

asm

COLLABORATORS

	<i>TITLE :</i> asm		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		July 10, 2022	

REVISION HISTORY

<i>NUMBER</i>	<i>DATE</i>	<i>DESCRIPTION</i>	<i>NAME</i>

Contents

1	asm	1
1.1	AT90S Cross assembler	1
1.2	copyright	1
1.3	intro	2
1.4	how	2
1.5	source	3
1.6	direct	4
1.7	errors	5
1.8	math	7
1.9	req	7
1.10	maccy	7
1.11	bugs	8
1.12	changes	8
1.13	contact	9

Chapter 1

asm

1.1 AT90S Cross assembler

Atmel AT90S Cross assembler.

Version: 1.04B

Contents:

Copyright/Disclaimer

Introduction

Command line

Source format

Assembler directives

Assembler errors

Math operators

Machine file 'Atmel.txt'

Requirements

Bugs/To do

Changes from V1.00B

Contact

1.2 copyright

Atmel AT90S Cross assembler Version 1.04 Beta

Copyright 2000 Loony Juice Software.
Written by Lee Atkins

Atmel is probably a registered trade mark of the Atmel corp.

This software is provided 'as is' and with no warranty.
Use of this software is entirely at the users own risk.

1.3 intro

Atmel cross assembler V1.04

This cross assembler produces code for the Atmel 90S range of processors. To get the full use from this assembler you will need the Atmel in circuit programmer so you can try your latest code.

The Atmel range of microcontrollers is a good alternative to other similar microcontrollers available. I say other because I am not going to mention names, it should become obvious.

Programming the Atmel micro's is simplicity its self, theres no stupid memory paging and it has a good instruction set. It is also very fast. An AT90S2313 with an 8MHz resonator will execute most instructions in 125ns. They are available in speeds up to 20MHz. All the 90S range have flash ROM so you can download code to the chip time and time again until you get it right without having to wait for the chip to be erased.

Included in this archive is everything you need to get going. Simply build the programming cable, get hold of some devices and yer off. There is also some example code which you can try out, examine and take to bits and use for your own projects. I have tried to include as much information as possible in this archive so there are no delays due to lack of info. I can't stand holding a project up because relevant information isn't available.

I decided to write this assembler because I was sick of the development environment I was using on the PC. After buying an in circuit programmer at work and using for a while I became fed up with the time it was taking to program the devices and also the inflexibility of the windoze software, so I wrote my own programmer and then ported it to the Amiga for use at home. So then I needed an assembler so I wrote one.

There is also a C compiler for these devices written by some jokers who expect you to fork out loads for the thing, and then when you discover the code it produces has errors they will SELL you an upgrade, no thanks. So look out for the C compiler I am also producing, which will be free.

Follows is a guide on how to use the assembler, some assembly knowledge is assumed!

1.4 how

ASSEMBLING

Start the assembler from the command line with
ASM <source> <out> <list>

<source> is the filename of your ASCII source file.
<out> and <list> are optional:-

<out> is the S Record that will be produced after assembly, this is the file you send to the chip using the programming software.
This file name is optional and if left out the S Record will be named a.out

<list> will produce an assembler listing on this file. If this is left out no listing will be produced. An <out> filename must be given to get to this option.

An example command line would be:-
for source file named LEDFLASH.ASM

```
ASM LEDFLASH.ASM LEDFLASH.S19 LEDFLASH.LST
```

will produce the files ledflash.s19 (the code) and ledflash.lst (the listing).

1.5 source

SOURCE FILE FORMAT

The source file is just any old ascii file which contains your code.
As usual the source is split into three fields :-

```
[LABELS] [OPCODE] [OPERAND]
```

These three fields must be separated by at least a single space character.
example:

```
LOOP:  
  dec R16  
  brne LOOP
```

is good

```
LOOP:  
  dec R16  
  brne LOOP
```

is bad, as is

```
LOOP:  
  dec R16  
  brne LOOP
```

etc..

Any text following a ; symol will be treated as a comment.

```
LOOP: dec R16      ;This is a comment for this delay.  
      brne LOOP
```

The assembler is not case sensitive so the label LOOP: above could be referenced by loop:

1.6 direct

There are a few assembler directives to get things done, these are commands to the assembler and do not produce any code.

ORG x

Starts the following code at the specified address
eg:

```
ORG $0000  
...code....
```

starts the code at address 0000

DB x (,y,z,zz)

DW x (,y,z,zz)

Places constant data at the current code address.

DB defines bytes, DW defines words.

eg:

```
DB 0,2,243,"Hello",3
```

places the constant data 0,2,243 and then the ASCII codes for the string 'Hello' (without a null terminator) and then a 3. Altogether will use up 9 bytes of ROM.

If the constant data does not end on an odd address (using DB) no packing byte is inserted eg.

```
DB 1,2,3  
DB 4
```

would produce the bytes 1,2,3,4 in memory in that sequence, some assemblers would have produced 1,2,3,0,4 in memory which in my view is wrong and very annoying.

INCLUDE "file" (or incude file)

Inserts another file specified by 'file' into the assembly.

The quotes round the filename are optional.

END

Ignored by this assembler.

EQU

Sets a label to a value.

eg

```
WALK equ 5
```

will make WALK equate to the value 5. Notice the label WALK is in the label field!

