Lecture 1: Cost/Performance, DLX, Pipelining, Caches, Branch Prediction

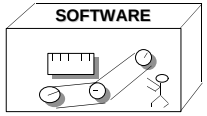
Prof. Fred Chong
ECS 250A Computer Architecture
Winter 1999

(Slides based upon Patterson UCB CS252 Spring 1998)

Computer Architecture Is …

the attributes of a [computing] system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation.

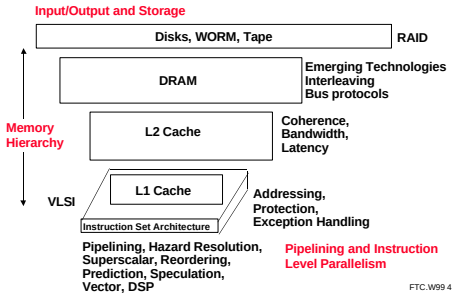
Amdahl, Blaaw, and Brooks, 1964



Computer Architecture's Changing Definition

- 1950s to 1960s: Computer Architecture Course
  Computer Arithmetic
- 1970s to mid 1980s: Computer Architecture Course
  Instruction Set Design, especially ISA appropriate for compilers
- 1990s: Computer Architecture Course
  Design of CPU, memory system, I/O system, Multiprocessors

Computer Architecture Topics



Computer Architecture Topics

Shared Memory, Message Passing, Data Parallelism

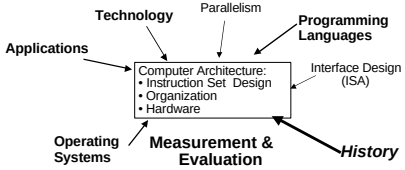
Network Interfaces

Topologies, Routing, Bandwidth, Latency, Reliability

Multiprocessors
Networks and Interconnections

ECS 250A Course Focus

Understanding the design techniques, machine structures, technology factors, evaluation methods that will determine the form of computers in 21st Century



Topic Coverage

Textbook: Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 2nd Ed., 1996.

- Performance/Cost, DLX, Pipelining, Caches, Branch Prediction
- ILP, Loop Unrolling, Scoreboarding, Tomasulo, Dynamic Branch Prediction
- Trace Scheduling, Speculation
- Vector Processors, DSPs
- Memory Hierarchy
- I/O
- Interconnection Networks
- Multiprocessors

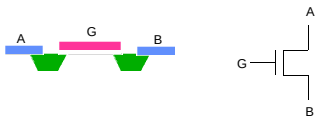
ECS250A: Staff

Instructor:	**Fred Chong**
	Office: EUII-3031 chong@cs
	Office Hours: Mon 4-6pm or by appt.
T. A:	**Diana Keen**
	Office: EUII-2239 keend@cs
	TA Office Hours: Fri 1-3pm
Class:	**Mon 6:10-9pm**
Text:	**Computer Architecture: A Quantitative Approach, Second Edition (1996)**
Web page:	**http://arch.cs.ucdavis.edu/~chong/250A/**
	Lectures available online before 1PM day of lecture
Newsgroup: ucd.class.cs250a{.d}	

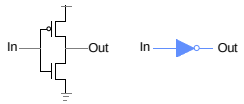
Grading

- **Problem Sets 35%**
- **1 In-class exam (prelim simulation) 20%**
- **Project Proposals and Drafts 10%**
- **Project Final Report 25%**
- **Project Poster Session (CS colloquium) 10%**

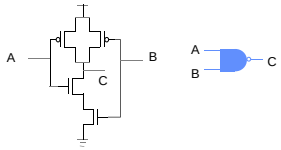
VLSI Transistors



CMOS Inverter



CMOS NAND Gate

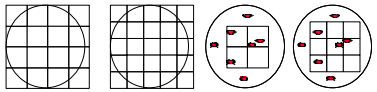


Integrated Circuits Costs

$$\text{IC cost} = \frac{\text{Die cost} + \text{Testing cost} + \text{Packaging cost}}{\text{Final test yield}}$$

