AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

LABORATORY MANUAL

COMPUTER ORGANIZATION AND ARCHITECTURE

SUBJECT CODE: 3140707

COMPUTER SCIENCE ENGINEERING

DEPARTMENT

B.E. 4TH SEMESTER

Name:____

Enrolment number:____________________________________

Batch:____

Year:____

Amiraj College of Engineering and Technology

Nr.Tata Nano Plant khoraj Sanand

AMIRAJ

COLLEGE OF ENGINEERING & TECHNOLOGY

CERTIFICATE

This is to certify that Mr. / Ms. ____

Of class__________________ Enrolment No ________________________has

Satisfactorily completed the course in ________________________________as

By the Gujarat Technological University for ____ Year (B.E.) semester___ of

Computer Science and Engineering in the Academic year ______.

Date of Submission:-

Faculty Name and Signature   Head of Department

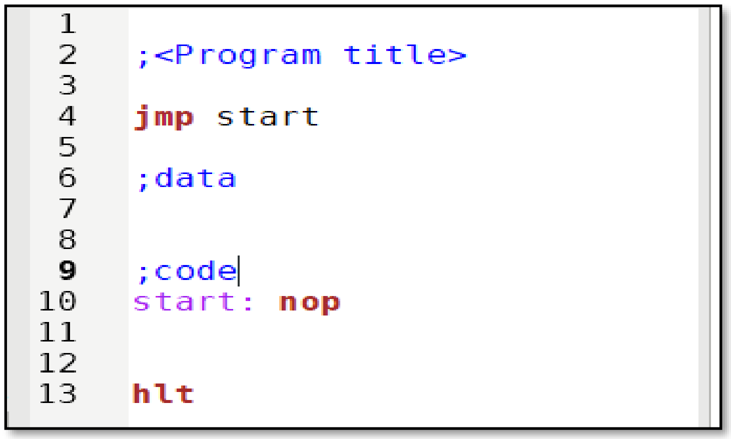
S Y JOSHI   (CSE)

Practical-1

Aim: To study simulator GNUsim8085

Theory:

GNUSim8085 is a graphical simulator, assembler and debugger for the Intel 8085 microprocessor in Linux and Windows. It contains a simple editor component with syntax highlighting. Also it has a keypad to input assembly language instructions with appropriate arguments.



```

;<Program title>

jmp start

;data


;code
start: nop


hlt
```

The image above shows default appearance of the GNUSim8085.

