Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology

– Computer Architecture – Semester 182 –

VERILOG-BASED SINGLE-CYCLE MIPS PROCESSOR

No.	Name	Student's ID
1	Nguyen Minh Dang	1752170
3	Le Nguyen An Khuong	1752305
4	Nhan Ngoc Thien	1752508

Instructor: Dr. Pham Quoc Cuong

Contents

1 Introduction

The report is about designing a MIPS Single Clock Cycle using Verilog HDL. The report involves using De2i board to simulate the result. The MIPS Single Clock Cycle processor is constructed by implementing 32-bit 32 registers, and 20 memory locations- each is 32 bit wide - (due to limitation of resources). The result is presented on De2i board by 3 methods: 26 leds, 7-segment leds and LCD.

2 Design

2.1 Datapath

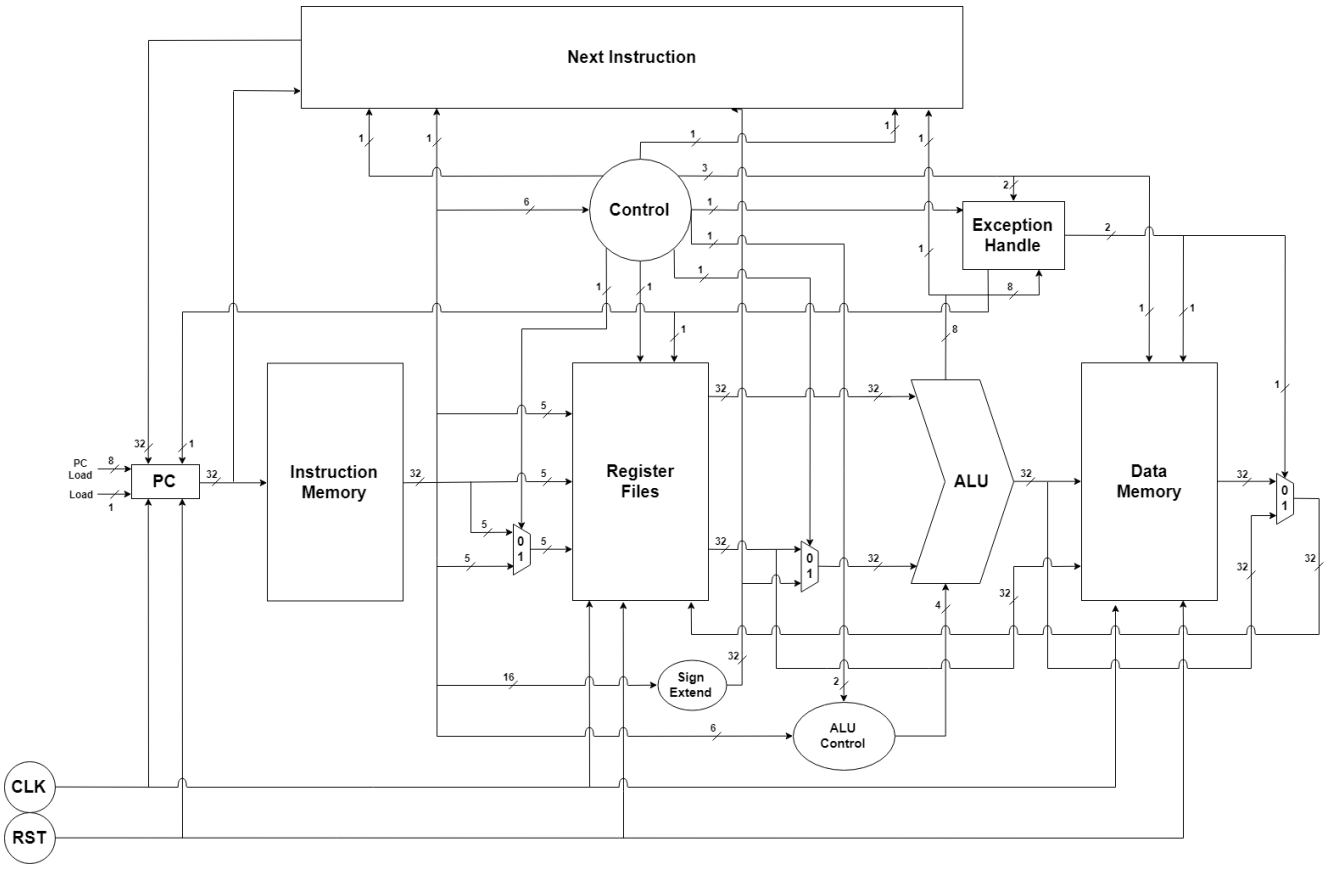

Figure 1: Single Cycle Full Datapath

2.2 Block Description

2.2.1 Module SYS Master

• Top-Level Entity module: connect all other modules: Instruction Memories, Register Files, ALU, Data Memory, Control Unit,...

