

**Fayoum University**

**Faculty Of Engineering**

**Computer Engineering Course**

**Project Phase 1**

**Single Cycle Processor Design**

**Supervision by:**

**Dr/ Gihan Naguib**

**ENG/ Jihad Awad**

**Group Names (ECE Department):**

**1: Ahmed osama Abd-Elgaffar**

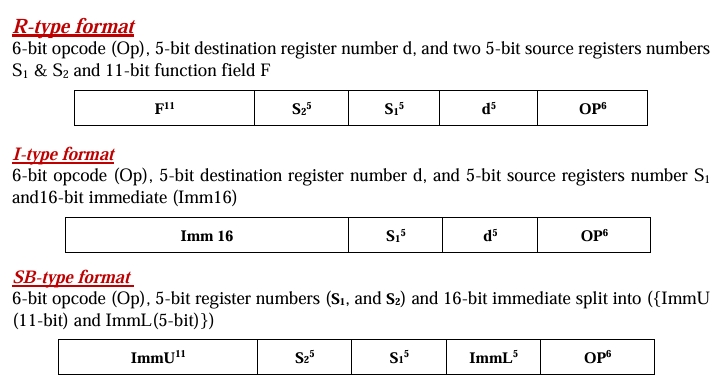
**2: Omer Ahmed Othman**

**3: Mohamed Nady Mahmoud**

|  |  |  |
| --- | --- | --- |
| **Single Cycle Processor** | | |
|  | [**Introduction**](#Itroduction) | **1** |
|  | **Quick Look on Whole Datapath** | **2** |
|  | **Single-Cycle Architecture** | **3** |
|  | [**Register File**](#RegisterFile) | **4** |
|  | [**Arithmetic & Logic Unit (ALU)**](#ArithmeticAndLogicUnit) | **5** |
|  | [**Instruction Memory**](#InstructionMemory) | **6** |
|  | [**Data Memory**](#DataMemory) | **7** |
|  | [**Program Counter (PC)**](#ProgramCounter) | **8** |
|  | [**Control Unit**](#ControlUnit) | **9** |
|  | **Simulation & Testing** | **10** |

**1- Introduction**

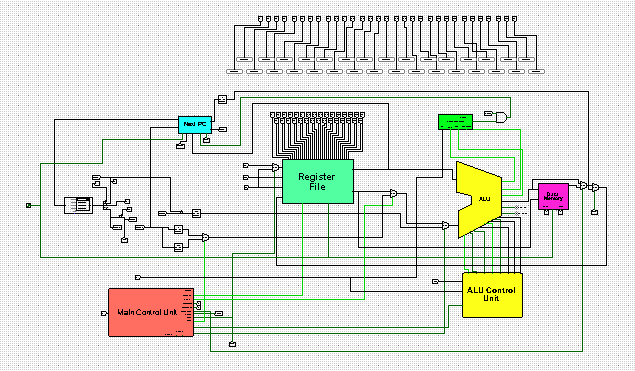
We were asked to design a simple 32-bit RISC processor with 32-bit general purpose registers: R1 through R31. R0 is hardwired to zero and cannot be written, we are left with seven registers. There is also one special-purpose 16-bit register, which is the program counter (PC). All instructions are only 32 bits. There are three instruction formats; R-type, I-type, and SB-type as shown below



We used blocks methodology in the whole project that allows us to design the logical circuit that does the functionality we want then put it in a block to use it in the datapath. This methodology prevents the complex design and view of the circuit because the wiring and details of the circuit are hidden inside the block.

**2-Quick Look on Whole Datapath**

This is the final design of the whole datapath and we will explain each part independently and in details:



**3- Single-Cycle Architecture :**

* [**Register File**](#RegisterFile)
* [**Arithmetic & Logic Unit (ALU)**](#ArithmeticAndLogicUnit)
* [**Instruction Memory**](#InstructionMemory)
* [**Data Memory**](#DataMemory)
* [**Program Counter (PC)**](#ProgramCounter)
* [**Control Unit**](#ControlUnit)

**4-Register File:**

Our implemented Register File boasts (32) 32-bit registers denoted as R1 to R31

It has two read ports and one write port. Notably, R0 is hardwired set to zero, serving as a default value

This register file features dual sources, a single destination input and a data write input. AddItionally, it is outfitted with two output buses to facilitate data output from the register file.

