ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÁO CÁO MILESTONE2

MÔN: KIẾN TRÚC MÁY TÍNH

Design of a Single Cycle RISC-V Processor

Giảng viên hướng dẫn:

Trần Hoàng Linh - Nguyễn Trung Hiếu

Lớp: L01_L02

Nhóm: 19

Sinh viên thực hiện:

STT	Họ tên	Mã số sinh viên
1	Phan Bảo Huân	2311114
2	Trần Chí Nhân	2312450
3	Nguyễn Trung Hiếu	2310964

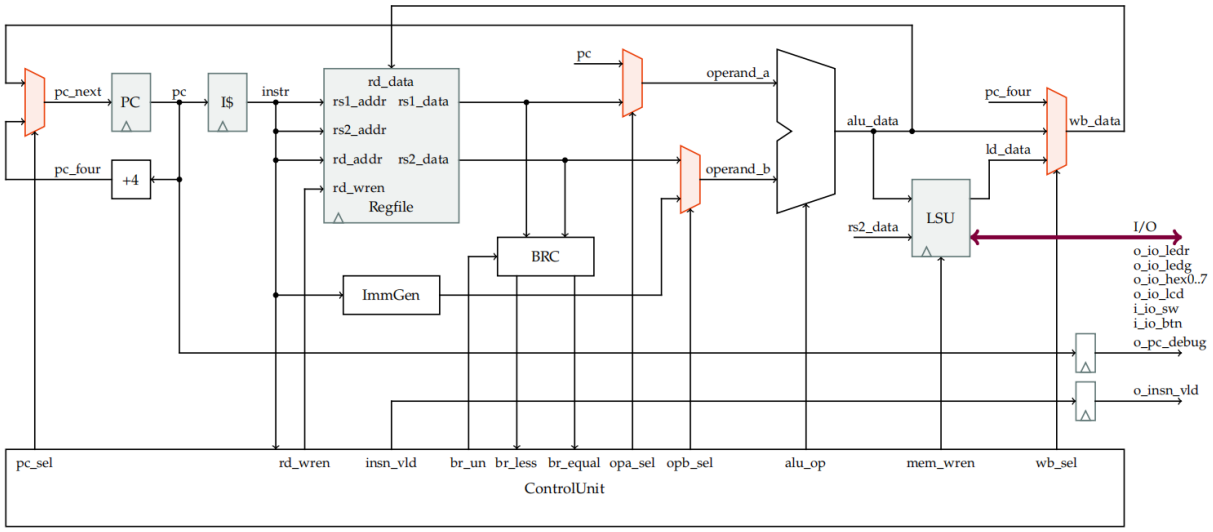
Thành phố Hồ Chí Minh – 10/2025

MỤC LỤC

DANH MỤC HÌNH ẢNH

I. Giới thiệu

Design of a Single Cycle RISC-V Processor



Hình 1: Single Cycle Processor

Báo cáo thí nghiệm này trình bày về thiết kế của một bộ xử lý đơn chu kỳ thực hiện ISA RV32I và có thể được tổng hợp trên FPGA. Bộ xử lý có thể triển khai các tập lệnh trong RV32I ngoại trừ các lệnh FENCE, ngoài ra còn có khả năng giao tiếp với các thiết bị ngoại vi như LED, LCD, LED 7 đoạn và Switch. Bộ xử lý sử dụng hai khối bộ nhớ có phần tử logic giống nhau: một khối read - only cho instruction memory, một khối read - write cho data memory.

II. Thiết kế

1. ALU và BRC

❖ Phân tích yêu cầu thiết kế ALU

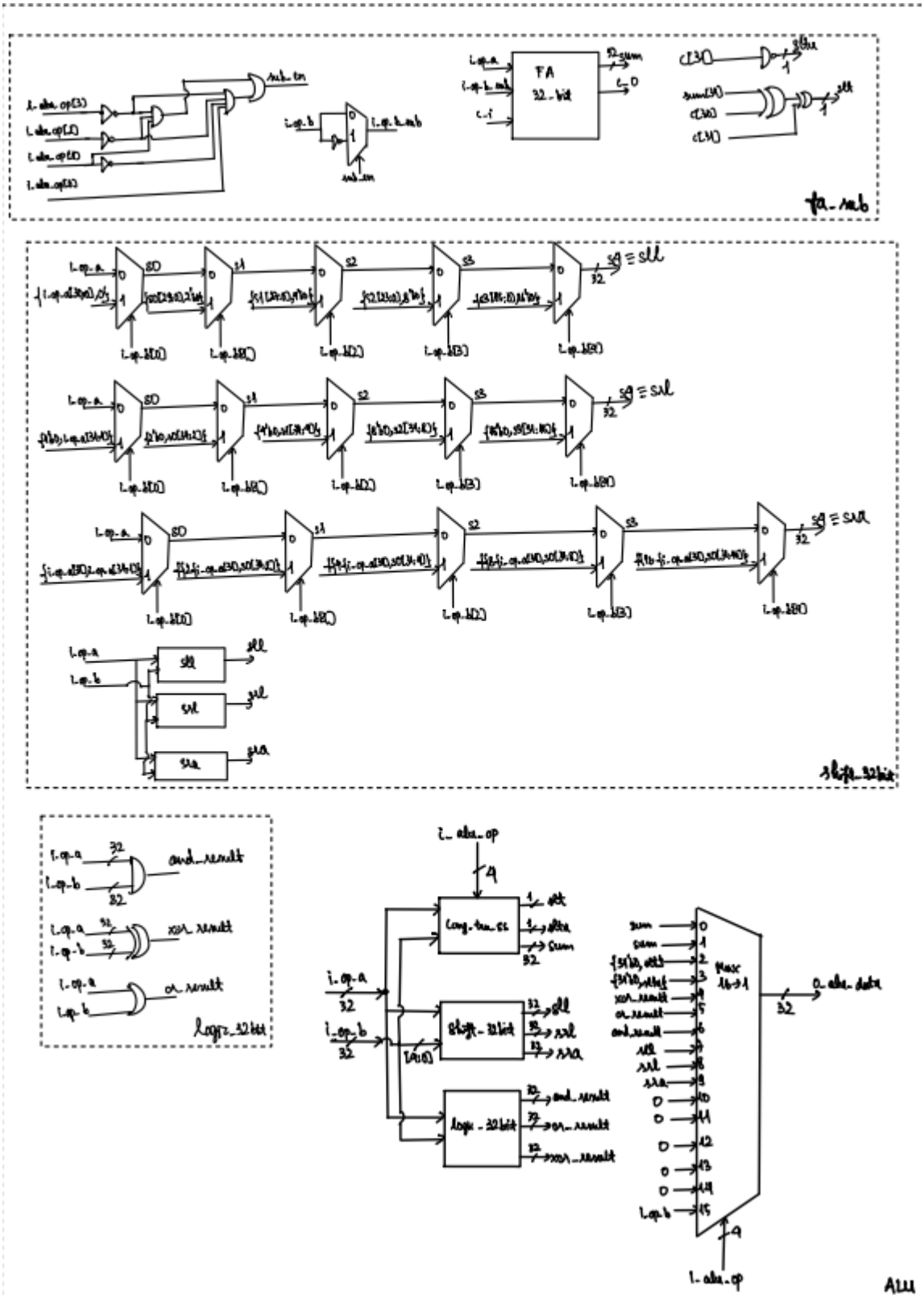
- Bộ ALU (Arithmetic Logic Unit) thực hiện các phép tính số học và logic trong tập lệnh RV32I. Bộ ALU nhận dữ liệu đầu vào từ thanh ghi, thực hiện các phép tính toán theo tín hiệu và trả ra kết quả.

