1. pipe hazards
14. memory, cache

Compiler optimizations, Avoid load-use stalls

- Move instruction between
  "Fill load-delay slot"

```
lw $1, offset($base)
add $3, $1, $2
and $4, $4, 0
```

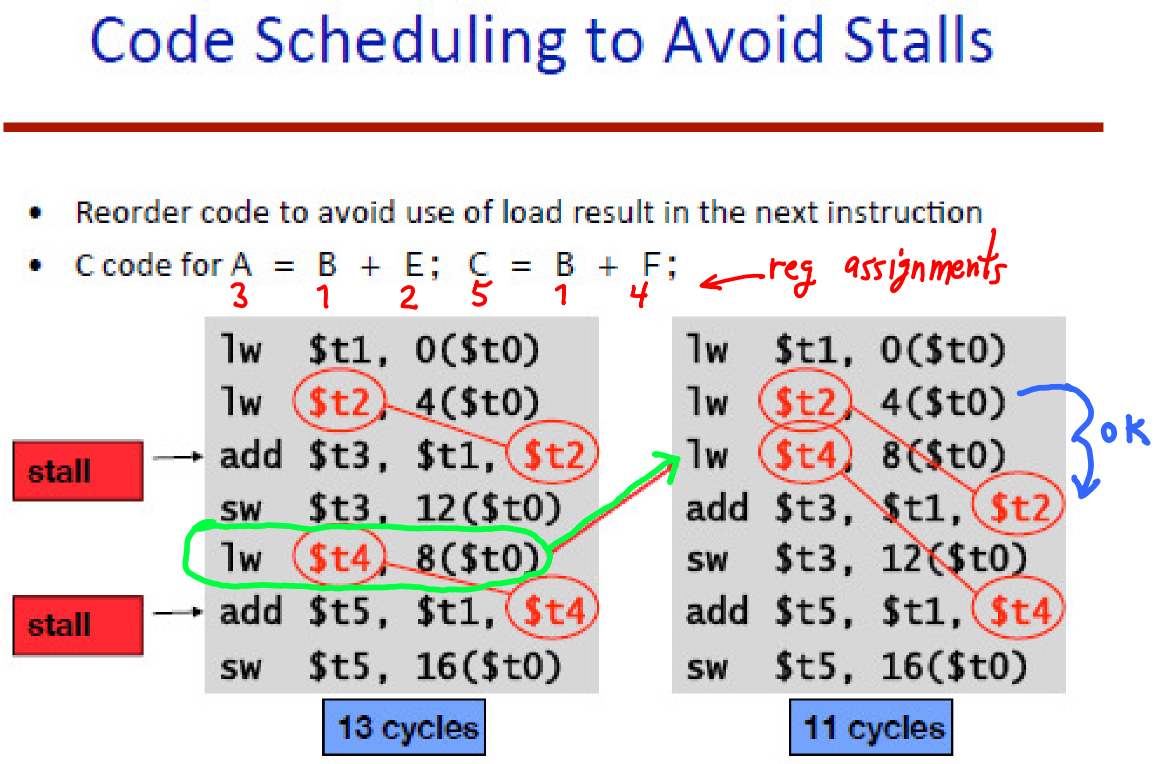
OR from before lw

E.G.

OK, if
- can find instruction w/o dependencies

- Let hardware insert Nop

- Compiler fill w/ nop
  HW doesn't have load-use detection



Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for A = B + E; C = B + F; ← reg assignments

$A=3,\ B=1,\ E=2;\ C=5,\ B=1,\ F=4$

```
        lw   $t1, 0($t0)
        lw   $t2, 4($t0)
stall → add  $t3, $t1, $t2
        sw   $t3, 12($t0)
        lw   $t4, 8($t0)
stall → add  $t5, $t1, $t4
        sw   $t5, 16($t0)
```

13 cycles

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
lw   $t4, 8($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

ok

11 cycles

pipe-fill: 5 stages ⇒ 4 nops
execution: 7 instructions
load-use stalls: 2 bubbles

13 ticks

$$\begin{array}{r} 4 \\ +\ 7 \\ \hline 11 \text{ ticks} \end{array}$$

$S = 13/11 = 1\ 2/11$

$\approx 15\%$

Control Hazards: BR

On clock tick:

1. Instructions written to next pipe stage reg
2. BR target address written to PC

BR L

SUB

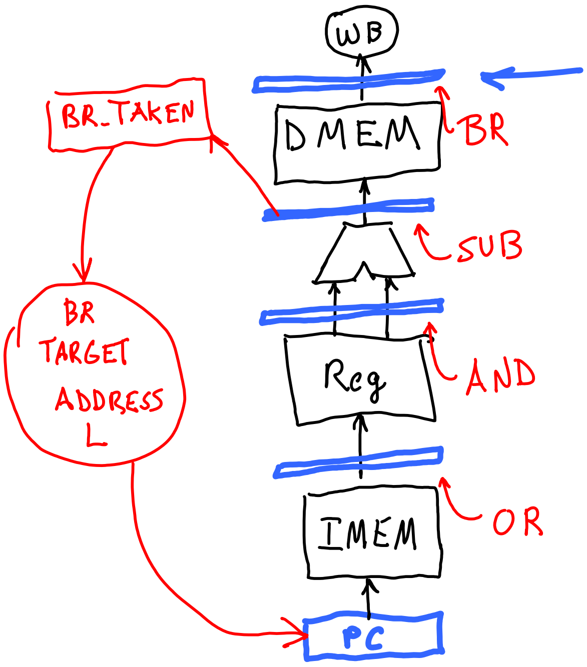
AND

OR

L:

ADD

do not execute? stall?

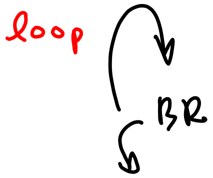


Pipe stage Reg

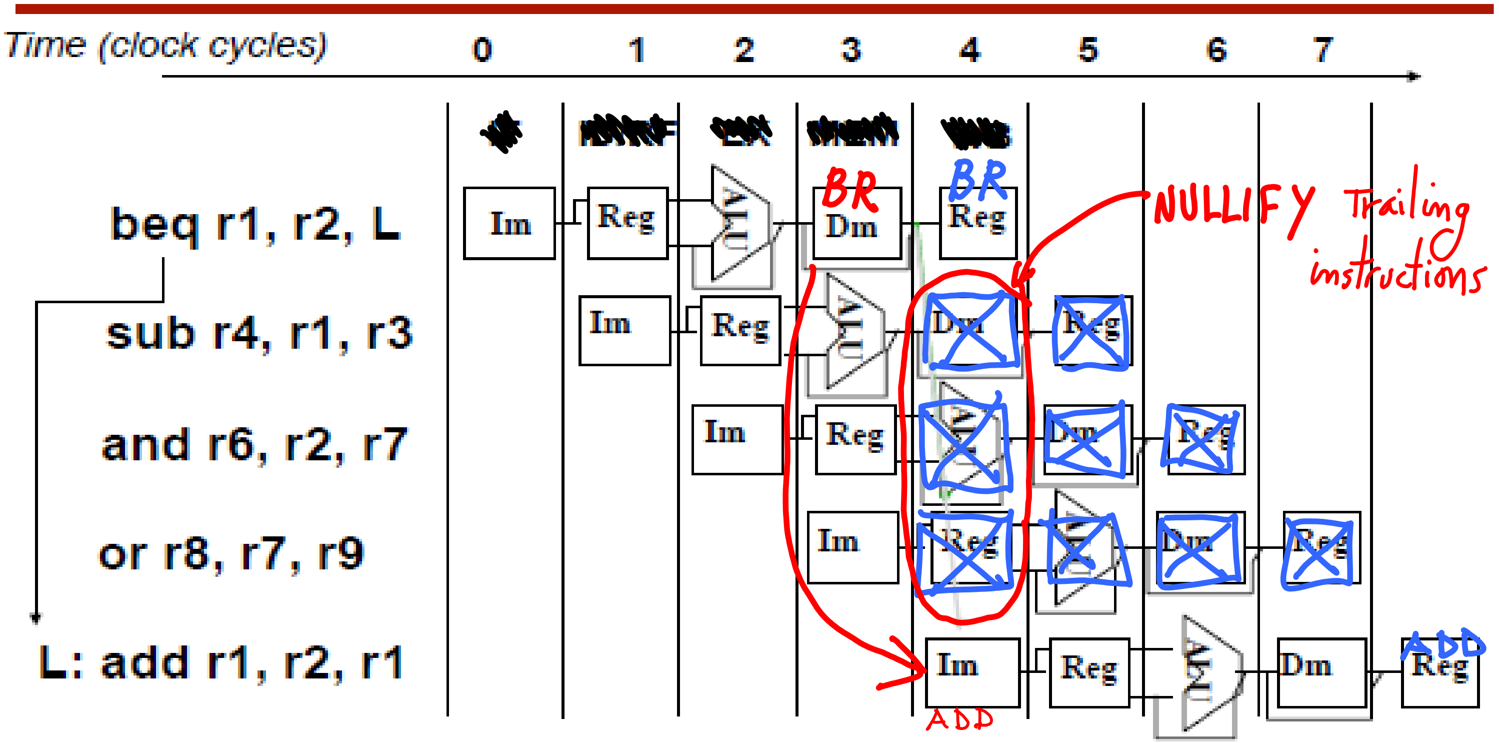
BR target feedback is a version of register forwarding to PC.

PC is a version of pipe stage register.

Overwrite NOPs To Pipe stages?



Branch Control Hazard



beq r1, r2, L

sub r4, r1, r3

and r6, r2, r7

or r8, r7, r9

L: add r1, r2, r1

CHOICES

Nulls : worry about data forwarding?

1• ~~Stall~~ on branches   EASY

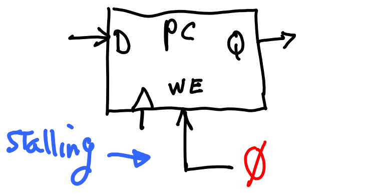
- Wait until you know the answer (branch CPI becomes ~~3~~ 4) ALL
- Control inserted in ID stage of the pipeline

→ STALL FETCHES, INSERT NULLS at ID

2• Predict not-taken   HARDER $CPI_{BR} = 4$ if Taken, 1 if not Taken

- Assume branch is not taken and continue with PC+4
- If branch is actually not taken, all is good
- Otherwise, nullify misfetched instructions and restart from PC+4+offset

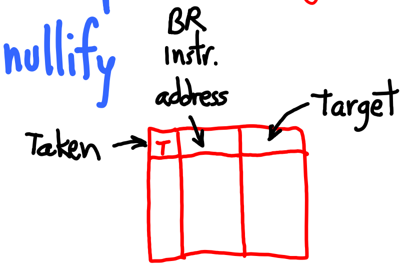
→ add Nullify HW To ID, REG EX



3• Predict taken   Can we do this? How?

