

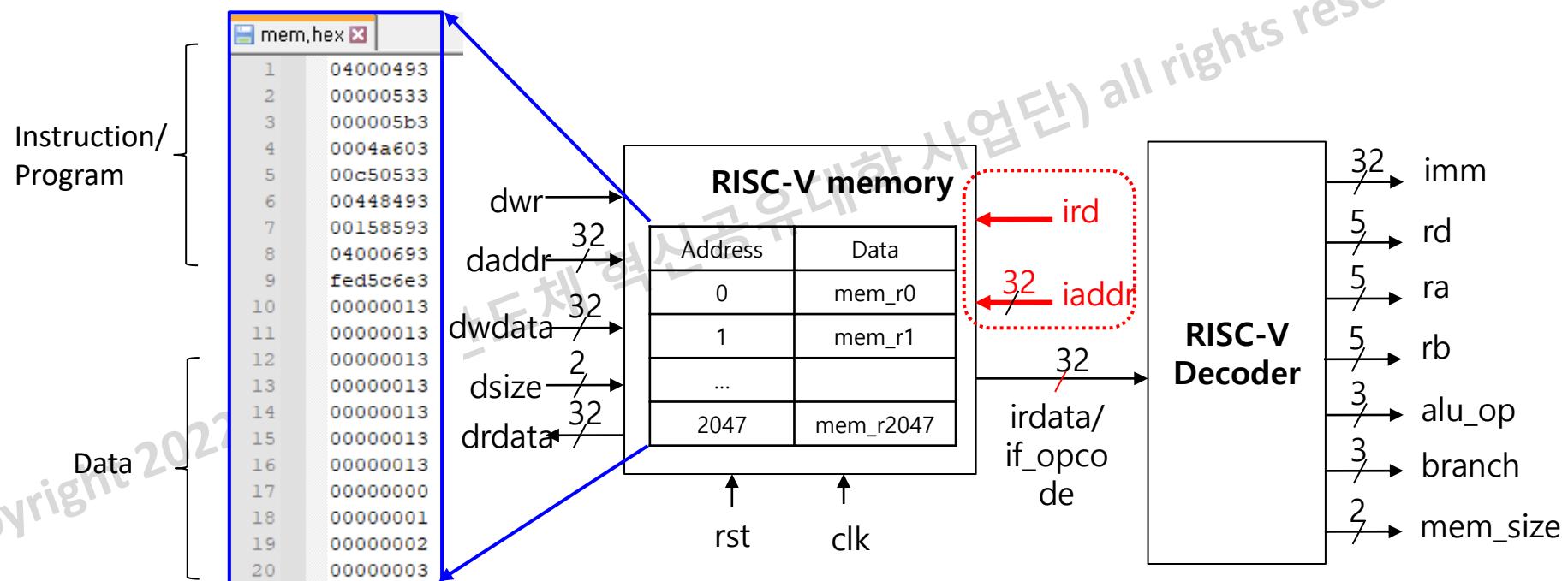
Program Counter, Branch Instructions, RISC-V core

Xuan-Truong Nguyen
2022.12.28 (Thu)



Recap: Memory, decoder

- Memory model: Instruction/Program, Data
- Instruction register, decoder
- What is the next instruction?
 - iaddr?



Lecture plan

Today we will:

- Review the RISC-V instruction format
- Introduce the program counter
- Introduce branch instructions
- Labs
 - Lab 1: Program counter
 - Lab 2: Program counter (Extended)
 - Lab 3: RISC-V core sim

Road map

RISC-V
RISC-V Simulator

Lab 1: Program Counter

Lab 2: Program Counter
(Extended)

Lab 3: RISC-V core sim

RISC-V Core Instruction Formats

- 32-bit instruction: Aligned on a four-byte boundary in memory.
- Formats: R, I, S, B, U, J-type

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R-type					funct7			rs2			rs1		funct3		rd			opcode															
I-type					imm[11:0]					rs1		funct3		rd			opcode																
S-type					imm[11:5]			rs2			rs1		funct3		imm[4:0]			opcode															
B-type					imm[12 10:5]			rs2			rs1		funct3		imm[4:1 11]			opcode															
U-type					imm[31 12]											rd			opcode														
J-type					imm[20 10:1 11 19:12]										rd			opcode															

RISC-V code generation

- Venus: A RISC-V Simulator
- <https://www.kvakil.me/venus/>

```
int A[64];
int sum = 0;
for (int i=0; i<64; i++)
    sum += A[i];
```

C code



```
addi x9, x0, 64 # x9=&A[0]
add x10, x0, x0 # sum=0
add x11, x0, x0 # i=0
Loop:
lw x12, 0(x9) # x12=A[i]
add x10,x10,x12 # sum+=
addi x9,x9,4 # &A[i++]
addi x11,x11,1 # i++
addi x13,x0,64 # x13=64
blt x11,x13,Loop
```

Assembly code

RISC-V code generation

- Manually generate the RISC-V compiled code from Assembly codes

```
addi x9, x0, 64 # x9=&A[0]
add x10, x0, x0 # sum=0
add x11, x0, x0 # i=0
Loop:
lw x12, 0(x9) # x12=A[i]
add x10,x10,x12 # sum+=
addi x9,x9,4 # &A[i++]
addi x11,x11,1 # i++
addi x13,x0,64 # x13=64
blt x11,x13,Loop
```



mem,hex	
1	04000493
2	00000533
3	000005b3
4	0004a603
5	00c50533
6	00448493
7	00158593
8	04000693
9	fed5c6e3
10	00000013
11	00000013
12	00000013
13	00000013
14	00000013
15	00000013
16	00000013
17	00000000
18	00000001
19	00000002
20	00000003

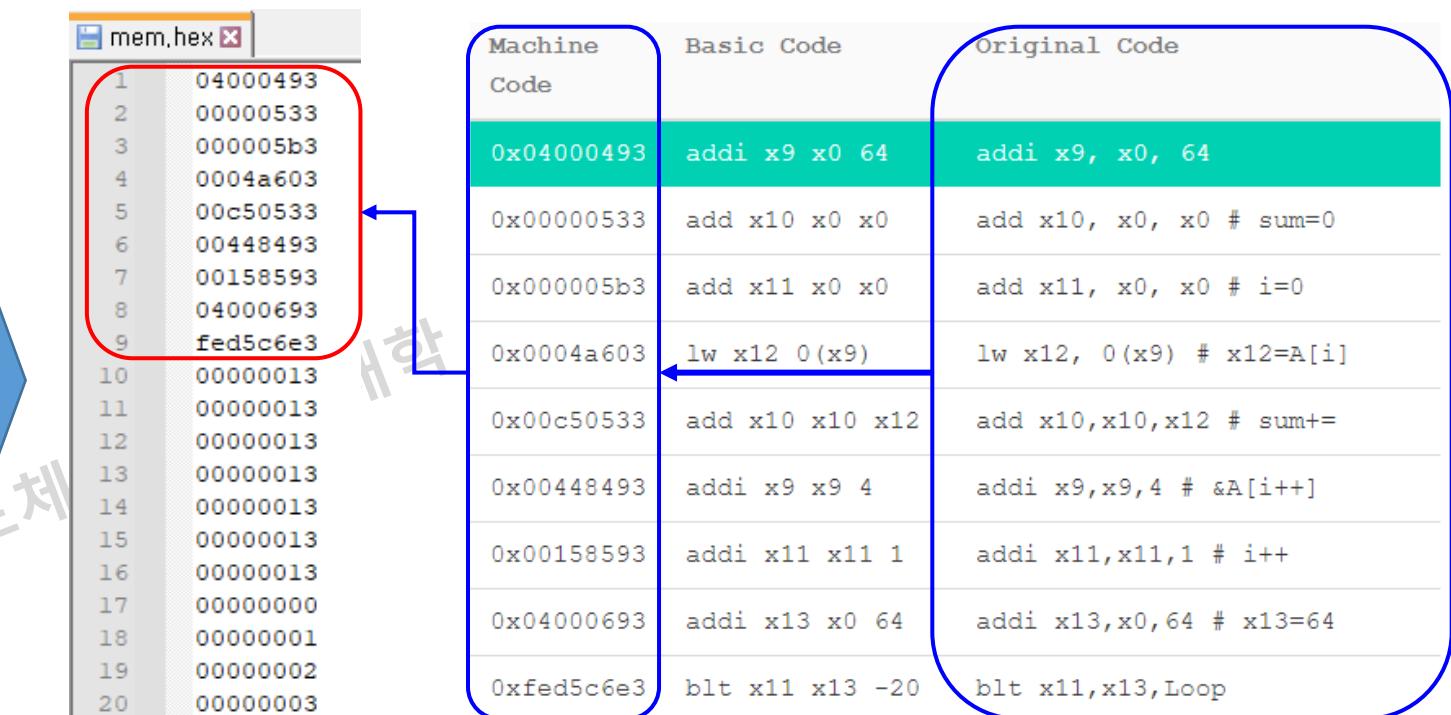
Instruction

Data

RISC-V code generation

- Manually generate the RISC-V codes from Assembly codes

```
addi x9, x0, 64 # x9=&A[0]
add x10, x0, x0 # sum=0
add x11, x0, x0 # i=0
Loop:
lw x12, 0(x9) # x12=A[i]
add x10,x10,x12 # sum+=
addi x9,x9,4 # &A[i++]
addi x11,x11,1 # i++
addi x13,x0,64 # x13=64
blt x11,x13,Loop
```



Execute this program

Control buttons



Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0x04000493	addi x9 x0 64	addi x9, x0, 64
0x00000533	add x10 x0 x0	add x10, x0, x0 # sum=0
0x000005b3	add x11 x0 x0	add x11, x0, x0 # i=0
0x0004a603	lw x12 0(x9)	lw x12, 0(x9) # x12=A[i]
0x00c50533	add x10 x10 x12	add x10,x10,x12 # sum+=
0x00448493	addi x9 x9 4	addi x9,x9,4 # &A[i++]
0x00158593	addi x11 x11 1	addi x11,x11,1 # i++
0x04000693	addi x13 x0 64	addi x13,x0,64 # x13=64
0xfed5c6e3	blt x11 x13 -20	blt x11,x13,Loop

Instructions

console output

Registers	Memory
zero	0
ra	0
(x1)	
sp	2147483632
(x2)	
gp	268435456
(x3)	
tp	0
(x4)	
t0	0
(x5)	
t1	0
(x6)	
t2	0
(x7)	
s0	0
(x8)	
s1	0
(x9)	
a0	0
(x10)	

Register file

Display Settings Unsigned

Unsigned integer

Execute this program

Current instruction
Next instruction

Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0x04000493	addi x9 x0 64	addi x9, x0, 64
0x000000533	add x10 x0 x0	add x10, x0, x0 # sum=0
0x000005b3	add x11 x0 x0	add x11, x0, x0 # i=0
0x0004a603	lw x12 0(x9)	lw x12, 0(x9) # x12=A[i]
0x00c50533	add x10 x10 x12	add x10,x10,x12 # sum+=
0x00448493	addi x9 x9 4	addi x9,x9,4 # &A[i++]
0x00158593	addi x11 x11 1	addi x11,x11,1 # i++
0x04000693	addi x13 x0 64	addi x13,x0,64 # x13=64
0xfed5c6e3	blt x11 x13 -20	blt x11,x13,Loop

console output

Registers Memory

zero	0
ra	0
(x1)	
sp	2147483632
(x2)	
gp	268435456
(x3)	
tp	0
(x4)	
t0	0
(x5)	
t1	0
(x6)	
t2	0
(x7)	
s0	0
(x8)	
s1	64
(x9)	
a0	0
(x10)	

Display Settings Unsigned

x9 is updated

Execute this program

Copyright 2022. (자)

Current instruction
Next instruction

Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0x04000493	addi x9 x0 64	addi x9, x0, 64
0x00000533	add x10 x0 x0	add x10, x0, x0 # sum=0
0x000005b3	add x11 x0 x0	add x11, x0, x0 # i=0
0x0004a603	lw x12 0(x9)	lw x12, 0(x9) # x12=A[i]
0x00c50533	add x10 x10 x12	add x10,x10,x12 # sum+=
0x00448493	addi x9 x9 4	addi x9,x9,4 # &A[i++]
0x00158593	addi x11 x11 1	addi x11,x11,1 # i++
0x04000693	addi x13 x0 64	addi x13,x0,64 # x13=64
0xfed5c6e3	blt x11 x13 -20	blt x11,x13,Loop

Registers Memory

zero	0
ra (x1)	0
sp (x2)	2147483632
gp (x3)	268435456
tp (x4)	0
t0 (x5)	0
t1 (x6)	0
t2 (x7)	0
s0 (x8)	0
s1 (x9)	64
a0 (x10)	0

Display Settings Unsigned

Console output

x10 is updated

Execute this program

Current instruction
Next instruction

Run Step Prev Reset Dump

Machine Code Basic Code Original Code

0x04000493	addi x9 x0 64	addi x9, x0, 64
0x00000533	add x10 x0 x0	add x10, x0, x0 # sum=0
0x000005b3	add x11 x0 x0	add x11, x0, x0 # i=0
0x0004a603	lw x12 0(x9)	lw x12, 0(x9) # x12=A[i]
0x00c50533	add x10 x10 x12	add x10,x10,x12 # sum+=
0x00448493	addi x9 x9 4	addi x9,x9,4 # &A[i++]
0x00158593	addi x11 x11 1	addi x11,x11,1 # i++
0x04000693	addi x13 x0 64	addi x13,x0,64 # x13=64
0xfed5c6e3	blt x11 x13 -20	blt x11,x13,Loop

console output

Registers Memory

zero	0
ra (x1)	0
sp (x2)	2147483632
gp (x3)	268435456
tp (x4)	0
t0 (x5)	0
t1 (x6)	0
t2 (x7)	0
s0 (x8)	0
s1 (x9)	64
a0 (x10)	0

Display Settings Unsigned

Execute this program

Current instruction
Next instruction

Run Step Prev Reset Dump

↓

Machine Code	Basic Code	Original Code
0x04000493	addi x9 x0 64	addi x9, x0, 64
0x00000533	add x10 x0 x0	add x10, x0, x0 # sum=0
0x000005b3	add x11 x0 x0	add x11, x0, x0 # i=0
0x0004a603	lw x12 0(x9)	lw x12, 0(x9) # x12=A[i]
0x00c50533	add x10 x10 x12	add x10,x10,x12 # sum+=
0x00448493	addi x9 x9 4	addi x9,x9,4 # &A[i++]
0x00158593	addi x11 x11 1	addi x11,x11,1 # i++
0x04000693	addi x13 x0 64	addi x13,x0,64 # x13=64
0xfed5c6e3	blt x11 x13 -20	blt x11,x13,Loop

console output

Registers Memory

zero	0
ra	0
(x1)	
sp	2147483632
(x2)	
gp	268435456
(x3)	
tp	0
(x4)	
t0	0
(x5)	
t1	0
(x6)	
t2	0
(x7)	
s0	0
(x8)	
s1	68
(x9)	
a0	0
(x10)	

Display Settings Unsigned

Execute this program

Copyright 2022. (자)

Next instruction

Current instruction

Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0x04000493	addi x9 x0 64	addi x9, x0, 64
0x00000533	add x10 x0 x0	add x10, x0, x0 # sum=0
0x000005b3	add x11 x0 x0	add x11, x0, x0 # i=0
0x0004a603	lw x12 0(x9)	lw x12, 0(x9) # x12=A[i]
0x00c50533	add x10 x10 x12	add x10,x10,x12 # sum+=
0x00448493	addi x9 x9 4	addi x9,x9,4 # &A[i++]
0x00158593	addi x11 x11 1	addi x11,x11,1 # i++
0x04000693	addi x13 x0 64	addi x13,x0,64 # x13=64
0xfed5c6e3	blt x11 x13 -20	blt x11,x13,Loop

Registers Memory

zero	0
ra	0
(x1)	
sp	2147483632
(x2)	
gp	268435456
(x3)	
tp	0
(x4)	
t0	0
(x5)	
t1	0
(x6)	
t2	0
(x7)	
s0	0
(x8)	
s1	68
(x9)	
a0	0
(x10)	

Display Settings Unsigned

console output

Road map

RISC-V
RISC-V Simulator

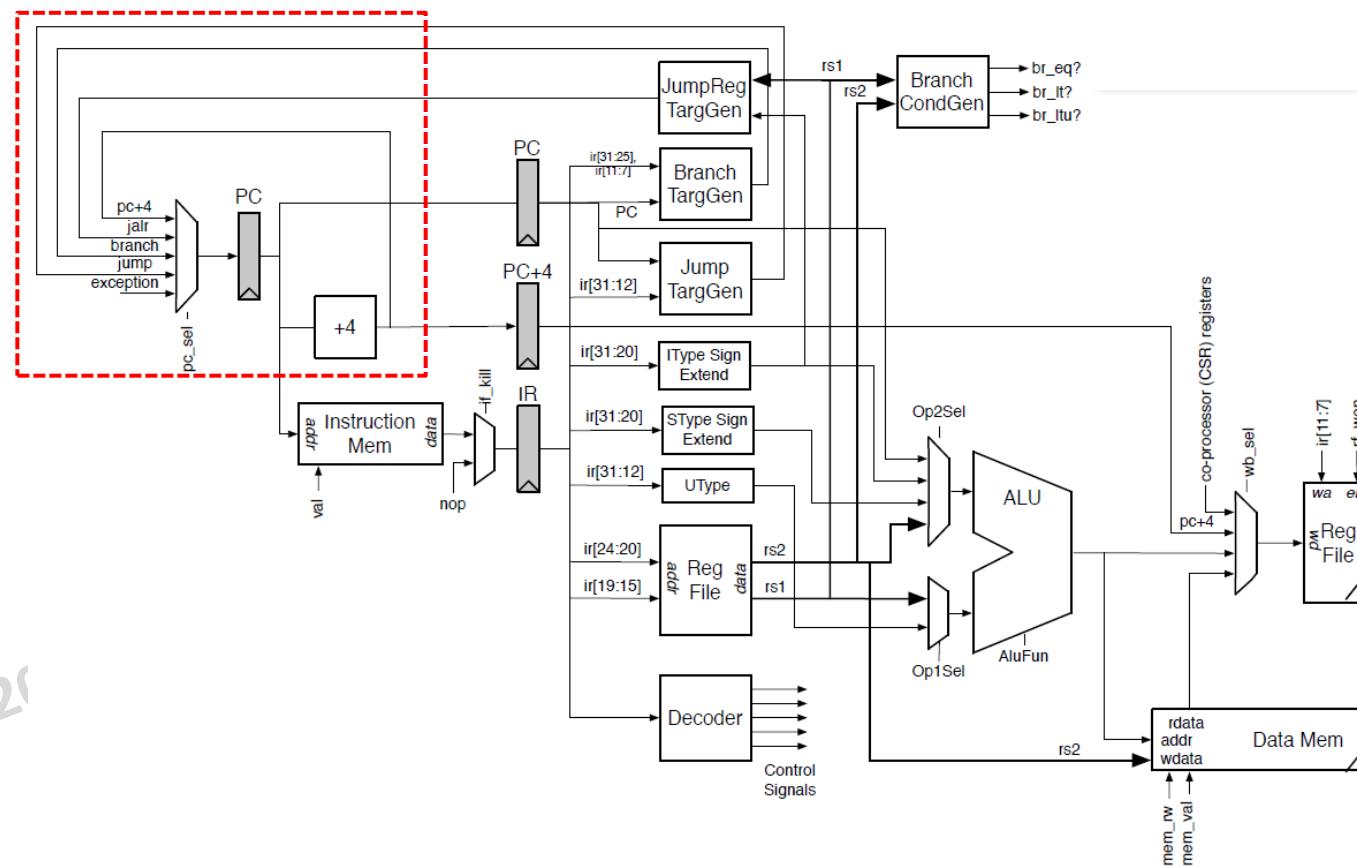
Lab 1: Program Counter

Lab 2: Program Counter
(Extended)

Lab 3: RISC-V core sim

What is the program counter?

