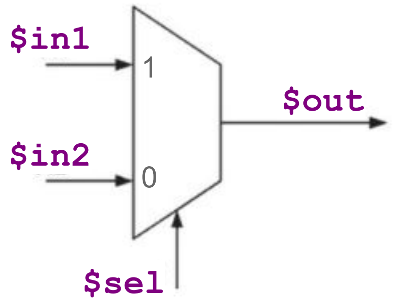
Building a RISC-V Core

RISC-V Based MYTH Workshop
MYTH - Microprocessor for You in Thirty Hours

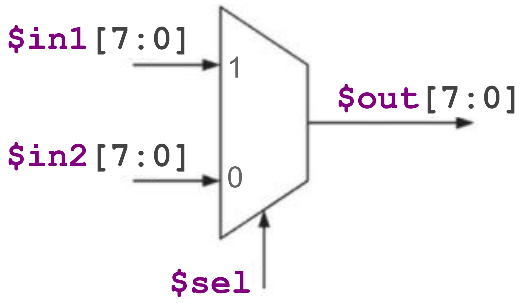
Steve Hoover
Founder, Redwood EDA
July 31, 2020

Lab: Mux

`$out = $sel ? $in1 : $in2`
creates a multiplexer.



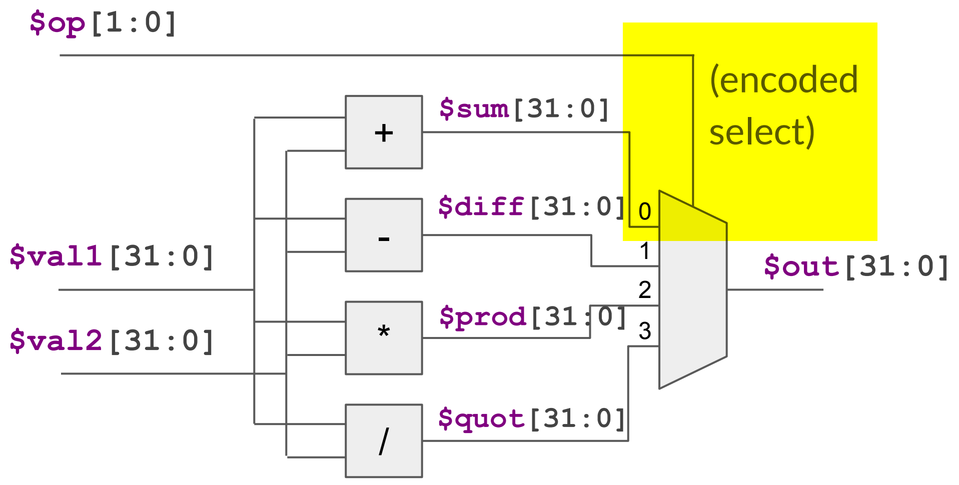
Modify this multiplexer to operate on vectors.



Note that bit ranges can generally be assumed on the left-hand side, but with no assignments to these signals, they must be explicit.

Lab: Combinational Calculator

This circuit implements a calculator that can perform +, -, *, / on two input values.



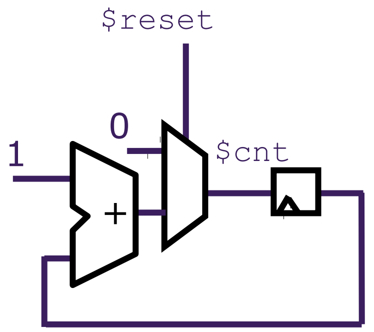
1. Implement this.
2. Use:

```
$val1[31:0] = $rand1[3:0];
$val2[31:0] = $rand2[3:0];
```

   for inputs to keep values small.
3. We'll return to this, so "Save as new project", bookmark, and open a new Makerchip IDE in a new tab.

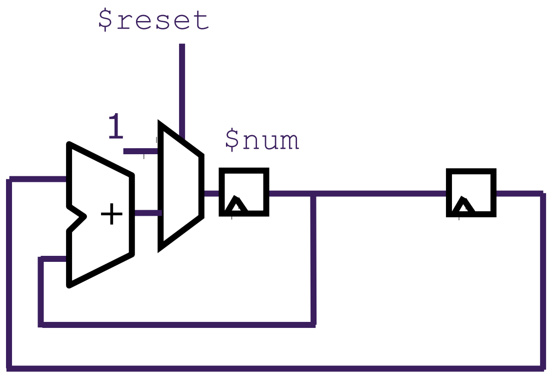
Lab: Counter

1. Design a free-running counter:



2. Include this code in your saved calculator sandbox for later (and confirm that it auto-saves).

Reference Example: Fibonacci Sequence (1, 1, 2, 3, 5, 8, ...)



```
\TLV
   $num[31:0] = $reset ? 1 : (>>1$num + >>2$num);
```

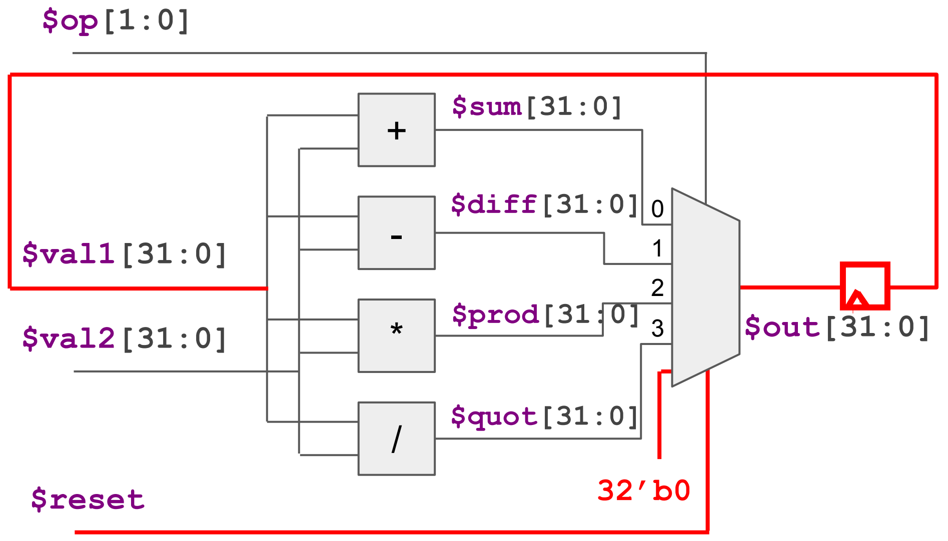
3-space indentation (no tabs)

(makerchip.com/sandbox/0/0wjhLP)

Lab: Sequential Calculator

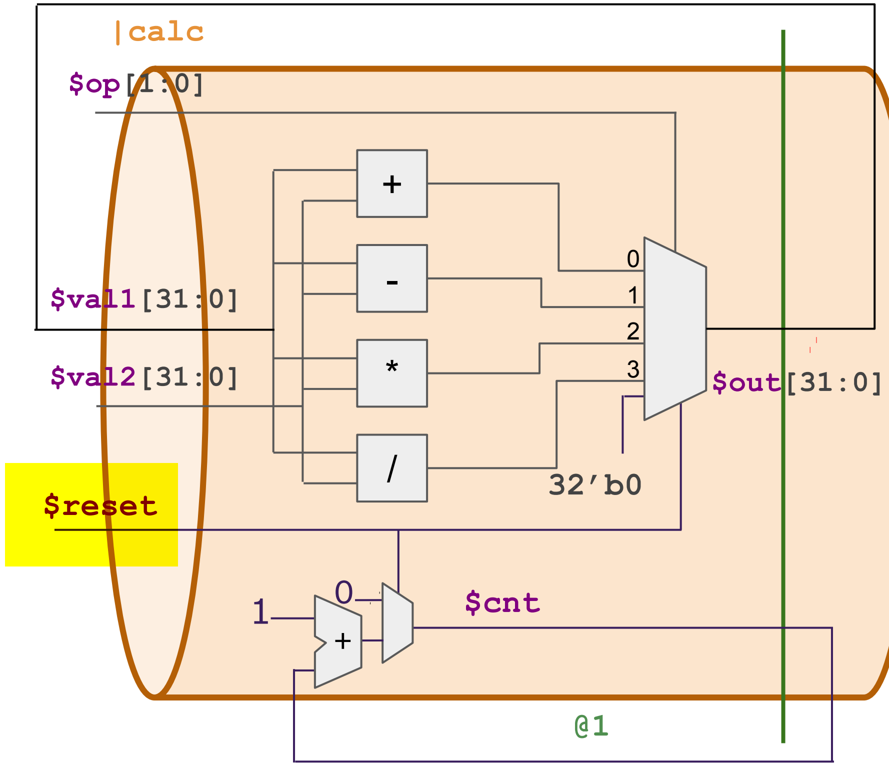
A real calculator remembers the last result, and uses it for the next calculation.

1. Return to the calculator.
2. Update the calculator to perform a new calculation each cycle where `$val1[31:0]` = the previous calculation.
3. Reset $out to zero.
4. Copy code and save outside of Makerchip (just to be safe).



Lab: Counter and Calculator in Pipeline

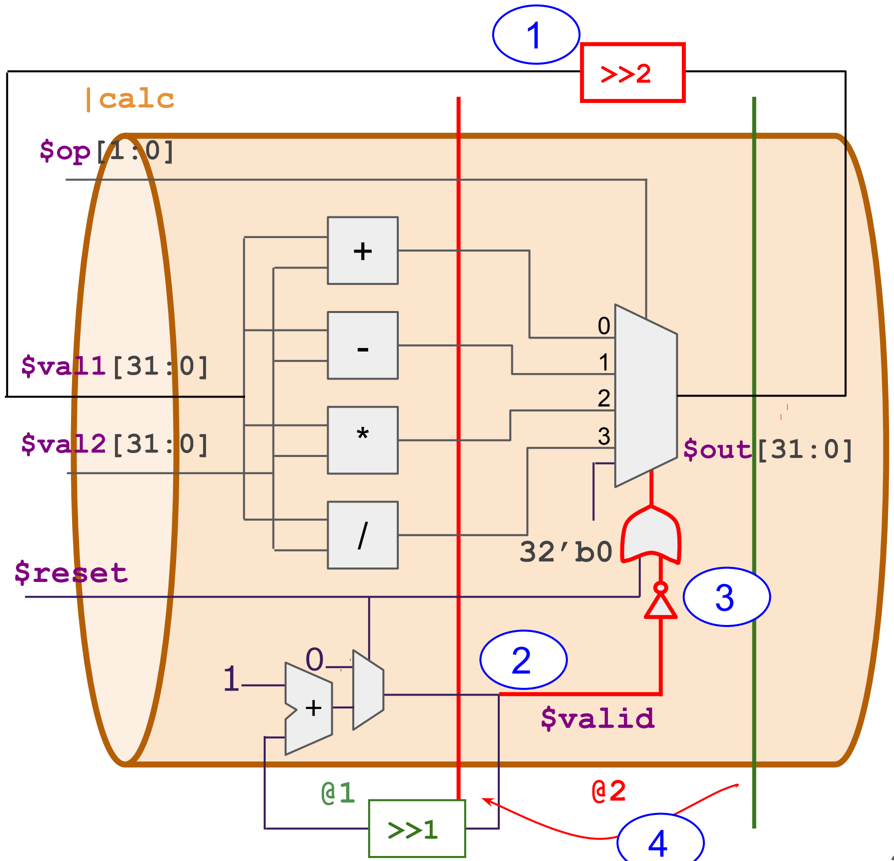
1. Put calculator and counter in stage `@1` of a `|calc` pipeline.
2. Check log, diagram, and waveform.
3. Confirm save.



