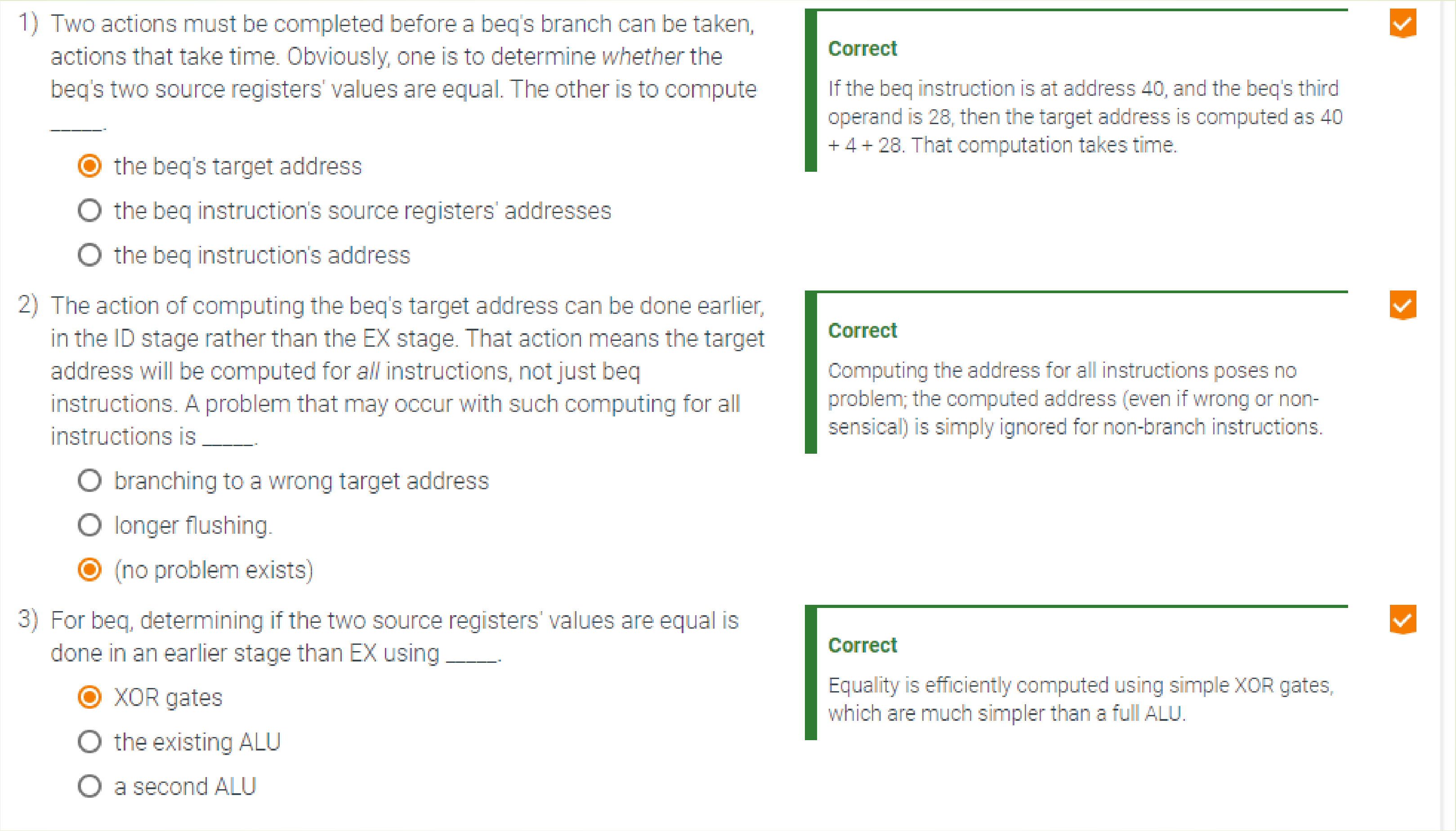
EGC442 Class Notes 4/12/2023

Baback Izadi
Division of Engineering Programs
bai@engr.newpaltz.edu

Test 2:

- Chapter 4
  - ALU design
- Chapter 5
  - Design of data path and control
  - Pipelined processor
  - Correcting for various hazards
  - Advanced pipeline concepts



1) Two actions must be completed before a beq's branch can be taken, actions that take time. Obviously, one is to determine *whether* the beq's two source registers' values are equal. The other is to compute _____.