- Các phép toán cần thực hiện:

alu_op	Description (R-type)	Description (I-type)
ADD	$rd \leftarrow rs1 + rs2$	$rd \leftarrow rs1 + imm$
SUB	$rd \leftarrow rs1 - rs2$	n/a
SLT	$rd \leftarrow (rs1 < rs2)?\ 1 : 0$	$rd \leftarrow (rs1 < imm)?\ 1 : 0$
SLTU	$rd \leftarrow (rs1 < rs2)?\ 1 : 0$	$rd \leftarrow (rs1 < imm)?\ 1 : 0$
XOR	$rd \leftarrow rs1 \oplus rs2$	$rd \leftarrow rs1 \oplus imm$
OR	$rd \leftarrow rs1 \vee rs2$	$rd \leftarrow rs1 \vee imm$
AND	$rd \leftarrow rs1 \wedge rs2$	$rd \leftarrow rs1 \wedge imm$
SLL	$rd \leftarrow rs1 \ll rs2[4:0]$	$rd \leftarrow rs1 \ll imm[4:0]$
SRL	$rd \leftarrow rs1 \gg rs2[4:0]$	$rd \leftarrow rs1 \gg imm[4:0]$
SRA	$rd \leftarrow rs1 \ggg rs2[4:0]$	$rd \leftarrow rs1 \ggg imm[4:0]$

Hình 2: Các phép toán của ALU

❖ Sơ đồ khối ALU



Hình 3: Sơ đồ khối ALU

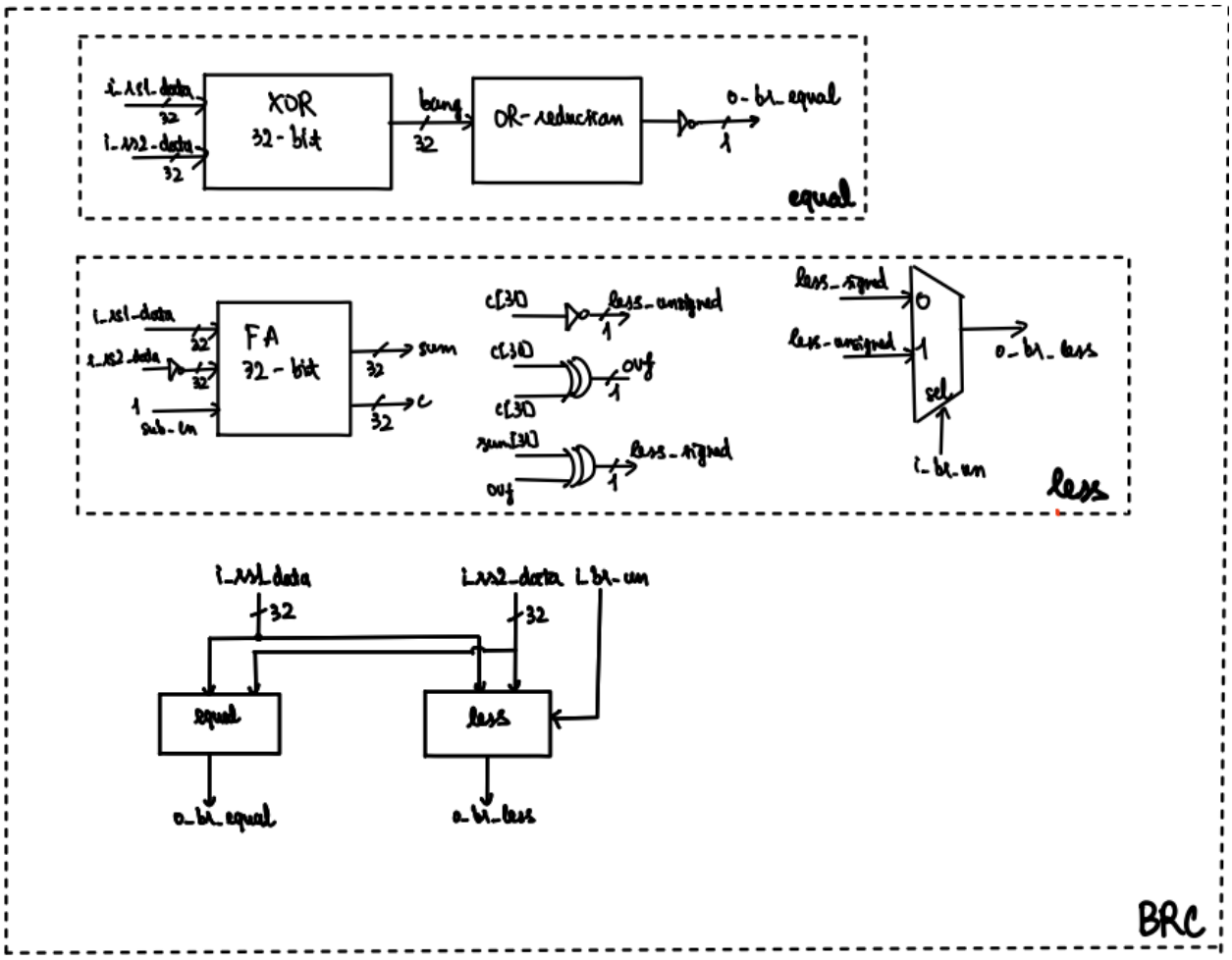
❖ ALU Specification

Signal name	Width	Direction	Description
i_op_a	32	input	First operand for ALU operations.
i_op_b	32	input	Second operand for ALU operations.
i_alu_op	4	input	The operation to be performed.
o_alu_data	32	output	Result of the ALU operation.

❖ Phân tích yêu cầu thiết kế BRC

- Bộ BRC (Branch Comparison) được thiết kế để thực hiện so sánh giá trị của hai thanh ghi i_rs1_data và i_rs2_data để xác định kết quả của các lệnh rẽ nhánh. Có hai đầu ra br_less nếu rs1 < rs2 (nếu o_br_less = 1 thì ra kết quả như trên, nếu o_br_less = 0 thì ngược lại rs1 > = rs2) và br_equal nếu rs1 = rs2.
- Nếu tín hiệu i_br_un = 0, thực hiện so sánh không dấu (unsigned). Ngược lại, nếu i_br_un = 1, thực hiện so sánh có dấu (signed).
- Khi thiết kế bộ BRC không được dùng các toán tử như phép trừ (-), phép so sánh (<, >), phép dịch (<<, >>, >>>), phép nhân (*), phép chia (/) và phép lấy dư (%).

