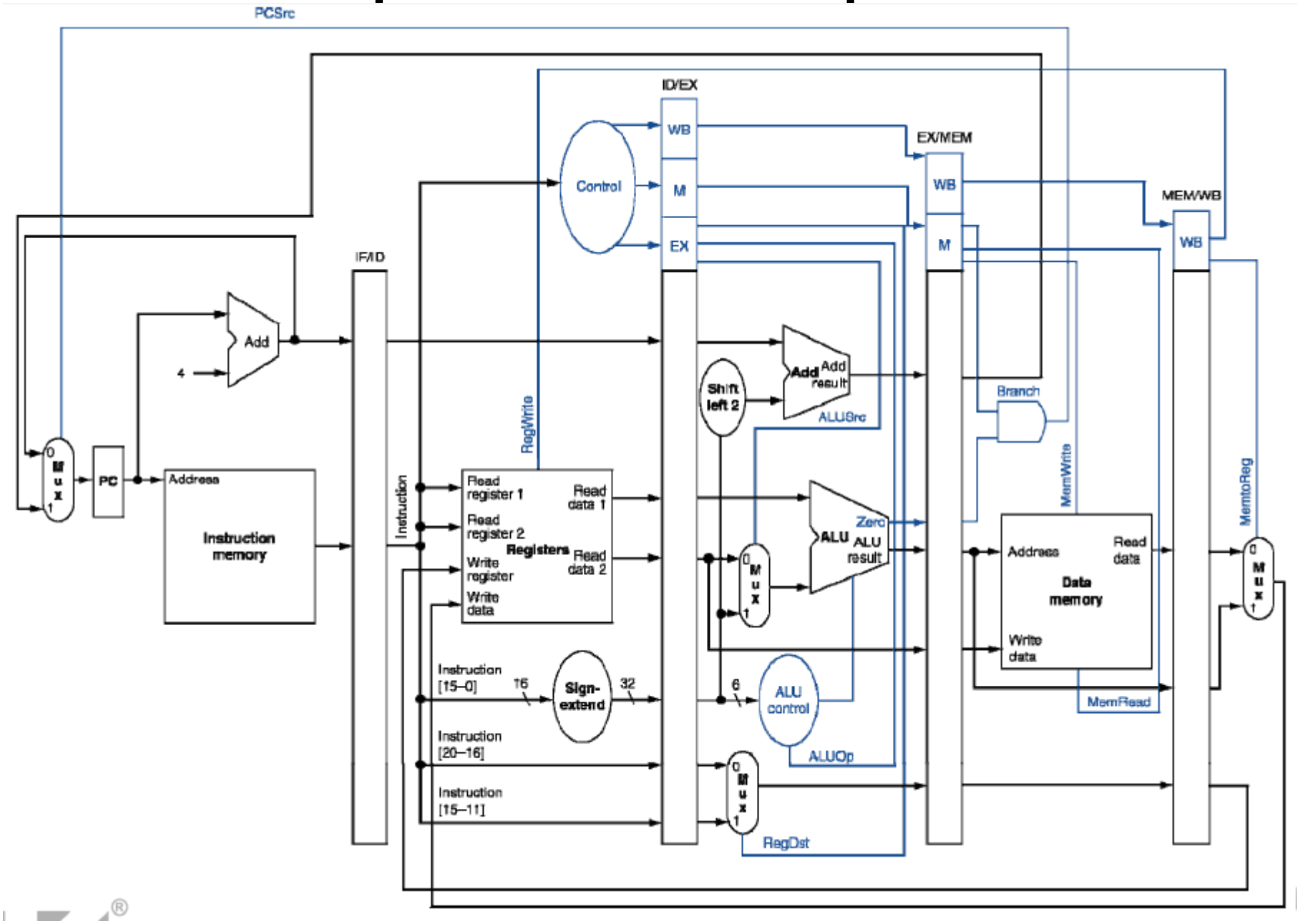
# CS M151B:
# Computer Systems Architecture