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} * \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi * (\text{Wafer\_diam} / 2)^2}{\text{Die Area}} - \frac{\pi * \text{Wafer\_diam}}{\sqrt{2 * \text{Die Area}}} - \text{Test dies}$$



$$\text{Die Yield} = \text{Wafer yield} * \left\{1 + \frac{\text{Defects\_per\_unit\_area} * \text{Die\_Area}}{\alpha}\right\}^{-\alpha}$$

Die Cost goes roughly with die area^4

Real World Examples

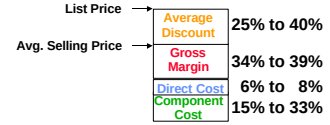
Chip	Metal layers	Line width	Wafer cost	Defect /cm²	Area mm²	Dies/ wafer	Yield	Die Cost
386DX	2	0.90	$900	1.0	43	360	71%	$4
486DX2	3	0.80	$1200	1.0	81	181	54%	$12
PowerPC 601	4	0.80	$1700	1.3	121	115	28%	$53
HP PA 7100	3	0.80	$1300	1.0	196	66	27%	$73
DEC Alpha	3	0.70	$1500	1.2	234	53	19%	$149
SuperSPARC	3	0.70	$1700	1.6	256	48	13%	$272
Pentium	3	0.80	$1500	1.5	296	40	9%	$417

– From "Estimating IC Manufacturing Costs," by Linley Gwennap, *Microprocessor Report*, August 2, 1993, p. 15

Cost/Performance

What is Relationship of Cost to Price?

- **Component Costs**
- **Direct Costs** (add 25% to 40%) recurring costs: labor, purchasing, scrap, warranty
- **Gross Margin** (add 82% to 186%) nonrecurring costs: R&D, marketing, sales, equipment maintenance, rental, financing cost, pretax profits, taxes
- **Average Discount** to get List Price (add 33% to 66%): volume discounts and/or retailer markup

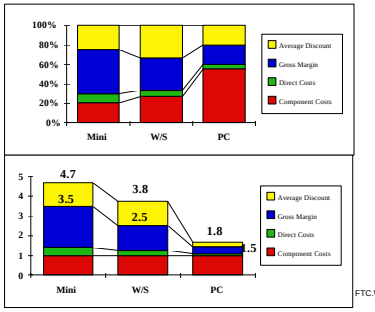


Chip Prices (August 1993)

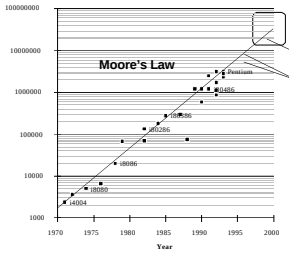
- **Assume purchase 10,000 units**

Chip	Area mm²	Mfg. cost	Price	Multi-plier	Comment
386DX	43	$9	$31	3.4	Intense Competition
486DX2	81	$35	$245	7.0	No Competition
PowerPC 601	121	$77	$280	3.6	
DEC Alpha	234	$202	$1231	6.1	Recoup R&D?
Pentium	296	$473	$965	2.0	Early in shipments

Summary: Price vs. Cost



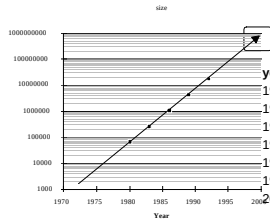
Technology Trends: Microprocessor Capacity



Alpha 21264: 15 million
Pentium Pro: 5.5 million
PowerPC 620: 6.9 million
Alpha 21164: 9.3 million
Sparc Ultra: 5.2 million

CMOS improvements:
- **Die size: 2X every 3 yrs**
- **Line width: halve / 7 yrs**

Memory Capacity (Single Chip DRAM)



year	size(Mb)	cyc time
1980	0.0625	250 ns
1983	0.25	220 ns
1986	1	190 ns
1989	4	165 ns
1992	16	145 ns
1996	64	120 ns
2000	256	100 ns

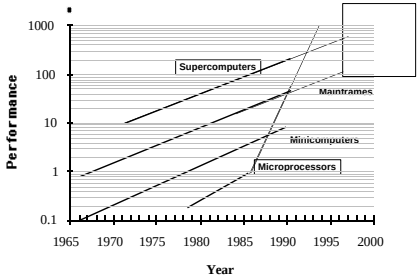
FTC.W99 19

Technology Trends (Summary)