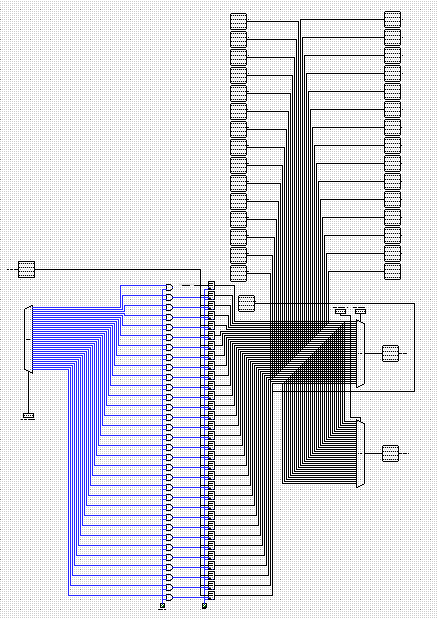
A pivotal control signal, RegWrite, is integrated to regulate the enabling or disabling of writing within the register file. Lastly, the Register File accommodates an input specifically designated for the Clock signal, ensuring synchronous operation

**Register File Structure**

In our design, we have implemented a decoder equipped with a Write Register selector, facilitating the seamless designation of the target register for writing operations. This decoder selects from R1 to R31, serves as a pivotal component in directing data to the appropriate register within the Register File.

To ensure precise control over the writing operations, each terminal of the decoder is intricately linked with the RegWrite signal through AND Gates. This configuration allows for effective regulation of writing Furthermore, we used two Multiplexers to enrich the functionality of our system by enabling the selection of output ports. This feature grants us to choose between utilizing one or both of the output ports depending on the instruction

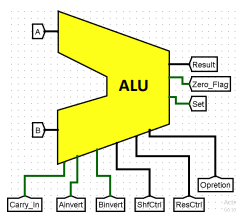
As part of our validation strategy during simulation, we have included seven output ports, each dedicated to a respective register within the Register File to just reading wile testing processing but it are not main components in the Register File.

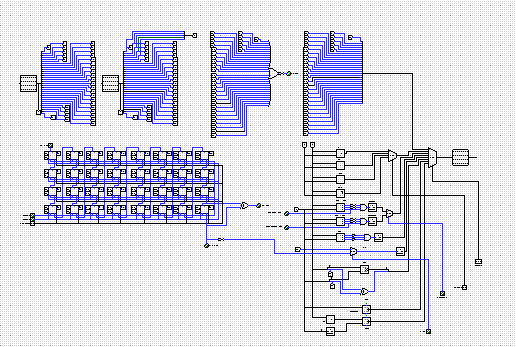
****

**5-Arithmetic and Logic Unit (ALU):**

**ALU Overview**

Embedded within the CPU, the Arithmetic and Logic Unit (ALU) executes arithmetic and logic operations on operands found in computer instruction words ALU has two inputs (A and B) and a single main output for result, so that ALU delivers the outcome of the operation It incorporates essential signals like Set and Zero Flag which used for branch instructions Governed by the ALU control unit (which will be explained later)the ALU receives the following signals (Carry In, Ainvert, Binvert, ResCtrl, ShfCtrl, and Operation), which determined by ALU control unit due to the instruction needed to execute.





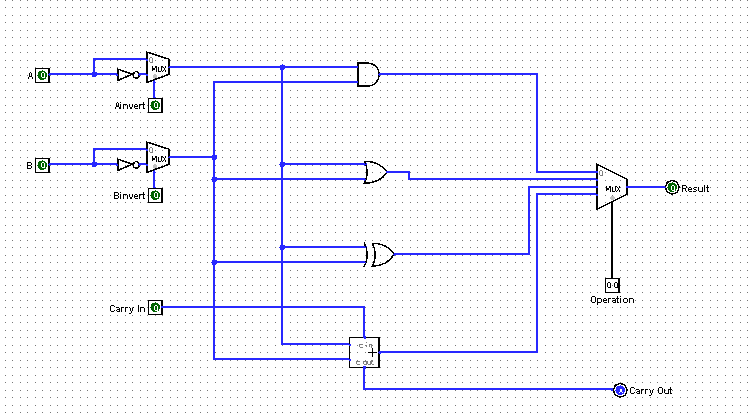
**ALU Structure**

Our ALU consists of 2 main parts, Arithmetic & Logic part and shift & Rotate Part, we will show each part in details

**(1) Arithmetic & Logic part**

This part is responsible for instructions that needs an arithmetical or logical operation (AND, OR, XOR, NOR, Add, Sub, SLT, SLTU, ANDI, ORI, XORI, AddI, LW, SW)

It consists of 32 1-Bit ALU (one for each bit ) linked together as carry out of each one is the carry in of the next one.



