

Branch Hazards

OR
 “Which way did he go?”

Control Dependence

- Just as an instruction will be dependent on other instructions to provide its operands (*dependence*), it will also be dependent on other instructions to determine whether it gets executed or not (*dependence* or *dependence*).
- Control dependences are particularly critical with _____ branches.

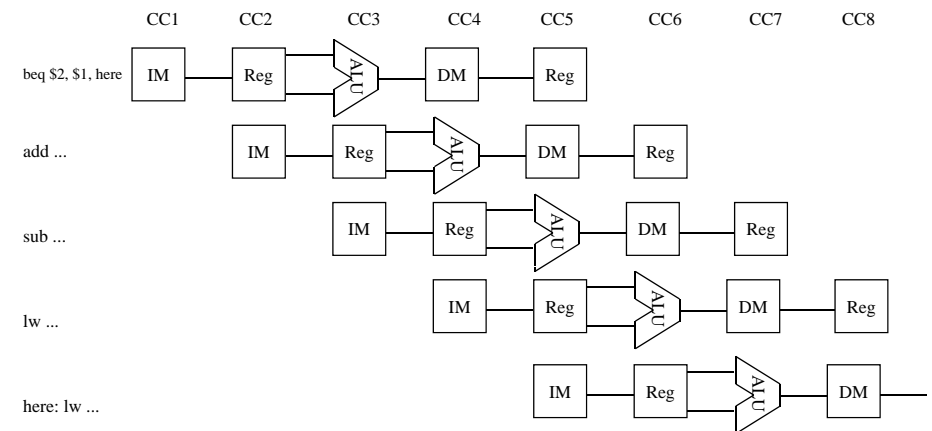
```

add $5, $3, $2
sub $6, $5, $2
beq $6, $7, somewhere
and $9, $6, $1
...
somewhere: or $10, $5, $2
add $12, $11, $9
...
    
```

Branch Hazards

- Branch dependences can result in branch hazards (when they are too close to be handled correctly in the pipeline).

Branch Hazards



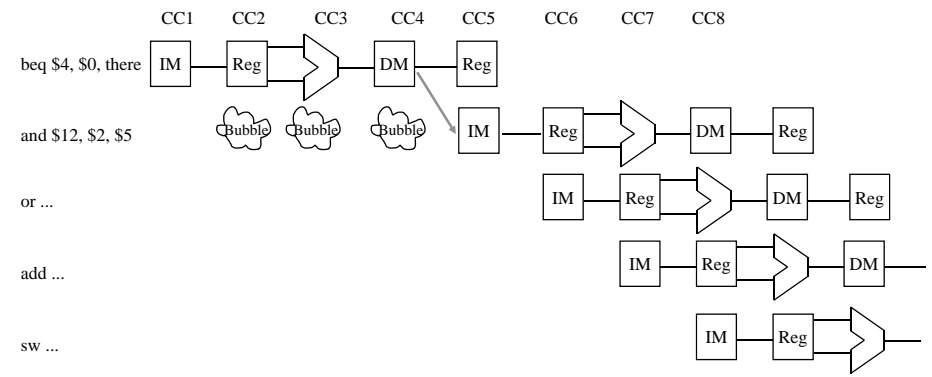
Dealing With Branch Hazards

- Hardware
 - stall until you know which direction
 - reduce hazard through earlier computation of branch direction
 - guess which direction
 - assume not taken (easiest)
 - more educated guess based on history (requires that you know it is a branch before it is even decoded!)
- Hardware/Software
 - nops, or instructions that get executed either way (delayed branch).

