Lab # 4

# OBJECTive

Identifies the organization of computer through Microprocessor Simulator

# THEORY

**TYPES OF PROGRAMMING LANGUAGE**

Programming languages can be classified into three basic categories on the basis of understanding level of users as well as the machine to which instructions has been given:

1. **HIGH LEVEL LANGUAGES**

A programming language that enables a programmer to write programs that are more or less independent of a particular type of computer and are designed to give a better program efficiency. Such languages are considered high-level because they are closer to human languages.

1. **LOW LEVEL LANGUAGES**

These are designed to have both: a relatively good programming efficiency and relatively good machine efficiency.

1. **MACHINE LANGUAGE**

Machine language is at the lowest level, because it is the actual binary code of 1s and 0s that the computer understands. These are designed to give a better machine efficiency.

REGISTERS CLASSIFICATION: The registers inside the microprocessor are classified according to the function they perform.

In general, they are classified as

1. General purpose registers
2. Address registers
3. Segment register
4. Offset registers
5. Status register

**SOME GENERAL PURPOSE REGISTERS:**

**AX (ACCUMULATOR REGISTER)**

* It is the preferred register to use in the arithmetic, logic and data transfer instructions because its use generates the shortest machine code.
* In multiplication and division operations, one of the numbers involved must be in AX or AL.
* Input and output operation also requires the use of AX and AL.

**BX (BASE REGISTER)**

* It is used to store the data also it serves as an address register.

**CX (COUNT REGISTER)**

* Program loop instructions are facilitated by the use of CX register, serves as a loop counter.
* Also used as a counter in the string operations.
* CL is used as count in instructions that shift and rotate bits.

**DX (DATA REGISTER)**

* It is used in multiplication and division operations.
* It is used in IO operation like DL in character output and DX in string output functions.

**REGISTER SIZE:**

* We have three different sizes of registers:
* 8-bit register: AH, AL, BH, BL, CH, CL, DH, DL
* 16-bit registers: AX, BX, CX, DX, SP, BP, SI, DI, SS, DS, CS, ES, FS, GS, IP, FLAGS
* 32-bit registers: EAX, EXB, ECX, EDX, ESI, EDI, ESP, EBP, EIP, and EFLAGS.

**BASIC MOV INSTRUCTION**

* The basic MOV instruction is used to transfer data between registers, between and memory locations, or to have a number directly to a register or memory location.

**Syntax:** **MOV Destination, Source**

**EXAMPLES:**

* MOV AH, BL ; 8-bits register to register
* MOV BX, AX ; 16-bits register to register
* MOV byte1, BL ; 8-bit register to memory
* MOV AX, word1 ;16-bit memory to register

**SOME ARITHMETIC INSTRUCTIONS**

* **ADD**: Add the contents of source operand1 to source operand 2 and result store in the source operand1.

**Syntax:** **ADD Source operand1, Source operand2**

**EXAMPLE: ADD AL, BL**

* **SUB**: Subtract the contents of source operand1 to source operand 2 and result store in the source operand1.

**Syntax:** **SUB Source operand1,Source operand2**

**EXAMPLE: SUB AL, BL**

**MICROPROCESSOR 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

* BTEC National Diploma for IT Practitioners (Computer Systems and Control Technology)
* AS and A2 Computing (Low Level Programming)
* Electronics Courses.
* Courses involving microcontrollers.
* Courses involving control systems.

# Description of the Simulator

The simulator includes a small sub-set of the full instruction set normally found with this style of processor. It includes advanced instructions such as CALL, RET, INT and IRET. There is a hardware timer interrupt simulation too.

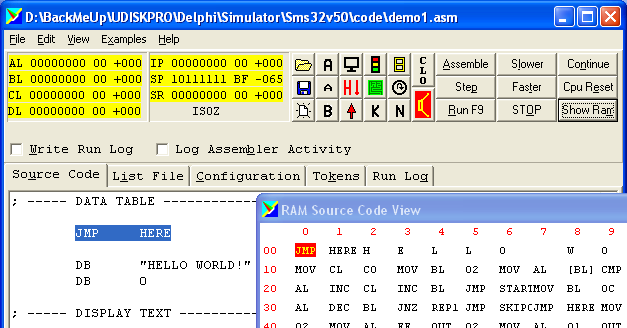
This simulator emulates an eight bit CPU that is similar to the low eight bits of the 80x86 family of chips. 256 bytes of RAM are simulated. It is surprising how much can be done with only 256 bytes or RAM.

The simulator is licensed under GNU / GPL making it freely available for use by students and educational institutions at zero cost.

# 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.
* Continuously run programs.
* Interrupt 02 triggered by a hardware timer (simulated).
* 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 and 99hoff.asm |
| Snake and Maze | 99snake.asm |
| Stepper Motor | 99step.asm |
| Memory Mapped VDU | 99keyb.asm |



|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**Sample Program**

To write and run a program using the simulator, select the source code editor tab by pressing **Alt+U**.

Type in your program. It is best to get small parts of the program working rather than typing it all in at once.

