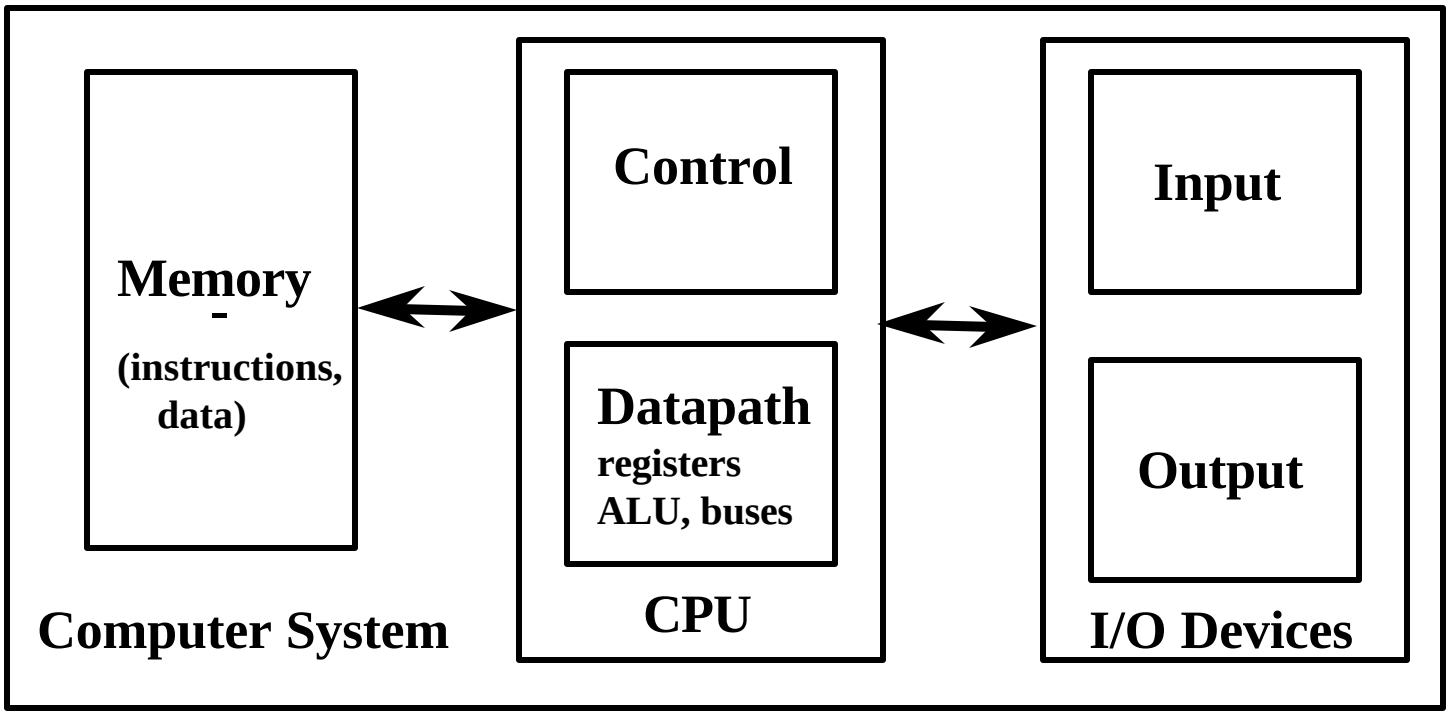
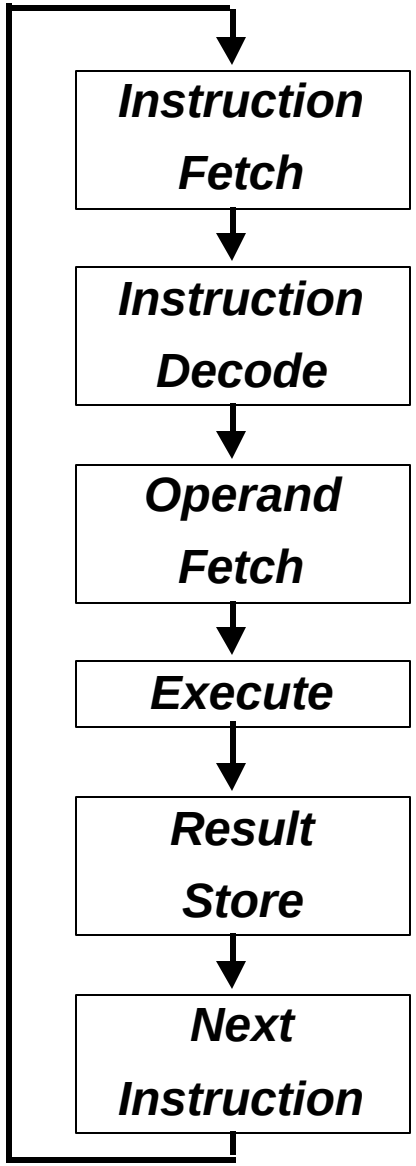
The Von Neumann Computer Model

- **Partitioning of the computing engine into components:**
  - **Central Processing Unit (CPU): Control Unit (instruction decode , sequencing of operations), Datapath (registers, arithmetic and logic unit, buses).**
  - **Memory: Instruction and operand storage.**
  - **Input/Output (I/O) sub-system: I/O bus, interfaces, devices.**
  - **The stored program concept: Instructions from an instruction set are fetched from a common memory and executed one at a time**



Generic CPU Machine Instruction Execution Steps



Obtain instruction from program storage

Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage for later use

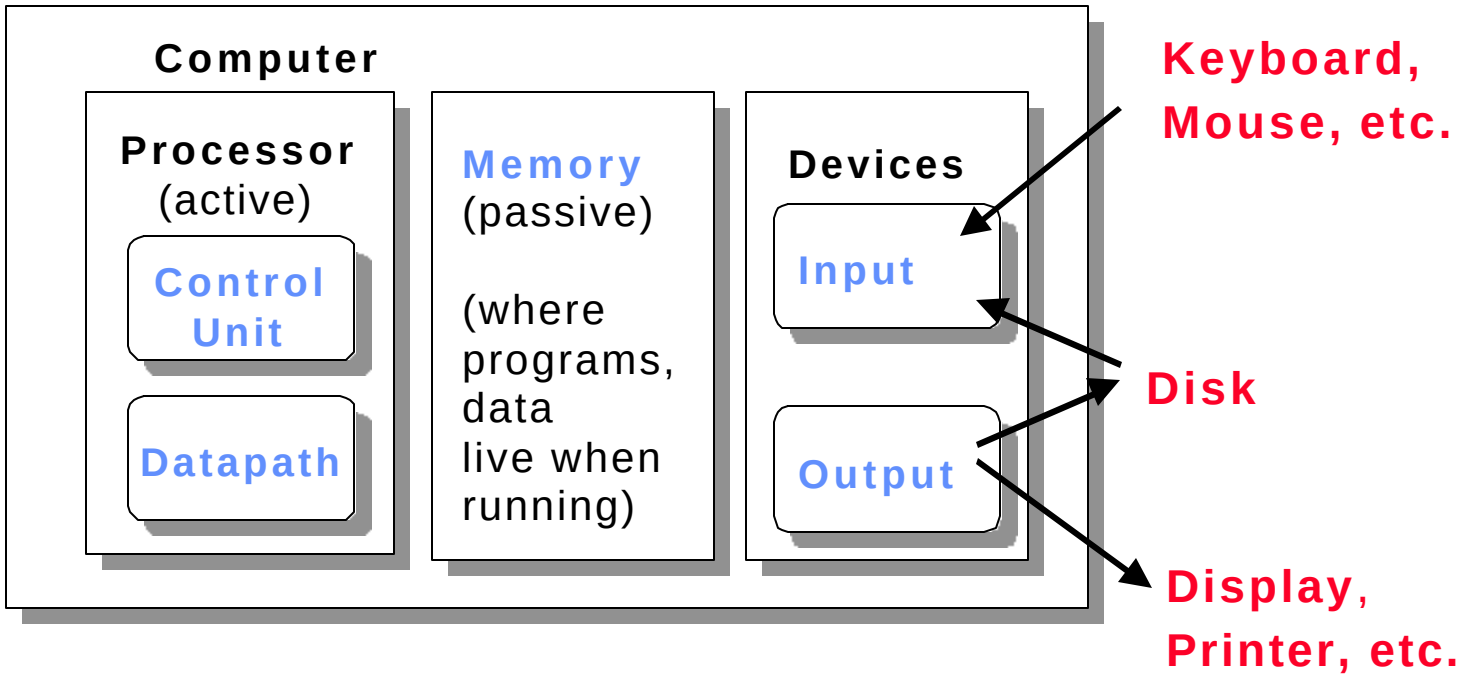
Determine successor or next instruction

Hardware Components of Any Computer

Five classic components of all computers:

