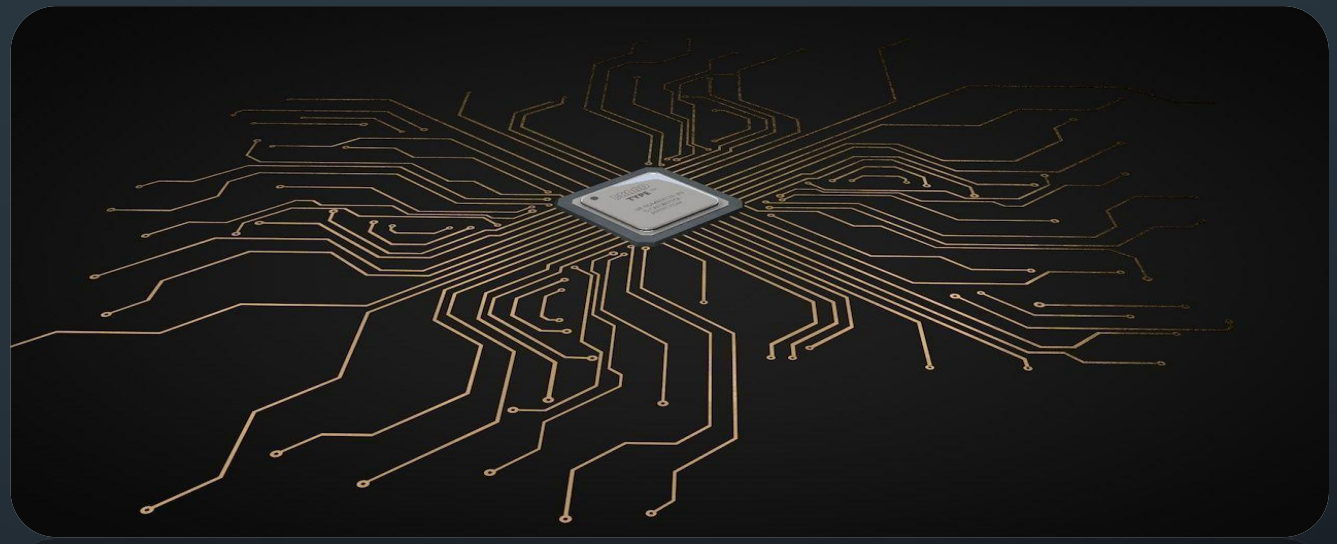
MIPS32 (MICROPROCESSOR WITHOUT INTERLOCKED PIPELINED STAGES) WITH GUI INTERFACE

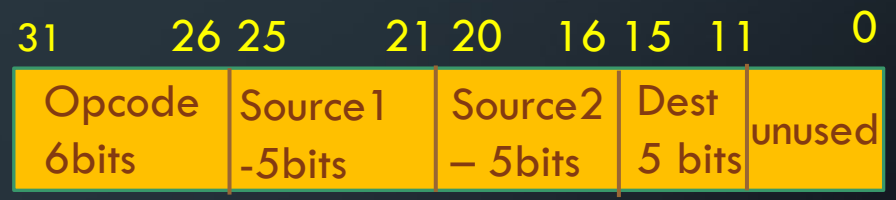


COURSE : HDL BASED SYSTEM DESIGN

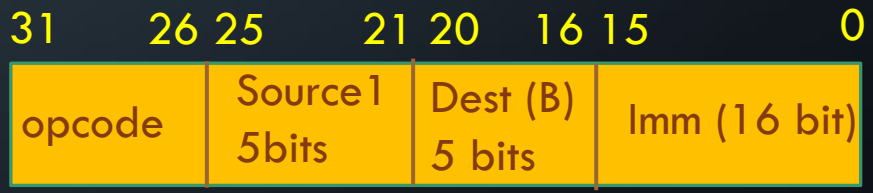
KEY POINTS ABOUT MIPS32 -

- RISC processor (Reduced instruction set Computer)
- Multicycle processor – Two non-overlapping clocks
- 5-stage pipelining
- 32-bit Processor
- Instruction format: IR Register

R-type instruction format



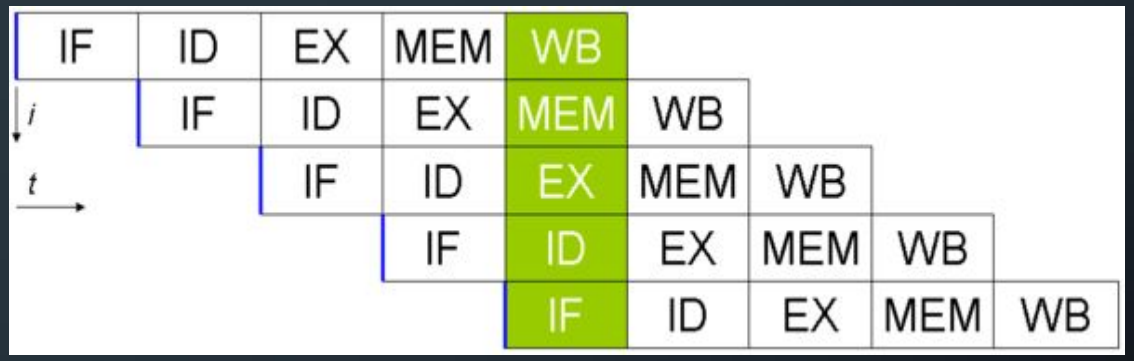
I-type Instruction Format



5- STAGE PIPELINED ARCHITECTURE

Five stages of instruction execution Cycle

1. Instruction fetch and PC increment
2. Cycle Reading sources from the register file
3. Cycle Performing an ALU computation
4. Cycle Reading or writing (data) memory
5. Cycle 5 Storing data back to the register file



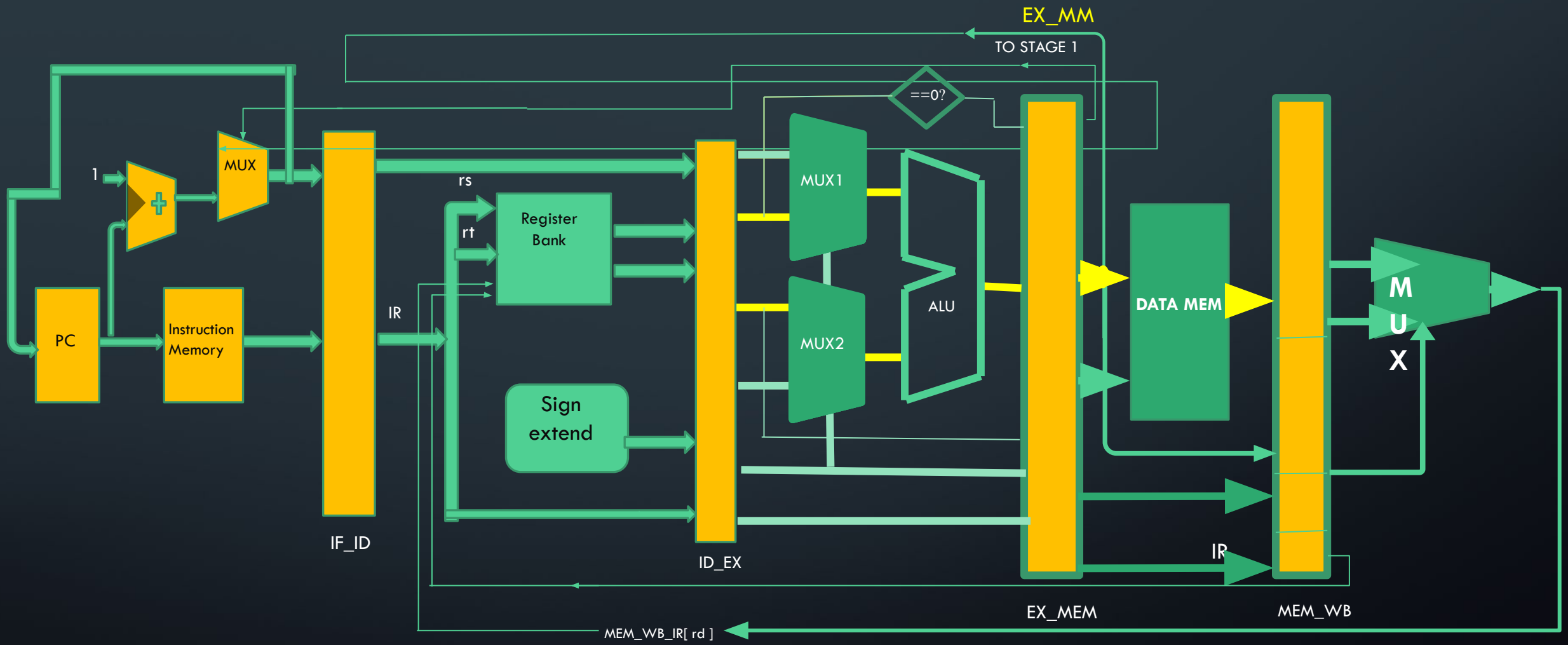
IF = Instruction Fetch
ID = Instruction Decode
EX = Execute
MEM = Memory access
WB = Register write back

