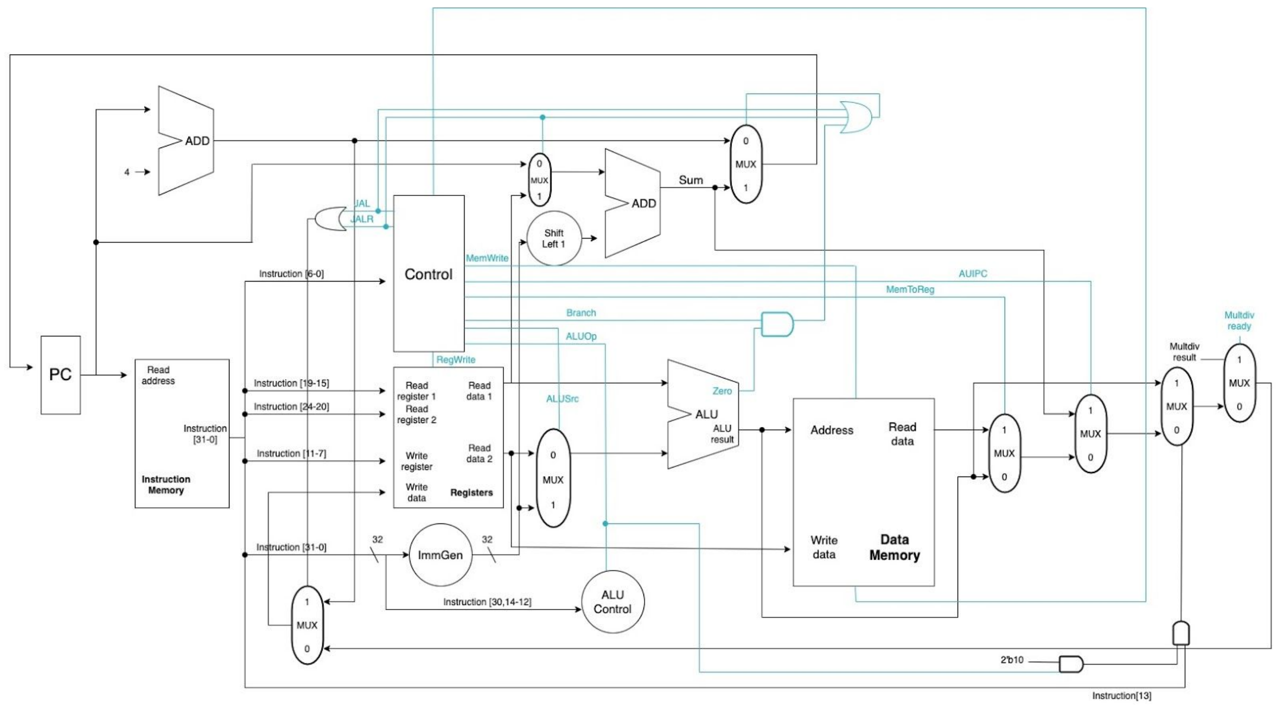
Computer Architecture Final Project

B04502040 徐瑞擇 B05901182 潘彥銘 B05901062 阮明皓

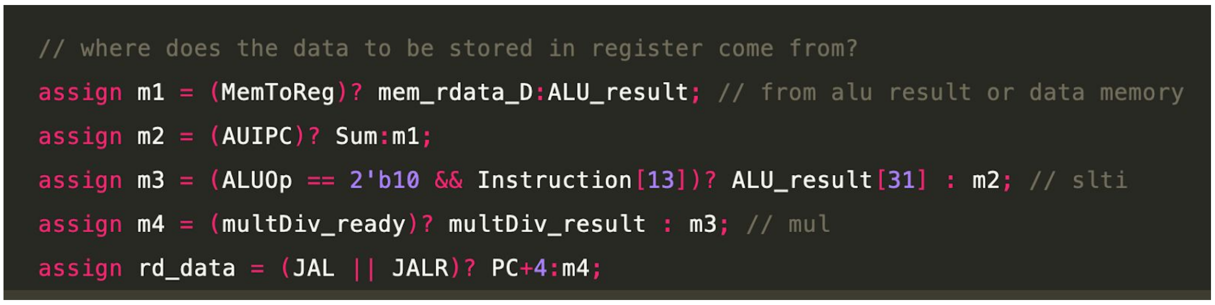
Briefly describe your CPU architecture

The CPU architecture has 2 parts: Combinational part and sequential part.