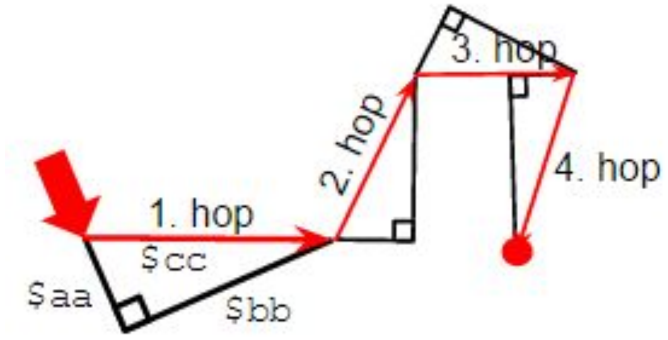
Lab: 2-Cycle Calculator

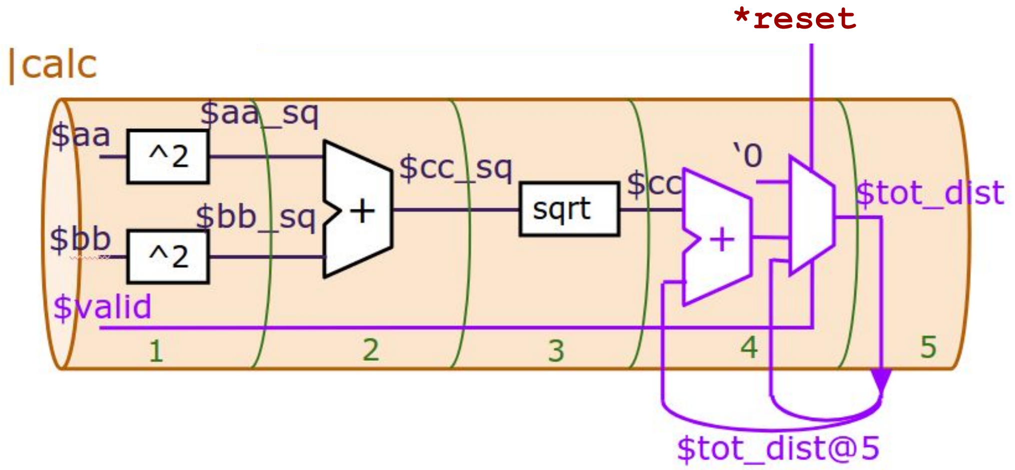
At high frequency, we might need to calculate every other cycle.

1. Change alignment of `$out` (to calculate every other cycle).
2. Change counter to single-bit (to indicate every other cycle).
3. Connect `$valid` (to clear alternate outputs).
4. Retime mux to `@2` (to ease timing; no functional change).
5. Verify behavior in waveform.
6. Save.



Total Distance (Makerchip walkthrough)





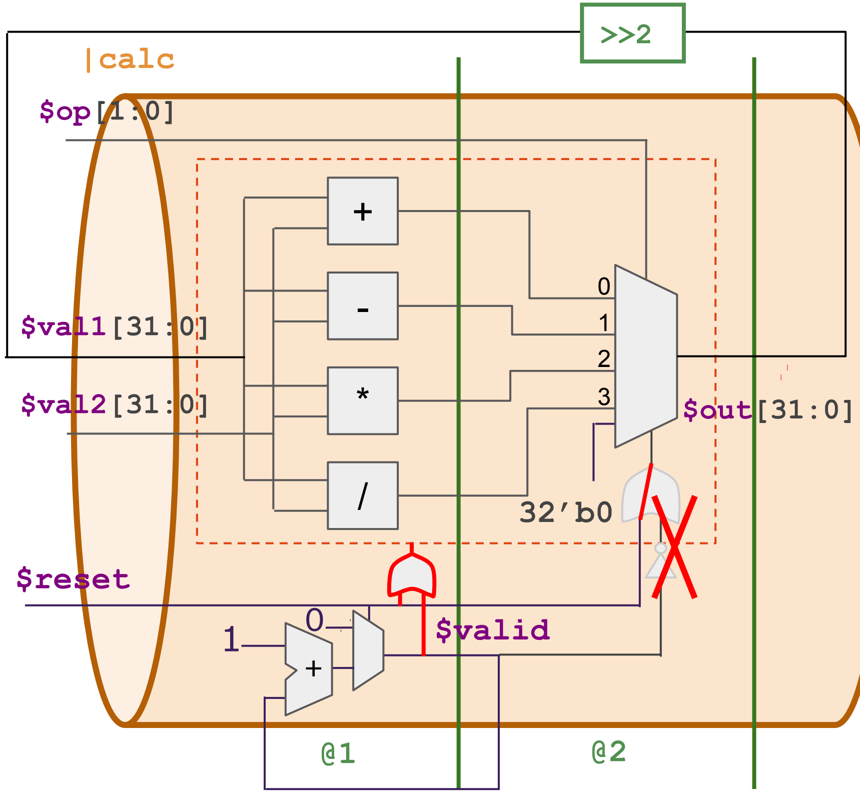
Lab: 2-Cycle Calculator with Validity

1. Use:
   `$valid_or_reset = $valid || $reset;`
   as a when condition for calculation instead of zeroing `$out`.

   For reference:

```
|calc
   @1
      $valid = ...;
   ?$valid
      @1
         $aa_sq[31:0] = $aa * $aa;
         ...
```

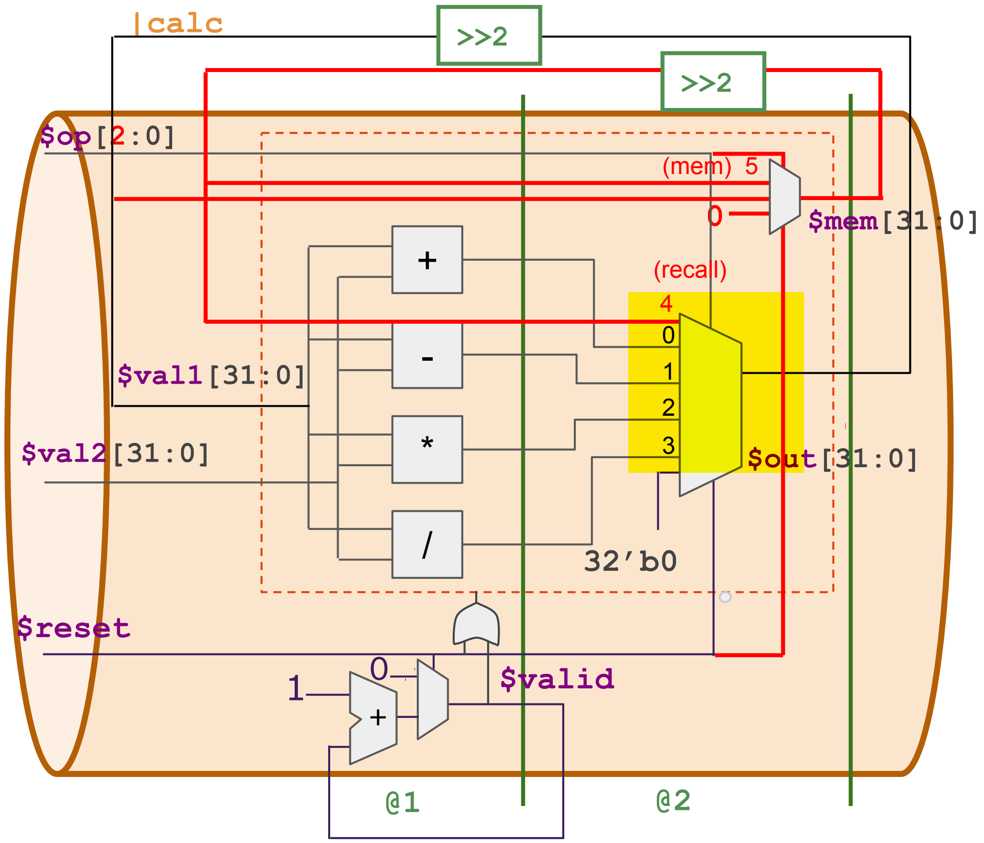
2. Verify behavior in waveform.



Lab: Calculator with Single-Value Memory

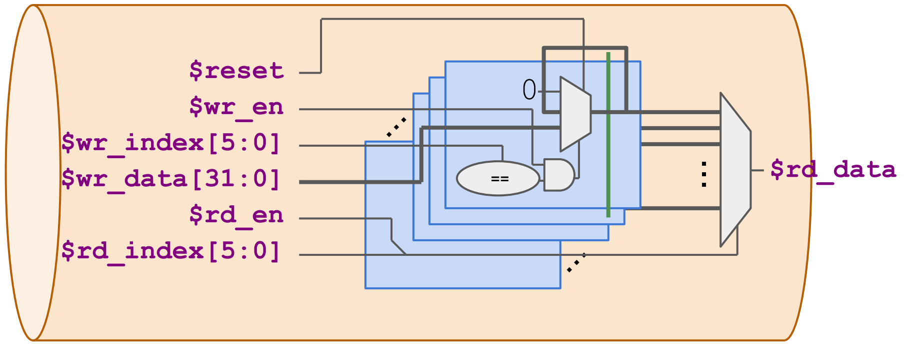
Calculators support “mem” and “recall”, to remember and recall a value.

