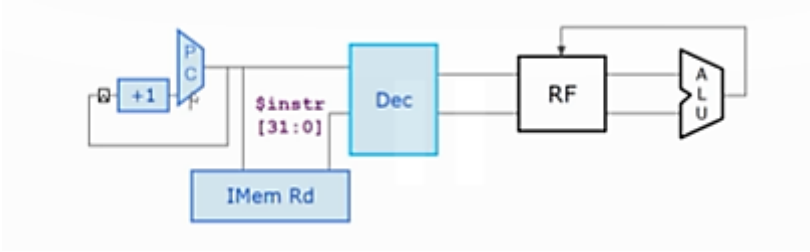
RISC-V Microarchitecture Design Report

Arijit Sengupta, 08th October 2020

Sandbox Link: https://myth2.makerchip.com/sandbox/05yf0hyMQ/0Nxh0D7



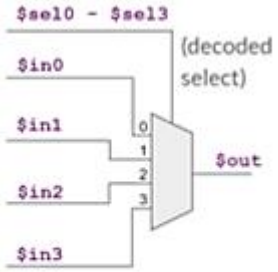
Instruction Types Decode:

	instr[4:2] instr[6:5]	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	-	-	-

$is_r_instr = ($instr[6:5] == 2'b01) || ($instr[6:5] == 2'b10);

$is_b_instr = $instr[6:5] == 2’b11;

Instruction Immediate Decode:

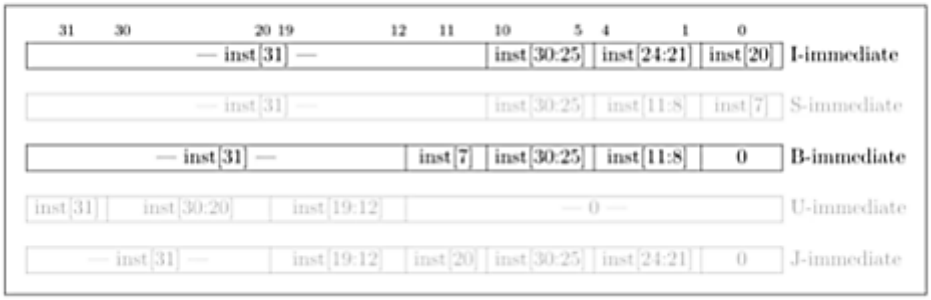


TL-Verilog:

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

(highest priority first.)

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
              $is_b_instr ? { {20{$instr[31]}}, $instr[7], $instr[30:25], $instr[11:8], 1'b0 } :    // B-type
              32'b0;   // Default (unused)
```

Instruction Field Decode:

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

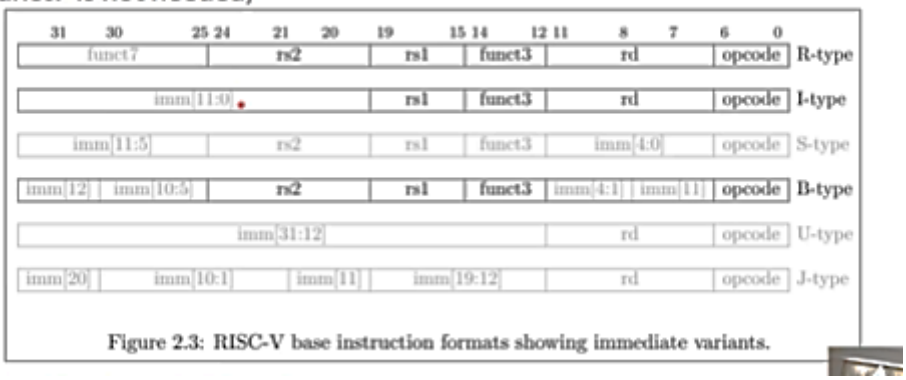


Figure 2.3: RISC-V base instruction formats showing immediate variants.

```
$rs2[4:0]    = $instr[24:20];
$rs1[4:0]    = $instr[19:15];
$funct3[2:0] = $instr[14:12];
$rd[4:0]     = $instr[11:7];
$opcode[6:0] = $instr[6:0];
```

Register Validity Decode:

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

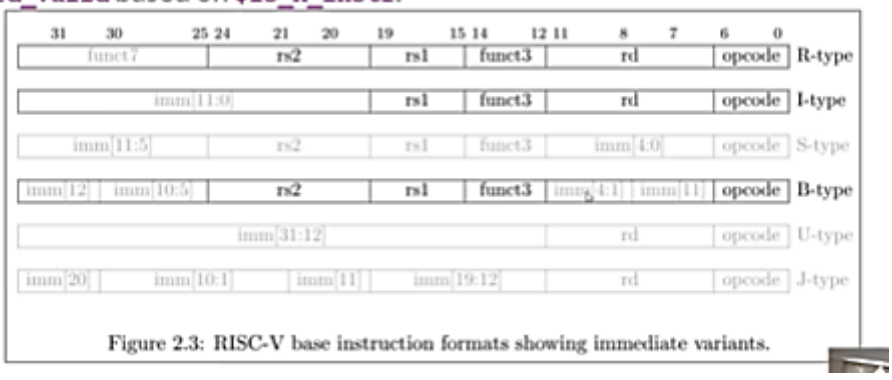
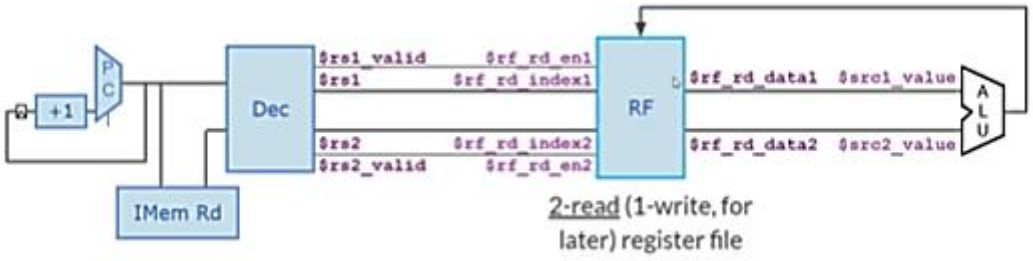
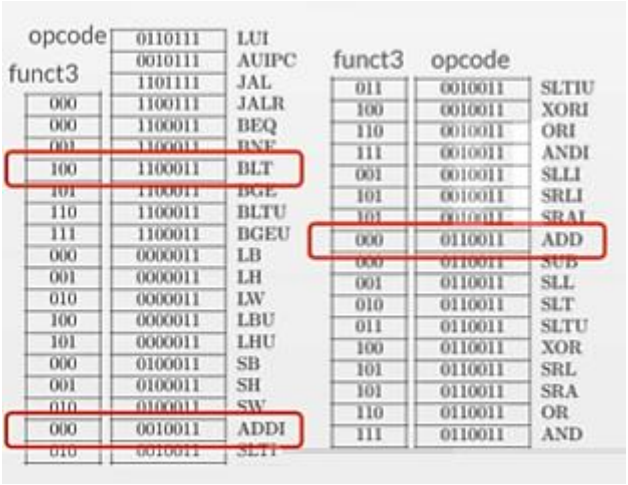


Figure 2.3: RISC-V base instruction formats showing immediate variants.

