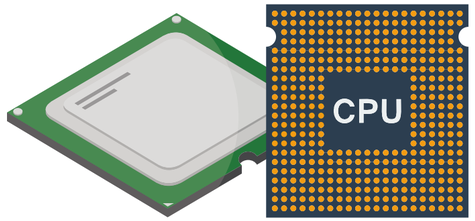
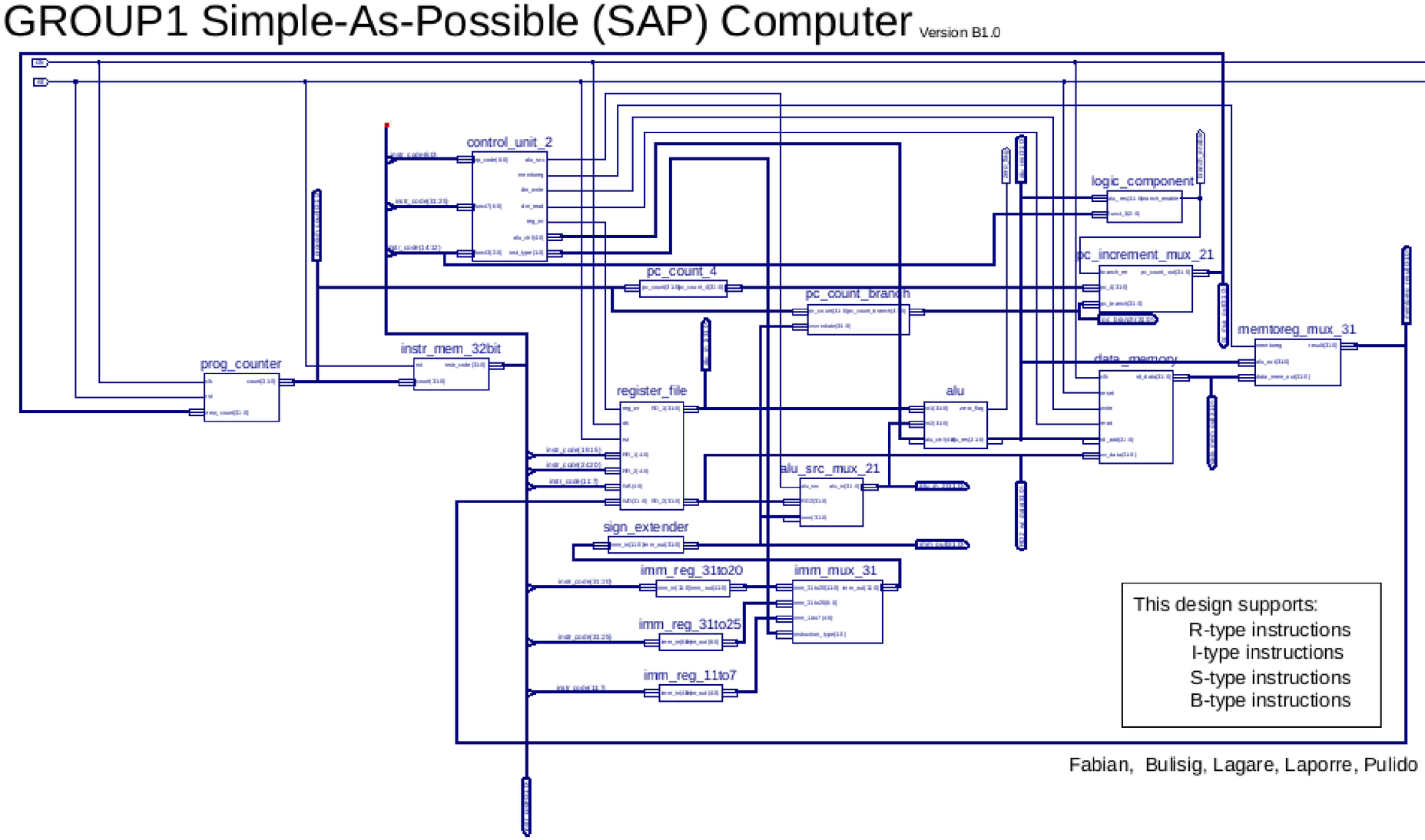


SAP (SIMPLE-AS-POSSIBLE) Computer: Design and Development

CPE131-L Project Presentation

Peter Miles Anthony Laporre
Florencio Enzo Pulido III
Louis Raphael Lagare
Bryxter Shem Bulisig
Ralph Edcel Fabian

Program Output



Program Output

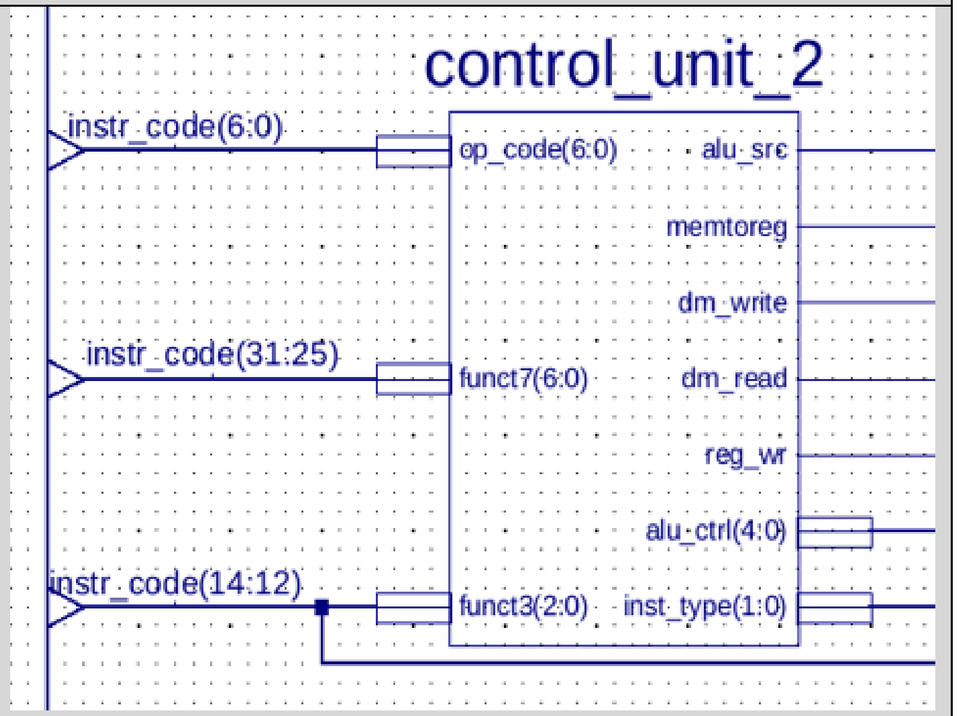
Control Unit

Component Description

The Control Unit sends control signals to the datapath components. This design explicitly extracts the data from the instruction rather than the whole.

