Single-cycle MIPS processor

1 exe/clk → future more

1. state holding eles

Prog. counter (register) holds addr of curr exe inst.

Inst. mem. read-only for prog. inst.

add → 32-bit inst.

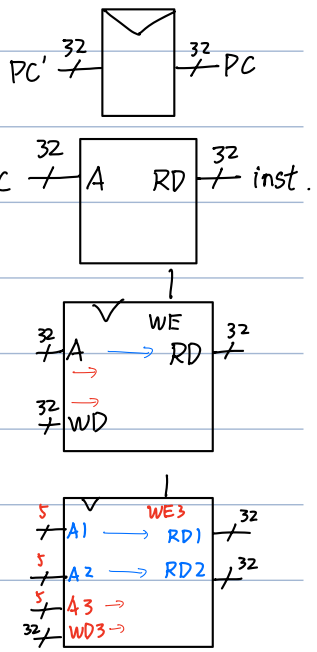
Data mem. place for data during exe

1R $A^{(32)} \rightarrow RD^{(32)}$ 1W $A^{(32)} + WD^{(32)}$

Reg. file (new!) holds 32 mips registers

2R $A1^{(5)} \rightarrow RD1^{(32)}$ $A2^{(5)} \rightarrow RD2^{(32)}$ 1W $A3^{(5)} + WD^{(32)}$

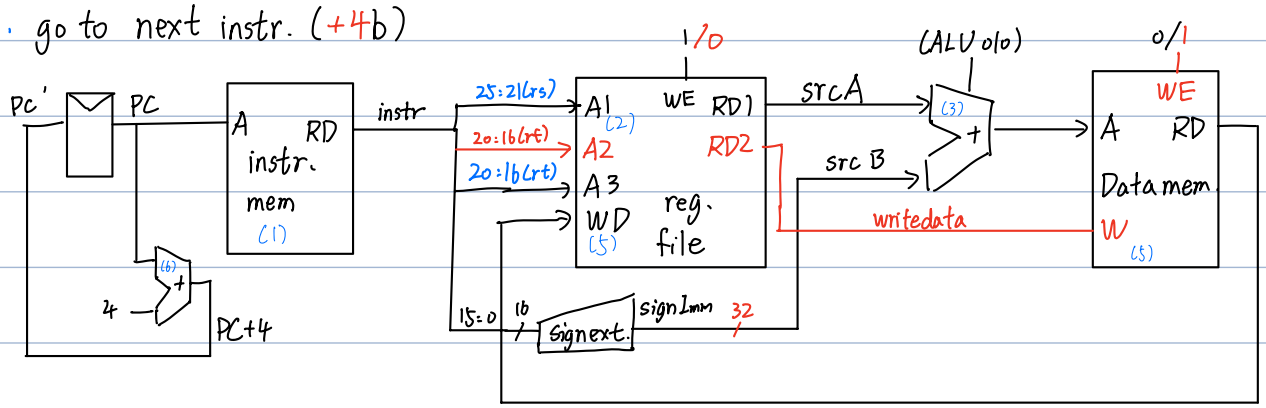
↳ 2 for R-type inst., sw, beq (2 source reg.)



2. build path to exe 1 inst.

lw op(6) rs(5) rt(5) imm(16)

1. Fetch instr. get out 32b
2. read rs (25:21)
3. add offset to imm
4. compute mem. addr (base + offset)
5. read dat. from mem and write to regfile.
6. go to next instr. (+4b)



3. extend datapath for more inst. (what's same/diff?)