- Assume branch is taken
- More complex, cause you also need to predict PC+4+offset ?
- Still need to nullify instructions if assumption is wrong

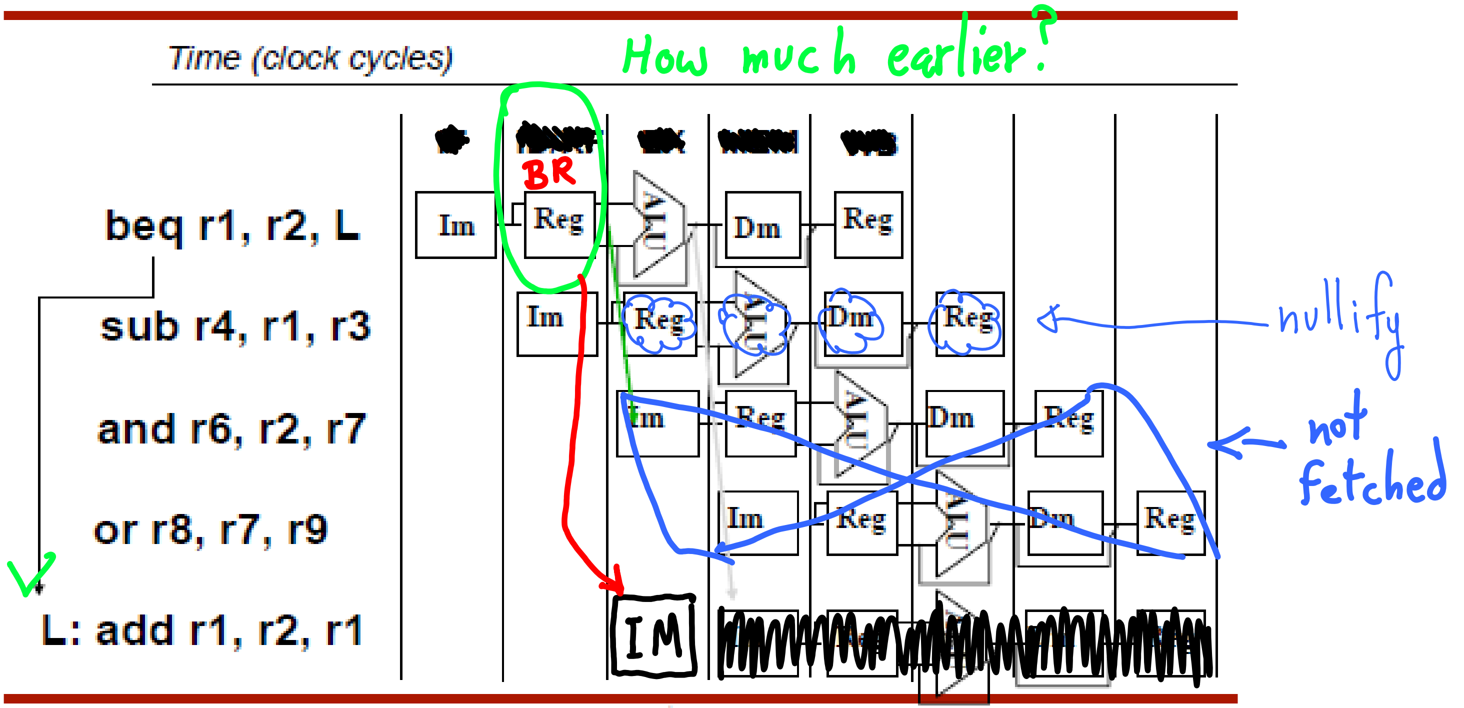
→ Pre-compute BR Target? + nullify



- Most machines do some type of prediction (taken, not-taken)

Can We Execute Branch Earlier?

How much earlier?



