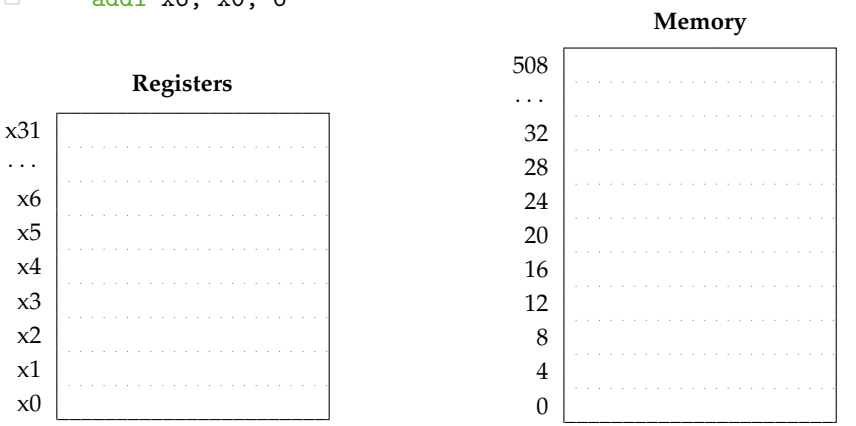
3.5. Control Dependencies

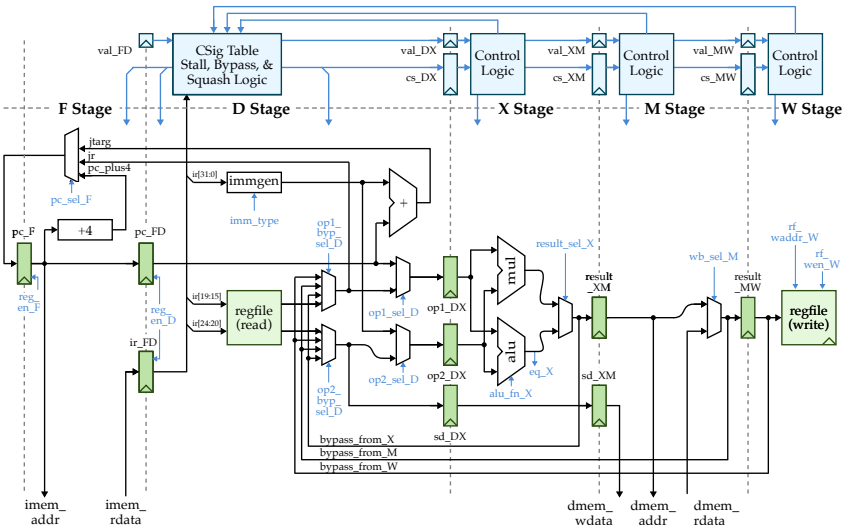
Control dependencies occur when whether or not an instruction should be executed depends on a control decision made by an earlier instruction. We use architectural dependency arrows to illustrate control dependencies in assembly code sequences.

```
    addi x1, x0, 1
    addi x2, x0, 2
    jal  x0, L1
    addi x3, x0, 3
    addi x4, x0, 4
L1: addi x5, x0, 5
    addi x6, x0, 6
```



- **Static instruction sequence** are instructions stored in memory
- **Dynamic instruction sequence** are instructions actually executed

Modifications to datapath/control to support jumps



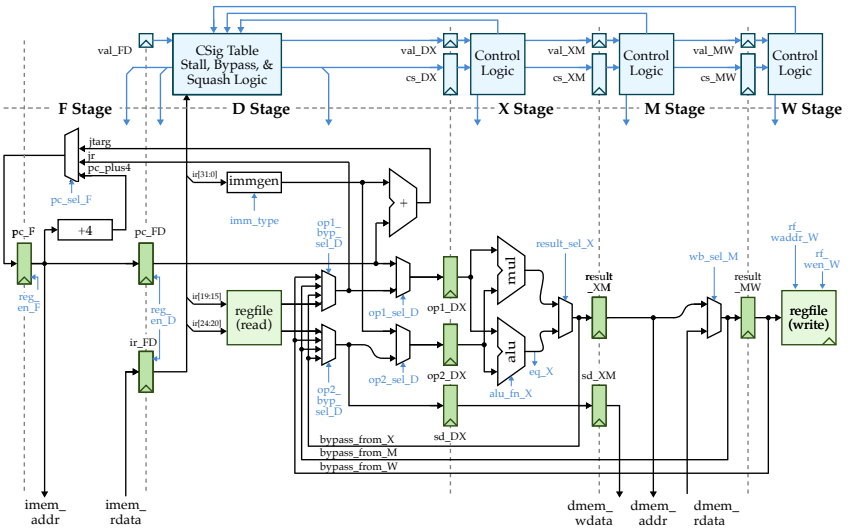
Using pipeline diagrams to illustrate control dependencies