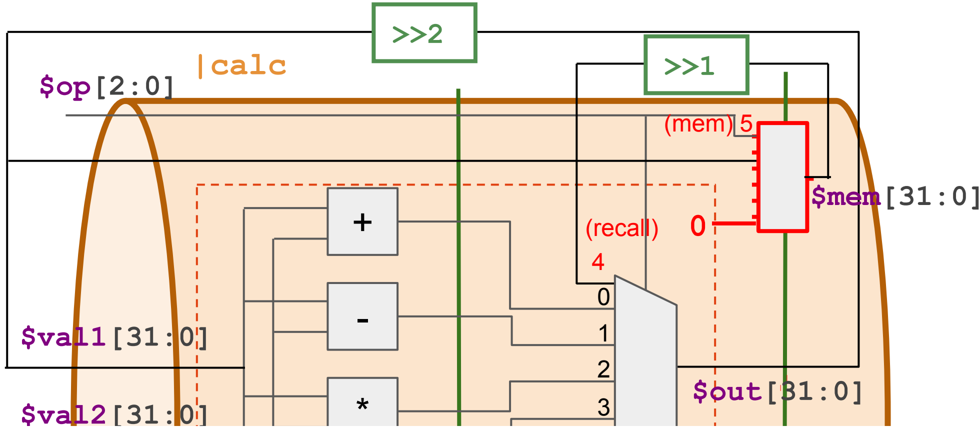
1. Extend $op to 3 bits.
2. Add memory MUX.
3. Select recall value in output MUX.
4. Verify behavior in waveform.



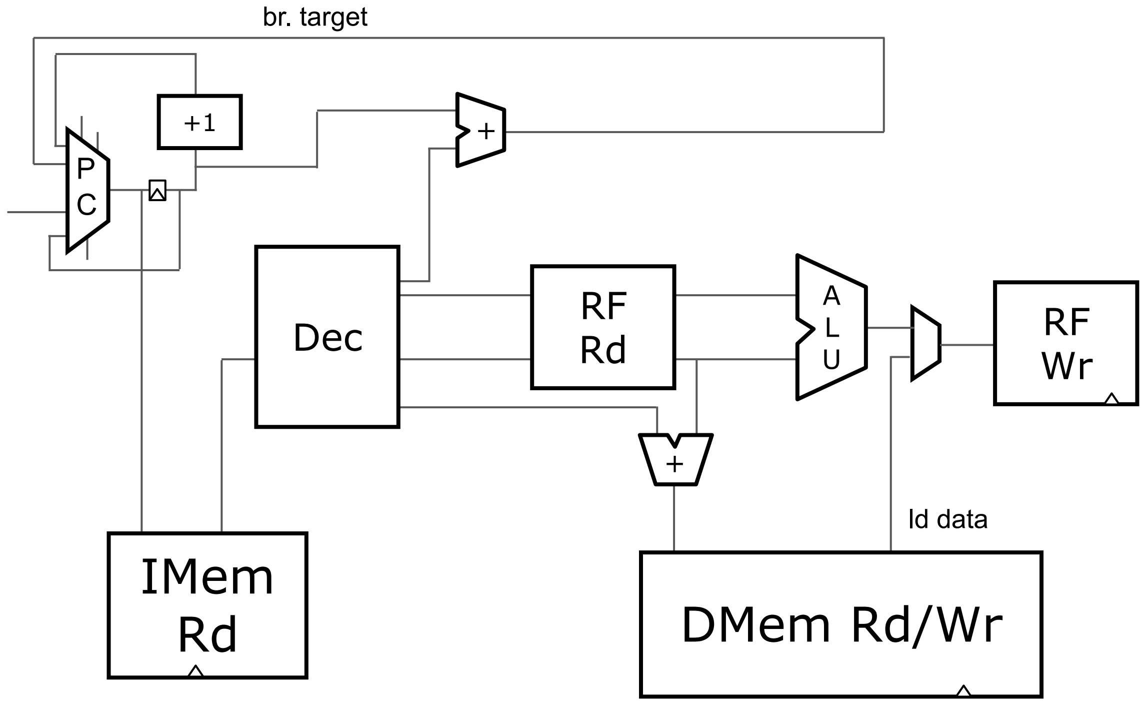
Lab: Calculator with Memory

1. Replace single-entry memory (`$mem[31:0]`) with an 8-entry memory.
2. `mem` and `recall` are wr_en/rd_en.
3. Let `$val1[2:0]` provide the rd/wr index.
4. Reference the previous slide to create and connect the memory.
5. Verify in simulation.

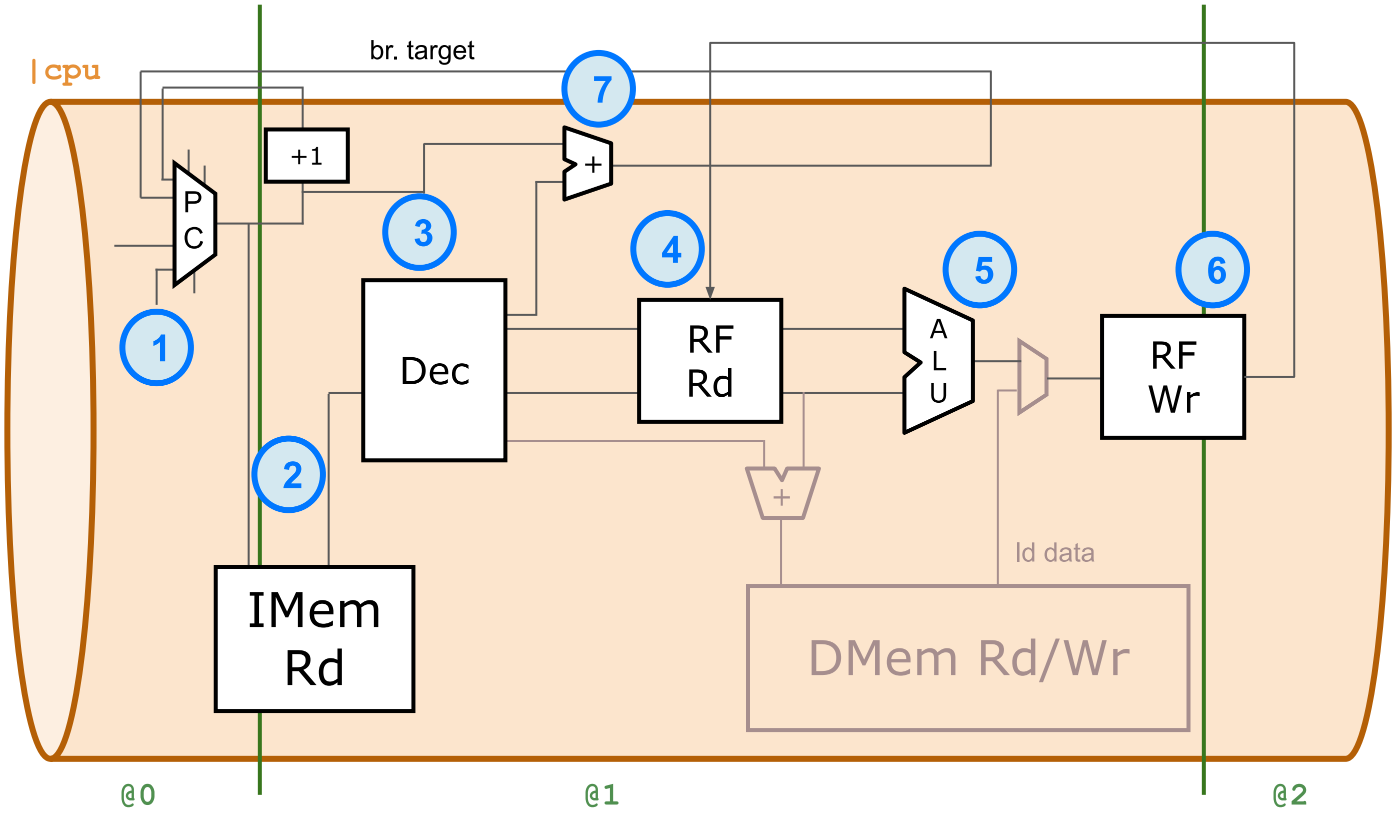




Example RISC-V Block Diagram



Implementation Plan



Lab: Instruction Types Decode

instr[6:2] determine instruction type: I, R, S, B, J, U

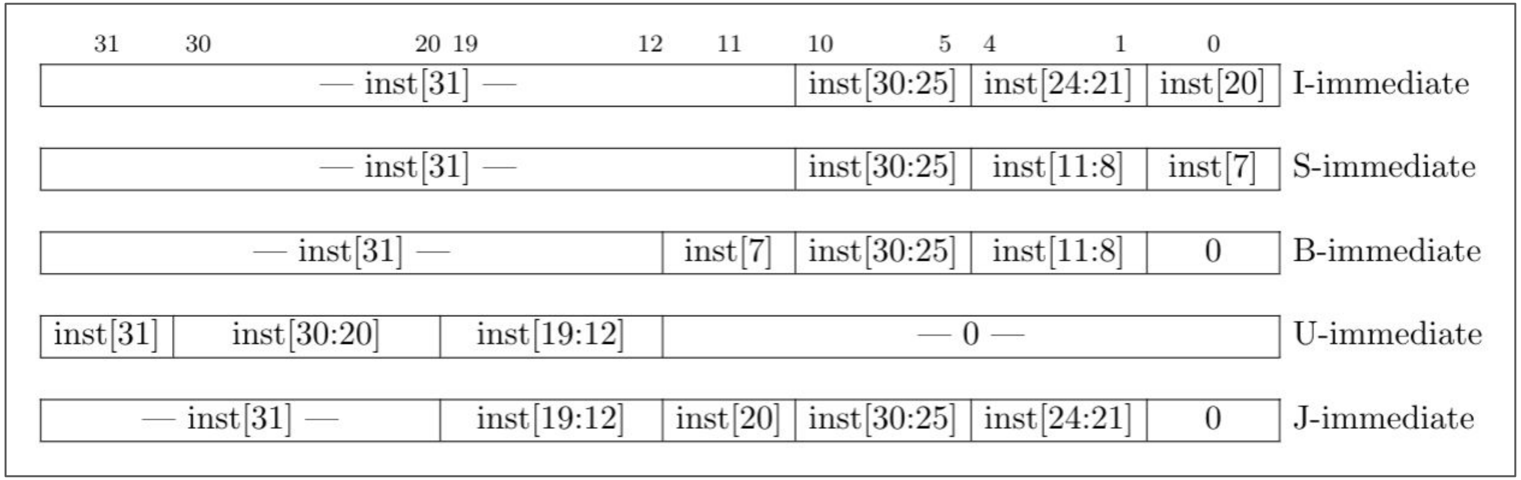
instr[4:2] / instr[6:5]	000	001	010	011	100	101	110	111
00	I	I	-	-	I	U	I	-
01	S	S	-	R	R	U	R	-
10	R4	R4	R4	R4	R	-	-	-
11	B	I	-	J	I (unused)	-	-	-

```
$is_i_instr = $instr[6:2] ==? 5'b0000x ||
              $instr[6:2] ==? 5'b001x0 ||
              ...;
...
```

Check behavior in simulation.