- Program Counter (PC): Store the next instruction address



Program counter

- Normal instruction
 - Go to the next instruction by incrementing 4
 - $PC \leftarrow PC + 4$
- Conditional branch instructions
 - Move to the next instruction by an offset
 - $PC \leftarrow PC + \text{offset}$
- JAL: Jump and link
 - $PC \leftarrow \text{target address}$

Conditional Branch Instructions

- Conditional Branches:

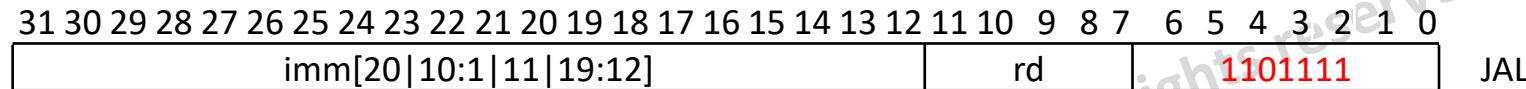
- Branch if = zero (BEQ)
- Branch if \neq zero (BNE)
- Branch if $<$ (signed) (BLT)
- Branch if \geq (signed) (BGE)
- Branch if $<$ (unsigned) (BLTU)
- Branch if \geq (unsigned) (BGEU)

B-type		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Instruction	Syntax	Description												Execution																			
BEQ	beq rs1, rs2, imm	Branch if = zero												PC \leq ($\text{reg}[rs1] == \text{reg}[rs2]$) ? PC + imm : (PC+4)																			
BNE	bne rs1, rs2, imm	Branch if \neq zero												PC \leq ($\text{reg}[rs1] != \text{reg}[rs2]$) ? PC + imm : (PC+4)																			
BLT	blt rs1, rs2, imm	Branch if $<$ (signed)												PC \leq ($\text{reg}[rs1] <_s \text{reg}[rs2]$) ? PC + imm : (PC+4)																			
BGE	bge rs1, rs2, imm	Branch if \geq (signed)												PC \leq ($\text{reg}[rs1] \geq_s \text{reg}[rs2]$) ? PC + imm : (PC+4)																			
BLTU	bltu rs1, rs2, imm	Branch if $<$ (Unsigned)												PC \leq ($\text{reg}[rs1] <_u \text{reg}[rs2]$) ? PC + imm : (PC+4)																			
BGEU	bgeu rs1, rs2, imm	Branch if \geq (Unsigned)												PC \leq ($\text{reg}[rs1] \geq_u \text{reg}[rs2]$) ? PC + imm : (PC+4)																			

Unconditional Branch Instructions

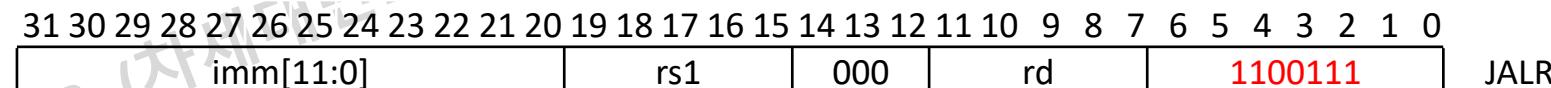
- Jump and Link (JAL)

- jal rd, imm
- $\text{reg}[rd] \leq \text{PC} + 4$
- $\text{PC} \leq \text{PC} + \text{imm}$



- Jump and Link Register (JALR)

- jal rd, imm
- $\text{reg}[rd] \leq \text{PC} + 4$
- $\text{PC} \leq \{(\text{reg}[rs1] + \text{imm})[31:1], 1'b0\}$



- Return

- $\text{PC} \leq \text{reg}[ra]$

Example

- A simple program calculates a sum of all elements in an array.

```
int A[64];  
  
int sum = 0;  
  
for (int i=0; i<64; i++)  
    sum += A[i];  
  
addi x9, x0, 64 # x9=&A[0]  
add x10, x0, x0 # sum=0  
add x11, x0, x0 # i=0  
  
Loop:  
    lw x12, 0(x9) # x12=A[i]  
    add x10,x10,x12 # sum+=  
    addi x9,x9,4 # &A[i++]  
    addi x11,x11,1 # i++  
    addi x13,x0,64 # x13=64  
    blt x11,x13,Loop
```

Example

- Program counter
 - Initialization: Store the base address of the Main function

PC →

Address	Instruction	Basic code	Original code	Comments
0000	00040493	addi x9 x0 64	addi x9, x0, 64	# x9=&A[0]
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# x12=A[i]
0010	00c50533	add x10 x10 x12	add x10,x10,x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9,x9,4	# &A[i++]
0018	00158593	addi x11 x11 1	addi x11,x11,1	# i++
001C	04000693	addi x13 x0 64	addi x13,x0,64	# x13=64
0020	fed5c6e3	blt x11 x13 -20	blt x11,x13,Loop	# Branch

Example

- Program counter
 - Normal instruction: $PC \leftarrow PC + 4$

PC →

Address	Instruction	Basic code	Original code	Comments
0000	00040493	addi x9 x0 64	addi x9, x0, 64	# x9=&A[0]
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# x12=A[i]
0010	00c50533	add x10 x10 x12	add x10,x10,x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9,x9,4	# &A[i++]
0018	00158593	addi x11 x11 1	addi x11,x11,1	# i++
001C	04000693	addi x13 x0 64	addi x13,x0,64	# x13=64
0020	fed5c6e3	blt x11 x13 -20	blt x11,x13,Loop	# Branch

Example

- Program counter
 - Normal instruction: $PC \leftarrow PC + 4$

PC →

Address	Instruction	Basic code	Original code	Comments
0000	00040493	addi x9 x0 64	addi x9, x0, 64	# x9=&A[0]
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# x12=A[i]
0010	00c50533	add x10 x10 x12	add x10,x10,x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9,x9,4	# &A[i++]
0018	00158593	addi x11 x11 1	addi x11,x11,1	# i++
001C	04000693	addi x13 x0 64	addi x13,x0,64	# x13=64
0020	fed5c6e3	blt x11 x13 -20	blt x11,x13,Loop	# Branch

Example

- Program counter
 - Normal instruction: $PC \leftarrow PC + 4$

Address	Instruction	Basic code	Original code	Comments
0000	00040493	addi x9 x0 64	addi x9, x0, 64	# x9=&A[0]
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# x12=A[i]
0010	00c50533	add x10 x10 x12	add x10,x10,x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9,x9,4	# &A[i++]
0018	00158593	addi x11 x11 1	addi x11,x11,1	# i++
001C	04000693	addi x13 x0 64	addi x13,x0,64	# x13=64
0020	fed5c6e3	blt x11 x13 -20	blt x11,x13,Loop	# Branch

PC →

Example

- Program counter
 - Branch instruction: $PC \leftarrow PC + \text{offset} (-20)$
`blt x11 x13 -20`

⇒ "if x_{11} is smaller than x_{13} (BLT), move to the next instruction at $PC - 20$ "

PC →

Address	Instruction	Basic code	Original code	Comments
0000	00040493	addi x9 x0 64	addi x9, x0, 64	# $x_9 = &A[0]$
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# $x_{12} = A[i]$
0010	00c50533	add x10 x10 x12	add x10, x10, x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9, x9, 4	# $&A[i++]$
0018	00158593	addi x11 x11 1	addi x11, x11, 1	# i++
001C	04000693	addi x13 x0 64	addi x13, x0, 64	# $x_{13} = 64$
0020	fed5c6e3	blt x11 x13 -20	blt x11, x13, Loop	# Branch

Lab 1: Program counter

- Lab 1: Program counter
 - Implement a PC module for RISC-V
 - Run a simulation with time = 1000 ns
 - Show the output result

H/W Implementation (riscv_pc.v)

- Program counter
 - Inputs
 - clock, reset
 - Instruction read request
 - branch_taken flag, and a target address for a jump.
 - Output:
 - Address for the next instruction

```
module riscv_pc #(parameter PC_SIZE = 32)
(
    input clk_i,                                // Clock
    input reset_i,                               // Reset
    input ird,                                   // Instruction Read request
    input branch_taken_w,                         // Jump instruction
    input [PC_SIZE-1:0] jump_addr_w,             // Jump address
    output [PC_SIZE-1:0] if_next_addr_w // Next instruction
);

reg [PC_SIZE-1:0] if_addr_r;
always @(posedge clk_i or negedge reset_i) begin
    assign if_next_addr_w = branch_taken_w ? jump_addr_w: if_addr_r;
endmodule
```

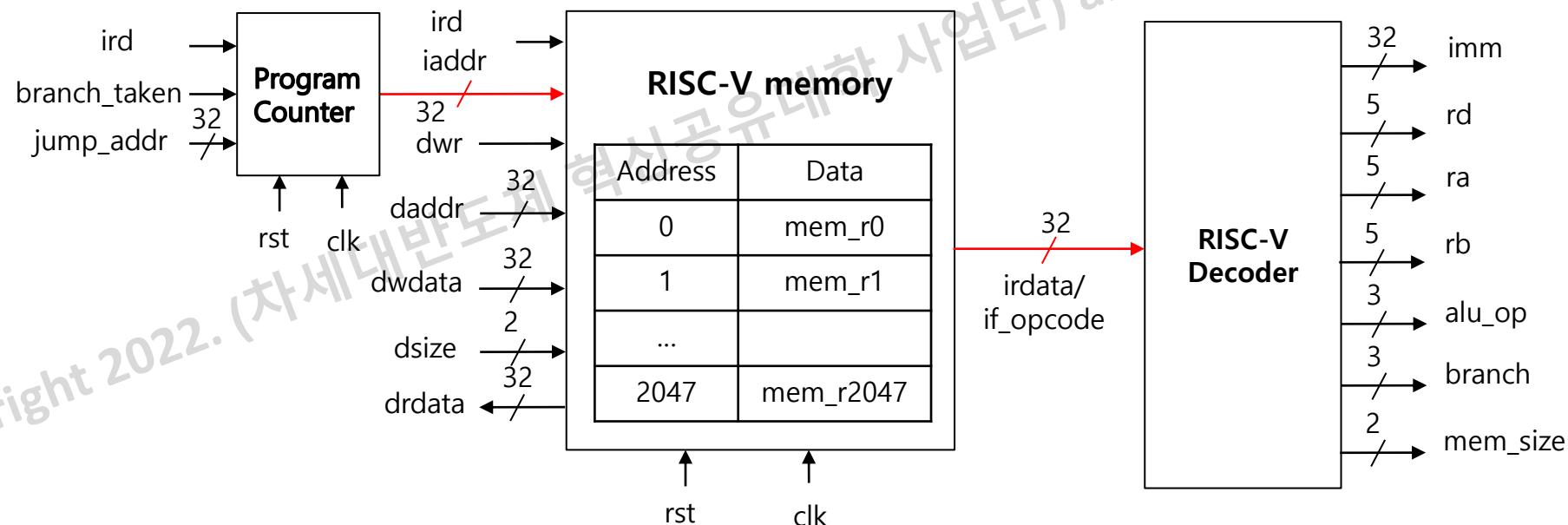
H/W Implementation (riscv_pc.v)

- Complete the missing codes to calculate the address for the next instruction

```
//-----  
// Program Counter  
//-----  
module riscv_pc #(  
    parameter RESET_SP = 32'h0000,  
    parameter PC_SIZE = 32)  
(  
    input clk_i,                                // Clock  
    input reset_i,                             // Reset  
    input ird,                                 // Instruction Read request  
    input branch_taken_w,                      // Jump instruction  
    input [PC_SIZE-1:0] jump_addr_w,           // Jump address  
    output [PC_SIZE-1:0] if_next_addr_w // Next instruction  
);  
  
reg [PC_SIZE-1:0] if_addr_r;  
always @(posedge clk i or negedge reset i) begin  
    if (~reset_i) begin  
        //Your code  
        //if_addr_r <= /*insert your code*/;  
    end  
    else begin  
        if (ird) begin  
            // Your code  
            //{{{{  
            if(branch_taken_w) begin  
                //if_addr_r <= /*insert your code*/;  
            end  
            else begin  
                //if_addr_r <= /*insert your code*/;  
            end  
            //}}}  
        end  
    end  
    end  
    end
```

Test bench (riscv_core_sim_tb.v)

- Three modules
 - Program counter: output the address (iaddr) to memory
 - Memory: get iaddr and return an instruction (irdata/if_opcode)
 - Decoder: parse an instruction



Test bench (riscv_pc_tb1.v)

- riscv_core_sim_tb1.v
 - Define internal signals
 - Clock, reset

```
module riscv_pc_tb1;
    reg reset_i;
    reg clk_i;

    // Input instruction
    reg [31:0] iaddr_i;
    reg      ird_i;
    reg [31:0] daddr_i;
    reg [31:0] dwdata_i;
    reg [1:0]  dszie_i;
    reg      drd_i;
    reg      dwr_i;
    // Outputs
    wire [31:0] irdata_o;
    wire [31:0] drdata_o;
    ...

```

```
wire [31:0] if_opcode_w;
assign if_opcode_w = irdata_o;
```

```
localparam
    SIZE_BYTE = 2'd0,
    SIZE_HALF = 2'd1,
    SIZE_WORD = 2'd2;

parameter p=10;
```

```
initial begin
    clk_i = 1'b0;
    forever #(p/2) clk_i = !clk_i;
end
```

```
initial begin
    reset_i = 1'b0;
    iaddr_i = 32'h0;
    ird_i = 32'h0;
    daddr_i = 32'h0;
    dwdata_i= 32'h0;
    drd_i = 1'b0;
    dwr_i = 1'b0;

    dszie_i=SIZE_WORD;

    branch_taken_w = 1'b0;
    jump_addr_w = 32'b0;
    #(4*p) reset_i = 1'b1;
```

Test bench (riscv_pc_tb1.v)

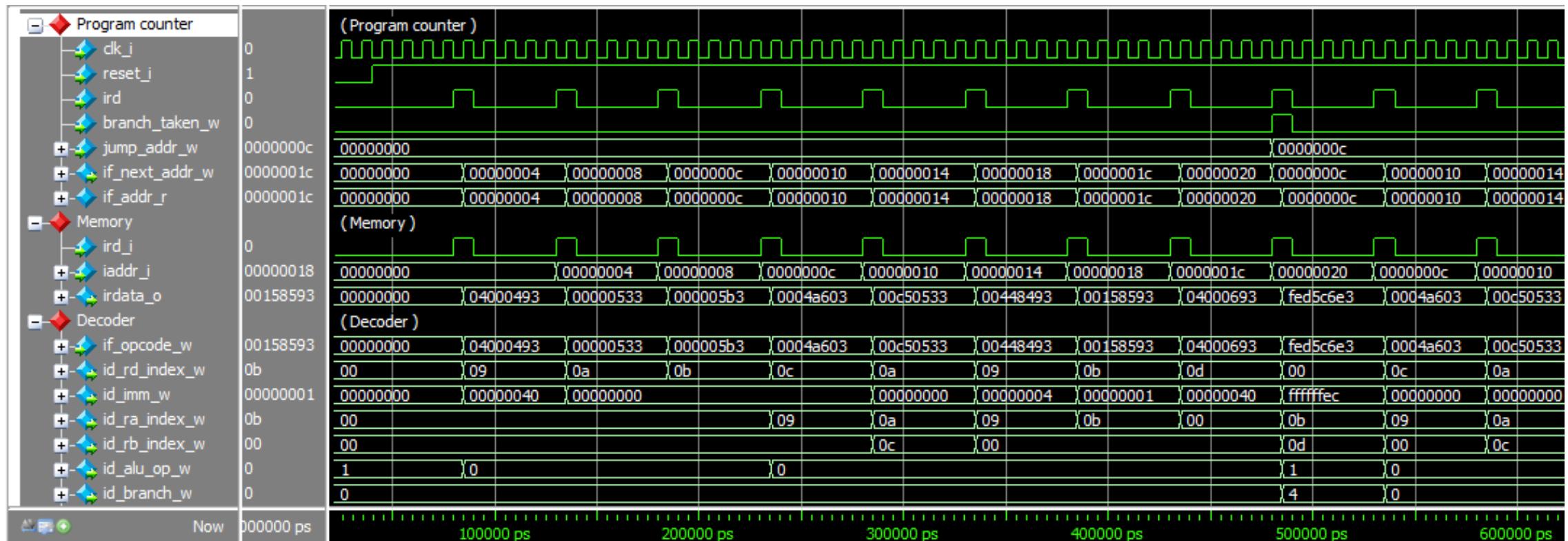
- Complete the missing code for test bench
 - Add connections for PC

```
//-----  
// Design  
//-----  
//{{{  
riscv_pc u_pc(  
    ./*input    *clk_i(clk_i)|  
    ,/*input    *reset_i(reset_i)  
    // Your code  
    //{{{  
    //}}} }  
};  
// Memory  
riscv_memory  
u_memory (  
    .clk_i(clk_i),  
    .reset_i(reset_i),  
    .iaddr_i(iaddr_i),  
    .ird_i(ird_i),  
    .daddr_i(daddr_i),  
    .dwdata_i(dwdata_i),  
    .dszie_i(dszie_i),  
    .drd_i(drd_i),  
    .dwr_i(dwr_i),  
    .irdata_o(irdata_o),  
    .drdata_o(drdata_o)  
);
```

