Branch Hazards

Consider executing this sequence of instructions in the pipeline:

```
address          instruction
-------------------------------
     36:          sub  $10, $4, $8
     40:          beq  $1,  $3, L1
     44:          and  $12, $2, $5
     48:          or   $13,$12,$13
     52:          add  $14, $4, $2
     56:          slt  $15, $6, $7
     ...
     72:    L1:   lw   $4, 50($7)
```

When `beq` moves into the ID stage:

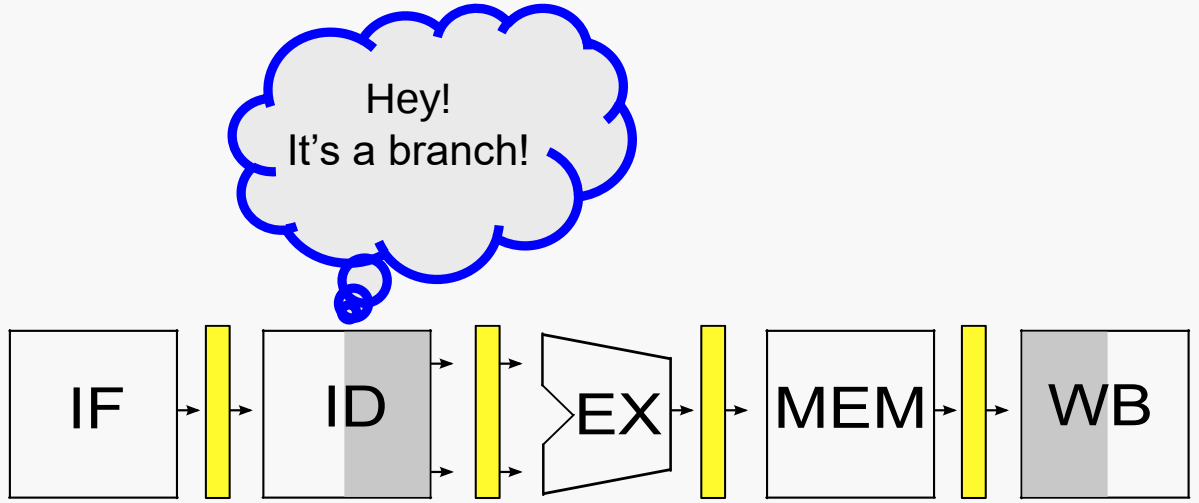
- should we fetch the instruction at address 44?
- it may or may not be executed next, depending on whether $1 == $3
- we don't know the address of the branch target instruction yet anyway

©2005-2020 WD McQuain

Branch Hazards

When `beq` enters the ID stage, we don't even know it is a conditional branch.

And… we won't know if the branch should be taken until `beq` reaches the end of EX.

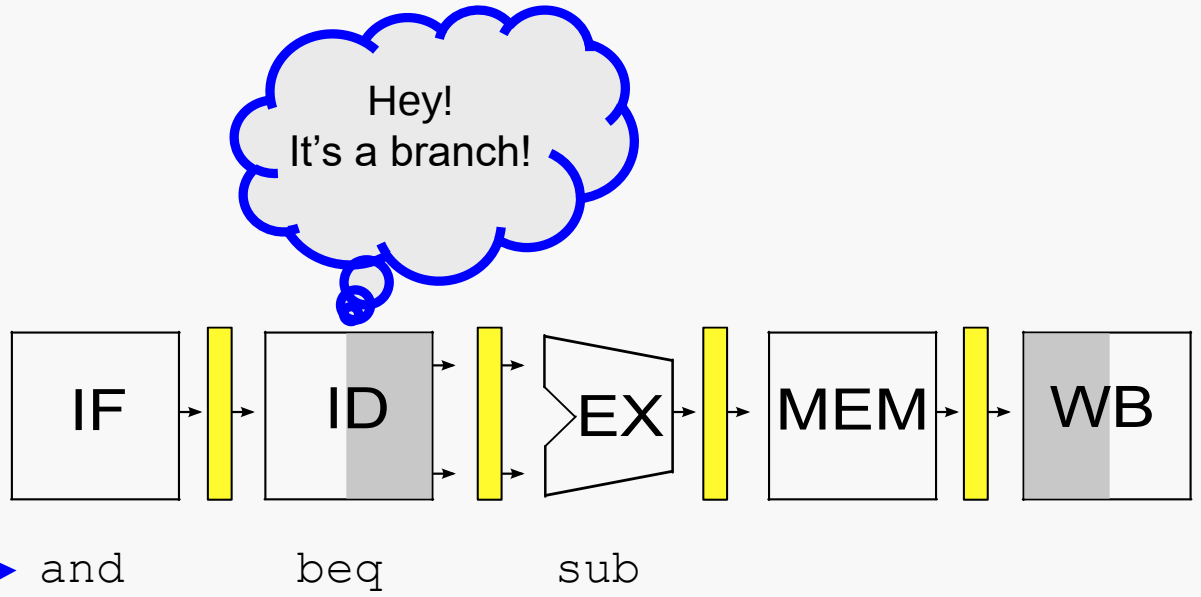



```
                          ???       beq       sub
address          instruction
-------------------------------
     36:          sub  $10, $4, $8
     40:          beq  $1,  $3, L1
     44:          and  $12, $2, $5
                  ...
     72:    L1:   lw   $4, 50($7)
```


©2005-2020 WD McQuain

Branch Hazards

So… we will have already fetched the next (sequential) instruction.