Lab: Instruction Immediate Decode

Form `$imm[31:0]` based on instruction type.

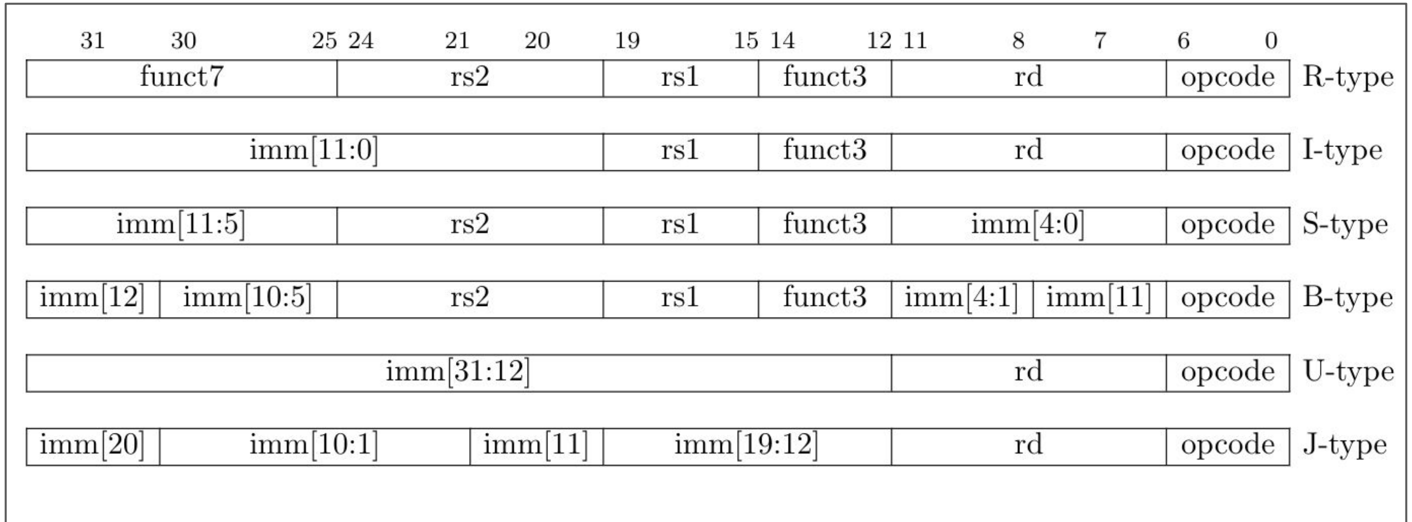


```
$imm[31:0] = $is_i_instr ? { {21{$instr[31]}}, $instr[30:20] } :
             $is_s_instr ? {...} :
             ...;
```

Check behavior in simulation.

Lab: Instruction Decode

Extract other instruction fields: `$funct7, $funct3, $rs1, $rs2, $rd, $opcode`



31	30 25	24 21	20	19 15	14 12	11 8	7	6 0	
funct7		rs2		rs1	funct3	rd		opcode	R-type
imm[11:0]				rs1	funct3	rd		opcode	I-type
imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode	S-type
imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
imm[31:12]						rd		opcode	U-type
imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode	J-type

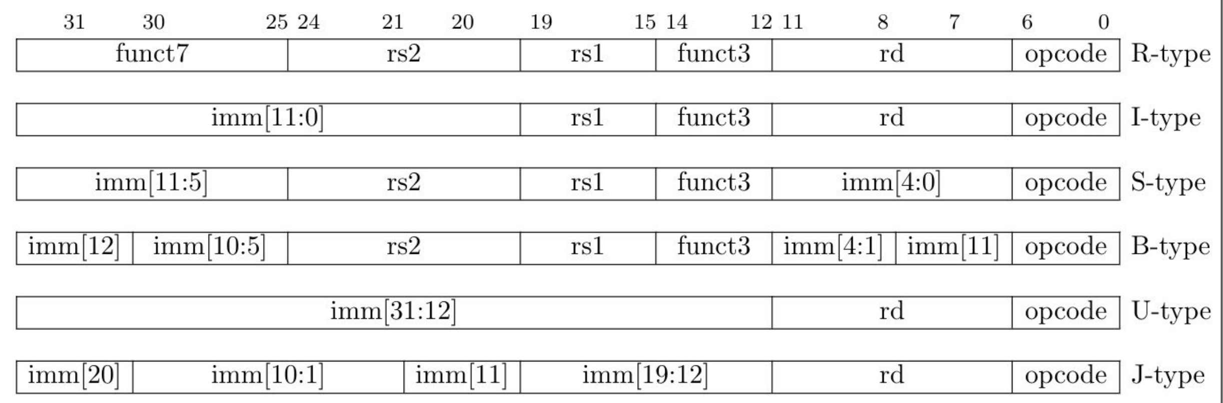
Figure 2.3: RISC-V base instruction formats showing immediate variants.

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

Check behavior in simulation.

Lab: RISC-V Instruction Field Decode

Let’s use when conditions



31	30 25	24 21	20	19 15	14 12	11 8	7	6 0	
funct7		rs2		rs1	funct3	rd		opcode	R-type
imm[11:0]				rs1	funct3	rd		opcode	I-type
imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode	S-type
imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
imm[31:12]						rd		opcode	U-type
imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode	J-type

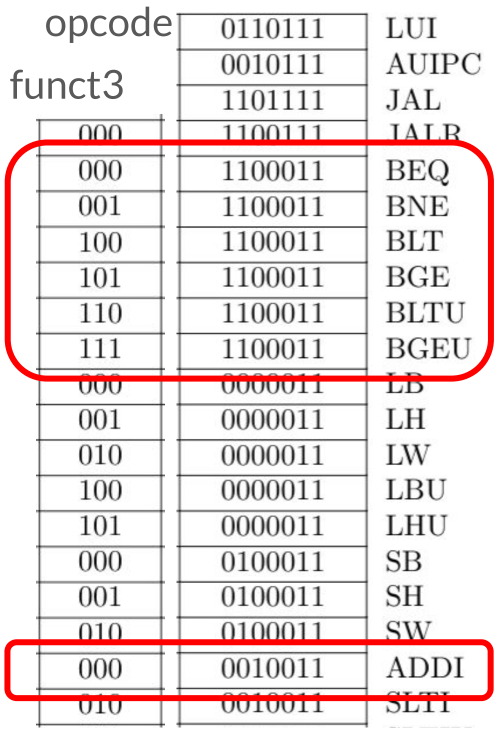
Figure 2.3: RISC-V base instruction formats showing immediate variants.

```
$rs2_valid = $is_r_instr || $is_s_instr || $is_b_instr;
?$rs2_valid
   $rs2[4:0] = $instr[24:20];
...
```

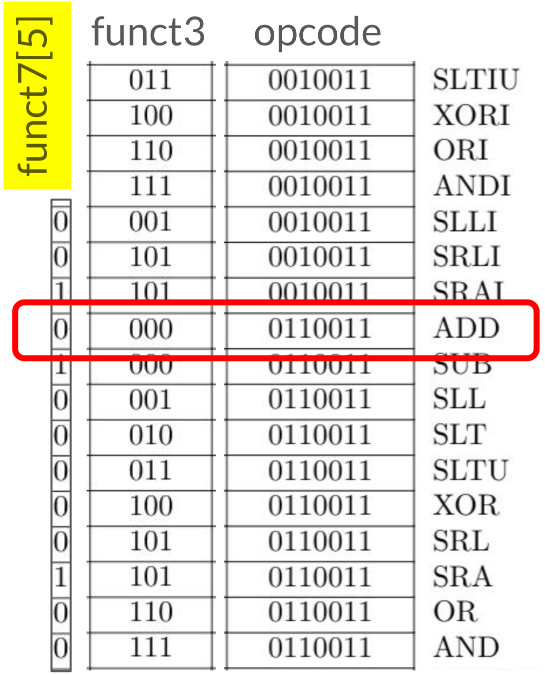
Check behavior in simulation.

Lab: Instruction Decode

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



funct3	opcode	
	0110111	LUI
	0010111	AUIPC
	1101111	JAL
000	1100111	JALR
000	1100011	BEQ
001	1100011	BNE
100	1100011	BLT
101	1100011	BGE
110	1100011	BLTU
111	1100011	BGEU
000	0000011	LB
001	0000011	LH
010	0000011	LW
100	0000011	LBU
101	0000011	LHU
000	0100011	SB
001	0100011	SH
010	0100011	SW
000	0010011	ADDI
010	0010011	SLTI



funct7[5]	funct3	opcode	
	011	0010011	SLTIU
	100	0010011	XORI
	110	0010011	ORI
	111	0010011	ANDI
0	001	0010011	SLLI
0	101	0010011	SRLI
1	101	0010011	SRAI
0	000	0110011	ADD
1	000	0110011	SUB
0	001	0110011	SLL
0	010	0110011	SLT
0	011	0110011	SLTU
0	100	0110011	XOR
0	101	0110011	SRL
1	101	0110011	SRA
0	110	0110011	OR
0	111	0110011	AND

Complete circled instructions.

```
$dec_bits[10:0] =
   {$funct7[5],$funct3,$opcode};
$is_beq = $dec_bits ==?
           11'bx_000_1100011;

// Until instrs are implemented,
// quiet down the warnings.
`BOGUS_USE($is_beq $is_bne ...)
```

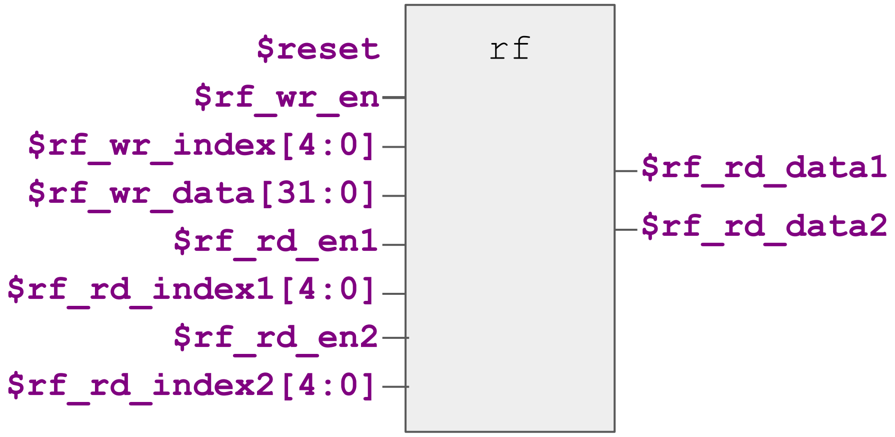
(no comma)

Check behavior in simulation and confirm save.