31:25	24:20	19:15	14:12	11:7	6:0	
funct7	rs2	rs1	funct3	rd	op	R-Type
$imm_{11:0}$		rs1	funct3	rd	op	I-Type
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	S-Type
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	B-Type

Component Photo



Program Output

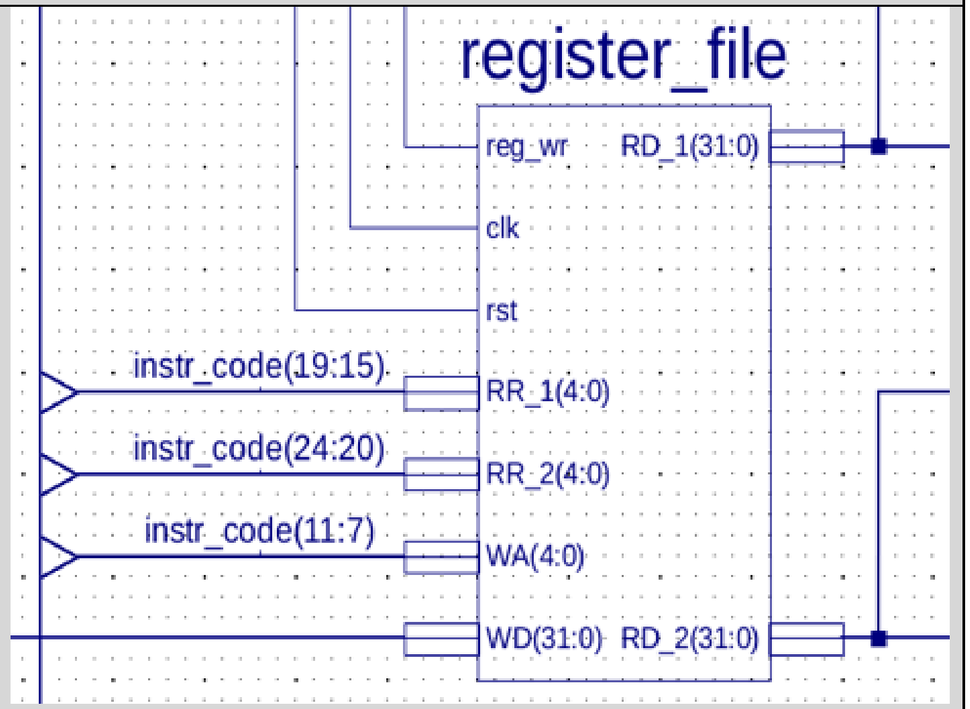
Register File

Component Description

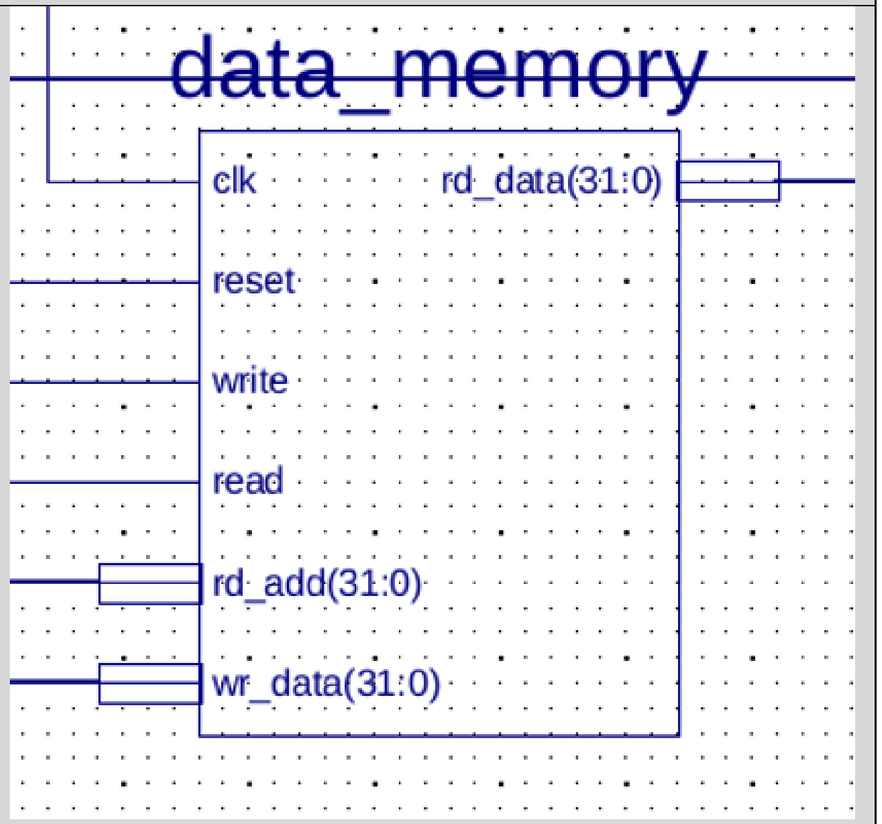
The register file stores the data to be feed to ALU or Data Memory. This design explicitly extracts the data from the instruction rather than the whole.

31:25	24:20	19:15	14:12	11:7	6:0	
funct7	rs2	rs1	funct3	rd	op	**R-Type**
$imm_{11:0}$		rs1	funct3	rd	op	**I-Type**
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	**S-Type**
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	**B-Type**

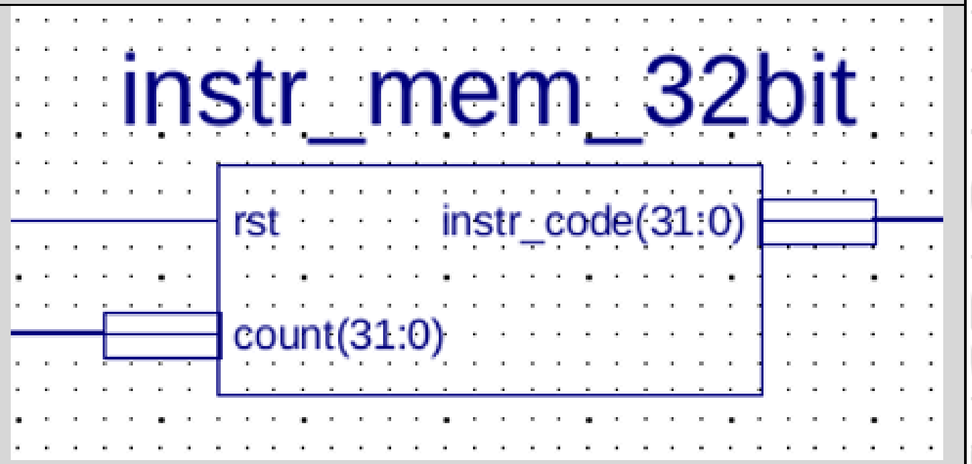
Component Photo



Program Output

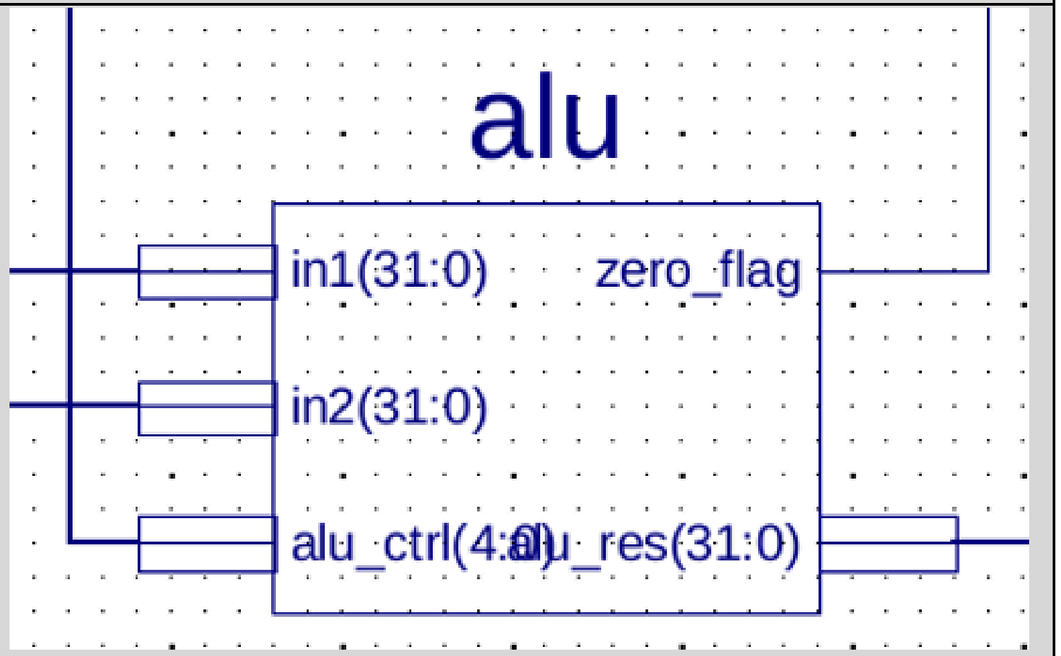
Data Memory	
Component Description	Component Photo
The data memory stores the data from Register File's RD2 to the effective address calculated by the ALU.	