NAMING CONVENTIONS -

- PC – Program counter

NPC – New Program Counter

COMPLETE 5-STAGE PIPELINED ARCHITECTURE



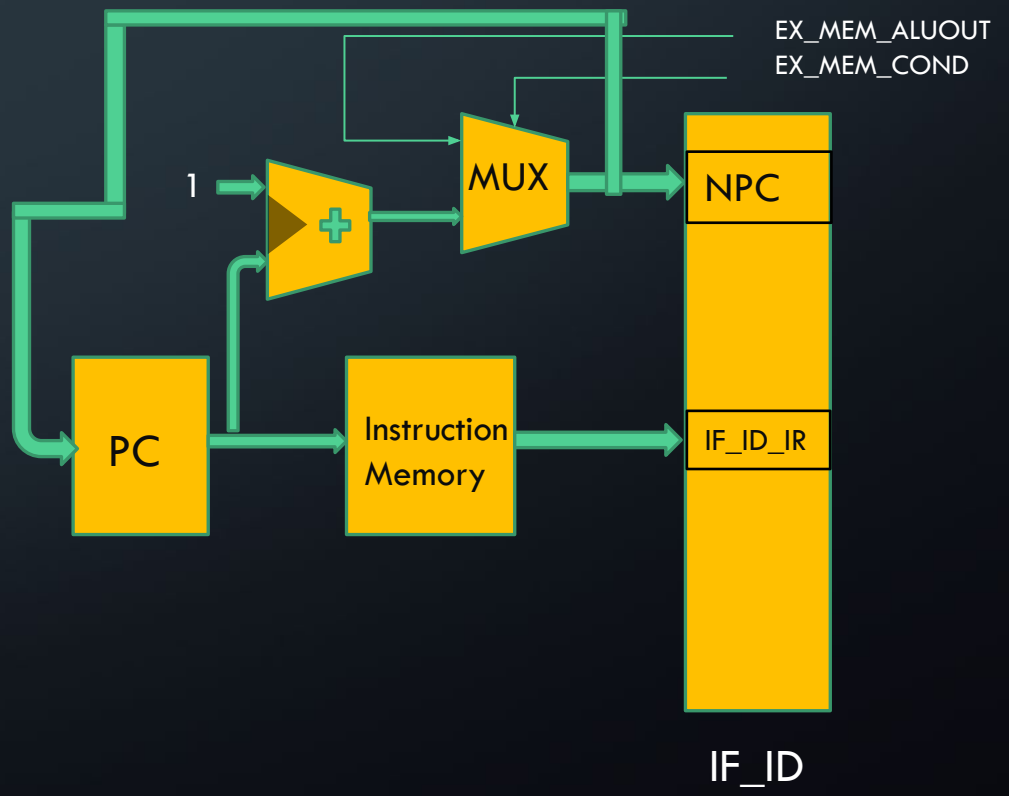
STAGE 1 - INSTRUCTION FETCH (IF)

MICRO-OPERATIONS:

IF_ID_IR ← Mem[PC]

PC, IF_ID_NPC ← if((EX_MEM_IR[opcode] == BRANCH) && EX_MEM_cond)) { EX_MEM_ALUOUT } // if branch then Effective address
else { PC + 1 }); // Next instruction

```
// STAGE 1  -->> IF  stage  - Instruction FETCH

always @(posedge clk1)
   if(PSW[1] == 0)
   begin
        if(( EX_MEM_IR[31:26] == BEQZ) && (EX_MEM_cond == 1)) ||
           ( EX_MEM_IR[31:26] == BNEQZ) && (EX_MEM_cond == 0)))
         begin
              IF_ID_IR        <= #2 Mem[EX_MEM_ALUOut];
              PSW[2]          <= #2 1'b1;
              IF_ID_NPC       <= #2 EX_MEM_ALUOut + 1;
              PC              <= #2 EX_MEM_ALUOut + 1;
         end
        else
         begin
              IF_ID_IR        <= #2 Mem[PC];
              IF_ID_NPC       <= #2 PC + 1;
              PC              <= #2 PC + 1;
         end
   end
```



STAGE 2 – INSTRUCTION DECODE

MICRO-OPERATIONS:

ID_EX_A ← Reg [IF_ID_IR[rs]]

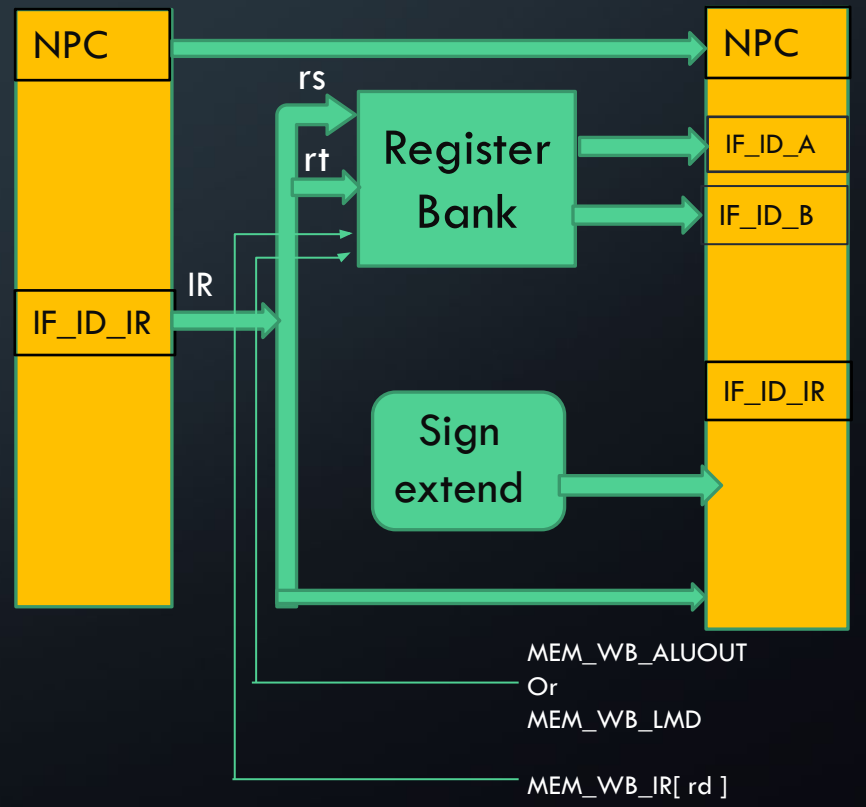
ID_EX_B ← Reg [IF_ID_IR[rt]]

ID_EX_NPC ← IF_ID_NPC

ID_EX_IR ← IF_ID_IR

ID_EX_Imm ← sign-extend($IF\_ID\_IR_{15..0}$)

```
// STAGE 2  -->> ID stage  - Instruction Decode

always @(posedge clk2)
 if(PSW[1] == 0)
    begin
      if( IF_ID_IR[25:21] == 5'b00000) ID_EX_A <= 0;          // "rs"
      else ID_EX_A  <= #2 Reg[IF_ID_IR[25:21]];

      if(IF_ID_IR[20:16] ==5'b00000) ID_EX_B <= 0;          //  "rt"
      else ID_EX_B  <= #2 Reg[IF_ID_IR[20:16]];

      ID_EX_NPC  <= #2 IF_ID_NPC;
      ID_EX_IR   <= #2 IF_ID_IR;
      ID_EX_Imm  <= #2 {{16{IF_ID_IR[15]}}, {IF_ID_IR[15:0]}};

    // Assigning type to the opcode which helps in later to optimize in
    // case statements

      case (IF_ID_IR[31:26])
         ADD, SUB, AND, OR, XOR, SLT, MUL, DIV:   ID_EX_type <= #2 RR_ALU;
         ADDI, SUBI, SLTI:                        ID_EX_type <= #2 RM_ALU;
         LW:                                      ID_EX_type <= #2 LOAD;
         SW:                                      ID_EX_type <= #2 STORE;
         BNEQZ, BEQZ:                             ID_EX_type <= #2 BRANCH;
         HLT:                                     ID_EX_type <= #2 HALT;
        default:                                  ID_EX_type <= #2 HALT;
  endcase
end
```



