



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Lecture 37: PIPELINE IMPLEMENTATION OF A PROCESSOR (PART 1)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Introduction

- We shall first look at the instruction set architecture of a popular *Reduced Instruction Set Architecture (RISC)* processor, viz. MIPS32.
 - It is a 32-bit processor, i.e. can operate on 32 bits of data at a time.
- We shall look at the instruction types, and how instructions are encoded.
- Then we can understand the process of instruction execution, and the steps involved.
- We shall discuss the pipeline implementation of the processor.
 - Only for a small subset of the instructions (and some simplifying assumptions).
- Finally, we shall present the Verilog design of the pipeline processor.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

2

A Quick Look at MIPS32

- MIPS32 registers:
 - a) 32, 32-bit general purpose registers (GPRs), *R0* to *R31*.
 - Register *R0* contains a constant 0; cannot be written.
 - b) A special-purpose 32-bit program counter (*PC*).
 - Points to the next instruction in memory to be fetched and executed.
- No flag registers (zero, carry, sign, etc.).
- Very few addressing modes (register, immediate, register indexed, etc.)
 - Only load and store instructions can access memory.
- We assume memory word size is 32 bits (*word addressable*).



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

3

The MIPS32 Instruction Subset Being Considered

- Load and Store Instructions


```
LW R2,124(R8)    // R2 = Mem[R8+124]
SW R5,-10(R25)    // Mem[R25-10] = R5
```
- Arithmetic and Logic Instructions (only register operands)

```
ADD R1,R2,R3      // R1 = R2 + R3
ADD R1,R2,R0      // R1 = R2 + 0
SUB R12,R10,R8    // R12 = R10 - R8
AND R20,R1,R5     // R20 = R1 & R5
OR  R11,R5,R6     // R11 = R5 | R6
MUL R5,R6,R7     // R5 = R6 * R7
SLT R5,R11,R12   // If R11 < R12, R5=1; else R5=0
```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

4

- Arithmetic and Logic Instructions (immediate operand)

```
ADDI R1,R2,25      // R1 = R2 + 25
SUBI R5,R1,150    // R5 = R1 - 150
SLTI R2,R10,10    // If R10<10, R2=1; else R2=0
```

- Branch Instructions

```
BEQZ R1,Loop        // Branch to Loop if R1=0
BNEQZ R5,Label      // Branch to Label if R5!=0
```

- Jump Instruction

```
J Loop              // Branch to Loop unconditionally
```

- Miscellaneous Instruction

```
HLT                  // Halt execution
```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

5

MIPS Instruction Encoding

- All MIPS32 instructions can be classified into three groups in terms of instruction encoding.
 - **R-type** (Register), **I-type** (Immediate), and **J-type** (Jump).
 - In an instruction encoding, the 32 bits of the instruction are divided into several fields of fixed widths.
 - All instructions may not use all the fields.
- Since the relative positions of some of the fields are same across instructions, instruction decoding becomes very simple.



IIT KHARAGPUR

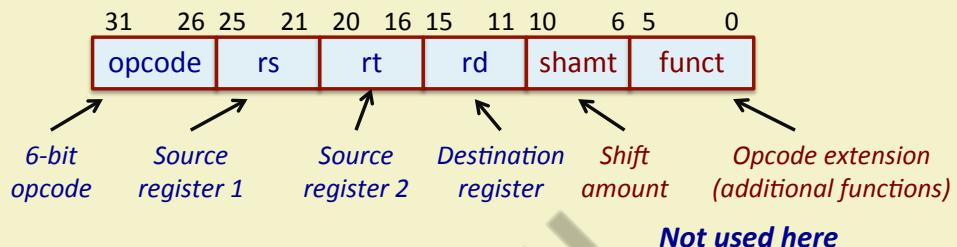
NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

6

(a) R-type Instruction Encoding

- Here an instruction can use up to three register operands.
 - Two source and one destination.
- In addition, for shift instructions, the number of bits to shift can also be specified (*we are not considering such instructions here*).



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

7

- R-type instructions considered with opcode:

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

```

SUB    R5 ,R12 ,R25
000001 01100 11001 00101 00000 000000
      SUB    R12    R25    R5
= 05992800 (in hex)
  
```



IIT KHARAGPUR

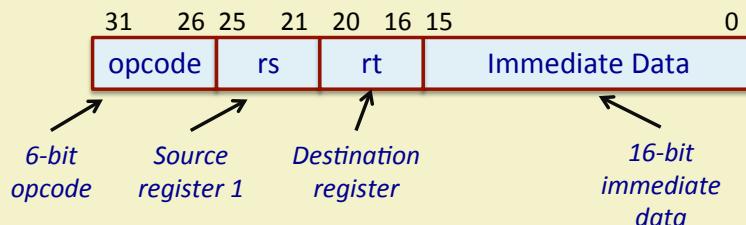
NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

8

(b) I-type Instruction Encoding

- Contains a 16-bit immediate data field.
- Supports one source and one destination register.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

9

- I-type instructions considered with opcode:

Instruction	opcode
LW	001000
SW	001001
ADDI	001010
SUBI	001011
SLTI	001100
BNEQZ	001101
BEQZ	001110

LW R20 , 84 (R9)

```
001000 01001 10100 0000000001010100
    LW      R9      R20      offset
= 21340054 (in hex)
```

BEQZ R25 , Label

```
001110 11001 00000 YYYYYYYYYYYYYYYY
    BEQZ   R25   Unused      offset
= 3b20YYYY (in hex)
```



IIT KHARAGPUR

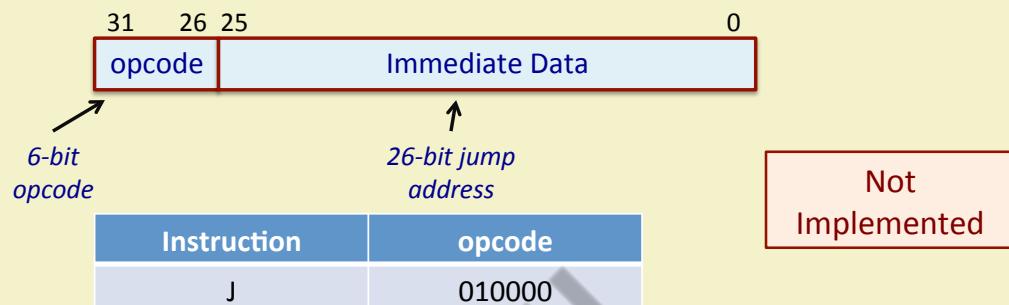
NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

10

(c) J-type Instruction Encoding

- Contains a 26-bit jump address field.
 - Extended to 28 bits by padding two 0's on the right.



A Quick View

	31	26	25	21	20	16	15	11	10	6	5	0
R-type	opcode	rs		rt		rd	shamt		funct			
I-type	31	26	25	21	20	16	15					0
I-type	opcode	rs		rt			Immediate Data					
J-type	31	26	25									0
J-type	opcode						Immediate Data					

- Some instructions require two register operands *rs* & *rt* as input, while some require only *rs*.
 - Gets known only after instruction is decoded.
 - While decoding is going on, we can prefetch the registers in parallel.
 - May or may not be required later.

- Similarly, the 16-bit and 26-bit immediate data are retrieved and sign-extended to 32-bits in case they are required later.

