

Pipelining: Control and Performance

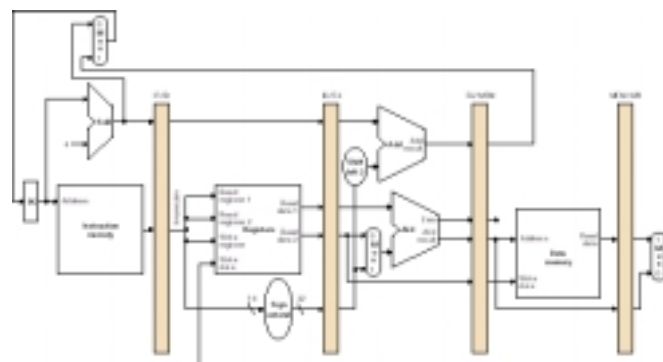
- ◆ **Last Time**
 - Pipelining and Speedup
 - Instruction Pipelining (datapath)
- ◆ **This Time**
 - Quiz
 - Control for Pipelining
 - Performance and Control Hazards
- ◆ **Reminders/Announcements**
 - HW3 out this afternoon (check the web)
 - Read rest of Chapter 6

CS 141 Chien

1

Feb 24, 2000

Pipelining (review)



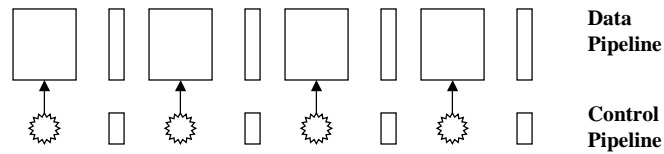
- ◆ **Datapath, instructions in sequence**
- ◆ **Fixing the register write address (pipeline the controls)**

CS 141 Chien

2

Feb 24, 2000

Pipelined Control (concept)



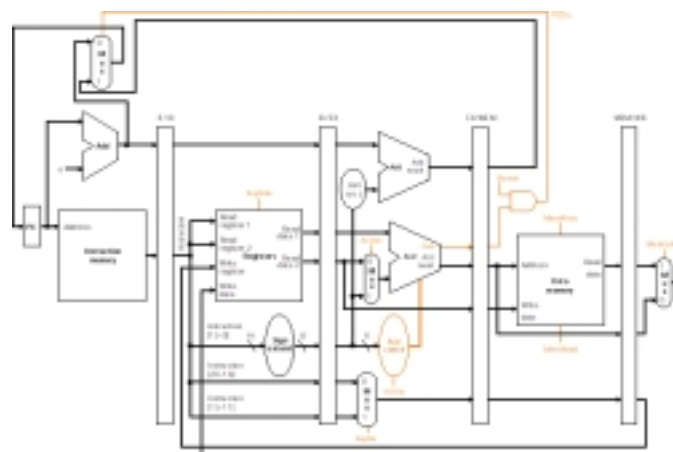
- ◆ Instructions flow down the pipeline
- ◆ Control flows along with it.
- ◆ Logic maps control -> datapath, and generates state for the next pipeline stage
- ◆ Everything gets done at just the the right time

CS 141 Chien

3

Feb 24, 2000

Pipelined Control (implementation)



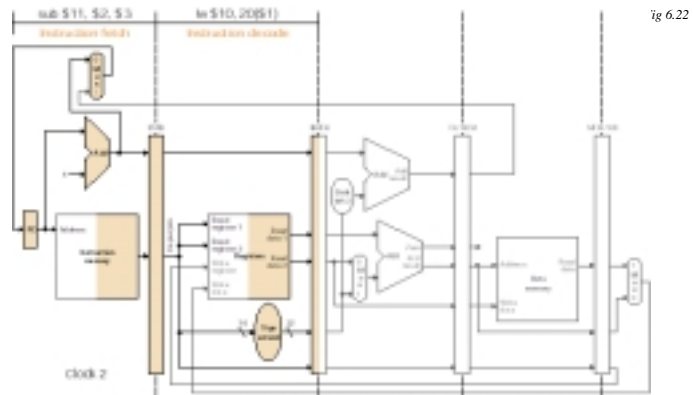
- ◆ Actual datapath and control pipeline