Copyright 2022. (차세대반도체 허
수) 김민수

Waveform

- Do a simulation with time = 1000 ns



Test cases

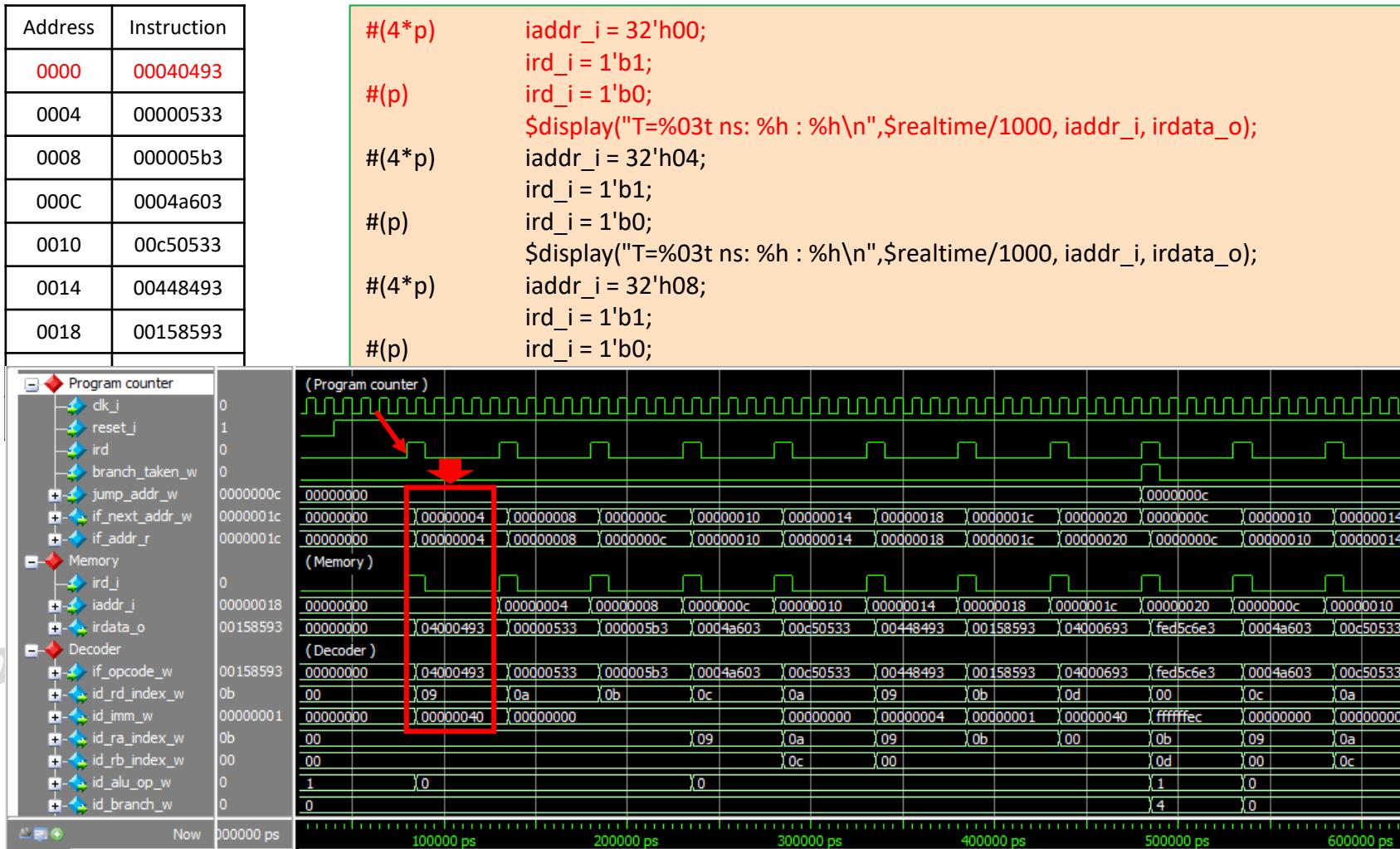
- Loop initialization
 - Three instructions

Address	Instruction
0000	00040493
0004	00000533
0008	000005b3
000C	0004a603
0010	00c50533
0014	00448493
0018	00158593
001C	04000693
0020	fed5c6e3

```
#(4*p)    iaddr_i = 32'h00;  
          ird_i = 1'b1;  
          ird_i = 1'b0;  
          $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr_i, irdata_o);  
#(p)  
#(4*p)    iaddr_i = 32'h04;  
          ird_i = 1'b1;  
          ird_i = 1'b0;  
          $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr_i, irdata_o);  
#(p)  
#(4*p)    iaddr_i = 32'h08;  
          ird_i = 1'b1;  
          ird_i = 1'b0;  
          $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr_i, irdata_o);  
#(p)
```

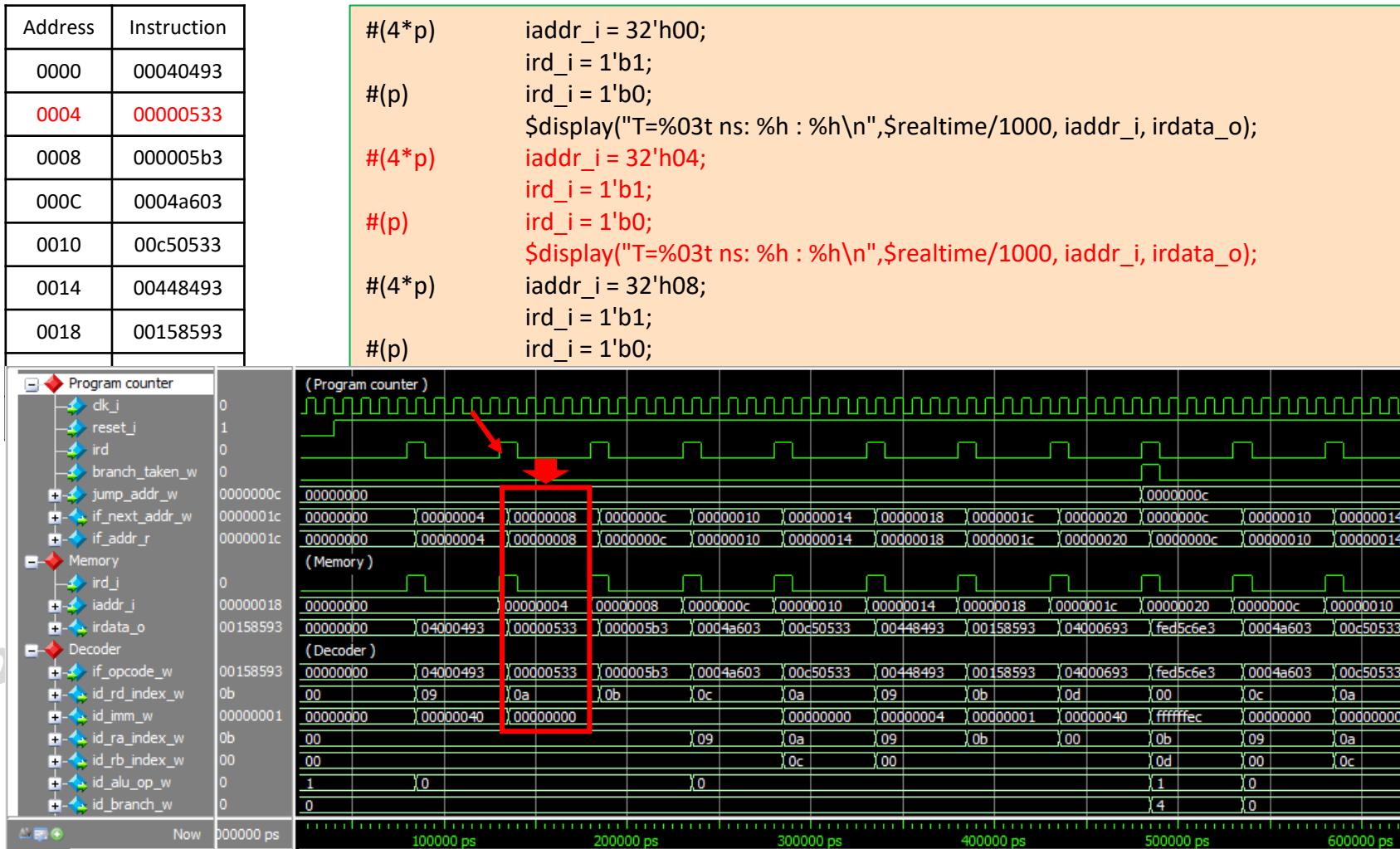
Waveform

- Initialization



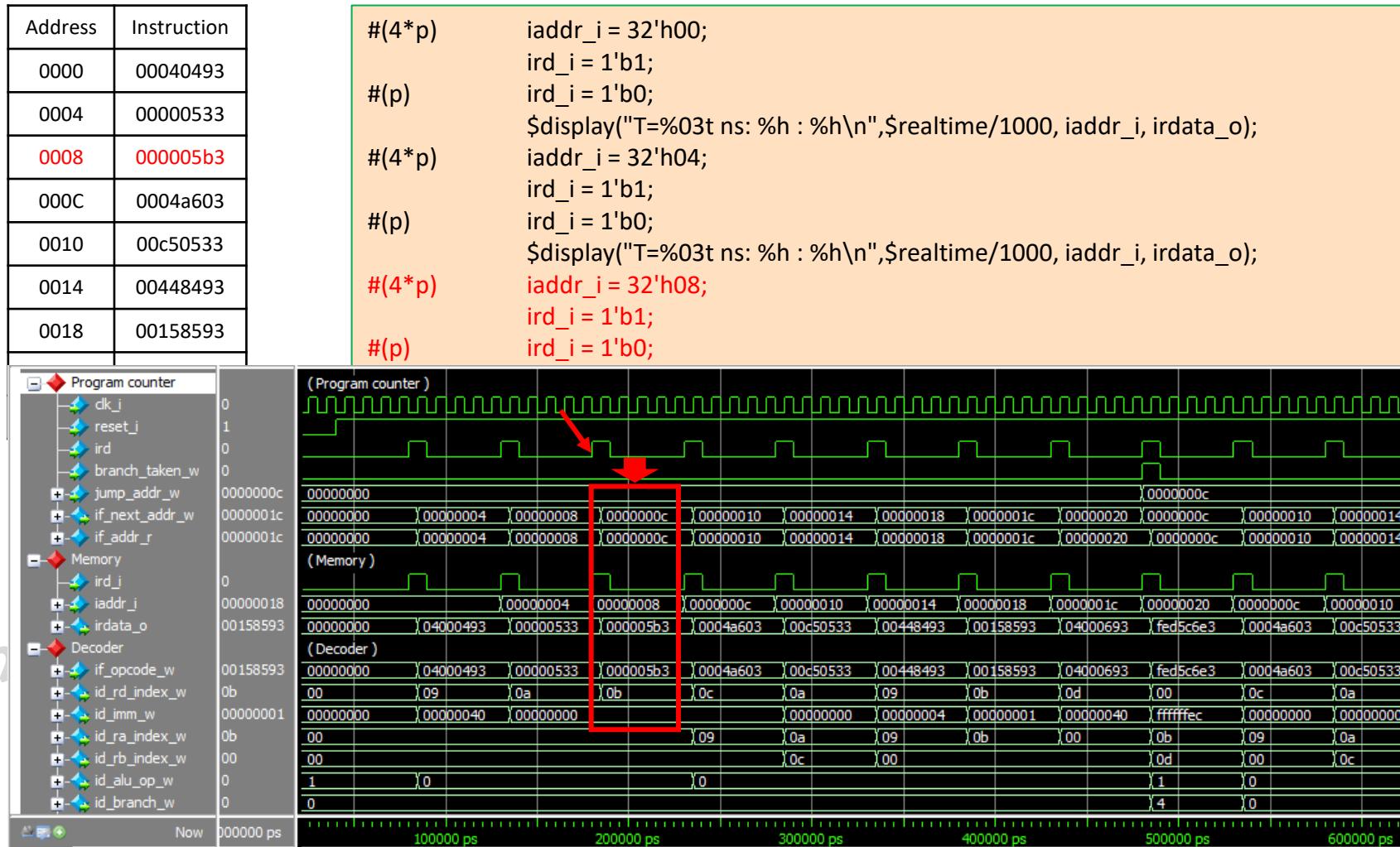
Waveform

■ Initialization



Waveform

- Initialization



Test cases

- Loop

Address	Instruction
0000	00040493
0004	00000533
0008	000005b3
000C	0004a603
0010	00c50533
0014	00448493
0018	00158593
001C	04000693
0020	fed5c6e3

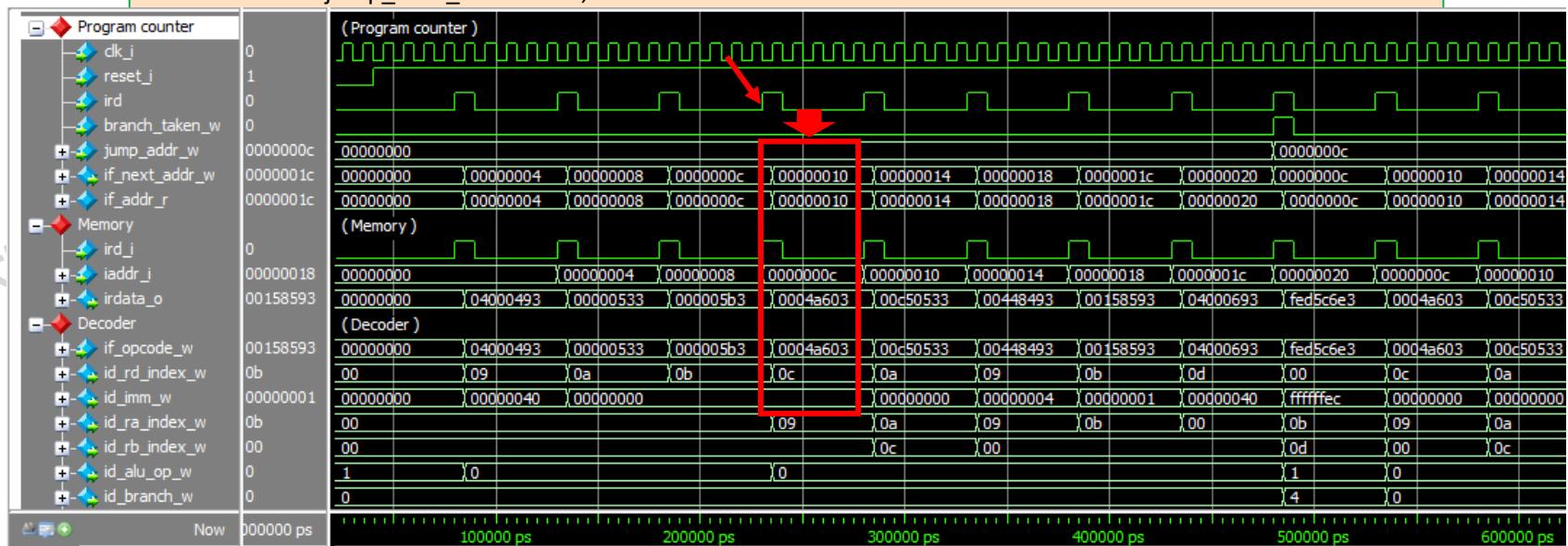
```
for (i = 0; i < 64; i = i+1) begin
#(4*p)      iaddr_i = 32'h0C;
            ird_i = 1'b1;
#(p)        iaddr_i = 32'h0C;
            ird_i = 1'b0;
            $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr_i, irdata_o);
#(4*p)      iaddr_i = 32'h10;
            ird_i = 1'b1;
#(p)        ird_i = 1'b0;
.....
#(4*p)      iaddr_i = 32'h20;
            ird_i = 1'b1;
            branch_taken_w = 1'b1;
            jump_addr_w = 32'h0C;
            ird_i = 1'b0;
            branch_taken_w = 1'b0;
            $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr_i, irdata_o
end
```

