JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

Department of CSE&IT

Lab Manual

Course Code: 15B17CI373

Course Name: Computer Organization and Architecture Lab

BTech CSE 3rd year (5th Semester)

Prepared by: Ms Amarjeet Kaur

Approved by: Dr. Vikas Saxena

HOD/CSE&IT

2022-23 (ODD2022)

VISION

Vision: To be a centre of excellence for providing quality education and carrying out cutting edge research to develop future leaders in all aspects of computing, IT and entrepreneurship.

MISSION

MISSION 1: To offer academic programme with state of art curriculum having flexibility for accommodating the latest developments in the areas of computer science and IT.

MISSION 2: To conduct research and development activities in contemporary and emerging areas of computer science & engineering and IT.

MISSION 3: To inculcate IT & entrepreneurial skills to produce professionals capable of providing socially relevant and sustainable solutions.

Programme Name: B.TECH. IN COMPUTER SCIENCE & ENGINEERING

PROGRAMME EDUCATIONAL OBJECTIVES:

PEO 1: To provide core theoretical and practical knowledge in the domain of Computer Science & Engineering for leading successful career in industries, pursuing higher studies or entrepreneurial endeavours.

PEO 2: To develop the ability to critically think, analyze and make decisions for offering techno-commercially feasible and socially acceptable solutions to real life problems in the areas of computing.

PEO 3: To imbibe lifelong learning, professional and ethical attitude for embracing global challenges and make positive impact on environment and society.

PROGRAMME OUTCOMES:

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAMME SPECIFIC OUTCOMES:

PSO 1: Able to identify suitable data structures and algorithms to design, develop and evaluate effective solutions for real-life and research problems.

PSO 2: Able to excel in various programming/project competitions and technological challenges laid by professional societies.

SEMESTER - III																			
Course Name & Code	Course Outcome	Cognitive Level	CO Code	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
Computer Organization and Architecture LAB,15B17CI 373	Implementation basic ALU of 2-bit and 4-bit computer using hardwired simulation tool	C3	C273.1	3	3	1	1	3	1			1				2	3
	Initialization and fetching of data from specific memory using various addressing mode of 8085 and 8086	C2	C273.2	3	3	3	1	2	1			1				2	3
	Develop 8086 assembly language programs using software interrupts and various assembler directives.	C3	C273.3	3	3	3	1	3	1			1				2	3
	Develop Microprocessor Interfacing program using PPI for various external devices	C3	C273.4	3	3	2	1	1	1			1	1		1	2	3
	Develop MIPS assembly language programs using software interrupts and various assembler directives.	C3	C273.5	3	3	2	1	1			1				1	2	3
	Create of application and its software using 8085/8086 microprocessor or microcontrollers	C6	C273.6	3	3	2	1	2	1	1		2	1			2	3
			C273	3	3	2	1	2	1	0	0	1	0	0	0	2	3

Programme Name: B.TECH. IN INFORMATION TECHNOLOGY

PROGRAMME EDUCATIONAL OBJECTIVES:

(Same as CSE PEO)

PROGRAMME OUTCOMES:

(Same as CSE PO)

PROGRAMME SPECIFIC OUTCOMES:

PSO 1: Able to acquire practical competency with emerging technologies, programming languages and open source platforms.

PSO 2: Able to assess hardware and software aspects necessary to develop IT based solutions.

SEMESTER - III																			
Course Name & Code	Course Outcome	Cognitive Level	CO Code	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
Computer Organization and Architecture LAB,15B17CI 373	Implementation basic ALU of 2-bit and 4-bit computer using hardwired simulation tool	C3	C273.1	3	3	1	1	3	1			1				2	3
	Initialization and fetching of data from specific memory using various addressing mode of 8085 and 8086	C2	C273.2	3	3	3	1	2	1			1				2	3
	Develop 8086 assembly language programs using software interrupts and various assembler directives.	C3	C273.3	3	3	3	1	3	1			1				2	3
	Develop Microprocessor Interfacing program using PPI for various external devices	C3	C273.4	3	3	2	1	1	1			1	1		1	2	3
	Develop MIPS assembly language programs using software interrupts and various assembler directives.	C3	C273.5	3	3	2	1	1			1				1	2	3
	Create of application and its software using 8085/8086 microprocessor or microcontrollers	C6	C273.6	3	3	2	1	2	1	1		2	1			2	3
			C273	3	3	2	1	2	1	0	0	1	0	0	0	2	3

COURSE OUTCOMES: Upon successful completion of this course it is expected that students will be able to:

S. No	DESCRIPTION	COGNITIVE LEVEL (BLLOMS TAXONOMY)
CO1	Implementation basic ALU of 2-bit and 4-bit computer using hardwired simulation tool	Apply (Level 3)
CO2	Initialization and fetching of data from specific memory using various addressing mode of 8085 and 8086	Understand (Level 2)
CO3	Develop 8086 assembly language programs using software interrupts and various assembler directives.	Apply (Level 3)
CO4	Develop Microprocessor Interfacing program using PPI for various external devices	Apply (Level 3)
CO5	Develop MIPS assembly language programs using software interrupts and various assembler directives.	Apply (Level 3)
CO6	Create of application and its software using 8085/8086 microprocessor or microcontrollers	Create (Level 6)

Direct and In-Direct Assessment Tools

Computer Organization and Architecture (15B17CI373)			
	Course Outcome	**Direct Assessment Tools (80%)**	**In-Direct Assessment Tools (20%)**
CO1	Implementation basic ALU of 2-bit and 4-bit computer using hardwired simulation tool	T1	Course Exit Survey
CO2	Initialization and fetching of data from specific memory using various addressing mode of 8085 and 8086	T1, Evaluation-1	Course Exit Survey
CO3	Develop 8086 assembly language programs using software interrupts and various assembler directives.	T2	Course Exit Survey
CO4	Develop Microprocessor Interfacing program using PPI for various external devices	Evaluation-2	Course Exit Survey
CO5	Develop MIPS assembly language programs using software interrupts and various assembler directives.	T2	Course Exit Survey
CO6	Create of application and its software using 8085/8086 microprocessor or microcontrollers	Project	Course Exit Survey

Detailed Syllabus

Lab-wise Breakup

Course Code	15B11CI373 **NBA CODE:C273**	**Semester ODD** **(specify Odd/Even)**	**Semester Fourth Session** 2022 -2023 Month from Aug to Dec 2022
Course Name	Computer Organization and Architecture Lab		
Credits	1	**Contact Hours**	2

Faculty (Names)	**Coordinator(s)**	Ms Amarjeet Kaur (62)
	Teacher(s) (Alphabetically)	Ms Amarjeet Kaur, Dr Hema N, Dr Janardan Verma, Dr Kapil Madan, Dr Pawan K. Upadhyay, Dr Taj Alam,

COURSE OUTCOMES		**COGNITIVE LEVELS**
C273.1	Implement basic ALU of 2-bit and 4-bit computer using hardwired simulation tool	Apply (Level 3)
C273.2	Initialization and fetching of data from specific memory using various addressing mode of 8085 and 8086	Understand (Level 2)
C273.3	Develop 8086 assembly language programs using software interrupts and various assembler directives.	Apply (Level 3)
C273.4	Develop Microprocessor Interfacing program using PPI for various external devices	Apply (Level 3)
C273.5	Develop MIPS assembly language programs using software interrupts and various assembler directives.	Apply (Level 3)
C273.6	Create application and its software using 8085/8086 microprocessor or microcontrollers	Create (Level 6)

Module No.	**Title of the Module**	**List of Experiments**	**CO**
1.	COA Hardwired simulation tool	Realize the truth table of various gates like as AND, OR, NOT, XOR, NAND and NOR., Conversion of universal gates, Design the half adder and full adder circuits, Ripple adder logic circuit, 4 x1 multiplexor circuit and realize the various input output logic based on control, 4X1 multiplexor	**C273.1**

		with NAND gates logic circuits	
2.	Combinational circuits	Design the subtractor circuits with defined bit logic, Adder-subtractor logic circuits, The odd frequency divider circuits, Carry lookup adder, Carry select and carry save, Adder circuits by modifying the ripple carry adder logic given in module-1.,Timing diagram of all four adder circuits and compare their performance, Decoder circuits with defined logic, 4-bit ALU circuits with defined operation logic.	**C273.1**
3.	8085 Simulator Introduction	Understanding Hardware Specification of the **8085 Simulator** in detail, Add two 8-bit numbers from load sample program from file menu, assemble and execute it step by step and view the contents of registers and memory., Basic Data transfer instructions, Arithmetic instructions, Logical instruction of 8085 using sample programs with note changes in flags.	**C273.2**
4.	8085 Programming (Simple)	8085 Assembly Programming: Basic Arithmetic (like addition, subtraction, multiplication, division etc), Array (sum , reverse, average copy etc) etc and explore more about Arithmetic , Logical and Flow control Instructions	**C273.2**
5.	8085 Programming (Complex)	8085 Assembly Programming: Logical and Data transfer (like Min, Max, Even/odd, Sorting etc), more complex program(like Factorial, Link list etc) , String etc and explore more about Arithmetic, Logical and Flow control Instructions, Interfacing with 8255	**C273.2, C273.4**
6.	8086(MASM/emu 86)	8086 Assembly Programming: Arithmetic (like addition, subtraction, multiplication, division etc), Logical and Data transfer (like Min, Max, Even/odd, Sorting etc), BIOS interrupt (I/O for read and write), String etc and explore more about Arithmetic, Logical, Flow control and Software Interrupt Instructions using MASM/emu86	**C273.3**
7.	MIPS(MARS) simulator	MIPS Assembly Programming: Arithmetic (like addition, subtraction, multiplication, division etc), Logical and Data transfer (like Min, Max, Even/odd, Sorting etc), Complex program (Factorial, Fibonacci etc), String etc and explore more about Arithmetic, Logical, Flow control Instructions using MARS Simulator.	**C273.5**
8.	Projects	Students are expected to create an hardware and software co-designed application based on 8085/ 8086/ MIPS/ Other controller (like Arduino) / Small Size computer (like Raspberry Pi **)** programming either in assembly or high level language.	**C273.6**
Project based learning: Project in COA lab is an integral part of the lab. Student form group size 2-3, and discuss the project idea with their lab faculty before finalizing. All projects are based on hardware and hardware components like microprocessor microcontrollers (like Arduino), microcomputer (like Raspberry pi), various sensors (like temperature sensor, humidity sensor etc), cams (like webcam), etc. are used. Programming language is used as per processor/controller. Students develop projects/prototypes to interact with physical environment, control physical object with software			

which is base of IoT and embedded system. Students learn various processor architecture as well as their programming languages. This helps students to understand how to develop IoT based products and embedded systems.

Evaluation Criteria

Components	**Maximum Marks**
Evaluation 1	**10**
Lab Test 1	**20**
Evaluation 2	**10**
Lab Test 2	**20**
Project / Assignments	**25**
Attendance	**15**
Total	**100**

Recommended Reading material: Author(s), Title, Edition, Publisher, Year of Publication etc. (Text books, Reference Books, Journals, Reports, Websites etc. in the IEEE format)

1.	William Stallings, Computer Organization and Architecture–Designing for Performance, 9th Edition, Pearson Education, 2013.
2.	Nicholas Carter, Schaum's outline of Computer Architecture, Tata McGraw Hill, 2017
3.	John L. Hennessy and David A Patterson, Computer Architecture A quantitative Approach, Morgan Kaufmann / Elsevier, Sixth Edition, 2017
4.	M. Morris Mano, Computer System Architecture, Prentice Hall of India Pvt Ltd, Fourth edition, 2002. ISBN: 81-203-0855-7.
5.	Microprocessor Architecture Programming and Applications with the 8085 [HB]-6/e. 25 September 2014. by Ramesh Gaonkar .
6.	The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro-Processor, Pentium II, Pentium III, Pentium 4, and Core2 with 64-bit Extensions : Architecture, Programming, and Interfacing. Barry B. Brey, Pearson Education India, 2009.
7.	http://nptel.ac.in/courses/Webcourse-contents/IIT-%20Guwahati/comp_org_arc/web/
8.	http://cs.nyu.edu/~gottlieb/courses/2010s/2011-12-fall/arch/class-notes.html
9.	http://www.cse.iitm.ac.in/~vplab/courses/comp_org/LEC_INTRO.pdf
10.	http://www.cs.iastate.edu/~prabhu/Tutorial/title.html
11.	http://www.cag.csail.mit.edu/
12.	http://www.research.ibm.com/compsci/arch

INTRODUCTION TO LABORATORY

One of the important courses of Computer Science and Engineering is Computer Organization and Architecture (COA) and its lab. We designed lab with set of experiments aimed at covering the classical computer functional units: processor, memory, and input/output system. Start of lab exercise is designed for simulating basic digital hardware with which the processor is built. The lab course goals complement those of the classroom course. We have designed and selected some experiments, trying to balance the course time among the mentioned functional units according to their importance. Computers are categorized as RISC and CISC. To simulating working of RISC and CISC architecture 8085, 8086 and MIPS are considered. When you are working close to hardware, the best programming language used is assembly programming. The aim is to acquire an elementary but complete knowledge about Computer Organization and Architecture working as well as its basic working principles and underlying design aspects. We also discuss the selection of a set of free software tools that allow those students requiring additional time, or those who show further interest, to continue their work at home. Students are expected to build hardware and software co-designed application based on 8085/8086/MIPS programming either in assembly or high level language. Application building exercise gives students a better opportunity to create their own system based on their interest.

Module-1

Hardwired Simulation Tool Introduction and Familiarization

Introduction:

This simulator provides an interactive environment for creating and conducting simulated experiments on computer organization and architecture. It supports gate level design to CPU design.

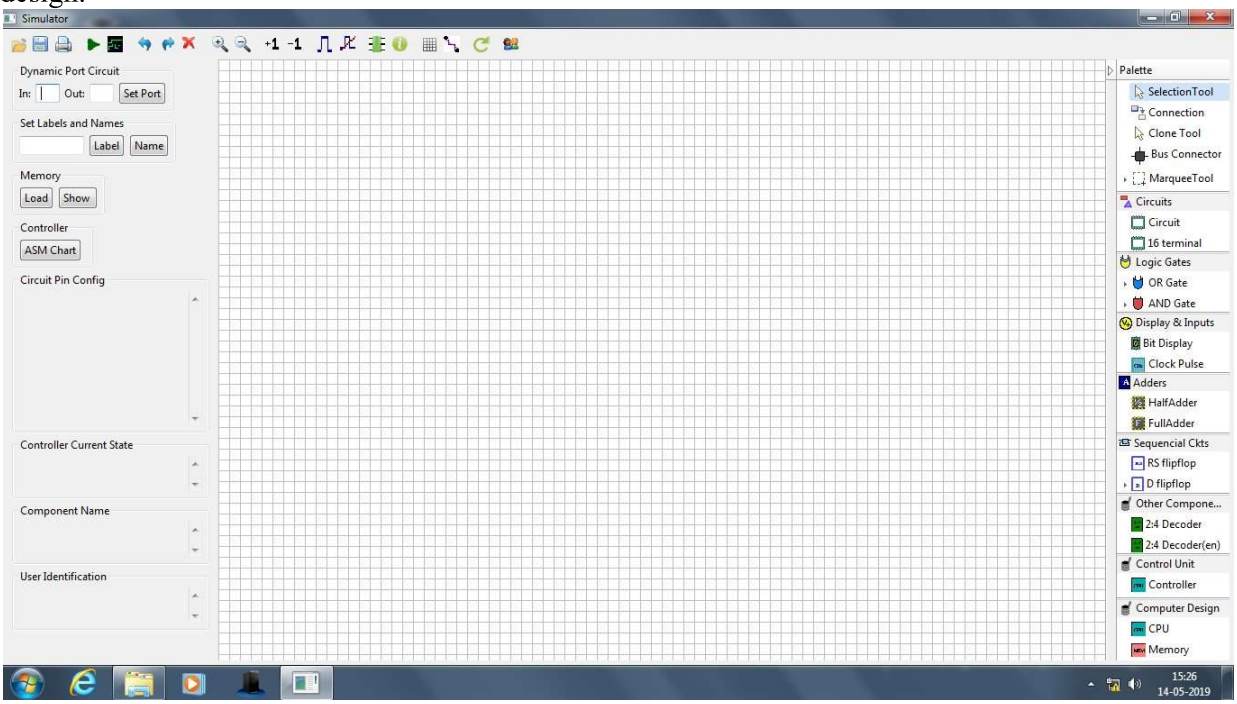


Figure 1: Main interface of the simulator

Features of the simulator:

The main features of the simulators are as follows:

Logic:

The simulator supports 5 valued logic. So the simulator supports wired AND for bus based design.

These 5 states along with their corresponding wire values are as follows:

- ***True (T)*** (wire color: blue)
- ***False (F)*** (wire color: black)
- ***High impedence (Z)*** (wire color: green)
- ***Unknown (X)*** (wire color: maroon)
- ***Invalid (I)*** (wire color: orange)

Truth table for 5 valued logic used in the simulator:

	0	1	X	Z	I
0	0	0	0	0	0
1	0	1	X	X	X
X	0	X	X	X	X
Z	0	X	X	X	X
I	0	X	X	X	X

AND

	0	1	X	Z	I
0	0	1	X	X	X
1	1	1	1	1	1
X	1	X	X	X	X
Z	1	X	X	X	X
I	1	X	X	X	X

OR

	0	1	X	Z	I
0	1	1	1	1	1
1	1	0	X	X	X
X	1	X	X	X	X
Z	1	X	X	X	X
I	1	X	X	X	X

NAND

	0	1	X	Z	I
0	1	0	X	X	X
1	0	0	0	0	0
X	X	0	X	X	X
Z	X	0	X	X	X
I	X	0	X	X	X

NOR

	0	1	X	Z	I
0	0	1	X	X	X
1	1	0	X	X	X
X	X	X	X	X	X
Z	X	X	X	X	X
I	X	X	X	X	X

XOR

	0	1	X	Z	I
0	1	0	X	X	X
1	0	1	X	X	X
X	X	X	X	X	X
Z	X	X	X	X	X
I	X	X	X	X	X

XNOR

in	out
0	1
1	0
X	X
Z	X
I	X

NOT

	0	1	X	Z	I
0	Z	0	X	X	X
1	Z	1	X	X	X
X	Z	X	X	X	X
Z	Z	X	X	X	X
I	Z	X	X	X	X

Tri State Buffer

Graphical organization of the simulator:

The simulator contains

- a pallete on the right hand side. This pallete contains all the components and tools. Tools are used to act up on the components.
- a toolbar on the top which contains several buttons. These buttons are:
  - *save/open*

◦ *simulate* (after creating a circuit, this button has to be pressed to simulate the circuit and to get output)

◦ *plot graph* (to plot input-output wave form)

◦ *undo/redo*

◦ *delete*

◦ *zoom in/zoom out*

◦ *increment/decrement LED* (for digital LED which can also be used as input and display)

◦ *start/stop clock* pulse

◦ *to check the name or pin configuration of a component*

◦ *changing connection types*

◦ *checking the user identification*

- a canvas in the middle where the circuits will be designed.
- A toolbar on the left side which contains the following buttons:

  ◦ *set port* to set the number of input and output ports for a circuit.

  ◦ *Set Label* and *set name* to set the label contents and the name of different components.

  ◦ *load memory* to load the memory content to the inbuilt memory (4 bit address and 12 bit data) for performing the computer design experiment. Data can be load either from file or through form.

  ◦ *Show memory* to show the content of the in built memory.

  ◦ *ASM chart* to load the ASM (algorithimic state machine) chart for a controller.

Tools:

- Different tools are:

  ◦ **Selection tool-** used for selecting components

  ◦ **Marquee tool-** used for selecting many components at a time by dragging the mouse in the design area(editor).

  ◦ **Connection tool-** used for connecting components.

  ◦ **Clonning tool –** used to create cloned components.

Components:

Components have been categorized according to their functionality and put into different drawers in the pallete. The area under every drawer is scrollable, if you are unable to see all the components in a particular drawer just click on the area and scroll. Different drawers:

- **Circuits-** contains 8 and 16 terminal circuits and flow container which can hold other circuit components.

- **Logic gates-** contain all kinds of basic logic gates with 2 and 3 inputs.
- **Display and inputs-** contains all kinds of component needed to give input to the circuit along with free running clock and displaying outputs of the circuit.
- **Adders-** contains different types of adder circuits.
- **Sequential ckt-** contains basic flipflops, registers for designing sequential circuits.
- **Other Components-** contains different kinds of components like decoders, multiplexers, arithmetic logic units(ALU), memory elements(RAM cell), cache memory(without any replacement policy)required to design combinational circuits.
- **Control Unit-** contains a controller whose state table(Moore m/c) can be loaded from the interface.
- **Computer Design-** contains a single instruction CPU and a Memory (4 bit address and 12bit data, can be loaded by user)

Building a circuit:

Adding components:

To add the components to the editor select any component(first click on the selection tool then click on the desired component) then finally click on the position of the editor window where you want to add the component.

Pin configuration for a pallete component:

The pin configuration of a component can be seen in two ways:

- selecting the component and press the 'show pinconfig' button in the left toolbar
- whenever the mouse is hovered on any canned component of the palette.

Pin numbering starts from 1 and from the bottom left corner(indicating with the circle) and increases anticlockwise.

Connecting components:

- To connect any two components select the Connection tool in the palette, and then click on the Source terminal and click on the target terminal(no drag and drop, simple click will serve the purpose).
- After the connection is over click the selection tool in the pallete.
- To move any components select the Selection Mode and drag the component after selecting it.
- Users can connect only from a source terminal to the destination terminal. This facility prevents the user from design errors.

- Any input terminal can be connected only once, but any output terminal can be connected to multiple number of terminals.
- For any component, all the upper terminals are input terminals and all lower terminals are output terminals.
- At any time instant, the value of a wire is denoted by the wire color. As the simulator supports 5 valued logic, a wire may have 5 different colors.
- Initially the wire value is set to *unknown*
- specific wire colors for 5 different states have been given at the beginning of this manual.

Other functionalities:

- undo, redo, delete, zoom in, zoom out facilities are provided.
- Users can save their circuits with .logic extension and reuse them.
- User can load data through interface or can load from text file. The text file must contain only binary values. The whole row of a data must be written in separate line. For example:
  - if memory location 0011 should contain a word of 100101111001, then the fourth line of the text file must contain only 100101111001. on other data should be written in that line.

