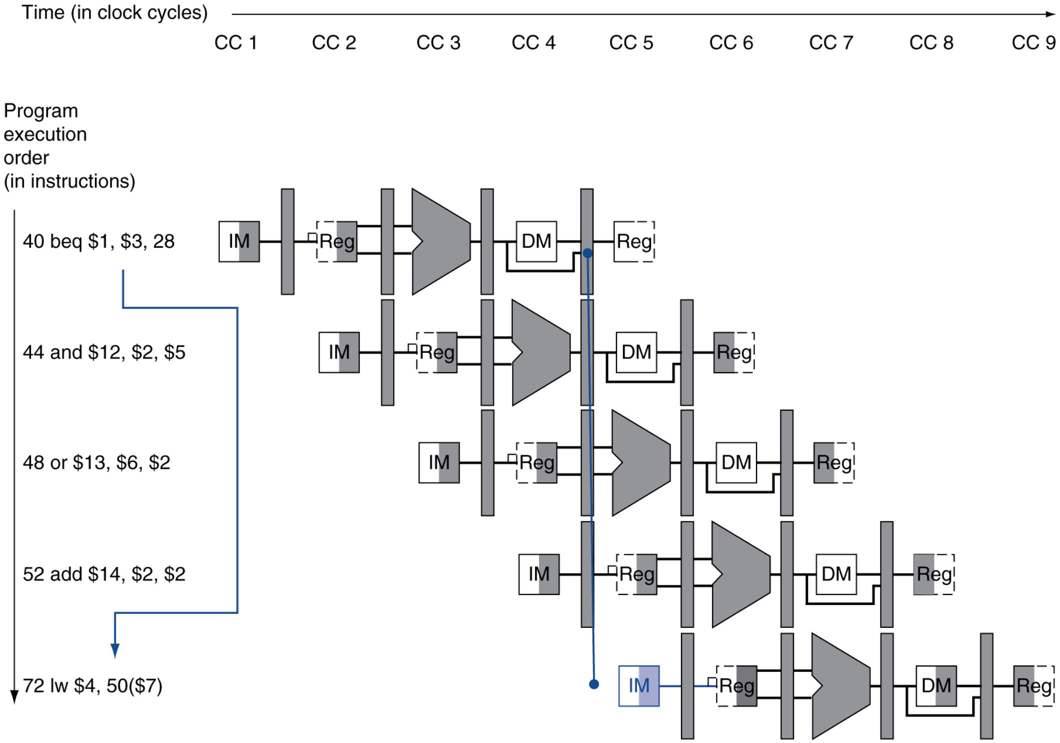
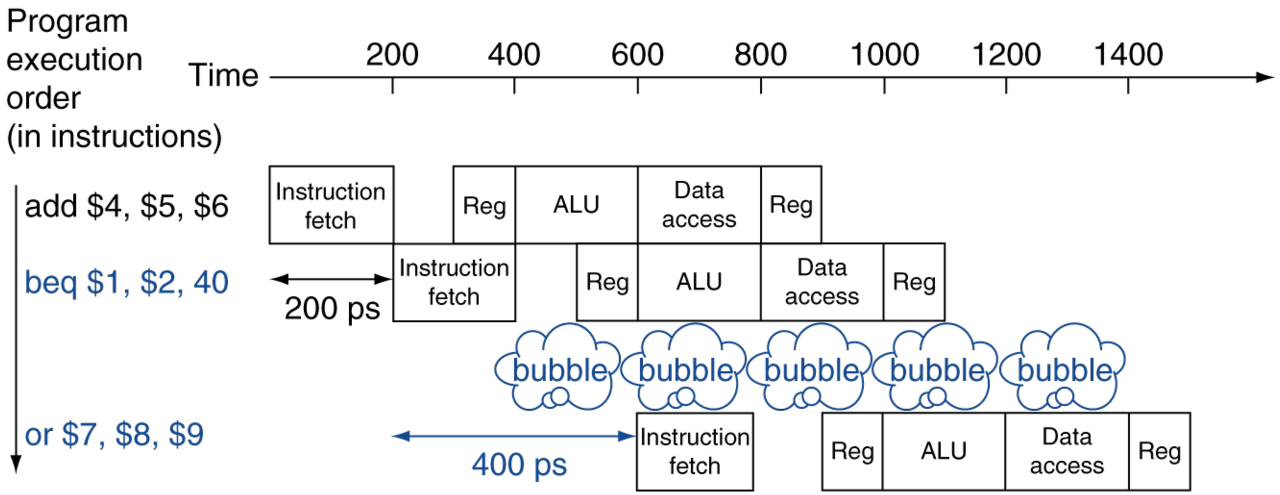
Control (Branch) Hazards

- Branch 명령은 flow of control 을 변경한다.
  - Fetch 할 다음 명령어의 주소는 branch 조건에 따라 달라진다.
- Solution to branch hazard
  - pipeline stall
  - branch prediction
    - static
      - as taken
      - as not taken
    - dynamic



Pipeline Stall on Branch Instr.

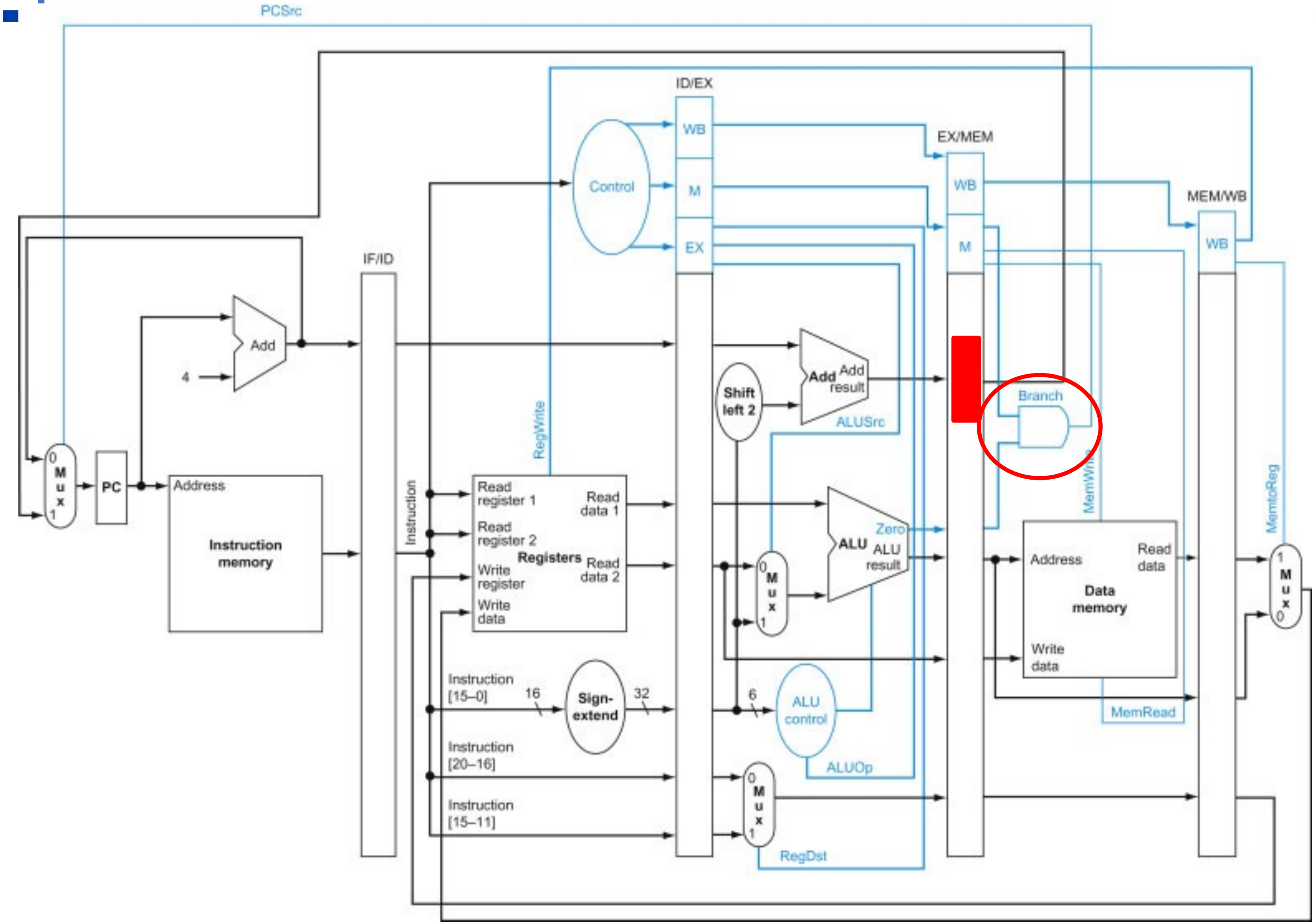
- Wait until branch outcome determined before fetching next instruction