Each 1-Bit ALU has 2 main inputs for operands A and B, and 1 main output is the result, there are also signals inputs ( AInvert, BInvert, Less ) which controls different operations

ALU operations divided into two types, Logical and Arithmetical as follow:

**1) Logic Operations:**

-Employs logic gates (AND, OR, XOR, NOR) for logical operations.

- For AndI, ORI, XORI, and AddI operations, similar to R-Type, the second ALU input originate from the extended immediate, differing only in the control signal (ALU Src) set to one.

**2) Arithmetic Operations:**

- Utilizes full Adders for AddItion and subtraction.

- AddItion involves summing the two inputs (A, B) with a carry-in of zero.

- Subtraction sets Carry in = 1, resulting in the multiplexer output being the inversion of B: A + B' + 1 = A + (B' + 1) = A + (-B) = A - B.

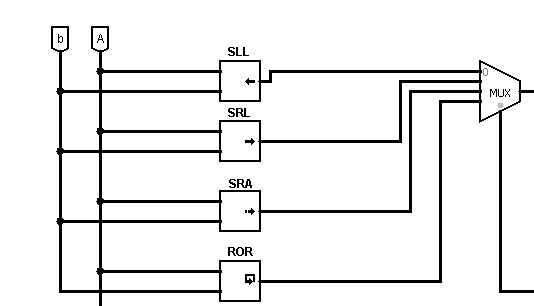
- SLT and SLTU implementation involves setting all output bits except the LSB to 0, with the LSB set to 1 if (A - B) is negative and 0 otherwise, aligning with the sign bit value of (A - B) where 1 denotes negative and 0 denotes positive. (but SLT deals with signed values ,while SLTU deals with unsigned values using comparator ).

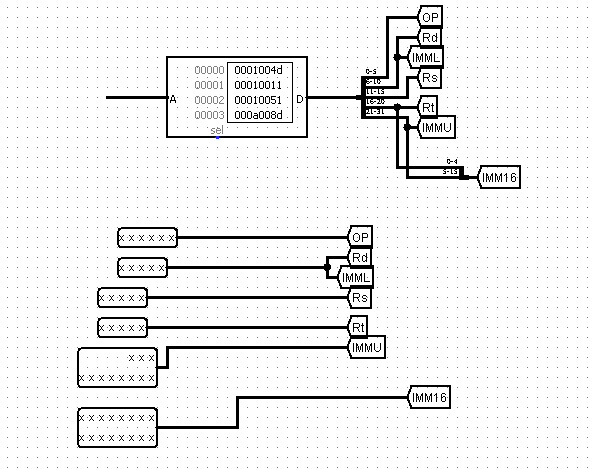
**(2) Shift & Rotate**

This part is responsible for instructions that use shifters ( SLL, SRL, SRA, ROR ) it also has a datapath for SLTU instruction.

It Executes logical shift left, logical shift right, shift right arithmatic, and rotate right operations. Shifting involves moving all bits within a register left or right. SLL denotes shift left logical (zero insertion from the right). SRL signifies shift right logical (zero insertion from the left). SRA denotes shift right arithmetic (sign bit insertion from the left).

Utilizes shifters for SLL, SRL, and ROR operations, with the shift and rotate amount being the unsigned immediate 5-bit value (bits 0-4) of the second ALU input (B) ; while shifter for SRA uses the signed immediate 5-bit value (bits 0-4) of the second ALU input (B).



**6-Instruction Memory:**

Instruction memory, depicted in the figure, functions as a ROM (Read-Only Memory) crucial for permanent data storage.

Stored data in ROM consists of four hex numbers (words), providing essential instructions for the system.

It operates on a word-Addressable basis, with input A sourced from the next PC. ensuring access to the memory.

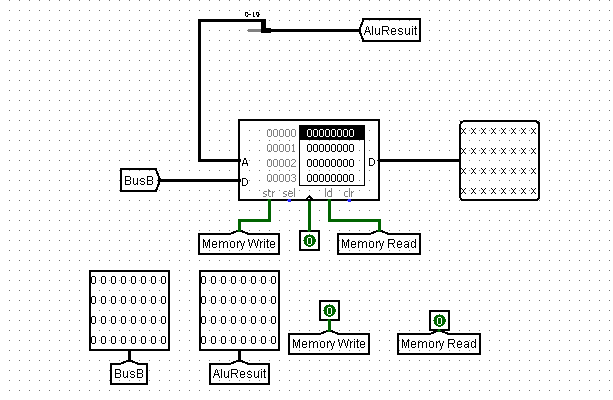
The output comprises 16 bits, representing:

• For R-type format: 5-bit opcode (Op), 3-bit destination register Rd, and two 3-bit source registers Rs & Rt and 2- bit function field.

