SIMULATION OF COMPUTER COMPONENTS

Aim:

To study the simulation of computer components.

Procedure:

1. Motherboard:

The motherboard is a paramount computer component of the central part where everything else is connecting to a motherboard is affable sized circuit board that allows other elements to interact it has sorts which are fairing outside a pc are so that you can plug to a monitor, charge computer or plug in a mouse, it also has slots for expansion so that you can install additional accessory ports if you wish to do so, the mother board store low level data like the system time even when a pc is switched off.

2. Power Supply:

The power supply is the device that powers all other mechanism of the pc, it generally plugs into the motherboard, it can connect to either a plug for an outlet (desktop) or an internal battery laptop.

3. Input and Output Devices:

Depending on particular pc, a variable of devices can be connected to send data into it or out from it, common place input data include Mia/laptop and touchpads web icons and ergodic keyboard white output devices are printers, monitors & speaker, removable media like SD cards and flash devices can also be utilized for transferring data between PCs can visit the website for more information.

4. CPU [Central Processing Unit]:

The CPU is the pc's brain since it works the hardest. A CPU does all the calculations needed for a system and varies in speed. The CPU generates that least, and that why a far is installed inside the pc. More powerful CPUs are required for instance computer work or work that necessitate programming multifaceted software or editing high-definition video.

SIMULATION OF PIPLINE

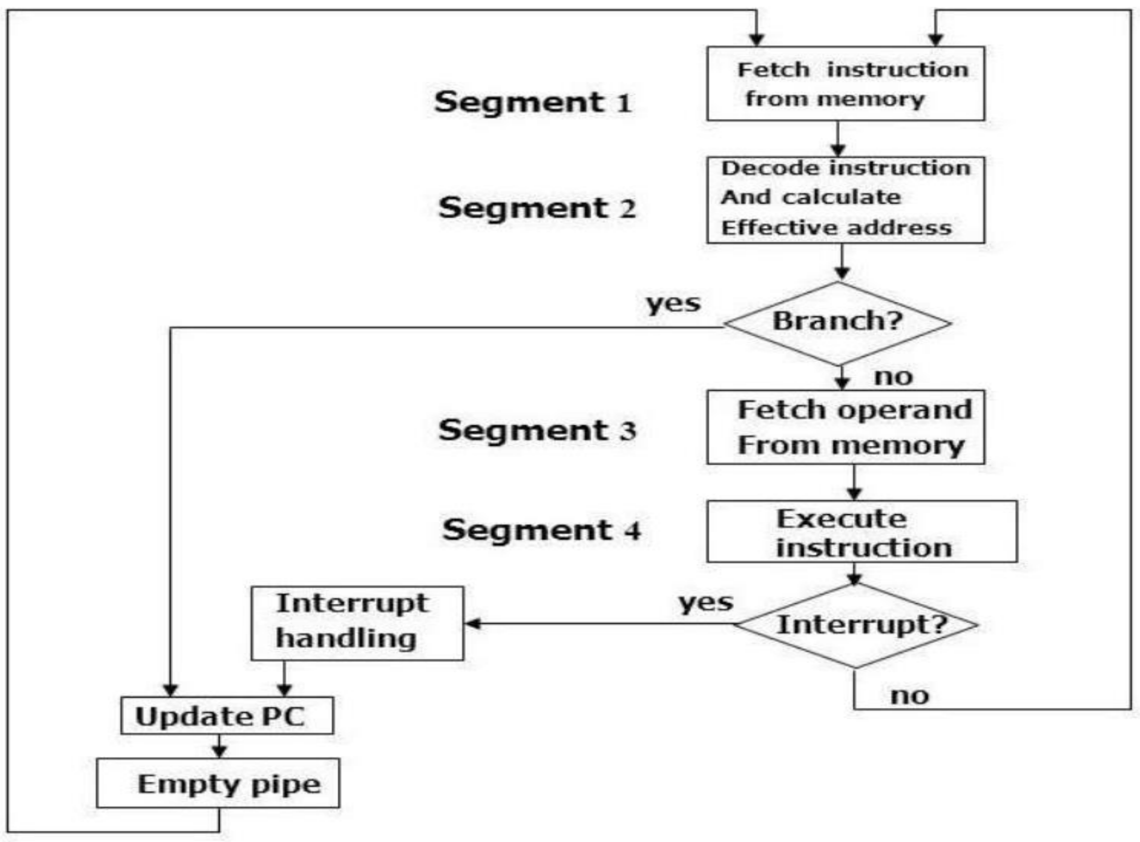
Aim:

To study pipeline feature and instructions pipeline.