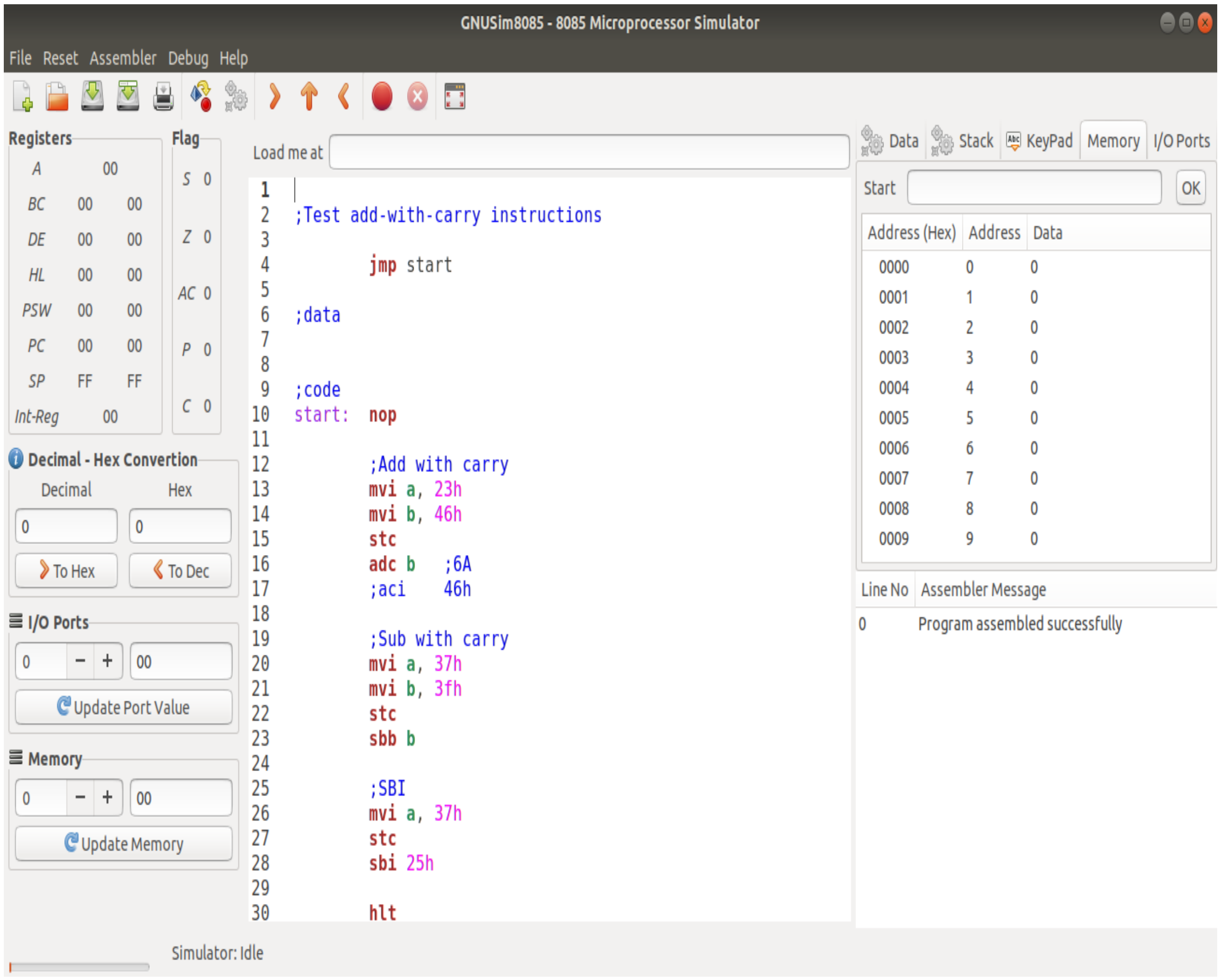
Below are the characteristics highlights of the simulator GNUSim8085

- A keypad to input assembly language instructions with appropriate arguments.
- Easy view of register contents.

  The register contents by default are set to zero
- Easy view of flag contents.

  The flag contents by default are set to zero
- Hexadecimal <–> Decimal converter.
- View of stack, memory and I/O contents.
- Support for breakpoints for program debugging.
- Stepwise program execution.
- One click conversion of assembly program to op code listing.

- Printing support.
- UI translated in various languages



Registers		
A	00	
BC	00	00
DE	00	00
HL	00	00
PSW	00	00
PC	00	00
SP	FF	FF
Int-Reg	00	

Flag	
S	0
Z	0
AC	0
P	0
C	0

```

;Test add-with-carry instructions

        jmp start

;data


;code
start:  nop

        ;Add with carry
        mvi a, 23h
        mvi b, 46h
        stc
        adc b   ;6A
        ;aci    46h

        ;Sub with carry
        mvi a, 37h
        mvi b, 3fh
        stc
        sbb b

        ;SBI
        mvi a, 37h
        stc
        sbi 25h

        hlt
```

Address (Hex)	Address	Data
0000	0	0
0001	1	0
0002	2	0
0003	3	0
0004	4	0
0005	5	0
0006	6	0
0007	7	0
0008	8	0
0009	9	0

Line No	Assembler Message
0	Program assembled successfully

The image above shows the GNUSim8085 sample addition with carry program; Registers and flags on the left size along with their status values; left below shows decimal to hex conversion with I/O ports and Memory ports with editing accessibility; on the right is loaded address contents.

Read Practical2 onwards for detailed understanding of 8085 architecture flag register etc details.