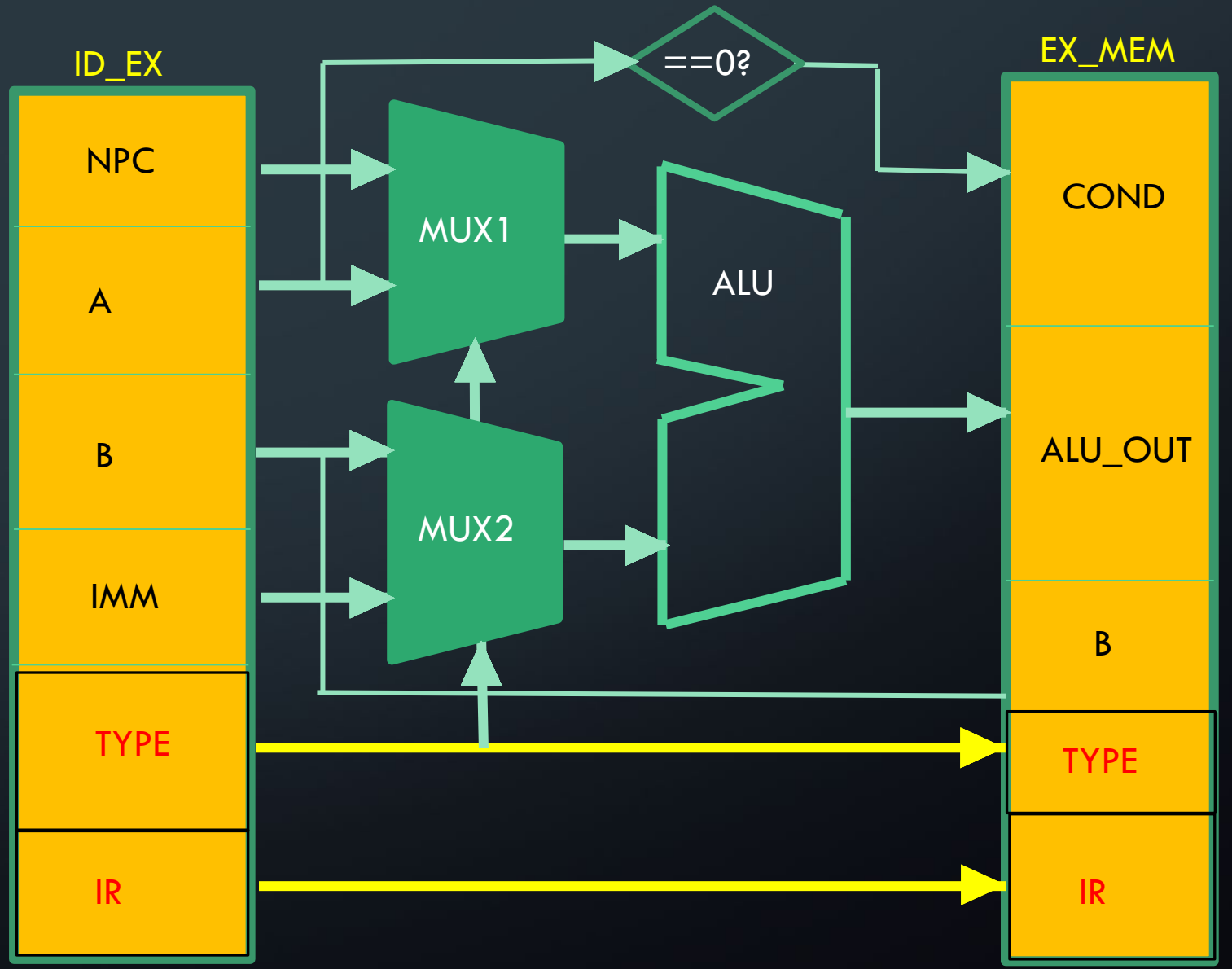
STAGE 3: EXECUTION

Initial Micro-operations

ID_EX_type => EX_MEM_type
ID_EX_type => EX_MEM_type

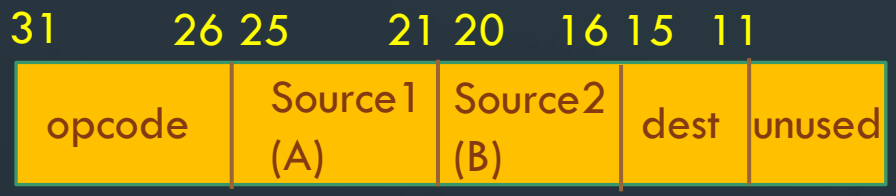
PSW[2] (taken branch flag) is reset

```
// STAGE 3  -->> EX stage  - Execution stage
   always @(posedge clk1)
      if(PSW[1] == 0)
        begin
        EX_MEM_type  <= #2 ID_EX_type;
        EX_MEM_IR    <= #2 ID_EX_IR;
        PSW[2]       <= #2 0;
        case (ID_EX_type)
        RR_ALU :
           begin
           case (ID_EX_IR[31:26])   // "opcode"
```

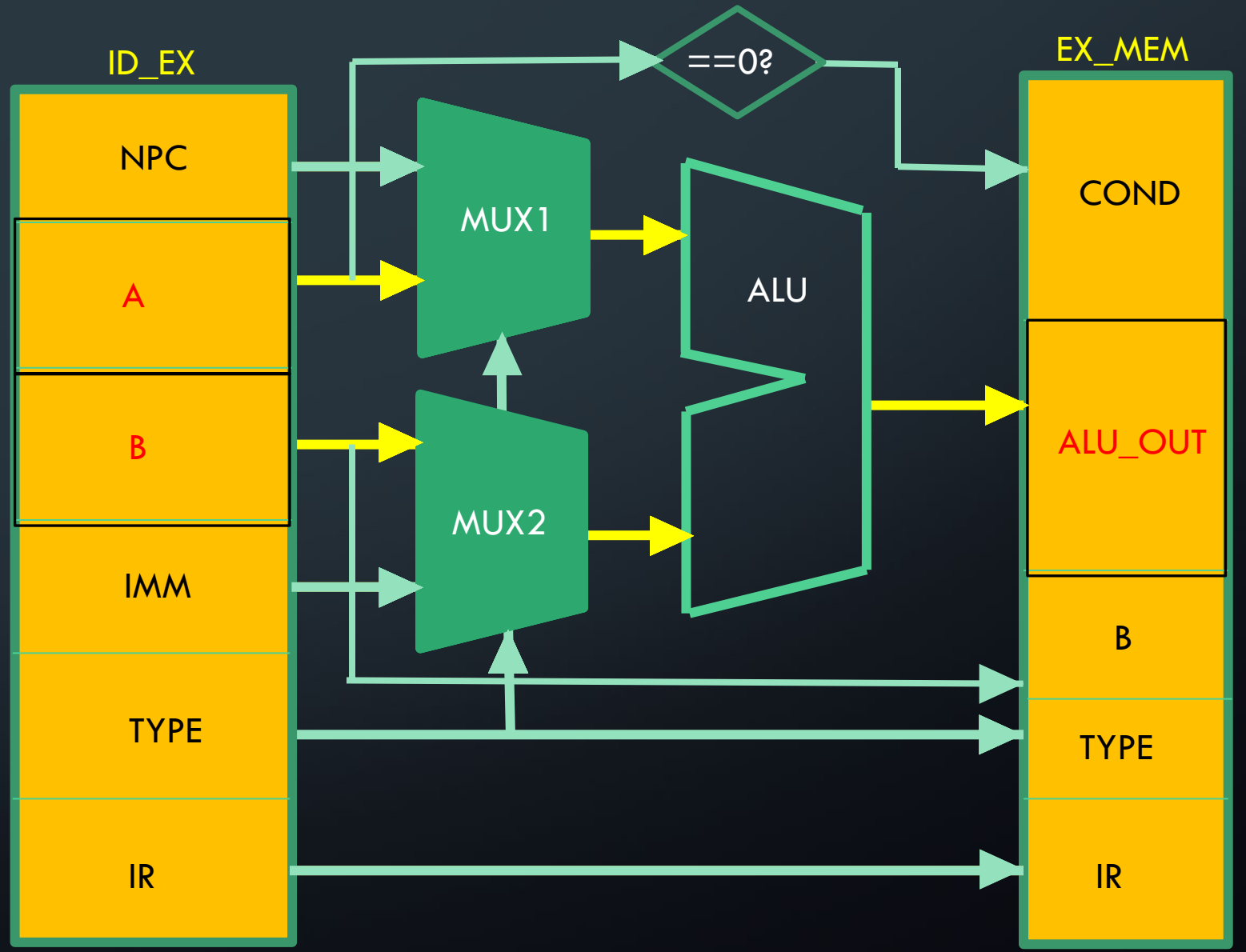


STAGE 3: EXECUTION

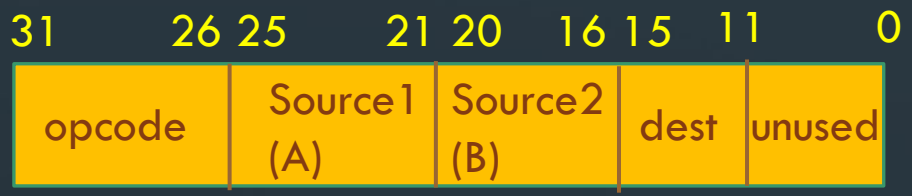
R type instruction format:



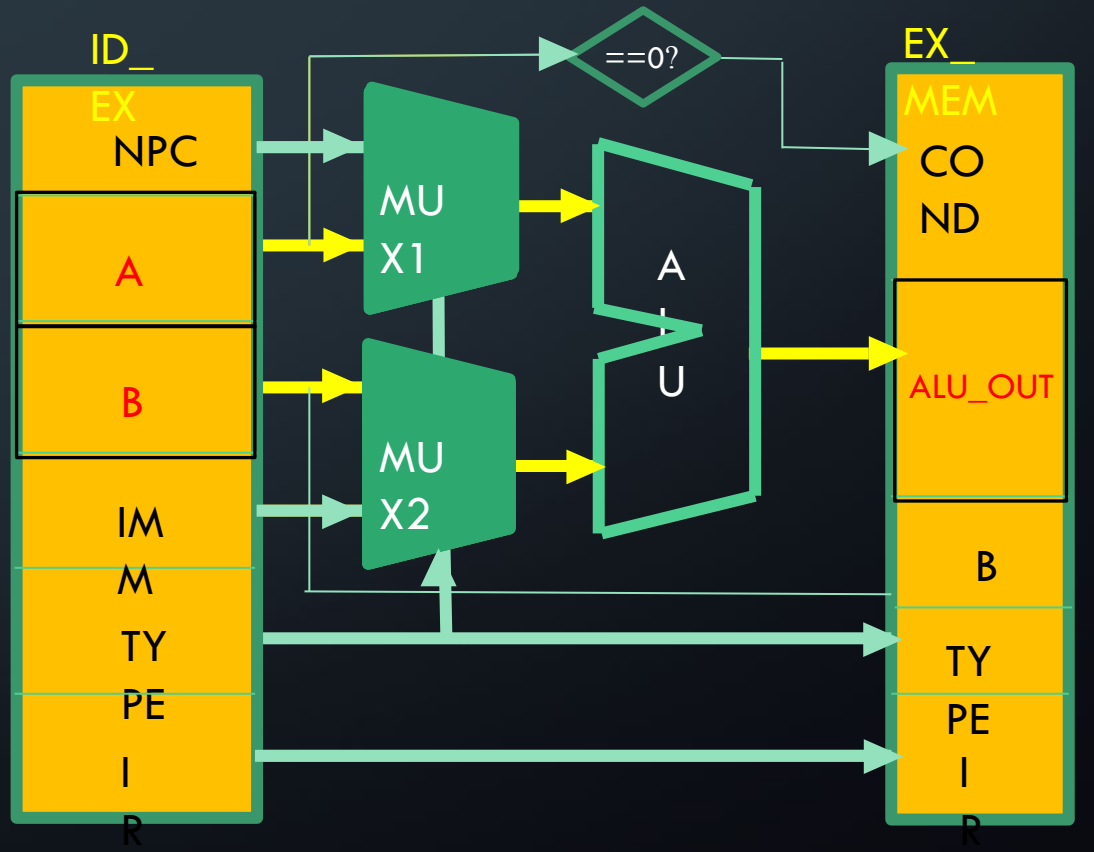
Instructions	opcode
ADD	000000
SUB	000001
AND	000010
OR	000011
SLT	000100
MUL	000101
HLT	111111



STAGE 3: EXECUTION

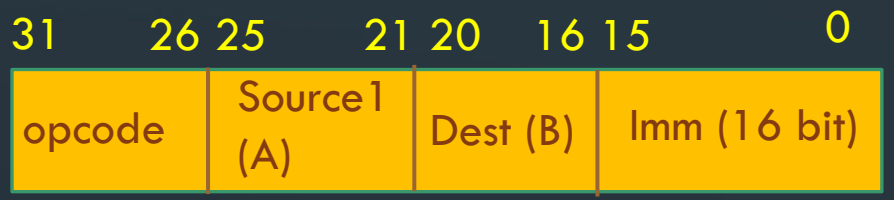



```
RR_ALU :
    begin
    case (ID_EX_IR[31:26])    // "opcode"
        ADD: {PSW[0],EX_MEM_ALUOut}     <= #2 ID_EX_A + ID_EX_B;
        SUB: {PSW[0],EX_MEM_ALUOut}     <= #2 ID_EX_A - ID_EX_B;
        AND:     EX_MEM_ALUOut          <= #2 ID_EX_A & ID_EX_B;
        OR:      EX_MEM_ALUOut          <= #2 ID_EX_A | ID_EX_B;
        XOR:     EX_MEM_ALUOut          <= #2 ID_EX_A ^ ID_EX_B;
        SLT:     EX_MEM_ALUOut          <= #2 ID_EX_A < ID_EX_B;
        MUL:     {PSW[0],EX_MEM_ALUOut} <= #2 ID_EX_A * ID_EX_B;
        DIV:
        begin
            if( ID_EX_B == 0)  //IF  DIVISOR IS PSW[4]
                begin
                EX_MEM_type   <= #2 HALT;           //CHANGE INSTRUTION TYPE TO HALT
                PSW[3]        <= #2 1;              // SET DIVIDE BY ZERO i.e PSW[4] FLAG TO 1
                EX_MEM_ALUOut <= #2 32'hxxxxxxxx;   //SET THE RESULT OF OPERATION AS UNKNOWN
                end
            else
                EX_MEM_ALUOut  <= #2 ID_EX_A / ID_EX_B;
        end
        default: EX_MEM_ALUOut <= #2 32'hxxxxxxxx;
     endcase
    if(EX_MEM_ALUOut==0) //checking for PSW[4] result
        PSW[4]<= #2 1;
    else
        PSW[4]<= #2 0;
    end
```