● the beq's target address

○ the beq instruction's source registers' addresses

○ the beq instruction's address

Correct ✓

If the beq instruction is at address 40, and the beq's third operand is 28, then the target address is computed as 40 + 4 + 28. That computation takes time.

2) The action of computing the beq's target address can be done earlier, in the ID stage rather than the EX stage. That action means the target address will be computed for *all* instructions, not just beq instructions. A problem that may occur with such computing for all instructions is _____.

○ branching to a wrong target address

○ longer flushing.

● (no problem exists)

Correct ✓

Computing the address for all instructions poses no problem; the computed address (even if wrong or non-sensical) is simply ignored for non-branch instructions.

3) For beq, determining if the two source registers' values are equal is done in an earlier stage than EX using _____.

● XOR gates

○ the existing ALU

○ a second ALU

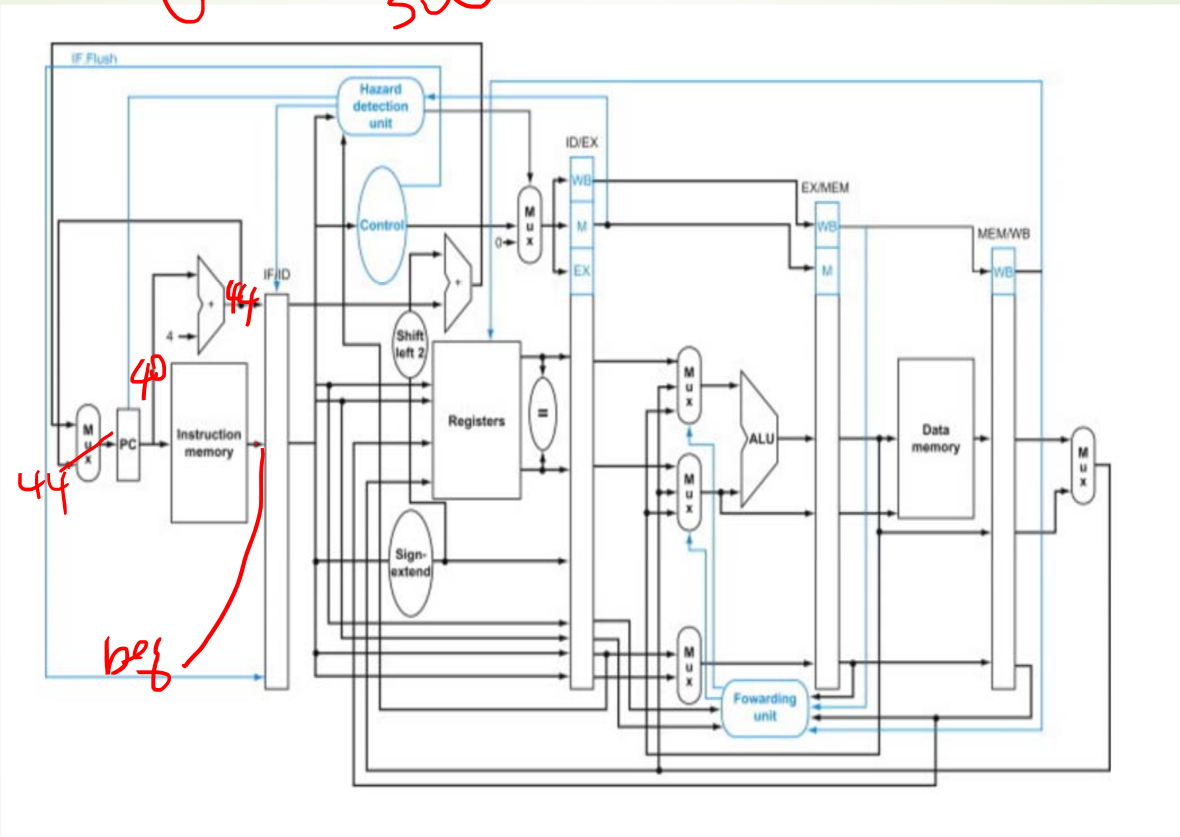
Correct ✓

Equality is efficiently computed using simple XOR gates, which are much simpler than a full ALU.

```
32   sub  $10, $4, $8
36   sw   $2, 2($8)
40   beq  $2,  $4, 3
44   and  $12, $2, $5
48   or   $13, $2, $6
52   add  $14, $4, $2
56   slt  $15, $6, $7
```