Test cases

- Loop

Address	Instruction
0000	00040493
0004	00000533
0008	000005b3
000C	0004a603
0010	00c50533
0014	00448493
0018	00158593
001C	04000693
0020	fed5c6e3

```
for (i = 0; i < 64; i = i+1) begin
#(4*p)      iaddr_i = 32'h0C;
            ird_i = 1'b1;
#(p)        ird_i = 1'b0;
$display("T=%03t ns: %h : %h\n", $realtime/1000, iaddr_i, irdata_o);
#(4*p)      iaddr_i = 32'h10;
            ird_i = 1'b1;
#(p)        ird_i = 1'b0;
.....
#(4*p)      iaddr_i = 32'h20;
            ird_i = 1'b1;
branch_taken_w = 1'b1;
jump_addr_w = 32'h0C;
```

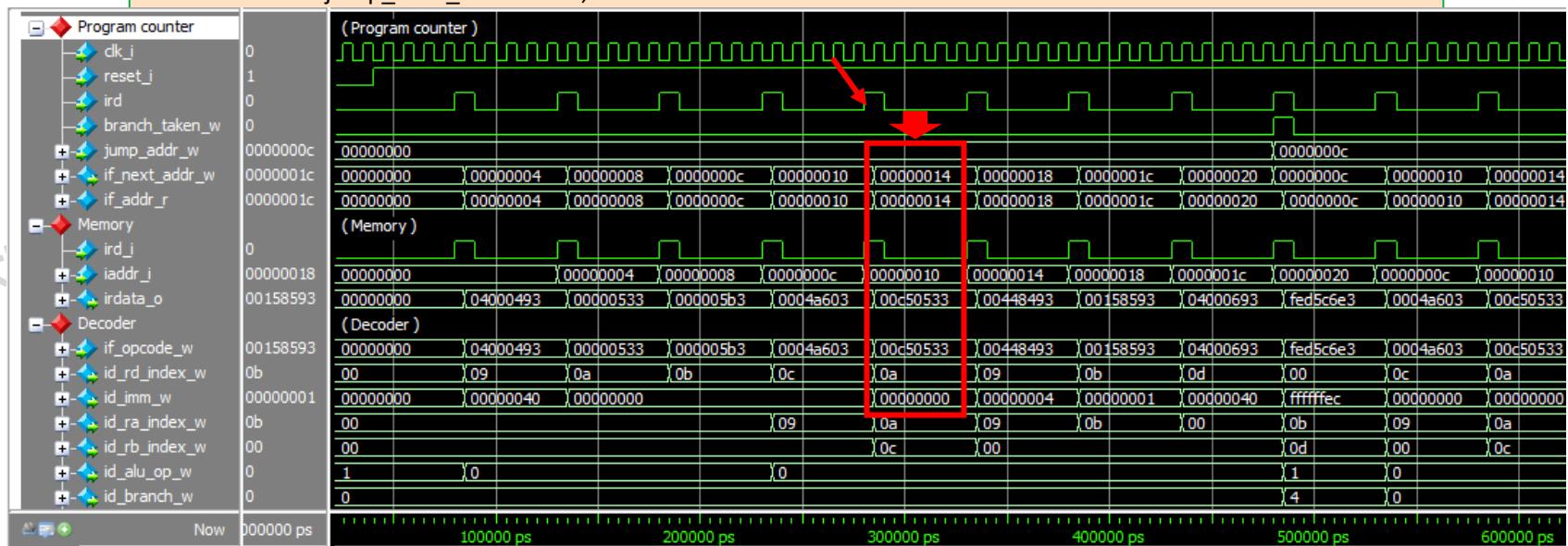


Test cases

- Loop

Address	Instruction
0000	00040493
0004	00000533
0008	000005b3
000C	0004a603
0010	00c50533
0014	00448493
0018	00158593
001C	04000693
0020	fed5c6e3

```
for (i = 0; i < 64; i = i+1) begin
#(4*p)      iaddr_i = 32'h0C;
            ird_i = 1'b1;
#(p)
            ird_i = 1'b0;
$display("T=%03t ns: %h : %h\n", $realtime/1000, iaddr_i, irdata_o);
#(4*p)      iaddr_i = 32'h10;
            ird_i = 1'b1;
#(p)
            ird_i = 1'b0;
.....
#(4*p)      iaddr_i = 32'h20;
            ird_i = 1'b1;
branch_taken_w = 1'b1;
jump_addr_w = 32'h0C;
```

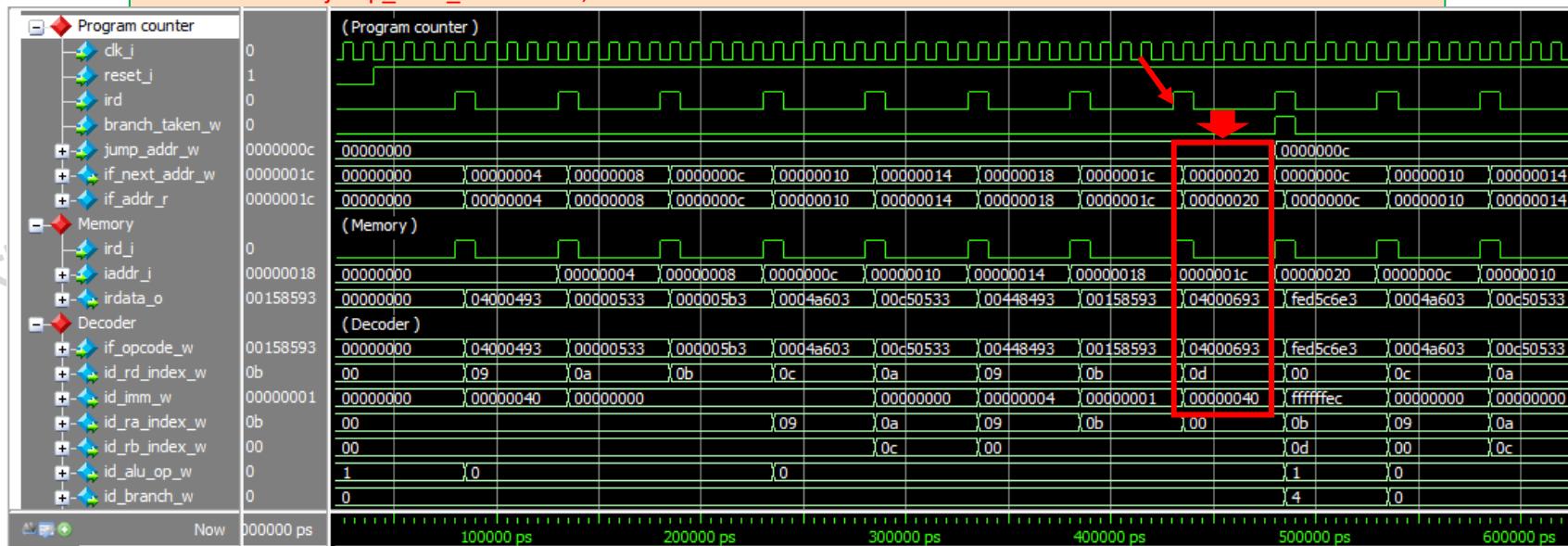


Test cases

- Loop

Address	Instruction
0000	00040493
0004	00000533
0008	000005b3
000C	0004a603
0010	00c50533
0014	00448493
0018	00158593
001C	04000693
0020	fed5c6e3

```
for (i = 0; i < 64; i = i+1) begin
#(4*p)      iaddr_i = 32'h0C;
            ird_i = 1'b1;
#(p)
            ird_i = 1'b0;
$display("T=%03t ns: %h : %h\n", $realtime/1000, iaddr_i, irdata_o);
#(4*p)      iaddr_i = 32'h10;
            ird_i = 1'b1;
#(p)
            ird_i = 1'b0;
#(4*p)      iaddr_i = 32'h20;
            ird_i = 1'b1;
branch_taken_w = 1'b1;
jump_addr_w = 32'h0C;
```



To do ...

- Implement riscv_pc.v and riscv_pc_tb1.v by completing the missing codes
- Do a simulation with time = 1000 ns
- Show the waveform

Copyright 2022. (차세대반도체 혁신공유대학 사업단) all rights reserved.

Road map

RISC-V
RISC-V Simulator

Lab 1: Program Counter

Lab 2: Program Counter
(Extended)

Lab 3: RISC-V core sim

Lab 2: Extended program counter

- Lab 2: Program counter
 - Implement a PC module for RISC-V
 - Run a simulation
 - Show the output result

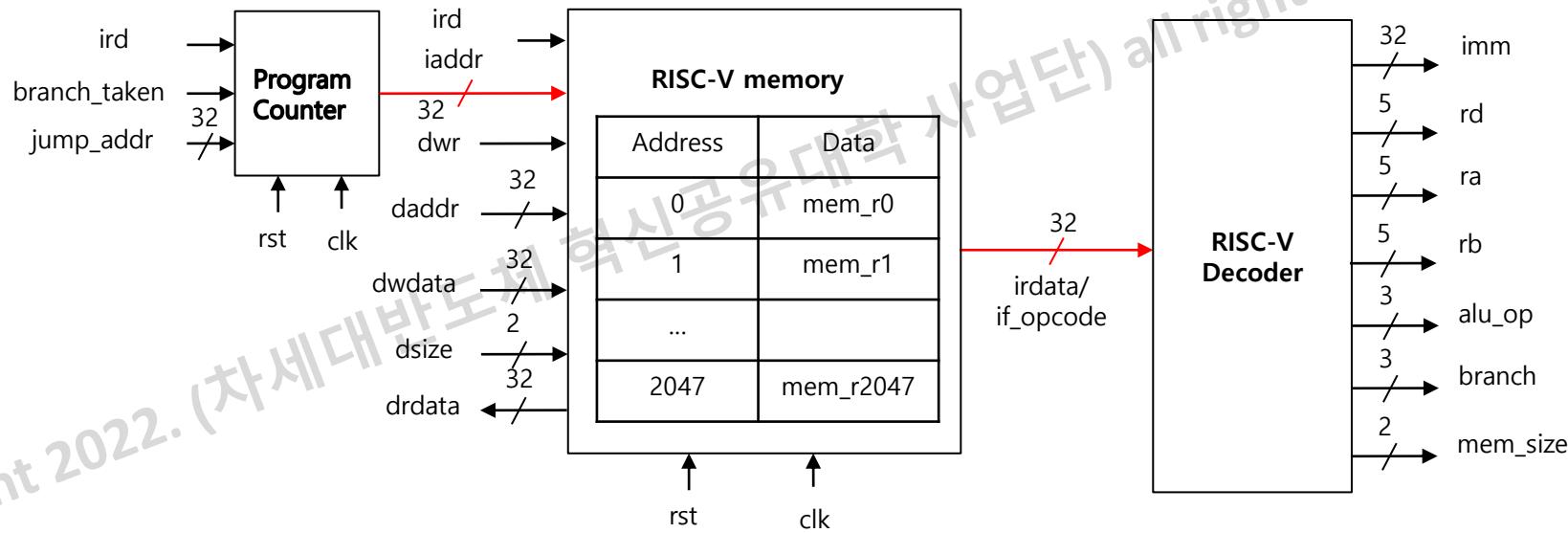
Test bench : ex2_program_counter

- Check : reset value assign (riscv_pc.v)

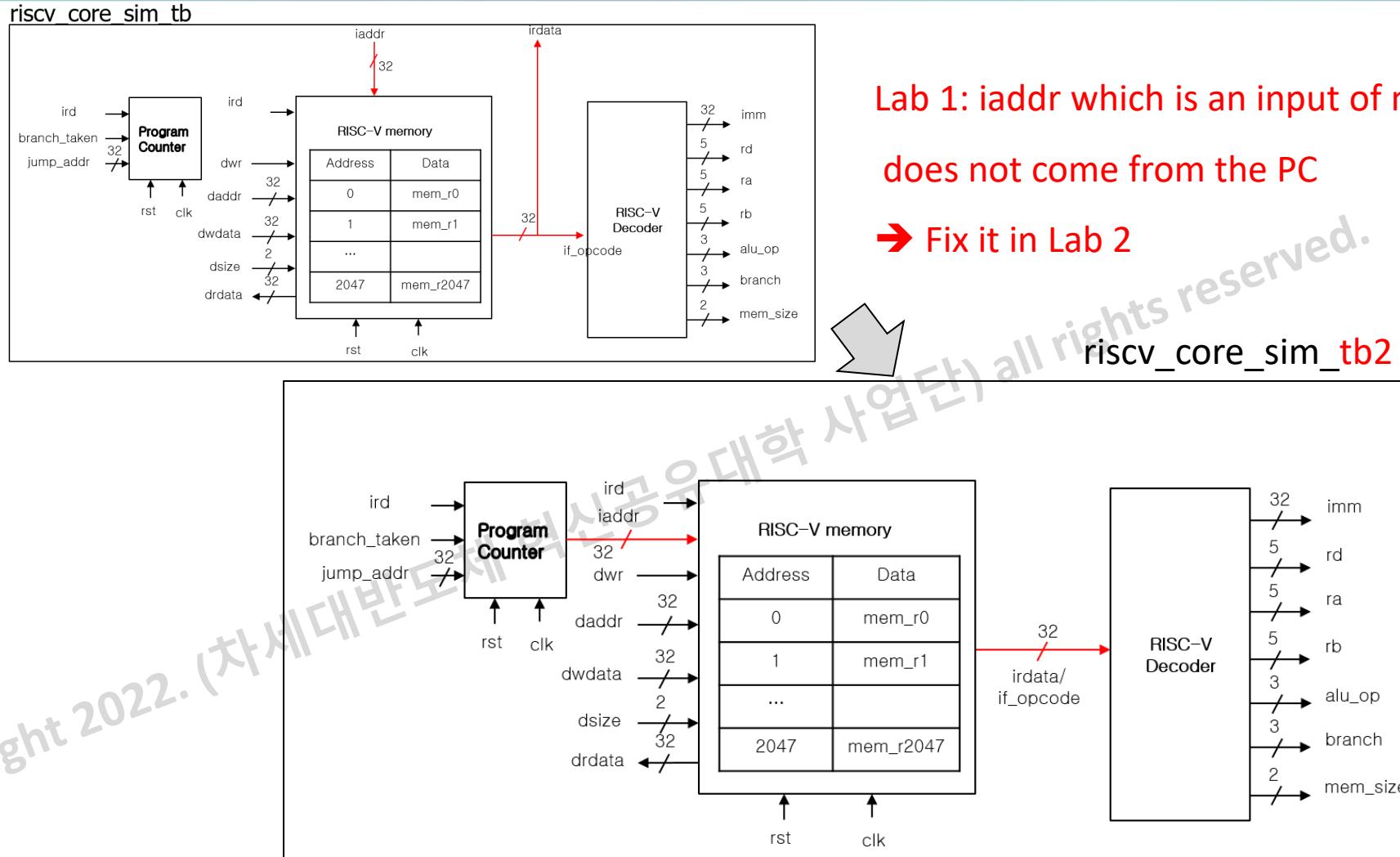
```
parameter RESET_SP = 32'h0000
reg [PC_SIZE-1:0] if_addr_r;
always @(posedge clk_i or negedge reset_i) begin
    if (~reset_i) begin
        if_addr_r <= RESET_SP;
    end
end
```

Test bench : ex2_program_counter

- Three modules with real connections
 - Program counter : to make next PC
 - Memory : to get address from Program counter
 - Decoder



Test bench: riscv_pc_tb1.v VS riscv_pc_tb2.v



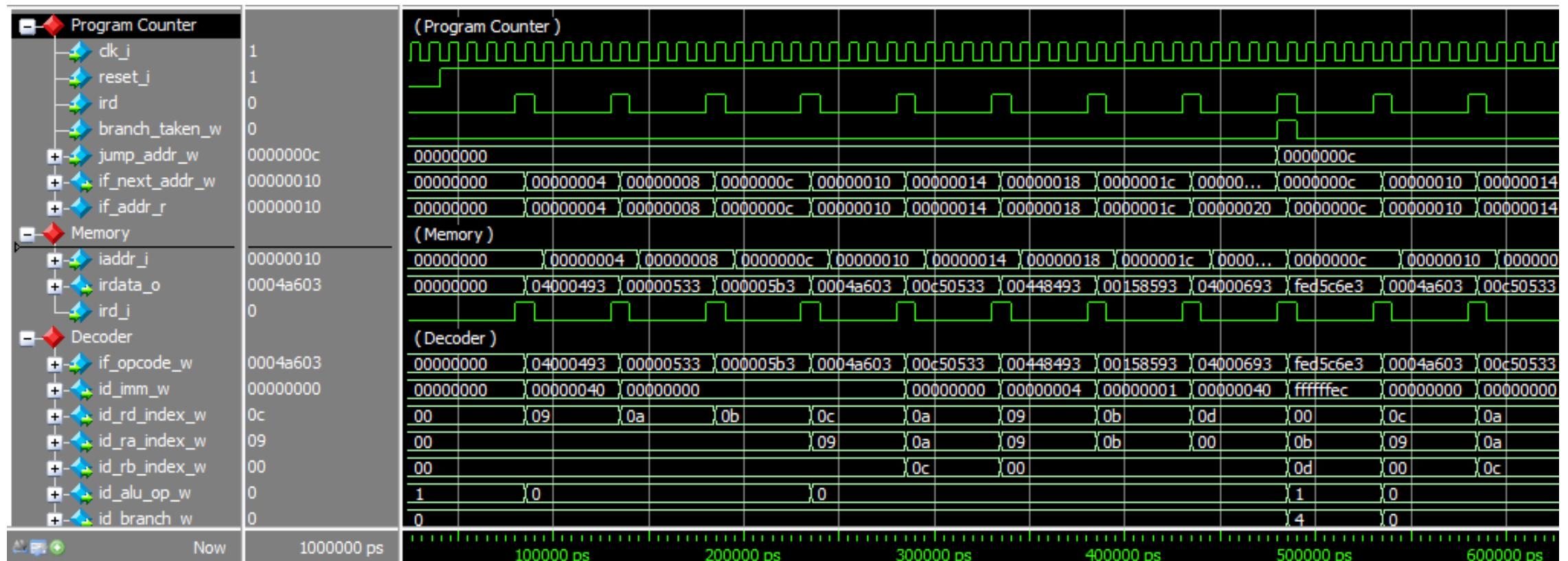
TODO ...

