

UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Computer Architecture
ECE 568

Part 1

Introduction

Israel Koren
Fall 2009

ECE568/Koren Part.1.1

Adapted from Patterson, Katz and Kubiawicz © UCB

Copyright 2005 UCB & Morgan Kaufmann

Coping with ECE 568

- ◆ **Students with varied background**
- ◆ **Prerequisites ECE221 and ECE232 or equivalent courses in Digital Design and Hardware Organization**
- ◆ **ECE 232**
"Computer Organization and Design" 2nd ed. by Patterson and Hennessy, ISBN 1-55860-428-6
 - **Chapters 1 to 7 + App. A (in "Computer Architecture: A Quantitative Approach") if never took ECE232**
 - **Chapter 2 in "Computer Architecture: A Quantitative Approach"**
- ◆ **First lectures- review of Performance and Pipelining (Chapter 1 + Appendix A)**

ECE568/Koren Part.1.2

Copyright 2008 Koren UMass

What you should know from ECE 221 and 232

- ◆ **Logic design**
 - logical equations, schematic diagrams, FSMs, components (MUX, ALU)
- ◆ **Basic machine structure**
 - processor (data path, control, arithmetic), memory, I/O
- ◆ **Read and write in an assembly language**
 - MIPS preferred
- ◆ **Understand the concepts of pipelining and virtual memory**

Textbook and references

- ◆ **Recommended book:** D.A. Patterson and J.L. Hennessy, *Computer Architecture: A Quantitative Approach*, 4th edition, Morgan-Kaufmann, 2007.
- ◆ **Recommended reading:**
 - J.P. Shen and M.H. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors*, McGraw-Hill, 2005.
 - M. J. Flynn, *Computer Architecture: Pipelined and Parallel Processor Design*, Jones and Bartlett, 1995.
 - V. P. Heuring and H. F. Jordan, *Computer Systems Design and Architecture*, Addison-Wesley, 1997.
 - K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, 1993.
 - Technical papers.

Course Outline

- ◆ I. Introduction
- ◆ II. Performance Analysis (Ch.1+)
- ◆ III. Processor Design: Instruction-level Parallelism, Pipelining (App.A,Ch.3-4)
- ◆ IV. Memory Design: Memory Hierarchy, Cache Memory, Secondary Memory (Ch.5)
- ◆ V. Storage systems and Input/Output (Ch.7)
- ◆ VI. Power-Aware Computing
- ◆ VII. Vector Computers (App.G)

Administrative Details

- ◆ Instructor: Prof. Israel Koren
- ◆ KEB 309E, Tel. 545-2643
- ◆ Email: koren@ecs.umass.edu
- ◆ Office Hours: 4:00-5:00 pm, Tues. & Thur.
- ◆ TA - None
- ◆ Course web page: Lecture notes, OWL assignments' due dates, technical papers and other details regarding the course will be available at:
<http://www.ecs.umass.edu/ece/koren/ece568>

Grading

- ◆ Midterm I - 25%
 - Wed. Oct. 21, 4-6pm (tentative)
- ◆ Midterm II - 25%
 - Mon. Nov. 30, 4-6pm (tentative)
- ◆ OWL quizzes - 15%
- ◆ Final Exam - 35%

