

SAP (SIMPLE-AS-POSSIBLE)

Computer: Design and Development

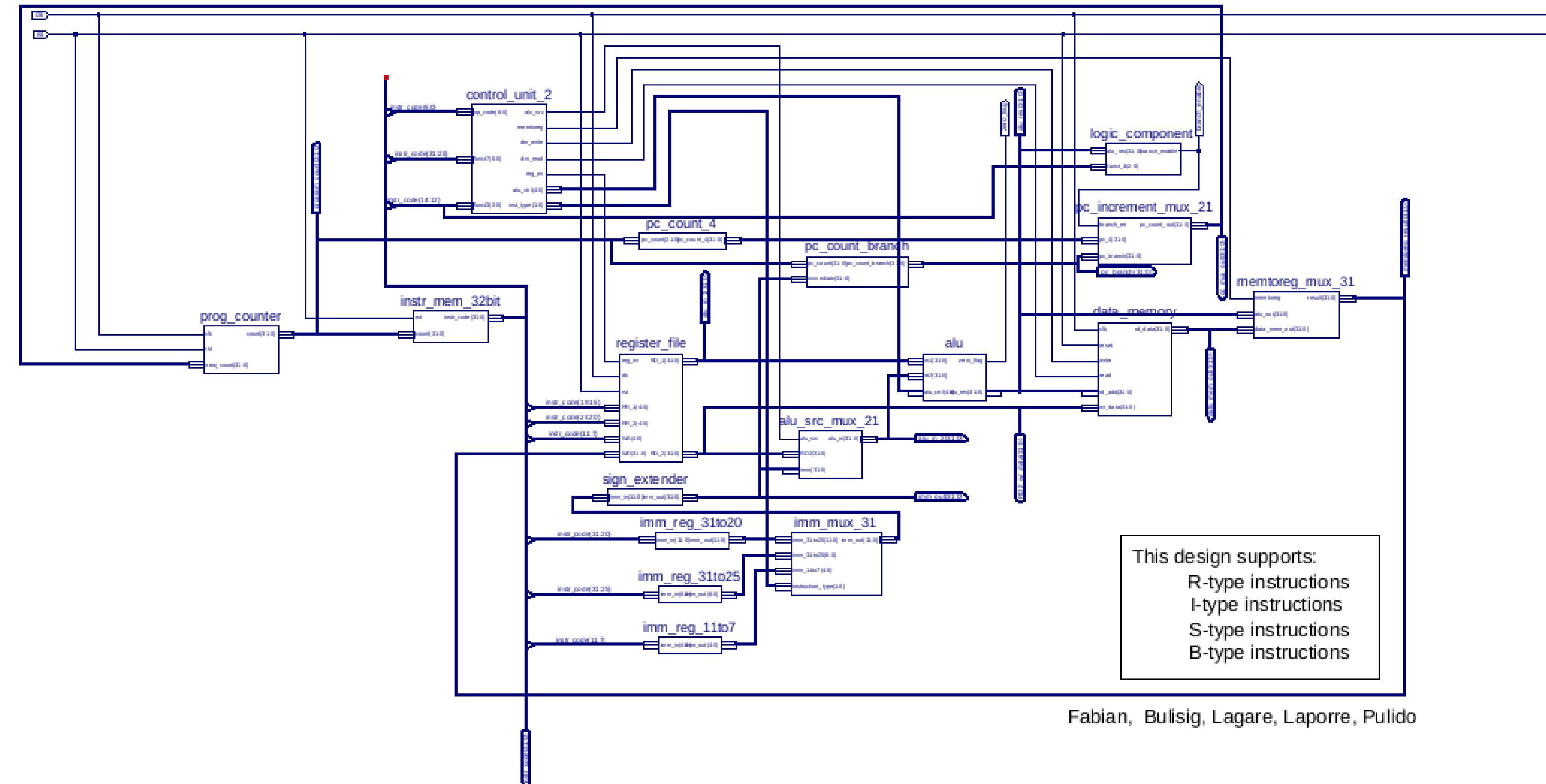
CPE131-L Project Presentation

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

Program Output

GROUP1 Simple-As-Possible (SAP) Computer

Version B1.0



Program Output

Control Unit																															
Component Description	Component Photo																														
<p>The Control Unit sends control signals to the datapath components. This design explicitly extracts the data from the instruction rather than the whole.</p>																															
<table border="1"><thead><tr><th>31:25</th><th>24:20</th><th>19:15</th><th>14:12</th><th>11:7</th><th>6:0</th></tr></thead><tbody><tr><td>funct7</td><td>rs2</td><td>rs1</td><td>funct3</td><td>rd</td><td>op</td></tr><tr><td>imm_{11:0}</td><td></td><td>rs1</td><td>funct3</td><td>rd</td><td>op</td></tr><tr><td>imm_{11:5}</td><td>rs2</td><td>rs1</td><td>funct3</td><td>imm_{4:0}</td><td>op</td></tr><tr><td>imm_{12,10:5}</td><td>rs2</td><td>rs1</td><td>funct3</td><td>imm_{4:1,11}</td><td>op</td></tr></tbody></table> <p>R-Type I-Type S-Type B-Type</p>	31:25	24:20	19:15	14:12	11:7	6:0	funct7	rs2	rs1	funct3	rd	op	imm _{11:0}		rs1	funct3	rd	op	imm _{11:5}	rs2	rs1	funct3	imm _{4:0}	op	imm _{12,10:5}	rs2	rs1	funct3	imm _{4:1,11}	op	<p>control_unit_2</p> <pre>graph LR; instr_code[instr_code(31:25)] --> op_code[op_code(6:0)]; instr_code --> funct7[funct7(6:0)]; instr_code --> imm40[imm<sub>4:0</sub>]; instr_code --> imm110[imm<sub>11:0</sub>]; instr_code --> imm12105[imm<sub>12,10:5</sub>]; op_code --> alu_src[alu_src]; op_code --> memtoreg[memtoreg]; op_code --> dm_write[dm_write]; op_code --> dm_read[dm_read]; op_code --> reg_wr[reg_wr]; op_code --> alu_ctrl[alu_ctrl(4:0)]; funct7 --> funct320[funct3(2:0)]; funct7 --> inst_type10[inst_type(1:0)]; imm40 --> rd[rd]; imm40 --> op[op]; imm110 --> rs2[rs2]; imm110 --> rs1[rs1]; imm12105 --> rs2[rs2]; imm12105 --> rs1[rs1]; imm12105 --> funct3[funct3];</pre>
31:25	24:20	19:15	14:12	11:7	6:0																										
funct7	rs2	rs1	funct3	rd	op																										
imm _{11:0}		rs1	funct3	rd	op																										
imm _{11:5}	rs2	rs1	funct3	imm _{4:0}	op																										
imm _{12,10:5}	rs2	rs1	funct3	imm _{4:1,11}	op																										

