the name you want for the document. Double-clicking on the document's icon will open it using the Notepad editor.

Be sure to save your program to drive H: which is your home directory.

3. Program Layout

A simple 6811 assembly language program is shown below. A line beginning with a ; is a comment. Blank lines can be inserted as appropriate for clarity. The ; should appear in the first character position.

```
; Pin toggle test program
; Author: Michael Miller
; Date: September 2, 1998

REGBAS EQU  $1000
PORTB  EQU  $04

DELAY  EQU  255

       LDX  #REGBAS
       LDAA #0
LOOP   STAA PORTB,X  / set Port A to 0
       LDY  #DELAY
WAIT   DEY
       BNE  WAIT     / delay by counting
       EORA #1       / flip bit 0 in ACCA
       JMP  LOOP
       END
```

Code lines have four fields: label, opcode, operands and comment. The fields must be separated by blanks or tabs. It is best to use tabs to align the fields.

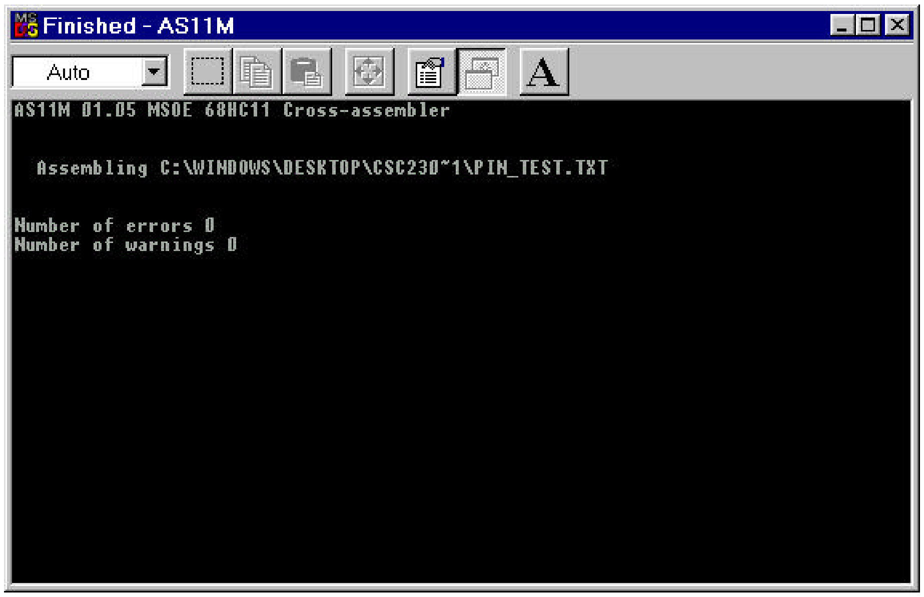
When present the symbol in the label field should start in the first character position. There must be whitespace at the start of a line with no label.

There should not be any blanks in the operand field. This is important. Some assemblers allow blanks between operands but this one does not.

Anything after the first blank or tab following the operands is taken as a comment. In the example above a / is used to delineate the comment. This is for clarity. It is not necessary but it is a good idea.

4. Assembling the Program

Open the CSC 230 folder on the desktop. It contains three icons. The AS11M assembler the WOOKIE – 6811 simulator and one called Drag and Drop. The simplest way to assemble your program is to drag its icon onto the Drag and Drop icon in the CSC 230 folder. A window similar to the one shown in Fig. 1 will be produced.



```
AS11M 01.05 MSOE 68HC11 Cross-assembler


  Assembling C:\WINDOWS\DESKTOP\CSC230~1\PIN_TEST.TXT


Number of errors 0
Number of warnings 0
```

Fig. 1

The assembler produces two files both with the same name as your source TXT file. One is an LST file and is the program listing. The second is an S19 file. This file contains the 6811 code for your assembled program. Both files are by default put in your H:\ directory regardless of the location of your TXT file. If you know enough about windows you can modify this. Easiest thing is to just accept that H:\ is where they go.

It is important that the S19 and LST files are in the same directory as that is what the WOOKIE -–6811 simulator requires.