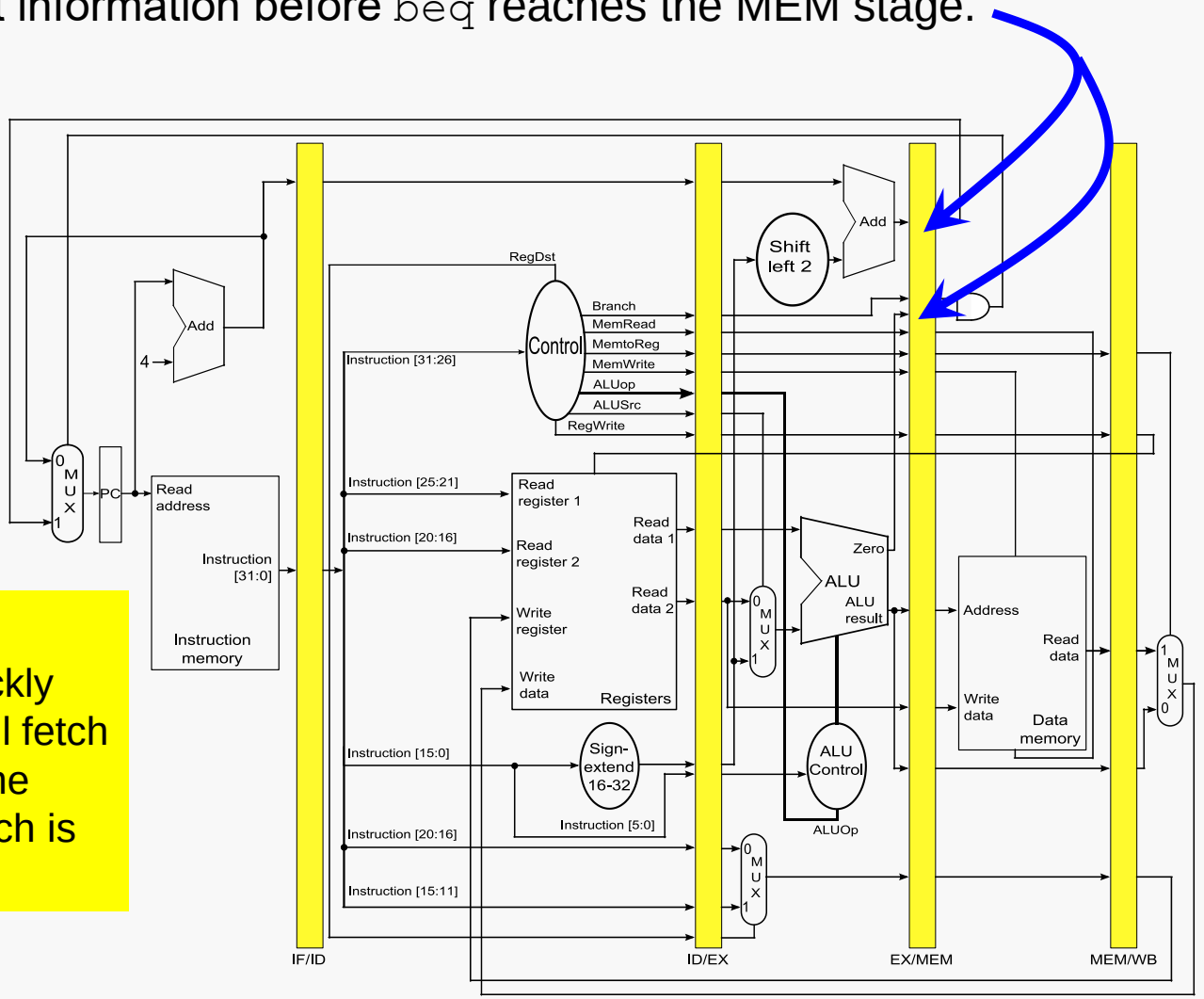

```
address          instruction
-------------------------------
     36:          sub  $10, $4, $8
     40:          beq  $1,  $3, L1
     44:          and  $12, $2, $5
                  ...
     72:    L1:   lw   $4, 50($7)
```


©2005-2020 WD McQuain

Stalling for Branch Hazards

Idea: insert stalls until we know if the branch will be taken.

We can't act on that information before `beq` reaches the MEM stage.



Note:

the AND gate fires quickly enough that we can still fetch the branch target, on the same cycle, if the branch is taken.