Procedure:

- Pipeline processing can occur not only in the data stream but in the instruction stream as well.
- Most of the digital computer with complex instructions require instructs pipeline to carry out operations like fetch, decode and execute instruction. Sequence steps.

  1. Fetch instruction from memory.
  2. Decode the instruction
  3. Calculate the effective address
  4. Fetch the operands
  5. Execute the instruction
  6. Store the secret in the proper place



SIMULATION OF INSTRUCTION LEVEL PARALLELISM

AIM:

To study the simulation of instruction level parallelism using c.

PROCEDURE:

- Pipelining can overlap the execution of construction when they are independent of one another. This potential overlap among instruction is called instruction level parallelism (IPL) since the instruction can be evaluated in parallel.
- The amount of parallelism available within a basic block is quite small. The average dynamic branch frequency in integer program was measured to be about 15%, meaning that about 7 instruction execute between a pair of branches.
- Since the instructions are likely to depend upon one another, the amount of overlap we can exploit within a basic block is likely to much less than 7.
- To obtain substantial performance enhancement, we must exploit IPL across multiple basic block.
- The simplest and most common way to increase the amount of parallelism available among instruction exploit parallelism among iteration of a loop. This type of parallelism often called loop-level parallelism.

SIMULATION OF CACHE MEMORY

Aim:

To study the simulation of cache memory.

Procedure:

1. Define Cache Parameters:

- Cache Size (S)
- Block/Line Size (B)
- Associativity (A)
- Replacement Policy (e.g., LRU, FIFO)
- Initialization (clear cache, load initial data, etc.)

2. Represent Memory Hierarchy:

- Consider the main memory and cache hierarchy.
- Break down main memory into blocks corresponding to cache lines.

3. Initialize Cache:

- Set all valid bits to false or initialize according to your model.
- If necessary, preload initial data into the cache.

4. Simulate Memory Access:

- For each memory access, determine the cache set using the index bits.
- Check if the desired data is present in the cache (cache hit).
- If hit, update cache state (e.g., LRU information).
- If miss, decide which block to replace (based on replacement policy).
- If replacement is required, load the new block into the cache.
- Update cache state accordingly.

5. Track Cache Hit/Miss Statistics:

- Maintain counters for cache hits and misses.
- Record additional information for analysis, such as access patterns.

SIMULATION OF MULTI PROCESSOR

AIM:

To simulation of multiprocessor in c.

PROCEDURE:

- Multiprocessing is how a computer is able to executed multiple program concurrently.
- One of the main workhouse function that makes multiprocessing possible is a function in c \ c++ that a process & pairing an identical copy. A process is a instance of a program.

DEFINE SYSTEM ARCHITECTURE:

Specify the number of processors, memory hierarchy, cache configurations, interconnection network, and shared resources. Determine the simulation time scale and granularity.

SELECT SIMULATION TOOLS OR FRAMEWORK:

Choose simulation tools or frameworks suitable for your modeling needs. Options include discrete event simulators, cycle-accurate simulators, or high-level architecture simulators.

PROCESSOR MODELING:

Model's each processor's architecture, instruction set, and pipeline structure. Include components like instruction fetch, decode, execution units, and caches. Choose an appropriate level of abstraction based on simulation goals.

MEMORY HIERARCHY MODELING:

Model the memory hierarchy, including caches, main memory, and any other levels such as shared caches. Specify cache coherence protocols if applicable.

INTERCONNECTION NETWORK:

Model the interconnection network that facilities communication between processors and memory. Choose the appropriate network topology and communication protocols.