	Capacity	Speed (latency)
Logic	2x in 3 years	2x in 3 years
DRAM	4x in 3 years	2x in 10 years
Disk	4x in 3 years	2x in 10 years

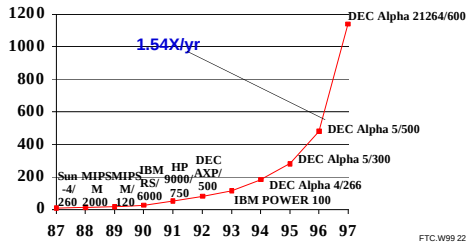
FTC.W99 20

Processor Performance Trends



FTC.W99 21

Processor Performance (1.35X before, 1.55X now)



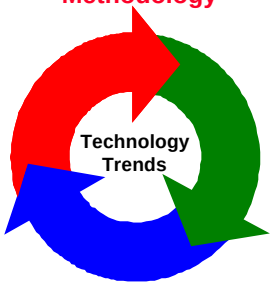
FTC.W99 22

Performance Trends (Summary)

- Workstation performance (measured in Spec Marks) improves roughly 50% per year (2X every 18 months)
- Improvement in cost performance estimated at 70% per year

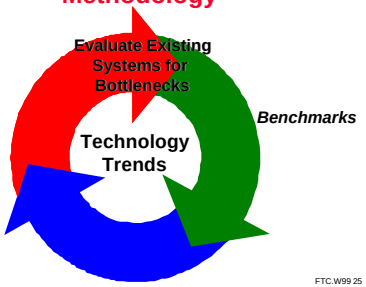
FTC.W99 23

Computer Engineering Methodology



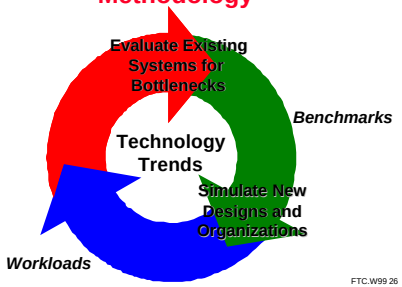
FTC.W99 24

Computer Engineering Methodology



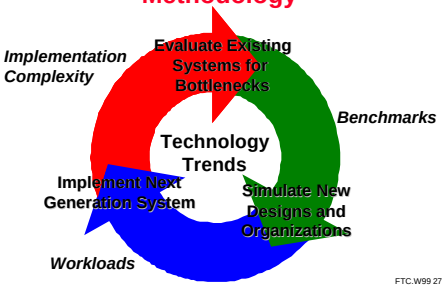
FTC.W99 25

Computer Engineering Methodology



FTC.W99 26

Computer Engineering Methodology



FTC.W99 27

Measurement Tools

- Benchmarks, Traces, Mixes
- Hardware: Cost, delay, area, power estimation
- Simulation (many levels)
  - ISA, RT, Gate, Circuit
- Queuing Theory
- Rules of Thumb
- Fundamental "Laws"/Principles

FTC.W99 28

The Bottom Line: Performance (and Cost)

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concodre	3 hours	1350 mph	132	178,200

- Time to run the task (ExTime)
  - Execution time, response time, latency
- Tasks per day, hour, week, sec, ns … (Performance)
  - Throughput, bandwidth

FTC.W99 29

The Bottom Line: Performance (and Cost)

"X is n times faster than Y" means

$$\frac{\text{ExTime(Y)}}{\text{ExTime(X)}} = \frac{\text{Performance(X)}}{\text{Performance(Y)}}$$

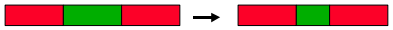
- Speed of Concorde vs. Boeing 747
- Throughput of Boeing 747 vs. Concorde

FTC.W99 30

Amdahl's Law

Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected

Amdahl's Law

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times \left[(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}} \right]$$

$$\text{Speedup}_{overall} = \frac{\text{ExTime}_{old}}{\text{ExTime}_{new}} = \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

Amdahl's Law

- **Floating point instructions improved to run 2X; but only 10% of actual instructions are FP**