1. Control Unit; 2. Datapath; 3. Memory; 4. Input; 5. Output

Processor



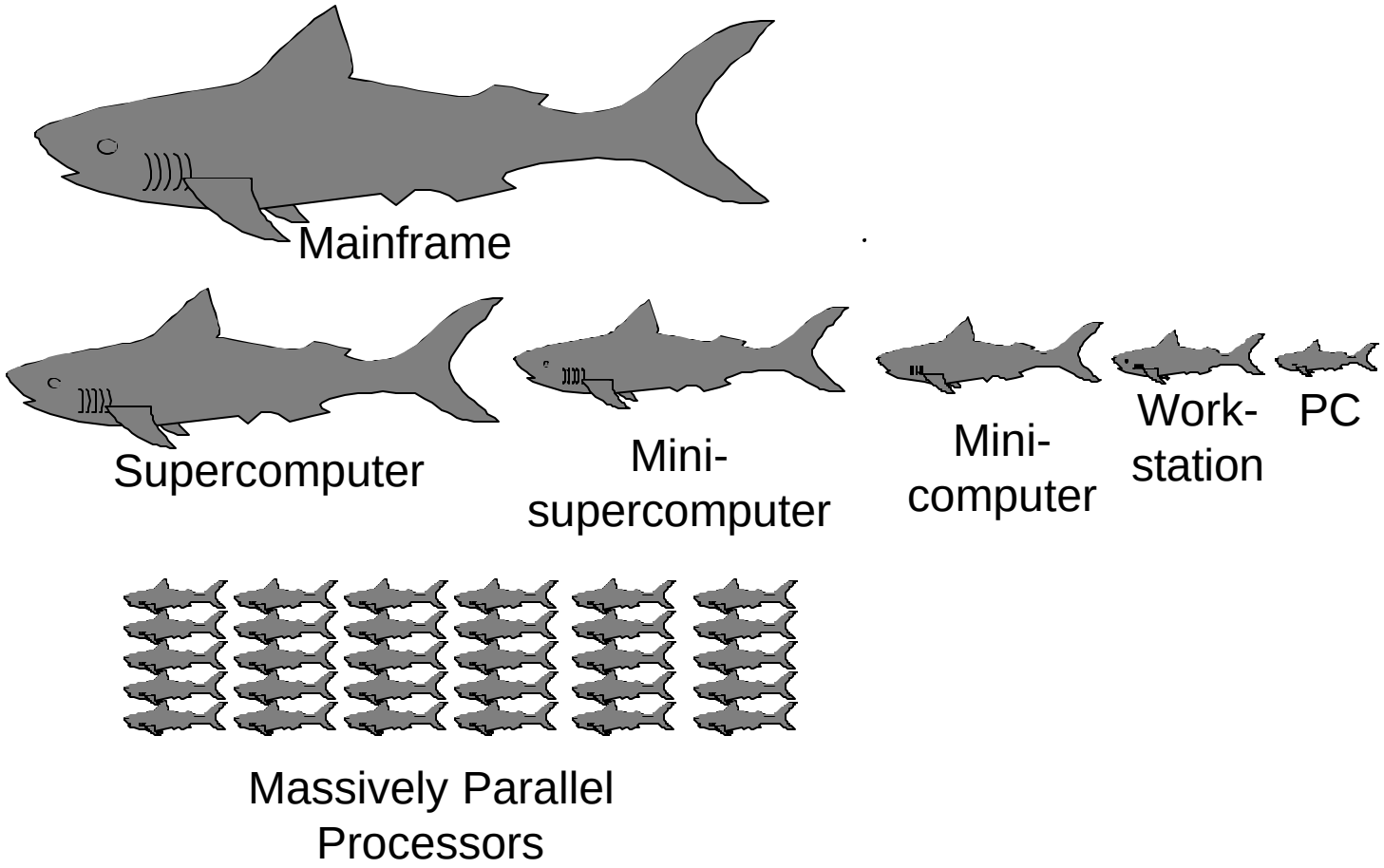
CPU Organization

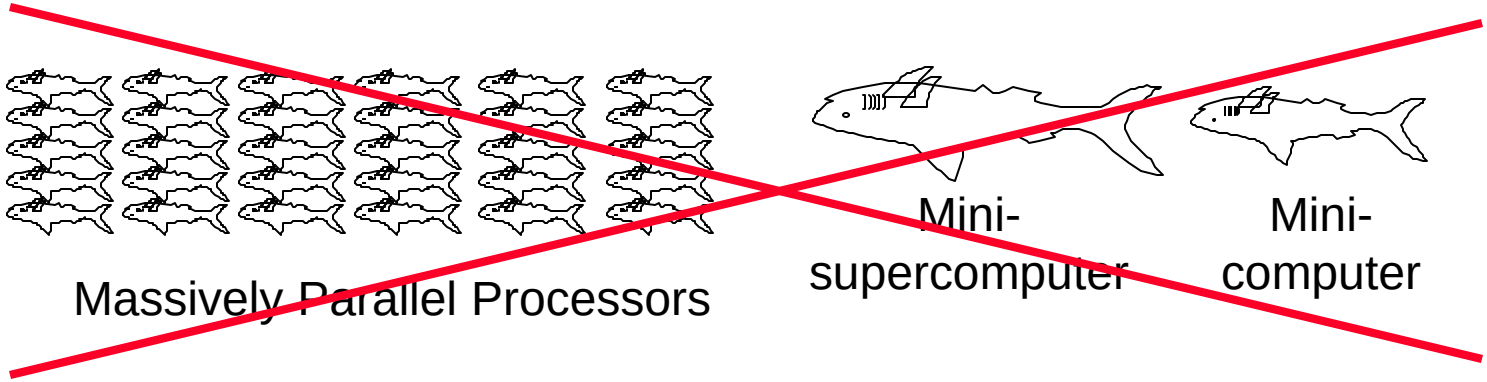
- **Datapath Design:**
  - **Capabilities & performance characteristics of principal Functional Units (FUs):**
    - **(e.g., Registers, ALU, Shifters, Logic Units, ...)**
  - **Ways in which these components are interconnected (buses connections, multiplexors, etc.).**
  - **How information flows between components.**
- **Control Unit Design:**
  - **Logic and means by which such information flow is controlled.**
  - **Control and coordination of FUs operation to realize the targeted Instruction Set Architecture to be implemented (can either be implemented using a finite state machine or a microprogram).**
- **Hardware description with a suitable language, possibly using Register Transfer Notation (RTN).**

Recent Trends in Computer Design

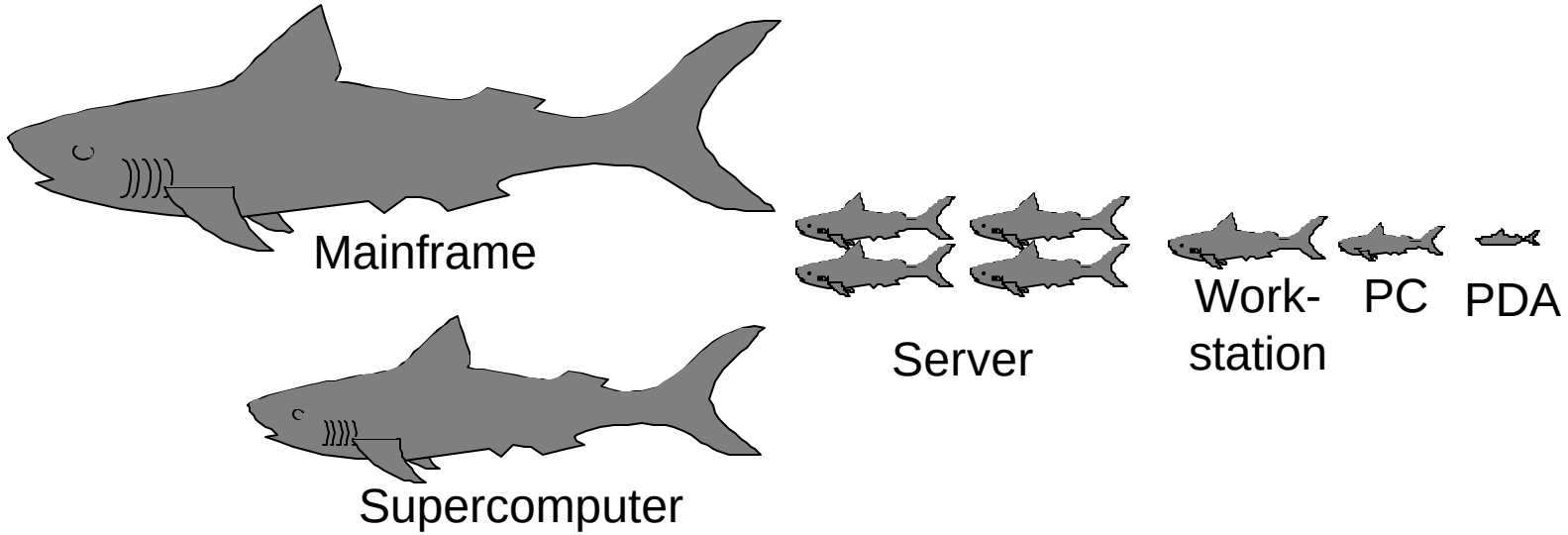
- The cost/performance ratio of computing systems have seen a steady decline due to advances in:
  - Integrated circuit technology: *decreasing feature size,* λ
    - Clock rate improves roughly proportional to improvement in λ
    - Number of transistors improves proportional to λ^2 (or faster).
  - Architectural improvements in CPU design.
- Microprocessor systems directly reflect IC improvement in terms of a yearly 35 to 55% improvement in performance.
- Assembly language has been mostly eliminated and replaced by other alternatives such as C or C++
- Standard operating Systems (UNIX, NT) lowered the cost of introducing new architectures.
- Emergence of RISC architectures and RISC-core architectures.
- Adoption of quantitative approaches to computer design based on empirical performance observations.