Program Output

Instruction Memory	
Component Description	Component Photo
The instruction memory stores the instruction. This SAP design doesn't support manual input of instructions due to Xilinx capability. Thus, all instructions are predefined. It uses Little Endian (LSB to MSB)	

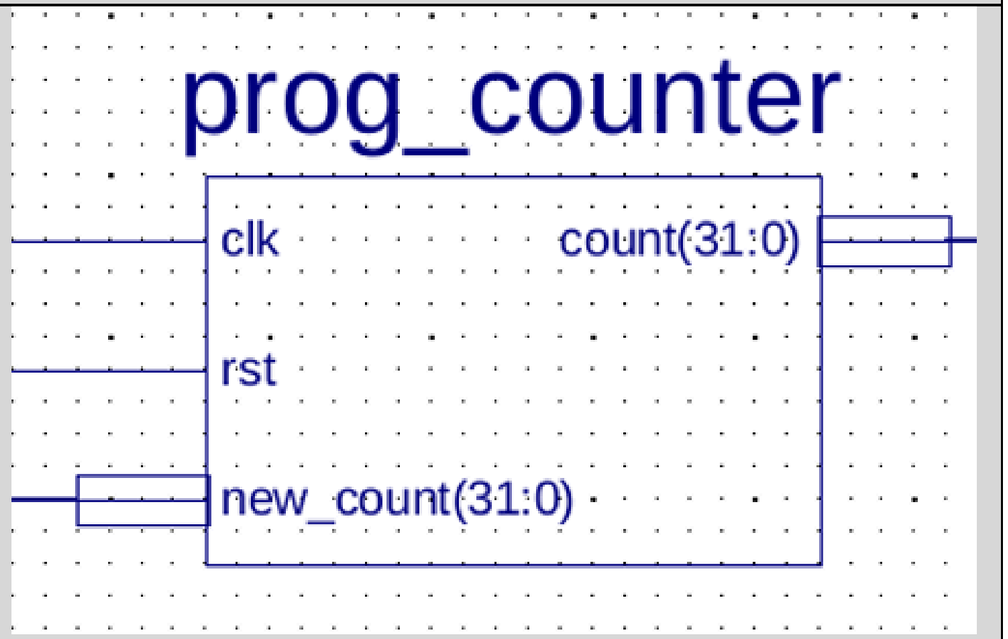

```
// Little-endian format (least significant byte first)

//sw x20, -1(x30) || x20 = 0xB; x30 = 0xA
mem[16'h0000] = 8'hA3;  // LSB (least significant byte)
mem[16'h0001] = 8'h2F;
mem[16'h0002] = 8'h4F;
mem[16'h0003] = 8'hFF;  // MSB (most significant byte)
```

Program Output

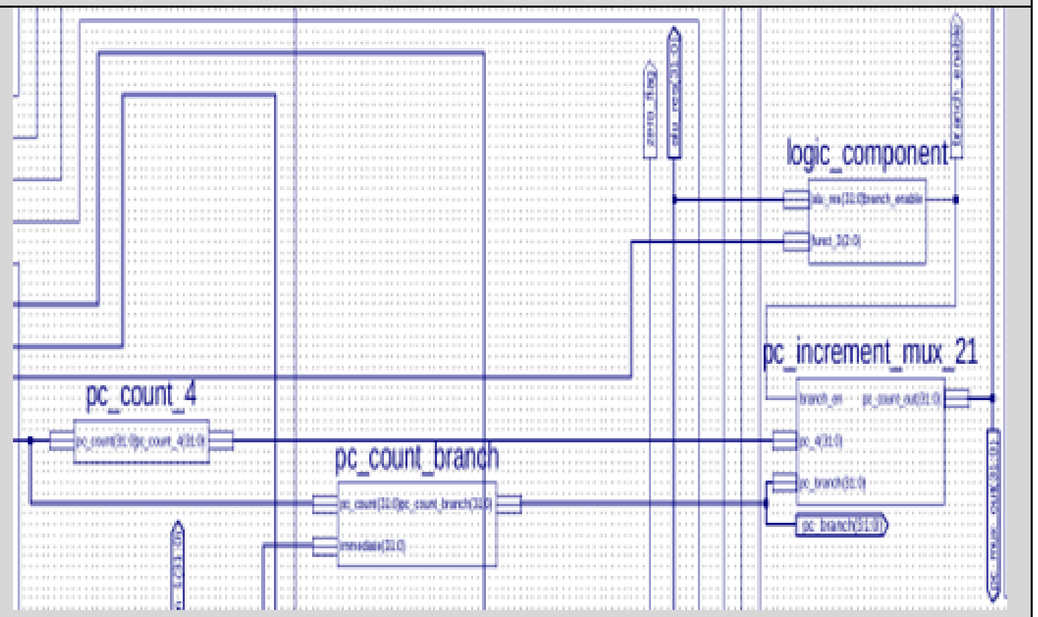
Arithmetic Logical Unit (ALU)	
Component Description	Component Photo
The component responsible for the calculation of the two inputs. The calculation differs from the alu_ctrl signal of the Control Unit. Examples are addittion, shift, and subtraction.	

Program Output

Program Counter	
Component Description	Component Photo

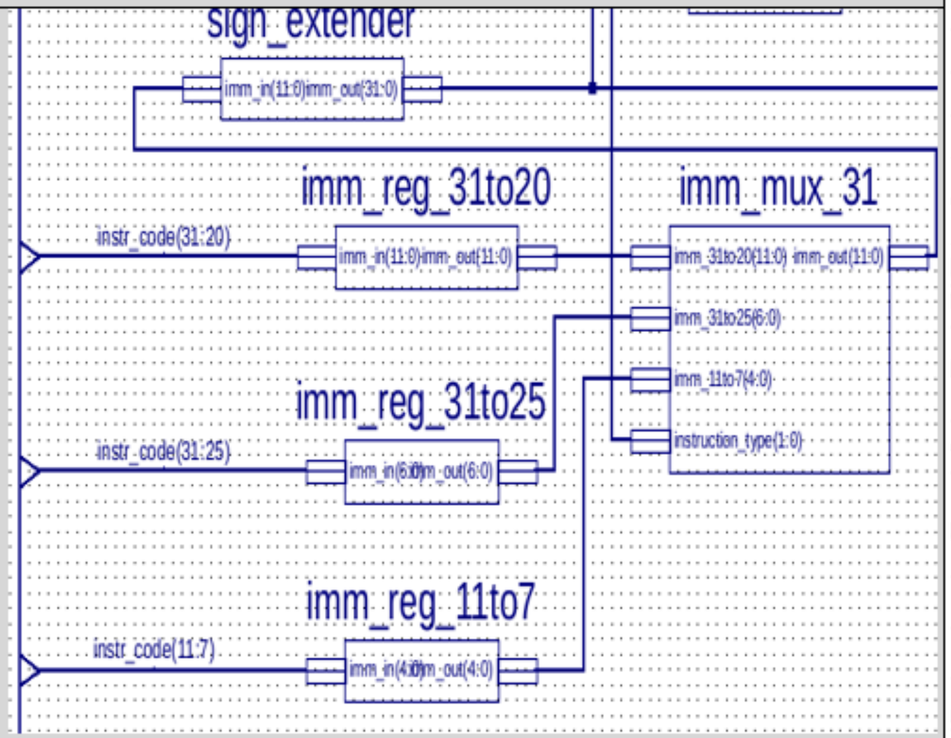
The program counter outputs the “new_count input” every posedge clk.
 	
