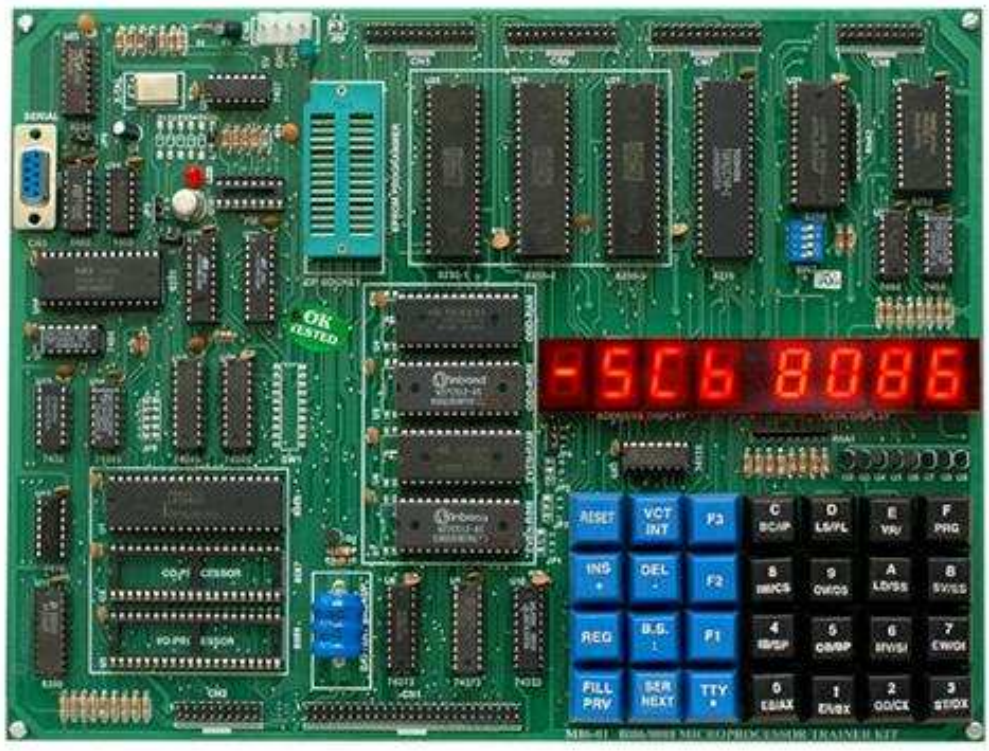
Haldia Institute of Technology
ICARE Complex, HIT Campus
P.O. HIT, Haldia, Pin – 721657
Dist. Purba Medinipur

Design and Implementation of 8085 Microprocessor Simulator



Project Report submitted to West Bengal University of Technology in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology

May, 2012

Submitted By,
Ashwani Kumar (08103001068)
Sourajyoti Datta (08103001076)
PranKrishna Dolai (08103001071)
Dept. of Computer Science & Engg.

HALDIA INSTITUTE OF TECHNOLOGY
ICARE Complex, HIT Campus
P.O. HIT, Haldia, Pin – 721657
Dist. Purba Medinipur

Design and Implementation of 8085 Microprocessor Simulator

Project Report submitted to West Bengal University of Technology in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology

May, 2012

Submitted by,

Ashwani Kumar (08103001068)
Sourajyoti Datta (08103001076)
PranKrishna Dolai (08103001071)
Dept. of Computer Science & Engg.

HALDIA INSTITUTE OF TECHNOLOGY
ICARE Complex, HIT Campus
P.O. HIT, Haldia, Pin – 721657
Dist. Purba Medinipur

Design and Implementation of 8085 Microprocessor Simulator

Project Report submitted to West Bengal University of Technology in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology under the supervision of Mrs. Gitika Maity

May, 2012

Submitted by,

Ashwani Kumar (08103001068)
Sourajyoti Datta (08103001076)
PranKrishna Dolai (08103001071)
Dept. of Computer Science & Engg.

ACKNOWLEDGEMENT

Before getting into the project work , we would like to add a few heartfelt words for the people who were a part of this project in numerous ways, people who gave unending support right from the stage the project idea was conceived.

First and foremost, we wish to thank Mrs. Gitika Maity, our project supervisor, whose inspiration, supervision, sincere appreciation and involvement has helped us to successfully carry on with our project.

We also express our thanks to all the teachers of the department for their kind co-operation and suggestions.

_____

Ashwani Kumar
(08103001068)

_____

Prankrishna Dolai
(08103001071)

_____

Sourajyoti Datta
(08103001076)

ABSTRACT

Microprocessors are generally very complicated integrated circuit chips which are a difficult subject to teach. It is an important topic in computer architectures, as it is the fundamental part of every computer system. Possibility of large calculations in a small amount of time, programming capabilities and the potential to communicate with other devices has increased the scale of applications the microprocessor has been used in. At the present time every intelligent electronic device such as a mobile phone or a navigation system does contain some kind of microprocessor. As a result of this every computer science student should know the general architecture of microprocessor and how it manages and processes data, and controls other parts of the system. Many methods for teaching microprocessors have been proposed, one important method is simulation. Cambridge Advanced Learner's Dictionary [2008] defines a simulation as-

"a model of a set of problems of events that can be used to teach someone how to do something, or the process of making such a model"

However, pure theoretical teaching of the subject does not fully clarifies what the student is expected to understand, and this is the time when simulation comes in, to bridge the gap between the theory and the real word. One of the most valuable features of simulation is that students have an opportunity to apply their theoretical knowledge in a safe realistic environment. Microprocessor simulation reveals real situations of inner microprocessor processes. The advantage of simulation is that user has a capability to graphically illustrate the whole process and display the internal state at any time of the execution. Additionally, instructions can be executed step-by-step method which stops the execution after each instruction, thus students have an opportunity to identify the changes that have been made after each step.

CONTENTS

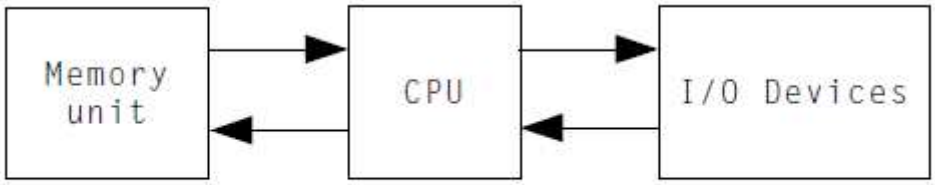
LIST of FIGURES

Chapter 1

INTRODUCTION

1.1 What is a Microprocessor?

A Microprocessor is the hearth of every computer based system. It processes all data within that system and controls all devices connected to it. Thus, it is also called the *central processing unit* (CPU). The system with a central processing unit, input/output device and memory storage can be visualized as in the following figure:



Computer based system

The CPU controls and processes all data sent by input devices such as a keyboard, mouse or thermometer. If for example, the temperature reaches the critical value, the CPU has to react to the new situation and perform required steps. In computer science these steps are called instructions and are stored in the memory unit. A sequence of instructions that performs the specified task is called a *computer program* (Stokes, 2007, p. 11). In the above example the CPU fetches the instruction specifically written to handle the critical temperature value from the memory unit. This instruction usually a set of instructions could for example increase the speed of a fan to reduce the temperature or simply activate an alarm.

1.2 What is Simulation?

Simulation is the imitation of the operation of a real-world process or system over time. The process of imitation always involves creating an artificial history which is then observed. The outcomes of the observation are then analyzed and used in the real system. Simulation is mostly used to describe the behavior of a model of a real or theoretical system (system to be implemented). The model then enables us to ask “what if” questions about the real operation of the system.

Simulation has been around for at least 2000 years. Board games such as Chess (in the West) and Go (in the East) were used as simulation of warfare for hundreds of years. However, simulation gains more importance in the beginning of 20th century, where it was mainly associated with the advent of flight simulation. In the new era where the first computers start to emerge, a computer based simulation arises alongside. Regardless of time, simulation has been one of the possible approaches to overcome a problem (e.g., plane crashes due to pilot inexperience or the need for improved decision making).

1.3 Why use Simulation?

Simulation is a very important part of research and system analysis. Five important advantages over other design techniques:

- *Simulation allows experimentation without disruptions to existing systems.* Implementing new ideas to an existing system may be costly or impossible to achieve. For example, in car manufacturing, where production is run constantly, implementing a new untested feature may cause production to disrupt.
- *A concept can be tested prior to installation.* Simulation testing may reveal unforeseen design flaws and allows the designer to improve the model.
- *Speed in analysis.* This is one of the most important advantages of simulation. After a model has been developed, it is possible to run the simulated system at speeds much higher or in some cases at lower speeds. For example, in a microprocessor moving data from one register to another is done in a fraction of microsecond. To animate this process, the simulation would be run at a much lower speed, so that the process can be captured by the user. In long-term processes, such as the earth revolving around the sun, the speed of simulation is increased and the simulation can be accomplished in a few seconds.
- *Force system definitions.* The development of simulation forces the designers to implement a fully defined model including all its expected parameters that are required to operate it. If the system has incomplete definitions, the output of the simulation will of course be incorrect and such a model should not be used as an analysis tool.
- *Enhances creativity.* Designing a model might reveal several possible solutions to a problem. One solution might be more expensive, but will work. The other one might require implementing a new technology and is less expensive compared to the first solution, but is somewhat risky.

Simulation is not a flawless approach which ensures that the system will behave correctly in every situation. It also has its limitations and disadvantages. Nevertheless, the number of advantages highly overcomes the number of disadvantages. Therefore, the list of disadvantages is considerably short:

- *Cost of development.* Models of complex systems are very difficult to create and often it is a very expensive method of analysis. In addition, software and in some cases, hardware has to be available or purchased. Personnel training learning curve expenses are also associated with the cost of development.
- *Time consuming.* Creation of a complicated model is time consuming and in most cases data collection, analysis and report generation will require considerable amount of time too.
- *Difficulty in interpreting results.* Because of the statistical nature of models, the simulation has to be run many times before it is possible to achieve reliable data.
- *Inappropriate use of modeling simulation.* Inappropriate use of modeling simulation when an analytical solution would be best suited.

1.4 Simulation in Educational Context.

Microprocessor architecture as a subject is taught at an undergraduate level at most universities in the UK. The content of the subject is primarily focused on computer hardware architecture and low-level programming. In practice it is a difficult subject to teach as it covers a large area from basic logic gates to complex digital circuits. Prior to the microprocessor architecture the student should have a working knowledge of digital circuits (taught in electronics courses), some background in understanding semiconductors and a knowledge of at least one high-level programming language is beneficial. A typical undergraduate course syllabus would include:

- Computer structure – processor, memory, I/O, buses, clock.
- Data representation – binary and hex.
- Memory system – principles of memory hierarchy, caches, main and virtual memory.
- CPU – simple internal structure, registers, arithmetic logic unit.
- Instruction sets – syntax, semantic, addressing modes.
- Data structure – stack machine.
- Basic processor implementation – basic steps of execution, interrupts.

At first sight a microprocessor may be seen by a student as a black-box whose contents is far too complex to understand. Teaching the subject properly in detail requires many hardware and software tools. The increase in use of microprocessor for the sole purpose of monitoring and

control has brought new trends in microprocessor education. These types of microprocessors are called *microcontrollers* (MCU). Microcontrollers have been widely adopted by most electrical engineering departments as a teaching tool for its relatively low price but the drawback of these tools is their simplicity and hardware limitations. MCU typically contains a processor code, memory and programmable input/output interface.

On the other hand there are microprocessor development systems which are standalone devices that can perform more complex operations in contrast to basic microcontrollers but the disadvantage is the high price. As a result students are then usually assigned to a group and the equipment is shared which shortens the time for each of the students to experiment with the device.

Chapter 2

BACKGROUND

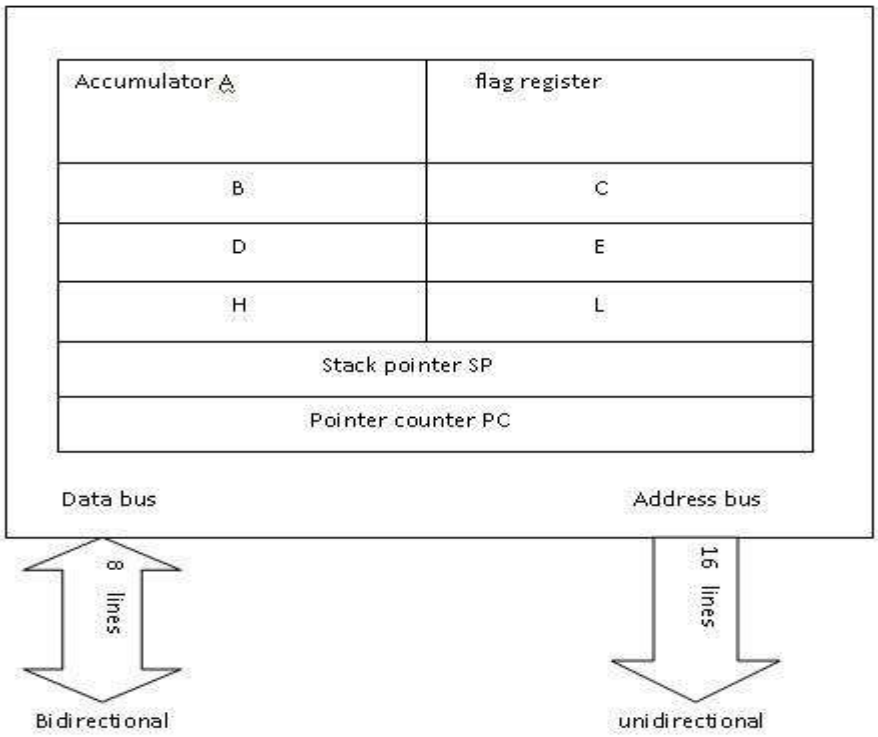
2.1 Introduction to the 8085 microprocessor

The Intel 8085 is an 8-bit microprocessor introduced by Intel in 1977. The 8085 is a conventional von Neumann design based on the Intel 8080.