Program Output

Register File																															
Component Description	Component Photo																														
<p>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.</p>																															
<table border="1"><thead><tr><th>31:25</th><th>24:20</th><th>19:15</th><th>14:12</th><th>11:7</th><th>6:0</th></tr></thead><tbody><tr><td>funct7</td><td>rs2</td><td>rs1</td><td>funct3</td><td>rd</td><td>op</td></tr><tr><td>imm_{11:0}</td><td></td><td>rs1</td><td>funct3</td><td>rd</td><td>op</td></tr><tr><td>imm_{11:5}</td><td>rs2</td><td>rs1</td><td>funct3</td><td>imm_{4:0}</td><td>op</td></tr><tr><td>imm_{12,10:5}</td><td>rs2</td><td>rs1</td><td>funct3</td><td>imm_{4:1,11}</td><td>op</td></tr></tbody></table> <p>R-Type I-Type S-Type B-Type</p>	31:25	24:20	19:15	14:12	11:7	6:0	funct7	rs2	rs1	funct3	rd	op	imm _{11:0}		rs1	funct3	rd	op	imm _{11:5}	rs2	rs1	funct3	imm _{4:0}	op	imm _{12,10:5}	rs2	rs1	funct3	imm _{4:1,11}	op	<p>register_file</p> <p>reg_wr RD_1(31:0)</p> <p>clk</p> <p>rst</p> <p>instr_code(19:15)</p> <p>RR_1(4:0)</p> <p>instr_code(24:20)</p> <p>RR_2(4:0)</p> <p>instr_code(11:7)</p> <p>WA(4:0)</p> <p>WD(31:0) RD_2(31:0)</p>
31:25	24:20	19:15	14:12	11:7	6:0																										
funct7	rs2	rs1	funct3	rd	op																										
imm _{11:0}		rs1	funct3	rd	op																										
imm _{11:5}	rs2	rs1	funct3	imm _{4:0}	op																										
imm _{12,10:5}	rs2	rs1	funct3	imm _{4:1,11}	op																										

Program Output

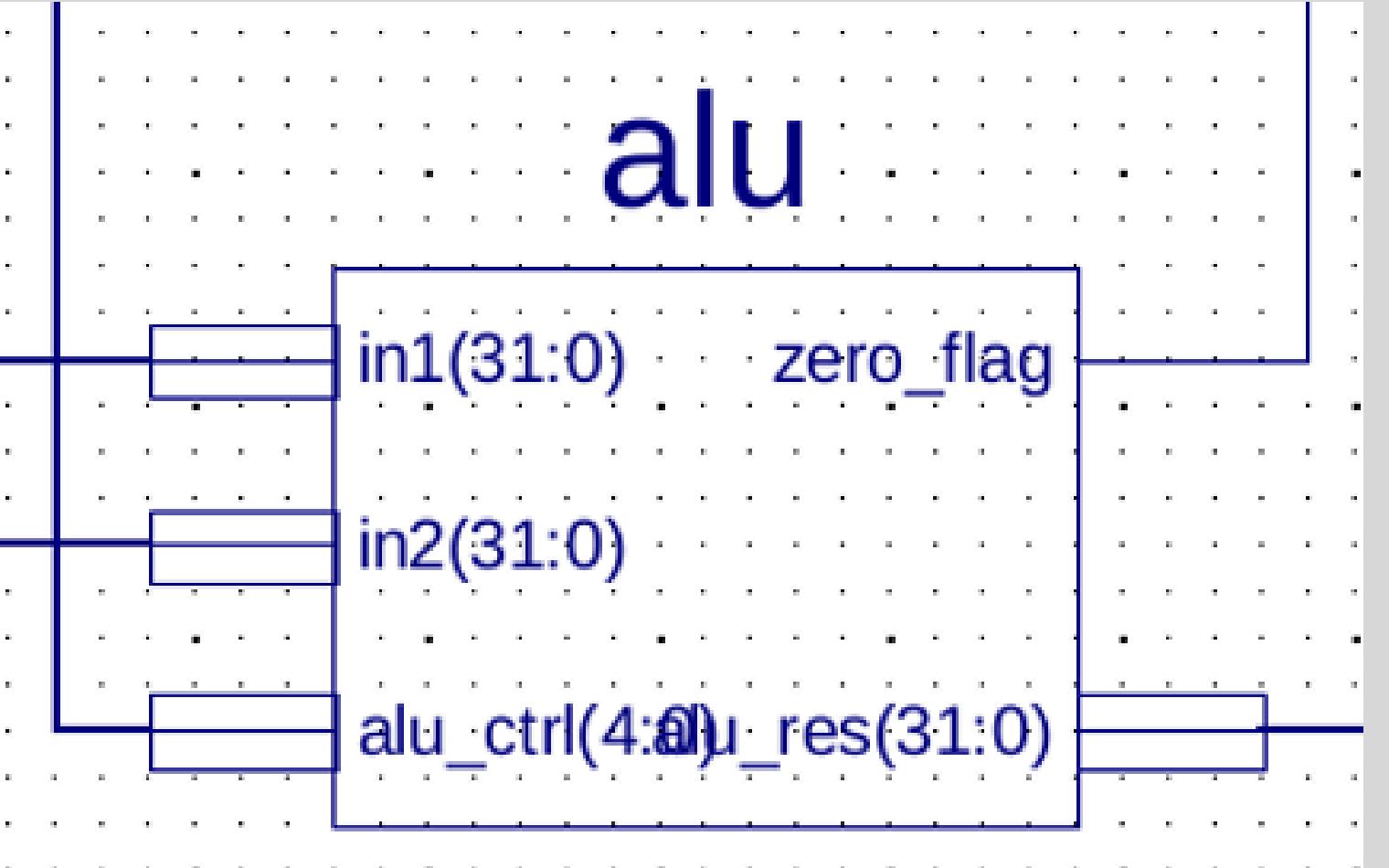
Data Memory	
Component Description	Component Photo
<p>The data memory stores the data from Register File's RD2 to the effective address calculated by the ALU.</p>	<p>The diagram illustrates a data memory component. It features a large grid of small squares representing memory cells. To the left of the grid, there are several control signal lines labeled in blue: clk, reset, write, read, rd_data(31:0), rd_add(31:0), and wr_data(31:0). The clk and rd_data lines are connected to the top edge of the grid, while the other control signals are connected to the left edge.</p>