beq r1, r2, L

sub r4, r1, r3

and r6, r2, r7

or r8, r7, r9

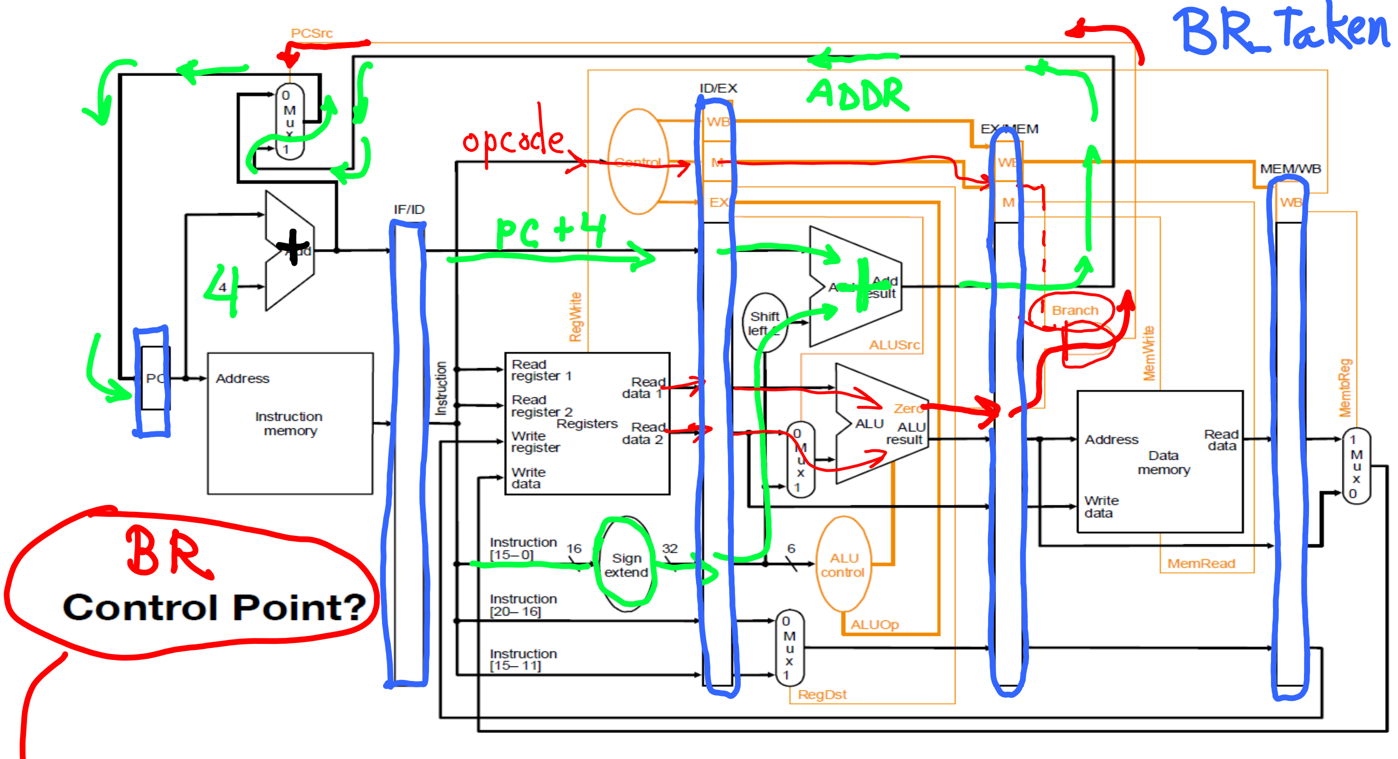
L: add r1, r2, r1

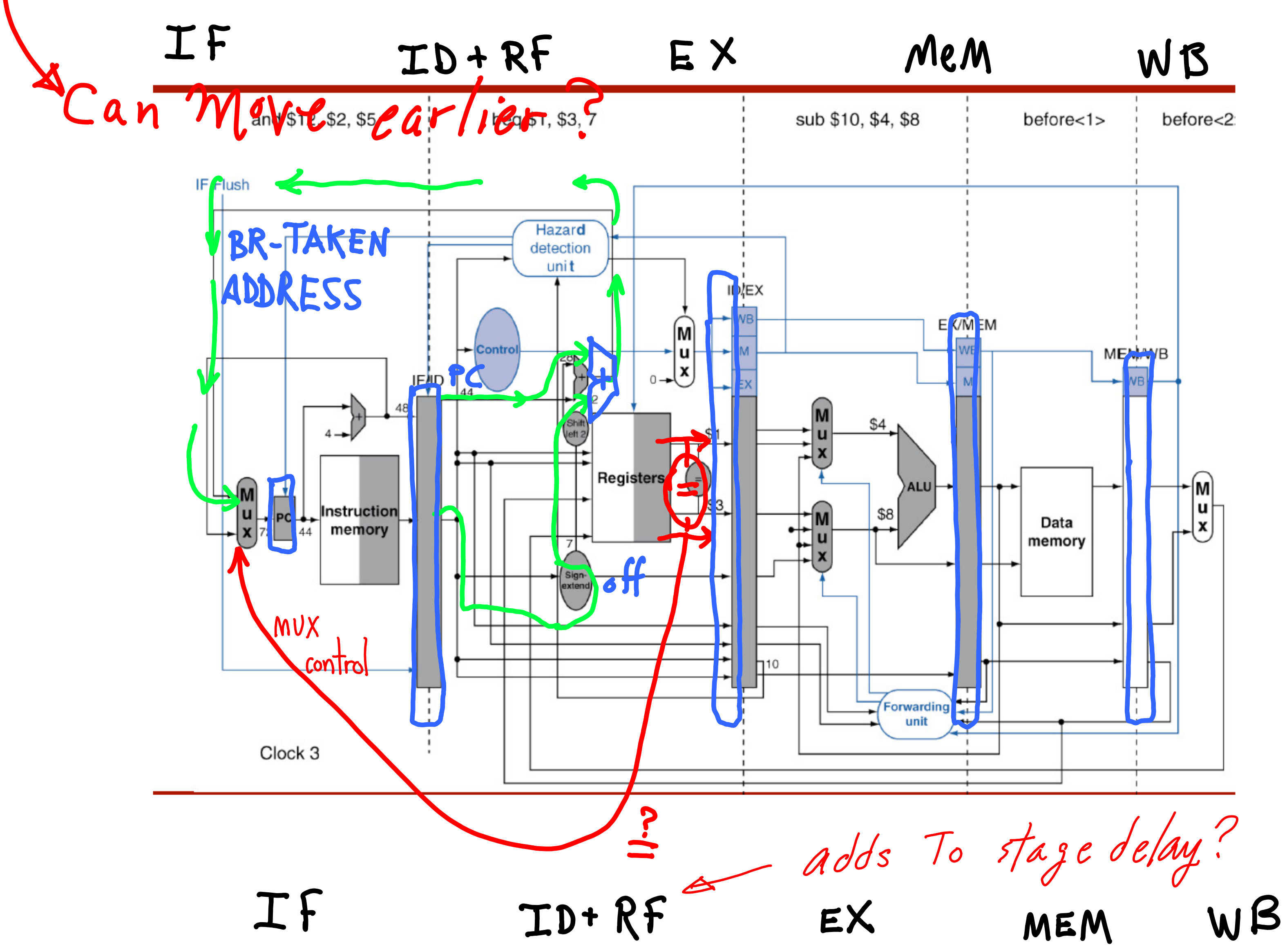
1   2   3   4

↑ fetch from L

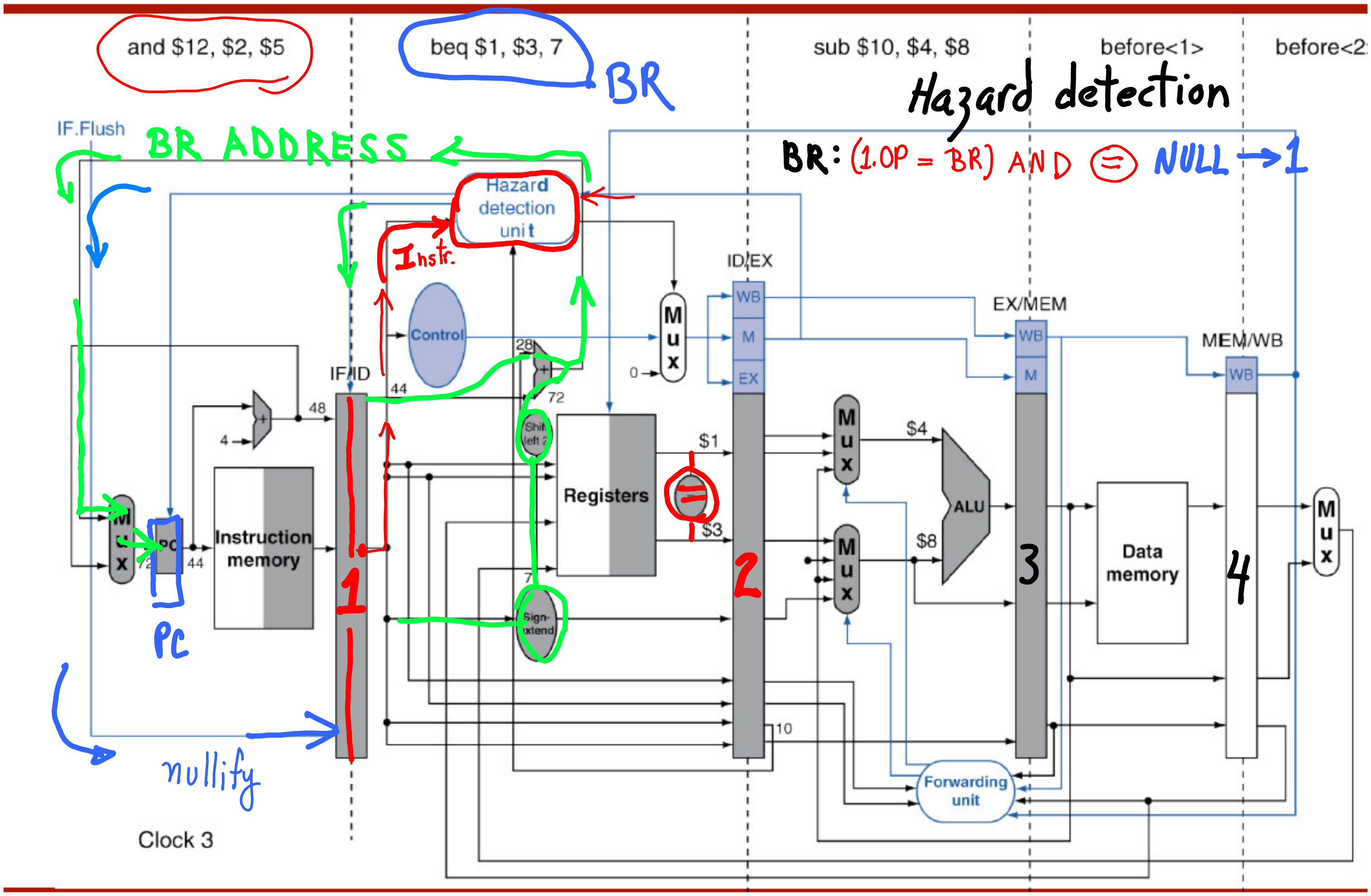
1 bubble for taken

$$CPI_{BR} = \begin{cases} 2, & \text{Taken} \\ 1, & \text{not Taken} \end{cases}$$

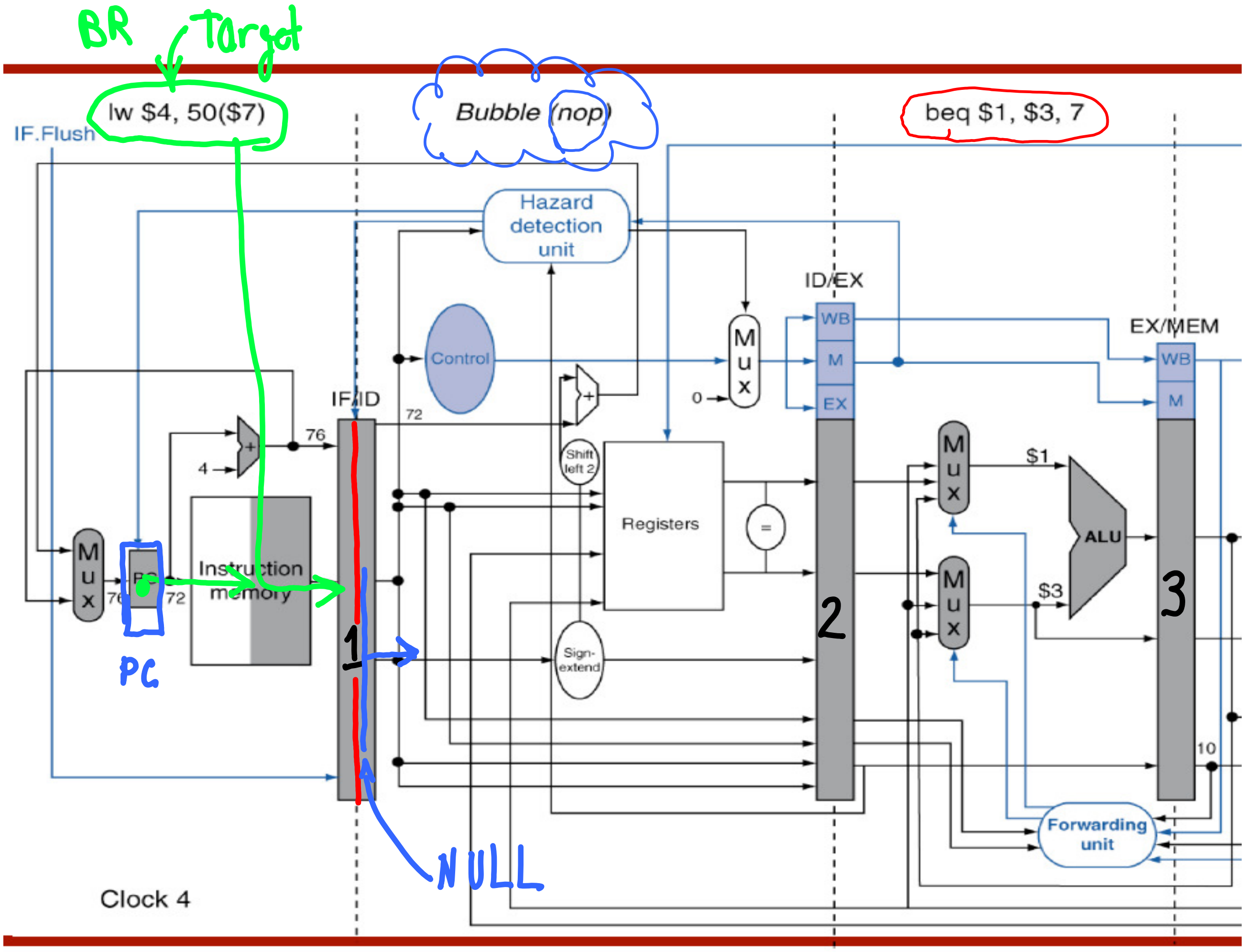




Added delays:
1. address adder after sign-extension.
2. EQUALS test after register fetch.



and $12, $2, $5
beq $1, $3, 7
BR
sub $10, $4, $8
before<1>
before<2
Hazard detection
BR: (1.OP = BR) AND (=) NULL → 1
IF.Flush
BR ADDRESS
Hazard detection unit
Instr.
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
M u x
WB
M
EX
0
28
72
44
48
4
Shift left 2
Registers
$1
$3
$4
$8
=
ALU
Data memory
Instruction memory
Sign-extend
PC
1
2
3
4
10
Forwarding unit
nullify
Clock 3



BR Target
lw $4, 50($7)
Bubble (nop)
beq $1, $3, 7
IF.Flush
Hazard detection unit
ID/EX
EX/MEM
Control
M u x
0
WB
M
EX
IF/ID
72
76
4
Shift left 2
Registers
=
$1
$3
ALU
Instruction memory
PC
Sign-extend
1
2
3
10
Forwarding unit
NULL
Clock 4

Suppose $BR = \begin{cases} 50\% \text{ taken} \\ 50\% \text{ not taken} \end{cases}$

Suppose trace T has 25% BR instructions.