1988 Computer Food Chain



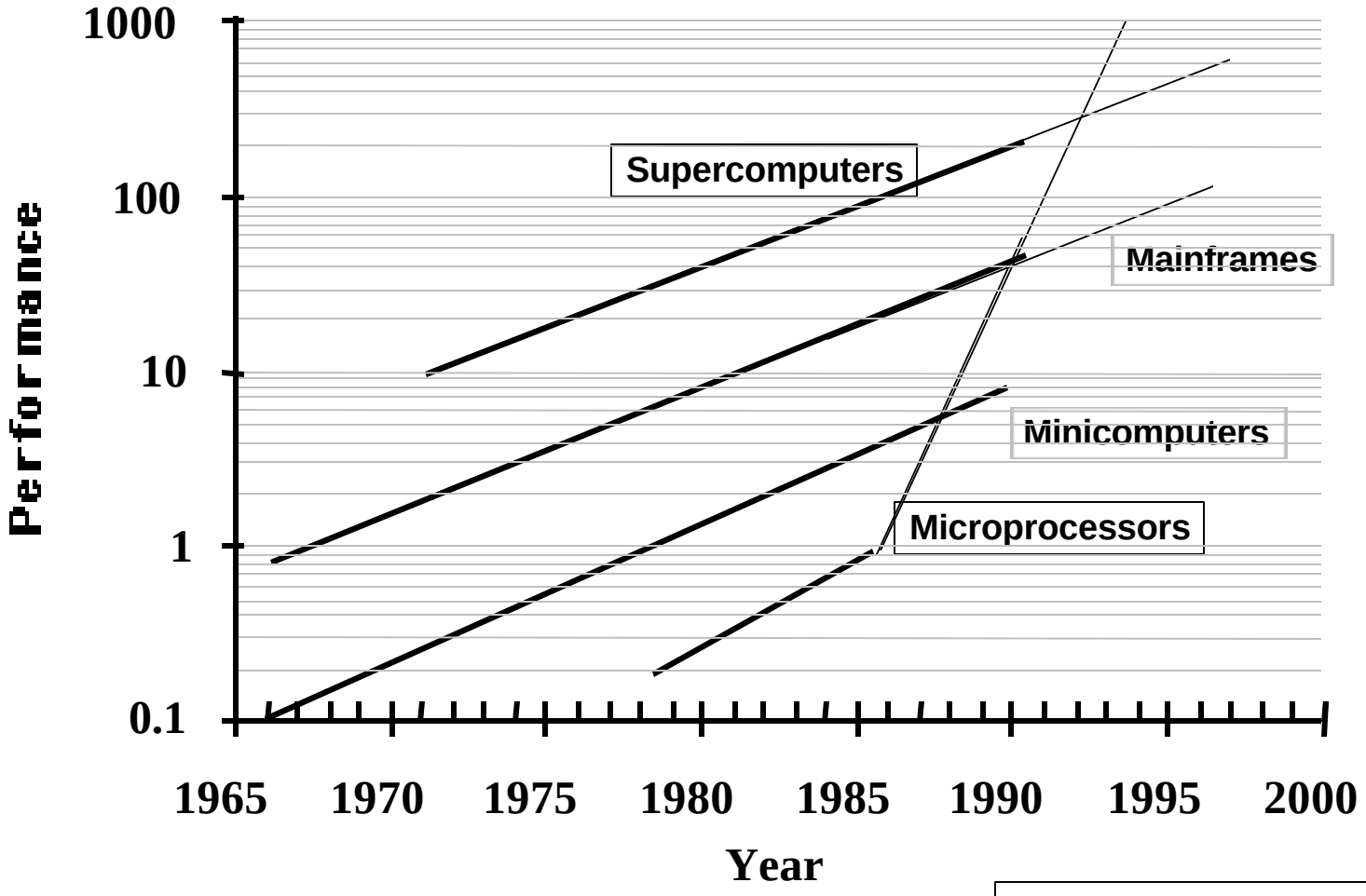


1997 Computer Food Chain

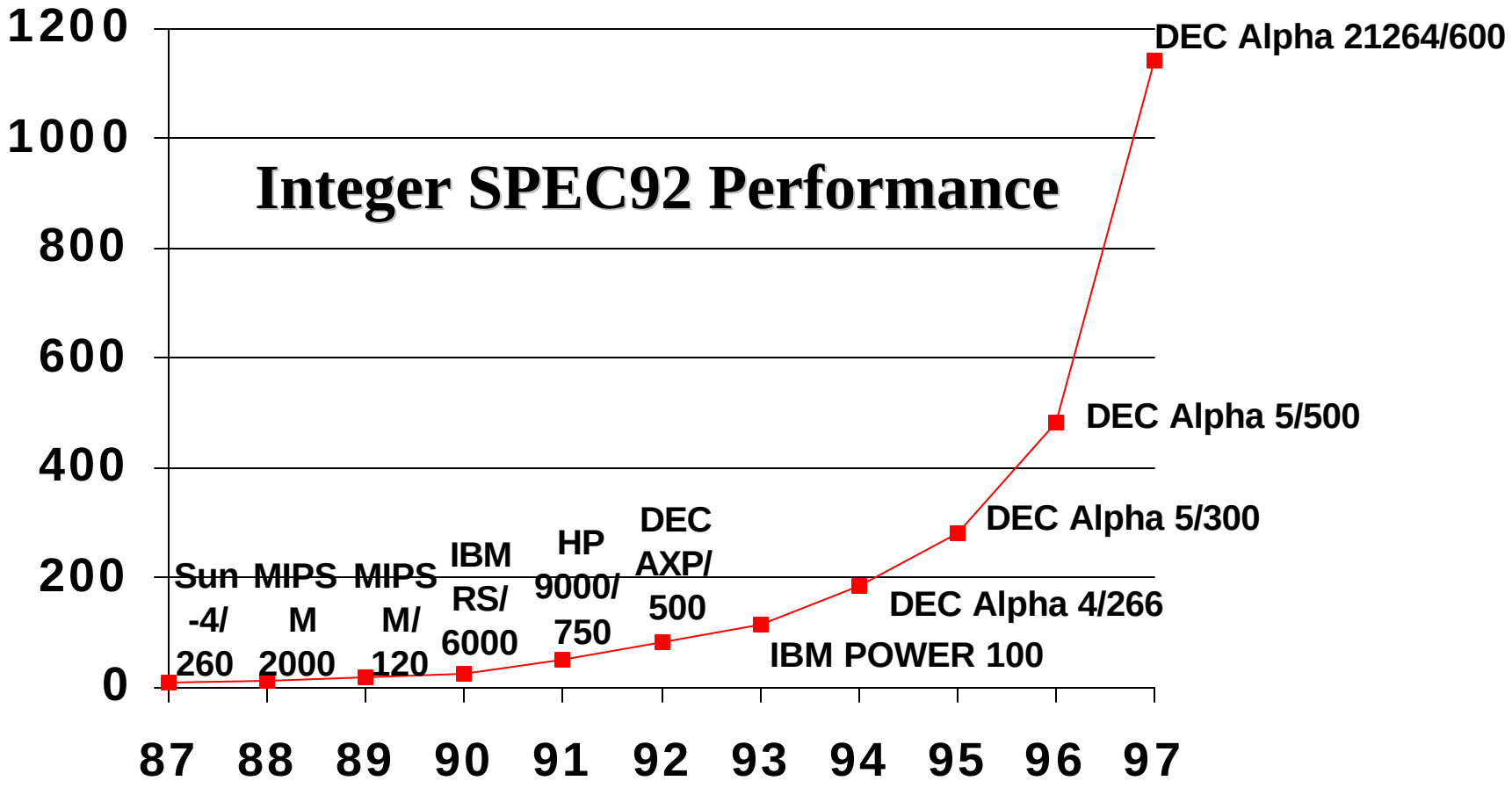


Processor Performance Trends

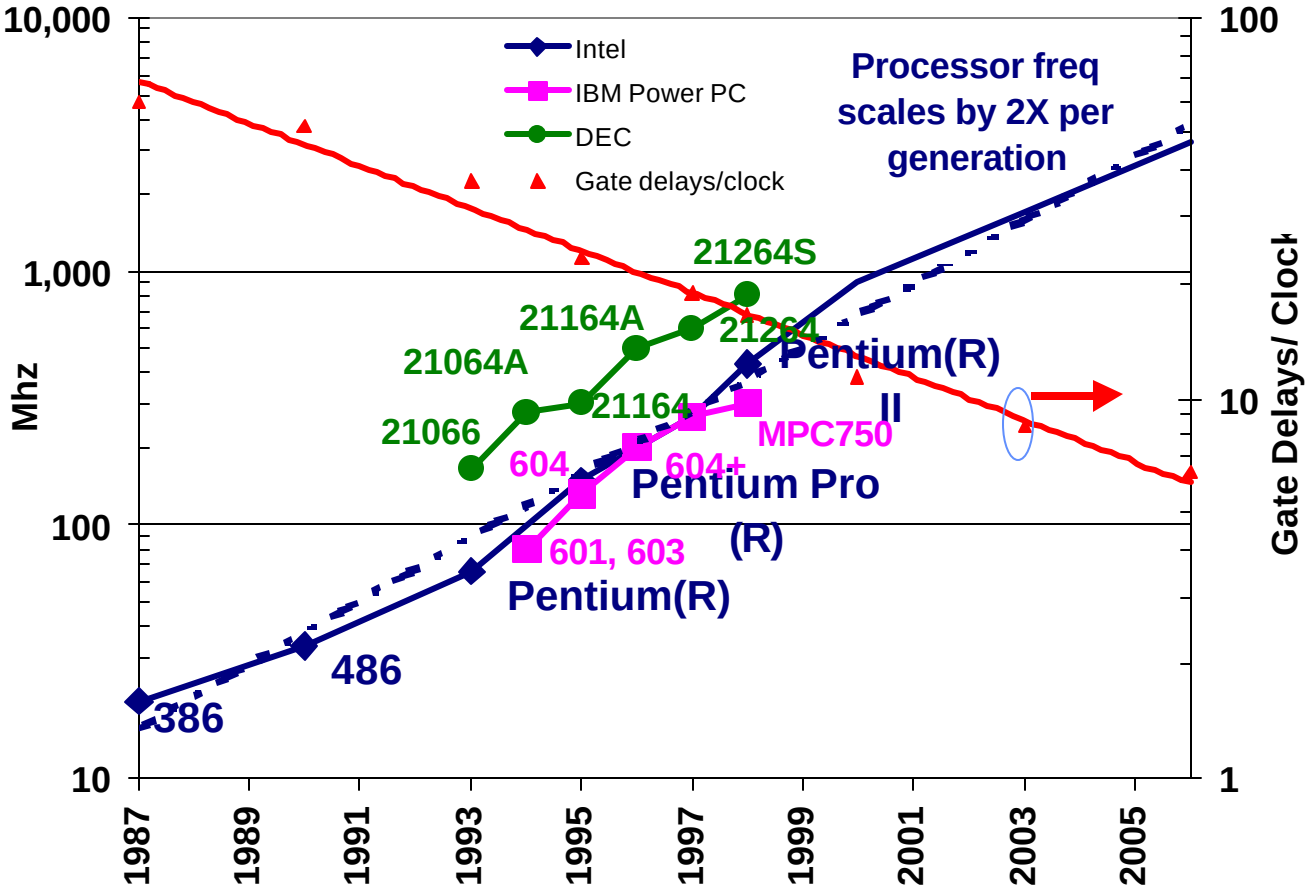
Mass-produced microprocessors a cost-effective high-performance replacement for custom-designed mainframe/minicomputer CPUs



