# Megaprocessor

--

# Simulator User Guide

May 2016

James Newman

# 1. Introduction

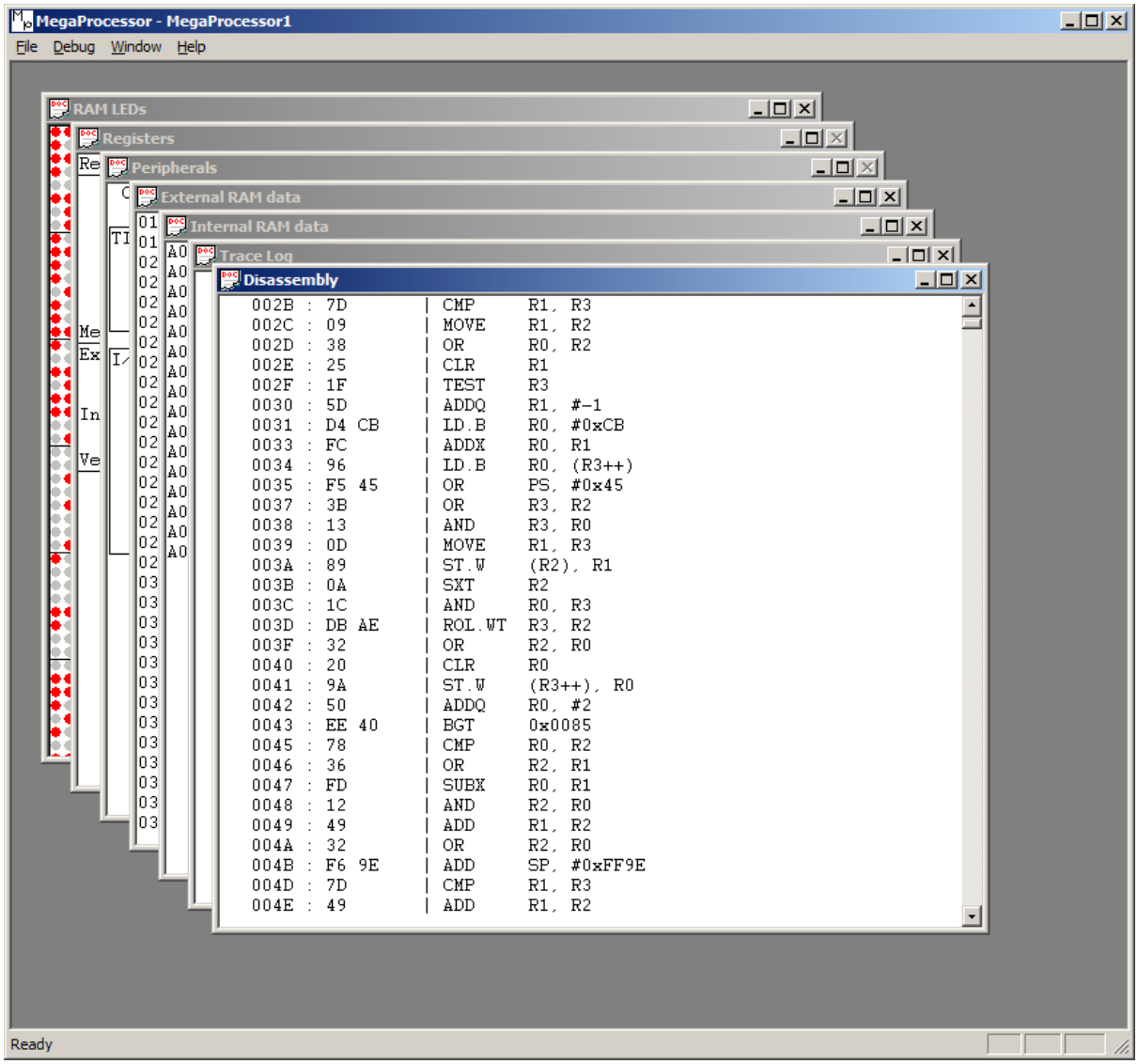
The simulator for the Megaprocessor is a Windows application It can be downloaded from www.megaprocessor.com. There is also an assembler and some example programs. Program images in Intel Hex format can be loaded and run. The simulation is cycle accurate. Some of the peripherals are emulated including memory, timers and GPIO. (Interrupts and the UART are not currently, May 2016, simulated). Breakpoints and single stepping are possible as well as free running.