S_1-2 = 4/3 ~ 33%

S_1-3 = 5/3 ~ 67%

S_2-3 = 5/4 ~ 25%

1. W/ BR hazard detection + early BR

$$CPI_{BR} = \begin{cases} 2, \text{ taken} \\ 1, \text{ not taken} \end{cases} \Rightarrow \overline{CPI}_{BR} = (0.5 \cdot 2 + 0.5 \cdot 1) = 1.5$$

2. Compiler inserts NOP for BR delay

$$CPI_{BR} = 2$$

3. W/ BR hazard detection + late BR

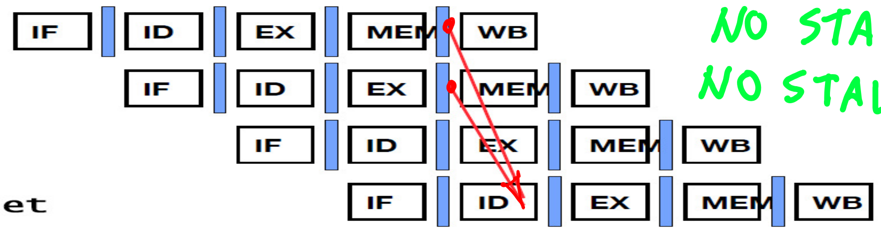
$$CPI_{BR} = \begin{cases} 4, \text{ taken} \\ 1, \text{ not taken} \end{cases} \Rightarrow \overline{CPI}_{BR} = (0.5 \cdot 4 + 0.5 \cdot 1) = 2.5$$

Q. Is (3.) a reasonable choice given its expense?
What if (1.) is not possible?

Data Hazards for Branches

- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction

```
ALU   add  $1,  $2,  $3
ALU   add  $4,  $5,  $6
      …
      beq  $1,  $4,  target
```



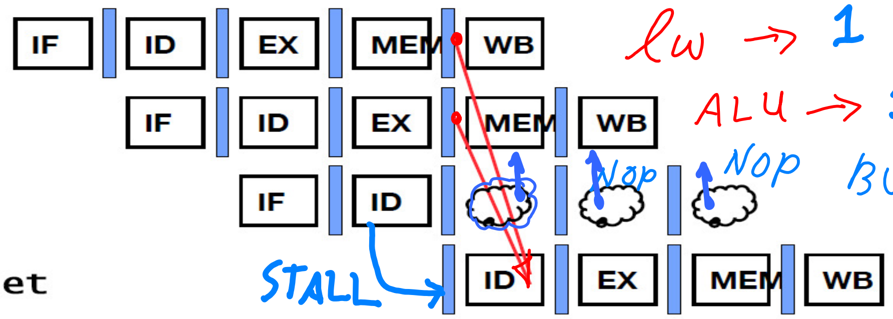
NO STALL

NO STALL

↑ BR needs values in ID, gets via forwarding

- Can resolve using forwarding
  - Additional datapaths and control

```
lw   $1,  addr
add  $4,  $5,  $6
beq  stalled
beq  $1,  $4,  target
```



lw → 1 stall

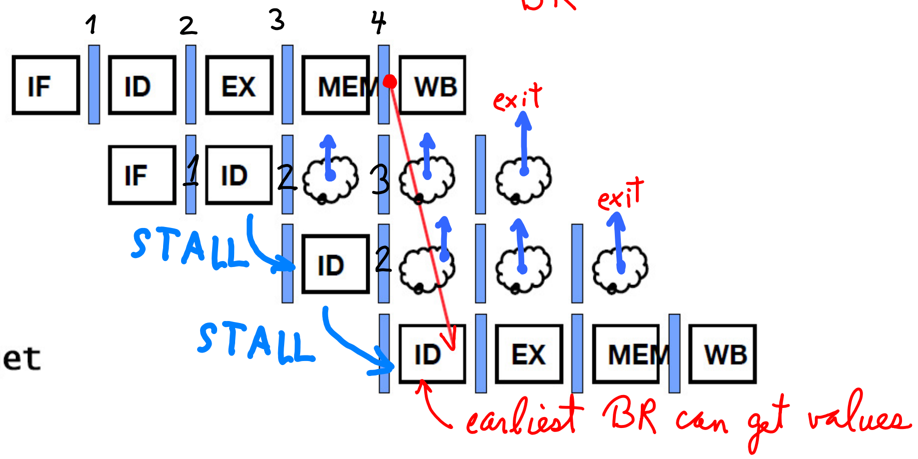
ALU → 1 stall

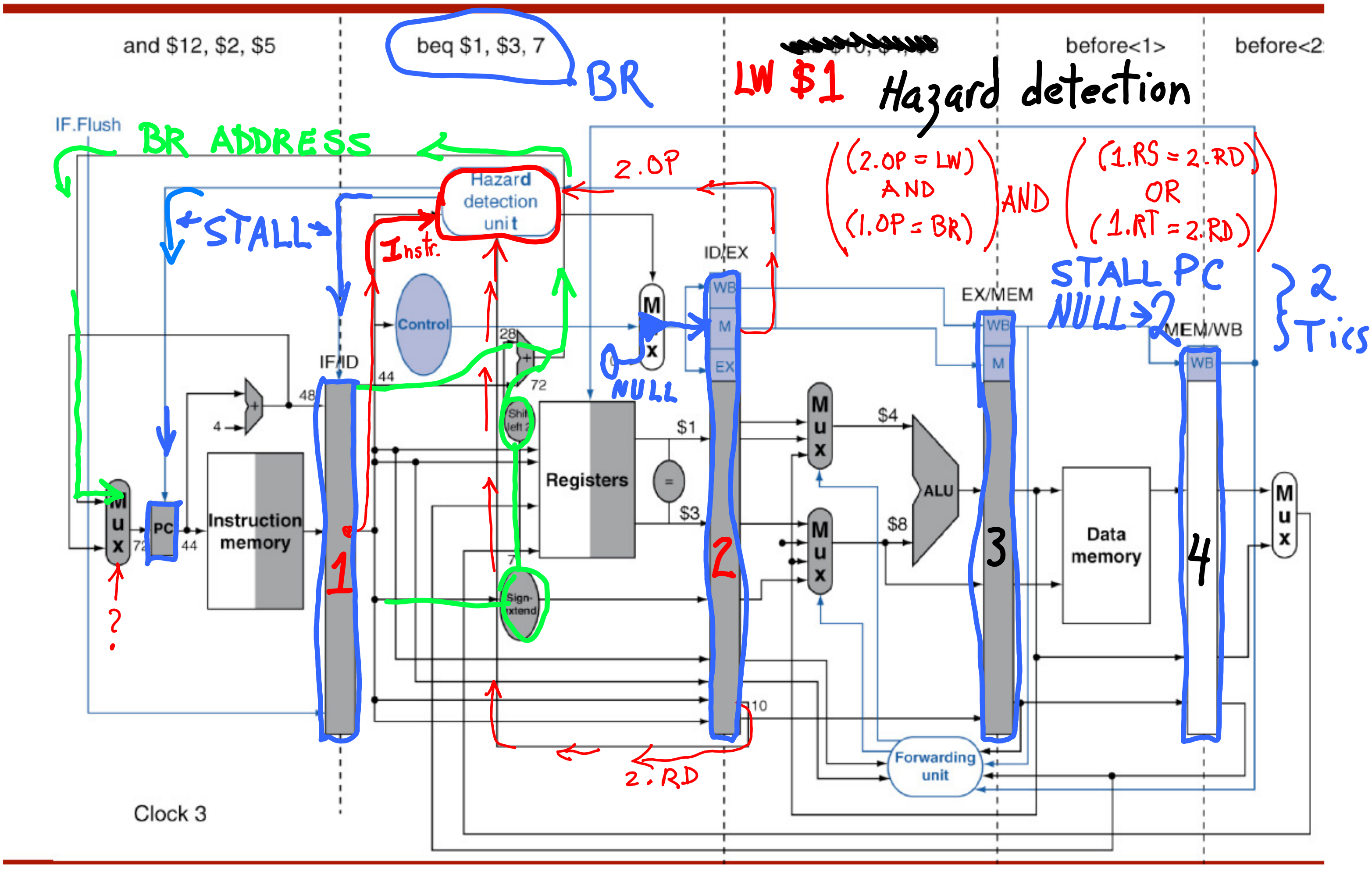
BUBBLES

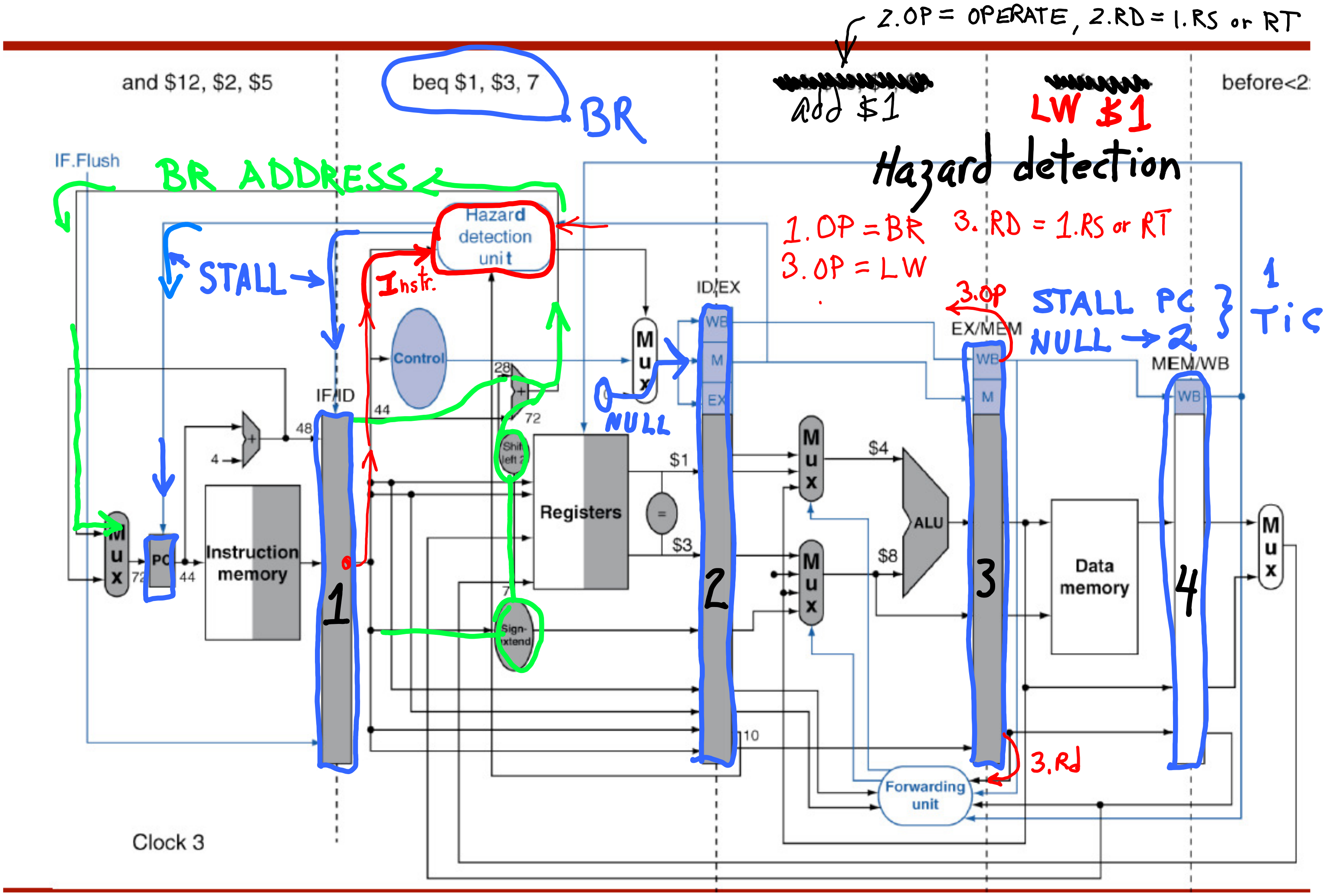
↑ earliest BR can get values

lw
BR ⇒ 2 stalls

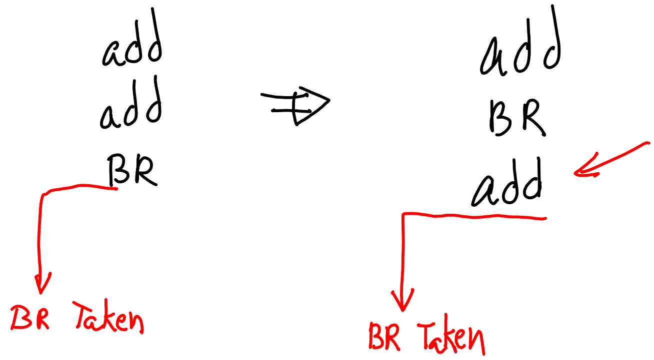
```
lw $1, addr
beq stalled
beq stalled
beq $1, $0, target
```






2.OP = OPERATE, 2.RD = 1.RS or RT
and $12, $2, $5
beq $1, $3, 7
BR
add $1
LW $1
before<2
Hazard detection
1. OP = BR
3. OP = LW
3. RD = 1.RS or RT
IF.Flush
BR ADDRESS
Hazard detection unit
STALL
Instr.
Control
ID/EX
EX/MEM
MEM/WB
3.OP
STALL PC
NULL → 2
1 Tic
NULL
IF/ID
M u x
WB
M
EX
28
72
44
48
4
Shift left 2
Sign-extend
Registers
=
$1
$3
$4
$8
ALU
Instruction memory
Data memory
PC
Mux
1
2
3
4
10
3.Rd
Forwarding unit
Clock 3

Delayed BR alternative



"delay slot" is always executed, no bubbles needed.