CS 141 Chien

4

Feb 24, 2000

Pipelined Control II



ig 6.22

- ◆ Fetch and decode
- ◆ Two instructions in the pipeline

CS 141 Chien

7

Feb 24, 2000

Pipelined Control III

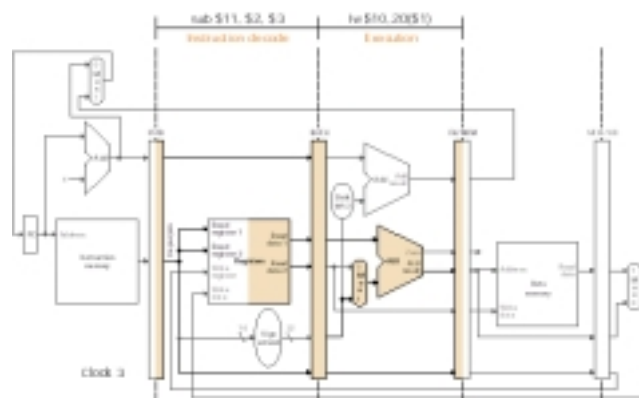


Fig 6.23

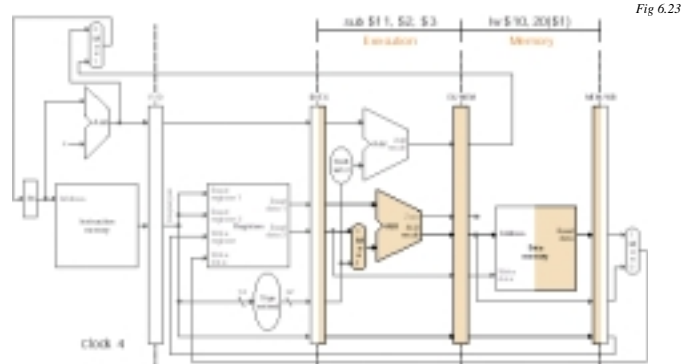
- ◆ Decode and Execute
- ◆ What does each instruction have to do?

CS 141 Chien

8

Feb 24, 2000

Pipelined Control IV



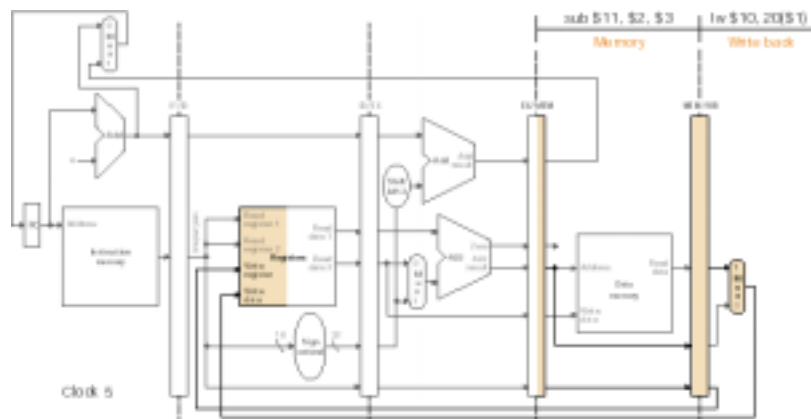
- ◆ Execute and memory
- ◆ What does each instruction have to do?

CS 141 Chien

9

Feb 24, 2000

Pipelined Control V



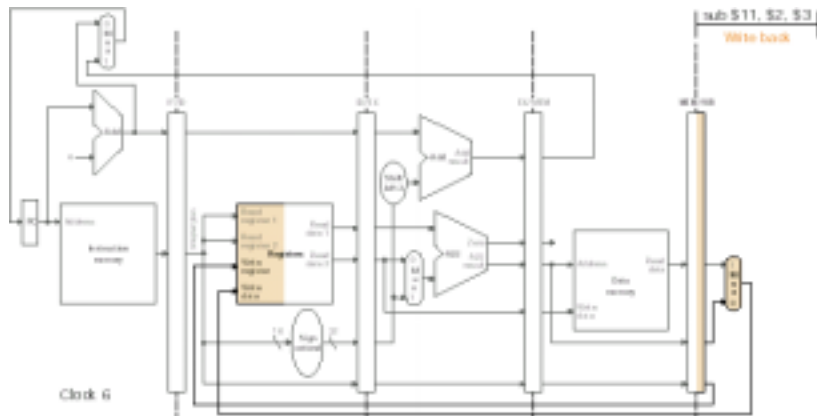
- ◆ Memory and Writeback, Instruction completion
- ◆ Did things happen in the right order?