TIP:

> The “Registers” and “Peripherals” windows are updated as each instruction is executed. This has a significant impact on speed of simulation. This can be reduced if those windows are minimised or closed.

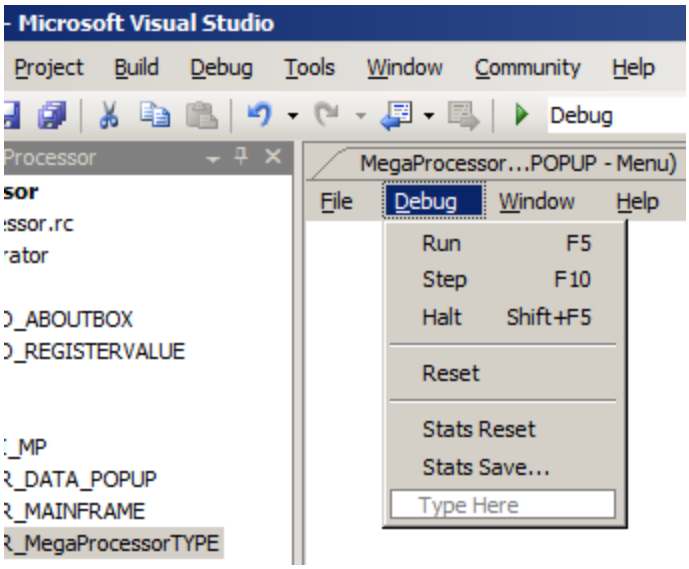
## 2. Instructions

On starting the simulator generates an initial random program and presents a set of windows which will look something like :




To simulate the execution of a particular program use File::Load. This will bring up a File Open dialog to find and select an Intel Hex format file (.hex extension) containing the image you wish to simulate. (This will have been created by the Assembler).

Loading a file will simulate a reset which will set the PC register to zero. The simulation is controlled through the debug menu :




Debug::Run will cause the simulation to execute instructions until either asked to halt, or a breakpoint is executed (see section on Disassembly window), or a memory violation occurs (see section on Registers window).

The simulation can also single step instructions using Debug::Step.

Debug::Reset simulates a reset. The PC is set to its vector base address and the simulated peripherals are also reset.

During simulation a count is kept of the number of times the processor executes an instruction at each address. The set of current counts can be saved to file using Debug::Stats Save. The counts can be cleared to zero using Debug::Stats Reset. This can be useful for performance profiling.