CSE 141

Dean Tullsen

Stalling for Branch Hazards



CSE 141

Dean Tullsen

Stalling for Branch Hazards

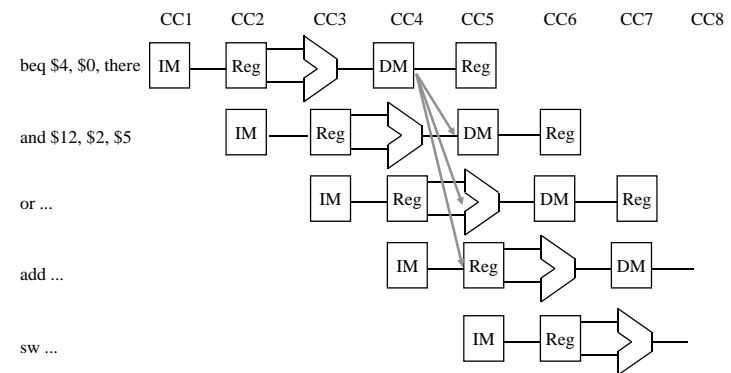
- Seems wasteful, particularly when the branch isn't taken.
- Makes all branches cost 4 cycles.

CSE 141

Dean Tullsen

Assume Branch *Not Taken*

- works pretty well when you're right

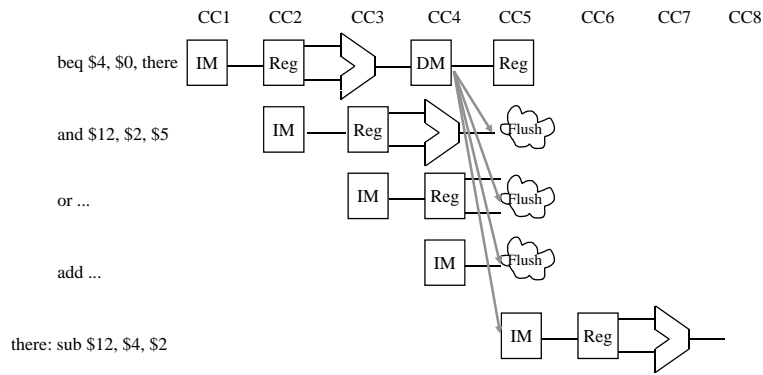


CSE 141

Dean Tullsen

Assume Branch Not Taken

- same performance as stalling when you're wrong



CSE 141

Dean Tullsen

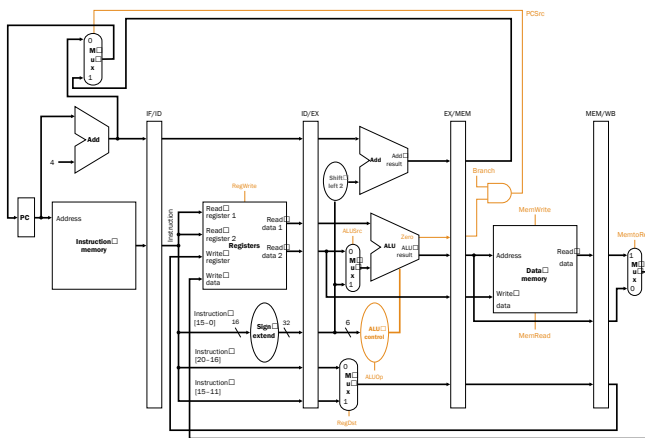
Assume Branch Not Taken

- Performance depends on percentage of time you guess right.
- Flushing an instruction means to prevent it from changing any permanent state (registers, memory, PC).
 - sounds a lot like a bubble...
 - But notice that we need to be able to insert those bubbles later in the pipeline