©2005-2020 WD McQuain

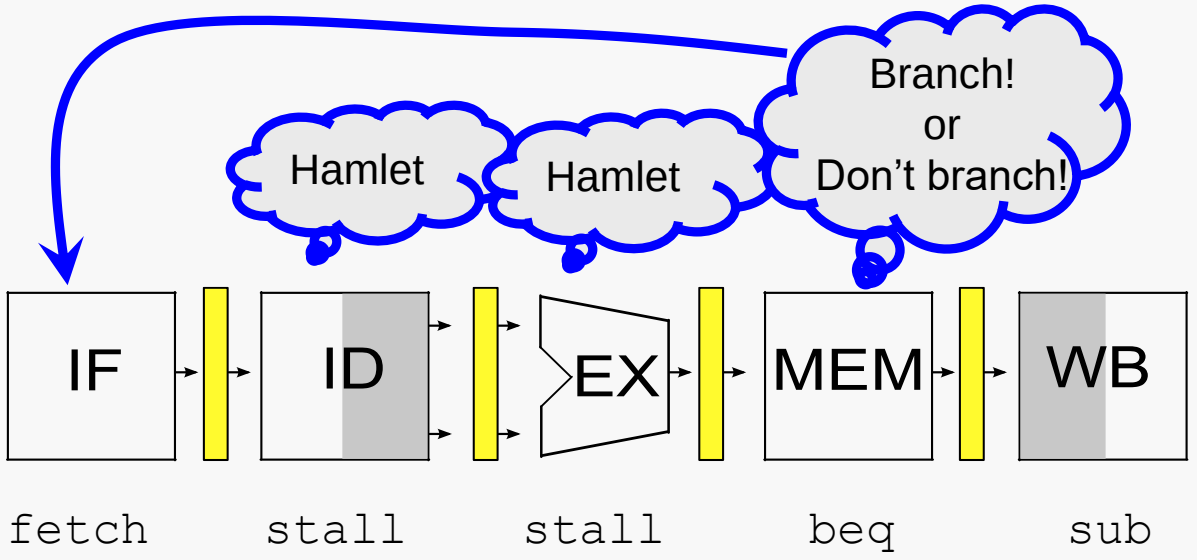
Stalling Until Branch Decision

Idea: insert stalls until we know if the branch will be taken.

```
cycle    action           beq
-------------------------------
   0:    fetch sub
   1:    fetch beq         IF
   2:    fetch and         ID
   3:    stall             EX
   4:    stall            MEM
   5:    fetch and/lw
```

That's expensive.

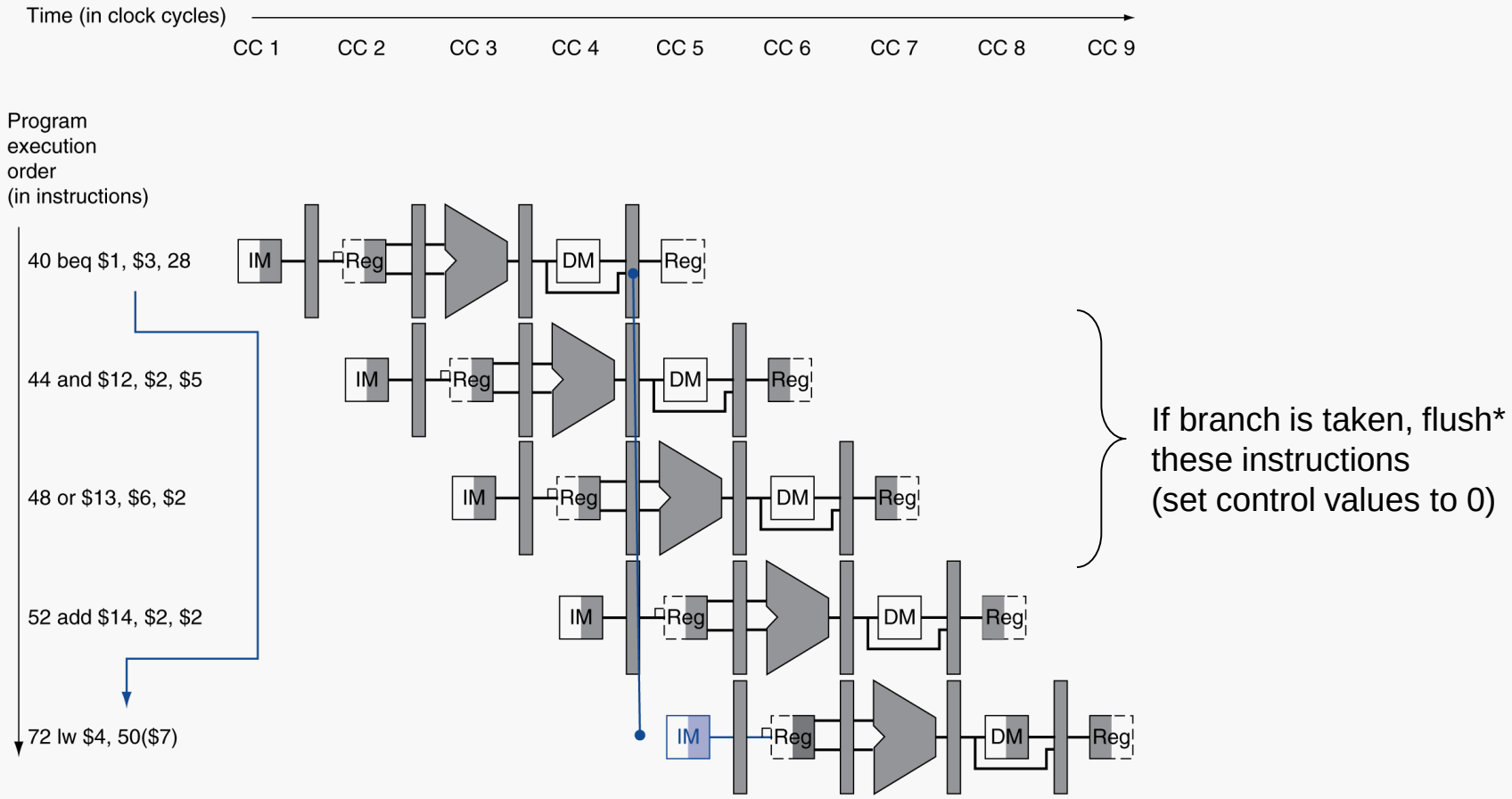
If we don't take the branch, we needlessly delayed the `and` instruction for 2 cycles.