## Week 6, Part 2

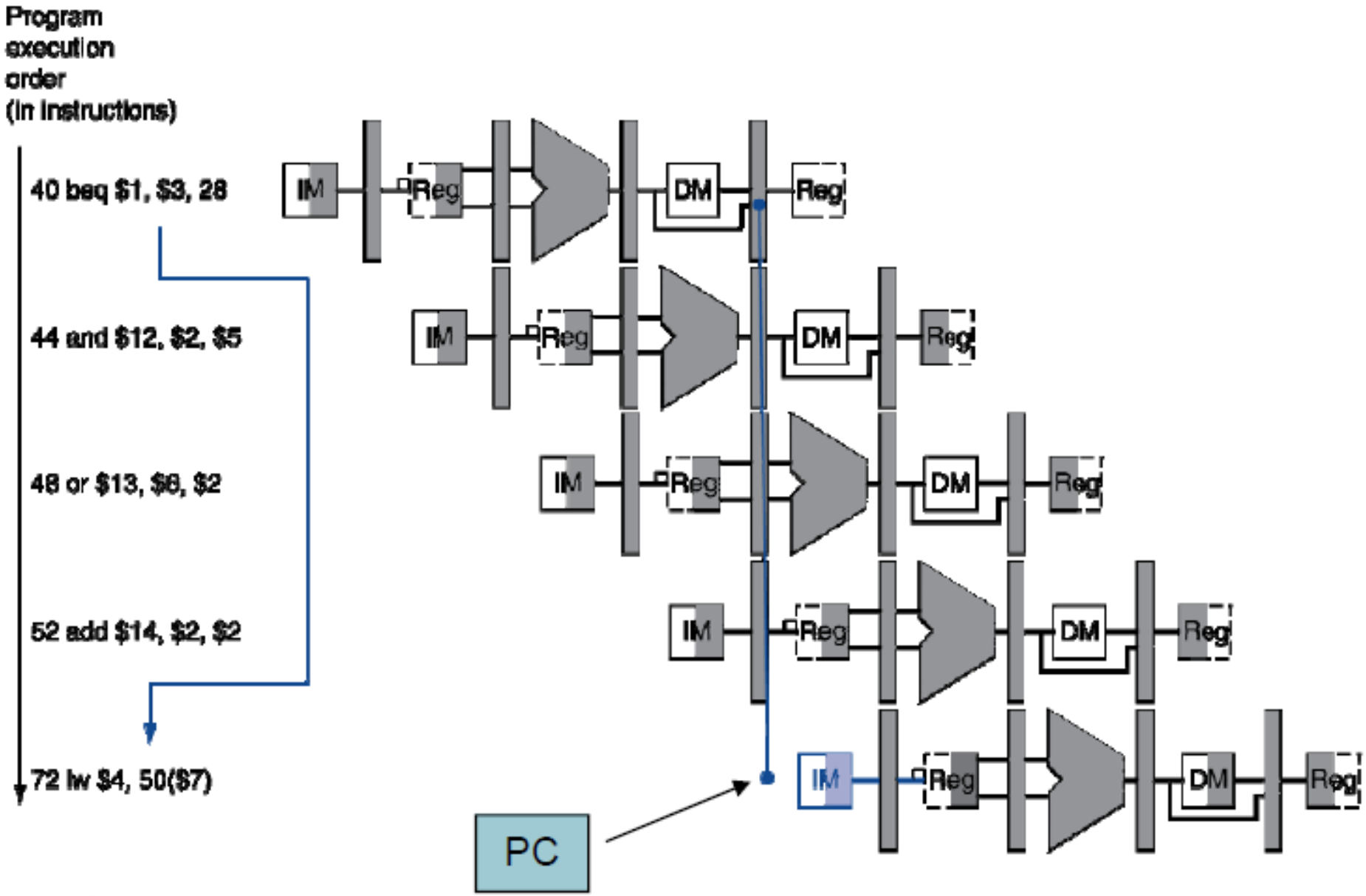
# Pipelined Datapath



# Control Hazards

- In this pipeline, whether you are to branch is calculated over the course of several stages.
  - Fetch the registers in IF
  - Retrieve register values in ID
  - Subtract the two inputs in EX
  - Calculate branch and update PC in the MEM stage.
- This means that upon hitting a branch, you don't execute the actual branch until the WB stage of the branch instruction.

# Control Hazards



# Control Hazards

- But what was in the pipeline after the branch? Is it right?