E. sw → now write to data mem. op(6) rs(5) rt(5) imm(16) read rs, rt simul

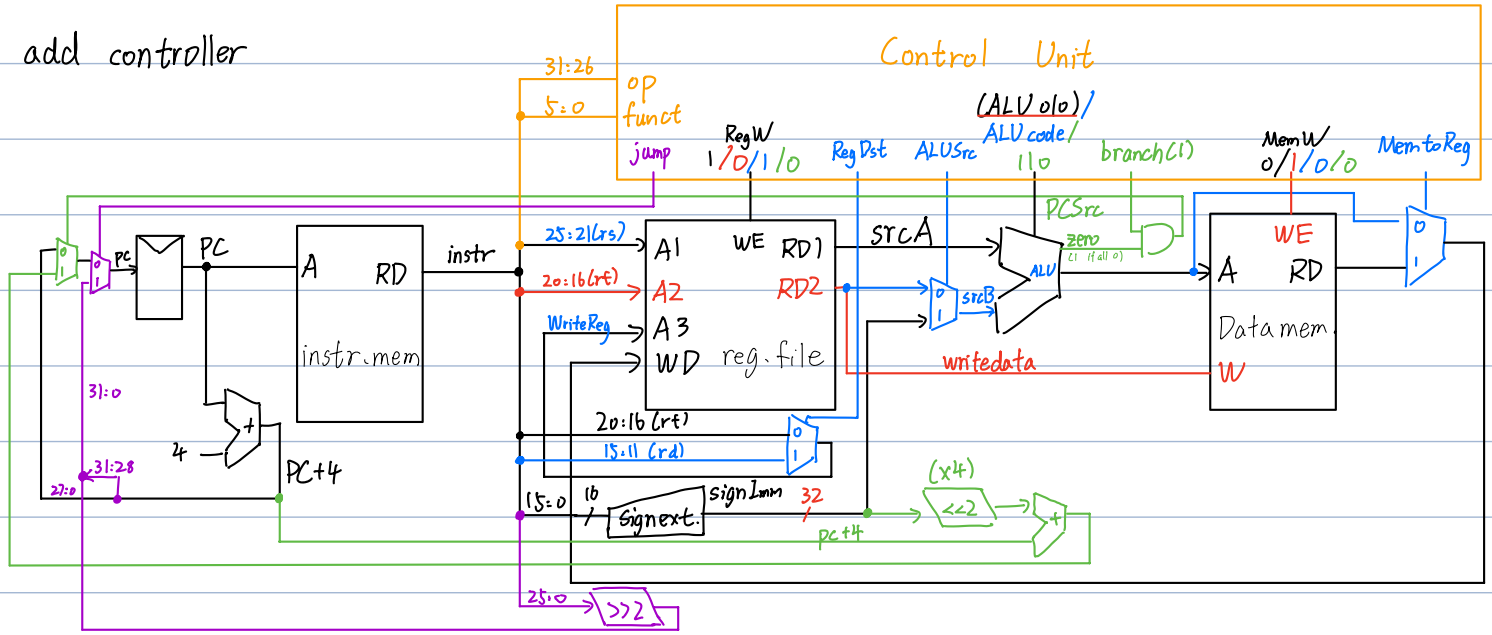
E. R-type class (diff: ALU op, same: source reg. reading from)

op(6) rs(5) rt(5) rd(5) shamt(5) funct(6)

E. beq op(6) rs(5) rt(5) imm(16) (offset from fallthru, not curr.) Reuse addr. calc and sgn extend.

+4 +4n

4. add controller



1. Main decoder Opcode 5:0

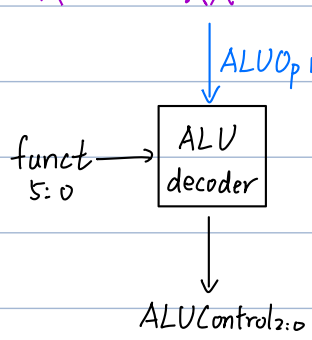
Inst	opcode	RegW	RegDst (0: rt, write to 1: rd)	ALUSrc (0: RD2, 1: imm)	Branch	MemW	Mem2Reg (0: ALU result, 1: memory content)	ALUop	j
R-type	000000	1	1	0	0	0	0	10 (check funct)	0
lw	100011	1	0	1	0	0	1	00 (+)	0
sw	101011	0	X	1	0	1	X	00 (+)	0
beq	000100	0	X	0	1	0	X	01 (-)	0
E. addiu	001000	1	0	1	0	0	0	00	0
E. j (<< 2, pad top 4 of prev. addr.)	000010	0 (just don't write)	X	X	X	0	X	XX	1