7 main components of 8085 microprocessor:

1. **ALU -** The ALU performs many of the function that involves arithmetic and logic operations. The Arithmetic unit also handles all data manipulation, such as shift left/right, rotate, and the 2's complement operations.
2. **Timing & Control Unit -** The timing & control unit of the 8085 is responsible for the timing of all components (within and outside the microprocessor), clock input, input and output traffic flow on all the busses.
3. **Instruction Register & Decoder -** When an instruction is fetched from memory, it is loaded in the instruction register. The decoder decodes the instruction and establishes the sequence of events to follow. The instruction register is not programmable and cannot be accessed through any instructions.
4. **Register Array -** Data registers are locations where data is stored temporarily within the microprocessor. These are a larger number of registers in the 8085, each with a specific function. Register are flip-flops configured as memory elements.
5. **Interrupt Control -** This control interrupts a process. Whenever the interrupt signal is enabled or requested the microprocessor shifts the control from main program to process the incoming request and after the completion of request, the control goes back to the main program.
6. **Serial I/O Control -** The input and output of serial data can be carried out using 2 instructions in 8085, SID (Serial Input Data) and SOD (Serial Output Data). Two more instructions are used to perform serial-parallel conversion for serial I/O devices, SIM and RIM.
7. **Internal BUS -** There are three types of internal bus, Address bus (16-bit Unidirectional), Data bus (8-bit Bidirectional), and the Control bus (6-bit). The address bus sends address to the memory, the data bus sends data from the ALU to the memory and vice-versa and the control bus is for communication with the peripherals.

2.2 The 8085 Programming Model



Memory

Program memory - program can be located anywhere in memory. Jump, branch and call instructions use 16-bit addresses, i.e. they can be used to jump/branch anywhere within 64 KB. All jump/branch instructions use absolute addressing.

Data memory - the data can be placed anywhere as the 8085 processor always uses 16-bit addresses.

Stack memory is limited only by the size of memory. Stack grows downward.

First 64 bytes in a zero memory page should be reserved for vectors used by RST instructions.

Interrupts

The 8085 microprocessor has 5 interrupts. They are presented below in the order of their priority (from lowest to highest):

INTR is maskable 8080A compatible interrupt. When the interrupt occurs the processor fetches from the bus one instruction, usually one of these instructions:

- One of the 8 RST instructions (RST0 - RST7). The processor saves current program counter into stack and branches to memory location N * 8 (where N is a 3-bit number from 0 to 7 supplied with the RST instruction).

- CALL instruction (3 byte instruction). The processor calls the subroutine, address of which is specified in the second and third bytes of the instruction.

RST5.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 2Ch (hexadecimal) address.

RST6.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 34h (hexadecimal) address.

RST7.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 3Ch (hexadecimal) address.

Trap is a non-maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 24h (hexadecimal) address.

All maskable interrupts can be enabled or disabled using EI and DI instructions. RST 5.5, RST6.5 and RST7.5 interrupts can be enabled or disabled individually using SIM instruction.

Registers

Accumulator or A register is an 8-bit register used for arithmetic, logic, I/O and load/store operations.

Flag is an 8-bit register containing 5 1-bit flags:

- Sign - set if the most significant bit of the result is set.
- Zero - set if the result is zero.
- Auxiliary carry - set if there was a carry out from bit 3 to bit 4 of the result.
- Parity - set if the parity (the number of set bits in the result) is even.
- Carry - set if there was a carry during addition, or borrow during subtraction/comparison.

General registers:

- 8-bit B and 8-bit C registers can be used as one 16-bit BC register pair. When used as a pair the C register contains low-order byte. Some instructions may use BC register as a data pointer.
- 8-bit D and 8-bit E registers can be used as one 16-bit DE register pair. When used as a pair the E register contains low-order byte. Some instructions may use DE register as a data pointer.
- 8-bit H and 8-bit L registers can be used as one 16-bit HL register pair. When used as a pair the L register contains low-order byte. HL register usually contains a data pointer used to reference memory addresses.

Stack pointer is a 16 bit register. This register is always incremented/decremented by 2.

Program counter is a 16-bit register.

Instruction Set

Instruction set of Intel 8085 microprocessor consists of the following instructions:

- Data moving instructions.
- Arithmetic - add, subtract, increment and decrement.
- Logic - AND, OR, XOR and rotate.
- Control transfer - conditional, unconditional, call subroutine, return and restarts.
- Input/Output instructions.
- Other - setting/clearing flag bits, enabling/disabling interrupts, stack operations, etc.

Addressing Modes

Register Direct - references the data in a register or in a register pair.

Register indirect - instruction specifies register pair containing address, where the data is located.

Immediate - 8 or 16-bit data.

I/O Ports

The 8085 supports up to 256 Input ports and 256 Output ports.

Chapter 3

REQUIREMENTS ANALYSIS

3.1 Problem Definition

The 8085 microprocessor Simulator is a standalone application which enables users to test their executable machine codes on the system, and compare the results of the execution. Sometimes it is not convenient to bring any machine or any object to the place where it is needed, so here comes the need of Simulator. We will prepare a Simulator which will work almost same as the machine which is required and it is prepared in the least amount of resources, where we don't need all the functions of the machine and only a few functions are needed, then instead of bringing that whole machine which might be costly and inconvenient to bring or arrange. Hence, we will prepare a smaller machine in the least possible resources so that it can provide us with all the functions that we need.

We selected Core Java for the development of the functional units of this application, as it is highly platform independent, and provides support for a large number of libraries and APIs.
Java Swing has been used for development of the user interface. Java Swing Framework, unlike java AWT, is platform independent, and hence provides a solid platform for GUI development.

3.2 Functional Requirements

The functional requirements of the application are as follows:

F-01: Memory edit by user

INPUT:	Memory address, Data Value
PROCESS:	Saves the data into the memory
OUTPUT:	Display memory in table
VALIDATION:	Check whether the data and memory are in hex, and are within the limits.

F-02: Memory view by user

INPUT:	Starting memory address
PROCESS:	Shows 256 data locations starting from starting address
OUTPUT:	Display memory in table
VALIDATION:	Check whether the memory address is in hex, and is within the limits.

F-03: Execute program by user

INPUT:	Start memory address
PROCESS:	Sequential execution of the program starting at the starting address is done.
OUTPUT:	Display all the register and flag values after completion.
VALIDATION:	Check whether the start memory address is in hex, and is within the limits.

F-04: Save program to file

INPUT:	File name
PROCESS:	Saves the entire memory into the file
OUTPUT:	A new file is created where the memory location data are stored.
VALIDATION:	Check for correct filenames.

F-05: Load program from file

INPUT:	File name
PROCESS:	Loads the entire memory with data from the saved file
OUTPUT:	The file is read, and the memory is filled with the data from the opened file.
VALIDATION:	Check for correct filenames. Check for correct content of the loaded file.

Chapter 4

DESIGN and ANALYSIS

4.1 Introduction

This chapter includes a short introduction into the Incremental and Iterative development model. Then the overall design and functionality of the microprocessor simulator is described. The subsections are devoted to detailed design and analysis of individual parts of the microprocessor simulator. The last part describes the design of the user interface.

The project's design is split into two major sections:

Simulator Design

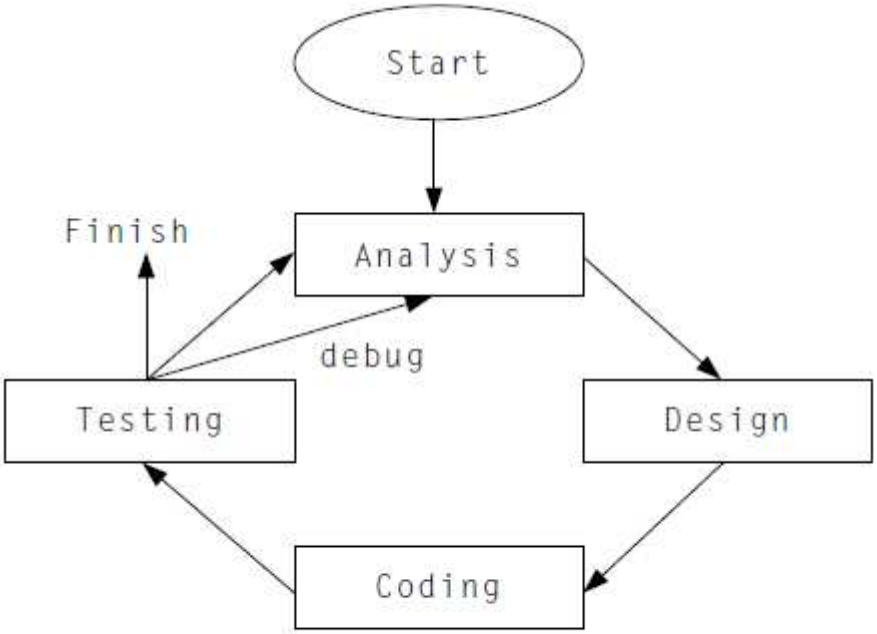
- Microprocessor
- Memory
- Stack
- Instruction set (Opcodes)
- I/O file

User Interface

- Graphical User Interface

4.2 Iterative & Incremental Model

The Incremental development model is a cyclic software development process. The main idea is to develop a system through repeated cycles (iterative) in small portions (incremental).
Development in this model starts with a developing small version of the program. The purpose of this is to have a working program even though it does not have all the required functions, which are developed gradually through cycles. At any time a working program can be presented to the customer for feedback. This has a beneficial affect on the overall development as alteration can be done immediately without a need for altering a huge amount of code.



Iterative & Incremental development model

The model is divided into four major stages:

- *Analysis* – decision over changes to be implemented in the next stage. Only small portions are implemented at a time.
- *Design* – changes that need to be done before a new feature can be coded if any.
- *Coding* – compiling the code to make sure that the coding is correct. The code is often recompiled after writing a few lines of code.
- *Testing* – involves running the program by using test data if it is requested. If an error occurs that the system has not recognised (typically memory leaks), the program must be debugged and either coding or design should be altered. However, small portions are implemented through a cycle. It is then very easy to define the source of the problem.

The development continues around this loop until the program is finished. The Iterative & Incremental approach means that there is a running program at the end of each iteration. It obviously does not contain all the features that have been proposed to be implemented. This gives an opportunity to evaluate the program after each cycle.

Advantages

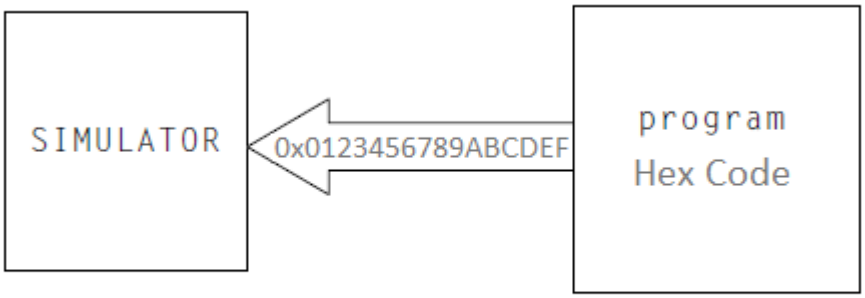
- Generates working software quickly and early during the software life cycle.
- More flexible – lost costly to change
- Easier to test and debug during a smaller iteration.
- Every iteration is an easily managed milestone.

Disadvantages

- Each phase of an iteration is rigid and do not overlap each other.
- Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.
- Can be a costly model to use.
- Project's success is highly dependent on the risk analysis phase.