Addressing Modes in MIPS32

- Register addressing ***ADD R1,R2,R3***
- Immediate addressing ***ADDI R1,R2, 200***
- Base addressing ***LW R5, 150(R7)***
 - Content of a register is added to a “base” value to get the operand address.
- PC relative addressing ***BEQZ R3, Label***
 - 16-bit offset is added to PC to get the target address.
- Pseudo-direct addressing ***J Label***
 - 26-bit offset is added to PC to get the target address.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

13

END OF LECTURE 37



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

14



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

Lecture 38: PIPELINE IMPLEMENTATION OF A PROCESSOR (PART 2)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MIPS32 Instruction Cycle

- We divide the instruction execution cycle into five steps:
 - a) IF : Instruction Fetch
 - b) ID : Instruction Decode / Register Fetch
 - c) EX : Execution / Effective Address Calculation
 - d) MEM : Memory Access / Branch Completion
 - e) WB : Register Write-back
- We now show the generic micro-instructions carried out in the various steps.



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

16

(a) IF : Instruction Fetch

- Here the instruction pointed to by PC is fetched from memory, and also the next value of PC is computed.
 - Every MIPS32 instruction is of 32 bits.
 - Every memory word is of 32 bits and has a unique address.
 - For a branch instruction, new value of the PC may be the target address. So PC is not updated in this stage; new value is stored in a register NPC .

IF: $IR \leftarrow \text{Mem}[PC];$
 $NPC \leftarrow PC + 1;$

For byte addressable memory, PC has to be incremented by 4.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

17

(b) ID : Instruction Decode

- The instruction already fetched in IR is decoded.
 - $Opcode$ is 6-bits (bits 31:26).
 - First source operand rs (bits 25:21), second source operand rt (bits 20:16).
 - 16-bit immediate data (bits 15:0).
 - 26-bit immediate data (bits 25:0).
- Decoding is done in parallel with reading the register operands rs and rt .
 - Possible because these fields are in a fixed location in the instruction format.
- In a similar way, the immediate data are sign-extended.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

18

ID:

```

A ← Reg [rs];
B ← Reg [rt];
Imm ← (IR15)16 ## IR15..0 // sign extend 16-bit immediate field
Imm1 ← (IR25)6 ## IR25..0 // sign extend 26-bit immediate field

```

A, B, Imm, Imm1 are temporary registers.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

19

(c) EX: Execution / Effective Address Computation

- In this step, the ALU is used to perform some calculation.
 - The exact operation depends on the instruction that is already decoded.
 - The ALU operates on operands that have been already made ready in the previous cycle.
 - A, B, Imm, etc.
- We show the micro-instructions corresponding to the type of instruction.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

20

Memory Reference:

$$\text{ALUOut} \leftarrow \text{A} + \text{Imm};$$

Example: LW R3, 100(R8)

Register-Register ALU Instruction:

$$\text{ALUOut} \leftarrow \text{A func B};$$

Example: SUB R2, R5, R12

Register-Immediate ALU Instruction:

$$\text{ALUOut} \leftarrow \text{A func Imm};$$

Example: SUBI R2, R5, 524

Branch:

$$\text{ALUOut} \leftarrow \text{NPC} + \text{Imm};$$

$$\text{cond} \leftarrow (\text{A op 0});$$

Example: BEQZ R2, Label
[op is ==]



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

21

(d) MEM: Memory Access / Branch Completion

- The only instructions that make use of this step are loads, stores, and branches.
 - The load and store instructions access the memory.
 - The branch instruction updates PC depending upon the outcome of the branch condition.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

22

Load instruction:

```
PC ← NPC;
LMD ← Mem [ALUOut];
```

Store instruction:

```
PC ← NPC;
Mem [ALUOut] ← B;
```

Other instructions:

```
PC ← NPC;
```

Branch instruction:

```
if (cond) PC ← ALUOut;
else PC ← NPC;
```



IIT KHARAGPUR

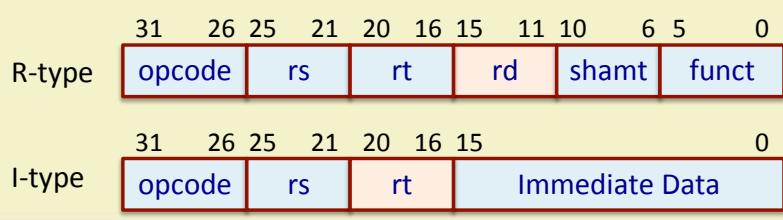
NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

23

(e) WB: Register Write Back

- In this step, the result is written back into the register file.
 - Result may come from the ALU.
 - Result may come from the memory system (viz. a LOAD instruction).
- The position of the destination register in the instruction word depends on the instruction → *already known after decoding has been done.*



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

24

Register-Register ALU Instruction:

Reg [rd] ← ALUOut;

Register-Immediate ALU Instruction:

Reg [rt] ← ALUOut;

Load Instruction:

Reg [rt] ← LMD;



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

25

SOME EXAMPLE INSTRUCTION EXECUTION



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

26

ADD R2, R5, R10

IF	IR \leftarrow Mem [PC]; NPC \leftarrow PC + 1 ;
ID	A \leftarrow Reg [rs]; B \leftarrow Reg [rt];
EX	ALUOut \leftarrow A + B;
MEM	PC \leftarrow NPC;
WB	Reg [rd] \leftarrow ALUOut;

ADDI R2, R5, 150

IF	IR \leftarrow Mem [PC]; NPC \leftarrow PC + 1 ;
ID	A \leftarrow Reg [rs]; Imm \leftarrow (IR ₁₅) ¹⁶ ## IR _{15..0}
EX	ALUOut \leftarrow A + Imm;
MEM	PC \leftarrow NPC;
WB	Reg [rt] \leftarrow ALUOut;



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

27

LW R2, 200 (R6)

IF	IR \leftarrow Mem [PC]; NPC \leftarrow PC + 1 ;
ID	A \leftarrow Reg [rs]; Imm \leftarrow (IR ₁₅) ¹⁶ ## IR _{15..0}
EX	ALUOut \leftarrow A + Imm;
MEM	PC \leftarrow NPC; LMD \leftarrow Mem [ALUOut];
WB	Reg [rt] \leftarrow LMD;

SW R3, 25 (R10)

IF	IR \leftarrow Mem [PC]; NPC \leftarrow PC + 1 ;
ID	A \leftarrow Reg [rs]; B \leftarrow Reg [rt]; Imm \leftarrow (IR ₁₅) ¹⁶ ## IR _{15..0}
EX	ALUOut \leftarrow A + Imm;
MEM	PC \leftarrow NPC; Mem [ALUOut] \leftarrow B;
WB	-