# Control Hazards

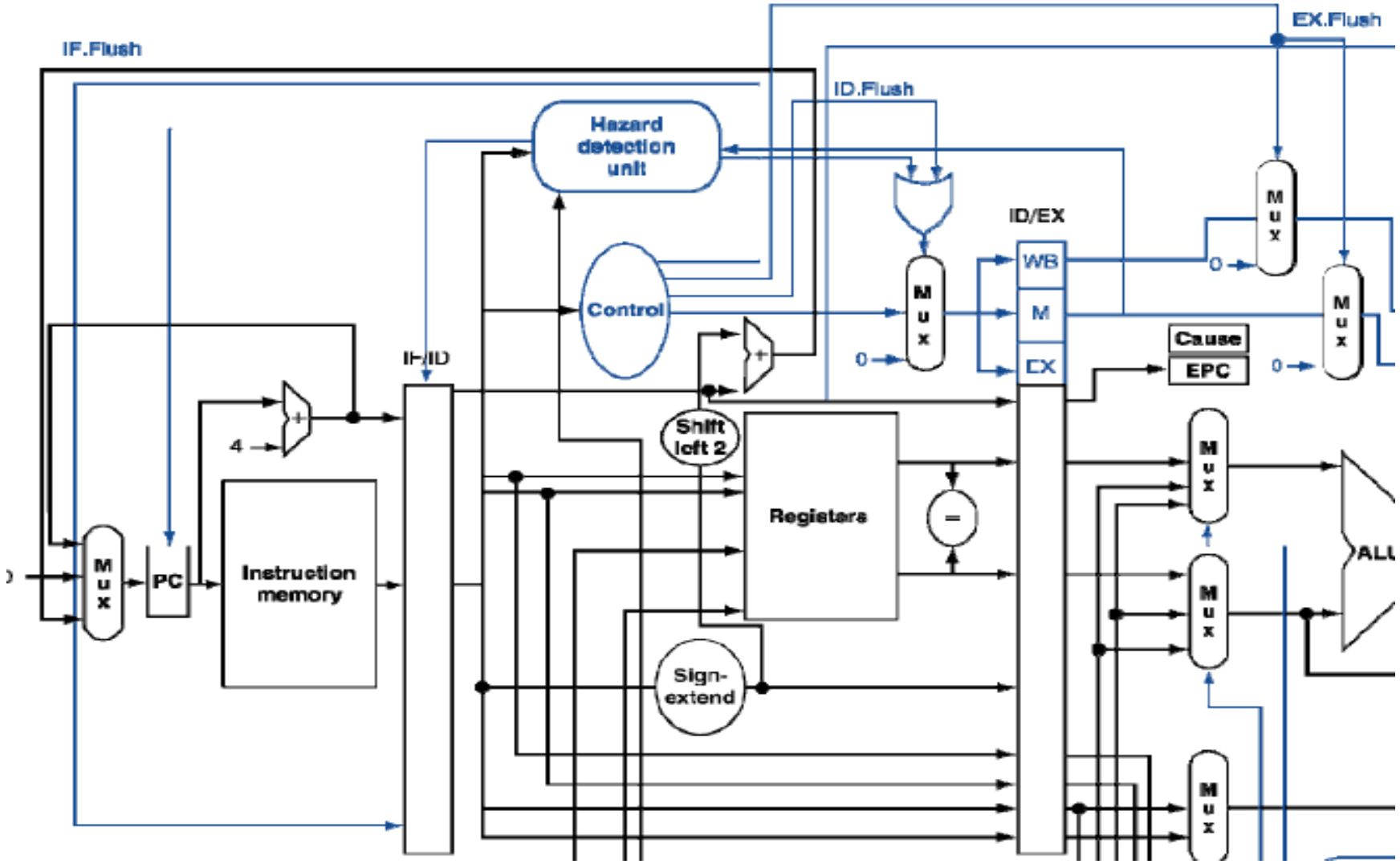
- But what was in the pipeline after the branch? Is it right?
  - By default in each cycle, PC + 4 is loaded into the PC.
  - It's correct if you weren't planning to branch.
  - But if you did branch, now you have some instruction in the ID stage, EX stage, and MEM (!) stage that you didn't plan to execute.
  - How do we fix this?

# Control Hazards: Resolving Incorrect Guesses

- Cancel a stage by flushing the pipeline by pushing all 0 control bits into the stage.
- We only have to account for ID, EX, and MEM stage.

# Control Hazards: Resolving Incorrect Guesses

- This simply means adding muxes to select 0's instead of the normal control signals



# Control Hazards

- This painlessly resolves the hazard and will produce accurate results.
- But because we're not happy simply with correctness.
- We need the speed.
- Is this fast?

# Control Hazards

- Each branch not taken means no penalty.
- Each branch taken means a penalty of 3 cycles before anything useful comes out of the pipeline again.
- That's not good enough.

# Control Hazards

- The problem can be examined from two angles
  - Frequency of Impact
  - Penalty

# Control Hazards

- Frequency of Impact
  - “How often you must pay the cost of a branch delay”
- Influenced by:
  - How you manage branching
  - Frequency of branches that appear in the code