4.3 Simulator Design

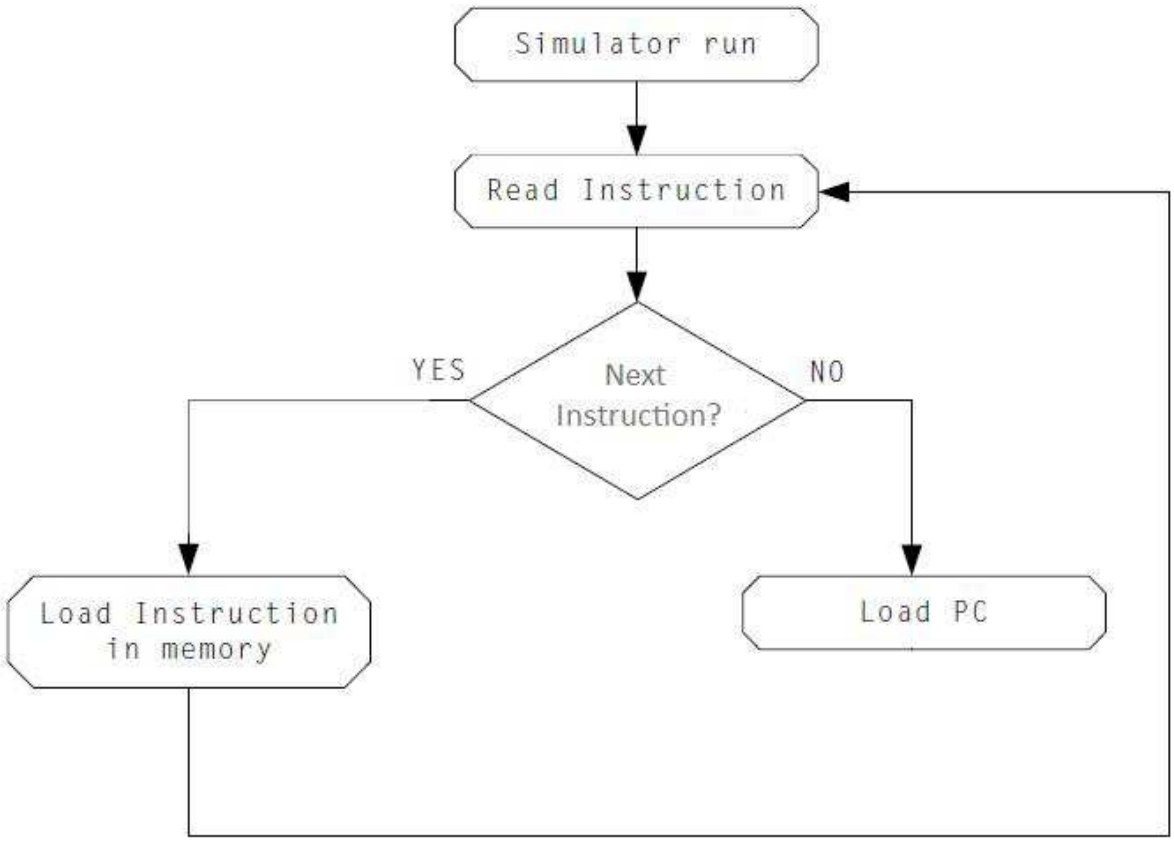
The simulator design consists of designing the overall functionality of the simulator, the way it input file is read. The simulator recognizes the file format and parses the instructions to the microprocessor. The high level architecture of the operation of the simulator:



High-level operation of the simulator

It is important to ascertain the two terms, "microprocessor" and "simulator". The term simulator refers to the implemented software that reads the input file or memory, and handles the user input commands. Whereas the microprocessor is a part of the simulator, the simulator passes the instructions to the microprocessor which then moves them to the memory and executes.

The simulator accepts a hex file which contains the hex op-codes executable on the microprocessor. The hex file is not a real file. It is a common text file that contains strings of hexadecimal opcodes. The file is read by the simulator and split into groups of 8 bits. Each group is then specifically assigned memory space starting from the address 0x0000h. When the terminating instruction is encountered the PC register is loaded with the address of the first instruction (0x0000) and the simulation starts.



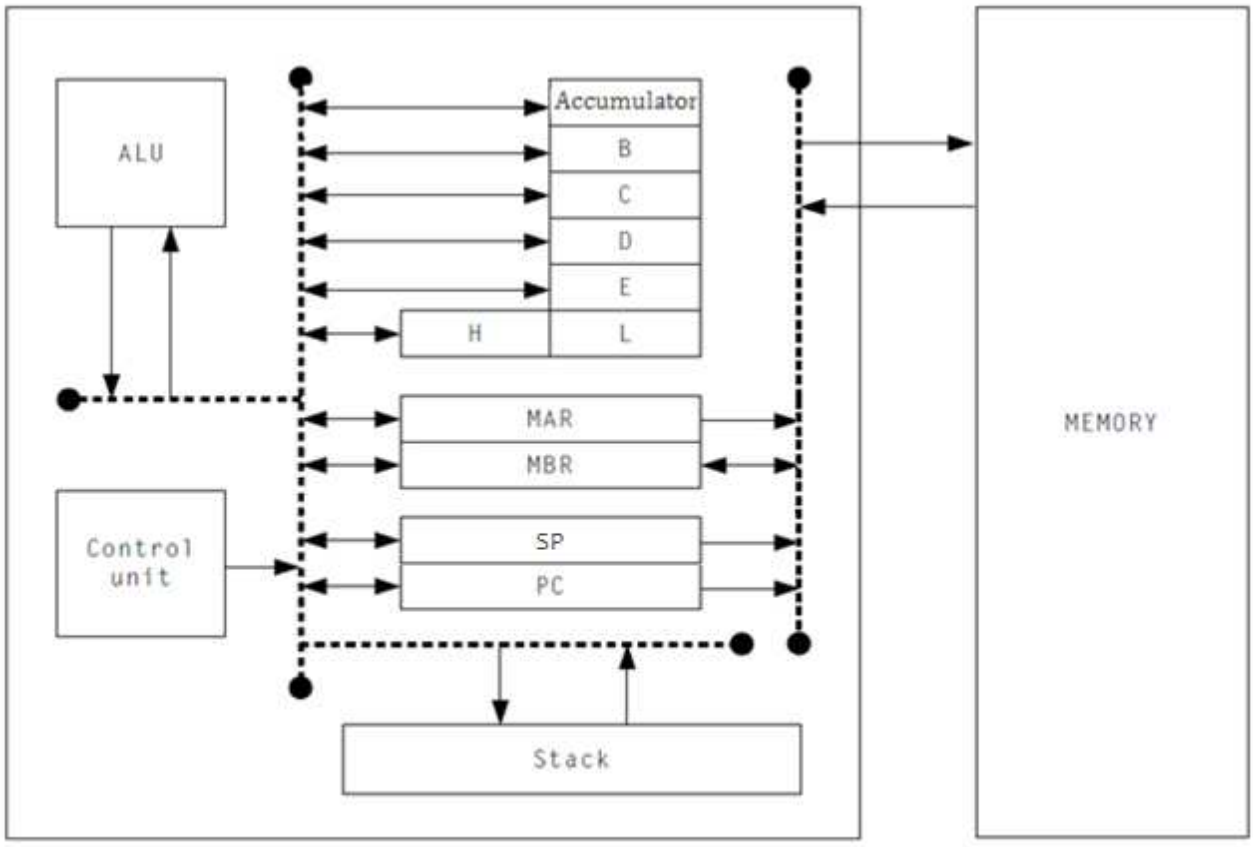
Procedure of loading instruction into memory

Instruction execution starts with reading the content of the address stored in the PC register (address of the first program instruction). In the next step the microprocessor decodes and recognizes the instruction. If the instruction is not the terminating instruction, the instruction is executed. An instruction can be made out of many microinstructions and however, each microinstruction is executed sequentially. After the instruction has been executed the PC is incremented and points to a new memory address where another instruction is stored. The process repeats until the terminating instruction has been reached. At the end a small report of states of all registers and specific memory address can be printed to the screen.

4.3.1 Microprocessor Design

The microprocessor consistes of these parts:

- Control Unit
- Registers
- Arithmetic Logic Unit
- Instruction Set
- Memory and Stack



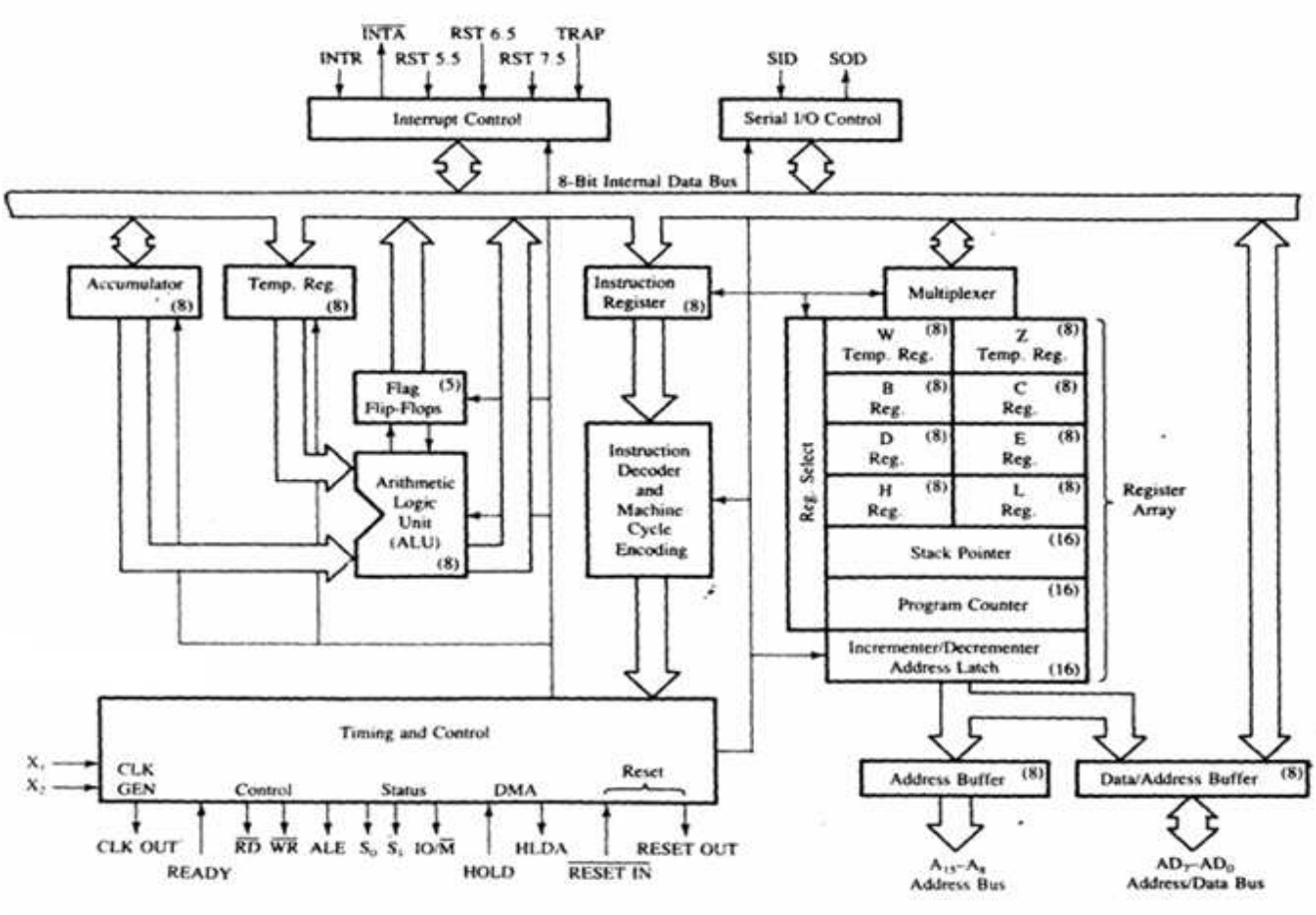
Block diagram of the microprocessor

The microprocessor has 6 general purpose registers B, C, D, E, H & L. An 8-bit accumulator register A is used to store the results of arithmetic and logic operations as no other register is able to do so.

Special purpose registers L and H are used to store 16-bit memory addresses. Registers MAR (Memory Address Register) and MBR (Memory Buffer Register) are associated with memory to store a memory address and memory data. These register are not directly accessible to the user but can be indirectly access only through the register H and L. The register PC (Program Counter) is used to store the instruction program pointer which points to a memory location with the program instructions. The user cannot directly alter the pointer and can only to do so through the control flow instructions. The microprocessor also contains a stack which is another way of storing and processing data compared to accumulator and register based machine.

4.3.1.1 Arithmetic Logic Unit

ALU is a component of microprocessor that performs arithmetic and logic operations.



Arithmetic Logic Unit architecture

The ALU contains two temporary registers (W and Z, both 8-bit) where the operands for the intended operation are stored. When an operation is performed both operands are read from the temporary registers and the operation is executed. The result of an operation is transferred to the internal data bus to the Accumulator register.

4.3.1.2 Register Design

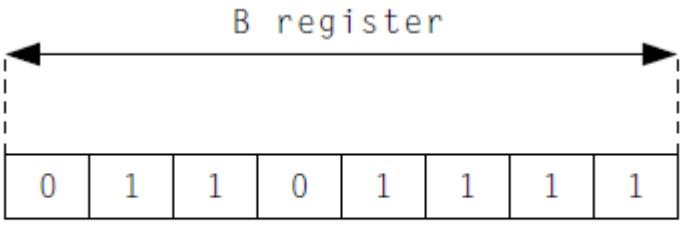
Registers are used to temporarily store information in binary format. The information stored in registers is then prepared to be processed by the microprocessor. There are different types of registers, general purpose and special registers. Special registers have an additional function for example may be used when accessing external memory or results of arithmetic operations are stored in the accumulator.

