

Advanced Pipelining

◆ Last Time

- Control Hazards
- Reduce, Stall, Make Explicit, Guess, Guess better

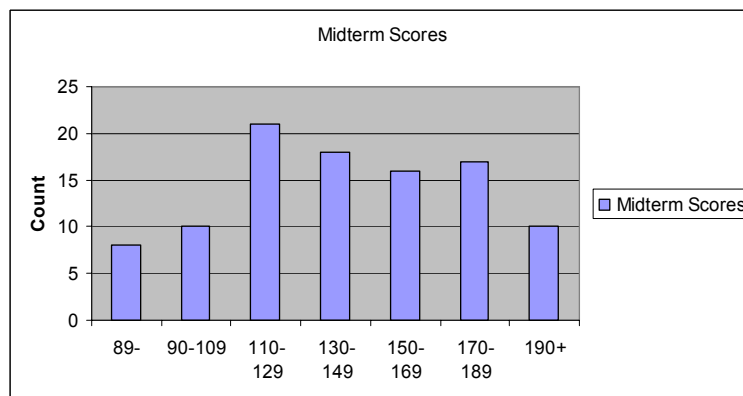
◆ This Time

- Pipelining and Exceptions
- More aggressive ILP (beyond basic pipelining)
- Superpipelining, Superscalar, Dynamic Pipelines

◆ Reminders/Announcements

- Homework #4 is on the Web, **Due 3/7/00 (one week)**
- Begin reading H&P Chapter 7.1 - 7.4 on memory hierarchies (we'll discuss beginning next lecture)

Midterm Scoring



◆ $n=100$, Avg = 142 out of 260, $\sigma = 36.7$

◆ Returned at end of Lecture Thursday

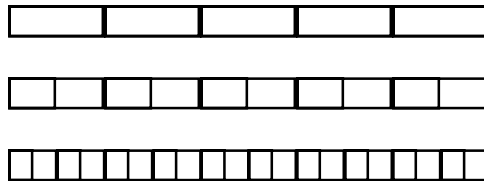
Pipelining and Exceptions

- ◆ Exceptions represent another form of control dependence.
- ◆ Therefore, they create a potential branch hazard
- ◆ Exceptions must be recognized early enough in the pipeline that subsequent instructions can be flushed before they change any permanent state.
- ◆ As long as we do that, everything else works the same as before.
- ◆ Exception-handling that always correctly identifies the offending instruction is called *precise interrupts*.

Pipelining in Today's Most Advanced Processors

- ◆ Not fundamentally different than the techniques we discussed
 - Deeper pipelines
- ◆ Pipelining is combined with
 - superscalar execution
 - out-of-order execution
 - VLIW (very-long-instruction-word)

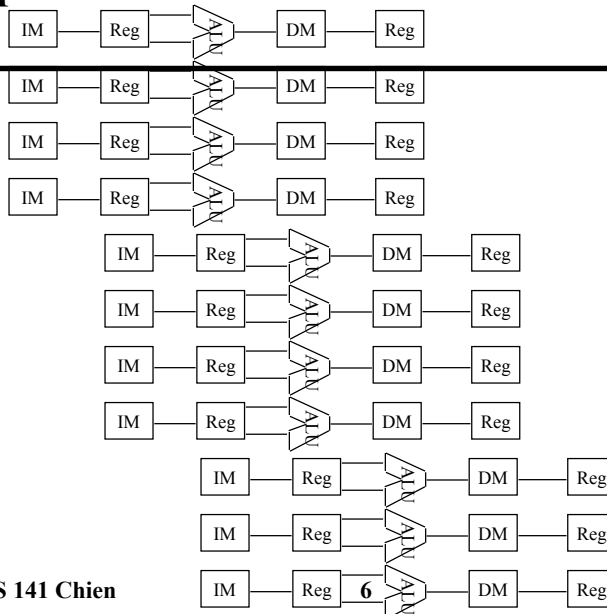
Deeper Pipelines



◆ How much deeper is productive? What are the limiting effects?