```
$rs1_valid = $is_r_instr || $is_i_instr || $is_b_instr;
$rs2_valid = $is_r_instr || $is_b_instr;
$rd_valid  = $is_r_instr || $is_i_instr;
```



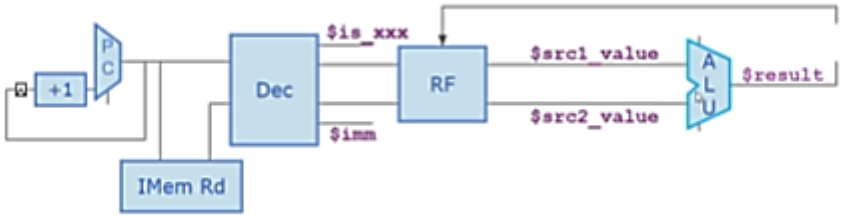
Instruction Decode:



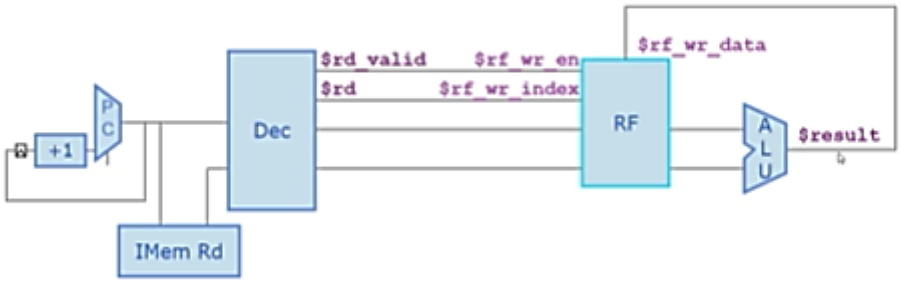
opcode			funct3	opcode	
	0110111	LUI			
	0010111	AUIPC			
	1101111	JAL			
funct3					
000	1100111	JALR			
000	1100011	BEQ			
001	1100011	BNE			
100	1100011	BLT			
101	1100011	BGE			
110	1100011	BLTU			
111	1100011	BGEU			
000	0000011	LB			
001	0000011	LH			
010	0000011	LW			
100	0000011	LBU			
101	0000011	LHU			
000	0100011	SB			
001	0100011	SH			
010	0100011	SW			
000	0010011	ADDI	011	0010011	SLTIU
010	0010011	SLTI	100	0010011	XORI
			110	0010011	ORI
			111	0010011	ANDI
			001	0010011	SLLI
			101	0010011	SRLI
			101	0010011	SRAI
			000	0110011	ADD
			000	0110011	SUB
			001	0110011	SLL
			010	0110011	SLT
			011	0110011	SLTU
			100	0110011	XOR
			101	0110011	SRL
			101	0110011	SRA
			110	0110011	OR
			111	0110011	AND

```
$dec_bits[9:0] = {$funct3, $opcode};

$is_blt  = $dec_bits == 10'b1001100011;

$is_addi = $dec_bits == 10'b0000010011;

$is_add  = $dec_bits == 10'b0000110011;
```

ALU:




```
$result[31:0] = $is_addi ? $src1_value + $imm :    // ADDI: src1 + imm
                $is_add ? $src1_value + $src2_value :    // ADD: src1 + src2
                32'b0;   // Default (unused)
```

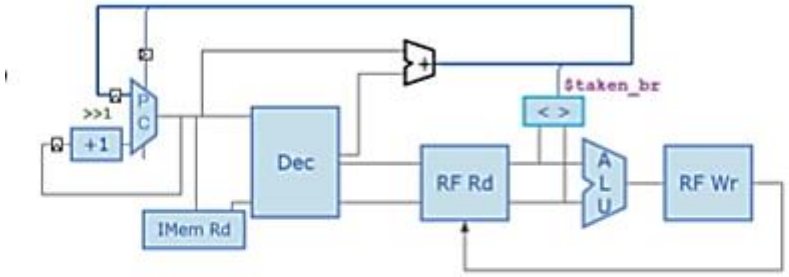


$rf_wr_en = $rd_valid;

$rf_wr_index[4:0] = $rd;

$rf_wr_data[31:0] = $result;

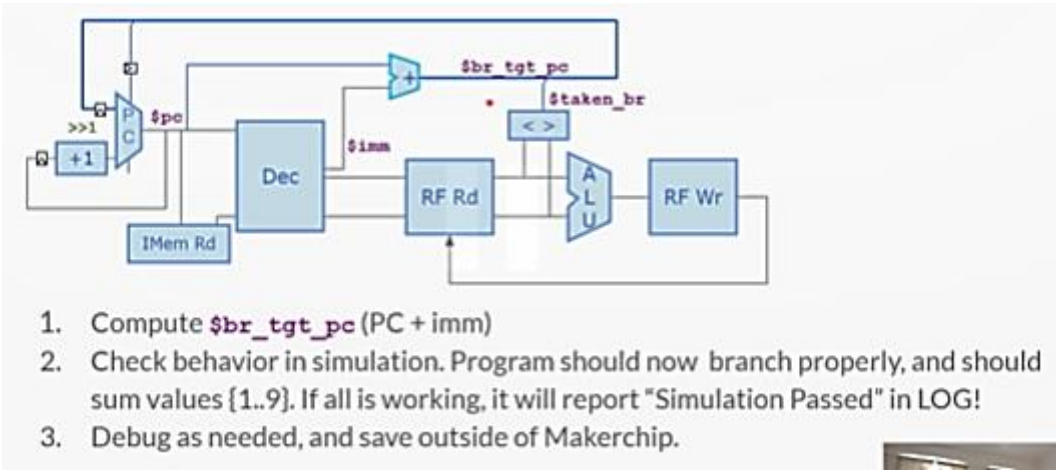
Branch Condition:



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

$taken_branch = ($src1_value < $src2_value) ? $is_blt : 0;

Branch Target:



$br_target_pc[31:0] = $pc[31:0] + $imm;

Completed Code:

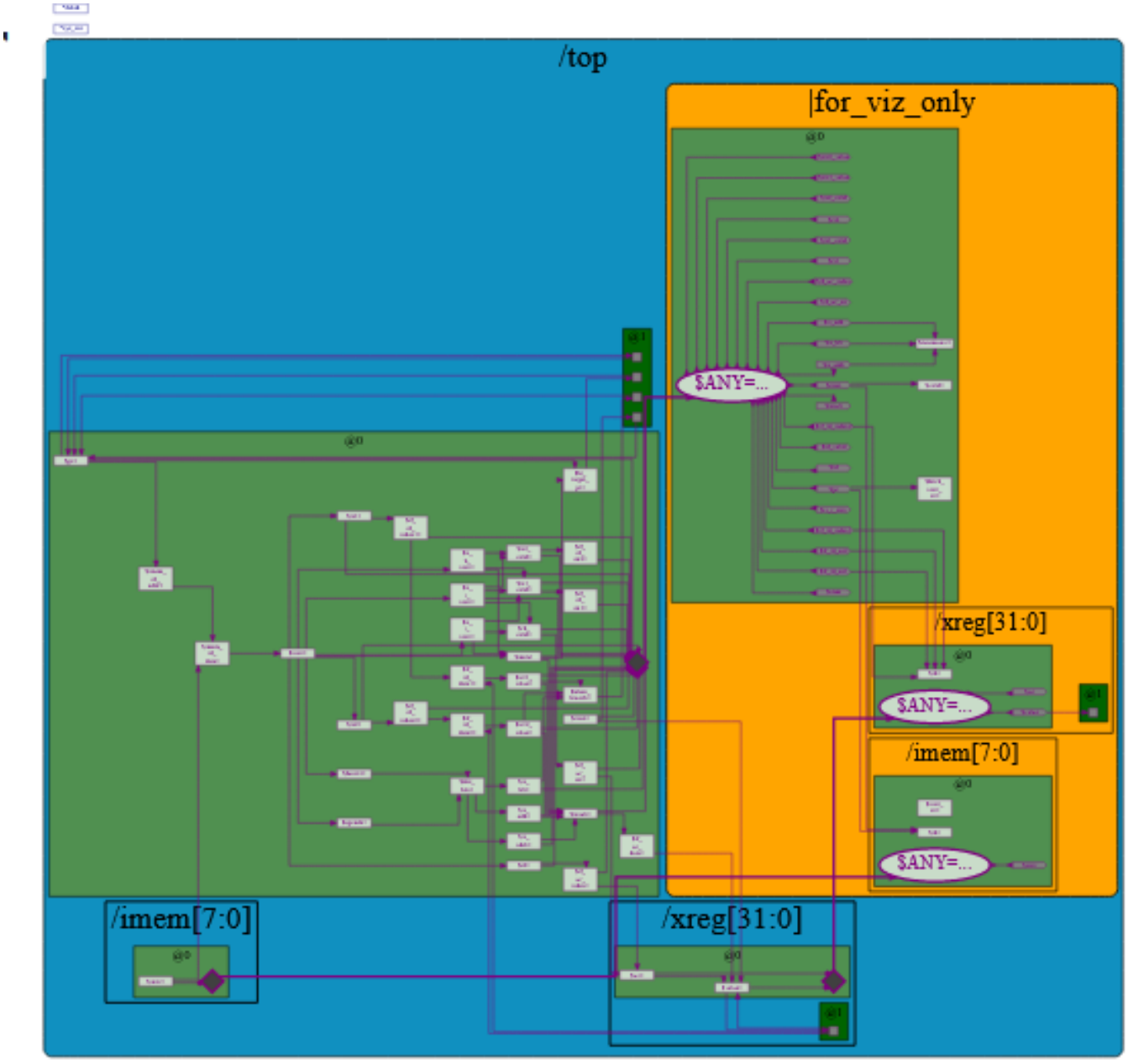
```
\m4_TLV_version 1d: tl-x.org
\SV
  // This code can be found in: https://github.com/stevehoover/VSDOpen2020_TLV_RISC-V_Tutorial
  m4_include_lib(['https://raw.githubusercontent.com/stevehoover/VSDOpen2020_TLV_RISC-
V_Tutorial/f5838e91523a968a41d00f25f025f819e3831046/lib/shell.tlv'])
\SV
  m4_makerchip_module   // (Expanded in Nav-TLV pane.)
  /* verilator lint_on WIDTH */
\TLV
  // ------------------------------------------------------
  //
  // /====================\
  // | Sum 1 to 9 Program |
  // \====================/
  //
  // Program for MYTH Workshop to test RV32I
  // Add 1,2,3,...,9 (in that order).
  //
  // Regs:
  //  r10 (a0): In: 0, Out: final sum
  //  r12 (a2): 10
  //  r13 (a3): 1..10
  //  r14 (a4): Sum
  //
  // External to function:
  m4_asm(ADD, r10, r0, r0)            // Initialize x10 to 0.
  // Function:
  m4_asm(ADD, r14, r10, r0)           // Initialize sum register x14 with 0x0
  m4_asm(ADDI, r12, r10, 1010)        // Store count of 10 in register x12.
  m4_asm(ADD, r13, r10, r0)           // Initialize intermediate sum register x13 with 0
  // Loop:
  m4_asm(ADD, r14, r13, r14)          // Incremental addition
  m4_asm(ADDI, r13, r13, 1)           // Increment intermediate register by 1
  m4_asm(BLT, r13, r12, 1111111111000) // If x13 is less than x12, branch to <loop>
  m4_asm(ADD, r10, r14, r0)           // Store final result to register x10 so that it can be read by main
program
  //
  // ------------------------------------------------------