- Complete the missing code for test bench
(and check the test module name "riscv_core_sim_tb2")

```
//-----  
// Design  
//-----  
riscv_pc u_pc(  
    ./*input */    clk_i(clk_i),           // Clock  
    ./*input */    reset_i(reset_i),        // Reset  
    //Your code  
    //{{{  
    ./*input */          ird( ),           // Instruction Read request  
    ./*input */          branch_taken_w( ), // Jump instruction  
    ./*input [PC_SIZE-1:0] */ jump_addr_w( ), // Jump address  
    ./*output [PC_SIZE-1:0] */ if_next_addr_w( ) // Next instruction  
    //}}}  
);  
// Memory  
riscv_memory  
u_memory (  
    .clk_i(clk_i),  
    .reset_i(reset_i),  
    // Your code  
    //{{{  
    .iaddr_i( ),  
    .ird_i( ),  
    //}}}  
    .daddr_i(daddr_i),  
    .dwdata_i(dwdata_i),  
    .dsize_i(dszie_i),  
    .drd_i(drd_i),  
    .dwr_i(dwr_i),  
    .irdata_o(irdata_o),  
    .drdata_o(drdata_o)  
);
```

Waveform

- Do a simulation with time = 1000 ns



Test cases

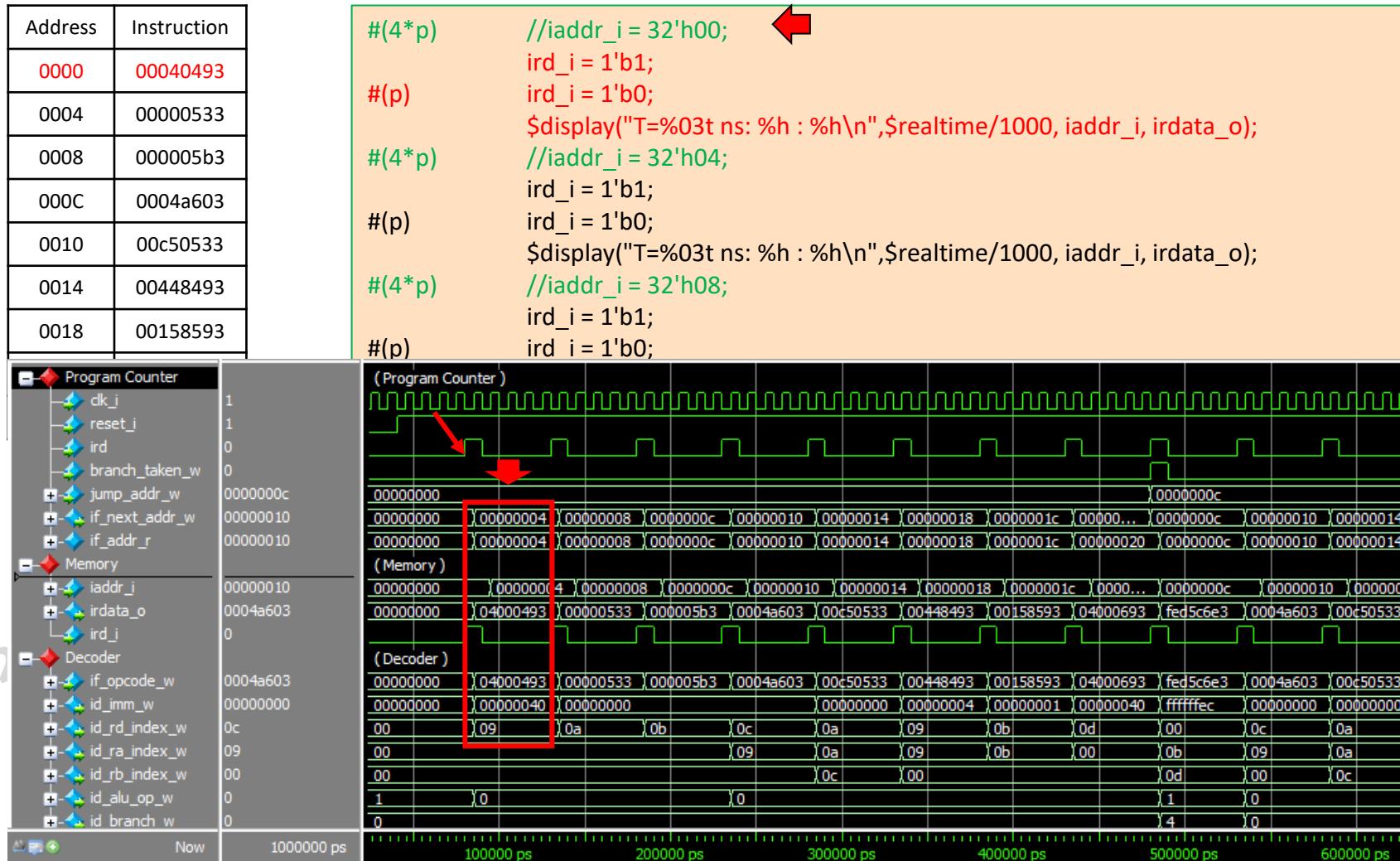
- Loop initialization
 - Three instructions

Address	Instruction
0000	00040493
0004	00000533
0008	000005b3
000C	0004a603
0010	00c50533
0014	00448493
0018	00158593
001C	04000693
0020	fed5c6e3

```
#(4*p)    //iaddr_i = 32'h00;      ←
          ird_i = 1'b1;
          ird_i = 1'b0;
          $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr_i, irdata_o);
#(p)
#(4*p)    //iaddr_i = 32'h04;      ←
          ird_i = 1'b1;
          ird_i = 1'b0;
          $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr_i, irdata_o);
#(p)
#(4*p)    //iaddr_i = 32'h08;      ←
          ird_i = 1'b1;
          ird_i = 1'b0;
          $display("T=%03t ns: %h : %h\n",$realtime/1000, iaddr_i, irdata_o);
```

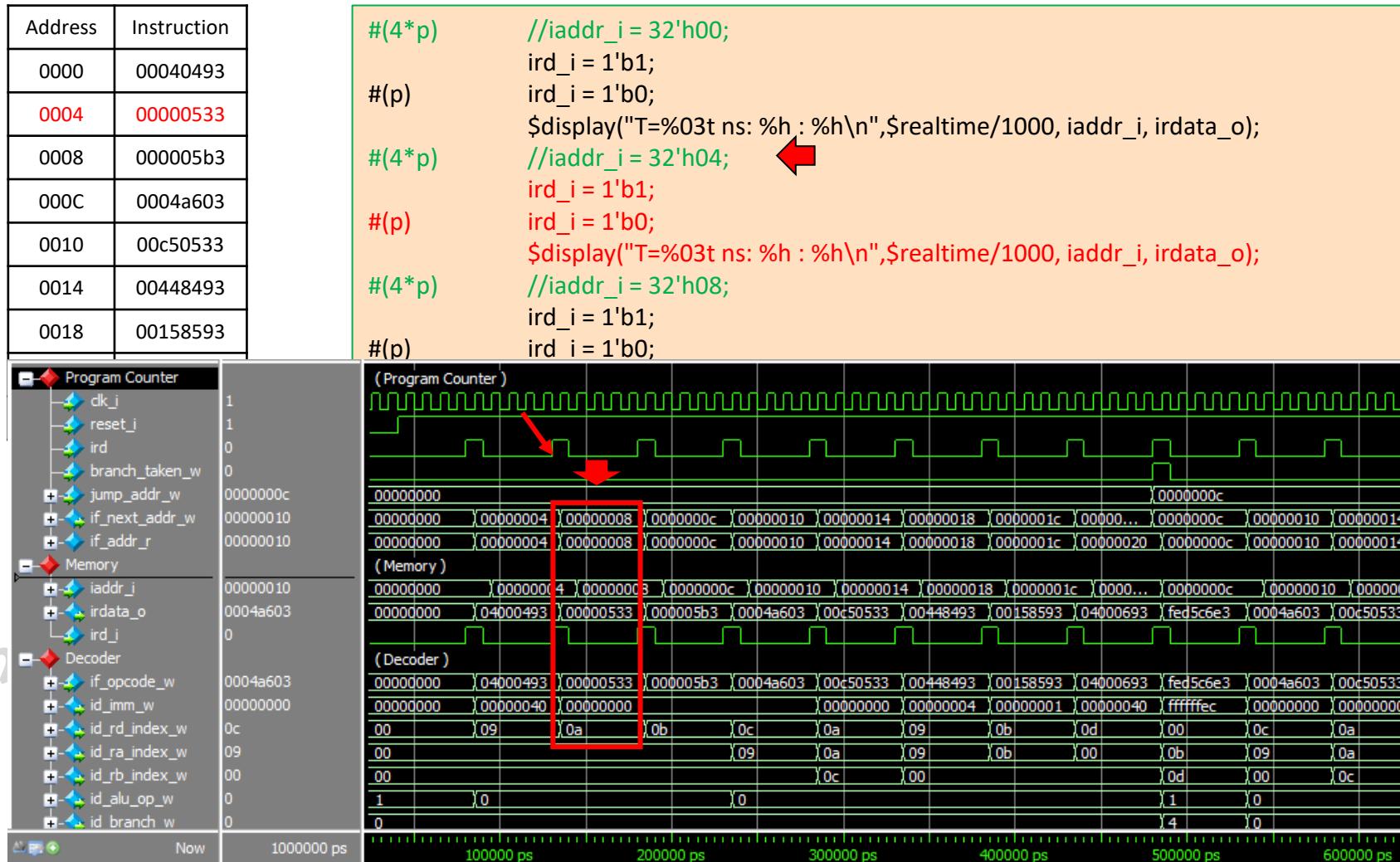
Waveform

■ Initialization



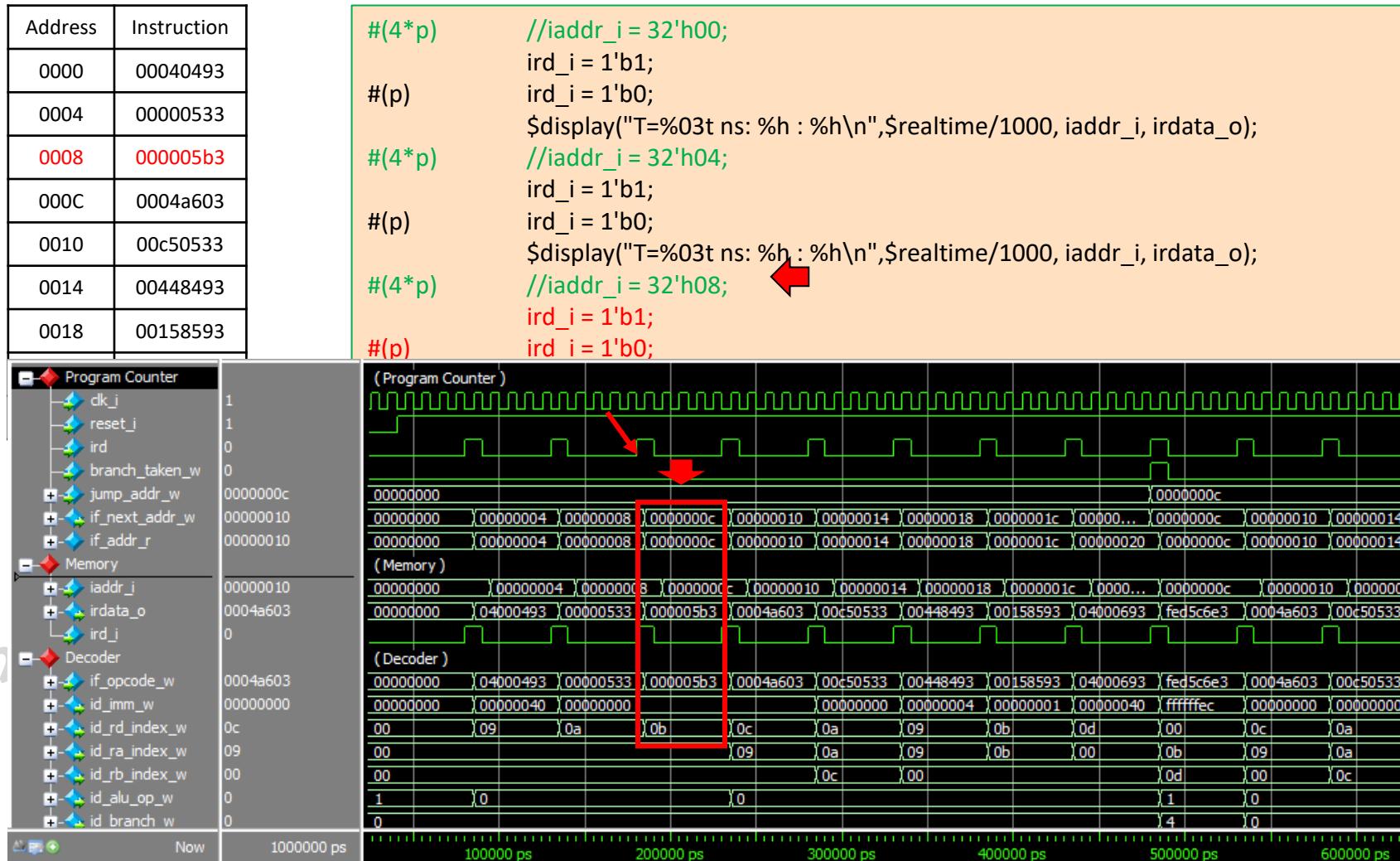
Waveform

■ Initialization



Waveform

■ Initialization



Test cases

- Loop

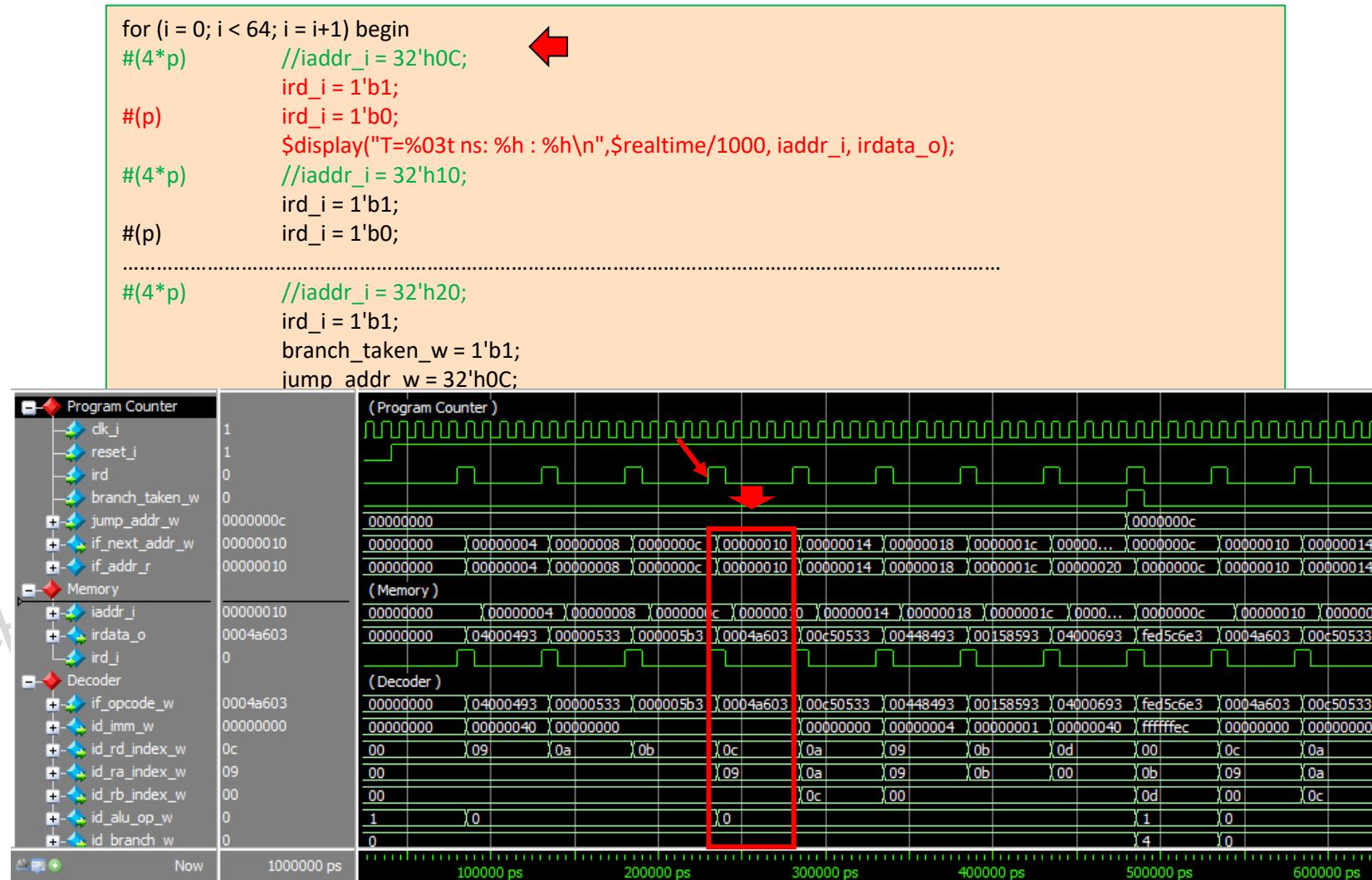
Address	Instruction
0000	00040493
0004	00000533
0008	000005b3
000C	0004a603
0010	00c50533
0014	00448493
0018	00158593
001C	04000693
0020	fed5c6e3

```
for (i = 0; i < 64; i = i+1) begin
  #(4*p)      //iaddr_i = 32'h0C;           ←
    ird_i = 1'b1;
    ird_i = 1'b0;
    $display("T=%03t ns: %h : %h\n", $realtime/1000, iaddr_i, irdata_o);
  #(4*p)      //iaddr_i = 32'h10;           ←
    ird_i = 1'b1;
    ird_i = 1'b0;
  .....
  #(4*p)      //iaddr_i = 32'h20;           ←
    ird_i = 1'b1;
    branch_taken_w = 1'b1;
    jump_addr_w = 32'h0C;
    ird_i = 1'b0;
    branch_taken_w = 1'b0;
    $display("T=%03t ns: %h : %h\n", $realtime/1000, iaddr_i, irdata_o
end
```