Program Output

Instruction Memory	
Component Description	Component Photo
<p>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)</p> <pre>// 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)</pre>	 <p>The diagram shows a rectangular component labeled "instr_mem_32bit". Inside the component, there are two ports: "rst" and "instr_code(31:0)". Below these ports is another port labeled "count(31:0)". The component is connected to a grid of dots representing memory storage.</p>

Program Output

Arithmetic Logical Unit (ALU)

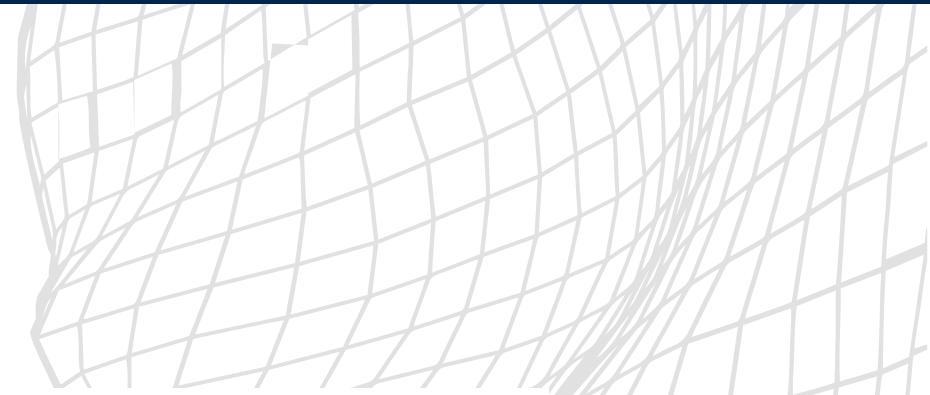
Component Description	Component Photo
<p>The component responsible for the calculation of the two inputs. The calculation differs from the alu_ctrl signal of the Control Unit. Examples are addition, shift, and subtraction.</p>	 <p>Block diagram of an ALU component. It has three inputs: <code>in1(31:0)</code>, <code>in2(31:0)</code>, and <code>alu_ctrl(4:0)</code>. The output is <code>u_res(31:0)</code>. The word alu is written above the component.</p>

Program Output

Program Counter

Component Description	Component Photo
<p>The program counter outputs the “new_count input” every posedge clk.</p> <pre>always @ (posedge clk) begin if (rst) begin count = 32'h0; end else begin count = new_count; end end</pre>	<p>prog_counter</p> <pre>graph LR clk((clk)) --> prog[prog_counter] rst((rst)) --> prog new_count((new_count(31:0))) --> prog prog --> count(count(31:0))</pre>

Program Output



Program Counter Counting Components (Program Count + 4, Program Count Branch, Logic Component, 2 to 1 Multiplexer)

Component Description	Component Photo
<p>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.</p>	<pre>graph LR PC[PC] --> PC4[pc_count_4] PC[PC] --> PCB[pc_count_branch] PC[PC] --> LC[logic_component] PC4 --> IMUX[pc_increment_mux_21] PCB --> IMUX LC --> IMUX LC --> BE[branch enable] BE --> IMUX IMUX --> PA[PC branch address]</pre>

Program Output

Immediate Related Components (Sign Extender, Immediate Multiplexer, and immediate registers)																															
Component Description	Component Photo																														
<p>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.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>31:25</th><th>24:20</th><th>19:15</th><th>14:12</th><th>11:7</th><th>6:0</th></tr> <tr> <td>funct7</td><td>rs2</td><td>rs1</td><td>funct3</td><td>rd</td><td>op</td></tr> <tr> <td>imm_{11:0}</td><td></td><td>rs1</td><td>funct3</td><td>rd</td><td>op</td></tr> <tr> <td>imm_{11:5}</td><td>rs2</td><td>rs1</td><td>funct3</td><td>imm_{4:0}</td><td>op</td></tr> <tr> <td>imm_{12,10:5}</td><td>rs2</td><td>rs1</td><td>funct3</td><td>imm_{4:1,11}</td><td>op</td></tr> </table> <p>R-Type I-Type S-Type B-Type</p>	31:25	24:20	19:15	14:12	11:7	6:0	funct7	rs2	rs1	funct3	rd	op	imm _{11:0}		rs1	funct3	rd	op	imm _{11:5}	rs2	rs1	funct3	imm _{4:0}	op	imm _{12,10:5}	rs2	rs1	funct3	imm _{4:1,11}	op	
31:25	24:20	19:15	14:12	11:7	6:0																										
funct7	rs2	rs1	funct3	rd	op																										
imm _{11:0}		rs1	funct3	rd	op																										
imm _{11:5}	rs2	rs1	funct3	imm _{4:0}	op																										
imm _{12,10:5}	rs2	rs1	funct3	imm _{4:1,11}	op																										

Program Output

MemtoReg Multiplexer	
Component Description	Component Photo
This multiplexer decides what values it will feed to the destination register on the register file.	<p>The diagram shows a logic block labeled "memtoreg_mux_31". It has three input ports: "memtoreg(31:0)", "alu_out(31:0)", and "data_mem_out(31:0)". The output port is "result(31:0)". The block is represented by a rectangle with a grid pattern inside, and the port labels are in blue text.</p>

Program Output

ALUSource Multiplexer	
Component Description	Component Photo
This multiplexer decides what values it will feed to the 2 nd input of the Arithmetic Logic Unit (ALU).	<p>The diagram shows a logic circuit for an ALU source multiplexer. It has three inputs: <code>alu_src</code>, <code>RD2(31:0)</code>, and <code>imm(31:0)</code>. The output is labeled <code>alu_in(31:0)</code>. The circuit consists of a 3-to-1 multiplexer where <code>alu_src</code> is the select line, <code>RD2</code> is the first data input, and <code>imm</code> is the second data input.</p>