©2005-2020 WD McQuain

Rollback for Branch Hazards

Idea: proceed as if the branch will not be taken;
turn mis-fetched instructions into `nop`s if wrong.



***QTP: will this be too late?**

©2005-2020 WD McQuain

Questions

Could we rearrange the datapath so that the branch decision could be made earlier?

Questions to ponder:

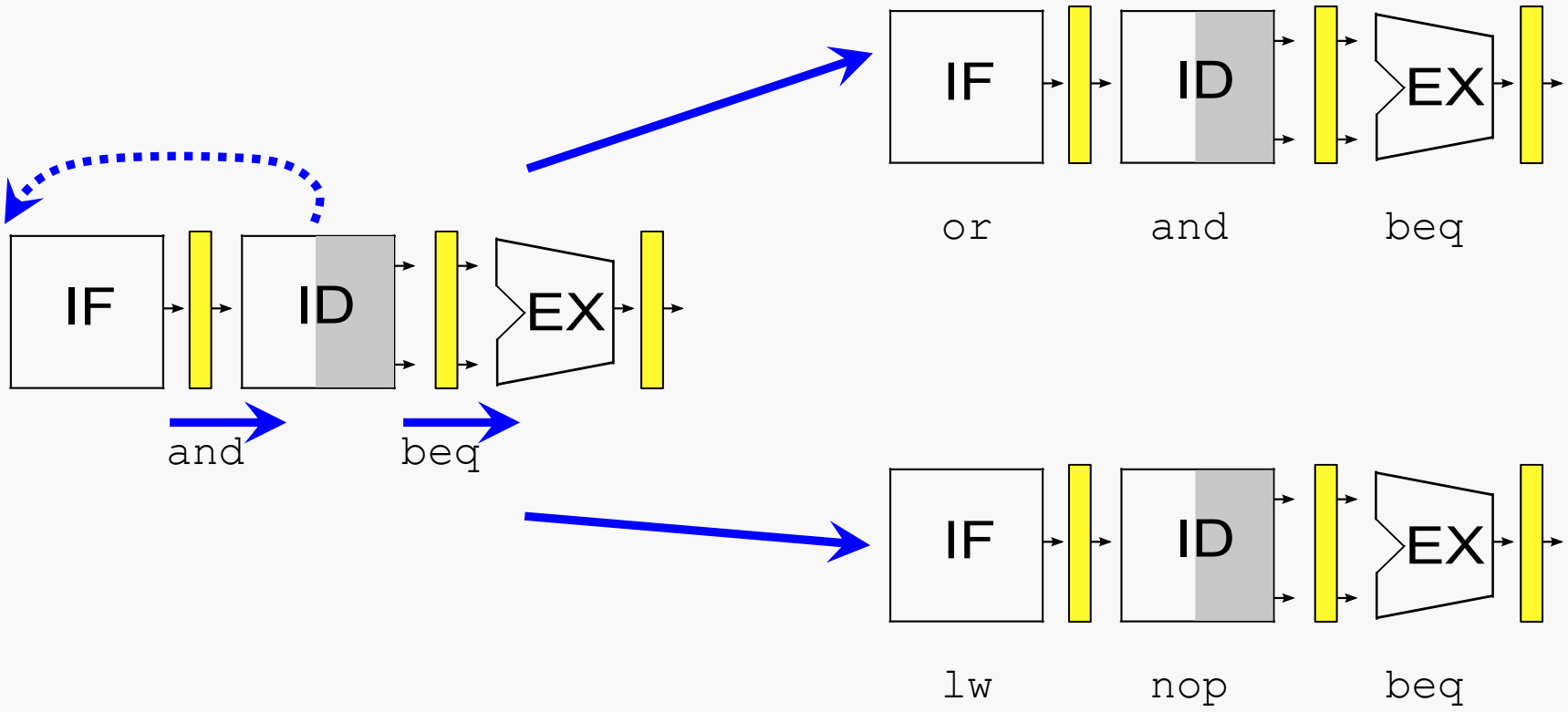
- What about calculating the branch target address?
  Can that be done in the ID stage?

- What about the register comparison?
  Can that be done in the ID stage?
  What about other kinds of conditional branches (e.g., `bgez`,)?

©2005-2020 WD McQuain

Making Branch Decisions Earlier

Stall if branch is taken, proceed normally otherwise:



Cost is now one stall if branch is taken, nothing if branch is not taken.