Are branches really 3 instructions:

1) condition, 2) address, 3) take branch

This is a version of 2: let compiler fill when possible, else fill w/ NOP

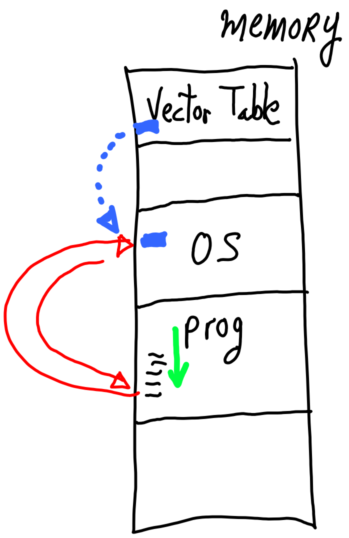
⇒ 50% NOPs in delay slot.

Q. Delayed BR has same performance as (1) above?

What about longer pipelines?

Control Hazards: Exceptions, Traps, Interrupts

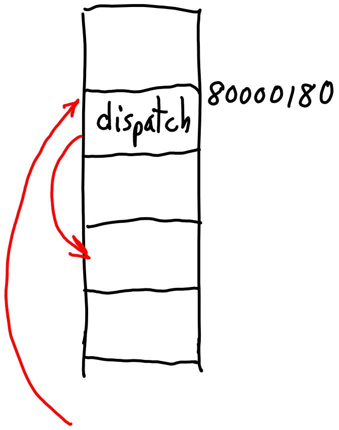
- Something happens
  - I/O device sends signal : INTERRUPT
  - CPU detects execution error : EXCEPTION
  - Execution of a sys call : TRAP



- Do something about it
  - Talk to device, get data, send data → jump back to prog.
  - Send error message, terminate program
  - Jump to OS routine, do service → jump back to prog.

OPTIONS FOR Control Transfer

- Hardwired : always go to 8000 0180
  - figure out what routine to jump to (use "cause" Reg.)
  - maybe a few Targets hardwired : 80000080 INT
    80000180 EXC
    80000280 TRAP



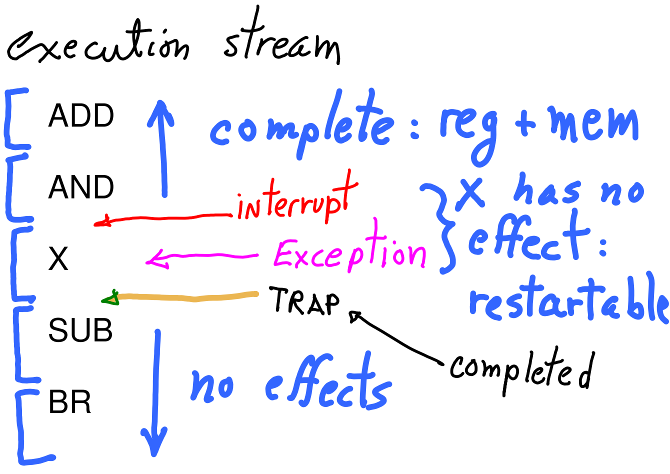
- Hardwired jump via jump Table (Vector Table)

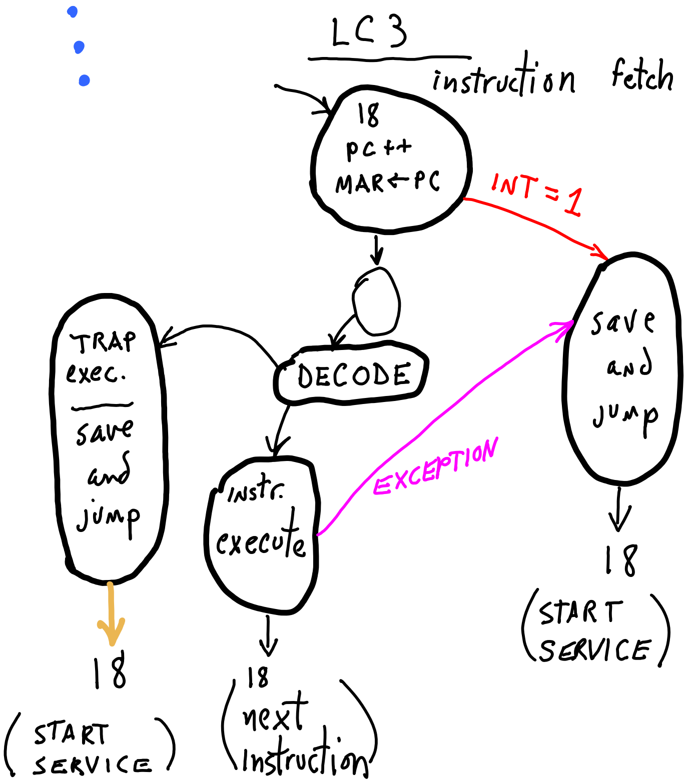
- Combination of these (dispatcher per vector)

Jump ≈ BR hazard

Precise Exceptions

- Definition: precise exceptions
  - All previous instructions had completed
  - The faulting instruction was not started
  - None of the next instructions were started
    - No changes to the architecture state (registers, memory)
- Why are precise exceptions desirable by OS developers?
- With a single cycle machine, precise exceptions are easy
  - Why?





possible context switch, w/ or w/o HW support

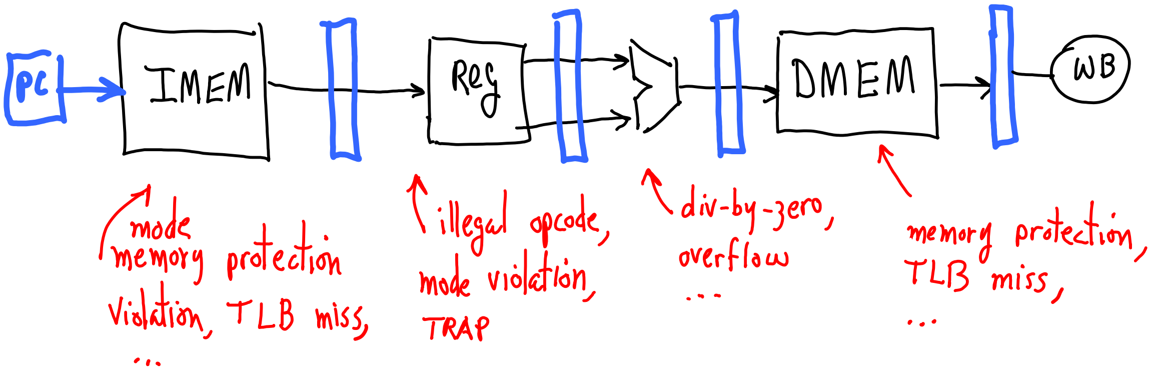
We (OS) need To know

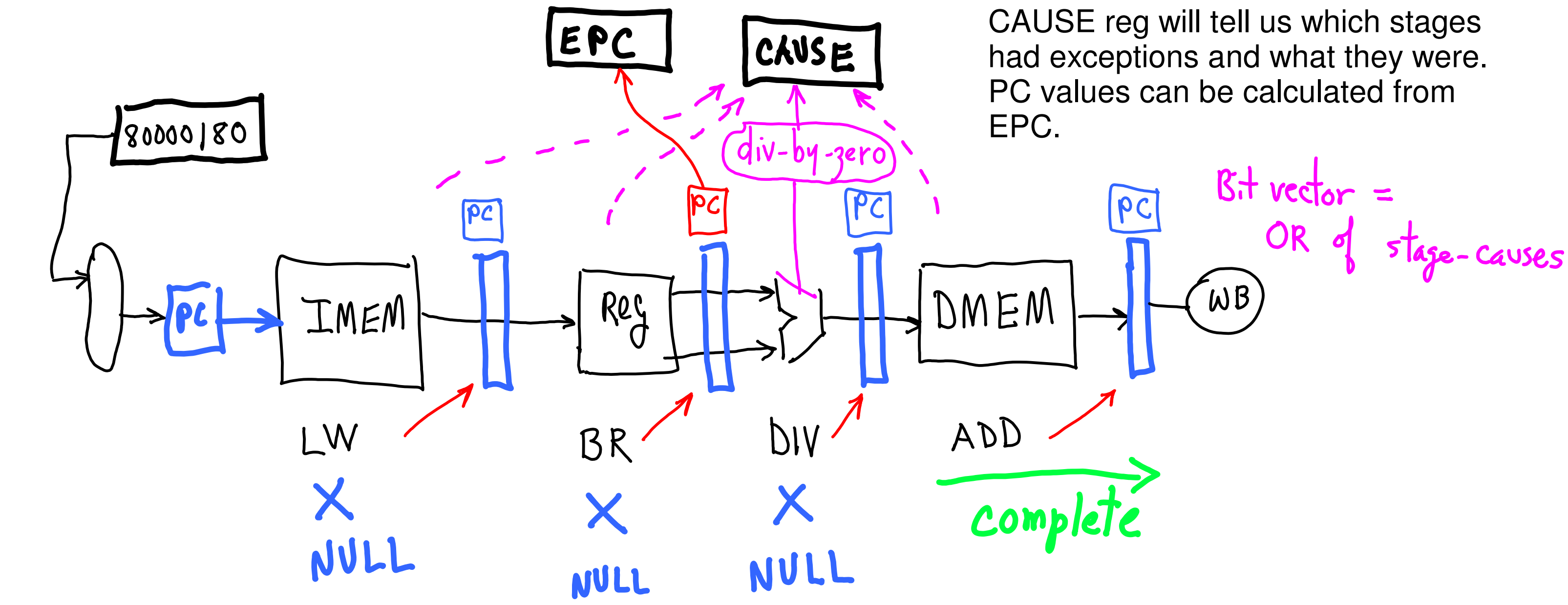
- what happened (INT, EXC, TRAP)
- How to restart (PC, ...)
- which instruction caused problem
- what data caused problem
- which device needs help