DEF

Defines a textual replacement (a micromacro !)

eg

```
TRIGGER_NUKE def PORTB,NUKE_PIN
```

will replace the text 'TRIGGER_NUKE' within the source with whatever it has been DEFined as.

1.7 errors

Atmel cross assembler error messages

1,Bad constant

A constant used on the error line contains illegal characters or just doesn't make sense (or is undefined!).

Valid constants are any decimal, hex or binary number (16 bit).

Decimal numbers have no type specifier (eg 1, 324, 42, 0)

Valid characters are 1234567890

Hex numbers can have either a leading \$ or trailing H.

(eg \$AA, \$0AA, 0aaH 54H) Hexadecimal numbers with a trailing H which start with a letter symbol must have a leading 0 (0ffH, 0a5H etc)

Valid characters are 1234567890ABCDEF

Binary numbers can either have a leading % or trailing B.

(eg %111111111, %10101010, 01010101B etc)

Valid characters are 10

2,File error

A file specified could not be opened. This is due to :

The file not existing.

Not enough memory.

The file is already in use by some other software.

3,Unrecognised opcode

An opcode used on the error line is not recognised and doesn't exist.

4,Operand error

There is something wrong with the operand. Usually where the assembler is expecting a constant and a register name has been given or there is too little data given in the operand.

5, Out of range

A constant value is too large. This is also valid for branches and relative jumps. The relative address could be out of range for that particular instruction.

6, Bad code generated

Rare, there has been no syntax errors found in the line but for some reason the assembler has failed to produce decent code for that line. It has produced something but what it is isn't complete.

7, Use registers >15

You have specified a register below R16, don't.

8, Use registers 24, 26, 28 and 30

This opcode can only be used with these 4 registers.

9, Redeclared label

A label or an equate has been defined twice.

10, Relative address must be even

A branch to an address was to an odd address.

11, Code must be at an even address

All code must be word aligned.

12, Can't open file for inclusion

A file specified to be included with the 'include' directive couldn't. See file error (2)

13, Out of memory

There was not enough available memory to complete assembly.

14, Missing quote's

You have missed out a set of quote's (") on a line.

15, Division by zero.

You have tried to divide by zero.

16, Missing bracket.

You have missed a bracket out of an expression.

17, Formulae too complex.

The maths equation you have specified is too complex.

Usually just too large.

1.8 math

Assembler maths operators.

As usual any labels or numbers can contain mathematical expressions.
Supported math functions are.

+ Add two numbers
- Subtract
/ Divide
* Multiply
& Logical bitwise AND
| Logical bitwise OR
<< Logical shift left
>> Logical shift right

eg

5+5 equates to 10
10-5 equates to 5
20/10 equates to 2
2*6 equates to 12
3&1 equates to 1
1|2 equates to 3
2<<1 equates to 4
16>>2 equates to 4

The usual operator precedence applies in this order

/*+-&|<<>>

so $1+2*3$ will be processed as $1+(2*3) = 7$
but $(1+2)*3$ will result in 9 as expected.

1.9 req

Requirements.

The assembler should run on any system and doesn't require anything special to use it.

The only file that is required as well as the assembler is ATMEL.TXT
This contains the code definitions for the processor you are assembling for. Dont change it unless you know what you are doing. The supplied file is written for processors in the AT90S range. Extra commands used by other processors can easily be added to this file.

1.10 maccy

The assembler will use an internal table of AVR opcodes. If a file 'Atmel.txt' exists in the working directory, this will be used instead of the internal table.

The file which comes with this archive was used to generate the internal table and can be modified if desired!

1.11 bugs

Known bugs.

The assembler can overwrite code its already produced! example:

```
org 5
nop
org 5
dec R16
```

List file is even worse than V1.00 (But does anyone really need one ? ;)

To do

Macors (Bigger ones)

1.12 changes

Changes from last version

V1.04

- 1) Was not freeing about 200 bytes of memory correctly in V1.03
- 2) The file 'Atmel.txt' is not needed for the assembler to function. This has been removed and is now internal to the assembler.

V1.03

- 1) Constant strings used to have spaces removed.
- 2) Text strings with commas processed wrong.

V1.02

- 1) Bug fix 7 in V1.01 didn't work ! it does now.
- 2) If a constant was used which was not defined and the first letter was a H or B the assembler didn't complain and evaluated it to 0
- 3) Temp. buffer increased to allocate in 8k blocks (used to be 1k blocks)

V1.01

- 1) Removed NULL at end of constant strings
 - 2) Shift Left/Right (< >) is now as for C (<< >>)
 - 3) Removed temporary file, now done to a buffer.
 - 4) No upper limit on number of labels (except for avail RAM).
 - 5) 'def' added to allow small text replacements.
 - 6) Maths precedence is now /*+--&|<>
 - 7) Source files needed a new line at the end of the file or else the last line would not be assembled, fixed.
-

1.13 contact

So you want to contact me?

If you are experiencing problems with the assembler or have found a bug or think its producing duff code, let me know.

Ensure you have fully verified any bugs and have good evidence of the bug.

You can write to:

Loony Juice Software
49 Park Lane
Oswaldtwistle
LANCASHIRE
BB5 3AF
ENGLAND

email : lee@loonyjuicesoftware.freemove.co.uk

ta muchly.

Lee.