©2005-2020 WD McQuain

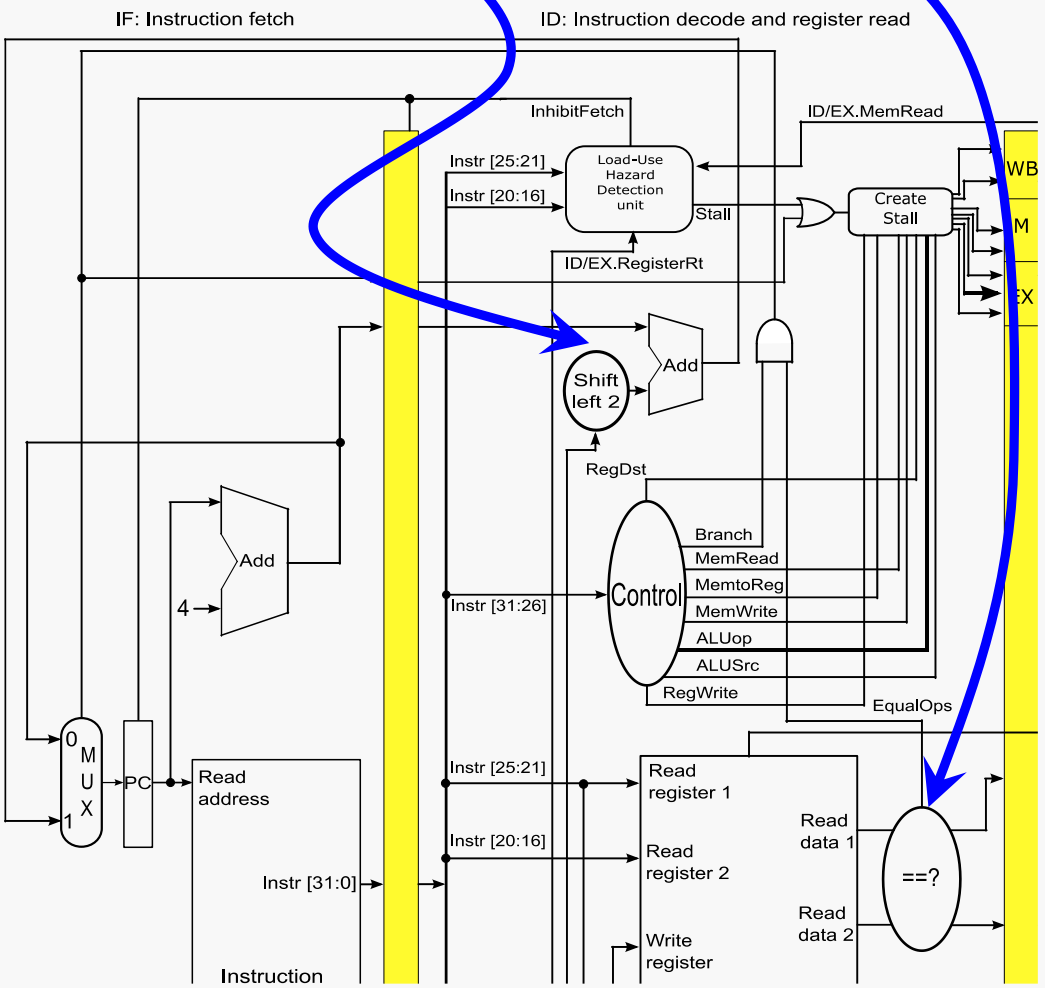
Making Branch Decisions Earlier

Ideas: simple hardware suffices to compare two registers

moving the branch adder is relatively simple

We can determine if the branch will be taken before `beq` reaches the EX stage.