<http://www.ecs.umass.edu/ece/koren/ece568>

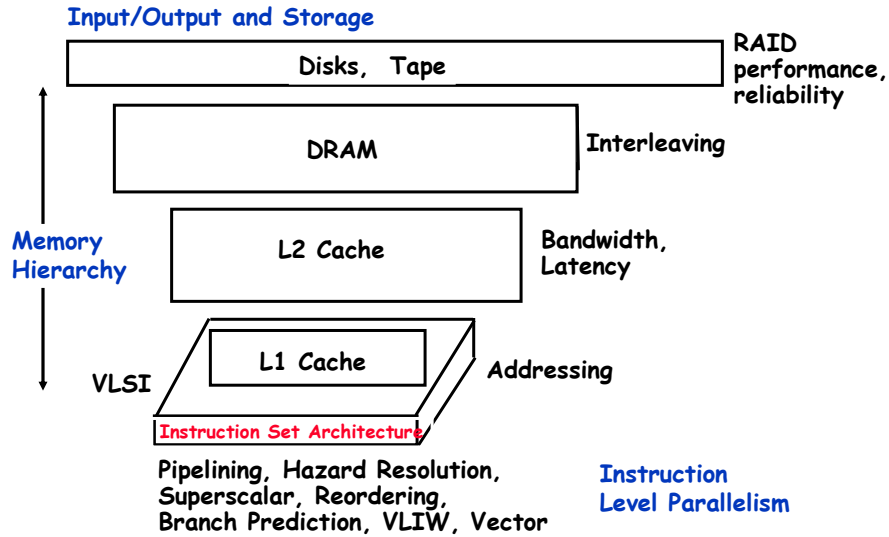
What is "Computer Architecture"

Computer Architecture =
Instruction Set Architecture +
Machine Organization
(e.g., Pipelining, Memory Hierarchy,
Storage systems, etc)

IBM 360 (minicomputer, mainframe, supercomputer)

Intel X86 vs. Transmeta

Computer Architecture Topics



ECE568/Koren Part.1.9

Adapted from Patterson, Katz and Kubiawicz © UCB

Copyright 2005 UCB & Morgan Kaufmann

Example ISAs (Instruction Set Architectures)

- ◆ Digital Alpha (v1, v3) 1992
- ◆ HP PA-RISC (v1.1, v2.0) 1986
- ◆ Sun Sparc (v8, v9) 1987
- ◆ MIPS (MIPS I, II, III, IV, V) 1986
- ◆ Intel (8086, 80286, 80386, 80486, Pentium, MMX, ...) 1978

RISC vs. CISC

ECE568/Koren Part.1.10

Adapted from Patterson, Katz and Kubiawicz © UCB

Copyright 2005 UCB & Morgan Kaufmann

Characteristics of RISC

- ◆ Only Load/Store instructions access memory
- ◆ A relatively large number of registers
- ◆
- ◆

Goals of new computer designs

- ◆ Higher performance
- ◆ More functionality (e.g., MMX)
- ◆ Other design objectives? (examples)
-
-

How to measure performance?

- Time to run the task
 - Execution time, response time, latency
 - Performance may be defined as $1 / \text{Ex_Time}$
- Tasks per day, hour, minute, ...
 - Throughput, bandwidth
- Individual user vs. "System manager"

Speedup

$$\text{performance}(x) = \frac{1}{\text{execution_time}(x)}$$

"Y is n times faster than X" means

$$n = \text{speedup} = \frac{\text{Execution_time}(\text{old / brand } x)}{\text{Execution_time}(\text{new / brand } y)} = 1 + w/100$$

Speedup must be greater than 1; speedup of $w\%$

$$T_x/T_y = 3/2 = 1.5 \quad \text{but not} \quad T_y/T_x = 2/3 = 0.67$$

MIPS and MFLOPS

- ◆ MIPS (Million Instructions Per Second)
 - Can we compare two different CPUs using MIPS?
- ◆ MFLOPS (Million Floating-point operations Per Sec.)
 - Application dependent (e.g., compiler)
- ◆ Still useful for benchmarks
- ◆ Benchmarks: e.g., **SPEC CPU 2000**: 26 applications (with inputs)
 - **SPECint2000**: Twelve integer, e.g., gcc, gzip, perl
 - **SPECfp2000**: Fourteen floating-point intensive, e.g., equake

SPEC CPU 2000

SPECint2000

SPECfp2000

Benchmark Language Category

164.gzip	C	Compression
175.vpr	C	FPGA Circuit Place& Route
176.gcc	C	C Compiler
181.mcf	C	Combinatorial Optimization
186.crafty	C	Game Playing: Chess
197.parser	C	Word Processing
252.eon	C++	Computer Visualization
253.perlbnk	C	PERL Prog Language
254.gap	C	Group Theory, Interpreter
255.vortex	C	Object-oriented Database
256.bzip2	C	Compression
300.twolf	C	Place and Route Simulator