IIT KHARAGPUR

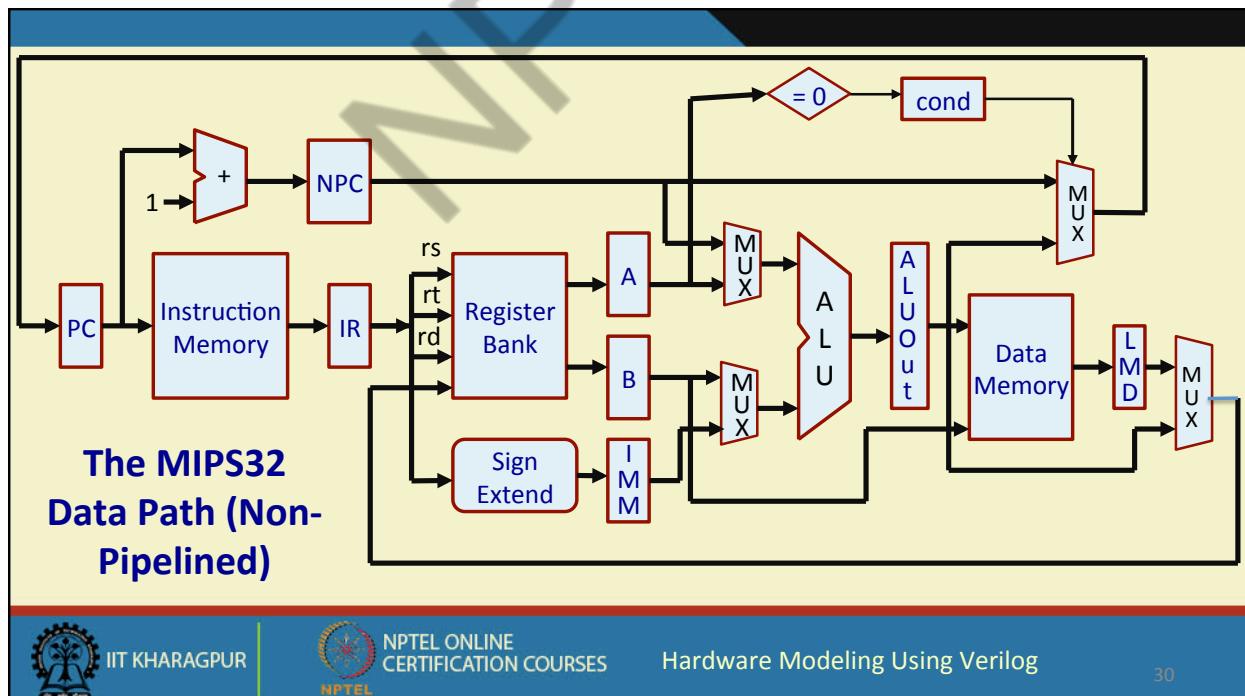
NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

28

BEQZ R3, Label

IF	$IR \leftarrow Mem[PC];$ $NPC \leftarrow PC + 1;$
ID	$A \leftarrow Reg[rs];$ $Imm \leftarrow (IR_{15})^{16} \# IR_{15..0}$
EX	$ALUOut \leftarrow NPC + Imm;$ $cond \leftarrow (A == 0);$
MEM	$PC \leftarrow NPC;$ if ($cond$) $PC \leftarrow ALUOut;$
WB	-



END OF LECTURE 38



IIT KHARAGPUR



NPTEL
NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

31



IIT KHARAGPUR



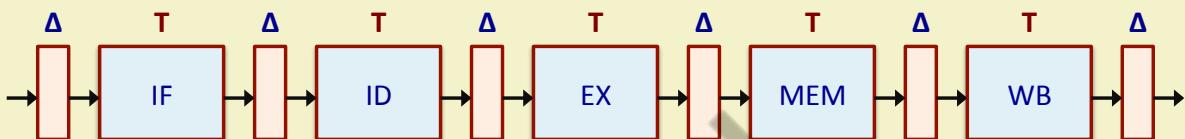
NPTEL ONLINE
CERTIFICATION COURSES
NPTEL

Lecture 39: PIPELINE IMPLEMENTATION OF A PROCESSOR (PART 3)

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Introduction

- Basic requirements for pipelining the MIPS32 data path:
 - We should be able to start a new instruction every clock cycle.
 - Each of the five steps mentioned before (IF, ID, EX, MEM and WB) becomes a pipeline stage.
 - Each stage must finish its execution within one clock cycle.



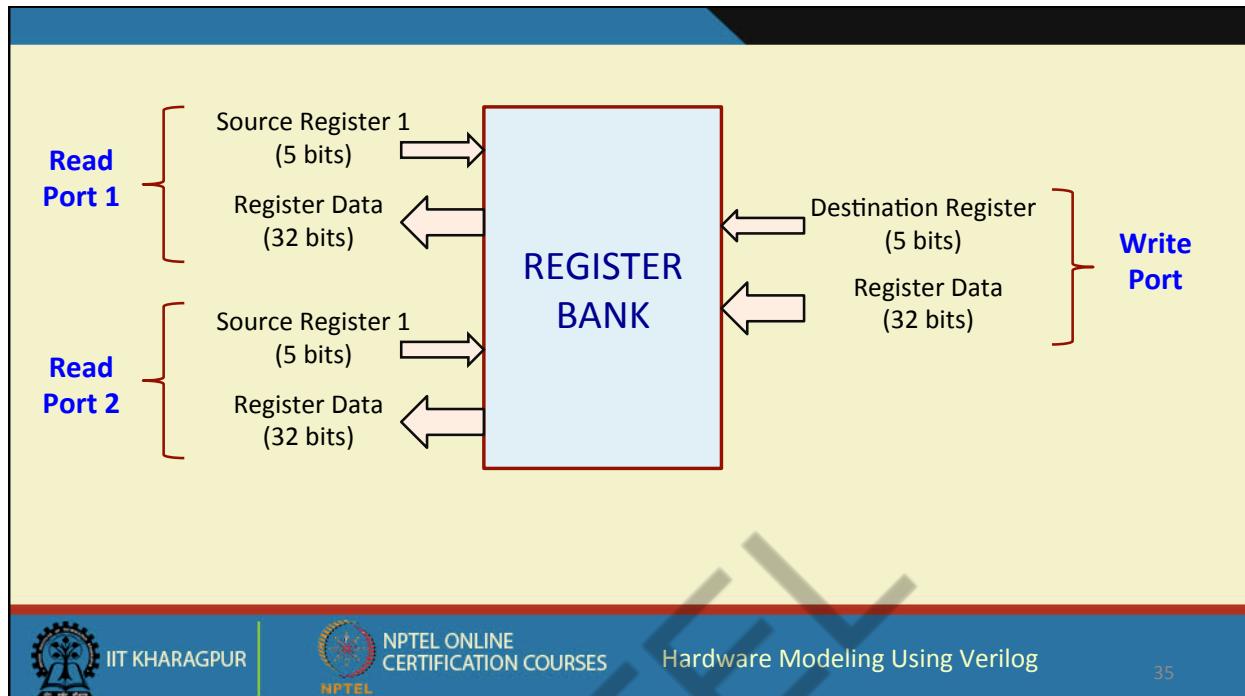
Clock Cycles								
Instruction	1	2	3	4	5	6	7	8
i	IF	ID	EX	MEM	WB			
$i+1$		IF	ID	EX	MEM	WB		
$i+2$			IF	ID	EX	MEM	WB	
$i+3$				IF	ID	EX	MEM	WB

