x86 MicroController Simulator

Contents

Who Should Use the Simulator... 2
Systems Architecture.. 4
Using the Simulator -Getting Started... 7
Tutorial 1 One - First Program. 2 + 2. 9
Tutorial 2 Two - Traffic Lights.. 12

Who Should Use the Simulator

The simulator is intended for any student studying low level programming, control or machine architecture for the first time.

The simulator can be used by students aged 14 to 16 to solve less complex problems such as controlling the traffic lights and snake.

More advanced students typically 16 or older can solve quite complex low level programming problems involving conditional jumps, procedures, software and hardware interrupts and Boolean logic. Although programs will be small, there is good scope for modular design and separation of code and data tables.

The simulator is suitable for courses such as

- An Introduction to Microprocessor Based Systems.
- Computer Systems.
- Electronics Courses.
- EdExcel GNVQ Low Level Programming Additional Unit 21.
- Courses involving microcontrollers.
- Courses involving control systems.
- Advanced Level Computing.

Description of the Simulator

In the shareware version the following instructions are not included. CALL, RET, INT and IRET. The hardware timer interrupt does not function because IRET can not be used either. The registered version includes these features. To Register click on this link.

This simulator emulates an eight bit CPU that is similar to the low eight bits of the 80x86 family of chips. 256 bytes of RAM are simulated. It is surprising how much can be done with only 256 bytes or RAM.

Features

8 bit CPU	16 Input Output ports. Not all are used.
Simulated peripherals on ports 0 to 5.	An assembler.
On-line help.	Single step through programs.
Interrupt 02 triggered by a hardware timer (simulated).	
Continuously run programs.	CPU Clock Speed can be altered.

Peripherals	Example Programs
Keyboard Input	99keyb.asm
Traffic Lights	99tlight.asm
Seven Segment Display	99sevseg.asm
Heater and Thermostat	99hon.asm 99hoff.asm
Snake and Maze	99snake.asm
Stepper Motor	99step.asm
Memory Mapped VDU	99keyb.asm

Documentation

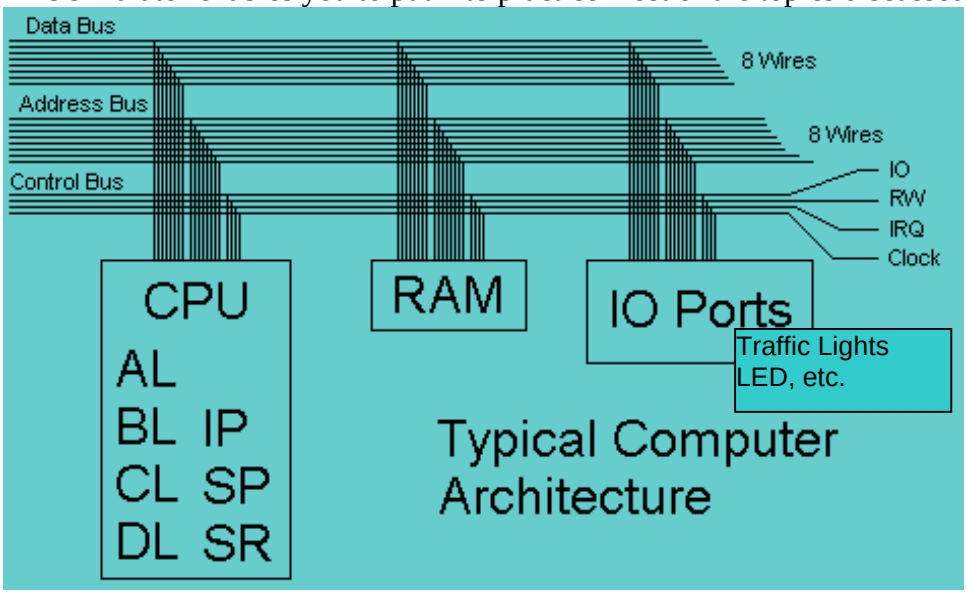
On-line hypertext help is stored in a Windows Help file. It is possible to copy from windows help and paste into word processor or text editor programs. The Rich Text Format source for the help file is also available. Many word processors including MS Word can load RTF files. Registered users have permission to modify help files for use by students and to print and or make multiple photocopies.

Disclaimer

This simulation software is not guaranteed in any way. It may differ from reality. It might not even work at all. Try it out and if you like it, please register.