MIPS

Use EPC - record PC of current? instruction

CAUSE-REG - record what happened

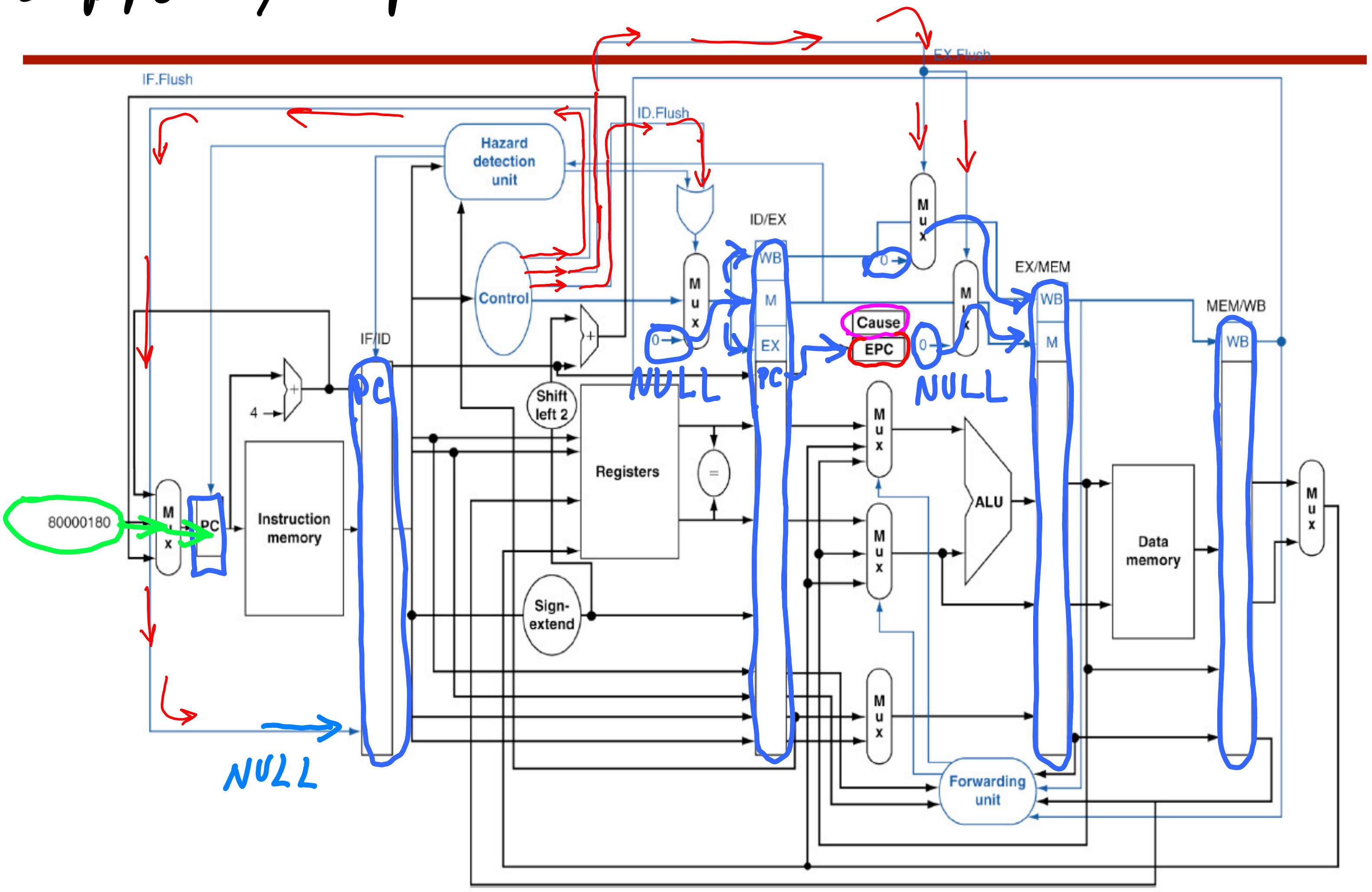

PC
IMEM
Reg
DMEM
WB
mode memory protection violation, TLB miss, ...
illegal opcode, mode violation, TRAP
div-by-zero, overflow ...
memory protection, TLB miss, ...



CAUSE reg will tell us which stages had exceptions and what they were. PC values can be calculated from EPC.

1. Let previous instructions complete
2. NULL following instructions
3. Save PC of problem instruction into EPC (PC+4)
4. Save exception code into EPC
5. NULL offending instruction (saves state = MEM+REG)
5. Jump to OS at 80000180
--- (or, freeze and start co-processor)
???---multiple exceptions?

5-pipe w/ exceptions



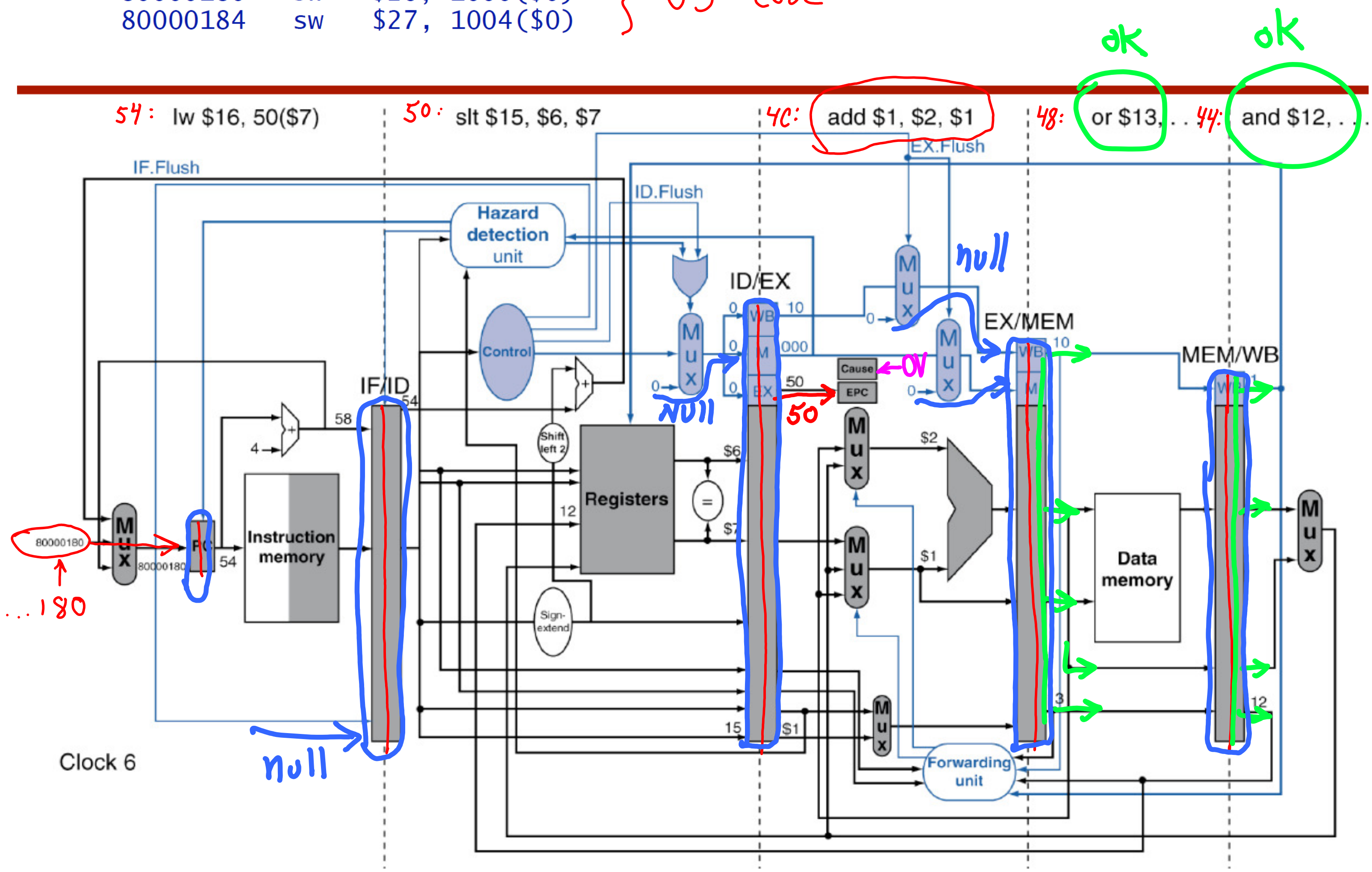
- Exception on add in

```
40     sub   $11, $2, $4
44     and   $12, $2, $5
48     or    $13, $2, $6
4C     add   $1,  $2, $1    ← overflow
50     slt   $15, $6, $7
54     lw    $16, 50($7)
…
```

- Handler

```
80000180    sw    $26, 1000($0)
80000184    sw    $27, 1004($0)
```