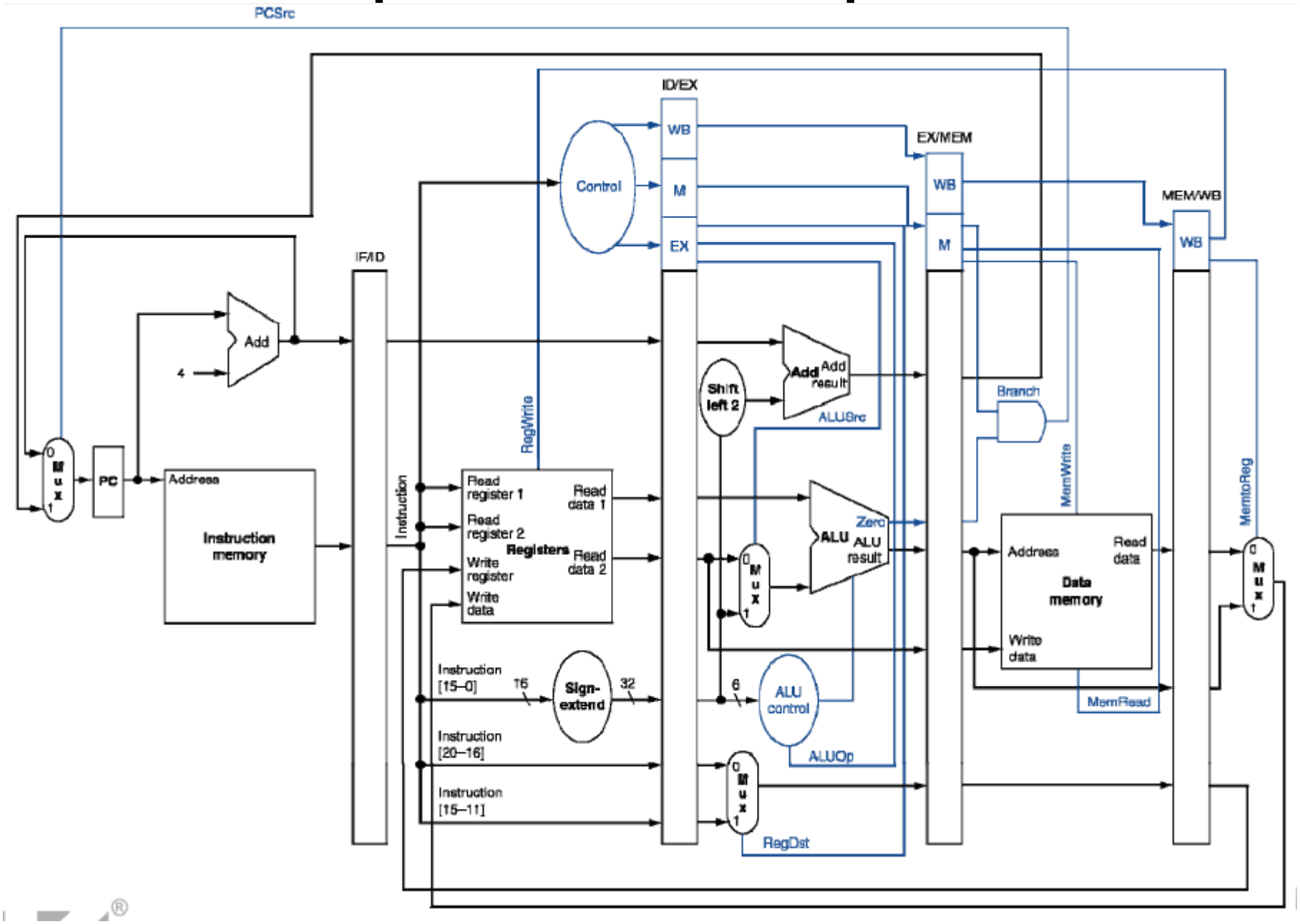
# Control Hazards

- Penalty
  - “What it costs when you're wrong about a prediction”
- Influenced by
  - How you manage the branches
  - Which stage that the branch is actually taken (AKA “Pipeline branch resolution location”)

# Control Hazards: Reducing Penalty

- Consider the “Pipeline branch resolution location” of the pipeline so far. The PC is set to branch in the MEM stage and taken during the WB stage.
- This means the penalty is 3 cycles.
- Can we improve the penalty by moving up the “Pipeline branch resolution location” (or... PBR location)?

# Pipelined Datapath



# Control Hazards: Reducing Penalty

- There's not really any reason that we need to set the PC for a branch in the MEM stage rather than the EX stage where the decision is actually made. (Except maybe reducing the amount of serial work that is done in the EX stage).
- Let's say pipeline resolution is in EX stage instead.
- Then Branch is taken in MEM stage.
- All we need to do is move the AND(Zero, Branch) gate to the EX stage.

# Control Hazards: Reducing Penalty

- That works fine. Now the delay is only two cycles.
- Why stop there?
- Let's keep moving it back, now to the ID stage.
- Is it as straightforward?