Here is a simple example. Also look at the tutorial example programs. You can type this into the simulator or copy and paste it. The assembly code has been annotated with comments that explain the code. These comments are ignored by the assembler program. Comments begin with a semicolon and continue to the end of the line.

|  |
| --- |
| MOV AL,0 ; Move 0 into the AL register    ADD AL,2 ; Add two to AL    END ; Program ends here |

### 

### Running a Program

|  |  |
| --- | --- |
| Step Button Image | To run a program, you can step through it one line at a time by pressing **Alt+P** or by clicking this button repeatedly. |
| Run Button Image | You can run a program continuously by pressing **F9** or **Alt+R** or by pressing this button |
| Faster and Slower Button Images | To speed up or slow down a running program use these buttons or type **Alt+L** or **Alt+T** |
| Stop Button Image | To stop a running program press **Alt+O** or click or press **Escape** or press this button. |
| Continue Button Image | To restart a paused program, continuing from where it left off, press **Alt+N** or click this button. |
| CPU Reset Button Image | To restart a program from the beginning, reset the CPU by pressing **Alt+E** or click this button. |
| Show Ram Button Image | To re-open the RAM display window, press **Alt+M** or click this button. |

### 

### Assembly Code

|  |  |
| --- | --- |
| Assemble Button Image | The code you type is called assembly code. This human-readable code is translated into machine code by the **Assembler.** The machine code (binary) is understood by the CPU. To assemble a program, press **Alt+A** or click this button. |
| Log Activity Button Image | You can see an animation of the assembler process by checking this box. |
| Run Button Image | When you run or setp a program, if necessary, the code is assembled. |

### 

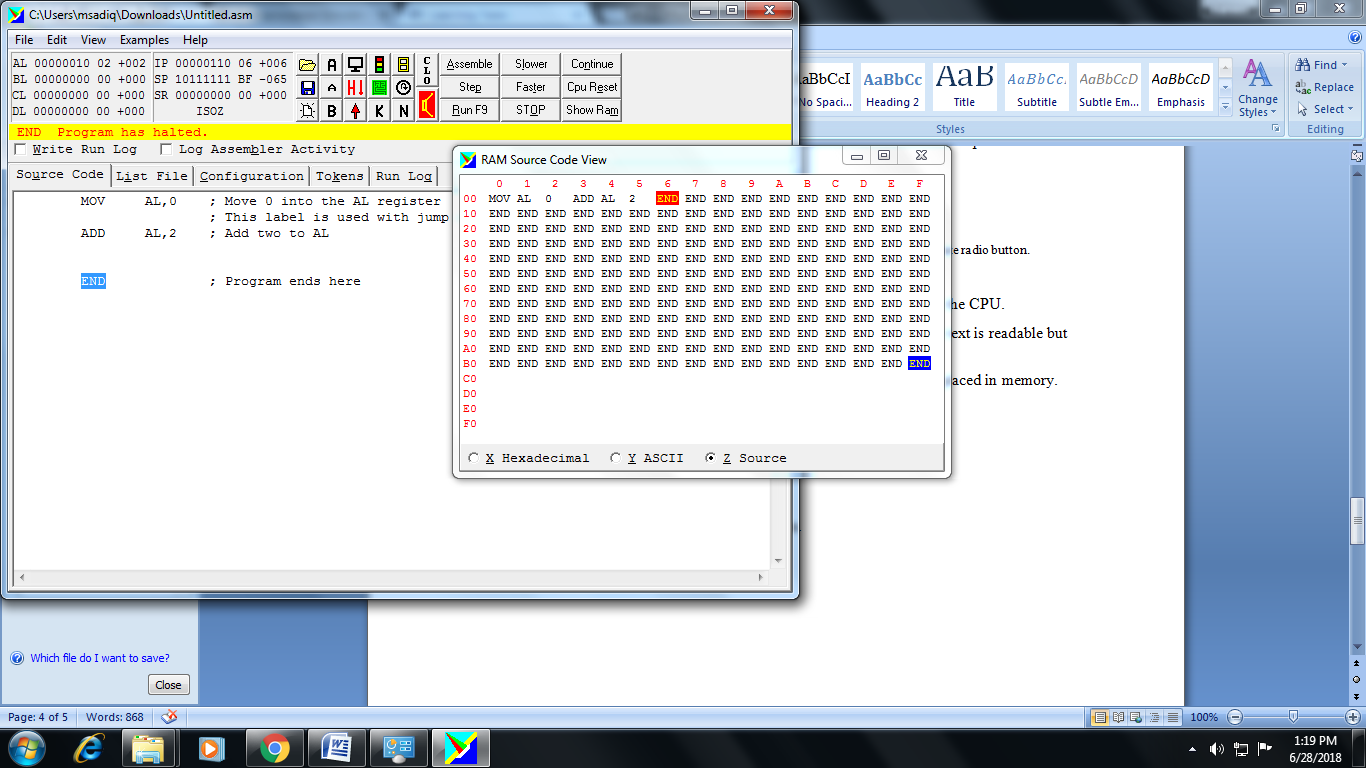
### Assembler Phases

There is short delay while the assembler goes through all the stages of assembling the program. The steps are

1. **Save** the source code.
2. Convert the source code into **tokens** (this simulator uses human readable tokens for educational value rather than efficiency).
3. **Parse** the source code and (if necessary) generate error messages. If there are no errors, generate the machine codes. This process could be coded more efficiently. If the tokens representing machine op codes like MOV and JMP were numerical, the assembler could look up the machine code equivalents in an array instead of plugging through many if-then-else statements. Once again, this has been done to demonstrate the process of assembling code for educational reasons.
4. **Calculate jumps**, the distances of the jump/branch instructions.

**Viewing Machine Code**

The machine code stored in RAM can be viewed in three modes by selecting the appropriate radio button.

**Hexadecimal** - This display corresponds exactly to the binary executed by the CPU.

**ASCII** - This display is convenient if your program is processing text. The text is readable but the machine codes are not.

**Source Code** - This display shows how the assembly code commands are placed in memory.

**LAB TASK**

1. Write a program that take 3 integers as hardcoded value and add them.