

HƯỚNG DẪN THÍ NGHIỆM

CẤU TRÚC MÁY TÍNH

Mục tiêu của bài tập lớn là giúp sinh viên hiểu và thiết kế được CPU chạy đơn chu kỳ và pipeline dựa trên cấu trúc CPU RV32.

1. TỔNG QUÁT VỀ CPU RV32

a. Tổng quát

- CPU RV32 có tổng cộng 32 lệnh hợp ngữ, trong đó mỗi lệnh có độ dài 32bit và có 7 bit [6:0] (opcode) để xác định loại lệnh.

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20:10:11:19:12]				rd	1101111	JAL
imm[11:0]				rd	1100111	JALR
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
imm[11:0]				rd	0000011	LB
imm[11:0]				rd	0000011	LH
imm[11:0]				rd	0000011	LW
imm[11:0]				rd	0000011	LBU
imm[11:0]				rd	0000011	LHU
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
imm[11:0]				rd	0010011	ADDI
imm[11:0]				rd	0010011	SLTI
imm[11:0]				rd	0010011	SLTIU
imm[11:0]				rd	0010011	XORI
imm[11:0]				rd	0010011	ORI
imm[11:0]				rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
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

- Tập lệnh của RV32 còn được gọi là tập lệnh kiểu load-store, điều đó có nghĩa là data trong bộ nhớ muốn được thực thi thì trước hết phải được lấy ra bỏ vào băng thanh ghi rồi mới được tính toán. Sau khi tính toán, data sẽ được lưu lại vào memory.
- Các thanh ghi trong băng thanh ghi (Register Bank) có độ dài 32 bits và có 32 thanh ghi (từ $x_0 - x_{31}$) \Rightarrow Cần có 5 bits để xác định địa chỉ của các thanh ghi trong băng thanh ghi. Trong đó, chức năng của 32 thanh ghi được cho như ở bảng dưới.

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller

Lưu ý: Thanh ghi x_0 luôn có giá trị bằng 0x00000000 và không thay đổi giá trị này.

- Dữ liệu trong cả bộ nhớ dữ liệu (DMEM) và bộ nhớ chương trình (IMEM) đều có độ dài 32bit và được sắp xếp theo kiểu *little endian*.

DMEM được định địa chỉ theo từng byte (= 8 bits) chứ không theo word (= 32 bits).

Nếu định địa chỉ theo word thì lấy địa chỉ của byte có trọng số thấp nhất.

Điều này được trình bày như ở hình dưới.

Least-significant byte in a word

...
15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	0

31 24 23 16 15 8 7 0

Least-significant byte

gets the smallest address

b. Nhóm lệnh R-Format:

Nhóm lệnh này bao gồm các lệnh có cấu trúc như ở hình sau:

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

Nhóm lệnh này có opcode là $[6:0] = 0110011$

Nhóm lệnh này thực hiện lấy hai giá trị lưu ở thanh ghi *rs1* và *rs2* thực hiện đưa vào khối ALU để tính toán, sau đó lưu kết quả vào thanh ghi *rd*.

c. Nhóm lệnh I (Tính toán)

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

Nhóm lệnh này có opcode là $[6:0] = 0010011$

Nhóm lệnh này (trừ 3 lệnh SRAI, SRLI, SLLI) thực hiện lấy giá trị lưu ở thanh ghi *rs1* và giá trị lưu ở *imm*[11:0] (được mở rộng dấu), thực hiện đưa vào khối ALU để tính toán. Kết quả được lưu vào thanh ghi *rd*.

d. Nhóm lệnh I (Load data)

<i>imm</i> [11:0]	<i>rs1</i>	000	<i>rd</i>	0000011	lb
<i>imm</i> [11:0]	<i>rs1</i>	010	<i>rd</i>	0000011	lh
<i>imm</i> [11:0]	<i>rs1</i>	011	<i>rd</i>	0000011	lw
<i>imm</i> [11:0]	<i>rs1</i>	100	<i>rd</i>	0000011	lbu
<i>imm</i> [11:0]	<i>rs1</i>	110	<i>rd</i>	0000011	lhu

Nhóm lệnh này có opcode là $[6:0] = 0000011$

Nhóm lệnh này thực hiện lấy hai giá trị lưu ở thanh ghi *rs1* và giá trị lưu ở *imm*[11:0] (được mở rộng dấu) để tính tổng $rs1 + ext(imm[11:0])$. Sau đó lấy giá trị lưu trong DMEM tại địa chỉ $rs1 + ext(imm[11:0])$, lưu vào thanh ghi *rd*.

e. Nhóm lệnh S (Store data)

<i>Imm</i> [11:5]	<i>rs2</i>	<i>rs1</i>	000	<i>imm</i> [4:0]	0100011	sb
<i>Imm</i> [11:5]	<i>rs2</i>	<i>rs1</i>	001	<i>imm</i> [4:0]	0100011	sh
<i>Imm</i> [11:5]	<i>rs2</i>	<i>rs1</i>	010	<i>imm</i> [4:0]	0100011	sw

Nhóm lệnh này có opcode là $[6:0] = 0100011$

Nhóm lệnh này thực hiện lấy hai giá trị lưu ở thanh ghi *rs1* và giá trị lưu ở *imm*[11:5] và *imm*[4:0] (ghép lại và mở rộng dấu) để tính tổng $rs1 + ext(imm[11:5]imm[4:0])$. Sau đó lấy giá trị lưu trong thanh ghi *rs2* lưu vào DMEM tại địa chỉ $rs1 + ext(imm[11:5]imm[4:0])$.

f. Nhóm lệnh B (rẽ nhánh)