$\text{ExTime}_{new} =$

$\text{Speedup}_{overall} =$

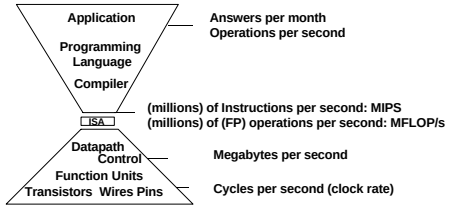
Amdahl's Law

- **Floating point instructions improved to run 2X; but only 10% of actual instructions are FP**

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times (0.9 + .1/2) = 0.95 \times \text{ExTime}_{old}$$

$$\text{Speedup}_{overall} = \frac{1}{0.95} = 1.053$$

Metrics of Performance



Aspects of CPU Performance

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

Cycles Per Instruction

"Average Cycles per Instruction"

CPI = (CPU Time * Clock Rate) / Instruction Count
= Cycles / Instruction Count

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^{n} \text{CPI}_i * I_i$$

"Instruction Frequency"

$$\text{CPI} = \sum_{i=1}^{n} \text{CPI}_i * F_i \quad \text{where} \quad F_i = \frac{I_i}{\text{Instruction Count}}$$

Invest Resources where time is Spent!

Example: Calculating CPI

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			1.5	

Typical Mix

SPEC: System Performance Evaluation Cooperative

- First Round 1989
  - 10 programs yielding a single number ("SPECmarks")
- Second Round 1992
  - SPECInt92 (6 integer programs) and SPECfp92 (14 floating point programs)
    - » Compiler Flags unlimited. March 93 of DEC 4000 Model 610:

      `spice:` `unix.c:/def=(sysv,has_bcopy,"bcopy(a,b,c)= memcpy(b,a,c)"`

      `wave5:` `/ali=(all,dcom=nat)/ag=a/ur=4/ur=200`

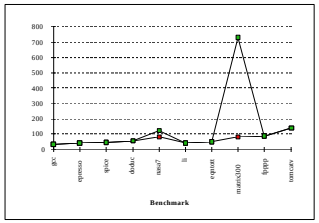
      `nasa7:` `/norecu/ag=a/ur=4/ur2=200/lc=blas`
- Third Round 1995
  - new set of programs: SPECint95 (8 integer programs) and SPECfp95 (10 floating point)
  - "benchmarks useful for 3 years"
  - Single flag setting for all programs: SPECint_base95, SPECfp_base95

How to Summarize Performance

- Arithmetic mean (weighted arithmetic mean) tracks execution time: $\Sigma(T_i)/n$ or $\Sigma(W_i*T_i)$
- Harmonic mean (weighted harmonic mean) of rates (e.g., MFLOPS) tracks execution time: $n/\Sigma(1/R_i)$ or $n/\Sigma(W_i/R_i)$
- Normalized execution time is handy for scaling performance (e.g., X times faster than SPARCstation 10)
- But do not take the arithmetic mean of normalized execution time, use the geometric: $(\Pi x_i)^{1/n}$

SPEC First Round

- One program: 99% of time in single line of code
- New front-end compiler could improve dramatically



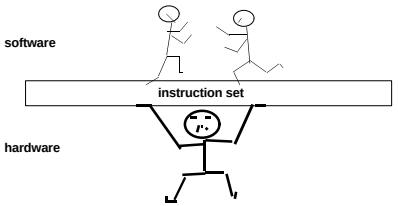
Impact of Means on SPECmark89 for IBM 550

	Ratio to VAX:		Time:		Weighted Time:	
Program	*Before*	*After*	*Before*	*After*	*Before*	*After*
gcc	30	29	49	51	8.91	9.22
espresso	35	34	65	67	7.64	7.86
spice	47	47	510	510	5.69	5.69
doduc	46	49	41	38	5.81	5.45
nasa7	78	144	258	140	3.43	1.86
li	34	34	183	183	7.86	7.86
eqntott	40	40	28	28	6.68	6.68
matrix300	78	730	58	6	3.43	0.37
fpppp	90	87	34	35	2.97	3.07
tomcatv	33	138	20	19	2.01	1.94
Mean	54	72	124	108	54.42	49.99
	Geometric		*Arithmetic*		*Weighted Arith.*	
	Ratio	1.33	Ratio	1.16	Ratio	1.09

Performance Evaluation

- "For better or worse, benchmarks shape a field"
- Good products created when have:
  - Good benchmarks
  - Good ways to summarize performance
- Given sales is a function in part of performance relative to competition, investment in improving product as reported by performance summary
- If benchmarks/summary inadequate, then choose between improving product for real programs vs. improving product to get more sales; Sales almost always wins!
- Execution time is the measure of computer performance!