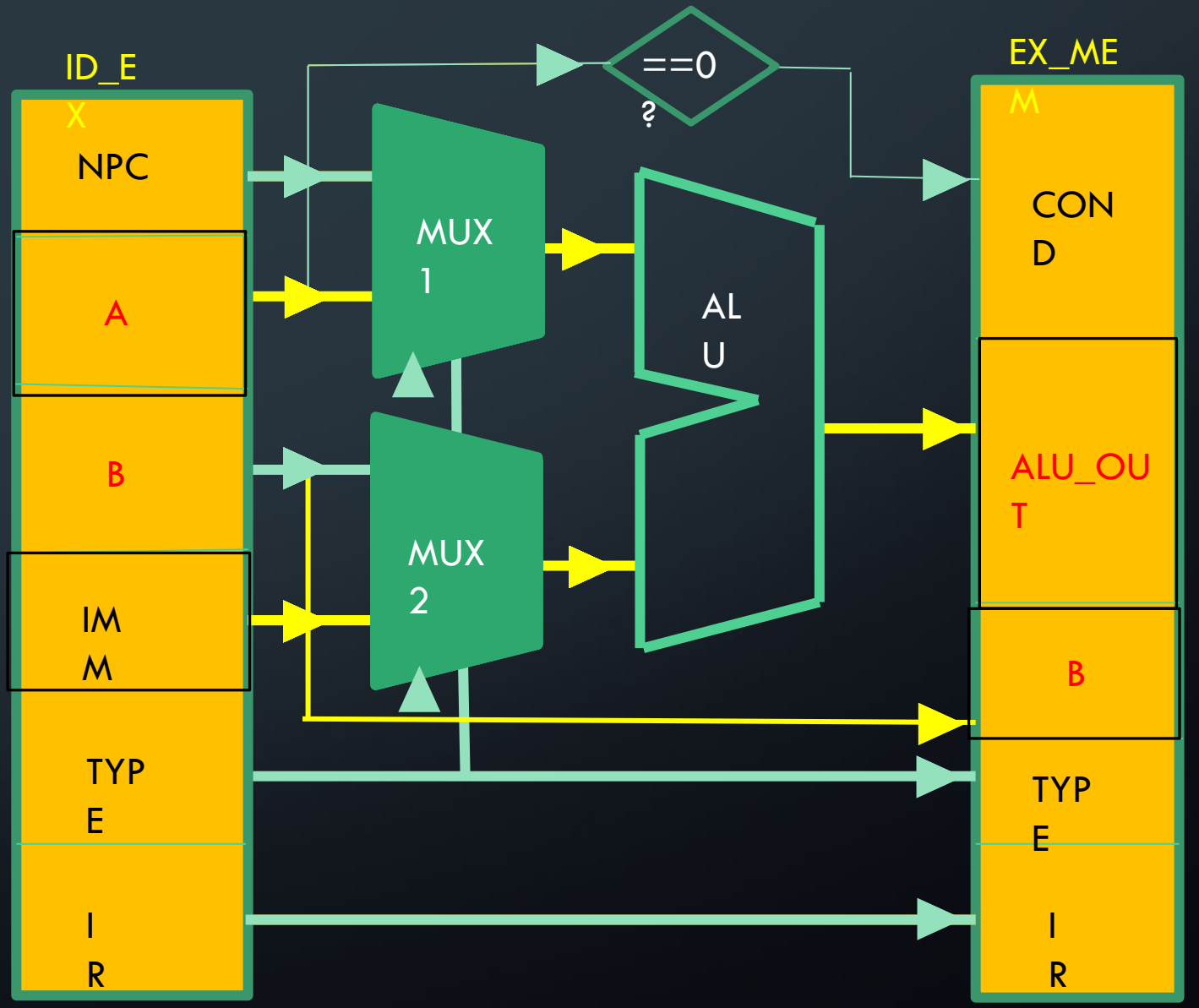


STAGE 3: EXECUTION

I type instruction format

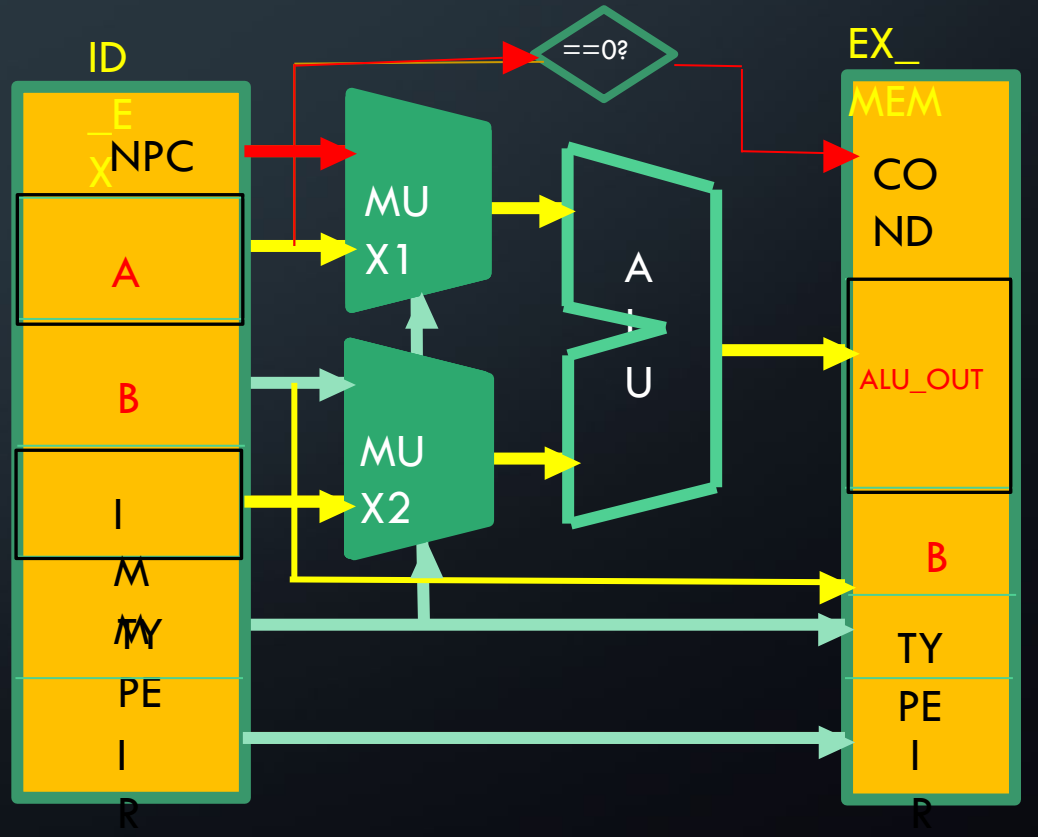


Instructions	opcode
LW	000000
SW	000001
ADDI	000010
SUBI	000011
SLTI	000100
BNEQZ	000101
BEQZ	111111



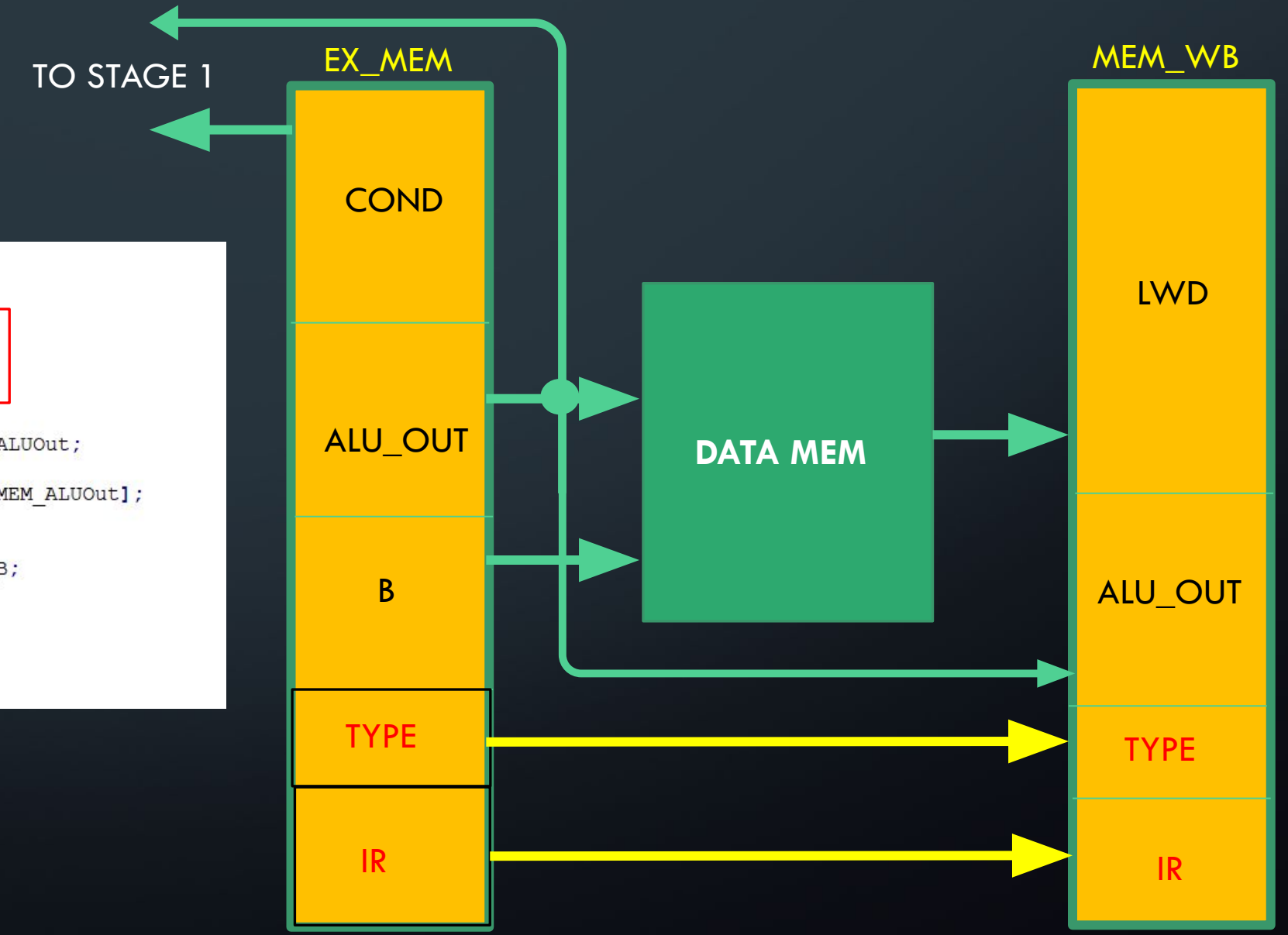
STAGE 3: EXECUTION


```
RM_ALU :
    begin
    case(ID_EX_IR[31:26])  // ""opcode"
        ADDI: {PSW[0],EX_MEM_ALUOut} <= #2 ID_EX_A + ID_EX_Imm;
        SUBI: {PSW[0],EX_MEM_ALUOut} <= #2 ID_EX_A - ID_EX_Imm;
        SLTI: EX_MEM_ALUOut          <= #2 ID_EX_A < ID_EX_Imm;
        default: EX_MEM_ALUOut       <= #2 32'hxxxxxxxx;
    endcase
    if(EX_MEM_ALUOut==0)  // checking whether the ALU result is zero or not
        PSW[4]<= #2 1;    // SET THE ZERO FLAG
    else
        PSW[4]<= #2 0;
    end

LOAD,STORE:
    begin
    EX_MEM_ALUOut <= #2 ID_EX_A + ID_EX_Imm;
    EX_MEM_B      <= #2 ID_EX_B;
    end

BRANCH:
    begin
    EX_MEM_ALUOut <= #2 ID_EX_NPC + ID_EX_Imm;
    EX_MEM_cond   <= #2 (ID_EX_A == 0);
    end
  endcase
```