Systems Architecture

This simulator enables you to put into practice most of the topics discussed below.



The simulator consists of a central processing unit (CPU), 256 bytes of random access memory (RAM) and 16 input output (IO) ports. Only five are used. There is a hardware timer that triggers interrupt 02 at regular time intervals that you can pre-set using the configuration tab.

The simulator is programmable in that you can run many different programs. In real life, the RAM would be replaced by read only memory (ROM) and the system would only ever run one program hard wired into the ROM. There are hundreds of examples of systems like this controlling traffic lights, CD players, simple games consoles, many children's games, TV remote controls, microwave oven timers, clock radios, car engine management systems, central heating controllers, environmental control systems and the list goes on.

The Central Processing Unit

The central processing unit is the "brain" of the computer. All calculations, decisions and data moves are made here. The CPU has storage locations called registers. It has an arithmetic and logic unit (ALU) where the processing is done. Data is taken from the registers, processed and results go back into the registers. Move (MOV) commands are used to transfer data between RAM locations and the registers. There are many instructions, each with a specific purpose. This collection is called the instruction set.

General Purpose Registers

The CPU has four general-purpose registers called AL, BL, CL and DL. These are eight bits or one byte wide. Registers can hold unsigned numbers in the range 0 to +255 and signed numbers in the range –128 to +127. These are used as temporary storage locations. Registers are used in preference to RAM locations because it takes a relatively long time to transfer data between RAM and the CPU. Faster computers generally have more CPU registers or memory on the CPU chip.

The registers are named AL, BL, CL and DL because the 16-bit version of this CPU has more registers called AH, BH, CH and DH. The 'L' means Low and the 'H' means High. These are the low and high ends of the 16-bit register.

Special Purpose Registers

The special purpose registers in the CPU are called IP, SR and SP.

IP is the Instruction pointer.

> This register contains the address of the instruction being executed. When execution is complete, IP is increased to point to the next instruction. Jump instructions alter the value of IP so the program flow jumps to a new position. CALL and INT also change the value stored in IP. In the RAM displays, the instruction pointer is highlighted red with yellow text.

SR is the Status Register.

> This register contains flags that report the CPU status.
> The 'Z' zero flag is set to one if a calculation gave a zero result.
> The 'S' sign flag is set to one if a calculation gave a negative result.
> The 'O' overflow flag is set if a result was too big to fit in a register.
> The 'I' interrupt is set if interrupts are enabled. See CLI and STI.

SP is the Stack Pointer.

> The stack is an area of memory organised using the LIFO last in first out rule. The stack pointer points to the next free stack location. The simulator stack starts at address BF just below the RAM used for the video display. The stack grows towards address zero. Data is pushed onto the stack to save it for later use. Data is popped off the stack when needed. The stack pointer SP keeps track of where to push or pop data items. In the RAM displays, the stack pointer is highlighted blue with yellow text.

Random Access Memory

The simulator has 256 bytes of ram. The addresses are from 0 to 255 in decimal numbers or from [00] to [FF] in hexadecimal. RAM addresses are usually given in square brackets such as [7C] where 7C is a hexadecimal number. Read [7C] as "the data stored at location 7C".

Busses

Busses are collections of wires used to carry signals around the computer. They are commonly printed as parallel tracks on circuit boards. Slots are sockets that enable cards to be connected to the system bus. An 8-bit computer typically has registers 8 bits wide and 8 wires in a bus. A 16-bit computer has 16 bit registers and 16 address and data wires and so on. The original IBM PC had 8 data wires and 20 address wires enabling one megabyte of RAM to be accessed. 32 bit registers and busses are now usual (1997).

The Data Bus is used to carry data between the CPU, RAM and IO ports. The simulator has an 8-bit data bus.

The Address Bus	is used to specify what RAM address or IO port should be used. The simulator has an 8-bit address bus.
The Control Bus	This has a wire to determine whether to access RAM or IO ports. It also has a wire to determine whether data is being read or written. The CPU reads data when it flows into the CPU. It writes data when it flows out of the CPU to RAM or the IO ports.
The System Clock	wire carries regular pulses so that all the electronic components can function at the correct times. Clock speeds between 100 and 200 million cycles per second are typical (1997). This is referred to as the clock speed in MHz or megahertz. The simulator runs in slow motion at about one instruction per second. This is adjustable over a small range.
Hardware Interrupts	require at least one wire. These enable the CPU to respond to events triggered by hardware such as printers running out of paper. The CPU processes some machine code in response to the interrupt. When finished, it continues with its original task. The IBM PC has 16 interrupts controlled by 4 wires.

