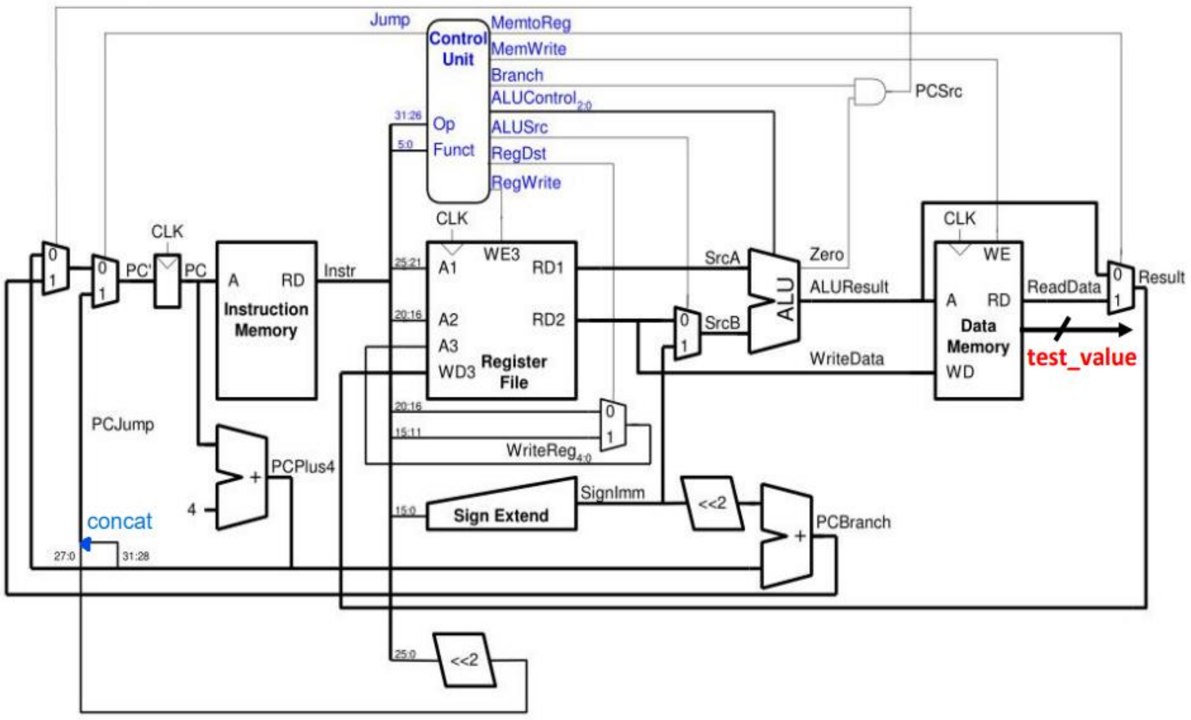
Phase 2, project 1: Single Cycle MIPS processor

Introduction:

You are required to implement a 32-bit single-cycle microarchitecture MIPS processor. The single cycle executes an entire instruction in one cycle. In other words, instruction fetch, instruction decode, execute, write back, and program counter update occurs within a single clock cycle.

