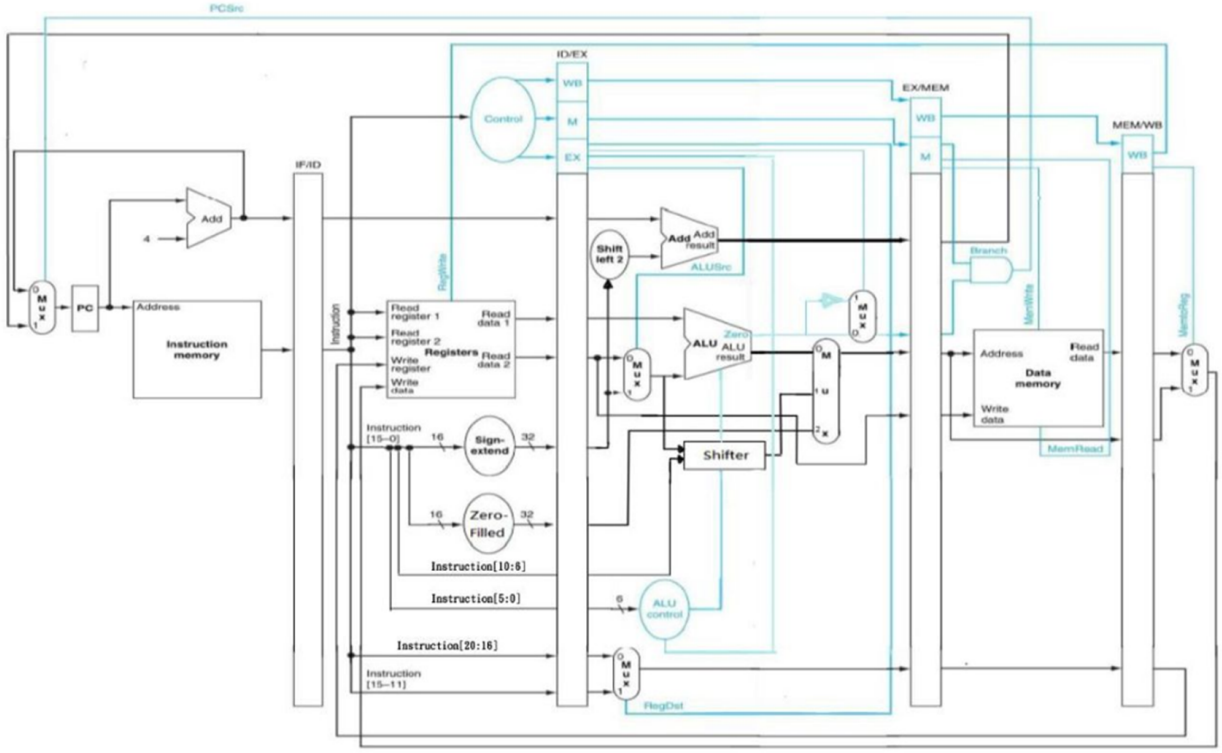
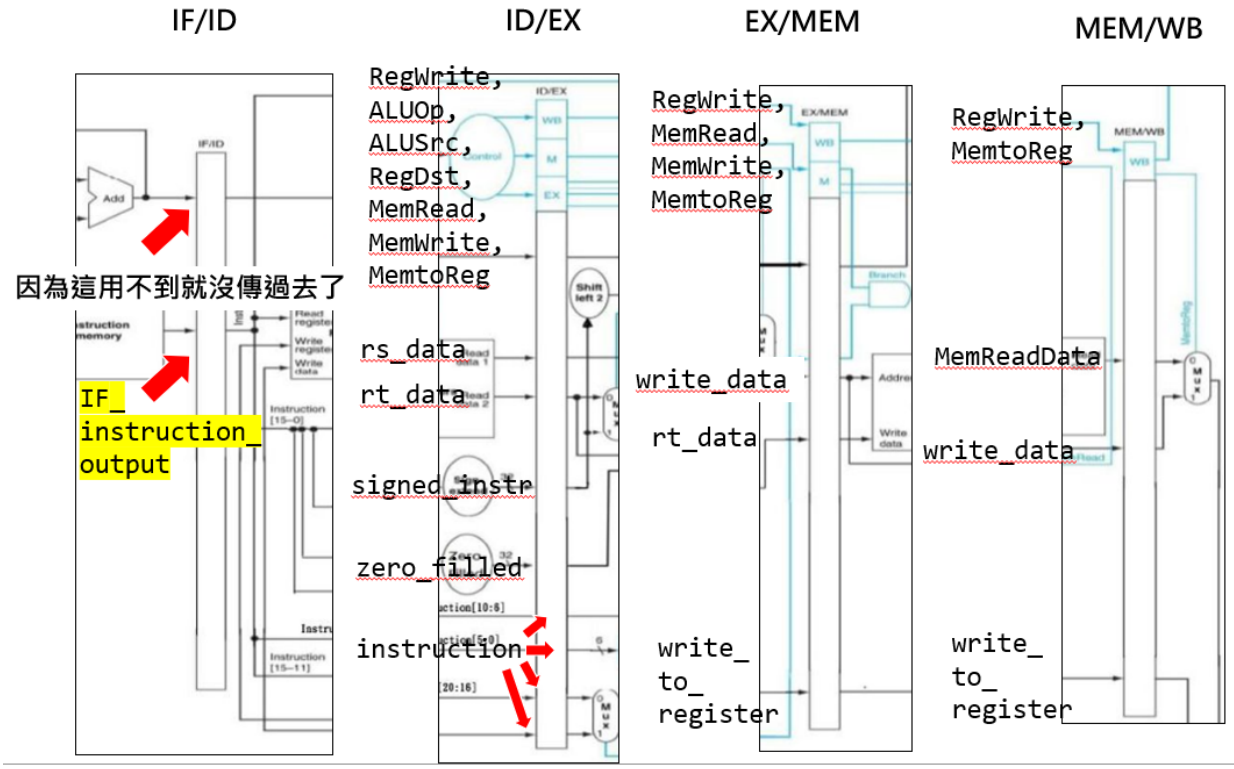
Q1: Describe the input fields of each pipeline register