`always @(posedge clk) begin`
`  if (rst) begin`
`  count = 32'h0;`
`  end`
`  else begin`
`  count = new_count;`
`  end`
`end`
	

Program Output

Program Counter Counting Components (Program Count + 4, Program Count Branch, Logic Component, 2 to 1 Multiplexer)	
Component Description	Component Photo
This SAP follow the datapath design for branch instructions. The Logic Component outputs the “branch enable” signal based on the ALU result and funct3. If enabled, the multiplexer will choose the branch address calculation; increments by 4 otherwise.	

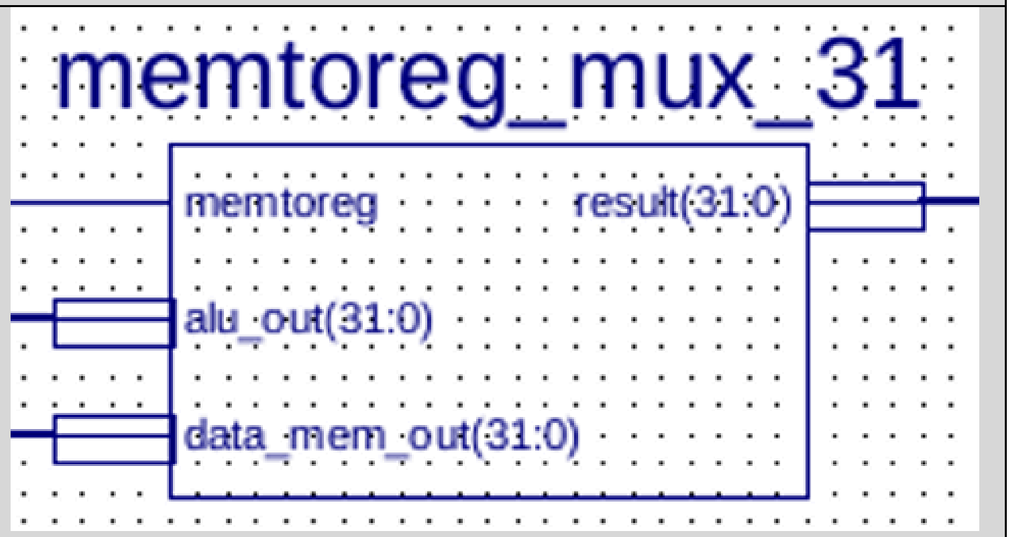
Program Output

Immediate Related Components
(Sign Extender, Immediate Multiplexer, and immediate registers)

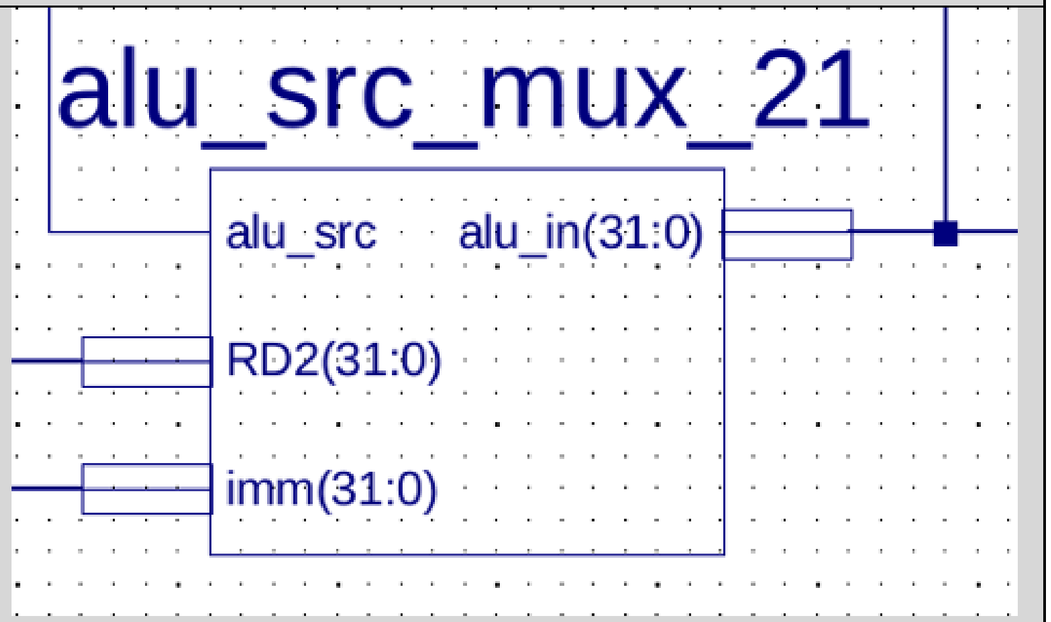
Component Description	Component Photo
This SAP design extracts the immediate values from different index. The immediate multiplexer decides what is the sequence of the bits based on the instruction type (signal from control unit). The sign extender pads the immediate value up to 32nd bit. The 12th bit is the padded bit.	

31:25	24:20	19:15	14:12	11:7	6:0	
funct7	rs2	rs1	funct3	rd	op	R-Type
$imm_{11:0}$		rs1	funct3	rd	op	I-Type
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	S-Type
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	B-Type

Program Output

MemtoReg Multiplexer	
Component Description	Component Photo
This multiplexer decides what values it will feed to the destination register on the register file.	

Program Output

ALUSource Multiplexer	
Component Description	Component Photo
This multiplexer decides what values it will feed to the 2nd input of the Arithmetic Logic Unit (ALU).	

Program Output

THE INSTRUCTIONS TABLE			
ASSUMED VALUES: **x30** = 0xA; **x20** = 0xB; **x31** = 0x9; **x21** = 0x5; **x1** = 0xE			
Count	**Abstract Code**	**Instruction**	**Interpretation**
0x0000	sw x20, -1(x30)	FF 4F 2F A3	@Data Memory: (0x9 = 0xB)
0x0004	sw x21, -2(x31)	FF 5F AF 23	@Data Memory: (0x7 = 0x5)
0x0008	lw x22, -1(x30)	FF FF 2B 03	@Register File: (x22 = 0xB)
0x000C	lw x23, -2(x31)	FF EF AB 83	@Register File: (x23 = 0x5)
0x0010	xor x24, x22, x23	01 7B 4C 33	1011_2 xor 0101_2 = 1110_2 = 0xE
0x0014	beq x24, x1, 8	00 1C 08 63	BTA = 0x0014 + (8_{10} * 2) = 0x0024
Skips (0x0018 – 0x0023)			
0x0024	addi x25, x24, -2	FF EC 0C 93	0xE + (-2_{10}) = 0xC