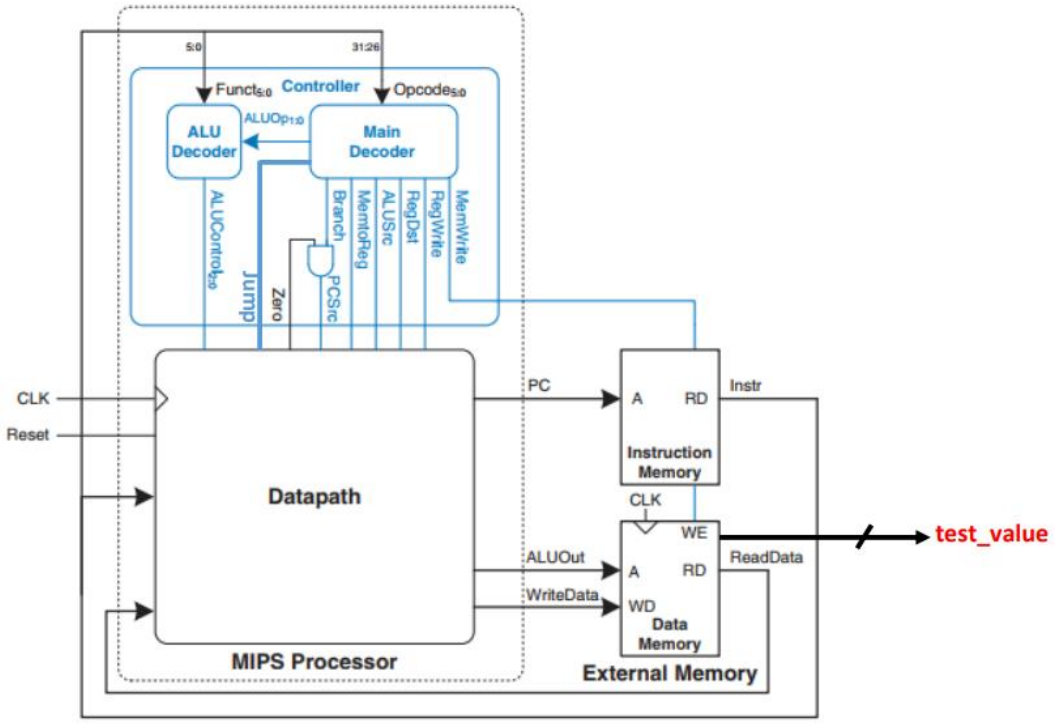
The objective is to write the RTL Verilog files for all sub-modules of the MIPS processor (Instruction memory, data memory, ALU, etc.) then implementing the top module of MIPS processor.



Complete single-cycle MIPS processor. (from David M. Harris, Sarah L. Harris - Digital Design and Computer Architecture)

Top module view:

The processor is composed of The processor is composed of a datapath and a controller. The controller, in turn, is composed of the main decoder and the ALU decoder. Here is a block diagram of interfacing with external memories.



MIPS single-cycle processor interfaced to external memory. (from David M. Harris, Sarah L. Harris - Digital Design and Computer Architecture)

Main modules:

1- ALU: The ALU forms the heart of most computer systems. The 3-bit ALUControl signal specifies the operation. The ALU generates 32-bit ALUResult and a zero flag to indicate if result == 0.

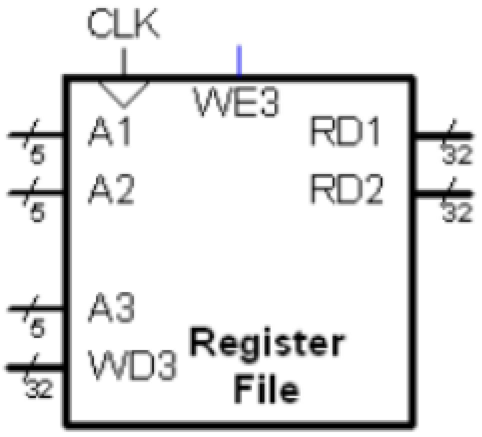
ALUControl	Function
000	Bitwise AND
001	Bitwise OR
010	Addition
011	Not used
100	Subtraction
101	Multiplication