**Instr-i
finishes**

*Instr-(i+
finishe*

\downarrow
*Instr-(i+2)
finishes*

1) *Instr-(i+3)*
finishes



Micro-operations for Pipelined MIPS32

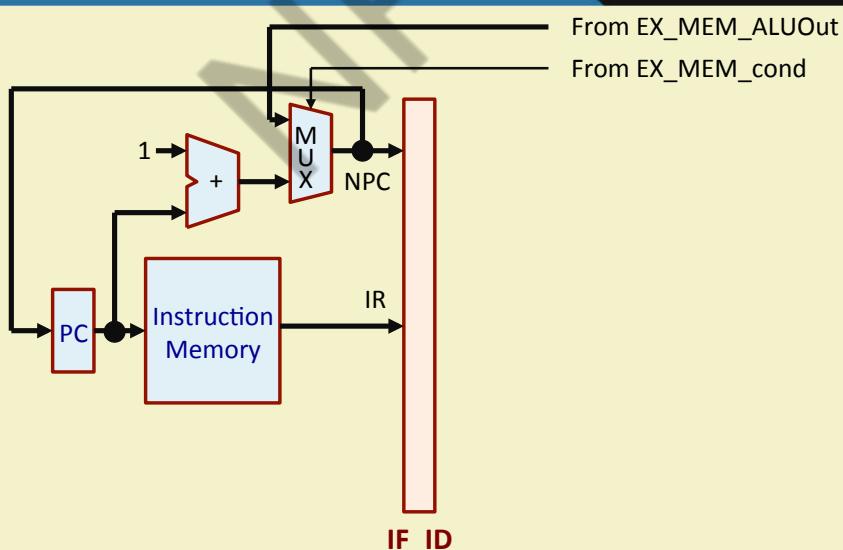
- Convention used:
 - Most of the temporary registers required in the data path are included as part of the inter-stage latches.
 - IF_ID:** denotes the latch stage between the IF and ID stages.
 - ID_EX:** denotes the latch stage between the ID and EX stages.
 - EX_MEM:** denotes the latch stage between the EX and MEM stages.
 - MEM_WB:** denotes the latch stage between the MEM and WB stages.
- Example:
 - ID_EX_A** means register **A** that is implemented as part of the **ID_EX** latch stage.



(a) Micro-operations for Pipeline Stage IF

```

IF_ID_IR      ← Mem [PC];
IF_ID_NPC,PC ← ( if ((EX_MEM_IR[opcode] == branch) & EX_MEM_cond)
                  { EX_MEM_ALUOut}
                  else {PC + 1} );
    
```

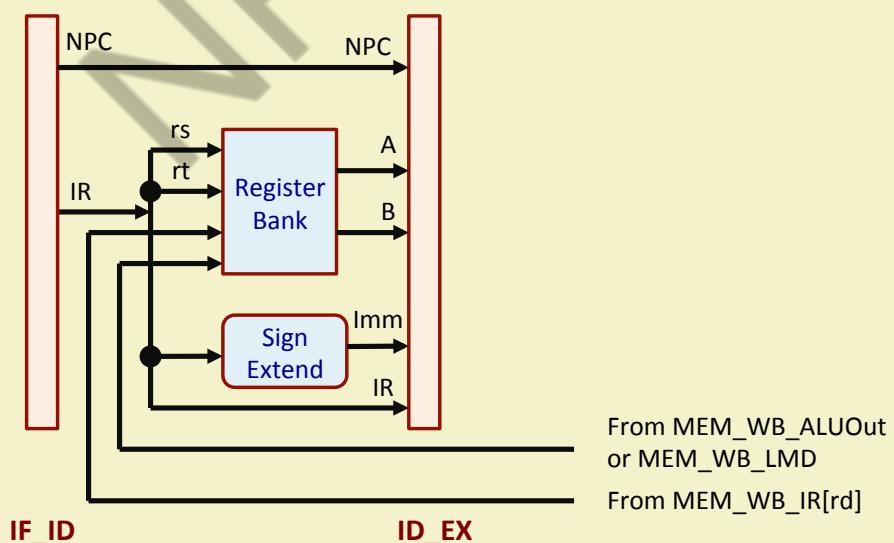


(b) Micro-operations for Pipeline Stage ID

```

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_IR15..0);

```



(c) Micro-operations for Pipeline Stage EX

EX_MEM_IR \leftarrow ID_EX_IR;
 EX_MEM_ALUOut \leftarrow ID_EX_A func ID_EX_B;

R-R ALU

EX_MEM_IR \leftarrow ID_EX_IR;
 EX_MEM_ALUOut \leftarrow ID_EX_A func ID_EX_Imm;

R-M ALU

EX_MEM_IR \leftarrow ID_EX_IR;
 EX_MEM_ALUOut \leftarrow ID_EX_A + ID_EX_Imm;
 EX_MEM_B \leftarrow ID_EX_B;

EX_MEM_ALUOut \leftarrow ID_EX_NPC +
 ID_EX_Imm;
 EX_MEM_cond \leftarrow (ID_EX_A == 0);

BRANCH

LOAD / STORE

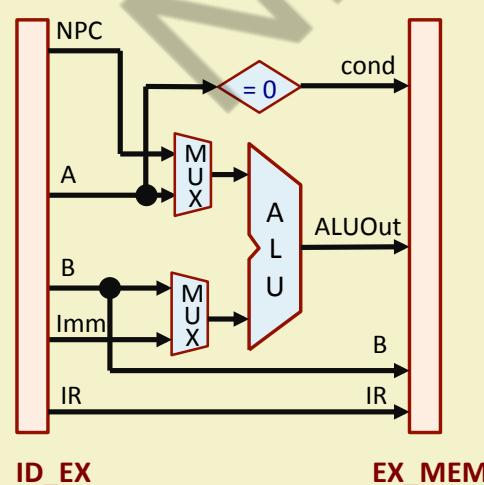


IIT KHARAGPUR

NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

41



IIT KHARAGPUR

NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

42

(d) Micro-operations for Pipeline Stage MEM

MEM_WB_IR \leftarrow EX_MEM_IR;
 MEM_WB_ALUOut \leftarrow EX_MEM_ALUOut;

ALU

MEM_WB_IR \leftarrow EX_MEM_IR;
 MEM_WB_LMD \leftarrow Mem [EX_MEM_ALUOut];

LOAD

MEM_WB_IR \leftarrow EX_MEM_IR;
 Mem [EX_MEM_ALUOut] \leftarrow EX_MEM_B;