beq

sw

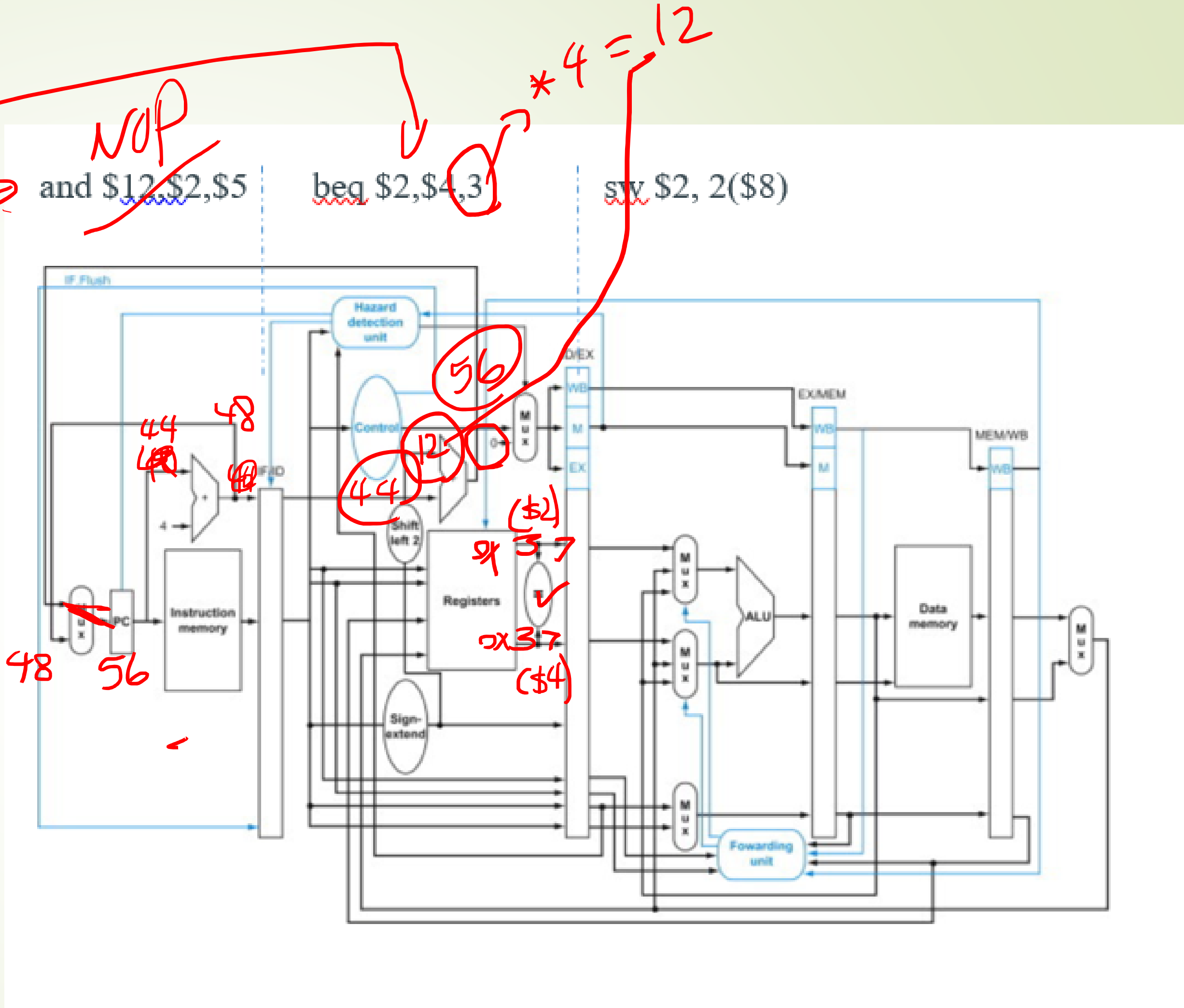


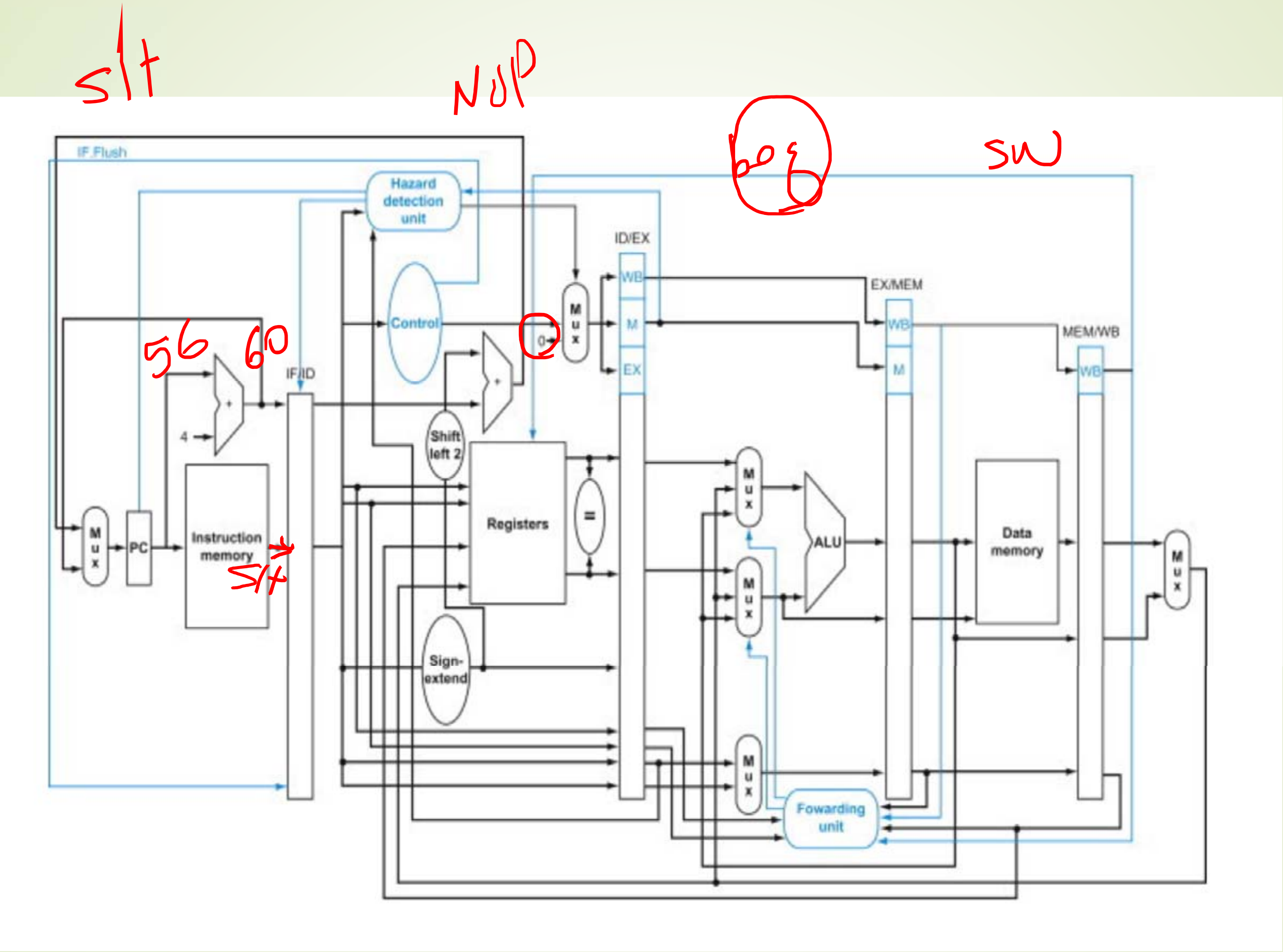
4) Assume the following sequence of instructions

```
32   sub   $10, $4, $8
36   sw    $2, 2($8)
40   beq   $2,  $4, 3
44   and   $12, $2, $5
48   or    $13, $2, $6
52   add   $14, $4, $2
56   slt   $15, $6, $7
```