CS 141 Chien

10

Feb 24, 2000

Pipelined Control VI



- ◆ Instruction completion
- ◆ Did things happen in the right order?

CS 141 Chien

11

Feb 24, 2000

Pipelining: Basic Control and Data

- ◆ Instructions move through the data and control pipeline
- ◆ Instruction latency is same
- ◆ Throughput increases by number of pipeline stages

- ◆ Is this the whole story?
 - What complications are there?
 - Are they fundamental? (would another datapath design eliminate them)

CS 141 Chien

12

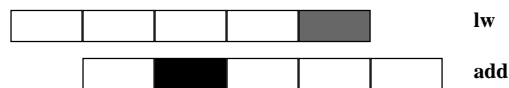
Feb 24, 2000

Dependences and Hazards

```
lw    $t0, 160($t1)
add   $t2,$t2,$t0
```

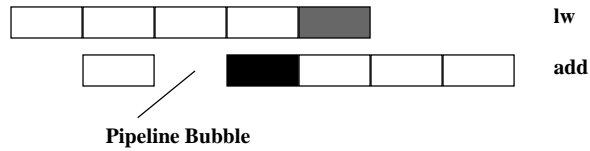
- ◆ Second instruction depends on completion of the first.
- ◆ Normal execution: no problem
- ◆ Pipelined Execution: incorrect answer?
- ◆ 0: ... lw completes in cycle 4, add needs the value at cycle 2

Data Hazard



- ◆ lw writes at cycle 4
- ◆ add reads at cycle 2
- ◆ dependence is backwards in time -> can't do this!
- ◆ Strictly: data available at end of cycle 4, needed at the start of cycle 3
- ◆ Still one cycle of "backward dependence"

Data Hazard Resolution



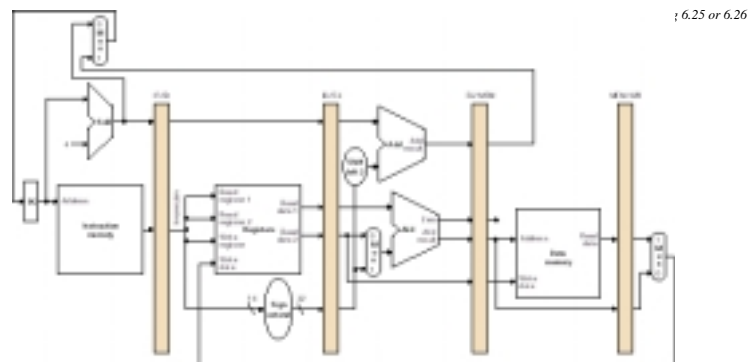
- ◆ Stop the add instruction in place for one cycle
- ◆ Minimum delay of one cycle, forwarding
- ◆ Dependence is correctly enforced (our less demanding requirements)
- ◆ Slight loss of performance “empty cycle”

CS 141 Chien

15

Feb 24, 2000

Pipeline Bubbles



- ◆ What happens to data pipeline?
- ◆ Bubble sets control to “do nothing”, travels through the pipeline

CS 141 Chien

16

Feb 24, 2000

What are possible sources of hazards?

- ◆ R then R, R then store, R then load; all are fine
- ◆ Load then R (data value dependence)
- ◆ Branch then R (control dependence)

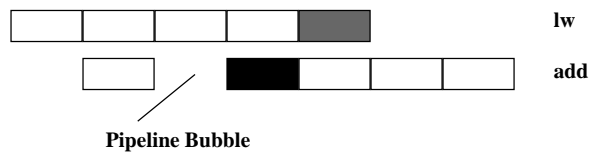
- ◆ These sequences of instructions cause pipeline bubbles.
- ◆ Each bubble reduces performance.
- ◆ Frequency of bubbles \leftrightarrow effectiveness of pipelining
- ◆ \Rightarrow Fraction of “ideal” benefit achieved

CS 141 Chien

17

Feb 24, 2000

Load then R-class



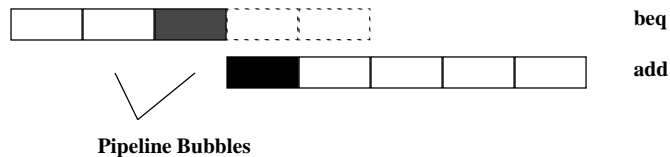
- ◆ Seen this before
- ◆ Causes one cycle of loss

CS 141 Chien

18

Feb 24, 2000

Branch then R



- ◆ Branch determines new PC in cycle 2
- ◆ Instruction Fetch must wait until after cycle 2
- ◆ Two bubbles must be introduced
- ◆ add instruction is fetched two cycles later

Detecting Hazards & Forwarding

- ◆ What is required to detect Load / R class hazards?
- ◆ What is required to detect Branch / anything hazards?
- ◆ What is required to do forwarding for R / R class instructions?