STORE

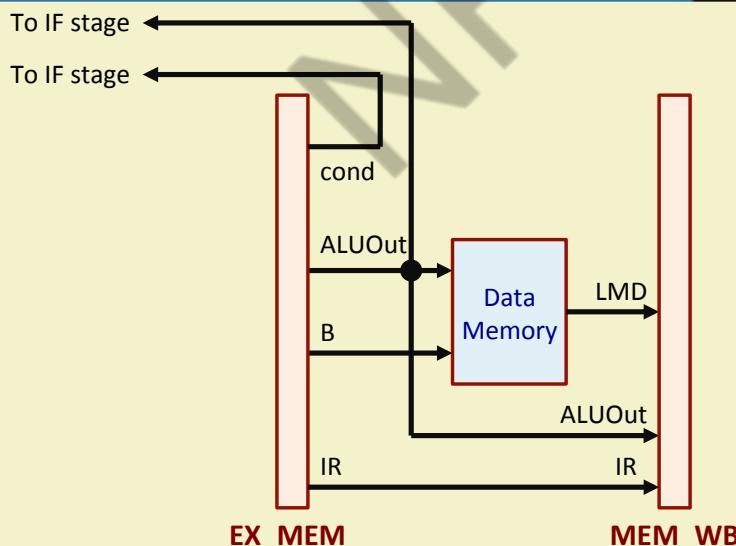


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

43



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

44

(e) Micro-operations for Pipeline Stage WB

Reg [MEM_WB_IR [rd]] \leftarrow MEM_WB_ALUOut; **R-R ALU**

Reg [MEM_WB_IR [rt]] \leftarrow MEM_WB_ALUOut; **R-M ALU**

Reg [MEM_WB_IR [rt]] \leftarrow MEM_WB_LMD; **LOAD**

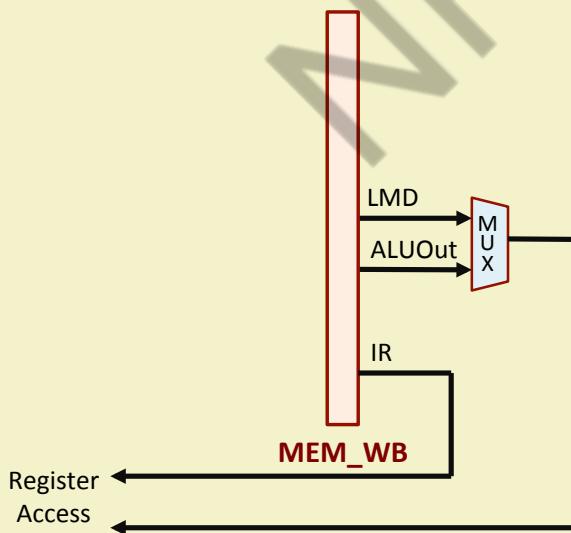


IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

45



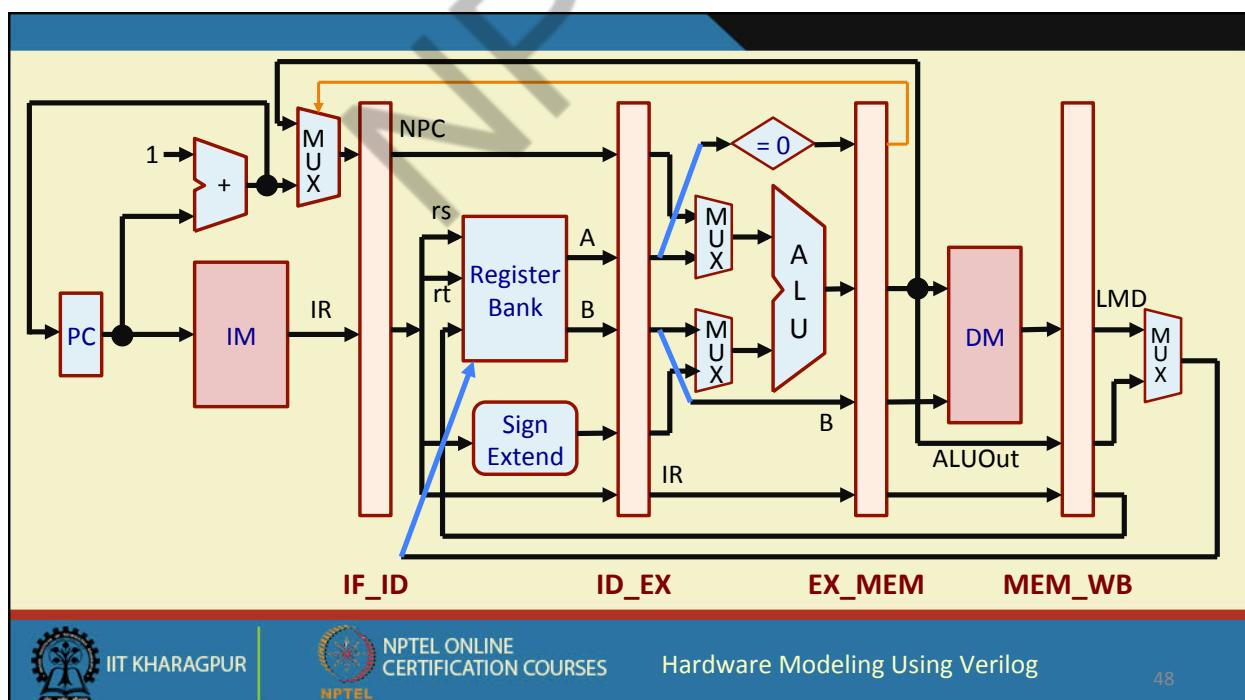
IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

46

PUTTING IT ALL TOGETHER :: MIPS32 PIPELINE



END OF LECTURE 39



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

49



IIT KHARAGPUR



NPTEL
ONLINE
CERTIFICATION COURSES

Lecture 40: VERILOG MODELING OF THE PROCESSOR (PART 1)

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Verilog Implementation of MIPS32 Pipeline

- Here we show the behavioral Verilog code to implement the pipeline.
- Two special 1-bit variables are used:
 - **HALTED** :: Set after a HLT instruction executes and reaches the WB stage.
 - **TAKEN_BRANCH** :: Set after the decision to take a branch is known. Required to disable the instructions that have already entered the pipeline from making any state changes.



```
module pipe_MIPS32 (clk1, clk2);
  input clk1, clk2;          // Two-phase clock

  reg [31:0] PC, IF_ID_IR, IF_ID_NPC;
  reg [31:0] ID_EX_IR, ID_EX_NPC, ID_EX_A, ID_EX_B, ID_EX_Imm;
  reg [2:0]  ID_EX_type, EX_MEM_type, MEM_WB_type;
  reg [31:0] EX_MEM_IR, EX_MEM_ALUOut, EX_MEM_B;
  reg       EX_MEM_cond;
  reg [31:0] MEM_WB_IR, MEM_WB_ALUOut, MEM_WB_LMD;

  reg [31:0] Reg [0:31];    // Register bank (32 x 32)
  reg [31:0] Mem [0:1023]; // 1024 x 32 memory

  parameter ADD=6'b000000, SUB=6'b000001, AND=6'b000010, OR=6'b000011,
           SLT=6'b000100, MUL=6'b000101, HLT=6'b111111, LW=6'6001000,
           SW=6'b001001, ADDI=6'b001010, SUBI=6'b001011, SLTI=6'b001100,
           BNEQZ=6'b001101, BEQZ=6'b001110;
```



```

parameter RR_ALU=3'b000, RM_ALU=3'b001, LOAD=3'b010, STORE=3'b011,
BRANCH=3'b100, HALT=3'b101;

reg HALTED;
// Set after HLT instruction is completed (in WB stage)

reg TAKEN_BRANCH;
// Required to disable instructions after branch

```



```

always @(posedge clk1)          // IF Stage
  if (HALTED == 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];
          TAKEN_BRANCH <= #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
  end

```



```

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

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

      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]}};
    end
  end
end

```



```

case (IF_ID_IR[31:26])
  ADD,SUB,AND,OR,SLT,MUL: 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;
                           // Invalid opcode
endcase
end

```



```

always @(posedge clk1)           // EX Stage
  if (HALTED == 0)
    begin
      EX_MEM_type <= #2 ID_EX_type;
      EX_MEM_IR   <= #2 ID_EX_IR;
      TAKEN_BRANCH <= #2 0;

      case (ID_EX_type)
        RR_ALU: begin
          case (ID_EX_IR[31:26]) // "opcode"
            ADD:     EX_MEM_ALUOut <= #2 ID_EX_A + ID_EX_B;
            SUB:     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;
            SLT:     EX_MEM_ALUOut <= #2 ID_EX_A < ID_EX_B;
            MUL:     EX_MEM_ALUOut <= #2 ID_EX_A * ID_EX_B;
            default: EX_MEM_ALUOut <= #2 32'hxxxxxxxxx;
          endcase
        end
    end

```

```

RM_ALU: begin
  case (ID_EX_IR[31:26]) // "opcode"
    ADDI:    EX_MEM_ALUOut <= #2 ID_EX_A + ID_EX_Imm;
    SUBI:    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'hxxxxxxxxx;
  endcase
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
end