Requirements

- 8-bit register size
- Write protection
- Internally the value is stored in binary format
- Register able to load and retrieve values

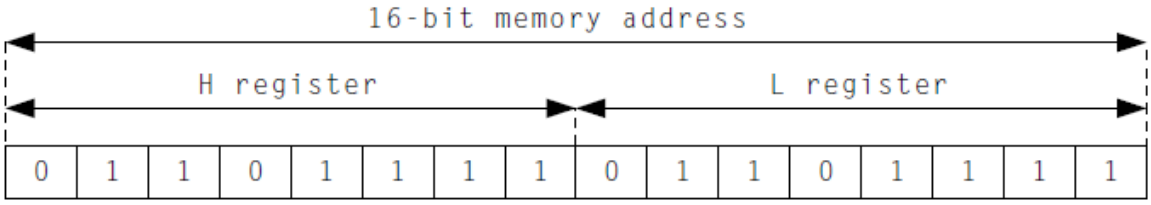
Design

All registers are 8-bit wide and thus the biggest possible number that can be stored is 255 or 0xFF. Write protection bit must be set prior to writing a new value to the register.



8-bit register diagram

16-bit register is constructed out of two 8-bit registers. For example, the registers H and L, which together are capable of storing a 16-bit value. When the microprocessor accesses the memory, H stands for higher 8-bits and L for lower 8-bits of the memory address.



16-bit register diagram

4.3.1.3 Instruction Set Design

Instruction set defines the operations that the microprocessor is able to perform. The design of the instruction set is mainly based upon the 8085 microprocessor instruction set.

Requirements

- Arithmetic and logical operations
- Register and memory transfer operations
- Control flow instructions

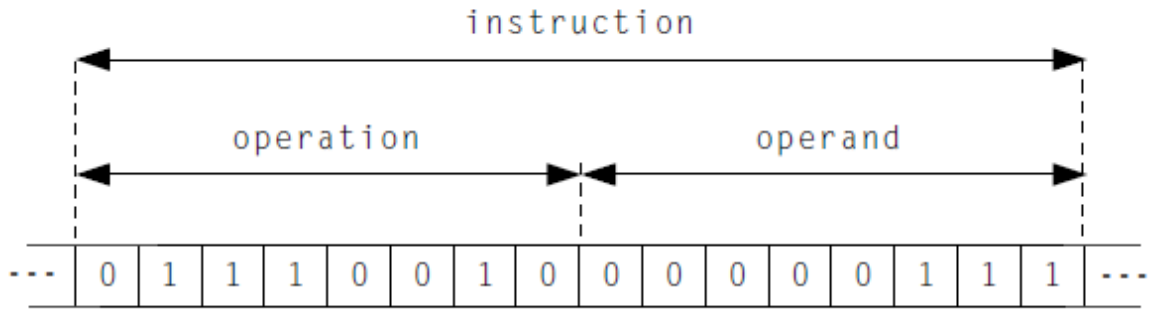
- Input/output instructions
- Others - set/clear flag bits, enable/disable interrupts, stack operations, etc.

Design

Instruction set design consists of defining:

- The size of instruction
- Operations

Each instruction consists of two parts: operation and operand. The length of the instruction is 16 bits, where the first 8 bits represent the operation and the lower 8 bits the operand if it is required. Since 8 bits are available to distinguish the operation, there is a maximum number of different operations which is 256. This number of instructions is large enough to accommodate all the planned instructions.



Instruction Design (2-byte instruction)

For example, in the above figure, the operation is $(01110010)_2$ which is a register transfer operation that moves the operand $(00000111)_2$ into A register. In symbolic form the operation would be written as:

A ← direct value

And in the assembly language:

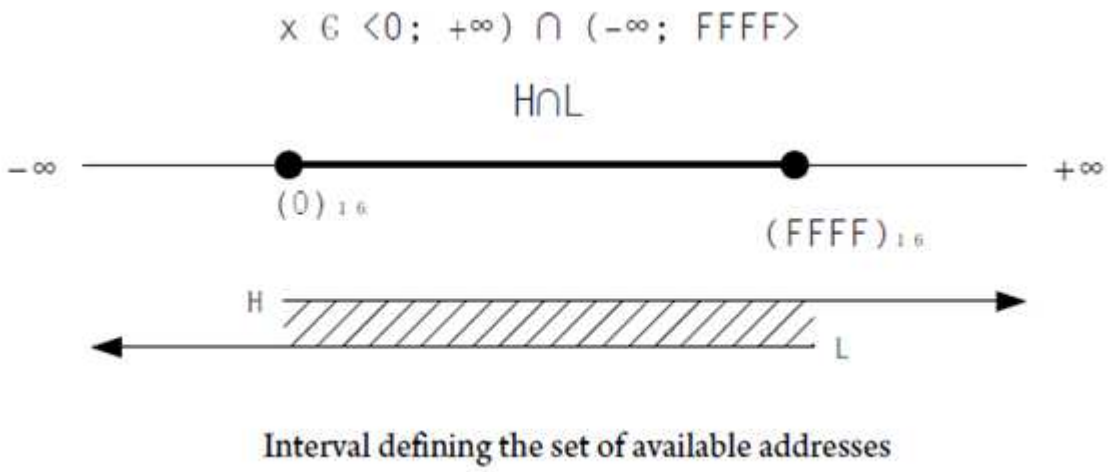
MOV A, 07_H

Register A, B, C, D, E, L and H can be used in every transfer instruction. However, when 16 bits are required to store or retrieve data from the memory HL register pair is capable of accepting and storing 16-bit values. Only Accumulator register is capable of performing arithmetic and logic operations. The result of every operation is stored in the Accumulator register.

4.3.2 External Memory Design

Memory is used to temporarily save data that may be retrieved at a later stage. Since the instruction operation is 8-bit wide and the operand can also acquire up to 8-bits, the required number of bits at every memory location is 8-bit. The memory address is uniquely defined by using 2 x 8-bits. Whereas, all registers are 8-bit wide, two registers must be used to access the memory. The first 8-bits are called higher bits (H) and the next 8-bits are lower bits (L). A memory address is traditionally expressed in hexadecimal format.

Mathematically expressed, each memory address x is from the interval.



Interval defining the set of available addresses

All the possible memory address locations can be expressed as an interval, as shown in the figure above.

The format of an address is $(XXXX)_{16}$, where X Є <0; F>. Examples of correct addresses: 3333, CCCC, FF23, 2210, denoting the hexadecimal numeric format: 0x3333, 0xCCCC, 0xFF23, 0x2210.

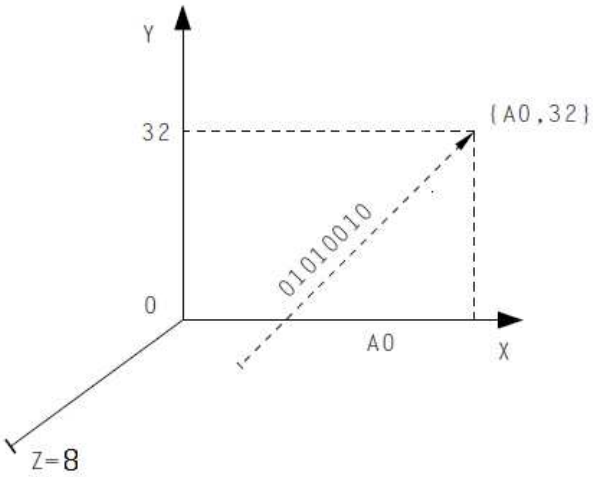
Requirements

- Memory address composed of 16-bits
- Each location to accommodate 8-bits
- WRITE – store information from register at a particular address
- READ – retrieve data from a specific address

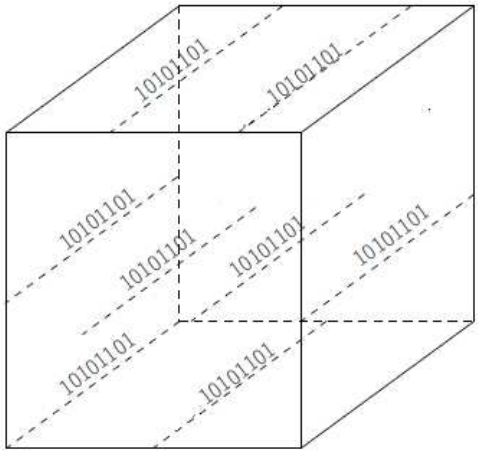
Analysis

Memory architecture is based on the 3-dimensional space, where the X and Y axis define the address and the information is stored at the Z axis. The following figure shows the memory architecture in a cubic form. The X axis defines the higher 8-bits and the Y axis the lower 8-bits of an address. The Z axis can hold up to 8-bits at each address. Thus, the union of all possible X and Y values can produce up to 65536 combinations. Each combination holds 8-bit (or 1-byte)

of data. Hence the available memory is of size 65536 X 1 byte = 65536 bytes = 65536/1024 KB = 64KB of memory.



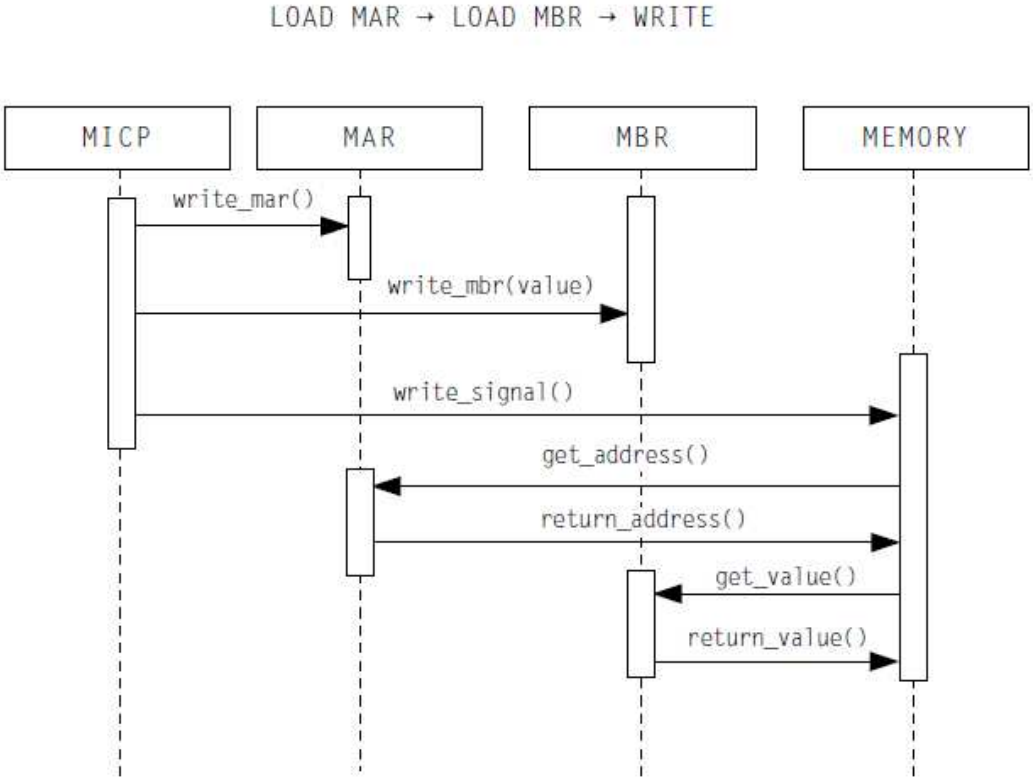
Memory architecture



Memory architecture in cubic expressed in cubic form

Two special registers are involved with memory read and write operations which are the MBR and MAR. The memory buffer register (MBR) is used to store the value to be written and values retrieved from memory and the memory address register (MAR) is used to store the address.