• For I-type format 5-bit opcode (Op), 3-bit destination register Rd, 3- bit source register Rs, and 5-bit immediate.

• For J-type format 5-bit opcode (Op) and 11-bit Immediate.

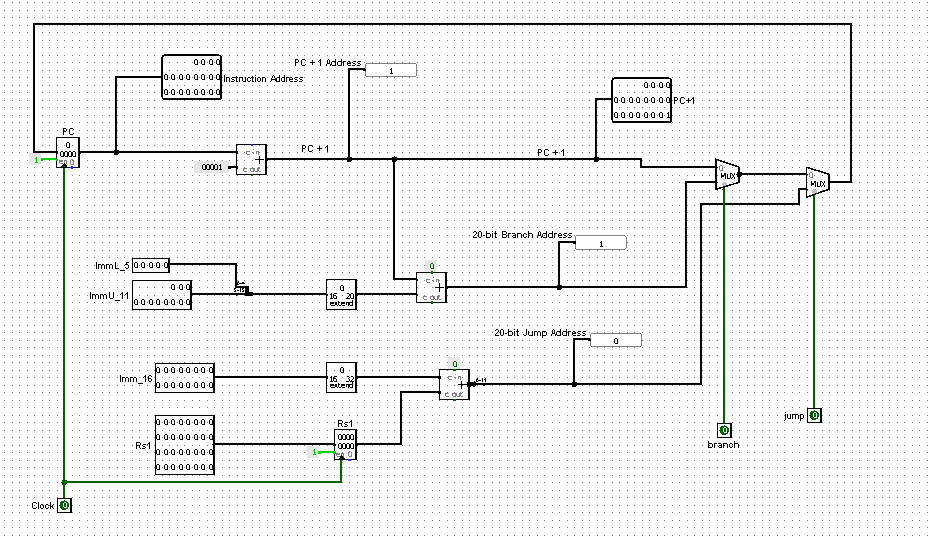
**7-Data Memory:**

 In our Processor, Data Memory serves as the repository for storing data post-instruction execution, facilitating its retrieval for subsequent use. To ensure optimal performance and versatility, we opted for RAM (Random Access Memory) as our choice for Data Memory. RAM's inherent speed and random access capabilities allow the processor to SWiftly access any location within the memory without the need to sequentially search through all Addresses.

Data manipulation within the Data Memory is orchestrated through the execution of LW (Load Word) and SW (Store Word) instructions, governed by the MemRd and MemWr control signals. Two primary inputs drive the Data Memory operations: firstly, the 16-bit result generated by the ALU determines the memory Address to be accessed; secondly, the Read Data 2, sourced from the Register File, is contingent upon the RegDst control signal, which designates whether it originates from Rd or Rt.

At the output stage of the Data Memory, a Multiplexer Selects between the ALU Result (when memory access is unnecessary) and the memory output (during data loading), guided by the MemtoReg control signal. Subsequently, another Mux is employed to determine whether to execute the JAL (Jump and Link) instruction, wherein the value of PC+1 from the Program Counter (PC) Block is selected, or to proceed with the first Mux value, initiating data writing into the Register File or execution of the LUI (Load Upper Immediate) instruction, contingent upon signals from the Control Unit.

**8-Program Counter (PC):**



This unit is responsible for producing the PC that will be used to indicate the next instruction and also to write it to the register in the case of JAL.

This unit performs many operations:

1. NextPc (PC +1).
2. Branching (PC + sign\_extend(Immediate 5)).
3. Jump (PC + sign\_extend(Immediate 11)S ).
4. JAL (PC = RS).

|  |  |
| --- | --- |
| **Operation** | **PCsrc** |
| **NextPC** | 00 |
| **J and JAL** | 01 |
| **Branch** | 10 |
| **JR (PC = RS)** | 11 |

**9-** **Control Unit:**

Control units generate required signals which determine suitable data path to execute each instruction in the processor. We have 3 control units each one generates special signals as follow:

**(1) Main Control Unit**

Signals: ( RegWr, RegDst, J, Jal, Lui, ExtOp, ALUSrc, MemRd, MemWr, MemtoReg )