Lab: Register File Read

1. Uncomment `//m4+rf(@1, @1)` instantiation. (Default values are provided for inputs.)
2. Provide proper input assignments to enable RF read (**`rd`**) of **`$rs1/2`** when **`$rs1/2_valid`**.
3. Debug in simulation. Note that, on reset, register values are set to their index (e.g. x5 = 32'd5) so you can see something meaningful in simulation.

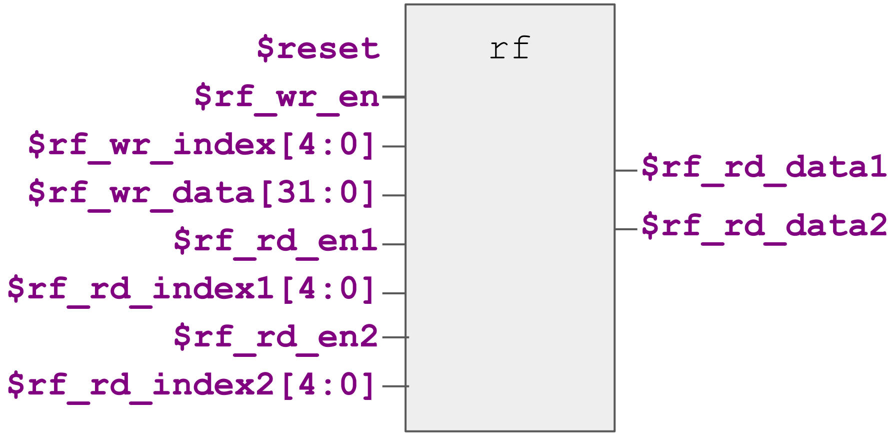
2-read, 1-write register file:



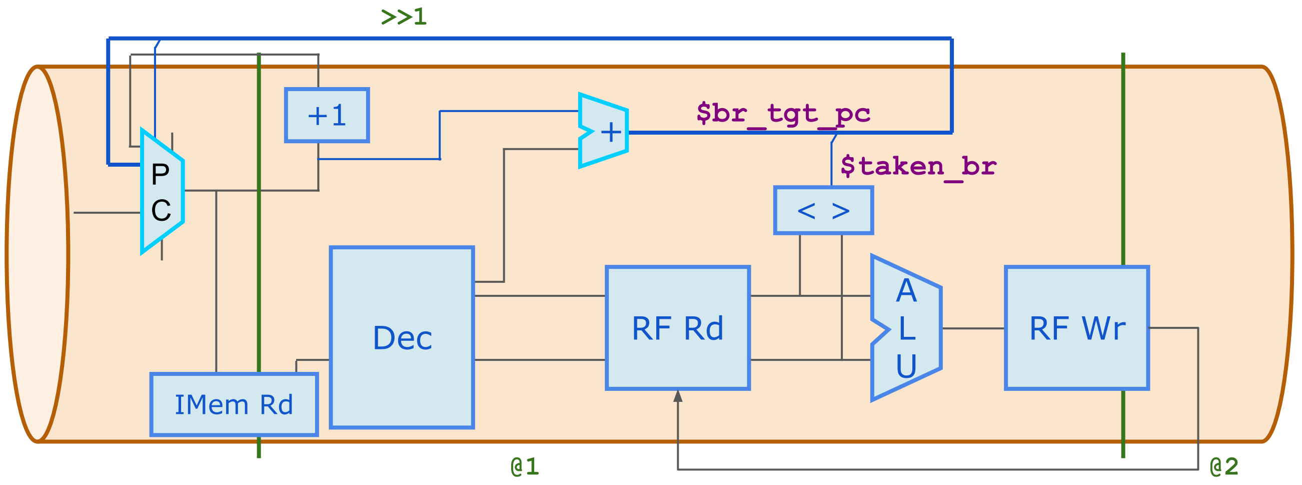
Lab: Register File Write

1. Provide proper input assignments to enable RF write (`wr`) of `$result` to `$rd` (dest reg) when `$rd_valid` for a valid instruction.
2. Debug in simulation. Should be writing and reading registers.
3. But wait, in RISC-V, x0 is "always-zero". Writes should be ignored. Add logic to disable write if `$rd` is 0.
4. Save outside of Makerchip.

2-read, 1-write register file:

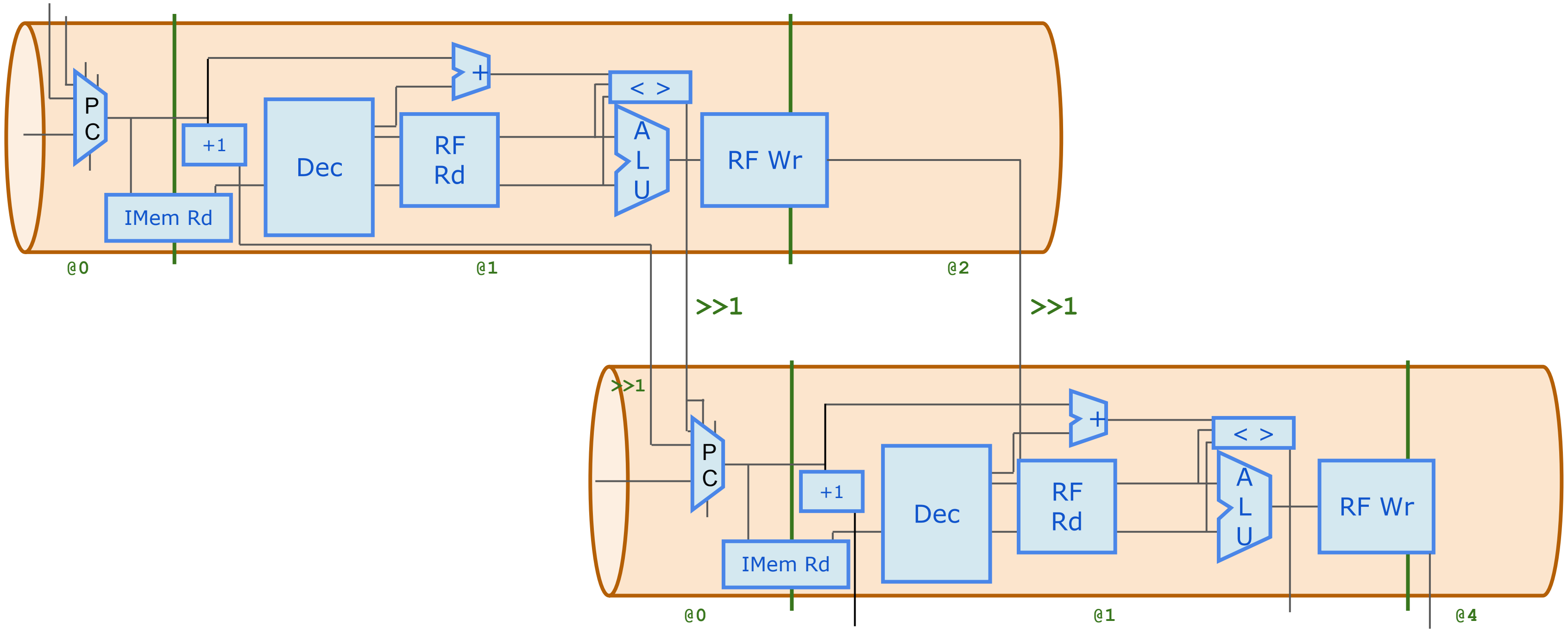


Lab: Branches

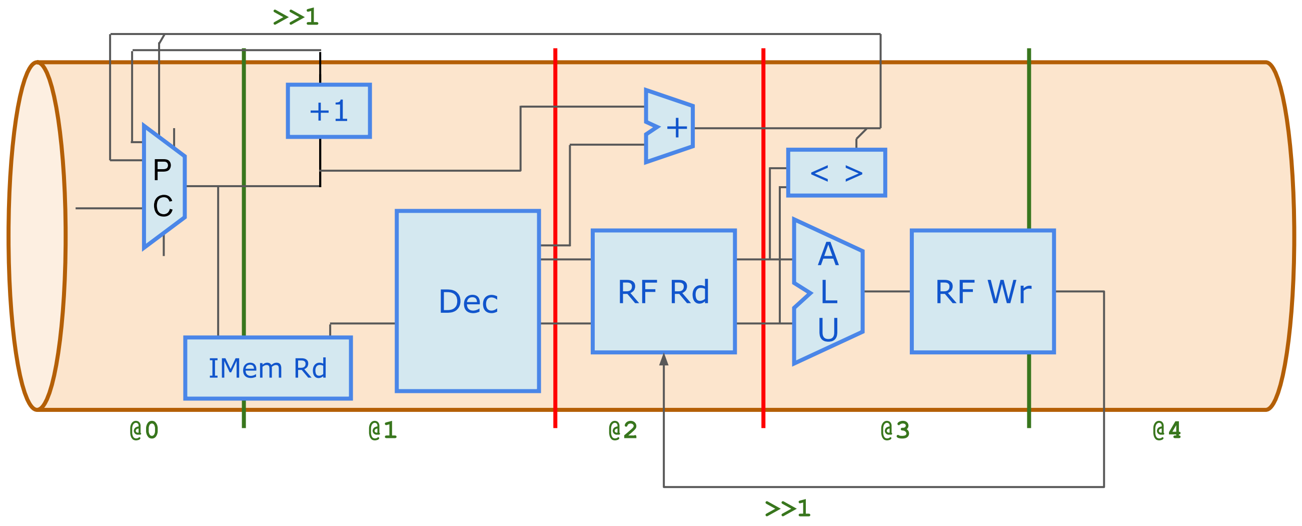


1. Compute **`$br_tgt_pc`** (PC + imm)
2. Modify **`$pc`** MUX expression to use the previous **`$br_tgt_pc`** when the previous instruction was **`$taken_br`**.
3. Check behavior in simulation. Program should now sum values {1..9}!
4. Debug as needed, and save outside of Makerchip.