Microprocessor Performance 1987-97

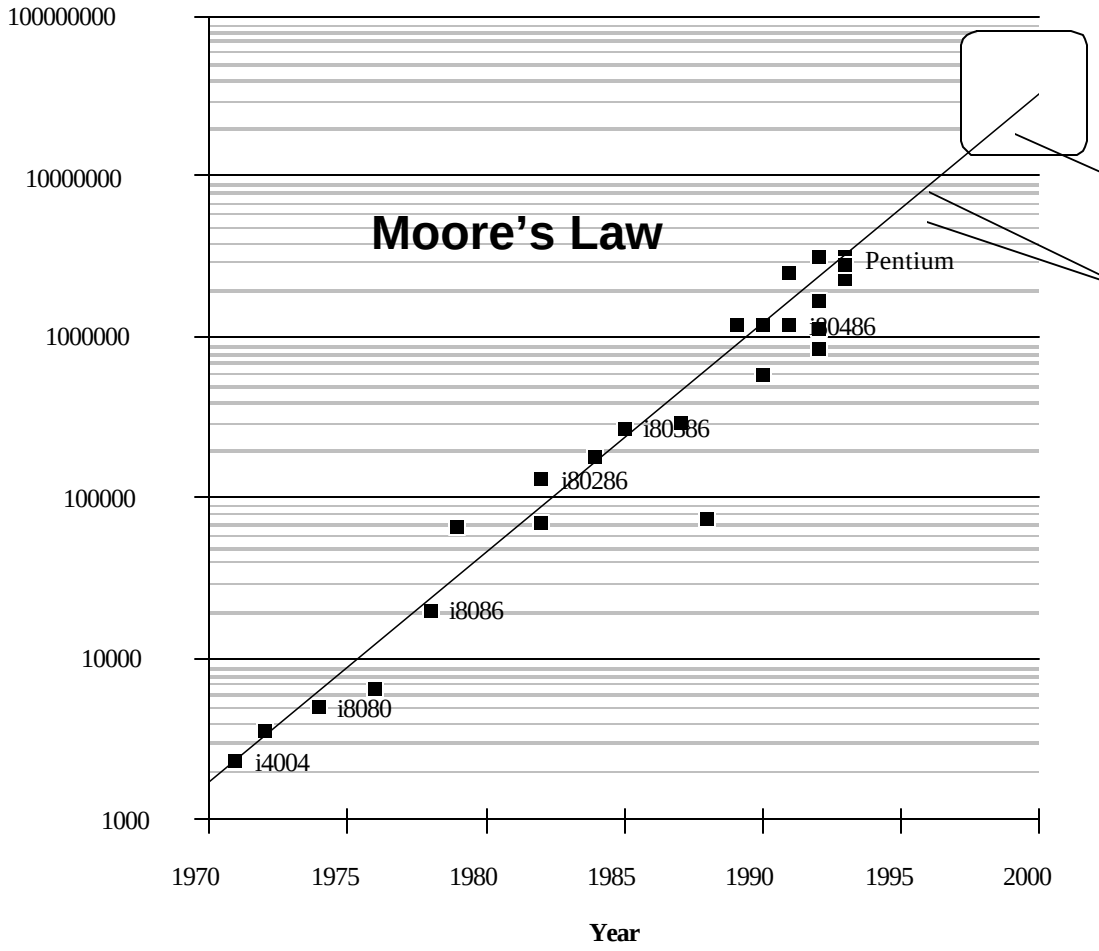


Microprocessor Frequency Trend



❶ Frequency doubles each generation
❷ Number of gates/clock reduce by 25%

Microprocessor Transistor Count Growth Rate

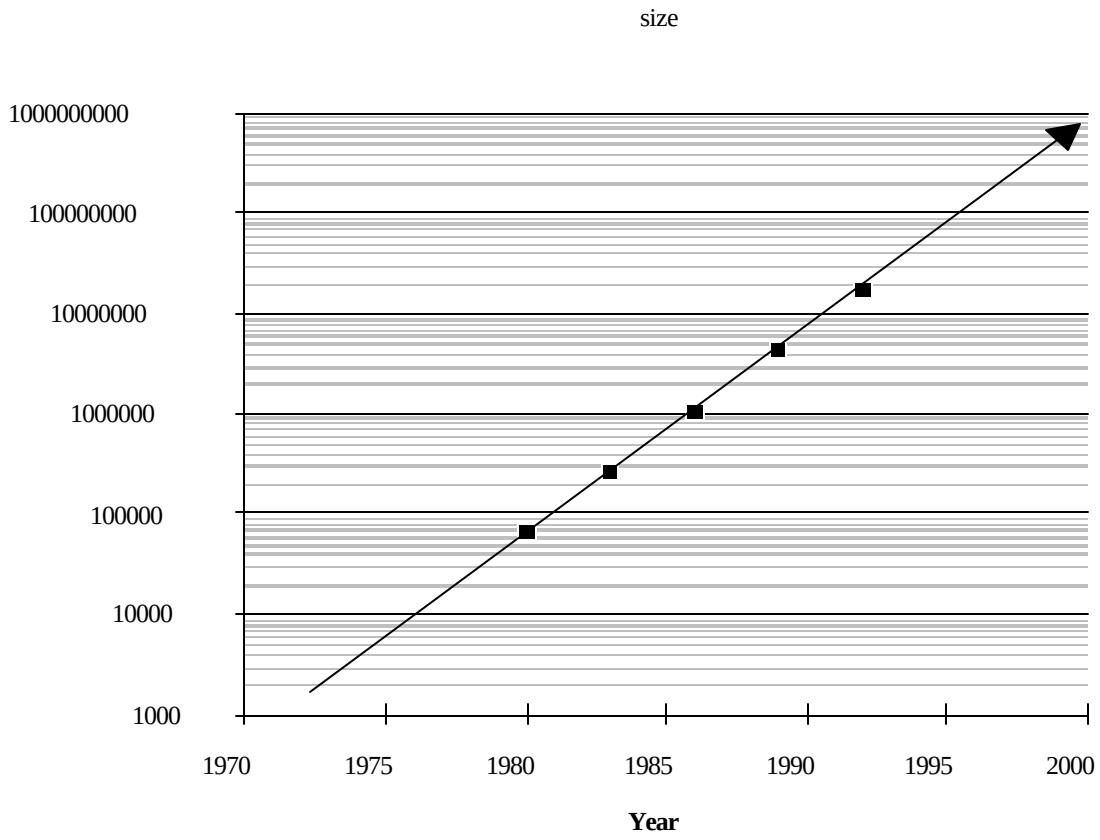


Alpha 21264: 15 million
Pentium Pro: 5.5 million
PowerPC 620: 6.9 million
Alpha 21164: 9.3 million
Sparc Ultra: 5.2 million

Moore's Law:

2X transistors/Chip
Every 1.5 years

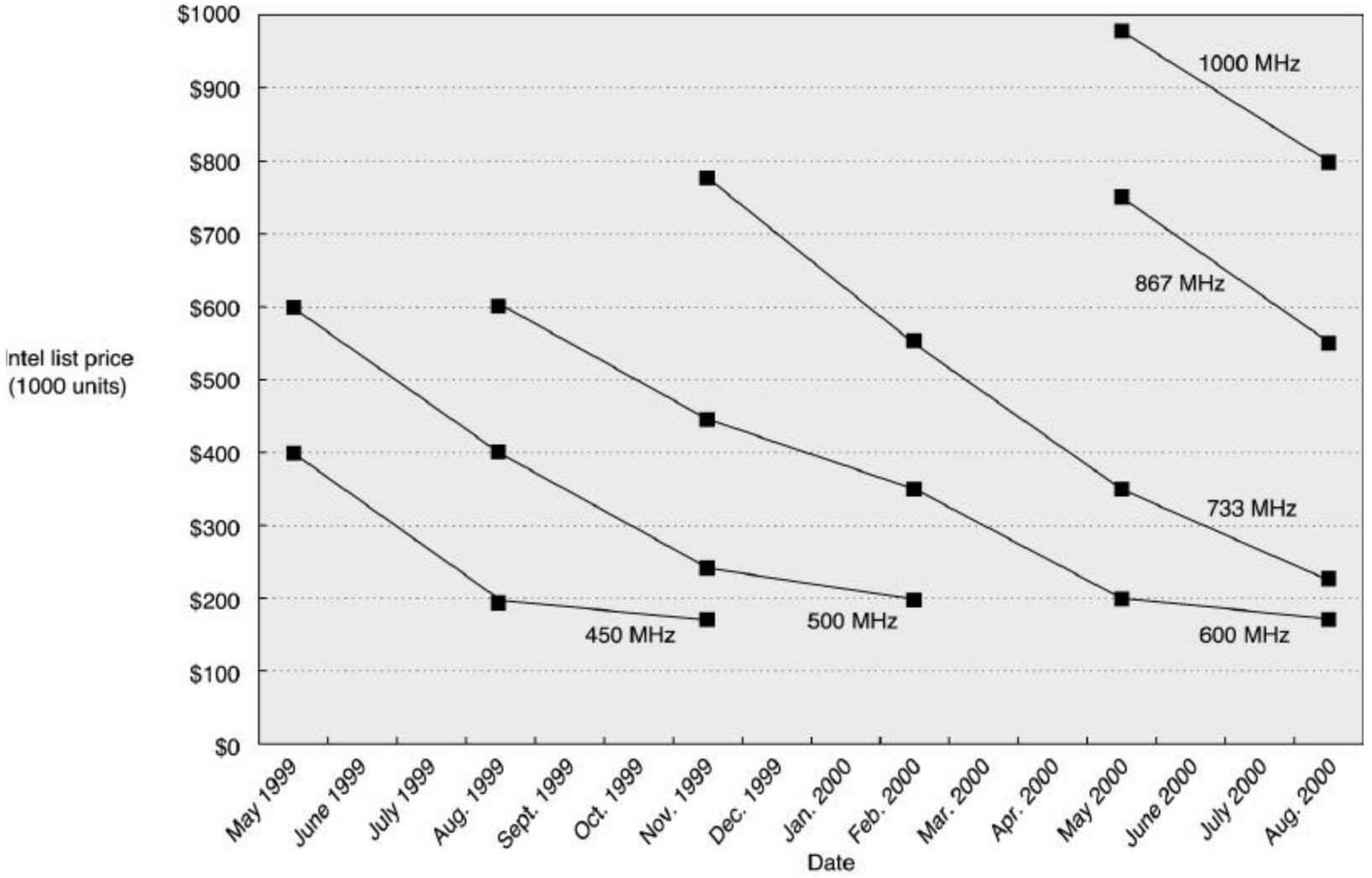
Increase of Capacity of VLSI Dynamic RAM Chips



year	size(Megabit)
1980	0.0625
1983	0.25
1986	1
1989	4
1992	16
1996	64
1999	256
2000	1024