Program Output

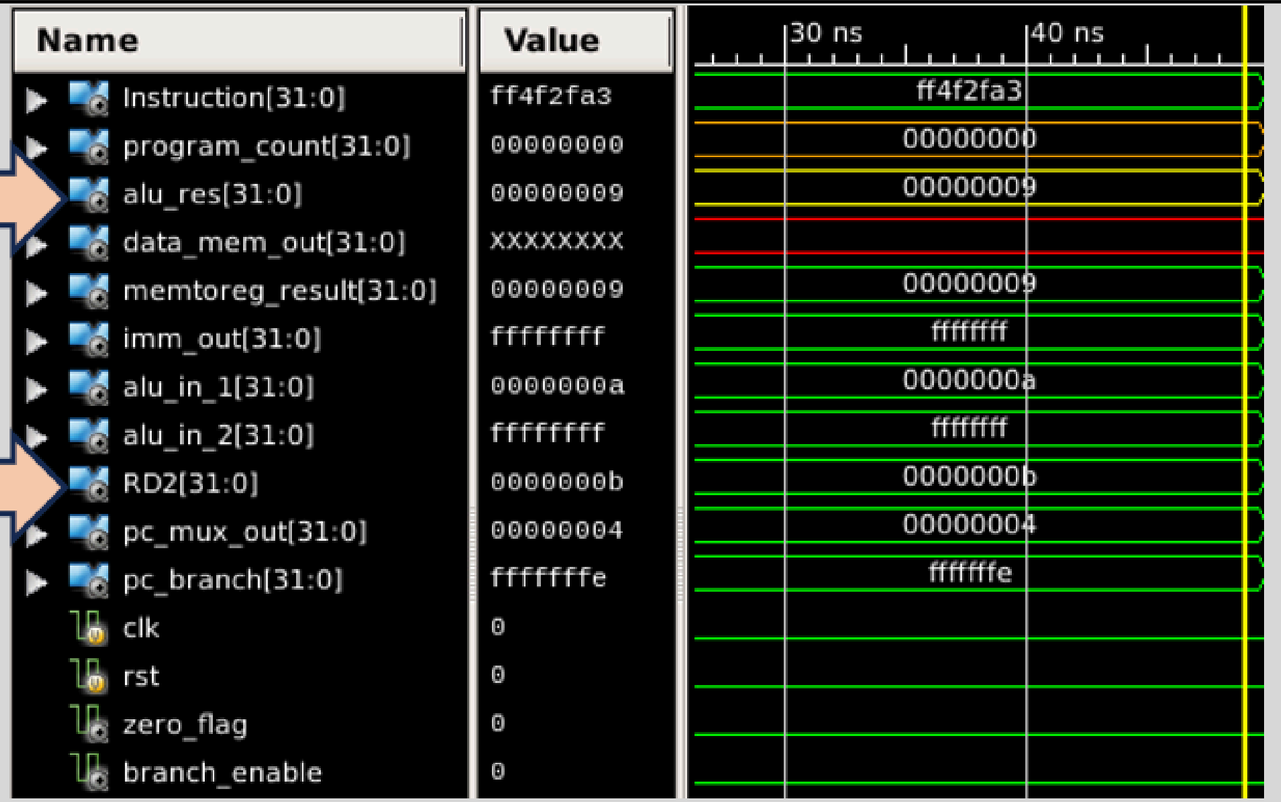
THE INSTRUCTIONS TABLE			
ASSUMED VALUES: **x30** = 0xA; **x20** = 0xB; **x31** = 0x9; **x21** = 0x5; **x1** = 0xE			
Count	**Abstract Code**	**Instruction**	**Interpretation**
0x0000	sw x20, -1(x30)	FF 4F 2F A3	@Data Memory: (0x9 = 0xB)
0x0004	sw x21, -2(x31)	FF 5F AF 23	@Data Memory: (0x7 = 0x5)
0x0008	lw x22, -1(x30)	FF FF 2B 03	@Register File: (x22 = 0xB)
0x000C	lw x23, -2(x31)	FF EF AB 83	@Register File: (x23 = 0x5)
0x0010	xor x24, x22, x23	01 7B 4C 33	1011_2 xor 0101_2 = 1110_2 = 0xE
0x0014	beq x24, x1, 8	00 1C 08 63	BTA = 0x0014 + (8_{10} * 2) = 0x0024
Skips (0x0018 – 0x0023)			
0x0024	addi x25, x24, -2	FF EC 0C 93	0xE + (-2_{10}) = 0xC

Program Output

Count	Abstract Code	Instruction	Interpretation
0x0000	sw x20, -1(x30)	FF 4F 2F A3	@Data Memory: (0x9 = 0xB)

The count starts from 0x0 with instruction code FF4F2FA3.

The calculated effective address of the ALU is stored with 0xB.

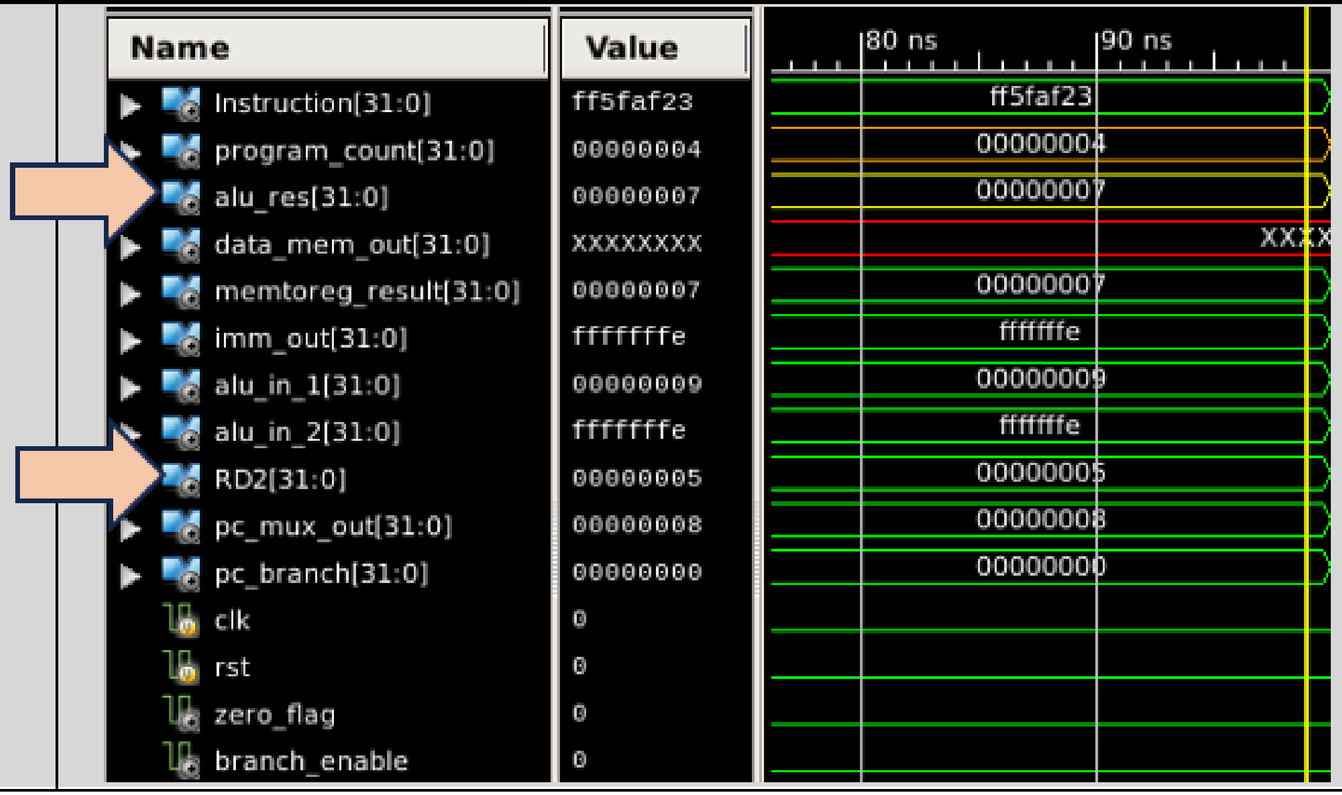


Program Output

Count	Abstract Code	Instruction	Interpretation
0x0004	sw x21, -2(x31)	FF 5F AF 23	@Data Memory: (0x7 = 0x5)

The count now is 0x4 with instruction code FF5FAF23.

The calculated effective address of the ALU is stored with 0x5.