SHARED RESOURCES MODELING:

If the multi-processors system includes shared resources (e.g., buses, I\O devices), model their behavior and contention. Implement any necessary arbitration or scheduling mechanisms.

EIGHT-BIT ADDER

Aim:

To implement Full Adder using Logisim software.

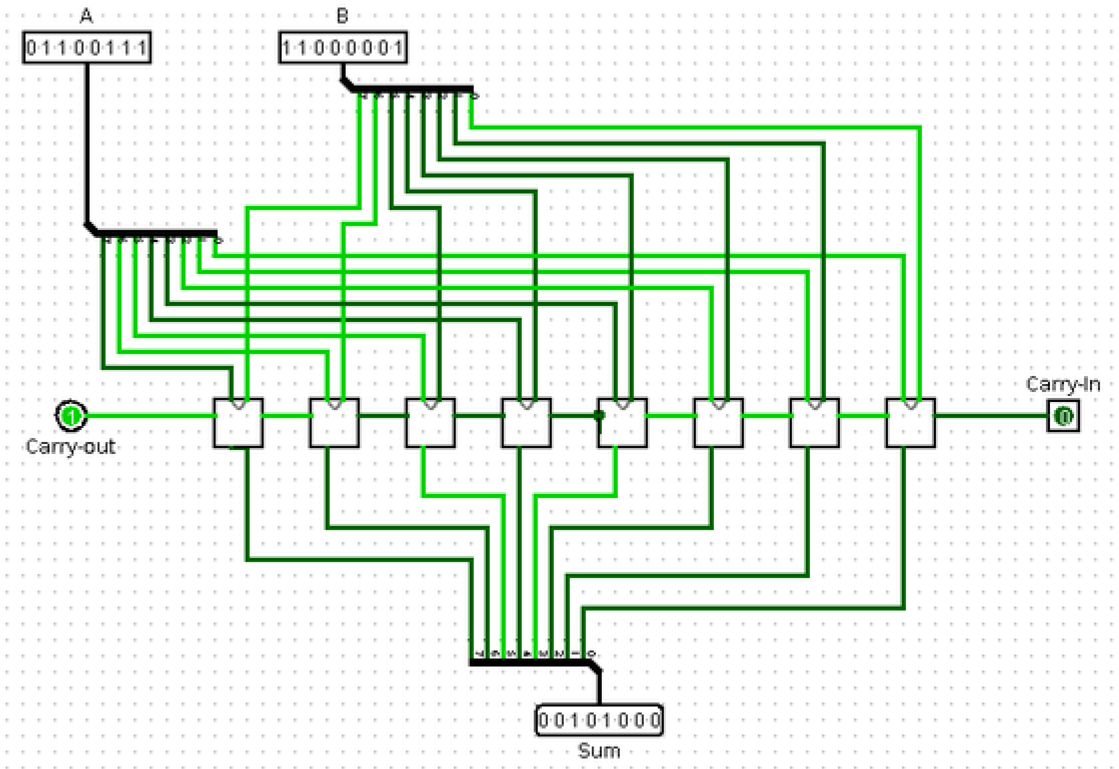
PROCEDURE:

Full Adder:

A Full Adder is composed of two half adders and an additional OR gate. It is used in many digital circuit such as microprocessor and calculator. It can also be combined with others full address to create larger binary adders for performing arithmetic operations or larger binary numbers.

1. The purpose of the circuit is to take two eight-bit numbers and a one-bit carry-in and to produce an eight-bit sum and a one-bit carry-out as output.
2. We saw in class how to construct multiple-bit adders from several one-bit adders. Use eight copies of the Adder that you created one-bit adder.

CIRCUIT DIAGRAM:



RESULT: The eight-bit adder circuit is executed successfully, and results are obtained.

ONE-BIT ADDER

Aim:

To implement One-Bit Adder using Logisim software.

PROCEDURE:

1. The input value of A and B goes into XOR gate and C-in also goes into XOR gate.
2. Both the XOR gate meets the AND gate which then goes to OR gate.
3. Finally output is obtained as sum and c-out.