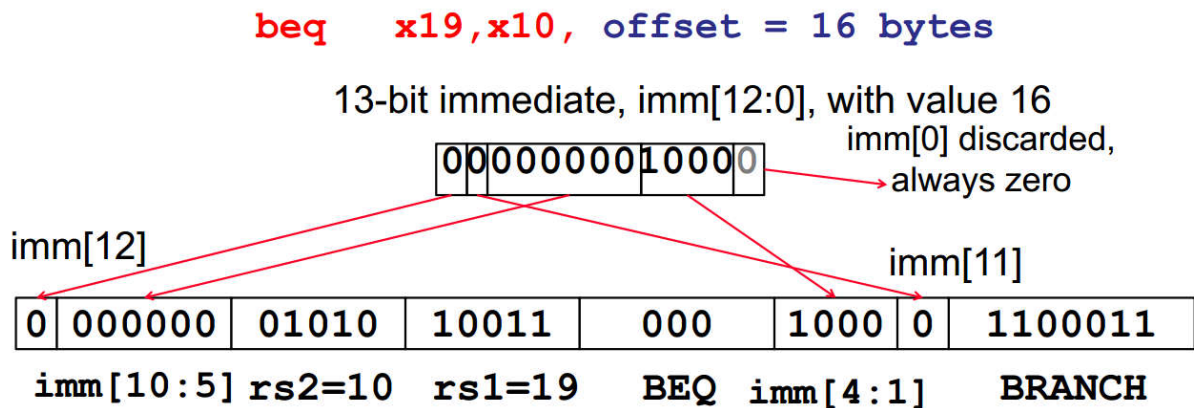
$imm[12:10:5]$	$rs2$	$rs1$	000	$imm[4:1:11]$	1100011	BEQ
$imm[12:10:5]$	$rs2$	$rs1$	001	$imm[4:1:11]$	1100011	BNE
$imm[12:10:5]$	$rs2$	$rs1$	100	$imm[4:1:11]$	1100011	BLT
$imm[12:10:5]$	$rs2$	$rs1$	101	$imm[4:1:11]$	1100011	BGE
$imm[12:10:5]$	$rs2$	$rs1$	110	$imm[4:1:11]$	1100011	BLTU
$imm[12:10:5]$	$rs2$	$rs1$	111	$imm[4:1:11]$	1100011	BGEU

Nhóm lệnh này có opcode là $[6:0] = 1100011$

Nhóm lệnh này sẽ thực hiện chuyển giá trị của thanh ghi PC thành giá trị được lưu trong các phần *imm* giá trị lưu trong *rs1* và *rs2* thỏa điều kiện câu lệnh (bằng, không bằng, lớn hơn hoặc bằng, ...).

Khi lấy giá trị lưu ở phần *imm* ta phải ghép lại cho đúng thứ tự và mở rộng dấu, bit LSB luôn luôn bằng 0.

Lấy ví dụ như ở hình dưới:



g. Nhóm lệnh U

31	12	11	7	6	0
imm[31:12]			rd	opcode	

- Với lệnh LUI

Opcode = 0110111

Lệnh này load giá trị $imm[31:12]000000000000$ vào thanh ghi *rd*.

- Với lệnh AUIPC

Opcode = 0010111

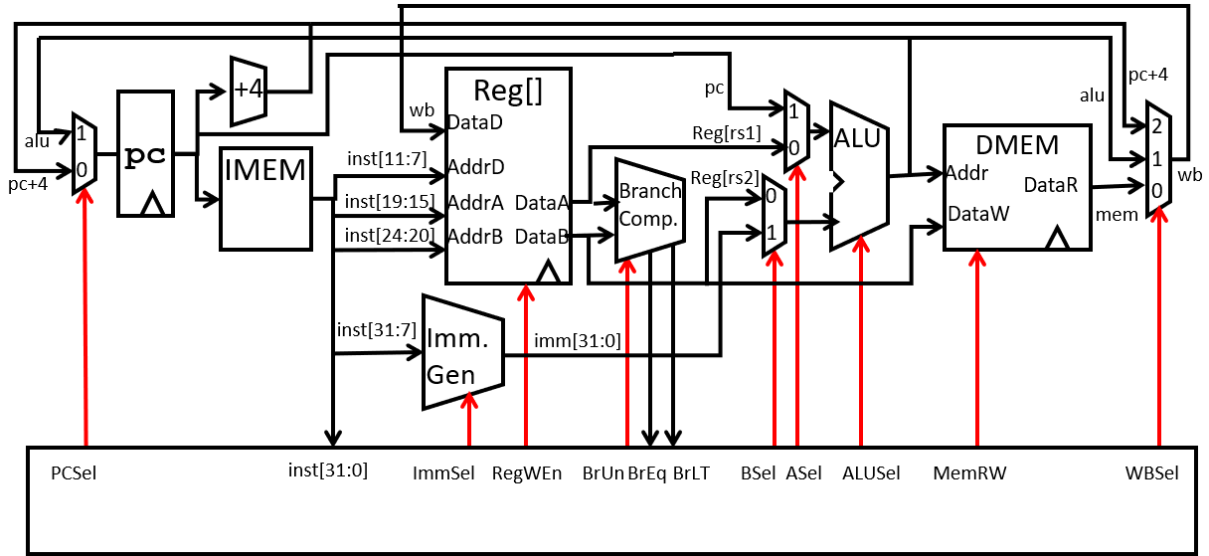
Lệnh này load giá trị ở *PC* vào thanh ghi *rd*.

h. Nhóm lệnh J (nhảy không điều kiện)

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

2. THIẾT KẾ CPU RV32 ĐƠN CHU KỲ

Sinh viên được khuyến khích thiết kế CPU RV32 đơn chu kỳ theo sơ đồ khối như hình:



Trong đó, các tín hiệu kết nối nên được đặt tên theo chuẩn ở bảng dưới:

a. Các tín hiệu giữa các khối (không tính đến khối control)

