Lecture #10

****
Review -- 1 min
****
Registers not bottleneck
Avoid WAR, WAW
Not limited to basic blocks
Lasting Contributions
    -- dynamic scheduling
    -- register renaming
    -- load/store disambiguation

****
Outline - 1 min
****
Tomasulo Loop Example
Exceptions
In-order Commit

****
Lecture - 20 min
****
Tomasulo Loop Example

****
Admin - 3 min
****
HW 2A Friday 5pm
HW 2B Next Friday *start of class*

****
Lecture - 24 min
****
Exceptions
----

example -- page fault, arithmetic exception

Goal

save state of processor
handle exception
restore state of processor

In Simple DLX Integer Pipeline

1. force trap "instruction" into pipeline
   store PC of faulting instruction
   "squash" faulting instruction and all later instructions in pipeline
      --> make them NO-OPs
      --> not allowed to change state
2. when trap reaches WB, jump to exception-handling routine
      --> handle traps in program order
3. exception handling routine saves PC of faulting instruction
4. handle exception
5. load PC with PC of faulting instruction and resume

Subtleties

1. with delayed branch PC of faulting instruction not enough
      (what if the instruction in delay slot faults?)
      --> need N PC's for N delay slots

2. Out-of-order completion/State changes before WB
      what if update state in different stages

      Ugh!

      1. get wrong answer
         ("It's OK b/c we're getting an exception
         and aborting computation anyway")
         e.g. imprecise floating point exceptions
         (Alpha 21064, Power-2, MIPS R8000)

      2. re-start in middle of instruction
         e.g. VAX move string instruction

      3. serialize state changes

e.g. "precise mode" on modern RISCs
can be > 10x slower than "imprecise mode"

4. "roll out" of changes
e.g. reorder buffer, history buffer,

Precise Exceptions with Out-of-order Execution

Out-of-order execution --> instruction i+1 faults *after* instruction i writes result

Motivation
1. Debugging -- imprecise exceptions bad
get a fault and the core dump is a state that is kinda like
the state where you had the fault
(e.g. "the PC was around here somewhere")

2. Correct restart -- imprecise unacceptable
e.g. page fault -- need to restart in correct state

3. speculative execution (branch prediction) -- slow recovery unacceptable
later we'll talk about executing instructions
before their control dependencies resolve
(really aggressive branch prediction)

This fails pretty often (1-2%) --> need cheap recovery
--> need hardware support

Requirements of Hardware
----
2 sets of state --> extra buffering
1. architectural state

visible to issuing instructions
2. In-order state (to handle exceptions)

Reorder Buffer: Implementation

Data structures:
1. Start w. tomasulo
reservation stations, register result status
2. add "Reorder Buffer"
lookahead state -- all pending changes to
in-order state

fields:

entry#	Busy	Instruction	State	Destination	Value

Approach: 4 steps
1. issue
add instruction to tail of reorder buffer
(instructions added to RB in program order)

send instruction to reservation stations

update register result status
→ the reorder buffer that will produce the result

2. execute
resolve RAW then execute
3. write result
send result to RB
4. commit

When an instruction reaches end of reorder buffer
--> write data to register
-->write writes to memory

QUESTION: How do you handle exception?
--> clear all later instructions in reorder buffer

Can do same thing on incorrect branch prediction

--> notice if clear reorder buffer, we can restart in previous processor state

no registers updated
no memory updated

Simple Example

		Issue	FinishEx	WB	Commit
ADDD	F0, F2, F4				
SUBD	F6, F0, F8				

Reorder Buffer

Entry	Busy	Instr	State	Destination	Value
1					
2					

Reservation Stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
Add1							
Add2							

Register Status

	F0	F2	F4	F6	F8	...
Reorder #:						

reorder buffer
get class to step through how reorder buffer would be implemented (when combined with Tomasulo) in particular -- on issue -- what do you write into register result status? (A: reorder buffer entry -- write change CDB into reorder buffer (changes CDB to "go to " bus)
Renaming -- new name for register is its reorder buffer entry

QUESTION: A STUMPER
On an exception, how do you determine the correct PC to restart
Answer 1: store PC with each instruction in reorder buffer
    DA: doubles amount of storage
Answer 2: keep a counter; update it on a taken branch
    increment it on each committed instruction

****
Summay - 1 min
****

Tomasulo --
- Only slow down for
  - Structural hazards
  - Data dependencies

Exceptions
- Want precise exceptions for debugging
- Need precise exceptions for restart
- Want fast recovery for speculative execution
  - (notice that starting an instruction before
    I complete previous one is speculating
    that there will be no exception)
- Reorder Buffer