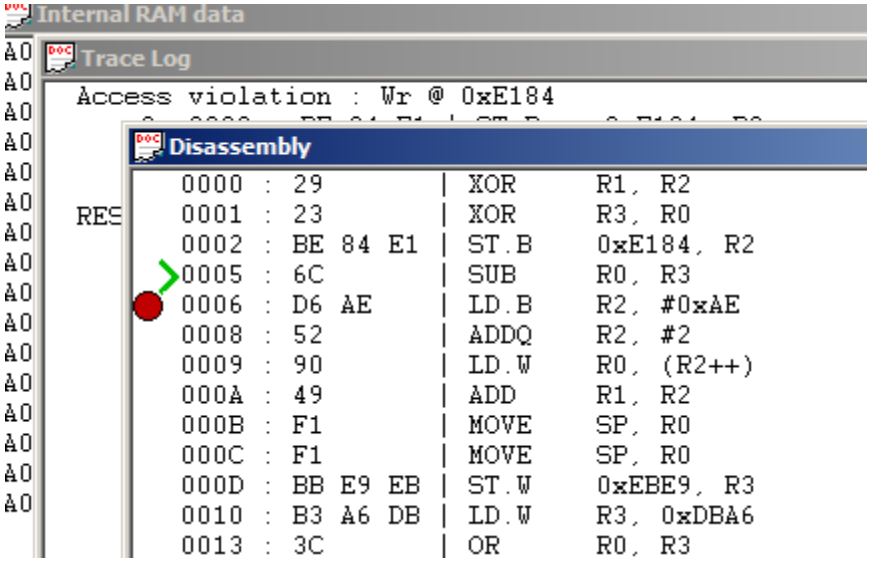
# 3. Windows

## a. Disassembly

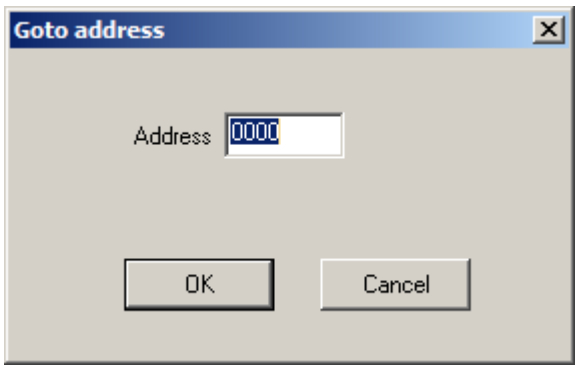
Each time an image is loaded the memory is disassembled. This disassembly can be viewed in a disassembly window.

If the instruction currently pointed to by the PC is in view it will be indicated by a green arrow.

Double clicking on a line will set a breakpoint on it. Double clicking on that line again will clear the breakpoint. Breakpoints are indicated by red circles.




Right clicking in the disassembly window will bring up a popup dialog:



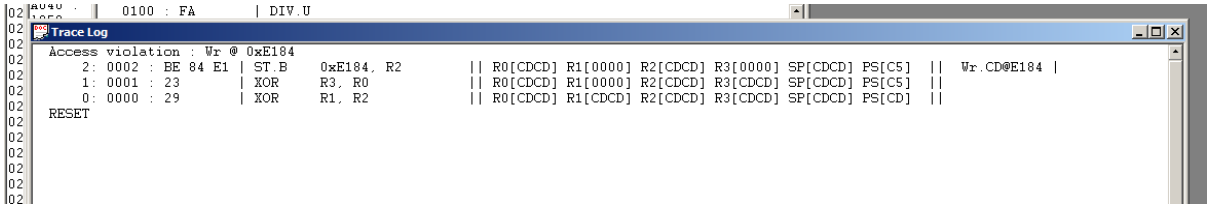

Entering an address (in hexadecimal) will cause the disassembly window to start displaying from that address.

NOTE: The disassembly listing is created when an image is loaded. If you mix code and data you may find that the disassembly is incorrect for the first one or two instructions located immediately after a data section. This is because the disassembler does not know what is what and treats everything as code. (The simulation will execute the correct instructions i.e. what is actually located at the address). This can be avoided by:

- not mixing code and data
- or  appending each data section with three NOP opcodes (0xFF).

## b. Trace Log

Each time an instruction is executed an entry is added to the trace log. These entries give the address and instruction executed as well as the values of the R0, R1, R2, R3, PS and SP registers. If a memory access occured then that is also shown. Entries are also added for a RESET event, and if a memory violation occurred (see section on registers window).