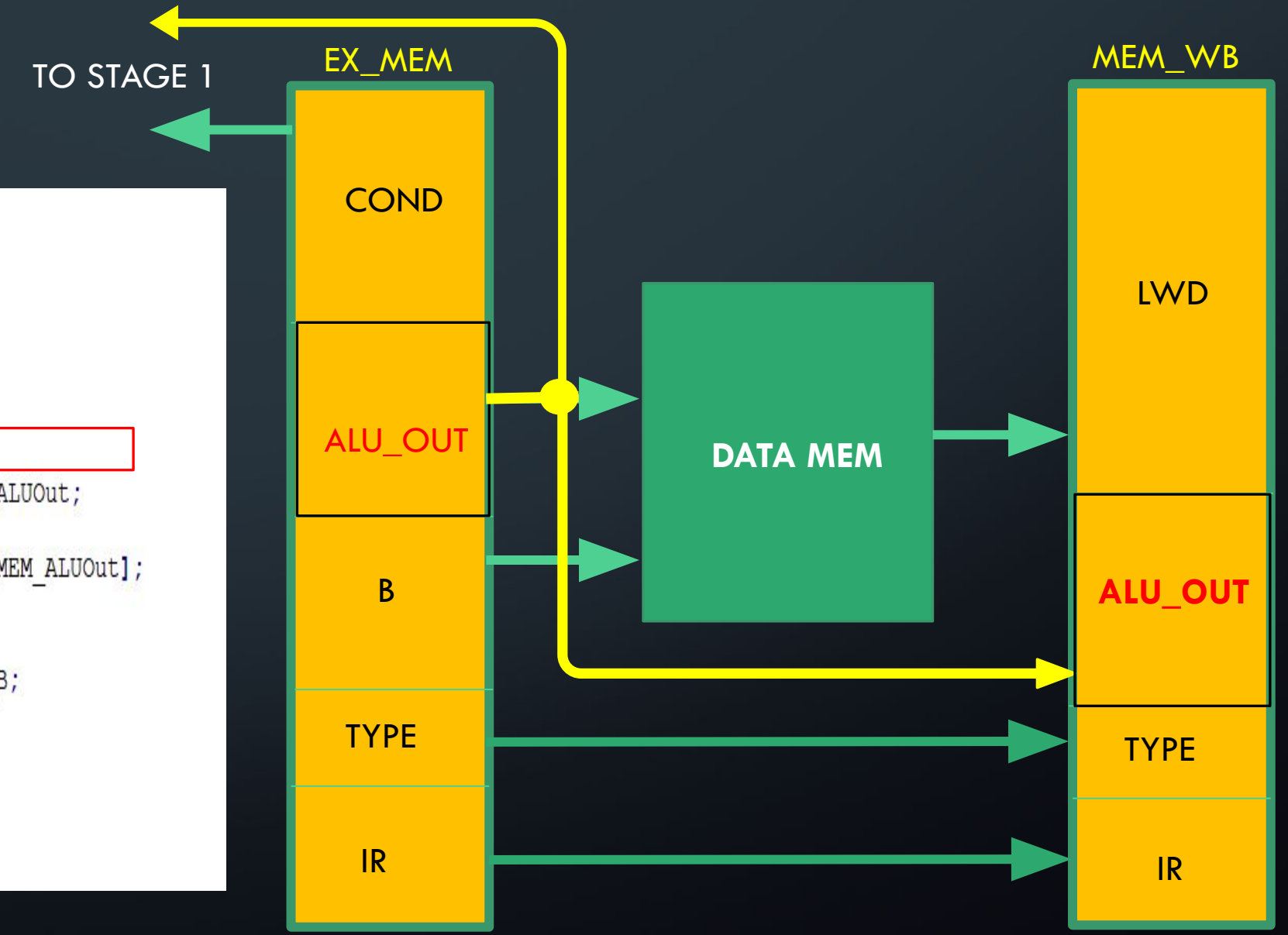
STAGE 4: MEMORY

```
always @(posedge clk2)      // MEM STAGE
  if(PSW[1] == 0 )
    begin
      MEM_WB_type   <= #2 EX_MEM_type;
      MEM_WB_IR     <= #2 EX_MEM_IR;

    case(EX_MEM_type)
     RR_ALU, RM_ALU: MEM_WB_ALUOut  <= #2 EX_MEM_ALUOut;

     LOAD:           MEM_WB_LMD     <= #2 Mem[EX_MEM_ALUOut];

     STORE: if(PSW[2] == 0 )
                Mem[EX_MEM_ALUOut]  <= #2 EX_MEM_B;

    endcase

end
```