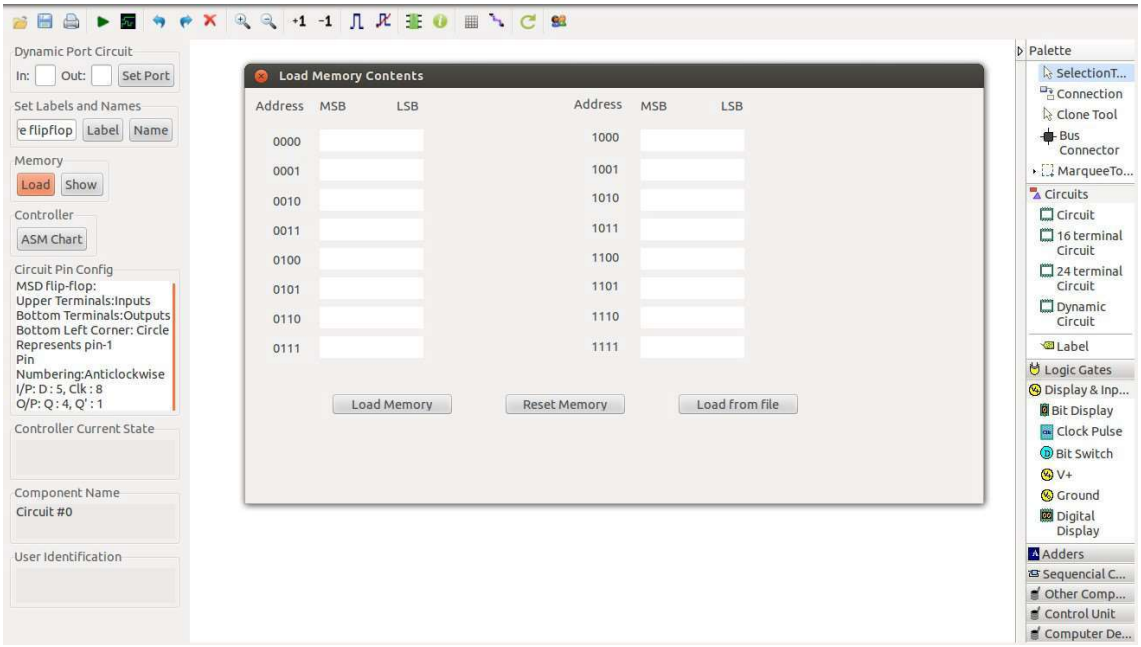


Figure 2. Interface to load memory for computer design experiment

- For loading ASM chart, enter the number of states, inputs and output control signals. The name of the control signals can be changed as per the requirement. Now, the state table can be added using the drop down box as shown in Figure 3.

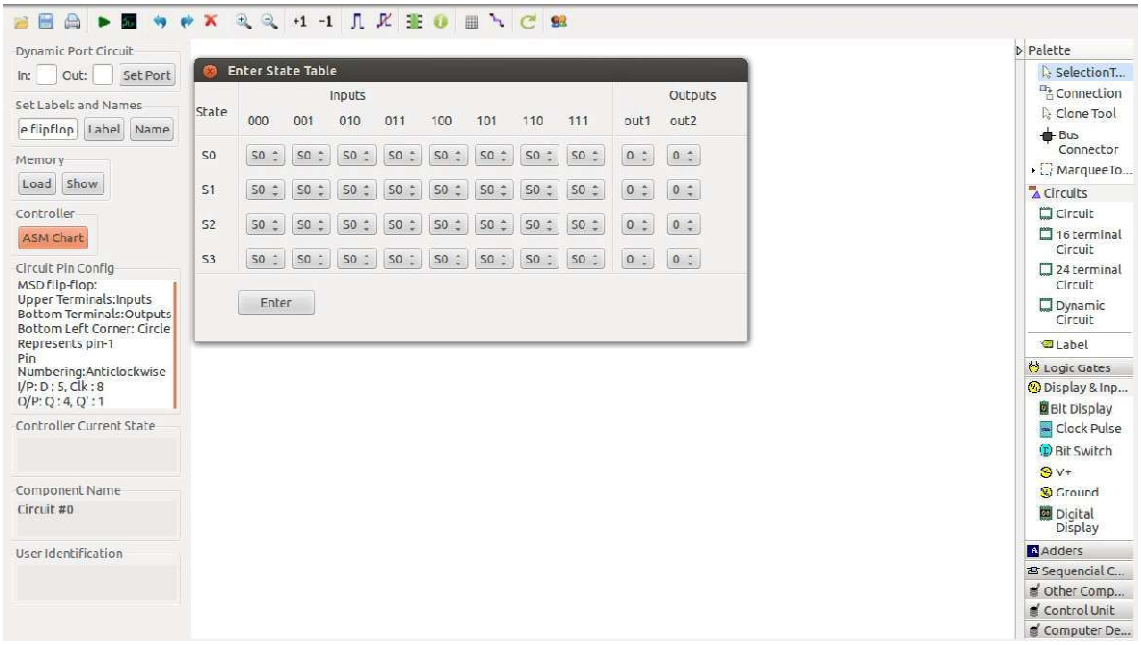


Figure 3. Interface to load state chart (ASM chart) to the controller for computer design experiment

- The circuit drawer contains a *label* component, which can be used to keep any note in the canvas during circuit design.

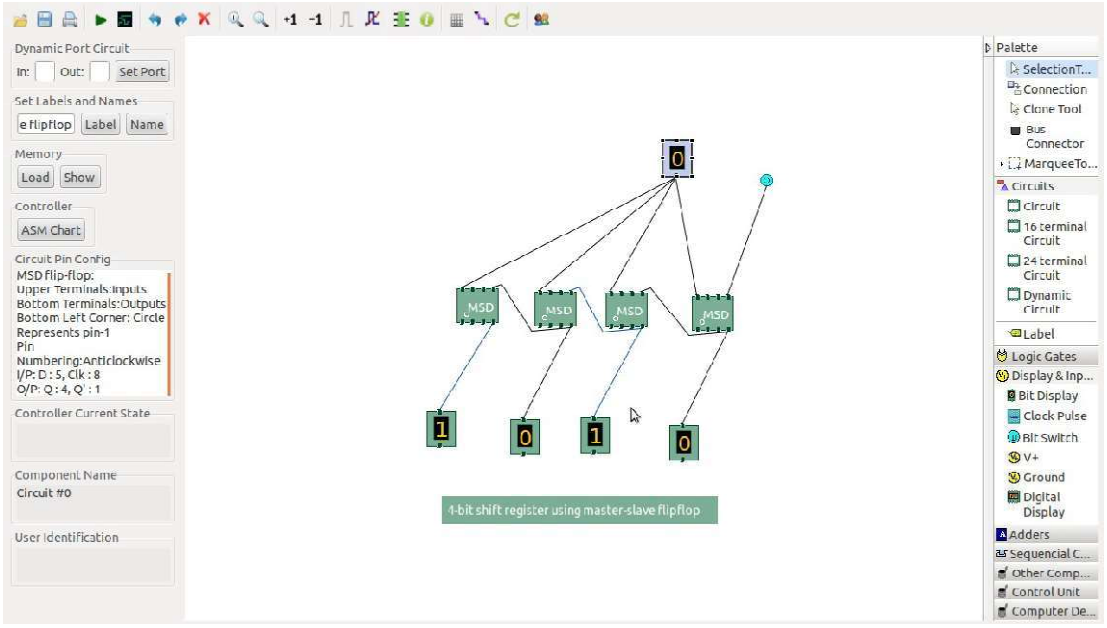


Figure 4. Screenshot of a running 4 bit shift register made up of master slave flipflops with a label at the bottom

Simulating a circuit:

- After building the circuit press the simulate button in the top toolbar to get the output.

- If the circuit contains a clock pulse input, then the 'start clock' button will start the simulation of the whole circuit. Then there is no need to again press the 'simulate' button.
- If the circuit contains multiple clock inputs, then select all of them using control key then press *start clock* button. *Start clock* button will generate continuous pulses to all the selected clock components, if any of the clock pulse is needed to be fixed in the desired value, then press control button and deselect the desired clock component. For starting again the deselected clock component press control key and select it.
- The *plot graph* button will give the input-output wave form of the circuit for a simulation. So if the circuit contains free running clock then the circuit will keep simulating itself and will generate a continuous wave form as shown in the following screenshot:

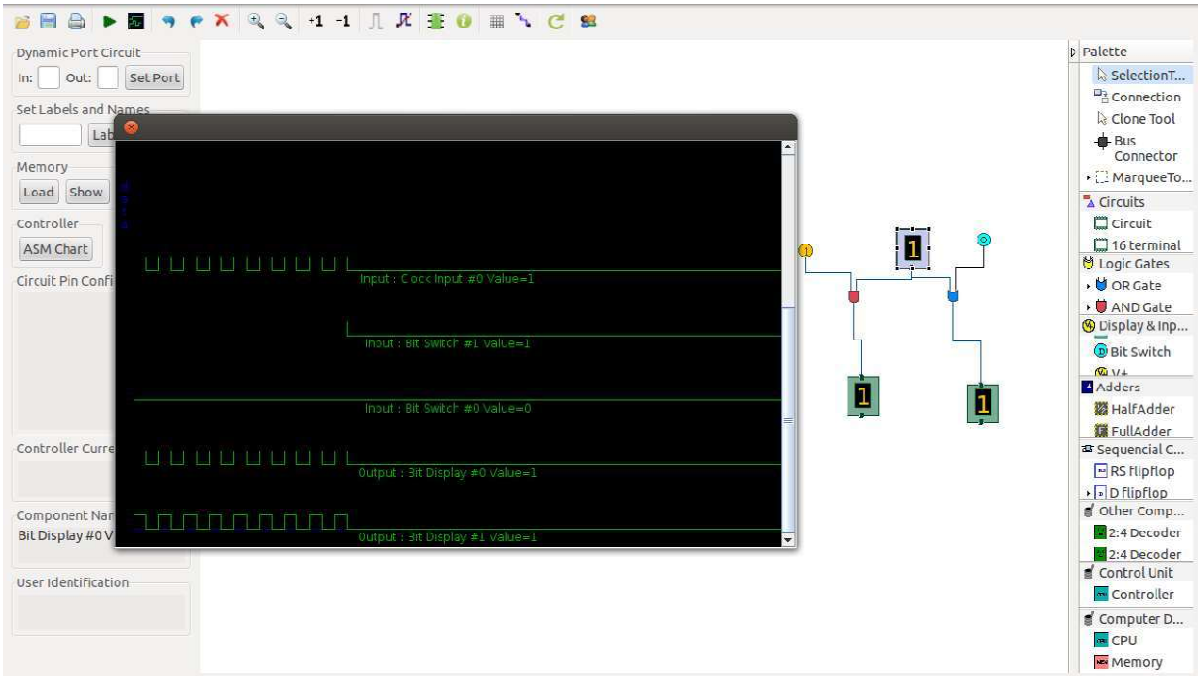


Figure 5. A small circuit with gates, free running clock, bit display and continuous wave form

Limitations:

1. Due to 5 valued logic gate level flipflop design is not supported by this simulator at present time. For designing sequential circuits, all types flipflops are provided as a module.

Description of Components:

General components:

1. **Digital display:** it can be used to give input and as well as to see the output in the decimal format, its right most terminal is the LSB(least significant bit) and the left most terminal is

the MSB(most significant bit), in the editor after selecting a particular digital display you can use 'Increment LED' and 'Decrement LED' buttons in the top left corner of the simulator to increment and decrement its value respectively.

2. **Bit display:** it displays a single bit value.
3. **V+:** it gives 1 as input.
4. **Ground:** it gives 0 as input.
5. **Bit switch:** it gives 1/0 input, it toggels its value with a double click.
6. **Free running clock:** generates clock pulses with clock frequencies among 1000/2000/3000/4000/5000/6000 ns

Specific components:

Pin numbering starts from 1 and from the bottom left corner(indicating with the circle) and increases anticlockwise. Pin configurations of all the components-

1. **Adder drawer:**

1. **Half adder:** i/p: 5, 8
   o/p: sum = 4, carry = 1
2. **Full adder:** i/p: 5, 6, 8
   o/p: sum = 4, carry = 1
3. **RCA 4 bit:** (4 bit ripple carry adder)
   i/p:
   A0 = 13, A1 = 14, A2 = 15, A3 = 16
   B0 = 17, B1 = 18, B2 = 19, B3 = 20
   C0 = 21
   o/p:
   S0 = 12, S1 = 11, S2 = 10, S3 = 9, Cout = 8
4. **Wallace tree adder:** (adds 3 4-bit numbers)
   i/p:
   A0 = 13, A1 = 14, A2 = 15, A3 = 16
   B0 = 17, B1 = 18, B2 = 19, B3 = 20
   C0 = 21, C1 = 22, C2 = 23, C3 = 24
   o/p:
   S0 = 12, S1 = 11, S2 = 10, S3 = 9, S4 = 8 , Cout=7

2. **Sequencial ckt drawer:**

1. **RS flipflop:** i/p: R = 5, S = 8, Clk = 7
   o/p: Q = 4, Q' = 1

2. **D flipflop:** i/p: D = 5, Clk = 8
   o/p: Q = 4, Q' = 1
3. **T flipflop:** i/p: T = 8, Clk = 7
   o/p: Q = 4, Q' = 1
4. **JK flipflop:** i/p: J = 5, K = 8, Clk = 7
   o/p: Q = 4, Q' = 1
5. **MSD flipflop:** Master slave flip flop
   i/p: D = 5, Clk = 8
   o/p: Q = 4, Q' = 1
6. **Shift register:** Shift Register
   i/p: data input =5, Clk=8
   o/p: data-output : 4

3. Other components drawer:

1. 2:4 Decoder:

i/p: A0 = 5, A1 = 7

o/p: D0 = 4, D1 = 3, D2 = 2, D3=1

2. 2:4 Decoder with enable:

i/p: A = 6, B = 5, Enable = 8

o/p: D0 = 4, D1 = 3, D2 = 2, D3 = 1

3. 4:1 Mux:

i/p:

I0=9, I1=10, I2=11, I3=12

S0=13, S1=14

o/p: F=8

4. Combinational Multiplier:

i/p:

multiplicand: A0 = 13, A1 = 14, A2 = 15, A3 = 16

Multiplier: B0 = 9, B1 = 10, B2 = 11, B3 = 12

o/p:

S0 = 8, S1 = 7, S2 = 6, S3 = 5, S4 = 4, S5 = 3, S6 = 2, S7 = 1

5. ALU 1 bit:

i/p:

A0 = 9, B0 = 10, C0 = 11

S0 = 12, S1 = 13

o/p:

F = 8, Cout = 7

6. **4 bit ALU:**

   i/p:

   A0 = 13, A1 = 14, A2 = 15, A3 = 16

   B0 = 17, B1 = 18, B2 = 19, B3 = 20

   C0 = 11

   S0 = 22, S1 = 23

   o/p:

   F0 = 12, F1=11, F2=10, F3=9

   Cout=8

7. **16 bit ALU:**

   i/p:

   A1 = 13, A2 = 15

   B1 = 14, B2 = 16

   Cin = 9

   S0 = 12, S1=11, S2 =10

   o/p:

   Cout = 6, F2 = 7, F1= 8

8. **RAM Cell:**

   i/p = 5

   select = 8

   R/W' = 6

   o/p = 4

   R/W' = 1 for read operation

   R/W' = 0 for write operation

9. **IC Memory:**

   R/W' = 16 Memory Enable = 15

   Address i/p =14, 13

   Data i/p = 12,11,10

   Data o/p = 6, 7, 8

   R/W' = 1 for read operation, R/W' = 0 for write operation

10. **Direct Mapped Cache:**

- pin-32 = S(selects whether user wants to perform cache write or cache mapping)

- pin-31 = R/W'A(selects whether user wants to input the address or cache mapping)
- pin-30 = A3, pin-29 = A2, pin-28 = A1, pin-27 = A0 (this 4 pins are used to give address input). A3 is the most significant bit and A0 is the least significant bit. A3 and A2 will be compared with the tag. A1 and A0 will select the corresponding set.
- Pin-26 = R/W'D(selects whether user wants to input in the set of cache or cache mapping)
- pin-25 = M1, pin-24 = M0 (M1 is the most significant bit and M0 is the least significant bit). These two bits are used for cache writhe purpose, it selects the particular set of which user wants to give inputs to the valid bit, tag bits and data bits.
- Pin-23 = Den(this is an enable input which has to set for any write purpose in the cache).
- Pin-21 = valid bit
- pin-20 = T1, pin-19=T0 (T1 is the most significant bit and T0 is the least significant bit). These are tag bits.
- Pin-18 = D1, pin-17 = D0 (D1 is the most significant bit and D0 is the least significant bit). These are data bits.
- Pin-14 = Hit/Miss bit (if it gives 1 then hit otherwise miss)
- pin-15 = F1, pin-16 = F0 (F1 is the most significant bit and F0 is the least significant bit). These are output data bits and will be given only when there is a hit.
- Essential pin configurations for writing in cache:
  S = 1, R/W'A = 0, R/W'D = 0, Den = 1
- Essential pin configurations for cache mapping:
  S=0, R/W'A = 1, R/W'D = 1, Den = 0

12. **Associative cache:** Associative Cache (memory address 4 bit, 2 bit data)

- Upper Terminals: Inputs
- Bottom Terminals: Outputs
- Bottom Left Corner: Circle Represents pin-1
- Pin Numbering: Anticlockwise
- S = 32
- R/W'A = 31
- A3 = 30, A2 = 29, A1 = 28, A0 = 27
- R/W'D = 26
- M1 = 25, M0 = 24
- Den = 23
- V = 21, T1 = 20, T0 = 19, D1 = 18, D0 = 17

o/p:

Hit/Miss bit = 14, F1 = 15, F0 = 16

- for cache write: S = 1, R/W'A = 0, R/W'D = 0, Den = 1
- for cache map: S = 0, R/W'A = 1, R/W'D = 1, Den = 0
- A3, A2, A1, A0 are memory address
- Conventions followed: A3 = MSB, A0 = LSB
- M1, M0 selecting set for cache write
- v: valid bit
- T1, T0 are tag
- D1, D0 are input data
- F1, F0 are output data

- **Computer design drawer:**

1. **CPU design:** Create a single instruction CPU (central processing unit) with 13 input pins and 18 output pins.

- The instruction is subneg (SBN) which is 'subtract and branch if negative'

  SBN a,b,c Mem[a] = Mem[a] – Mem[b]

  if (Mem[a] < 0) then goto c
- This CPU contains a controller. The state chart for the controller has to be input from outside interface.
- input pin (upper) :

  data input = 20-31

  clock input = 19
- output pin(lower) :

  memory enable = 1

  R/W' (read/write) = 2

  address = 3-6

  data output = 7-18

2. **Memory: Create a memory with data bit: 12 and address bit:4**

- input pins **= 18**
- output pins **= 12**
- input ports(upper ports) :

- memory enable = 30
- R/W' = 29
- address = 25-28

- data = 13-24
- output pins(lower) : 1-12 : data output

Module 2:

Combinational

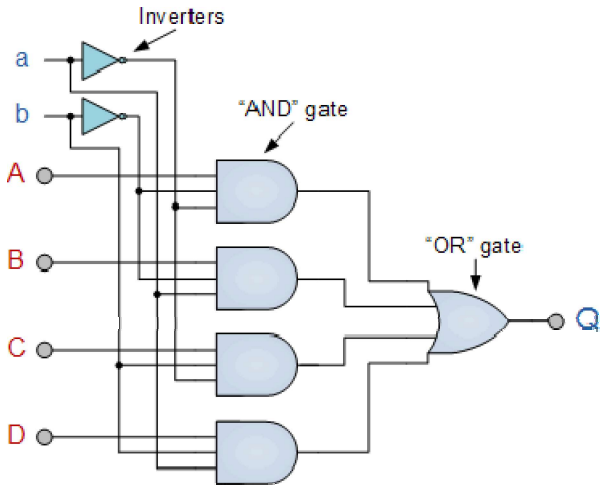
Circuits Simulation

Multiplexer Implementation

Introduction:

The *multiplexer*, shortened to "MUX" or "MPX", is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called "channels" one at a time to the output.

 # Design of 4 Channel Multiplexer using Logic Gates (AND, OR, NOT)



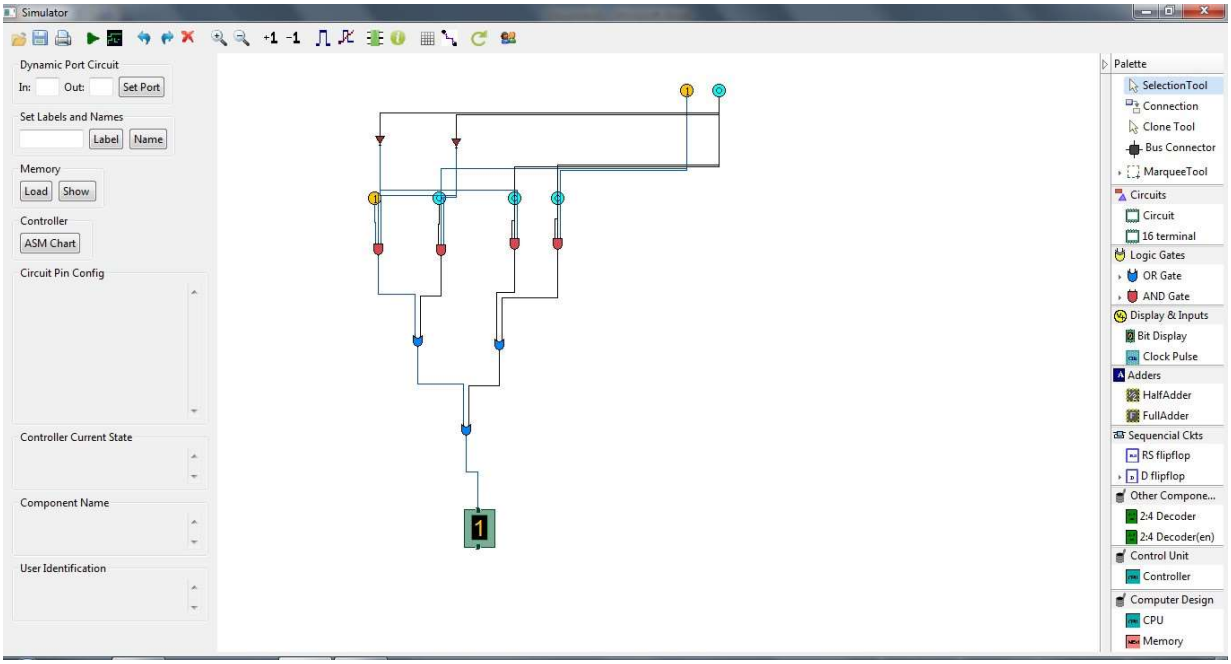
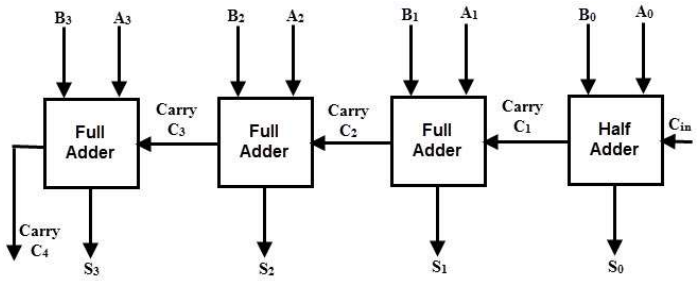


Figure 6 : MUX implementation using logical Gates

Design of 4 bit binary adder using full and half adders

The figure below shows a parallel 4 bit binary adder which has three full adders and one half-adder. The two binary numbers to be added are A3A2A1A0 and B3B2B1B0 which are applied to the corresponding inputs of full adders. This parallel adder produces their sum as C4S3S2S1S0 where C4 is the final carry.