- Display 26 bits low of output on leds, display 32 bits of output on 7-segment leds and display 32 bits output, PC, and 8 bits of SYS_output_sel on LCD.
- SYS_clk works as clock triggered to execute instructions.
- SYS_load is used to load 8 bits low of PC to demo the wanted instructions.
- SYS_output_sel is describe as following:
    + 0: Instructions Memory output is displayed.
    + 1: output-1 of Register Files is displayed.
    + 2: ALU result is displayed.
    + 3: ALU status is displayed.
    + 4: Data Memory result is displayed.
    + 5: Control Unit output is displayed.
    + 6: ALU Control output is displayed.
    + 7: Next PC address is displayed.
    + 8: output-2 of Register Files is displayed.
    + 9: Exception Handle output is displayed.
    + 10: Enable signal from Exception Handle is displayed.
    + 11: Error signal from Register Files is displayed.

2.2.2 Instruction Memory

● Store the machine code of the program.
● Receive the address of the current instruction and transfer the instruction to other block for execution.

2.2.3 Module Register Files

● Receive the address of the instruction and decode the instruction of the register (there are 32 32-bit registers). Output is the value of the register.
● If there is write signal, the input value will be written into the destination register.
● If the write register is $0 then the error signal will be active and send signal to Exception Handle module.

2.2.4 Module ALU

● Execute operations (add, addi, sub, and, or, slt, branch, beq, jump,...).
● Output is the result of the calculation and status signal if there are any exception (divided by 0, overflow,...)
● ALU status is displayed as followed:

Name	Bit	Description	Exception
Zero	7	Active when ALU result equals zero	
Overflow	6	Active when ALU result overflows	x
Carry	5	Active when ALU result has carry bit	
Negative	4	Active when ALU result is negative	
Invalid Address	3	Active when ALU result is not aligned	x
Undefined	[2:0]	No description	

2.2.5 Module Data Memory

• Create memory space: 20 memory locations, each is 32 wide bit due to the resources limitation.
• Access and write into the input location of the memory.
• When exception occurs, Data Memory hold, means that the module will not access and write into memory location.

2.2.6 Control Unit

• Receive the Operation bit from Instruction Memory to transfer the signal for other blocks to execute the given instruction.
• The output of Control Unit is described as followed:

Name	Bit
Jump	10
Branch	9
MemRead	8
MemWrite	7
Mem2Reg	6
ALUOp	[5:4]
Exception	3
ALUsrc	2
RegWrite	1
RegDst	0

2.2.7 Module PC

• Output is the current executed instruction. Input is the next instruction to be executed. PC can load an instruction for the purpose of testing.
• If an exception occurs, PC won't be directed to the next instruction.

2.2.8 Next Instruction

• Receive three set of bits: current address of PC, Sign-extend output and 26-jump bits and three control signal: Jump and Branch and ALU status, which is implemented as following:
• Output is the address for PC to point to in the next clock cycle.

2.2.9 Module ALU Control

• Receive function from Instruction Memory and ALU OpCode from the Control Unit.
• Giving ALU block the compatible signal to perform the equivalent arithmetic operation.

2.2.10 Module Exception Handle

• Receive 8 bit of ALU status, 4 bit: Exception, MemRead, MemWrite, MemRead, Mem2Reg from Control Unit, and 1 bit from Register Files. Output is the enable signal which is active low when an exception occur.
• Exceptions need to be handled:
    - Instruction syntax error.
    - Hardware error (write on register $zero, unaligned).
    - Arithmetic exception (overflow).
• The system will enable block the Mem2Reg and MemWrite signal to prevent output error, and will pause the whole system until reset signal is active.

2.2.11 Module Mux

• Selection module with two inputs.

2.2.12 Module Sign Extend

• Input is an 16-bit integer, 5 bits shamt, output is an 32-bit extended integer number.

