Lab 3
Instruction Memory and Data Memory

Fall 2008

低功率高效能VLSI設計實驗室
NCKU, Low power, high performance VLSI design lab

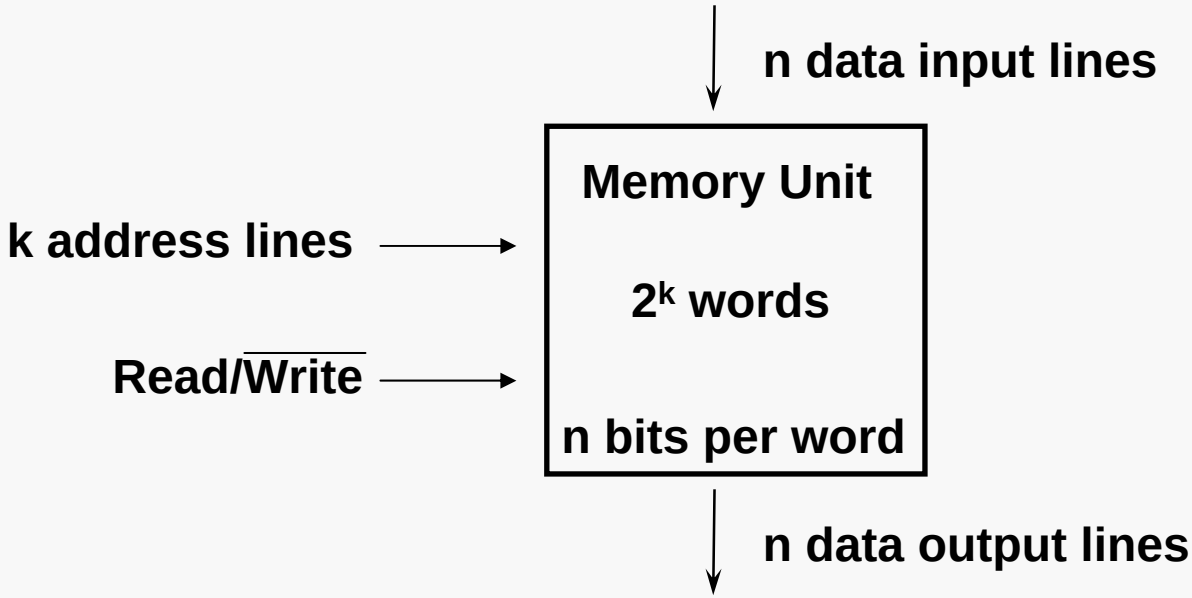
Lab Overview

- Target Design
  - Add instruction memory and data memory to the datapath+controller
- This ppt first reviews the memory and ways to use them in Verilog.
- You perform simulation to verify the correctness.
- Combine all components together to form a complete SMILE CPU, the controller may need to be modified.
- Detailed description for each step are in Lab3_handout.

Outline

- Memory Elements
- Loading Data into Memory
- Simulation Results
- Reference Codes

Memory Elements (1/3)