In the 4 bit adder, first block is a half-adder that has two inputs as A0B0 and produces their sum S0 and a carry bit C1. Next block should be full adder as there are three inputs applied to it. Hence this full adder produces their sum S1 and a carry C2. This will be followed by other two full adders and thus the final sum is C4S3S2S1S0.

Most commonly Full adders designed in dual in-line package integrated circuits. A typical 74LS283 is a 4 bit full adder. Arithmetic and Logic Unit of a unit computer consist of these parallel adders to perform the addition of binary numbers.

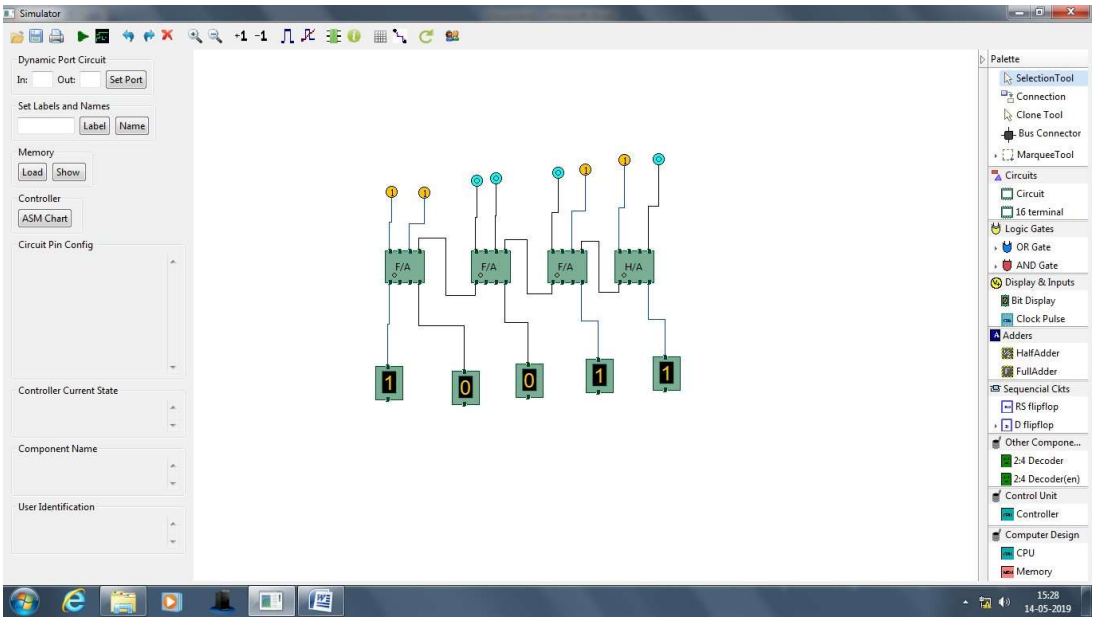
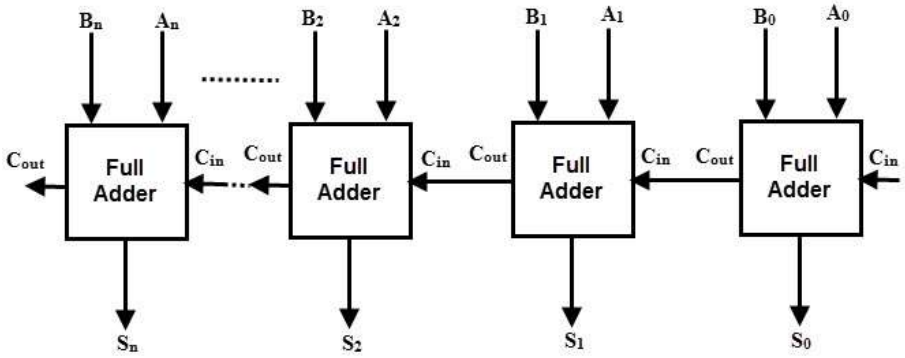


Figure 7 : Design of 4-bit binary adder

Design of Parallel Binary Adders

As we discussed that a single full adder performs the addition of two one bit numbers and an input carry. For performing the addition of binary numbers with more than one bit, more than one full adder is required depends on the number bits. Thus, a parallel adder is used for adding all bits of the two numbers simultaneously.

By connecting a number of full adders in parallel, n-bit parallel adder is constructed. From the below figure, it is to be noted that there is no carry at the least significant position, hence we can use either a half adder or made the carry input of full adder to zero at this position.



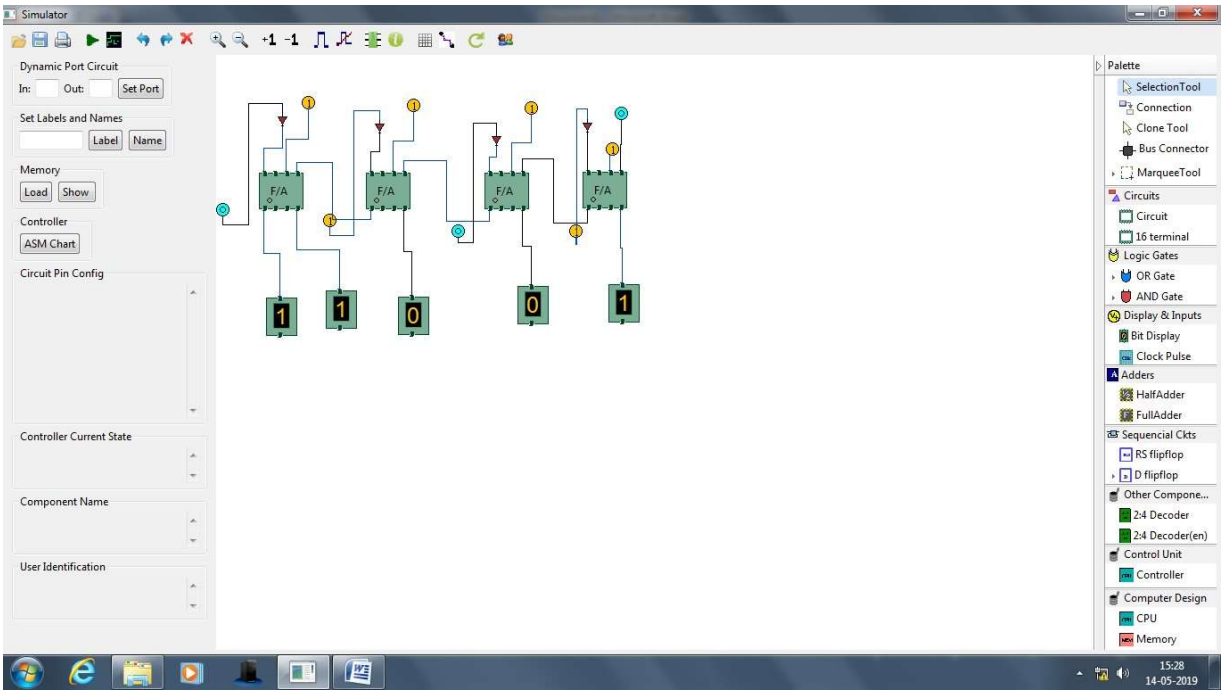


Figure 8: 4-bit binary adder using only full adders

Module 3:

Introduction to 8085 Simulator. Simple Instruction to access Memory, view the Register content, Flag, PC and Stack

Introduction:

The hardware components of the Basic Computer are specified in the Mano book:

1. A memory unit with 64K words of 8- bits each (64KB). (emulated in software)
2. Processor 8 registers (bits): A (Accumulator 8-bits), B (8-bits),C (8-bits),D (8-bits),E(8-bits),F(Flags 8-bits used only 5-bits),H(8-bits),L(8-bits), SP(Stack Pointer 16-bits), PC(Program Counter 16-bits).

   PSW(16-bits) <--- A+F

   HL(16-bits) <--- H+L

   BC(16-bits) <---B+C

   DE(16-bits) <---D+E

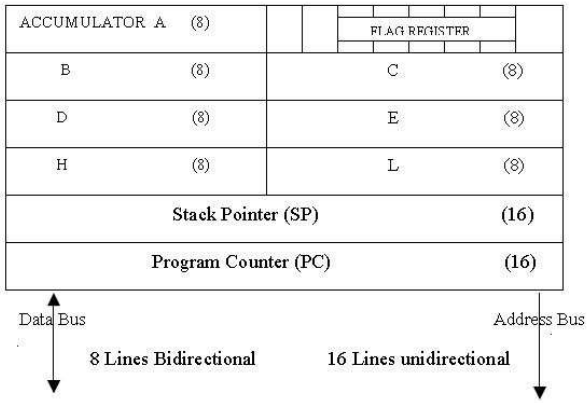


Figure 9: Registers of 8085 Architecture

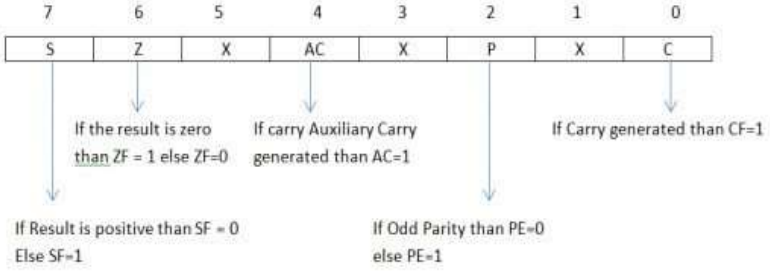


Figure 10: 8085 Flag Register Details

Download 8085 Simulator version 2 designed by Jubin MITRA at the following location:

https://archive.codeplex.com/?p=8085simulator

With the academic learning purpose in mind the 8085 simulator software is designed. It helps in get started easily with example codes, and to learn the architecture playfully. It also provides a trainer kit as an appealing functional alternative to real hardware. The users can write assembly code easily and get results quickly without even having the actual hardware.

The software is shared under **opensource** GNU license.

How to run the Program?

Simply double clicking the program, it should run.

But if it opens like a zip file, then you can be rest assured , that you do not have java installed on your machine.
Then download it from the link given below.

System Requirements:

1. Need Java 6 Update 16 http://javadl.sun.com/webapps/download/AutoDL?BundleId=33889

Features:

The main feature of the 8085 are listed as follows:

Assembler Editor

- Can load Programs written in other simulator
- Auto-correct and auto-indent features
- Supports assembler directives
- Number parameters can be given in binary, decimal and hexadecimal format
- Supports writing of comments
- Supports labelling of instructions, even in macros
- Has error checking facility
- Syntax Highlighting

Disassembler Editor

- Supports loading of Intel specific hex file format
- It can successfully reverse trace the original program from the assembly code, in most of the cases
- Syntax Highlighting and Auto Spacing

Screenshot of Simulator:

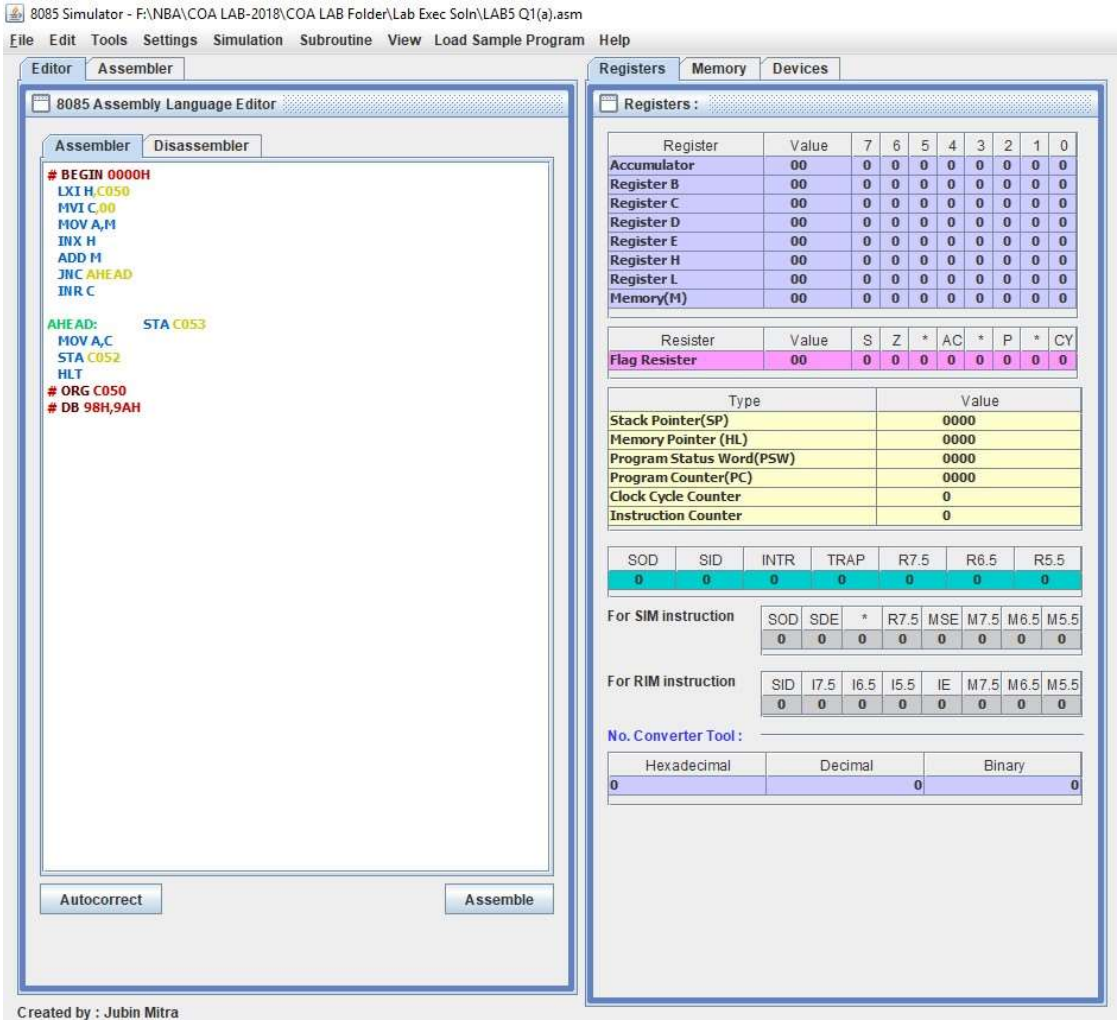


Figure 12: Jubin Mitra 8085 Simultor view

Assembler Workspace

- Contains the Address field, Label, Mnemonics, Hex-code, Mnemonic Size, M-Cycles and T-states
- Static Timing diagram of all instruction sets are supported
- Dynamic Timing diagram during step by step simulation
- It has error checking facility also

Memory Editor

- Can directly update data in a specified memory location
- It has 3 types of interface, user can choose from it according to his need.
    - Show entire memory content
    - Show only loaded memory location

- Store directly to specified memory location
- Allows user to choose memory range

I/O Editor

- It is necessary for peripheral interfacing.
- Enables direct editing of content

Interrupt Editor

- All possible interrupts are supported. Interrupts are triggered by pressing the appropriate column (INTR, TRAP, RST 7.5, RST 6.5, RST 5.5) on the interrupt table. The simulation can be reset any time by pressing the clear memory in the settings tab.

Debugger

- Support of breakpoints
- Step by step execution/debugging of program.
- It supports both forward and backward traversal of programs.
- Allows continuation of program from the break-point.

Simulator

- There are 3 level of speed for simulation:
  - Step-by-step : Automatic line by line execution with each line highlighting. The time to halt at each line is be decided by the user.
  - Normal : Full execution with reflecting intermittent states periodically.
  - Ultimate : Full execution with reflecting final state directly.

- There are 2 modes of simulator engine:
  - Run all at a Time : It takes the current settings from the simulation speed level and starts execution accordingly.
  - Step by Step : It is manual mode of control of FORWARD and BACKWARD traversal of instruction set. It also displays the in-line comment if available for currently executed instruction.
  - Allows setting of starting address for the simulator
  - Users can choose the mnemonic where program execution should terminate

Helper

- Help on the mnemonics is integrated
- CODE WIZARD is a tool added to enable users with very little knowledge of assembly code could also 8085 assembly programs.
- Already loaded with plenty SAMPLE programs
- Dynamic loading of user code if placed in user_code folder
- It also includes a user manual

Printing

- Assembler Content
- Workspace Content
- Register Bank --> Each register content is accompanied with its equivalent binary value*
    - Accumulator, Reg B, Reg C, Reg D, Reg E, Reg H, Reg L, Memory (M)
    - Flag Register
    - Stack Pointer (SP)
    - Memory Pointer (HL)
    - Program Status Word (PSW)
    - Program Counter (PC)
    - Clock Cycle Counter
    - Instruction Counter
    - Special blocks for monitoring Flag register and the usage of SIM and RIM instruction

Crash Recovery

- Can recover programs lost due to sudden shutdown or crash of application

8085 TRAINER KIT

- It simulates the kit as if the user is working in the lab. It basically uses the same simulation engine at the back-end

TOOLS

- Insert DELAY Subroutine TOOL

- It is a powerful wizard to generate delay subroutine with user defined delay using any sets of register for a particular operating frequency of 8085 microprocessor.

- Interrupt Service Subroutine TOOL
  - It is a handy way to set memory values at corresponding vector interrupt address
- Number Conversion Tool
  - It is a portable inter-conversion tool for Hexadecimal, decimal and binary numbers. So, that user does not need to open separate calculator for it.

Simulator View with code and registers

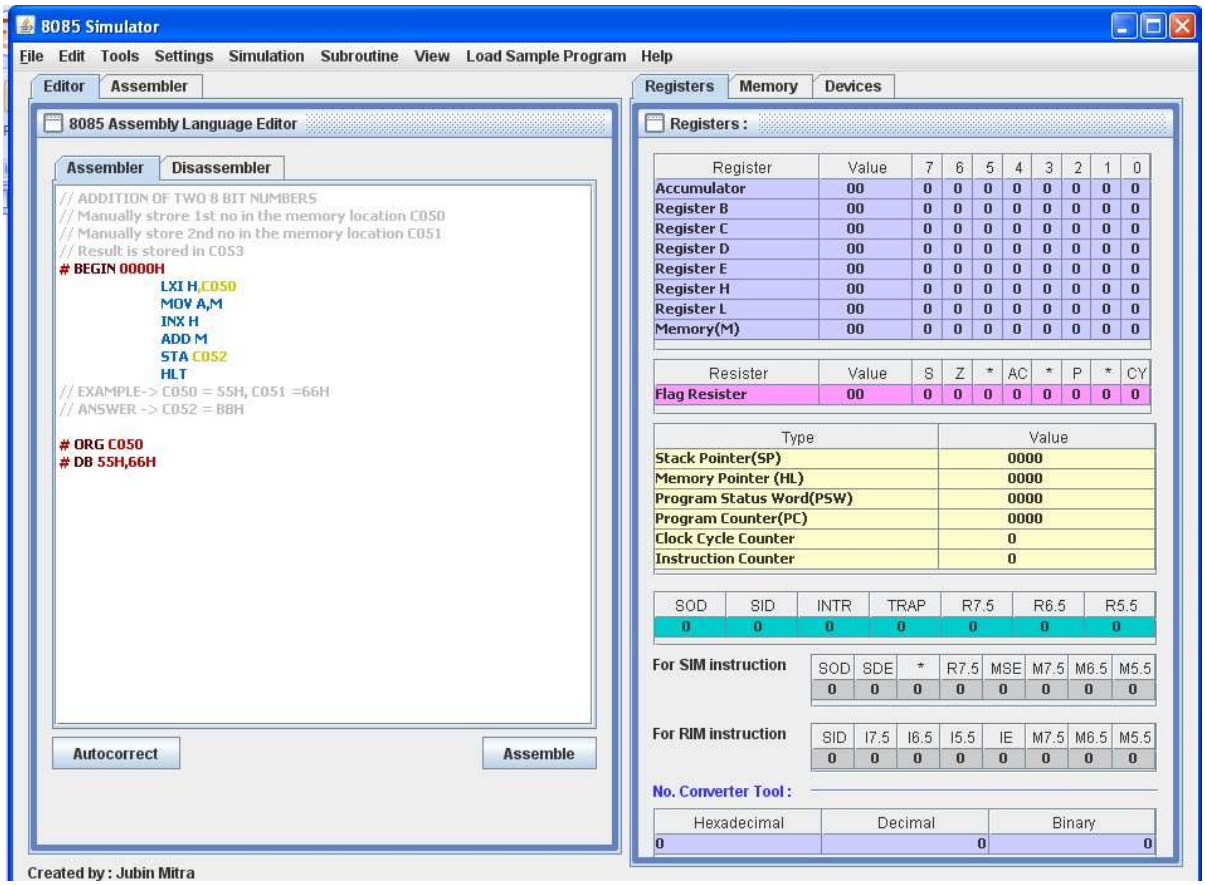


Figure 13: Simulator view with assembly code and register

Assembling the code

After loading code, go to TOOLS from the File Menu and select ASSEMBLE (F7) for assembling the code.