```



```

always @(posedge clk2)          // MEM Stage
if (HALTED == 0)
begin
    MEM_WB_type <= 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 (TAKEN_BRANCH == 0) // Disable write
                    Mem[EX_MEM_ALUOut] <= #2 EX_MEM_B;
    endcase
end

```



```

always @(posedge clk1)          // WB Stage
begin
  if (TAKEN_BRANCH == 0)      // Disable write if branch taken
    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:    HALTED <= #2 1'b1;
    endcase
  end
endmodule

```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

61

END OF LECTURE 40



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

62



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Lecture 41: VERILOG MODELING OF THE PROCESSOR (PART 2)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Running Example Programs on the Processor

- We shall show how test benches can be written for verifying the operation of the processor model.
- What will be the test benches like?
 - Load a program from a specific memory address (say, 0).
 - Initialize PC with the starting address of the program.
 - The program starts executing, and will continue to do so until the HLT instruction is encountered.
 - We can print the results from memory locations or registers to verify the operation.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

64

Example 1

- Add three numbers 10, 20 and 30 stored in processor registers.
- The steps:
 - Initialize register R1 with 10.
 - Initialize register R2 with 20.
 - Initialize register R3 with 30.
 - Add the three numbers and store the sum in R4.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

65

Assembly Language Program	Machine Code (in Binary)
ADDI R1,R0,10	001010 00000 00001 0000000000001010
ADDI R2,R0,20	001010 00000 00010 00000000000010100
ADDI R3,R0,25	001010 00000 00011 00000000000011001
ADD R4,R1,R2	000000 00001 00010 00100 00000 000000
ADD R5,R4,R3	000000 00100 00011 00101 00000 000000
HLT	111111 00000 00000 00000 00000 000000



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

66

```

module test_mips32;

reg clk1, clk2;
integer k;

pipe_MIPS32 mips (clk1, clk2);

initial
begin
    clk1 = 0; clk2 = 0;
    repeat (20)                                // Generating two-phase clock
        begin
            #5 clk1 = 1; #5 clk1 = 0;
            #5 clk2 = 1; #5 clk2 = 0;
        end
    end
end

```

TEST BENCH



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

67

```

initial
begin
    for (k=0; k<31; k++)
        mips.Reg[k] = k;

    mips.Mem[0] = 32'h2801000a; // ADDI R1,R0,10
    mips.Mem[1] = 32'h28020014; // ADDI R2,R0,20
    mips.Mem[2] = 32'h28030019; // ADDI R3,R0,25
    mips.Mem[3] = 32'h0ce77800; // OR   R7,R7,R7 -- dummy instr.
    mips.Mem[4] = 32'h0ce77800; // OR   R7,R7,R7 -- dummy instr.
    mips.Mem[5] = 32'h00222000; // ADD  R4,R1,R2
    mips.Mem[6] = 32'h0ce77800; // OR   R7,R7,R7 -- dummy instr.
    mips.Mem[7] = 32'h00832800; // ADD  R5,R4,R3
    mips.Mem[8] = 32'hfc000000; // HLT

```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

68

```

mips.HALTED = 0;
mips.PC = 0;
mips.TAKEN_BRANCH = 0;

#280
for (k=0; k<6; k++)
    $display ("R%1d - %2d", k, mips.Reg[k]);
end

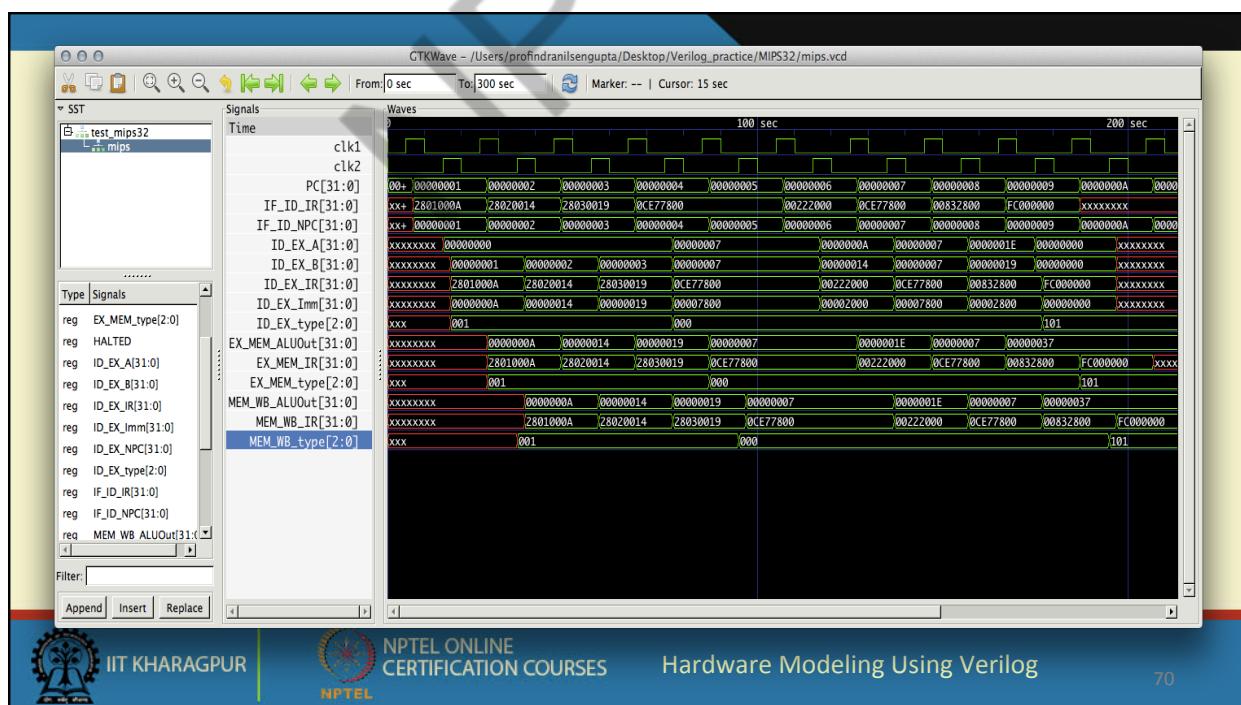
initial
begin
    $dumpfile ("mips.vcd");
    $dumpvars (0, test_mips32);
    #300 $finish;
end

endmodule

```

SIMULATION OUTPUT

R0	- 0
R1	- 10
R2	- 20
R3	- 25
R4	- 30
R5	- 55



Example 2

- Load a word stored in memory location 120, add 45 to it, and store the result in memory location 121.
- The steps:
 - Initialize register R1 with the memory address 120.
 - Load the contents of memory location 120 into register R2.
 - Add 45 to register R2.
 - Store the result in memory location 121.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

71

Assembly Language Program	Machine Code (in Binary)
ADDI R1,R0,120	001010 00000 00001 0000000001111000
LW R2,0(R1)	001000 00001 00010 0000000000000000
ADDI R2,R2,45	001010 00010 00010 0000000000101101
SW R2,1(R1)	001001 00010 00001 0000000000000001
HLT	111111 00000 00000 00000 00000 00000



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

72

```

module test_mips32;

reg clk1, clk2;
integer k;

pipe_MIPS32 mips (clk1, clk2);

initial
begin
    clk1 = 0; clk2 = 0;
    repeat (50)                                // Generating two-phase clock
        begin
            #5 clk1 = 1; #5 clk1 = 0;
            #5 clk2 = 1; #5 clk2 = 0;
        end
    end
end