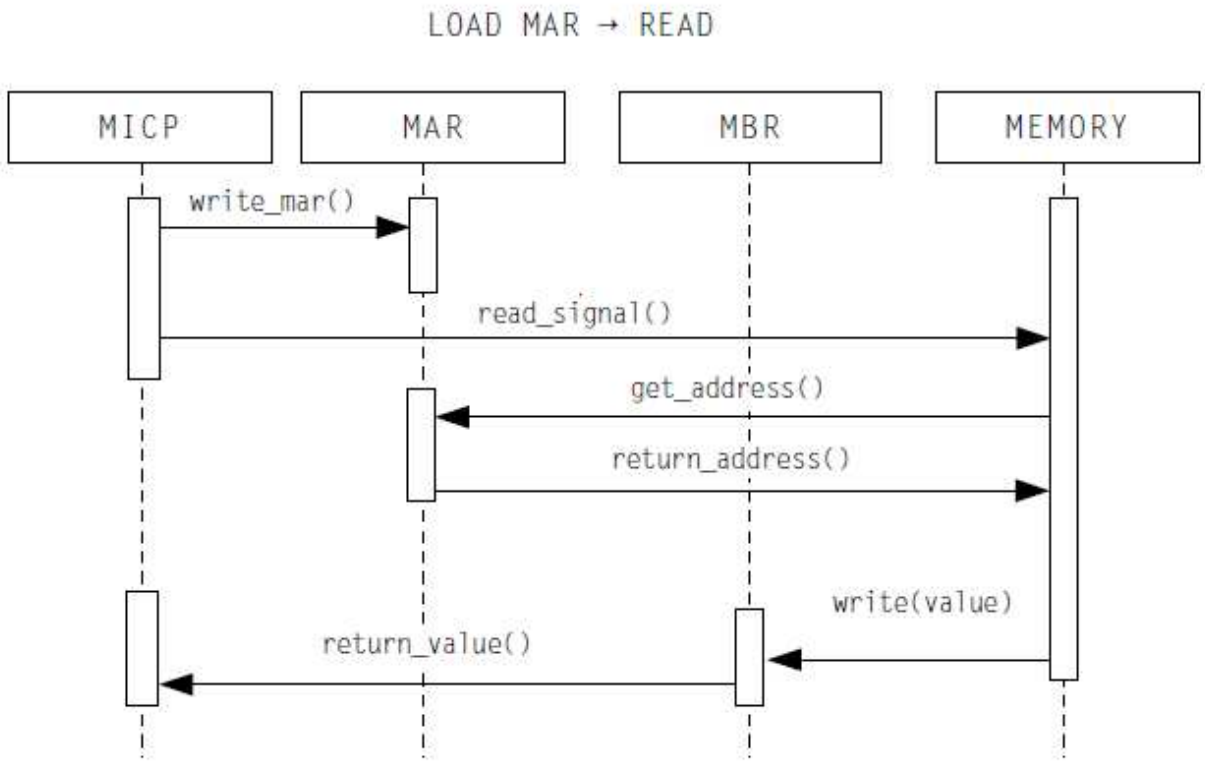
Write Operation



Memory write operation

The microprocessor writes the required address into MAR register and the value to be stored into the MBR. Then a write_signal() is sent to the memory. The memory then takes control over the writing procedure. Firstly, it retrieves the value stored in the MAR and uses it as a pointer to the memory location. Then, it retrieves the value from the MBR and writes the memory at the required location (MAR).

Read Operation



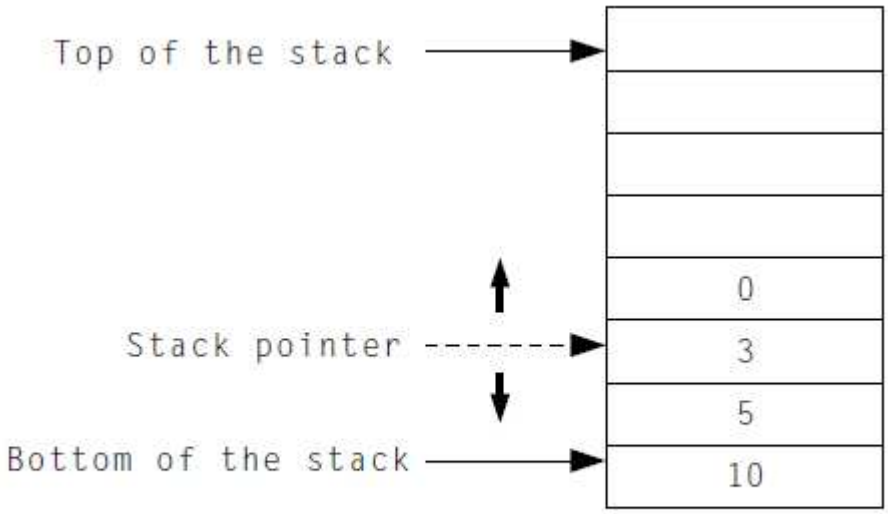
Memory read operation

To retrieve the data stored at a particular address in the memory, the microprocessor writes the required address into the MAR and sends a read signal to the memory. The memory then dispatches a get_signal() which tells the MAR to return the value it contains (memory address). The memory then uses the MAR's value to determine the right memory location, retrieves the data and writes it into the MBR. The MBR then implicitly returns the value to the microprocessor.

4.3.3 Stack Design

The stack is of first-in first-out data structure and it is an alternative way of data processing. The information is stored in last-in first-out manner where the last item pushed on to the stack is at the same time the first item to be retrieved from the stack. A very simplified model of a stack is shown in the figure below. The stack pointer always holds the address of lastly inserted item.

The operations associated with the stack are insertion and deletion of items. The operation of insertion is called *PUSH* and the operation of deletion is called *POP*. These operations are only simulated and no memory deletion is performed. The operations only increment or decrement the stack pointer (address).



Fundamental architecture of a stack

The stack in the above figure contains 8 trays which can be filled up with values. The stack pointer is simply a register that contains the address of the lastly inserted item. Since, the stack has 8 trays and thus 8 memory locations the stack pointer must at least contain 3 bits to be able to retrieve all data values:

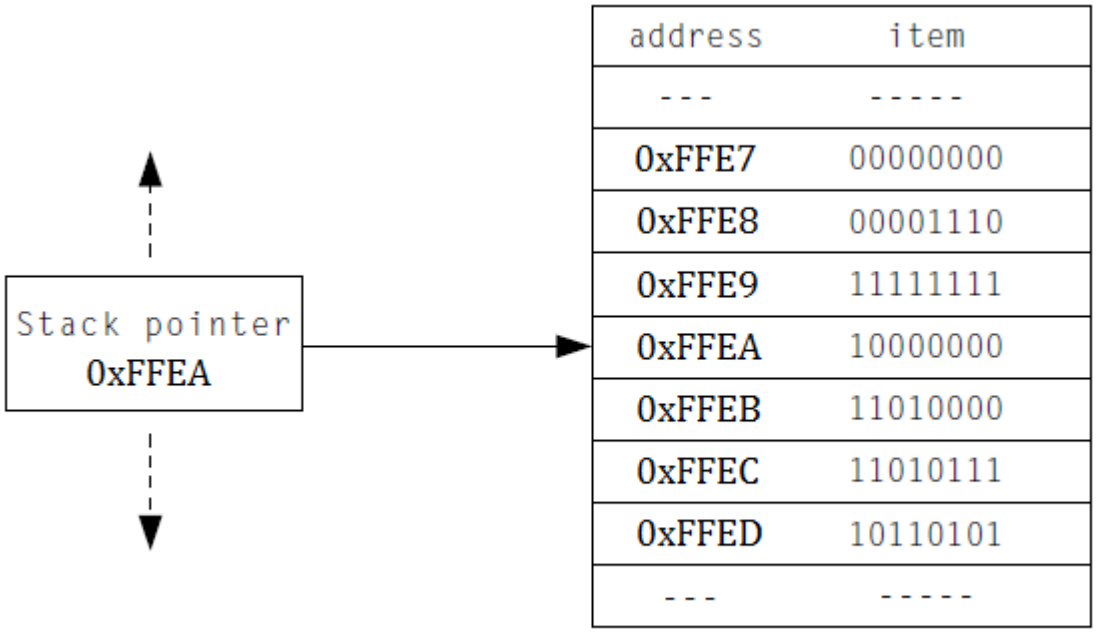
$$\log_2(8) = 3 \text{ bits}$$

Requirements

- Able to store at least 64 items
- Each location holds 8-bits of data
- Able to perform simple arithmetic and logic operations
- Able to push an item onto the stack from any register or directly
- Able to pop an item out from the stack to any register

Design

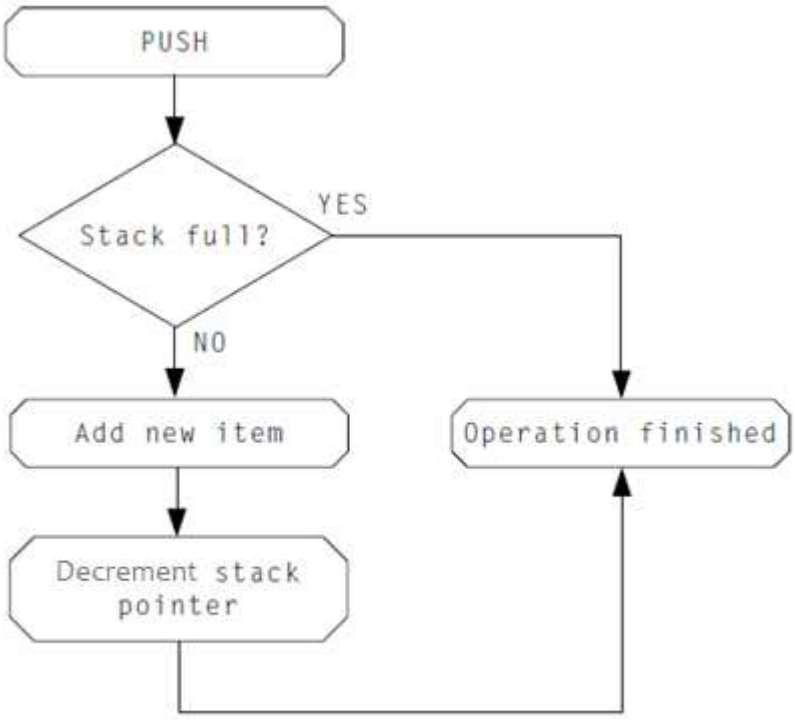
The stack will be implemented as a standalone component contrary to implementation as a part of memory. The design is similar to the memory described in the previous sub-chapter.



Stack design

The stack pointer can move up or down alongside the stack. However, when it reaches the bottom of the stack it cannot move further down and stays at the bottom position. The same rule applies when it reaches the top of the stack (stack is full).

Push operation

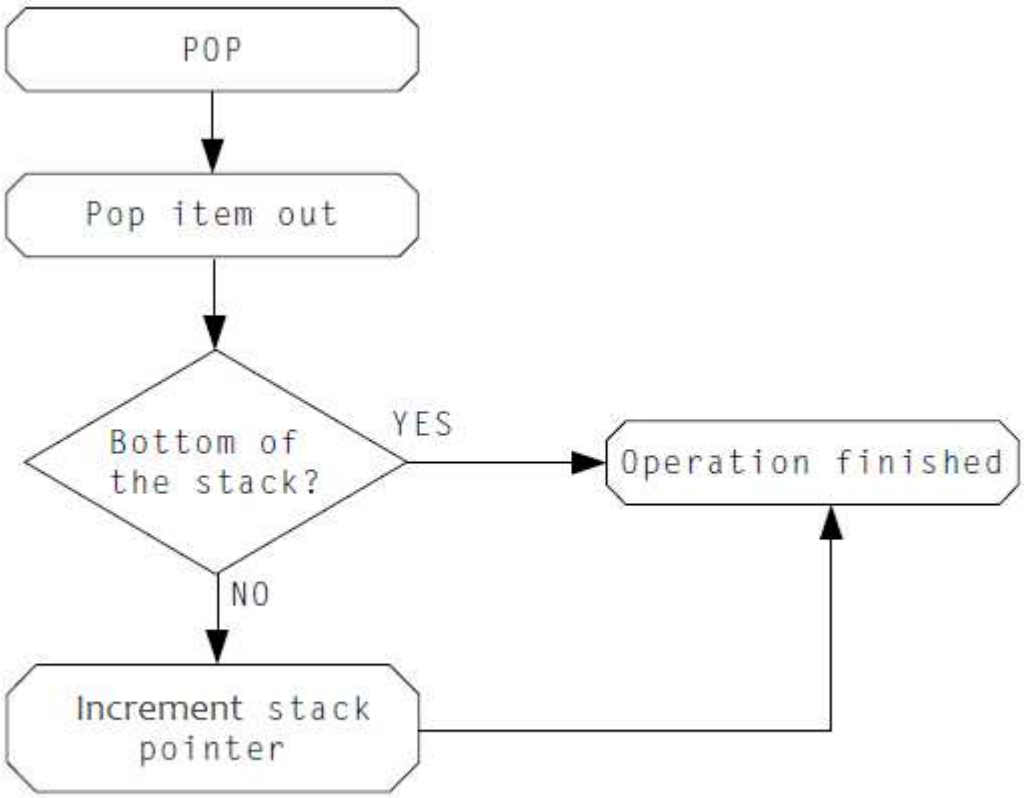


Stack push operation

The flowchart in the above figure shows the procedure of calling the PUSH operation. When inserting a new item onto the stack, the stack checks whether there is available space. If yes, it adds the new item on the top of the stack and decrements the stack pointer so it points at the new item. If the stack is full, no item is added onto the stack unless an item has been popped out from the stack.

Pop operation

POP function retrieves an item from the stack. Then the stack checks if the stack pointer has reached the bottom of the stack. If no, the stack pointer is incremented and points at the next item below. When the stack pointer reached the bottom of the stack and pop function is called the item is still retrieved but no increment of the stack pointer is performed and the same item is retrieved whenever the pop function is called.