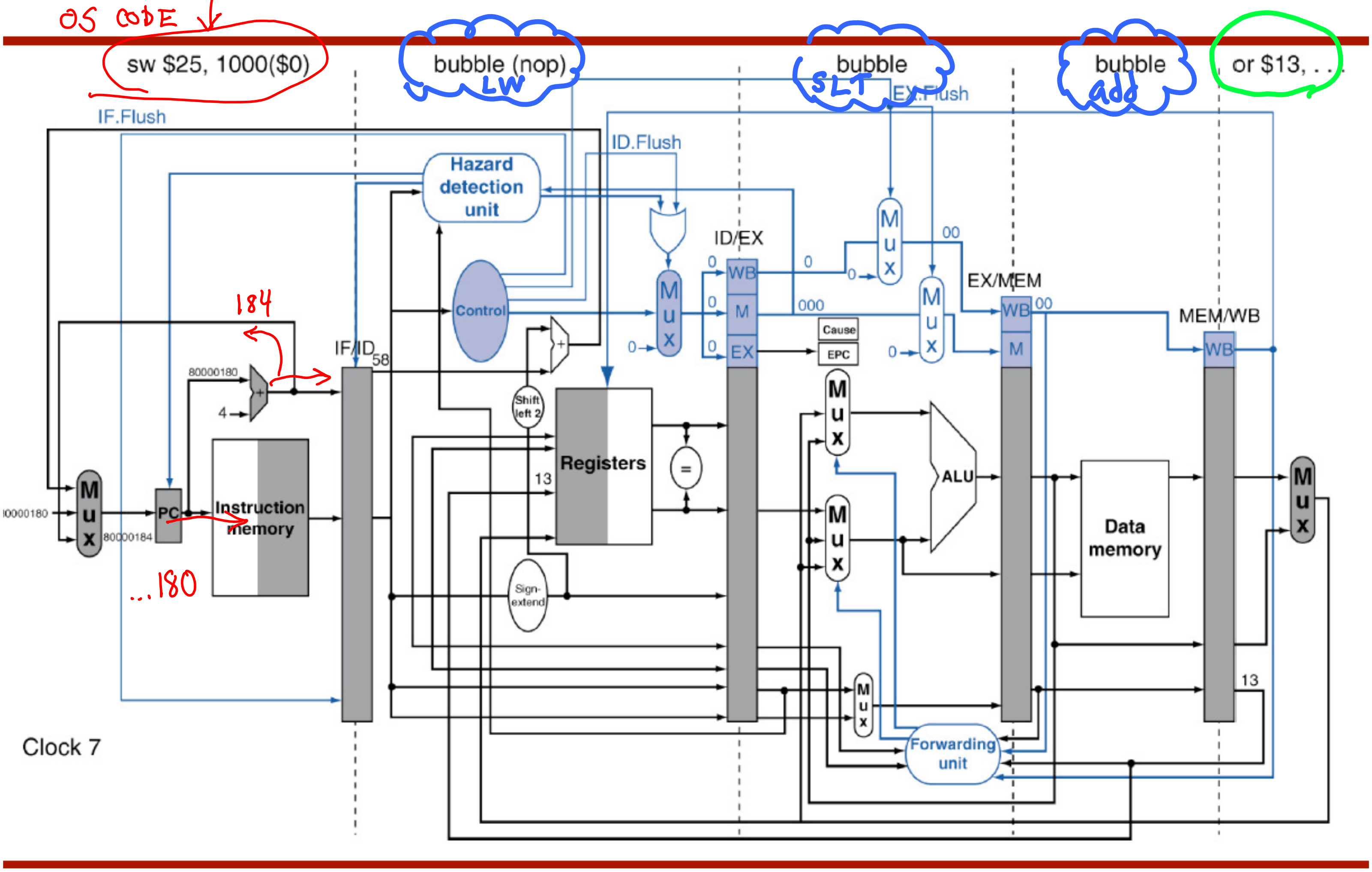
} OS code



access cause register:

mfc0 $18, $13

co-processor register = $13 = cause register



- Pipelining overlaps multiple instructions
  - Could have multiple exceptions at once

overflow + seg fault + illegal opcode

- Simple approach: deal with exception from earliest instruction
  - Flush subsequent instructions
  - Necessary for "precise" exceptions

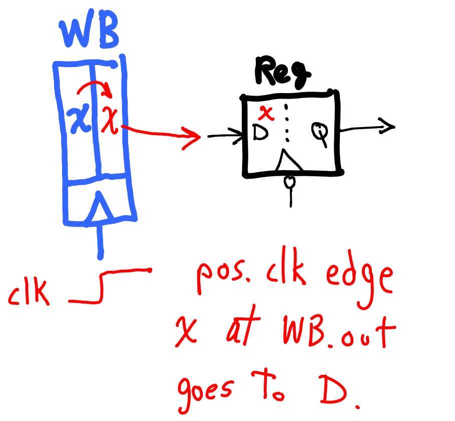
restarting? – later instructions ⇒ EXC.
– might not restart,

- In more complex pipelines
  - Multiple instructions issued per cycle
  - Out-of-order completion
  - Maintaining precise exceptions is difficult!

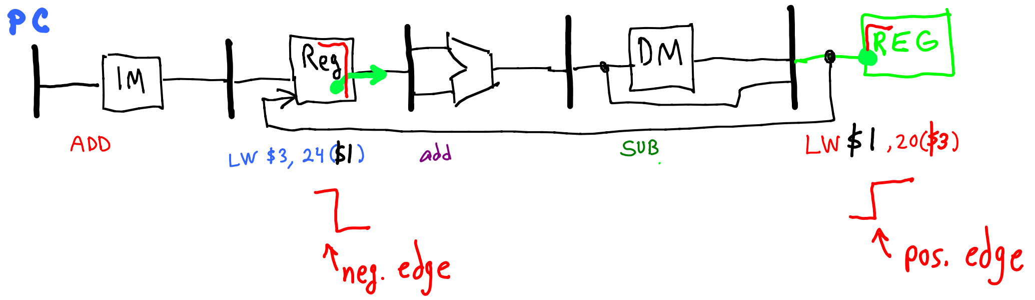
Imprecise Exceptions

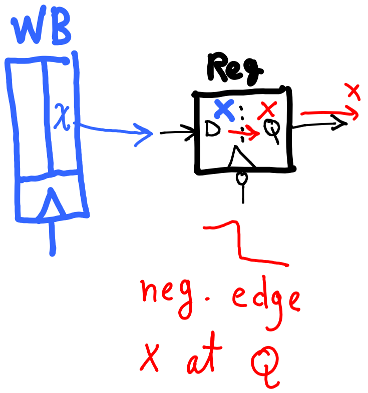
- Actually available in several processors
- Just stop pipeline and save pipeline state
  - Including exception cause(s)
- Let the handler work out
  - Which instruction(s) had exceptions
  - Which to complete or flush
    - May require "manual" completion
- Simplifies hardware, but more complex handler software
- Not feasible for complex multiple-issue out-of-order pipelines

Pipe Summary

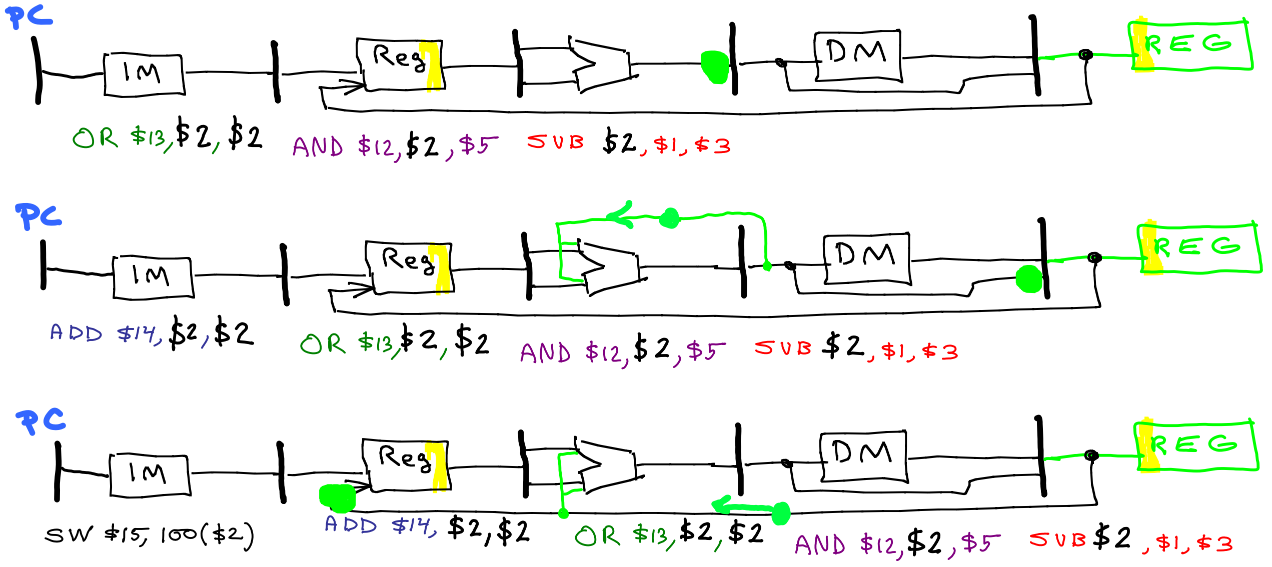


Shorten feedback through register file using neg. edge triggered FFs.

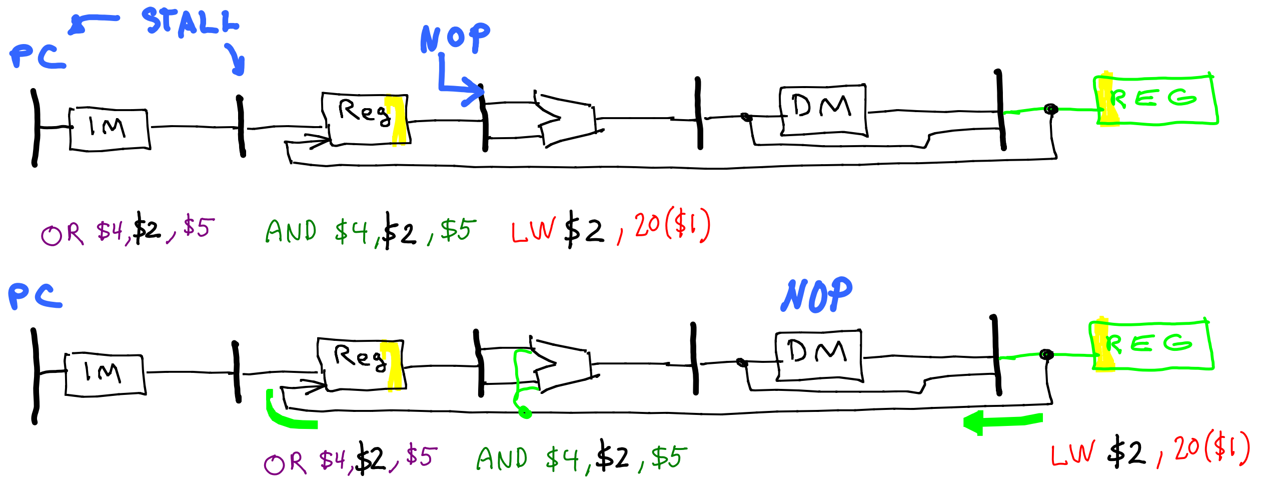




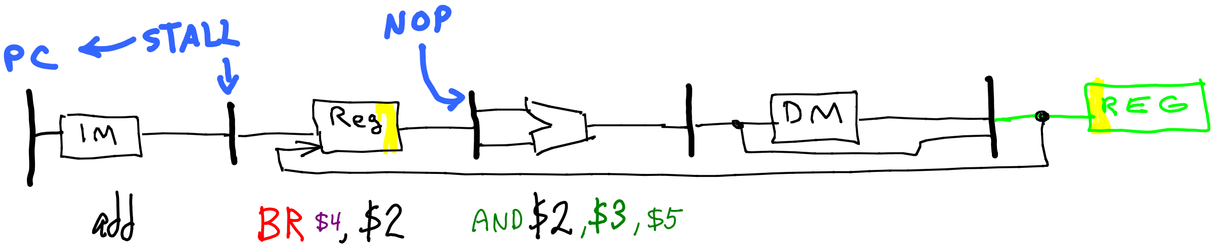
Data hazard detection can forward data without bubbles for operate instructions.