Using the Simulator -Getting Started

On Line Help
Press F1 to get on line help.

Switch back and forth between the help system and the simulator using Alt+Tab.

While typing in your program, if you highlight an assembly code command like MOV or INT and you press F1, a help page will pop up with descriptions of the machine command. If you mis-spell a command and try to get help, you will see a list of all the available commands.

Use Alt+Tab to get back to the simulator.

Writing a Program
To write and run a program using the simulator, select the text editor by pressing

Alt+D

and type in the program. It is best to get small parts of the program working rather than typing it all in at once. To get started, look at the tutorial example programs. Base your first programs on these. You can switch backwards and forwards between help and your program by pressing Alt+Tab

Running a Program
To run a program, you can step through it one command at a time by pressing

Alt+P

repeatedly. You can run a program without single stepping by pressing

Alt+R or F9

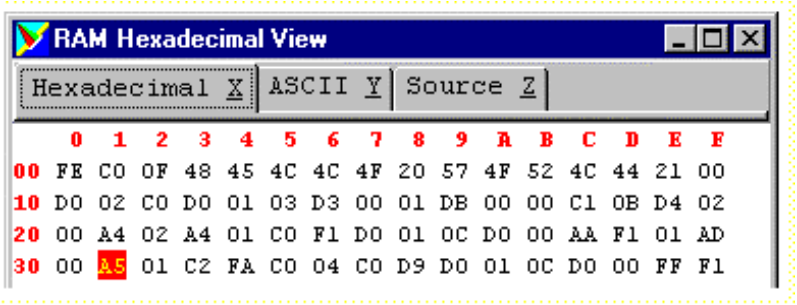
Assembly Code
The code you type is called assembly code. You can translate the assembly code into machine code understood by the CPU by pressing

Alt+A

If necessary, this is done automatically when you press Run or Step.

Viewing Machine Code

The machine code stored in RAM can be viewed in three modes by selecting the Tab of your choice.



The Hexadecimal display corresponds exactly to the binary executed by the CPU. The ASCII display is convenient if your program is processing text. The text is readable but the machine codes are not. The source code display shows how the assembly code commands are placed in memory. In normal use, these displays work quite well. It is possible to deliberately confuse the display.

For example MOV AL,15 translates into D0 00 15.

Instead of typing in the MOV command as above try this.

```
DB    D0
DB    00
DB    15
```

The same machine codes will be generated. The program will work in the same way. The displays will no longer recognise that it was a MOV command.

Tutorial Examples

The tutorial examples provide a step by step introduction to the commands and techniques of low level programming. Each program has one or more learning tasks associated with it. Some of the tasks are simple. Some are real brain teasers.

Tutorial 1 **One - First Program. 2 + 2**

Program uses
CLO, MOV, ADD, SUB, MUL, DIV and END

Click on the link below. When doing the learning exercises, add to and modify your own copy of the program. You can copy from the help page and paste into the program source code editor.

Here is the example program. 01first.asm (see below)

To run the program press the Step button repeatedly until the program ends. Watch the registers change as you step through the program. The altered registers at the top of the window are highlighted yellow.

What you need to know

Comments
Any text after a semicolon is not part of the program and is ignored by the simulator. These comments are used for explanations of what the program is doing. Good programmers make extensive use of comments. Good comments should not just repeat the code. Good comments should explain why things are begin done.

CLO
The CLO command is unique to this simulator. It closes any window that is not needed while a program is running. This command makes it easier to write nice demonstration programs. It also avoids having to close several windows manually.

MOV
The MOV command is short for Move. In this example numbers are being copied into registers where arithmetic can be done. Mov copies data from one location to another. The data in the original location is left intact by the mov command.

Registers
Registers are storage locations where 8 bit binary numbers are stored. The central processing unit in this simulator has four general purpose registers called AL, BL, CL and DL. These registers can, with a few exceptions, be used for any purpose.

Newer central processing unit (CPU) chips have 16, 32 or even 64 bit registers. These work in the same way but more data can be moved in one step so there is a speed advantage.

Wider registers can store larger integer (whole) numbers. This simplifies many programming tasks. The other three registers SP, IP and SR are described elsewhere.