We use microarchitectural dependency arrows to illustrate control dependencies on pipeline diagrams.

`addi x1, x0, 1`													
`addi x2, x0, 2`													
`jal x0, L1`													
`addi x5, x0, 5`													
`addi x6, x0, 6`													

Arrow points down means there is a control hazard!

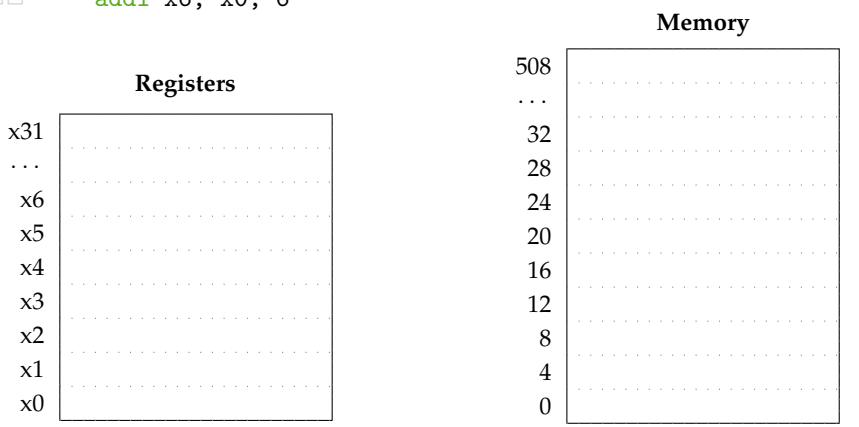
Using speculation to resolve control hazards



`addi x1, x0, 1`													
`addi x2, x0, 2`													
`jal x0, L1`													
`addi x3, x0, 3`													
`addi x5, x0, 5`													
`addi x6, x0, 6`													

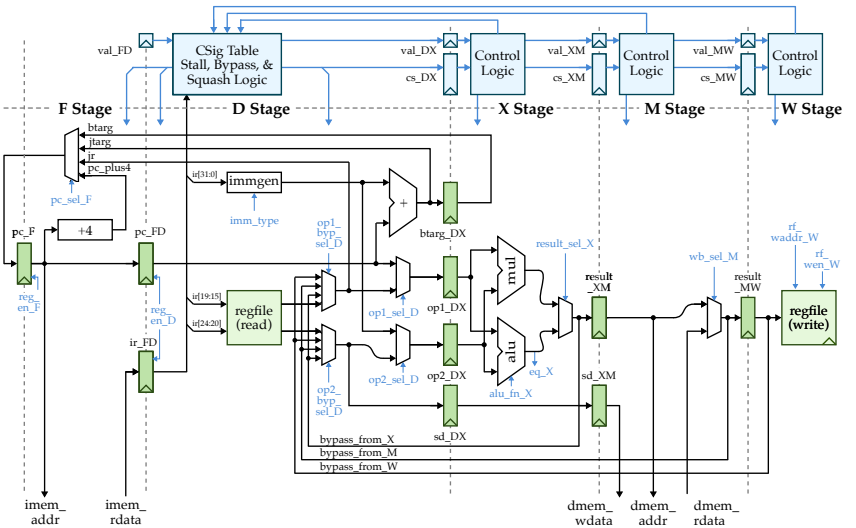
How should we handle control dependencies from conditional branches?

```
    addi x1, x0, 1
    addi x2, x0, 2
    bne  x1, x2, L1
    addi x3, x0, 3
    addi x4, x0, 4
L1: addi x5, x0, 5
    addi x6, x0, 6
```



- **Static instruction sequence** are instructions stored in memory
- **Dynamic instruction sequence** are instructions actually executed