TRUTH TABLE:

INPUTS **OUTPUTS**

A	**B**	**C-In**	**SUM**	**C-Out**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

EIGHT-BIT SUBTRACTOR

Aim:

To implement Full Subtractor using Logisim software.

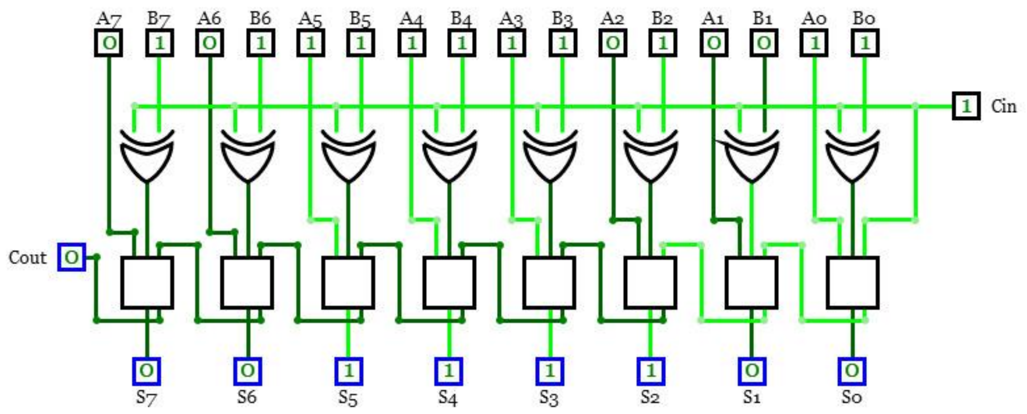
PROCEDURE:

Half Subtractor:

A half subtractor is a digital circuit that performs binary subtraction on two single bit inputs produce a difference and a borrow output. It is composed of a XOR gate and a NAND gate. It is used in many digital circuit, such as microprocessor and calculator.

1. since subtracting A−B is the same as adding A+(−B). However, −B can be computed by negating B and adding 1. This means that A+(−B) = A+(NOT B)+1.
2. To make a subtraction circuit, start by adding an 8-bit adder to the circuit. Add two 8-bit inputs and one 8-bit output. The first input can be directly connected to the first input of the adder.
3. The output can be directly connected to the output from the adder.
4. We actually have a nice way to deal with the +1 at the end: The adder circuit has a carry-in input. By connecting a constant value 1 to the adder's carry-in, we can add the final 1 that we need for subtraction.

CIRCUIT DIAGRAM:



RESULT: The Tiny ALU circuit is executed successfully, and results are obtained.

TINY ALU

Aim:

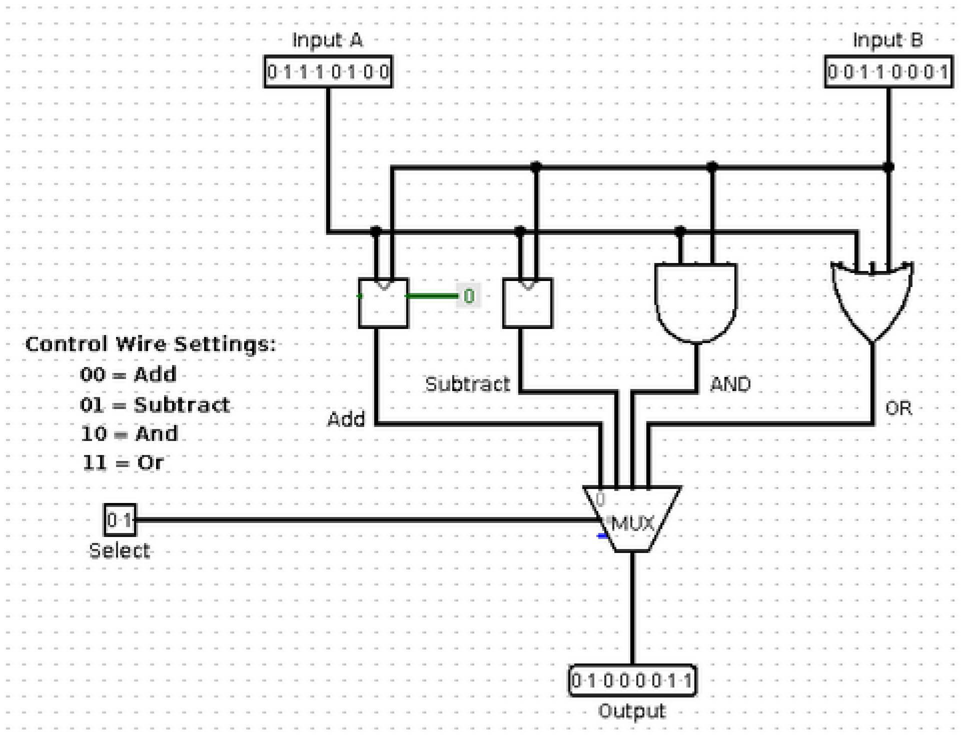
To build a Tiny ALU using Logisim software.

