**Instruction set architecture (RISC)**

**Instruction formats**

1. RRR type

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Opcode Reg A Reg B Reg C Unused

1. RI type

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Opcode Reg A Immediate

1. I type

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Opcode Immediate

**Architecture**

Data Memory (RAM) – 8M x 16 bits

Instruction Memory (ROM) – 4K x 24 bits

MAR – 24-bit Register

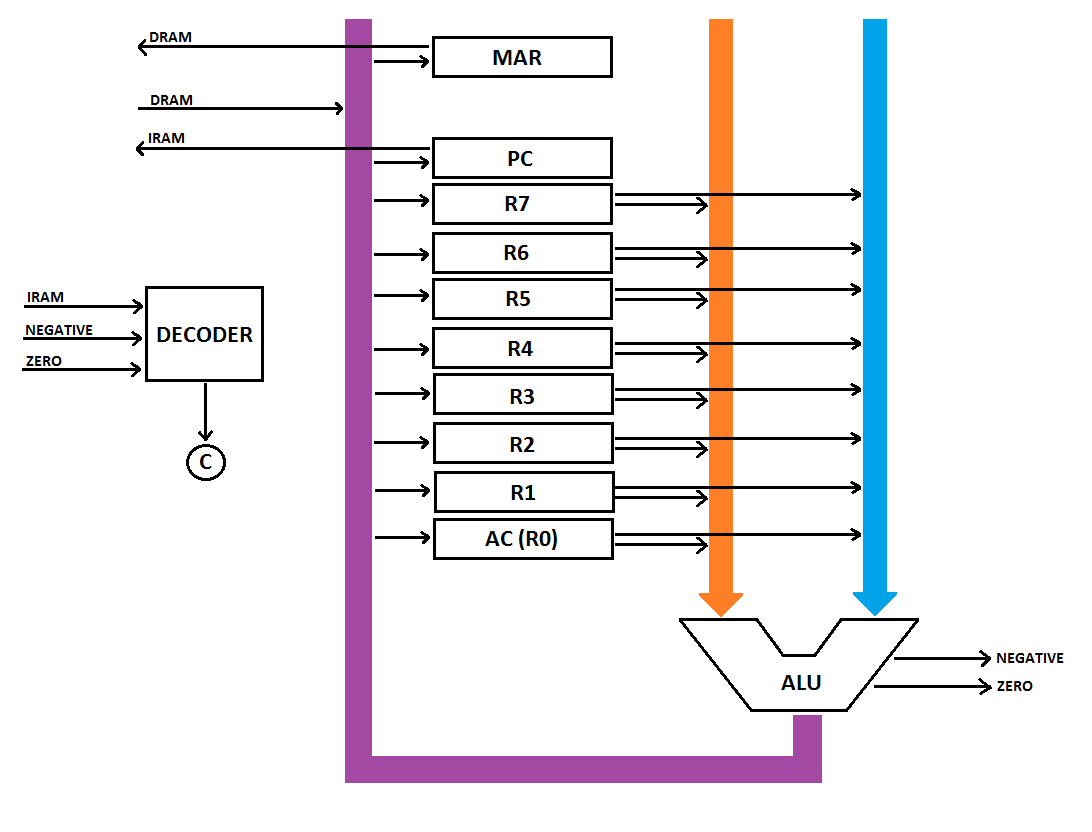
PC – 24-bit Register

Accumulator (AC, R0) – 24-bit Register

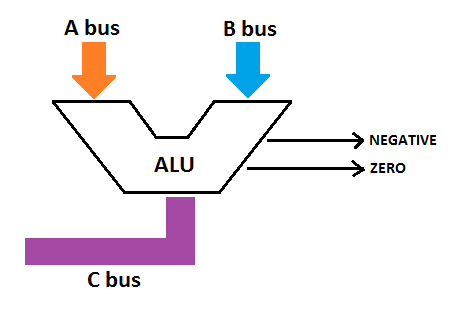
R1 – R7 General Purpose Registers – 24-bit