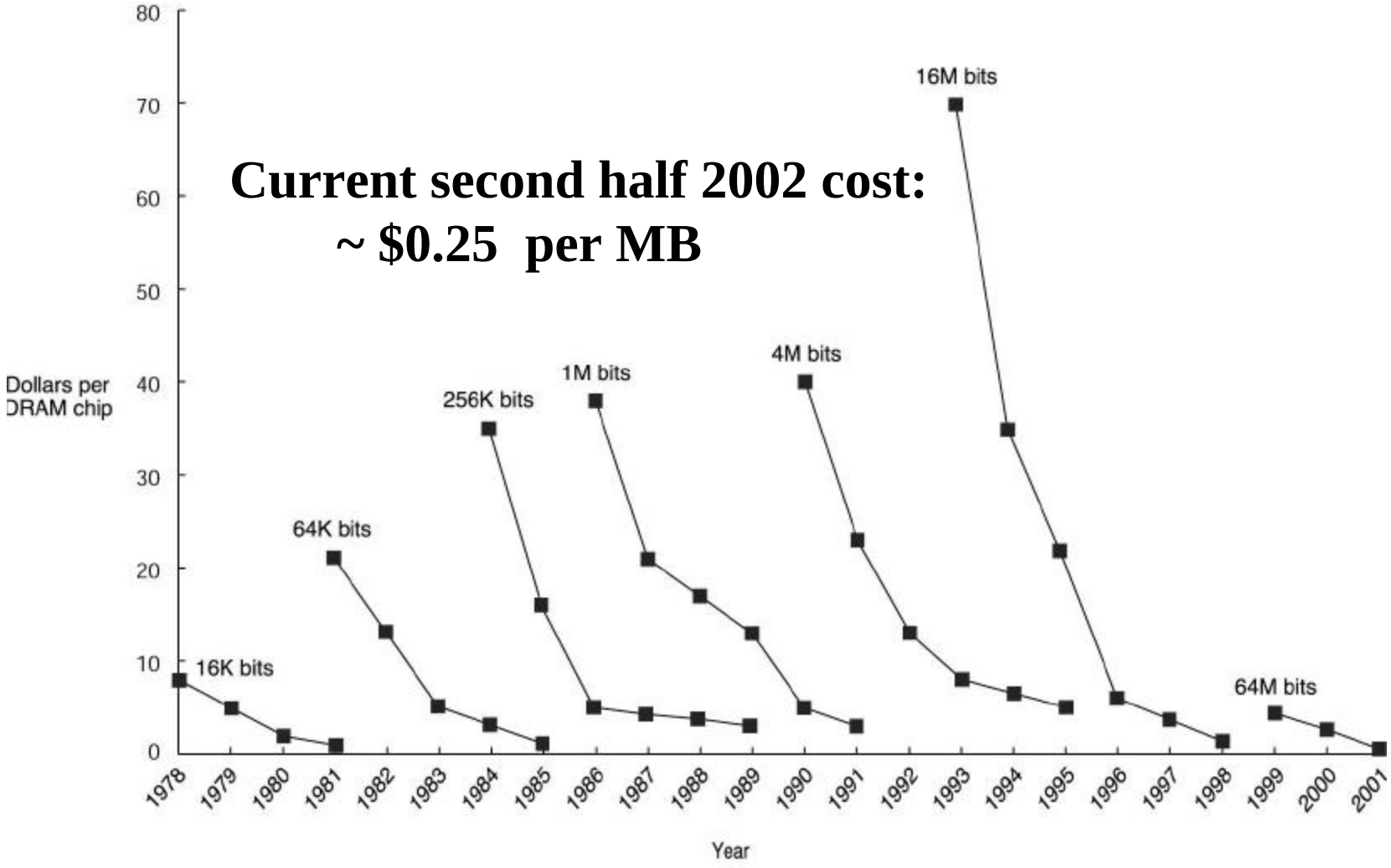
1.55X/yr,
or doubling every 1.6 years

Microprocessor Cost Drop Over Time
Example: Intel PIII



DRAM Cost Over Time

Current second half 2002 cost:
~ $0.25 per MB



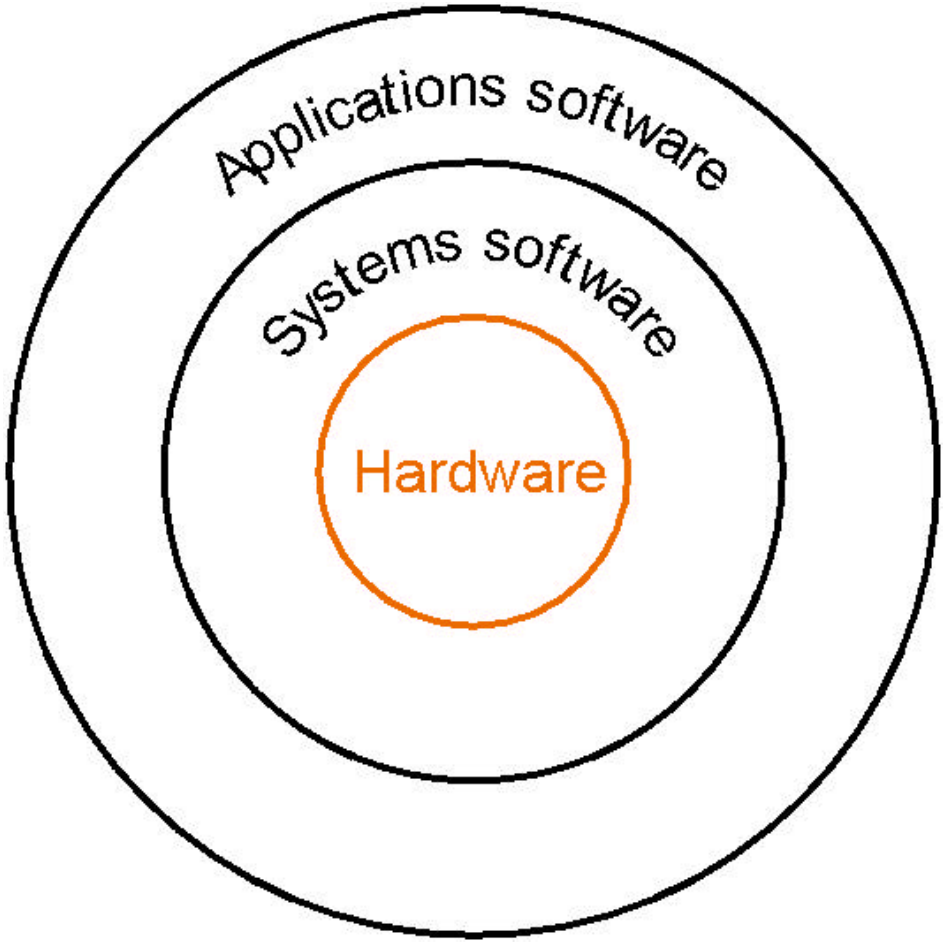
Recent Technology Trends (Summary)

	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

Computer Technology Trends: *Evolutionary but Rapid Change*

- **Processor:**
  - **2X in speed every 1.5 years; 100X performance in last decade.**
- **Memory:**
  - **DRAM capacity: > 2x every 1.5 years; 1000X size in last decade.**
  - **Cost per bit: Improves about 25% per year.**
- **Disk:**
  - **Capacity: > 2X in size every 1.5 years.**
  - **Cost per bit: Improves about 60% per year.**
  - **200X size in last decade.**
  - **Only 10% performance improvement per year, due to mechanical limitations.**
- **Expected State-of-the-art PC by end of year 2003 :**
  - **Processor clock speed: > 3400 MegaHertz (3.4 GigaHertz)**
  - **Memory capacity: > 4000 MegaByte (4 GigaBytes)**
  - **Disk capacity: > 300 GigaBytes (0.3 TeraBytes)**

A Simplified View of The Software/Hardware Hierarchical Layers



A Hierarchy of Computer Design

Level	Name	Modules	Primitives	Descriptive Media
1	Electronics	Gates, FF's	Transistors, Resistors, etc.	Circuit Diagrams
2	Logic	Registers, ALU's ...	Gates, FF's ….	Logic Diagrams
3	Organization	Processors, Memories	Registers, ALU's …	Register Transfer Notation (RTN)

Low Level - Hardware

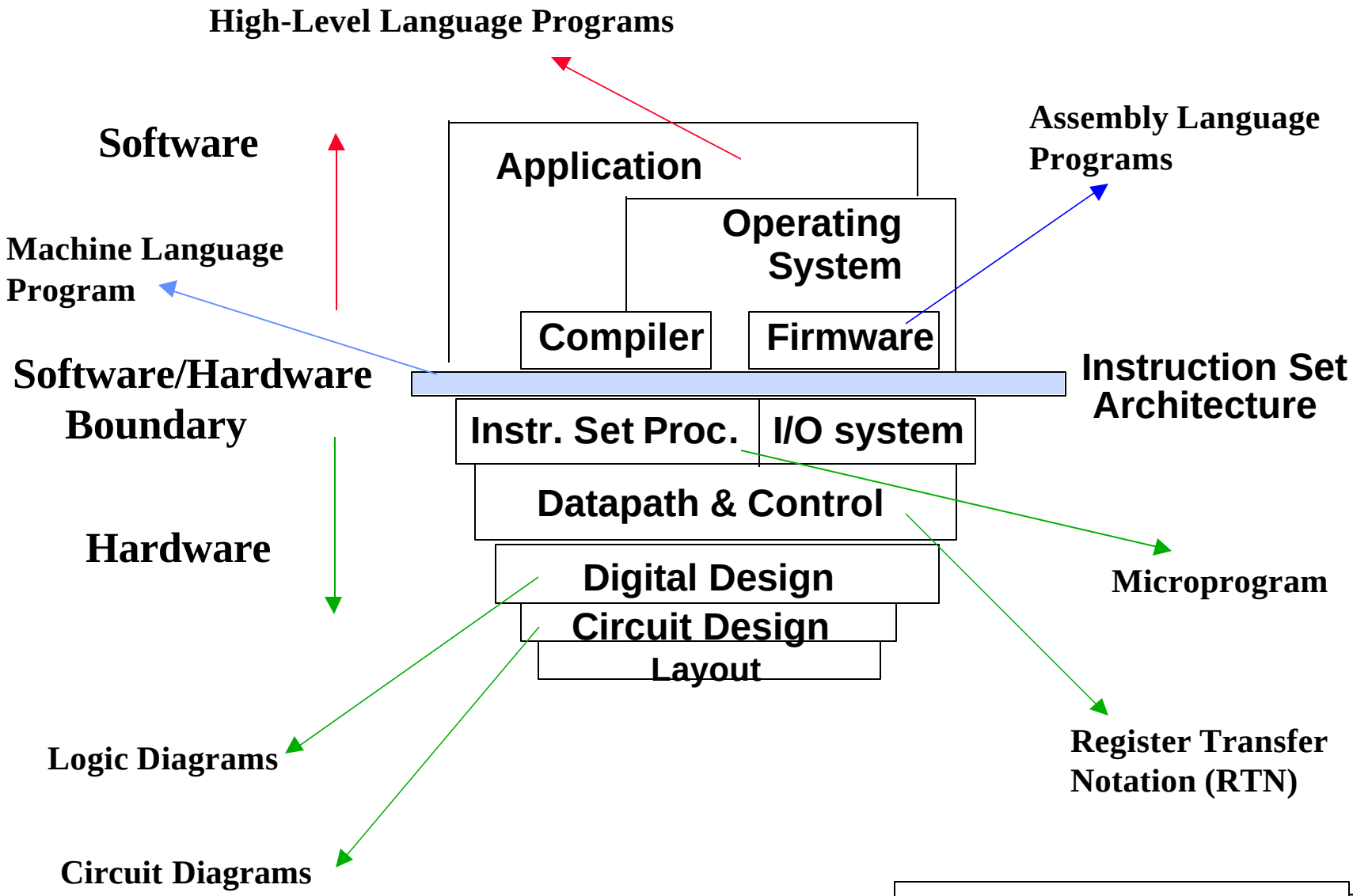
4	Microprogramming	Assembly Language	Microinstructions	Microprogram

Firmware

5	Assembly language programming	OS Routines	Assembly language Instructions	Assembly Language Programs
6	Procedural Programming	Applications Drivers ..	OS Routines High-level Languages	High-level Language Programs
7	Application	Systems	Procedural Constructs	Problem-Oriented Programs

High Level - Software

Hierarchy of Computer Architecture



Computer Architecture Vs. Computer Organization

- The term Computer architecture is sometimes erroneously restricted to computer instruction set design, with other aspects of computer design called implementation
- More accurate definitions:
  - Instruction set architecture (ISA): The actual programmer-visible instruction set and serves as the boundary between the software and hardware.
  - Implementation of a machine has two components:
    - Organization: includes the high-level aspects of a computer's design such as: The memory system, the bus structure, the internal CPU unit which includes implementations of arithmetic, logic, branching, and data transfer operations.
    - Hardware: Refers to the specifics of the machine such as detailed logic design and packaging technology.
- In general, Computer Architecture refers to the above three aspects:

  Instruction set architecture, organization, and hardware.