Test cases

- Loop

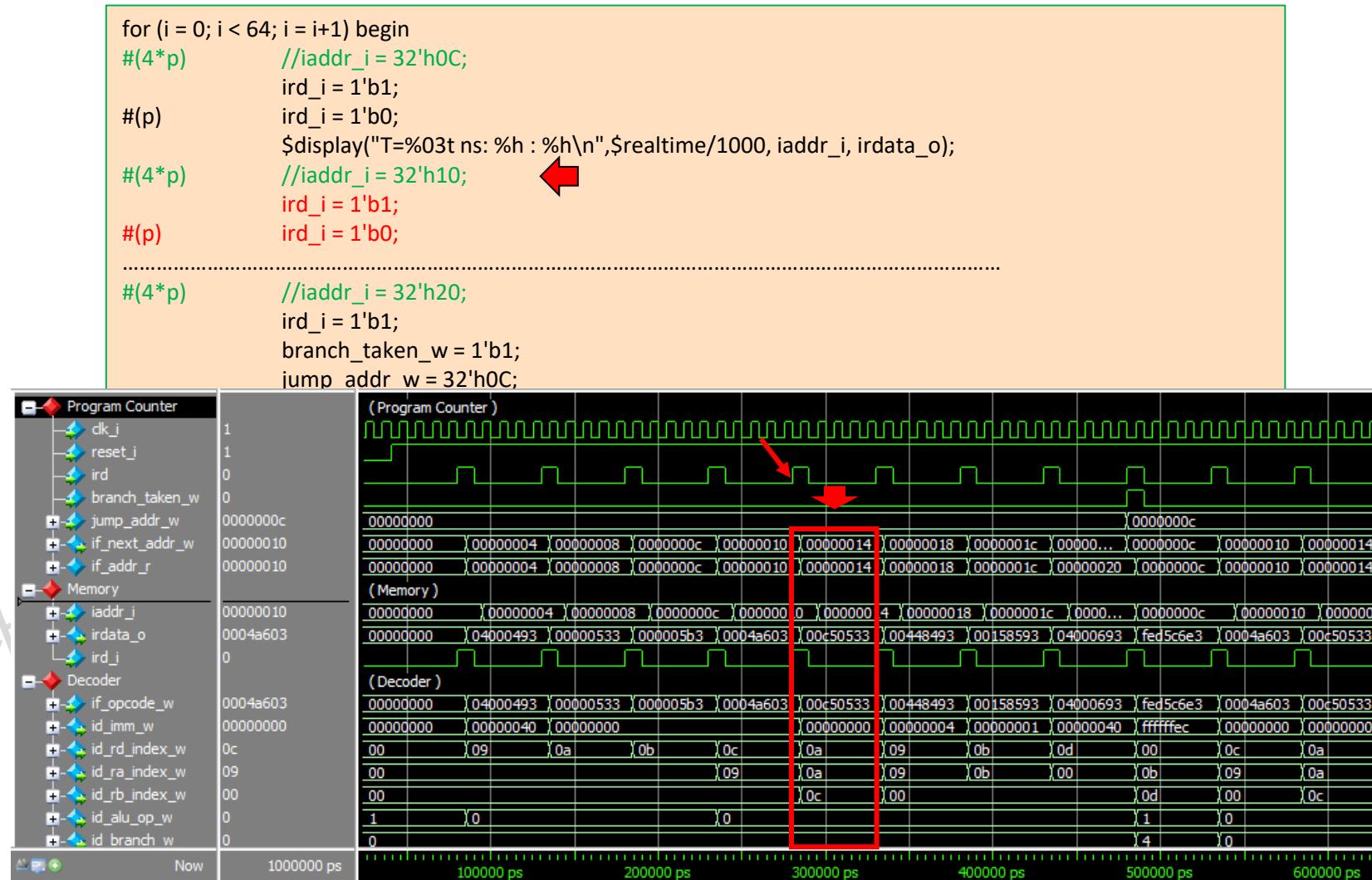
Address	Instruction
0000	00040493
0004	00000533
0008	000005b3
000C	0004a603
0010	00c50533
0014	00448493
0018	00158593
001C	04000693
0020	fed5c6e3



Test cases

- Loop

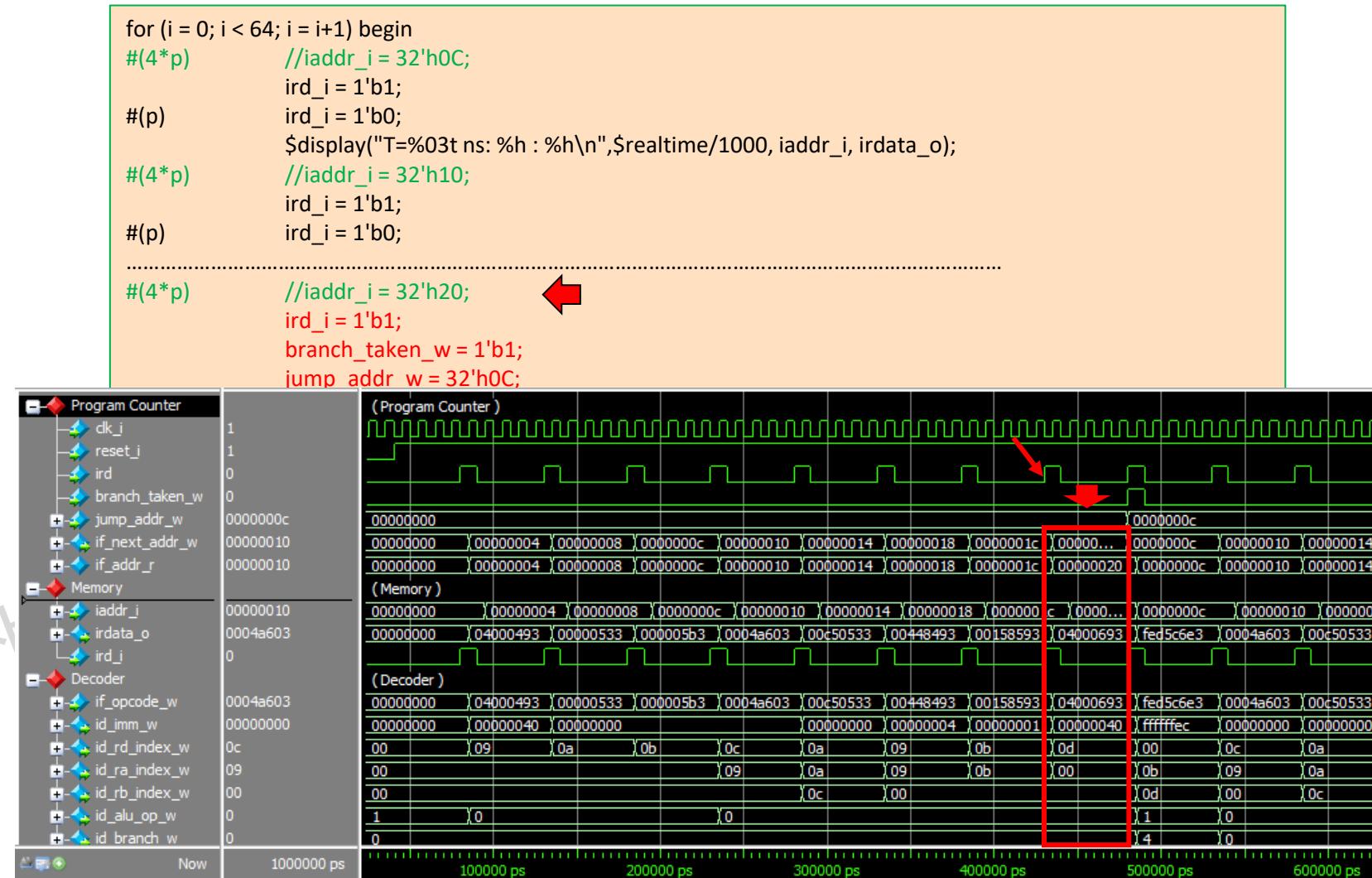
Address	Instruction
0000	00040493
0004	00000533
0008	000005b3
000C	0004a603
0010	00c50533
0014	00448493
0018	00158593
001C	04000693
0020	fed5c6e3



Test cases

- Loop

Address	Instruction
0000	00040493
0004	00000533
0008	000005b3
000C	0004a603
0010	00c50533
0014	00448493
0018	00158593
001C	04000693
0020	fed5c6e3



TODO ...

- Reuse riscv_pc.v from Lab 1
- Implement riscv_pc_tb2.v by completing the missing codes
- Do a simulation with time = 1000 ns
- Show the waveform

Copyright 2022. (차세대반도체 혁신공유대학 사업단) all rights reserved.

Road map

RISC-V
RISC-V Simulator

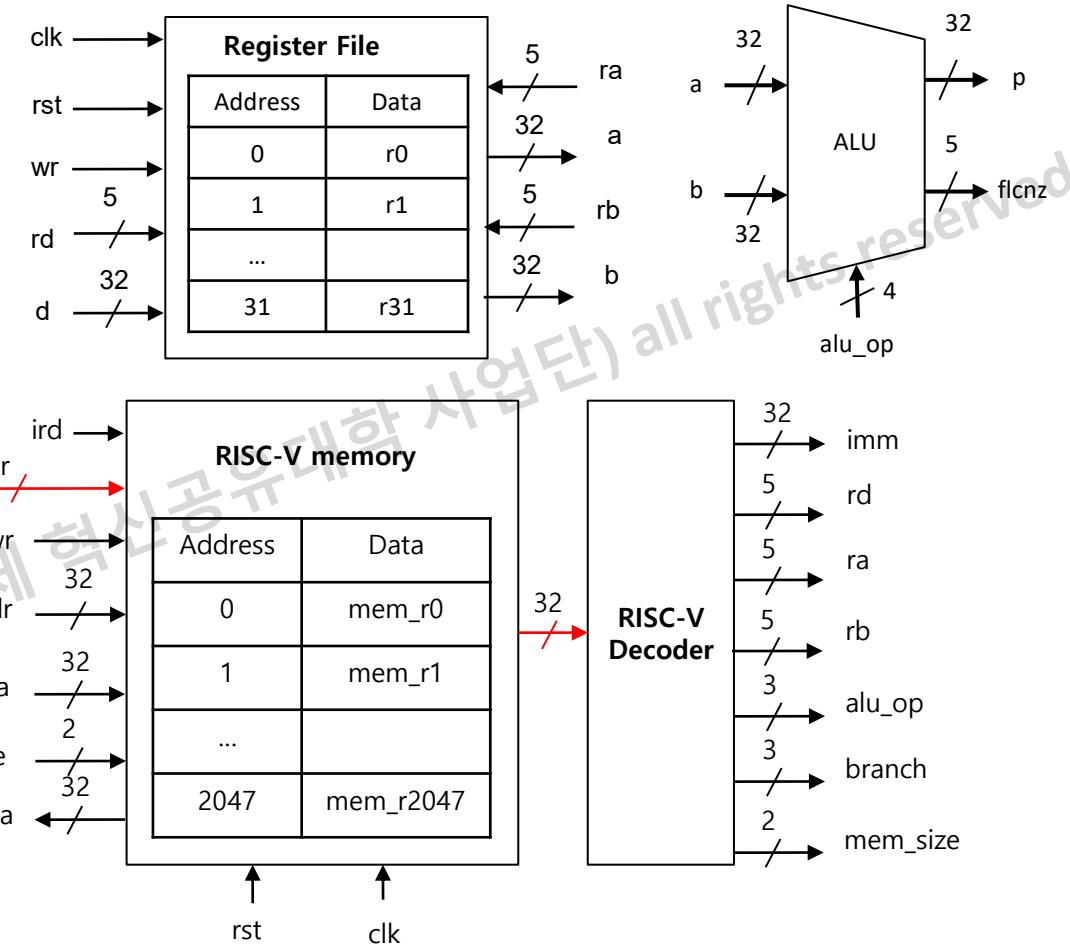
Lab 1: Program Counter

Lab 2: Program Counter
(Extended)

Lab 3: RISC-V core sim

Simplified RISC-V

- Five modules
 - Program counter
 - Memory
 - Decoder
 - ALU
 - Register File



Lab 3: Simplified RISC-V core

- Lab 3: RISC-V core
 - Implement a simplified RISC-V core
 - Run a simulation with time = 1000 ns
 - Show the output result

Simplified RISC-V

- RISC-V Core is connected to Memory
 - Instruction ports
 - Data port
- ⇒ There is no signal from ALU, register file, decoder, program counter.

```
module riscv_core_sim #(
    parameter PC_SIZE = 32,
    parameter RESET_SP = 32'h0000
) (
    input      clk_i,      // Clock
    input      reset_i,    // Reset
    output     lock_o,     // Lock

    output [31:0] iaddr_o, // Instruction from Memory
    input  [31:0] irdata_i, // Instruction address
    output     ird_o,      // Read request

    output [31:0] daddr_o, // Read/Write address
    output [31:0] dwdata_o, // Write Data
    input  [31:0] drdata_i, // Read data
    output [1:0]  dsizes_o,
    output      drd_o,     // Write request
    output      dwr_o      // Read/Write Enable
);
```

Instruction
Program

Data

	mem.hex	riscv
1	04000493	
2	00000533	
3	000005b3	
4	0004a603	
5	00c50533	
6	00448493	
7	00158593	
8	04000693	
9	fed5c6e3	
10		
11		
12		
13		
14		
15		
16	00000000	
17	00000001	
18	00000002	
19	00000003	
20	00000004	
21	00000005	
22	00000006	
23	00000007	
24	00000008	

Instruction Fetch

- Program counter
 - Store the next instruction
 - If (branch_taken_w)
 - PC \leftarrow jump_addr
 - Else
 - PC \leftarrow curr_addr + 4

```
//-----
// TODO: Instruction Fetch
//-----
always @ (posedge clk_i or negedge reset_i) begin
    if (~reset_i) begin
        if_pc_w <= RESET_SP;
    end
    else begin
        if_pc_w <= if_next_addr_w;
    end
end
assign iaddr_o = if_pc_w;
assign ird_o  = 1'bl;
assign lock_o = 1'b0;
assign if_opcode_w = irdata_i;

//-----
// Program Counter
//-----
riscv_pc #(.RESET_SP(RESET_SP))
u_pc(
    /*input          */clk_i(clk_i),
    /*input          */reset_i(reset_i),
    /*input          */ird(ird_o),
    /*input          */branch_taken_w(branch_taken_w),
    /*input [31:0]   */jump_addr_w(jump_addr_w),
    /*output [31:0] */if_next_addr_w(if_next_addr_w)
);
//-----
```

Instruction Fetch

- Program counter
 - Store the next instruction
- For an instruction address,
the core fetches an instruction from memory.

```
//-----
// TODO: Instruction Fetch
//-----

always @ (posedge clk_i or negedge reset_i) begin
    if (~reset_i) begin
        if_pc_w <= RESET_SP;
    end
    else begin
        if_pc_w <= if_next_addr_w;
    end
end
assign iaddr_o = if_pc_w;
assign ird_o  = 1'b1;
assign lock_o = 1'b0;
assign if_opcode_w = irddata_i;

//-----
// Program Counter
//-----

riscv_pc #(.RESET_SP(RESET_SP))
u_pc(
    /*input      */clk_i(clk_i),
    /*input      */reset_i(reset_i),
    /*input      */ird(ird_o),
    /*input      */branch_taken_w(branch_taken_w),
    /*input [31:0] */jump_addr_w(jump_addr_w),
    /*output [31:0]*/if_next_addr_w(if_next_addr_w)
);
//-----
```

Decode

- For an instruction, Decoder extracts all information including:
 - Addresses of source and dest. registers for Register File (ra, rb, rd).
 - ALU opcode (alu_op)
 - Immediate (imm)
 - Branch, jump & link types (branch)

```
//-----  
// TODO: Instruction Fetch  
//-----  
always @ (posedge clk_i or negedge reset_i) begin  
    if (~reset_i) begin  
        if_pc_w <= RESET_SP;  
    end  
    else begin  
        if_pc_w <= if_next_addr_w;  
    end  
end  
assign iaddr_o = if_pc_w;  
assign ird_o = 1'b1;  
assign lock_o = 1'b0;  
assign if_opcode_w = iridata_i;
```

```
//-----  
// Decoder  
//-----  
riscv_decoder  
u_decoder  
(  
    /*input [31:0]*/if_opcode_w (if_opcode_w ),  
    /*output [31:0]*/id_imm_w (id_imm_w ),  
    /*output [4:0] */id_rd_index_w (id_rd_index_w ),  
    /*output [4:0] */id_ra_index_w (id_ra_index_w ),  
    /*output [4:0] */id_rb_index_w (id_rb_index_w ),  
    /*output [3:0] */id_alu_op_w (id_alu_op_w ),  
    /*output [2:0] */id_branch_w (id_branch_w ),  
    /*output [1:0] */id_mem_size_w (id_mem_size_w ),  
    /*output */mulh_w (mulh_w ),  
    /*output */mulhsu_w (mulhsu_w ),  
    /*output */div_w (div_w ),  
    /*output */rem_w (rem_w ),  
    /*output */sra_w (sra_w ),  
    /*output */srai_w (srai_w ),  
    /*output */alu_imm_w (alu_imm_w ),  
    /*output */jal_w (jal_w ),  
    /*output */load_w (load_w ),  
    /*output */store_w (store_w ),  
    /*output */lbu_w (lbu_w ),  
    /*output */lhu_w (lhu_w ),  
    /*output */jalr_w (jalr_w ),  
    /*output */id_illegal_w (id_illegal_w )  
);
```