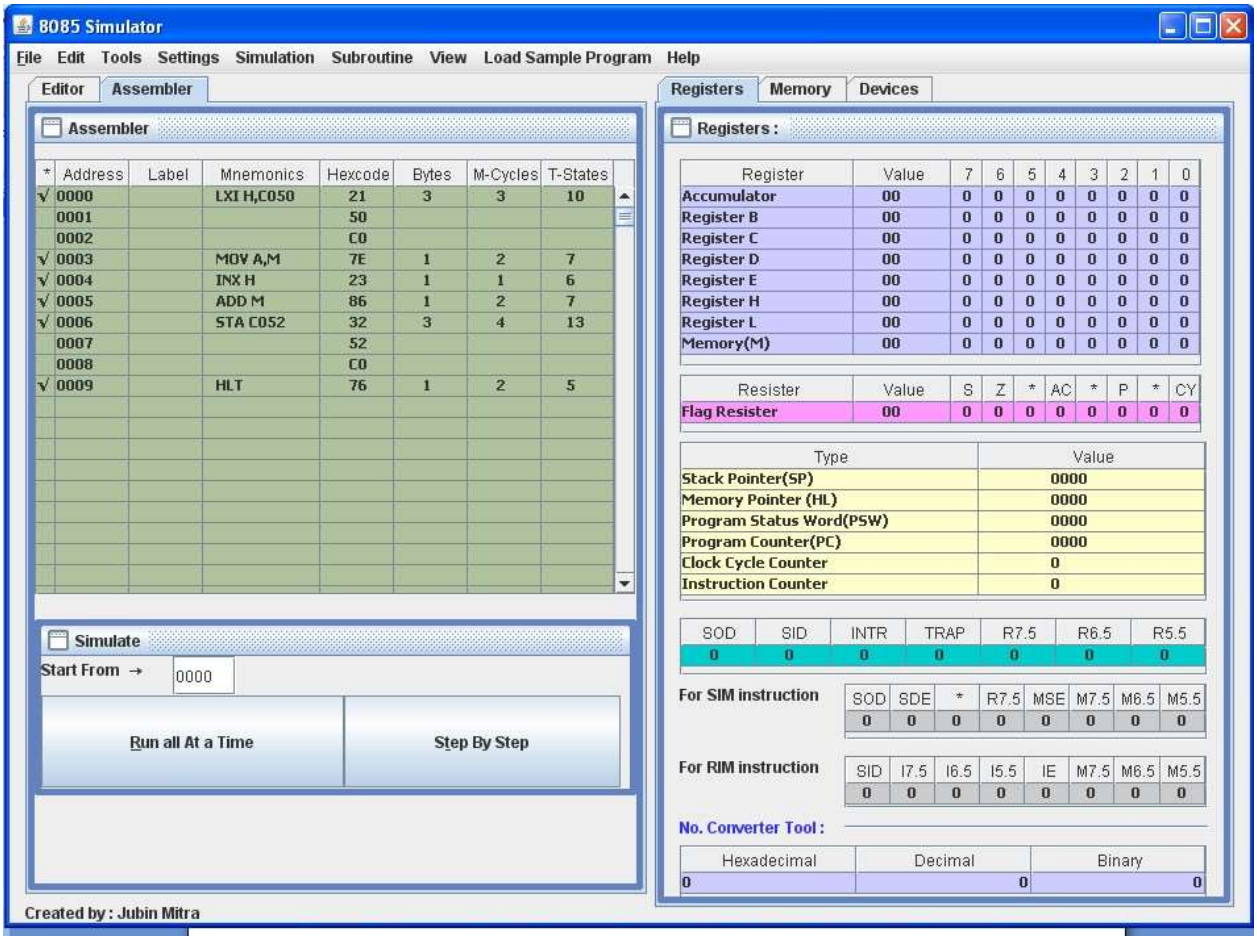


Figure 14: After assembling the code, view of Mnemonics, Hex-code (Machine Code) and Register view

Execution Modes:

Once code is assembled, to simulate and run program we have following two option

1. Run all At a Time
2. Step by Step

Step by Step Running

If you are running program as Novice it's better to go for option "Step by Step" , where you can run program line by line and check the corresponding changes in register and correctness of the program. This execution is used for debugging mode.

Run all At a Time

If you have already executed program once and all syntax errors are removed then "Run all AT a Time" execution mode can be used. Final output of the program can be viewed in this execution mode.

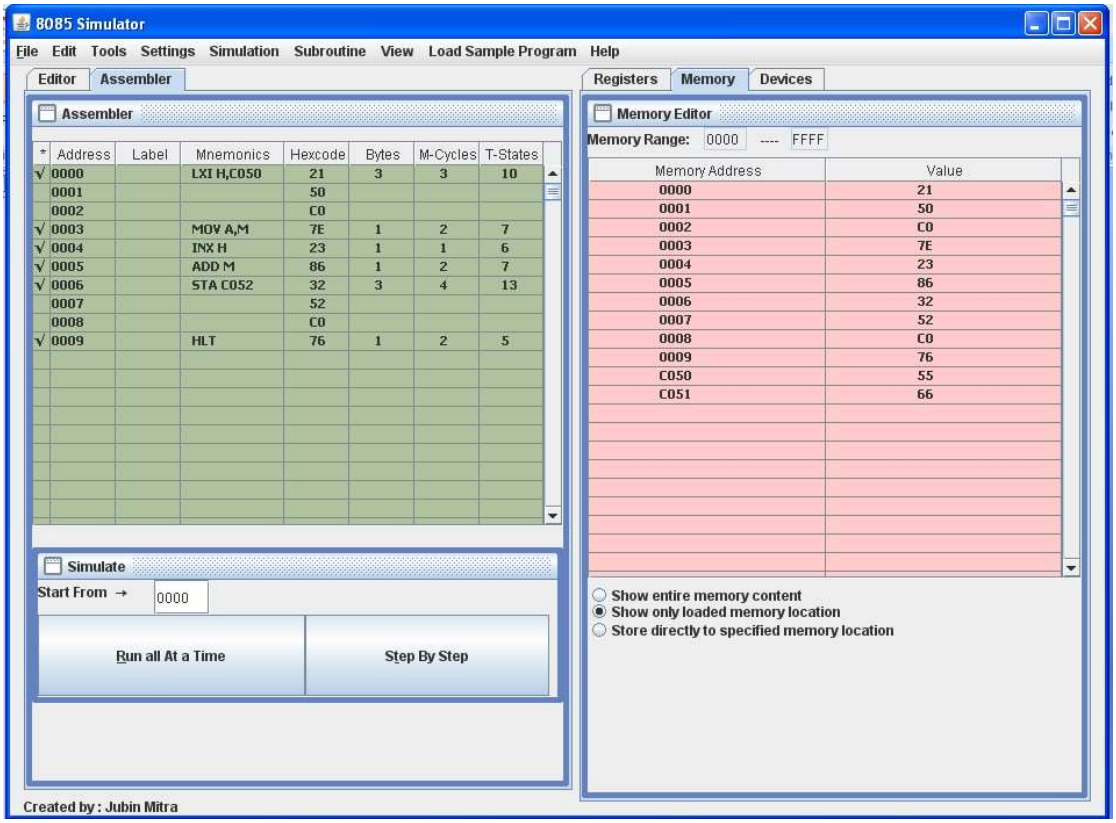


Figure 15: Execution Modes of the Simulator

Memory View of the code

Once the program runs, you can select "Memory" and it has three options to view entire memory, only loaded memory or store at specific location memory.

- Load add two 8-bit numbers from **load sample program** from file menu, assemble and execute it step by step and see the contents of registers and memory.
- Study the basic data transfer instructions of 8085 and perform the following:

Addressing mode Instruction used

Addressing mode used in sample program is

1. **Immediate data transfer instruction**

2. **Register to Register data transfer instruction**

Sample code:

```
BEGIN #0
MVI B,5
MVI A,10
MOV A,B
HLT
```

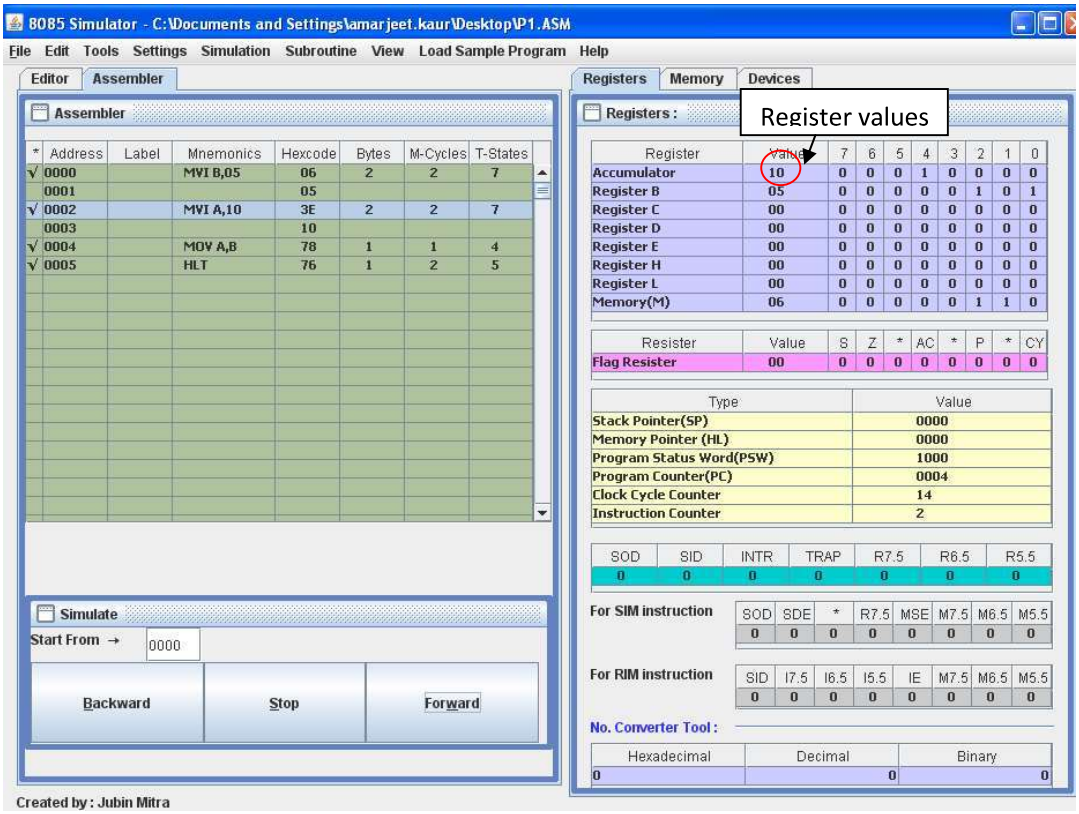


Figure 16: Assembling sample program and viewing the register content

Other Addressing Modes Instruction

1. Mem to reg Example : MOV A,M or M VI A,32H
2. Reg to mem Example : MOV M,A
3. Mem to mem (what syntax error)
4. Load and store instruction Example : STA 2000H, LDA 200H

Flag Registers

Study the basic Arithmetic instruction of 8085 and perform the following and note the changes in the flag register.

ADD instruction simulation

BEGIN #0
MVI B,5
MVI A,10
ADD B
HLT

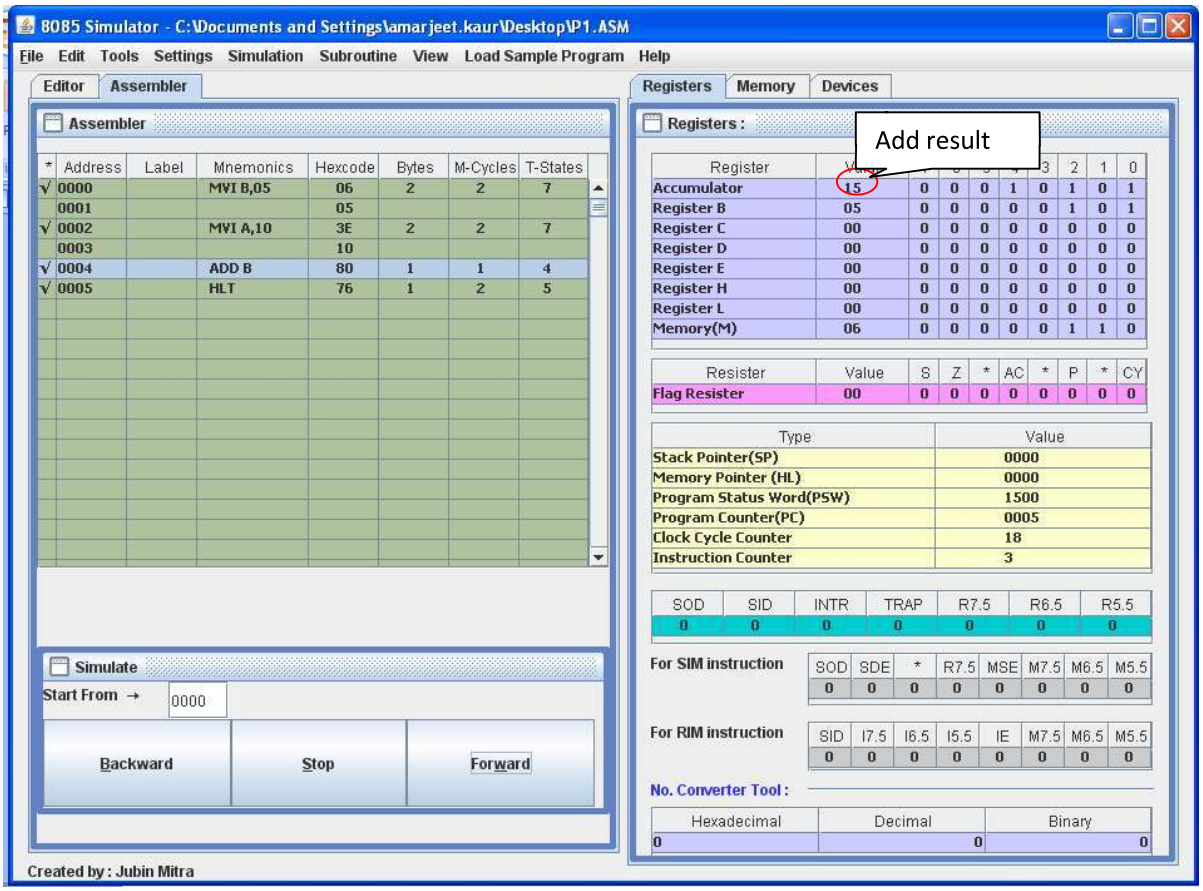


Figure 17: Sample program to add two numbers

Module 4 -8085 Programming to Read and Store Memory

Design 8085 program to read two memory location starting from CO50H and store the result along with at location starting from CO52 H

```
# BEGIN 0000H
   LXI H,C050
   MVI C,00
   MOV A,M
   INX H
   ADD M
   JNC AHEAD
   INR C

AHEAD:
   STA C053
   MOV A,C
   STA C052
   HLT
# ORG C050
# DB 98H,9AH
```

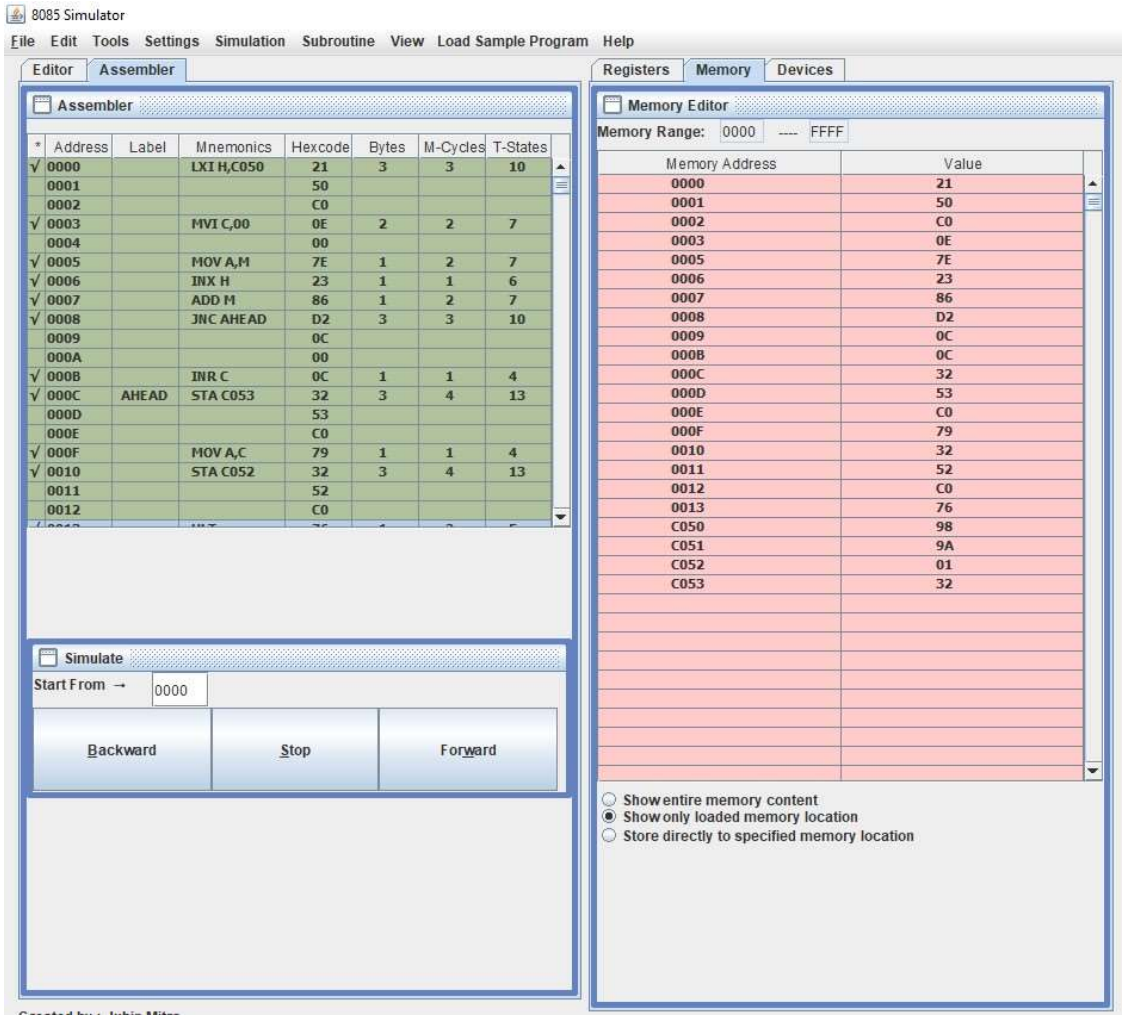


Figure 18 : Memory Read and Write Program

Module 5-8085 Programming to Find the largest of the number, sorting the list and generation Fibonacci series

8085 Program to find largest number in a given array

```
# BEGIN 0000H
        LXI  H, 4200  ;Set pointer for array
         MOV B,M   ; Load the Count
         INX H       ;Set 1st element as largest data
         MOV A,M
         DCR B       ; Decrements the count
LOOP:  INX H
         CMP M        ; f A- reg > M go to AHEAD
         JNC AHEAD
         MOV A,M   ; Set the new value as largest
AHEAD:DCR B
         JNZ LOOP    ; Repeat comparisons till count = 0
         STA 4025    ;Store the largest value at 4300
         HLT
# ORG 4200
# DB 05H,0AH,0BH,02H,06H,01H
```

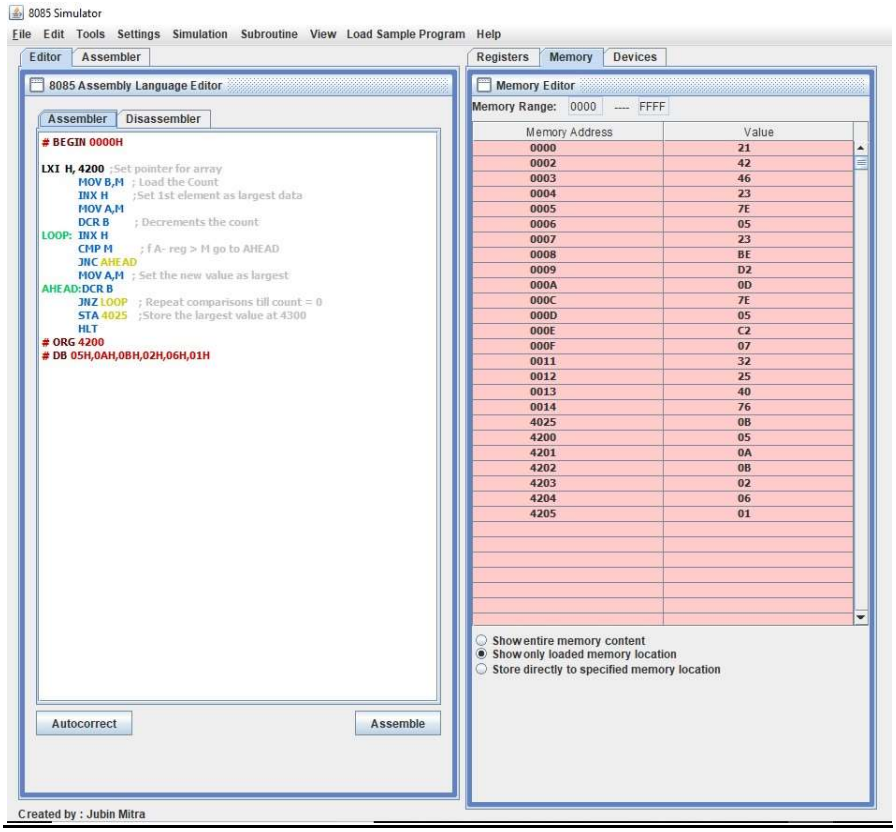


Figure 19: 8085 Program to find largest number

8085 Program to generate FIBONACCI Series and store the result at Memory location C050

```
# BEGIN 0000H
START:        MVI C,09      // Counter
         LXI H,C050    // Memory Pointer

X:       MOV A,M
         INX H
         MOV B,M
         INX H
         ADD B
         DAA
         MOV M,A
         DCX H
         DCR C
         JNZ X
         RST 1
// To run the Program simply load at memory location C050=01,C051=01

# ORG C050
# DB 01H,01H
```