Name	Khối bắt đầu	Khối đích	Ý nghĩa
clk	Input của CPU	PC, register bank, DMEM	Xung clock điều khiển chu kỳ lệnh
pc_in	PCmux	PC	PC của lệnh tiếp theo
pc_out	PC	IMEM, PC+4, ALUmux1	PC hiện tại
pc_plus4_out	PC+4	PCmux	$PC \leftarrow PC+4$
rs1	IMEM	Register bank	Địa chỉ của rs1
rs2	IMEM	Register bank	Địa chỉ của rs2
rsd	IMEM	Register bank	Địa chỉ của rsd
rs1_out	Register bank	BranchComp, ALUmux1	Data của rs1
rs2_out	Register bank	BranchComp, ALUmux2, DMEM	Data của rs2
imm_in	IMEM	ImmGen	Data vào ImmGen
imm_out	ImmGen	ALUmux2	Data sau khi qua khối ImmGen
alumux1_out	ALUmux1	ALU	Toán hạng 1 vào ALU
alumux2_out	ALUmux2	ALU	Toán hạng 2 vào ALU

aluout	ALU	DMEM, Wbmux, PCmux	Ngõ ra của ALU
dmem_out	DMEM	Wbmux	Data đọc của DMEM
wb_out	Wbmux	Register bank	Data ghi ngược

b. Tín hiệu đưa vào khối Control

<i>Name</i>	<i>Type</i>	<i>Chú thích</i>
inst		Câu lệnh đưa vào khối control
br_eq	0: DataA \neq DataB 1: DataA = DataB	Tín hiệu kết quả ở khối Branch Comp
br_lt	0: DataA \geq DataB 1: DataA < DataB	Tín hiệu kết quả ở khối Branch Comp

c. Tín hiệu từ khối Control tới các khối còn lại

<i>Name</i>	<i>Type</i>	<i>Chú thích</i>
pcmux_sel PCSel	1: chọn từ ALU 0: chọn từ pc+4	Tín hiệu điều khiển mux trước PC
imm_sel immSel	000: chọn tạo Imm theo kiểu I-format 001: chọn tạo Imm theo kiểu S-format 010: chọn tạo Imm theo kiểu B-format 011: chọn tạo Imm theo kiểu U-format 100: chọn tạo Imm theo kiểu J-format	Tín hiệu điều khiển bộ Imm Gen
regfilemux_sel RegWEn	0: only read 1: read write enable	Tín hiệu điều khiển read/write băng thanh ghi
cmpop BrUn	0: so sánh không dấu <i>doi bang 1</i> 1: so sánh có dấu <i>doi bang 0</i>	Tín hiệu điều khiển bộ so sánh BranchComp
alumux1_sel ASel	0: lấy dữ liệu từ băng thanh ghi 1: lấy dữ liệu từ PC	Tín hiệu điều khiển chọn toán hạng 1 đưa vào ALU
alumux2_sel BSel	0: lấy dữ liệu từ băng thanh ghi 1: lấy từ khối ImmGen	Tín hiệu điều khiển chọn toán hạng 2 đưa vào ALU
aluop ALUSel	Sinh viên tự mã hóa dựa theo chức năng câu lệnh	Tín hiệu chọn phép toán thực hiện trong ALU

dmem_sel MemRW	0: cho phép read 1: cho phép write	Tín hiệu điều khiển đọc ghi DMEM
wbmux_sel WBSEL	00: Lấy dữ liệu từ DMEM để ghi ngược 01: Lấy dữ liệu từ ALU để ghi ngược 10: Lấy dữ liệu từ PC+4 để ghi ngược	Tín hiệu điều khiển việc ghi ngược lại.

3. TIẾN HÀNH THIẾT KẾ

a. Thiết kế khối Control

Tiến hành lập bảng bao gồm các lệnh, các tín hiệu vào và tín hiệu ra như hình.

No	Type	Inst[30]	Inst[14:12]	Inst[6:2]	BrEq	BrLT	PCSel	ImmSel	RegWEn	BrUn	Bsel	Asel	ALUSel	MemRW	DataIn	DataOutAddj	WBSEL
0	R	ADD	0	000	01100												
1	R	SUB	1	000	01100												
2	R	SLL	0	001	01100												
3	R	SLT	0	010	01100												
4	R	SLTU	0	011	01100												
5	R	XOR	0	100	01100												
6	R	SRL	0	101	01100												
7	R	SRA	1	101	01100												
8	R	OR	0	110	01100												
9	R	AND	0	111	01100												
10	I	ADDI	x	000	00100												
11	I	SLTI	x	010	00100												
12	I	SLTIU	x	011	00100												
13	I	XORI	x	100	00100												
14	I	ORI	x	110	00100												
15	I	ANDI	x	111	00100												
16	I	SLLI	0	001	00100												
17	I	SRLI	0	101	00100												
18	I	SRAI	1	101	00100												
19	I	LB	x	000	00000												
20	I	LH	x	001	00000												
21	I	LW	x	010	00000												
22	I	LBU	x	100	00000												
23	I	LHU	x	101	00000												

Phân tích từng lệnh và điền vào bảng như hình:

No	Type	Inst[30]	Inst[14:12]	Inst[6:2]	BrEq	BrLT	PCSel	ImmSel	RegWEn	BrUn	Bsel	Asel	ALUSel	MemRW	DataIn	DataOutAddj	WBSEL
0	R	ADD	0	000	01100	x	x	0	x		0	0	0000	0	x	x	01
1	R	SUB	1	000	01100	x	x	0	x		0	0	0001	0	x	x	01
2	R	SLL	0	001	01100	x	x	0	x		0	0	0010	0	x	x	01
3	R	SLT	0	010	01100	x	x	0	x		0	0	0011	0	x	x	01
4	R	SLTU	0	011	01100	x	x	0	x		0	0	0100	0	x	x	01
5	R	XOR	0	100	01100	x	x	0	x		0	0	0101	0	x	x	01
6	R	SRL	0	101	01100	x	x	0	x		0	0	0110	0	x	x	01
7	R	SRA	1	101	01100	x	x	0	x		0	0	0111	0	x	x	01
8	R	OR	0	110	01100	x	x	0	x		0	0	1000	0	x	x	01
9	R	AND	0	111	01100	x	x	0	x		0	0	1001	0	x	x	01
10	I	ADDI	x	000	00100												
11	I	SLTI	x	010	00100												
12	I	SLTIU	x	011	00100												
13	I	XORI	x	100	00100												
14	I	ORI	x	110	00100												
15	I	ANDI	x	111	00100												
16	I	SLLI	0	001	00100												
17	I	SRLI	0	101	00100												
18	I	SRAI	1	101	00100												
19	I	LB	x	000	00000												
20	I	LH	x	001	00000												
21	I	LW	x	010	00000												
22	I	LBU	x	100	00000												
23	I	LHU	x	101	00000												

