Lecture #5

****
Review -- 1 min
****
Pipelining -- as natural as doing laundry
Simple instructions, Simple Datapath
Limits to pipelining
- helps throughput
- limited by slowest stage
- parallelism
- max speedup = # pipe stages
- "fill" and "drain" time limit speedup

****
Outline - 1 min
****

<good news/bad news>
- Speedup Equation
- 3 Hazards
  - structural
  - data
  - control
- Data Hazards

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

Speed Up Equation for Pipelining

Speedup = Time_unpipelined / Time_pipelined

CPU_Time = #Instr * CPI * Clock Cycle

#INstructions unchanged

IdealCPI * Pipeline Depth ClockCycle_unpipelined

$$\text{Speedup} = \frac{}{\text{IdealCPI + Pipe stall CPI}} * \frac{}{\text{Clock Cycle_pipelined}}$$

-----> <show picture of pipelined datapath to explain equation>

Why "IdealCPI * Piipeline Depth"
Why would clock cycle change?
Why are there "pipe stall CPI"?

For this chapter,

IdealCPI = 1

$$\text{Speedup} = \frac{\text{PipelineDepth}}{\text{1+PipeStallCPI}} * \frac{\text{ClockCycle_unpipelined}}{\text{ClockCycle_pipelined}}$$

Limits to pipelining: Hazards cause Pipeline Stalls
--> Increase PipeStallCPI

3 types of hazard
1. Structural Hazard -- no hardware available
2. Data hazard -- instruction depends on result of prior instruction
3. Control Hazard -- pipelining of instructions that change PC (e.g. branch)

Remember: Max Speedup = pipeline depth
Hazards are reason we don't have 1000-deep pipelines

What if we had a 1000-deep pipeline?
Structural -- need memory interface that can accept 1000 requests

as quickly as it now accepts 5

Data hazard --

Laundry lesson -- speedup comes from parallelism

to have 1000-deep pipeline, need 1000 instructions that you can execute in parallel
--> stall pipeline (e.g. stop) if instruction depends on any of the 1000 preceding instructions

Control Hazard --

Laundry lesson -- <the fill/drain laundry problem>

drain/fill pipeline (1000 cycles) on every branch

Relative Impact of Hazards

Structural -- not all that interesting
--> just spend more $'s on HW
(Tech trend -- # transistors going up fast)

Data hazard --

problem: dependencies among instructions

a = b + c
d = a + g

Can't calculate d until you know a

These dependencies put limit on how quickly you could execute a program
no matter how much hardware you have

Solution: execute second equation as soon as possible after first

Control hazard -- branches

hard problem -- basic blocks in computer programs are short

Solution -- start filling pipeline again as soon after a branch as possible
(avoid complete fill/drain)

****
Admin - 3 min
****

HW2 available on line

anyone need project partner?

comments on HW1

programming problem -- goal of class is to get hands dirty
most everyone figured out what was tricky

need to time lots of instructions to amortize timer overhead
--> need loop

need to unroll loops to amortize branch overhead

Superscalar processors adds one more trick:
to get MHz -- need to limit parallelism (1 instruction per cycle)
to get MIPS need to maximize parallelism

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

Talk about the 3 types of hazard

Structural Hazards

Book steps through example -- suppose you can only do one memory access per cycle
--> MEM phase of a load stalls FETCH the next instructions
--> solution = separate I and D cache

Not very interesting -- just throw more HW at the problem

Architectural question -- is extra hardware worth it?

General approach
1. calculate speedup with and without hazard
2. calculate cost of eliminating hazard
3. compare cost/performance

--> all current processors have separate I and D caches
--> most current processors do not fully pipeline floating point units

Data Hazards
---

add r1, r2, r3	IF	ID	EX	MEM	WB				
sub r4, r1, r5		IF	ID	EX	MEM	WB			
and r6, r1, r7			IF	ID	EX	MEM	WB		
or r8, r1, r9			IF	ID	EX	MEM	WB		
xor r10, r1, r11					IF	ID	EX	MEM	WB

Straw man solution -- stall

add r1, r2, r3	IF	ID	EX	MEM	WB				
sub r4, r1,									
r5	stall	stall	stall	stall	IF	ID	EX	MEM	WB
and r6, 41, r7								IF	

ID EX ...

Better solution: forward

<draw arrows in picture 1 -- add instruction knows answer after EX stage
--> "forward" result to later instructions>

How does forwarding work?

<use datapath + pipeline registers picture from last lecture>

1. result of EX stored in latch
2. add path to forward that result to input of EX
3. create paths from EX/MEM --> EX, MEM/WB-->EX, also, if WB in first part of cycle, can read registers in second part

<optional: depending on time>
Really, how does it work

if dest of prev instruction = one of my source registers, forward from EX/MEM to EX

if dest of 2-prev instruction = one of my source registers, forward from MEM/WB to EX

<end optional>

Load Stalls -- data hazard even with forwarding

---

lw r1, 0(r2)	IF	ID	EX	MEM	WB			
sub r4, r1, r5		IF	ID	stall	EX	MEM	WB	
<next instruction can't proceed b/c sub is stuck in pipeline>								
and r6, r7, r8			IF	stall	ID	EX	MEM	WB

Solution: Sofware scheduling

(don't use result of load in next instruction)
e.g.
lw r1, O(r2)
and r6, r7, r8
sub r4, r1, r5

Taxonomy of Data Hazards

<convention -- hazards named after the order that you must preserve for correct operation>

RAW -- read after write
previous examples -- need a previous result
true dependency

WAR -- write after read

e.g. add r1, r2, r3
sub r2, r4, r5

--> must make sure that add reads r2 before sub overwrites it
Not a problem in DLX pipeline -- earlier instruction alwas reads inputs before a later instruction writes outputs

WAW -- write after write

e.g. add r1, r2, r3
sub r1, r4, r5

r1 had better end up with r4 - r5

not a problem on dlx pipeline

Will see WAR and WAW later

Note: never see RAR

****
Summay - 1 min
****
<good news/bad news>

- Speedup Equation
- 3 Hazards
  - structural
  - data
  - control
- Data Hazards