  // PC
  $pc[31:0] = >>1$reset       ? 32'0 :
              >>1$taken_branch ? >>1$br_target_pc :    // (initially $taken_branch == 0)
                                 >>1$pc + 32'b100;

  // IMem Hookup
  $imem_rd_addr[2:0] = $pc[4:2];
  $instr[31:0] = $imem_rd_data;

  // **Lab: Instruction Types Decode
  $is_i_instr = $instr[6:5] == 2'b00;
```

```
$is_r_instr = ($instr[6:5] == 2'b01) || ($instr[6:5] == 2'b10);
$is_b_instr = $instr[6:5] == 2'b11;

// **Lab: Instruction Immediate Decode
$imm[31:0]  = $is_i_instr ? { {21{$instr[31]}}, $instr[30:20] } :   // I-type
          $is_b_instr ? { {20{$instr[31]}}, $instr[7], $instr[30:25], $instr[11:8], 1'b0 } :   // B-type
          32'b0;   // Default (unused)

// **Lab: Instruction Field Decode
$rs2[4:0]    = $instr[24:20];
$rs1[4:0]    = $instr[19:15];
$funct3[2:0] = $instr[14:12];
$rd[4:0]     = $instr[11:7];
$opcode[6:0] = $instr[6:0];

// **Lab: Register Validity Decode
$rs1_valid = $is_r_instr || $is_i_instr || $is_b_instr;
$rs2_valid = $is_r_instr || $is_b_instr;
$rd_valid  = $is_r_instr || $is_i_instr;

// Register File Read Hookup
$rf_rd_en1        = $rs1_valid;
$rf_rd_en2        = $rs2_valid;
$rf_rd_index1[4:0] = $rs1;
$rf_rd_index2[4:0] = $rs2;
$src1_value[31:0] = $rf_rd_data1;
$src2_value[31:0] = $rf_rd_data2;

// **Lab: Instruction Decode
$dec_bits[9:0] = {$funct3, $opcode};
$is_blt  = $dec_bits == 10'b1001100011;
$is_addi = $dec_bits == 10'b0000010011;
$is_add  = $dec_bits == 10'b0000110011;

// **Lab: ALU
$result[31:0] = $is_addi ? $src1_value + $imm :    // ADDI: src1 + imm
            $is_add ? $src1_value + $src2_value :   // ADD: src1 + src2
                  32'b0;   // Default (unused)

// Register File Write Hookup
$rf_wr_en        = $rd_valid;
$rf_wr_index[4:0] = $rd;
$rf_wr_data[31:0] = $result;

// **Lab: Branch Condition
$taken_branch = ($src1_value < $src2_value) ? $is_blt : 1'b0;