a. Using the following diagram, assuming ($2)= 0x37 ($4)= 0x37, show the next three cycling steps:

b. Repeat a for ($2)= 0x37 ($4)= 0x7



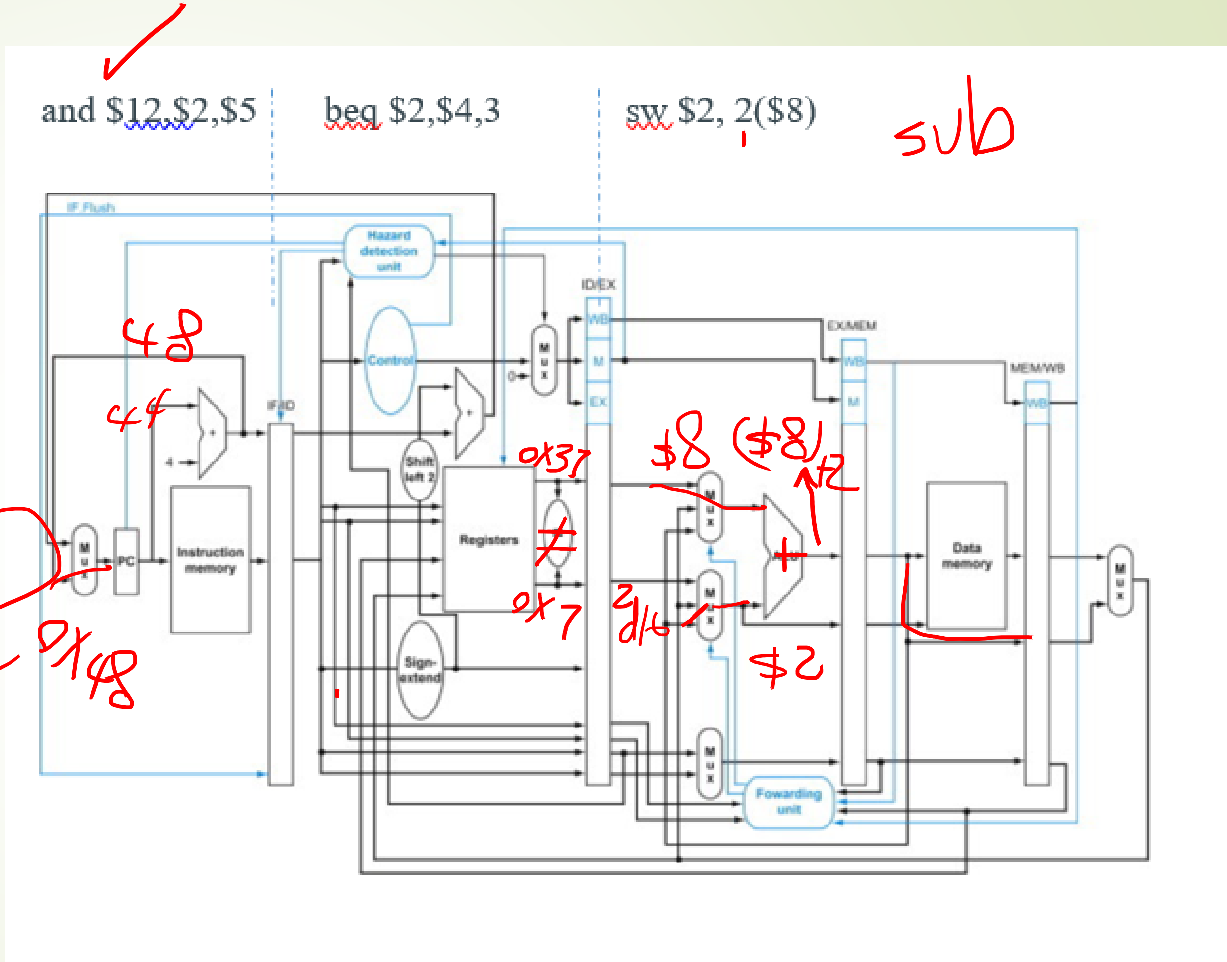
slt
NOP
beq
sw
IF.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
WB
M
WB
56
60
IF/ID
0
M u x
M u x
PC
4
+
+
Instruction memory
slt
Shift left 2
Registers
=
Sign-extend
M u x
ALU
M u x
Data memory
M u x
M u x
Fowarding unit