Sau đó, chuyển bảng vừa lập thành khối control. Viết theo kiểu ROM, ví dụ:

```
module ROMControl(data,addr);
parameter      addrwidth=6;
parameter      datawidth=20;
output reg      [datawidth-1:0]data;
input          [addrwidth-1:0]addr;

always@(addr)
begin
    case(addr)
        // R type
        6'd0 : data=20'b0_000_1_0_0_0_0000_0_00_000_01;
        6'd1 : data=20'b0_000_1_0_0_0_0001_0_00_000_01;
        6'd2 : data=20'b0_000_1_0_0_0_0010_0_00_000_01;
        6'd3 : data=20'b0_000_1_0_0_0_0011_0_00_000_01;
        6'd4 : data=20'b0_000_1_0_0_0_0100_0_00_000_01;
        6'd5 : data=20'b0_000_1_0_0_0_0101_0_00_000_01;
        6'd6 : data=20'b0_000_1_0_0_0_0110_0_00_000_01;
        6'd7 : data=20'b0_000_1_0_0_0_0111_0_00_000_01;
        6'd8 : data=20'b0_000_1_0_0_0_1000_0_00_000_01;
        6'd9 : data=20'b0_000_1_0_0_0_1001_0_00_000_01;
```

b. Viết từng khối chức năng

Phân tích từng khối: chức năng, ngõ vào, ngõ ra và viết riêng từng module.

Ví dụ 1: Các khối mux trước PC và ALU có chức năng chọn 1 trong 2 ngõ vào tùy thuộc vào tín hiệu select.

Input: 2 ngõ vào 32 bit, 1 ngõ chọn tín hiệu.

Output: 1 ngõ ra 32 bit.

```
module mux2(out,in0,in1,sel);
parameter      WIDTH_DATA_LENGTH = 32;
output          [WIDTH_DATA_LENGTH-1:0]out;
input          [WIDTH_DATA_LENGTH-1:0]in0,in1;
input          sel;

assign          out = (sel)?in1:in0;

endmodule
```

Ví dụ 2: Khối IMEM có chức năng lưu các lệnh của chương trình, khi cho địa chỉ (PC) của lệnh sẽ xuất ra lệnh tại địa chỉ đó.

Input: 1 ngõ vào 32 bit (PC).

Output: 1 ngõ ra 32 bit (data).


```

module IMEM(inst,PC);
parameter      INST_WIDTH_LENGTH = 32;
parameter      PC_WIDTH_LENGTH = 32;
parameter      MEM_WIDTH_LENGTH = 32;
parameter      MEM_DEPTH = 1<<18;
output reg      [INST_WIDTH_LENGTH-1:0]inst;
input          [PC_WIDTH_LENGTH-1:0]PC;

//***** Instruction Memory *****/
reg            [MEM_WIDTH_LENGTH-1:0]IMEM[0:MEM_DEPTH-1];

wire           [17:0]pWord;
wire           [1:0]pByte;

assign         pWord = PC[19:2];
assign         pByte = PC[1:0];

initial begin
$readmemh("D:/Documents_Study/Computer Architecture/RISC-V_project/TestCode/pipeline_test.txt", IMEM);
end

always@(PC)
begin
    if (pByte == 2'b00)
        inst <= IMEM[pWord];
    else
        inst <= 'hz;
end

endmodule

```

Trong đó lệnh *\$readmemh* dùng để load chương trình vào IMEM.

c. Viết đoạn chương trình test

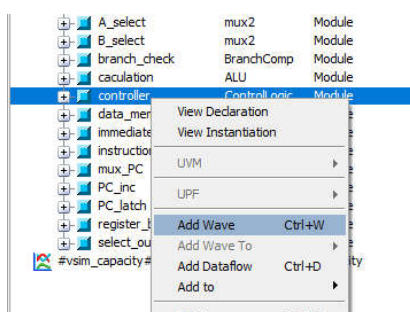
Viết đoạn chương trình test trên phần mềm RISC-V, hướng dẫn sử dụng phần mềm nằm ở phần phụ lục sau.

Các đoạn chương trình khi test nộp bài phải là các đoạn có nhiều loại lệnh, thực hiện khối lượng lớn, cụ thể:

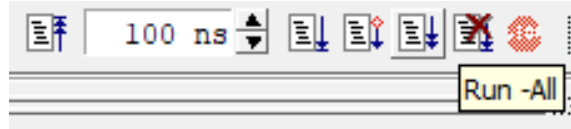
- Lấy 10 số lưu trong DMEM và sắp xếp lại rồi lưu vào DMEM ở 10 vị trí tiếp theo.
- Tính giai thừa số lớn nhất và lưu ở vị trí tiếp theo.
- Tính số Fibonanci của số lớn nhất và lưu ở vị trí tiếp theo.

