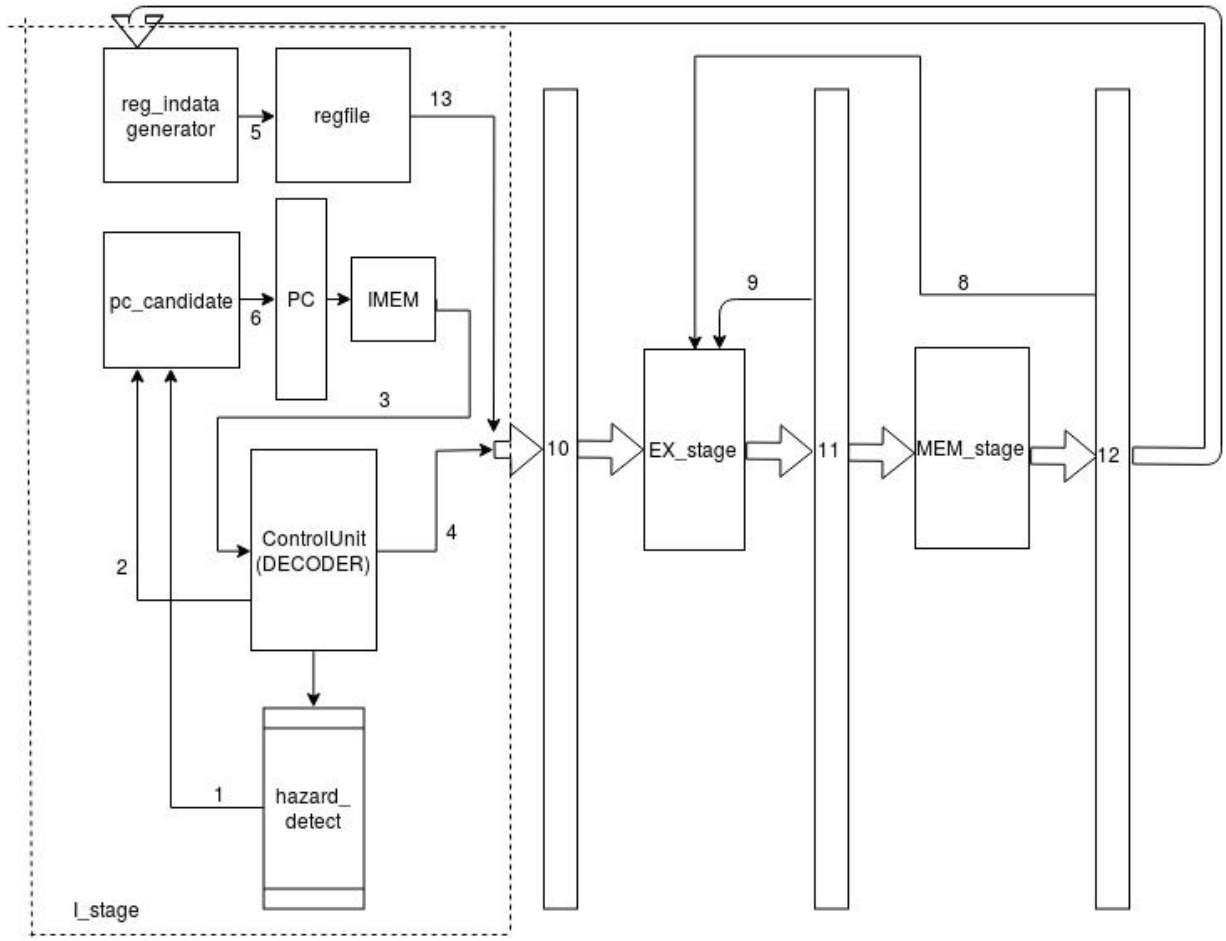
COMPUTER ORGANISATION
ASSIGNMENT 4 REPORT

SUBMITTED BY **EE17B114**(Pruthvi Raj R G) , **EE17B069**(Sundar Raman)

13th Nov 2019

The **modular view** of our CPU is as follows. The various control signals involved are mentioned below.



