

Building a Simple CPU

VSDOpen2020



Steve Hoover

Founder, Redwood EDA

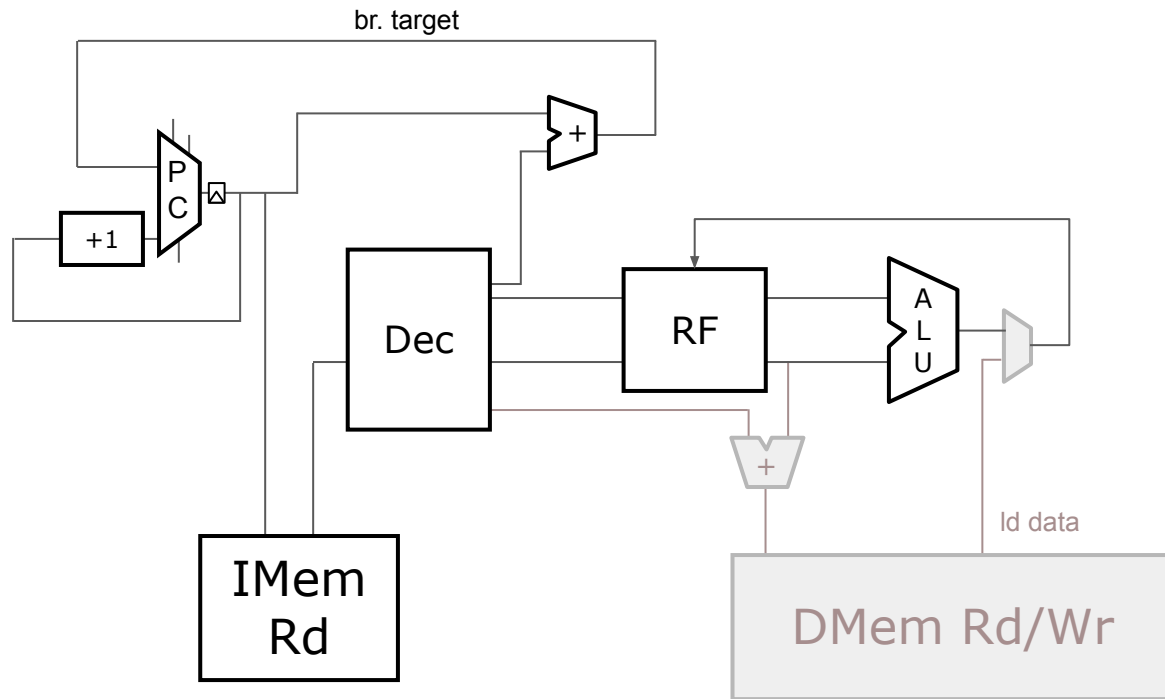
Oct. 8, 2020



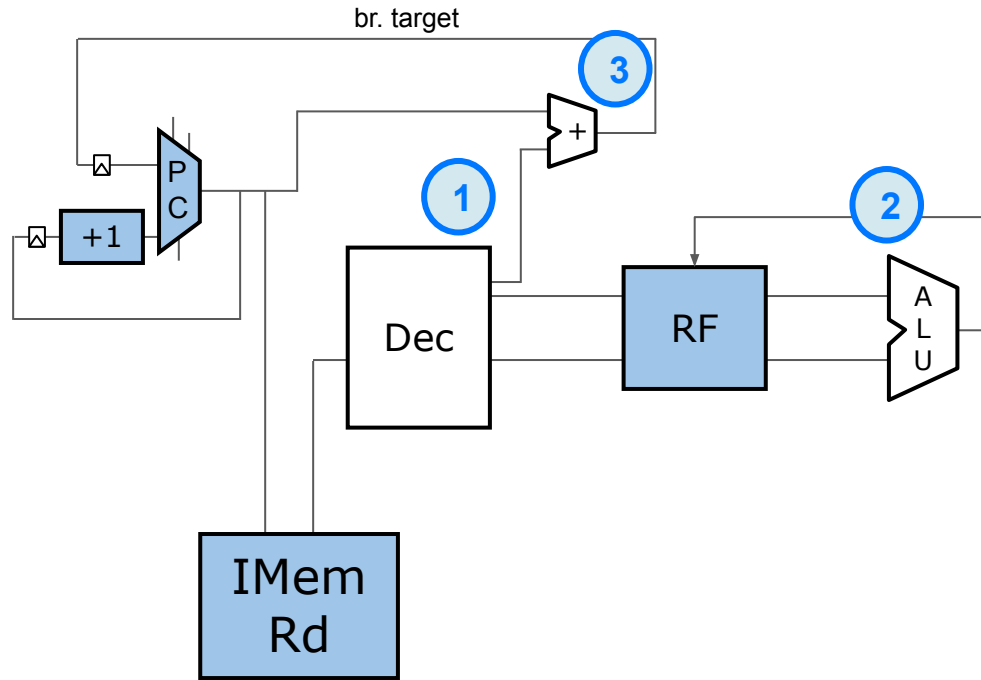
Agenda

- RISC-V
- Makerchip.com
- Combinational logic in TL-Verilog
- Building Your Own Simple RISC-V-Subset Core

Simple CPU Block Diagram



Implementation Plan



Lab: Setup

Github repo can be ignored.
Starting-point code should already be loaded.

- Visit:
github.com/stevehoover/VSDOpen2020_TLV_RISC-V_Tutorial
- Open starting-point code.
- Click “Save as new project” (upper right), and bookmark the new URL.
- In the remaining labs, you will:
 - complete “TBD” logic expressions
 - fix errors in LOG (it will report “Simulation FAILED!!!” until the final lab is completed successfully)
 - examine DIAGRAM, VIZ, and WAVEFORM to ensure correct operation and debug