Benchmark Language Category

168.wupwise	Fortran77	Quantum Chromodynamics
171.swim	Fortran77	Shallow Water Modeling
172.mgrid	Fortran77	Multi-grid Solver
173.applu	Fortran77	Partial Differential Equations
177.mesa	C	3-D Graphics Library
178.galgel	Fortran90	Fluid Dynamics
179.art	C	Image Recognition /Neural Nets
183.equake	C	Seismic Wave Propagation
187.facerec	Fortran 90	Face Recognition
188.amp	C	Computational Chemistry
189.lucas	Fortran90	Primality Testing
191.fma3d	Fortran90	Finite-element Crash - Nuclear Physics
200.sixtrack	Fortran77	Accelerator Design
301.apsi	Fortran77	Meteorology: Pollutant Distribution

www.specbench.org/cpu2000

ECE568/Koren Part.1 .15

Copyright 2008 Koren UMass

Other Benchmarks

Workload Category

Example Benchmark Suite

Source: L. John, "Performance Evaluation: Techniques, Tools and Benchmarks," The Computer Engineering Handbook, CRC Press, 2001.

www.ece.utexas.edu/projects/ece/lca/courses/382m/

CPU Benchmarks - Uniprocessor

SPEC CPU 2000
Java Grande Forum Benchmarks
SciMark, ASCI
SPLASH, NASPAR
MediaBench
EEMBC benchmarks
BDTI benchmarks
SPECjvm98, CaffeineMark
SPECjBB2000, VolanoMark
Java Grande Forum Benchmarks
SciMark

CPU - Parallel Processor
Multimedia
Embedded

Digital Signal Processing
Java - Client side
Java - Server side
Java - Scientific

Transaction Processing

On-Line Transaction Processing
Transaction Processing
Decision Support Systems

Web Server
Electronic commerce
Mail-server
Network File System
Personal Computer

TPC-C, TPC-W

TPC-H, TP-R
SPEC web99, TPC-W, VolanoMark
TPC-W, SPECjBB2000
SPECmail2000
SPEC SFS 2.0
SYSMARK, WinBench, DMarkMAX99

ECE568/Koren Part.1 .16

Copyright 2008 Koren UMass

Synthetic Benchmarks

Whetstone Benchmark

www.cse.clrc.ac.uk/disco/Benchmarks/whetstone.shtml

Rank	Machine	Mflop ratings (VI=1024)			Total CPU (seconds)	MWIPS
		N2	N3	N8		
1	Intel Woodcrest 3GHz 4MBL2	1966	4588	2907	3.3	10560
2	Intel Woodcrest 3GHz-533	1966	4588	3069	3.3	10451
3	IBM eServer p5	1966	1966	1625	6.2	6219
4	SunFire V20 2.2GHz	1311	1298	1481	7.7	4496
5	IBM eServer p5 575/1.5	1966	1529	1315	7.8	4874
6	AMD Opteron852/2600	1513	1547	1771	8.1	4488
7	HP DL380 Pentium4/3600	1966	1720	607	8.4	4408
8	Dell PowerEdge 1MBL2	1966	1720	607	8.4	4351
9	Dell PowerEdge 2MBL2	1966	1720	607	8.5	4370
10	AMD Opteron875/2200	1311	1251	1497	8.6	4543

Core	DMIPS /MHz	Freq. (MHz)	DMIPS	Inline DMIPS/MHz	Inline DMIPS
4Kc™	1.3	300	390	1.6	480
4KEc™	1.35	300	405	1.8	540
5Kc™	1.4	350	490	2.0	700
5Kf™	1.4	320	448	2.0	640
20Kc™	1.7	600	1020	2.2	1320

Dhrystone Benchmark

MIPS TECHNOLOGIES

ECE568/Koren Part.1 .17

Copyright 2008 Koren UMass

How do we design faster CPUs?