Control Dependence

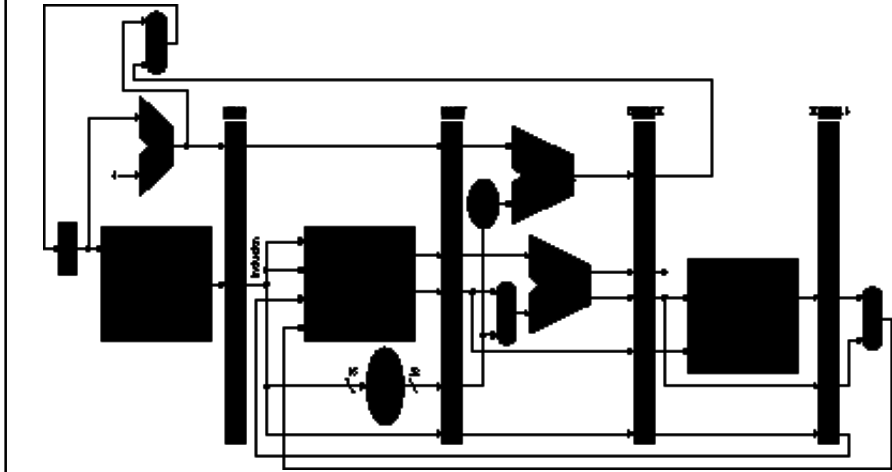
- ◆ Just as an instruction will be dependent on other instructions to provide its operands (*data dependence*), it will also be dependent on other instructions to determine whether it gets executed or not (*branch dependence* or *control dependence*).
- ◆ Control dependences are particularly critical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

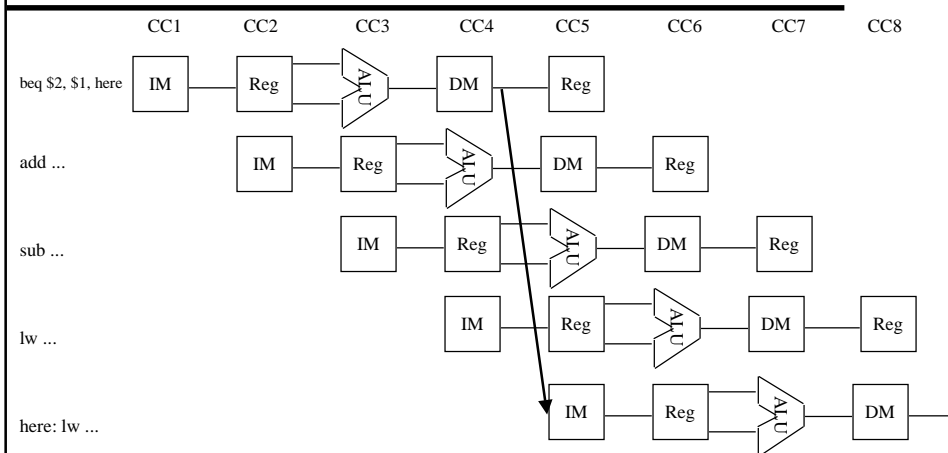
Branch Hazards

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

The Pipeline in Execution



Branch Hazards



Dealing With Branch Hazards

◆ Software

- nops, or instructions that get executed either way (delayed branch).