BTW, the logic inside the yellow “|for_viz_only” box in the DIAGRAM provides the VIZ functionality. It is not part of the hardware model and can be ignored.

Verilog/TL-Verilog Operators Reference

Try other operators until you feel comfortable.

Operation	Boolean	Bitwise
NOT	!	~
AND	&&	&
OR		
XOR	^	^

Use () to group sub-expressions.

Operation	Operator
Arithmetic	+, -, *, /
Bit Shift	<<, >>
Compare	==, !=
Bit extract	<i>sig</i> [<i>max-bit</i> : <i>min-bit</i>]
Concatenate	{ <i>sig1</i> , <i>sig2</i> }
Replicate	{ <i>count</i> { <i>sig</i> }

Constants

16'b10

16-bit binary value

For example:

```
$out[4:0] = $in1 + 5'b01001;
```

Binary
(base 2)

0: 0000
1: 0001
2: 0010
3: 0011
4: 0100
5: 0101
6: 0110
...

Decimal
(base 10)

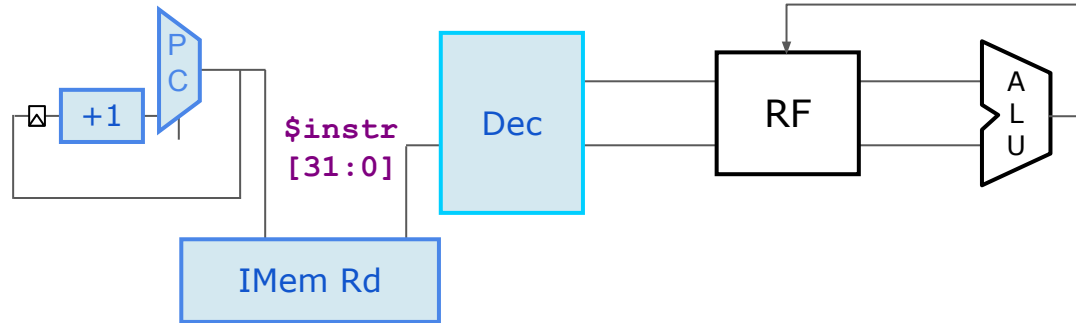
0: 00
1: 01
...
9: 09
10: 10
...
...

Hexadecimal
(base 16)

0: 00
1: 01
...
9: 09
10: 0A
...
15: 0F
16: 10
...

Decode

Be sure to click the button with the videos to load the starting code.