4) Assume the following sequence of instructions

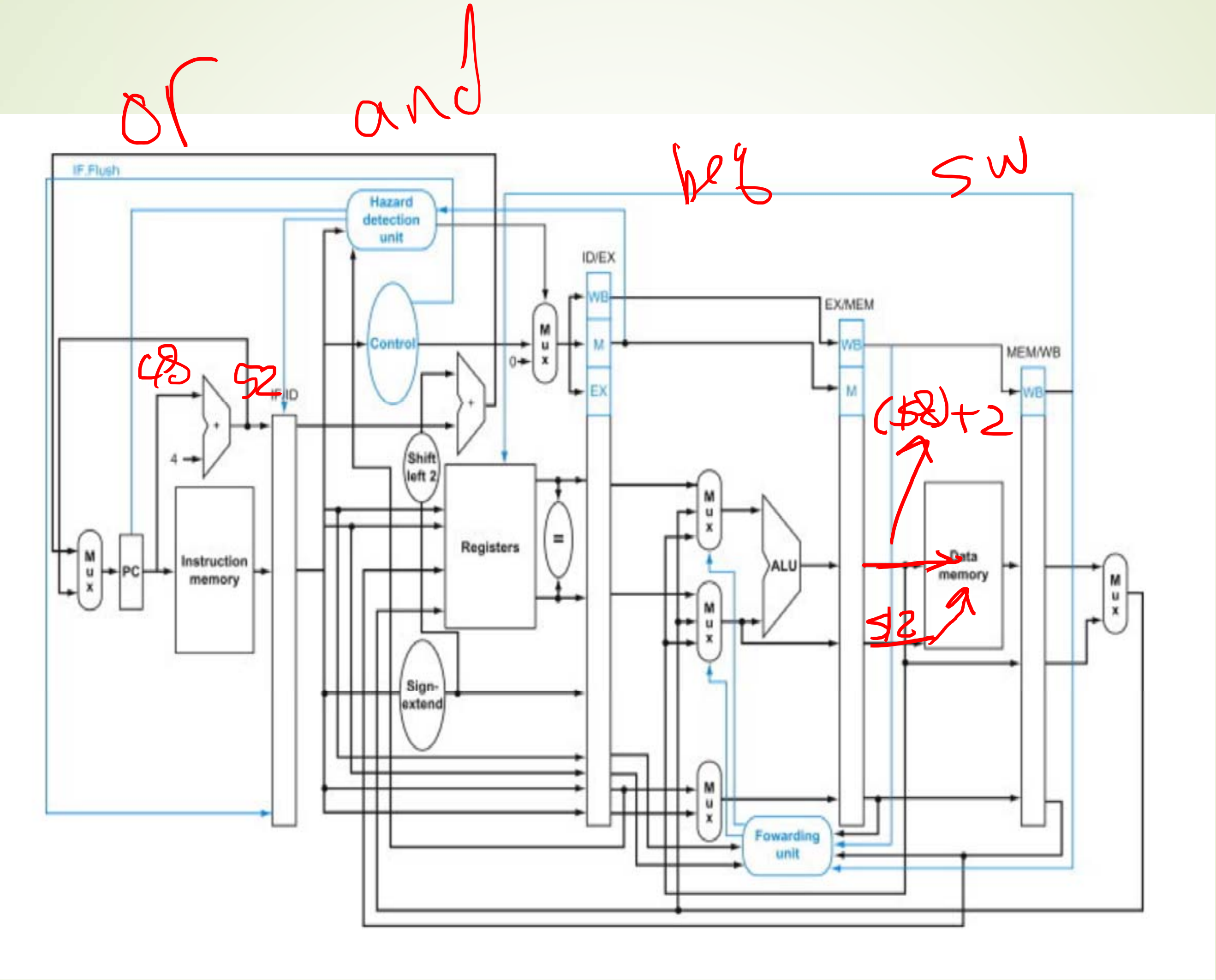
32 sub $10, $4, $8
36 sw $2, 2($8)
40 beq ~~$2, $4, 3~~
44 and $12, $2, $5
48 or $13, $2, $6
52 add $14, $4, $2
56 slt $15, $6, $7