```

TEST BENCH



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

73

```

initial
begin
    for (k=0; k<31; k++)
        mips.Reg[k] = k;

    mips.Mem[0] = 32'h28010078; // ADDI R1,R0,120
    mips.Mem[1] = 32'h0c631800; // OR   R3,R3,R3 -- dummy instr.
    mips.Mem[2] = 32'h20220000; // LW   R2,0(R1)
    mips.Mem[3] = 32'h0c631800; // OR   R3,R3,R3 -- dummy instr.
    mips.Mem[4] = 32'h2842002d; // ADDI R2,R2,45
    mips.Mem[5] = 32'h0c631800; // OR   R3,R3,R3 -- dummy instr.
    mips.Mem[6] = 32'h24220001; // SW   R2,1(R1)
    mips.Mem[7] = 32'hfc000000; // HLT

    mips.Mem[120] = 85;

```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

74

```

mips.PC = 0;
mips.HALTED = 0;
mips.TAKEN_BRANCH = 0;

#500 $display ("Mem[120]: %4d \nMem[121]: %4d",
               mips.Mem[120], mips.Mem[121]);
end

initial
begin
$dumpfile ("mips.vcd");
$dumpvars (0, test_mips32);
#600 $finish;
end

endmodule

```

SIMULATION OUTPUT

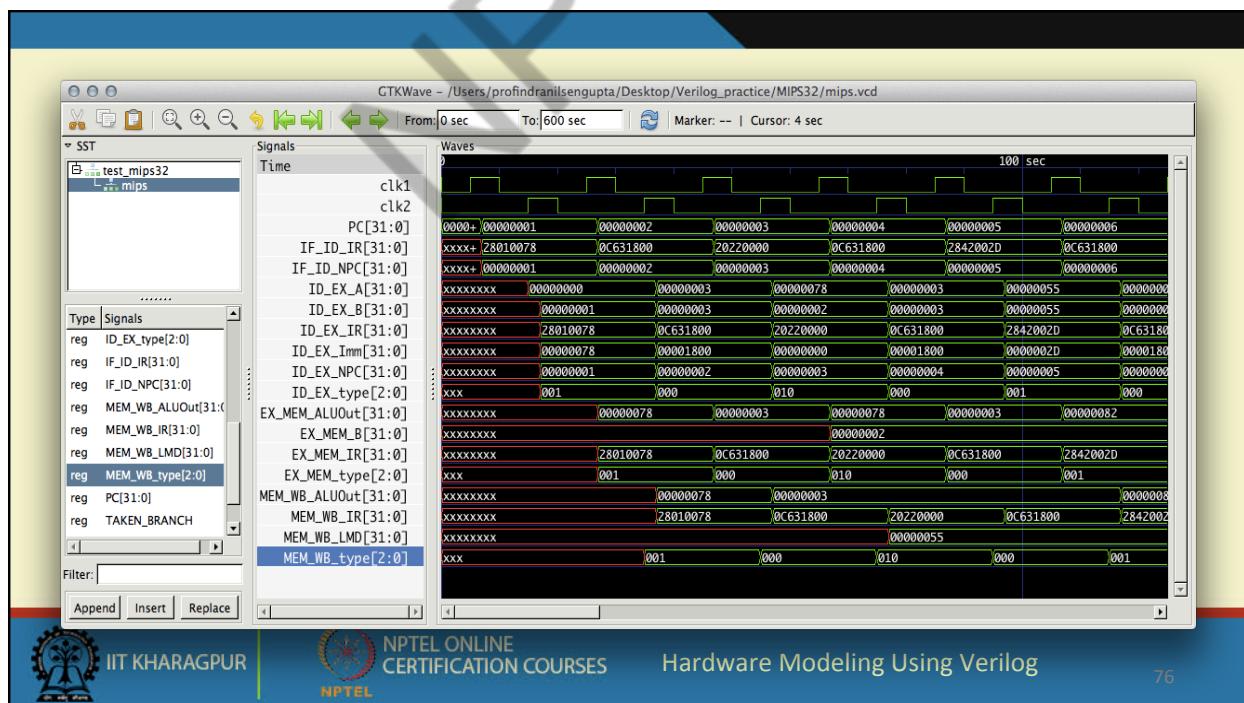
Mem[120]: 85
Mem[121]: 130



NPTEL ONLINE CERTIFICATION COURSES

Hardware Modeling Using Verilog

75



Example 3

- Compute the factorial of a number N stored in memory location 200. The result will be stored in memory location 198.
- The steps:
 - Initialize register R10 with the memory address 200.
 - Load the contents of memory location 200 into register R3.
 - Initialize register R2 with the value 1.
 - In a loop, multiply R2 and R3, and store the product in R2.
 - Decrement R3 by 1; if not zero repeat the loop.
 - Store the result (from R3) in memory location 198.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

77

Assembly Language Program		Machine Code (in Binary)
ADDI	R10,R0,200	001010 00000 01010 0000000011001000
ADDI	R2,R0,1	001010 00000 00010 0000000000000001
LW	R3,0(R10)	001000 01010 00011 0000000000000000
Loop:	MUL R2,R2,R3	000101 00010 00011 00010 00000 000000
SUBI	R3,R3,1	001011 00011 00011 0000000000000001
BNEQZ	R3,Loop	001101 00011 00000 1111111111111101
SW	R2,-2(R10)	001001 00011 01010 1111111111111110
	HLT	111111 00000 00000 00000 00000 000000



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

78

```

module test_mips32;

reg clk1, clk2;
integer k;

pipe_MIPS32 mips (clk1, clk2);

initial
begin
    clk1 = 0; clk2 = 0;
    repeat (50)                                // Generating two-phase clock
        begin
            #5 clk1 = 1; #5 clk1 = 0;
            #5 clk2 = 1; #5 clk2 = 0;
        end
    end
end

```

TEST BENCH



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

79

```

initial
begin
    for (k=0; k<31; k++)
        mips.Reg[k] = k;

    mips.Mem[0] = 32'h280a00c8; // ADDI R10,R0,200
    mips.Mem[1] = 32'h28020001; // ADDI R2,R0,1
    mips.Mem[2] = 32'h0e94a000; // OR R20,R20,R20 -- dummy instr.
    mips.Mem[3] = 32'h21430000; // LW R3,0(R10)
    mips.Mem[4] = 32'h0e94a000; // OR R20,R20,R20 -- dummy instr.
    mips.Mem[5] = 32'h14431000; // Loop: MUL R2,R2,R3
    mips.Mem[6] = 32'h2c630001; // SUBI R3,R3,1
    mips.Mem[7] = 32'h0e94a000; // OR R20,R20,R20 -- dummy instr.
    mips.Mem[8] = 32'h3460ffff; // BNEQZ R3,Loop (i.e. -4 offset)
    mips.Mem[9] = 32'h2542ffff; // SW R2,-2(R10)
    mips.Mem[10] = 32'hfc000000; // HLT

```



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

80

```

mips.Mem[200] = 7;      // Find factorial of 7

mips.PC = 0;
mips.HALTED = 0;
mips.TAKEN_BRANCH = 0;

#2000 $display ("Mem[200] = %2d, Mem[198] = %6d",
                  mips.Mem[200], mips.Mem[198]);
end

initial
begin
$dumpfile ("mips.vcd");
$dumpvars (0, test_mips32);
$monitor ("R2: %4d", mips.Reg[2]);
#3000 $finish;
end
endmodule