// **Lab: Branch Target
$br_target_pc[31:0] = $pc[31:0] + $imm;
// Note: $taken_branch and $br_target_pc control the PC mux.
```

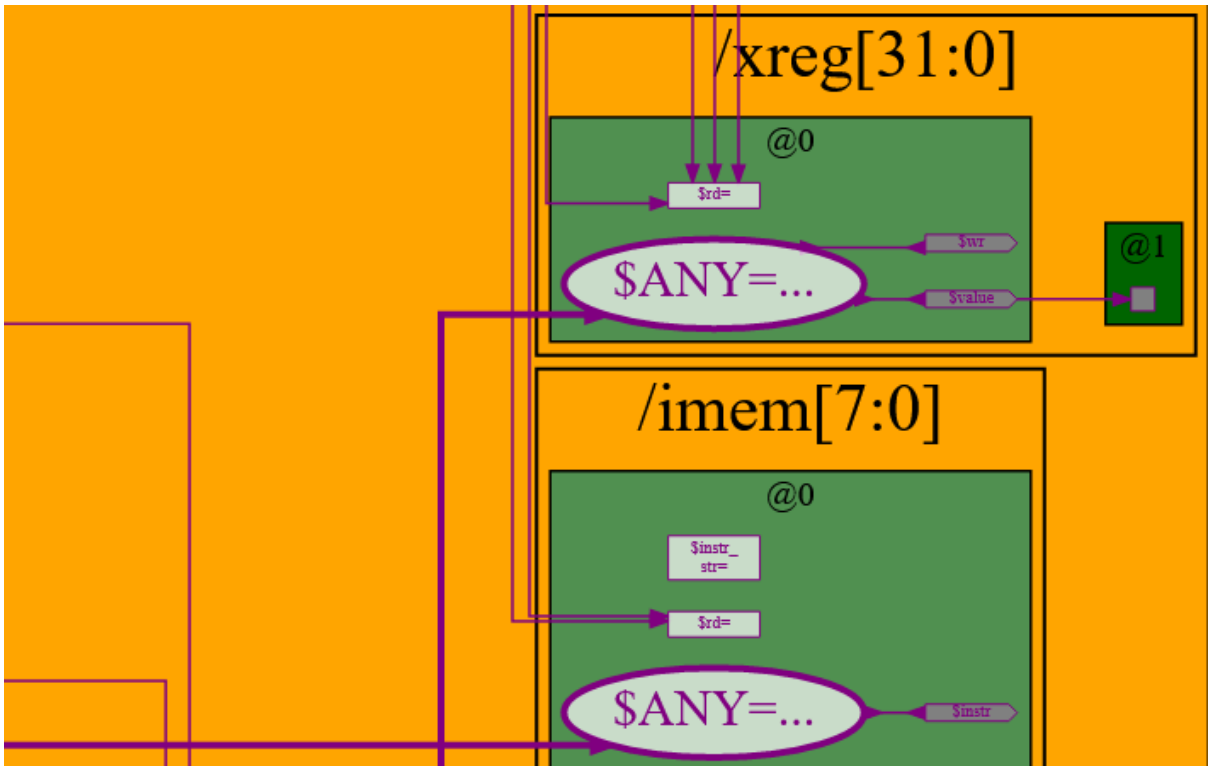
m4+shell()

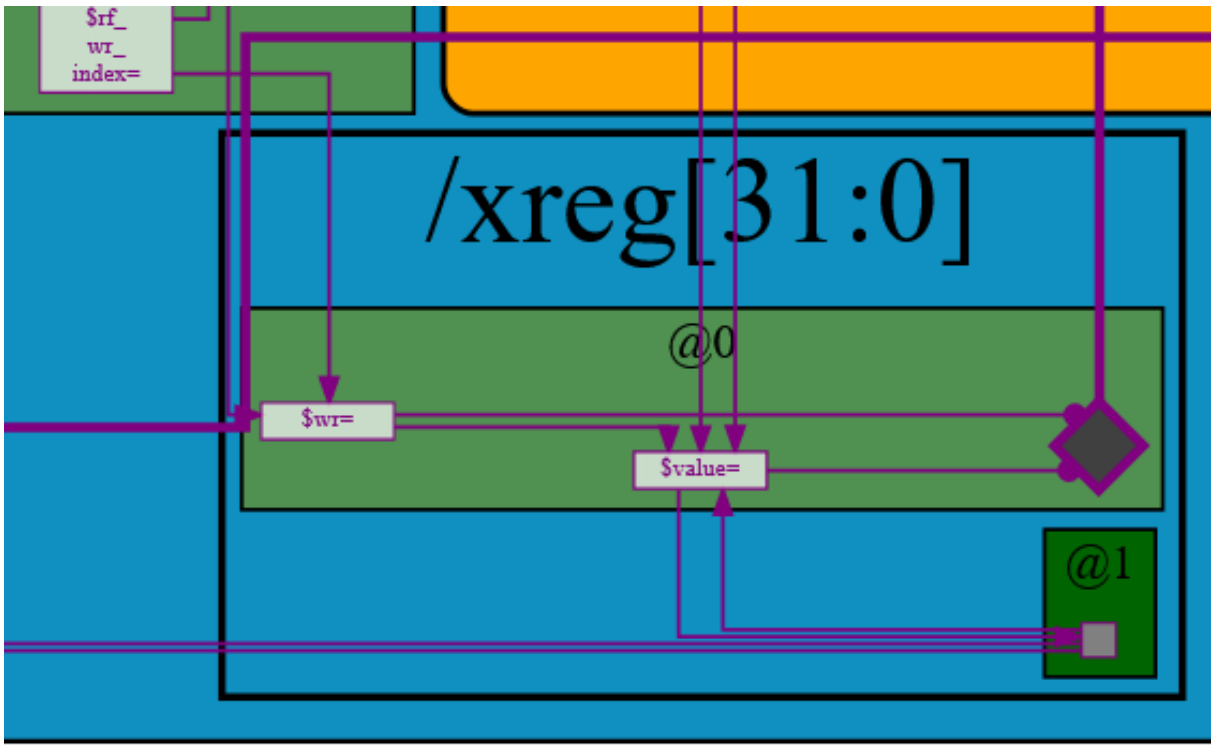
\SV
Endmodule