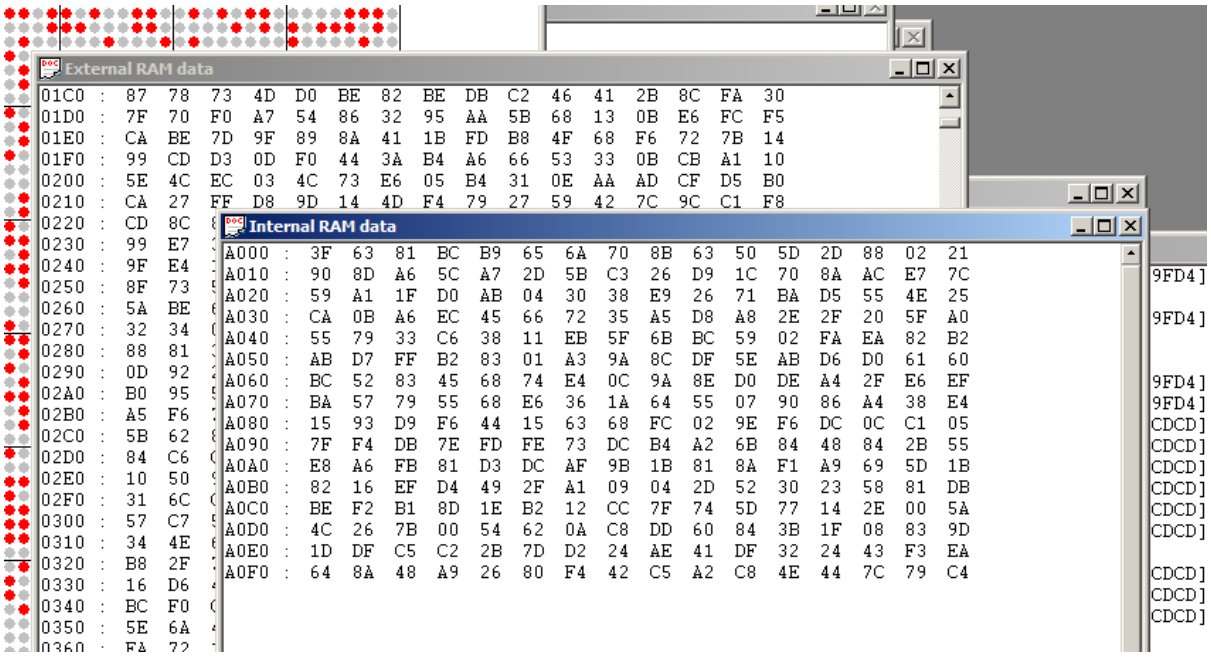



## c. External/Internal Ram data

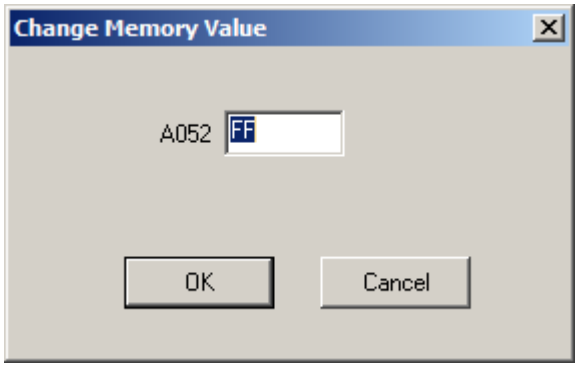
These are windows onto memory.

The Internal RAM window displays the contents of the 256 bytes of memory based at 0xA000 which corresponds to the Megaprocessor RAM built from discrete components.

The External RAM window displays the contents of the 32K bytes of memory based at 0x0000 which corresponds to the RAM built from a RAM chip.