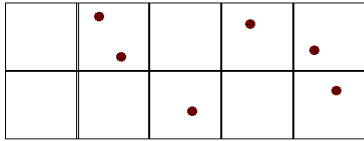
- ◆ **Faster technology** - used to be the main approach
 - (a) getting more expensive
 - (b) reliability & yield
 - (c) speed of light ($3 \cdot 10^8$ m/sec)
- ◆ **Larger dies (SOC - System On a Chip)**
 - less wires between ICs but - low yield (next slide)
- ◆ **Parallel processing** - use n independent processors
 - limited success
- ◆ **n -issue superscaler microprocessor (currently $n=4$)**
 - Can we expect a Speedup = n ?
- ◆ **Pipelining**
- ◆ **Multi-threading**

ECE568/Koren Part.1 .18

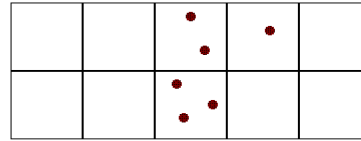
Copyright 2008 Koren UMass

Integrated Circuits Yield

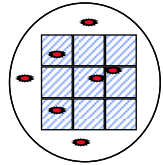
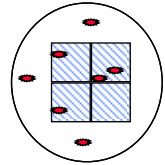
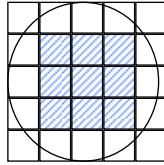
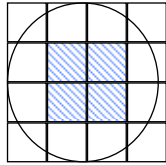
$$\text{Die Yield} = \text{Wafer_yield} \times \left\{ 1 + \left(\frac{\text{Defect_Density} \times \text{Die_area}}{\alpha} \right) \right\}^{-\alpha}$$



(a) Non-clustered faults,
Chip_Yield = 0.5



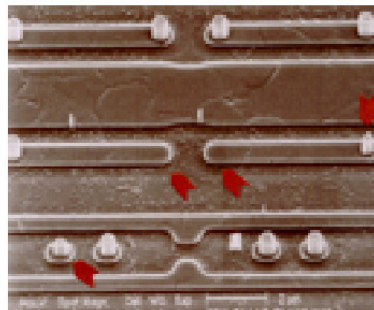
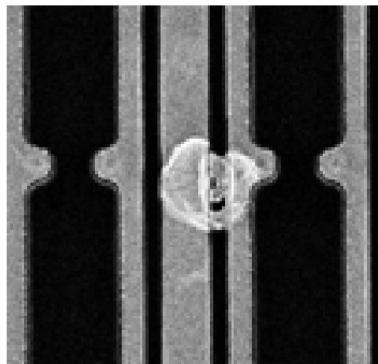
(b) Clustered faults,
Chip_Yield = 0.7



ECE568/Koren Part.1 .19

Copyright 2008 Koren UMass

Integrated Circuits Yield - Defects



ECE568/Koren Part.1 .20

Copyright 2008 Koren UMass

Integrated Circuits Costs

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

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} \times \text{Die Yield}}$$

$$\text{Dies per wafer} = \frac{\pi (\text{Wafer_diam}/2)^2}{\text{Die_Area}} - \frac{\pi \times \text{Wafer_diam}}{\sqrt{2} \times \text{Die_Area}} - \text{Test_Die}$$