❖ **Sơ đồ khối BRC**



Hình 4: Sơ đồ khối BRC

❖ **BRC Specification**

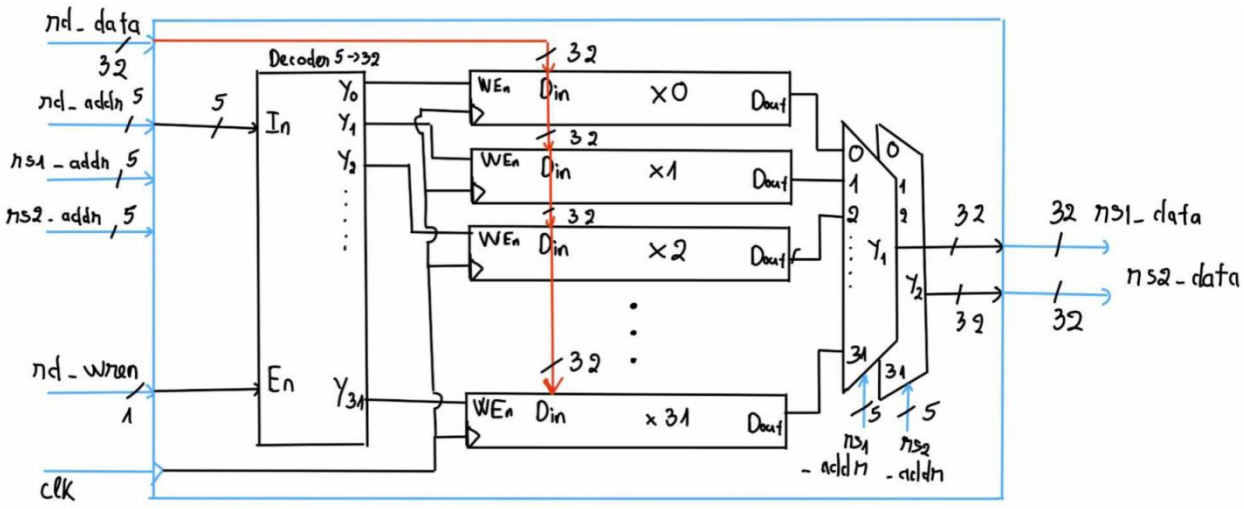
Signal name	Width	Direction	Description
`i_rs1_data`	32	input	Data from the first register.
`i_rs2_data`	32	input	Data from the second register.
`i_br_un`	1	input	Comparison mode (1 if signed, 0 if unsigned).
`o_br_less`	1	output	Output is 1 if $rs1 < rs2$.
`o_br_equal`	1	output	Output is 1 if $rs1 = rs2$.

2. Regfile, Memory và LSU

❖ Phân tích yêu cầu thiết kế Regfile

- Thiết kế một tệp 32 thanh ghi, mỗi thanh ghi có độ rộng 32 bit trong đó thanh ghi 0 luôn có giá trị 0. Tệp thanh ghi này có hai cổng đọc và một cổng ghi. Regfile này có chức năng trả ra data khi ta đưa vào địa chỉ từ rs1 và rs2.
- Nhóm thiết kế tệp thanh ghi gồm một decoder 32 sang 1, 32 thanh ghi 32 bit và hai mux 2 sang 1. Khi ghi thì đồng bộ, dữ liệu được trả về chờ sẵn ngoài khối regfile khi có xung clk thì dữ liệu được ghi vào regfile; còn đọc thì không cần tín hiệu xung clk, khi nào i_rd_wren = 0 thì regfile sẽ tự động đọc dữ liệu từ thanh ghi.

❖ Sơ đồ khối Regfile:



Hình 5: Sơ đồ khối Regfile

❖ Regfile Specification

Signal name	Width	Direction	Description
i_clk	1	input	Global clock.
i_reset	1	input	Global active reset.
i_rs1_addr	5	input	Address of the first source register.
i_rs2_addr	5	input	Address of the second source register.
o_rs1_data	32	output	Data from the first source register.
o_rs2_data	32	output	Data from the second source register.
i_rd_addr	5	input	Address of the destination register.
i_rd_data	32	input	Data to write to the destination register.
i_rd_wren	1	input	Write enable for the destination register.

❖ Phân tích yêu cầu thiết kế Memory

- Thiết kế Memory trong đó có hai phần là Data memory có dung lượng 2kB, Instruction memory có dung lượng 8kB; Data Memory nằm trong khối LSU để đọc/ghi và kết nối với các ngoại vi. Instruction memory dùng để đọc mã máy (instruction) tại địa chỉ mà PC gửi vào.

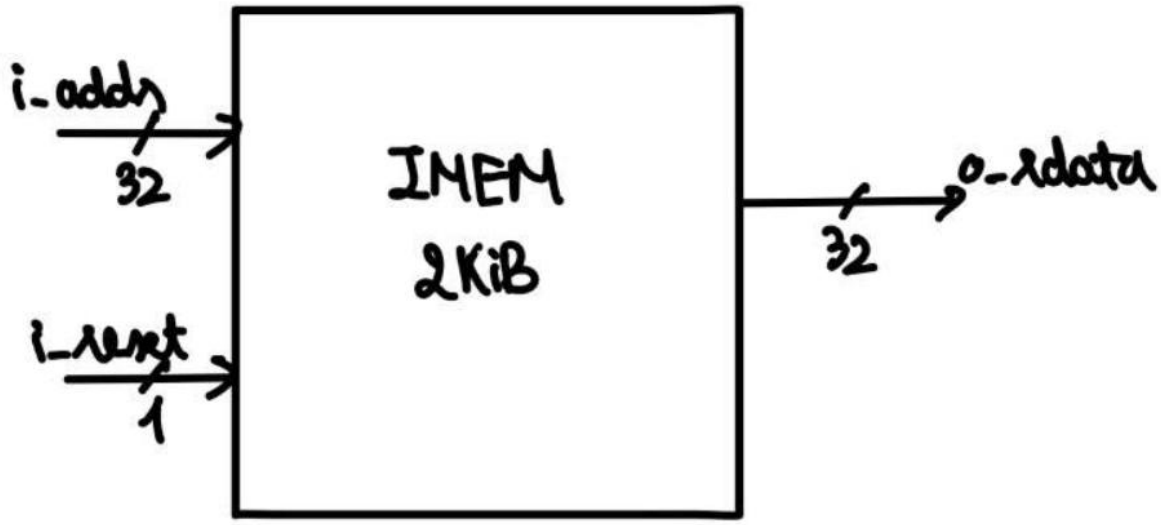
- Nhóm dự định thiết kế khối Instruction Memory (8KB), mỗi ô nhớ có 8 bit. Vì PC xuất dữ liệu là địa chỉ byte (mỗi lần giá trị đếm PC tăng thêm 4) nên khối Instruction memory phải đáp ứng với mỗi địa chỉ từ PC (địa chỉ byte) phải xuất ra được một mã máy 32 bit (word).

- Nhóm khai báo Instruction memory là mảng 2 chiều gồm 2048 word với độ rộng mỗi word là 32 bit. Muốn chuyển từ địa chỉ byte của PC qua địa chỉ word, ta bỏ đi 2 bit của của địa chỉ đó bằng cách dịch phải hai lần.

❖ Memory Specification

Signal name	Width	Direction	Description
i_clk	1	input	Global clock.
i_reset	1	input	Global active reset.
i_addr	32	input	Address for both read and write operations.
o_rdata	32	output	Instruction

❖ Sơ đồ khối Memory



Hình 6: Sơ đồ khối Memory

❖ Phân tích yêu cầu thiết kế LSU

- Nhóm thiết kế khối LSU (Load - Store Unit) gồm:

+ Bao gồm: bộ nhớ Data memory (2kB), Input Buffer (1 thanh ghi), Output Buffer (11 thanh ghi).