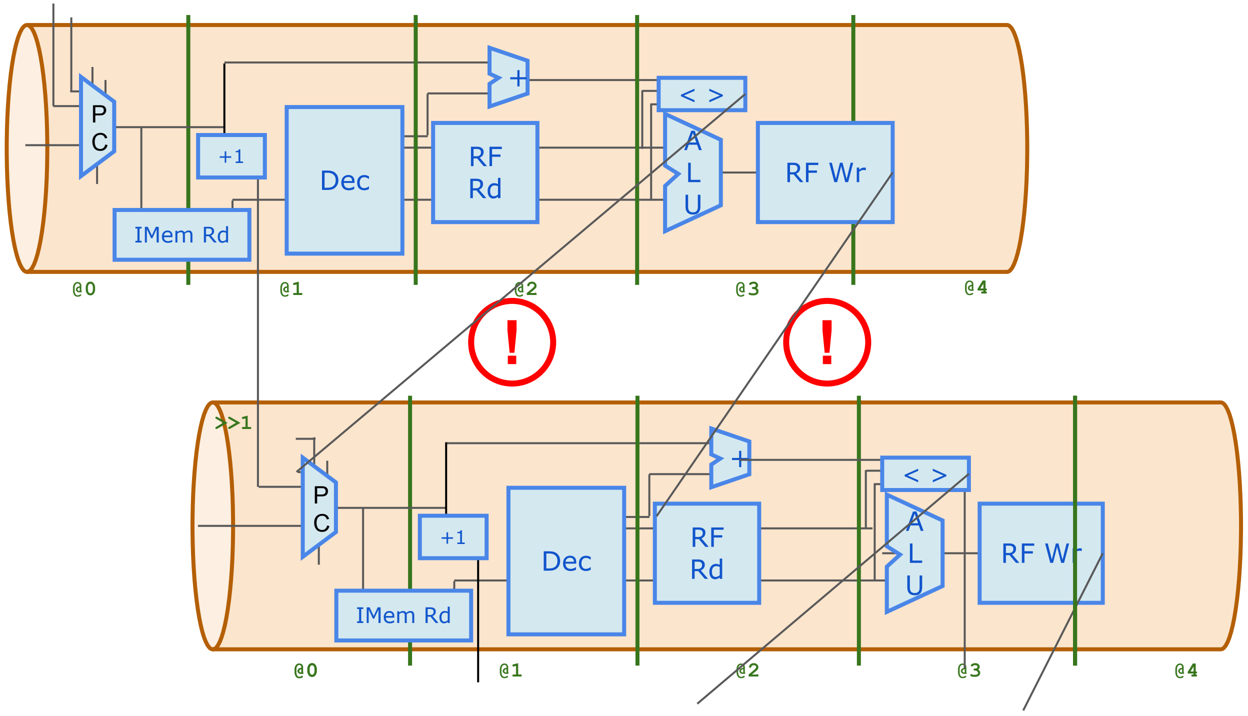
Waterfall Logic Diagram



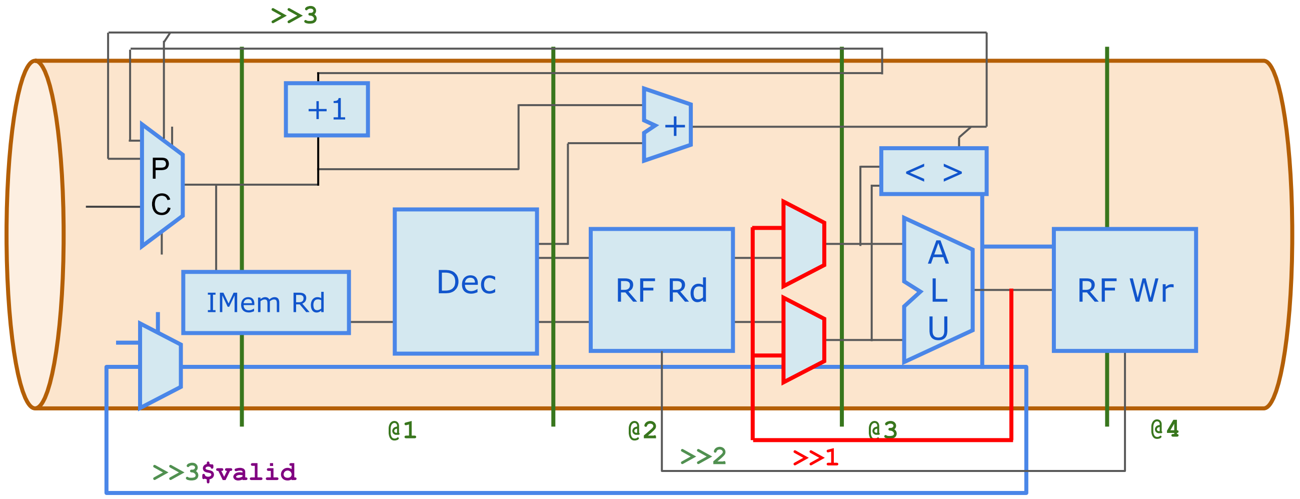
Pipelining Your RISC-V



Waterfall Logic Diagram

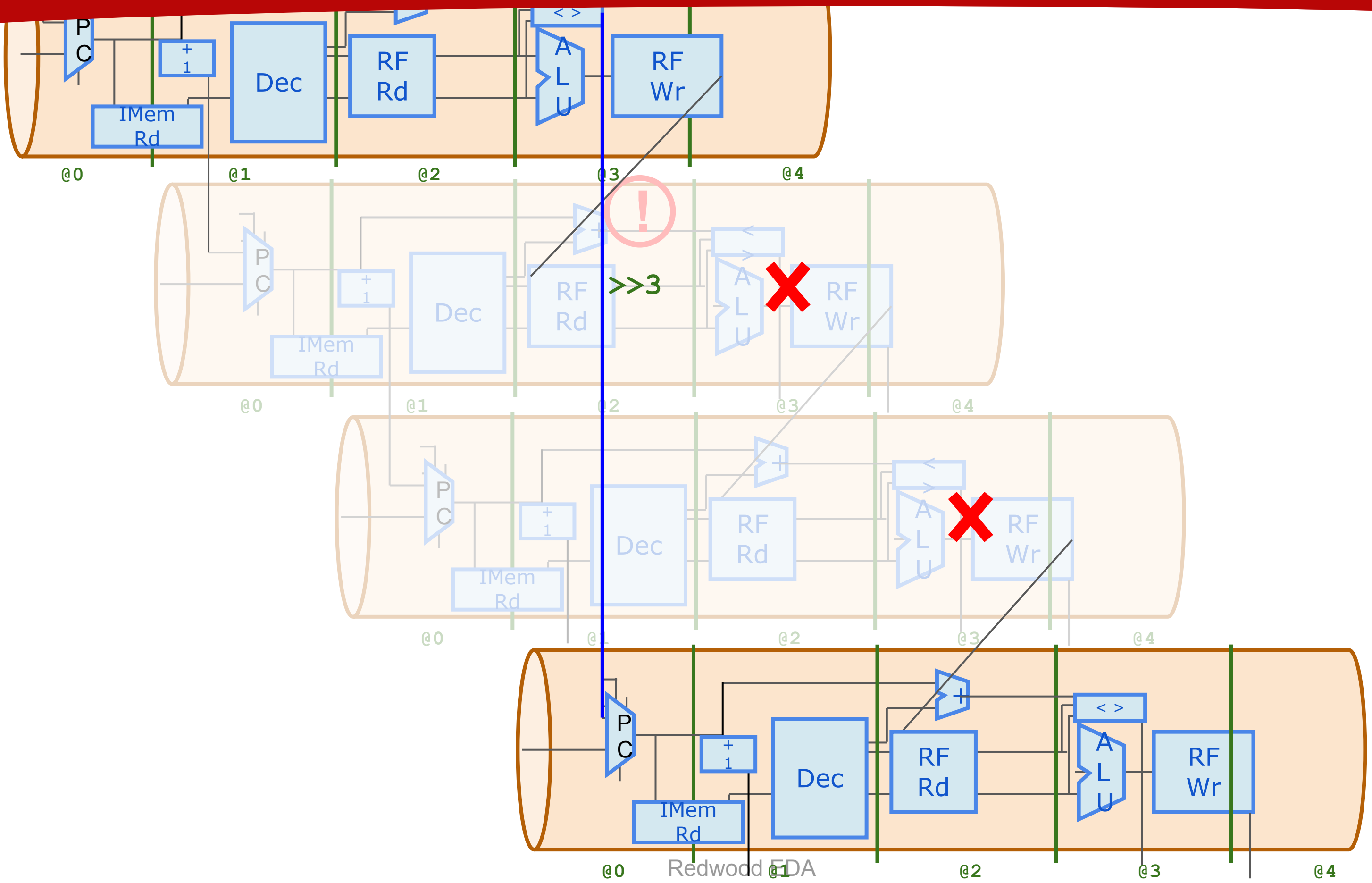


Lab: Register File Bypass

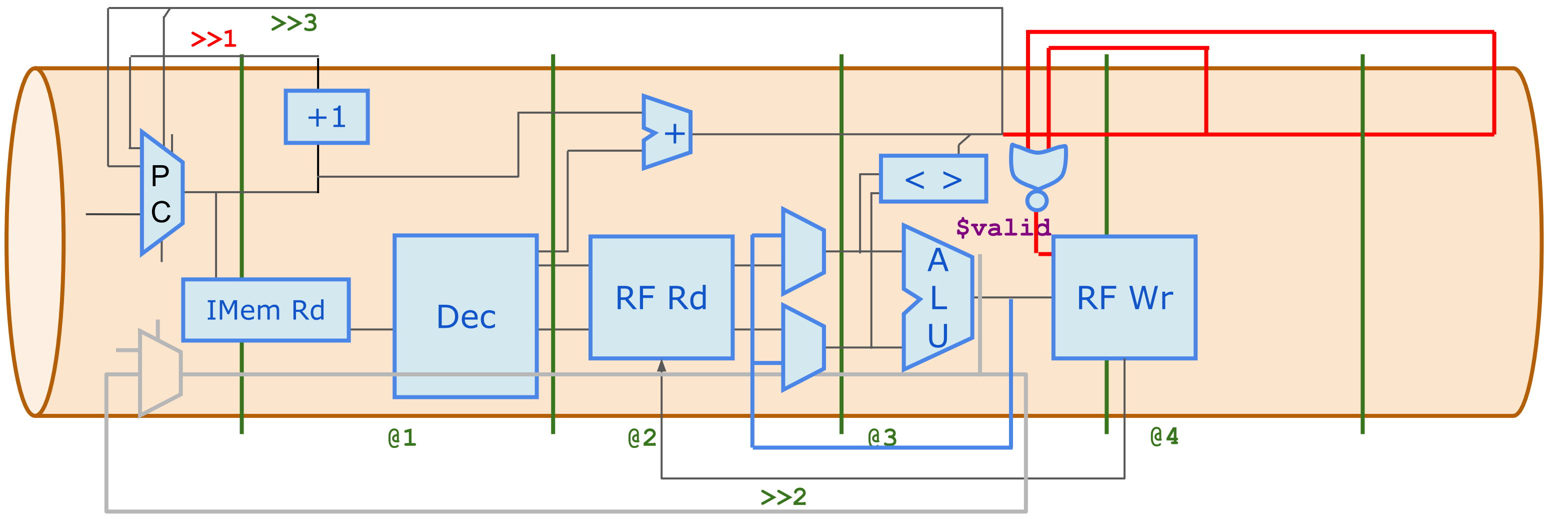


1. RF read uses RF as written 2 instructions ago (already correct).
2. Update expressions for **$srcX_value** to select previous **$result** if it was written to RF (write enable for RF) and if previous **$rd** == **$rsX**.
3. (Should have no effect yet)

Branches



Lab: Branches



1. Replace `@1 $valid` assignment with `@3 $valid` assignment based on the non-existence of a valid `$taken_br`'s in previous two instrutions.
2. Increment PC every cycle (not every 3 cycles)
3. (PC redirect for branches is already 3-cycle. No change.)
4. Debug. Save outside of Makerchip.