Branch Prediction

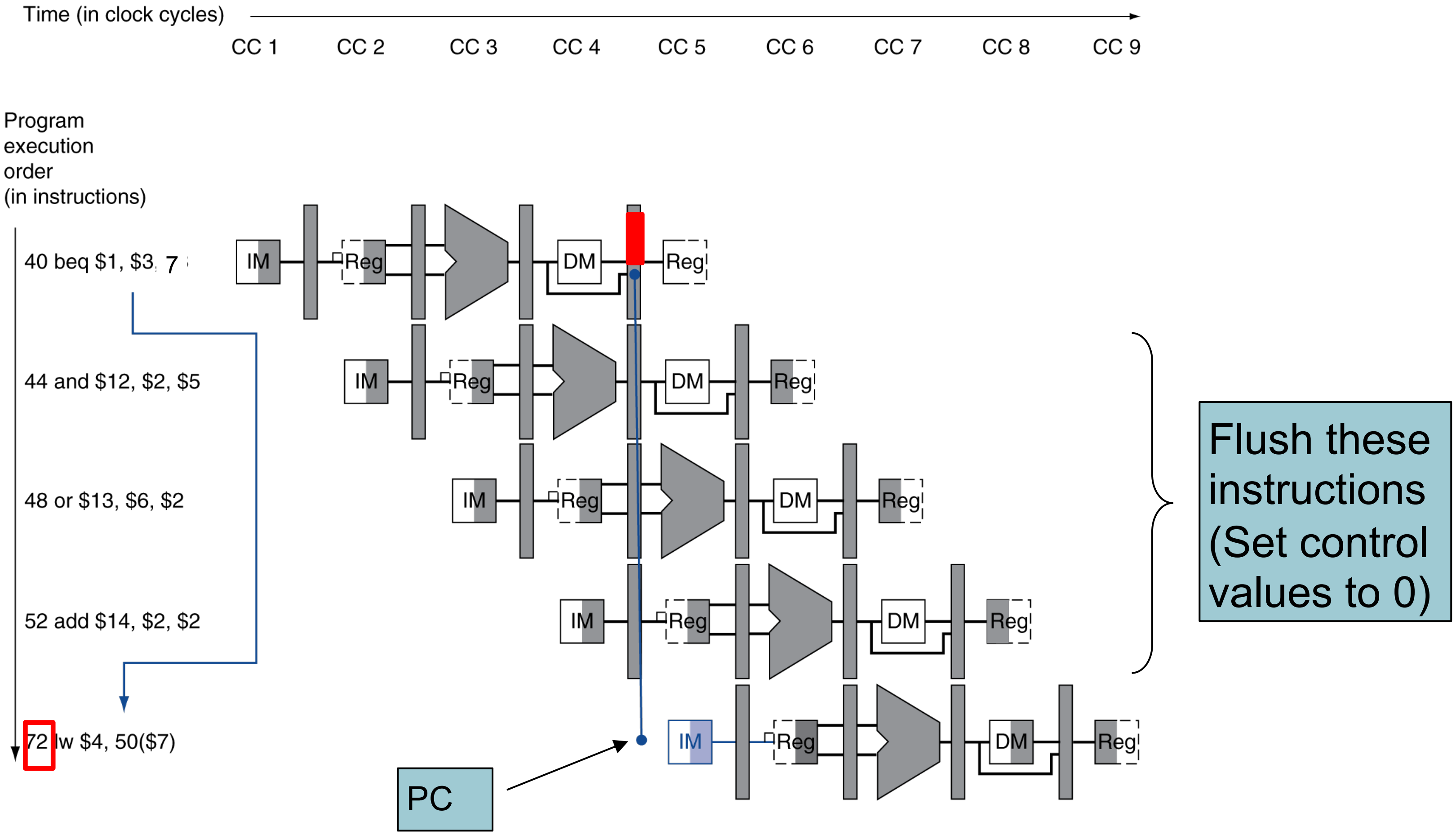
- Longer pipelines can’t readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if the prediction is wrong
- In the textbook pipeline processor
  - predict branches as not taken

Pipelined Control



Branch Hazards

- If branch outcome determined in MEM



Flush these instructions (Set control values to 0)

PC

Reducing Branch Delay

- Modification of the hardware
  - BTA adder in ID stage
  - Register comparator in ID stage
- Example: branch taken

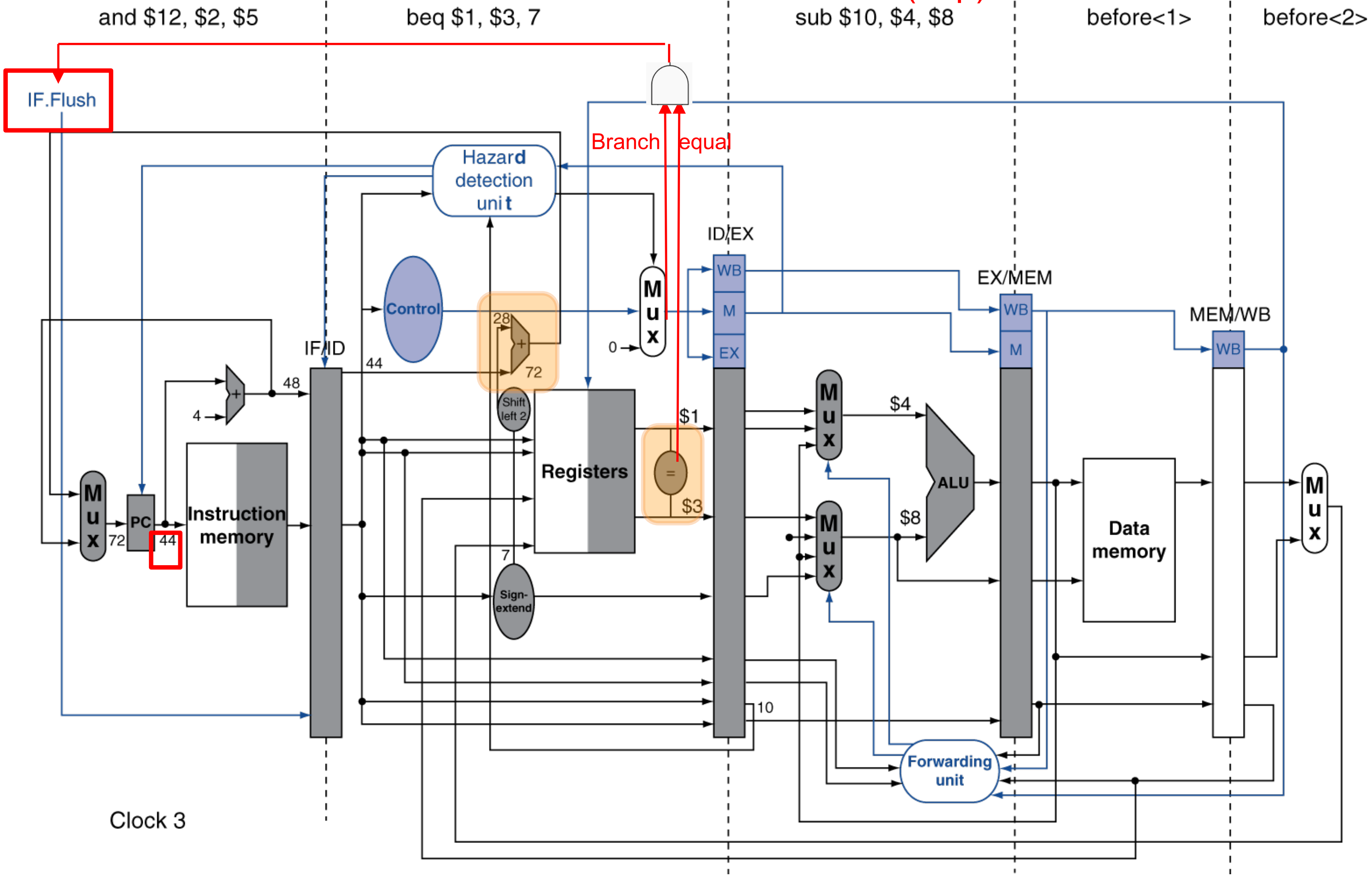
```
36:  sub  $10, $4, $8
40:  beq  $1,  $3, 7
44:  and  $12, $2, $5
48:  or   $13, $2, $6
52:  add  $14, $4, $2
56:  slt  $15, $6, $7
    ...
72:  lw   $4, 50($7)
```