*Note: All images and content copyrights to gnusim8085.srid.ca

<u>**Practical-2**</u>

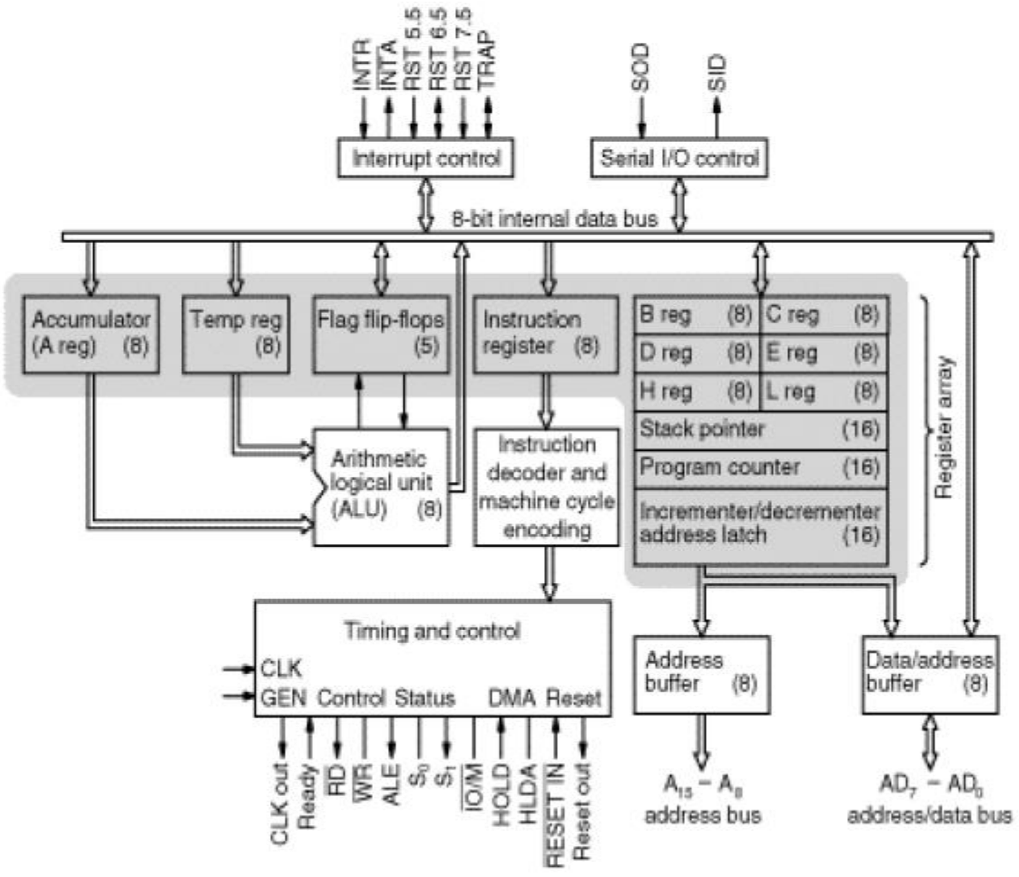
Aim: To study basic architecture of 8085

The 8085 microprocessor was made by Intel in mid 1970s. It was binary compatible with 8080 microprocessor but required less supporting hardware thus leading to less expensive microprocessor systems.

It is a general purpose microprocessor capable of addressing 64k of memory. The device has 40 pins, require a +5V power supply and can operate with 3 MHz single phase clock.

It has also a separate address space for up to 256 I/O ports. The instruction set is backward compatible with its predecessor 8080 even though they are not pin-compatible.

8085 Internal Architecture is shown in (Fig: 1)



The 8085 has a 16 bit address bus which enables it to address 64 KB of memory, a data bus 8 bit wide and control buses that carry essential signals for various operations. It also has a built in register array which are usually labelled A(Accumulator), B, C, D, E, H, and L. Further special-purpose registers are the 16-bit Program Counter (PC), Stack Pointer (SP), and 8-bit flag register F. The microprocessor has three maskable interrupts (RST 7.5, RST 6.5 and RST 5.5), one Non-Maskable interrupt (TRAP), and one externally serviced interrupt (INTR). The RST n.5 interrupts refer to actual pins on the processor a feature which permitted simple systems to avoid the cost of a separate interrupt controller chip.

Control Unit

Generates signals within microprocessor to carry out the instruction, which has been decoded. In reality causes certain connections between blocks of the processor be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Arithmetic Logic Unit

The ALU performs the actual numerical and logic operation such as 'add', 'subtract', 'AND', 'OR', etc. Uses data from memory and from Accumulator to perform arithmetic and always stores the result of operation in the Accumulator.