Stack pop operation

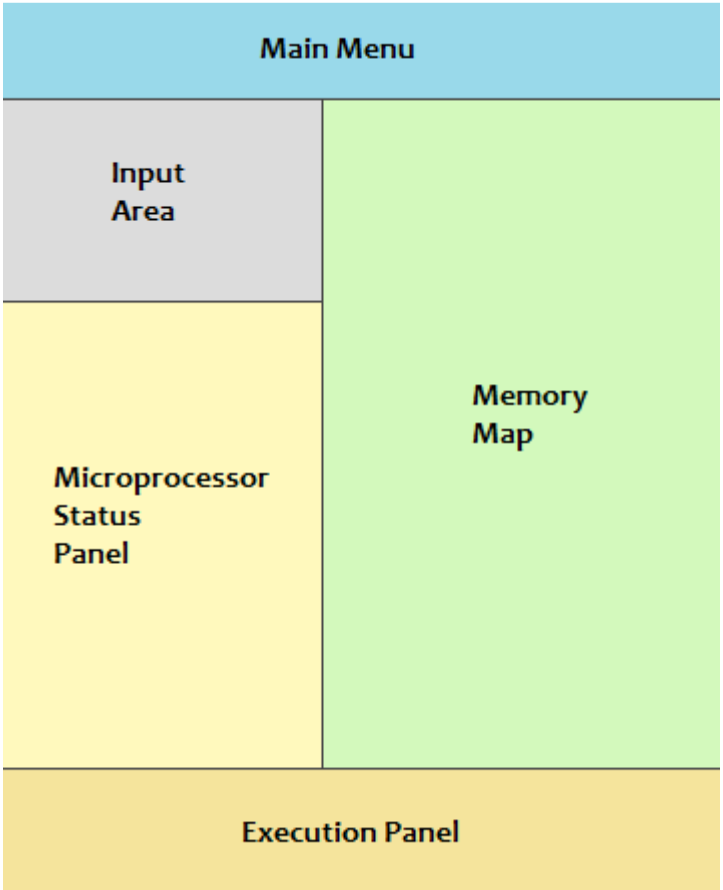
In case of stack operations, the stack pointer, at the beginning, points to the last stack address (i.e. 0xFFFF in our case), which acts as the bottom of the stack. As items are pushed into the stack, the value of the stack pointer is decremented.

4.4 User Interface Layout

The user interface must provide an easy access and flexibility. The user is allowed to write the opcodes into the memory and execute the code by providing the execution starting address. The interface should provide a simple notification about execution errors. All internal contents of the implemented registers, stack and memory should be displayed and updated during the execution to allow the user to immediately identify the changes. A message should be shown at the end of the execution informing the user about successful simulation.

The user interface consists of 5 main parts:

- Main menu
- Input area
- Microprocessor status panel
- Memory map
- Execution panel



The input area provides the user the option to input code (in the form of opcodes) into the memory. The microprocessor components – registers, flags, and stack are situated in the box "Microprocessor status panel". The memory map contains the contents of all memory location.

4.4.1 Main Menu Layout

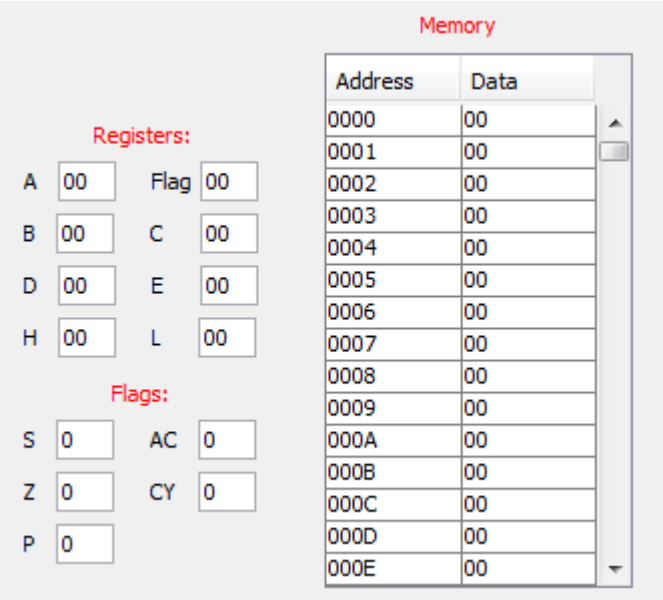
The main menu is composed of two sub-menus: File and Help. The *File Menu* is intended to provide a standard file handling options – Load, Save and Exit. The Help menu contains an option about which pops up a new window with the author's details and website link to the project's website.

FILE	HELP
	About
Load	
Save	
Quit	

Main menu options

4.4.2 Microprocessor Status Panel & Memory Map layout

Register values are shown in hexadecimal format. The intention is to provide a fixed layout for registers where the user is not required to scroll down to determine the values. The registers are split into three groups. The first group contains general purpose registers that are most important to the user. Special registers MAR, MBR, PC etc. are grouped in the second group. The last in hexadecimal format (0000 - FFFF). The hexadecimal value on the right-hand side is the actual value stored at that memory location. Each memory location holds one hexadecimal value that is 1 byte wide.



Chapter 5

IMPLEMENTATION

5.1 Simulator Operation

The simulator is or a container that contains the implemented microprocessor and the input/output communication.

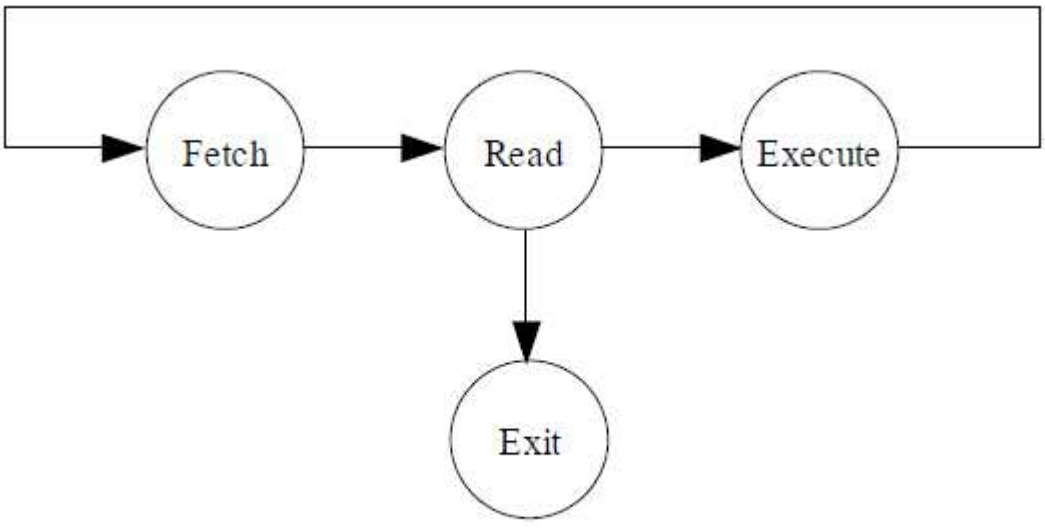
The simulator accepts a simple text file with the any extension. It is assumed that the file contains strings hexadecimal opcodes which defines the instructions. These instructions are passed to the microprocessor which decodes and executes each instruction.

All programs written on this simulator should be written in machine language form (i.e. opcodes), as there is no compilation module designed to compile Assembly Language Programs.

The microprocessor operates in the following manner:

- Fetch an instruction from the memory unit
- Read the instruction
- Decode and execute the instruction

The instruction fetching continues until the terminating instruction is read. The execution is then halted by the simulator.



Instruction execution procedure

Each of the steps is executed at a certain speed. However, this speed by default, would be the speed of the computer itself. The rate at which the execution is performed is determined by the clock and timing of the implemented microprocessor.

5.2 Register Implementation

Each register is of the type short, as it supports exactly 1 byte. The method execute () contains codes to handle all kinds operations related to the registers. Every register can hold a value between 0 and 255, edge inclusive.

Registers Accumulator (A), B, C, D, E, H, L, Program Counter (PC), Stack Pointer (SP) have been implemented.

Registers MBR and MAR have not been implemented, as their requirement is only while hardware interfaced with memory device.

Use of register pairs BC, DE and HL have also been implemented by the simple concatenation of the original two registers.

All these registers are by default set to 00_H.

5.3 Number Base Conversions

Conversions between different numerical systems are a crucial part of this implementation. Java provides inbuilt libraries for successfully converting data from one system to the other.

Numerical systems used in this implementation are:

- Decimal
- Hexadecimal
- Binary

Hexadecimal values can hold ordinary alphabet letters (A, B, C, D, E, F) and thus similarly to binary it is stored as string of characters.

5.4 Memory Implementation

Although, the external memory devices are based on a 3-directional cubic architecture, it has been implemented as a single one dimensional array of Strings.

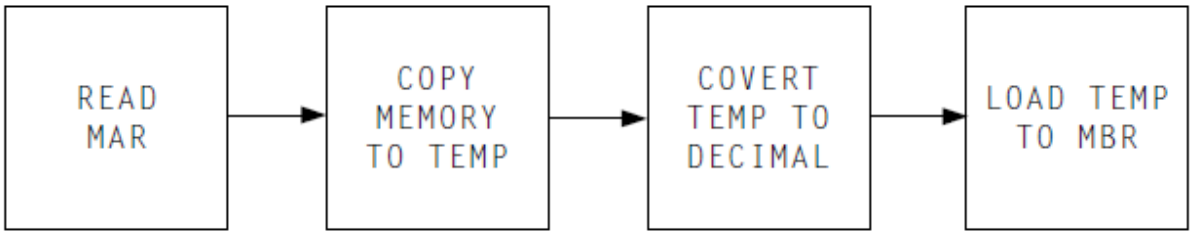
The dimensions are 16^4 X 8, where the first dimension represents the memory address, and the second dimension tells about the no. of bits that shall be stored per memory location.

Memory essentially performs only two operations:

- Read memory
- Write memory

Read Operation

The process of read operations is:



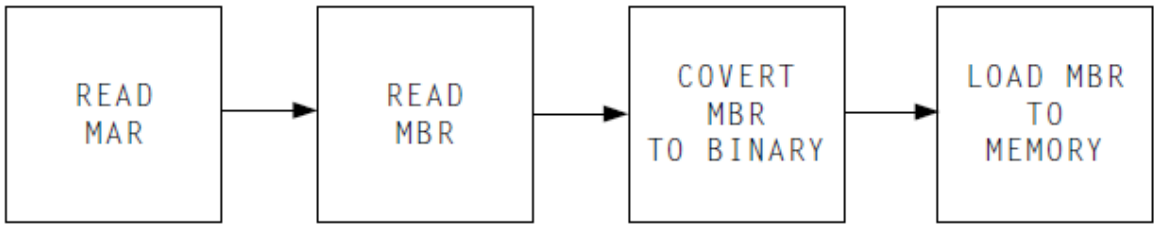
Read operation procedure

The address from MAR register is first read into temporary variables where decimal value is obtained. Hexadecimal representation is then copied into temporary registers. The hexadecimal representation is then converted into decimal form and stored in MBR.

However, in this actual implementation, the MAR and MBR registers have been implemented as temporary registers for ease of memory operations.

Write Operation

Write operation process:



Write operation procedure

The address is retrieved from the MAR in the same way as in the read operation. The value to be stored in the memory is located in the MBR. The value in the MBR, which is in decimal, is converted into hexadecimal format. Finally, all 8-bits are copied into memory in a similar way as in read operation.

The function “void setMemory(int addr, int val)” is implemented here to store data “val” into the memory at address “addr”.

5.5 Stack Implementation

The stack is of the type FIFO (First In, First Out). It consists of a defined number of memory locations. Since the stack pointer is of 16-bit, hence a virtual stack of 2^{16} memory locations has been implemented. Every memory location in the stack is able to store 1 byte of data, in hexadecimal numerical system.

Two elementary operations are associated with stack - push and pop. Push method accepts an integer to be pushed on the top of the stack.

void push(int val)

The decimal value val is inserted onto the top of the stack, which is pointed to by the value of the stack pointer. Before inserting the data, the system checks whether the stack is full or not. If the stack is full, warning message shall be passed on to the user.

If in any case, the stack pointer's value decreases to below 0000_H, it shall be automatically rounded up to point to the bottom of the stack (here, $FFFF_H$) for complete execution of the code.

int pop()

Pop operation pops out a value from the top of the stack. If the pop operation is successful, then the value of the stack pointer is incremented. If the value of stack pointer exceeds $FFFF_H$, the value is rounded off to 0000_H. Before the execution of the pop statement, the system checks whether the bottom of the stack has been reached, and generates a warning message in case of erroneous execution. After a value is popped, the actual value still remains in the stack's memory location, but it is no more accessible for the programmer.

5.6 Arithmetic Logic Unit implementation

The ALU consists of temporary registers, the Accumulator and the flag register. The result of any operation is reflected by the value in the accumulator register and the flag register. Additionally, the result of an operation can be stored in the stack or the memory.

The ALU can perform two arithmetic operations: addition and subtraction, and logical operations: AND, OR and XOR.