← immediate field 에 저장된 offset value

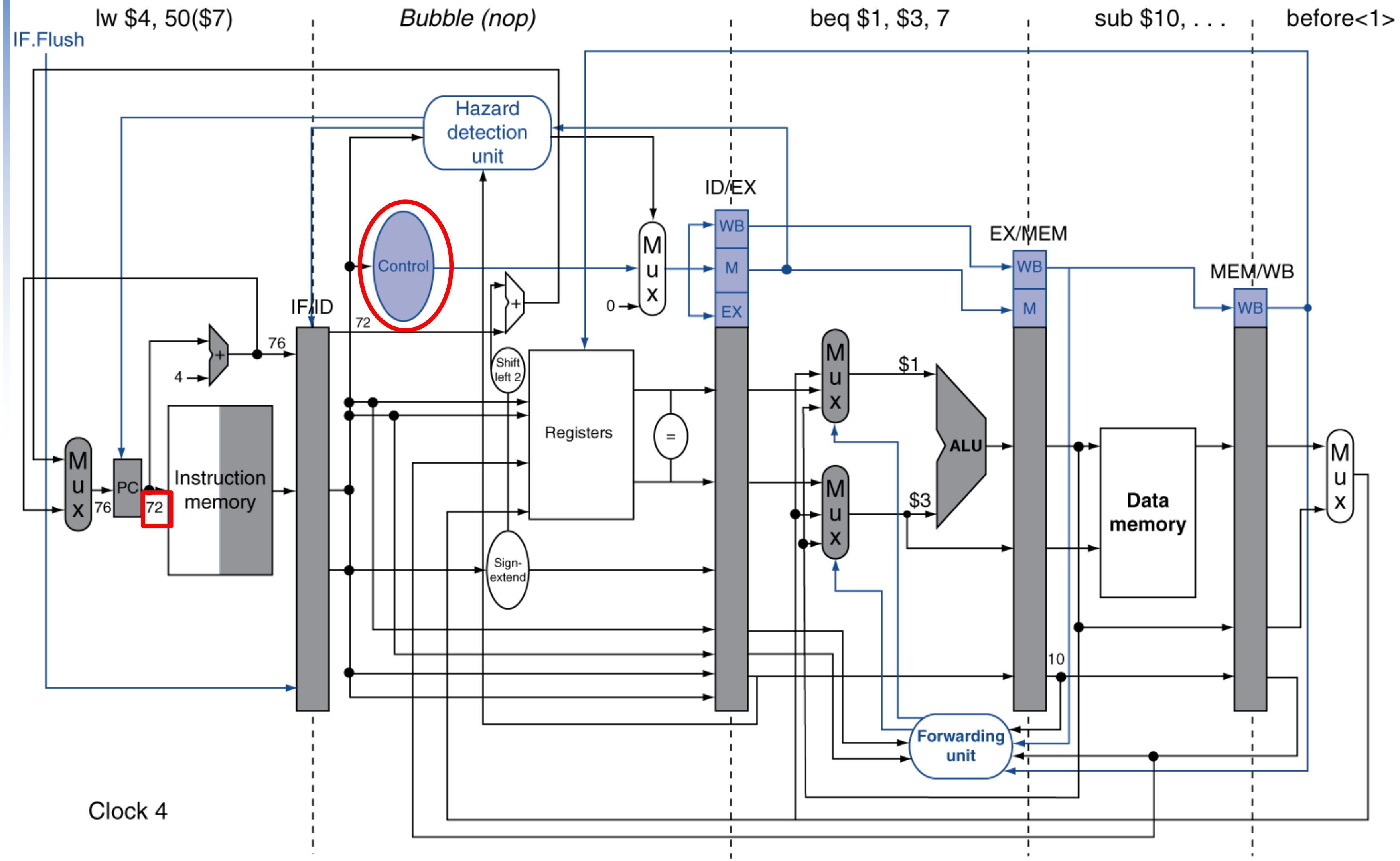
44 + 7 x 4

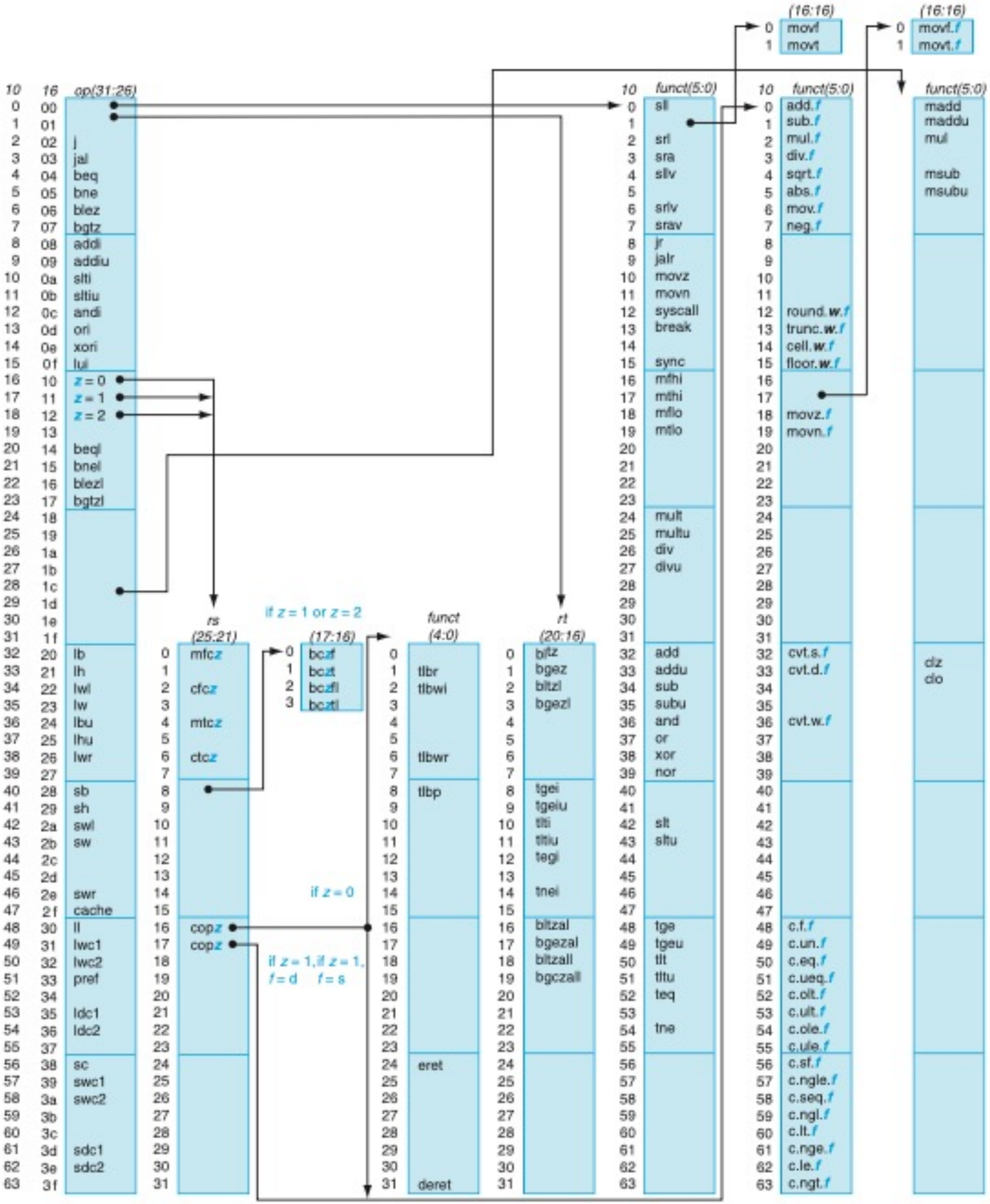
Example: Branch Taken (CC3)