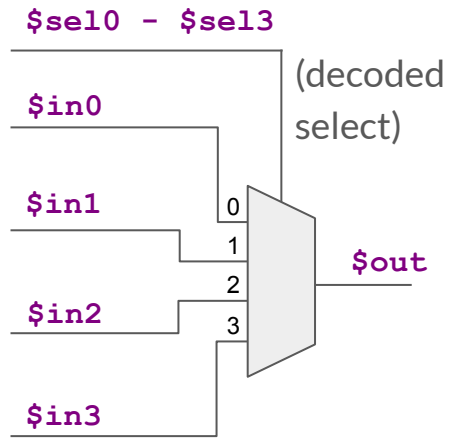
Lab: Instruction Types Decode

In RV32I, `instr[6:2]` determine instruction type: I, R, S, B, J, U.
For our RISC-V subset, `instr[6:5]` determines instruction type.

	<code>instr[4:2]</code> <code>instr[6:5]</code>	000	001	010	011	100	101	110	111
I	00	I	I	-	-	I	U	I	-
R	01	S	S	-	R	R	U	R	-
R	10	R4	R4	R4	R4	R	-	-	-
B	11	B	I	-	J	I	-	-	-

```
$is_i_instr = $instr[6:5] == 2'b00;  
$is_r_instr = TBD;  
$is_b_instr = TBD;
```

Multiplexers



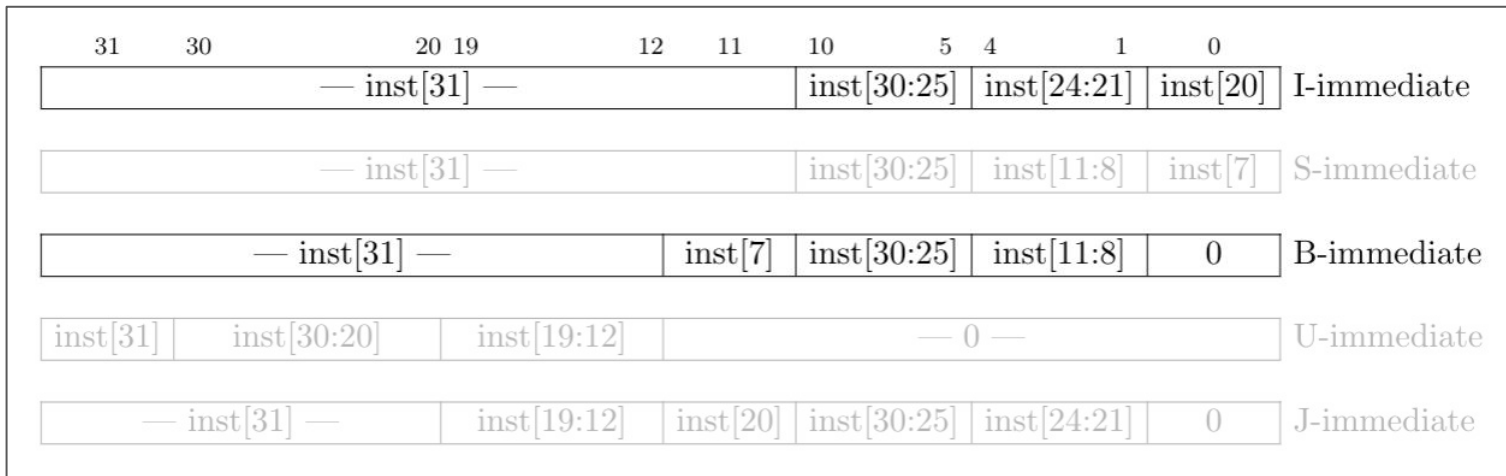
TL-Verilog:

```
$out =  
    $sel0 ? $in0 :  
    $sel1 ? $in1 :  
    $sel2 ? $in2 :  
    $in3 ;
```

(highest priority first.)