Hexadecimal Numbers

In the command MOV AL,2 the 2 is a hexadecimal number. The hexadecimal number system is used in low level programming because there is a very convenient conversion between binary and hex. Study the Hexadecimal and Binary number systems.

Arithmetic

ADD is used to add two registers together. Another version of add is used to add a number to a register. You can get help on ADD, SUB, MUL and DIV by highlighting the command and pressing F1. This works both here and within the program source code editor.

END

The last command in a program should be END. Any text after the END keyword is ignored.

Your Task

Most of the example programs include a learning task. You should study the example and if you can complete the task, it is likely that your understanding of the example is good. In this case your task is to use all the registers AL, BL, CL and DL and to use ADD, SUB, MUL and DIV. Find out what happens if you try to divide by zero.

```
; _____ WORK OUT 2 PLUS 2 _____________________________________

        CLO             ; Close unwanted windows.

        MOV   AL,2      ; Copy a 2 into the AL register.
        MOV   BL,2      ; Copy a 2 into the BL register.
        ADD   AL,BL     ; Add AL to BL.  Answer goes into AL.

        END             ; Program ends

; _____ Program Ends ___________________________________________
```

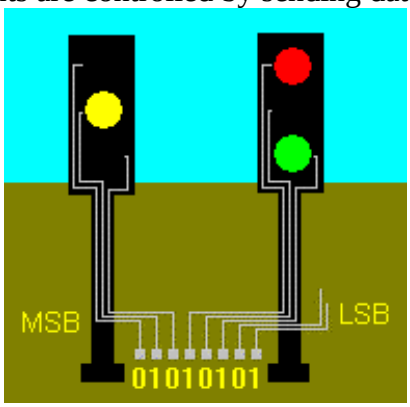
YOUR TASKs

1) Write a program that subtracts using SUB
2) Write a program that multiplies using MUL
3) Write a program that divides using DIV
4) Write a program that divides by zero. Make a note to avoid doing this in real life.

Tutorial 2 Two - Traffic Lights

Program uses
CLO, MOV, OUT, JMP and END

The traffic lights are controlled by sending data to input output (IO) port One.



There are six lamps to control. Red, Amber and Green for a pair of lights. This can be achieved with a single byte of data where two bits are unused.

By setting the correct bits to One, the correct lamps come on.

Here is the example program. 02tlight.asm (see below)

Click on the link above. Copy the program from the help page and paste it into the program source code editor.

To run the program press the Step button repeatedly or press the Run button.

To stop the program, press Stop. When the program is running, click the RAM-Source or RAM-Hex or RAM-ASCII tabs. These give alternative views of the contents of random access memory (RAM).

Also click the List File tab to see the machine code generated by the simulator.

What you need to know

Labels and the JMP command
Labels mark positions that are used by Jump commands. All the commands in this program are repeated for ever or until Stop is pressed. Label names must start with a letter or _ character. Label names must not start with a digit.

JMP Start
causes the program to jump back and re-do the earlier commands.

Destination labels end in a colon. For example

Start:

Controlling the Lights
If you look carefully at the traffic lights display, you can see which bit controls each light bulb. Work out the pattern or noughts and ones needed to turn on a sensible set of bulbs. Use the Hexadecimal and Binary numbers table to work out the hexadecimal equivalent. Move this hexadecimal number into AL.

OUT 01

This command copies the contents of the AL register to Output Port One. The traffic lights are connected to port one. A binary one causes a bulb to switch on. A nought causes it to turn off.

```
; _____ CONTROL THE TRAFFIC LIGHTS _____________________________

        CLO             ; Close unwanted windows.
Start:
                        ; Turn off all the traffic lights.
        MOV   AL,0      ; Copy 00000000 into the AL register.
        OUT   01        ; Send AL to Port One (The traffic lights).

                        ; Turn on all the traffic lights.
        MOV   AL,FC     ; Copy 11111100 into the AL register.

        OUT   01        ; Send AL to Port One (The traffic lights).

        JMP   Start     ; Jump back to the start.

        END             ; Program ends.

; _____ Program Ends ___________________________________________
```

YOUR TASK

5) Use the help page on hexadecimal and Binary numbers. Work out what hexadecimal numbers will activate the correct traffic lights. Modify the program to step the lights through a realistic sequence.

Checkout the other Tutorials Geezer!