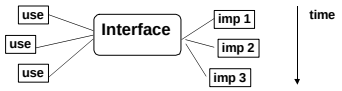
Instruction Set Architecture (ISA)



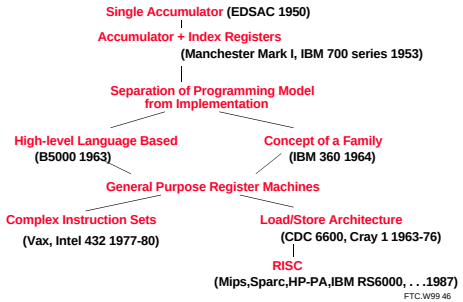
Interface Design

A good interface:

- Lasts through many implementations (portability, compatibility)
- Is used in many differeny ways (generality)
- Provides convenient functionality to higher levels
- Permits an efficient implementation at lower levels



Evolution of Instruction Sets



Evolution of Instruction Sets

- Major advances in computer architecture are typically associated with landmark instruction set designs
  - Ex: Stack vs GPR (System 360)
- Design decisions must take into account:
  - technology
  - machine organization
  - programming languages
  - compiler technology
  - operating systems
- And they in turn influence these

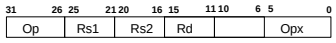
A "Typical" RISC

- 32-bit fixed format instruction (3 formats)
- 32 32-bit GPR (R0 contains zero, DP take pair)
- 3-address, reg-reg arithmetic instruction
- Single address mode for load/store: base + displacement
  - no indirection
- Simple branch conditions
- Delayed branch

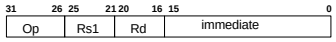
see: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC, CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

Example: MIPS

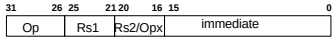
Register-Register



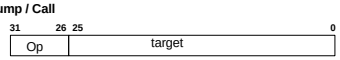
Register-Immediate



Branch



Jump / Call



FTC.W99 49

Summary, #1

- Designing to Last through Trends

	Capacity	Speed
Logic	2x in 3 years	2x in 3 years
DRAM	4x in 3 years	2x in 10 years
Disk	4x in 3 years	2x in 10 years

- 6yrs to graduate => 16X CPU speed, DRAM/Disk size
- Time to run the task
  - Execution time, response time, latency
- Tasks per day, hour, week, sec, ns, …
  - Throughput, bandwidth
- "X is n times faster than Y" means

$$\frac{\text{ExTime(Y)}}{\text{ExTime(X)}} = \frac{\text{Performance(X)}}{\text{Performance(Y)}}$$

FTC.W99 50

Summary, #2

- Amdahl's Law:

$$\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

- CPI Law:

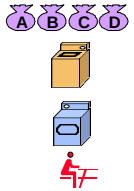
$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

- Execution time is the REAL measure of computer performance!
- Good products created when have:
  - Good benchmarks, good ways to summarize performance
- Die Cost goes roughly with die area^4
- Can PC industry support engineering/research investment?

FTC.W99 51

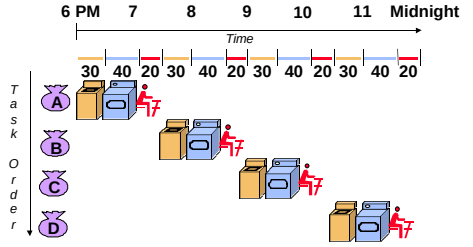
Pipelining: Its Natural!