IF.Flush 가 1이면 IF/ID 의 instruction 부분에 0x00000000 (nop) 이 써짐



Example: Branch Taken (CC4)





10	16	op(31:26)
0	00	
1	01	
2	02	j
3	03	jal
4	04	beq
5	05	bne
6	06	blez
7	07	bgtz
8	08	addi
9	09	addiu
10	0a	slti
11	0b	sltiu
12	0c	andi
13	0d	ori
14	0e	xori
15	0f	lui
16	10	z = 0
17	11	z = 1
18	12	z = 2
19	13	
20	14	beql
21	15	bnel
22	16	blezl
23	17	bgtzl
24	18	
25	19	
26	1a	
27	1b	
28	1c	
29	1d	
30	1e	
31	1f	
32	20	lb
33	21	lh
34	22	lwl
35	23	lw
36	24	lbu
37	25	lhu
38	26	lwr
39	27	
40	28	sb
41	29	sh
42	2a	swl
43	2b	sw
44	2c	
45	2d	
46	2e	swr
47	2f	cache
48	30	ll
49	31	lwc1
50	32	lwc2
51	33	pref
52	34	
53	35	ldc1
54	36	ldc2
55	37	
56	38	sc
57	39	swc1
58	3a	swc2
59	3b	
60	3c	
61	3d	sdc1
62	3e	sdc2
63	3f	

	rs (25:21)
0	mfcz
1	
2	cfcz
3	
4	mtcz
5	
6	ctcz
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	copz
17	copz
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	

	(17:16)
0	bczf
1	bczt
2	bczfl
3	bcztl

	funct (4:0)
0	
1	tlbr
2	tlbwi
3	
4	
5	
6	tlbwr
7	
8	tlbp
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	eret
25	
26	
27	
28	
29	
30	
31	deret

	rt (20:16)
0	bltz
1	bgez
2	bltzl
3	bgezl
4	
5	
6	
7	
8	tgei
9	tgeiu
10	tlti
11	tltiu
12	tegi
13	
14	tnei
15	
16	bltzal
17	bgezal
18	bltzall
19	bgczall
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	

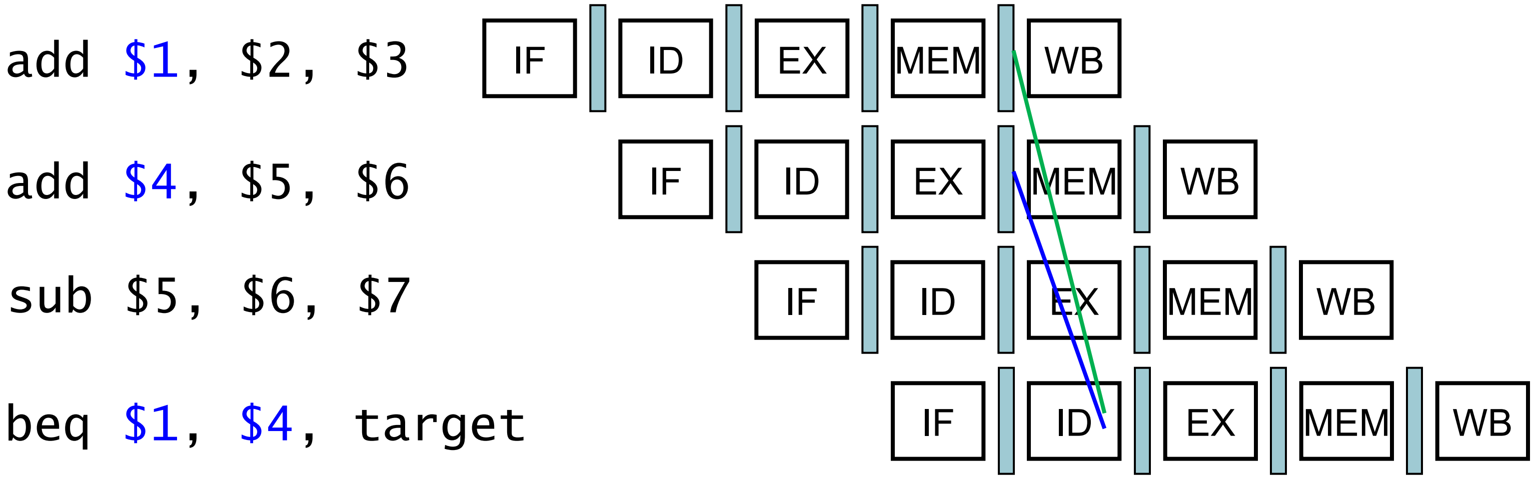
	(16:16)
0	movf
1	movt

	(16:16)
0	movf.f
1	movt.f

10	funct(5:0)	10	funct(5:0)	funct(5:0)
0	sll	0	add.f	madd
1		1	sub.f	maddu
2	srl	2	mul.f	mul
3	sra	3	div.f	
4	sllv	4	sqrt.f	msub
5		5	abs.f	msubu
6	srlv	6	mov.f	
7	srav	7	neg.f	
8	jr	8		
9	jalr	9		
10	movz	10		
11	movn	11		
12	syscall	12	round.w.f	
13	break	13	trunc.w.f	
14		14	cell.w.f	
15	sync	15	floor.w.f	
16	mfhi	16		
17	mthi	17		
18	mflo	18	movz.f	
19	mtlo	19	movn.f	
20		20		
21		21		
22		22		
23		23		
24	mult	24		
25	multu	25		
26	div	26		
27	divu	27		
28		28		
29		29		
30		30		
31		31		
32	add	32	cvt.s.f	clz
33	addu	33	cvt.d.f	clo
34	sub	34		
35	subu	35		
36	and	36	cvt.w.f	
37	or	37		
38	xor	38		
39	nor	39		
40		40		
41		41		
42	slt	42		
43	sltu	43		
44		44		
45		45		
46		46		
47		47		
48	tge	48	c.f.f	
49	tgeu	49	c.un.f	
50	tlt	50	c.eq.f	
51	tltu	51	c.ueq.f	
52	teq	52	c.olt.f	
53		53	c.ult.f	
54	tne	54	c.ole.f	
55		55	c.ule.f	
56		56	c.sf.f	
57		57	c.ngle.f	
58		58	c.seq.f	
59		59	c.ngl.f	
60		60	c.lt.f	
61		61	c.nge.f	
62		62	c.le.f	
63		63	c.ngt.f	