Combinational part (Part other than multiplication and division):



The Control lines are shown as above. First, we added JAL, JALR and AUIPC outputs to the Control module. Whenever a JAL/JALR/AUIPC instruction is detected, the line corresponding will turn up and send out control signals.
To determine which data should be written back to the register file, we added several MUXs in sequence. The code looks like this:

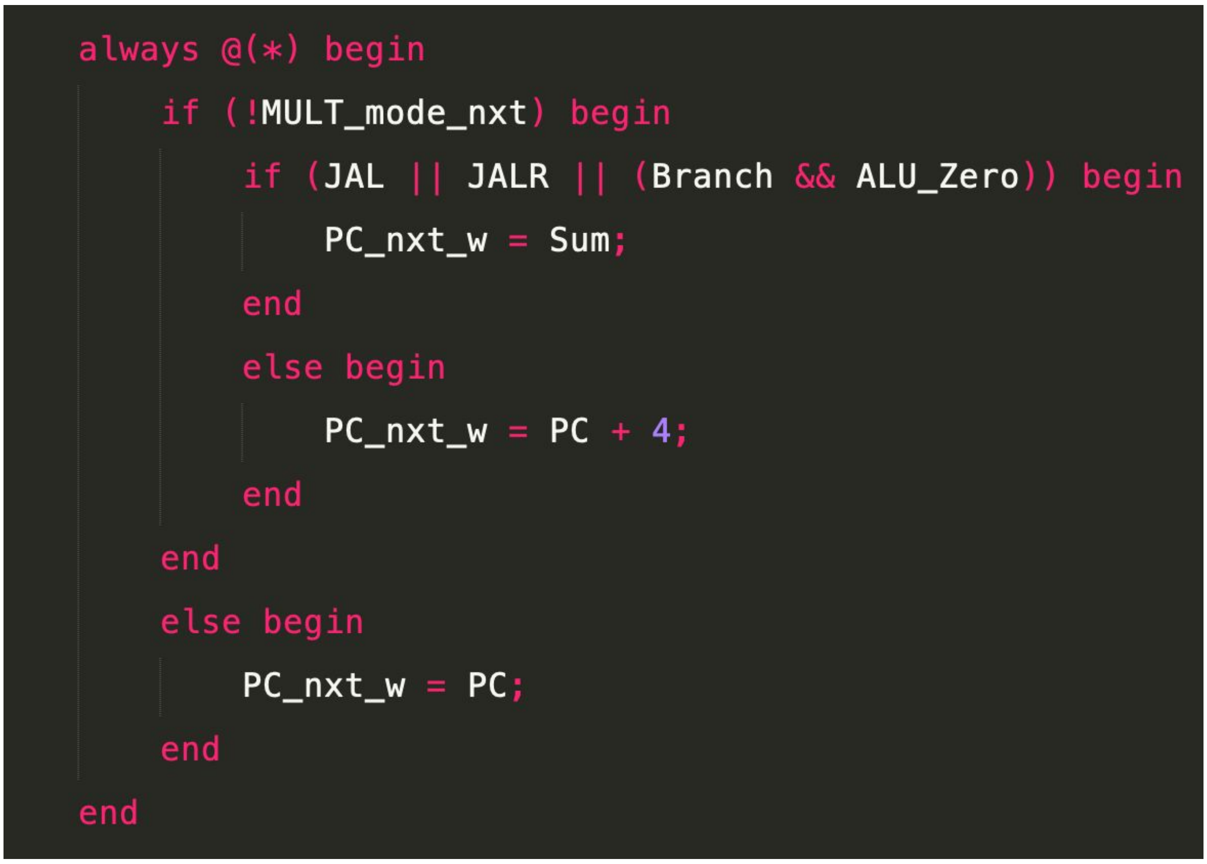


```
// where does the data to be stored in register come from?
assign m1 = (MemToReg)? mem_rdata_D:ALU_result; // from alu result or data memory
assign m2 = (AUIPC)? Sum:m1;
assign m3 = (ALUOp == 2'b10 && Instruction[13])? ALU_result[31] : m2; // slti
assign m4 = (multDiv_ready)? multDiv_result : m3; // mul
assign rd_data = (JAL || JALR)? PC+4:m4;
```