Modifications to datapath/control to support branches

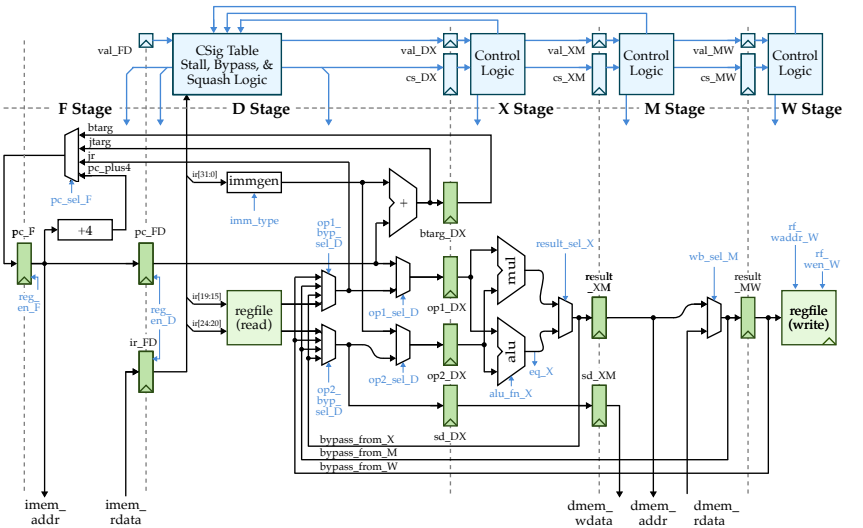


Using pipeline diagrams to illustrate control dependencies

`addi x1, x0, 1`													
`addi x2, x0, 2`													
`bne x1, x2, L1`													
`addi x5, x0, 5`													
`addi x6, x0, 6`													

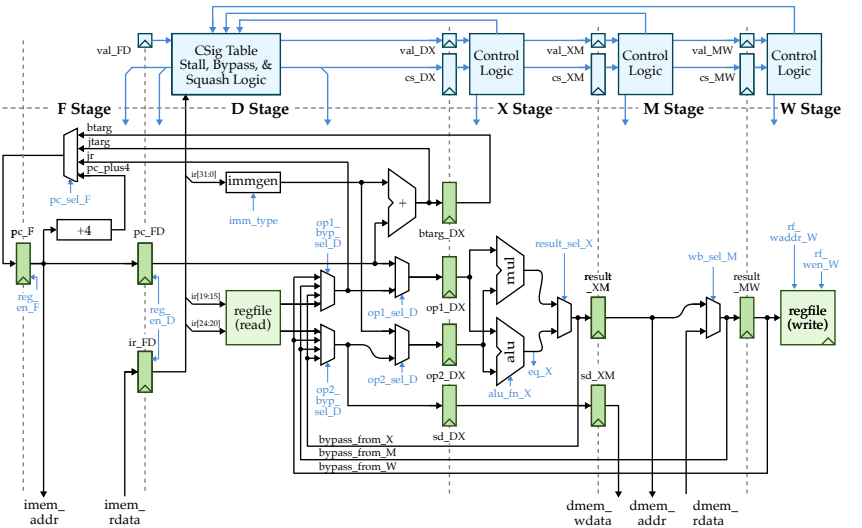
Arrow points backwards means there is a control hazard!

Using speculation to resolve control hazards



`addi x1, x0, 1`													
`addi x2, x0, 2`													
`bne x1, x2, L1`													
`addi x3, x0, 3`													
`addi x4, x0, 4`													
`addi x5, x0, 5`													
`addi x6, x0, 6`													

Deriving the squash signals

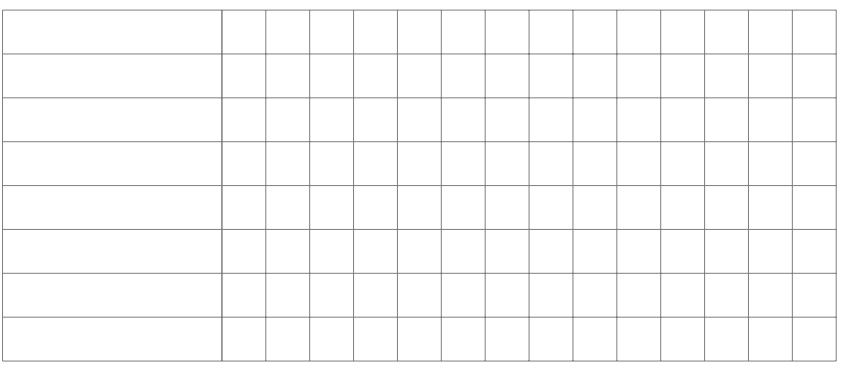



```
squash_D = val_X && (op_X == bne) && !eq_X
squash_F = squash_D || (val_D && ((op_D == jal) || (op_D == jr)))
```