Die Cost goes up roughly with (Die_Area)³

Amdahl's Law - Basics

Example: Executing a program on n independent processors

$\text{Fraction}_{\text{enhanced}}$ = parallelizable part of program

$\text{Speedup}_{\text{enhanced}} = n$

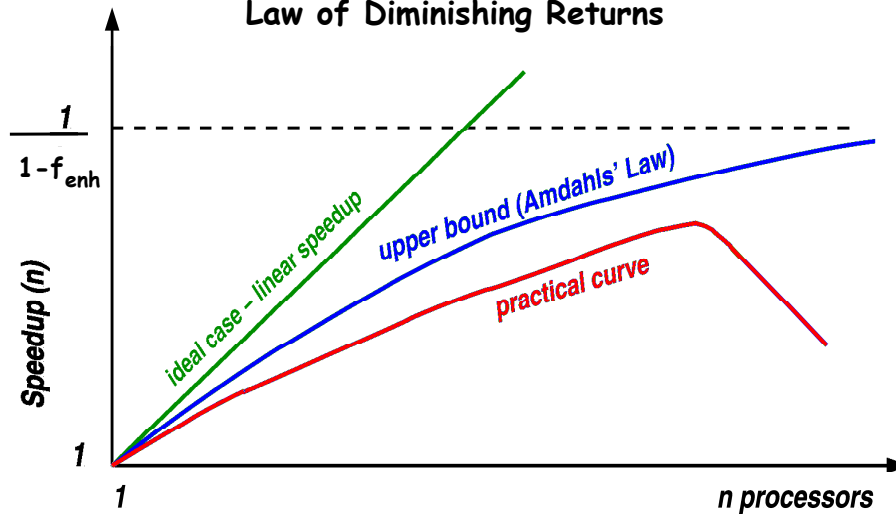
$$\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} (1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{ExTime}_{\text{old}} \times \text{Fraction}_{\text{enhanced}}}{n}$$

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

$$\lim_{n \rightarrow \infty} \text{Speedup}_{\text{overall}} = 1 / (1 - \text{Fraction}_{\text{enhanced}})$$

Amdahl's Law - Graph

Law of Diminishing Returns



ECE568/Koren Part.1 .23

Copyright 2008 Koren UMass

Amdahl's Law - Extension

- ◆ Example: Improving part of a processor (e.g., multiplier, floating-point unit)

Fraction_{enhanced} = part of program to be enhanced

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

$$< 1 / (1 - \text{Fraction}_{\text{enhanced}})$$

A given signal processing application consists of 40% multiplications.

An enhanced multiplier will execute 5 times faster

$$\text{Speedup}_{\text{overall}} = 1 / (\quad + \quad) = 1.47 < 1/0.6 = 1.66$$

ECE568/Koren Part.1 .24

Copyright 2008 Koren UMass

Instruction execution

◆ **Components of average execution time (CPI Law)**

- Average CPU time per program

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



- ◆ The **"End to End Argument"** is what RISC was ultimately about - it is the performance of the complete system that matters, not individual components!

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			
Compiler			
Inst. Set.			
Organization			
Technology			

Cycles Per Instruction

“Average Cycles per Instruction”

$$\text{CPI} = \text{Total_No_of_Cycles} / \text{Instruction Count}$$

“CPI of Individual Instructions”

CPI_j - CPI for instruction j (j=1,...,n)

I_j - # of times instruction j is executed

$$\text{CPU time} = \text{Cycle Time} \times \sum_{j=1}^n \text{CPI}_j \times I_j$$

“Instruction Frequency”

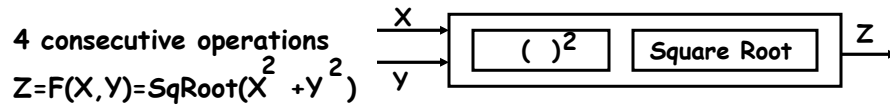
$$\text{CPI} = \sum_{j=1}^n \text{CPI}_j \times F_j \quad \text{where } F_j = \frac{I_j}{\text{Instruction Count}}$$

Example: Calculating CPI

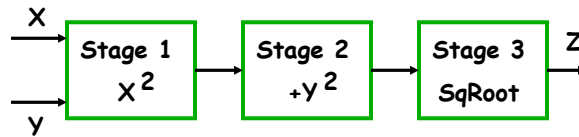
Base Machine (Reg / Reg)				
Op	Freq	Cycles	CPI _j * F _j	(% Time)
ALU	50%	1	.5	()
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)