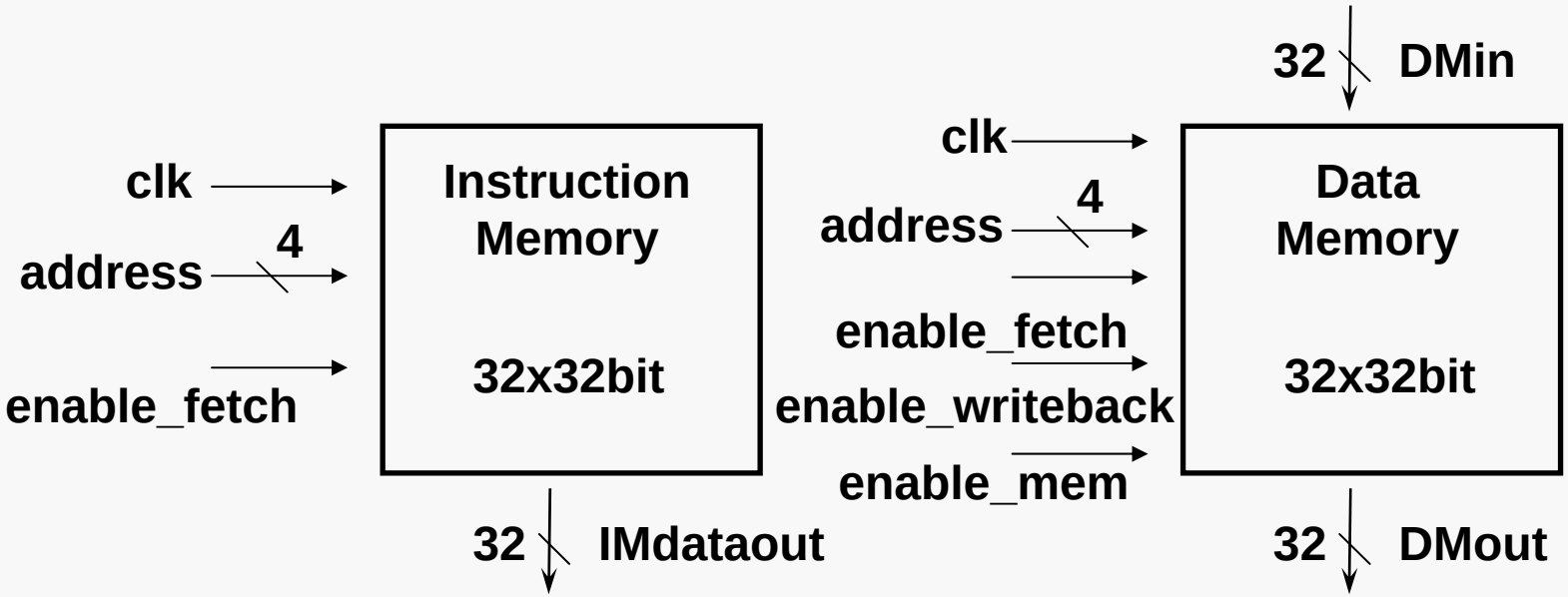


- Memory is a collection of binary storage cells together with associated circuits needed to transfer information
- During Read operation, a specific word is selected by applying k-bit address and the selected word will be put on Data Output
- During Write operation, the input address selects the memory location to be written with the data appear at the Data Input

Memory Elements (2/3)

- CPU consists of two memories, one for storage of instructions (Instruction Memory) and the other for storage of data (Data Memory)
- The *access time* of a memory read operation is the maximum time from the application of the address to the appearance of the Data Output
- The *write cycle time* is the maximum time from the application of the address to the completion of all internal memory operations required to store a word
- CPU must provide memory control signals to synchronize its own internal clock operations by devoting a fix number of clock pulses to each memory request
- SRAM vs DRAM
- Volatile memory vs Non-volatile memory

Memory Elements (3/3)




```
always@(negedge clk)
begin
  if(enable_fetch) begin
    IMdataout=mem_data1[address];
  end
end
```

```
always@(posedge clk)begin
  if( rst )begin
    for(i=0 ; i<memSize ; i=i+1 )
          mem_data[i]=0;
          DMout = 0;  end
  else if( enable_mem==1 ) begin
          if( enable_fetch==1 )
          DMout = mem_data[ADDR];
          else if( enable_writeback==1 )
          mem_data[ADDR] = DMin;  end
```

Loading Data into Memory (1/3)

- Test program “mins.prog” contains the operating instructions to be executed by CPU
- Convert the instructions into binary machine codes
- Added the following block into the testbench to load mins.prog into Instruction memory

```
initial
begin : prog_load
  $readmemb("mins.prog",u_memory_Instruction.mem_data1);
end
```

- The system task $readmemb (reads in binary) and $readmemh (reads in hex) will read in a file.
- It takes 2 arguments, the file name and the memory structure name. It reads the file into the memory structure.

Loading Data into Memory (2/3)

mins.prog content

- 0. LD R0 <=21
- 1. SW M0<=R0
- 2. SW M1<=2'hff
- 3. LD R1<= M1
- 4. LD R0<= M0
- 5. LD R1<=10
- 6. ADD R1=R1+R0;
- 7. SUB R1=R1-R0
- 8. AND R1=R0 & R1
- 9. OR R1=R0 || R1
- 10. LD R0<=7
- 11. XOR R0=R1^R0
- 12. ADD R0=R0+4'b1101
- 13. NOP
- 14. SUB R1=R1-5'b10000
- 15. SLL R0=R0 SLL(R1)
- 16. SRL R0=R0 SRL(R1)
- 17. NOP
- 18. LD R1<=5'b11100
- 19. RLL R0=R0 RLL(R0)
- 20. RRL M0=R0 RRL(3)
- 21. LD R2<= M0
- 22. ADD R2=R2+4'b1010

Loading Data into Memory (3/3)