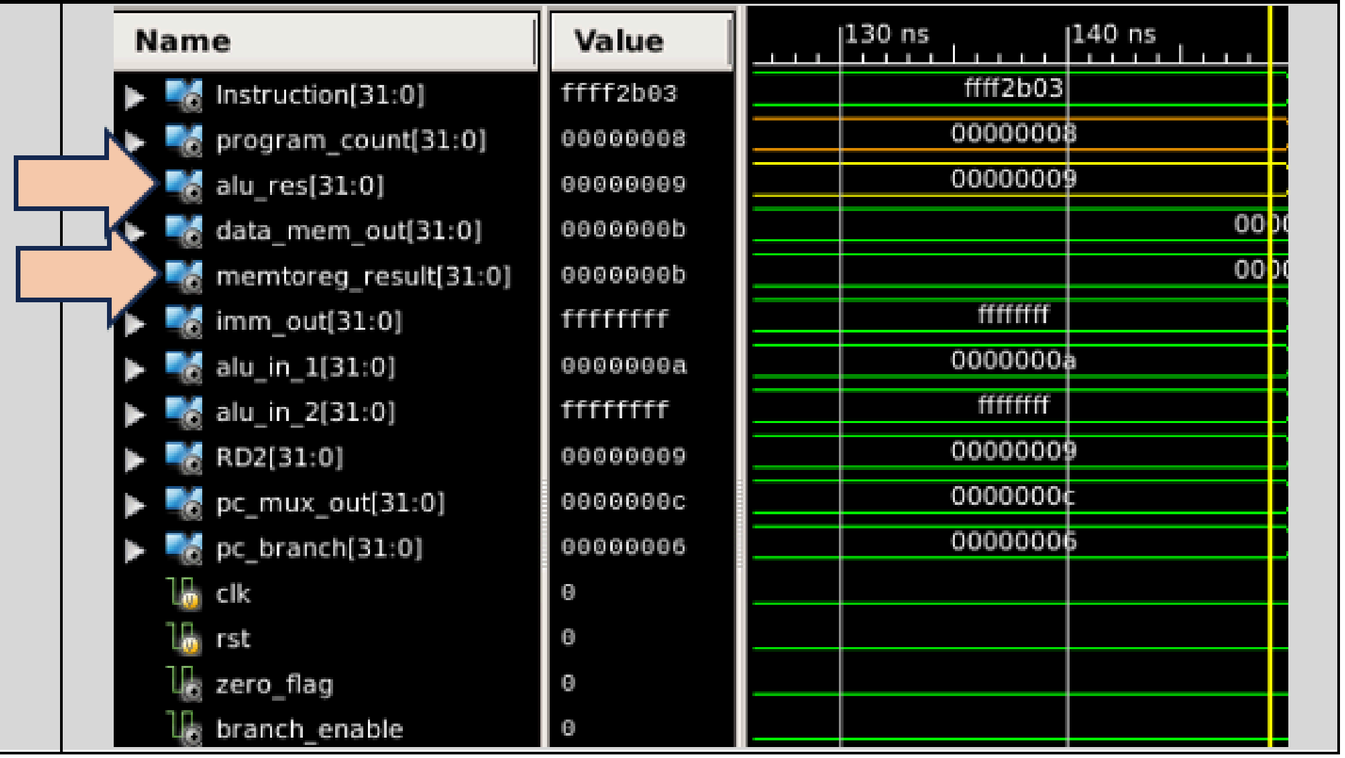


Program Output

Count	Abstract Code	Instruction	Interpretation
0x0008	lw x22, -1(x30)	FF FF 2B 03	@Register File: (x22 = 0xB)

The count now is 0x8 with instruction code FFFF2B03.

The value of the calculated effective address (0x9) in the data memory is loaded to the register file (x22 = 0xB).

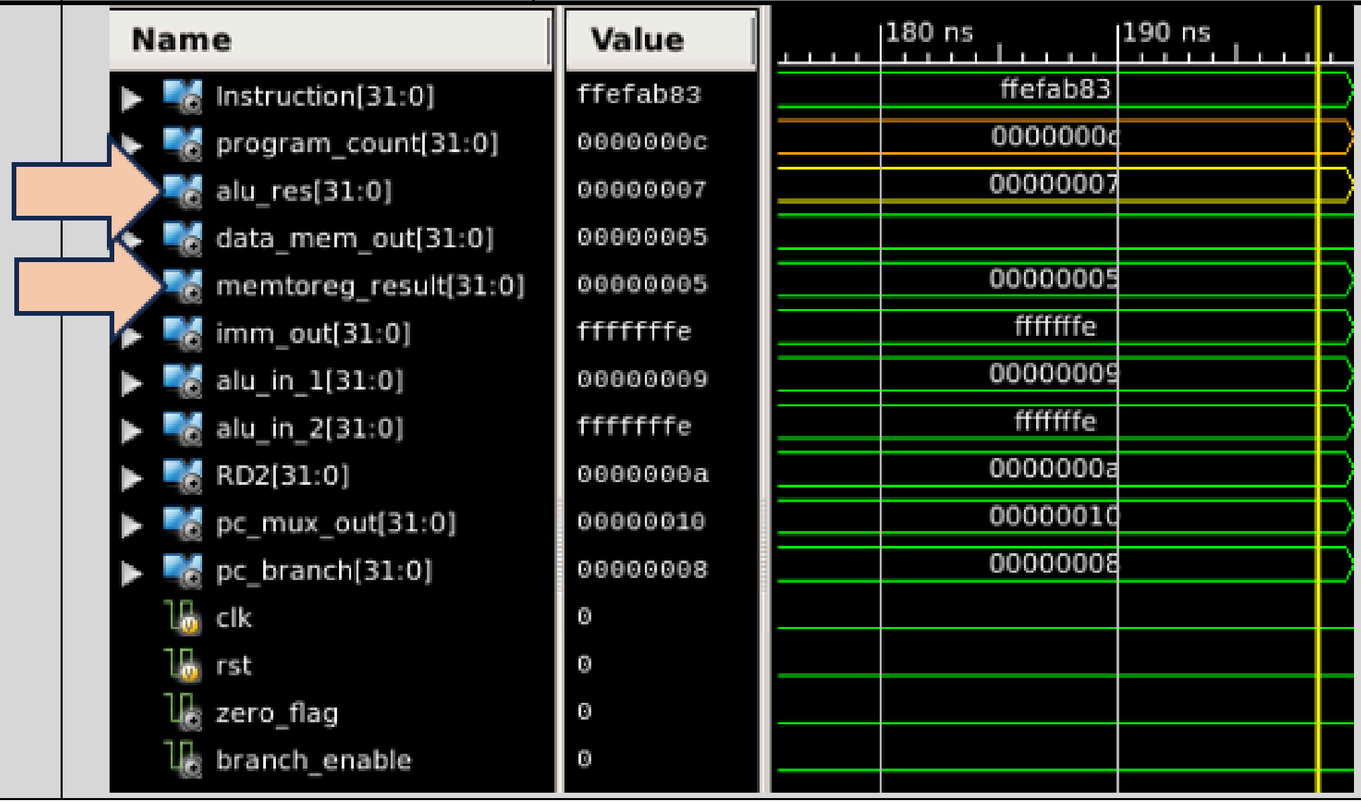


Program Output

Count	Abstract Code	Instruction	Interpretation
0x000C	lw x23, -2(x31)	FF EF AB 83	@Register File: (x23 = 0x5)

The count now is 0xC with instruction code FFEFAB83.

The value of the calculated effective address (0x7) in the data memory is loaded to the register file (0x7 = 0x5).