But we also need to create a stall for one cycle…

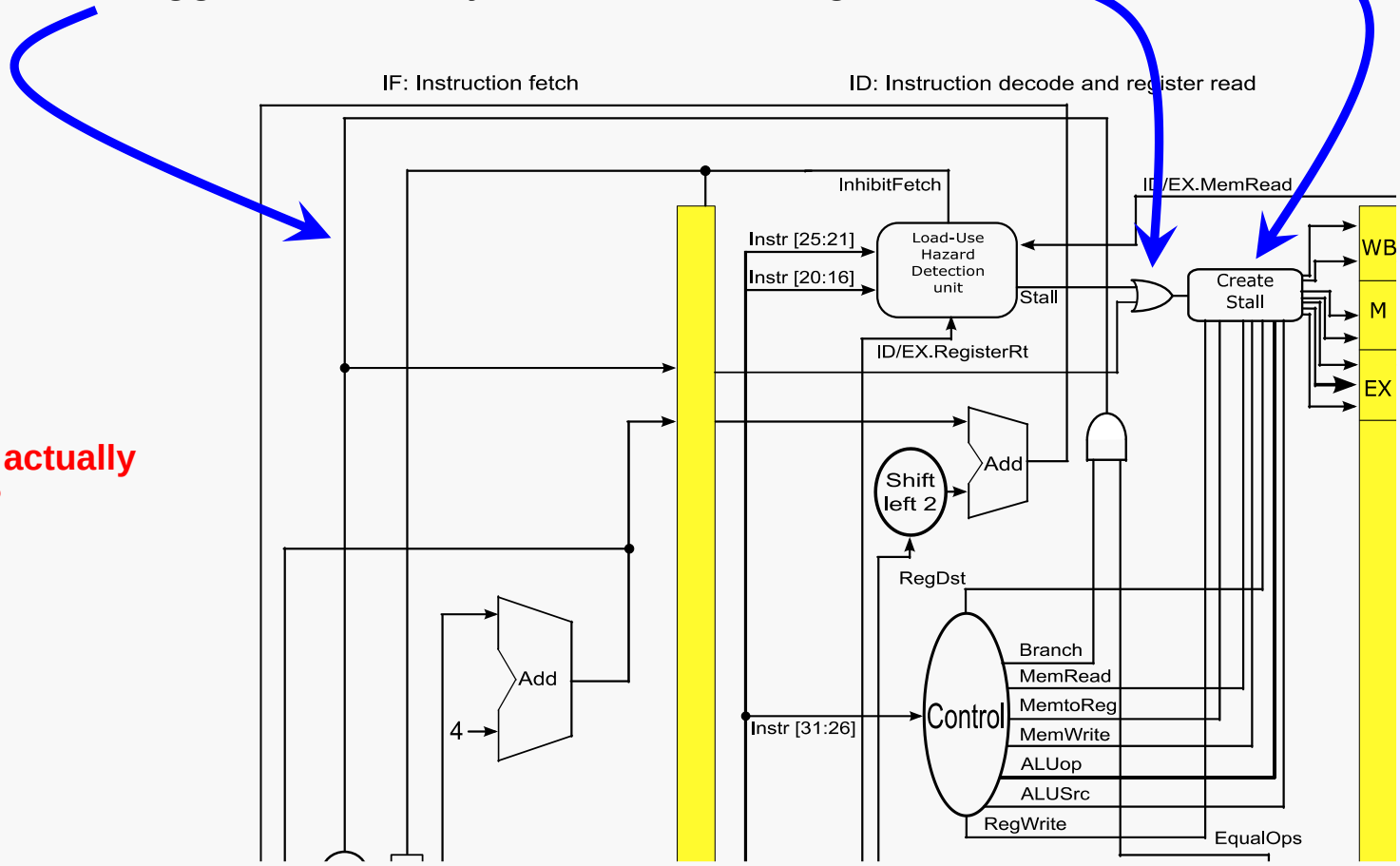


©2005-2020 WD McQuain

Stall-on-Branch

Whenever we decode a `beq` instruction:

- we need to stall the instruction that follows the `beq`
- we have the ability to turn the instruction in the ID stage into a `nop`
- but we need to trigger that one cycle after decoding the `beq`