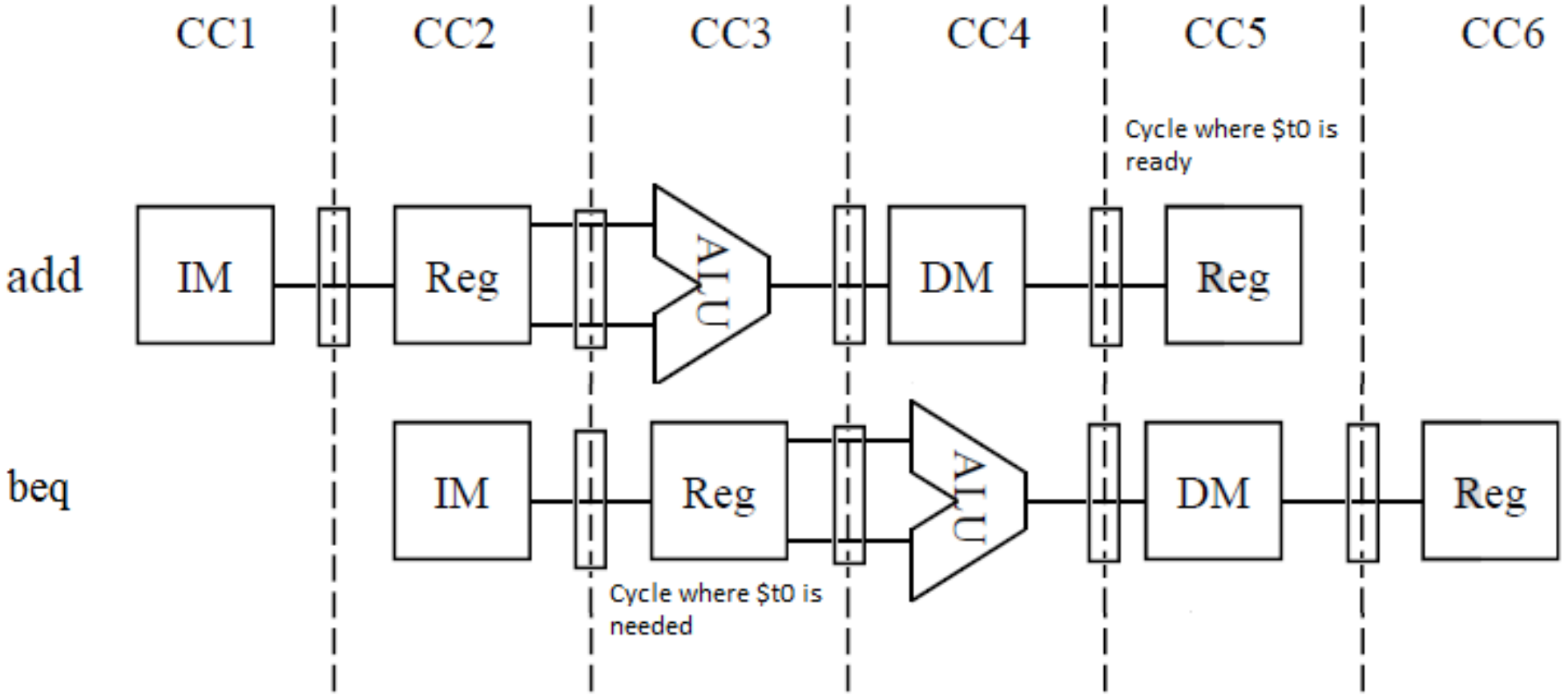
# Control Hazards: Reducing Penalty

- Is it straightfoward?
  - The ALU is in the EX stage so we'll need a new specialized comparator unit and ALU for (SE(I) + PC + 4) in the ID stage.
  - That's not too bad, now we use the comparator to determine if we want to branch in the ID stage.
  - The branch is executed in the EX stage
  - Only one cycle! That's pretty good.
  - And it was simple! Just like everything else in this chapter! Sometimes life is good.