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes



FTC.W99 52

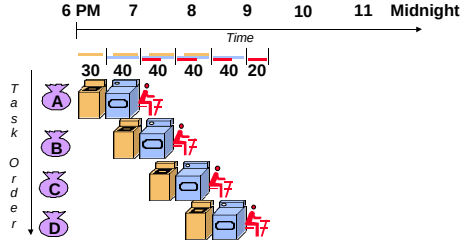
Sequential Laundry



- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

FTC.W99 53

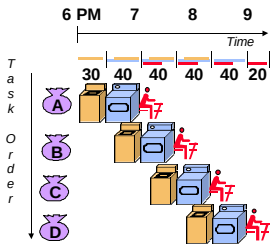
Pipelined Laundry Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

FTC.W99 54

Pipelining Lessons



- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

FTC.W99 55

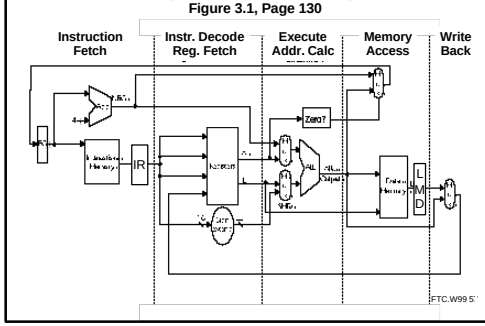
Computer Pipelines

- Execute billions of instructions, so throughput is what matters
- DLX desirable features: all instructions same length, registers located in same place in instruction format, memory operands only in loads or stores

FTC.W99 56

5 Steps of DLX Datapath

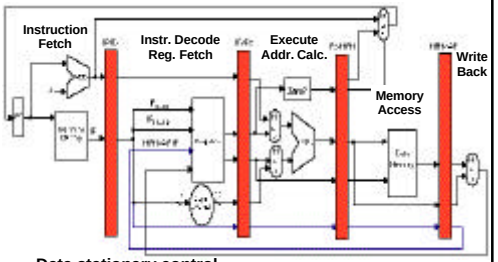
Figure 3.1, Page 130



FTC.W99 57

Pipelined DLX Datapath

Figure 3.4, page 137

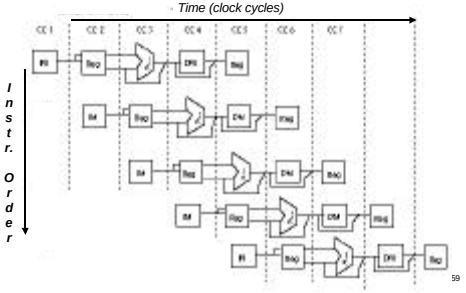


- Data stationary control
  - local decode for each instruction phase / pipeline stage

FTC.W99 58

Visualizing Pipelining

Figure 3.3, Page 133



59

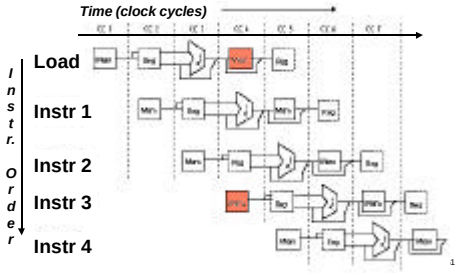
Its Not That Easy for Computers

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support this combination of instructions (single person to fold and put clothes away)
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline (missing sock)
  - Control hazards: Pipelining of branches & other instructionsstall the pipeline until the hazardbubbles" in the pipeline

FTC.W99 60

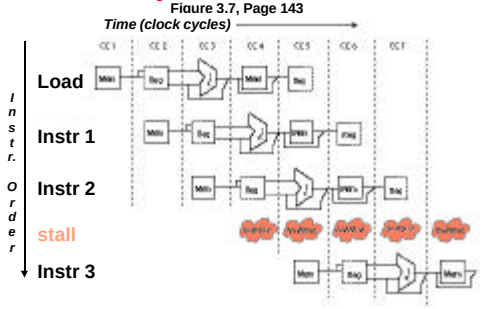
One Memory Port/Structural Hazards

Figure 3.6, Page 142



One Memory Port/Structural Hazards

Fiqure 3.7, Page 143



Speed Up Equation for Pipelining

$$CPI_{pipelined} = \text{Ideal CPI} + \text{Pipeline stall clock cycles per instr}$$

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle}_{unpipelined}}{\text{Clock Cycle}_{pipelined}}$$

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle}_{unpipelined}}{\text{Clock Cycle}_{pipelined}}$$