CSE 141

Dean Tullsen

Reducing the Branch Delay

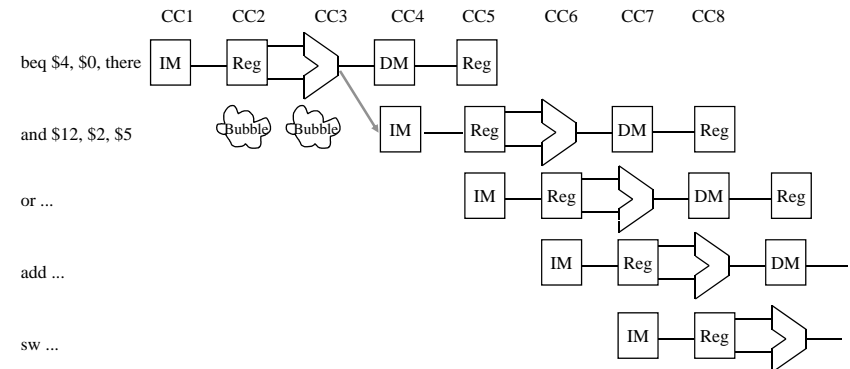


- can easily get to 2-cycle stall

CSE 141

Dean Tullsen

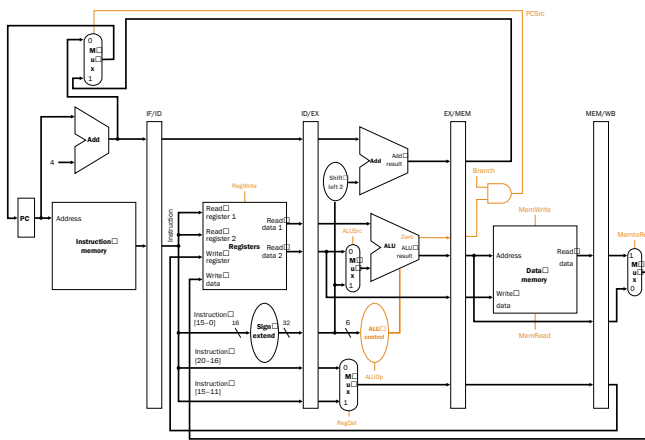
Stalling for Branch Hazards



CSE 141

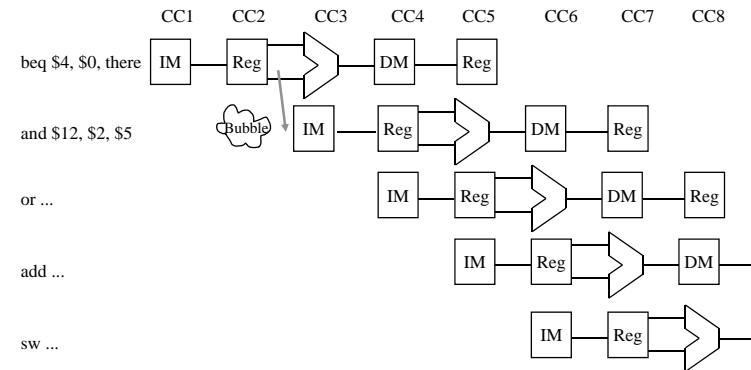
Dean Tullsen

Reducing the Branch Delay

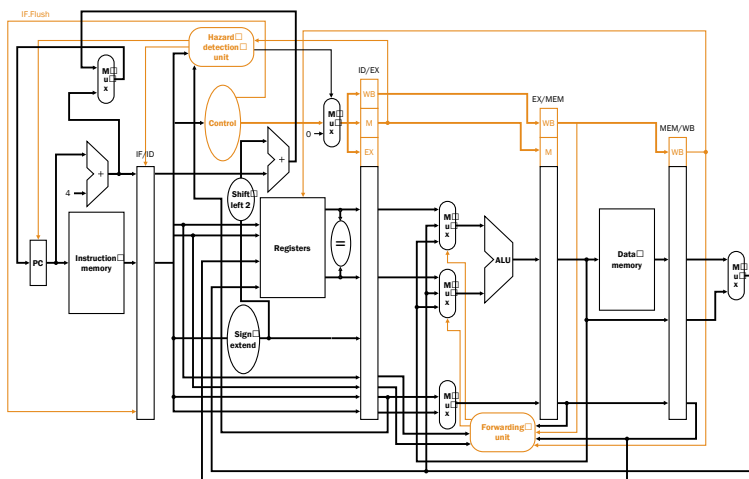


- Harder, but possible, to get to 1-cycle stall

Stalling for Branch Hazards



The Pipeline with flushing for taken branches

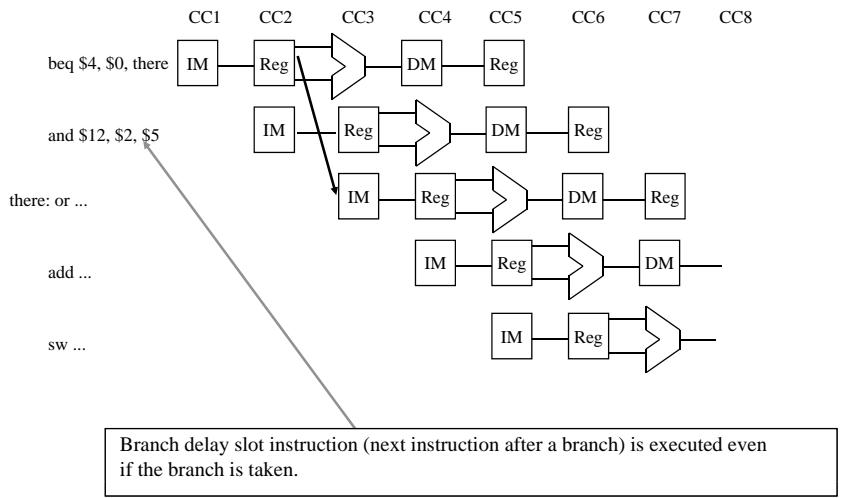


- Notice the IF/ID flush line added.

Eliminating the Branch Stall

- There's no rule that says we have to see the effect of the branch immediately. Why not wait an extra instruction before branching?
- The original SPARC and MIPS processors each used a single *branch delay slot* to eliminate single-cycle stalls after branches.
