Pipelining

-and-

Review for Final Exam

March 15, 2013

Outline

Pipelining

Goal: execute programs faster!

Basic idea

- separate processor into stages
- overlap the execution of consecutive instructions

Laundry room analogy

- it takes 60 minutes for one person to do their laundry
    - 20 minutes each for washer, dryer, and folding
- at 1pm, 5 people all want to wash their laundry ASAP

Laundry room analogy

One person in the laundry room at a time (not pipelined)

1pm	2pm	3pm	4pm	5pm
W D F				
	W D F			
		W D F		
			W D F	
				W D F

- 5 people = 5h
- 1 person / hour

Separate laundry room into three stations (pipelined)

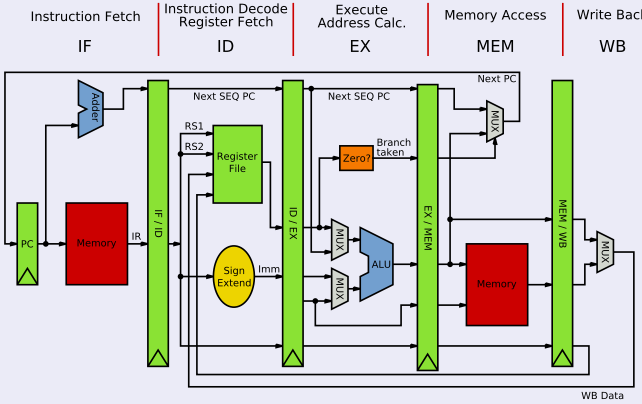
1pm	2pm	3pm
W D F		
W D	F	
W	D F	
	W D F	
	W D	F

- 5 people = 2h20m
- $\lim_{n \to \infty}$ 1 person / 20m

MIPS is designed for pipelining

Originally: **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages

MIPS 5-stage pipeline



Instruction pipelining

Pipeline stages

1. IF – **I**nstruction **F**etch
2. ID – **I**nstruction **D**ecode
3. EX – **EX**ecute
4. ME – **ME**mory access
5. WB – **W**rite **B**ack

Example

Clock cycle

	1	2	3	4	5	6	7	8
`lw $s0, 20($sp)`	IF	ID	EX	ME	WB			
`lw $s1, 24($sp)`		IF	ID	EX	ME	WB		
`lw $s2, 28($sp)`			IF	ID	EX	ME	WB	
`lw $s3, 32($sp)`				IF	ID	EX	ME	WB

Hazards

Hazard: a dependency that breaks pipelining

Two kinds of hazards

- **Data hazard** – need a value that hasn't been computed yet
- **Control hazard** – don't know which instruction comes next

Example: Data hazard

	Clock cycle						
	1	2	3	4	5	6	
`add  $t1, $t2, $t3`	IF	ID	EX	ME	WB		$t1 set here
`addi $t4, $t1, 1`		IF	ID	EX	ME	WB	

$t1 read here

Resolving hazards

Example: Data hazard

	Clock cycle						
	1	2	3	4	5	6	
`add $t1, $t2, $t3`	IF	ID	EX	ME	$\boxed{\text{WB}}$		$t1 set here
`addi $t4, $t1, 1`		IF	$\boxed{\text{ID}}$	EX	ME	WB	

$t1 read here

Solution: Processor inserts delays

	Clock cycle								
	1	2	3	4	5	6	7	8	9
`add $t1, $t2, $t3`	IF	ID	EX	ME	WB				
`addi $t4, $t1, 1`		IF	XX	XX	XX	ID	EX	ME	WB

Avoiding hazards

Delays negate the benefit of pipelining!

- would rather avoid data hazards then resolve them

Can reorder instructions to avoid data hazards

```
lw   $ra, 16($sp)                      add  $t1, $t2, $t3
lw   $t2, 20($sp)                      lw   $ra, 16($sp)
lw   $t3, 24($sp)          ⇝           lw   $t2, 20($sp)
add  $t1, $t2, $t3                     lw   $t3, 24($sp)
addi $t4, $t1, 1                       addi $t4, $t1, 1
```

Outline

I will provide . . .

- ASCII character encoding table
- interface of any relevant system calls
- a diagram of the stack frame layout
    - same as in slides, but black & white
- instruction format diagrams
- table of op/funct codes and syntax of any instructions you will need to assemble

You should bring . . .

- a pencil and eraser
- one page of notes (optional)
- a calculator (optional)

Focusing your study

Every problem on the final will be either . . .

- similar to a problem on Midterm 1
- similar to a problem on Midterm 2
- related to material covered since Midterm 2

From Midterm 1

- computer architecture vocabulary
- data representation
  - converting between bases
  - representing integers (two's complement)
  - null-terminated ASCII strings
- binary arithmetic (addition, multiplication)
- implementing math expressions in assembly

From Midterm 2

- procedure call vocabulary
- array addressing
  - integer arrays
  - strings (character arrays)
- implementing control structures in assembly
  - if-then, if-then-else, do-while, while, for
- procedure calling conventions
  - calling and returning from procedures
  - argument passing and return values
  - saving registers and managing the stack

Since Midterm 2

- subroutine linkage (implements the calling conventions)
- function pointers
- assembling to machine code
  - instruction formats
  - assembling R-type instructions
  - assembling I-type instructions (both math and branches)
  - (I won't have you assemble a J-type instruction)
- (nothing about pipelines either)

Exercise (basically how it will look on the final)

R:	0	rs	rt	rd	sh	fn

I:	op	rs	rt	addr/imm

Relevant instructions

```
# Instruction syntax       # op/fn
add  $rd, $rs, $rt         #  32
addi $rt, $rs, imm         #   8
beq  $rs, $rt, offset      #   4
bne  $rs, $rt, offset      #   5
```

Assemble the following program:

```
        bne    $t0, $t1, label
        addi   $t0, $t0, 10
label:  add    $t2, $t0, $t1
        beq    $t2, $t3, label
```

Name	Number
`$zero`	0
`$v0–$v1`	2–3
`$a0–$a3`	4–7
`$t0–$t7`	8–15
`$s0–$s7`	16–23
`$t8–$t9`	24–25
`$sp`	29
`$ra`	31