2. ALU decoder Funct 5:0 (if needed)

ALUOp	funct	ALUControl	
00	xxxxxx	010	Add
01	xxxxxx	110	Subtract
10	100001	010	Add
10	100011	110	Subtract
10	100100	000	And
10	100101	001	Or
10	101010	111	Slt

I: lw/sw, beq

R



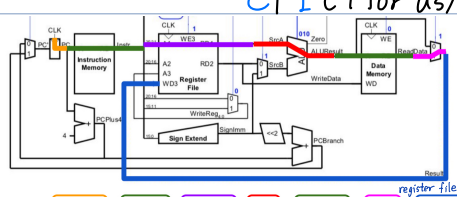
Performance

$$\frac{\text{sec}}{\text{prog}} = \frac{\text{instr.}}{\text{prog}} \cdot \frac{\text{clk cycles}}{\text{instr.}} \cdot \frac{\text{sec}}{\text{clk cycle}}$$

(clk cycles/instr.: diff. for pipeline)

CPI (1 for us) ↳ clk period (crit. path, ---)

E. longest: lw



T_C = $t_{pcq\text{-}PC}$ + $t_{mem\text{-}I}$ + t_{RFread} + t_{ALU} + $t_{mem\text{-}D}$ + t_{mux} + $t_{RFsetup}$ (register file setup time for FFs)

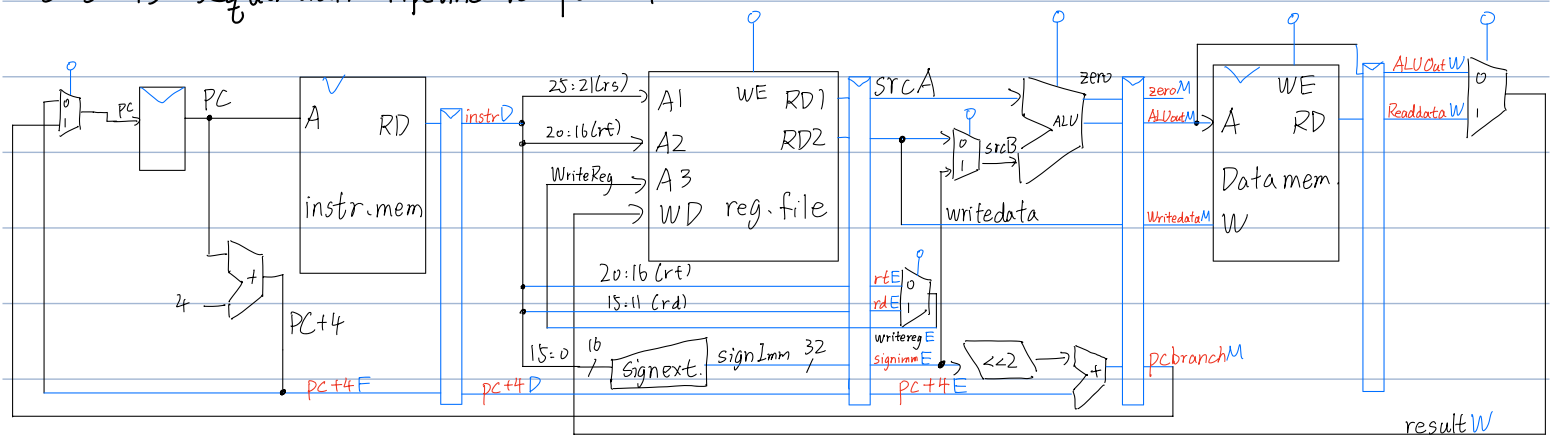
Add all delays

Take longest

11/7

Pipeline (not 1-cycle)