PROCEDURE:

An ALU (arithmetic-logic unit) in a computer performs arithmetic and logical operations on numbers. The same ALU can perform several different operations, and it has control wire inputs to tell it which operation to perform.

1. The ALU that you build for this exercise will do four operations on 8-bit numbers: Add, Subtract, And, and Or.
2. To select which operation to perform, there will be two wires, which can be represented by one 2-bit input.
3. If the control wires are set to 00, the ALU should add its two inputs; if they are set to 01, the ALU should subtract; if they are set to 10, it should apply AND; and if they are set to 11, it should apply OR.
4. To add and subtract numbers in your ALU, we can use the 8-bit adder and the 8-bit subtracter that you created earlier in the lab.

CIRCUIT DIAGRAM:



RESULT: The Tiny ALU circuit is executed successfully, and results are obtained.

XOR GATE

AIM:

To build an XOR gate out of AND, OR and NOT gate.

PROCEDURE:

A XOR Gate is digital logic gate that has two inputs and one output of an XOR gate is true only when its inputs are different.

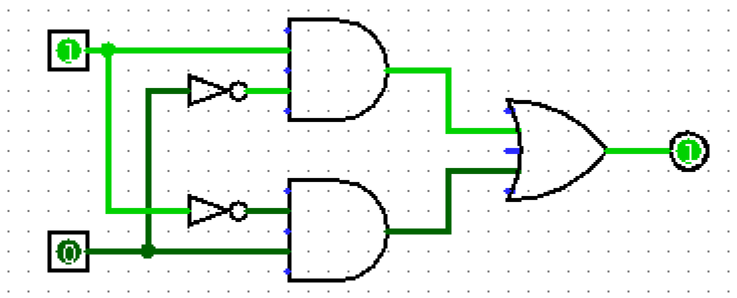
$Y=A\oplus B=A{}^{-}B+{}^{-}AB$

1. A and B are the two input variables to the XOR gate.
2. Y is the output variable of the XOR gate.
3. The output expression of the XOR gate is read as Y is equal to A ex-or B.
4. Using two NOT and NAND gates for both the inputs A and B.
5. Finally the two NAND gates goes into OR gate which produces the Output.

TRUTH TABLE:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

CIRCUIT DIAGRAM:



RESULT: The logic circuit is executed successfully, and results are obtained.

SIMULATION OF VECTOR PROCESSOR

AIM:

To study of vector processor.

PROCEDURE:

A vector processor consists of scalar processor and a vector unit, which could be thought of as an independent functional unit capable of efficient vector operation.

VECTOR HARDWARE:

Vector computer have hardware to perform the vector operation effectively. Operands cannot be used directly from memory rather are loaded into register and are loaded into register and are put back in register after the operation.

PIPELINING:

- The pipeline is divided up into individual segment, each of which is completely independent and involves no hardware sharing.
- This ability enable it to produce in result per clock as soon as the pipeline is full.
- The processing of a number of operands may carried out simultaneously.
- The loading of a vector register is itself a pipelined operation, with the ability to load one element each clock period after some initial startup overhead.