Data Hazards for Branches

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



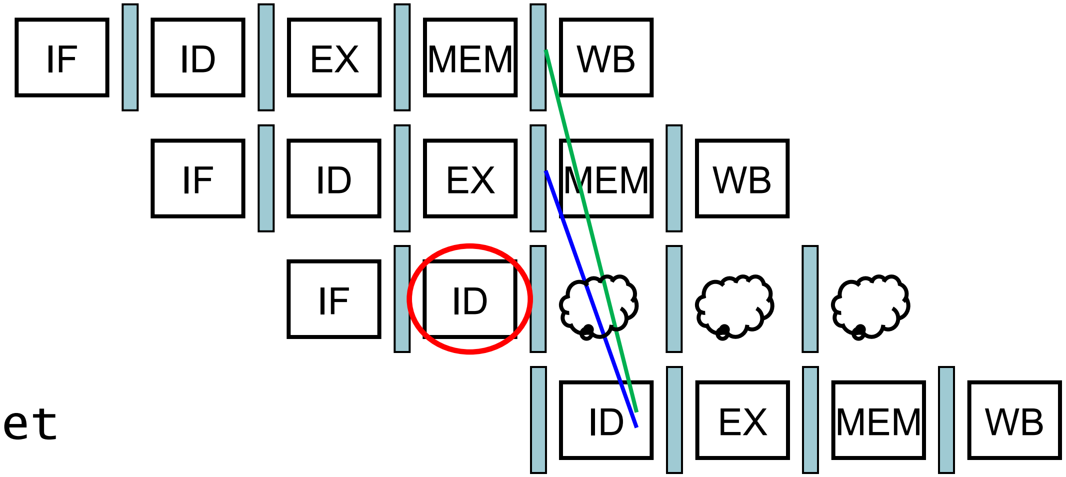
```
add $1, $2, $3
add $4, $5, $6
sub $5, $6, $7
beq $1, $4, target
```

- Can resolve using forwarding

Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
  - Need 1 stall cycle

```
lw  $1, 0($8)
add $4, $5, $6
beq stalled
beq $1, $4, target
```



Data Hazards for Branches

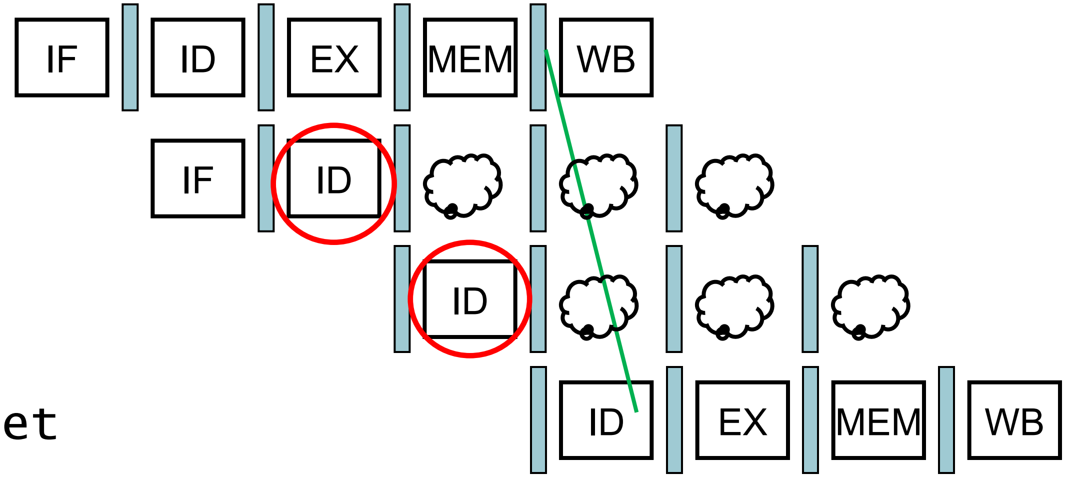
- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles

```
lw   $1, 0($8)

beq  stalled

beq  stalled

beq  $1, $0, target
```



More-Realistic Branch Prediction

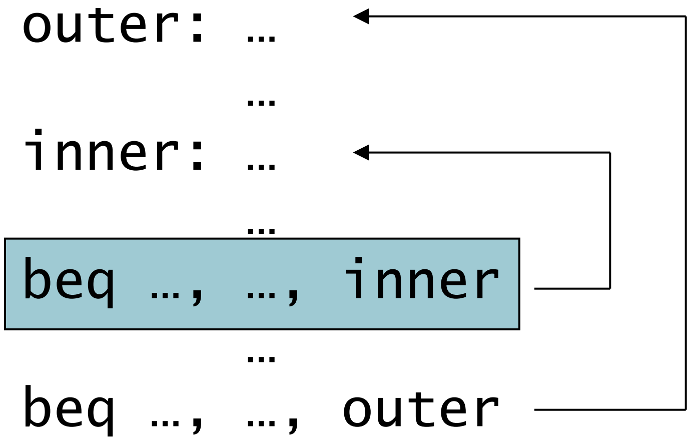
- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history

Dynamic Branch Prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction

1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!

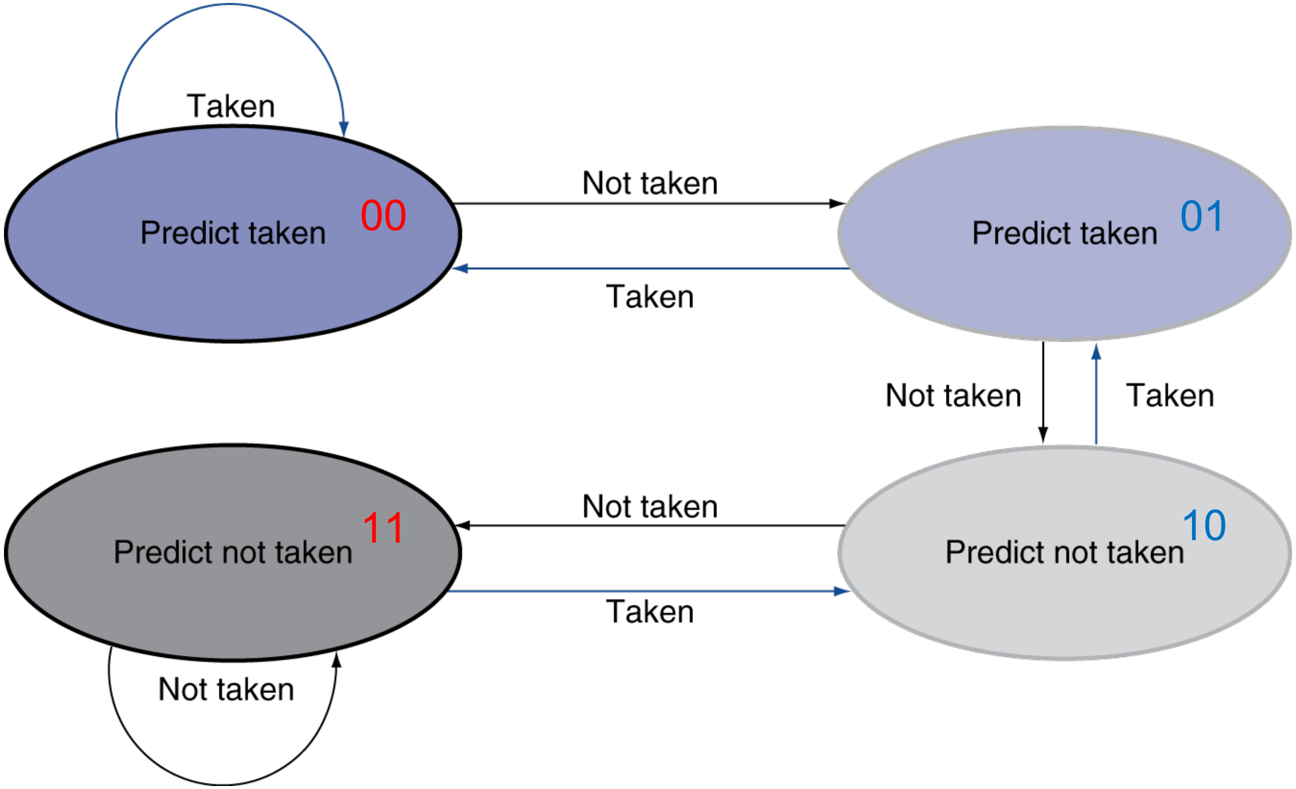


```
outer: …
       …
inner: …
       …
beq …, …, inner
       …
beq …, …, outer
```