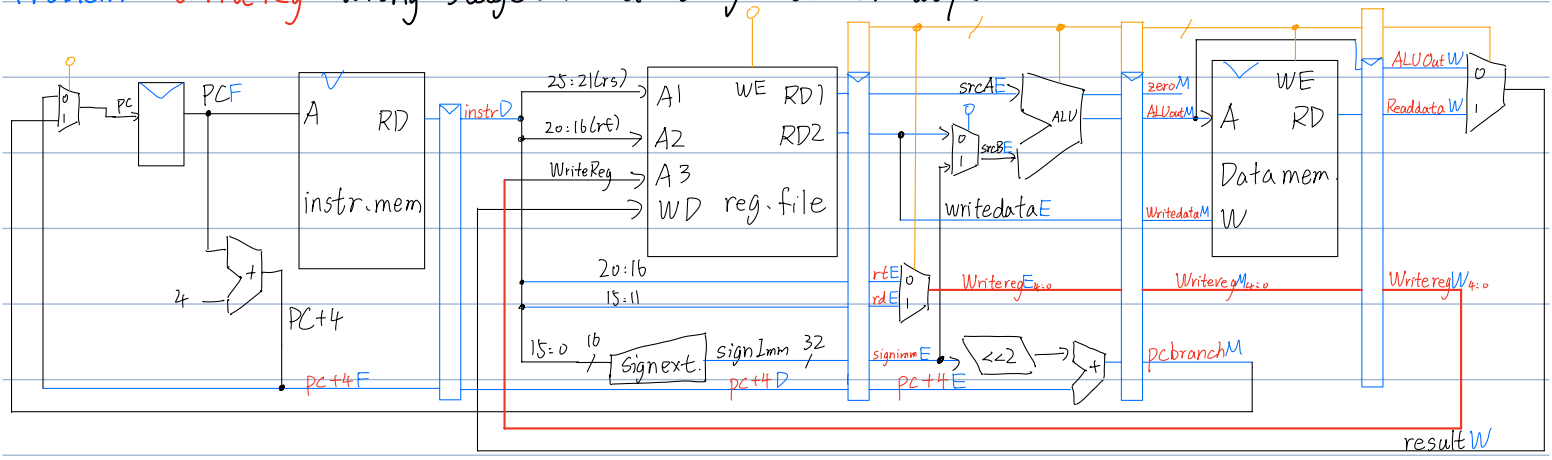
S-S is sequential. Pipeline is parallel



5	4	3	2	1
Fetch from instr. mem	Decode rs & control sig.	Execute: ALU	Memory: R/W	Writeback
IM	RF	+	DM DM	RF
F	D	E/X	M	W

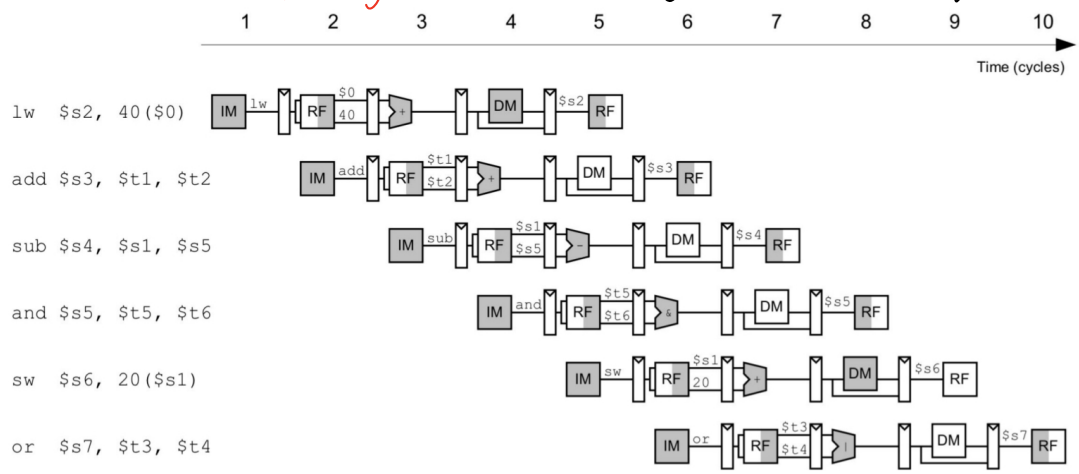
Use pipereg to store partial results (ana shift register)

Problem Writereg wrong stage! Need to go all the way!