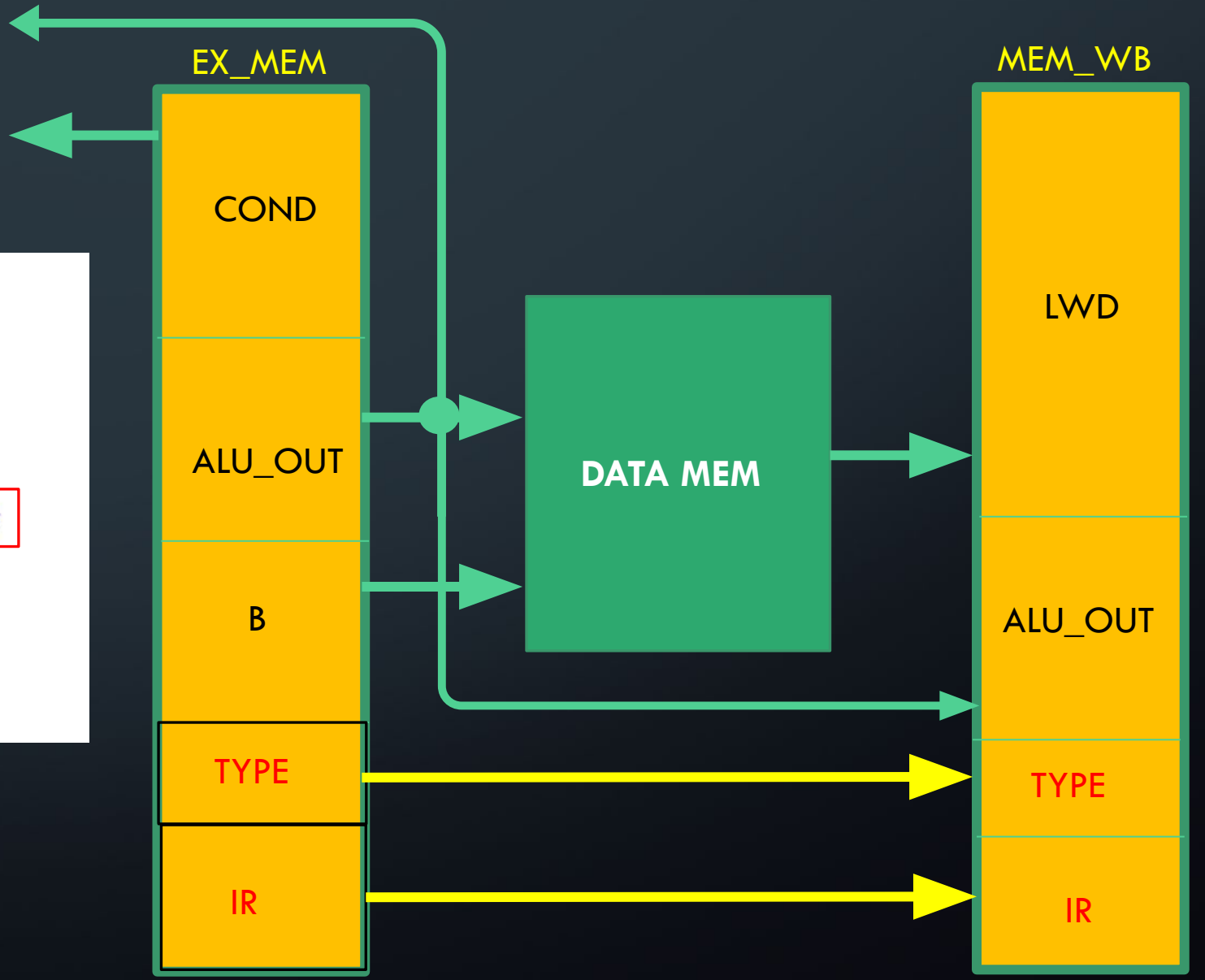
STAGE 4: MEMORY (CONT.)

```
always @(posedge clk2)      // MEM STAGE
  if(PSW[1] == 0 )
    begin
      MEM_WB_type  <= #2 EX_MEM_type;
      MEM_WB_IR    <= #2 EX_MEM_IR;

    case(EX_MEM_type)
    RR_ALU, RM_ALU: MEM_WB_ALUOut  <= #2 EX_MEM_ALUOut;

    LOAD:           MEM_WB_LMD     <= #2 Mem[EX_MEM_ALUOut];

    STORE: if(PSW[2] == 0 )
                Mem[EX_MEM_ALUOut]  <= #2 EX_MEM_B;

    endcase

end
```



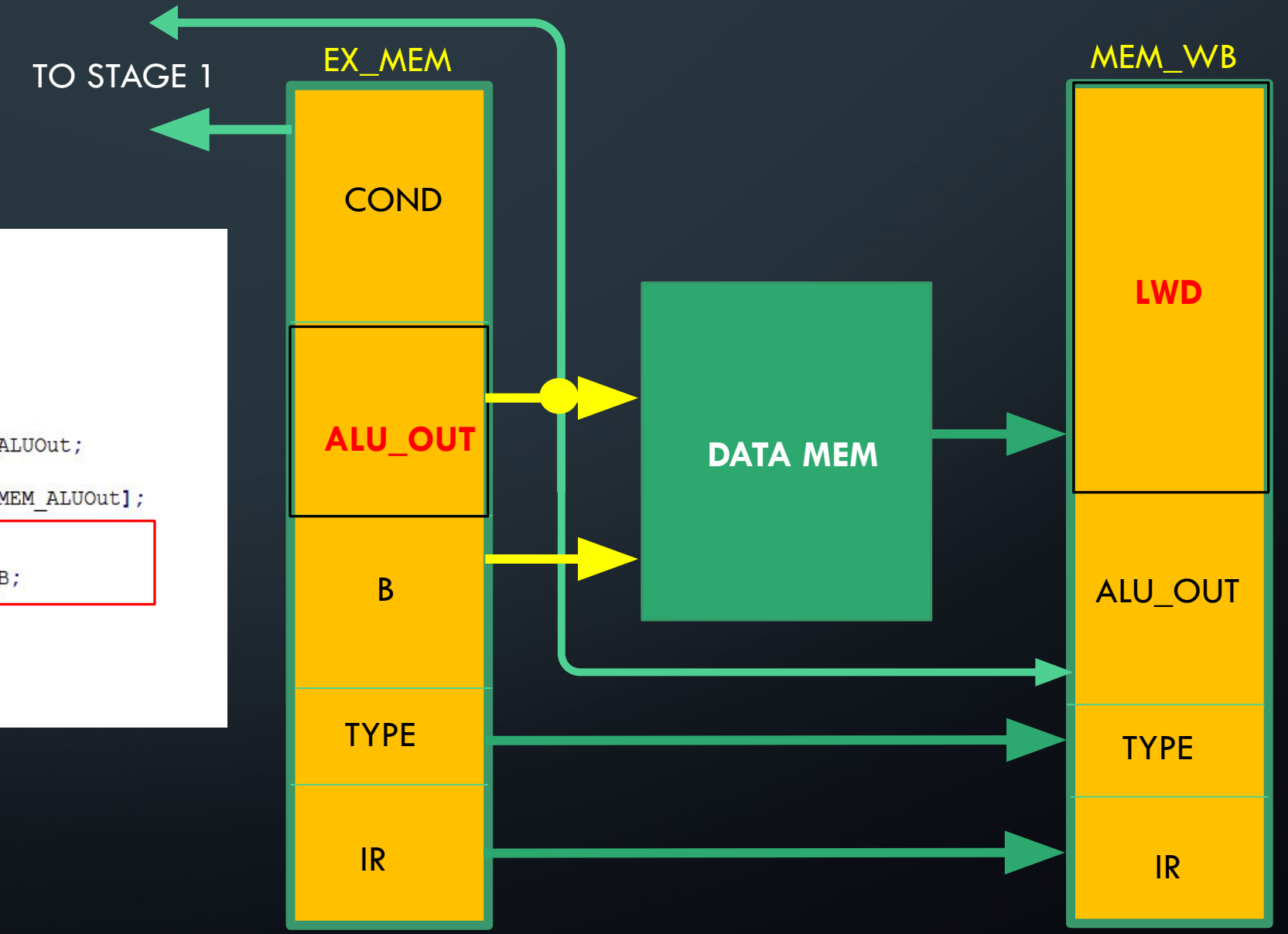
STAGE 4: MEMORY (CONT.)

```
always @(posedge clk2)      // MEM STAGE
  if(PSW[1] == 0 )
    begin
      MEM_WB_type  <= #2 EX_MEM_type;
      MEM_WB_IR    <= #2 EX_MEM_IR;

    case(EX_MEM_type)
     RR_ALU, RM_ALU: MEM_WB_ALUOut  <= #2 EX_MEM_ALUOut;

     LOAD:           MEM_WB_LMD     <= #2 Mem[EX_MEM_ALUOut];

     STORE: if(PSW[2] == 0 )
                Mem[EX_MEM_ALUOut]  <= #2 EX_MEM_B;

    endcase

end
```