◆ Hardware

- reduce hazard through earlier computation of branch direction
- stall until you know which 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!)
- ignore the branch for a cycle (branch delay slot)

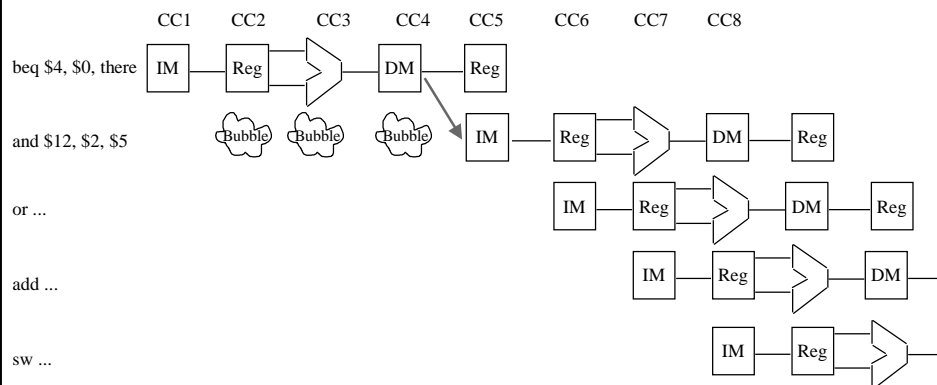
We'll talk about these

CS 141 Chien

25

Feb 24, 2000

Stalling for Branch Hazards



CS 141 Chien

26

Feb 24, 2000

Stalling for Branch Hazards

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

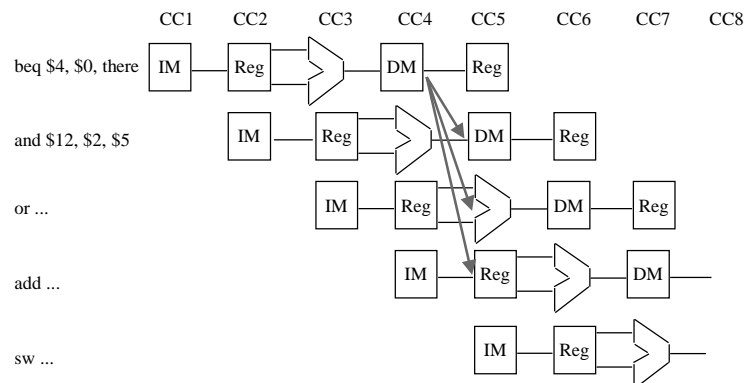
CS 141 Chien

27

Feb 24, 2000

Assume Branch *Not Taken*

- ◆ works pretty well when you're right



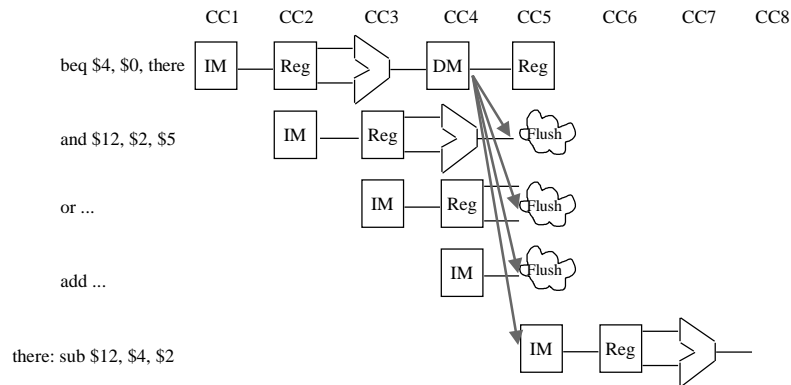
CS 141 Chien

28

Feb 24, 2000

Assume Branch *Not Taken*

- ◆ same performance as stalling when you're wrong



CS 141 Chien

29

Feb 24, 2000

Assume Branch *Not Taken*

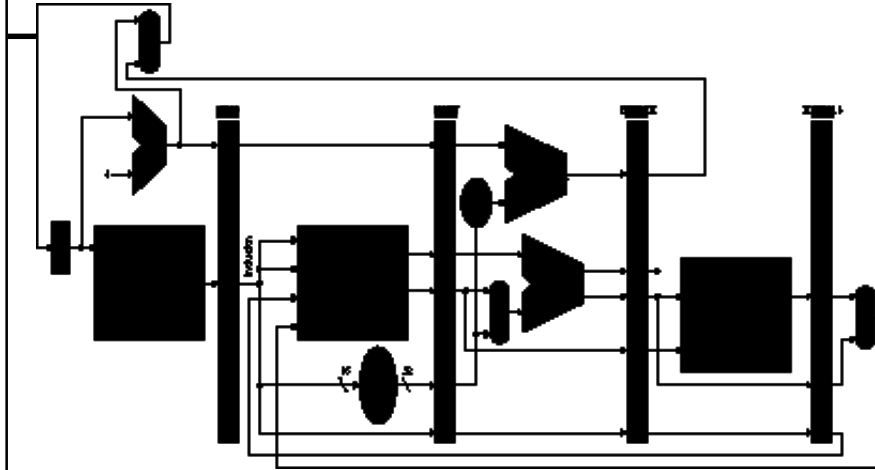
- ◆ Performance depends on percentage of time you guess right.
- ◆ Flushing an instruction means to prevent it from changing any permanent state (registers, memory, PC).
 - sounds a lot like a bubble...
 - But notice that we need to be able to insert those bubbles later in the pipeline