DEFINE ARCHITECTURE:

Understand the architecture of the vector processor you want to simulate. This includes the structure of vector registers, instruction set, pipeline stages, and memory access patterns.

SELECT SIMULATION:

Choose a simulation environment or platform. This could be general-purpose programming language (e.g., C++, Python) or specialized tools designed for processor simulation (e.g., Gem5, Simics).

SIMULATION OF THREAD LEVELPARALLELISM

Aim:

To study simulation of thread level parallelism.

Procedure:

A program can be quickly executed by introducing concurrency using threads. Each thread can execute part of the same task or it can allocate the same task for different clients in a client server architecture.

1. **Header File:**

   Include the header file p thread.h
   #include<p thread.h>

2. **The ID For a Thread**

   Each thread has an object of type p thread t associated with it that tells its ID.
   p thread_t id [2];

3. **Creating a thread:**

   A thread is created and starts using the function p thread_create(). It takes for parameter.
   - ID _parameter – reference (or pointer) to the ID of the thread.
   - Starting routine – void*_this name of the function that the thread starts to execute if the function's return type is void.
   - Argument – void*_this is the argument that the starting routine tasks. If the task multiple arguments, a struct is used.

4. **Existing a Thread:**

   P thread_exit() is used to exit a thread. //Global Variable;
   Int i=1;

5. **Waiting for a Thread:**

   A parent thread is made to wait for a child using p thread join(). The two parameters of the functions are. Thread ID_p thread_t – The ID of the thread that the parent waits for Reference to return value – void*_the value returned by the existing thread is caught by this pointer.
   Int*ptr;
   pthread_join(id, & ptr)

SIMULATION OF DATA PARALLELISM

AIM:
To study simulation of data parallelism.

PROCEDURE:
- Data parallelism is parallelization across multiple process in parallel.
- It focuses on distributing the data across different nodes, which operate on the data in parallel.
- It contrasts task parallelism as another from of parallelism.
- A data parallel job on an array of n element can be divided equally among all the process.
- Let us assume we want to sum all the element of the given array and the for a single addition operation is time unit.
- One important that they to node is that the locality of the data reference play an important part in evaluating the performance of a data parallel programming model.
- In a multiprocessor register executing a single set of instruction (SIMD), data parallelism is achieve when each processor perform the same task on different distributed data.

DEFINE THE PROBLEM:

Clearly define the problem you want to simulate with data-level parallelism. Identify the data-level parallelism opportunities within the algorithm or application.

SELECT A SIMULATION TOOL OR ENVIRONMENT:

Choose a simulation tool or environment that supports the modeling of parallel processing. This could be a general-purpose simulation tool, a parallel programming framework, or a domain-specific simulation platform.

MODEL THE ALGORITHM:

Develop a model or representation of the algorithm you want to simulate. Identify the sections of the algorithm where data-level parallelism can be exploited. This may involve breaking down the algorithm into tasks or operations that can be performed concurrently.

PARALLELIZE DATA OPERATIONS:

Identify data-level parallelism within the algorithm and parallelize the relevant data operations. This might include tasks like vectorization, SIMD (Single Instruction, Multiple Data) parallelism, or using parallel processing constructs in your chosen simulation environment.

MICROPROCESSOR-8086

Aim:

To study the simulation of Microprocessor-8086

Procedure:

Instruction Set:

- **Data Transfer Instructions**

  These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group-

- **Arithmetic Instructions**

  These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.

- **Bit Manipulation Instructions**

  These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.

- **String Instructions**

  String is a group of bytes/words and their memory is always allocated in a sequential order.

- **Program Execution Transfer Instructions (Branch & Loop Instructions)**

  These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

- **Processor Control Instructions**

  These instructions are used to control the processor action by setting/resetting the flag values.

- **Iteration Control Instructions**

  These instructions are used to execute the given instructions for number of times.

- **Interrupt Instructions**

  These instructions are used to call the interrupt during program execution