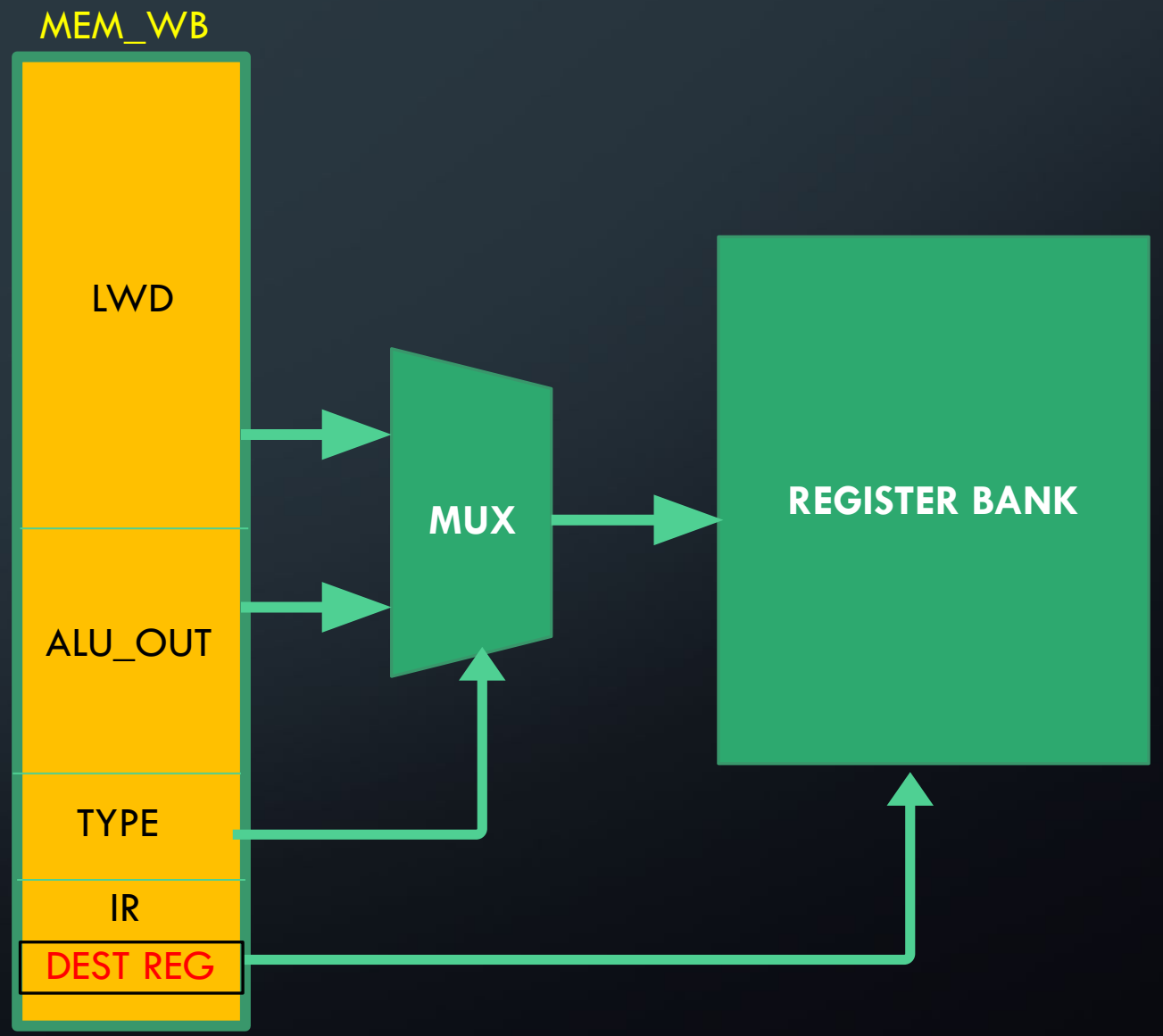
STAGE 4: MEMORY (CONT.)

```
always @(posedge clk2)      // MEM STAGE
  if(PSW[1] == 0 )
    begin
      MEM_WB_type  <= #2 EX_MEM_type;
      MEM_WB_IR    <= #2 EX_MEM_IR;

    case(EX_MEM_type)
     RR_ALU, RM_ALU: MEM_WB_ALUOut  <= #2 EX_MEM_ALUOut;

     LOAD:           MEM_WB_LMD     <= #2 Mem[EX_MEM_ALUOut];

     STORE: if(PSW[2] == 0 )
                Mem[EX_MEM_ALUOut]  <= #2 EX_MEM_B;

    endcase

end
```



STAGE 5: WRITE BACK

```
// stage 5: WB stage Write Back Stage

always @(posedge clk1)
 begin
   if(PSW[2] == 0)
     case (MEM_WB_type)
        RR_ALU: Reg[MEM_WB_IR[15:11]] <= #2 MEM_WB_ALUOut;   // "rd"
        RM_ALU: Reg[MEM_WB_IR[20:16]] <= #2 MEM_WB_ALUOut;   // "rt"
        LOAD  : Reg[MEM_WB_IR[20:16]] <= #2 MEM_WB_LMD;      // "rt"
        HALT  : PSW[1] <= #2 1'b1;
     endcase
 end

endmodule
```

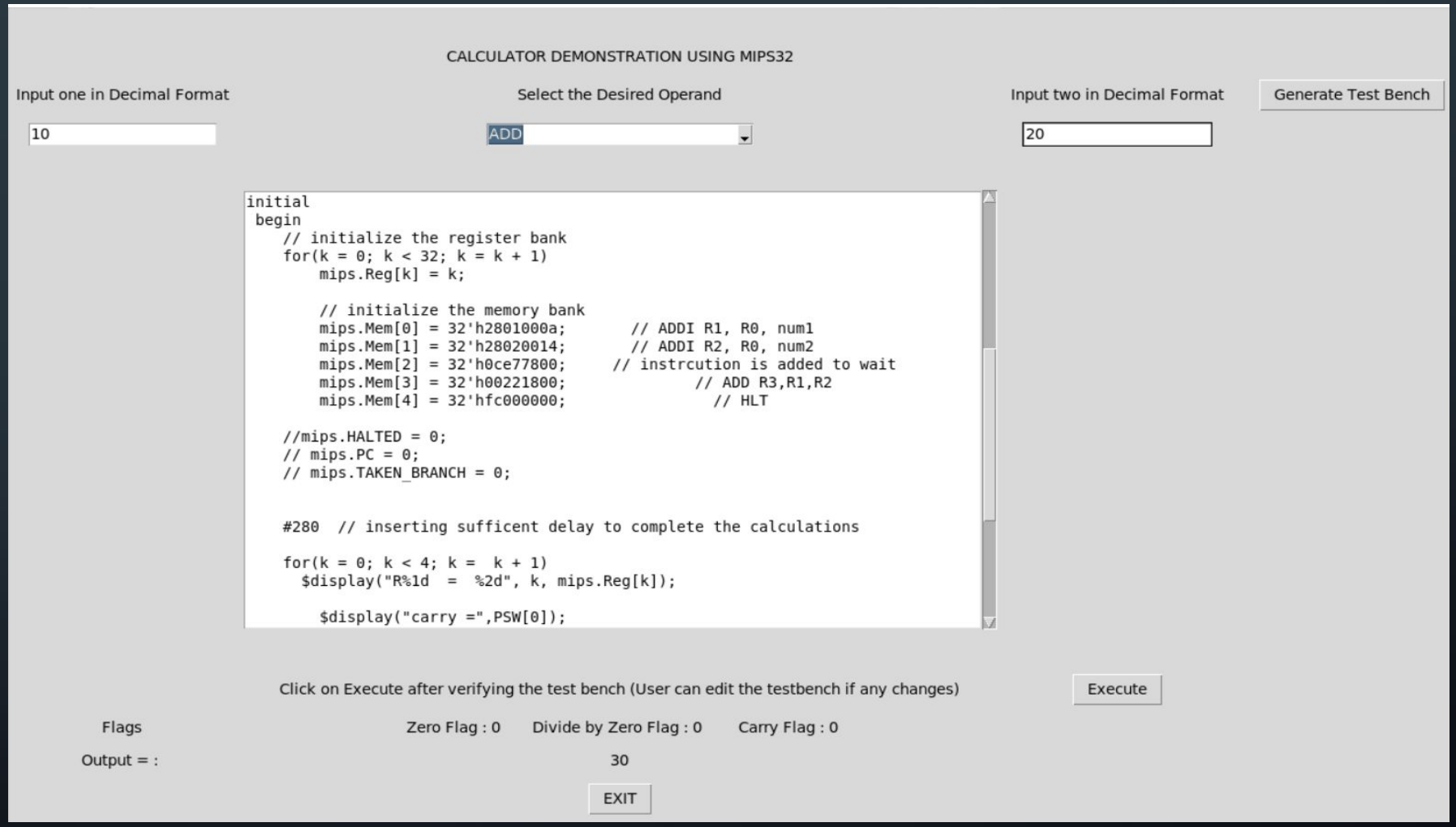


EXAMPLE – ADD TWO NUMBERS

```
// initialize the memory bank  i.e ROM
mlips.Mem[0] = 32'h2801000a;       // ADDI R1, R0, num1
mips.Mem[1] = 32'h28020014;        // ADDI R2, R0, num2
mips.Mem[2] = 32'h0ce77800;        // NOP instruction
mips.Mem[3] = 32'h00221800;        // ADD R3,R1,R2
mips.Mem[4] = 32'hfc000000;        // HALT
```

OPCODE	SOURCE A	SOURCE B(target)	DESTINATI	NOT USED	NOT USED
			16-bit operand for Immediate operand		
001010	00000	00001	00000	00000	01010
001010	00000	00010	00000	00000	10100
000011	00111	00111	00000	00000	00000
000000	00001	00010	00011	00000	00000
111111	000000	00000	00000	00000	00000

GUI FOR MIPS32



THANK YOU