d. Test dạng sóng trên ModelSim

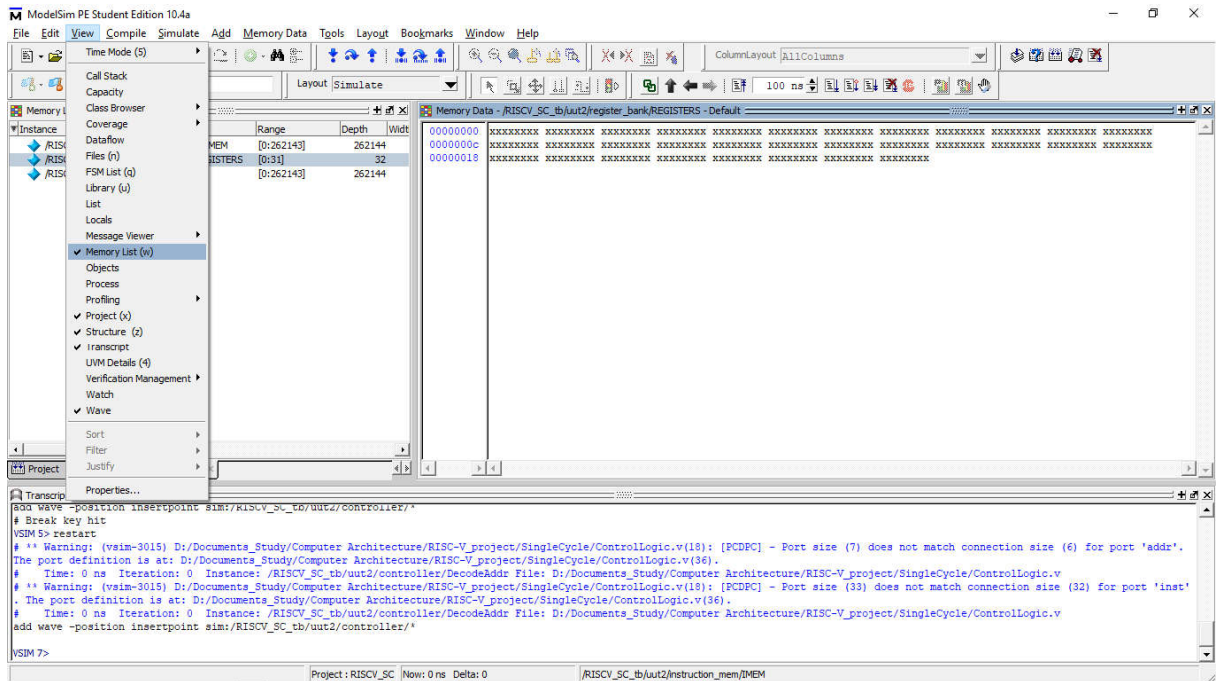
Sau khi simulate, chọn vào khối cần quan sát dạng sóng và add wave:



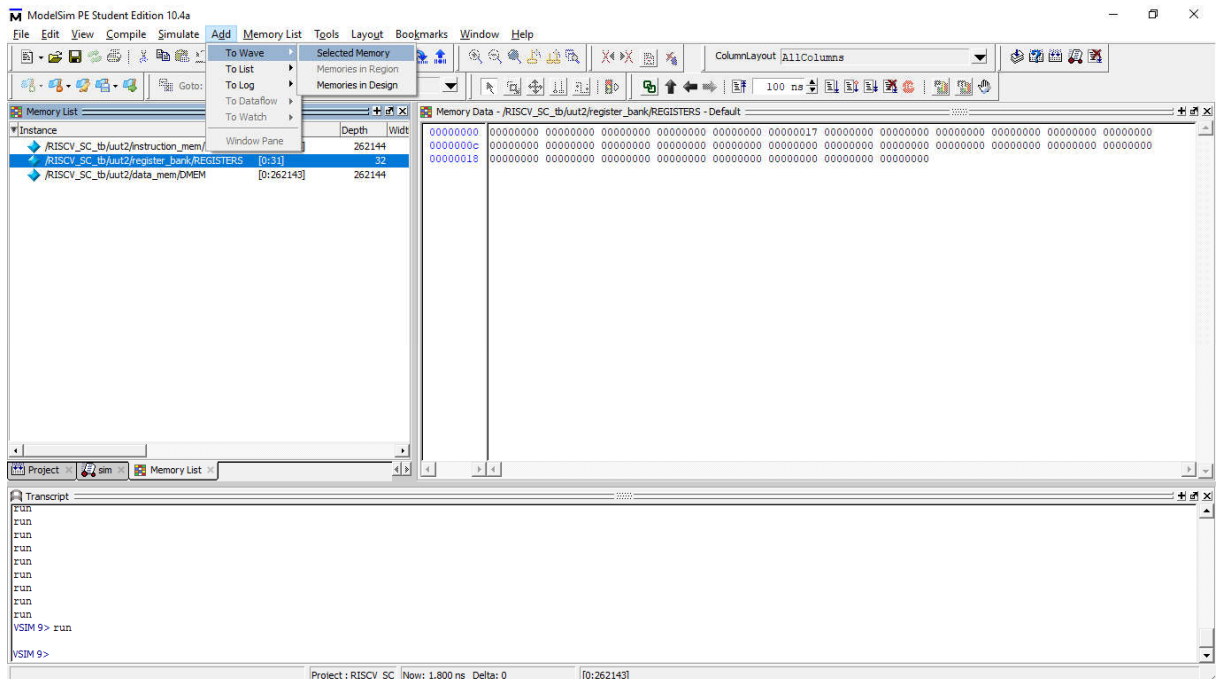
Sau đó chọn, Run hoặc RunAll để mô phỏng dạng sóng và quan sát:



Vào View→MemoryList để quan sát các khối dạng MEM như IMEM, DMEM và RegisterBank.



Để quan sát dạng sóng của memory, chọn memory vào Add→ToWave→Selected Memory



PHỤ LỤC

4. CÀI ĐẶT JAVA SE

Lưu ý: Nếu bạn đã cài JAVA SE có thể bỏ qua mục này.

Truy cập link: <https://www.oracle.com/technetwork/java/javase/downloads/index.html>

Java SE 8u201 / Java SE 8u202
Java SE 8u201 / Java SE 8u202 includes important bug fixes. Oracle strongly recommends that all Java SE 8 users upgrade to this release.
[Learn more](#)

<ul style="list-style-type: none">Installation InstructionsRelease NotesOracle LicenseJava SE Licensing Information User Manual<ul style="list-style-type: none">Includes Third Party LicensesCertified System ConfigurationsReadme Files<ul style="list-style-type: none">JDK ReadMeJRE ReadMe	JDK DOWNLOAD
	Server JRE DOWNLOAD
	JRE DOWNLOAD

Which Java package do I need?