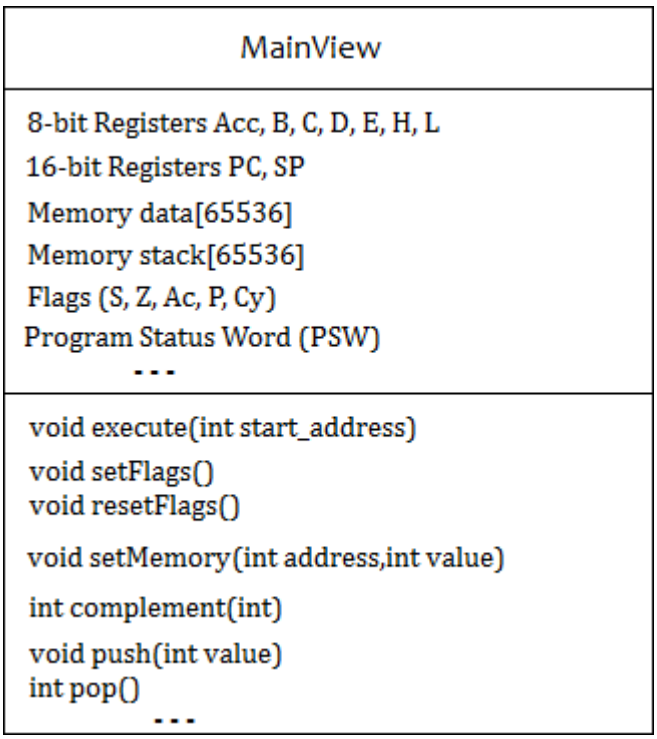
Arithmetic operations addition, multiplication and subtraction can alter the flags - comparison, zero and carry. The ALU implementation contains two references to the stack and a register. Both are required as the result of all operations is copied to either of the two.

The ALU can perform only 8-bit arithmetic on whole numbers.

5.7 Microprocessor Implementation

The class “MainView” inside the 8085 project acts as a container which holds and links every component of the implemented microprocessor. It also provides instruction decoding and execution functions. A very basic example of running a simulation can be achieved by defining the instruction in its hexadecimal format and store it as a string array (string instructions).

An instance of the microprocessor gets created at the start of the simulation. Then the microprocessor is ready to load the instructions into the memory. When all instructions have been loaded the microprocessor is ready to execute them. Member function execute() provides all necessary steps to accomplish the execution.



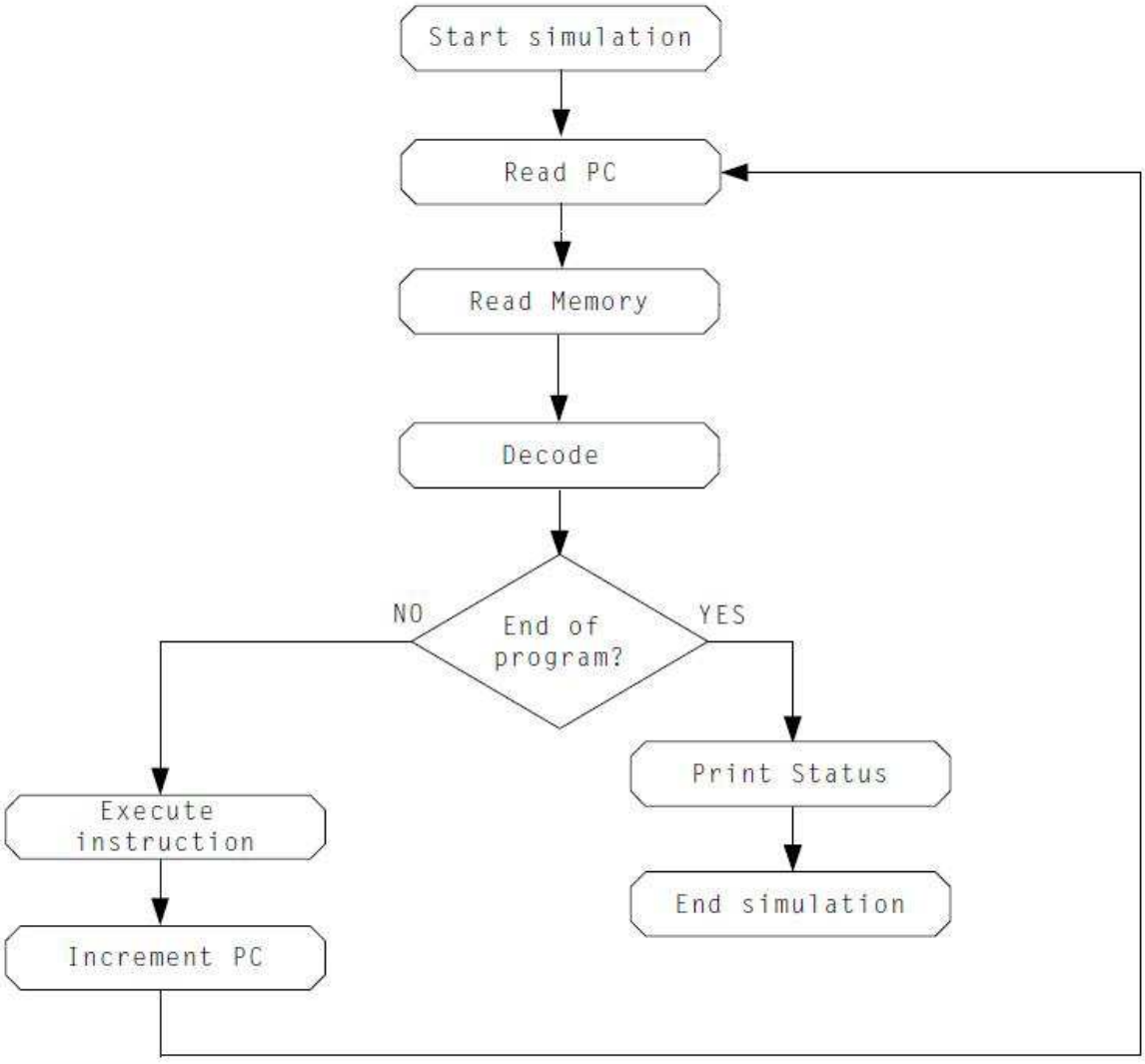
5.7.1 Instruction Execution

The microprocessor loads the program counter register with a value 0000_H or with the value provided by the user. Program counter is a register which always points at the next instruction to be executed by the microprocessor. The value is automatically incremented after each instruction has been executed.

Then the MAR (Memory Address Register) is loaded with the value contained within the PC register. MAR stores an address location to be access by the memory. The content stored at the memory location is copied into MBR (Memory Buffer Register). The content is obviously an 8-bit instruction. The content of the MBR contains the instruction that is read by the

microprocessor, decoded and executed. PC register is then incremented by 1 and the process is repeated.

The instruction in hexadecimal format is converted into an integer. Then the microprocessor enters an switch/case statement which contains the implemented instruction set architecture.



Instruction execution flow chart

5.7.2 Register and Memory Data Transfer Operation

These operations let a programmer move data between registers at his/her own choice. Data can be moved to and from every register that has been provided to the programmer.

Data can also be moved directly from any memory location to any register by using the HL register pair as a 16-bit memory address holder.

Data can also be moved to any register directly, i.e. the data source being the source program (memory) itself.

5.7.3 Arithmetic Operation

Every arithmetic operation has been implemented as one of the cases of the switch/case block in the execute() method.

The operations are made up of various micro-operations:

1. Loading the value of Accumulator into a temporary register
2. Loading the second operand into temporary register
3. Executing the instruction
4. Store result in accumulator
5. Modify flag bits of the flag register according to the operation and result

5.7.4 Control Flow Instructions

Program counter is the register responsible for the sequential execution of instructions present in the memory. The value of the program counter is changed every time an instruction is executed.

At any point of time during execution, the program counter stores the memory address of the next instruction that is to be executed.

Control flow statements, control the flow of execution of the program by changing the values of the program counter, and/or by storing the value of the program counter in stack.

Jump statements are simple branching instructions which simply change the direction of the execution only.

Call and Ret statements are used for subroutine designing. These statements store and retrieve respectively, the program counter value in/from stack, before branching.

All these three instructions can be used as conditional branching statements by checking the condition of sign, zero, parity and carry flags.

5.8 Graphical User Interface Implementation

One of the most popular GUI frameworks on any platform is the Java Swing Framework. It is much sophisticated than the older AWT (Abstract Window Toolkit), and provides several advanced GUI components. Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written completely in Java, and hence are platform independent.

5.8.1 Java Swing Framework

Java Swing follows a single-threaded programming model. It is a highly modular architecture, which allows for the “plugging” of various custom implementations of specified framework interfaces. It is a component based framework.

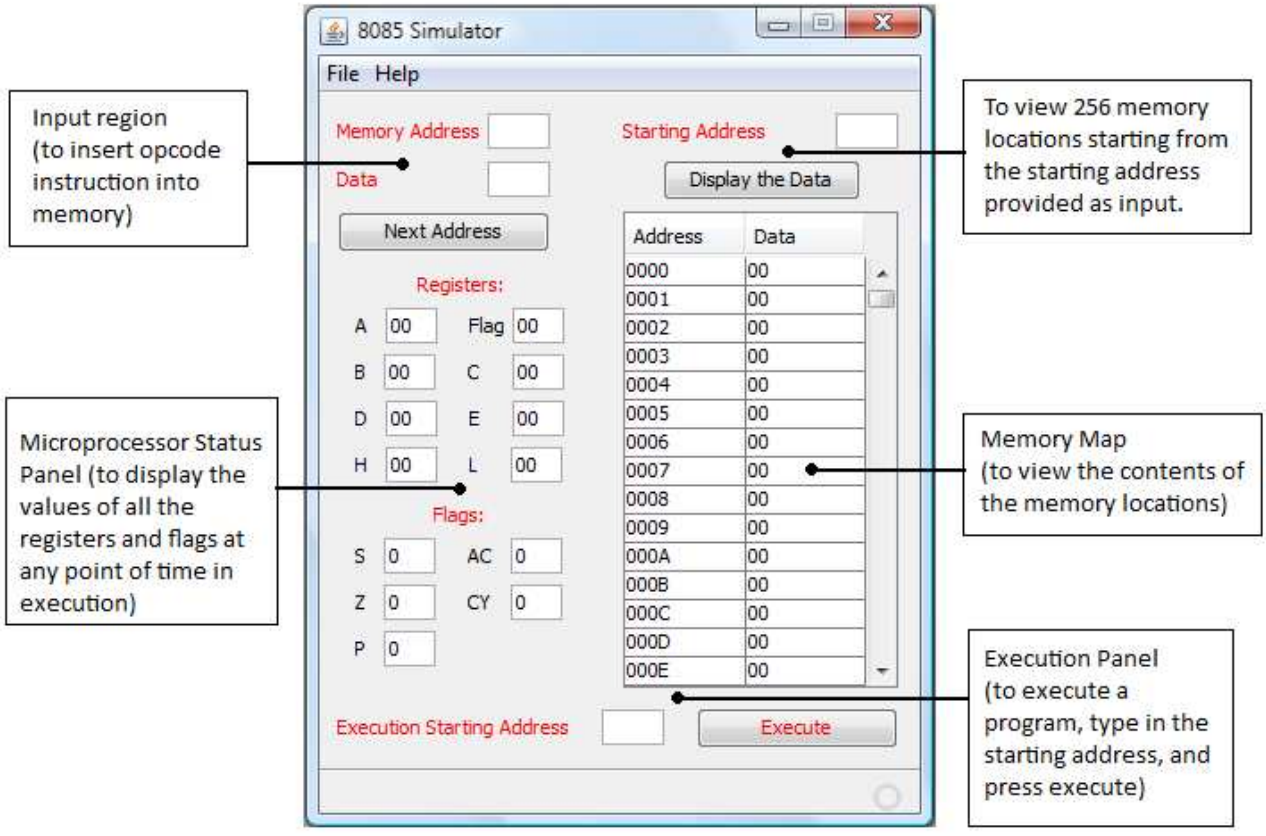
Given the programmatic rendering model of the Swing framework, fine control over the details of rendering of a component is possible in Swing. It helps create a dynamic user interface, thus improving highly, the look and feel of the application interface. Moreover, the Swing library makes heavy use of the Model/View/Controller software design patterns.

5.8.2 NetBeans IDE 6.7.1

NetBeans IDE is a free and open-source software development tool for developing enterprise, web, desktop and mobile applications. This IDE provides end-to-end solution for all Java Development Platforms.

NetBeans support development of applications in Java, C, C++, Ruby, C#, etc, and hence can be used for integration of a large variety of complex systems.

5.8.3 The Simulator GUI



The interface for this simulator application has been designed very neatly and simply, for easy understanding by users. Data can be inserted only by using the Input Region. Whereas, to view memory, use the memory map area. The Microprocessor Status Panel provides the output of the program after successful completion of execution.

Chapter 6

RESULTS and ANALYSIS

6.1 Simulation Analysis

Simulation analysis is mainly concerned with the correctness of the simulation processes, such as simulation output and the correctness of the output. Small programs written in Assembly Language, and then compiled into opcodes shall be used to test the correctness of the Simulator.