FTC.W99 63

Example: Dual-port vs. Single-port

- Machine A: Dual ported memory
- Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Ideal CPI = 1 for both
- Loads are 40% of instructions executed

$$\text{SpeedUp}_A = \text{Pipeline Depth}/(1 + 0) \times (\text{clock}_{unpipe}/\text{clock}_{pipe}) = \text{Pipeline Depth}$$

$$\text{SpeedUp}_B = \text{Pipeline Depth}/(1 + 0.4 \times 1) \times (\text{clock}_{unpipe}/(\text{clock}_{unpipe} / 1.05)) = (\text{Pipeline Depth}/1.4) \times 1.05 = 0.75 \times \text{Pipeline Depth}$$

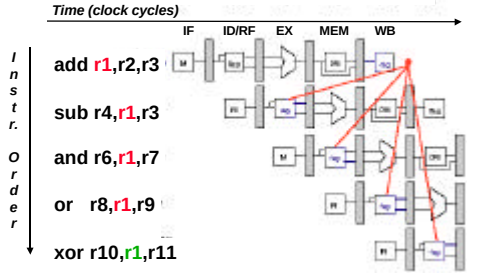
$$\text{SpeedUp}_A / \text{SpeedUp}_B = \text{Pipeline Depth}/(0.75 \times \text{Pipeline Depth}) = 1.33$$

- Machine A is 1.33 times faster

FTC.W99 64

Data Hazard on R1

Figure 3.9, page 147



Three Generic Data Hazards

$Instr_I$ followed by $Instr_J$

- Read After Write (RAW)
  $Instr_J$ tries to read operand before $Instr_I$ writes it

FTC.W99 66

Three Generic Data Hazards

$Instr_I$ followed by $Instr_J$

- Write After Read (WAR)
  $Instr_J$ tries to write operand ***before*** $Instr_I$ reads i
  - Gets wrong operand
- Can't happen in DLX 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5

FTC.W99 67

Three Generic Data Hazards

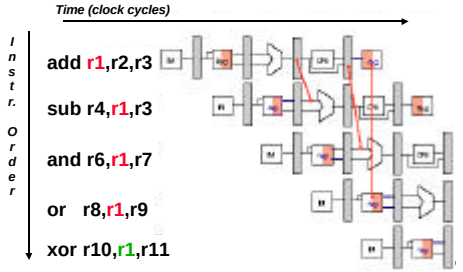
$Instr_I$ followed by $Instr_J$

- Write After Write (WAW)
  $Instr_J$ tries to write operand ***before*** $Instr_I$ writes it
  - Leaves wrong result ($Instr_I$ not $Instr_J$)
- Can't happen in DLX 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Writes are always in stage 5
- Will see WAR and WAW in later more complicated pipes

FTC.W99 68

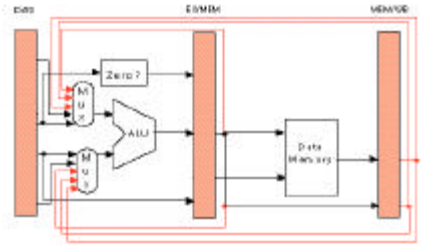
Forwarding to Avoid Data Hazard

Figure 3.10, Page 149



HW Change for Forwarding

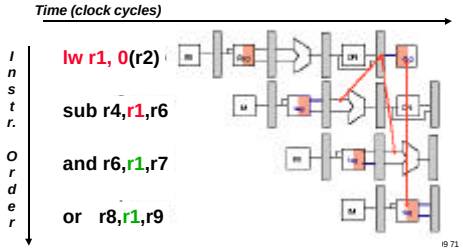
Figure 3.20, Page 161



FTC.W99 70

Data Hazard Even with Forwarding

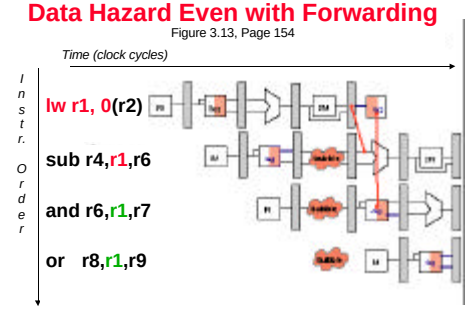
Figure 3.12, Page 153



I9 71

Data Hazard Even with Forwarding

Figure 3.13, Page 154



Software Scheduling to Avoid Load Hazards

Try producing fast code for

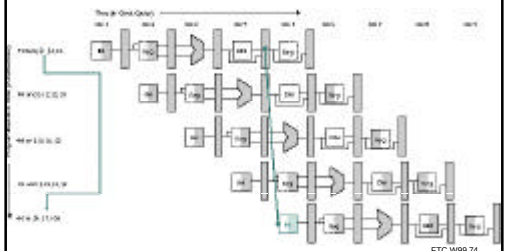
a = b + c;