- Software Developers: JDK** (Java SE Development Kit). For Java Developers. Includes a complete JRE plus tools for developing, debugging, and monitoring Java applications.
- Administrators running applications on a server: Server JRE** (Server Java Runtime Environment) For deploying Java applications on servers. Includes tools for JVM monitoring and tools commonly required for server applications, but does not include browser integration (the Java plug-in), auto-update, nor an installer. [Learn more](#)
- End user running Java on a desktop: JRE** (Java Runtime Environment). Covers most end-users needs. Contains everything required to run Java applications on your system.

Tải phần mềm cài đặt tại mục Java SE 8u201 / Java SE 8u202

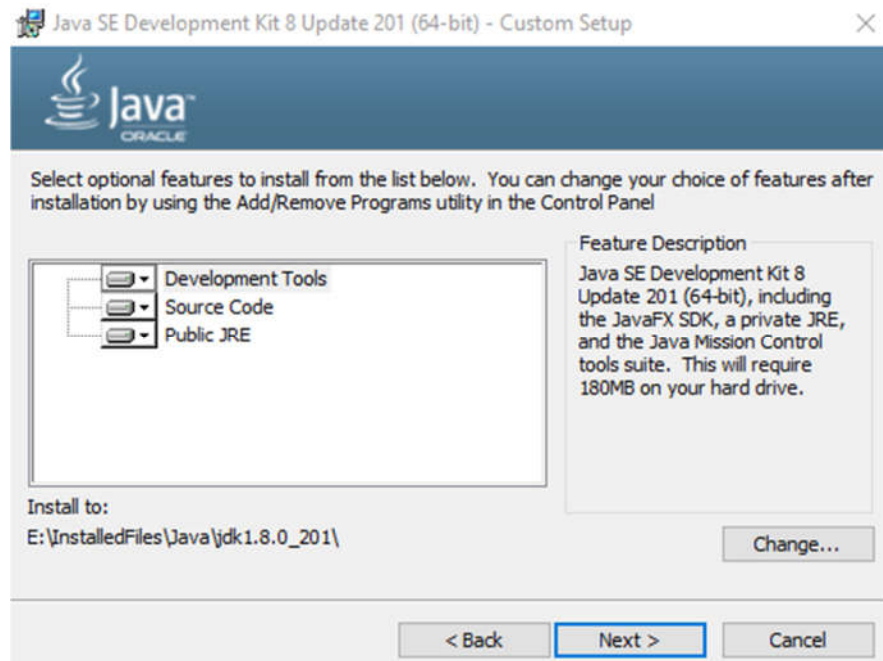
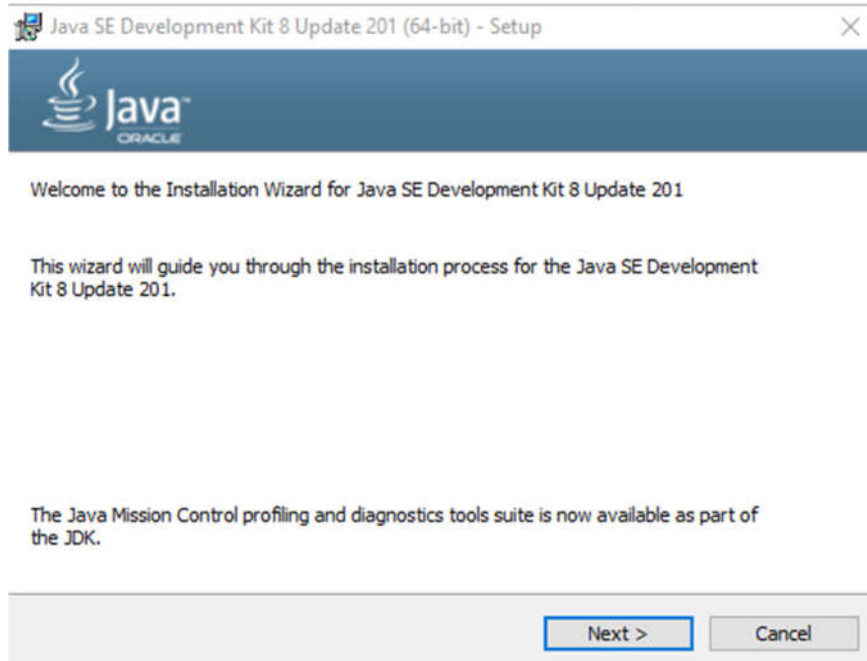
Java SE 8u201 / Java SE 8u202
Java SE 8u201 / Java SE 8u202 includes important bug fixes. Oracle strongly recommends that all Java SE 8 users upgrade to this release.
[Learn more](#)

<ul style="list-style-type: none">Installation InstructionsRelease NotesOracle LicenseJava SE Licensing Information User Manual<ul style="list-style-type: none">Includes Third Party LicensesCertified System ConfigurationsReadme Files<ul style="list-style-type: none">JDK ReadMeJRE ReadMe	JDK DOWNLOAD
	Server JRE DOWNLOAD
	JRE DOWNLOAD

Tiến hành cài đặt phần mềm vừa tải về:

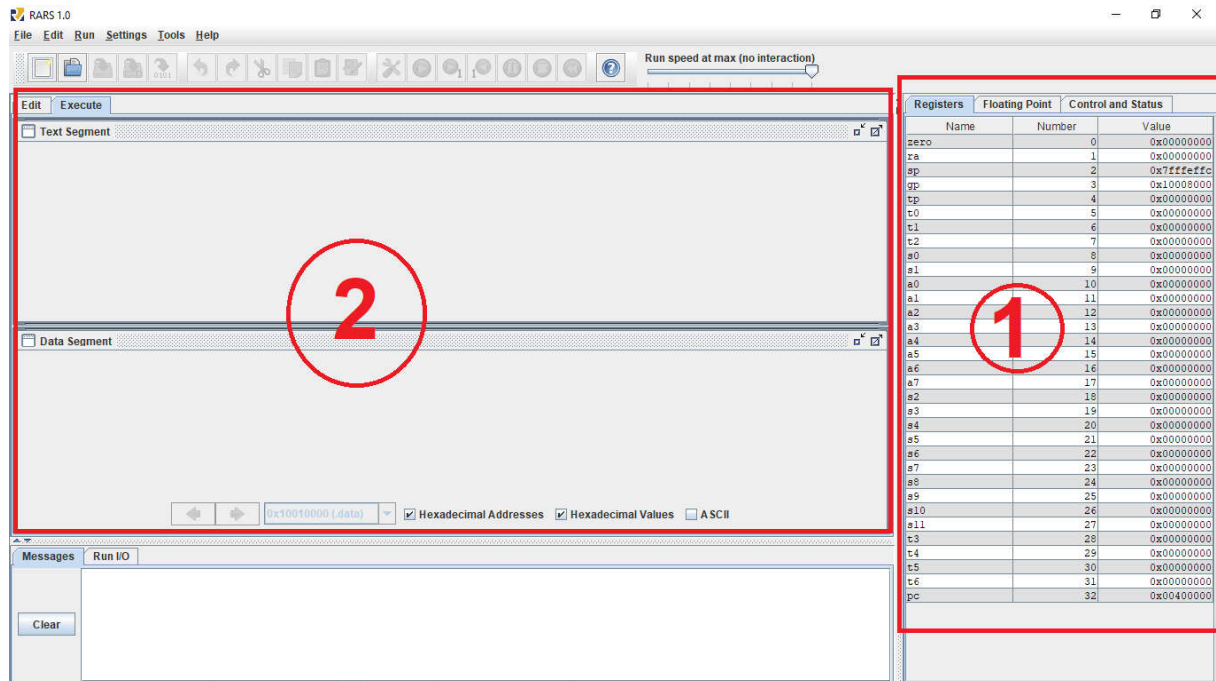


Các bước cài đặt Java SE trên hệ điều hành Window:



5. CHƯƠNG TRÌNH MÔ PHỎNG RISC-V

- Sau khi cài đặt Java, ta tiến hành mở chương trình mô phỏng.
- Chương trình có giao diện như hình dưới.



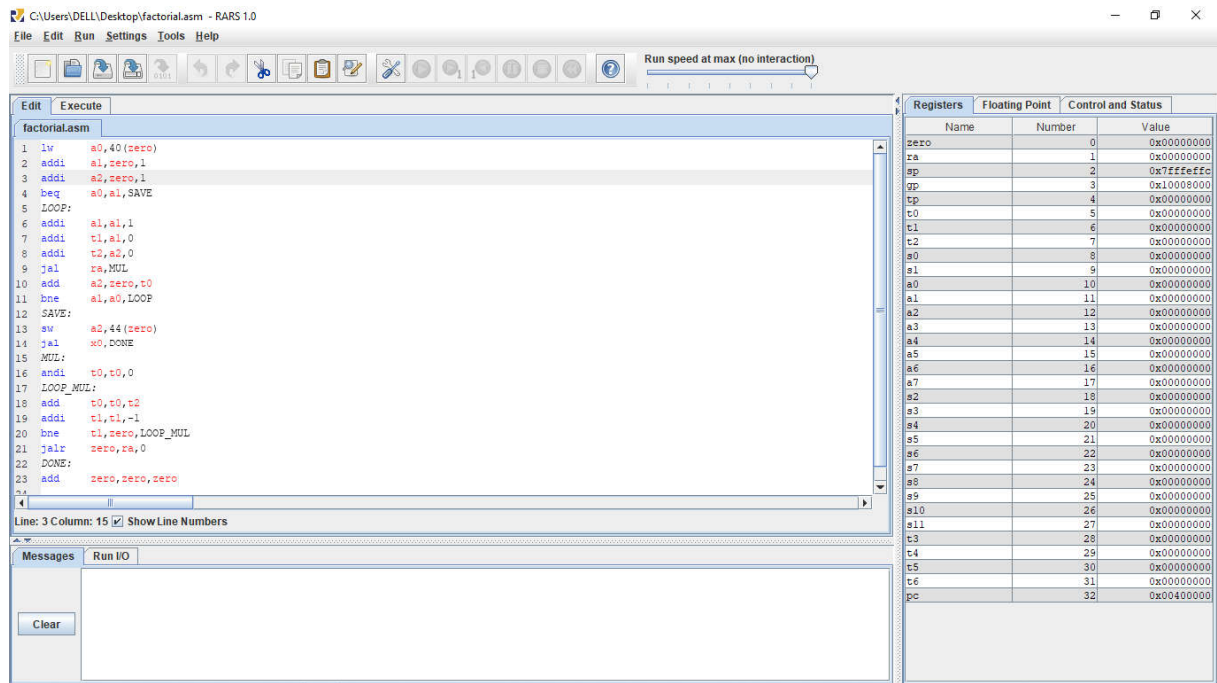
Trong giao diện gồm có 2 phần chính:

- Phần 1: Dùng để mô phỏng các thanh ghi quan trọng trong CPU.
- Phần 2: gồm 2 tab:
 - Tab Edit: dùng để nhập đoạn chương trình code dùng để mô phỏng.
 - Tab Execute: dùng để biên dịch chương trình, xem vị trí lệnh trong I-MEM và xem mã hex của lệnh.