- **Example of machine codes to be used in the lab session:**

- **0011_10__00000_0000_1101__00000_0000_0000 //ADD R0=R0+4'b1101**
- **0011_10__00000_0000_1100__00000_0000_0001 //ADD R1=R1+4'b1100**
- **0011_10__00000_0000_1000__00000_0000_0010 //ADD R2=R2+4'b1000**
- **0011_10__00000_0000_0100__00000_0000_0011 //ADD R3=R3+4'b0100**
- **0011_10__00000_0000_0010__00000_0000_0100 //ADD R4=R4+4'b0010**
- **0011_10__00000_0000_0001__00000_0000_0101 //ADD R5=R5+4'b0001**
- **0011_10__00000_0000_1001__00000_0000_0110 //ADD R6=R6+4'b1001**
- **0011_00__00000_0000_0000__00000_0000_0001 //ADD R1=R1+R0;**
- **0100_00__00000_0000_0000__00000_0000_0001 //SUB R1=R1-R0**
- **0101_00__00000_0000_0000__00000_0000_0001 //AND R1=R0 & R1**

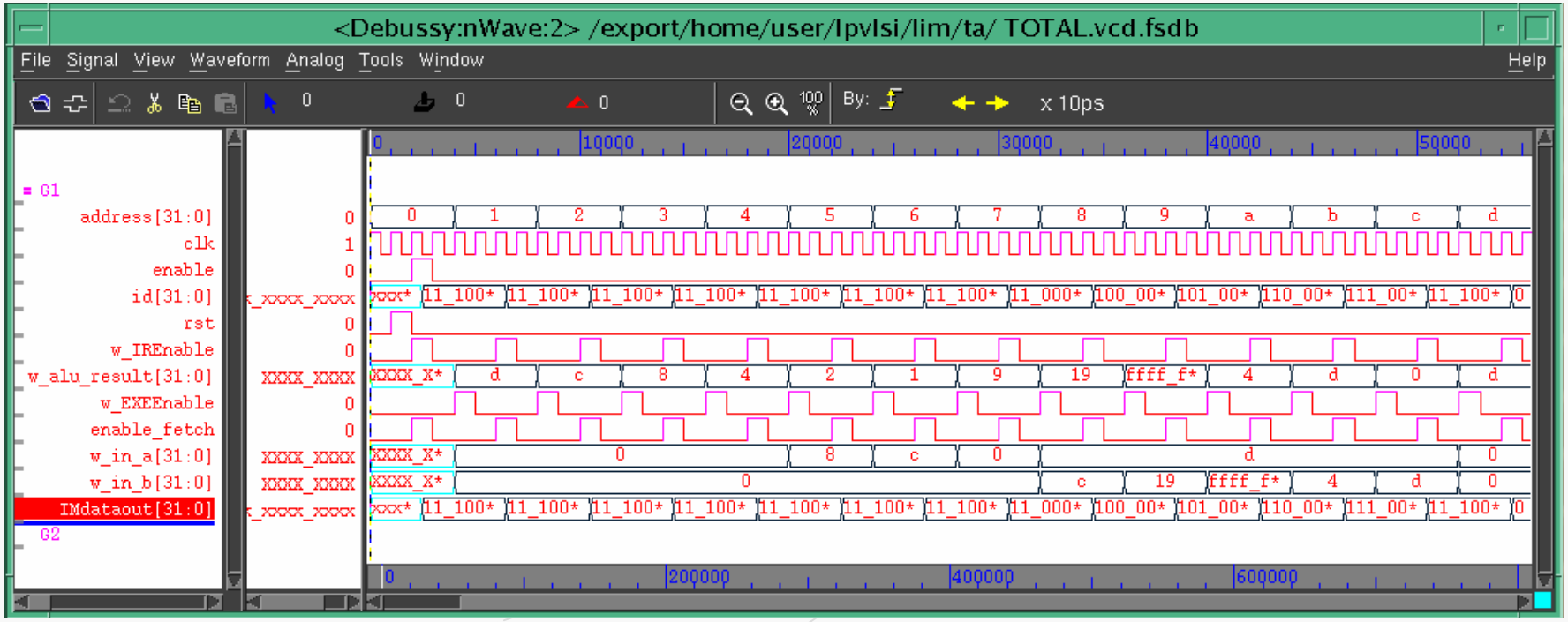
…

…

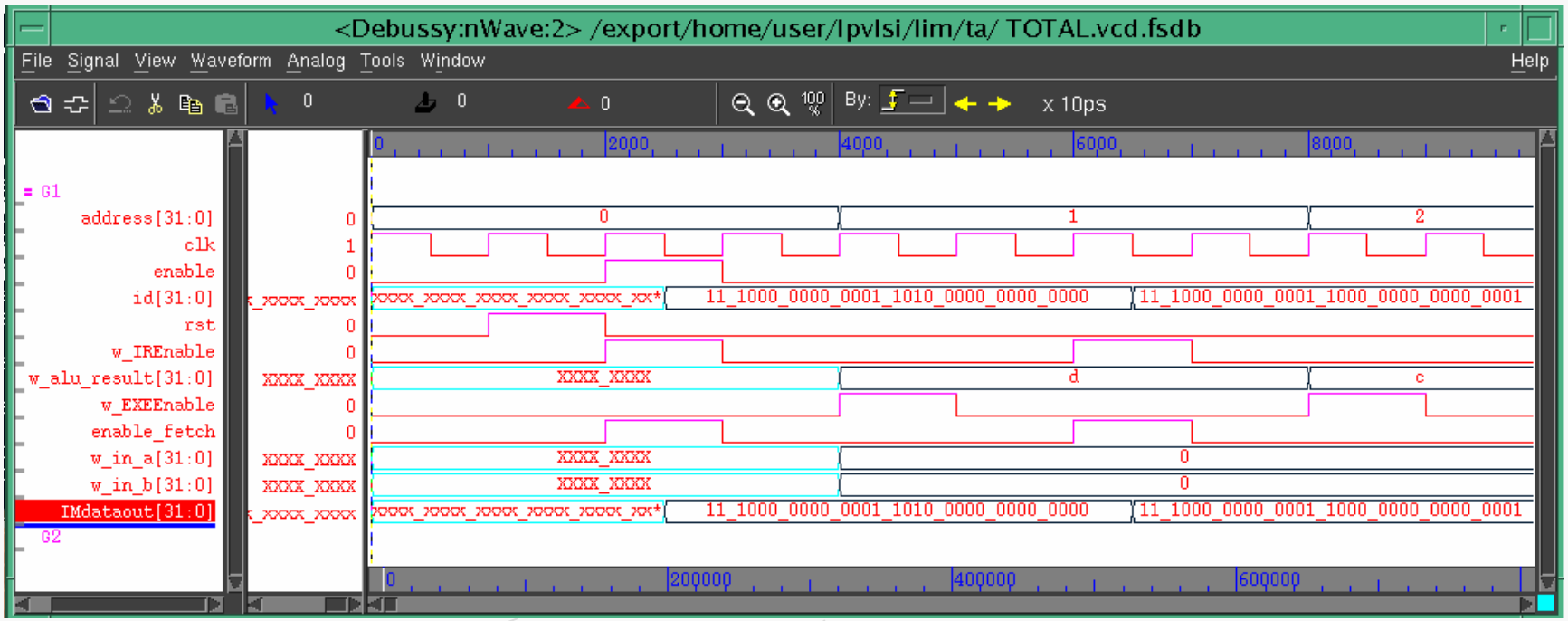
…

…

Simulation Results (1/2)



Simulation Results (2/2)



Reference Code (Instruction Memory)

```
module Instruction_memory (clk, address, IMdataout, enable_fetch);
input   clk,    enable_fetch;
input   [4:0]   address; //from PCounter
output  [31:0]  IMdataout;

