Megaprocessor

--

Assembler User Guide

May 2016

James Newman

1. Introduction

The assembler for the Megaprocessor is a Windows application. It can be downloaded from www.megaprocessor.com. It runs in a command prompt window (DOS box) with the command line:

```
MPasm <root_name>
```

The source file should be root_name.asm. The assembler will produce two files:

- `root_name.hex` : the image including both code and data in Intel Hex format
- `root_name.lst` : a listing showing machine code produced

The .hex file can be loaded into the simulator (available at www.megaprocessor.com and also downloaded to the Megaprocessor itself.

Several example programs will have been included in the zip file for the assembler:

`opcodes.asm`	Example of all instructions and some of the assembler directives
`snail.asm`	Clears the internal ram and the runs a “snail” up and down it. (This was my first “substantive” program.)
`life.asm`	Implementation of John Conways “Game of Life”. Implementation of the video game. Uses the discrete RAM LEDs for a display.
`tetris.asm`	Implementation of the video game. Uses the discrete RAM LEDs for a display.
`tic_tac_toe_2.asm`	Implementation of tic-tac-toe (noughts and crosses). Uses the discrete RAM LEDs for a display.

Note that at time of writing (May 2016) these have only been run in the simulator, not yet on actual hardware.

2. Usage

a. comments

Comments are introduced with //. All following text to the end of the line is ignored. e.g.

```
    ld.w    r1,prev_key;
    cmp     r0,r1;
    bne     do_key_press;   // a different key

    // same key...is it time for auto repeat
    ld.w    r1,TIME_BLK_COUNTER;
```

b. labels

An assembler statement defining either code or data can start with a label which is indicated by following it with a colon character ":". e.g.

```
key_was_pressed:
    ld.w    r1,prev_key;
```

It takes the value of the address location at the start of the code/data defined.

c. constants

Constants can be defined using the EQU keyword. e.g.

```
// register locations for the GPIO...
GEN_IO_OUTPUT                   equ     GEN_IO_BASE + 0;
GEN_IO_INPUT                    equ     GEN_IO_BASE + 2;
GEN_IO_CTR                      equ     GEN_IO_BASE + 4;
```

d. Radix

By default numbers are treated as being decimal. To define a hexadecimal number prefix it with 0x, to define a binary number prefix it with 0b. e.g.

```
        db      1,2,3,4,0b1010;
        dw      21,500,0xdeadbeef,0xCAFe;
```

e. location

The current memory location can be accessed with the pseudo-variable $. e.g.

```
end_of_variables    equ $;
```

The current location can be changed to a new value using the org directive. e.g.

```
// tables and variables....
        org   0x10;
```

f. include

An assembler file can include another. For example :

```
// ***********************************
// Start with shared definitions...
include "Megaprocessor_defs.asm";

// ***********************************
```

g. data

There are several directives for creating space for variables. DB, DW, DL define space for bytes, words and longs respectively. If no initialisation value is provided then space for one variable is created. Several variables of a given size can be created by providing initialisation data for each using a comma separated list. The DM directive inserts a message (string) as a sequence of bytes. e.g.

```
        db      1,2,3,4,0b1010;
        dw      21,500,0xdeadbeef,0xCAFe;
        dl      0x12345678, 0xdeadbeef;
        dm      "Hi there";
```

produces the listing

```
1:  333 [0197] 00                       -       db;
1:  334 [0198] 01 02 03 04 0A           -       db      1,2,3,4,0b1010;
1:  335 [019D] 15 00 F4 01 EF BE FE CA  -       dw      21,500,0xdeadbeef,0xCAFe;
1:  336 [01A5] 78 56 34 12 EF BE AD DE  -       dl      0x12345678, 0xdeadbeef;
1:  337 [01AD] 48 69 20 74 68 65 72 65  -       dm      "Hi there";
```

Alternately an arbitrary amount of space can be allocated with the DS directive. An optional initialisation value (byte sized) may be provided.

```
        ds      10;     // allocate 10 bytes
        ds      20, 55; // 20 bytes filled with 55
```