- Pipeline latching overhead
- Losses due to stalls and hazards
- Clock Speeds achievable

Superscalar Execution



Superscalar Scheduling

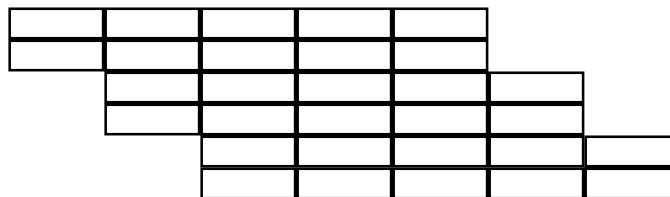
◆ assume in-order, 2-issue, ld-store followed by integer

lw \$6, 36(\$2)
 add \$5, \$6, \$4
 lw \$7, 1000(\$5)
 sub \$9, \$12, \$5

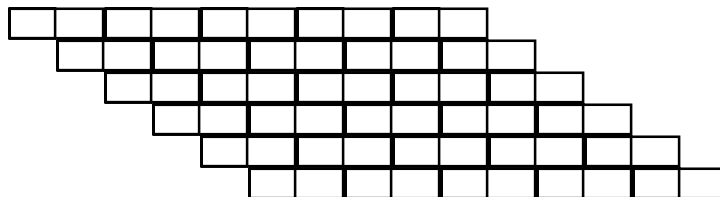
◆ assume 4-issue, any combination (VLIW?)

lw \$6, 36(\$2)
 add \$5, \$6, \$4
 lw \$7, 1000(\$5)
 sub \$9, \$12, \$5
 sw \$5, 200(\$6)
 add \$3, \$9, \$9
 and \$11, \$7, \$6

Superscalar vs. Superpipelined



(multiple instructions in the same stage, same CR as scalar)



(more total stages, faster clock rate)

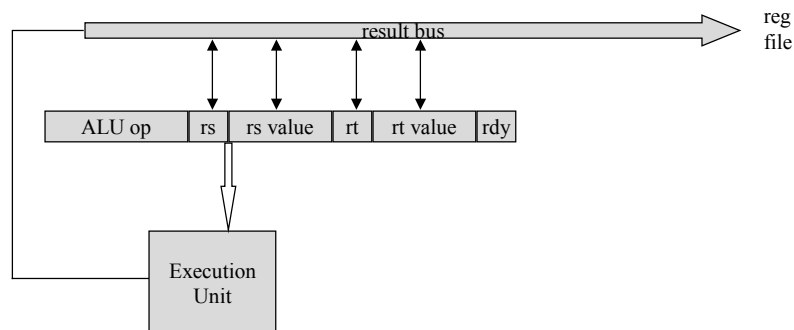
Dynamic or Out-of-Order Scheduling

- ◆ Issues (begins execution of) an instruction as soon as all of its dependences are satisfied, even if prior instructions are stalled.

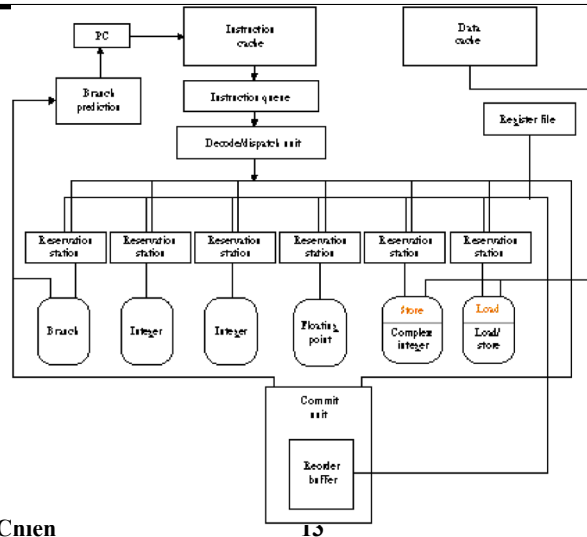
```
lw $6, 36($2)
add $5, $6, $4
lw $7, 1000($5)
sub $9, $12, $5
sw $5, 200($6)
add $3, $9, $9
and $11, $7, $6
```

Reservation Stations

- ◆ are a mechanism to allow dynamic scheduling



PowerPC 604, Intel Pentium Pro



CS 141 Chien

13

Feb 29, 2000

Dynamic Pipelining

- ◆ **Instruction issue enforced the value relationships which maintain sequential execution order semantics.**
- ◆ **Results are written to the register file in order**
 - Instructions are “retired” in order
 - Values in the meantime come thru the result bus
- ◆ **So, what really determines the order of the execution of the operations?**
- ◆ **Imagine doing this with precise exceptions.... !**

CS 141 Chien

14

Feb 29, 2000

Pipelining Summary

- ◆ $ET = \text{Number of instructions} * CPI * \text{cycle time}$
- ◆ *Data hazards* and *branch hazards* prevent CPI from reaching 1.0, but *forwarding* and *branch prediction* get it pretty close.
- ◆ Data hazards and branch hazards need to be detected by hardware.
- ◆ Pipeline control uses combinational logic. All data and control signals move together through the pipeline.
- ◆ Pipelining attempts to get CPI close to 1. To improve performance we must reduce CT (superpipelining) or CPI below one (superscalar, VLIW).