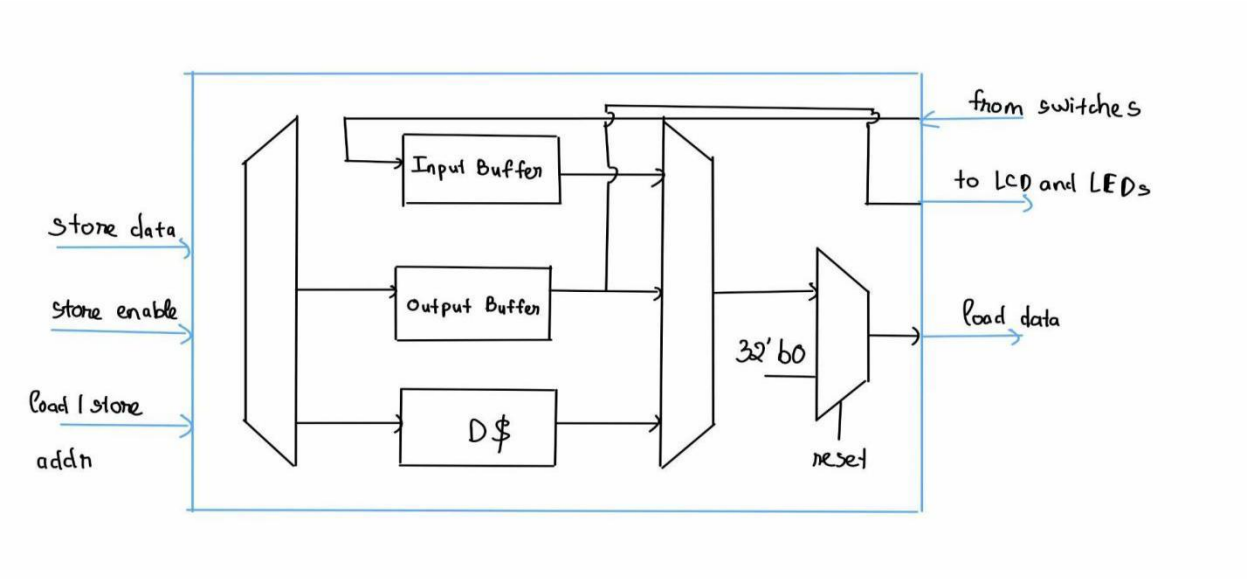
+ Data memory có chức năng đọc/ghi dữ liệu, Output Buffer ghi dữ liệu (store) vào Led, Lcd, Led 7 đoạn, còn Input Buffer đọc dữ liệu từ Switches.

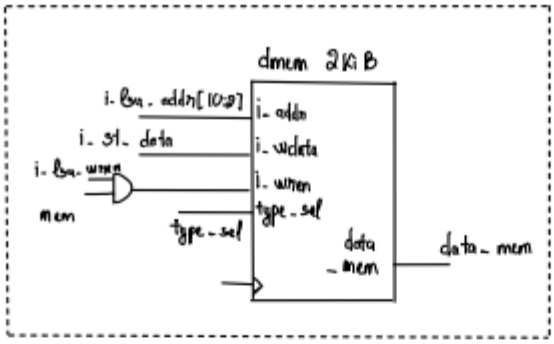
- Nhóm sử dụng mảng 2 chiều để khai báo Data memory gồm 512 word với mỗi word là 32 bit.

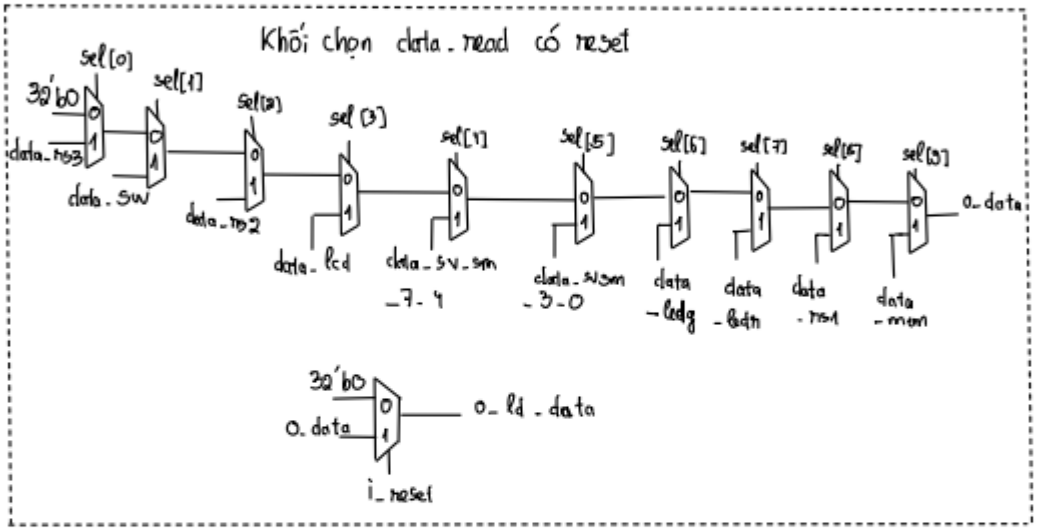
❖ LSU Specification

Signal name	Width	Direction	Description
i_clk	1	input	Global clock, active on the rising edge.
i_reset	1	input	Global active reset.
i_lsu_addr	32	input	Address for data read/write.
i_st_data	32	input	Data to be stored.
i_lsu_wren	1	input	Write enable signal (1 if writing).
o_ld_data	32	output	Data read from memory.
o_io_ledr	32	output	Output for red LEDs.
o_io_ledg	32	output	Output for green LEDs.
o_io_hex0..7	7	output	Output for 7-segment displays.
o_io_lcd	32	output	Output for the LCD register.
i_io_sw	32	input	Input for switches.
type_sel	2	input	Sel type store/load