# Control Hazards: Reducing Penalty

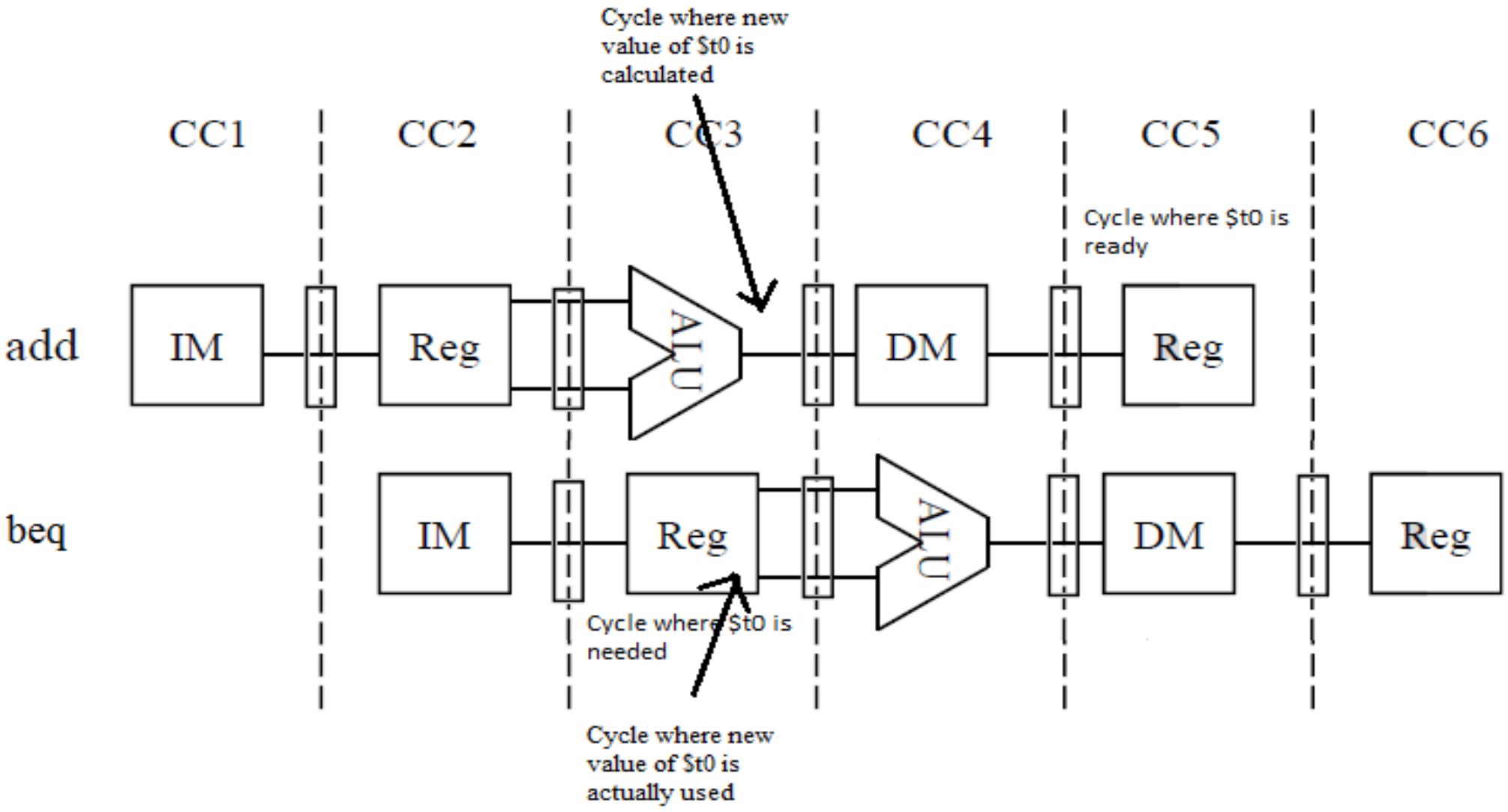
- Consider:
- add $t0, $t1, $t1
- beq $t0, $t2, LABEL

# Control Hazards: Reducing Penalty



# Control Hazards: Reducing Penalty

- Son of a...



# Control Hazards: Reducing Penalty

- This is a problem for two reason.

1. We have to forward data from the EX/MEM latch of the first instruction. By the time we do that, the second instruction is in the EX stage and by then it's too late. We have to stall.
2. Even if we do stall, we haven't implemented forwarding techniques to forward to the ID stage.

- Ugh. Can't we just ignore this?

# Control Hazards: Reducing Penalty

- For once, yes.
- We've finally reached the apathy threshold. It's messy enough were we don't care about the details.
- The point of this is to demonstrate that this method of improving branches will work, but will add complexity.
- As you can imagine, the new forwarding mechanism is just gonna be a bunch of muxes and the stalling mechanism is another signal that can tell existing muxes to insert a stall.

# Control Hazards

- Can we approach the topic of improving on branches from some other perspective?
- Let's focus our attention on how we manage branches.
- What happens when we reach a branch determines whether we need to incur a branch penalty.

# Control Hazards: Prediction

- What actions can we take when we reach a branch:
  - Always stall. This is a poor solution. Incur the cost at every branch. Also more complicated than simply always predicting branch not taken, which is done via normal execution.
  - Continue execution, but based on a prediction of whether or not we wanted to take the branch. This one is probably the winner.

# Control Hazards: Prediction

- Prediction
  - Software based static prediction
  - Hardware based dynamic prediction
  - Always taken/not taken

# Control Hazards: Static

- Software based static prediction
- Based on the static instructions.
- Ex:
  - Consider the direction of the branch. If the branch direction is to a higher memory address, then it is likely a loop. Loops are generally executed many times if at all, thus predict taken.

# Control Hazards: Static Prediction

- Static Prediction Tradeoffs?

# Control Hazards: Static Prediction

- Static Prediction tradeoffs?
  - Must consider the application/software.
  - Pro: simple, based on things like displacement and opcode.
  - Pro(?): Consistent, won't change behavior during execution.
  - Con: The static code can be very different from the actual instructions executed during instruction.
  - ...maybe we want the prediction to change during execution.