2.2.13 Module LCD

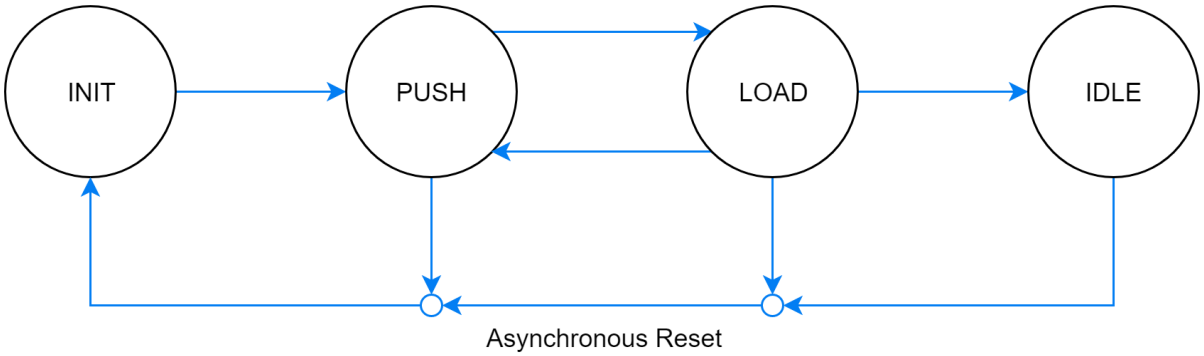

Figure 2: LCD Finite State Machine

• Manipulate the LCD to display necessary information of the system.
• The LCD works by receiving and processing instructions from user. This module automates the process of sending the instructions into the LCD.
• The figure above demonstrate the LCD's Finite State Machine
  • **INIT:** In this state the module initialized setup instructions to push into the LCD.
  • **PUSH:** In this state the module push instruction into the LCD (one instruction per time)
  • **LOAD:** In this state the module load the next instruction to be pushed.
  • **IDLE:** After pushing all the instructions, the module goes to IDLE state, which it will do nothing and wait for next set of instructions.

Finally, all state can transit to INIT state with an Asynchronous reset signal.

3 Implementation

3.1 Prototype

3.1.1 Module SYS-Master

```
module SYS_Master(
  output [26:0] SYS_leds,
  output [0:6] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7,
  output [7:0] LCD_DATA,
  output    LCD_RW,LCD_EN,LCD_RS,
  input SYS_clk,
  input SYS_reset,
  input SYS_load,
  input [7:0] SYS_pc_load,
  input [7:0] SYS_output_sel,
  input CLOCK_50, SW
);
```

3.1.2 Module Instructions Memory

```
module inst_mem(
  output [31:0] imem_instruction,
  input  reset,
  input  [31:0] imem_pc
);
```

3.1.3 Module Register Files

```
module register_files(
  output [31:0] read_data_1,
  output [31:0] read_data_2,
  output error_toggle,
  input [4:0]   read_register_1,
  input [4:0]   read_register_2,
  input [4:0]   write_register,
  input         write_switch,
  input [31:0] write_data,
  input clk, reset, enable
);
```

3.1.4 Module ALU

```
module alu(
  output [31:0] alu_result,
  output [7:0] alu_status,
  input signed [31:0] alu_operand_1, alu_operand_2,
  input  [3:0] alu_control
);
```

3.1.5 Module Data Memory

```
module data_memory(
  output [31:0] data_out,
  input[31:0] address,
  input[31:0] data_in,
  input mem_read, mem_write, clk, reset
);
```

3.1.6 Module Control Unit

```
module control_unit(
  output [10:0] control_signal,
  input [5:0] opcode
);
```

3.1.7 Module LCD

```
module lcd(
  input CLOCK,
  input RST,
  input CLK,
  input LOAD,
  input SW,
  input [31:0] DATA,
  input [31:0] PC,
  input [7:0]  SEL,
  output reg LCD_EN,
  output reg LCD_RS,
  output LCD_RW,
  output reg [7:0] LCD_DATA
);
```

4 Result and Simulation

The MIPS Single Clock Cycle is able to execute basic operations: add, addi, sub, or, and, slt, beq, j, lw, sw. The output of each module can be selected and is display on 26 leds, 7-segment leds and LCD.