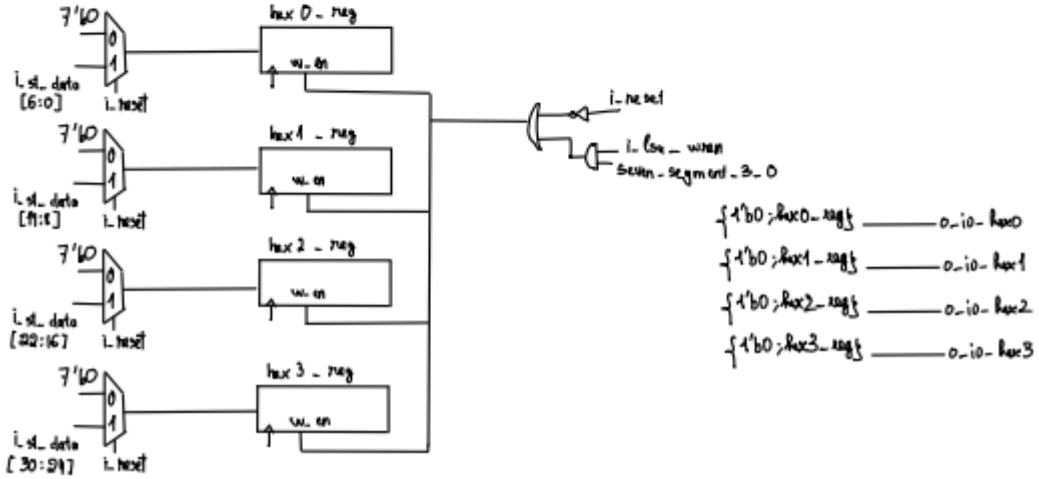
❖ Sơ đồ khối LSU

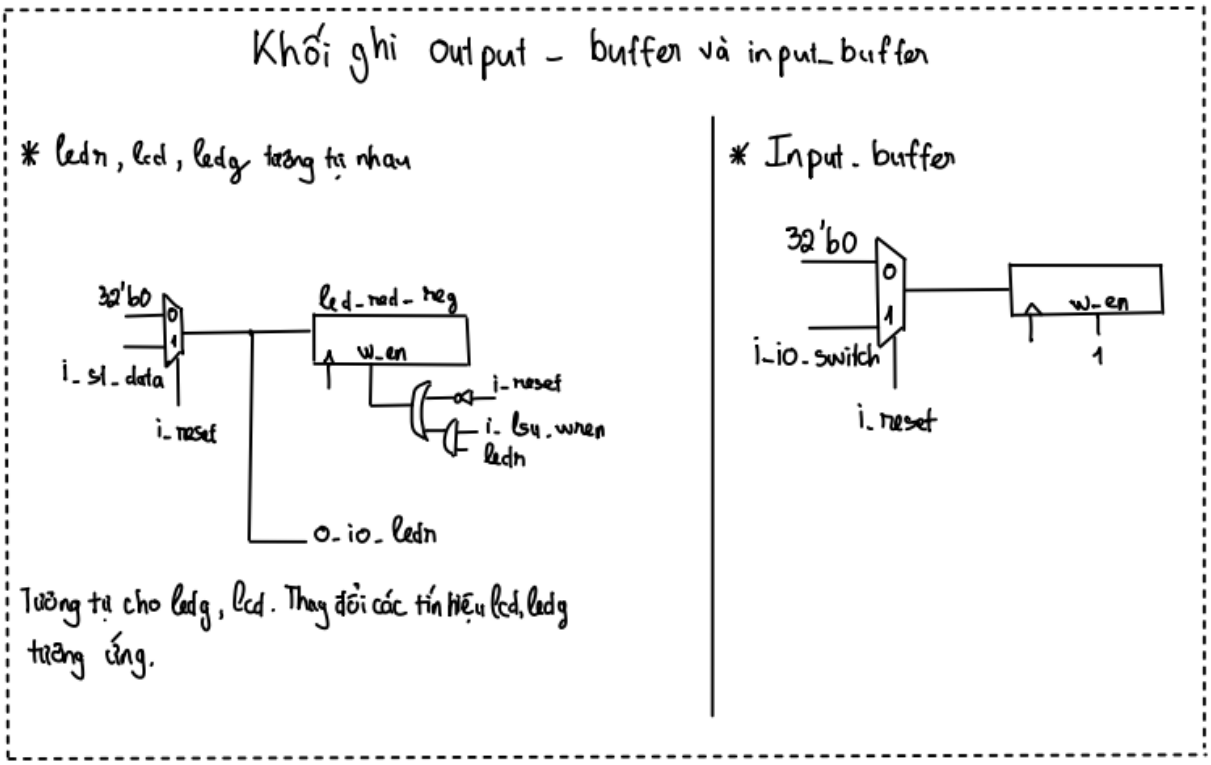






Seven_segment 3_0 và 7_4





Hình 7: Sơ đồ khối LSU

3. Control logic

❖ Phân tích yêu cầu thiết kế Control Unit

- Nhóm sẽ chia các nhóm lệnh dựa trên opcode của bit [6:0], gồm có 6 nhóm lệnh chính là R-format, I-format(layout và load), S-format, B-format, J-format, U-format. Trong mỗi nhóm lệnh sẽ dùng funct 3 để phân biệt, nếu 2 câu lệnh giống nhau thì dùng thêm bit 5 của funct 7 để phân biêt.

R-format:

funct7			funct3		opcode	
0000000	rs2	rs1	000	rd	0110011	add
0100000	rs2	rs1	000	rd	0110011	sub
0000000	rs2	rs1	001	rd	0110011	sll
0000000	rs2	rs1	010	rd	0110011	slt
0000000	rs2	rs1	011	rd	0110011	sltu
0000000	rs2	rs1	100	rd	0110011	xor
0000000	rs2	rs1	101	rd	0110011	srl
0100000	rs2	rs1	101	rd	0110011	sra
0000000	rs2	rs1	110	rd	0110011	or
0000000	rs2	rs1	111	rd	0110011	and

I-format layout:

		funct3		opcode	
imm[11:0]		rs1	000	rd	0010011	addi
imm[11:0]		rs1	010	rd	0010011	slti
imm[11:0]		rs1	011	rd	0010011	sltiu
imm[11:0]		rs1	100	rd	0010011	xori
imm[11:0]		rs1	110	rd	0010011	ori
imm[11:0]		rs1	111	rd	0010011	andi
0000000	shamt	rs1	001	rd	0010011	slli
0000000	shamt	rs1	101	rd	0010011	srli
0100000	shamt	rs1	101	rd	0010011	srai

I-format load:

	funct3		opcode	
imm[11:0]	rs1	000	rd	0000011	lb
imm[11:0]	rs1	001	rd	0000011	lh
imm[11:0]	rs1	010	rd	0000011	lw
imm[11:0]	rs1	100	rd	0000011	lbu
imm[11:0]	rs1	101	rd	0000011	lhu

S-Format:

		funct3		opcode	
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	sb
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	sh
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	sw

B-format:

			funct3		opcode	
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	bgeu

J-format:

imm[20|10:1|11|19:12]		rd	1101111	JAL
imm[11:0]	rs1 000	rd	1100111	JALR

U-format:

imm[31:12]	rd	0110111	LUI
imm[31:12]	rd	0010111	AUIPC

- Nhóm thực hiện khối Control Unit bằng cách:

+ Đầu tiên là xác định opcode để phân biệt các format khác nhau.

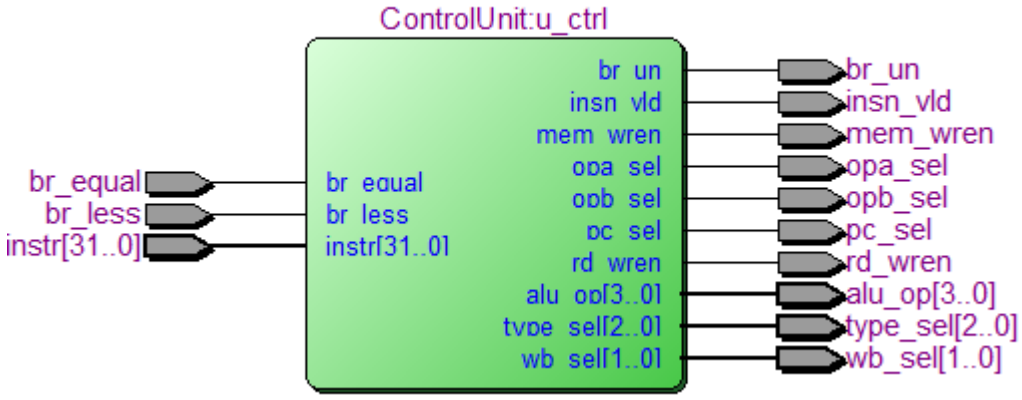
+ Sau đó trong mỗi format sẽ dùng funct3 để khác định các câu lệnh trong đó, nếu các câu lệnh có cùng funct3 thì nhóm sẽ xem trong funct7 coi bit nào khác nhau.