Registers

The 8085 microprocessor includes six registers, one accumulator, and one flag register, as shown in Fig 1. In addition, it has two 16-bit registers: the stack pointer and the program counter. The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H, and L as shown in Fig 1. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.

Accumulator

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flag Registers

The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero(Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

Stack Pointer (SP)

The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

Instruction Register / Decoder

This is a temporary store for the current instruction of a program. Latest instruction is sent to here from memory prior to execution. Decoder then takes instruction and ‘decodes’ or interprets the instruction. Decoded instruction is then passed to next stage.

Memory Address Register (MAR)

Holds addresses received from PC for eg: of next program instruction. MAR feeds the address bus with address of the location of the program under execution.

Control Generator

Generates signals within microprocessor to carry out the instruction which has been decoded. In reality it causes certain connections between blocks of the processor to be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Register Selector

This block controls the use of the register stack. Just a logic circuit which switches between different registers in the set will receive instructions from Control Unit.

8085 System Bus

The microprocessor performs four operations primarily.

- Memory Read
- Memory Write

- I/O Read
- I/O Write

All these operations are part of the communication processes between microprocessor and peripheral devices. The 8085 performs these operations using three sets of communication lines called buses - the address bus, the data bus and the control bus.

Address Bus

The address bus is a group of 16 lines. The address bus is unidirectional: bits flow only in one direction – from the 8085 to the peripheral devices. The microprocessor uses the address bus to perform the first function: identifying a peripheral or memory location. Each peripheral or memory location is identified by a 16 bit address. The 8085 with its 16 lines is capable of addressing 64 K memory locations.

Data Bus

The data bus is a group of eight lines used for dataflow. They are bidirectional: data flows in both direction between the 8085 and memory and peripheral devices. The 8 lines enable the microprocessor to manipulate 8-bit data ranging from 00 to FF.

Control Bus

The control bus consists of various single lines that carry synchronization signals. These are not groups of lines like address of data bus but individual lines that provide a pulse to indicate an operation. The 8085 generates specific control signal for each operation it performs. These signals are used to identify a device type which the processor intends to communicate

Practical-3

Aim: To write an assembly language code in GNUsim8085 to implement arithmetic instruction –ADD

Algorithm:

- Pass two binary values (8bit binary numbers) to two corresponding registers using suitable data transfer instructions
- Use arithmetic instruction add instruction to add the binary numbers
- Check for carry if any
- Store the result
- Terminate the program.

Code:

Input:

Output:

Practical-4

Aim: To write an assembly language code to implement arithmetic instruction –SUB

Algorithm:

- Pass two binary values (8bit binary numbers) to two corresponding registers using suitable data transfer instructions
- Use arithmetic instruction add instruction to perform subtraction the binary numbers
- Check for borrow if any
- Store the result
- Terminate the program.

Code:

Input:

Output:

Practical-5

Aim: To write an assembly language code to implement logical instruction –OR

Algorithm:

- Pass two binary values (8bit binary numbers) to two corresponding registers using suitable data transfer instructions
- Use logical instruction OR instruction to perform logical or of the binary numbers
- Check for S,Z,P flags
- Store the result
- Terminate the program.

Code:

Input:

Output:

Practical-6

Aim: To write an assembly language code to implement logical instruction –AND

Algorithm:

- Pass two binary values (8bit binary numbers) to two corresponding registers using suitable data transfer instructions
- Use logical instruction AND instruction to perform logical anding of the binary numbers
- Check for S,Z,P flags
- Store the result
- Terminate the program.