Control: same unit. Each pipereg stores ctrl. signals of its stage

Shorthand



Hazards Result of instr. needed before it's produced → Add hazard unit

1. data result needed before available

E. add \$s0, \$s2, \$s3

and \$t0, \$s0, \$s1 → not yet ready! OK if same cycle

2. ctrl next addr. not known at fetch E. beq.

Sln. 1. Insert nops to delay later instr. Can put useful work there.

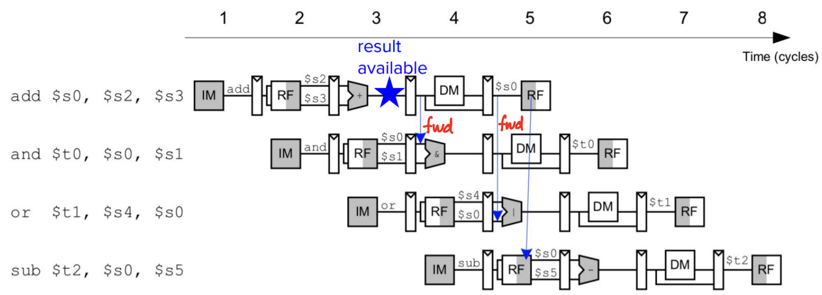
:(bad abstraction, what if not run on piped?

2. Forwarding (bypassing)

pick value before write back

bypassing register

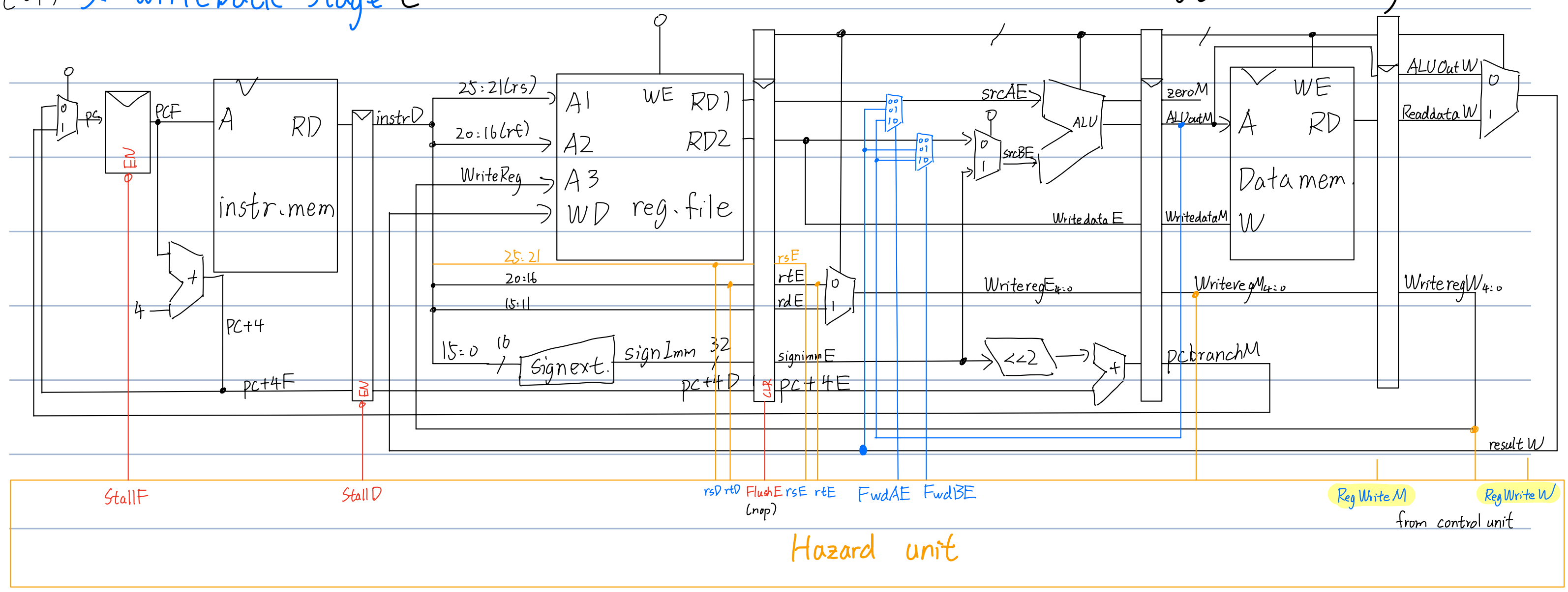
ALU input can come from:



(00) 1. register file (general)

M→E
(10) 2. memory stage (instr. in E reads a reg. that the instr. in M will write)

W→E
(01) 3. writeback stage (~ W ~)



FwdAE

10 M→E if (RegWriteM) and (rsE ≠ 0) and (rsE = WriteReg M)
(writes to reg. file; not \$0; E source = M dest)

01 W→E if (RegWrite W) and (~) and (~ W)

FwdBE

10 ~ rtE ~ rtE ~

01 ~

X if producer is a load

Can't fwd, must stall!

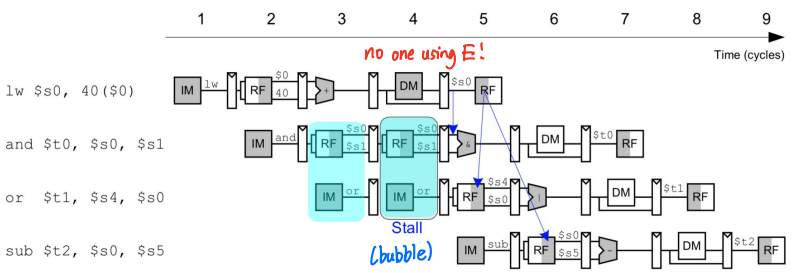
Detect at runtime, insert nop.

Stall tells pipeline and stuff behind to wait!

disable pipe reg and clear E pipe reg → nop

StallF, StallD, FlushE — lw? — lw dest reg = either D src. reg?

lwstall = MemToReg E and ((rsD = rtE) or (rtD = rtE))



Control hazard: branch

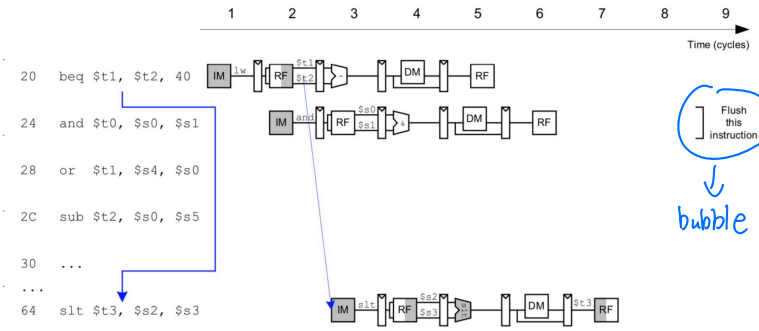
1. Early branch resolution → decide in D, before E

New hazard: may need value M→D. No need for E→D

FwdAD M→D if (RegWrite M) and (rsD ≠ 0) and (rsD = WriteReg M) (usual)

instr. in M writes to Reg? — Did I just read a reg? — D src = M dest

FwdBD ——— (t ~ t ———



Flush this instruction → bubble

C2 resolves branch

instr. fetched in C3.

speculate whether branch is taken (guess just +4 no)

✓: good, already in pipeline

X: flush wrong instr., zero reg. → bubble

branch Stall

cycle	1	2	3	4	5
lw $t0, 4($0)	F	D	E	M	W
add $t1, $t1, $t2		F	D	E	M
beq $t0, $t1, target			F	D	E