Lab: Complete ALU

Assign **`$result`** for other instrs

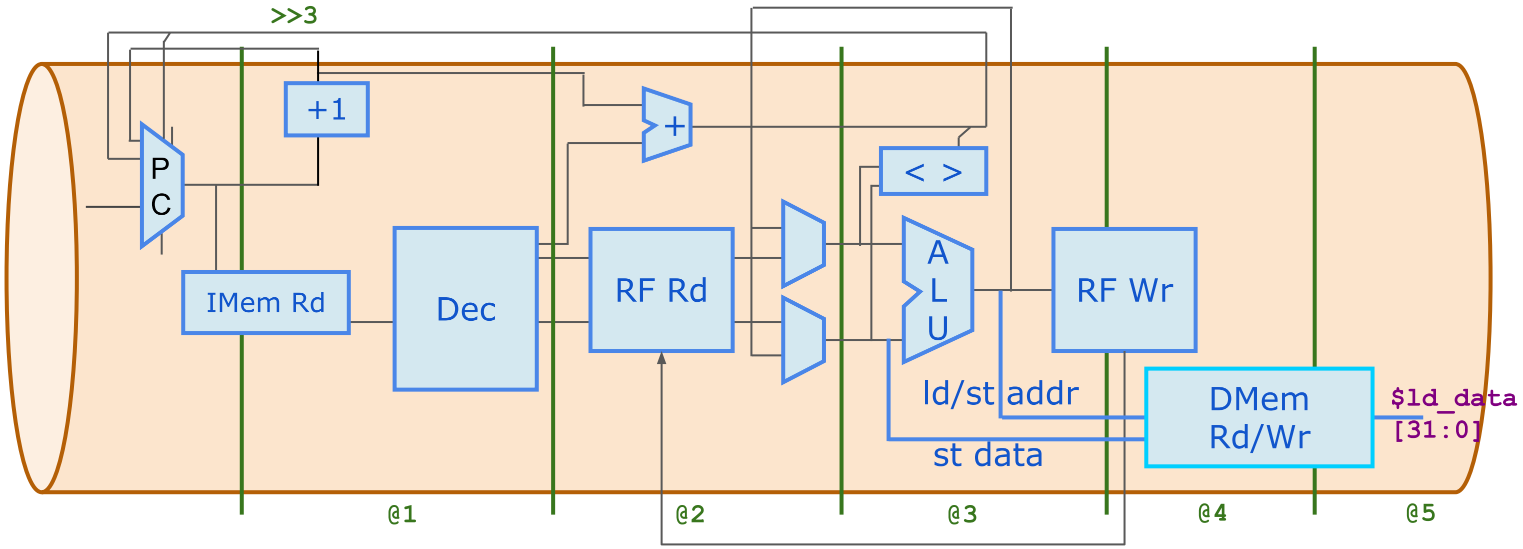
ANDI	`$src1_value & $imm;`
ORI	`$src1_value | $imm;`
XORI	`$src1_value ^ $imm;`
ADDI	`$src1_value + $imm;`
SLLI	`$src1_value << $imm[5:0];`
SRLI	`$src1_value >> $imm[5:0];`
AND	`$src1_value & $src2_value;`
OR	`$src1_value | $src2_value;`
XOR	`$src1_value ^ $src2_value;`

ADD	`$src1_value + $src2_value;`
SUB	`$src1_value - $src2_value;`
SLL	`$src1_value << $src2_value[4:0];`
SRL	`$src1_value >> $src2_value[4:0];`
SLTU	`$src1_value < $src2_value;`
SLTIU	`$src1_value < $imm;`
LUI	`{$imm[31:12], 12'b0};`
AUIPC	`$pc + $imm;`
JAL	`$pc + 4;`
JALR	`$pc + 4;`

SRAI	`{ {32{$src1_value[31]}}, $src1_value} >> $imm[4:0];`
SLT	`($src1_value[31] == $src2_value[31]) ? $sltu_rslt : {31'b0,$src1_value[31]};`
SLTI	`($src1_value[31] == $imm[31]) ? $sltiu_rslt : {31'b0,$src1_value[31]};`
SRA	`{ {32{$src1_value[31]}}, $src1_value} >> $src2_value[4:0];`

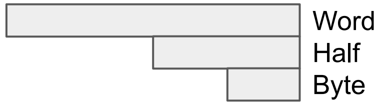
Need intermediate result signals for these.

Loads/Stores



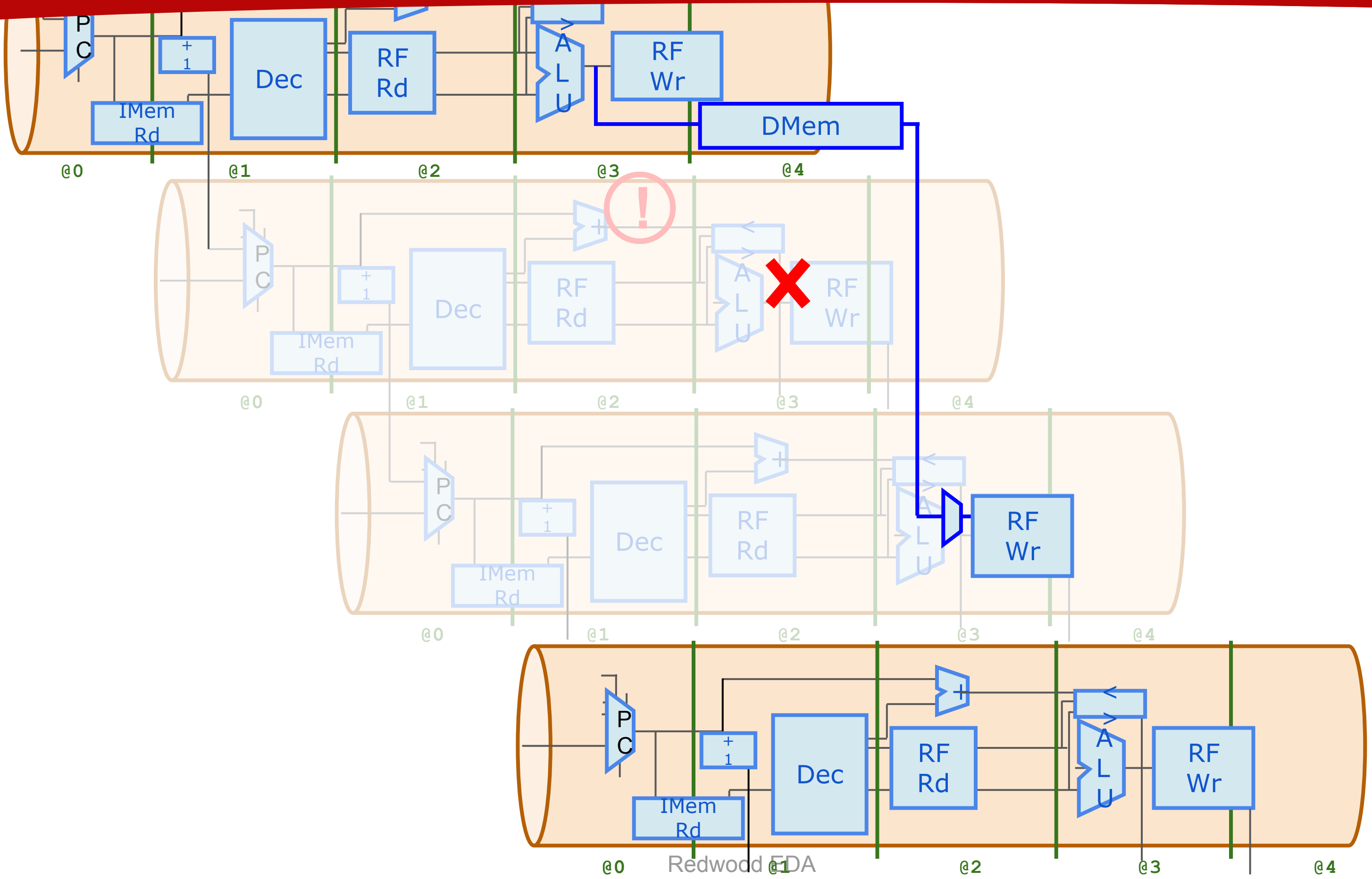
`LOAD (LW,LH,LB,LHU,LBU)`
`LOAD rd, imm(rs1)`
rd <= DMem[addr]