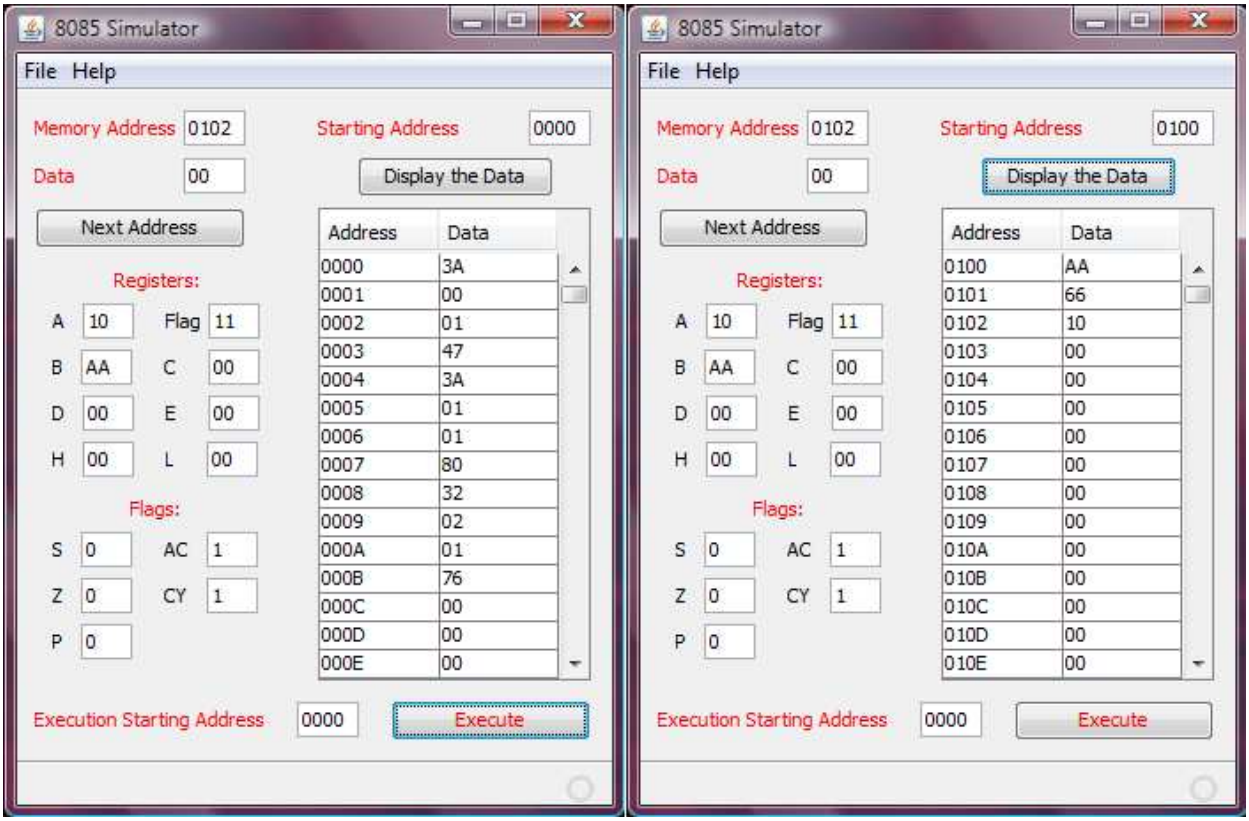
Assembly script to add two 8-bit numbers and store it in memory:
(Considering the two numbers are at memory locations 0100_H and 0101_H)

```
Begin
        LDA 0100H
        MOV B,A
        LDA 0101H
        ADD B
        STA 0102H
        HLT
End
```

The corresponding opcodes for the above program:
(The program is loaded at memory location 0000_H)

0000_H – 3A
0001_H – 00
0002_H – 01
0003_H – 47
0004_H – 3A
0005_H – 01
0006_H – 01
0007_H – 80
0008_H – 32
0009_H – 02
$000A_H$ – 01
$000B_H$ – 76

Output for the previous program:



As we can see from the above output of the simulator, the instructions were correctly executed.

The value at 0100_H – AA_H, and the value at 0101_H – 66_H.

The addition of these two values should produce the following output:

Value at 0102_H – 10_H

Carry and Auxiliary Carry flags are set.

Sign, Zero and Parity flags are reset.

Hence, the simulation processes is working correctly, and is producing correct results for the test program used.

We can also see that the data transfer instructions & the arithmetic instructions are executing properly.

6.2 Microprocessor Analysis

Microprocessor analysis deals completely with the correct execution of all the different types of instructions present in the microprocessor, like arithmetic & logical operations, control flow operations, register & memory data transfer instructions. It deals with the correctness of data processing and resultant output of the test program.

Assembly script to compare 2 numbers and find the smaller of the two.

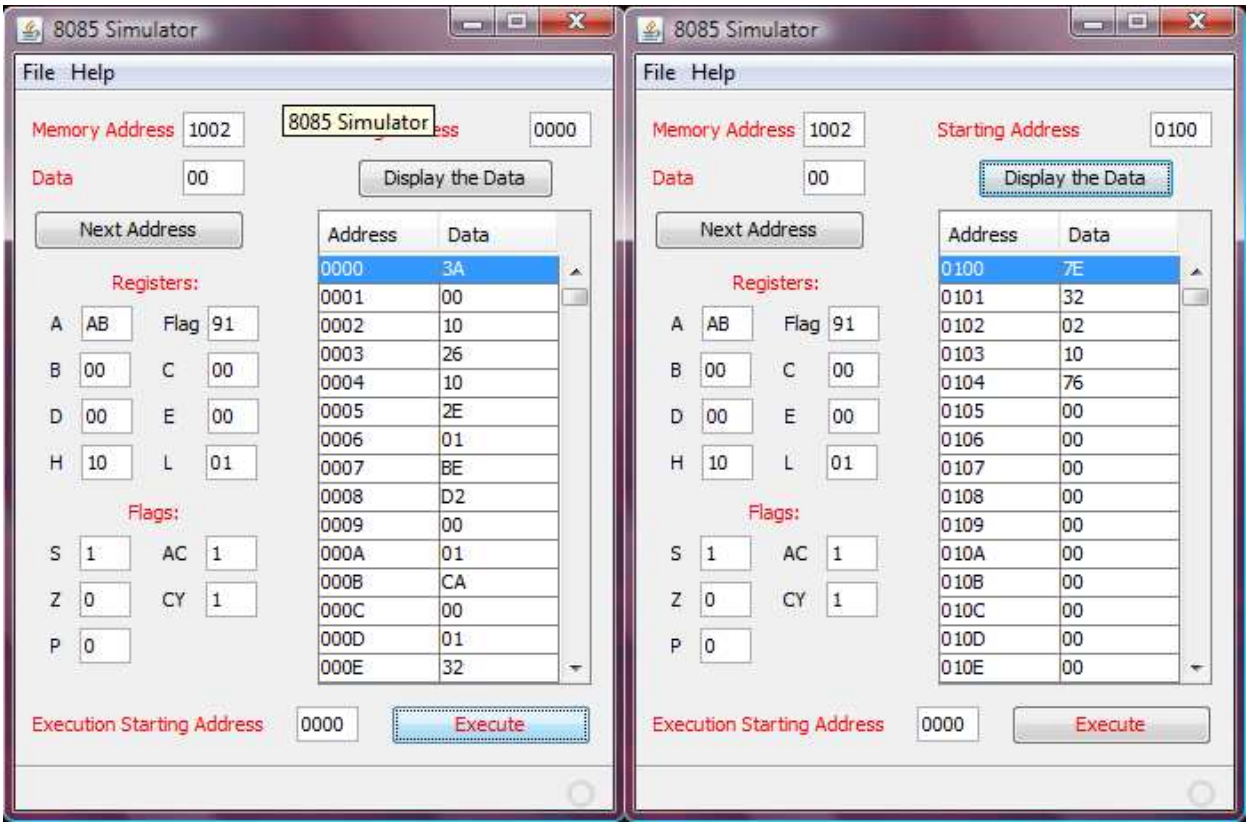
(Considering the two numbers are at location 1000_H and 1001_H, and the result is to be stored at 1002_H)

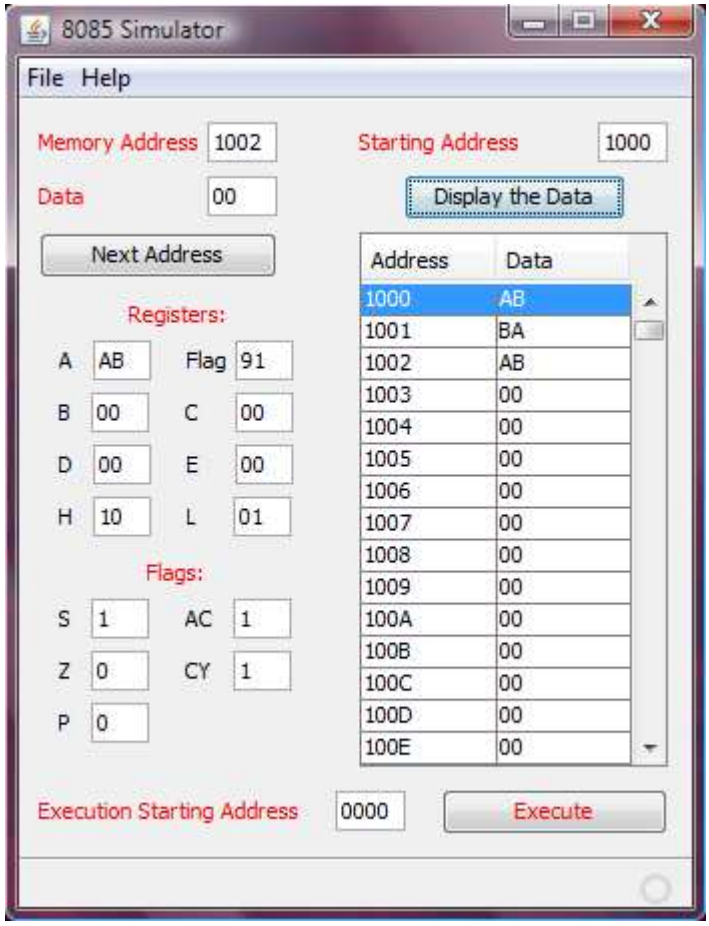
```
Begin (at 0000H)
        LDA 1000H
        MVI H, 10H
        MVI L, 01H
        CMP M
        JNC Label1
        JZ Label1
        STA 1002H
        HLT
End
Begin (at 0100H)
Label1:     MOV A,M
            STA 1002H
            HLT
End
```

The corresponding opcodes for the above program:

0000_H – 3A	$000F_H$ – 02	0100_H – 7E
0001_H – 00	0010_H – 10	0101_H – 32
0002_H – 10	0011_H – 76	0102_H – 02
0003_H – 26		0103_H – 10
0004_H – 10		0104_H – 76
0005_H – 2E		
0006_H – 01		
0007_H – BE		
0008_H – D2		
0009_H – 00		
$000A_H$ – 01		
$000B_H$ – CA		
$000C_H$ – 00		
$000D_H$ – 01		
$000E_H$ – 32		

Output of this program:





As we can see from the above output of the simulator, the instructions were correctly executed.
The two numbers are at locations: 1000_H – AB_H, and 1001_H – BA_H.
The comparison of these two values should produce the following output:
Value at 1002_H – AB_H
Carry, Sign and Auxiliary Carry flags are set.
Zero and Parity flags are reset.

Hence, the simulation processes is working correctly, and is producing correct results for the test program used.
We can also see that the data transfer instructions, the control flow instructions, the register & memory data transfer instructions are executing properly.

Chapter 7

CONCLUSION

“Happiness lies in the joy of achievement and the thrill of creative efforts”

Our project “8085 Microprocessor Simulator” has given us tremendous exposure in the field of software development. We have actually put our theoretical knowledge to practice. We have come to realize the importance of software development life cycle and various stages in it. In addition to exposure to the professional world this project has helped us to learn a lot about team spirit, and that a single person cannot complete any task wholly and successfully. It has to be a joint effort.

- Although the 8085 µP has become a outdated, it is still studied to understand the basic working and architecture of a processor.
- Use of Simulators can help students who are in the process of learning microprocessor programming.
- Educational labs and training institutes can implement use of such simulators at no extra cost.
- Our project, as of now, shall cover the basic necessities and hence would have potential for further development.

Chapter 8

FUTURE WORKS

Since, this simulator application has been developed within a limited period of time; hence it possesses tremendous scope for future enhancements, and additions to the application. Some of the possible enhancements that could not be done, but can be implemented in future are:

- Assembler: The application developed here is a pretty simple one. It can take hexadecimal numbers as input and produce result accordingly. But as a part of future work, it can be done as if the developed assembler could take the assembly level instructions as the input and produce the result after necessary compilation of the entered codes there.

- Graphical User Interface: As a part of future work, the GUI can be made much more attractive to the users. A feature where the user could be able to write his/her assembly level code on the Text Editor provided there on the interface so that syntax errors could easily be solved if any.

- Animation: An animation showing the internal flow of information and data would increase the overall understating of the microprocessor architecture. Each instruction would have a defined procedure to animate the process.

- Microprocessor: Currently, the Simulator can handle only the software interrupts. It cannot handle the hardware interrupts used in the microprocessor simulator because here hardware interfacing of this simulator is not possible. If any future works are to be done then this must be the area of interest. Currently, the Simulator can display a maximum of 256 address locations starting from the starting address at a time on the Memory Map area provided there. But as a future work, it can be done that we can display the full address locations available to the user.

Chapter 9

REFERENCES

Books:

1. “Software Engineering” By Rajib Mall
2. “JAVA - The Complete Reference” by Herbert Schildt
3. “Microprocessor Architecture, Programming, and Applications with the 8085” by Ramesh Gaonkar

Websites:

1. http://en.wikipedia.org/wiki/Intel_8085