Using the following diagram, assuming ($2)= 0x37 ($4)= 0x37, show the next three cycling steps;

Repeat a for ($2)= 0x37 ($4)= 0x7


and $12,$2,$5 | beq $2,$4,3 | sw $2, 2($8)

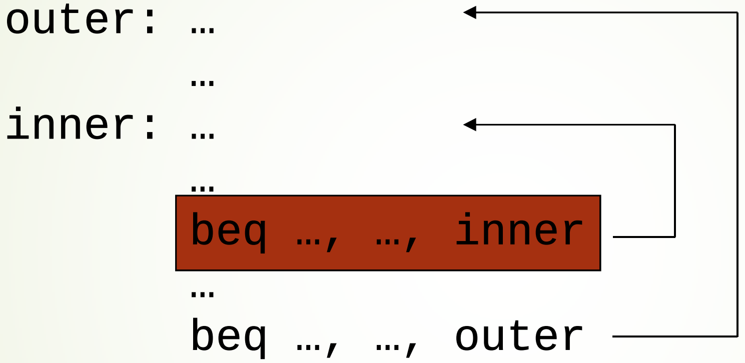
sub


or
and
beq
sw
IF.Flush
Hazard detection unit
ID/EX
EX/MEM
MEM/WB
WB
M
EX
Control
0
M u x
48
52
IF/ID
4
Shift left 2
Registers
=
PC
Instruction memory
Sign-extend
ALU
($8)+2
$2
Data memory
Fowarding unit

1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!