Lab: Instruction Immediate Decode

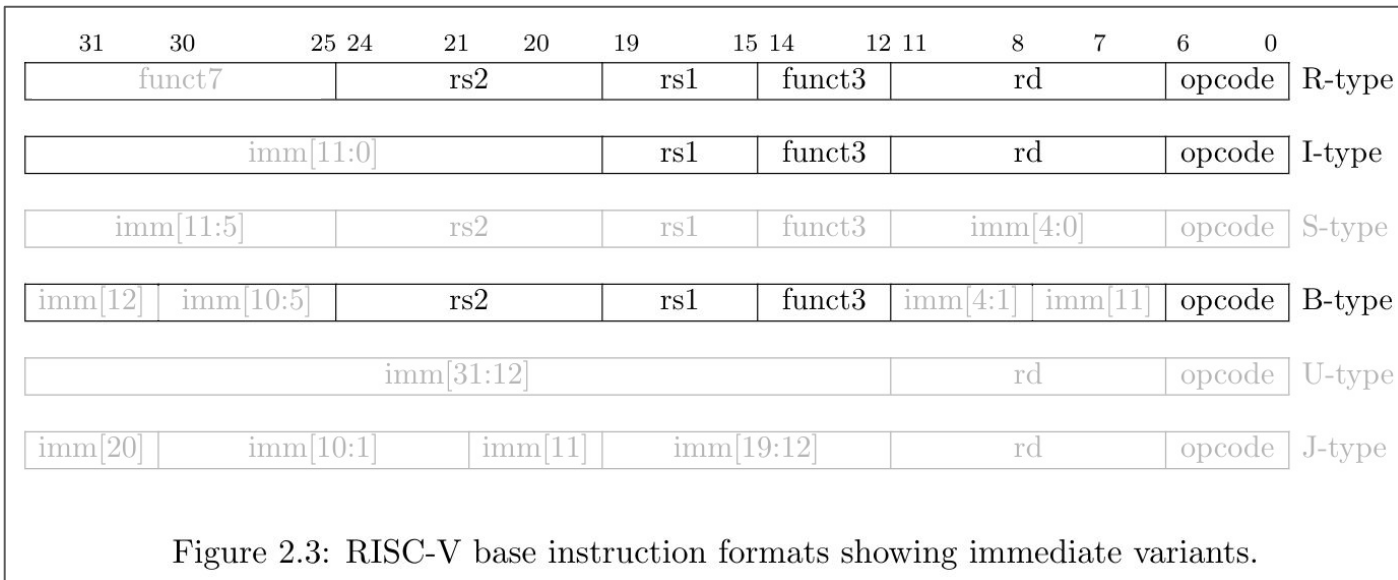
Form `$imm[31:0]` based on instruction type. (R-type has no immediate.)



```
$imm[31:0] = $is_i_instr ? { {21{$instr[31]}}, $instr[30:20] } : // I-type
                //TBD                                         // B-type
                32'b0;
```

Lab: Instruction Field Decode

Extract other instruction fields: $\$rs2$, $\$rs1$, $\$funct3$, $\$rd$, $\$opcode$ (funct7 is not needed)

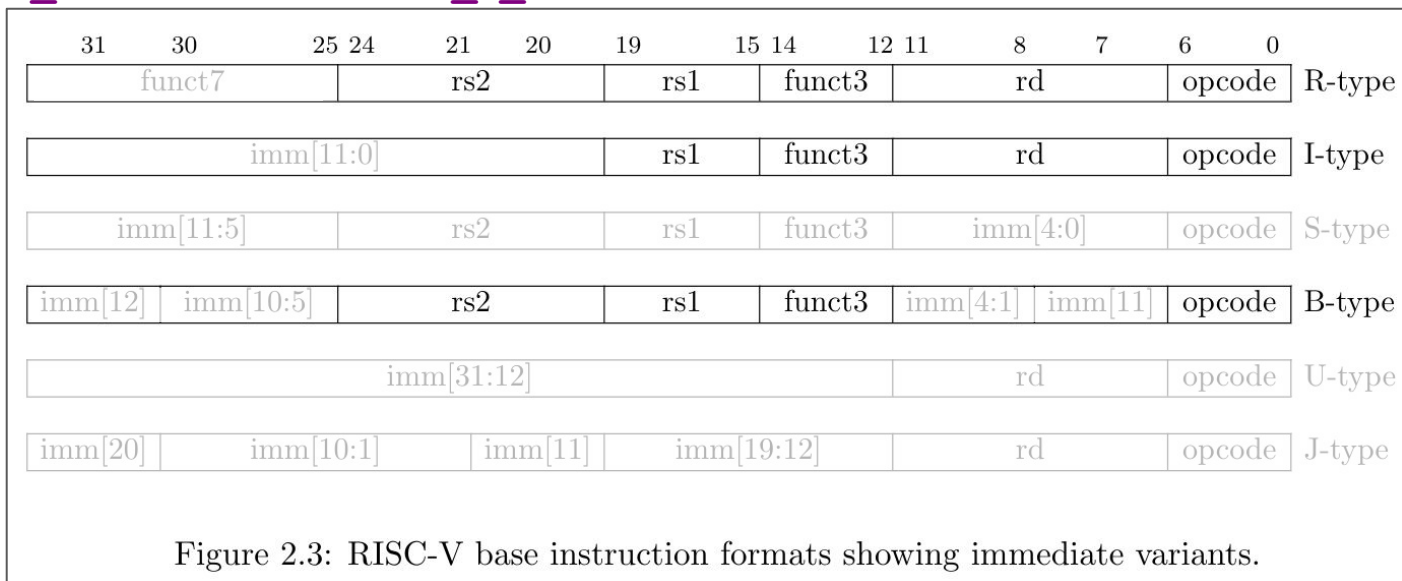


```
 $\$rs2[4:0] = \$instr[24:20];$ 
```

```
...
```