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **operation** |
| NOP | 00000 | No operation |
| LOAD | 00001 | Reg A ← RAM[MAR] |
| STORE | 00010 | RAM[MAR] ← Reg B |
| MOVE | 00011 | Reg A ← Reg B |
| LDMAR | 00100 | MAR ← Reg A |
| LDMARI | 00101 | MAR ← signed immediate (19-bit) |
| LOADI | 00110 | Reg A ← signed immediate (16-bit) |
| LDACI | 00111 | AC ← signed immediate (19-bit) |
|  |  |  |
| ADD | 01000 | Reg A ← Reg B + Reg C |
| SUB | 01001 | Reg A ← Reg B - Reg C |
| MUL | 01010 | Reg A ← Reg B << Reg C |
| DIV | 01011 | Reg A ← Reg B >> Reg C |
| INC | 01100 | Reg A ← Reg A + 1 |
| DEC | 01101 | Reg A ← Reg A - 1 |
| NEG | 01110 | Reg A ← -Reg B |
| NOT | 01111 | Reg A ← Reg B (NOT) Reg C |
| AND | 10000 | Reg A ← Reg B (AND) Reg C |
| OR | 10001 | Reg A ← Reg B (OR) Reg C |
| XOR | 10010 | Reg A ← Reg B (XOR) Reg C |
|  |  |  |
| JGT | 10011 | If ALU out > 0 then PC ← IMM19 else PC ← PC + 1 |
| JEQ | 10100 | If ALU out = 0 then PC ← IMM19 else PC ← PC + 1 |
| JGE | 10101 | If ALU out >= 0 then PC ← IMM19 else PC ← PC + 1 |
| JLT | 10110 | If ALU out < 0 then PC ← IMM19 else PC ← PC + 1 |
| JNE | 10111 | If ALU out != 0 then PC ← IMM19 else PC ← PC + 1 |
| JLE | 11000 | If ALU out <= 0 then PC ← IMM19 else PC ← PC + 1 |
| JMP | 11001 | PC ← Reg A (Unconditional Jump) |

**Data path**



**Arithmetic and Logic Unit (ALU)**

****

|  |  |  |
| --- | --- | --- |
| **Opcode** | **ALU control bits** | **Operation** |
| - | 0000 | No Operation |
| 01000 | 0001 | ADD (A + B) |
| 01001 | 0010 | SUB (A – B) |
| 01010 | 0011 | MUL (A << B) |
| 01011 | 0100 | DIV (A >> B) |
| 01100 | 0101 | INC (A + 1) |
| 01101 | 0110 | DEC (A – 1) |
| 01110 | 0111 | NEG (–A) |
| 01111 | 1000 | NOT (!A) |
| 10000 | 1001 | AND (A & B) |
| 10001 | 1010 | OR (A | B) |
| 10010 | 1011 | XOR (A ^ B) |