110	Set less than (aluresult = 1 if srca < srcb)
111	Not used

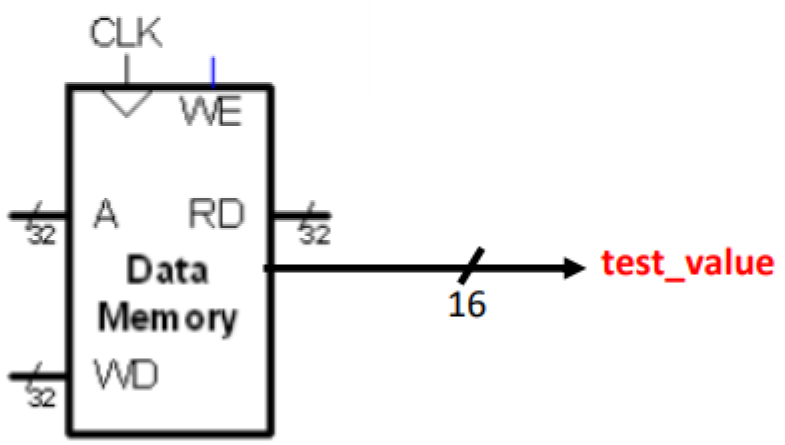
2- **Program counter:** program counter register contains the 32-bit address to execute. The PC is updated at the rising edge of clk and cleared whenever the active low reset is asserted asynchronously.

3- **Instruction memory:** PC is connected to the address input of the instruction memory. The instruction memory fetches the 32-bit instruction. It is a ROM that holds the program that your CPU will execute. This ROM has width = 32 bits and depth = 256 entries (1KB ROM), i.e, you will only connect bits 9 to 2 in the address bus as it is byte aligned. Instructions are read asynchronously.

4- **Register File:** The Register file contains the 32 32-bit MIPS registers. The register file has two read output ports (RD1 and RD2) and a single input write port (WD3). The register file is read asynchronously and written synchronously at the rising edge of the clock. The register file supports simultaneous reads and writes. The register file has width = 32 bits and depth = 32 entries. Note that there is a write enable signal that is used to enable writing the new value on the data bus (WD3) to the specified register address (A3).



5- **Data memory:** It is a RAM that provides a store for the CPU to load from and store to. Reads are asynchronous and writes are synchronous to the rising edge of the clk signal. It is the same width and depth of instruction memory. Note that test_value is read from address 0x0000_0000 (first location on memory) the least 16 significant bits. This is for testing purposes only.



6- Control unit:

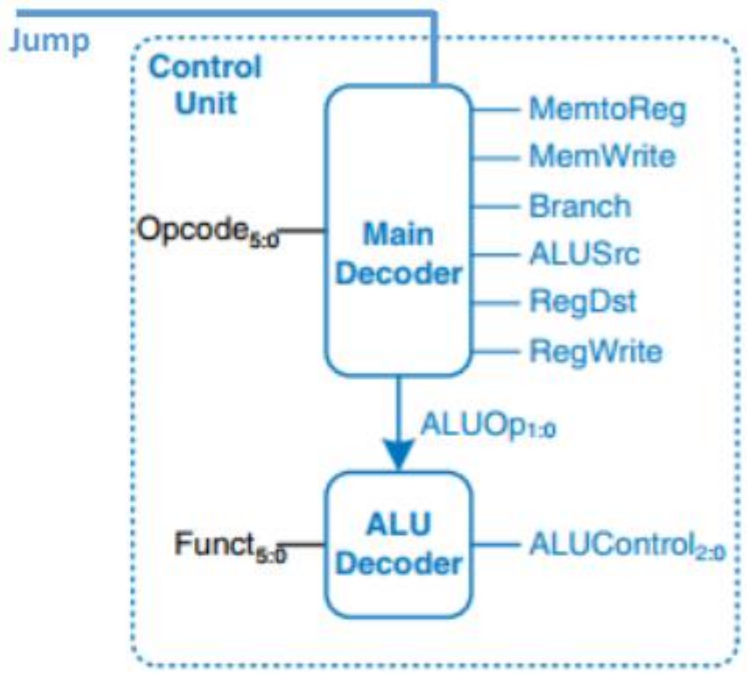
the control unit computes the control signals based on the opcode and funct fields of the instrucrion, instr$_{31:26}$ and instr$_{5:0}$. Most of the control information comes from the opcode, but R-type instructions also use the funct field to determine the ALU operation. Thus, we will simplify that by factoring control unit into two blocks: ALU decoder and main decoder.