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	$1011_2 \text{ xor } 0101_2 = 1110_2 = 0xE$
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			
Count	Abstract Code	Instruction	Interpretation
0x0000	sw x20, -1(x30) 0xA	FF 4F 2F A3	@Data Memory: (0x9 = 0xB)
0x0004	sw x21, -2(x31) 0x5	FF 5F AF 23	@Data Memory: (0x7 = 0x5)
0x0008	lw x22, -1(x30) 0xB	FF FF 2B 03	@Register File: (x22 = 0xB)
0x000C	lw x23, -2(x31) 0x5	FF EF AB 83	@Register File: (x23 = 0x5)
0x0010	xor x24, x22, x23 0xE 0xE 0x5	01 7B 4C 33	$1011_2 \text{ xor } 0101_2 = 1110_2 = 0xE$
0x0014	beq x24, x1, 8 0xE 0xE	00 1C 08 63	BTA = $0x0014 + (8_{10} * 2) = 0x0024$
Skips (0x0018 – 0x0023)			
0x0024	addi x25, x24, -2 0xE	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.	<table border="1"> <thead> <tr> <th>Name</th><th>Value</th></tr> </thead> <tbody> <tr> <td>► Instruction[31:0]</td><td>ff4f2fa3</td></tr> <tr> <td>► program_count[31:0]</td><td>00000000</td></tr> <tr> <td>► alu_res[31:0]</td><td>00000009</td></tr> <tr> <td>► data_mem_out[31:0]</td><td>XXXXXXXX</td></tr> <tr> <td>► memtoreg_result[31:0]</td><td>00000009</td></tr> <tr> <td>► imm_out[31:0]</td><td>ffffffffff</td></tr> <tr> <td>► alu_in_1[31:0]</td><td>0000000a</td></tr> <tr> <td>► alu_in_2[31:0]</td><td>ffffffffff</td></tr> <tr> <td>► RD2[31:0]</td><td>0000000b</td></tr> <tr> <td>► pc_mux_out[31:0]</td><td>00000004</td></tr> <tr> <td>► pc_branch[31:0]</td><td>fffffffffe</td></tr> <tr> <td>► clk</td><td>0</td></tr> <tr> <td>► rst</td><td>0</td></tr> <tr> <td>► zero_flag</td><td>0</td></tr> <tr> <td>► branch_enable</td><td>0</td></tr> </tbody> </table>	Name	Value	► Instruction[31:0]	ff4f2fa3	► program_count[31:0]	00000000	► alu_res[31:0]	00000009	► data_mem_out[31:0]	XXXXXXXX	► memtoreg_result[31:0]	00000009	► imm_out[31:0]	ffffffffff	► alu_in_1[31:0]	0000000a	► alu_in_2[31:0]	ffffffffff	► RD2[31:0]	0000000b	► pc_mux_out[31:0]	00000004	► pc_branch[31:0]	fffffffffe	► clk	0	► rst	0	► zero_flag	0	► branch_enable	0	<p>Timing Diagram:</p> <ul style="list-style-type: none"> 30 ns: ff4f2fa3, 00000000, 00000009, XXXXXX, 00000009, ffffffff, 0000000a, ffffffff, 0000000b, 00000004, fffffffe 40 ns: ff4f2fa3, 00000000, 00000009, 00000009, ffffffff, 0000000a, ffffffff, 0000000b, 00000004, fffffffe
Name	Value																																		
► Instruction[31:0]	ff4f2fa3																																		
► program_count[31:0]	00000000																																		
► alu_res[31:0]	00000009																																		
► data_mem_out[31:0]	XXXXXXXX																																		
► memtoreg_result[31:0]	00000009																																		
► imm_out[31:0]	ffffffffff																																		
► alu_in_1[31:0]	0000000a																																		
► alu_in_2[31:0]	ffffffffff																																		
► RD2[31:0]	0000000b																																		
► pc_mux_out[31:0]	00000004																																		
► pc_branch[31:0]	fffffffffe																																		
► clk	0																																		
► rst	0																																		
► zero_flag	0																																		
► branch_enable	0																																		

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.			<table border="1"> <thead> <tr> <th>Name</th><th>Value</th></tr> </thead> <tbody> <tr> <td>Instruction[31:0]</td><td>ff5faf23</td></tr> <tr> <td>program_count[31:0]</td><td>00000004</td></tr> <tr> <td>alu_res[31:0]</td><td>00000007</td></tr> <tr> <td>data_mem_out[31:0]</td><td>XXXXXXXX</td></tr> <tr> <td>memtoreg_result[31:0]</td><td>00000007</td></tr> <tr> <td>imm_out[31:0]</td><td>ffffffe</td></tr> <tr> <td>alu_in_1[31:0]</td><td>00000009</td></tr> <tr> <td>alu_in_2[31:0]</td><td>ffffffe</td></tr> <tr> <td>RD2[31:0]</td><td>00000005</td></tr> <tr> <td>pc_mux_out[31:0]</td><td>00000008</td></tr> <tr> <td>pc_branch[31:0]</td><td>00000000</td></tr> <tr> <td>clk</td><td>0</td></tr> <tr> <td>rst</td><td>0</td></tr> <tr> <td>zero_flag</td><td>0</td></tr> <tr> <td>branch_enable</td><td>0</td></tr> </tbody> </table>	Name	Value	Instruction[31:0]	ff5faf23	program_count[31:0]	00000004	alu_res[31:0]	00000007	data_mem_out[31:0]	XXXXXXXX	memtoreg_result[31:0]	00000007	imm_out[31:0]	ffffffe	alu_in_1[31:0]	00000009	alu_in_2[31:0]	ffffffe	RD2[31:0]	00000005	pc_mux_out[31:0]	00000008	pc_branch[31:0]	00000000	clk	0	rst	0	zero_flag	0	branch_enable	0			
Name	Value																																					
Instruction[31:0]	ff5faf23																																					
program_count[31:0]	00000004																																					
alu_res[31:0]	00000007																																					
data_mem_out[31:0]	XXXXXXXX																																					
memtoreg_result[31:0]	00000007																																					
imm_out[31:0]	ffffffe																																					
alu_in_1[31:0]	00000009																																					
alu_in_2[31:0]	ffffffe																																					
RD2[31:0]	00000005																																					
pc_mux_out[31:0]	00000008																																					
pc_branch[31:0]	00000000																																					
clk	0																																					
rst	0																																					
zero_flag	0																																					
branch_enable	0																																					
The calculated effective address of the ALU is stored with 0x5.																																						