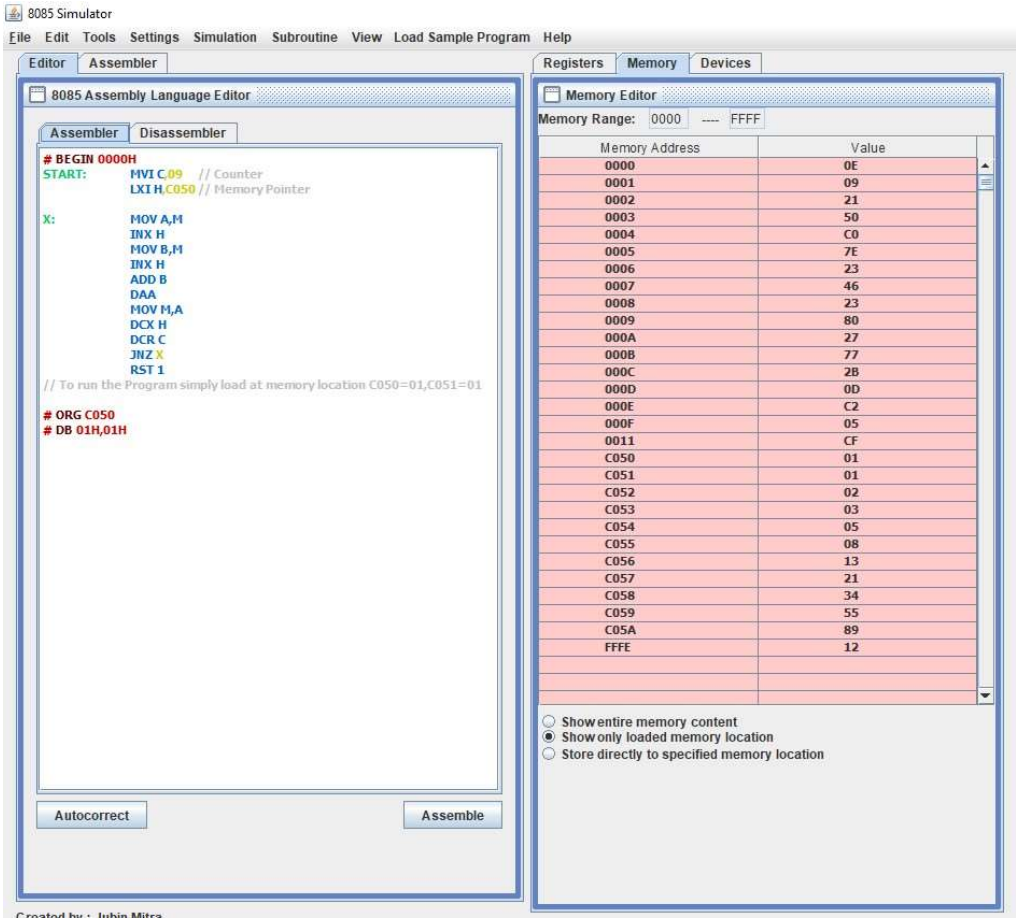


Figure 20: FIBONACCI Series Generation

8085 Program to sort array using BUBBLE sorting

```
// Store from memory location C020 five consecutive array numbers to be
sorted in ascending order

# BEGIN 0000H
START:   MVI D,05     // Counter

W:       LXI H,C020
         MVI C,05     // Counter

X:       MOV A,M
         INX H
         MOV B,M
         CMP B
         JM Y
         MOV M,A
         DCX H
         MOV M,B
         INX H

Y:       DCR C
         JNZ X
         DCR D
         JNZ W
         HLT
// EXAMPLE C020 -> BBH,AAH,99H,88H,77H,66H
// ANSWER C020 -> 66H,77H,88H,99H,AAH,BBH

# ORG C020
# DB BBH,AAH,99H,88H,77H,66H
```

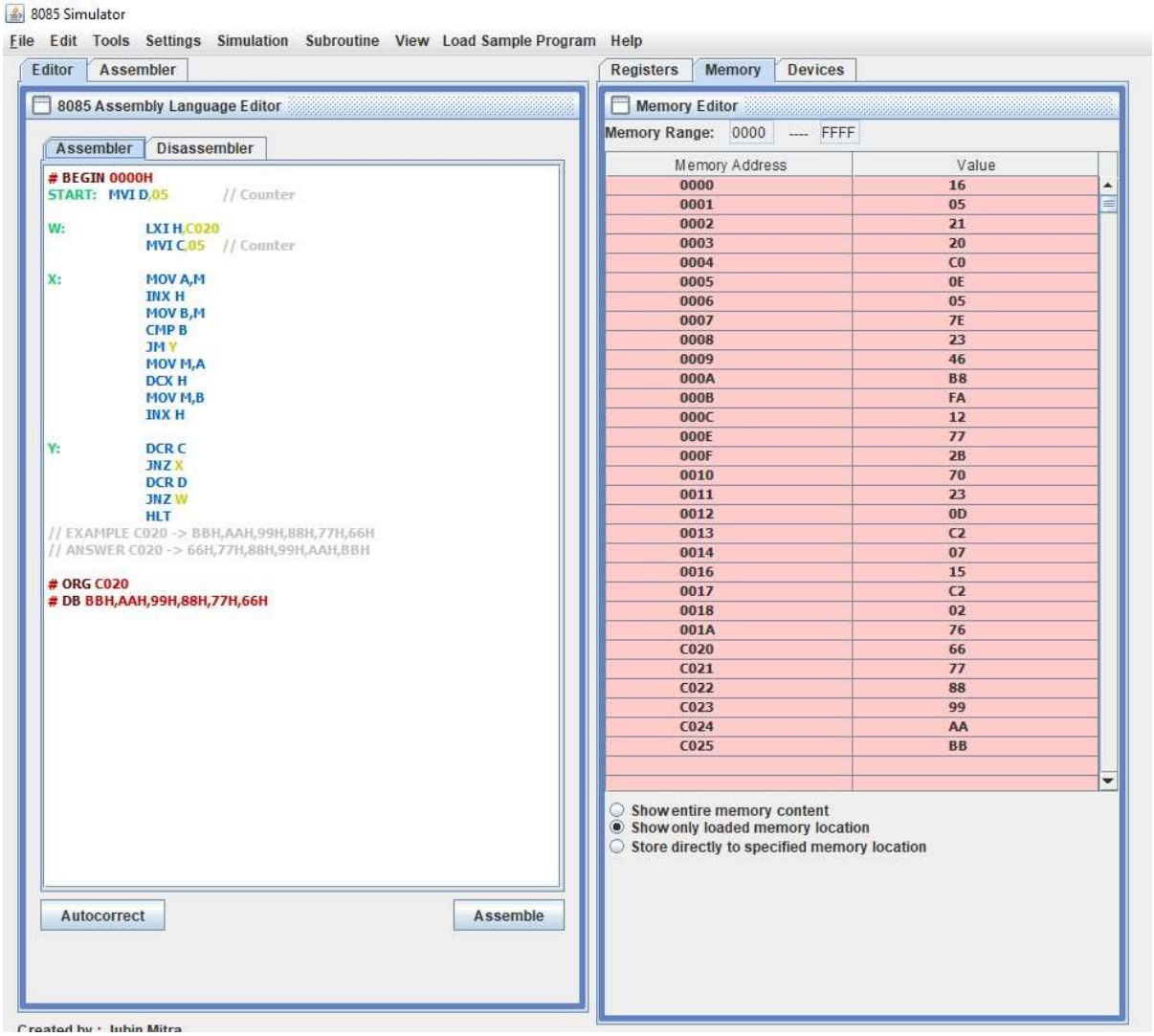


Figure 21 : Sorting of numbers

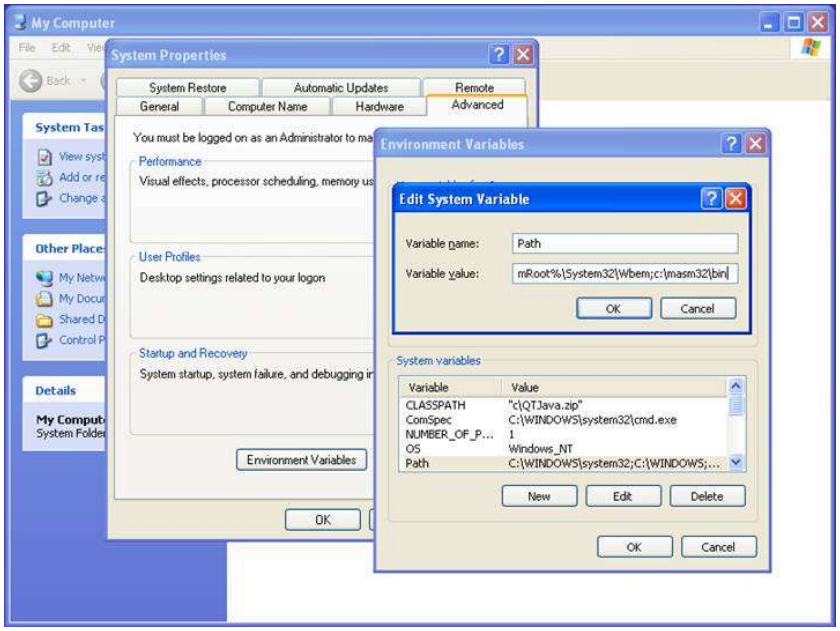
Module 6-
8086(MASM/emu86)
Introduction to
MASM and EMU86
simulators, Basic
programming and
BIOS based programs

Microsoft Assembler (MASM)

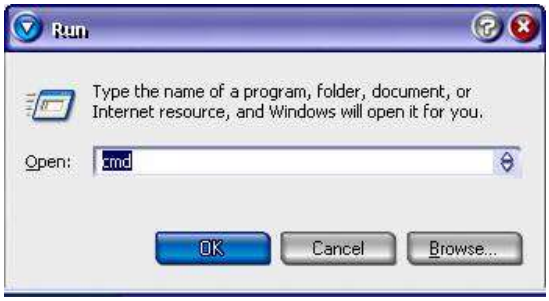
1. Download MASM from http://www.masm32.com/

2. Unzip the package and run install.exe

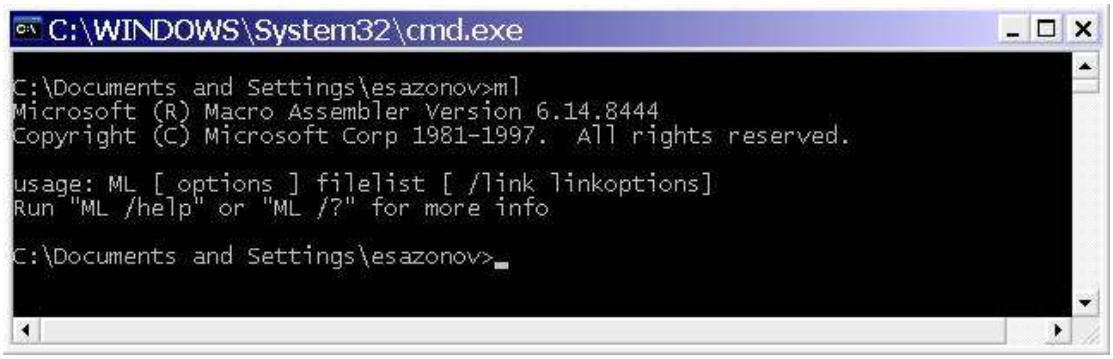
3. Set the path to the compiler. Open “My computer”, right click and select “Properties”. Select “Advanced” -> “Environment variables”->”Path”. Click “Edit” and add “; c:\masm32\bin” to the path



4. Check the installation by opening the command prompt window (Start->Run->cmd)



and typing ML at the command prompt



5. Download 16-bit version of the link.exe from the Microsoft's website

http://download.microsoft.com/download/vc15/Update/1/WIN98/EN-US/Lnk563.exe

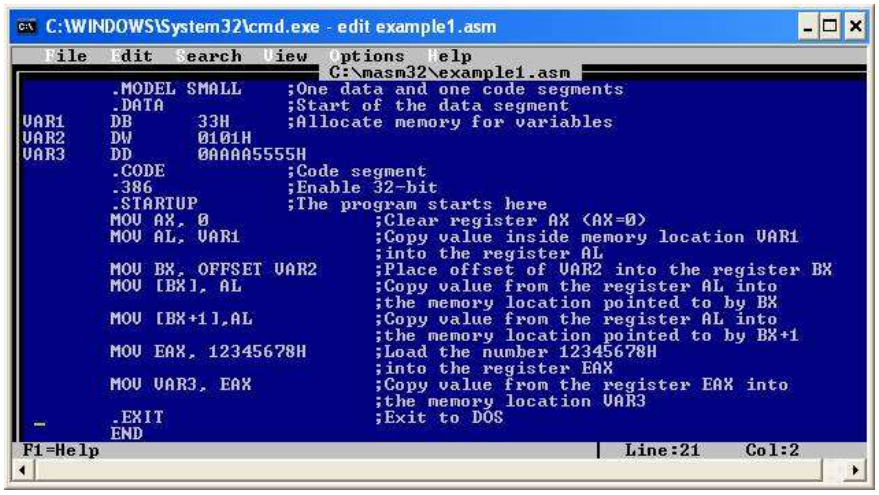
Copy this file to C:\MASM32\BIN and run it. Answer Yes when asked whether to overwrite existing files.

6. Download Code View Debugger

http://www.nuvisionmiami.com/books/asm/cv/cv41patch.exe

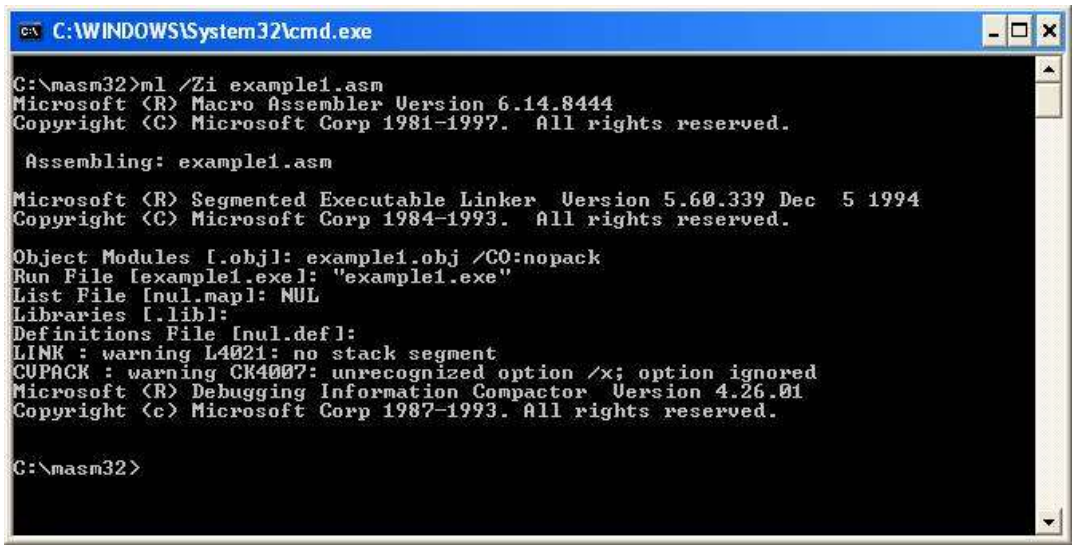
Unzip all files to c:\masm32\bin

7. Now you can use almost any text editor to create an assembly program. In this example, we will use Microsoft's EDIT. Type "edit example1.asm" on the command prompt and enter the text of the program.



Save the file by “Alt-F”,”Alt+S”. Exit “Alt-F”,“Alt-X”

8. Compile and link the assembly file by issuing “ml /Zi example1.asm”

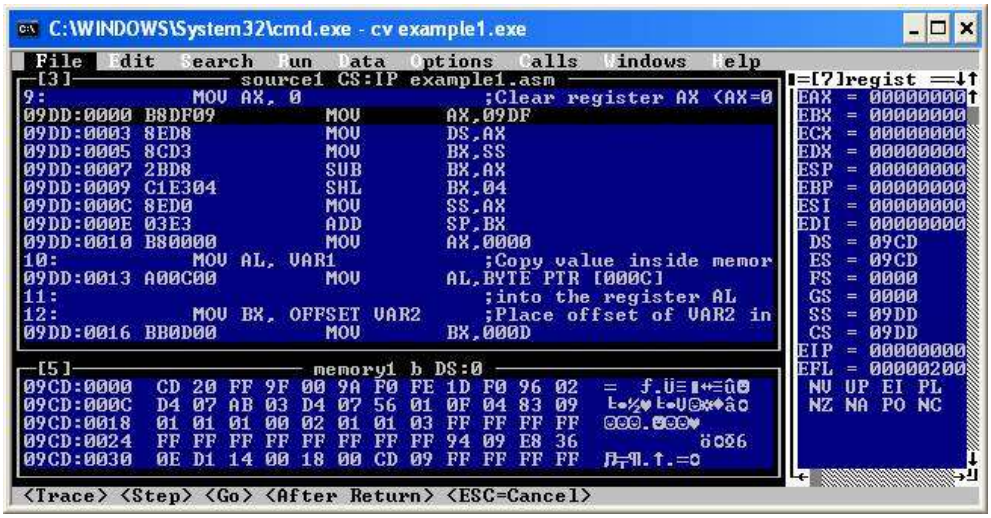


9. Now lets start and configure the Code View debugger. Type “cv example1.exe’ at the command prompt.

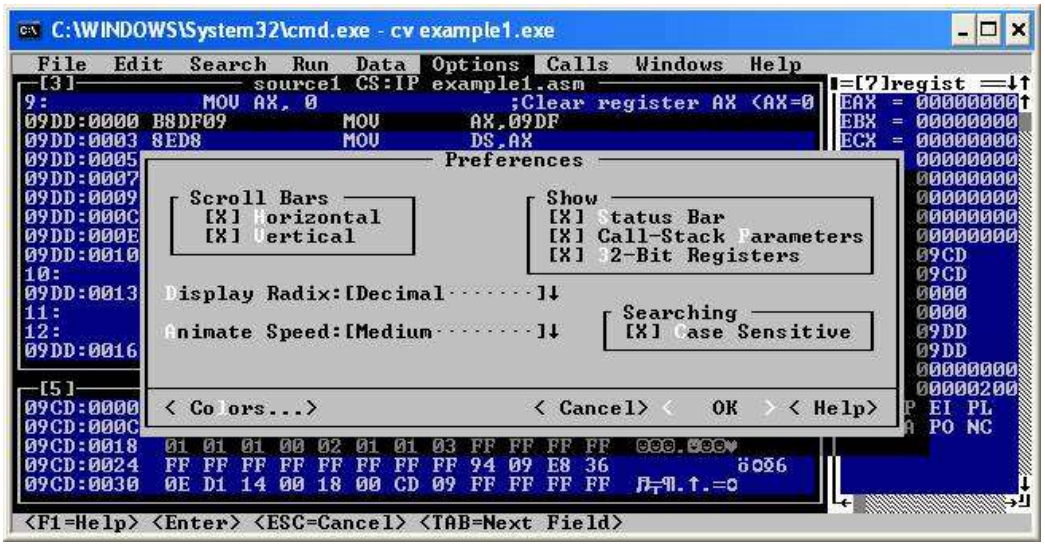
Enter “Alt-W” and make sure that you have the following windows on the screen:

- Code 1
- Registers
- Memory 1

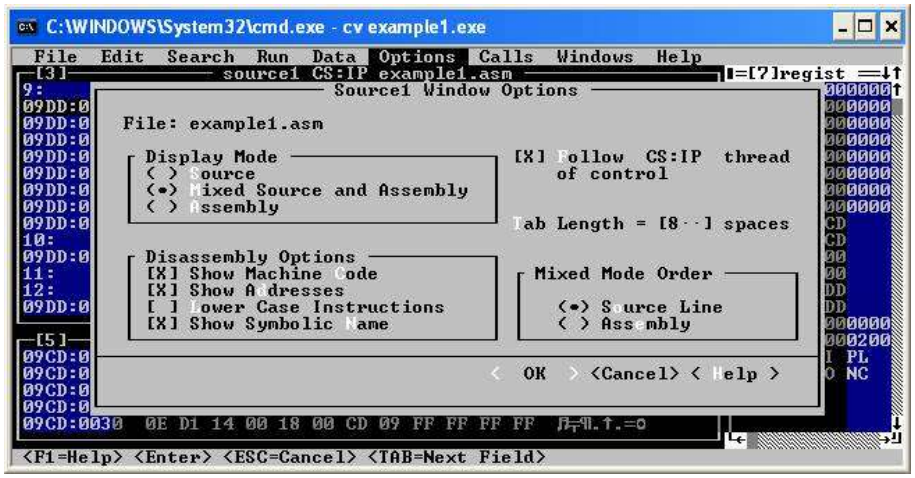
Press “Alt-F5” to arrange the windows on the screen.



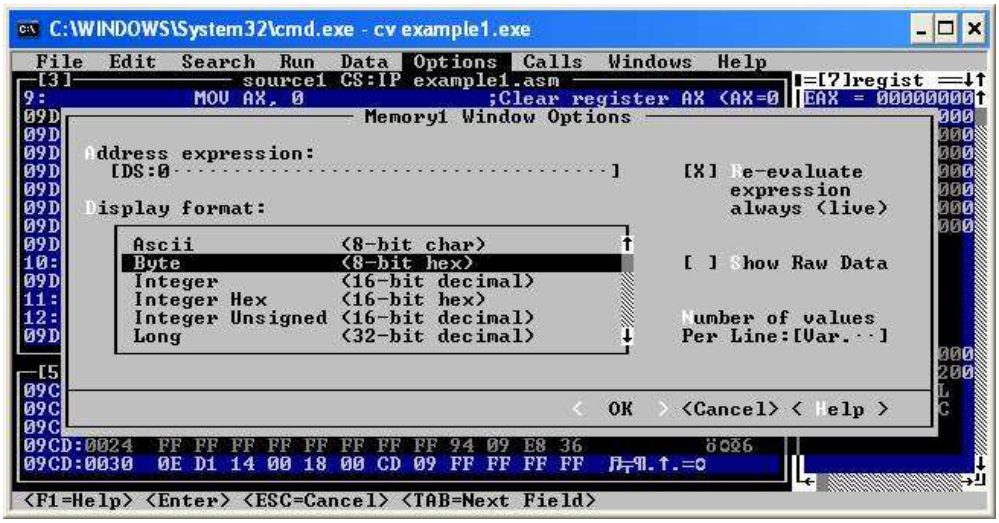
Now lets set the options. “Alt-O” -> Preferences. Set the options as shown and click “ok”.