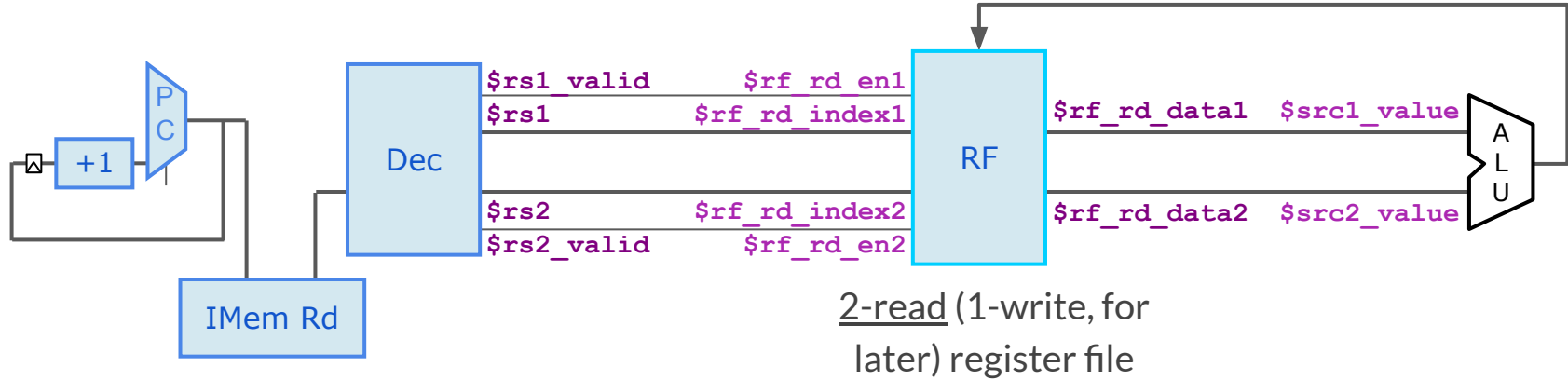
Lab: Register Validity Decode

Determine when register fields are valid: `$rs2_valid`, `$rs1_valid`, `$rd_valid` based on `$is_x_instr`.



```
$rs1_valid = $is_r_instr || $is_i_instr || $is_b_instr;  
...
```

Register File Read Hookup



Lab: Instruction Decode

RV32I Base Instruction Set (except FENCE, ECALL, EBREAK):

opcode		
01101111		LUI
00101111		AUIPC
11011111		JAL
11001111		JALR
000	11000111	BEQ
000	11000111	BEQ
001	11000111	RNE
100	11000111	BLT
101	11000111	BGE
110	11000111	BLTU
111	11000111	BGEU
000	00000111	LB
001	00000111	LH
010	00000111	LW
100	00000111	LBU
101	00000111	LHU
000	01000111	SB
001	01000111	SH
010	01000111	SW
000	00100111	ADDI
010	00100111	SLTI

funct3	opcode	
011	00100111	SLTIU
100	00100111	XORI
110	00100111	ORI
111	00100111	ANDI
001	00100111	SLLI
101	00100111	SRLI
101	00100111	SRAI
000	01100111	ADD
000	01100111	SUB
001	01100111	SLL
010	01100111	SLT
011	01100111	SLTU
100	01100111	XOR
101	01100111	SRL
101	01100111	SRA
110	01100111	OR
111	01100111	AND