Typical Mix of instruction types in program

Pipelining - Basics



If each step takes $1T$ then one calculation takes $3T$, four take $12T$

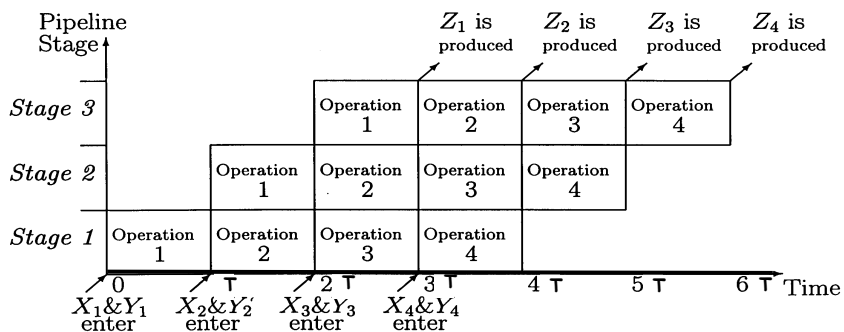


Assuming ideally that each stage takes $1T$

What will be the **latency** (time to produce the first result)?

What will be the **throughput** (pipeline rate in the steady state)?

Pipelining - Timing



Total of $6T$; Speedup = ?

For n operations: $3T + (n-1)T = \text{latency} + \frac{n-1}{\text{throughput}}$

$$\text{Speedup} = \frac{n \cdot 3T}{3T + (n-1)T} = \frac{3n}{n+2} \xrightarrow{n \rightarrow \infty} \boxed{} \text{ \# of stages}$$

Pipelining - Non ideal

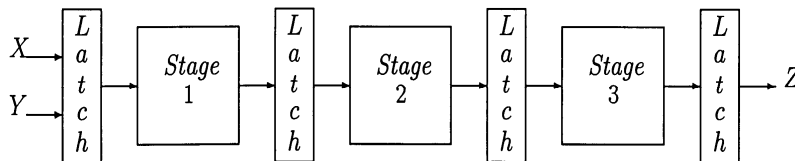
Non-ideal situation:

1. Steps take T_1, T_2, T_3 Rate = $1 / \max T_i$

Slowest unit determines the throughput

2. To allow independent operation must add latches

$$\tau = \tau_{\text{latch}} + \max T_i$$

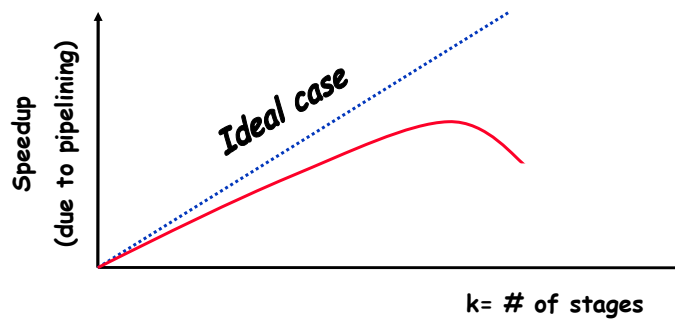


ECE568/Koren Part.1 .31

Copyright 2008 Koren UMass

k-stage Pipeline - Speedup

$$\text{Speedup} = \frac{n k T}{k \tau + (n-1) \tau} = \frac{T n k}{\tau (k+n-1)} \xrightarrow{n \rightarrow \infty} (T/\tau) k$$