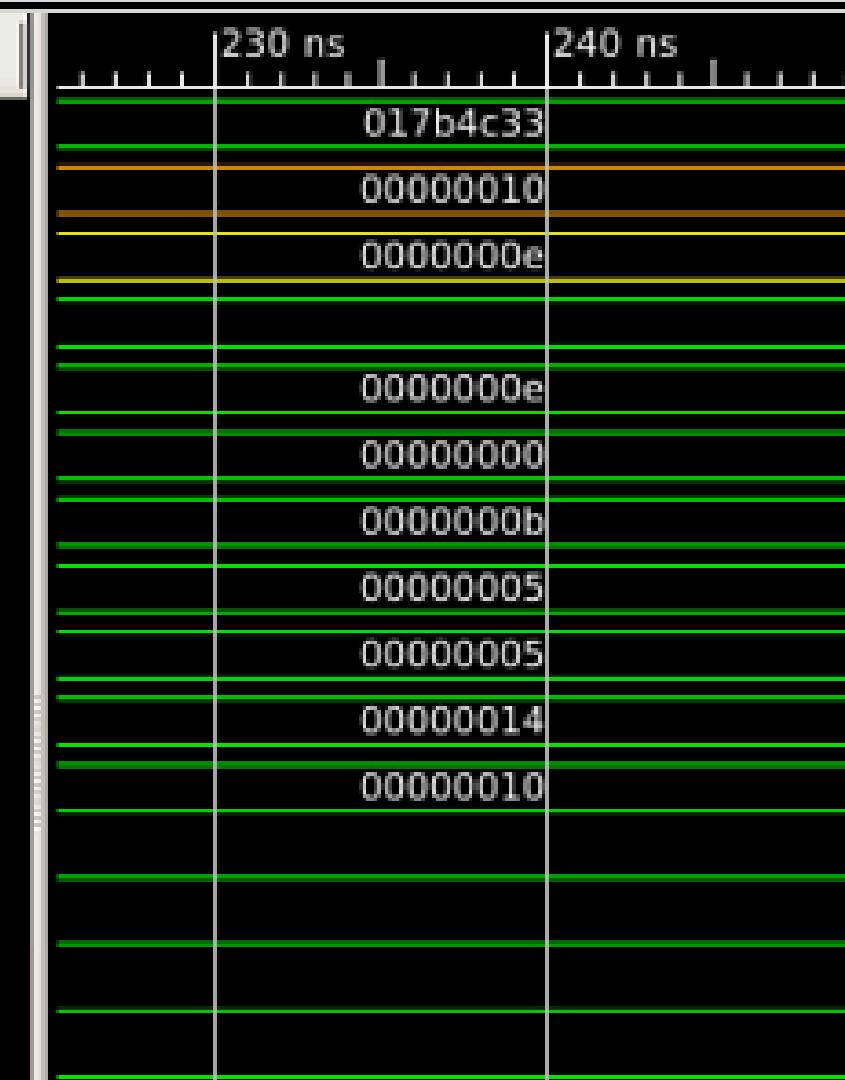
Program Output

Count	Abstract Code	Instruction	Interpretation																																
0x0008	lw x22, -1(x30)	FF FF 2B 03	@Register File: (x22 = 0xB)																																
<p>The count now is 0x8 with instruction code FFFF2B03.</p> <p>The value of the calculated effective address (0x9) in the data memory is loaded to the register file (x22 = 0xB).</p>			<table border="1"> <thead> <tr> <th>Name</th><th>Value</th></tr> </thead> <tbody> <tr> <td>Instruction[31:0]</td><td>ffff2b03</td></tr> <tr> <td>program_count[31:0]</td><td>00000008</td></tr> <tr> <td>alu_res[31:0]</td><td>00000009</td></tr> <tr> <td>data_mem_out[31:0]</td><td>0000000b</td></tr> <tr> <td>memtoreg_result[31:0]</td><td>0000000b</td></tr> <tr> <td>imm_out[31:0]</td><td>ffffffff</td></tr> <tr> <td>alu_in_1[31:0]</td><td>0000000a</td></tr> <tr> <td>alu_in_2[31:0]</td><td>ffffffff</td></tr> <tr> <td>RD2[31:0]</td><td>00000009</td></tr> <tr> <td>pc_mux_out[31:0]</td><td>0000000c</td></tr> <tr> <td>pc_branch[31:0]</td><td>00000006</td></tr> <tr> <td>clk</td><td>0</td></tr> <tr> <td>rst</td><td>0</td></tr> <tr> <td>zero_flag</td><td>0</td></tr> <tr> <td>branch_enable</td><td>0</td></tr> </tbody> </table>	Name	Value	Instruction[31:0]	ffff2b03	program_count[31:0]	00000008	alu_res[31:0]	00000009	data_mem_out[31:0]	0000000b	memtoreg_result[31:0]	0000000b	imm_out[31:0]	ffffffff	alu_in_1[31:0]	0000000a	alu_in_2[31:0]	ffffffff	RD2[31:0]	00000009	pc_mux_out[31:0]	0000000c	pc_branch[31:0]	00000006	clk	0	rst	0	zero_flag	0	branch_enable	0
Name	Value																																		
Instruction[31:0]	ffff2b03																																		
program_count[31:0]	00000008																																		
alu_res[31:0]	00000009																																		
data_mem_out[31:0]	0000000b																																		
memtoreg_result[31:0]	0000000b																																		
imm_out[31:0]	ffffffff																																		
alu_in_1[31:0]	0000000a																																		
alu_in_2[31:0]	ffffffff																																		
RD2[31:0]	00000009																																		
pc_mux_out[31:0]	0000000c																																		
pc_branch[31:0]	00000006																																		
clk	0																																		
rst	0																																		
zero_flag	0																																		
branch_enable	0																																		