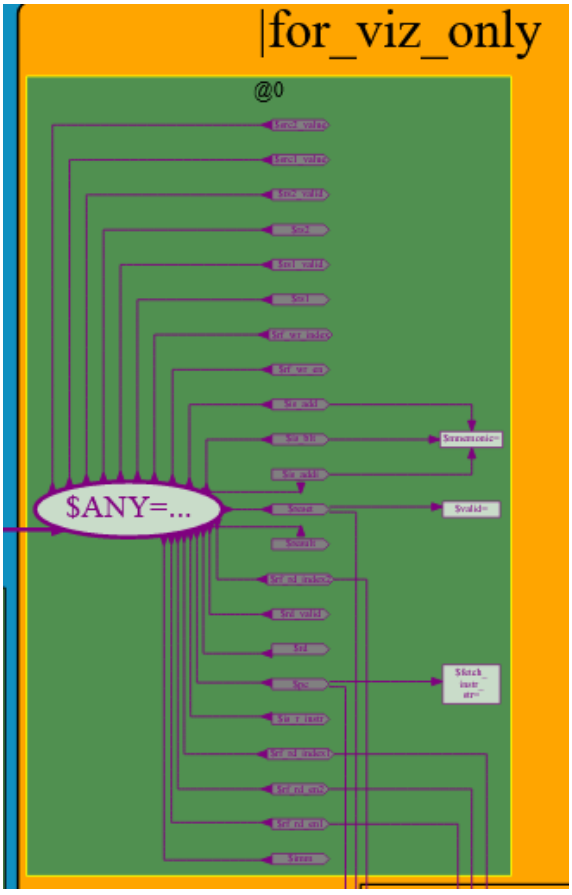
The completed CPU looks as follows:

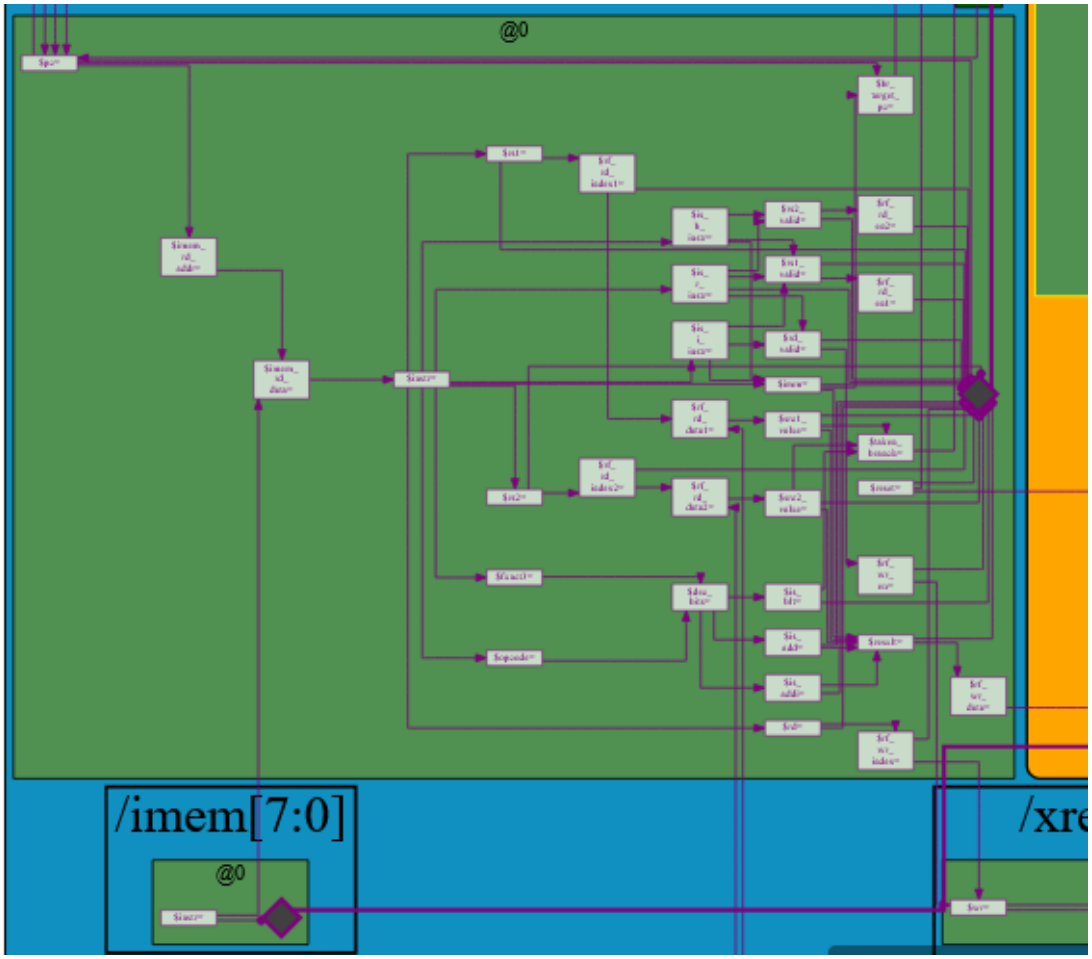


Zooming in -

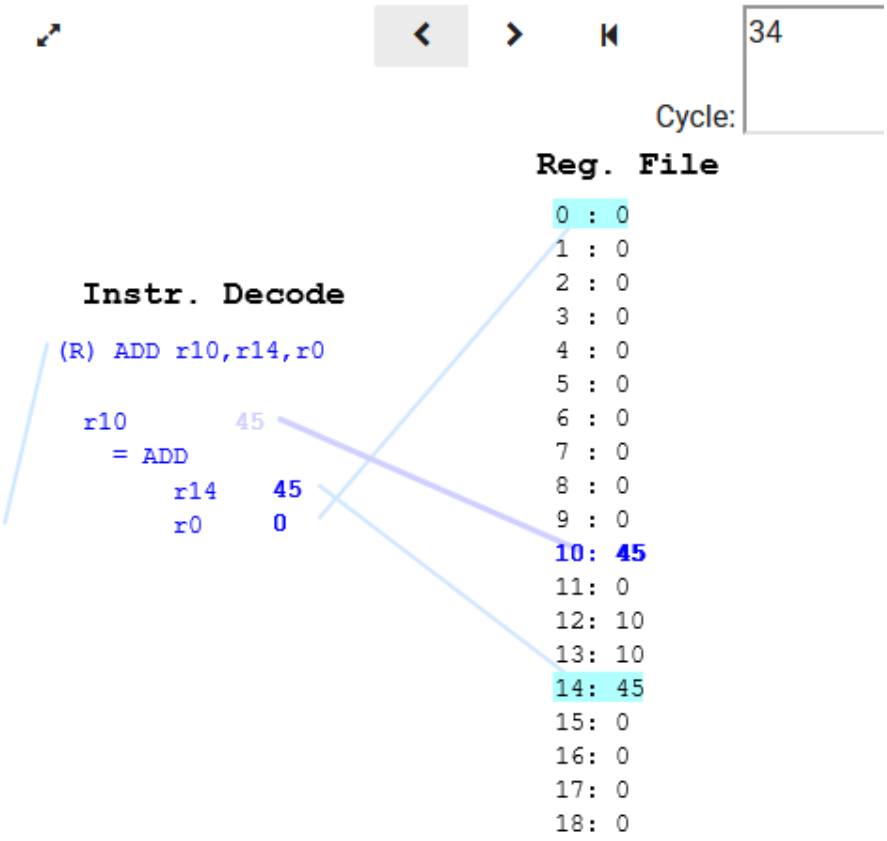

/xreg[31:0]
@0
$rd=
$wr
$ANY=...
$value
@1
/imem[7:0]
@0
$instr_
str=
$rd=
$ANY=...