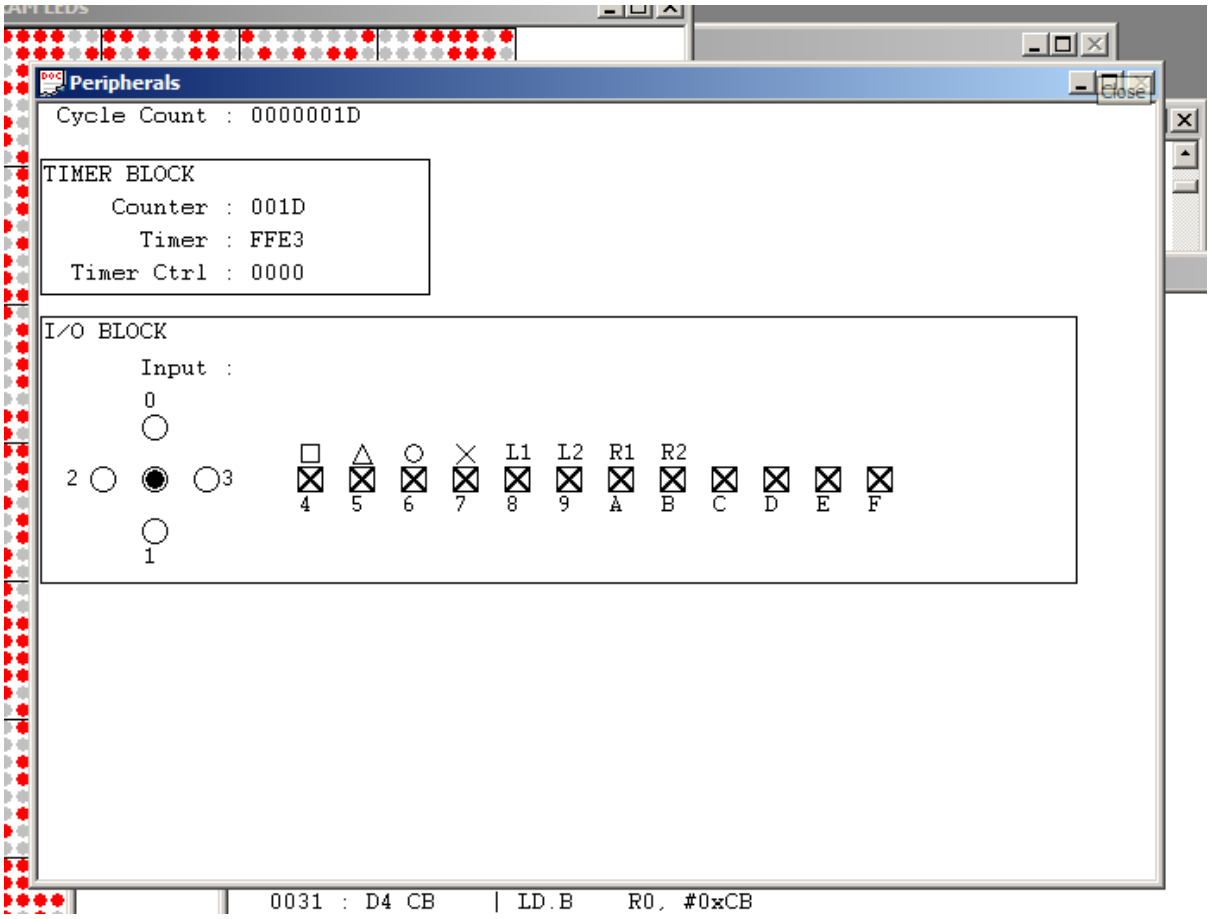

Double clicking on a byte value will bring up a dialog allowing you to change the value at that location (value in hexadecimal).




Right clicking will bring up the dialog to enter an address (in hexadecimal) for the window display to start from.

### d. Peripherals

This window allows the simulation of some of the peripherals. In particular I/O, counter and timer. Interrupts and the UART are not currently (May 2016) simulated.




The timer block simulates the counter and timer. These will update each cycle in the same way as the real hardware does. There is no interaction with these values through the GUI.