For interest, the contents of the S19 file produced for the program shown earlier are

```
S1170000CE10008600A70418CE00FF180926FC88017E0005A5
S9030000FC
```

The LST file is rather more meaningful. It shows the code produced for your program associated with the original source code. Any error or warning messages would appear at the appropriate position in the code. The LST file for the program above is as follows:

```
AS11M 01.05  Sun Sep 6, 1998 21:21  PIN_TEST.lst

0001                    ; Pin toggle test program
0002                    ; Author: Michael Miller
0003                    ; Date: September 2, 1998
0004
0005 1000               REGBAS  EQU   $1000
0006 0004               PORTB   EQU   $04
0007
0008 00ff               DELAY   EQU   255
0009
0010 0000 ce 10 00              LDX   #REGBAS
0011 0003 86 00                 LDAA  #0
0012 0005 a7 04         LOOP    STAA  PORTB,X / set Port A to 0
0013 0007 18 ce 00 ff           LDY   #DELAY
0014 000b 18 09         WAIT    DEY
0015 000d 26 fc                 BNE   WAIT / delay by counting
0016 000f 88 01                 EORA  #1   / flip bit 0 in ACCA
0017 0011 7e 00 05              JMP   LOOP
0018 0014                       END


Number of errors 0
Number of warnings 0
```

5. 6811 Assembly Language Rules

6811 assembly language programming will be illustrated in class, in the labs and there are several examples in the text. This document covers some basics to get you started.

5.1 Numbers: Decimal numbers are written in the usual manner without a decimal point. Hexadecimal numbers are preceded by a $. Binary numbers are preceded by a %. Note that # denotes immediate addressing mode and does not precede every number in a program.

5.2 Symbols: Choose meaningful symbols. They can contain letters, digits and the character _ but must not start with a digit. Uppercase and lowercase letters are distinct. Avoid using assembler opcodes as labels. The assembler doesn’t care, but you and the marker will have great difficulty reading the program. Note that symbols are defined throughout the entire assembly language program. There are no scoping rules as in high-level languages.

5.3 Assembler Directives: An assembler directive has a layout similar to an executable instruction but provides direction to the assembler rather than producing machine language. The supported directives include:

ORG **Set location counter to ORiGin.**
ex: START ORG $C000
This allows the user to set the location counter to a specified value. This is commonly used to ensure that code is assembled for the correct addresses which correspond to your memory map. The above example sets the location counter to the address $C000. The location counter is a value internal to the assembler that keeps track of where the code or data associated with each instruction or directive is to be placed in memory.

FCC **Form Constant Character string.**
ex: STRING FCC 'A string of ASCII'
This causes the label to be assigned the address of the first letter in the string. The string is stored in successive bytes with its ASCII value. The quotes are not part of the string.

FCB **Form Constant Byte.**
ex: TABLE FCB 0,1,$02,'A'
This causes the assembler to store the operands in successive 8-bit bytes. The operands must be 8-bit values or single character constants. The values are separated by commas.

FDB **Form Double Byte.**
ex: CONSTS FDB 0000,$1234
Function is similar to that of FCB except 16-bit values are stored sequentially in memory.

EQU **EQUate symbol to a value.**
ex: LABEL EQU $1017
This causes the assembler to add the label to the symbol table and to equate it to the given value. In this example, LABEL equates to $1017. The label cannot be redefined elsewhere. The value must not be forward-referenced or undefined.

RMB **Reserve Memory Bytes.**
ex: VARS RMB 3
This directive allows the user to reserve a block of memory and associate a label to the address of the block. It is used for variables, tables, etc. This example reserves the next three sequential bytes for variables and associates the first address with the symbol VARS.

BSZ **Block Set Zero**.
ex: BLK BSZ 10
This directive reserves memory and initializes it to 0. This example reserves 10 sequential bytes, sets them to 0, and associates the first address with the symbol BLK.

FILL.
ex: BLK FILL 10,15
This directive reserves memory. It reserves the number of bytes specified by the second operand and fills each with the value given by the first operand. This example reserves 15 bytes all initialized to 10 and associates the symbol BLK with the address of the first byte.

END **End of assembler input.**
ex: END
This causes the assembler to stop reading the file. All lines following this line are comments. If you do not use an END directive the assembler reads until the end of file.

Note that a label is not mandatory in an assembly directive except for EQU. END should not have a label. Also note that when a label is present, its value is the current value of the assembler location counter except for the EQU directive which is the only one that explicitly assigns a value to the specified label.

5.4 Assembler Arithmetic: The assembler supports simple arithmetic expressions involving symbols and constants. For example:

```
LDAA #X+3

CMPA Y+5
```

The expression must be directly computable at assembly time and is used as the value of the operand.

6. Comments and Corrections

This document is intended for use by students in C SC 230 and for others who might use the AS11M assembler for educational work at UVic. Comments and corrections should be sent to mmiller@csr.uvic.ca.