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.			<table border="1"> <thead> <tr> <th>Name</th><th>Value</th></tr> </thead> <tbody> <tr> <td>Instruction[31:0]</td><td>ffefab83</td></tr> <tr> <td>program_count[31:0]</td><td>0000000c</td></tr> <tr> <td>alu_res[31:0]</td><td>00000007</td></tr> <tr> <td>data_mem_out[31:0]</td><td>00000005</td></tr> <tr> <td>memtoreg_result[31:0]</td><td>00000005</td></tr> <tr> <td>imm_out[31:0]</td><td>ffffffe</td></tr> <tr> <td>alu_in_1[31:0]</td><td>00000009</td></tr> <tr> <td>alu_in_2[31:0]</td><td>ffffffe</td></tr> <tr> <td>RD2[31:0]</td><td>0000000a</td></tr> <tr> <td>pc_mux_out[31:0]</td><td>00000010</td></tr> <tr> <td>pc_branch[31:0]</td><td>00000008</td></tr> <tr> <td>clk</td><td>0</td></tr> <tr> <td>rst</td><td>0</td></tr> <tr> <td>zero_flag</td><td>0</td></tr> <tr> <td>branch_enable</td><td>0</td></tr> </tbody> </table>	Name	Value	Instruction[31:0]	ffefab83	program_count[31:0]	0000000c	alu_res[31:0]	00000007	data_mem_out[31:0]	00000005	memtoreg_result[31:0]	00000005	imm_out[31:0]	ffffffe	alu_in_1[31:0]	00000009	alu_in_2[31:0]	ffffffe	RD2[31:0]	0000000a	pc_mux_out[31:0]	00000010	pc_branch[31:0]	00000008	clk	0	rst	0	zero_flag	0	branch_enable	0
Name	Value																																		
Instruction[31:0]	ffefab83																																		
program_count[31:0]	0000000c																																		
alu_res[31:0]	00000007																																		
data_mem_out[31:0]	00000005																																		
memtoreg_result[31:0]	00000005																																		
imm_out[31:0]	ffffffe																																		
alu_in_1[31:0]	00000009																																		
alu_in_2[31:0]	ffffffe																																		
RD2[31:0]	0000000a																																		
pc_mux_out[31:0]	00000010																																		
pc_branch[31:0]	00000008																																		
clk	0																																		
rst	0																																		
zero_flag	0																																		
branch_enable	0																																		
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	$1011_2 \text{ xor } 0101_2 = 1110_2 = 0xE$																																
The count now is 0x10 with instruction code 017B4C33.			<table border="1"> <thead> <tr> <th>Name</th><th>Value</th></tr> </thead> <tbody> <tr> <td>Instruction[31:0]</td><td>017b4c33</td></tr> <tr> <td>program_count[31:0]</td><td>00000010</td></tr> <tr> <td>alu_res[31:0]</td><td>0000000e</td></tr> <tr> <td>data_mem_out[31:0]</td><td>00000005</td></tr> <tr> <td>memtoreg_result[31:0]</td><td>0000000e</td></tr> <tr> <td>imm_out[31:0]</td><td>00000000</td></tr> <tr> <td>alu_in_1[31:0]</td><td>0000000b</td></tr> <tr> <td>alu_in_2[31:0]</td><td>00000005</td></tr> <tr> <td>RD2[31:0]</td><td>00000005</td></tr> <tr> <td>pc_mux_out[31:0]</td><td>00000014</td></tr> <tr> <td>pc_branch[31:0]</td><td>00000010</td></tr> <tr> <td>clk</td><td>0</td></tr> <tr> <td>rst</td><td>0</td></tr> <tr> <td>zero_flag</td><td>0</td></tr> <tr> <td>branch_enable</td><td>0</td></tr> </tbody> </table>	Name	Value	Instruction[31:0]	017b4c33	program_count[31:0]	00000010	alu_res[31:0]	0000000e	data_mem_out[31:0]	00000005	memtoreg_result[31:0]	0000000e	imm_out[31:0]	00000000	alu_in_1[31:0]	0000000b	alu_in_2[31:0]	00000005	RD2[31:0]	00000005	pc_mux_out[31:0]	00000014	pc_branch[31:0]	00000010	clk	0	rst	0	zero_flag	0	branch_enable	0
Name	Value																																		
Instruction[31:0]	017b4c33																																		
program_count[31:0]	00000010																																		
alu_res[31:0]	0000000e																																		
data_mem_out[31:0]	00000005																																		
memtoreg_result[31:0]	0000000e																																		
imm_out[31:0]	00000000																																		
alu_in_1[31:0]	0000000b																																		
alu_in_2[31:0]	00000005																																		
RD2[31:0]	00000005																																		
pc_mux_out[31:0]	00000014																																		
pc_branch[31:0]	00000010																																		
clk	0																																		
rst	0																																		
zero_flag	0																																		
branch_enable	0																																		
xor x24, x22, x23 $0xE = 0xB \text{ xor } 0x5$																																			

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.			<table border="1"> <thead> <tr> <th>Name</th><th>Value</th></tr> </thead> <tbody> <tr> <td>Instruction[31:0]</td><td>001c0863</td></tr> <tr> <td>program_count[31:0]</td><td>00000014</td></tr> <tr> <td>alu_res[31:0]</td><td>00000000</td></tr> <tr> <td>data_mem_out[31:0]</td><td>00000005</td></tr> <tr> <td>memtoreg_result[31:0]</td><td>00000000</td></tr> <tr> <td>imm_out[31:0]</td><td>00000008</td></tr> <tr> <td>alu_in_1[31:0]</td><td>0000000e</td></tr> <tr> <td>alu_in_2[31:0]</td><td>0000000e</td></tr> <tr> <td>RD2[31:0]</td><td>0000000e</td></tr> <tr> <td>pc_mux_out[31:0]</td><td>00000024</td></tr> <tr> <td>pc_branch[31:0]</td><td>00000024</td></tr> <tr> <td>clk</td><td>0</td></tr> <tr> <td>rst</td><td>0</td></tr> <tr> <td>zero_flag</td><td>1</td></tr> <tr> <td>branch_enable</td><td>1</td></tr> </tbody> </table>	Name	Value	Instruction[31:0]	001c0863	program_count[31:0]	00000014	alu_res[31:0]	00000000	data_mem_out[31:0]	00000005	memtoreg_result[31:0]	00000000	imm_out[31:0]	00000008	alu_in_1[31:0]	0000000e	alu_in_2[31:0]	0000000e	RD2[31:0]	0000000e	pc_mux_out[31:0]	00000024	pc_branch[31:0]	00000024	clk	0	rst	0	zero_flag	1	branch_enable	1
Name	Value																																		
Instruction[31:0]	001c0863																																		
program_count[31:0]	00000014																																		
alu_res[31:0]	00000000																																		
data_mem_out[31:0]	00000005																																		
memtoreg_result[31:0]	00000000																																		
imm_out[31:0]	00000008																																		
alu_in_1[31:0]	0000000e																																		
alu_in_2[31:0]	0000000e																																		
RD2[31:0]	0000000e																																		
pc_mux_out[31:0]	00000024																																		
pc_branch[31:0]	00000024																																		
clk	0																																		
rst	0																																		
zero_flag	1																																		
branch_enable	1																																		
<p>The Branch is enabled. Program Counter Multiplexer outputs 0x24 as the next Program Count</p> <p>BTA = 0x0014 + (8₁₀ * 2) = 0x0024</p>																																			