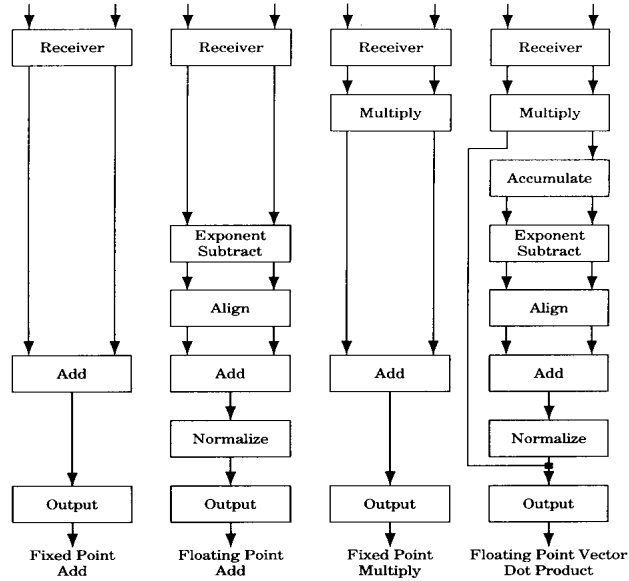


ECE568/Koren Part.1 .32

Copyright 2008 Koren UMass

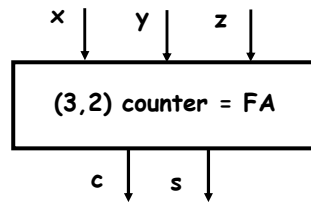
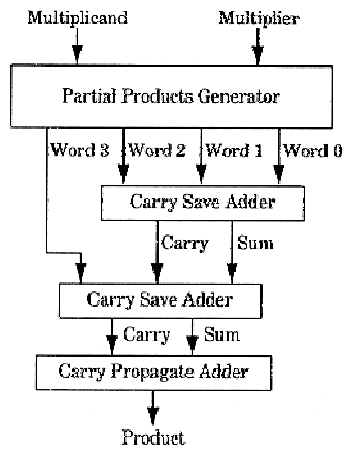
Arithmetic Pipelines

TI - ASC



ECE568/Koren Part.1 .33

Multiplier



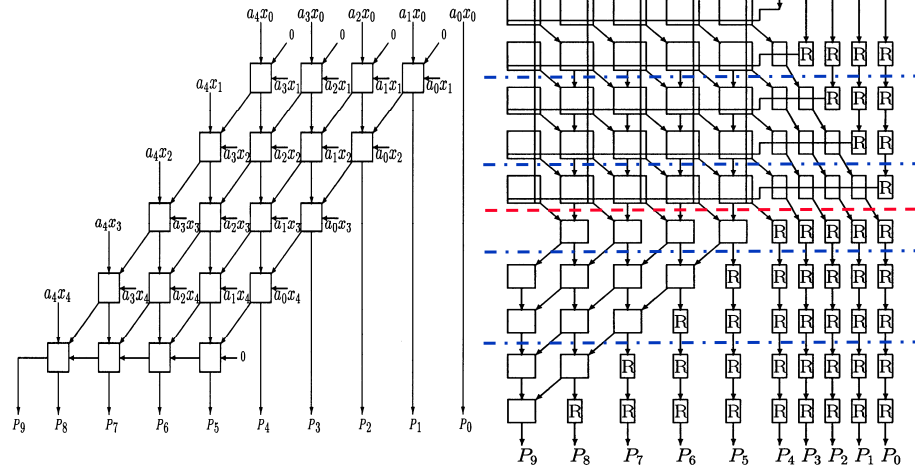
	a_4	a_3	a_2	a_1	a_0	
\times	x_4	x_3	x_2	x_1	x_0	
	$a_4 \cdot x_0$	$a_3 \cdot x_0$	$a_2 \cdot x_0$	$a_1 \cdot x_0$	$a_0 \cdot x_0$	
	$a_4 \cdot x_1$	$a_3 \cdot x_1$	$a_2 \cdot x_1$	$a_1 \cdot x_1$	$a_0 \cdot x_1$	
	$a_4 \cdot x_2$	$a_3 \cdot x_2$	$a_2 \cdot x_2$	$a_1 \cdot x_2$	$a_0 \cdot x_2$	
	$a_4 \cdot x_3$	$a_3 \cdot x_3$	$a_2 \cdot x_3$	$a_1 \cdot x_3$	$a_0 \cdot x_3$	
	$a_4 \cdot x_4$	$a_3 \cdot x_4$	$a_2 \cdot x_4$	$a_1 \cdot x_4$	$a_0 \cdot x_4$	
	P_9	P_8	P_7	P_6	P_5	P_4
						P_3
						P_2
						P_1
						P_0

ECE568/Koren Part.1 .34

Copyright 2008 Koren UMass

Pipelined Multiplier

Source: "Computer Arithmetic Algorithms," I. Koren, 2002.



ECE568/Koren Part.1 .35

Copyright 2008 Koren UMass