Code:

Input:

Output:

<u>Practical-7</u>

Aim: To write an assembly language code to find the factorial of a number

Algorithm:

Factorial (!) of a number n is:

n!= n X (n-1) X (n-2) …… X (n-(n-1))

- Load the data(whose factorial is to be found out) into register1
- To start multiplication set register2 to 01H
- Decrement register1 contents to multiply to a previous number
- Continue till value of register1 is greater than zero
- Take memory pointer to next location and store result
- Terminate the program
- Hint: Use a multiplication subroutine for easiness

Code:

Input:

Output:

Practical-8

Aim: To write an assembly language code to store numbers in reverse order in memory location

Algorithm:

- Initialize the memory pointer
- Get the count in any (B) register
- Increment the pointer
- Get the first data in A register
- Decrement the count
- Increment the pointer
- Pass the values to appropriate registers
- Check for least value of count
- Pass the values to corresponding memory location using opposite procedure
- Store the result
- Terminate the program.

Code:

Input:

Output:

Practical-9

Aim:To study Verilog HDL

Theory:

Verilog is a **HARDWARE DESCRIPTION LANGUAGE (HDL).**

It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip−flop.

It means, by using a HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

Verilog supports a design at many levels of abstraction.

The major three are −

- Behavioral level
- Register-transfer level
- Gate level

Behavioral level
This level describes a system by concurrent algorithms (Behavioural). Every algorithm is sequential, which means it consists of a set of instructions that are executed one by one. Functions, tasks and blocks are the main elements. There is no regard to the structural realization of the design.

Register−Transfer Level
Designs using the Register−Transfer Level specify the characteristics of a circuit using operations and the transfer of data between the registers. Modern definition of an RTL code is "Any code that is synthesizable is called RTL code".

Gate Level
Within the logical level, the characteristics of a system are described by logical links and their timing properties. All signals are discrete signals. They can only have definite logical values (`0', `1', `X', `Z`). The usable operations are predefined logic primitives (basic gates). Gate level modelling may not be a right idea for logic design. Gate level code is generated using tools like synthesis tools and his netlist is used for gate level simulation and for backend.

Lexical Tokens

Verilog language source text files are a stream of lexical tokens. A token consists of one or more characters, and each single character is in exactly one token.

The basic lexical tokens used by the Verilog HDL are similar to those in C Programming Language. Verilog is case sensitive. All the key words are in lower case.

White Space

White spaces can contain characters for spaces, tabs, new-lines and form feeds. These characters are ignored except when they serve to separate tokens.

White space characters are Blank space, Tabs, Carriage returns, New line, and Form feeds.

Comments

There are two forms to represent the comments

- 1) Single line comments begin with the token // and end with carriage return.

Ex.: //this is single line syntax

- 2) Multiline comments begins with the token /* and end with token */

Ex.: /* this is multiline Syntax*/

Numbers

You can specify a number in binary, octal, decimal or hexadecimal format. Negative numbers are represented in 2's compliment numbers. Verilog allows integers, real numbers and signed & unsigned numbers.

The syntax is given by − <size> <radix> <value>

Size or unsized number can be defined in <Size> and <radix> defines whether it is binary, octal, hexadecimal or decimal.

Identifiers

Identifier is the name used to define the object, such as a function, module or register. Identifiers should begin with an alphabetical characters or underscore characters. Ex. A_Z, a_z,_

Identifiers are a combination of alphabetic, numeric, underscore and $ characters. They can be up to 1024 characters long.

Operators

Operators are special characters used to put conditions or to operate the variables. There are one, two and sometimes three characters used to perform operations on variables.

Ex. >, +, ~, &! =.

Verilog Keywords

Words that have special meaning in Verilog are called the Verilog keywords. For example, assign, case, while, wire, reg, and, or, nand, and module. They should not be used as identifiers. Verilog keywords also include compiler directives, and system tasks and functions.

