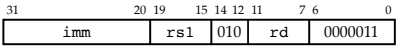
Adding LW Instruction

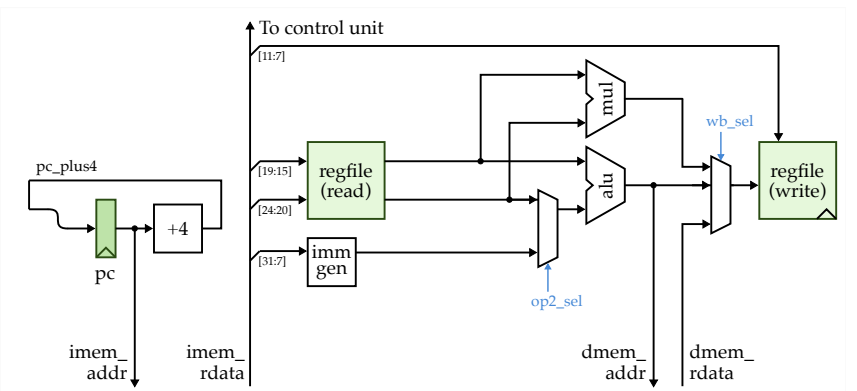
LW

`lw rd, imm(rs1)`

R[`rd`] ← M[R[`rs1`] + sext(`imm`)]
PC ← PC + 4



inst	op2 sel	wb sel	dmem val
`addi`	imm	alu	0
`add`	rf	alu	0
`mul`	x	mul	0
`lw`			



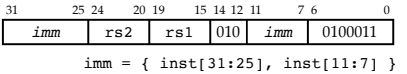
Adding SW Instruction

SW

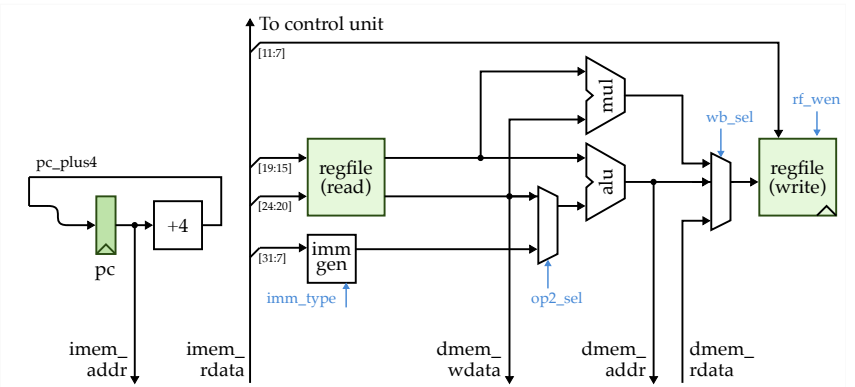
`sw rs2, imm(rs1)`

M[R[`rs1`] + sext(`imm`)] ← R[`rs2`]

PC ← PC + 4



inst	imm type	op2 sel	wb sel	rf wen	dmem val	dmem type
`addi`	I	imm	alu	1	0	x
`add`	x	rf	alu	1	0	x
`mul`	x	x	mul	1	0	x
`lw`						
`sw`						



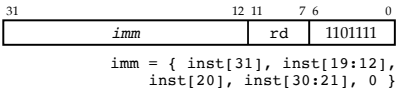
Adding JAL Instruction

JAL

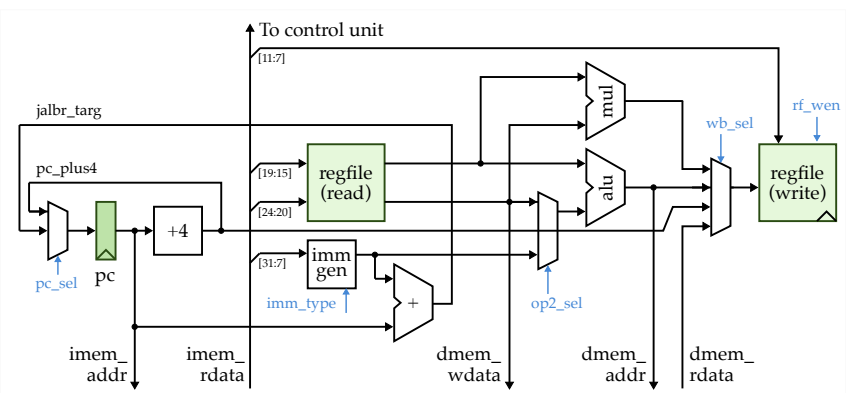
`jal rd, imm`

R[`rd`] ← PC + 4

PC ← PC + sext(`imm`)



inst	pc sel	imm type	op2 sel	wb sel	rf wen	dmem val	dmem type
`addi`	pc+4	I	imm	alu	1	0	x
`add`	pc+4	x	rf	alu	1	0	x
`mul`	pc+4	x	x	mul	1	0	x
`lw`							
`sw`							
`jal`							