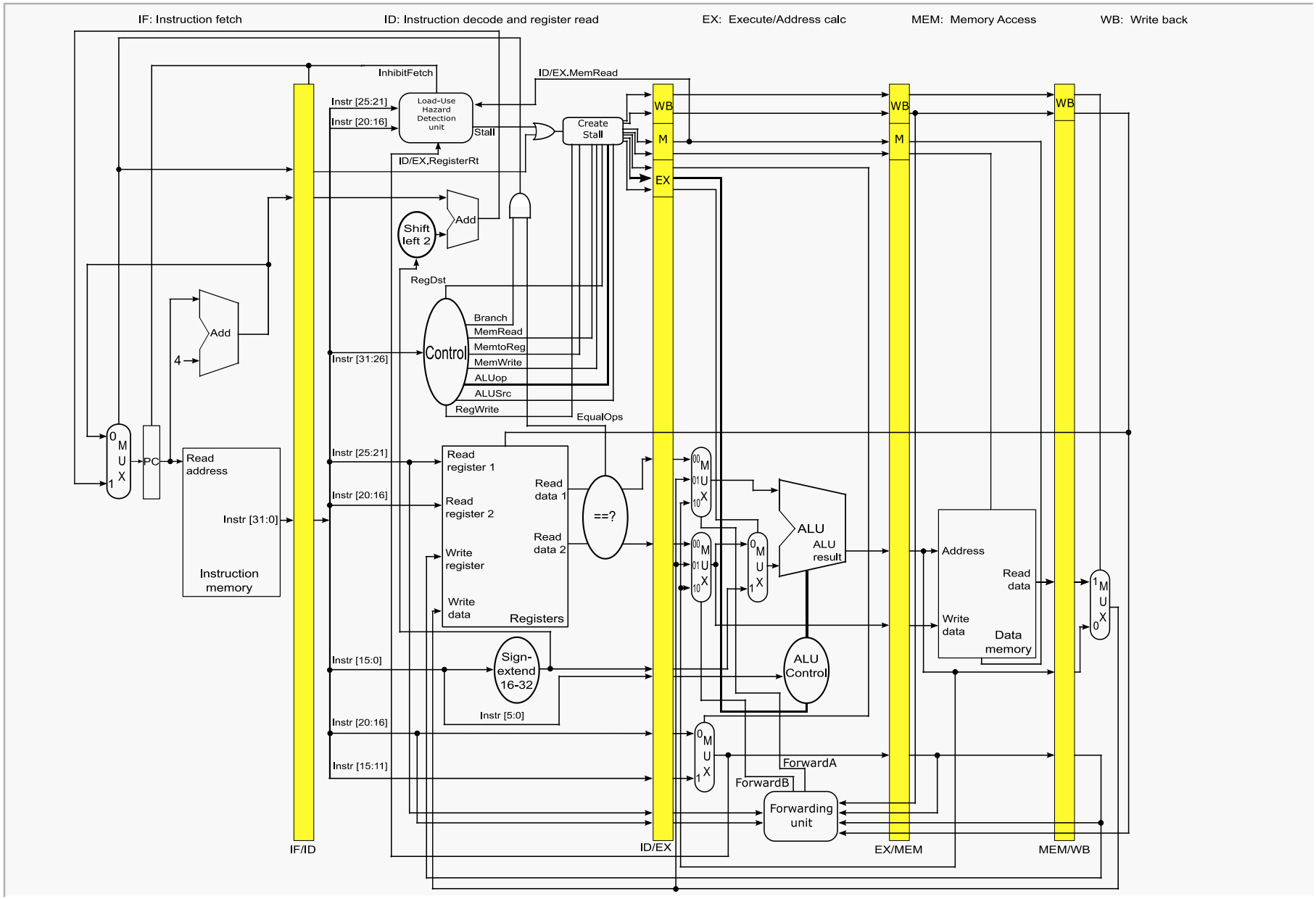


QTP

Does this design actually accomplish that?

©2005-2020 WD McQuain

The Big (but not quite final) Picture

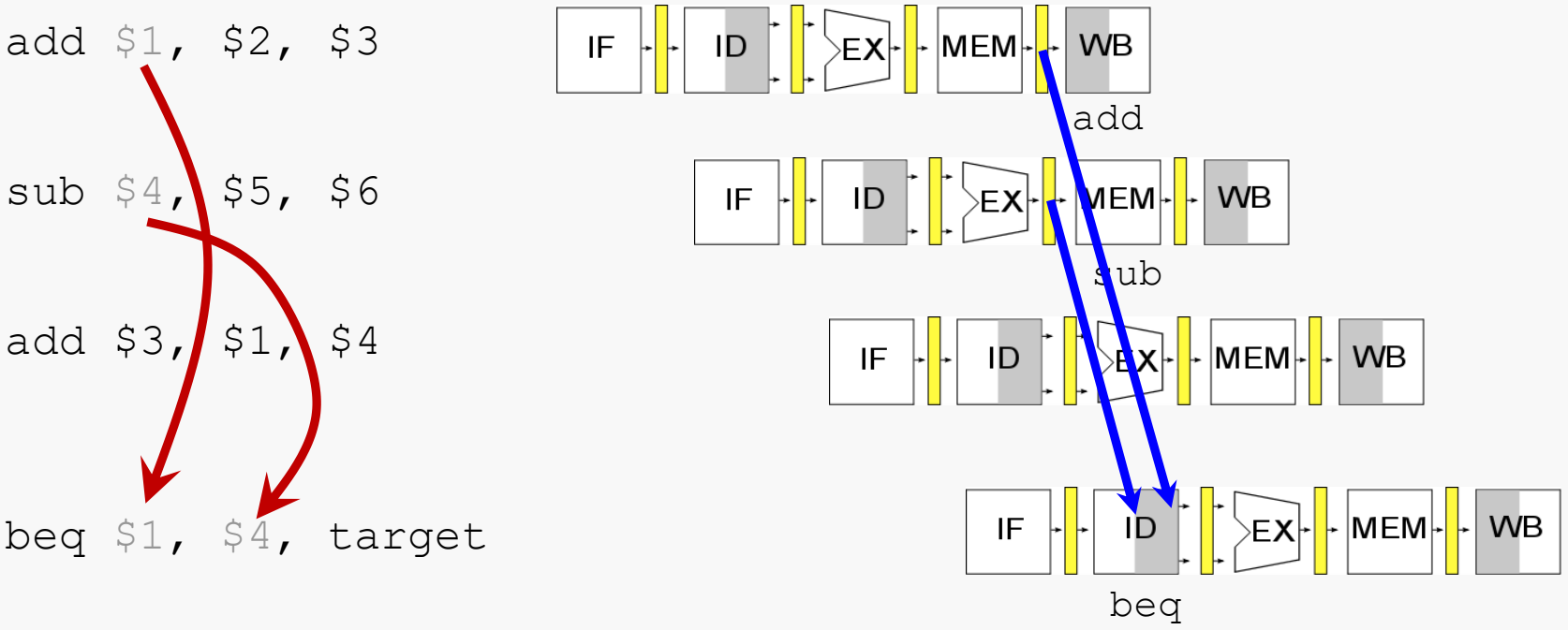


©2005-2020 WD McQuain

Data Hazards for Branches

If a comparison register in `beq` is a destination of 2nd or 3rd preceding R-type instruction