for(i=0; i<100; i++){



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

Not branch

}

100

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

and

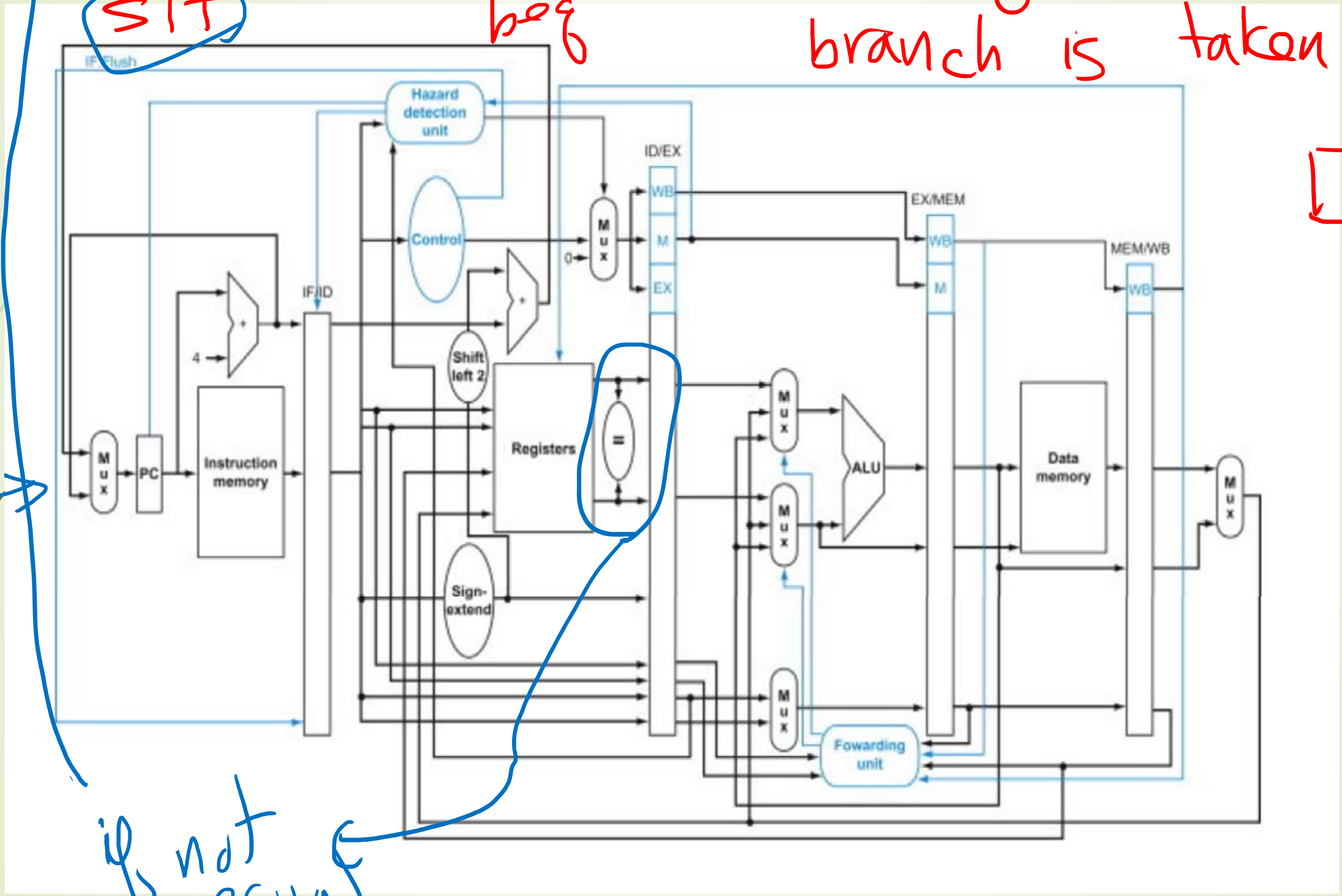
change it to NOP

slt

beq

assuming that branch is taken

branch



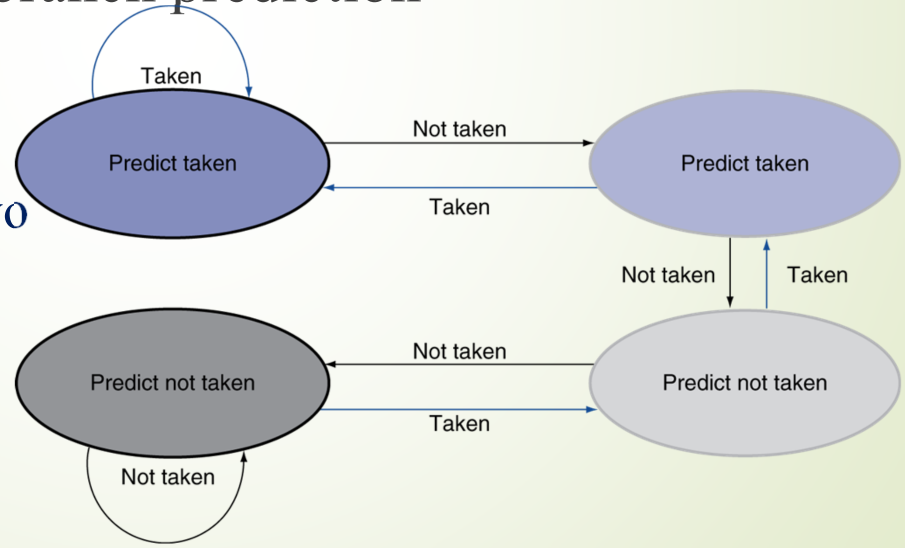
144

if not equal

2-Bit Predictor

- If the branch is taken, we have a penalty of one cycle
- For our simple design, this is reasonable
- With deeper pipelines, penalty increases and static branch prediction drastically hurts performance
- Solution: Use 2-bit branch prediction

Only change prediction on two successive mispredictions



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

Branch Prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use behavioral branch 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