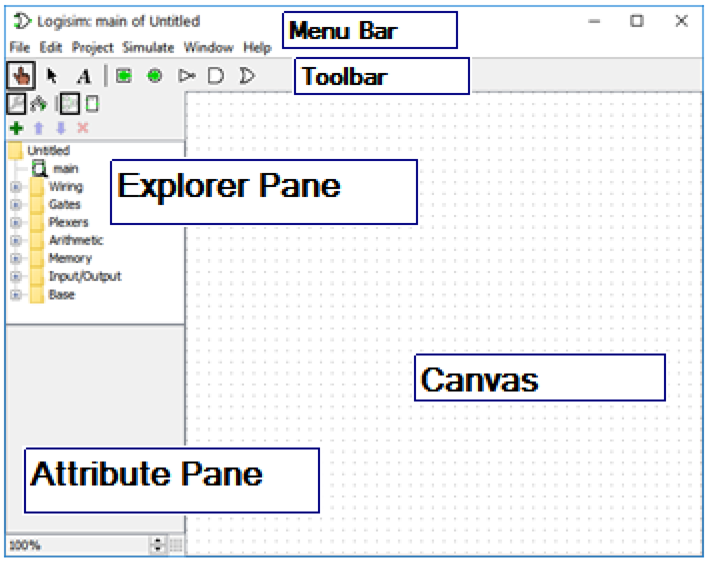
Practical-10

Aim:To study Logisim

Theory:
LogiSim is a LOGic SIMulator.

When learning computer architecture and logic circuits, one needs a real-world, graphical example of what you are studying. Text and diagrams only go so far. A helpful tool for designing and simulating logic circuits is **Logisim**.

In logisim, circuits will be arranged in the canvas as shown in the typical logisim layout as shown in below figure.



The explorer pane contains all of the elements needed to create circuits. For example, gates, plexers etc. In order to add an element from the explorer pane, you select the item in the pane. Next, you click on the canvas to drop the element onto your design.

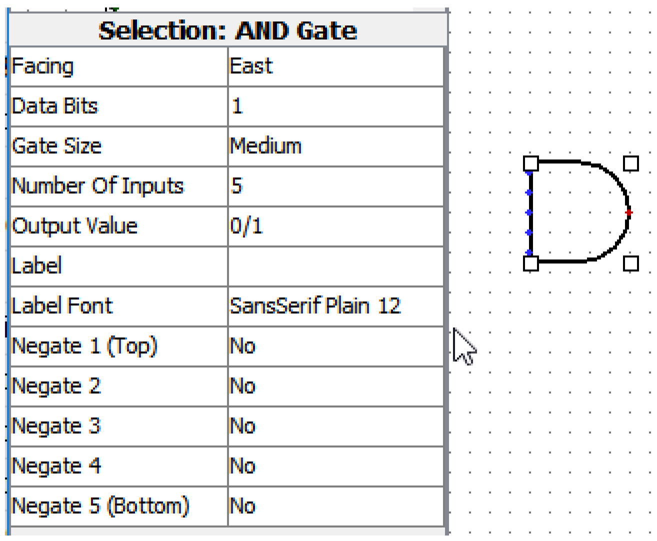
Now to add an element from the explorer pane, you need to select any item in the pane. Next, you click on the canvas to drop the element onto your design.

Suppose, we choose and drop an AND gate onto the canvas.

Explorer pane>>gates>>AND gate

The attributes pane will get activated i.e. will have some options.

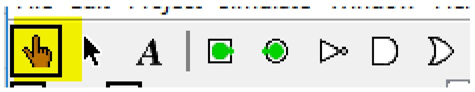
The figure here shows the AND gate into canvas and also attribute along with options.



Selection: AND Gate	
Facing	East
Data Bits	1
Gate Size	Medium
Number Of Inputs	5
Output Value	0/1
Label	
Label Font	SansSerif Plain 12
Negate 1 (Top)	No
Negate 2	No
Negate 3	No
Negate 4	No
Negate 5 (Bottom)	No

We can change the direction the gate faces (east, west, north, south), or the number of data bits (the default is 1). As you build circuits, you will be changing the attributes of your elements here.

Changing Values in Circuits
In the toolbar, you'll notice a little hand icon (shown in Figure). This icon is used to change values in circuits. This is how you test your circuits.

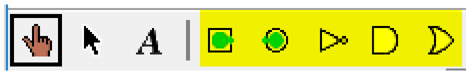


Logisim Testing Tool

To continue moving/adding elements, click the arrow icon in the toolbar. You can click the A icon in the toolbar, then click on the canvas to add the text. You change the actual text and font in the attributes pane.

Common Tools

You'll also notice a set of common tools up in the tool bar as shown in Figure


Logisim common tools

From left to right, there are two pins, a NOT gate, an AND gate, and an OR gate.
These are the Logisim common tools

To create simple combinational circuit firstly we enlist and add the tools from the toolbars.