d = e – f;

assuming a, b, c, d ,e, and f in memory.

Slow code:		Fast code:	
LW	Rb,b	LW	Rb,b
LW	Rc,c	LW	Rc,c
ADD	Ra,Rb,Rc	LW	Re,e
SW	a,Ra	ADD	Ra,Rb,Rc
LW	Re,e	LW	Rf,f
LW	Rf,f	SW	a,Ra
SUB	Rd,Re,Rf	SUB	Rd,Re,Rf
SW	d,Rd	SW	d,Rd

Control Hazard on Branches Three Stage Stall

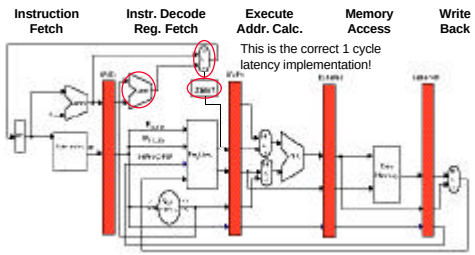


Branch Stall Impact

- If CPI = 1, 30% branch, Stall 3 cycles => new CPI = 1.9!
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- DLX branch tests if register = 0 or ° 0
- DLX Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

Pipelined DLX Datapath

Figure 3.22, page 163



Four Branch Hazard Alternatives

#1: Stall until branch direction is clear

#2: Predict Branch Not Taken
- Execute successor instructions in sequence
- "Squash" instructions in pipeline if branch actually taken
- Advantage of late pipeline state update
- 47% DLX branches not taken on average
- PC+4 already calculated, so use it to get next instruction

#3: Predict Branch Taken
- 53% DLX branches taken on average
- But haven't calculated branch target address in DLX
  - » DLX still incurs 1 cycle branch penalty
  - » Other machines: branch target known before outcome

Four Branch Hazard Alternatives

#4: Delayed Branch
- Define branch to take place AFTER a following instruction

```
branch instruction
  sequential successor1
  sequential successor2
  ........
  sequential successorn
branch target if taken
```

Branch delay of length *n*

- 1 slot delay allows proper decision and branch target address in 5 stage pipeline
- DLX uses this

Delayed Branch

- **Where to get instructions to fill branch delay slot?**
  - Before branch instruction
  - From the target address: only valuable when branch taken
  - From fall through: only valuable when branch not taken
  - Cancelling branches allow more slots to be filled
- **Compiler effectiveness for single branch delay slot:**
  - Fills about 60% of branch delay slots
  - About 80% of instructions executed in branch delay slots useful in computation
  - About 50% (60% x 80%) of slots usefully filled
- **Delayed Branch downside: 7-8 stage pipelines, multiple instructions issued per clock (superscalar)**

Evaluating Branch Alternatives

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

Scheduling scheme	*Branch penalty*	*CPI*	*speedup v. unpipelined*	*speedup v. stall*
Stall pipeline	3	1.42	3.5	1.0
Predict taken	1	1.14	4.4	1.26
Predict not taken	1	1.09	4.5	1.29
Delayed branch	0.5	1.07	4.6	1.31

Conditional & Unconditional = 14%, 65% change PC

Pipelining Summary

- **Just overlap tasks, and easy if tasks are independent**
- **Speed Up Š Pipeline Depth; if ideal CPI is 1, then:**

$$\text{Speedup} = \frac{\text{Pipeline Depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle Unpipelined}}{\text{Clock Cycle Pipelined}}$$

- **Hazards limit performance on computers:**
  - Structural: need more HW resources
  - Data (RAW,WAR,WAW): need forwarding, compiler scheduling
  - Control: delayed branch, prediction