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.

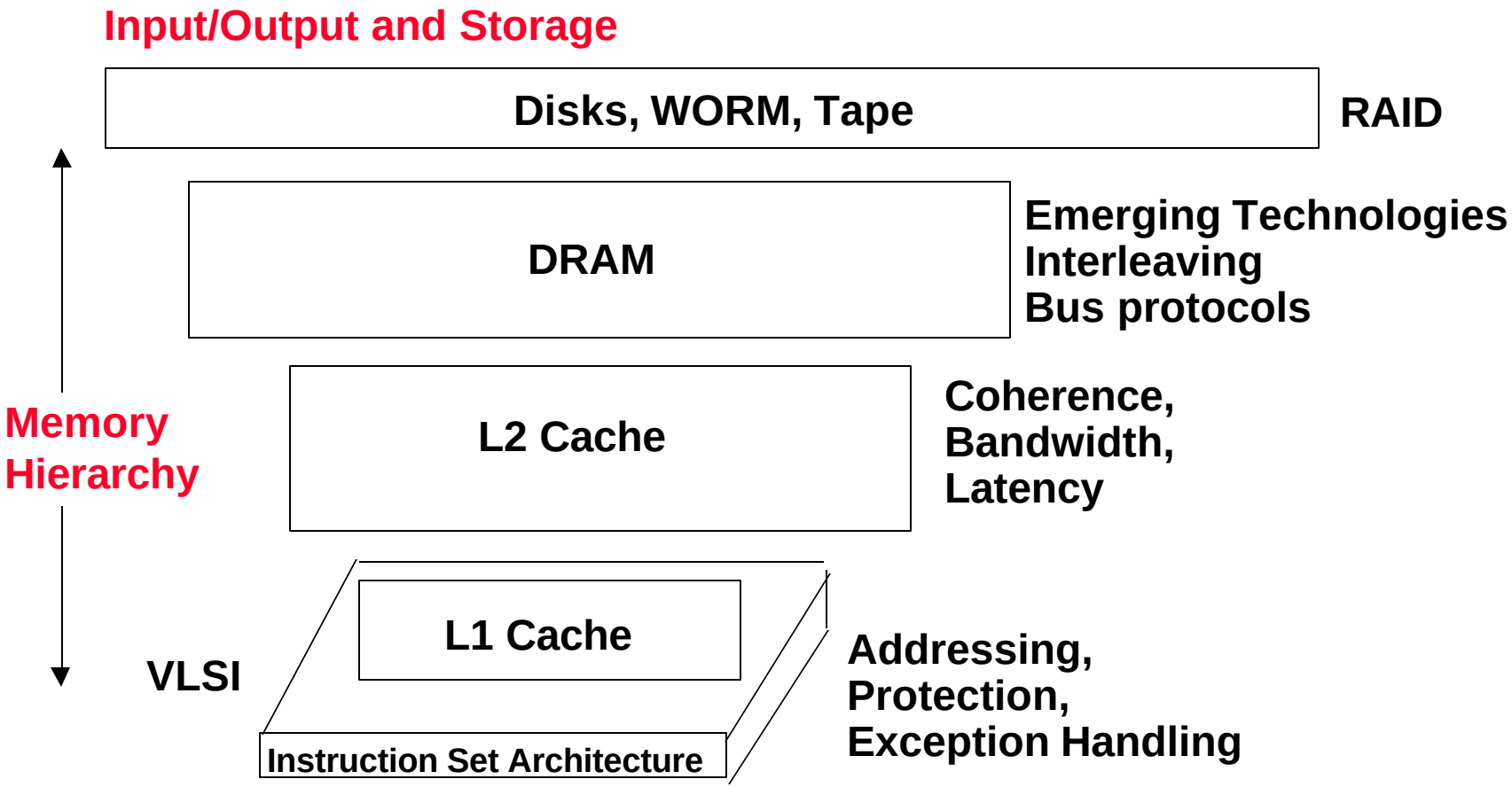
The Task of A Computer Designer

- Determine what attributes that are important to the design of the new machine.
- Design a machine to maximize performance while staying within cost and other constraints and metrics.
- It involves more than instruction set design.
  - Instruction set architecture.
  - CPU Micro-Architecture.
  - Implementation.
- Implementation of a machine has two components:
  - Organization.
  - Hardware.

Recent Architectural Improvements

- Increased optimization and utilization of cache systems.
- Memory-latency hiding techniques.
- Optimization of pipelined instruction execution.
- Dynamic hardware-based pipeline scheduling.
- Improved handling of pipeline hazards.
- Improved hardware branch prediction techniques.
- Exploiting Instruction-Level Parallelism (ILP) in terms of multiple-instruction issue and multiple hardware functional units.
- Inclusion of special instructions to handle multimedia applications.
- High-speed bus designs to improve data transfer rates.

Current Computer Architecture Topics



Computer Performance Evaluation: Cycles Per Instruction (CPI)

- **Most computers run synchronously utilizing a CPU clock running at a constant clock rate:**

  where: Clock rate = 1 / clock cycle

- **A computer machine instruction is comprised of a number of elementary or micro operations which vary in number and complexity depending on the instruction and the exact CPU organization and implementation.**
  - **A micro operation is an elementary hardware operation that can be performed during one clock cycle.**
  - **This corresponds to one micro-instruction in microprogrammed CPUs.**
  - **Examples: register operations: shift, load, clear, increment, ALU operations: add , subtract, etc.**

- **Thus a single machine instruction may take one or more cycles to complete termed as the Cycles Per Instruction (CPI).**

Computer Performance Measures: Program Execution Time

- For a specific program compiled to run on a specific machine "A", the following parameters are provided:
  - The total instruction count of the program.
  - The average number of cycles per instruction (average CPI).
  - Clock cycle of machine "A"
- How can one measure the performance of this machine running this program?
  - Intuitively the machine is said to be faster or has better performance running this program if the total execution time is shorter.
  - Thus the inverse of the total measured program execution time is a possible performance measure or metric:

$$\text{Performance}_A = 1 \;/\; \text{Execution Time}_A$$

How to compare performance of different machines?

What factors affect performance? How to improve performance?

Measuring Performance

- For a specific program or benchmark running on machine x:

  $$\text{Performance} = 1 / \text{Execution Time}_x$$

- To compare the performance of machines X, Y, executing specific code:

  $$n = \text{Execution}_y / \text{Execution}_x$$
  $$= \text{Performance}_x / \text{Performance}_y$$

- System performance refers to the performance and elapsed time measured on an unloaded machine.
- CPU Performance refers to user CPU time on an unloaded system.
- Example:

For a given program:
Execution time on machine A: $\text{Execution}_A = 1$ second
Execution time on machine B: $\text{Execution}_B = 10$ seconds
$\text{Performance}_A / \text{Performance}_B = \text{Execution Time}_B / \text{Execution Time}_A = 10 / 1 = 10$

The performance of machine A is 10 times the performance of machine B when running this program, or: Machine A is said to be 10 times faster than machine B when running this program.

CPU Performance Equation

CPU time = CPU clock cycles for a program
X Clock cycle time

or:

CPU time = CPU clock cycles for a program / clock rate

CPI (clock cycles per instruction):

CPI = CPU clock cycles for a program / I

where I is the instruction count.

CPU Execution Time: The CPU Equation

- A program is comprised of a number of instructions, I
  - Measured in: instructions/program
- The average instruction takes a number of cycles per instruction (CPI) to be completed.
  - Measured in: cycles/instruction
- CPU has a fixed clock cycle time C = 1/clock rate
  - Measured in: seconds/cycle
- CPU execution time is the product of the above three parameters as follows:

CPU Time = I x CPI x C

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

CPU Execution Time

For a given program and machine:

CPI = Total program execution cycles / Instructions count

$\rightarrow$ CPU clock cycles = Instruction count x CPI

CPU execution time =

= CPU clock cycles x Clock cycle

= Instruction count x CPI x Clock cycle

= I x CPI x C

CPU Execution Time: Example

- **A Program is running on a specific machine with the following parameters:**
  - **Total instruction count: 10,000,000 instructions**
  - **Average CPI for the program: 2.5 cycles/instruction.**
  - **CPU clock rate: 200 MHz.**
- **What is the execution time for this program:**