Program Output

Count	Abstract Code	Instruction	Interpretation
0x0010	xor x24, x22, x23	01 7B 4C 33	1011_2 xor 0101_2 = 1110_2 = 0xE

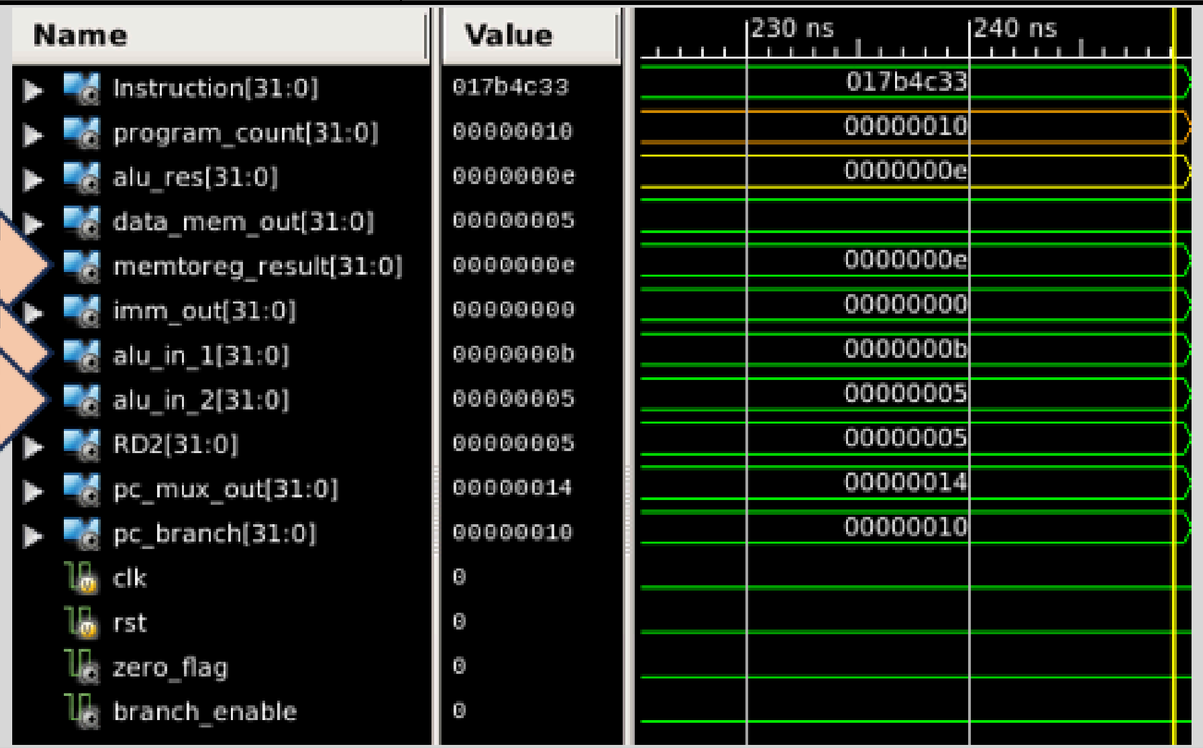
The count now is 0x10 with instruction code 017B4C33.

The value of the of x22 and x23 are with xored.

xor x24, x22, x23

0xE = 0xB xor 0x5

1011
0101
1110_2



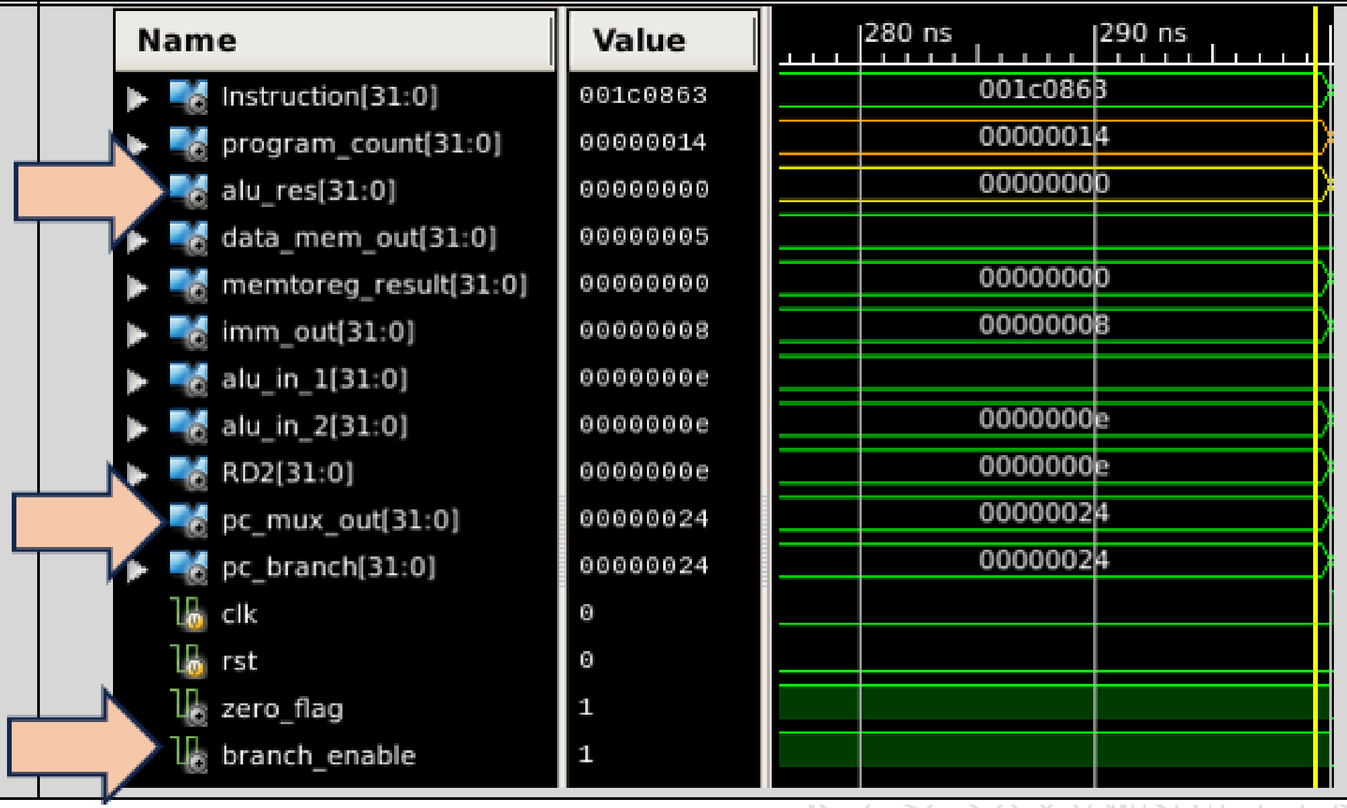
Program Output

Count	Abstract Code	Instruction	Interpretation
0x0014	beq x24, x1, 8	00 1C 08 63	BTA = 0x0014 + (8_{10} * 2) = 0x0024

The count now is 0x14 with instruction code 001C0863.

The Branch is enabled. Program Counter Multiplexer outputs 0x24 as the next Program Count