CS 141 Chien

30

Feb 24, 2000

Reducing the Branch Delay



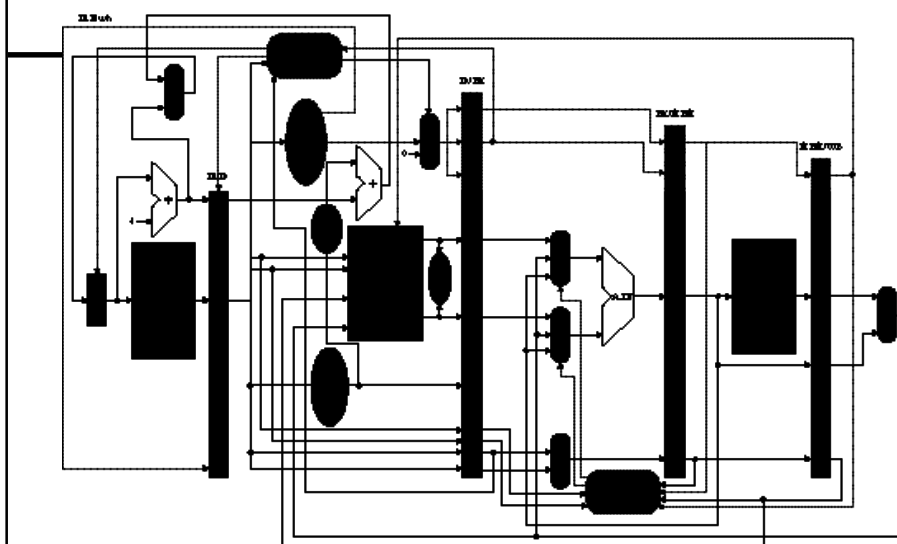
•can easily get to 2-cycle stall

CS 141 Chien

31

Feb 24, 2000

Reducing the Branch Delay

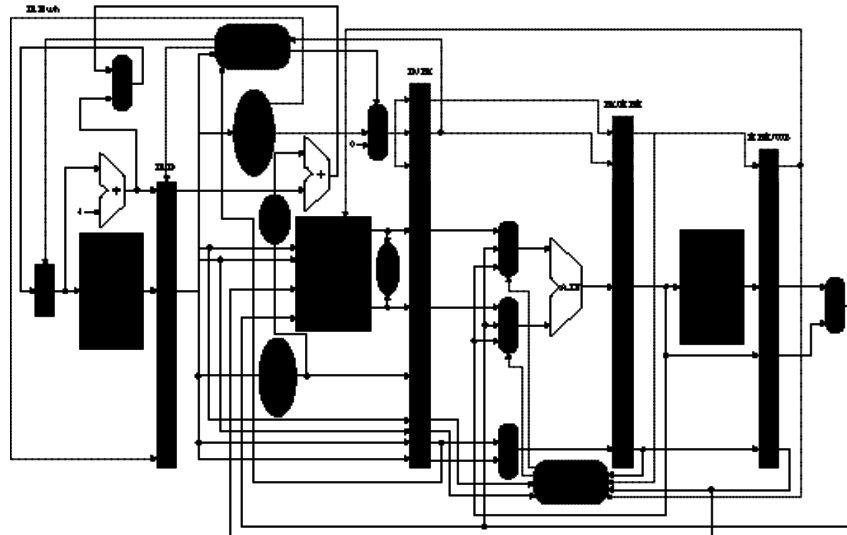


CS 141 Chien

32

Feb 24, 2000

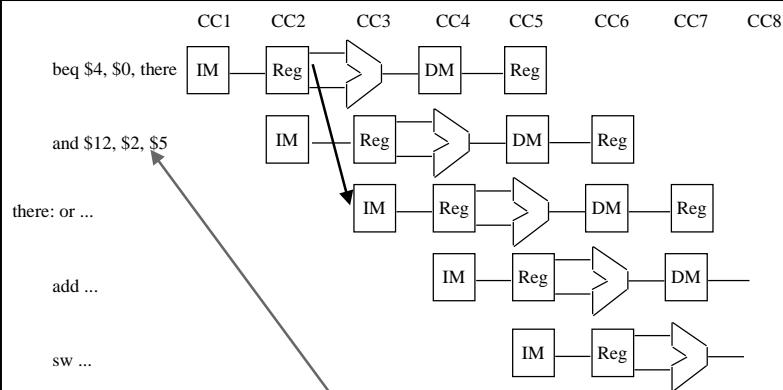
The Pipeline with flushing for taken branches



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



Branch delay slot instruction (next instruction after a branch) is executed even if the branch is taken.

CS 141

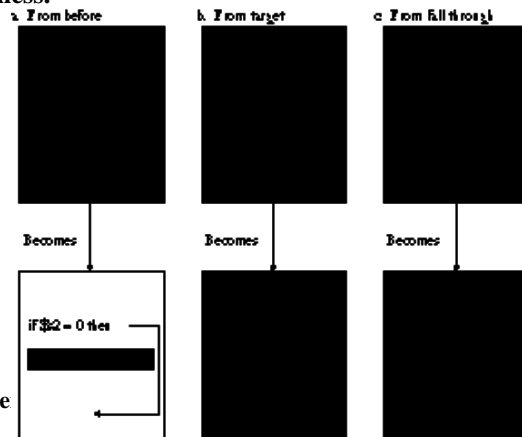
Filling the branch delay slot

```

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
    
```

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.



CS 141 Chie

Feb 24, 2000

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???

CS 141 Chien

38

Feb 24, 2000

Branch Prediction

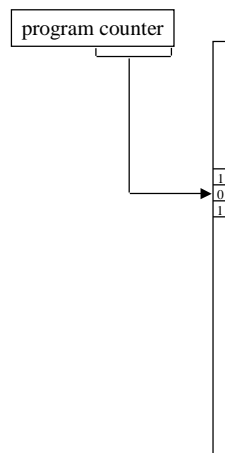
- ◆ Always assuming the branch is not taken is a crude form of *branch prediction*.