# Control Hazards: Dynamic Prediction

- Hardware based dynamic branch prediction
  - Don't need to examine software. Instead, predict the branches based on information gathered during execution.
  - General idea: hardware maintains a table. When a branch occurs, consult the table to see if that particular branch should be taken. Update the table according to if the branch was/was not taken.
  - Note this tradeoff. A table needs to be physically implemented in hardware.

# Control Hazards: Dynamic Prediction

- 1-bit predictor
  - Have a hash table that you can access into via the address of the offending branch.
  - For each branch, the table contains a bit. If the bit is 1, predict that the branch will be taken. If the bit is 0, predict the branch will not be taken.
  - If the branch is actually taken set the bit to 1. If the branch isn't taken, set the bit to 0.

# Control Hazards: Dynamic Prediction

- 1-bit predictor tradeoffs?

# Control Hazards: Dynamic Prediction

- 1-bit predictor tradeoffs?
  - Use of a hash table implies that there could be collisions. Be aware, but don't have to worry about this.
  - Based only on previous action without considering past history. There's no “hysteresis”.
  - A branch that is taken many times will not bias the decision any more than if that branch that is not taken once.

# Control Hazards: Dynamic Prediction

- 1-bit predictor tradeoffs?
  - Thrashing

```
while(...)
{
   if(i%2 == 0)
   {
     <branch not taken>
   }
   <branch taken>
   i++;
}
```

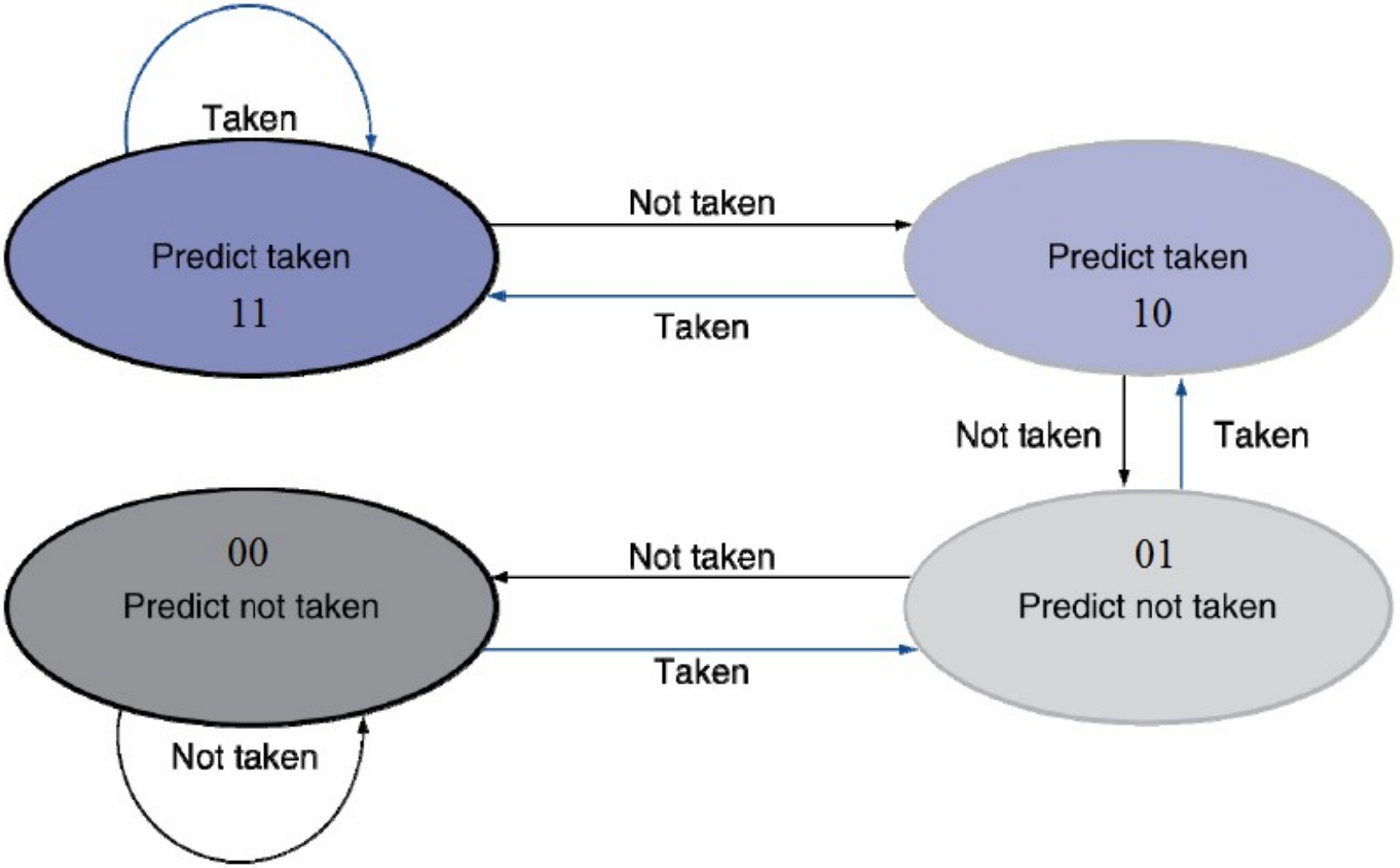
At best, this will be correct once and then always wrong afterwards. Otherwise, it is always wrong.