Register file

- Given addresses of source and dest. registers (ra, rb, rd),

Register File can

- Output a value of a register.
- Store or update a new value of a register

```
//-----  
// Register File  
//-----  
riscv_regfile  
u_regfile  
(  
    ./*input          */clk_i(clk_i)  
    ./*input          */rstn_i(reset_i)  
    ./*input [ 4:0]   */rd0_i(rd_index_w)  
    ./*input [ 31:0]  */rd0_value_i(rd_value_w)  
    ./*input [ 4:0]   */ra0_i(id_ra_index_w)  
    ./*input [ 4:0]   */rb0_i(id_rb_index_w)  
    ./*input          */wr(rd_we_w)  
    ./*output [ 31:0] */ra0_value_o(ra_value_r)  
    ./*output [ 31:0] */rb0_value_o(rb_value_r)  
);  
  
// Dummy register file ports  
always@(*) begin  
    rd_index_w = 5'h0;  
    rd_value_w = 32'h0;  
    rd_we_w   = 1'b0;  
    // Insert your code  
    //{{{  
    //rd_index_w = id_rd_index_w;  
    //rd_value_w = alu_p;  
    //rd_we_w   = 1'b1;  
    //}}}  
end
```

ALU

- ALU calculates the results
 - Inputs:
 - Registers from Register File
 - Operation from Decoder
 - Output
 - ALU output (alu_p).
 - Flags (flcnz).

```
//-----  
// ALU  
//-----  
riscv_alu  
u_alu  
(  
    /*input [ 3:0] */alu_op_i(alu_op)  
    /*input [ 31:0] */alu_a_i(alu_a)  
    /*input [ 31:0] */alu_b_i(alu_b)  
    /*output [ 31:0] */alu_p_o(alu_p)  
    /*output reg [4:0] */flcnz(flcnz)  
) ;
```

Test bench

- Test bench includes
 - A memory
 - A RISC-V core

```
module riscv_memory_tb;
    reg reset_i;
    reg clk_i;

    // Input instruction
    reg [31:0] iaddr_i;
    reg        ird_i;
    reg [31:0] daddr_i;
    reg [31:0] dwdata_i;
    reg [1:0]  dszie_i;
    reg        drd_i;
    reg        dwr_i;
    // Outputs
    wire [31:0] irdata_o;
    wire [31:0] drdata_o;
    ...
```

```
riscv_memory #(.FIRMWARE("mem.hex"))
u_riscv_memory
(
    /*input      */clk_i(clk_i),
    /*input      */reset_i(reset_i),
    /*input [31:0] */iaddr_i(iaddr),
    /*output [31:0] */irdata_o(irdata),
    /*input      */ird_i(ird),
    /*input [31:0] */daddr_i(daddr),
    /*input [31:0] */dwdata_i(dwdata),
    /*output [31:0] */drdata_o(drdata),
    /*input [1:0]  */dszie_i(dszie),
    /*input      */drd_i(drd),
    /*input      */dwr_i(dwr)
);

riscv_core_sim
u_riscv_core_sim
(
    /*input      */clk_i(clk_i),
    /*input      */reset_i(reset_i),
    /*output      */lock_o(lock),
    /*output [31:0] */iaddr_o(iaddr),
    /*input [31:0] */irdata_i(irdata),
    /*output      */ird_o(ird),
    /*output [31:0] */daddr_o(daddr),
    /*output [31:0] */dwdata_o(dwdata),
    /*input [31:0] */drdata_i(drdata),
    /*output [1:0]  */dszie_o(dszie),
    /*output      */drd_o(drd),
    /*output      */dwr_o(dwr)
);
```

Test bench

- Test bench includes
 - A memory
 - A RISC-V core
- For test cases, only clock and reset signals are required

```
// CLOCK
initial begin
clk_i = 0;
forever #5 clk_i = ~clk_i;
end

// Testcase
initial
begin
    reset_i = 1'b0;

    #20 reset_i = 1'b1;
    // Reset

end
```

```
riscv_memory #(FIRMWARE("mem.hex"))
u_riscv_memory
|(
    /*input      */clk_i(clk_i),
    /*input      */reset_i(reset_i),
    /*input [31:0] */iaddr_i(iaddr),
    /*output [31:0] */irdata_o(irdata),
    /*input      */ird_i(ird),
    /*input [31:0] */daddr_i(daddr),
    /*input [31:0] */dwdata_i(dwdata),
    /*output [31:0] */drdata_o(drdata),
    /*input [1:0]  */dszie_i(dszie),
    /*input      */drd_i(drd),
    /*input      */dwr_i(dwr)
);

riscv_core_sim
u_riscv_core_sim
|(
    /*input      */clk_i(clk_i),
    /*input      */reset_i(reset_i),
    /*output      */lock_o(lock),
    /*output [31:0] */iaddr_o(iaddr),
    /*input [31:0] */irdata_i(irdata),
    /*output      */ird_o(ird),
    /*output [31:0] */daddr_o(daddr),
    /*output [31:0] */dwdata_o(dwdata),
    /*input [31:0] */drdata_i(drdata),
    /*output [1:0]  */dszie_o(dszie),
    /*output      */drd_o(drd),
    /*output      */dwr_o(dwr)
);
```

Dummy vs TODO

- Dummy code/connections
 - Maintain the dataflow among modules, i.e. PC, Decoder, ALU, RF.
 - Some logics are incomplete.
- TODO
 - Some logics are missing, so that they need to be filled out.

Dummy

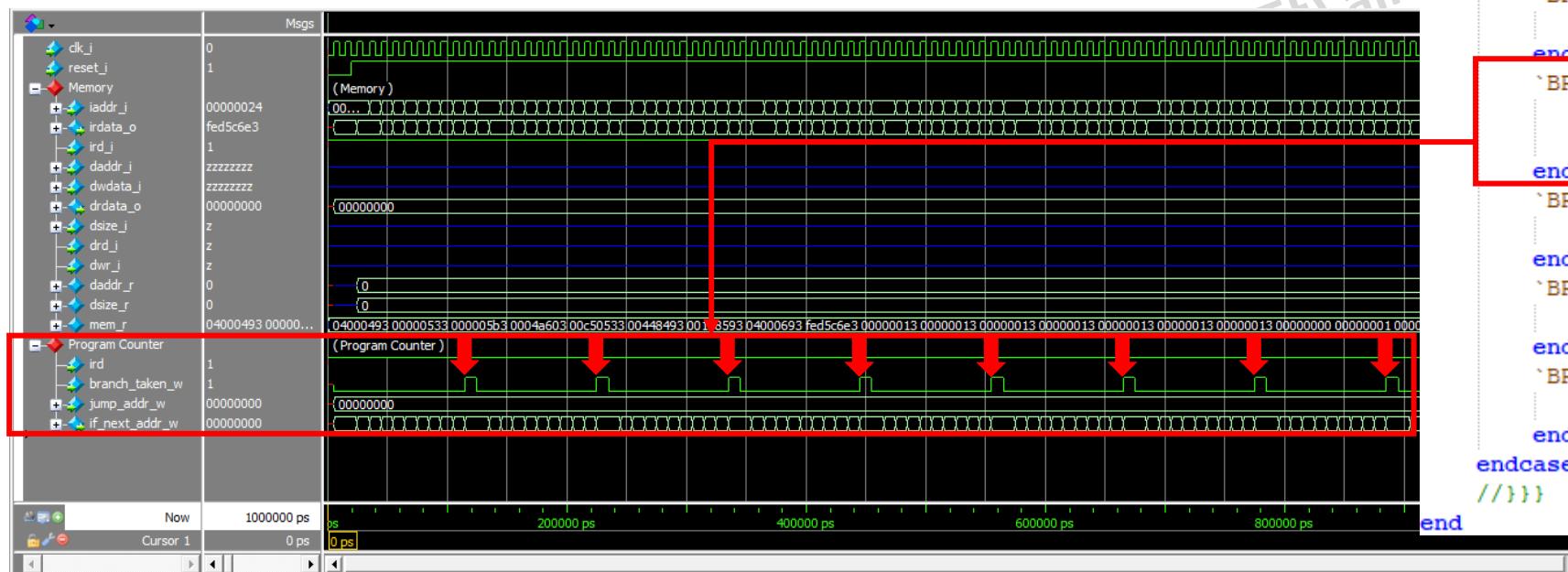
- Dummy code for "branch if less than (BR_LT)"
 - branch_taken_w = 1'b1
 - ⇒ Set the flag to HIGH for "branch if less than (BR_LT)"
 - ⇒ **Dummy: Branch-Less-Than is an unconditional instruction**

Address	Instruction	Basic code	Original code	Comments
0000	00004093	addi x9 x0 64	addi x9, x0, 64	# x9=&A[0]
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# x12=A[i]
0010	00c50533	add x10 x10 x12	add x10,x10,x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9,x9,4	# &A[i++]
0018	00158593	addi x11 x11 1	addi x11,x11,1	# i++
001C	04000693	addi x13 x0 64	addi x13,x0,64	# x13=64
0020	fed5c6e3	blt x11 x13 -20	blt x11,x13,Loop	# Branch

```
/* TODO: Branch, Jump and Link instructions */
always @ (*) begin
    branch_taken_w = 1'b0;
    jump_addr_w = 32'h0;
    // Insert your code
    //{{{
    case(id_branch_w)
        `BR_JUMP: begin
            end
        `BR_EQ: begin
            end
        `BR_NE: begin
            end
        `BR_LT: begin
            // Dummy Branch
            branch_taken_w = 1'b1;
        end
        `BR_GE: begin
            end
        `BR_LTU: begin
            end
        `BR_GEU: begin
            end
    endcase
    //}}}
end
```

Baseline code: Dummy

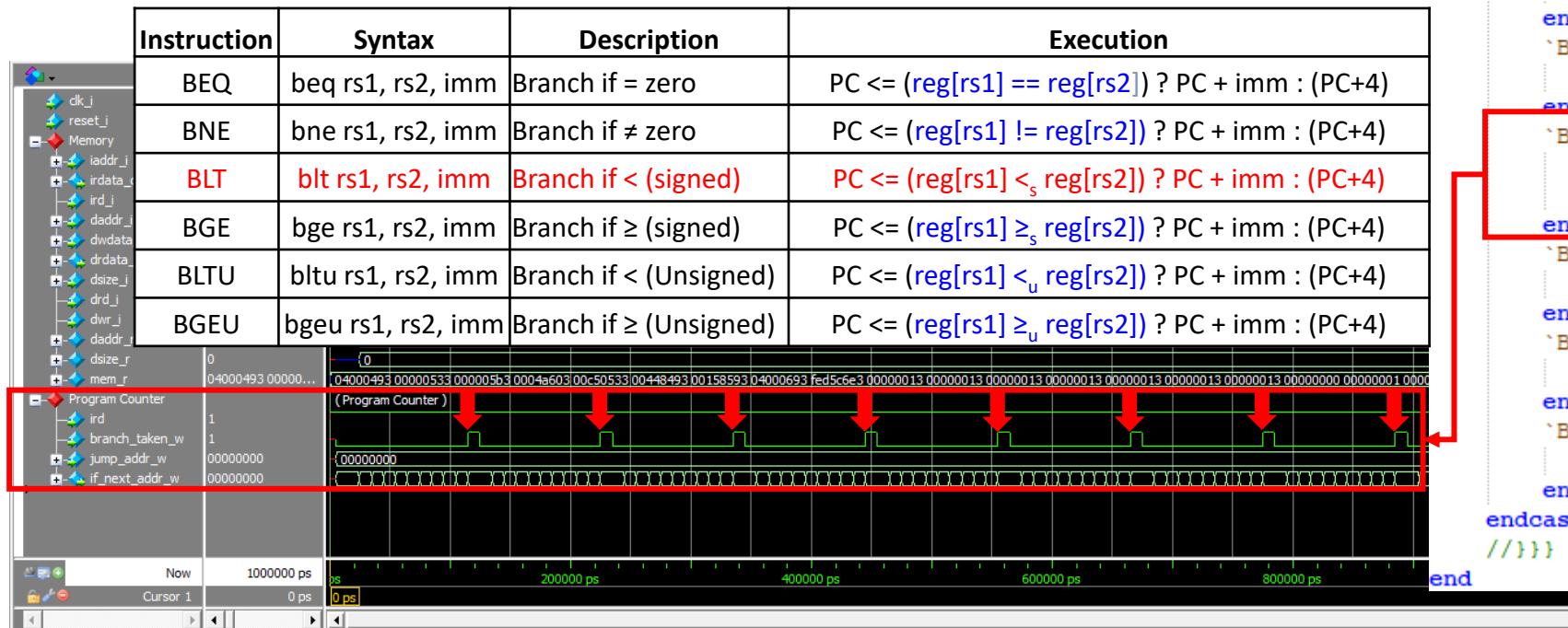
- Dummy code/connections
 - Maintain the dataflow among modules
 - Some logics are incomplete.



```
/* TODO: Branch, Jump and Link instructions */
always @ (*) begin
    branch_taken_w = 1'b0;
    jump_addr_w = 32'h0;
    // Insert your code
    //////
    case(id_branch_w)
        `BR_JUMP: begin
        end
        `BR_EQ: begin
        end
        `BR_NE: begin
        end
        `BR_LT: begin
            // Dummy Branch
            branch_taken_w = 1'bl;
        end
        `BR_GE: begin
        end
        `BR_LTU: begin
        end
        `BR_GEU: begin
        end
    endcase
    //}}}
end
```

How about TODO?

- TODO
 - Some logics are missing, so that they need to be filled out.
 - Example: Complete "branch if less than (BR_LT)"
- Calculate a target address and its enable for branch instructions



```
/* TODO: Branch, Jump and Link instructions */
always @ (*) begin
    branch_taken_w = 1'b0;
    jump_addr_w = 32'h0;
    // Insert your code
    //////
    case(id_branch_w)
        `BR_JUMP: begin
        end
        `BR_EQ: begin
        end
        `BR_NE: begin
        end
        `BR_LT: begin
            // Dummy Branch
            branch_taken_w = 1'b1;
        end
        `BR_GE: begin
        end
        `BR_LTU: begin
        end
        `BR_GEU: begin
        end
    endcase
    //////
end
```

TODO ...

- Add register file and ALU from Lecture 2, decoder and memory from Lecture 3
- Add your Program Counter (riscv_pc.v) in the previous exercise

```
always @(posedge clk_i or negedge reset_i) begin
    if (~reset_i) begin
        //Your code
    end
    else begin
        if (ird) begin
            // Your code
            //{{{
            //}}}
        end
    end
end
```

- Uncomment the following codes in riscv_core_sim:
 - Connect Register File and Decoder to ALU

```
/* TODO: ALU */
always@(*) begin
    alu_op = `ALU_ADD;
    alu_a = 32'h0;
    alu_b = 32'h0;

    // Insert your code
    //{
    //alu_op = id_alu_op_w;
    //alu_a = ra_value_r;
    //alu_b = rb_value_r;
    //}
end
```

```
// Dummy register file ports
always@(*) begin
    rd_index_w = 5'h0;
    rd_value_w = 32'h0;
    rd_we_w   = 1'b0;
    // Insert your code
    //{{{
    //rd_index_w = id_rd_index_w;
    //rd_value_w = alu_p;
    //rd_we_w   = 1'bl;
    //}}}
end
```

TODO : Calculate the address

- Conditional Branches:
 - Branch if = zero (BEQ)
 - Branch if \neq zero (BNE)
 - Branch if $<$ (signed) (BLT)
 - Branch if \geq (signed) (BGE)
 - Branch if $<$ (unsigned) (BLTU)
 - Branch if \geq (unsigned) (BGEU)

B-type	imm[12:10:5]	rs2	rs1	funct3	imm[4:1 11]	opcode
--------	--------------	-----	-----	--------	-------------	--------

Instruction	Syntax	Description	Execution
BEQ	beq rs1, rs2, imm	Branch if = zero	$PC \leq (\text{reg}[rs1] == \text{reg}[rs2]) ? PC + \text{imm} : (PC+4)$
BNE	bne rs1, rs2, imm	Branch if \neq zero	$PC \leq (\text{reg}[rs1] != \text{reg}[rs2]) ? PC + \text{imm} : (PC+4)$
BLT	blt rs1, rs2, imm	Branch if $<$ (signed)	$PC \leq (\text{reg}[rs1] <_s \text{reg}[rs2]) ? PC + \text{imm} : (PC+4)$
BGE	bge rs1, rs2, imm	Branch if \geq (signed)	$PC \leq (\text{reg}[rs1] \geq_s \text{reg}[rs2]) ? PC + \text{imm} : (PC+4)$
BLTU	bltu rs1, rs2, imm	Branch if $<$ (Unsigned)	$PC \leq (\text{reg}[rs1] <_u \text{reg}[rs2]) ? PC + \text{imm} : (PC+4)$
BGEU	bgeu rs1, rs2, imm	Branch if \geq (Unsigned)	$PC \leq (\text{reg}[rs1] \geq_u \text{reg}[rs2]) ? PC + \text{imm} : (PC+4)$

Example

- Program counter
 - Initialization: Store the base address of the Main function

PC →

Address	Instruction	Basic code	Original code	Comments
0000	00400493	addi x9 x0 64	addi x9, x0, 64	# x9=&A[0]
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# x12=A[i]
0010	00c50533	add x10 x10 x12	add x10,x10,x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9,x9,4	# &A[i++]
0018	00158593	addi x11 x11 1	addi x11,x11,1	# i++
001C	04000693	addi x13 x0 64	addi x13,x0,64	# x13=64
0020	fed5c6e3	blt x11 x13 -20	blt x11,x13,Loop	# Branch