SIGNALS INVOLVED:

1: hazard_type

2: imm,jump,jalr,auipc

3: Instruction

4: RegDst, RegWrite, ALUSrc, MemRead, MemWrite,MemToReg, Branch, Jump,imm,auipc,rs1,rs2,load_type,ALUop,we,jalr,hazard_type,current_instruction_type

5: indata_parsed

6: pc_candidate

8: dmem_out_MEM_WB_reg_wire(Used in register forwarding of LW-ALU operations)

9: daddr_EX_MEM_reg_wire(Used in register forwarding of ALU-ALU operations)

10:
MemWrite_I_EX_reg
RegWrite_I_EX_reg
ALUSrc_I_EX_reg
MemToReg_I_EX_reg
Jump_I_EX_reg
auipc_I_EX_reg
jalr_I_EX_reg
we_I_EX_reg
load_type_I_EX_reg
hazard_type_I_EX_reg
ALUop_I_EX_reg
RegDst_I_EX_reg
imm_I_EX_reg
rv1_I_EX_reg
rv2_I_EX_reg
pc_I_EX_reg

11:
auipc_EX_MEM_reg
jalr_EX_MEM_reg
MemToReg_EX_MEM_reg
MemWrite_EX_MEM_reg
Jump_EX_MEM_reg
we_EX_MEM_reg
load_type_EX_MEM_reg
imm_EX_MEM_reg
daddr_EX_MEM_reg
dwdata_EX_MEM_reg
pc_EX_MEM_reg
RegDst_EX_MEM_reg

RegWrite_EX_MEM_reg
12: auipc_MEM_WB_reg
MemToReg_MEM_WB_reg
RegWrite_MEM_WB_reg
Jump_MEM_WB_reg
jalr_MEM_WB_reg
load_type_MEM_WB_reg
imm_MEM_WB_reg
dmem_out_MEM_WB_reg
daddr_MEM_WB_reg
pc_MEM_WB_reg
RegDst_MEM_WB_reg

13: rv1,rv2

- The module names mentioned in the diagram correspond to the same blocks of codes as their names.

CHOICE OF PIPELINE:
4 Stage pipeline including the I_stage, EX_stage, MEM_stage and the WB_stage. The I_stage includes the instruction fetch and instruction decode stages.

TYPES OF HAZARDS HANDLED CURRENTLY:

1. **ALU-ALU(data_hazard)** : This hazard is caused when the rs1 or rs2 of the second ALU operation depends on the rd of the previous ALU instruction. To avoid this we have used register forwarding by using the daddr(alu output) directly as the input to the ALU in the next clock cycle without waiting for the value to be written into register.

2. **BRANCH(control_hazard):** This hazard is caused because the **alu_zero** signal which decides whether to take the branch or not, is only **available after the EX_stage**,but by that time the next instruction would have already entered the I-stage. We use the value of alu_zero generated by the EX_stage(**combinational output, not from the pipeline register**) to assign the new value to the pc(via the pc_candidate generation module) if the branch is supposed to be taken.We also use **alu_zero to nullify the MemWrite and RegWrite signals** before writing them into the pipeline register. This ensures that the other instruction which was already loaded will not harm the state of the system.

3. **LW/LH/LB-ALU(data_hazard)**: This hazard is caused when the rs1 or rs2 of the ALU operation depends on the rd of the load instruction. To avoid this we have used register forwarding by using the **drdata**(dmem output) directly as the input to the ALU in the next clock cycle without waiting for the value to be written into register. For the **stalling** operation that is required while the LW operation moves into the MEM_stage, we make use of **hazard_type signal** generated by the hazard_detect module. We can nullify the write enables of the instruction which is passed into the ALU_stage(which were about to be written into the pipeline registers) and we also **retain the value of pc.**

WORK DISTRIBUTION:

EE17B114 : Ideation,Implementation of basic pipeline registers, wiring and ALU-ALU hazards implementation,testing and debugging.

EE17B069: Ideation, Implementation of LW-ALU and Branch hazards,testing and debugging.