- The instruction after a conditional branch is *always executed* in those machines, regardless of whether the branch is taken or not!

Branch Delay Slot



CSE 141

Dean Tullsen

Filling the branch delay slot

```

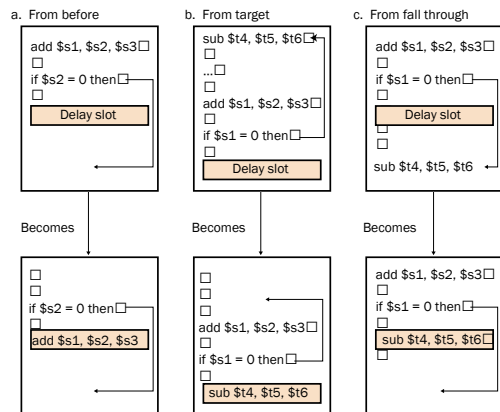
add $5, $3, $7
sub $6, $1, $4
and $7, $8, $2
beq $6, $7, there
nop /* branch delay slot */
add $9, $1, $2
sub $2, $9, $5
...
there:
mult $2, $10, $11
    
```

CSE 141

Dean Tullsen

Filling the branch delay slot

- The branch delay slot is only useful if you can find something to put there.
- If you can't find anything, you must put a *nop* to insure correctness.



CSE 141

Dean Tullsen

Branch Delay Slots

- This works great for this implementation of the architecture, but becomes a permanent part of the ISA.
- What about the MIPS R10000, which has a 5-cycle branch penalty, and executes 4 instructions per cycle???