Program Output

Count	Abstract Code	Instruction	Interpretation																																
0x0024	addi x25, x24, -2	FF EC 0C 93	$0xE + (-2_{10}) = 0xC$																																
<p>The count now is 0x24 after branching with instruction code FFEC0C93</p> <p>The x25 of register file is stored with the addition of stored value in x24 and the immediate -2.</p> <p>$0xE + (-2_{10}) = 0xC$</p>			<table border="1"> <thead> <tr> <th>Name</th><th>Value</th></tr> </thead> <tbody> <tr> <td>Instruction[31:0]</td><td>ffec0c93</td></tr> <tr> <td>program_count[31:0]</td><td>00000024</td></tr> <tr> <td>alu_res[31:0]</td><td>0000000c</td></tr> <tr> <td>data_mem_out[31:0]</td><td>00000005</td></tr> <tr> <td>memtoreg_result[31:0]</td><td>0000000c</td></tr> <tr> <td>imm_out[31:0]</td><td>ffffffe</td></tr> <tr> <td>alu_in_1[31:0]</td><td>0000000e</td></tr> <tr> <td>alu_in_2[31:0]</td><td>ffffffe</td></tr> <tr> <td>RD2[31:0]</td><td>0000000a</td></tr> <tr> <td>pc_mux_out[31:0]</td><td>00000028</td></tr> <tr> <td>pc_branch[31:0]</td><td>00000020</td></tr> <tr> <td>clk</td><td>0</td></tr> <tr> <td>rst</td><td>0</td></tr> <tr> <td>zero_flag</td><td>0</td></tr> <tr> <td>branch_enable</td><td>0</td></tr> </tbody> </table>	Name	Value	Instruction[31:0]	ffec0c93	program_count[31:0]	00000024	alu_res[31:0]	0000000c	data_mem_out[31:0]	00000005	memtoreg_result[31:0]	0000000c	imm_out[31:0]	ffffffe	alu_in_1[31:0]	0000000e	alu_in_2[31:0]	ffffffe	RD2[31:0]	0000000a	pc_mux_out[31:0]	00000028	pc_branch[31:0]	00000020	clk	0	rst	0	zero_flag	0	branch_enable	0
Name	Value																																		
Instruction[31:0]	ffec0c93																																		
program_count[31:0]	00000024																																		
alu_res[31:0]	0000000c																																		
data_mem_out[31:0]	00000005																																		
memtoreg_result[31:0]	0000000c																																		
imm_out[31:0]	ffffffe																																		
alu_in_1[31:0]	0000000e																																		
alu_in_2[31:0]	ffffffe																																		
RD2[31:0]	0000000a																																		
pc_mux_out[31:0]	00000028																																		
pc_branch[31:0]	00000020																																		
clk	0																																		
rst	0																																		
zero_flag	0																																		
branch_enable	0																																		