$t0, t1 not available. Need stall

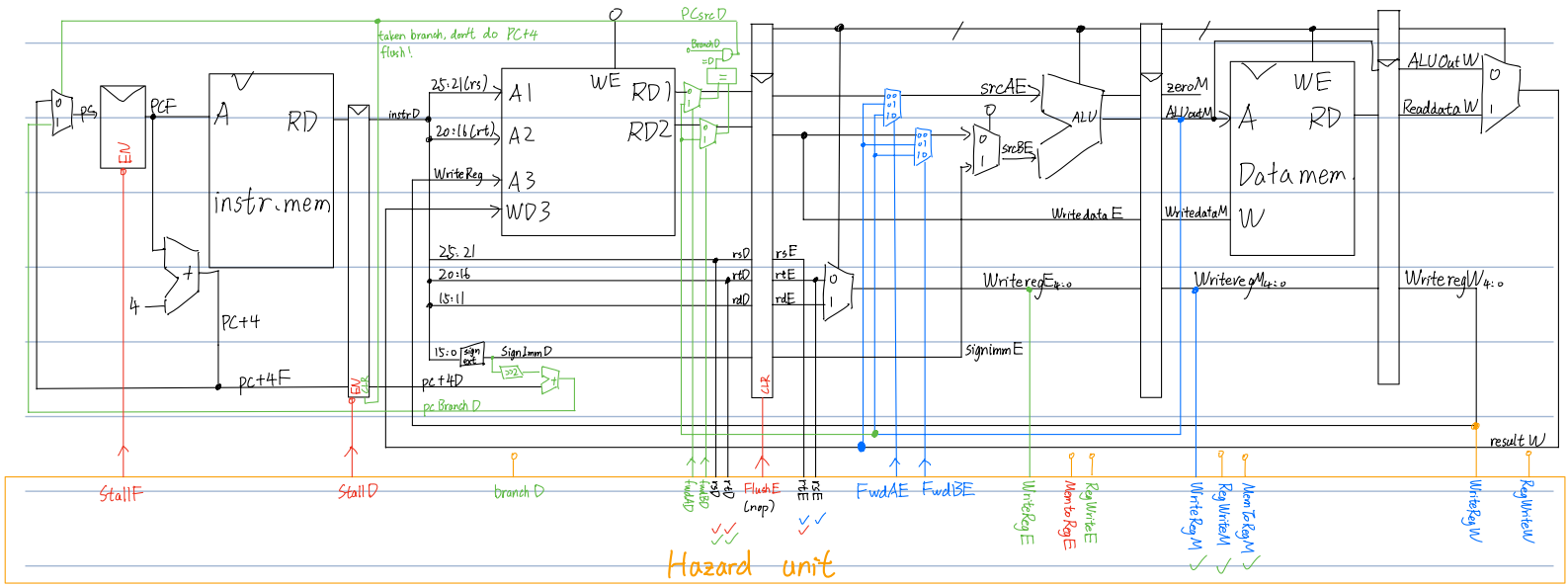
StallF, StallD, StallE = lwstall + branchstall (if instr. in F. produce)

? branch in decode — ? write to mem — is dest reg. = either branch source?

branchstall = Branch D and RegWrite E and ((WriteReg E = rsD) or (WriteReg E = rtD))

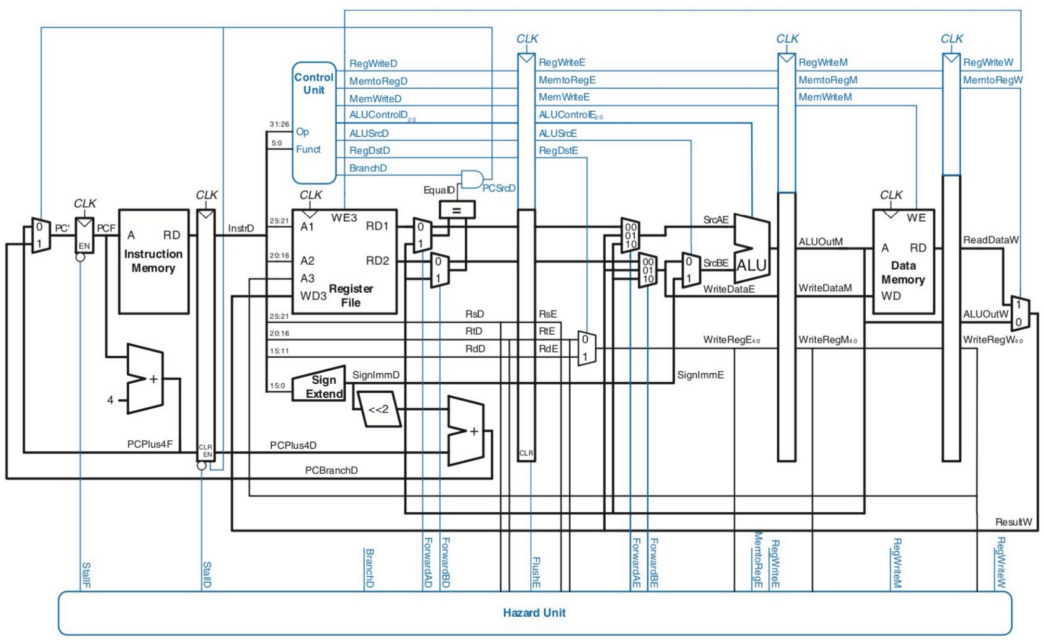
? is load in M?