- ◆ What about loops that are *taken 95%* of the time?
 - we would like the option of assuming *not taken* for some branches, and *taken* for others, depending on ???

CS 141 Chien

39

Feb 24, 2000

Branch Prediction



```
for (i=0;i<10;i++) {  
...  
...  
}
```



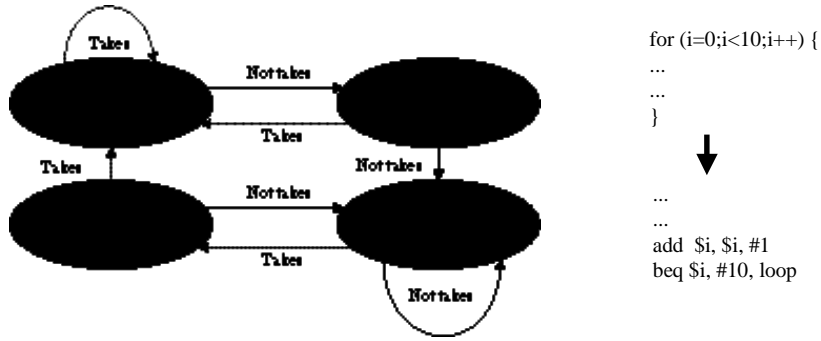
```
...  
...  
add $i, $i, #1  
beq $i, #10, loop
```

CS 141 Chien

40

Feb 24, 2000

Two-bit predictors give better loop prediction



CS 141 Chien

41

Feb 24, 2000

Pipeline performance

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

CS 141 Chien

42

Feb 24, 2000

Pipelining Control Summary

- ◆ Control follows pipeline with data, hardware control in similar to multicycle implementation
- ◆ Hazards require pipeline bubbles, stop some instructions
- ◆ Forwarding and Hazard detection requirements
- ◆ Pipelined control must stop instructions in place, introducing do-nothing instructions
- ◆ Bubbles reduce performance, the game is to minimize the bubbles...
- ◆ 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