Important: PC select logic must give priority to older instructions (i.e., prioritize branches over jumps)! *Good exam question?*

Draw the pipeline diagram assuming control hazards are resolved with hardware speculation

```
    addi x1, x0, 0
    bne  x1, x0, L1
    addi x2, x0, 1
    addi x3, x0, 1
L1: bne  x2, x0, L2
    addi x4, x0, 1
    addi x5, x0, 1
L2: addi x6, x0, 1
```

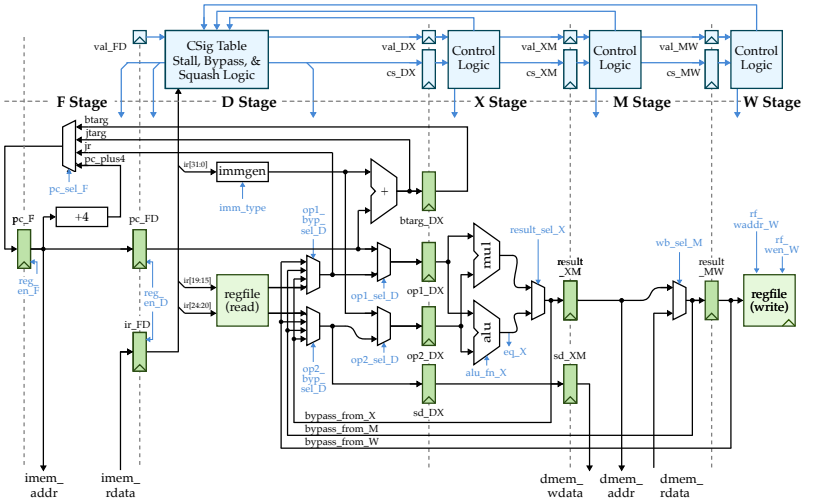


3.6. Analyzing Performance

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

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

Estimating minimum clock period (cycle time)



	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

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

Pseudo-Code

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

Assembly Code

```
 # addr(src0[i]):x1, addr(src1[i]):x2
 # addr(dest[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
```

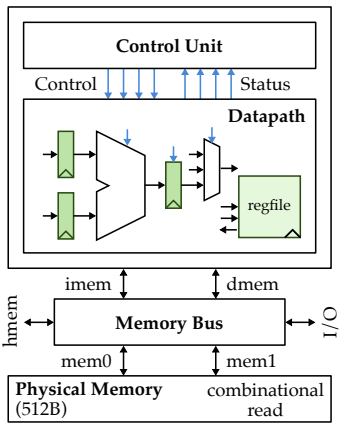
`lw`																			
`lw`																			
`add`																			
`sw`																			
`addi`																			
`addi`																			
`addi`																			
`addi`																			
`bne`																			
`opA`																			
`opB`																			
`lw`																			
`lw`																			

Results for vector-vector-add micro-benchmark

Microarchitecture	Inst/Prog	Cycle/Inst	Time/Cycle	Exec Time
Single-Cycle	576	1.0	366 τ	211 kτ
15-State Multi-Cycle	576	4.2	151 τ	367 kτ
63-State Multi-Cycle	576	6.7	231 τ	891 kτ
2-Stage Pipelined	576	1.1	223 τ	143 kτ
5-Stage Pipelined	576			

4. TinyRV1 Embedded System

- Memory-mapped input/output enables pipelined processor to external devices by reading/writing memory addresses
- Supporting the memory wait signal requires additional stall logic
  - Stall logic should *only* stall earlier stages, never later stages
  - Instruction memory wait needs to stall the F stage
  - Data memory wait needs to stall the F, D, and X stages



- Now possible to support physical memory with sequential read (send memory read request in stage before stage requiring read data)