BTA = 0x0014 + (810 * 2) = 0x0024



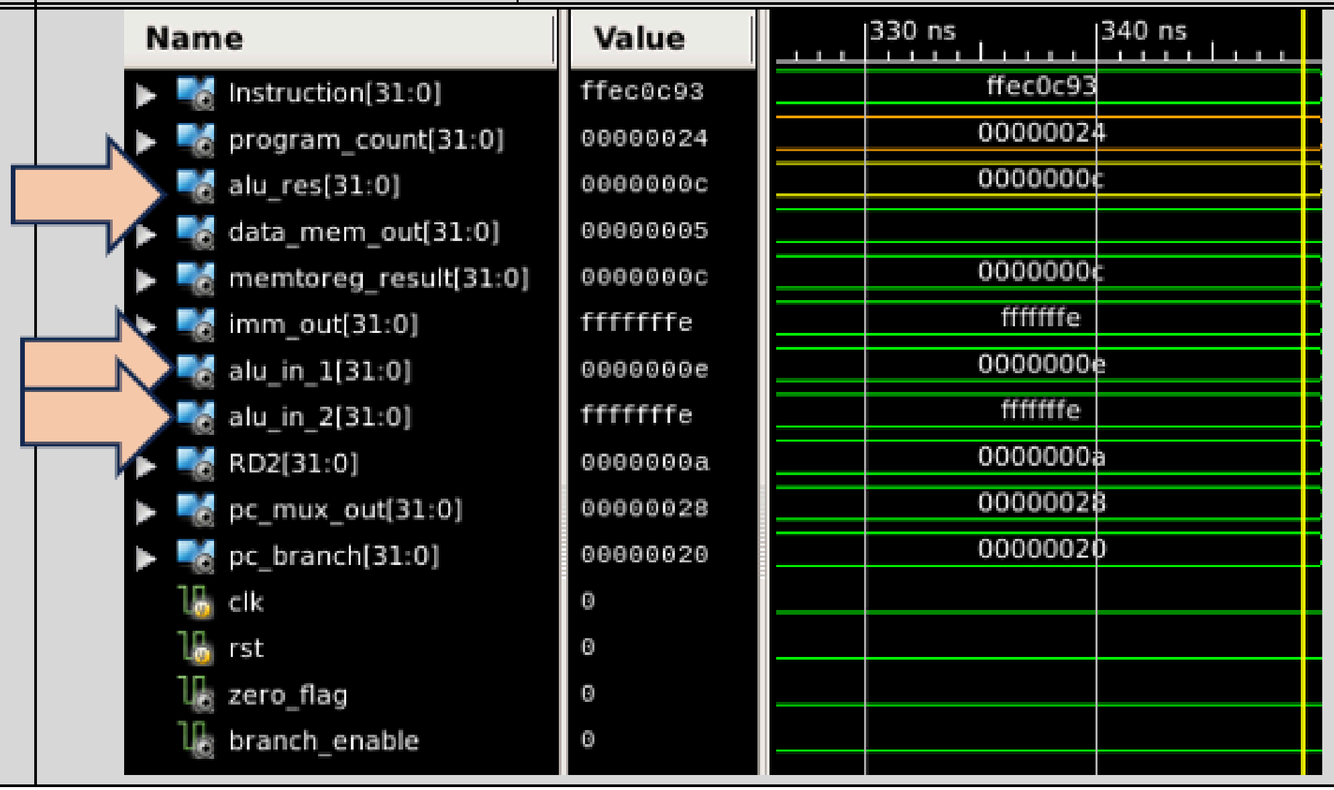
Program Output

Count	Abstract Code	Instruction	Interpretation
0x0024	addi x25, x24, -2	FF EC 0C 93	0xE + (-2_{10}) = 0xC

The count now is 0x24 after branching with instruction code FFEC0C93

The x25 of register file is stored with the addition of stored value in x24 and the immediate -2.

0xE + (-2_{10}) = 0xC



Findings, Observations, and Comments

1. Verilog to Schematic Approach
2. System First, Control Unit Later
3. Xilinx Virtual Machine Memory Constraints
4. Learning Concepts Along the Way
5. RISB Type Instruction Support

Appendices

Rollback 1
1 item

Rollback 2
1 item

v4_Fix Branch and Add logic unit
1 item

v5 Final Alpha
1 item

v6 test
1 item

v7 with logic controller
1 item

v8_branch with complete
1 item

V1 M1 Exam
120 items

V2 No control Unit
2 items

V3_Control Unit Addition
1 item

new file
20 bytes

Readme
20 bytes

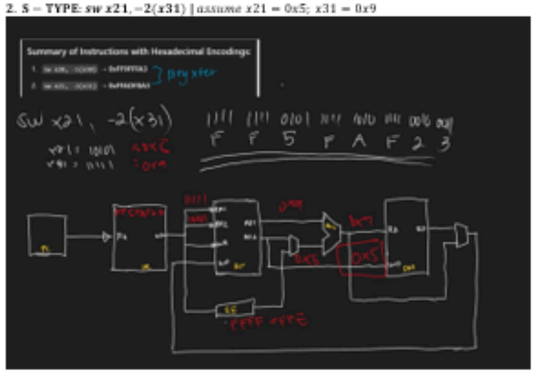
Appendices

31

APPENDICES

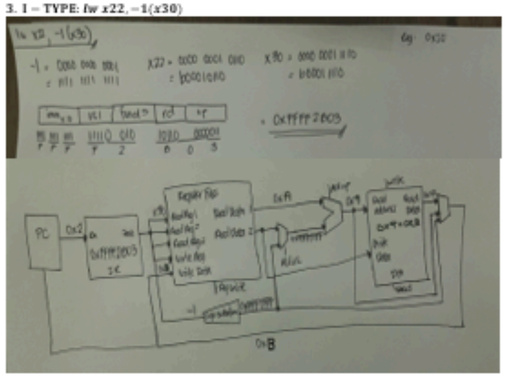
1. S – TYPE: *sw x20, −1(x30)* | *assume x20 = 0xB; x30 = 0xA*

2. S – TYPE: *sw x21, −2(x31)* | *assume x21 = 0x5; x31 = 0x9*

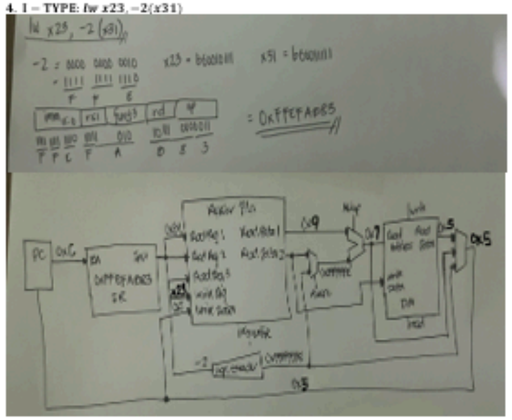


32

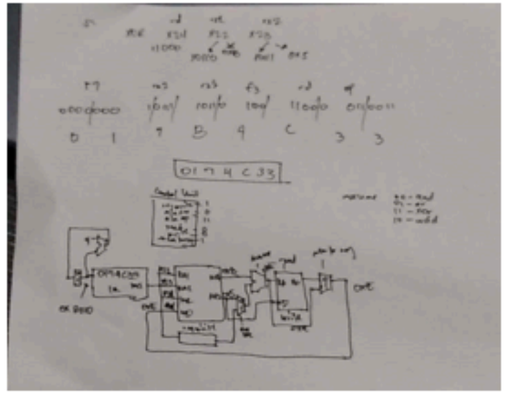
3. I – TYPE: *lw x22, −1(x30)*



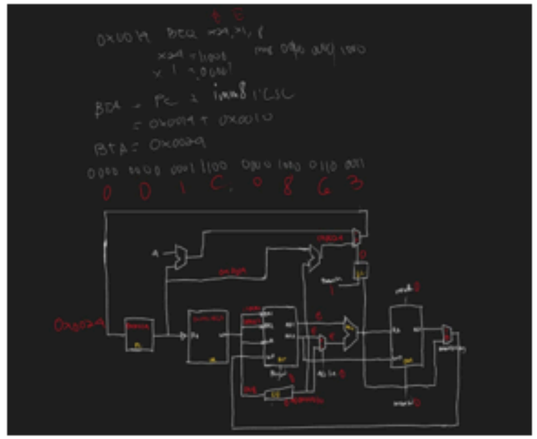
4. I – TYPE: *lw x23, −2(x31)*

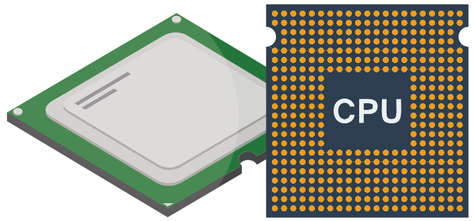


33



5. B – TYPE: *beq x24, x1, 8*





SAP (SIMPLE-AS-POSSIBLE) Computer: Design and Development

Thank you for Listening!