- Mispredict as taken on last iteration of inner loop
- Then mispredict as not taken on first iteration of inner loop next time around

2-Bit Predictor

- Only change prediction on two successive mispredictions



Calculating the Branch Target

- Even with predictor, still need to calculate the target address
  - 1-cycle penalty for a taken branch
- Branch target buffer
  - Cache of target addresses
  - Indexed by PC when instruction fetched
    - If hit and instruction is branch predicted taken, can fetch target immediately

Branch instruction 의 빈도를 줄이는 방법

- conditional move instruction :
  - movn : move if not zero
  - movz : move if zero
  - `movn  $8, $11, $4 # MIPS later version`
  - `CSEL  X8, X11, X4, NE # ARMv8`

Pipeline Summary

The BIG Picture

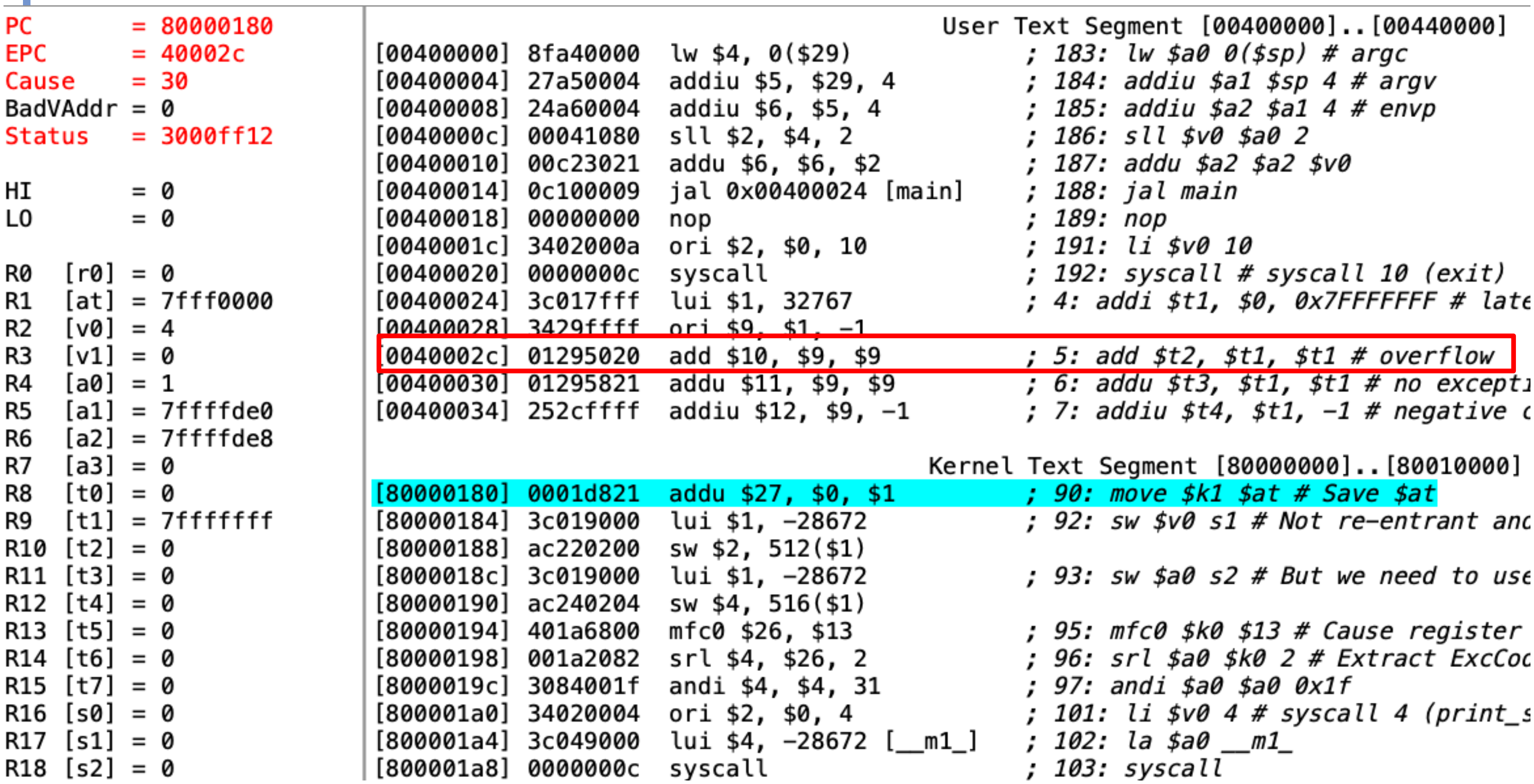
- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Subject to hazards
  - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

Exceptions and Interrupts

- “Unexpected” events requiring change in flow of control
  - Different ISAs use the terms differently
- Exception
  - Arises within the CPU
    - e.g., undefined opcode, overflow,…
- Interrupt
  - From an external I/O controller
    - e.g., system reset, I/O device request, …
- Dealing with them without sacrificing performance is hard

Handling Exceptions

- In MIPS, exceptions managed by a System Control Coprocessor (CP0)
- Save PC of offending (or interrupted) instruction
  - In MIPS: Exception Program Counter (EPC)
- Save indication of the problem
  - In MIPS: Cause register
- Jump to handler at 8000 0180