ALU decoder truth table:

ALUOp	funct	ALUControl
00	xxxxxx	010
01	xxxxxx	100
10	add = 6’b10_0000	010
	sub = 6’b10_0010	100
	slt = 6’b10_1010	110
	mul = 6’b01_1100	101
Default	xxxxxx	010

opcode	jump	aluop	memwrite	regwrite	regdest	alusrc	memtoreg	branch
lw = 6’b10_0011	0	00	0	1	0	1	1	0
sw = 6’b10_1011	0	00	1	0	0	1	1	0
R type = 6’b00_0000	0	10	0	1	1	0	0	0
addi = 6’b00_1000	0	00	0	1	0	1	0	0
beq = 6’b00_0100	0	01	0	0	0	0	0	1
Jal& j = 6’b00_0010	1	00	0	0	0	0	0	0
Default	0	00	0	0	0	0	0	0

Main decoder truth table:



Small modules:

1. **Sign Extend:** Sign extension copies the MSB of a short input (16 bits) into all the upper bits of the longer output (32 bits).
2. **Shift_left_twice:** you need to make this block parametrized to use in two different versions of data input width.
3. **Adder**
4. **MUX:** you are also required to make a parametrized 2x1 mux.

Testing programs machine codes:

1- Number Factorial: (hex format)

```
00008020
20100007        // change the least two significant decimals with the number
00008820
20110001
12000003
0230881C
2210FFFF
08000004
AC110000
```

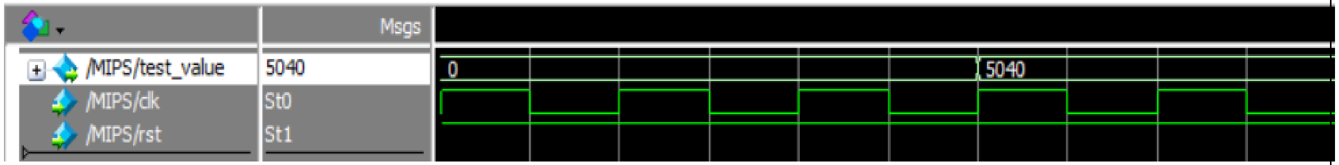
2- Greatest Common Divisor: (hex format)

```
00008020
20100078             // change the least two significant decimals with the number
00008820
201100B4             // change the least two significant decimals with the number.
00009020
12110006
0211482A
11200002
02308822
08000005
02118022
08000005
00109020
AC120000
```

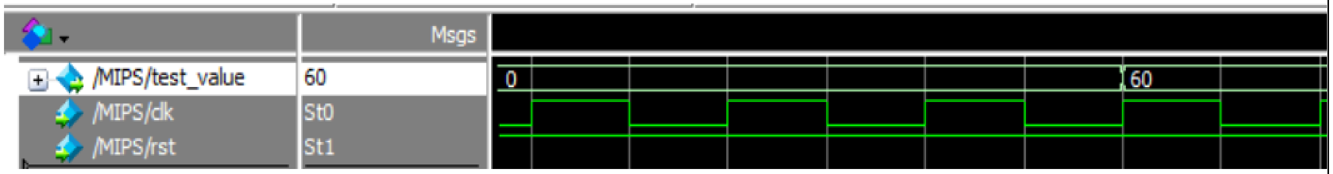
These programs you need to compile and load it onto the instruction memory and reset your processor to start executing from memory location 0000 0000h.

Examples of simulation:

1- Factorial of 7:



2- GCD of 120 and 180



Details of submission:

- [individual] you need to submit a pdf that contains codes and simulation on canvas.
- Due date: 5/10/2023, 11:59 PM