4.1 List of instructions

$$
\begin{cases}
addi & \$t0, & \$t0, & 10 \\
addi & \$t1, & \$0, & 5 \\
add & \$t2, & \$t1, & \$t0 \\
add & \$t2, & \$t2, & \$t2 \\
sub & \$t3, & \$t1, & \$t2 \\
slt & \$t4, & \$t3, & \$t2 \\
and & \$t4, & \$t1, & \$t3 \\
or & \$t5, & \$t2, & \$t4 \\
sub & \$t5, & \$t5, & \$t4 \\
j & \text{0xFFFFFF6} &&
\end{cases}
\qquad
\begin{cases}
addi & \$0, & \$0, & 0 \\
\text{Unknown} & \text{instruction} &&\\
lw & \$t1, 2 & \$(t0) &\\
sw & \$t1, 2 & \$(t0) &\\
addi & \$t0, & \$0, & 4 \\
addi & \$t1, & \$0, & 4 \\
sw & \$t0, 0 & \$(t0) &\\
lw & \$t3, 0 & \$(t0) &\\
add & \$t3, & \$t3, & \$t3 \\
beq & \$t0, & \$t1, & \text{0xFFFA}
\end{cases}
$$

4.2 Simulation Waveform

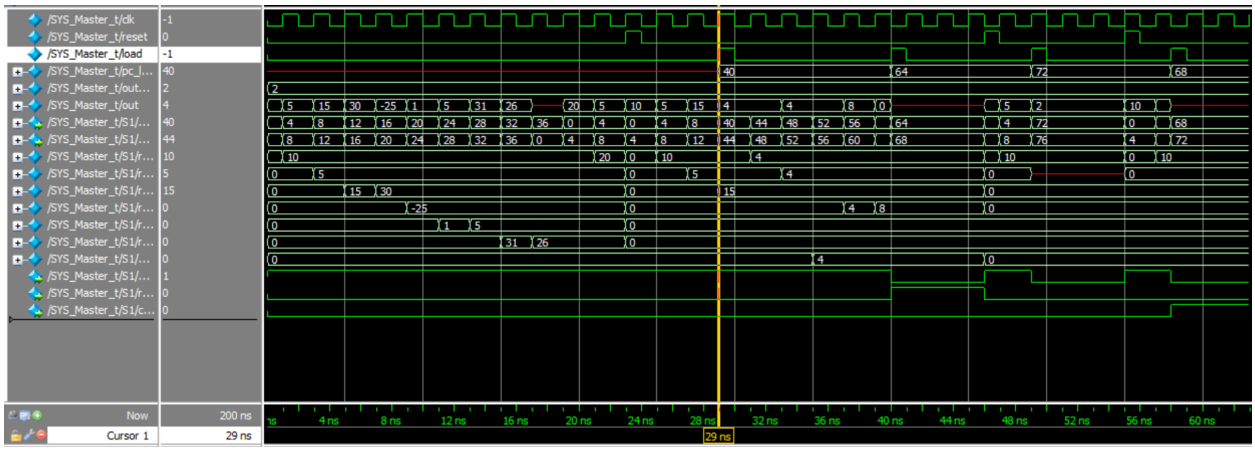

Figure 3: Waveform Simulation

4.3 Simulation On De2i

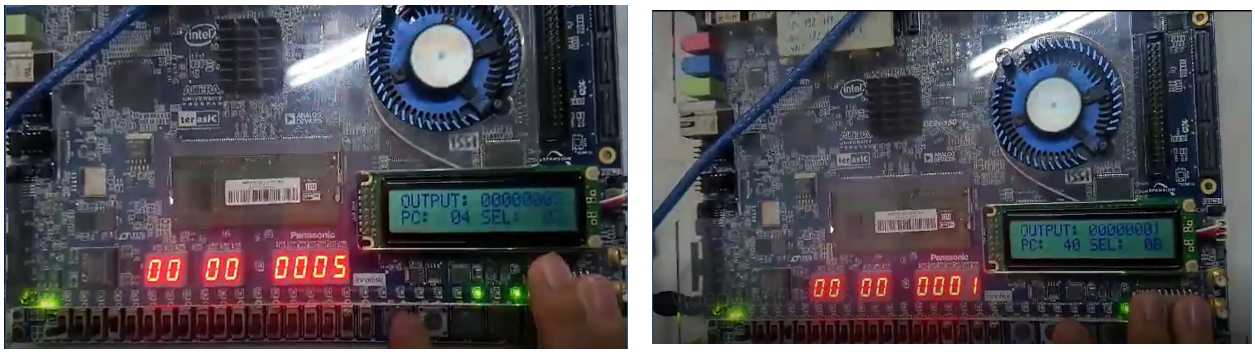

Figure 4: Simulation on De2i

5 Conclusions

The project provide the opportunity to comprehend the function of a normal MIPS instruction, study deep into how the computer when each line of code is compiled; a long with that is the chance of practicing with Verilog. This project gives us the base knowledge for future programming as well as much greater project.