**If ALU output = 0, then Z = 1 else Z = 0**

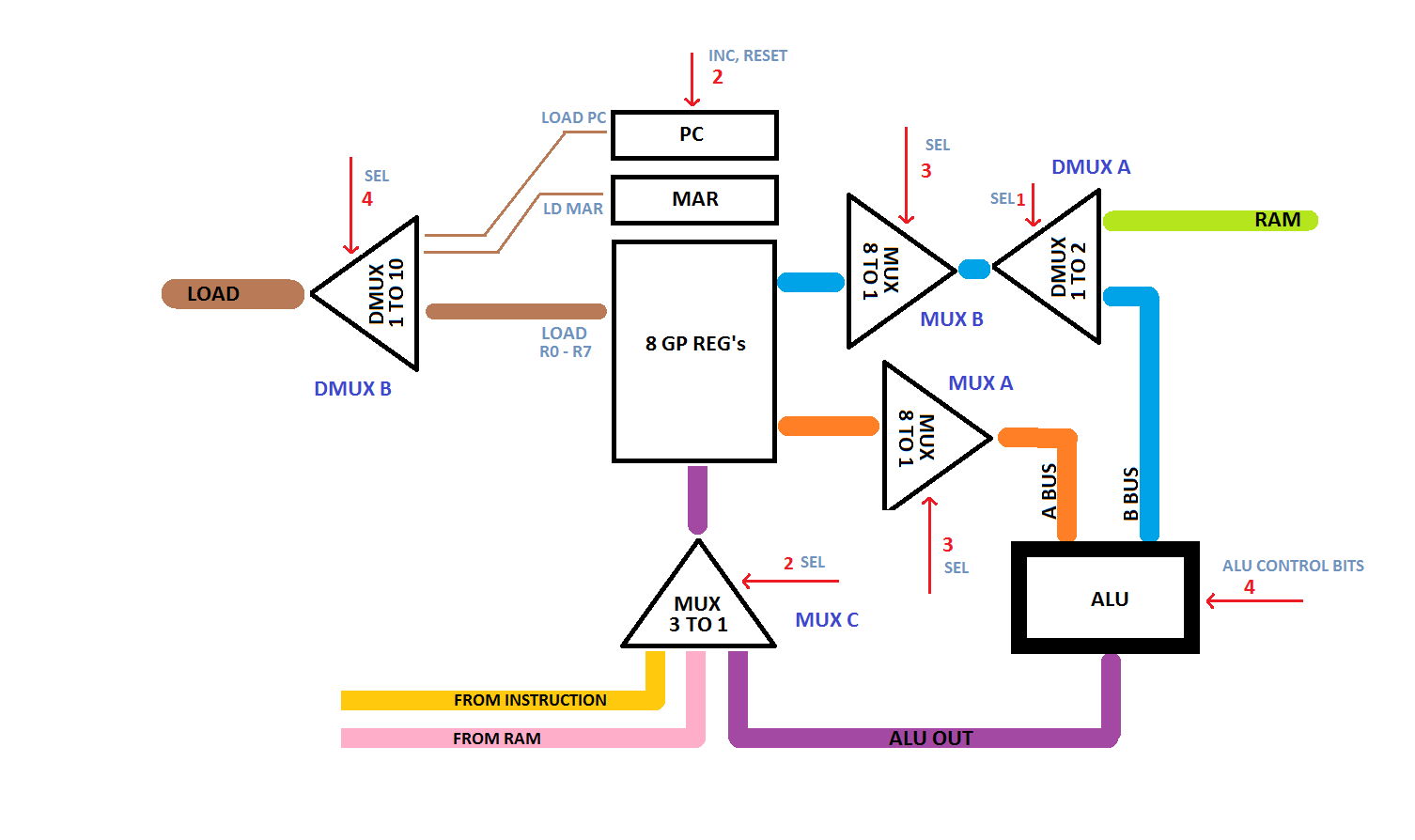
**If ALU output < 0, the N = 1 else N = 0**

