|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| NOT | 1 | 1 | 0 | 0 | 0 |  |
| INC | 1 | 1 | 0 | 0 | 1 |  |
| DEC | 1 | 1 | 0 | 1 | 0 |  |
| MOV | 1 | 1 | 0 | 1 | 1 |  |
| ADD | 1 | 1 | 1 | 0 | 0 |  |
| IADD | 1 | 1 | 1 | 0 | 0 | 1 |
| SUB | 1 | 1 | 1 | 0 | 1 |  |
| AND | 1 | 1 | 1 | 1 | 0 |  |
| OR | 1 | 1 | 1 | 1 | 1 |  |
| PUSH | 0 | 1 | 0 | 0 | 0 |  |
| POP | 0 | 1 | 0 | 0 | 1 |  |
| LDM | 0 | 1 | 0 | 1 | 0 | 1 |
| LDD | 0 | 1 | 0 | 1 | 1 |  |
| STD | 0 | 1 | 1 | 0 | 0 |  |
| JZ | 1 | 0 | 0 | 0 | 0 |  |
| JC | 1 | 0 | 0 | 0 | 1 |  |
| JMP | 1 | 0 | 0 | 1 | 0 |  |
| CALL | 1 | 0 | 0 | 1 | 1 |  |
| RET | 1 | 0 | 1 | 0 | 0 |  |
| RTI | 1 | 0 | 1 | 0 | 1 |  |
| NOP | 0 | 0 | 0 | 0 | 0 |  |
| SETC | 0 | 0 | 0 | 0 | 1 |  |
| CLRC | 0 | 0 | 0 | 1 | 0 |  |
| IN | 0 | 0 | 0 | 1 | 1 |  |
| OUT | 0 | 0 | 1 | 0 | 0 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| OPCODE | | | | | Immediate bit | R destination | | | R source 1 | | | R source 2 | | | unused |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **RegWrite** | **PCSrc** | **MemRead (used by memory)** | **MemWrite**  **(used by memory)** | **MemToReg**  **(used by WB)** | **IN**  **(used by WB)** | **OUT**  **(used by WB)** | **SP+**  **(used by memory)** | **SP-**  **(used by memory)** |
| NOT | 1 |  |  |  |  |  |  |  |  |
| INC |  |  |  |  |  |  |  |  |
| DEC |  |  |  |  |  |  |  |  |
| MOV |  |  |  |  |  |  |  |  |
| ADD |  |  |  |  |  |  |  |  |
| IADD |  |  |  |  |  |  |  |  |
| SUB |  |  |  |  |  |  |  |  |
| AND |  |  |  |  |  |  |  |  |
| OR |  |  |  |  |  |  |  |  |
| PUSH |  |  |  | 1 |  |  |  |  | 1 |
| POP | 1 |  | 1 |  | 1 |  |  | 1 |  |
| LDM | 1 |  |  |  |  |  |  |  |  |
| LDD | 1 |  | 1 |  | 1 |  |  |  |  |
| STD |  |  |  | 1 |  |  |  |  |  |
| JZ |  | 1 |  |  |  |  |  |  |  |
| JC |  |  |  |  |  |  |  |  |
| JMP |  |  |  |  |  |  |  |  |
| CALL |  |  | 1 |  |  |  |  | 1 |
| RET |  | 1 |  |  |  |  | 1 |  |
| RTI |  | 1 |  |  |  |  | 1 |  |
| NOP |  |  |  |  |  |  |  |  |  |
| SETC |  |  |  |  |  |  |  |  |  |
| CLRC |  |  |  |  |  |  |  |  |  |
| IN | 1 |  |  |  |  | 1 |  |  |  |
| OUT |  |  |  |  |  |  | 1 |  |  |

|  |  |  |
| --- | --- | --- |
| **Register** | **Inputs** | **Outputs** |
| IF/ID | * Instruction * Immediate * PC | * Instruction * Immediate * PC * RS1 data * RS2 data |
| ID/EX | * RS1 data * RS2 data * Instruction as a whole [will split inside] * Immediate * All control signals * PC | * RS1 data * RS2 data * Opcode * Immediate * isImmediate * Rd * Rs1 * Rs2 * PC * All control signals |
| EX/MEM1 | * Alu output * Flag register [checked in execution stage for jumps but still needs to be propagated to be saved in the stack] * Rs2Data [for memory write] * Rs1Data [for out – used at write back] * Memory signals * PC | * Alu output * Rs2 * Memory signals * PC |
| MEM1/MEM2 | * Memory output (integrated within the memory stage) * MemToReg * IN * OUT | * Delayed memory output (delays memory output by one cycle) * MemToReg * IN * OUT |
| MEM2/WB | * Delayed memory output * MemToReg * IN * OUT | * Delayed memory output * MemToReg * IN * OUT |

|  |  |
| --- | --- |
| **Hazards** | **Details** |
| **Data Hazards** | **Forwarding unit at ALU:**  Checks for RegWrite signal at all subsequent buffers  Checks for Rd for all subsequent buffers  Checks for Rs1,Rs2 of current execution  And acts accordingly -> checks closer buffers first  Potential instructions to cause hazards [All R-type, POP, IN, LDD, LDM]  **Forwarding unit at Memory:**  Checks for RegWrite signal at MEM2/WB buffer  Checks for Rd at MEM2/WB buffer  Checks for Rs1, Rs2 of current STORE operation  And acts accordingly  Potential instructions to cause hazards [a load (**LDD**) instruction followed by a store (**STD**)] |
| **Hazard Detection unit at Decode:**  Checks for MemRead (only by LDD and POP) signal at ID/EX buffer  Checks for Rd at ID/EX buffer  Checks for Rs1, Rs2 of current fetched instruction (from IF/ID buffer)  And acts accordingly  Potential instructions to cause hazards [a load (**LDD**) instruction or a pop (**POP**)] |
| **Structural Hazards** | **Hazard Detection unit at Decode:**  If current MemRead or MemWrite at decode stage are ‘1’  Checks for MemRead or MemWrite at ID/EX buffer if equal ‘1’  Then stall once |
| **Control Hazards** | **JMP [decode]**  Change PC -> to RD  Flush IF/ID buffer (1 Stall)  **Call [decode]**  Change PC -> to RD  Flush IF/ID buffer  Propagate old PC till Mem stage to save it |
| **JMP conditional [execute] (JZ – JC)**  Change PC -> to RD  Flush IF/ID & ID/EX (2 stalls) |
| **RET/RTI [memory]**  Change PC -> to Datamem[SP]  Flush IF/ID & ID/EX & EX/Mem1 & Mem1/Mem2 (5 stalls) |

1. Control signals
2. Add Rs1,Rs2,Rd registers to ALL buffers
3. Add flags to output (propagate form EXEC)
4. .MEM file (later)
5. PC version 2008 (fix) (later)