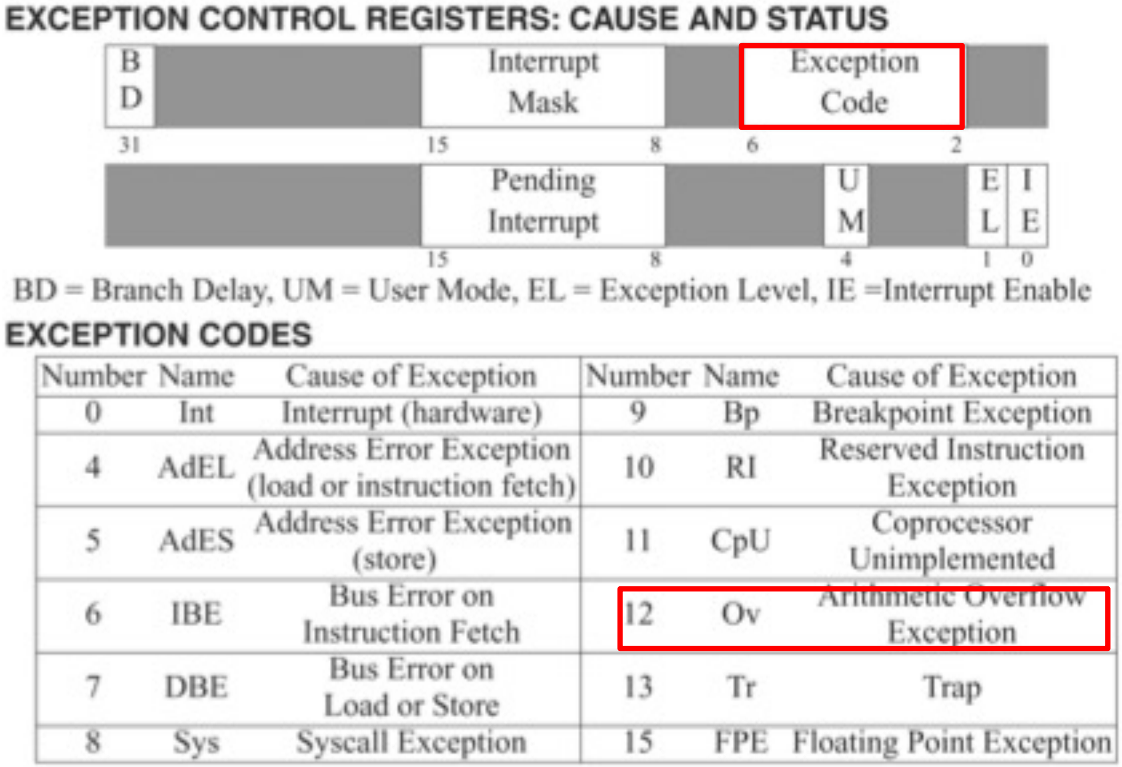
```
PC       = 80000180
EPC      = 40002c
Cause    = 30
BadVAddr = 0
Status   = 3000ff12

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 7fff0000
R2  [v0] = 4
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 7ffffde0
R6  [a2] = 7ffffde8
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 7fffffff
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
```

```
                                        User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)          ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4       ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4        ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2          ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2        ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]  ; 188: jal main
[00400018] 00000000  nop                    ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10         ; 191: li $v0 10
[00400020] 0000000c  syscall                ; 192: syscall # syscall 10 (exit)
[00400024] 3c017fff  lui $1, 32767          ; 4: addi $t1, $0, 0x7FFFFFFF # late
[00400028] 3429ffff  ori $9, $1, -1
[0040002c] 01295020  add $10, $9, $9        ; 5: add $t2, $t1, $t1 # overflow
[00400030] 01295821  addu $11, $9, $9       ; 6: addu $t3, $t1, $t1 # no excepti
[00400034] 252cffff  addiu $12, $9, -1      ; 7: addiu $t4, $t1, -1 # negative c

                                       Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1       ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672         ; 92: sw $v0 s1 # Not re-entrant and
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672         ; 93: sw $a0 s2 # But we need to use
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13          ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2         ; 96: srl $a0 $k0 2 # Extract ExcCod
[8000019c] 3084001f  andi $4, $4, 31        ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4          ; 101: li $v0 4 # syscall 4 (print_s
[800001a4] 3c049000  lui $4, -28672 [__m1_] ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                ; 103: syscall
```



EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS

B D | Interrupt Mask | Exception Code

31 15 8 6 2

Pending Interrupt | U M | E L | I E

15 8 4 1 0

BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =Interrupt Enable

EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

Vectored interrupt

- Vectored Interrupts
  - Handler address determined by the cause
- Example:

Undefined opcode:	C000 0000
Overflow:	C000 0020
…:	C000 0040

- Instructions in the handler routine either
  - Deal with the interrupt, or
  - Jump to real handler

Handler Actions

- Read cause, and transfer to relevant handler
- Determine action required
- If restartable
    - Take corrective action
    - use EPC to return to program
- Otherwise
    - Terminate program
    - Report error using EPC, cause, …

Exceptions in a Pipeline

- Another form of control hazard
- Consider overflow on add in EX stage

  `add $1, $2, $1`
  - Prevent $1 from being written
  - Complete previous instructions
  - Flush add and subsequent instructions
  - Set Cause and EPC register values
  - Transfer control to handler
- Similar to mispredicted branch
  - Use much of the same hardware

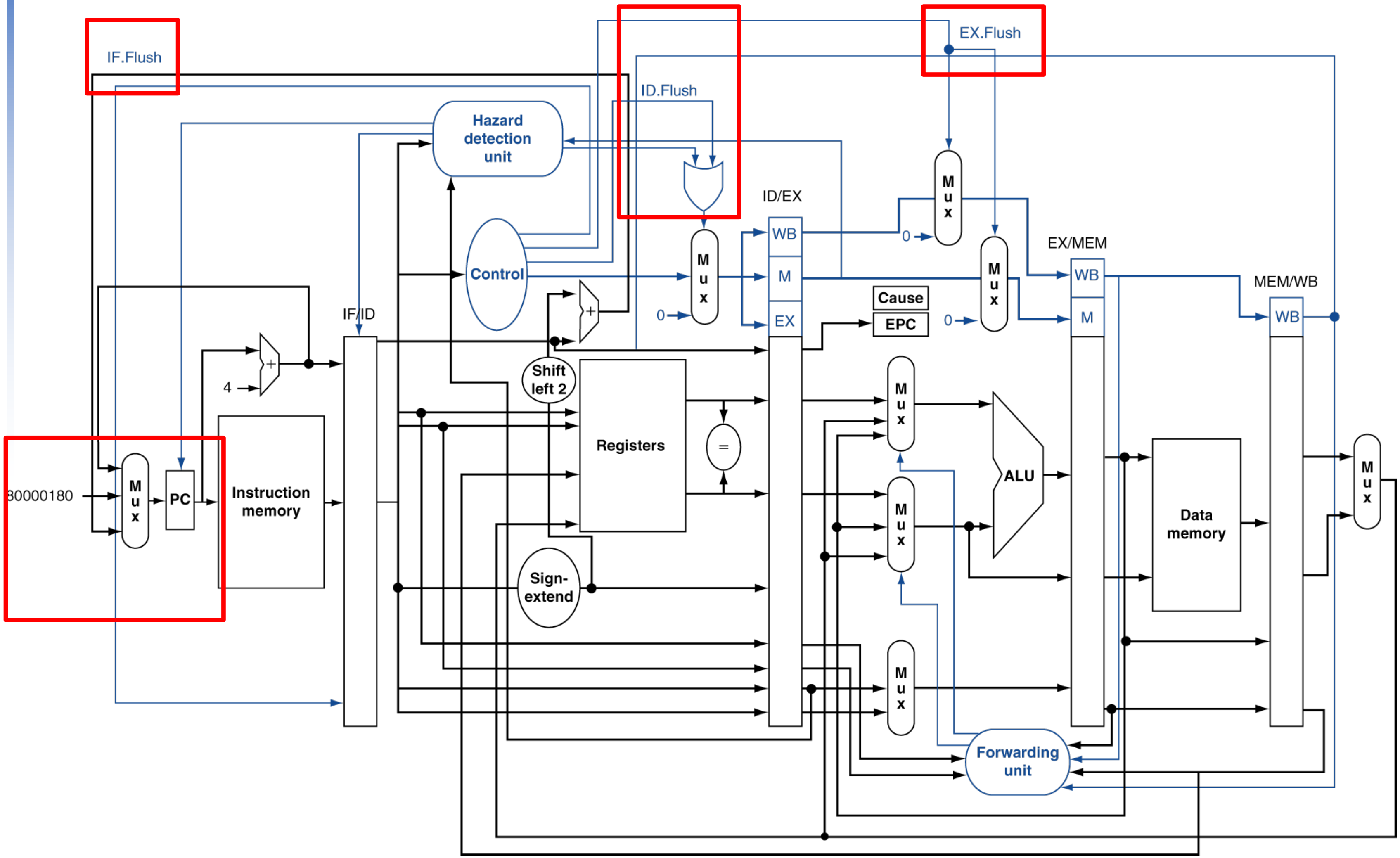
- Overflow Exception on add in

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

- Handler

-
  ```
  80000180 sw $25, 1000($0)
  80000184 sw $26, 1004($0)
  …
  ```