To determine the next address PC points to, we need to know whether the instruction is JAL or JALR.

```
assign Add1 = (JALR)?rs1_data:PC;
assign Sum = (JALR||JAL)?Add1 + ImmGen_out:Add1 + (ImmGen_out << 1);
```

The next PC is then chosen between Sum and PC+4 based on JAL, JALR, Branch and ALU_Zero signals:



```
always @(*) begin
    if (!MULT_mode_nxt) begin
        if (JAL || JALR || (Branch && ALU_Zero)) begin
            PC_nxt_w = Sum;
        end
        else begin
            PC_nxt_w = PC + 4;
        end
    end
    else begin
        PC_nxt_w = PC;
    end
end
```

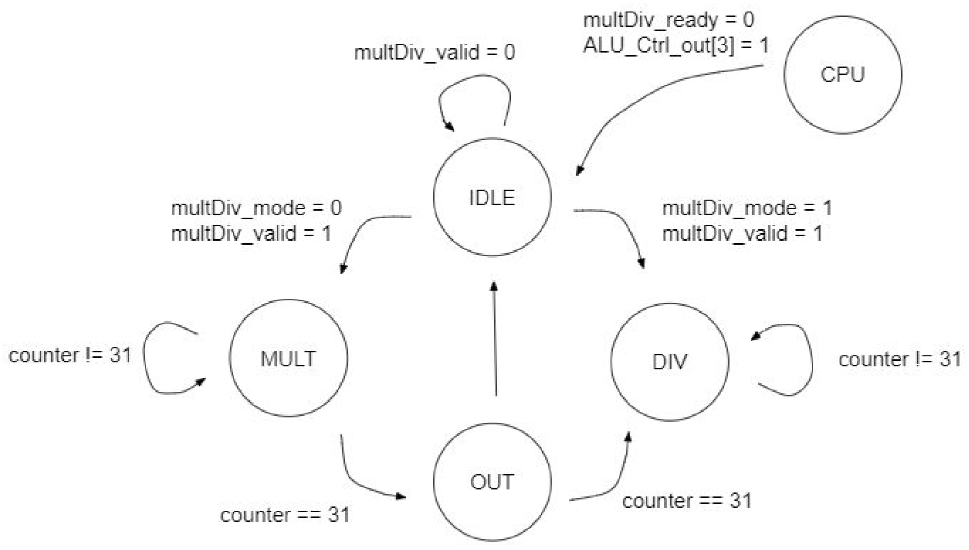
Control:

Instruction	AluSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0	JAL	JALR	AUIPC
Add/Sub	0	0	1	0	0	0	1	0	0	0	0
addi/slti	1	0	1	0	0	0	1	0	0	0	0
lw	1	1	1	1	0	0	0	0	0	0	0
sw	1	x	0	0	1	0	0	0	0	0	0
beq	0	x	0	0	0	1	0	1	0	0	0
jal	x	x	1	0	0	0	x	x	1	0	0
jalr	x	x	1	0	0	0	x	x	0	1	0
auipc	1	x	1	0	0	0	x	x	0	0	1

ALU_Control:

Instruction Opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
lw	OO	Load word	x	x	add	OO1O
sw	OO	Store word	x	x	add	OO1O
beq	O1	Branch if equal	x	x	subtract	O11O
add	1O	add	OOOOOOO	OOO	add	OO1O
sub	1O	sub	O1OOOOO	OOO	subtract	O11O
addi	1O	add immediate	x	OOO	add	OO1O
slti	1O	store 1 if less than imm	x	O1O	subtract	O11O
jal	OO	Jump and Link	x	x	add	OO1O
jalr	OO	Indirect Jump	x	x	add	OO1O
auipc	OO	Store pc-relatice addr	x	x	add	OO1O
mul	1O	multiply	OOOOOO1	OOO	mult	1OOO