`STORE (SW,SH,SB)`
`STORE rs2, imm(rs1)`
DMem[addr] <= rs2

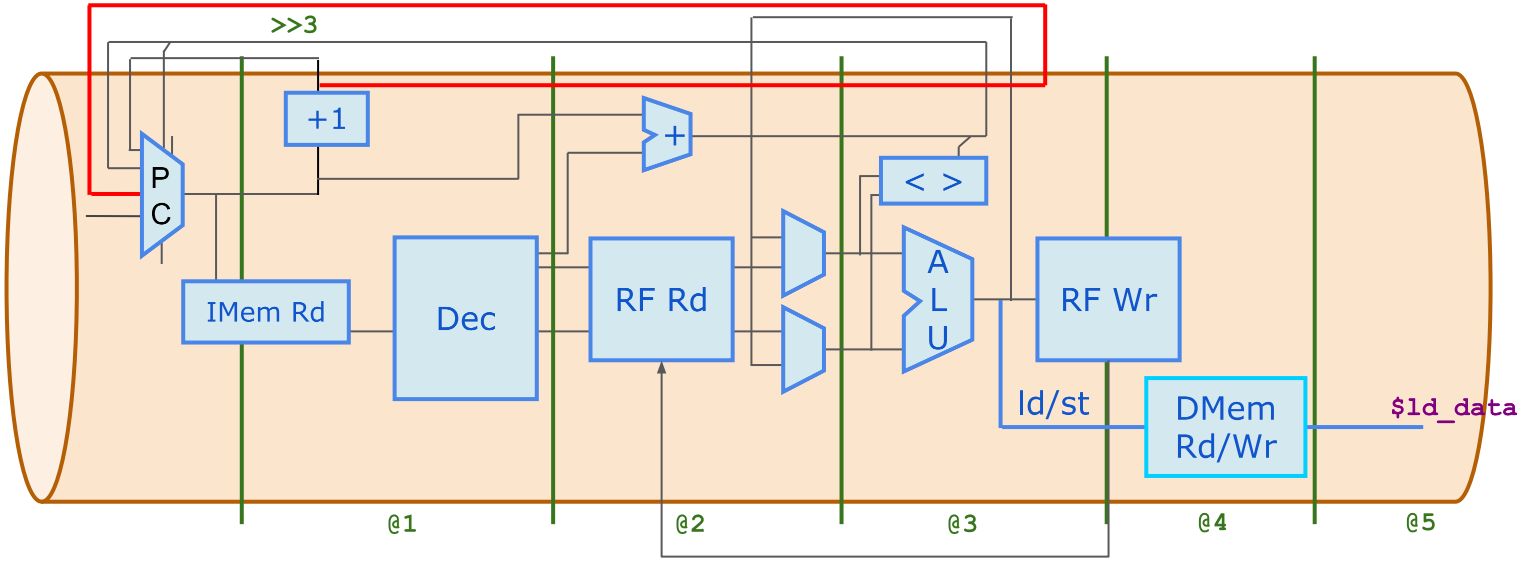


where, addr <= rs1 + imm (like addi)

Loads

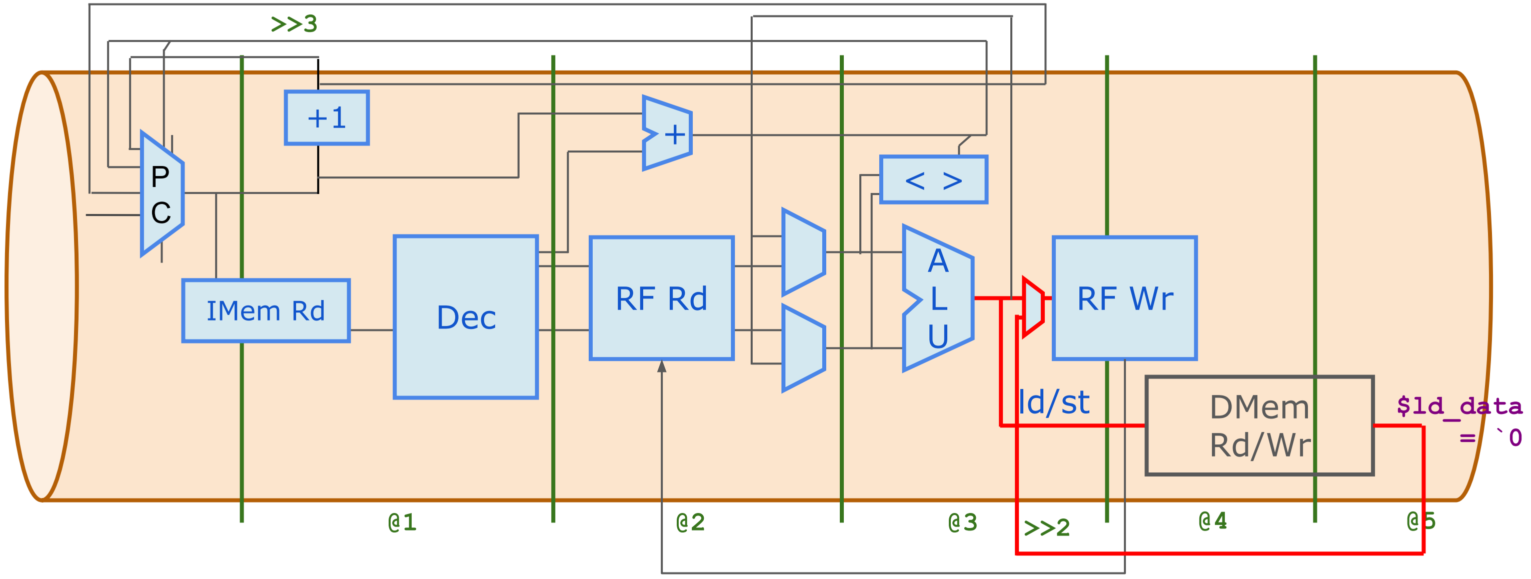


Lab: Redirect Loads



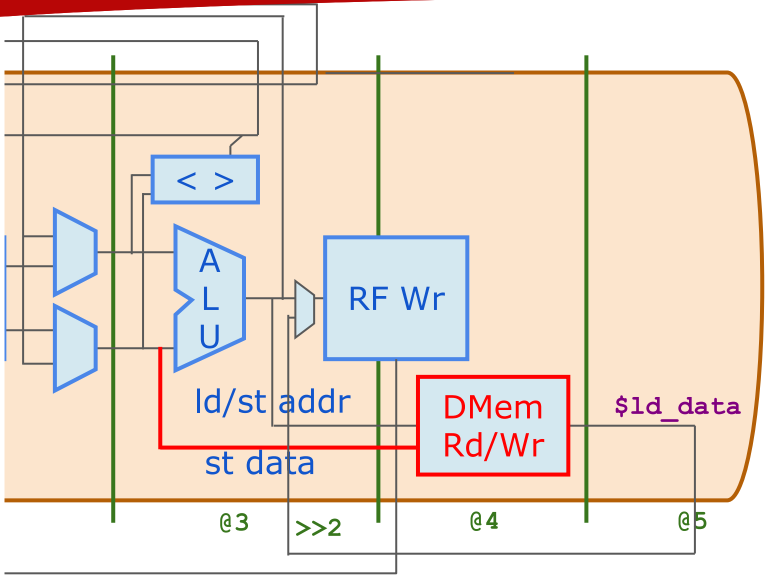
1. Clear **`$valid`** in the "shadow" of a load (like branch).
2. Select **`$inc_pc`** from 3 instructions ago for load redirect.
3. Debug. Confirm save.

Lab: Load Data

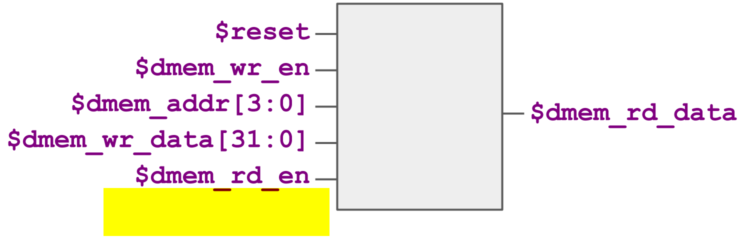


1. For loads/stores (**$is_load**/**$is_s_instr**), compute same result as for `addi`.
2. Add the RF-wr-data MUX to select **>>2$ld_data** and **>>2$rd** as RF inputs for **!$valid** instructions.
3. Enable write of **$ld_data** 2 instructions after valid **$load**.
4. Confirm save.

Lab: Load Data

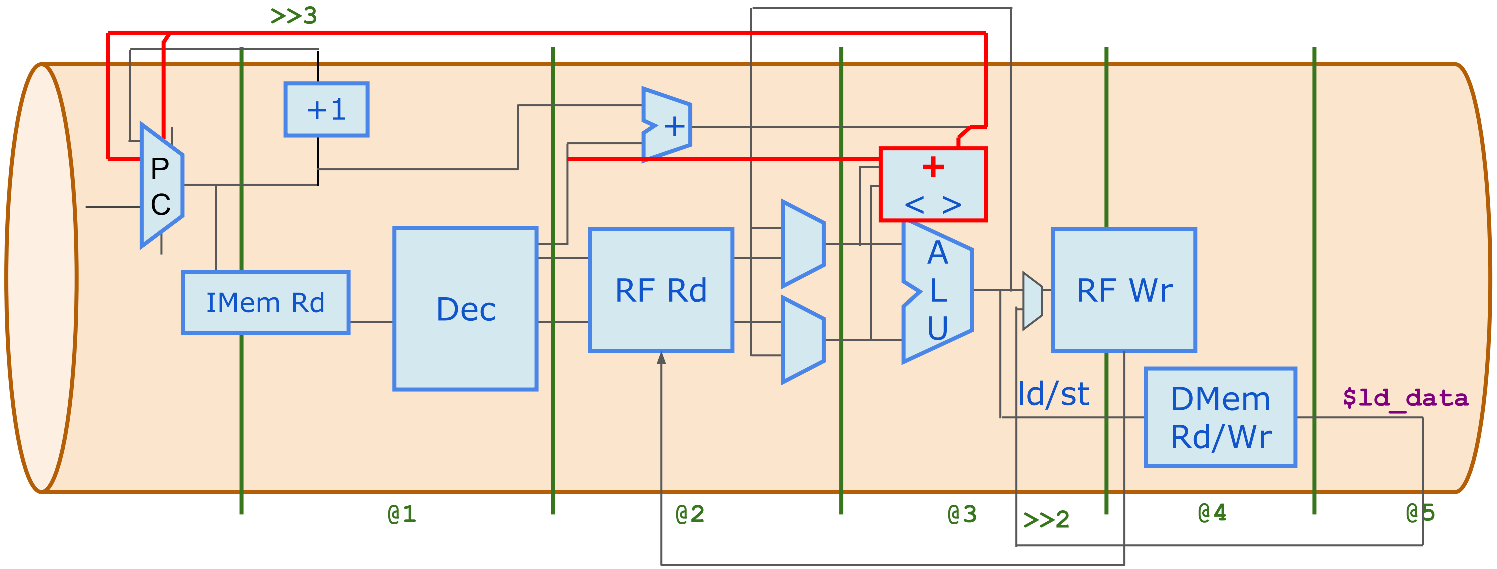


dmem: mini 1-R/W memory
(16-entries, 32-bits wide)



1. Uncomment `//m4+dmem(@4)`.
2. Connect interface signals above using address bits [5:2] to perform load and store (when valid).

Lab: Jumps



1. Define **$is_jump** (JAL or JALR), and, like **$taken_br**, create invalid cycles.
2. Compute **$jalr_tgt_pc** (SRC1 + IMM).
3. Select correct **$pc** for JAL (>>3**$br_tgt_pc**) and JALR (>>3**$jalr_tgt_pc**).
4. Save.