6. SỬ DỤNG PHẦN MỀM VÀ CHƯƠNG TRÌNH MẪU

a. Sử dụng phần mềm

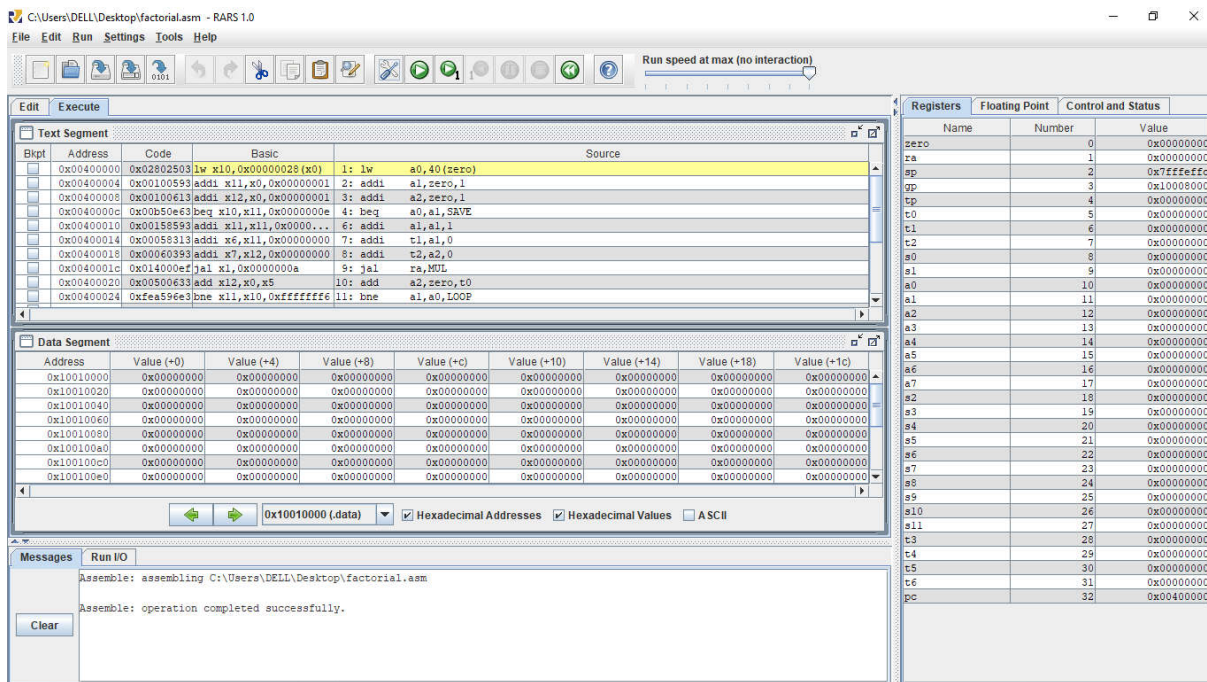
- Đầu tiên, ta vào File → chọn New, trang dùng để soạn thảo chương trình hợp ngữ sẽ xuất hiện tại tab Edit. Ở đây ta tiến hành nhập chương trình dưới dạng hợp ngữ.
- Nếu đã có sẵn file dưới dạng .asm (hoặc .s), ta có thể chọn File → Open để mở file đó lên.
- Ta có thể code chương trình bằng mã giả hay bất cứ lệnh nào có trong bảng tập lệnh.



- Sau khi đã viết xong chương trình, vào File → chọn Save để lưu và đặt tên.

- Để biên dịch, ta nhấn vào biểu tượng  trên thanh công cụ.


- Sau khi ấn, màn hình sẽ chuyển sang tab Execute như hình:





Trong hình vẽ:

- Cột Address là địa chỉ của mỗi lệnh trong IMEM (mặc định địa chỉ bắt đầu của IMEM trong chương trình mô phỏng là 0x00400000).
- Cột code là mã hex của lệnh.
- Cột basic là lệnh gốc của máy CPU tương ứng với lệnh bên cột Source.


- Khung Data Segment mô phỏng DMEM của CPU bắt đầu từ địa chỉ 0x10010000 và mỗi địa chỉ tiếp theo sẽ cộng thêm 4.

- Để xem mỗi lệnh chạy như thế nào, ta nhấn vào nút  để chạy từng lệnh hoặc

ấn vào nút  để chạy tất cả các lệnh trong chương trình. Trong quá trình chạy, ta sẽ quan sát được quá trình các thanh ghi và DMEM thay đổi dữ liệu.

Lưu ý: Khi chạy từng lệnh, ta hạn chế nhấn nút  với tốc độ nhanh vì nó sẽ gây ra treo máy

- Để biên dịch ra file .txt chứa mã máy dùng để nạp vào chạy mô phỏng dạng sóng,

ta chọn nút , sau đó chọn tên cần lưu.

b. Chương trình mẫu

Trong thư mục mà sinh viên được gửi sẽ có 4 file:

- Chương trình rars_v0
- File factorial.asm và factorial.txt
- File IMEM.v

Trong đó:

- File factorial.asm chứa chương trình tính số giai thừa dưới mạng hợp ngữ và file factorial.txt là file chứa mã máy đã được biên dịch.
- File IMEM.v là file verilog viết cho IMEM của RISC-V.

Để nạp chương trình trong file .txt cho IMEM ta làm như sau:

- Mở file bằng modelSim (hoặc Quartus,...).

```

module IMEM(inst,PC);
  parameter INST_WIDTH_LENGTH = 32;
  parameter PC_WIDTH_LENGTH = 32;
  parameter MEM_WIDTH_LENGTH = 32;
  parameter MEM_DEPTH = 1<<18;
  output reg [INST_WIDTH_LENGTH-1:0] inst;
  input [PC_WIDTH_LENGTH-1:0] PC;

  /***** Instruction Memory *****/
  reg [MEM_WIDTH_LENGTH-1:0] IMEM[0:MEM_DEPTH-1];

  wire [17:0] pWord;
  wire [1:0] pByte;

  assign pWord = PC[19:2];
  assign pByte = PC[1:0];

  initial begin
    $readmemh("D:/Documents_Study/Computer Architecture/RISC-V_project/TestCode/pipeline_test.txt", IMEM);
  end

  always@(PC)
  begin
    if (pByte == 2'b00)
      inst <= IMEM[pWord];
    else
      inst <= 'hz;
    end
  end
endmodule

```

- Tại dòng:

```
$readmemh("D:/Documents_Study/Computer Architecture/RISC-V_project/TestCode/pipeline_test.txt", IMEM);
```

Ta thay phần nằm trong dấu nháy đôi bằng đường dẫn chữ file .txt.

Nếu thay đổi tên file IMEM thì ta phải thay đổi phần đằng sau dấu phẩy thành tên tương ứng.