Again, “Alt-O” -> “Source 1 window”

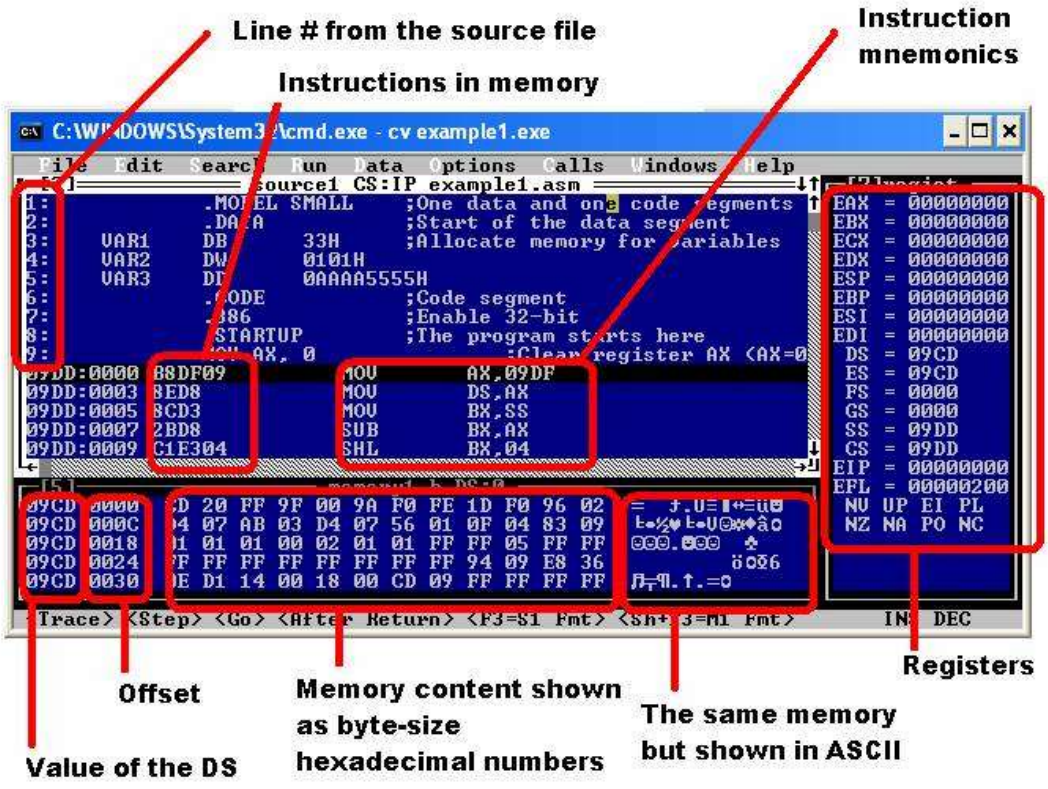


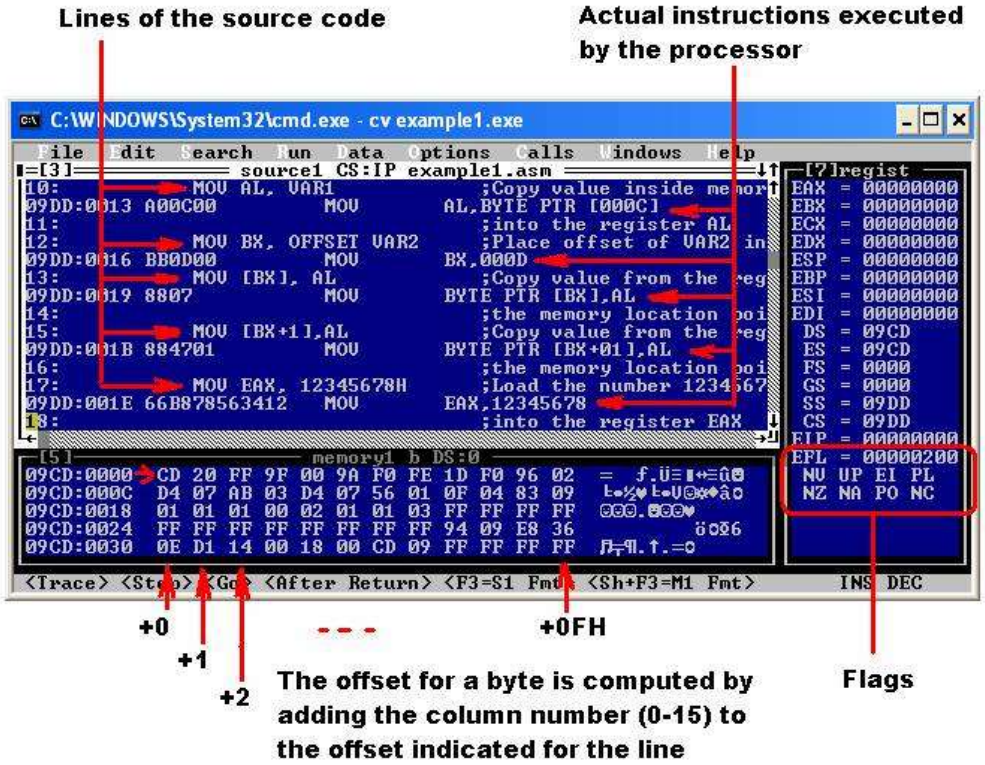
"Alt-O" - >"Memory 1 window"

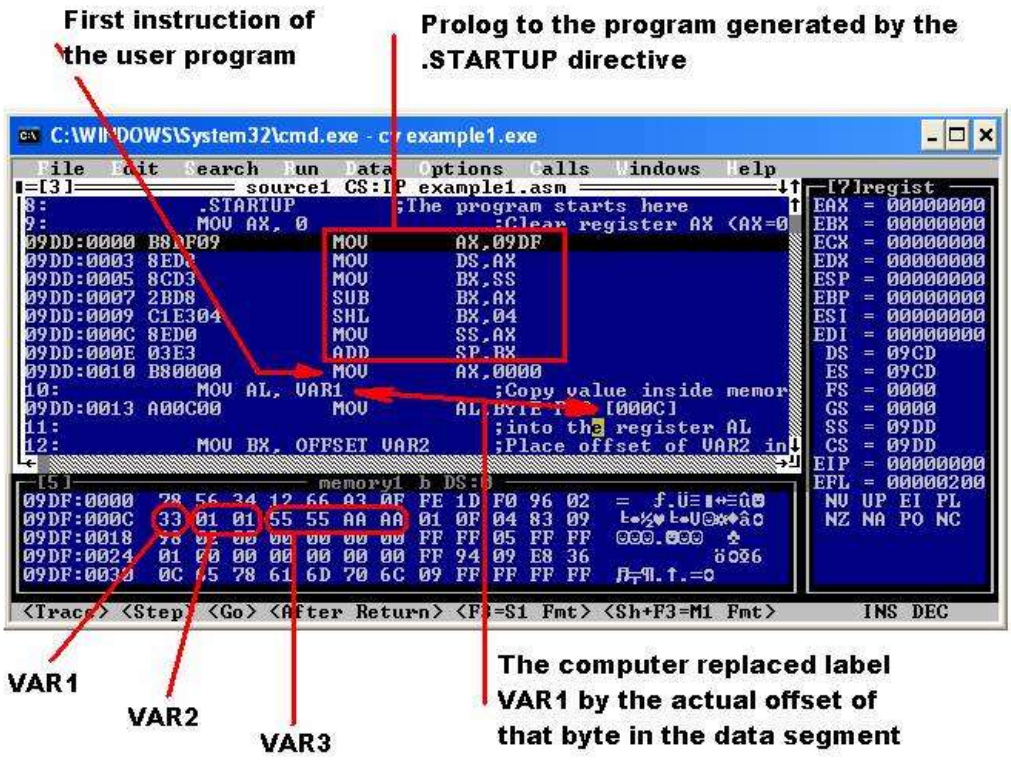


The configuration is now complete.

10. Lets look at the program.

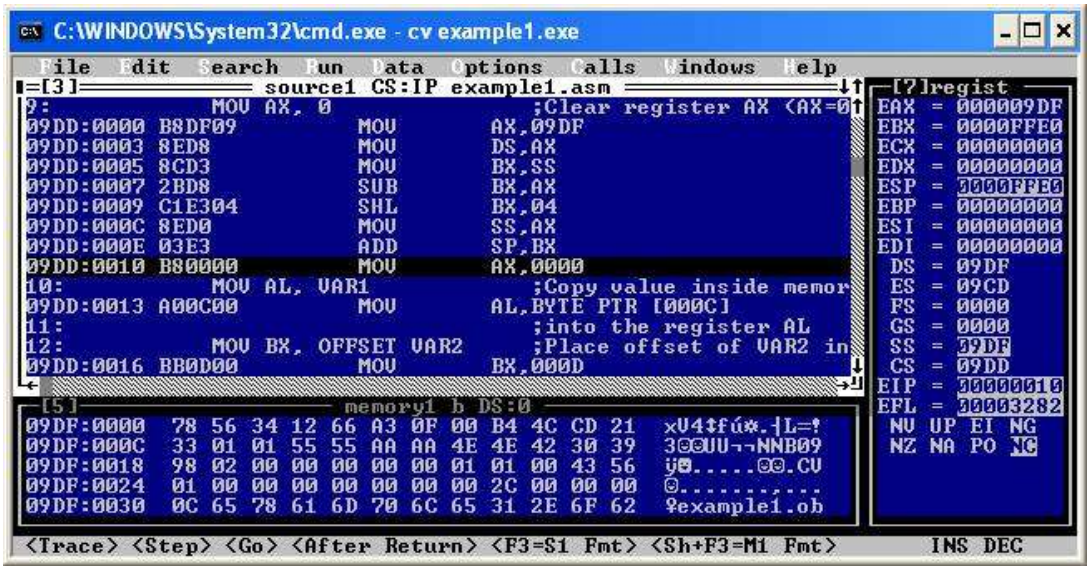

Line # from the source file
Instructions in memory
Instruction mnemonics
Offset
Value of the DS
Memory content shown as byte-size hexadecimal numbers
The same memory but shown in ASCII
Registers


Lines of the source code
Actual instructions executed by the processor
+0
+1
+2
+0FH
The offset for a byte is computed by adding the column number (0-15) to the offset indicated for the line
Flags

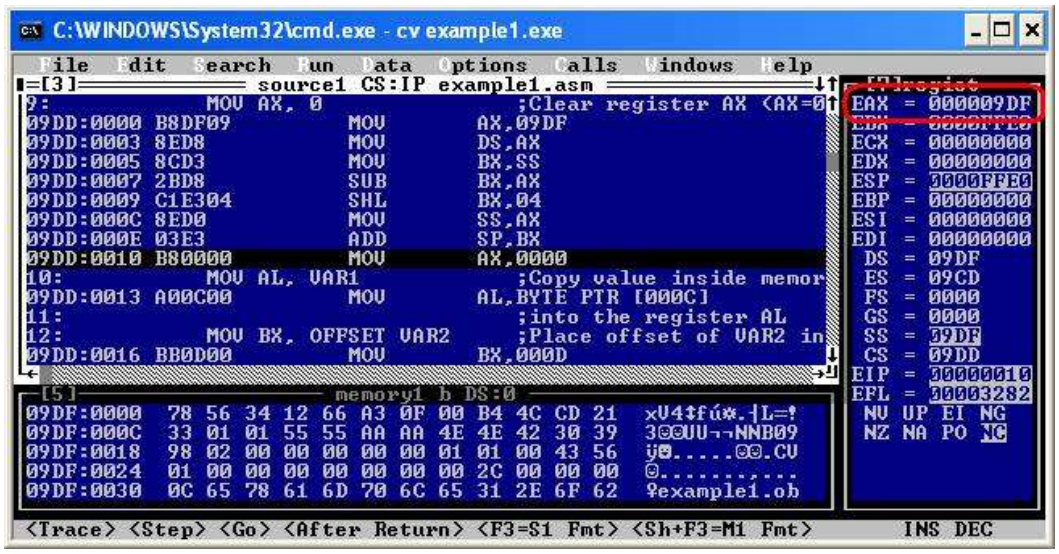


11. Now lets step through the program and observe execution of each instruction.

- Press “F10”.
- The debugger will show execution of the first line of the prolog.
- Press “F10” until instruction “MOV AX,0” is highlighted. This is the first instruction of your program.

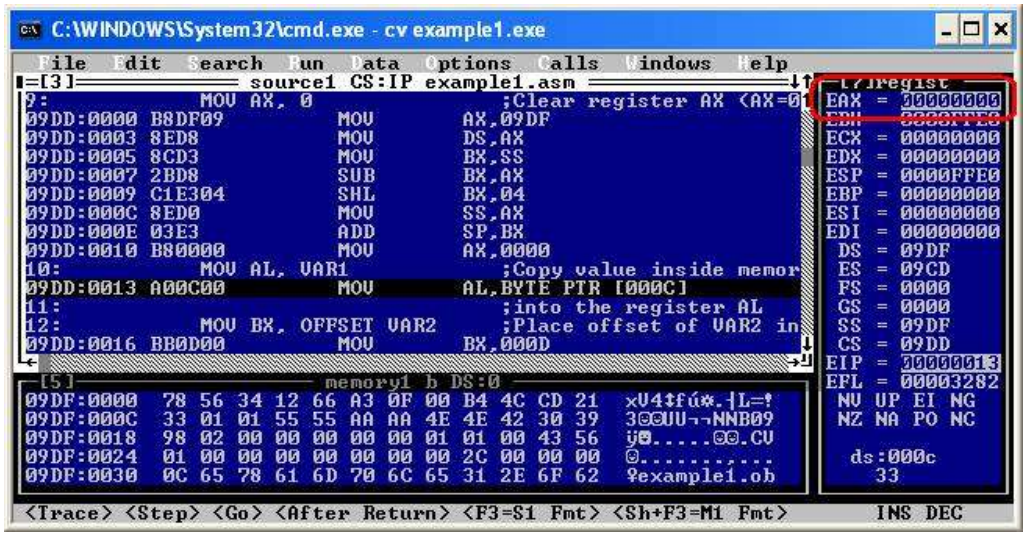


Observe the value in the register AX. Register AX contains number 09DFH.



Now press "F10". The debugger will execute the highlighted instruction.

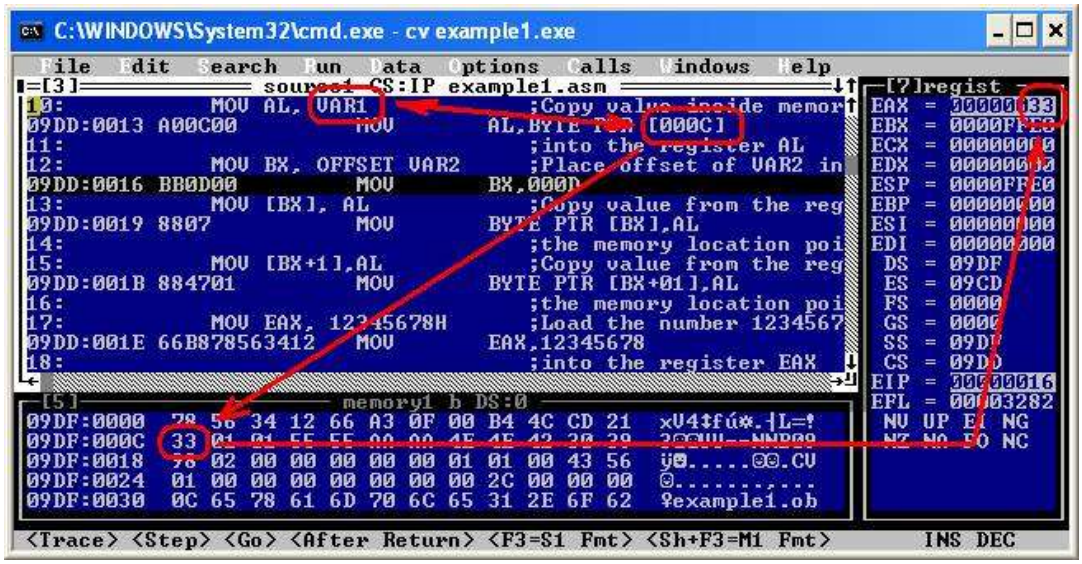
Note the change in the content of EAX and the fact that the register has been highlighted by the debugger, indicating the change.



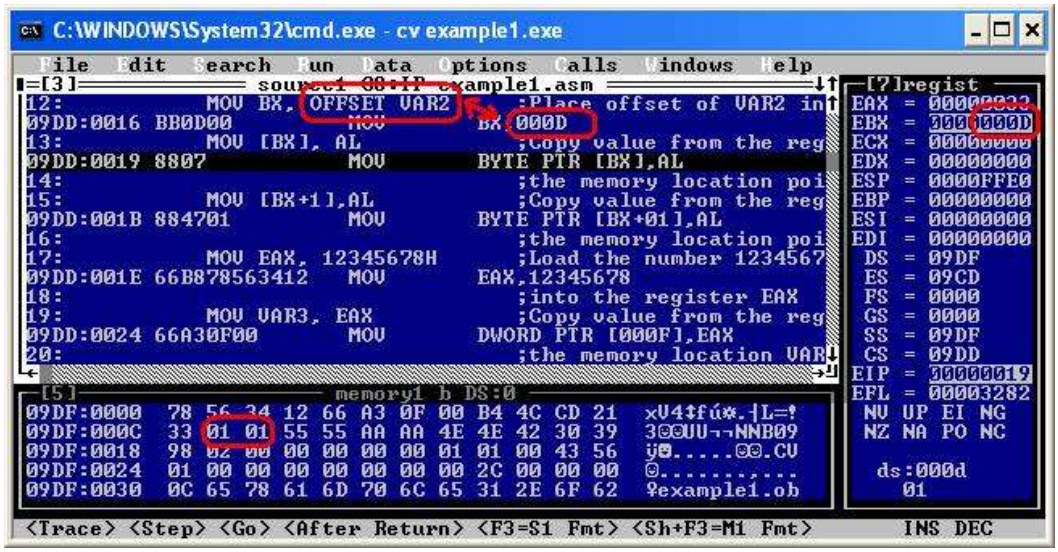
The highlighting the code window moved to the next instruction.

Note that the line of the source code “MOV AL, VAR1” became “MOV AL, [000C] where 000CH is the actual offset of VAR1 in the data segment. You can check that this is true by checking the content of memory location DS:000CH in the data window.

Now execute this instruction by pressing “F10”. Content of the register AL changed, taking the value from the VAR1.



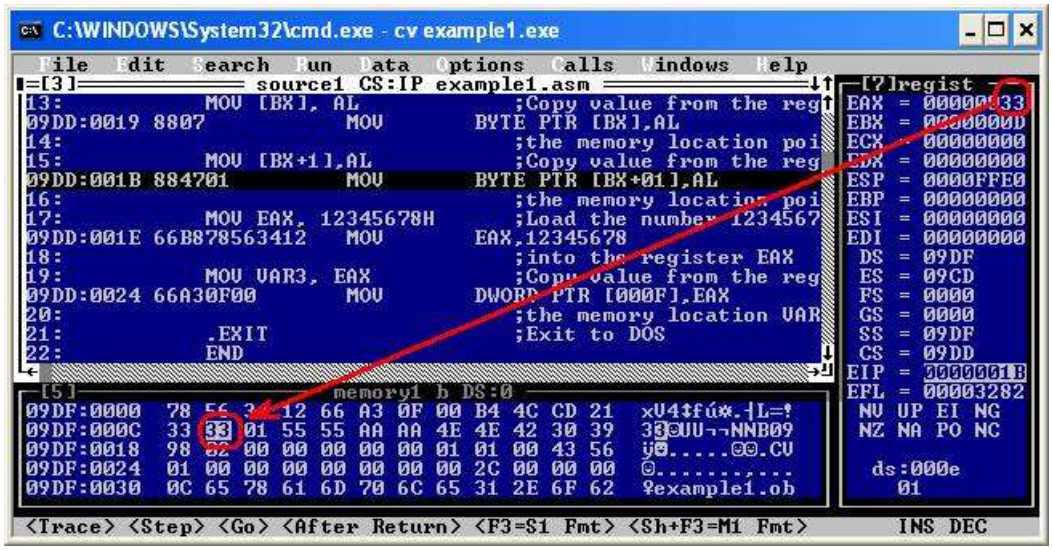
The next instruction is “MOV BX, OFFSET VAR2”. VAR2 follows VAR1 in memory and has offset of 000DH. This is the value that will be placed into the BX upon execution of this instruction. Press “F10” to execute.



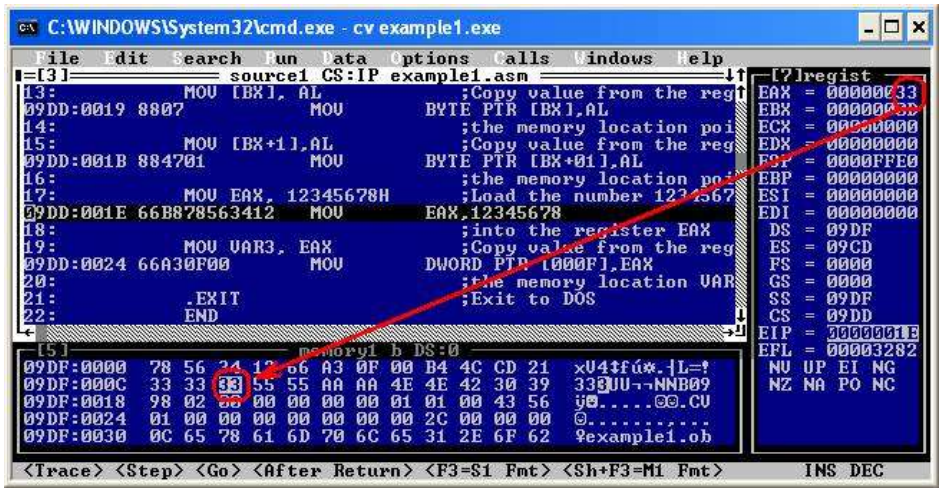
The following instruction “MOV [BX], AL” will copy the content of AL into the memory location pointed by BX within the data segment. After the

previous instruction BX contains the offset of the first byte of VAR2 or 000DH. That is where the data from AL will appear. Press “F10” to execute.

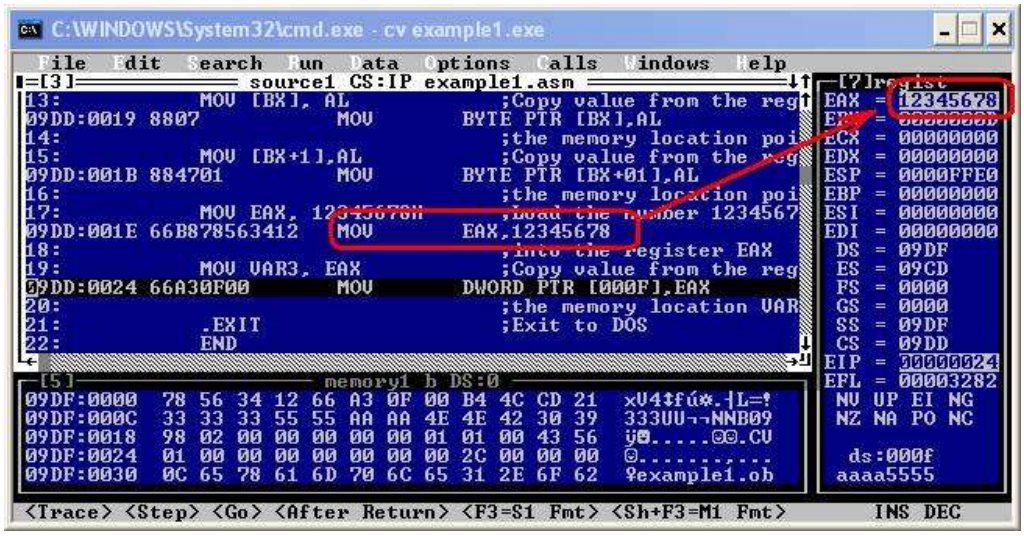
Note the debugger also highlighted changes in the data window.