wire    [4:0]   address;
reg     [31:0]  mem_data1[31:0];
reg     [31:0]  IMdataout;

always@(negedge clk)
begin
  if(enable_fetch) begin
    IMdataout=mem_data1[address];
  end
end
endmodule
```

Reference Code (Data Memory)

```
module memory_Data (clk, rst, id, enable_mem, enable_fetch, enable_writeback,
                DMin, DMout, ADDR);

parameter memSize = 32;
parameter DataSize = 32;

input clk;
input rst;
input enable_mem;
input enable_fetch;
input enable_writeback;
input  [DataSize-1:0]   id;
input  [DataSize-1:0]   DMin;
input  [DataSize-1:0]   ADDR;
output [DataSize-1:0]   DMout;
reg    [DataSize-1:0]   DMout;
reg    [DataSize-1:0]   mem_data[memSize-1:0],                    //for 32*32

integer i;
always@(posedge clk)begin
  if( rst )begin
     for(i=0 ; i<memSize ; i=i+1 )
                mem_data[i]=0;
                DMout = 0;
                end
  else if( enable_mem==1 ) begin
                if( enable_fetch==1 )
                DMout = mem_data[ADDR];
                else if( enable_writeback==1 )
                mem_data[ADDR] = DMin;
                end
end
endmodule
```