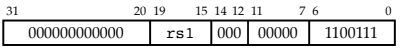


Adding JR Instruction

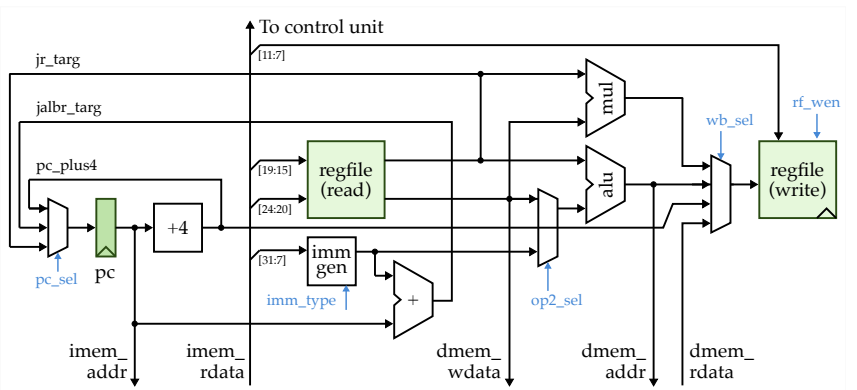
JR

`jr rs1`

PC ← R[`rs1`]



inst	pc sel	imm type	op2 sel	wb sel	rf wen	dmem val	dmem type
`addi`	pc+4	I	imm	alu	1	0	x
`add`	pc+4	x	rf	alu	1	0	x
`mul`	pc+4	x	x	mul	1	0	x
`lw`							
`sw`							
`jal`							
`jr`							

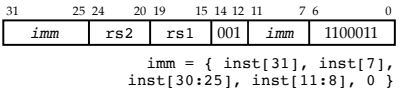


Adding BNE Instruction

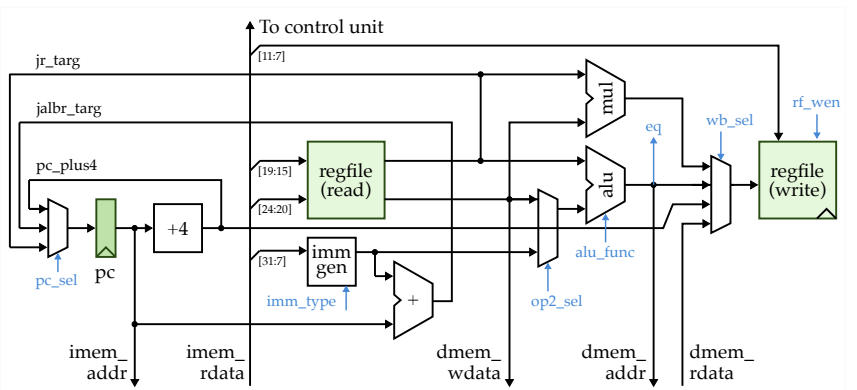
BNE

`bne rs1, rs2, imm`

if (R[`rs1`] != R[`rs2`]) PC ← PC + sext(`imm`)
else PC ← PC + 4



inst	pc sel	imm type	op2 sel	alu func	wb sel	rf wen	dmem val	dmem type
`addi`	pc+4	I	imm	add	alu	1	0	x
`add`	pc+4	x	rf	add	alu	1	0	x
`mul`	pc+4	x	x	x	mul	1	0	x
`lw`								
`sw`								
`jal`								
`jr`								
`bne`								