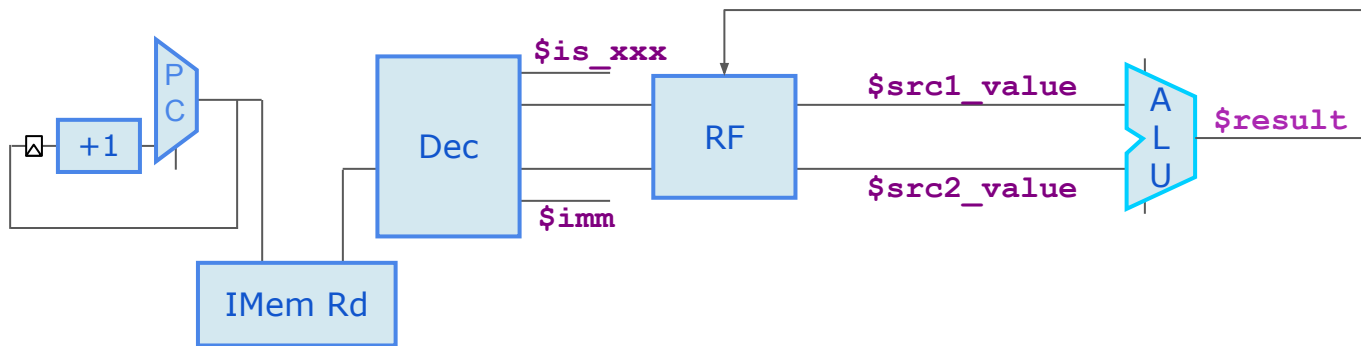
Complete circled instructions.

```
$dec_bits[9:0] = {$funct3, $opcode};  
$is_blt = $dec_bits ==  
           10'b1001100011;  
$is_addi = TBD;  
$is_add  = TBD;
```

Check VIZ and debug decode logic as necessary.

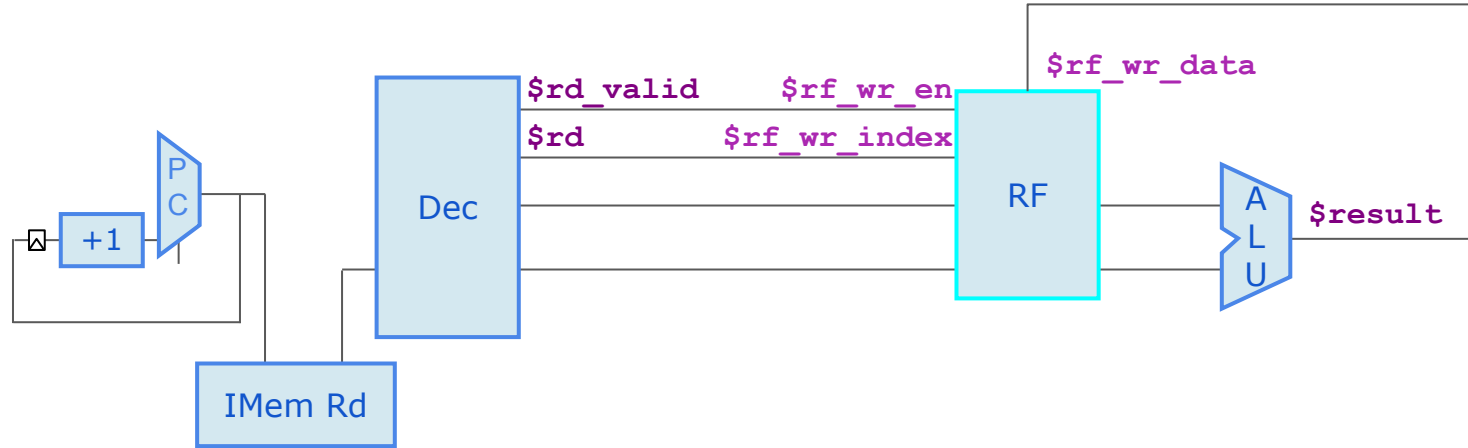
Lab: ALU

Assign the ALU `$result` for ADD and ADDI. (BLT has no result; default to 0.)