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

CPU time = Instruction count x CPI x Clock cycle

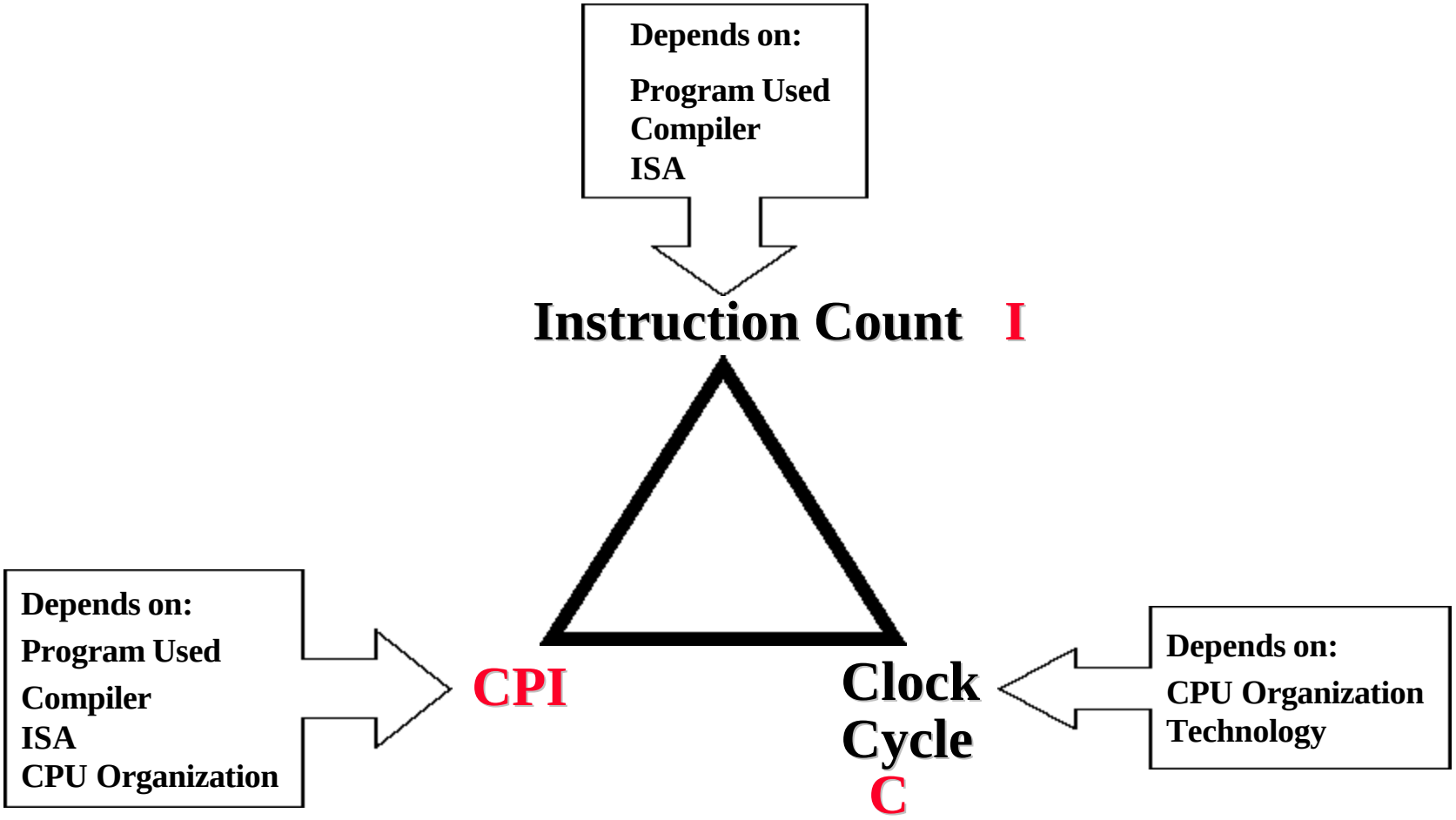
= 10,000,000 x 2.5 x 1 / clock rate

= 10,000,000 x 2.5 x 5×10^{-9}

= .125 seconds

Aspects of CPU Execution Time

CPU Time = Instruction count x CPI x Clock cycle



Factors Affecting CPU Performance

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

	Instruction Count I	CPI	Clock Cycle C
Program	X	X	
Compiler	X	X	
Instruction Set Architecture (ISA)	X	X	
Organization		X	X
Technology			X

Performance Comparison: Example

- From the previous example: A Program is running on a specific machine with the following parameters:
  - Total instruction count: 10,000,000 instructions
  - Average CPI for the program: 2.5 cycles/instruction.
  - CPU clock rate: 200 MHz.
- Using the same program with these changes:
  - A new compiler used: New instruction count 9,500,000

    New CPI: 3.0
  - Faster CPU implementation: New clock rate = 300 MHZ
- What is the speedup with the changes?

$$\text{Speedup} = \frac{\text{Old Execution Time}}{\text{New Execution Time}} = \frac{I_{old} \times CPI_{old} \times \text{Clock cycle}_{old}}{I_{new} \times CPI_{new} \times \text{Clock Cycle}_{new}}$$

$$\text{Speedup} = (10{,}000{,}000 \times 2.5 \times 5\times10^{-9}) / (9{,}500{,}000 \times 3 \times 3.33\times10^{-9})$$

$$= .125 / .095 = 1.32$$

or 32 % faster after changes.

Instruction Types & CPI

- **Given a program with *n* types or classes of instructions with the following characteristics:**

C_i = **Count of instructions of type$_i$**

CPI_i = **Cycles per instruction for type$_i$**

Then:

CPI = CPU Clock Cycles / Instruction Count I

Where:

$$CPU\ clock\ cycles = \sum_{i=1}^{n} \left(CPI_i \times C_i \right)$$

Instruction Count I = S C_i

Instruction Types And CPI: An Example

- An instruction set has three instruction classes:

Instruction class	CPI
A	1
B	2
C	3

- Two code sequences have the following instruction counts:

	Instruction counts for instruction class		
Code Sequence	A	B	C
1	2	1	2
2	4	1	1

- CPU cycles for sequence 1 = 2 x 1 + 1 x 2 + 2 x 3 = 10 cycles

  CPI for sequence 1 = clock cycles / instruction count

  = 10 /5 = 2

- CPU cycles for sequence 2 = 4 x 1 + 1 x 2 + 1 x 3 = 9 cycles

  CPI for sequence 2 = 9 / 6 = 1.5

Instruction Frequency & CPI

- **Given a program with *n* types or classes of instructions with the following characteristics:**

  C_i = Count of instructions of type$_i$

  CPI_i = Average cycles per instruction of type$_i$

  F_i = Frequency of instruction type$_i$

  = C_i / total instruction count

Then:

$$CPI = \sum_{i=1}^{n} (CPI_i \times F_i)$$

Instruction Type Frequency & CPI: A RISC Example

Base Machine (Reg / Reg)

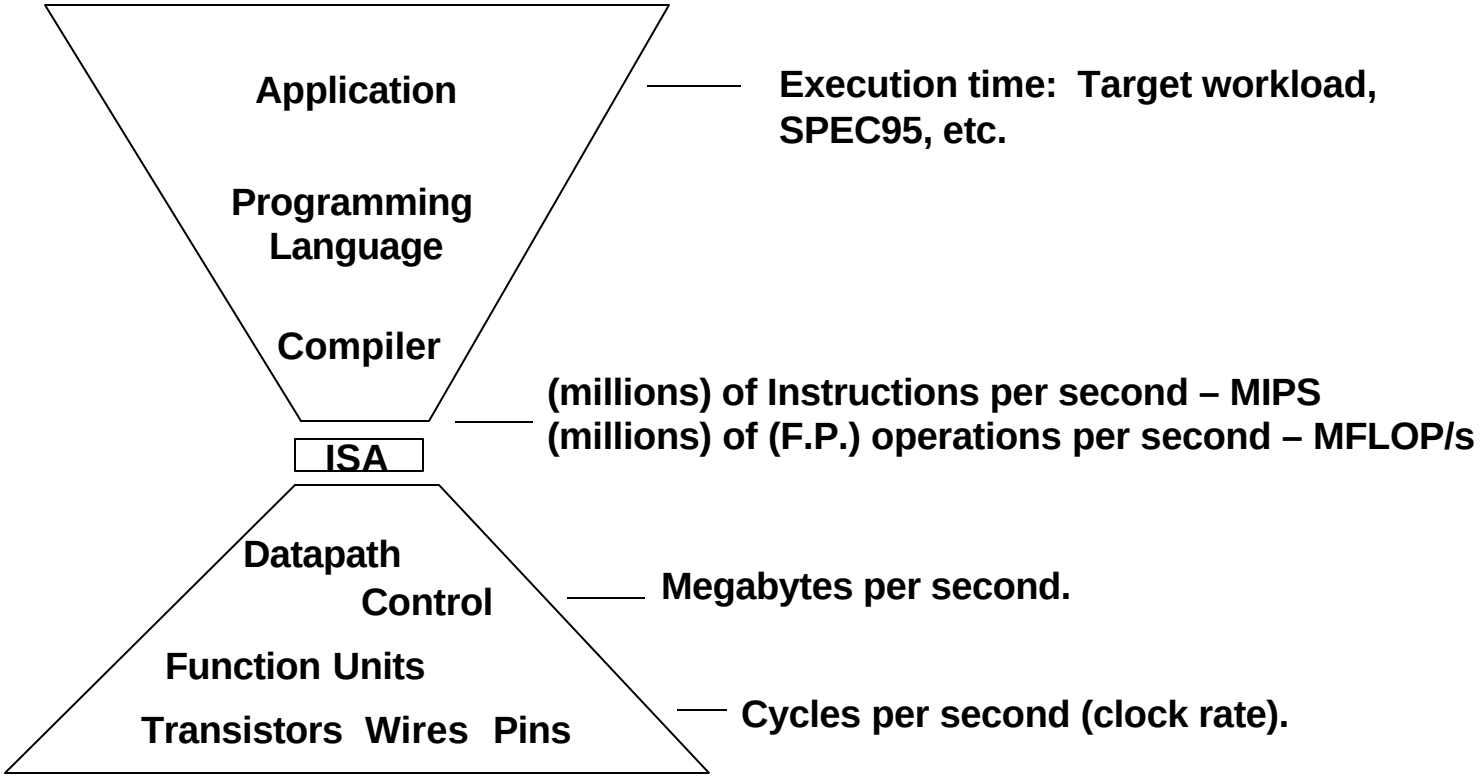
Op	Freq, F_i	CPI_i	$CPI_i \times F_i$	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%

Typical Mix

$$CPI = \sum_{i=1}^{n} (CPI_i \times F_i)$$

CPI = .5 x 1 + .2 x 5 + .1 x 3 + .2 x 2 = 2.2

Metrics of Computer Performance



Each metric has a purpose, and each can be misused.

Choosing Programs To Evaluate Performance

Levels of programs or benchmarks that could be used to evaluate performance:

- **Actual Target Workload: Full applications that run on the target machine.**
- **Real Full Program-based Benchmarks:**
  - **Select a specific mix or suite of programs that are typical of targeted applications or workload (e.g SPEC95, SPEC CPU2000).**
- **Small "Kernel" Benchmarks:**
  - **Key computationally-intensive pieces extracted from real programs.**
    - **Examples: Matrix factorization, FFT, tree search, etc.**
  - **Best used to test specific aspects of the machine.**
- **Microbenchmarks:**
  - **Small, specially written programs to isolate a specific aspect of performance characteristics: Processing: integer, floating point, local memory, input/output, etc.**

Types of Benchmarks

Pros		Cons
• Representative	Actual Target Workload	• Very specific. • Non-portable. • Complex: Difficult to run, or measure.
• Portable. • Widely used. • Measurements useful in reality.	Full Application Benchmarks	• Less representative than actual workload.
• Easy to run, early in the design cycle.	Small “Kernel” Benchmarks	• Easy to “fool” by designing hardware to run them well.
• Identify peak performance and potential bottlenecks.	Microbenchmarks	• Peak performance results may be a long way from real application performance

SPEC: System Performance Evaluation Cooperative

The most popular and industry-standard set of CPU benchmarks.

