Lecture #7

****
Review -- 1 min
****
branches
    two part solution
        determin taken/not taken AND destination

    strategies: predict not taken, branch delay, (or predict taken)

****
Outline - 1 min
****
Next several days:
    longer pipeline
    data hazards
    control hazards

Today:
- o longer pipelines
    - MIPS R4000
    - floating point
- o ILP: compiler perspective

****
Lecture - 20 min
****

MIPS R4000

Fast clock rate
- --> get less done per stage
- --> more stages (8)
- --> memory access takes 3 cycles
  - for a cache HIT
  - (note: we've been ignoring cache misses so far
  - that's chapter 5
  - realize -- very unrealistic -- can take hundreds of
    - instruction-times
      - for a miss

8 Stage Pipeline

----

1. IF -- first half of fetch
   - -- pc selection
   - -- first half of mem access
2. IS -- second half of Icache access
3. RF -- instruction decode and reg fetch
4. EX -- execute
   - ALU op
   - effective address calculation
   - branch target calculation
   - condition evaluation
5. DF -- data fetch (1st half)
6. DS -- data fetch (2nd half)
7. TC -- tag check
8. WB -- write back for loads and register-register ops

BRANCH DELAY

Picture:

branch	IF	IS	RF	EX	DF	DS	TC	WB

QUESTION: what is branch
delay?
ANSWER: 3 cycles

Know branch target and condition after EX stage

stall stall stall IF IS RF ...

QUSTION: What techniques would help branches?
ANSWER:

(1) Predict not taken

(2) delayed branch

but -- we could only fill 50% of 1 delay slot
what are we going to do with 3 delay slots?

(3) cancelling branch ("branch likely")

→ can put “unsafe” instructions from dest in slot

--> they use 1 delay slot + predict taken + branch likely

o taken branch = delay slot + stall + stall
o untaken branch = delay slot

Anecdote -- they wanted to put a branch prediction hardware on chip, but they ran out of die space and cut it
--> branch cost is higher than the designers expected it to be.

(for spec, CPI = 1.54
branch accounts for .38 of that)

LOAD DELAY

Picture:

load r3, r2(0)	IF	IS	RF	EX	DF	DS	TC	WB
add r4, r3, r5		???						

QUESTION: what is load delay?
A: 2 cycles (might guess 3 but can start EX of dependent instruction during TC phase
    if tag mismatch --> don't unstall pipleline

Floating point

FLoating point is complicated
    -> either use lots of chip area or use lots of time

Even though transistors are cheap and getting cheaper,
    FP operation still relatively slow

(MIPS R4000)

Instruction	Clock Cycles
Add, subtract	4
Multiply	8
Divide	36
Square root	112
Compare	3

Pipeline picture -- think of it like regular DLX pipeline, but instruction
stays in EX stage for a long time

IF ID EX EX EX MEM WB

Different instructions may go to diferent EX units

		ADD_EX		
IF	ID	MULT_EX	MEM	WB
		DIV_EX		
		INT_EX		

<repeat as needed>

Implication: out of order completion
--> can get WAW hazards
e.g.
SQRT f1, f2
ADD f1, f2, f3

Stages can be *fully piplined*
--> start a new square root each cycle

In practice -- not worth the extra hardware -- partially piplelined
(back to back to back to back divides not that common)

In addition to latency, there is *initiation interval*
e.g. in MIPS R4000 -- can do 1 multiply in 8 cycles
but can start new multiply every 4 cycles

----> STRUCTURAL HAZARDS

MIPS R4000

Actually go the other way --
for most fp units, initiation rate = latency
FP units share hardware
(e.g. can't start an Add right after a multiply)

****
Admin - 3 min
****
project proposals Friday

Dont forget about HW2

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

Instruction level parallelism/ Loop parallelism

Motivation
ILP = Overlap execution of unrelated instructions

Within basic blocks not enough
gcc 17% control transfer
--> 5 instructions per branch

One opportunity: loops

Today -- compiler techniques
Next week -- hardware
(ironically -- hardware stuff was invented first
even though more complex

<Slide 1:
QUESTION: Where are dependencies?
QUESTINO: Where are stalls?

Compiler perspectives
Compiler worries about dependencies
whether or not they cause hazards

dependencies limit reordering of instructions
stalls/hazards depend on given pipeline
(stalls/hazards scheduling happens late in compilation
-- pipeline dependent phase
e.g. different for Pentium than PPro

True Dependencies/RAW Hazards
instruction i produces result used by instruction j

--> can't execute dependent instructions in parallel

Easy to determine for registers (fixed names)
Hard for memory
does 100(R4) = 20(R6)?
for different loop iterations, does 20(R6) = 20(R6)?

end slide 1>

<slide 2 -- revised FP loop minimizing stalls>
next step -- unroll loop

<slide 3: naive unrolled loop>

  got rid of fluff --> data dependencies cause hazards again
  want to reorder to avoid data dependency hazards
    but name dependencies (+ data dependencies) put
    partial order on instructions

Name dependence

  Antidependence -- WAR
    instruction j writes location that instruction i reads
    and instruction i is first

  output dependence
    instruction i and instruction j warite a location
    ordering must be preserved

QUESTION: where are name dependecies

Again, name dependencies hard for memory accesses
  does 100(R4) == 20(R6)?
  from different loop iterations, does 20(R6) = 20(R6)

  Our example -- compiler had to know
    0(R1) != -8(R1) != -16(R1) != -24(R1)

Control Dependence
----

example
    if(p1) {s1;}
    if(p2) {s2;}

S1 is control dependent on p1
S2 is control dependent on p2
S2 is not control dependent on p1

Obvious constraints on compiler
    cannot move instruction controlled by a branch before the branch
    cannot move instruction that is not controlled by a branch after a branch

Sometime relax control dependencies for better parallelism
    must preserve order of exceptions
    must preserve data flow

Loop-carried dependencies

<slide>

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

MIPS R4000 -

--- worse data and control hazards

floating point

-- out-of-order data hazards

-- structural hazards

compiler reordering

data dependence

name dependence

control dependence