❖ Control Unit Specification:

Signal name	Width	Direction	Description
instr	32	input	Instruction 32 bit
br_less	1	input	Compare flag (1 if rs1 < rs2)
br_equal	1	input	Compare flag (1 if rs1 = rs2)
pc_sel	1	output	Pc select (0 when PC+4; 1 when branch)
rd_wren	1	output	Write enable (1 if writing)
insn_vld	1	output	Instruction valid (1 if valid)
br_un	1	output	Compare unsign (1 if

			unsign)
opa_sel	1	output	Operand select a (1 if rs1, 0 if PC)
opb_sel	1	output	Operand select b (1 if immediate, 0 if rs2)
alu_op	4	output	Select ALU operation (add, sub, and…)
mem_wren	1	output	Write enable (1 if writing)
type_sel	3	output	Type select
wb_sel	2	output	Write back select

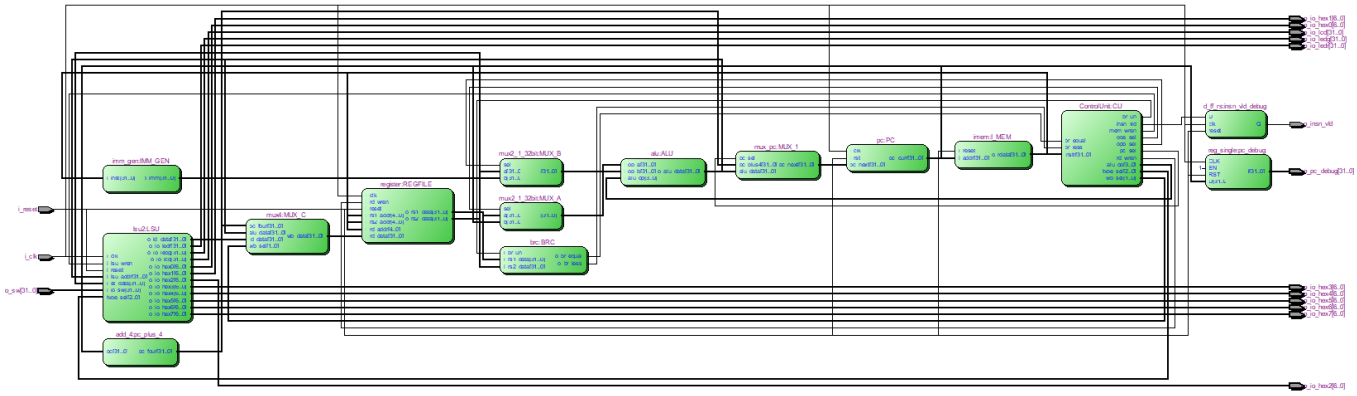
❖ **Sơ đồ khối Control Unit:**



Hình 8: Sơ đồ khối Control Unit

4.Single-Cycle Processor

❖ Sơ đồ khối Single-cycle Processor:



Hình 9: Sơ đồ khối Single-cycle Processor

❖ Single_Cycle Processor Specification:

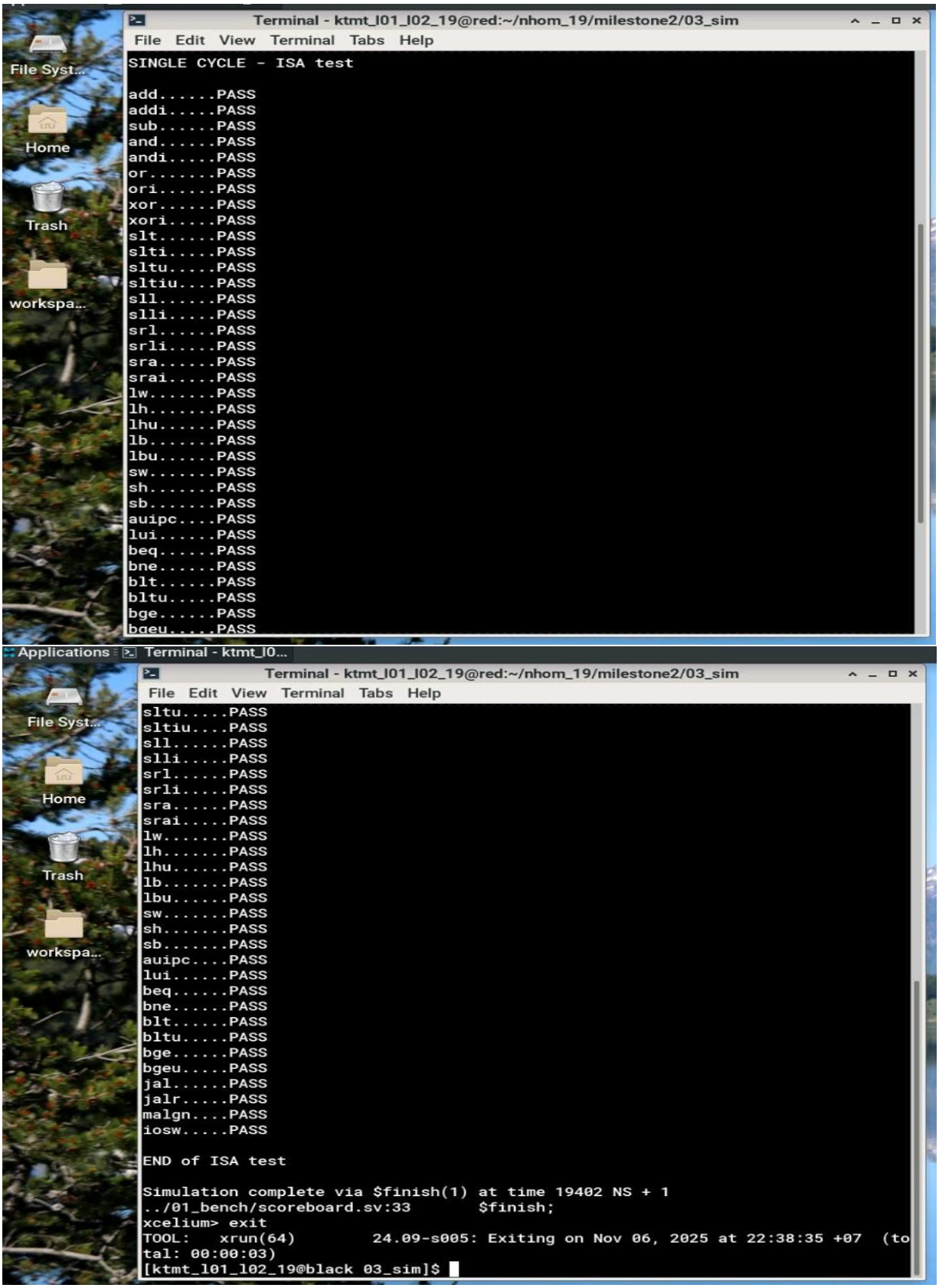
Signal name	Width	Direction	Description
i_clk	1	input	Clock, active on rising edge
i_reset	1	input	Active reset
i_io_sw	32	input	Switch
o_pc_debug	32	output	PC output for debugging
o_insn_vld	1	output	Instruction valid
o_io_ledr	32	output	Red LEDs
o_io_ledg	32	output	Green LEDs
o_io_hex0…7	7	output	7-segment
o_io_lcd	32	output	LCD

❖ Test trên server:

