1. In the following figure, explain the purpose of each numbered items.

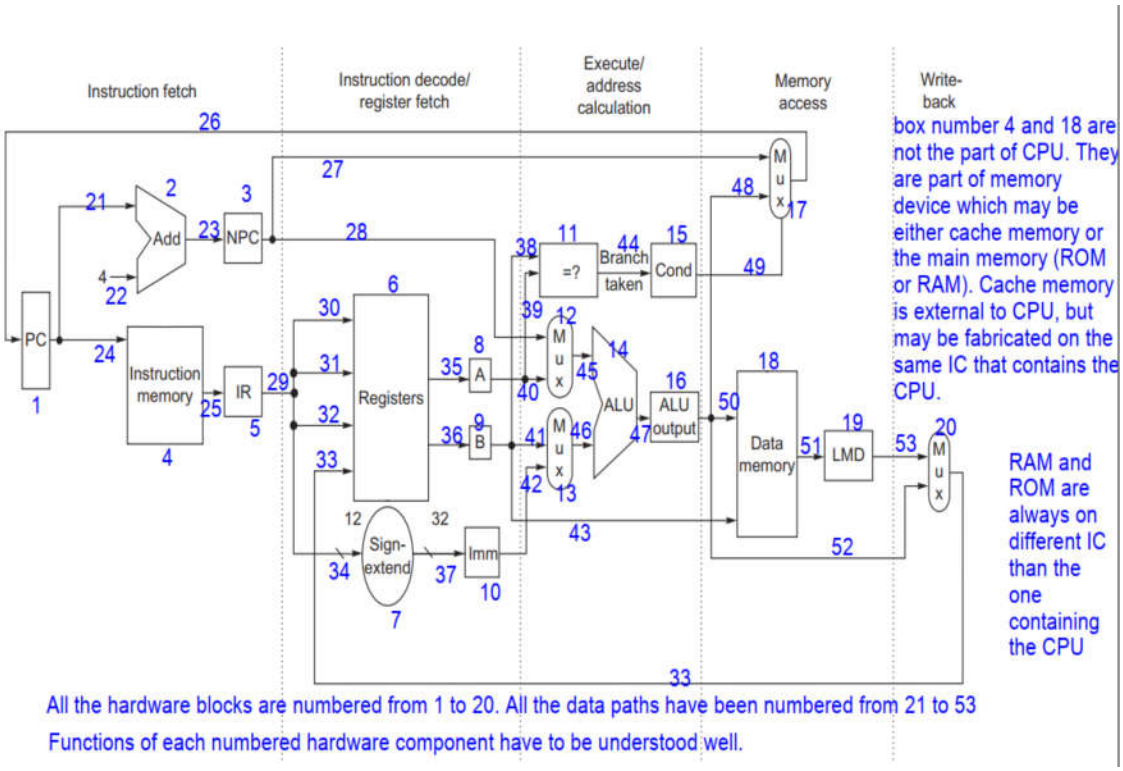

Figure C.18 The implementation of the RISC V data path allows every instruction to be executed in 4 or 5 clock cycles. Although the PC is shown in the portion of the data path that is used in instruction fetch and the registers are shown in the portion of the data path that is used in instruction decode/register fetch, both of these functional units are read as well as written by an instruction. Although we show these functional units in the cycle corresponding to where they are read, the PC is written during the memory access clock cycle and the registers are written during the write-back clock cycle. In both cases, the writes in later pipe stages are indicated by the multiplexer output (in memory access or write-back), which carries a value back to the PC or registers. These backward-flowing signals introduce much of the complexity of pipelining, because they indicate the possibility of hazards.

2. Explain the following micro-operations of the pipelined stages of RISC V processor.

Stage	Any instruction		
IF	IF/ID.IR ← Mem[PC] IF/ID.NPC,PC ← (if ((EX/MEM.opcode == branch) & EX/MEM.cond){EX/MEM.ALUOutput} else {PC+4});		
ID	ID/EX.A ← Regs[IF/ID.IR[rs1]]; ID/EX.B ← Regs[IF/ID.IR[rs2]]; ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← sign-extend(IF/ID.IR[immediate field]);		
	ALU instruction	**Load instruction**	**Branch instruction**
EX	EX/MEM.IR ← ID/EX.IR; EX/MEM.ALUOutput ← ID/EX.A func ID/EX.B; or EX/MEM.ALUOutput ← ID/EX.A *op* ID/EX.Imm;	EX/MEM.IR to ID/EX.IR EX/MEM.ALUOutput ← ID/EX.A+ID/EX.Imm; EX/MEM.B ← ID/EX.B;	EX/MEM.ALUOutput ← ID/EX.NPC + (ID/EX.Imm<<2); EX/MEM.cond ← (ID/EX.A == ID/EX.B);
MEM	MEM/WB.IR ← EX/MEM.IR; MEM/WB.ALUOutput ← EX/MEM.ALUOutput;	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD ← Mem[EX/MEM.ALUOutput]; or Mem[EX/MEM.ALUOutput] ← EX/MEM.B;	
WB	Regs[MEM/WB.IR[rd]] ← MEM/WB.ALUOutput;	For load only: Regs[MEM/WB.IR[rd]] ← MEM/WB.LMD;	

Figure C.20 Events on every pipe stage of the RISC V pipeline. Let's review the actions in the stages that are specific to the pipeline organization. In IF, in addition to fetching the instruction and computing the new PC, we store the incremented PC both into the PC and into a pipeline register (NPC) for later use in computing the branch-target address. This structure is the same as the organization in Figure C.19, where the PC is updated in IF from one of two sources. In ID, we fetch the registers, extend the sign of the 12 bits of the IR (the immediate field), and pass along the IR and NPC. During EX, we perform an ALU operation or an address calculation; we pass along the IR and the B register (if the instruction is a store). We also set the value of cond to 1 if the instruction is a taken branch. During the MEM phase, we cycle the memory, write

the PC if needed, and pass along values needed in the final pipe stage. Finally, during WB, we update the register field from either the ALU output or the loaded value. For simplicity we always pass the entire IR from one stage to the next, although as an instruction proceeds down the pipeline, less and less of the IR is needed.