# Control Hazards: Dynamic Prediction

- We don't have enough hysteresis!
- It's not enough to just look at the last action.
- We must make the drastic move of looking at... the second to last action.

# Control Hazards: Dynamic Prediction

- 2-Bit Predictor:



# Control Hazards: Dynamic Prediction

- 2-Bit Predictor:
  - Now we have a little hysteresis.
  - If the same action is taken for a branch twice in a row, we can bias it so that we will predict taken until we have two not takens in a row
  - Due to the bias, this predictor may perform with 50% accuracy rather than ~0% (it may still be 0% though based on initial prediction).

# Control Hazards: Dynamic Prediction

Consider:

```
LOOP:
lw $t0, 0($t1)
lw $t2, 8($t0)
beq $t2, $s0, EXIT
lw $t1, 16($t0)
bne $t1, $s1, LOOP
EXIT:
sw $t0, 0($s2)
```

- Full forwarding
- Branches resolved in MEM (instruction branched to executes in WB)
- 2-bit dynamic branch prediction, all init 00.
- beq not taken twice.
- bne taken first time, not taken second time

# Control Hazards: Dynamic Prediction

Consider:

```
LOOP:
lw $t0, 0($t1)
lw $t2, 8($t0)
beq $t2, $s0, EXIT
lw $t1, 16($t0)
bne $t1, $s1, LOOP
EXIT:
sw $t0, 0($s2)
```

- beq not taken twice.
- bne taken first time, not taken second time
- How are the branches predicted?

# Control Hazards: Dynamic Prediction

Consider:

```
LOOP:
lw $t0, 0($t1)
lw $t2, 8($t0)
beq $t2, $s0, EXIT
lw $t1, 16($t0)
bne $t1, $s1, LOOP
EXIT:
sw $t0, 0($s2)
```

- beq not taken twice.
- bne taken first time, not taken second time
- How are the branches predicted?
  - beq 1st time 00, Pred = NT, Actual = NT
  - beq 2nd time 00, Pred = NT, Actual = NT
  - bne 1st time 00, Pred = NT,
  - Actual = T
  - bne 2nd time, 01, Pred = NT,
  - Actaul = NT

# Control Hazards: Dynamic Prediction

Consider:

```
LOOP:
lw $t0, 0($t1)
lw $t2, 8($t0)
beq $t2, $s0, EXIT
lw $t1, 16($t0)
bne $t1, $s1, LOOP
EXIT:
sw $t0, 0($s2)
```

Actual:

```
lw $t0, 0($t1)
lw $t2, 8($t0)
beq $t2, $s0, EXIT
lw $t1, 16($t0)
bne $t1, $s1, LOOP
sw $t0, 0($s2) //nulled
next insn //nulled
next insn //nulled
lw $t0, 0($t1)
lw $t2, 8($t0)
beq $t2, $s0, EXIT
lw $t1, 16($t0)
bne $t1, $s1, LOOP
sw $t0, 0($s2)
```

# Control Hazards: Dynamic Prediction

Actual:

Dependencies?

```
lw $t0, 0($t1)
lw $t2, 8($t0)
beq $t2, $s0, EXIT
lw $t1, 16($t0)
bne $t1, $s1, LOOP
sw $t0, 0($s2) //nulled
next insn //nulled
next insn //nulled
lw $t0, 0($t1)
lw $t2, 8($t0)
beq $t2, $s0, EXIT
lw $t1, 16($t0)
bne $t1, $s1, LOOP
sw $t0, 0($s2)
```

# Control Hazards: Dynamic Prediction

Actual:

```
lw $t0, 0($t1)
lw $t2, 8($t0)
beq $t2, $s0, EXIT
lw $t1, 16($t0)
bne $t1, $s1, LOOP
sw $t0, 0($s2) //nulled
next insn //nulled
next insn //nulled
lw $t0, 0($t1)
lw $t2, 8($t0)
beq $t2, $s0, EXIT
lw $t1, 16($t0)
bne $t1, $s1, LOOP
sw $t0, 0($s2)
```

Dependencies?
Consult examples...

End of

# The Sixth Week

-Four Weeks Remain-