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

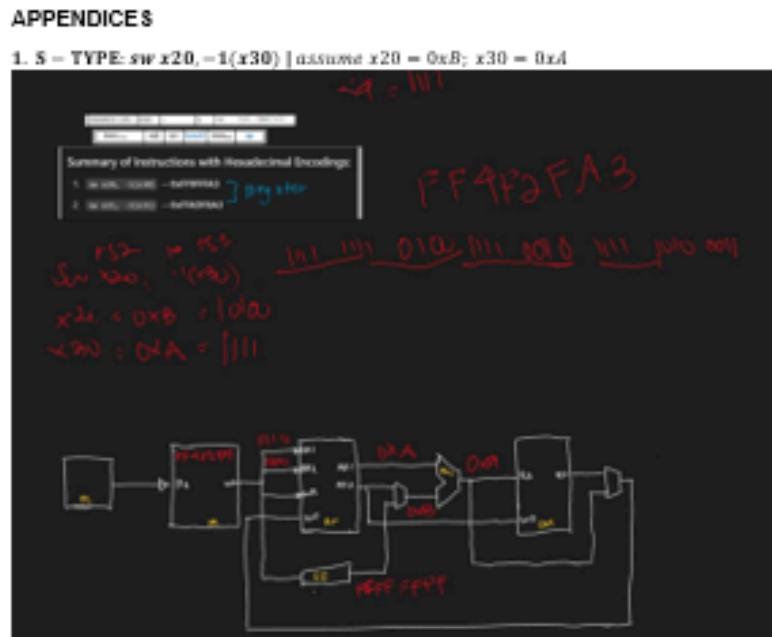


Readme
20 bytes

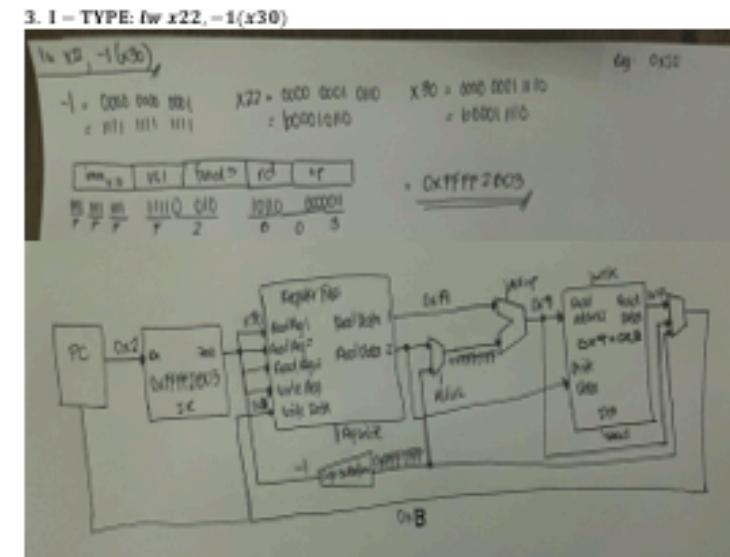
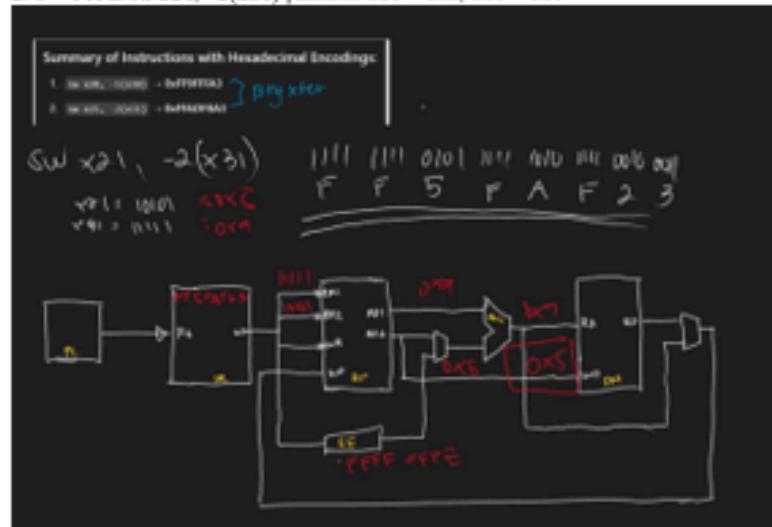


Appendices

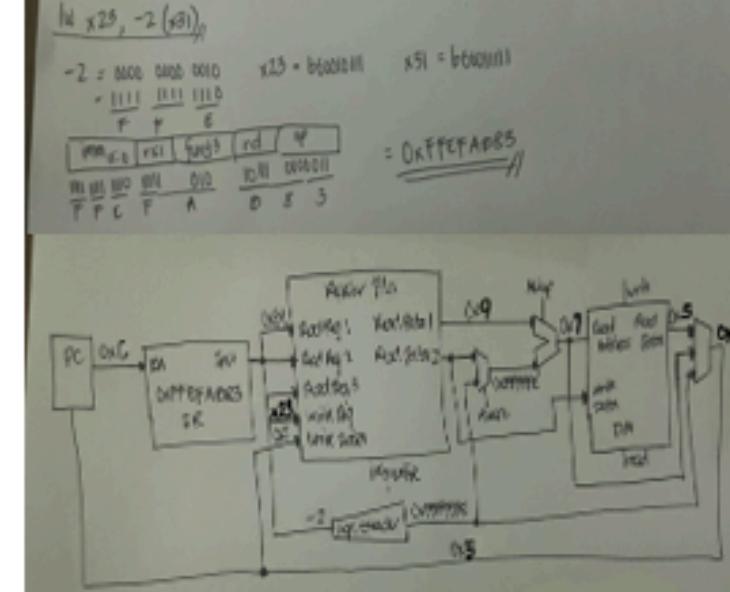
3



2. S = TYPE: sw x21,-2(x31) | assume x21 = 0x5; x31 = 0



4. I - TYPE: fw x23,-2/x31



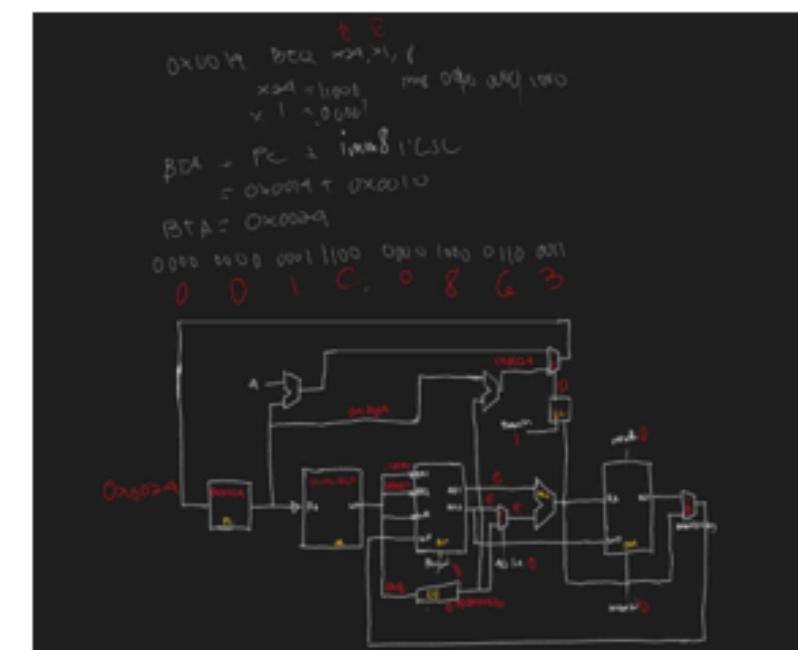
S₁ S₂ S₃ S₄
 R₁ R₂ R₃ R₄
 11000 11111 11111 11111
 10000 11111 11111 11111
 00000 10000 10000 10000
 0 1 7 B 4 C 3 3

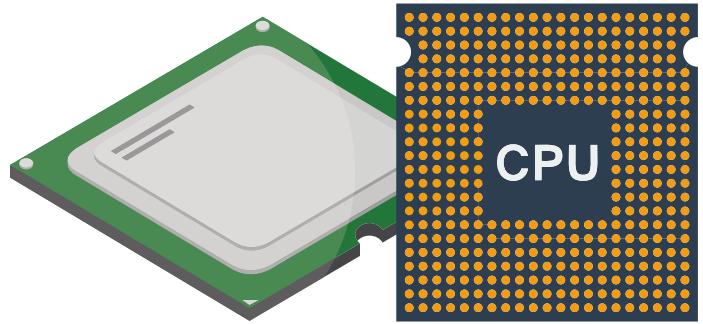
01114C33

Initial State
1
0
1
1
1
0
1
1
1
0
1
1
1
0
1
1
1
0

outputs:
 00 = red
 01 = green
 11 = yellow
 10 = white

5. B = TYPE: *beg x24, x1, 8*





SAP (SIMPLE-AS-POSSIBLE)

Computer: Design and Development

Thank you for Listening!