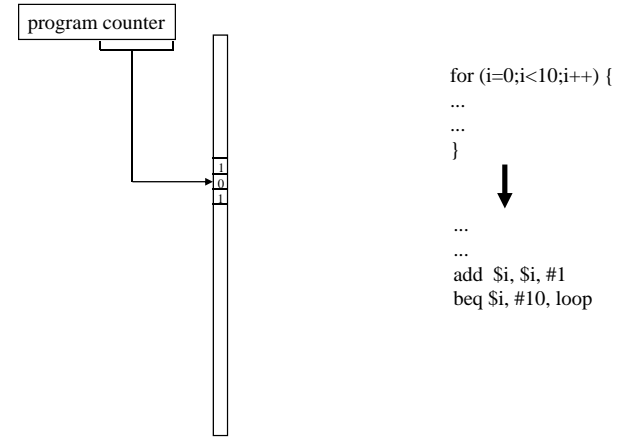
CSE 141

Dean Tullsen

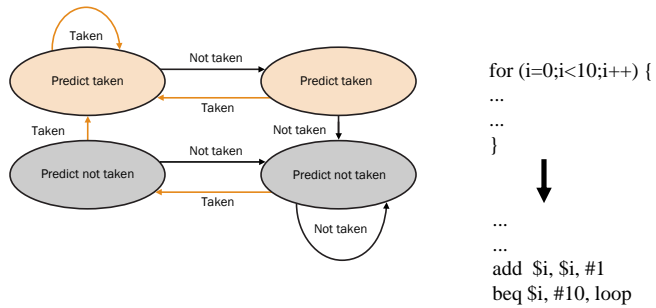
Branch Prediction

- Always assuming the branch is not taken is a crude form of _____.
- What about loops that are 95% of the time?
 - we would like the option of assuming *not taken* for some branches, and *taken* for others, depending on ???

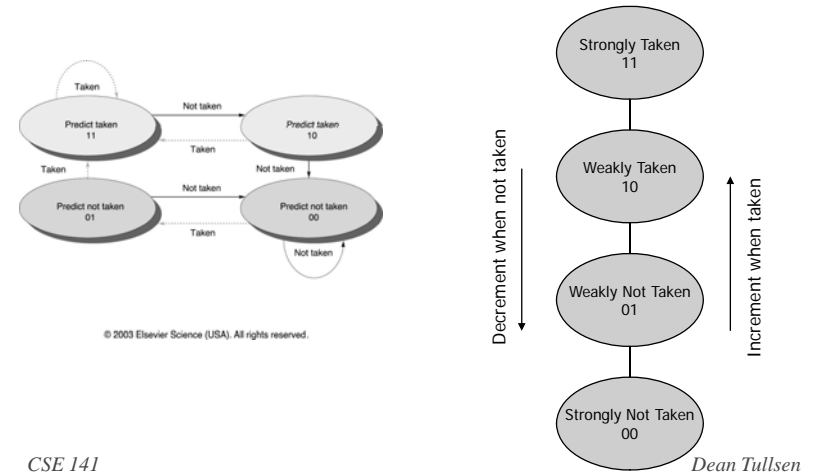
Branch Prediction



Two-bit predictors give better loop prediction

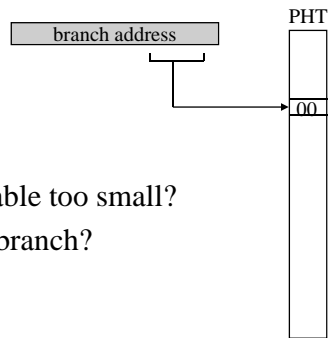


Two different 2-bit schemes



Branch History Table

- has limited size
- 2 bits by N (e.g. 4K)
- uses low bits of branch address to choose entry



- what happens when table too small?
- what about even/odd branch?

CSE 141

Dean Tullsen

- Latest branch predictors significantly more sophisticated, using more advanced correlating techniques, larger structures, and soon possibly using AI techniques.
- Presupposes what two pieces of information are available at fetch time?
 -
 -
- Branch Target Buffer supplies this information.

CSE 141

Dean Tullsen

Pipeline performance

```
loop: lw $15, 1000($2)
      add $16, $15, $12
      lw $18, 1004($2)
      add $19, $18, $12
      beq $19, $0, loop:
```

CSE 141

Dean Tullsen

Control Hazards -- Key Points

- Control (or branch) hazards arise because we must fetch the next instruction before we know if we are branching or where we are branching.
- Control hazards are detected in hardware.
- We can reduce the impact of control hazards through:
 - early detection of branch address and condition
 - branch prediction
 - branch delay slots

CSE 141

Dean Tullsen