Can resolve using forwarding, without any stalls



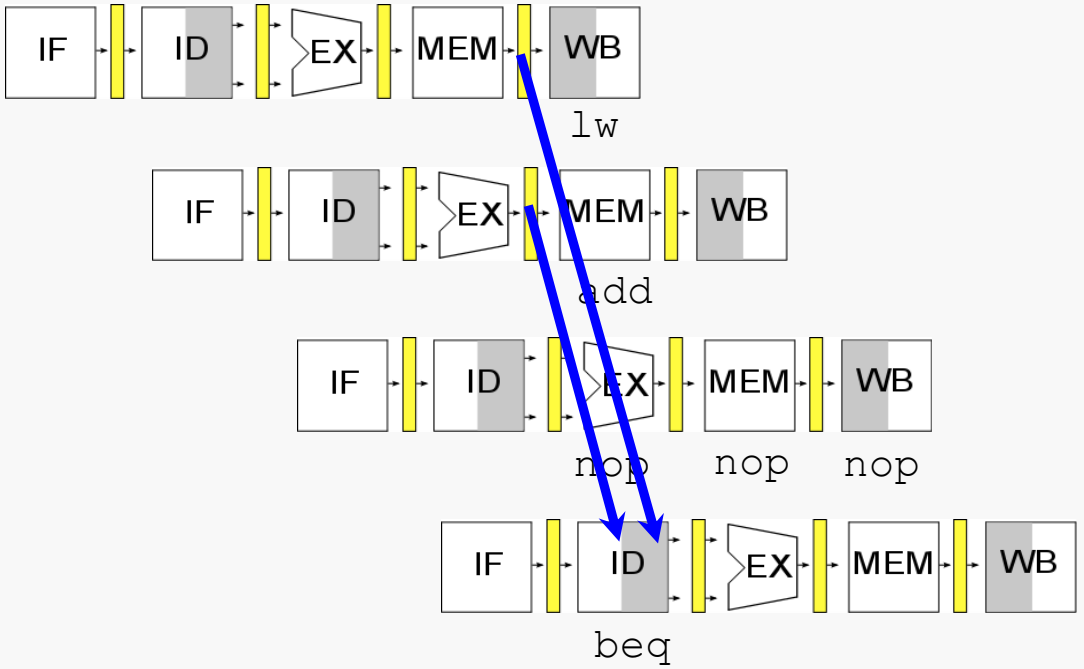
©2005-2020 WD McQuain

Data Hazards for Branches

If a comparison register is a destination of the immediately-preceding type instruction or 2nd preceding load instruction

- Need to stall `beq` for 1 cycle

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



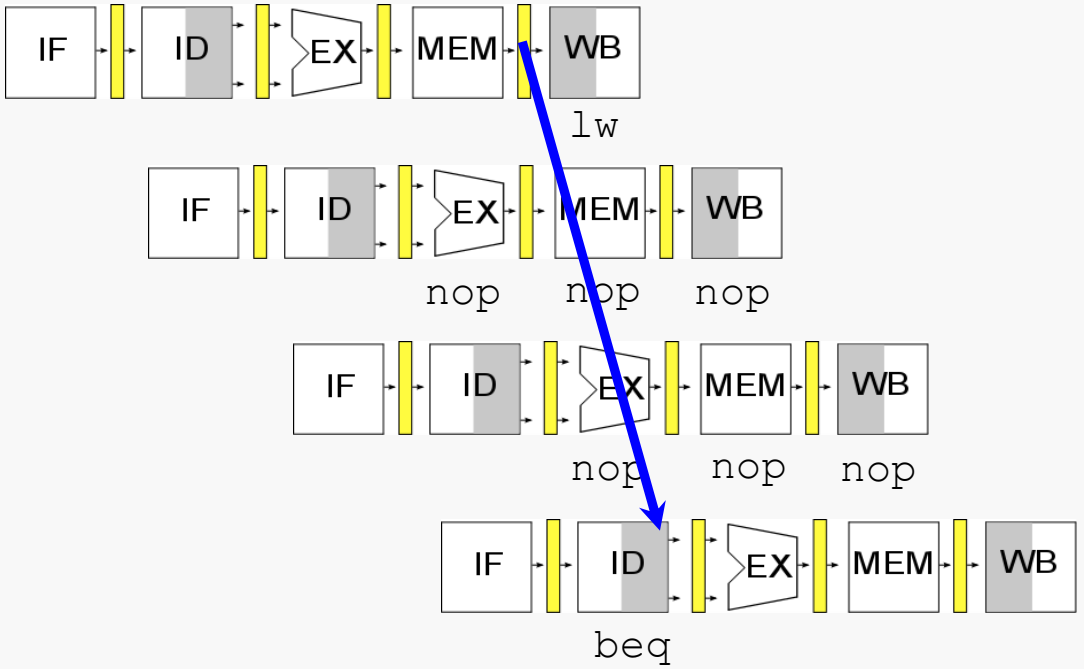
©2005-2020 WD McQuain

Data Hazards for Branches

If a comparison register is a destination of immediately preceding load instruction

- Need to stall for 2 cycles

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



©2005-2020 WD McQuain

Data Hazards for Branches

So, the latest version of the datapath will require more modifications:

- these hazards must be detected in by new hardware in the ID stage
- the actual forwarding (substitution) operations also require new hardware in the ID stage
- handling these hazards requires stalling, as with `lw`, but…
  - if the writing instruction is one cycle ahead of `beq` we need 2 stall cycles
  - if the writing instruction is two cycles ahead of `beq` we need 1 stall cycle

These changes are similar to the earlier discussion of forwarding.

Details are left as an exercise.

©2005-2020 WD McQuain