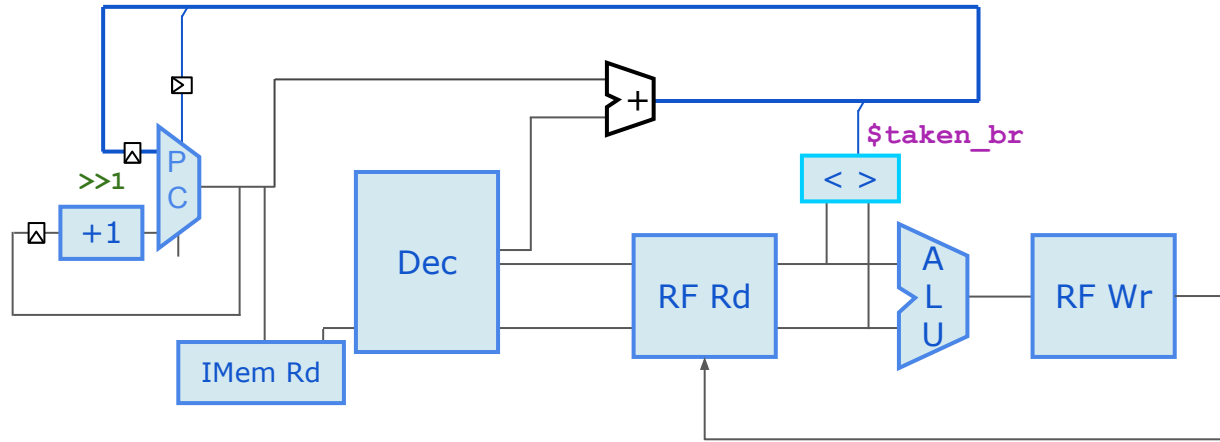


```
$result[31:0] =  
    $is_addi ? $src1_value + $imm :  
    //TBD  
    32'b0;
```

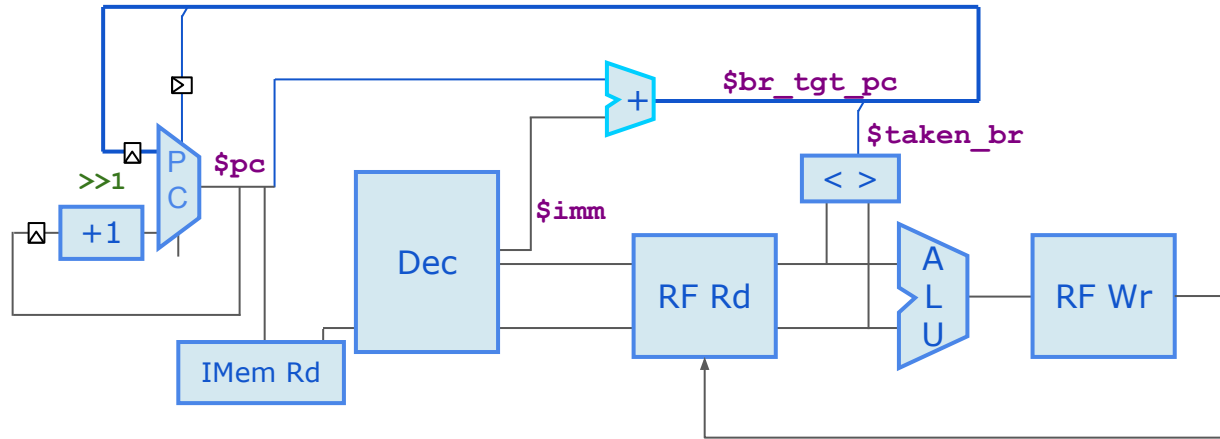

Register File Write Hookup



Lab: Branch Condition



1. Assert $\$taken_br$ for BLT instructions ($\$is_blt$) with $\$src1_value < \$src2_value$.
2. Confirm save.



1. Compute `$br_tgt_pc` (PC + imm)
2. Check behavior in simulation. Program should now branch properly, and should sum values {1..9}. If all is working, it will report "Simulation Passed" in LOG!
3. Debug as needed, and save outside of Makerchip.



YOU DID IT!!!!!!

Technologies/Skills You've Discovered

- Digital logic design
- RISC-V and CPU microarchitecture
- TL-Verilog
- Makerchip

Funda-
mental
Skills!

Latest
Technology!

Crazy-
Hot
Topic!

Skills that will put you ahead of the curve!!!



What's Next

- Brag about your accomplishments on social media!
- Capture your work in GitHub -- it's the new resume. Make a nice README.
- Participate in the next MYTH Workshop.
- Learn more about RISC-V at riscv.org.
- Learn more TL-Verilog using the tutorials in Makerchip.
 - Pipelines
 - State
 - Validity
 - Hierarchy
 - Transactions