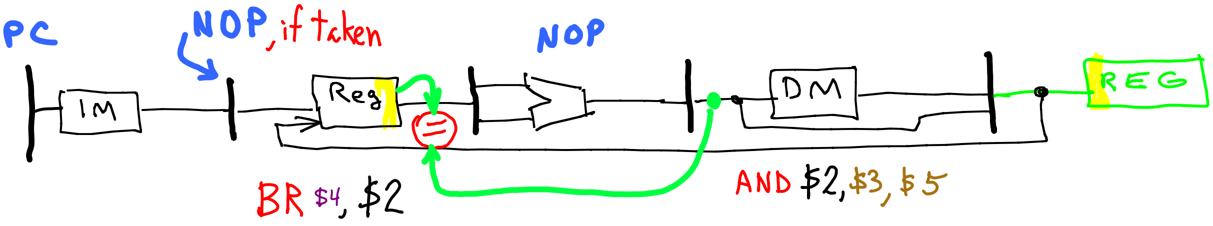


Load-use delay causes a bubble (unless compiler fills slot), then forwarding used.

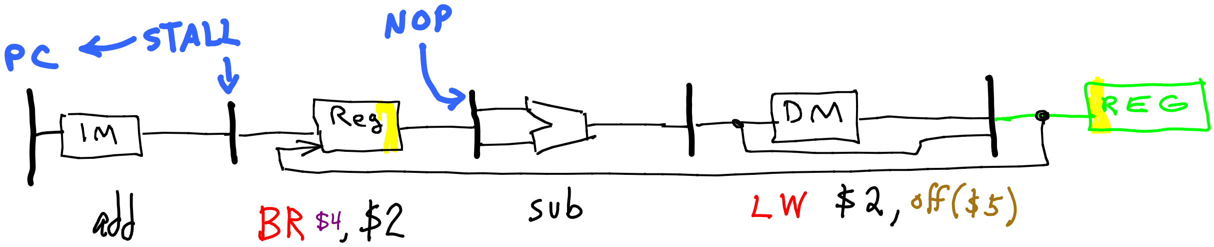


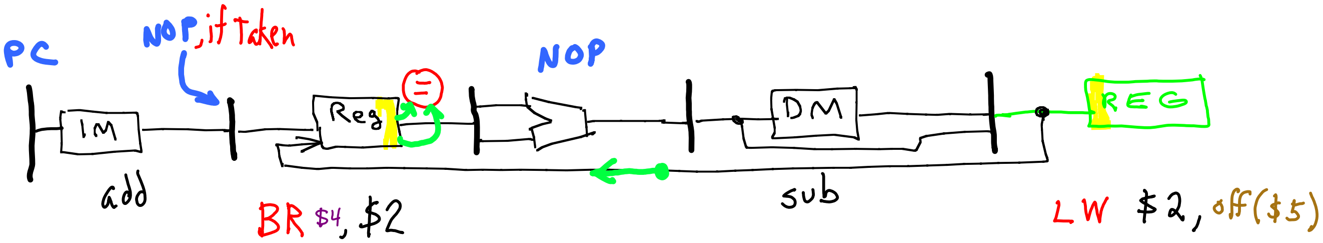
Branch data hazard from operate instruction cause stall and one bubble, then uses forwarding. Almost the same as load-use delay. Inserts NOP if branch taken.



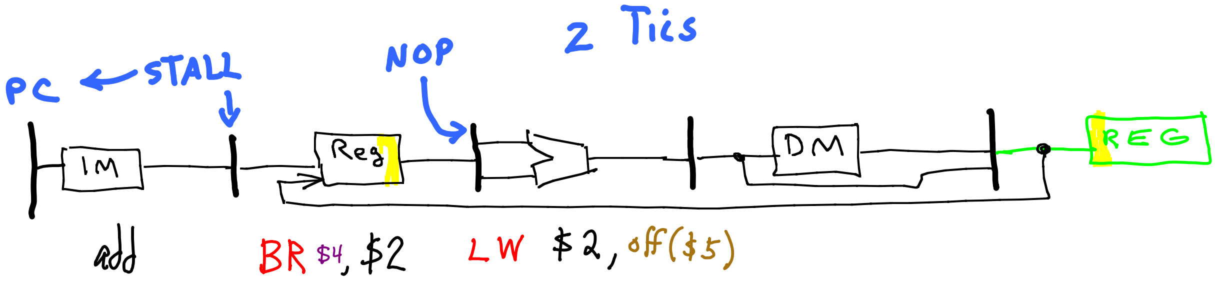


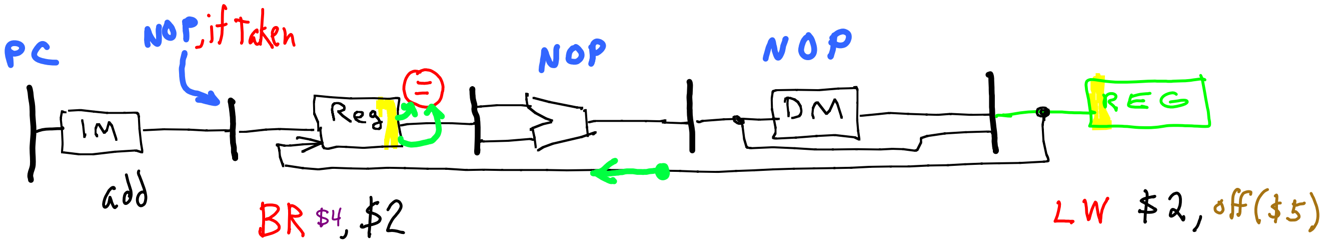
Branch data hazard from LW instruction in DMEM causes stall and one bubble, then uses forwarding. Same as operate data hazard.



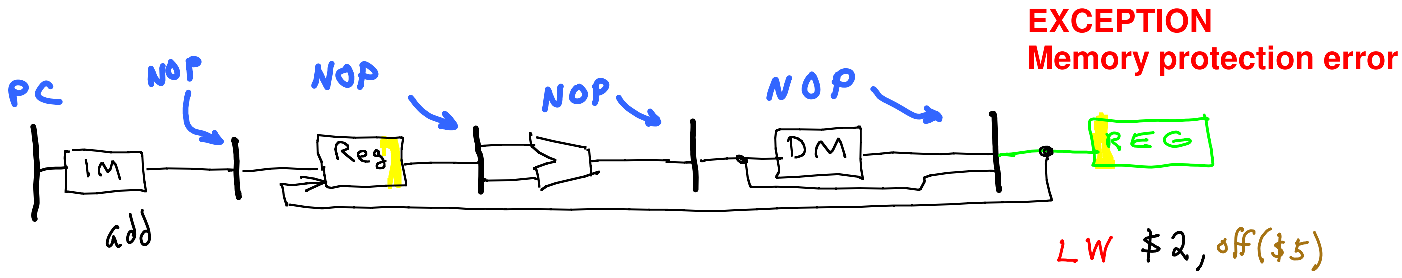


Branch data dependency with LW in EX causes two bubbles, then forwarding.

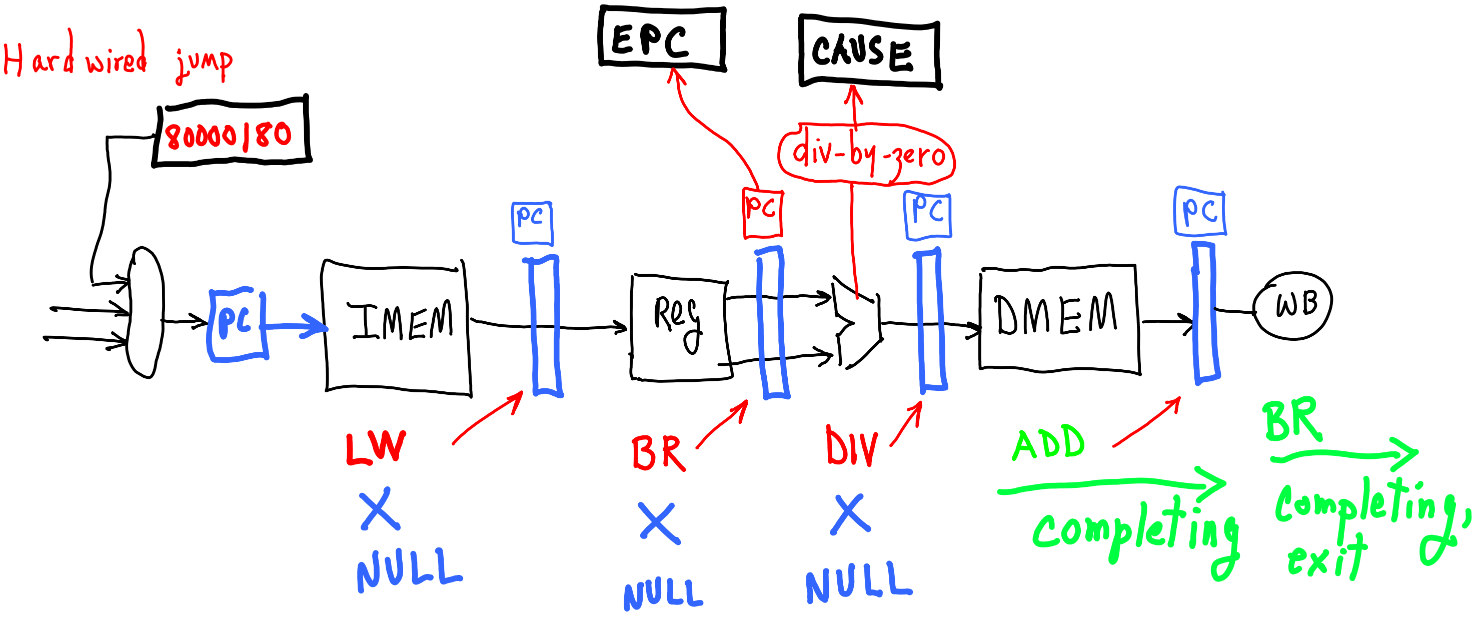




Exceptions, traps, and interrupts can cause many bubbles.


EXCEPTION
Memory protection error

For (precise) exceptions,
--- 1. stage.PC ===> EPC   (cause code #) ===> CAUSE register.
--- 2. upstream instructions <=== NULL   (let downstream instructions complete)
--- 3. stage.INSTR_OP <=== NULL
--- 3 Jump to OS for service.



Questions

1. What to do w/ multiple exceptions during same clock cycle?
2. What to do w/ exceptions for completing instructions?
3. How to know what happened?
4. What about nested exceptions; i.e., exceptions occuring during exception handling?