Pipeline with Exceptions



Exception Example

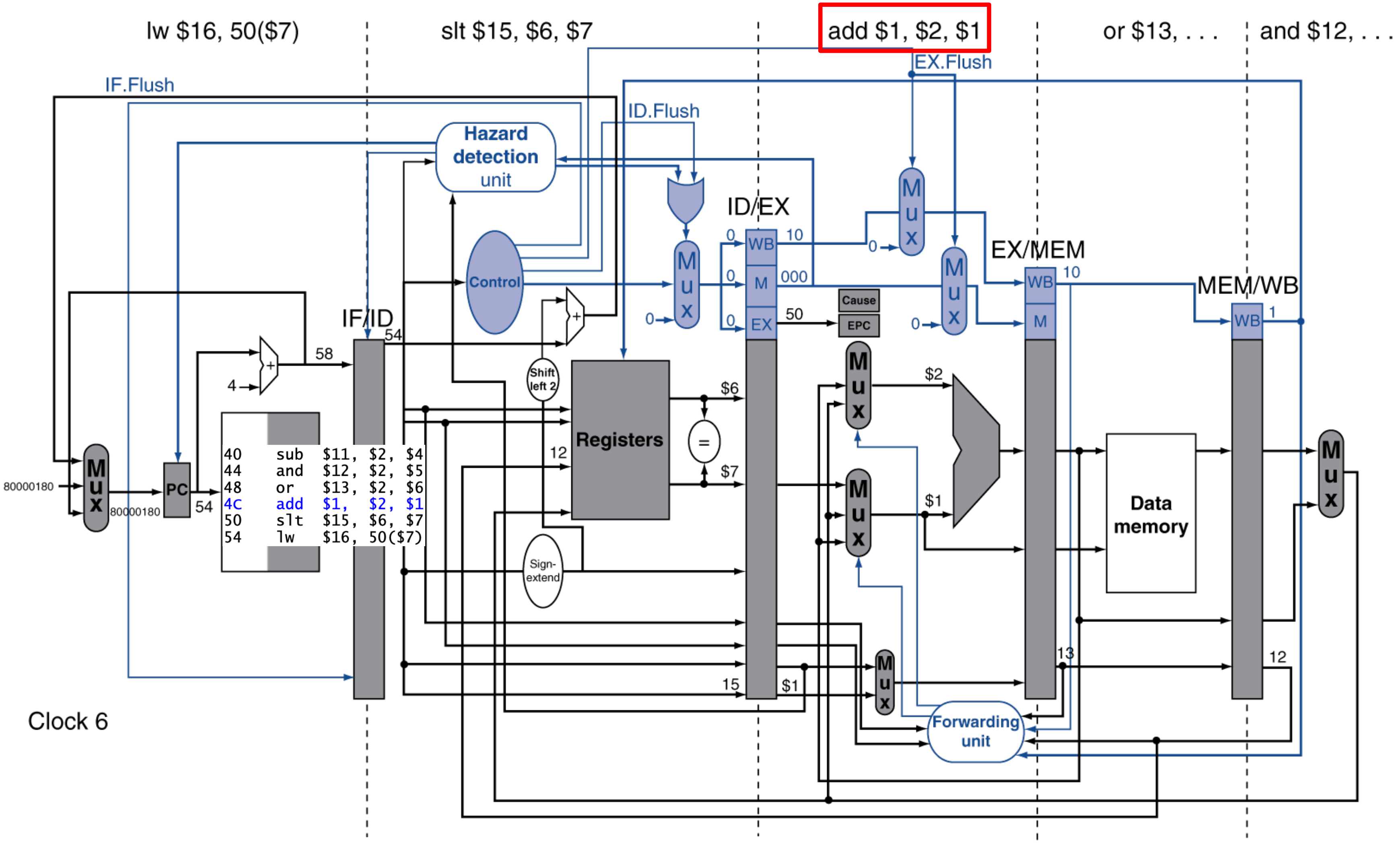
- Overflow Exception on add in

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

- Handler

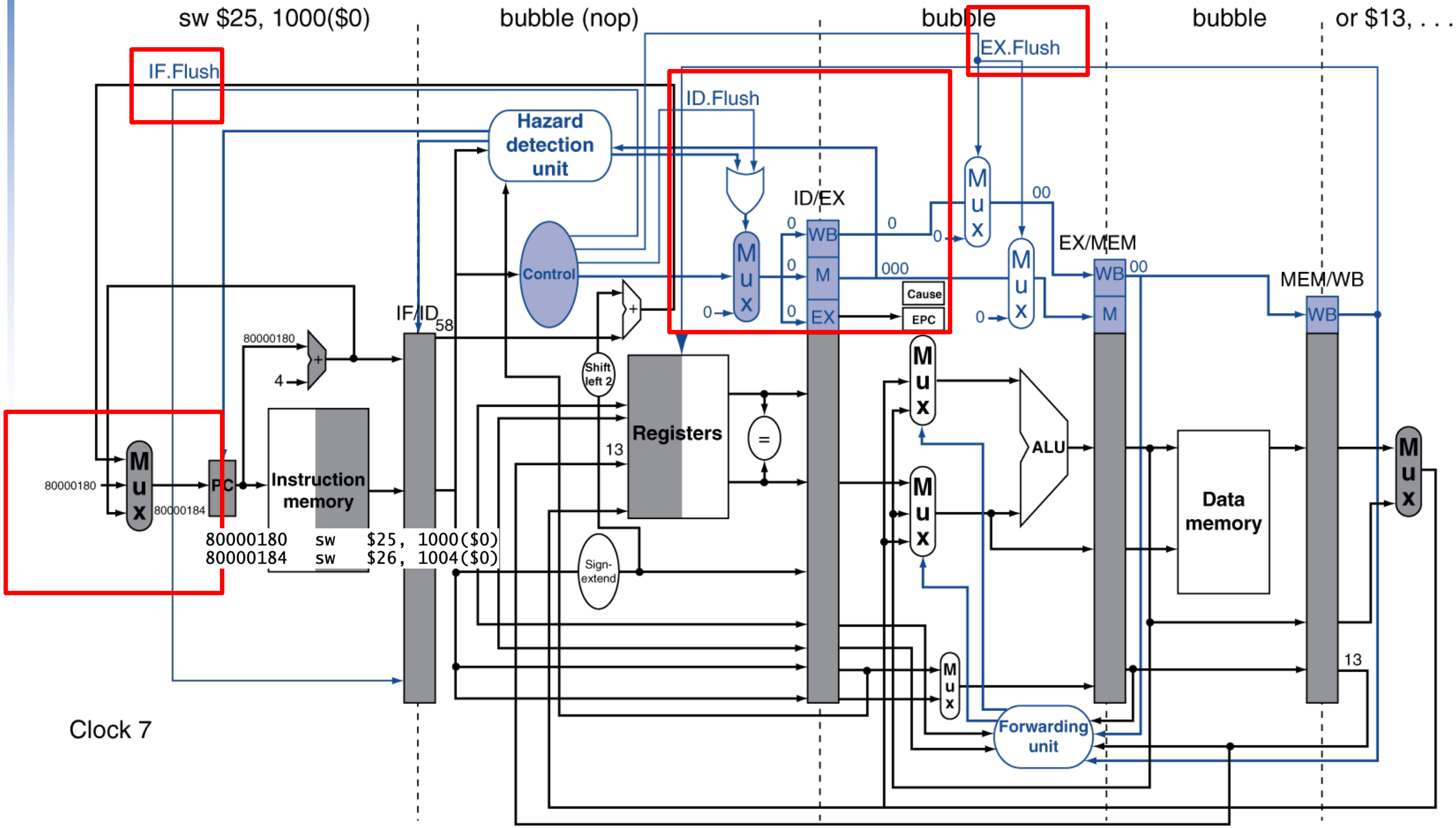
```
80000180   sw   $25, 1000($0)
80000184   sw   $26, 1004($0)
…
```

Exception Example : Overflow



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

Exception : flushing instructions



```
80000180    sw    $25, 1000($0)
80000184    sw    $26, 1004($0)
```

Multiple Exceptions

- Pipelining overlaps multiple instructions
  - Could have multiple exceptions at once
- Simple approach: deal with exception from earliest instruction
  - Flush subsequent instructions

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

Imprecise Exceptions

- Just stop pipeline and save 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