Instruction “MOV [BX+1], AL” will copy the content of the register AL into the memory location with offset equal whatever the number is in BX plus 1. In our case BX=000DH, then the offset is 000DH+0001H=000EH. That is the second byte of the VAR2. Press “F10” to execute. Note the change in the memory content.



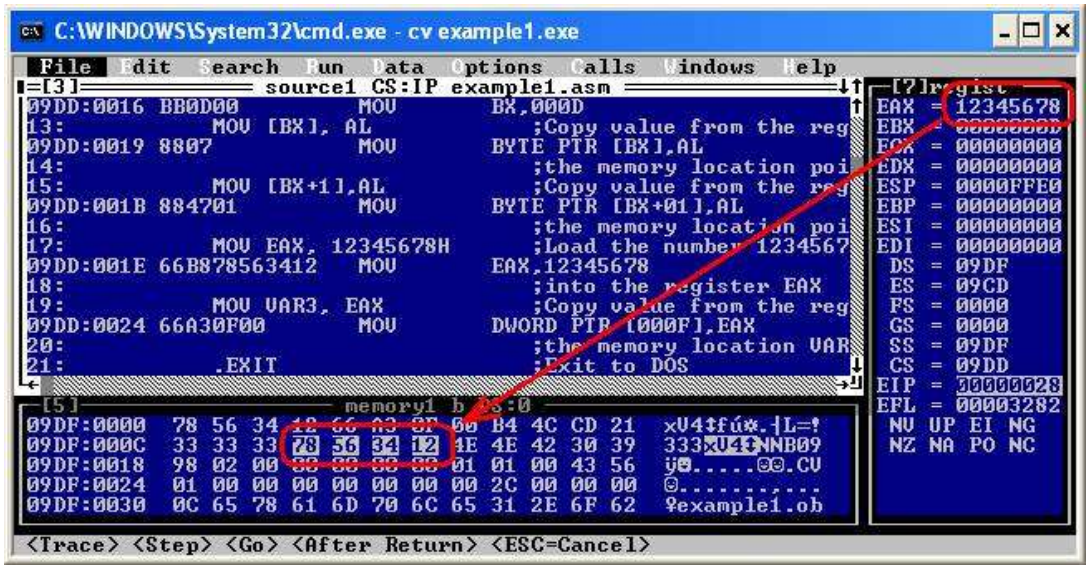
Instruction “MOV EAX, 12345678H” will place number 12345678H into the register EAX. Press “F10” to execute.



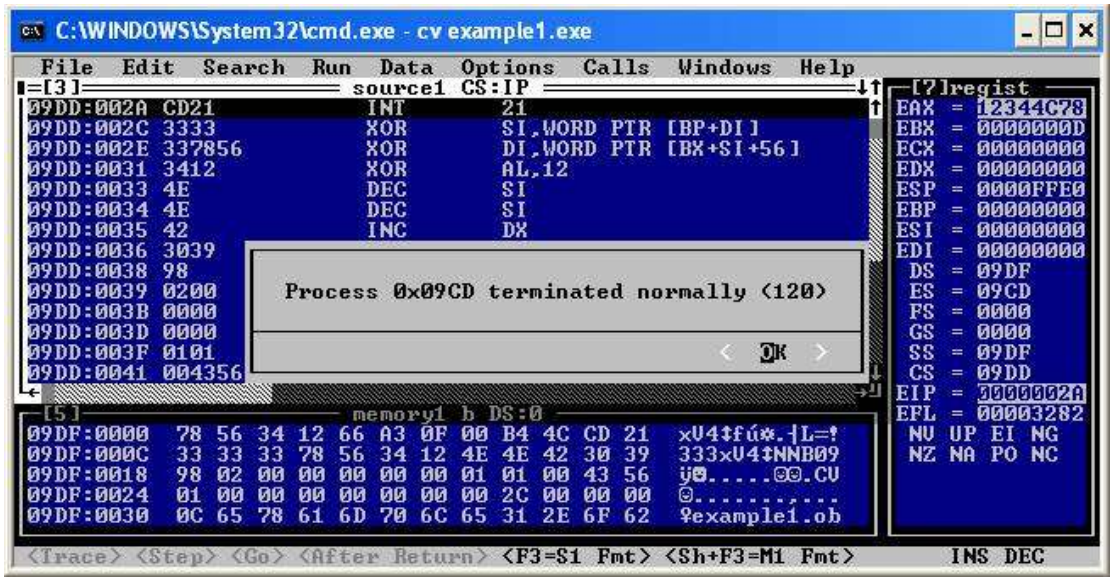
The instruction “MOV VAR3, EAX” became
“MOV DWORD PTR [000F], EAX”.

VAR3 has been replaced by the actual offset (000FH) of VAR3 in the data memory. This instruction will take the content of the EAX and place into the

four consecutive bytes of memory (a 32-bit variable) starting with the offset 000FH. Press “F10” to execute.



That was the last instruction of the user program. The remaining instructions are generated by the .EXIT directive and serve to terminate the program. Press “F10” until the process terminates.



Still not clear how to work with the CodeView debugger?

Here is additional tutorials you can go through.

CodeView tutorial

http://www.nuvisionmiami.com/books/asm/cv/index.htm

Debugging

http://www.math.uaa.alaska.edu/~afkjm/cs221/handouts/debugging.pdf

Emu8086 Assembler and Microprocessor Emulator - Overview

Everything for learning assembly language in one pack! Emu8086 combines an advanced source editor, assembler, disassembler, software emulator (Virtual PC) with debugger, and step by step tutorials. This program is extremely helpful for those who just begin to study assembly language. It compiles the source code and executes it on emulator step by step. Visual interface is very easy to work with. You can watch registers, flags and memory while your program executes. Arithmetic & Logical Unit (ALU) shows the internal work of the central processor unit (CPU).

Emulator runs programs on a Virtual PC, this completely blocks your program from accessing real hardware, such as hard-drives and memory, since your assembly code runs on a virtual machine, this makes debugging much easier.

8086 machine code is fully compatible with all next generations of Intel's micro-processors, including Pentium II and Pentium 4, I'm sure Pentium 5 will support 8086 as well. This makes 8086 code very portable, since it runs both on ancient and on the modern computer systems. Another advantage of 8086 instruction set is that it is much smaller, and thus easier to learn.

Before we start... (Download)

Make sure you have the latest *Emu8086* pack (assembler and emulator), you can download it from here

http://www.emu8086.com.

https://qpdownload.com/8086-microprocessor-emulator/

https://emu8086-microprocessor-emulator.en.softonic.com/

You can also use any other assembler, but this could be more difficult since you may need to work from the command prompt and *Emu8086* is a Windows based application so you can enjoy user-friendly Windows environment.

Emu8086 has a much easier syntax than any other assemblers, but generates the same machine code, it's very handy for beginners, however professionals may find Emu8086 useful as well. Also in case you are not using original *Emu8086* to compile, you won't be able to step through the

actual source while running it. *Emu8086* has a much easier syntax than any of the major assemblers, but will still generate a program that can be executed on any computer that runs 8086 machine code; a great combination for beginners!

Note: If you don't use *Emu8086* to compile the code, you won't be able to step through your actual source code while running it.

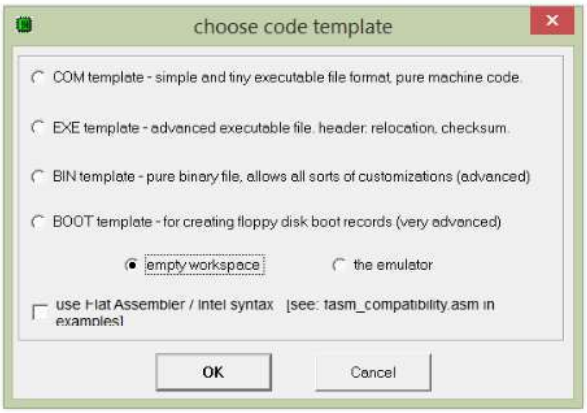
Where to start?

1. Start *Emu8086* by selecting its icon from the start menu, or by running **Emu8086.exe**.
2. Select "**Samples**" from "**File**" menu.
3. Click **[Compile and Emulate]** button (or press **F5** hot key).
4. Click [**Single Step**] button (or press **F8** hot key), and watch how the code is being executed.
5. Try opening other samples, all samples are heavily commented, so it's a great learning tool.

The architecture of 8086 Intel Microprocessor is named Von Neumann architecture after the mathematician who conceived of the design. The CPU can interpret the contents of memory either as instructions or as data. Because there is no difference between data and instructions for the processor, program can even re-write its own instructions and then execute them.

How to write and run a code on emu86:

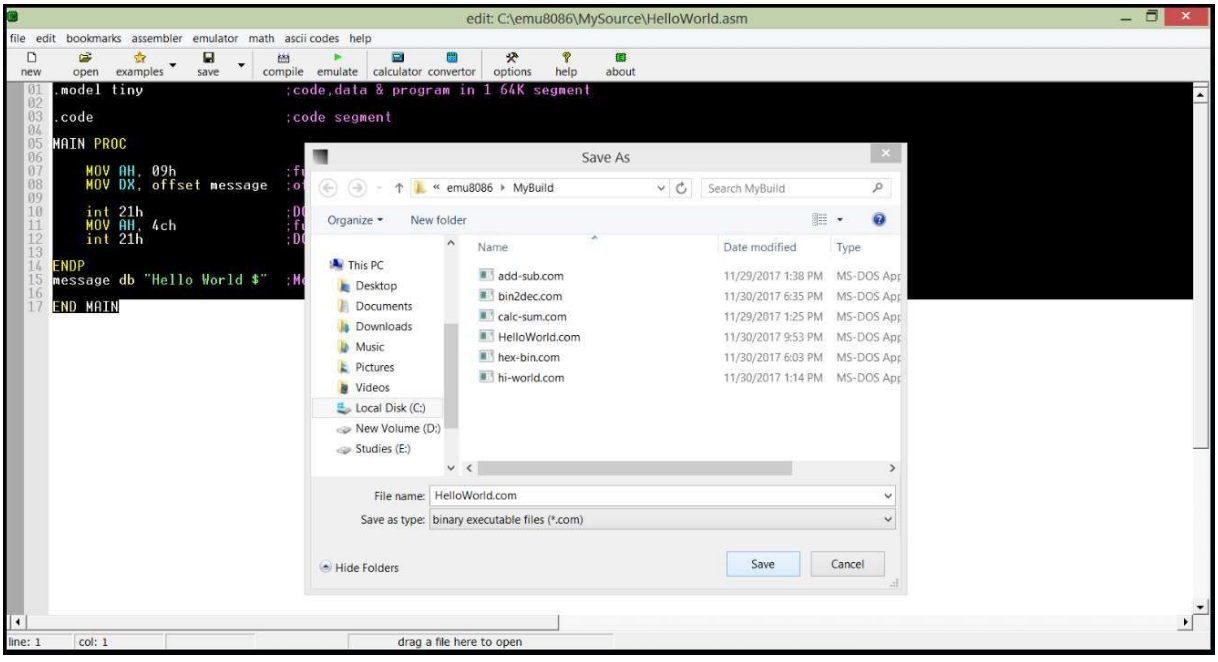
1. Create new file and save it as Hello World.asm



2. Copy the following code into that file.

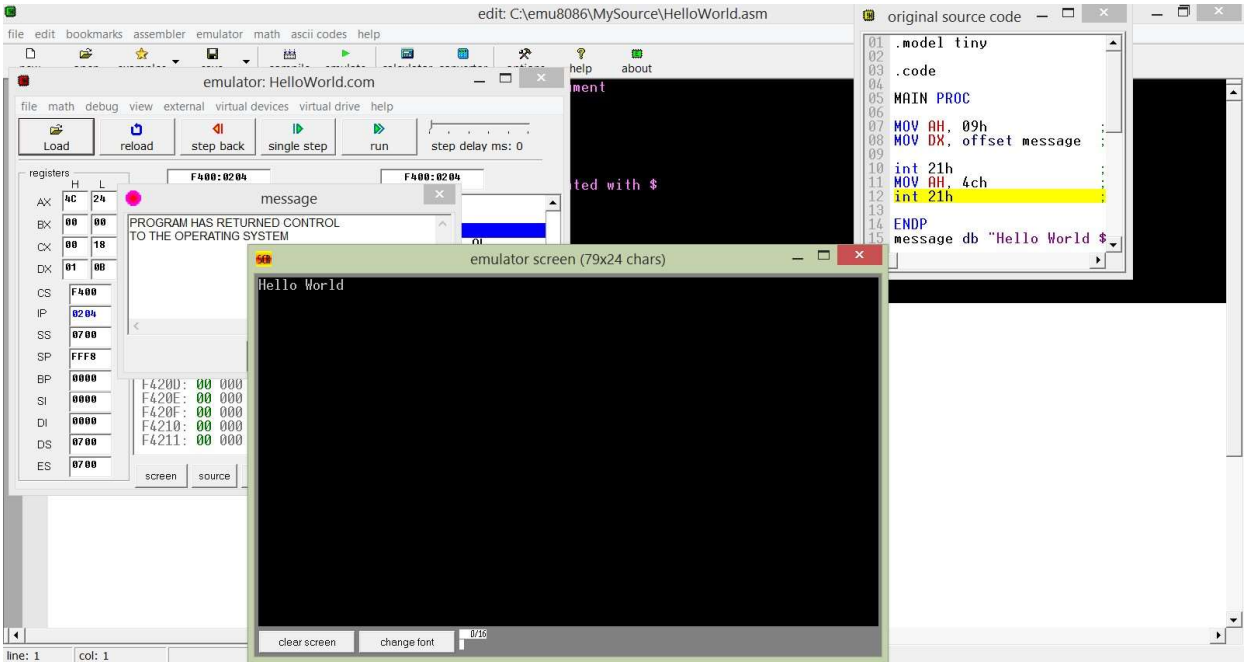
```
.model tiny                    ;code, data & program in 1 64K segment
.code                          ;code segment
MAIN PROC

   MOV AH, 09h                 ;function to display a string
   MOV DX, offset message      ;offset of Message string terminated with $
    int 21h                    ;DOS interrupt
   MOV AH, 4ch                 ;funtion to terminate
   int 21h                     ;DOS interrupt
 ENDP
message db "Hello World $"     ;Message to be displayed terminated with $(indicates end of string)
END MAIN
```

3. Now compile the program and save the HelloWorld.com file.



4. Run it.

5. You can see the output on the screen.



Module 7-

MIPS(MARS)

Simulator

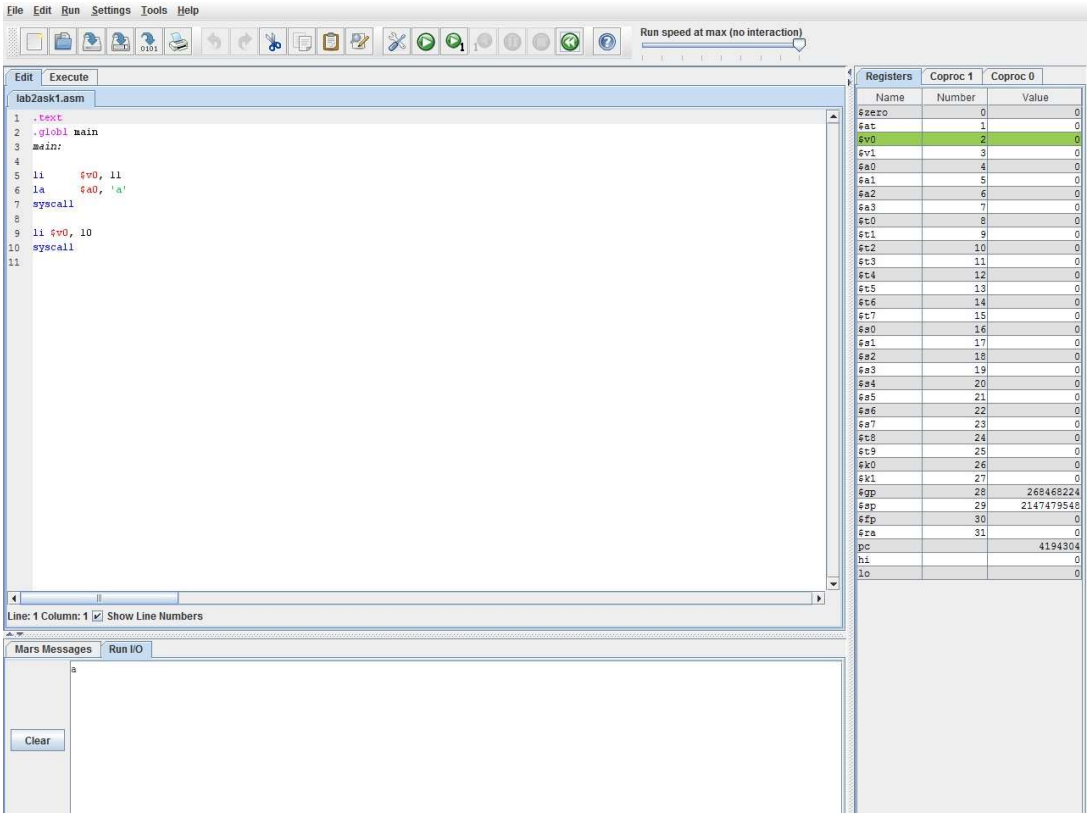
Introduction and Basic programming

Introduction:

MARS MIPS simulator is an assembly language editor, assembler, simulator & debugger for the MIPS processor, developed by Pete Sanderson and Kenneth Vollmar at Missouri State University (src).

You get the MARS for free here. As for installing the 4.5 version, you might need the suitable Java SDK for your system from here

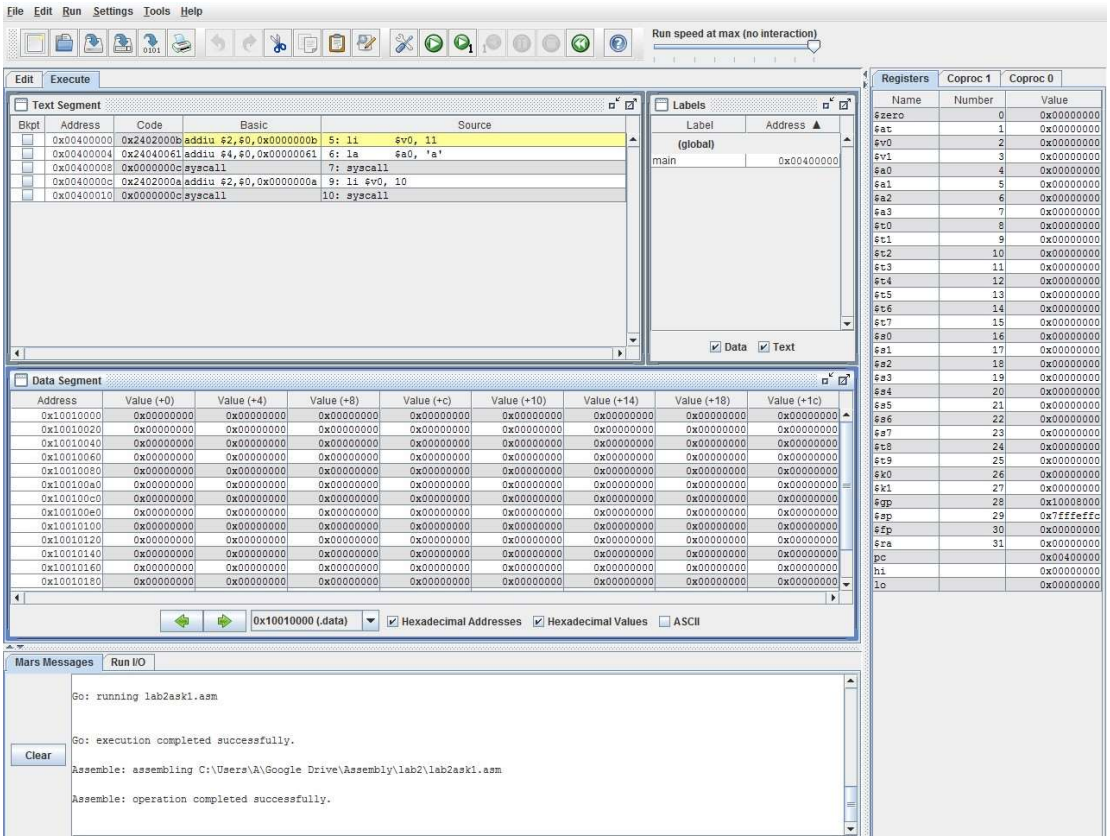
Before assembling, the environment of this simulator can be simplisticly split to three segments: the *editor* at the upper left where all of the code is being written, the compiler/output right beneath the editor and the *list of registers* that represent the "CPU" for our program.



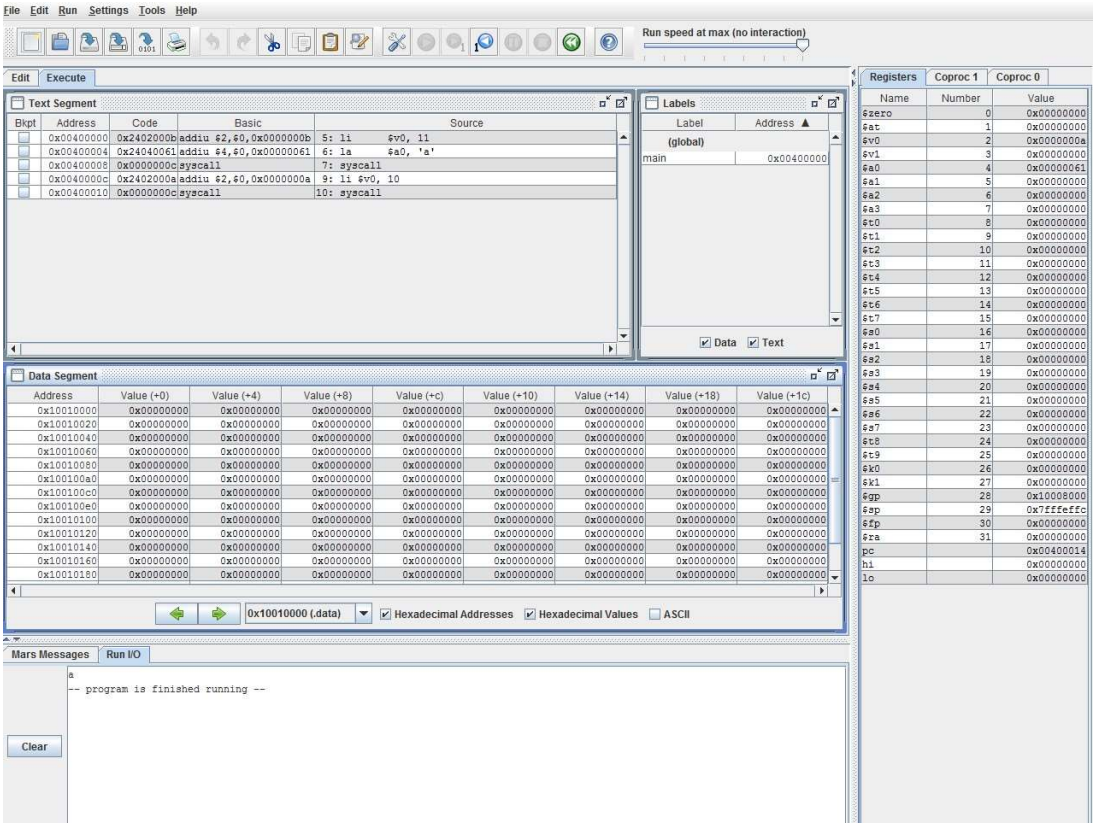
After assembling (by simply pressing F3) the environment changes, with two new segments getting the position of the editor: the *text segment* where

i) each line of assembly code gets cleared of "pseudoinstructions" (we'll talk about those in a sec) at the "basic" column and

ii) the machine code for each instruction at the "code" column, and the *data segment* where we can have a look at a representation of the memory of a processor with little-endian order.