- Đầu tiên phải load module xcelium

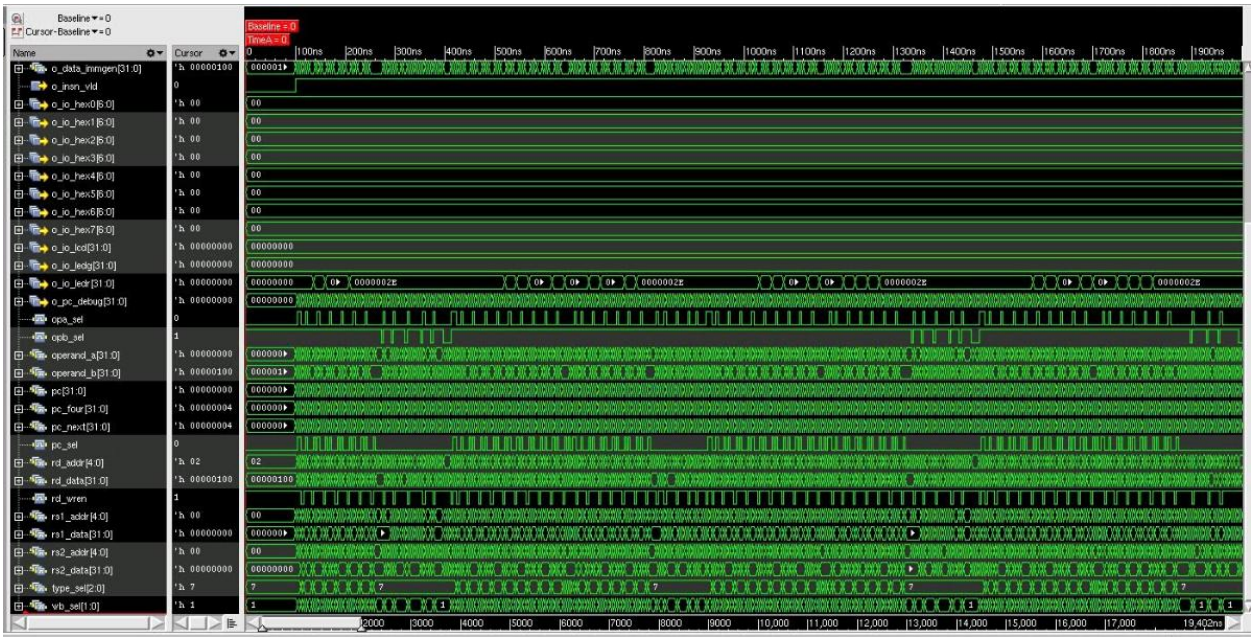
- Tiếp theo phải đến nơi chứa file mô phỏng và dùng lệnh make sim để chạy mô phỏng, còn make wave để xem được dạng sóng.

Dưới đây là phần chạy mô phỏng của nhóm:



Hình 10: Mô phỏng trên server

Tiếp theo là phần xem dạng sóng:



Hình 11: Mô phỏng dạng sóng

5.I/O System Conventions

- Các quy ước khi thiết lập ngoại vi với kit DE2:

+ i_io_ledr

Bits	Usage
31 - 17	(Reserved)
16 - 0	17-bit data connected to the array of 17 red LEDs in order.

+ i_io_ledg

Bits	Usage
31 - 8	(Reserved)
7 - 0	8-bit data connected to the array of 8 green LEDs in order.

+ Seven segment

Address	**0x1000_2000**
Bits	**Usage**
31	(Reserved)
30 - 24	7-bit data to HEX3.
23	(Reserved)
22 - 16	7-bit data to HEX2.
15	(Reserved)
14 - 8	7-bit data to HEX1.
7	(Reserved)
6 - 0	7-bit data to HEX0.

Address	**0x1000_3000**
Bits	**Usage**
31	(Reserved)
30 - 24	7-bit data to HEX7.
23	(Reserved)
22 - 16	7-bit data to HEX6.
15	(Reserved)
14 - 8	7-bit data to HEX5.
7	(Reserved)
6 - 0	7-bit data to HEX4.

+ LCD

Bits	**Usage**
31	ON
30 - 11	(Reserved)
10	EN
9	RS
8	R/W
7 - 0	Data.

+ Switches

Bits	Usage
31 - 18	(Reserved)
17	Reset.
16 - 0	17-bit data from SW16 to SW0 respectively.

6. Application

- Trong phần này nhóm đã sử dụng hợp ngữ để viết ra ứng dụng cho Single-cycle Processor. Sau khi viết bằng hợp ngữ thì nhóm sử dụng venus để có thể chuyển sang mã hex rồi từ đó đổ lên kit DE2.

- Đề bài: cho trước hai số A và B, nhập từ Switches giá trị của C (switch nào được bật thì ledr tại đó sáng lên). Tìm xem khoảng cách từ C đến hai số cho trước thì C gần số nào hơn và xuất lên LCD. (switch17 được dùng như để xuất giá trị lên LCD khi nhập xong giá trị của C, gạt lên thì LCD sáng còn muốn reset để nhập lại giá trị của C thì bấm Key0)