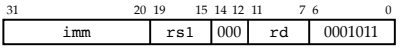


Add a New Auto-Incrementing Load Instruction

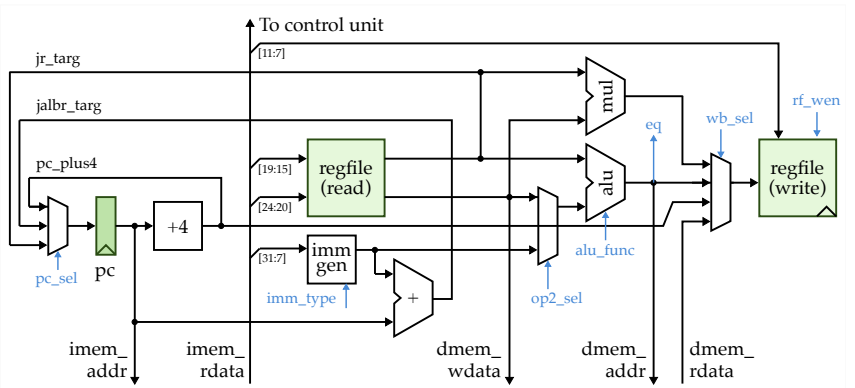
LW.AI

`lw.ai rd, imm(rs1)`

R[`rd`] ← M[R[`rs1`] + sext(`imm`)]
R[`rs1`] ← R[`rs1`] + 4
PC ← PC + 4



inst	pc sel	imm type	op2 sel	alu func	wb sel	rf wen	dmem val	dmem type
`lw.ai`								

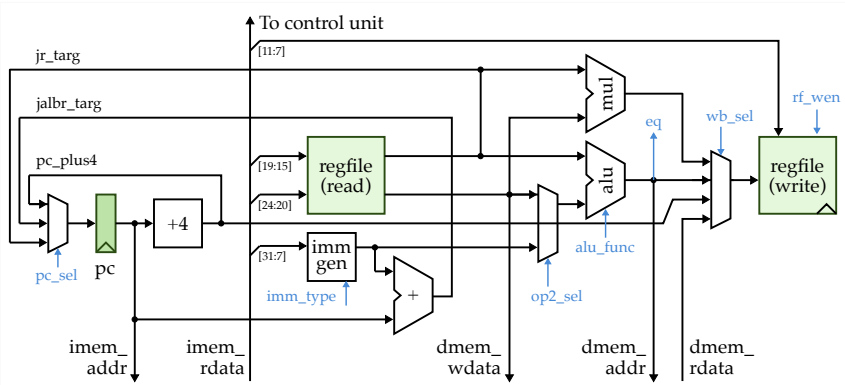


3. Analyzing Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycles}}$$

- Instructions / program depends on source code, compiler, ISA
- Cycles / instruction (CPI) depends on ISA, microarchitecture
- Time / cycle depends upon microarchitecture and implementation

Estimating minimum T_C (i.e., clock period, cycle time, or time/cycle)



	t_{pd}
32-bit 2-to-1 Mux	4τ
32-bit 4-to-1 Mux	8τ
32-bit Adder	60τ
32-bit ALU	64τ
32-bit Multiplier	100τ
32-bit +4 Unit	30τ
ImmGen Unit	12τ
32-bit Reg (t_{cq})	9τ
Register File Read	25τ
Memory Read	120τ

32-bit Reg (t_{setup})	10τ
Register File (t_{setup})	20τ
Memory (t_{setup})	120τ

Estimating execution time (i.e., time/program)

How long in units of τ will it take to execute the vector-vector add program assuming `n` is 64?

Pseudo-Code

```
for i in range(n):
  dest[i] = src0[i] + src1[i]
```

Assembly Code

```
 # addr(dest[i]):x1, addr(src0[i]):x2
 # addr(src1[i]):x3, n:x4
loop:
 lw    x5, 0(x1)
 lw    x6, 0(x2)
 add   x7, x5, x6
 sw    x7, 0(x3)
 addi  x1, x1, 4
 addi  x2, x2, 4
 addi  x3, x3, 4
 addi  x4, x4, -1
 bne   x4, x0, loop
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
`lw x5, 0(x1)`														
`lw x6, 0(x2)`														
`add x7, x5, x6`														
`sw x7, 0(x3)`														
`addi x1, x1, 4`														
`addi x2, x2, 4`														
`addi x3, x3, 4`														
`addi x4, x4, -1`														
`bne x4, x0, loop`														
`lw x5, 0(x1)`														
`lw x6, 0(x2)`														
`add x7, x5, x6`														
`sw x7, 0(x3)`														

Analyzing impact of adding a new auto-incrementing load instruction

- Step 0. Rewrite vector-vector add loop using `lw.ai`
- Step 1. Estimate new instructions/program
- Step 2. Estimate new cycles/instruction
- Step 3. Estimate new time/cycle
- Step 4. Estimate new time/program
- Step 5. Estimate new total area

Pseudo-Code

```
for i in range(n):
  dest[i] = src0[i] + src1[i]
```

Assembly Code

```
 # addr(dest[i]):x1, addr(src0[i]):x2
 # addr(src1[i]):x3, n:x4
loop:
 lw    x5, 0(x1)
 lw    x6, 0(x2)
 add   x7, x5, x6
 sw    x7, 0(x3)
 addi  x1, x1, 4
 addi  x2, x2, 4
 addi  x3, x3, 4
 addi  x4, x4, -1
 bne   x4, x0, loop
```