. OR ~ MemToReg M ——— M ~ ——— M ~



Summary 4 types

consumer \ producer	lw (M) later	lw
beq	1	2
beq (D) earlier	3	4



1. ✓✓, fwd M → E

F D E M W — detect → fwd

beq lw — M-E fwd

2. lw

beq lw

beq X lw — lwstall

beq X lw (only available here) — W-E fwd

3. beq

beq lw — branchstall

beq X lw — M-D fwd

4. XX

beq lw — branchstall and lwstall

beq X lw (load in M → Bstall) — branchstall

beq X X lw — <register file will handle it>

Performance Analysis

CPI (cycles-per-instr.), ideal = 1, > 1 for bubbles

E. 54% R, 25% load, 10% store, 11% branch (dynamic exe count, loops --- included)

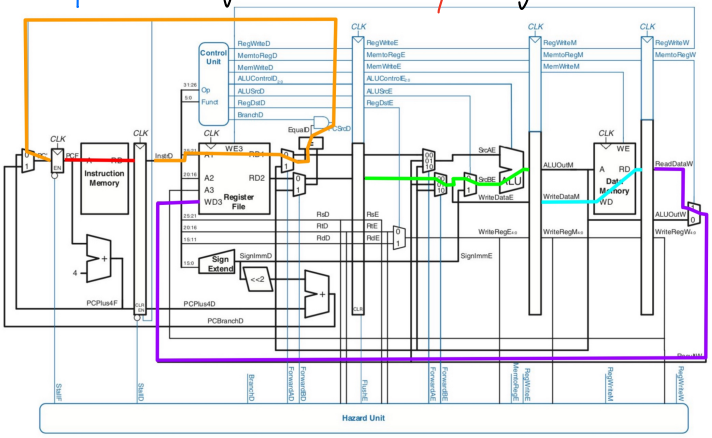
if 40% loads used by nxt. instr.; 25% branches taken

load CPI $0.4 \cdot 2 + 0.6 \cdot 1 = 1.4$

b CPI (flush) $0.25 \cdot 2 + 0.75 \cdot 1 = 1.25$

$\Sigma = 0.54 \cdot 1 + 0.25 \cdot 1.4 + 0.1 \cdot 1 + 0.11 \cdot 1.25 = 1.1275$!

Crit path: longest thru any stage



Element		Delay
Register clk-to-Q	t_{pcq}	30 ps
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory Read	$t_{memread}$	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20
Equality	t_{eq}	40
AND gate	t_{AND}	15
Memory Write	$t_{memwrite}$	220
Register file write	$t_{RFwrite}$	100

$$T_c = \max \left\{ \begin{array}{ll} t_{pcq} + t_{memread} + t_{setup} & \textit{Fetch} \\ 2(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup}) & \textit{Decode} \\ t_{pcq} + t_{mux} + t_{mux} + t_{ALU} + t_{setup} & \textit{Execute} \\ t_{pcq} + t_{memwrite} + t_{setup} & \textit{Memory} \\ 2(t_{pcq} + t_{mux} + t_{RFwrite}) & \textit{Writeback} \end{array} \right\}$$

$$= 2(150 + 25 + 40 + 15 + 25 + 20)\ \text{ps} = 550\ \text{ps} \qquad \textit{Decode}$$

CPI ↑, delay ↓ (chopped 5 ports!), product ↓