Example

- Program counter
 - Normal instruction: $PC \leftarrow PC + 4$

PC →

Address	Instruction	Basic code	Original code	Comments
0000	00400493	addi x9 x0 64	addi x9, x0, 64	# x9=&A[0]
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# x12=A[i]
0010	00c50533	add x10 x10 x12	add x10,x10,x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9,x9,4	# &A[i++]
0018	00158593	addi x11 x11 1	addi x11,x11,1	# i++
001C	04000693	addi x13 x0 64	addi x13,x0,64	# x13=64
0020	fed5c6e3	blt x11 x13 -20	blt x11,x13,Loop	# Branch

Example

- Program counter
 - Normal instruction: $PC \leftarrow PC + 4$

PC →

Address	Instruction	Basic code	Original code	Comments
0000	00400493	addi x9 x0 64	addi x9, x0, 64	# x9=&A[0]
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# x12=A[i]
0010	00c50533	add x10 x10 x12	add x10,x10,x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9,x9,4	# &A[i++]
0018	00158593	addi x11 x11 1	addi x11,x11,1	# i++
001C	04000693	addi x13 x0 64	addi x13,x0,64	# x13=64
0020	fed5c6e3	blt x11 x13 -20	blt x11,x13,Loop	# Branch

Example

- Program counter
 - Normal instruction: $PC \leftarrow PC + 4$

Address	Instruction	Basic code	Original code	Comments
0000	00400493	addi x9 x0 64	addi x9, x0, 64	# x9=&A[0]
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# x12=A[i]
0010	00c50533	add x10 x10 x12	add x10,x10,x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9,x9,4	# &A[i++]
0018	00158593	addi x11 x11 1	addi x11,x11,1	# i++
001C	04000693	addi x13 x0 64	addi x13,x0,64	# x13=64
0020	fed5c6e3	blt x11 x13 -20	blt x11,x13,Loop	# Branch

PC →

Example

- Program counter
 - Branch instruction: $PC \leftarrow PC + \text{offset} (-20)$
`blt x11 x13 -20`

⇒ "if x_{11} is smaller than x_{13} (BLT), move to the next instruction at $PC - 20$ "

PC →

Address	Instruction	Basic code	Original code	Comments
0000	00400493	addi x9 x0 64	addi x9, x0, 64	# $x_9 = &A[0]$
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# $x_{12} = A[i]$
0010	00c50533	add x10 x10 x12	add x10, x10, x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9, x9, 4	# $&A[i++]$
0018	00158593	addi x11 x11 1	addi x11, x11, 1	# i++
001C	04000693	addi x13 x0 64	addi x13, x0, 64	# $x_{13} = 64$
0020	fed5c6e3	blt x11 x13 -20	blt x11, x13, Loop	# Branch

Test case

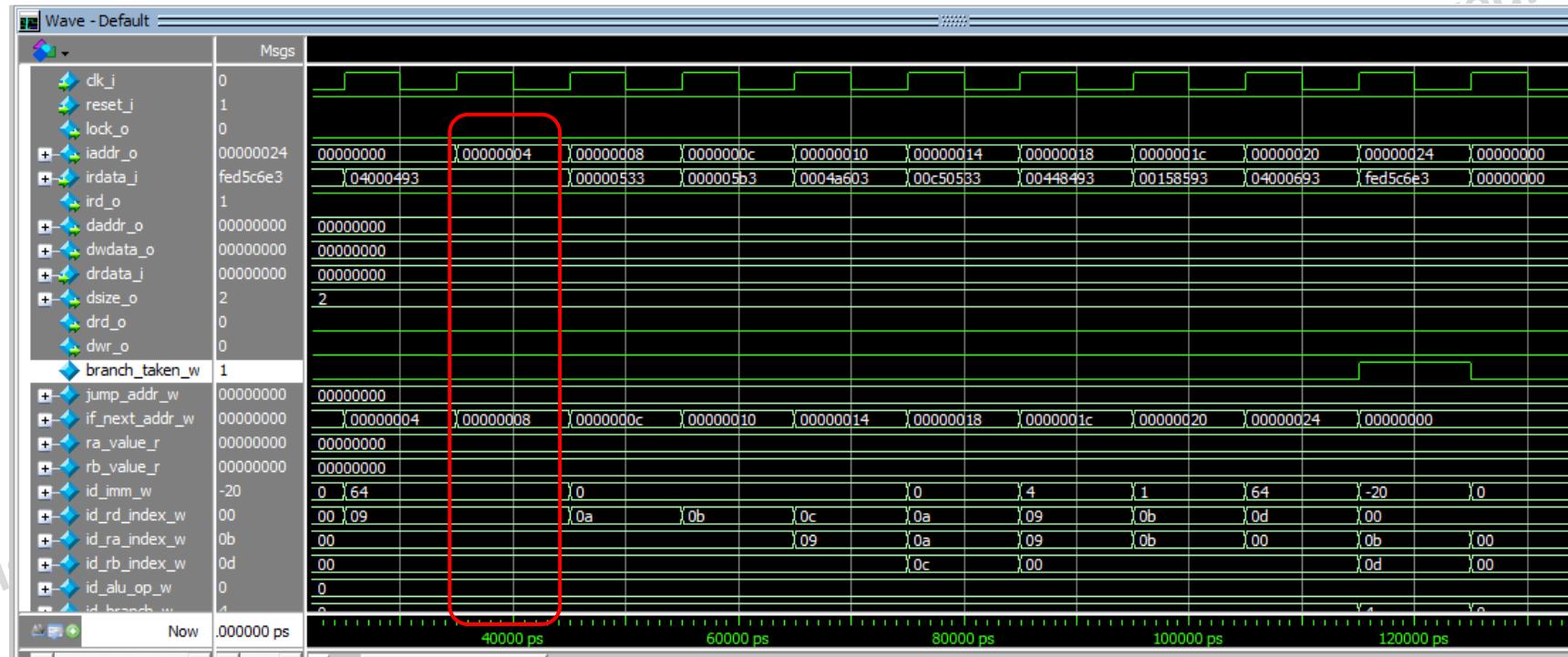
- Program counter
 - Initialization: Store the base address of the Main function

PC →

Address	Instruction	Basic code	Original code	Comments
0000	00400493	addi x9 x0 64	addi x9, x0, 64	# x9=&A[0]
0004	00000533	add x10 x0 x0	add x10, x0, x0	# sum=0
0008	000005b3	add x11 x0 x0	add x11, x0, x0	# i=0
000C	0004a603	lw x12 0(x9)	lw x12, 0(x9)	# x12=A[i]
0010	00c50533	add x10 x10 x12	add x10,x10,x12	# sum+=
0014	00448493	addi x9 x9 4	addi x9,x9,4	# &A[i++]
0018	00158593	addi x11 x11 1	addi x11,x11,1	# i++
001C	04000693	addi x13 x0 64	addi x13,x0,64	# x13=64
0020	fed5c6e3	blt x11 x13 -20	blt x11,x13,Loop	# Branch

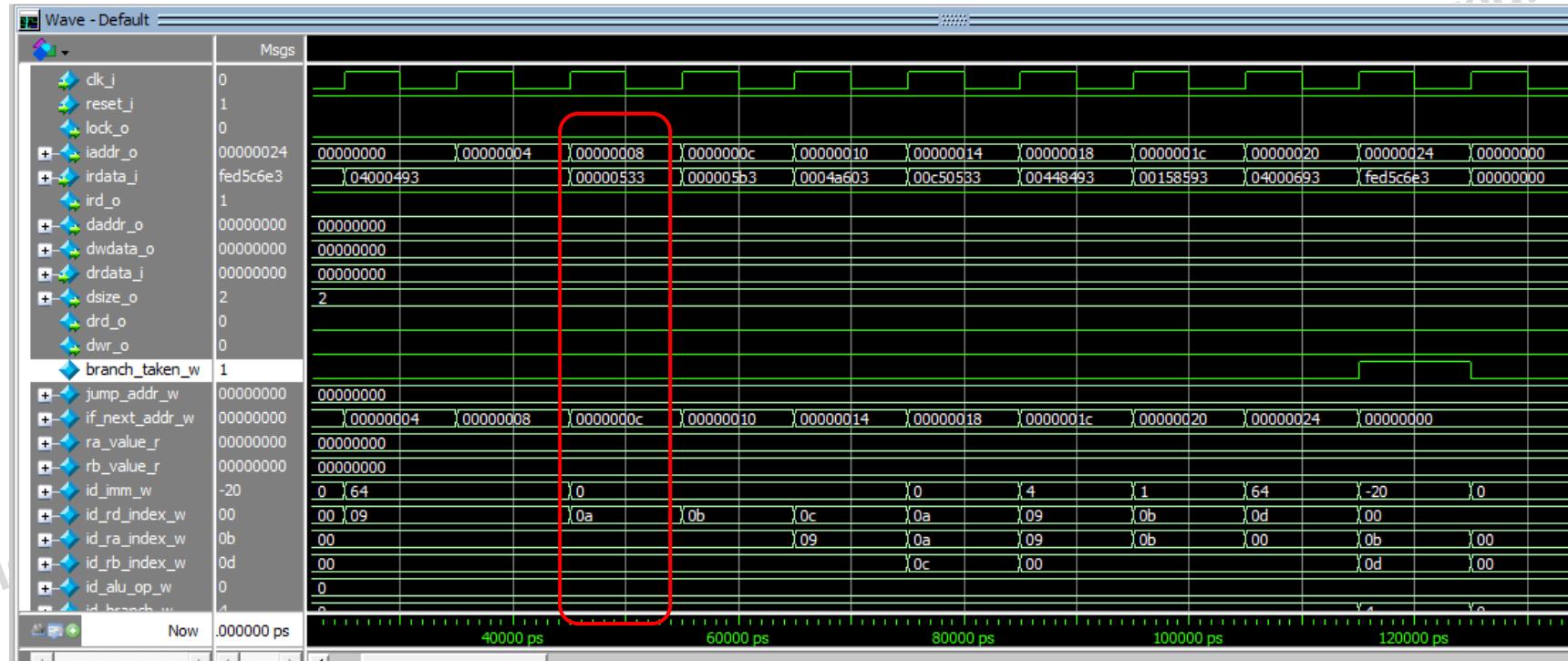
Waveform

- iaddr=0000
 - **addi x9, x0, 64 # x9=&A[0]**



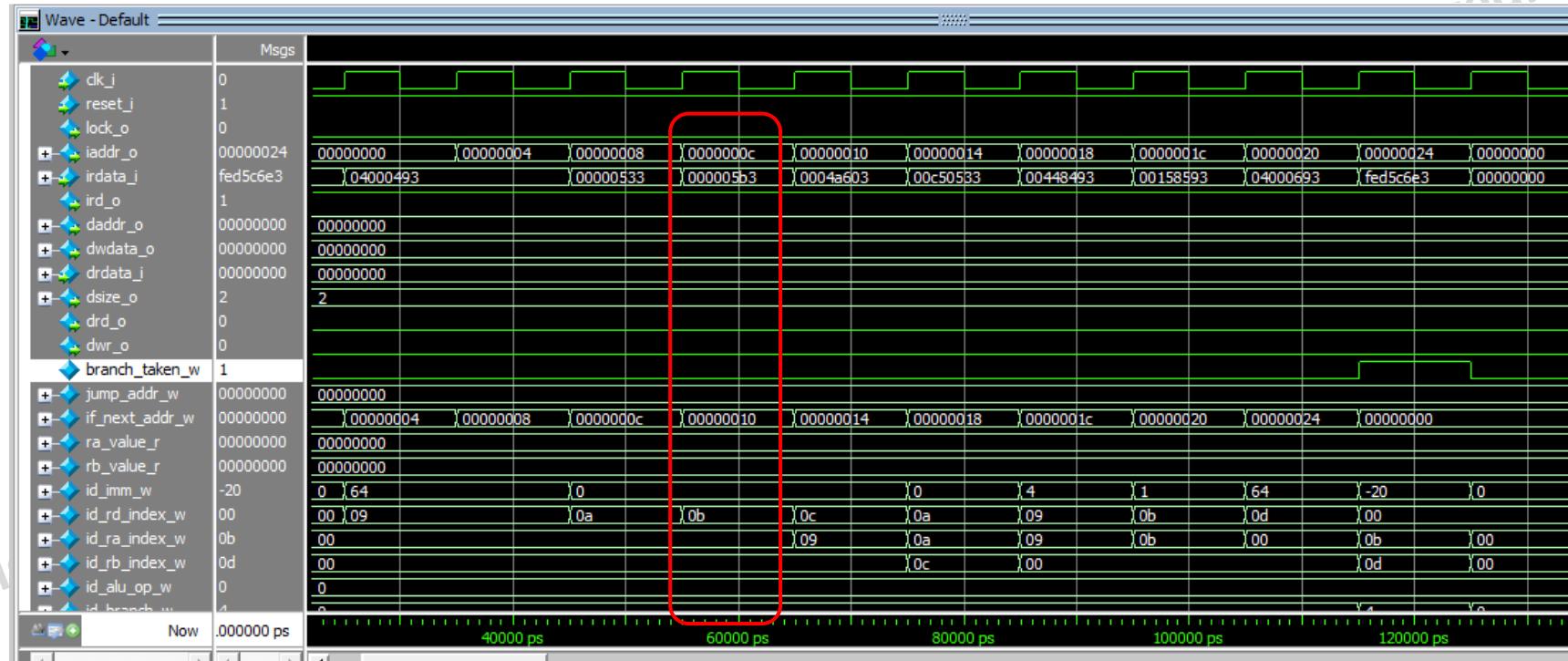
Waveform

- iaddr=0004
 - **add x10, x0, x0 # sum=0**



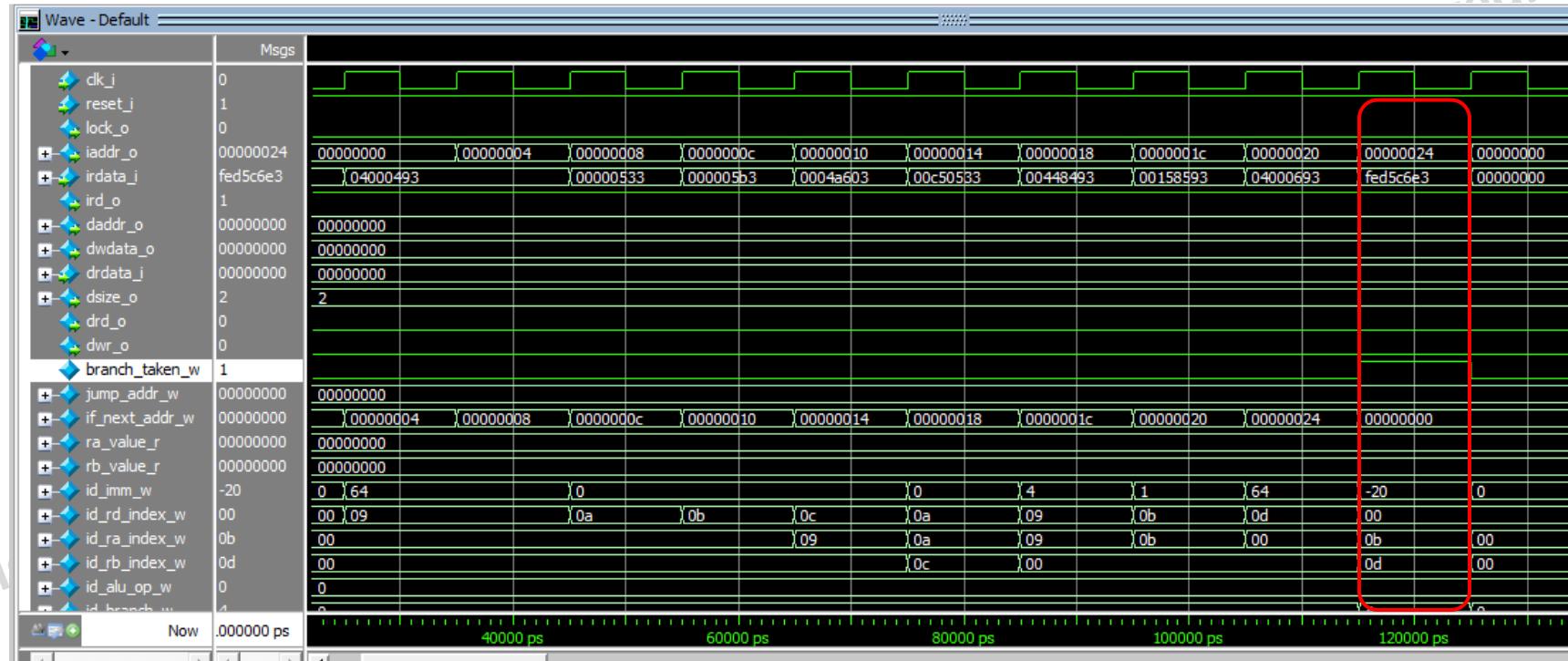
Waveform

- laddr=0008
 - **add x11, x0, x0 # i=0**



Waveform

- laddr=0020
 - **blt x11,x13,Loop**



TODO ...

- Reuse some codes:
 - riscv_regfile.v, riscv_alu.v from Lec#2
 - riscv_memory.v, riscv_decoder.v from Lec#3
 - riscv_pc from Lab1
- Implement riscv_core_sim.v by completing the missing codes
- Do simulation
- Show the waveform

 mem.hex	✓
 riscv_alu.v	✓
 riscv_core_sim.v	✓
 riscv_core_sim_bne_tb.v	✓
 riscv_core_sim_tb.v	✓
 riscv_decoder.v	✓
 riscvDefines.v	✓
 riscv_memory.v	✓
 riscv_pc.v	✓
 riscv_regfile.v	✓