```



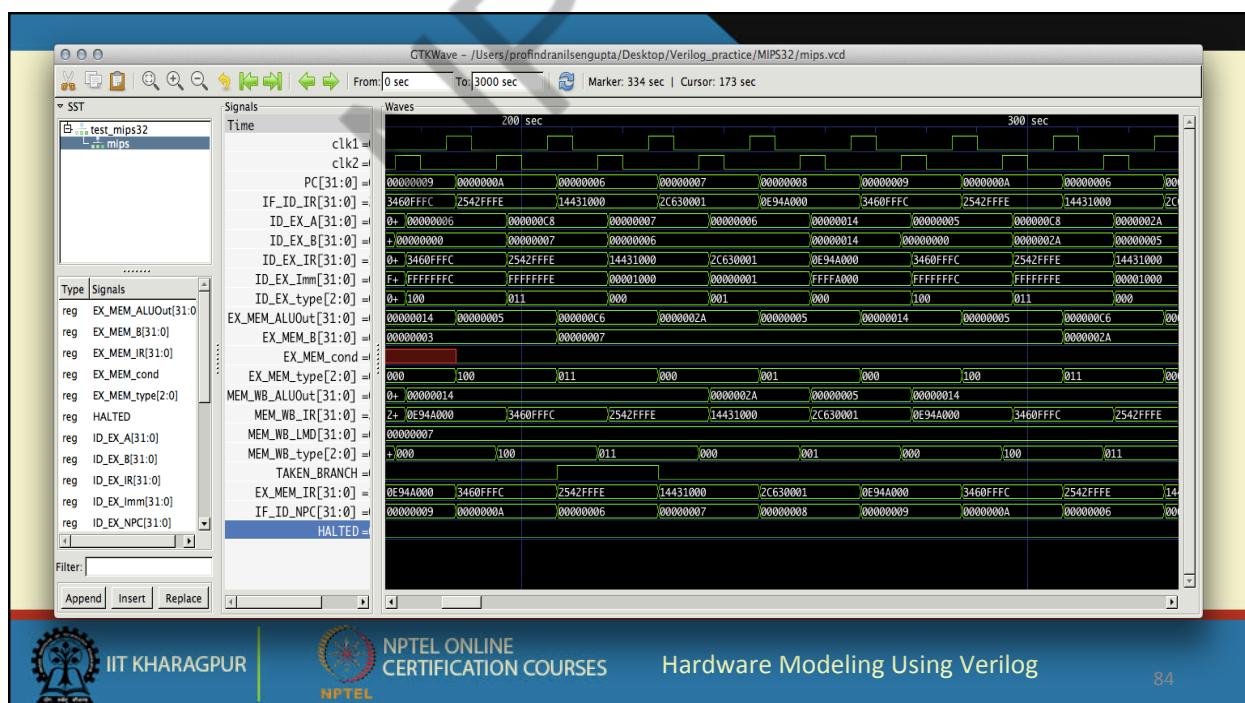
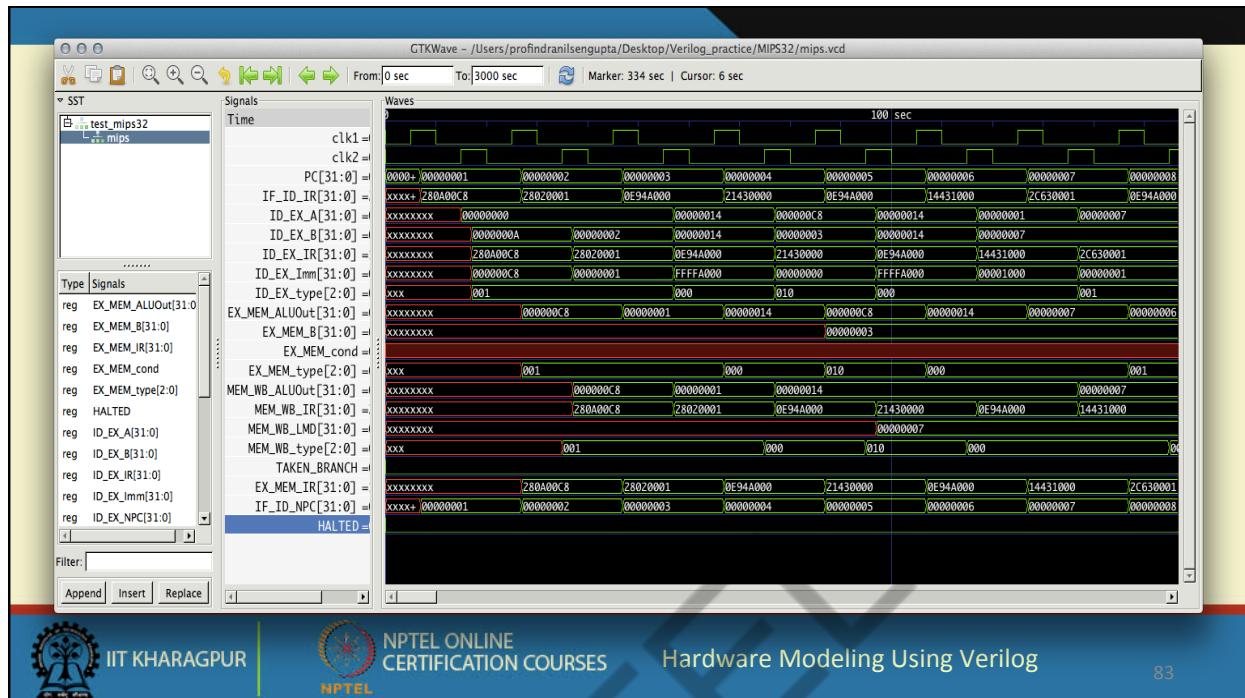
SIMULATION OUTPUT

```

R2:    2
R2:    1
R2:    7
R2:   42
R2:  210
R2:  840
R2: 2520
R2: 5040
R2: 5040
Mem[200] = 7, Mem[198] = 5040

```





Point to Note

- We have not considered the methods for avoiding hazards in pipelines.
- For the examples shown, we have inserted dummy instructions between dependent pairs of instructions.
 - So that data hazard does not lead to incorrect results.
- Also, we have modeled the processor using behavioral code.
 - In a real design where the target is to synthesize into hardware, structural design of the pipeline stages is usually used.
 - The Verilog code will be generating the control signals for the pipeline data path in the proper sequence.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

85

END OF LECTURE 41



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

86



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Lecture 42: SUMMARIZATION OF THE COURSE

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Topics Covered

- Basic introduction to Verilog language and features.
- Difference between behavioral and structural representations.
- Using continuous dataflow assignments in Verilog code.
- Modeling combinational and sequential circuits.
- Procedural blocks, blocking and non-blocking assignments.
- Writing test benches.



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

88

Other Topics Discussed

- Modeling finite state machines and controllers.
- Partitioning a design into data and control paths.
- User defined primitives.
- Switch level modeling.
- Basic concepts of pipelining.
- Pipelined design and implementation of a subset of the MIPS32 instruction set.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

89

What has not been covered!!

- Use of synthesis tools, for FPGA and/or ASIC implementations.
- Complex structural designs, as they are difficult to discuss on slides.
- Expertise comes from experience.
 - Designing for complex problems gives a lot more insight than a text book or a course can teach.



IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

90

END OF THE COURSE

THANK YOU FOR ATTENDING



IIT KHARAGPUR



NPTEL
NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

91