Overall Pipelined CPU Architecture Diagram



Each Pipeline register with their input



From the provided diagram of the pipeline CPU, we have pipeline registers between each of the main stages: IF_ID, ID_EX, EX_MEM, and MEM_WB. Let's identify the input fields for each of these pipeline registers:

1. Pipeline_IF_ID:

- Input Fields:
  - IF_instruction_output: This is the instruction output from the instruction memory (IF_instruction_output). It contains the instruction that will be decoded in the next ID (Instruction Decode) stage.

2. Pipeline_ID_EX:

- Input Fields:
  - Control fields:
    - ID_RegWrite_signal: Indicates if a register should be written to.
    - ID_ALUOp_signal: The operation for the ALU.
    - ID_ALUSrc_signal: Specifies the ALU source.
    - ID_RegDst_signal: Specifies the destination register.
    - ID_MemRead_signal: Indicates if memory should be read.
    - ID_MemWrite_signal: Indicates if memory should be written to.
    - ID_MemtoReg_signal: Specifies if the data should come from memory or the ALU result.
  - ID_rs_data: Data corresponding to the rs field of the instruction.
  - ID_rt_data: Data corresponding to the rt field of the instruction.
  - ID_signed_instr: The sign-extended 32-bit immediate field from the instruction.
  - ID_zero_filled_instr: The zero-filled 32-bit immediate field from the instruction.
  - ID_instruction_input: The entire instruction itself.

3. Pipeline_EX_MEM:

- Input Fields:
  - Control fields:
    - EX_RegWrite_signal: Indicates if a register should be written to.
    - EX_MemRead_signal: Indicates if memory should be read.
    - EX_MemWrite_signal: Indicates if memory should be written to.
    - EX_MemtoReg_signal: Specifies if the data should come from memory or the ALU result.
  - EX_write_data: The result from the ALU or shifter or zero-filled instruction, depending on the operation.
  - EX_rt_data: Data corresponding to the rt field of the instruction, forwarded to the next stage.

- EX_write_to_register: Specifies which register should be written to.

4. Pipeline_MEM_WB:

- Input Fields:
  - Control fields:
    - MEM_RegWrite_signal: Indicates if a register should be written to.
    - MEM_MemtoReg_signal: Specifies if the data should come from memory or the result of the previous stage.
  - MEM_write_data: The result data that might be written back to a register.
  - Mem_read_data: The data read from the data memory.
  - MEM_write_to_register: Specifies which register should be written to in the Write-Back (WB) stage.

Q2: Explain your control signals in the sixth cycle(both test data test_1.txt and test_2.txt are needed)

Test1.txt

In the sixth cycle

```
010011|00000|00001|0000000000011111 = addi $1, $0, 31
010011|00000|00010|0000000000010001 = addi $2, $0, 17  ---> WB
010011|00000|00100|0000000000001110 = addi $4, $0, 14 ---> MEM
000000|00101|00110|00011|00000|011111 = and $3, $5, $6 ---> EX
000000|00001|00000|00101|00000|101111 = or $5, $1, $0 ---> ID
000000|00010|00001|00110|00000|010100 = slt $6, $2, $1 ---> IF
```

EX(and)			**MEM(addi)**			**WB(addi)**	
ALUOP	ALUSrc	Regdst	MemRead	MemWrt	branch	MemtoReg	RegWrt
010	0	1	0	0	0	0	1

Test2.txt

In the sixth cycle

```
010011|00000|00010|0000000000000111 = addi $2, $0, 7
010011|00000|00011|0000000000001000 = addi $3, $0, 8 ---> WB
010011|00000|00100|0000000000001001 = addi $4, $0, 9 ---> MEM
010011|00000|00101|0000000000001010 = addi $5, $0, 10 ---> EX
101000|00000|00010|0000000000000100 = sw $2, 4($0) ---> ID
101000|00011|00011|0000000000001000 = sw $3, 8($3) ---> IF
```

EX(addi)			**MEM(addi)**			**WB(addi)**	
ALUOP	ALUSrc	Regdst	MemRead	MemWrt	branch	MemtoReg	RegWrt
011	1	0	0	0	0	0	1

Ref ALUOP code

```
assign ALUOp_o =  (instr_op_i == RType) ? 3'b010 :
                  (instr_op_i == addi) ? 3'b011 :
                  (instr_op_i == lw) || (instr_op_i == sw) ? 3'b000
                  (instr_op_i == beq) ? 3'b001 :
                  (instr_op_i == bne) || (instr_op_i == bnez) ? 3'b110 :
                  (instr_op_i == blt) ? 3'b100 :
                  (instr_op_i == bgez) ? 3'b101 : 3'b000;
```