$instr


$rf_
wr_
index=
/xreg[31:0]
@0
$wr=
$value=
@1

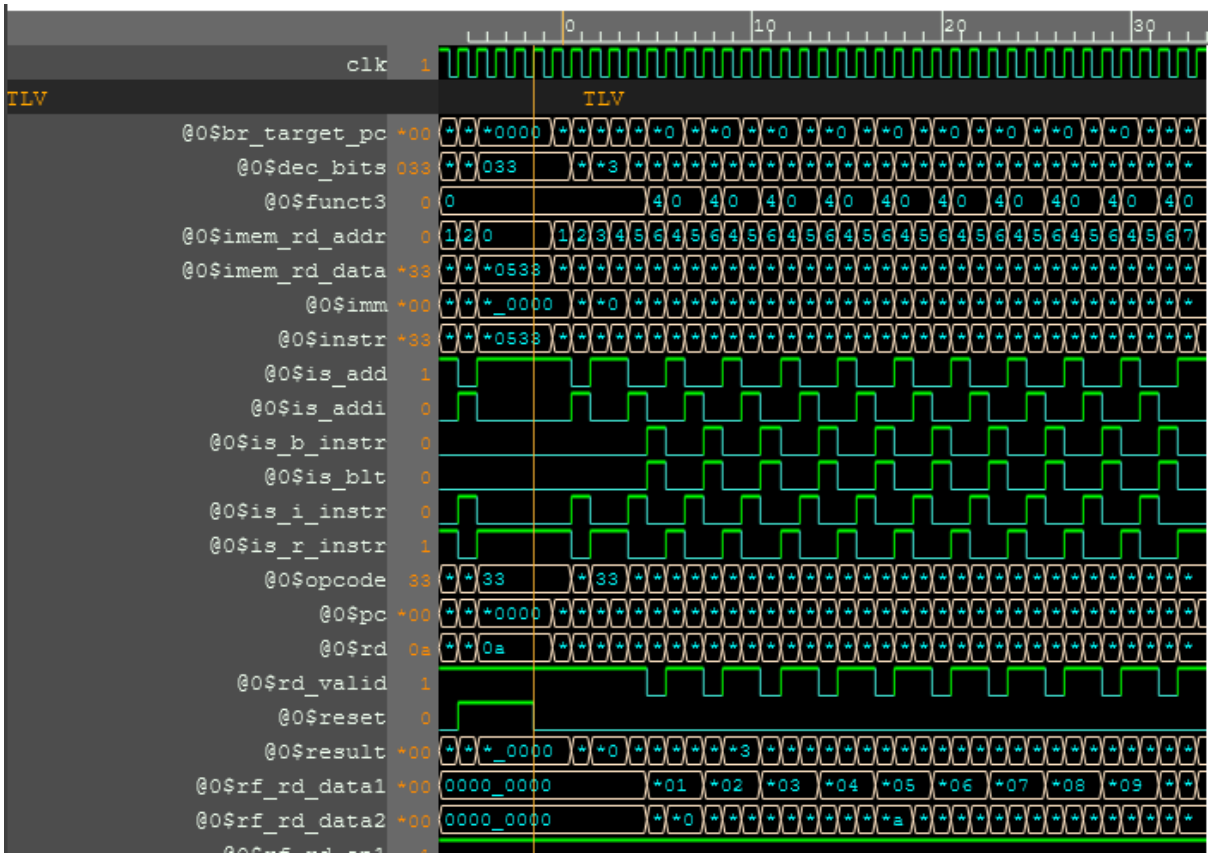

|for_viz_only
@0
$ANY=...

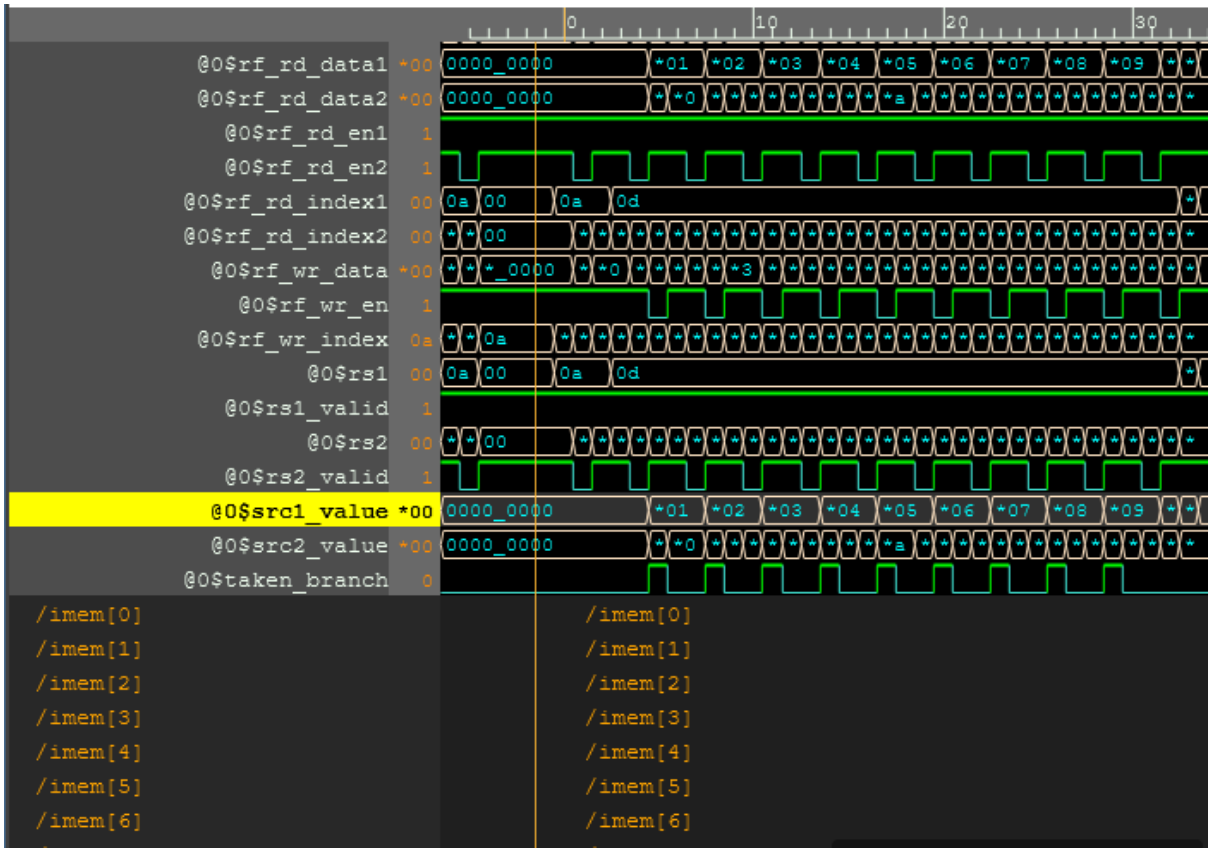

@0
/imem[7:0]
@0

The VIZ window:



The waveform window:





Output Log:

```
    SYSTEMC_INCLUDE     =
    SYSTEMC_LIBDIR      =
    VERILATOR_ROOT      =
    VERILATOR_BIN       =
[Inferior 1 (process 21473) exited normally]
No stack.
verilator --debug --debugi 0 --gdbbt --no-dump-tree --cc +librescan +libext+.sv --top-
No stack.
Starting Verilator 4.027 devel rev v4.026-36-g22088c9
Starting Verilator 4.027 devel rev v4.026-36-g22088c9
[Inferior 1 (process 21488) exited normally]
No stack.
cd obj_dir ; cp /src/sim_main.cpp . ; /usr/bin/make -j 3 -f /src/Makefile_obj make[1]:
g++  -I.  -MMD -I/usr/local/share/verilator/include -I/usr/local/share/verilator/inclu
g++  -I.  -MMD -I/usr/local/share/verilator/include -I/usr/local/share/verilator/inclu
/usr/bin/perl /usr/local/share/verilator/bin/verilator_includer -DVL_INCLUDE_OPT=inclu
/usr/bin/perl /usr/local/share/verilator/bin/verilator_includer -DVL_INCLUDE_OPT=inclu
g++  -I.  -MMD -I/usr/local/share/verilator/include -I/usr/local/share/verilator/inclu
g++  -I.  -MMD -I/usr/local/share/verilator/include -I/usr/local/share/verilator/inclu
ar -cr Vmakerchip__ALL.a Vmakerchip__ALLcls.o Vmakerchip__ALLsup.o
ranlib Vmakerchip__ALL.a
g++    -g sim_main.o verilated.o verilated_vcd_c.o Vmakerchip__ALL.a    -o simx -lm -l
make[1]: Leaving directory '/src/obj_dir' obj_dir/simx +verilator+error+limit+100
Enabling waves...
Simulation PASSED!!!
```