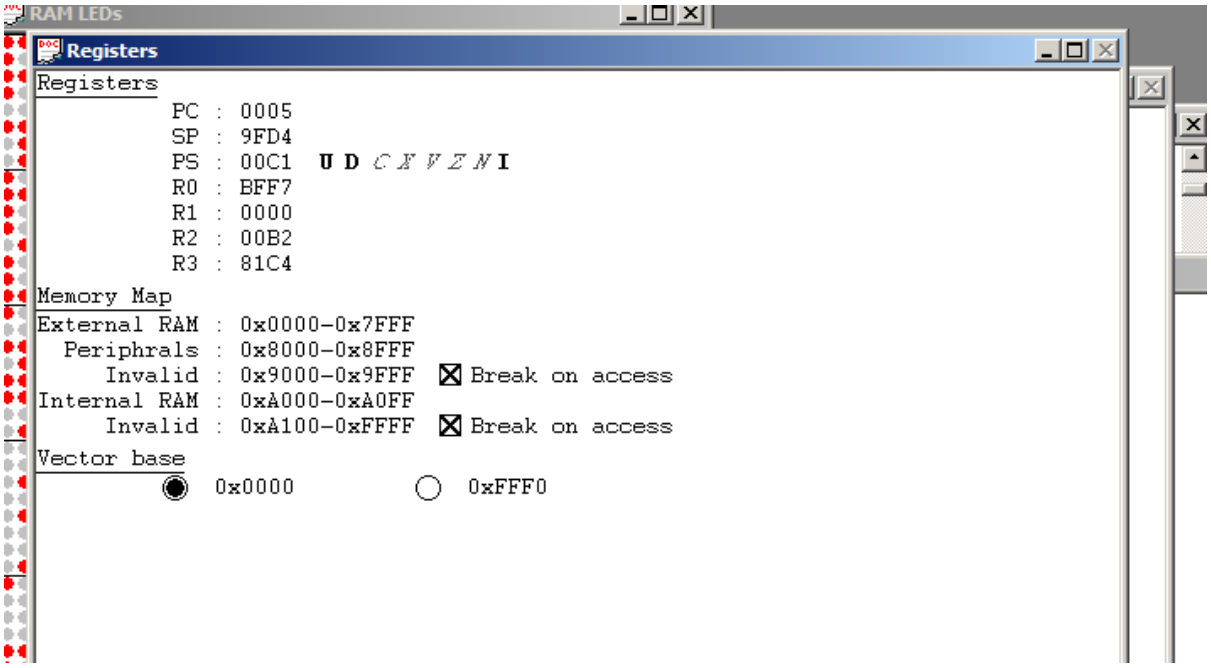
The I/O block simulates the modified Venom Arcade Stick attached to the input lines of the Control and I/O frame. (NOTE: on the real hardware there are pullup resistors on all of the input lines. Therefore the default value in real life that will be read is 0xFFFF, and that is simulated here). The

pattern of 5 radio buttons on the left represent the joystick. This is attached to the 4 LS input bits. Selecting one of the outer radio buttons will cause the associated bit to go LOW. Selecting the centre button represents the joystick being centred and all 4 simulated input bits will be HIGH. Bits 4..B represent the 8 buttons (and are labelled the same way). When the checkbox has a cross the bit will read HIGH, when there is no cross the bit will read LOW.


Cycle count is a count of the number of cycles since the last RESET. It does not represent any part of the hardware but is provided for debug. It's a 32 bit counter and so does not wrap as quickly as the simulated counter does.

### e. Registers

This window shows the values of the processor registers and also the memory map.



Double clicking on a register value will bring up a dialog allowing you to change it (new value entered in hexadecimal).

There are some parts of the memory map which in “real life” have no hardware. For debug the simulator can break if they are accessed. In real life the processor will read garbage, and writes will have no effect.

The Megaprocessor can locate its vector table at either 0x0000 or 0xFFF0 and the simulator provides a mechanism to control this. Until I build the ROM frame the only sensible option is 0x0000.

### f. RAM LEDs

This window shows the contents of the 256 byte RAM built from discrete components as a set of red dots. This is to emulate the appearance of the memory frame. Byte 0 bit 0 is top left (address 0xA000). Top right is byte 3 bit 7.