a) Code:

```
#1lenh ~ 60ns
#gia tri cho san A = s10, B = s11
    addi s10, zero, 1000
    addi s11, zero, 500
#  ĐỊA CHỈ NGOẠI VI
    lui   t1, 0x10010   # SW_ADDR
    lui   t2, 0x10002    # HEX03_ADDR
    lui   t3, 0x10004   #LCD
    lui   t4, 0x10000   #led red
    addi a0, zero, 1
#khoi dong lcd
    jal ra, delay_20ms
    jal ra, delay_20ms
```

```
addi s1, zero, 0x238
sw s1, 0(t3)
addi s1, zero, 0x038
sw s1, 0(t3)
jal ra, delay_20ms          #ghi ma lenh 0x38

addi s1, zero, 0x20c
sw s1, 0(t3)
addi s1, zero, 0x00c
sw s1, 0(t3)
jal ra, delay_20ms          #ghi ma lenh 0x0c

addi s1, zero, 0x206
sw s1, 0(t3)
addi s1, zero, 0x006
sw s1, 0(t3)
jal ra, delay_20ms          #ghi ma lenh 0x06

addi s1, zero, 0x201
sw s1, 0(t3)
addi s1, zero, 0x001
sw s1, 0(t3)
jal ra, delay_20ms          #ghi ma lenh 0x01

addi s1, zero, 0x282
sw s1, 0(t3)
addi s1, zero, 0x082
sw s1, 0(t3)
```

```
 jal ra, delay_20ms            #ghi ma lenh 0x82

#XUAT GIA TRI A VA B RA DONG 1 LCD
#xuatA
#xuat A=
addi s2, zero, 0x41
ori s2, s2, 0x300
sw s2, 0(t3)
andi s2, s2 0x5ff
sw s2, 0(t3)
jal ra, delay_20ms

addi s2, zero, 0x3d
 ori s2, s2, 0x300
 sw s2, 0(t3)
 andi s2, s2 0x5ff
 sw s2, 0(t3)
 jal ra, delay_20ms

mv s3, s10
jal ra, tach_so
  #xuat lcd
  #hang nghin
 addi s4, s4, 0x30
 ori s4, s4, 0x300
 sw s4, 0(t3)
 andi s4, s4, 0x5ff
 sw s4, 0(t3)
 jal ra, delay_20ms
```

```
#hang tram
 addi s5, s5, 0x30
ori s5, s5, 0x300
sw s5, 0(t3)
andi s5, s5, 0x5ff
sw s5, 0(t3)
jal ra, delay_20ms
#hang chuc
 addi s6, s6, 0x30
ori s6, s6, 0x300
sw s6, 0(t3)
andi s6, s6, 0x5ff
sw s6, 0(t3)
jal ra, delay_20ms
#hang don vi
 addi s7, s7, 0x30
ori s7, s7, 0x300
sw s7, 0(t3)
andi s7, s7, 0x5ff
sw s7, 0(t3)
jal ra, delay_20ms

#gui data dau cach
addi s2, zero, 0x20
 ori s2, s2, 0x300
sw s2, 0(t3)
andi s2, s2 0x5ff
sw s2, 0(t3)
jal ra, delay_20ms
```

```
#xuat B
#xuat B=
addi s2, zero, 0x42
 ori s2, s2, 0x300
 sw s2, 0(t3)
 andi s2, s2 0x5ff
 sw s2, 0(t3)
 jal ra, delay_20ms

 addi s2, zero, 0x3d
  ori s2, s2, 0x300
 sw s2, 0(t3)
 andi s2, s2 0x5ff
 sw s2, 0(t3)
 jal ra, delay_20ms

 mv s3, s11
 jal ra, tach_so
  #xuat lcd
  #hang nghin
 addi s4, s4, 0x30
 ori s4, s4, 0x300
 sw s4, 0(t3)
 andi s4, s4, 0x5ff
 sw s4, 0(t3)
 jal ra, delay_20ms
 #hang tram
  addi s5, s5, 0x30
```

```
    ori s5, s5, 0x300
    sw s5, 0(t3)
    andi s5, s5, 0x5ff
    sw s5, 0(t3)
    jal ra, delay_20ms
    #hang chuc
     addi s6, s6, 0x30
    ori s6, s6, 0x300
    sw s6, 0(t3)
    andi s6, s6, 0x5ff
    sw s6, 0(t3)
    jal ra, delay_20ms
    #hang don vi
     addi s7, s7, 0x30
    ori s7, s7, 0x300
    sw s7, 0(t3)
    andi s7, s7, 0x5ff
    sw s7, 0(t3)
    jal ra, delay_20ms

    #xuong hang
    addi s2, zero, 0x2c2     #0x10_1100_0000
    sw s2, 0(t3)
    andi s2, s4, 0x5ff
    sw s2, 0(t3)
    jal ra, delay_20ms



#kiem tra gia tri switch voi gia tri cho san (lay 13 bit cuoi cua switch)
```

```
main:
  lw s2, 0(t1)
  srli s8, s2, 17
  bne s8, a0, main
  li a0, 0x01fff
  and s2, s2, a0
  sw s2, 0(t4)
  addi s3, s2, 0
  j xuat_led_7_doan
  here:
  sub s3, s2, s10
  blt s3, zero, bu_2_s3
  back_1:
  sub s4, s2, s11
  blt s4, zero, bu_2_s4
  back_2:
  blt s3, s4, gan_A_hon
  beq s3, s4, nam_giua
  j gan_B_hon



  gan_A_hon:
  addi s3, zero, 1
  addi s2, zero, 10       #gom 10 ki tu "gan A hon!"
  addi a0, zero, 128      #luu ma ascci tu dong 21 cua dmem (n-1)x4 = địa chỉ byte
  tiep_A:
  lw s1, 0(a0)
  ori s1, s1, 0x300
  sw s1, 0(t3)
```

```
andi s1, s1, 0x5ff
sw s1, 0(t3)
jal ra, delay_20ms
addi a0, a0, 4
addi s2, s2, -1
blt s2, s3, main
j tiep_A



gan_B_hon:
addi s3, zero, 1
addi s2, zero, 10        #gom 10 ki tu "gan B hon!"
addi a0, zero, 248
tiep_B:
lw s1, 0(a0)
ori s1, s1, 0x300
sw s1, 0(t3)
andi s1, s1, 0x5ff
sw s1, 0(t3)
jal ra, delay_20ms
addi a0, a0, 4
addi s2, s2, -1
blt s2, s3, main
j tiep_B



nam_giua:
addi s3, zero, 1
addi s2, zero, 12        #gom 10 ki tu "k/c nhu nhau"
```

```
    addi a0, zero, 40
    tiep_C:
    lw s1, 0(a0)
    ori s1, s1, 0x300    #11_0000_0000
    sw s1, 0(t3)
    andi s1, s1, 0x5ff
    sw s1, 0(t3)
    jal ra, delay_20ms
    addi a0, a0, 4
    addi s2, s2, -1
    blt s2, s3, main
    j tiep_C



    xuat_led_7_doan:
     # ---- Reset các chữ số ----
    addi s4, zero, 0        # nghìn
    addi s5, zero, 0        # trăm
    addi s6, zero, 0        # chục
    addi s7, zero, 0        # đơn vị

# ---- NGHÌN ----
thousand_loop:
    li   s9, 1000
    blt  s3, s9, hundred_loop
    addi s4, s4, 1
    sub  s3, s3, s9
    j    thousand_loop
```

```
# ---- TRĂM ----
hundred_loop:
    li   s9, 100
    blt  s3, s9, ten_loop
    addi s5, s5, 1
    sub  s3, s3, s9
    j    hundred_loop

# ---- CHỤC ----
ten_loop:
    li   s9, 10
    blt  s3, s9, unit_done
    addi s6, s6, 1
    sub  s3, s3, s9
    j    ten_loop

# ---- ĐƠN VỊ ----
unit_done:
    addi s7, s3, 0          # còn lại là đơn vị



# ==== TRA BẢNG LED 7 ĐOẠN ====
    addi   a0, zero, 0       # a0 = base address of table

    slli s4, s4, 2
    slli s5, s5, 2
    slli s6, s6, 2
    slli s7, s7, 2
```

```
    add  a1, a0, s4
    lw   s4, 0(a1)
    add  a1, a0, s5
    lw   s5, 0(a1)
    add  a1, a0, s6
    lw   s6, 0(a1)
    add  a1, a0, s7
    lw   s7, 0(a1)



# ==== GỘP 4 CHỮ SỐ VÀ GHI RA HEX ====
    slli s4, s4, 24
    slli s5, s5, 16
    slli s6, s6, 8
    or   s4, s4, s5
    or   s4, s4, s6
    or   s4, s4, s7

    sw   s4, 0(t2)          # ghi ra HEX0–HEX3
    j here

    bu_2_s3:
    sub s3, zero, s3
    j back_1

    bu_2_s4:
    sub s4, zero, s4
    j back_2
```

```
   #tach so A va B
  tach_so:
   addi s4, zero, 0        # nghìn
   addi s5, zero, 0        # trăm
   addi s6, zero, 0        # chục
   addi s7, zero, 0        # đơn vị
   nghin_loop:
   li   s9, 1000
   blt  s3, s9, tram_loop
   addi s4, s4, 1
   sub  s3, s3, s9
   j    nghin_loop

# ---- TRĂM ----
tram_loop:
   li   s9, 100
   blt  s3, s9, chuc_loop
   addi s5, s5, 1
   sub  s3, s3, s9
   j    tram_loop

# ---- CHỤC ----
chuc_loop:
   li   s9, 10
   blt  s3, s9, donvi_done
   addi s6, s6, 1
   sub  s3, s3, s9
   j    chuc_loop
```

```
# ---- ĐƠN VỊ ----
donvi_done:
   addi s7, s3, 0          # còn lại là đơn vị
   ret

   delay_20ms:
   li   t5, 166666         # số vòng lặp 20ms
delay_loop:
   addi t5, t5, -1
   bnez t5, delay_loop
   ret
```

b) Kết quả thực tế khi chạy trên kit DE2

- Với bài này nhóm đang cho A=1000 và B=500

Hình 12: Kết quả thực tế

III. Đánh giá

Nhóm đã thiết kế được thành công một bộ xử lý đơn chu kỳ (single-cycle processor) bao gồm các khối như yêu cầu. Qua việc kiểm tra và mô phỏng, nhóm đã đảm bảo được tính chính xác và hoạt động đúng của single-cycle processor. Bài ứng dụng mà nhóm làm ra thì đảm bảo được đúng yêu cầu ban đầu mà nhóm đề ra cũng như chạy được đúng chính xác với yêu cầu đề bài.