After assembling, we can execute our code either all at once (F5) or step by step (F7), as well as rewinding the execution several steps backwards to the back (F8).



Now, let's see the example code from above and explain each line:

```
.text
.globl main
main:           #main function

li    $v0, 11    #11=system code for printing a character, $v0=register that gets the system code for printing as value
la    $a0, 'a'   #'a'=our example character, $a0=register that accepts the character for printing
syscall          #Call to the System to execute our instructions and print the character at the a0 register

li $v0, 10       #11=system code for terminating, $v0=register that gets the system code for terminating (optional, but
desirable)
syscall          #Call to the System to terminate the execution
```

MARS accepts and exports files with the .asm filetype

But the code above prints just a character, what about the good ol' "Hello World"? What about, dunno, adding a number or something? Well, we can change what we had a bit for just that:

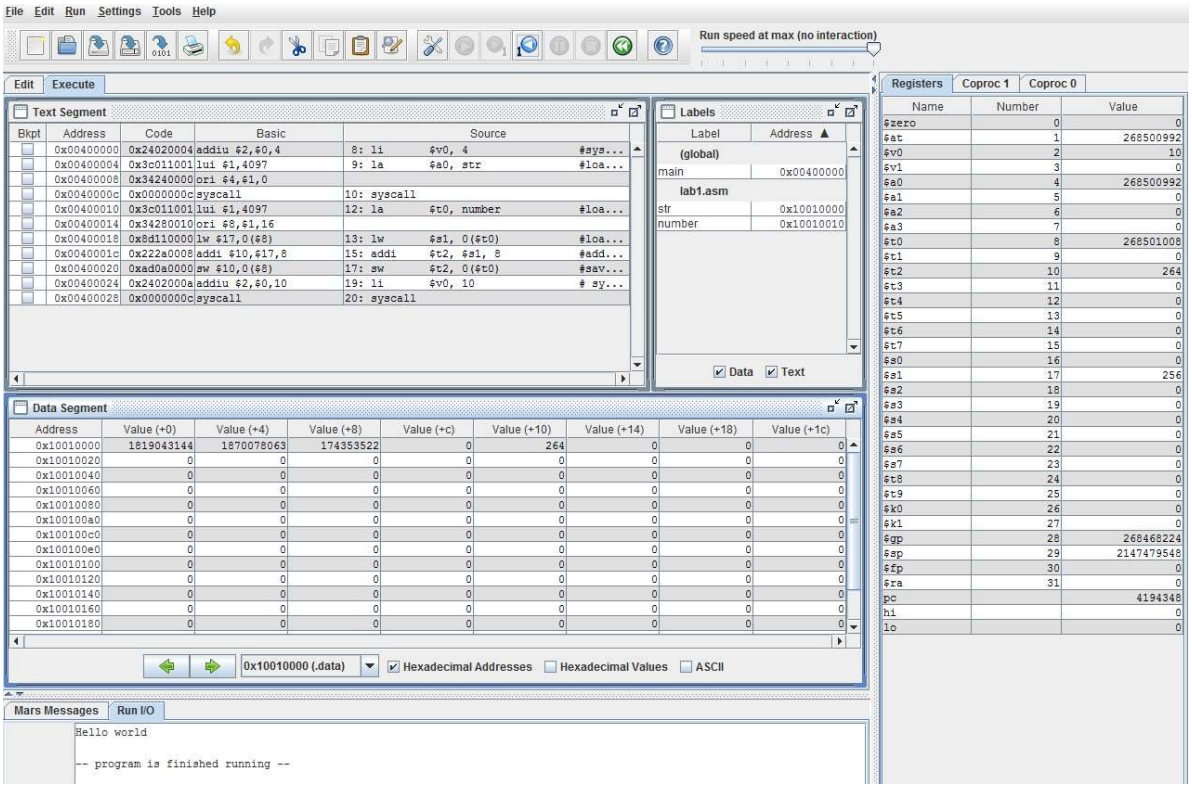
```
.data              #data section
str: .asciiz "Hello world\n"
number: .word 256

.text              #code section
.globl main
main:
li       $v0, 4              #system call for printing strings
la       $a0, str            #loading our string from data section to the $a0 register
syscall
```

```
la      $t0, number       #loading our number from data section to the $t0 register
lw      $s1, 0($t0)       #loading our number as a word to another register, $s1

addi    $t2, $s1, 8       #adding our number ($s1) with 8 and leaving the sum to register $t2

sw      $t2, 0($t0)       #storing the sum of register $t2 as a word at the first place of $t0

li      $v0, 10           # system call for terminating the execution
syscall
```

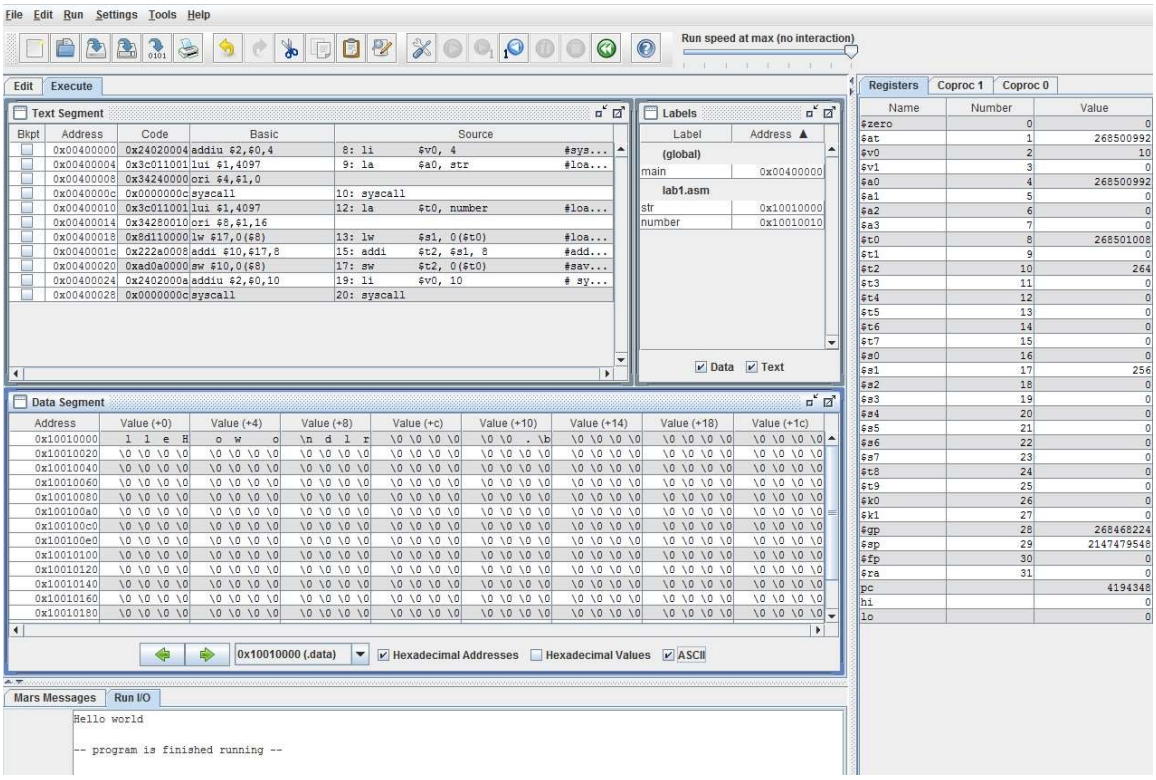
Before illustrating the results through MARS, a little more explanation about these commands is needed:

- ***System calls*** are a set of services provided from the operating system. To use a system call, a *call code* is needed to be put to $v0 register for the needed operation. If a system call has arguments, those are put at the $a0-$a2 registers. Here are all the system calls.
- li (load immediate) is a pseudo-instruction (we'll talk about that later) that instantly loads a register with a value. la (load address) is also a pseudo-instruction that loads an address to a register. With li $v0, 4 the $v0 register has now 4 as value, while la $a0, str loads the string of str to the $a0 register.
- A ***word*** is (as much as we are talking about MIPS) a 32 bits sequence, with bit 31 being the Most Significant Bit and bit 0 being the Least Significant Bit.
- lw (load word) transfers from the memory to a register, while sw (store word) transfers from a register to the memory. With the lw $s1, 0($t0) command, we loaded to $s1 register the value that was at the LSB of the $t0 register (thats what the 0 symbolizes here, the offset of the word), aka 256. $t0 here has the address, while $s1 has the value. sw $t2, 0($t0) does just the opposite job.
- MARS uses the Little Endian, meaning that the LSB of a word is stored to the smallest byte address of the memory.
- MIPS uses **byte addresses**, so an address is apart of its previous and next by 4.

By assembling the code from before, we can further understand how memory and registers exchange, disabling "Hexadecimal Values" from the Data Segment:



or enabling "ASCII" from the Data Segment:



Start it like this

$ java -jar Mars4_5.jar

Create this file and save it.

```
   .text
main:
   li    $s0,0x30
loop:
   move    $a0,$s0        # copy from s0 to a0

   li    $v0,11         # syscall with v0 = 11 will print out
   syscall            # one byte from a0 to the Run I/O window

   addi    $s0,$s0,3    # what happens if the constant is changed?

   li    $t0,0x5d
   bne    $s0,$t0,loop
   nop            # delay slot filler (just in case)

stop:    j    stop        # loop forever here
   nop            # delay slot filler (just in case)
```

Module 8- Create of Application and its Software using 8085/8086 Microprocessor or Microcontrollers

Sample Project Report

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, Noida

Department of CSE & IT

Project Report

Computer Organisation and Architecture

(15B17CI373)

Basic Computer Simulator

ODD SEM 2018

Submitted By:

Utsav Gupta – 17103040

Rohan Jain – 17103056

Rajat Kumar Garg – 17103062

Hardik Bhardwaj - 17103064

(Batch – B10)

Submitted To:

Dr. Hema N

(CSE Faculty)

Introduction

The Basic Computer Simulator is an 8 bit microprocessor that can handle a memory of 64KB.

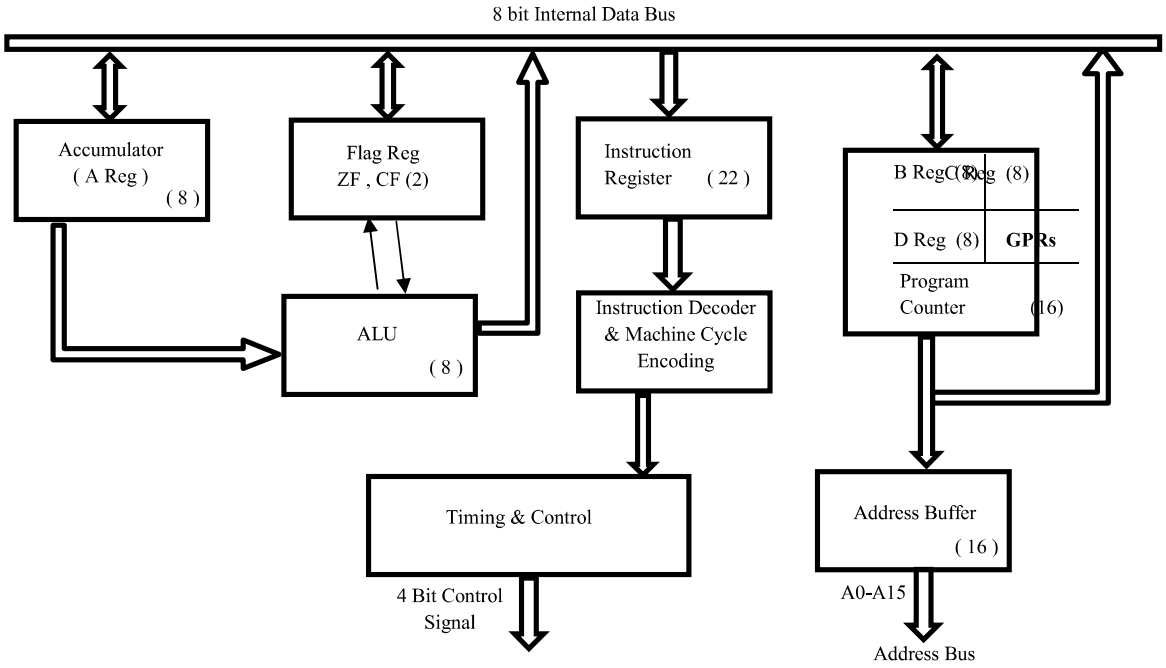
Microprocessor consists of:

Control unit: control microprocessor operations and generate Control Signals

ALU: performs data processing function.

Registers: provide storage internal to CPU.

Internal data bus: (Bidirectional)to transfer Data



Architecture

The Basic Computer Simulator has the following characteristics:

1. Data Bus – 8 bits
2. Address Bus – 16 bits
3. Word Size – 8 bits
4. Total Registers – 4
   a. General Purpose Registers (**B** Register, **C** Register, **D** Register)
   b. Special Purpose Registers (**A**ccumulator)
5. Flags – 2 (**Z**ero **F**lag, **C**arry **F**lag)
6. Memory Size – 64 KB of memory : Total 2^{16} memory location each capable of storing a word i.e. 8 bits.

Instruction Type

Total Instructions can be classified into four types:

1. Data Transfer Instructions → Data Transfer Instructions are the instructions in which data transfer takes place between Registers and Memory Locations.
   a. **MOV –** Move values from Source Register (latter) to Destination Register.
      Eg. MOV A B;
   b. **MVI –** Move 8 bit immediate values in a register
      Eg.MVI A 05;
      MVI B 45;
   c. **LDA –** Load Accumulator with contents of a 16 bit Address
      Eg. LDA 200F;
   d. **STA –** Store the content of accumulator at a 16 bit Address
      Eg. STA AA45;
2. Arithmetic Instructions → Arithmetic Instructions are the instructions which performs mathematical operations such as Addition, Subtraction & Multiplication.
   a. **ADD –** Add the contents of source register and destination register and store the result in destination register.
      Eg. ADD A B;
   b. **ADI –** Add the immediate 8 bit value to the register and store the result back in register.
      Eg. ADI A 12;
   c. **SUB –** Subtract the contents of source register from destination register and store the result in destination register.
      Eg. SUB A B;
   d. **SBI –** Subtract the 8 bit immediate value from the register and store the result in destination register.
      Eg. SUB A 05;

e. **MUL –** Multiply the contents of source and destination register and store the result in destination register.

Eg. MUL A B;

f. **MLI –** Multiply 8 bit value with the register and store the result in Destination register.

Eg. MLI A 06;

3. Logical Instructions→Logical Instructions are the instructions which performs logical operations such as AND, OR.
   a. **AND –** Take the AND of the contents of Source and Destination register and then store the result in destination register.

      Eg. AND A B;

   b. **ANI –** Take the AND of 8 bit value with the contents of register and store the result in destination register.

      Eg. ANI A 05;

   c. **ORR –** Take the OR of the contents of Source and Destination register and then store the result in destination register.

      Eg. ORR A B;

   d. **ORI –**Take the OR of 8 bit value with the contents of register and store the result in destination register.

      Eg. ORI A 35;

4. Machine Control Instruction →The Control Instructions guide your machine to operate in a specific manner.
   a. **HLT –** Stop executing the program.

INSTRUCTION MANUAL

DATA TRANSFER INSTRUCTIOS	ARITHMETIC INSTRUCTIONS	LOGICAL INSTRUCTIONS
MOV A B;	ADD B C;	ORR A B;
MVI A 45;	ADI B C;	ORI A 45;
LDA 2000;	SUB B C;	AND A B;
STA 2000;	SBI B 45;	ANI A 45;
	MUL A B;	
	MLI A 45;	
	INC B;	
	DEC B;	

HLT - To Terminate the Program !

*Immediate Data is in Hexa Decimal Form.

Instruction Set Architecture

The simulator follows a variable length Instruction Set Architecture.

i) Zero Operand Addressing Mode : 6 bit opcode

Addressing Mode Opcode	Instruction Opcode
B5-B3	B2-B0

ii) One Operand Addressing Mode : 8 bit opcode

Addressing Mode Opcode	Instruction Opcode	Register
B7-B5	B4-B2	B1-B0

iii) Two Operand Addressing Mode : 10 bit opcode

Addressing Mode Opcode	Instruction Opcode	Destination Register	Source Register
B9-B7	B6-B4	B3-B2	B1-B0

iv) Direct 8 bit Data Addressing Mode : 16 bit opcode

Addressing Mode Opcode	Instruction Opcode	Destination Register	Immediate Data
B15-B13	B12-B10	B9-B8	B7-B0

v) Direct 16 bit Memory Addressing Mode : 22 bit opcode

Addressing Mode Opcode	Instruction Opcode	Direct Address
B21-B19	B18-B16	B15-B0

The Most Significant 3 bits determine the Addressing Mode

Addressing Mode	
Addressing Mode	*Opcode IT*
Zero Operand	000
One Operand	001
Two Operand	010
Direct 8 Bit Data	010
Direct 16 Bit Memory	100

Two Bits are dedicated to Register Identification opcode.

Registers' Opcode	
Register	*Opcode R*
A	00
B	01
C	10
D	10

The next three bits identify instruction opcode in each Addressing Mode.

Addressing Mode	*Instruction*	*Instruction Opcode*
Zero Operand	HLT	000

Addressing Mode	*Instruction*	*Instruction Opcode*
One Operand	INC	000
	DEC	001

Addressing Mode	*Instruction*	*Instruction Opcode*
Two Operand	MOV	000
	ADD	001
	SUB	010
	MUL	011
	ORR	100
	AND	101

Addressing Mode	*Instruction*	*Instruction Opcode*
Direct 8 Bit Data	MVI	000
	ADI	001
	SBI	010
	MLI	011
	ORI	100
	ANI	101

Addressing Mode	*Instruction*	*Instruction Opcode*
Direct 16 Bit Memory	LDA	000
	STA	001

Control ROM

Total 16 control ROM signals are generated which are as follows:

0000 – Data Transfer register to register

0001 – ALU Operation Add

0010 – ALU Operation Subtract

0011 – ALU Operation Multiply

0100 – ALU Operation Bitwise Or

0101 – ALU Operation Bitwise And

0110 – Data Transfer Memory to Register

0111 – ALU Operation Add with Immediate Data

1000 – ALU Operation Subtract with Immediate Data

1001 – ALU Operation Multiply with Immediate Data

1010 – ALU Operation Bitwise Or with Immediate Data

1011 – ALU Operation Bitwise And with Immediate Data

1100 – ALU Operation Increment by 1

1101 – ALU Operation Decrement by 1

1110 – Load Data from Memory

1111 – Store Data into Memory

Based on the variable length instruction set architecture, one can easily calculate the instruction opcode.

Ex: **MOV C D :**

B9-B7	B6-B4	B3-B2	B1-B0
010	**000**	**10**	**11**

LDA FFA2 ;

B21-B19	B18-B16	B15-B0
100	**000**	**1111111110100010**

ANI B 62;

B15-B13	B12-B10	B9-B8	B7-B0
011	**101**	**01**	**01100010**

DEC A;

B7-B5	B4-B2	B1-B0
001	**001**	**00**

Sample Code and Output:

1. **To Store the first 4 multiples of a number kept in B register at memory location starting from 2000H.**

```
MOV A B;
STA 2000;
ADD A B;
STA 2001;
ADD A B;
STA 2002;
ADD A B;
STA 2003;
HLT;
```

```
*********************************************************************
*                                                                   *
*                        Registers Content                          *
*                                                                   *
*       A - 14H         B - 05H         C - 00H         D - 00H     *
*                                                                   *
*                                                                   *
*                          Flags Content                            *
*                                                                   *
*                 CF - 00               ZF - 00                     *
*                                                                   *
*                                                                   *
*                        Memory Content                             *
*                                                                   *
*                 Program Counter - 000AH                           *
*                       [2000H] - 05H                               *
*                       [2001H] - 0AH                               *
*                       [2002H] - 0FH                               *
*                       [2003H] - 14H                               *
*                                                                   *
*********************************************************************
Press any key to continue . . .
```

When Run All at a Time

```
[0005] >> ADD A B;

        Instruction Type : 010
        Instruction Opcode : 001
        Destination Register Code : 00
        Source Register Code : 01
        Control ROM Content : 01, Rs = 1, Rd = 0, imm data = 0, Mem Index = 0

        *********************************************************************
        *                                                                   *
        *                        Registers Content                          *
        *                                                                   *
        *       A - 0AH         B - 05H         C - 00H         D - 00H     *
        *                                                                   *
        *                                                                   *
        *                          Flags Content                            *
        *                                                                   *
        *                 CF - 00               ZF - 00                     *
        *                                                                   *
        *                                                                   *
        *                        Memory Content                             *
        *                                                                   *
        *                 Program Counter - 0006H                           *
        *                       [2000H] - 05H                               *
        *                                                                   *
        *********************************************************************
```

When Run Step By Step

Learning Outcomes

The making of this project led to:

1. Better Understanding of Instruction Set Architecture
2. Discussion on Variable Length Instruction Set
3. Understanding of Control Unit
4. Better Understanding of Addressing Modes and various instruction type.