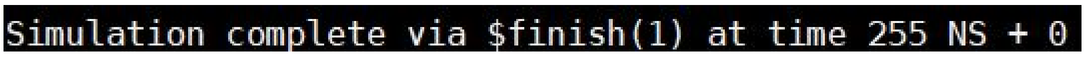
Sequential part (FSM for multiplication and division):



We first detect if the ready bit is 0 and the LSB of ALU control output is 1, if both of the conditions above satisfy, then our CPU will do multi-cycle operation, which depends on the mulDiv_mode. If mulDiv_mode is 0, our CPU will do multiplication. If mulDiv_mode is 1, out CPU will do division. The remaining steps are same as that in HW3.

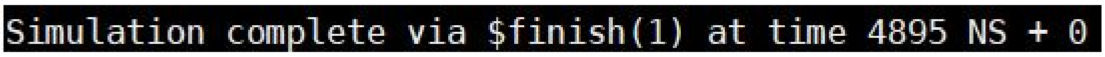
Record total simulation time (CYCLE = 10 ns)

- Leaf: a = 1, b = 9, c = 2, d = 2



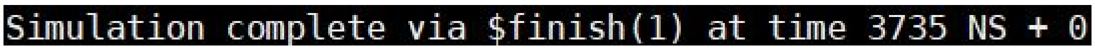
```
Simulation complete via $finish(1) at time 255 NS + 0
```

- Fact: n = 10



```
Simulation complete via $finish(1) at time 4895 NS + 0
```

- (Bonus) HW1: n = 10



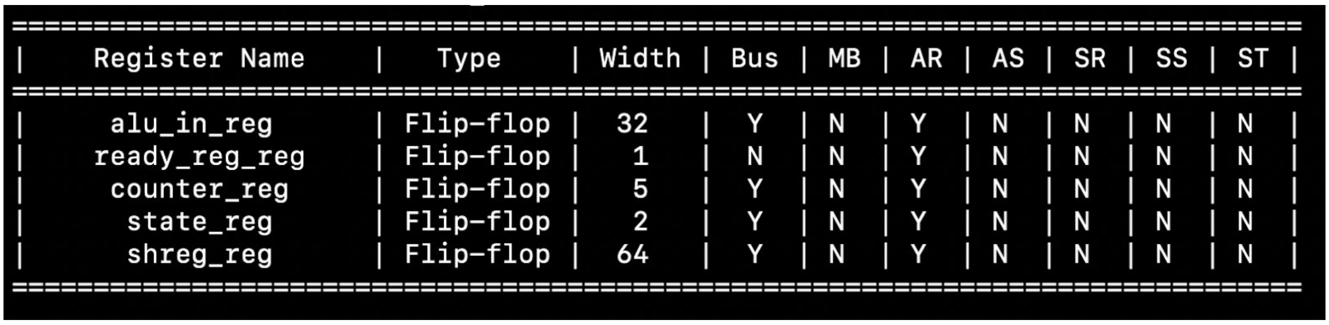
```
Simulation complete via $finish(1) at time 3735 NS + 0
```

Describe your observation

At first we did not design our control classification properly. Therefore, we have to additionally add a few MUXs to implement the functions, which increases the datapath.

Also, for bonus, we have to adjust the data size and the stack size so that the program can execute seamlessly.

Snapshot the “Register table” in Design Compiler (p. 22)



```
==================================================================================
|    Register Name     |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
==================================================================================
|     alu_in_reg       | Flip-flop |  32   |  Y  | N  | Y  | N  | N  | N  | N  |
|    ready_reg_reg     | Flip-flop |   1   |  N  | N  | Y  | N  | N  | N  | N  |
|     counter_reg      | Flip-flop |   5   |  Y  | N  | Y  | N  | N  | N  | N  |
|      state_reg       | Flip-flop |   2   |  Y  | N  | Y  | N  | N  | N  | N  |
|      shreg_reg       | Flip-flop |  64   |  Y  | N  | Y  | N  | N  | N  | N  |
==================================================================================
```

List a work distribution table

	徐瑞擇	阮明皓	潘彥銘
工作分配	想架構、接上大家的module、寫report	接上MUL、 做bonus	寫code、寫report