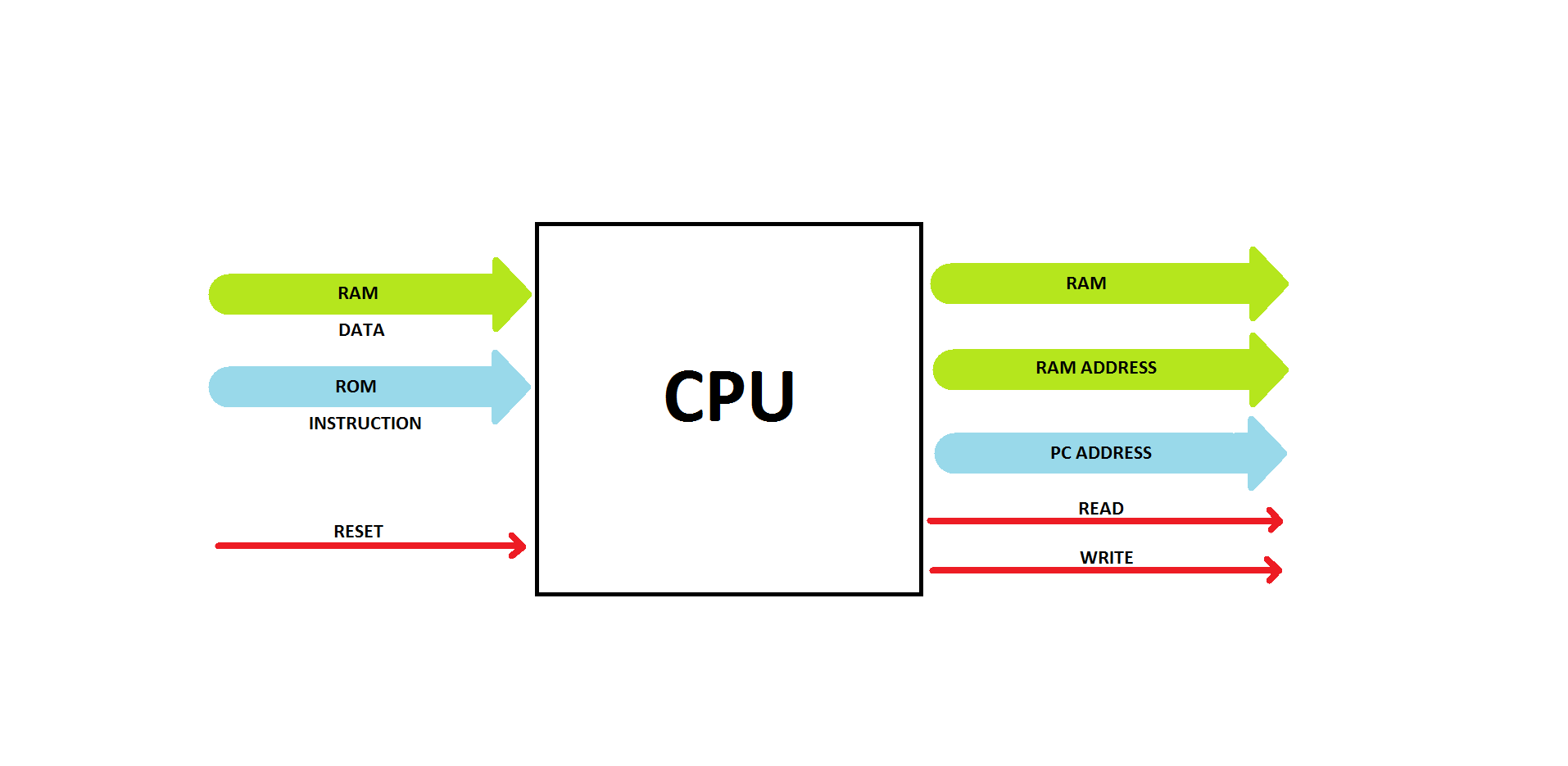
**Jump Logic**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Opcode** | **Operation** | **Z flag** | **N flag** | **Jump** |
| JGT | If ALU out > 0 | 0 | 0 | PC ← Reg A |
| JEQ | If ALU out = 0 | 1 | x | PC ← Reg A |
| JGE | If ALU out >= 0 | x | 0 | PC ← Reg A |
| JLT | If ALU out < 0 | 0 | 1 | PC ← Reg A |
| JNE | If ALU out != 0 | 0 | x | PC ← Reg A |
| JLE | If ALU out <= 0 | 1/0 | 1 | PC ← Reg A |
| JMP | Unconditional Jump | x | x | PC ← Reg A |

**Control bits**

Depending on the opcode and ALU flags the decoder will generate the control bits

****

****