**(2) ALU Control Unit**

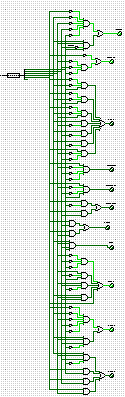
Signals: ( Carry\_In, Ainvert, Binvert, ShfCtrl, ResCtrl, Operation, Jr )

**(3) Branch Control Unit**

Signals: ( Branch )

**(1) Main Control Unit:**

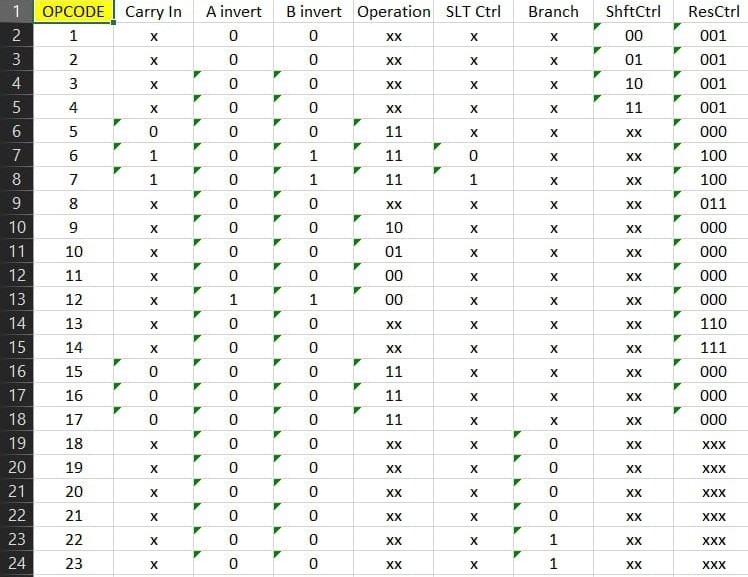
We used a splitter ( instead of decoder ) to take only last 5 bits in the instructions as the Op code which generates the signals. We didn’t Add R-Type signals as ALU control unit can take last 5 bits in the instructions as the Op code directly.



**(2) ALU Control Unit**

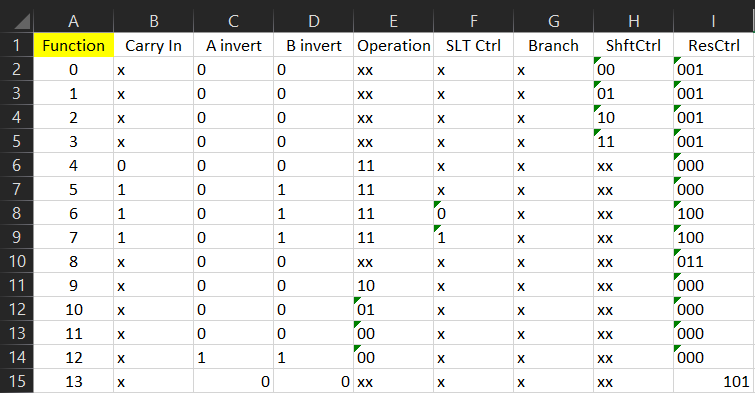
We used 2 splitter ( instead of decoder ), upper one take only last 5 bits in the instructions as the Op code and lower one take Bits 9 and 10 as the Function and both together determine which instruction to be executed in the ALU.

**Instructions Truth Table**

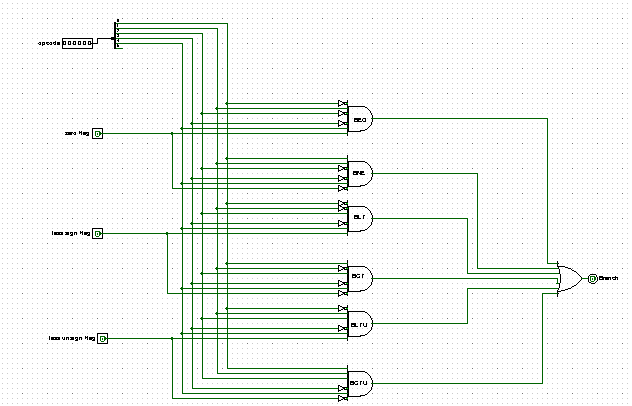


**(3) Branch Control Unit**

It is the unit that executes branch instructions. It has 3 Inputs, Set and Zero Flag are signals generated by ALU and the same 5-Bit Op code.

****

It has 1 output Branch signal that execute the branch at Next PC Block.



**10-Simulation & Testing.**

While we were testing we found a problem with the branch instruction

All instruction happens to extract the output individually correct.

Test code is working fine , with a little modification on Branch we will provide a simulation screenshots.