- **SPECmarks, 1989:**
  - **10 programs yielding a single number ("SPECmarks").**
- **SPEC92, 1992:**
  - **SPECInt92 (6 integer programs) and SPECfp92 (14 floating point programs).**
- **SPEC95, 1995:**
  - SPECint95 (8 integer programs):
    - **go, m88ksim, gcc, compress, li, ijpeg, perl, vortex**
  - SPECfp95 (10 floating-point intensive programs):
    - **tomcatv, swim, su2cor, hydro2d, mgrid, applu, turb3d, apsi, fppp, wave5**
  - Performance relative to a Sun SuperSpark I (50 MHz) which is given a score of SPECint95 = SPECfp95 = 1
- **SPEC CPU2000, 1999:**
  - **CINT2000 (11 integer programs). CFP2000 (14 floating-point intensive programs)**
  - **Performance relative to a Sun Ultra5_10 (300 MHz) which is given a score of SPECint2000 = SPECfp2000 = 100**

SPEC CPU2000 Programs

	Benchmark	Language	Descriptions
CINT2000 (Integer)	**164.gzip**	**C**	**Compression**
	175.vpr	**C**	**FPGA Circuit Placement and Routing**
	176.gcc	**C**	**C Programming Language Compiler**
	181.mcf	**C**	**Combinatorial Optimization**
	186.crafty	**C**	**Game Playing: Chess**
	197.parser	**C**	**Word Processing**
	252.eon	**C++**	**Computer Visualization**
	253.perlbmk	**C**	**PERL Programming Language**
	254.gap	**C**	**Group Theory, Interpreter**
	255.vortex	**C**	**Object-oriented Database**
	256.bzip2	**C**	**Compression**
	300.twolf	**C**	**Place and Route Simulator**
CFP2000 (Floating Point)	**168.wupwise**	**Fortran 77**	**Physics / Quantum Chromodynamics**
	171.swim	**Fortran 77**	**Shallow Water Modeling**
	172.mgrid	**Fortran 77**	**Multi-grid Solver: 3D Potential Field**
	173.applu	**Fortran 77**	**Parabolic / Elliptic Partial Differential Equations**
	177.mesa	**C**	**3-D Graphics Library**
	178.galgel	**Fortran 90**	**Computational Fluid Dynamics**
	179.art	**C**	**Image Recognition / Neural Networks**
	183.equake	**C**	**Seismic Wave Propagation Simulation**
	187.facerec	**Fortran 90**	**Image Processing: Face Recognition**
	188.ammp	**C**	**Computational Chemistry**
	189.lucas	**Fortran 90**	**Number Theory / Primality Testing**
	191.fma3d	**Fortran 90**	**Finite-element Crash Simulation**
	200.sixtrack	**Fortran 77**	**High Energy Nuclear Physics Accelerator Design**
	301.apsi	**Fortran 77**	**Meteorology: Pollutant Distribution**

Source: http://www.spec.org/osg/cpu2000/

Top 20 SPEC CPU2000 Results (As of March 2002)

Top 20 SPECint2000

#	MHz	Processor	int peak	int base
1	1300	POWER4	814	790
2	2200	Pentium 4	811	790
3	2200	Pentium 4 Xeon	810	788
4	1667	Athlon XP	724	697
5	1000	Alpha 21264C	679	621
6	1400	Pentium III	664	648
7	1050	UltraSPARC-III Cu	610	537
8	1533	Athlon MP	609	587
9	750	PA-RISC 8700	604	568
10	833	Alpha 21264B	571	497
11	1400	Athlon	554	495
12	833	Alpha 21264A	533	511
13	600	MIPS R14000	500	483
14	675	SPARC64 GP	478	449
15	900	UltraSPARC-III	467	438
16	552	PA-RISC 8600	441	417
17	750	POWER RS64-IV	439	409
18	700	Pentium III Xeon	438	431
19	800	Itanium	365	358
20	400	MIPS R12000	353	328

Top 20 SPECfp2000

MHz	Processor	fp peak	fp base
1300	POWER4	1169	1098
1000	Alpha 21264C	960	776
1050	UltraSPARC-III Cu	827	701
2200	Pentium 4 Xeon	802	779
2200	Pentium 4	801	779
833	Alpha 21264B	784	643
800	Itanium	701	701
833	Alpha 21264A	644	571
1667	Athlon XP	642	596
750	PA-RISC 8700	581	526
1533	Athlon MP	547	504
600	MIPS R14000	529	499
675	SPARC64 GP	509	371
900	UltraSPARC-III	482	427
1400	Athlon	458	426
1400	Pentium III	456	437
500	PA-RISC 8600	440	397
450	POWER3-II	433	426
500	Alpha 21264	422	383
400	MIPS R12000	407	382

Source: http://www.aceshardware.com/SPECmine/top.jsp

Computer Performance Measures : MIPS (Million Instructions Per Second)

- For a specific program running on a specific computer is a measure of millions of instructions executed per second:

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution Time} \times 10^6} = \frac{\text{Instruction count}}{\text{CPU clocks} \times \text{Cycle time} \times 10^6} = \frac{\text{Instruction count} \times \text{Clock rate}}{\text{Instruction count} \times \text{CPI} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- Faster execution time usually means faster MIPS rating.
- Problems:
  - No account for instruction set used.
  - Program-dependent: A single machine does not have a single MIPS rating.
  - Cannot be used to compare computers with different instruction sets.
  - A higher MIPS rating in some cases may not mean higher performance or better execution time. i.e. due to compiler design variations.

Compiler Variations, MIPS, Performance: An Example

- For the machine with instruction classes:

Instruction class	CPI
A	1
B	2
C	3

- For a given program two compilers produced the following instruction counts:

	Instruction counts (in millions) for each instruction class		
Code from:	A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

- The machine is assumed to run at a clock rate of 100 MHz

Compiler Variations, MIPS, Performance: An Example (Continued)

MIPS = Clock rate / (CPI x 10^6) = 100 MHz / (CPI x 10^6)

CPI = CPU execution cycles / Instructions count

$$CPU\ clock\ cycles = \sum_{i=1}^{n} \left(CPI_i \times C_i \right)$$

CPU time = Instruction count x CPI / Clock rate

- **For compiler 1:**
  - **CPI_1 = (5 x 1 + 1 x 2 + 1 x 3) / (5 + 1 + 1) = 10 / 7 = 1.43**
  - **MIP_1 = 100 / (1.428 x 10^6) = 70.0**
  - **CPU $time_1$ = ((5 + 1 + 1) x 10^6 x 1.43) / (100 x 10^6) = 0.10 seconds**
- **For compiler 2:**
  - **CPI_2 = (10 x 1 + 1 x 2 + 1 x 3) / (10 + 1 + 1) = 15 / 12 = 1.25**
  - **MIP_2 = 100 / (1.25 x 10^6) = 80.0**
  - **CPU $time_2$ = ((10 + 1 + 1) x 10^6 x 1.25) / (100 x 10^6) = 0.15 seconds**

Computer Performance Measures :

MFOLPS (Million FLOating-Point Operations Per Second)

- A floating-point operation is an addition, subtraction, multiplication, or division operation applied to numbers represented by a single or double precision floating-point representation.
- MFLOPS, for a specific program running on a specific computer, is a measure of millions of floating point-operation (megaflops) per second:

$$\text{MFLOPS} = \text{Number of floating-point operations} \;/\; (\text{Execution time} \times 10^6)$$

- A better comparison measure between different machines than MIPS.
- Program-dependent: Different programs have different percentages of floating-point operations present. i.e compilers have no such operations and yield a MFLOPS rating of zero.
- Dependent on the type of floating-point operations present in the program.

Quantitative Principles of Computer Design

- **Amdahl's Law:**

The performance gain from improving some portion of a computer is calculated by:

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement}}{\text{Performance for the entire task without using the enhancement}}$$

$$\text{or Speedup} = \frac{\text{Execution time without the enhancement}}{\text{Execution time for entire task using the enhancement}}$$

Performance Enhancement Calculations: Amdahl's Law

- The performance enhancement possible due to a given design improvement is limited by the amount that the improved feature is used
- Amdahl's Law:

Performance improvement or speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{Execution Time without E}}{\text{Execution Time with E}} = \frac{\text{Performance with E}}{\text{Performance without E}}$$

  - Suppose that enhancement E accelerates a fraction F of the execution time by a factor S and the remainder of the time is unaffected then:

$$\text{Execution Time with E} = ((1-F) + F/S) \times \text{Execution Time without E}$$

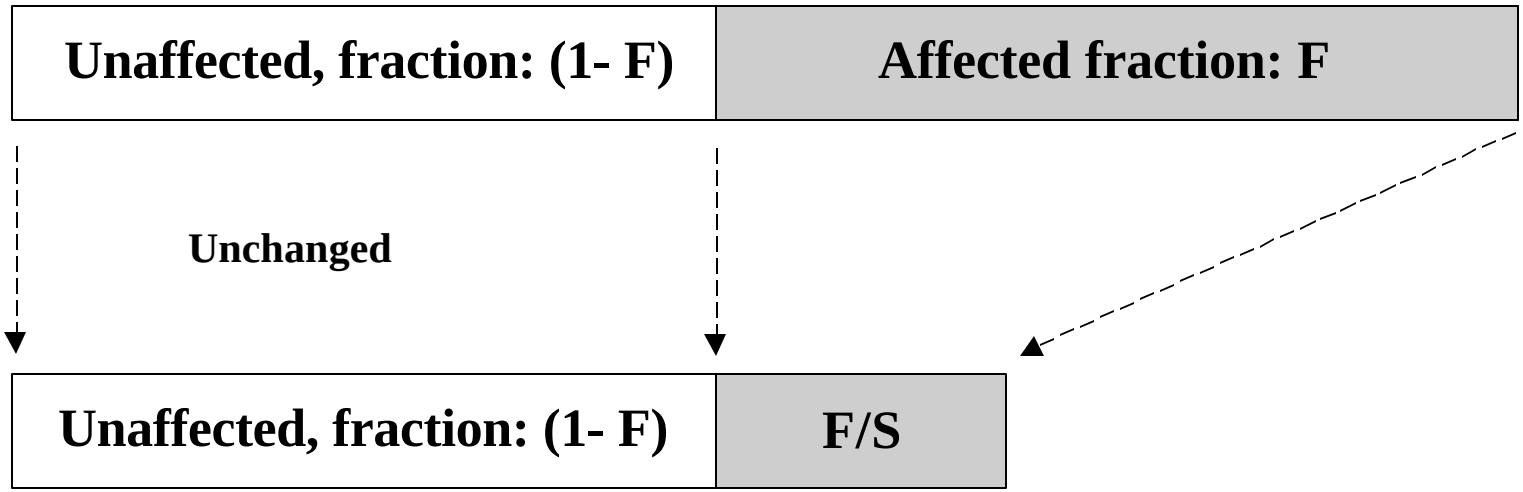
Hence speedup is given by:

$$\text{Speedup(E)} = \frac{\cancel{\text{Execution Time without E}}}{((1 - F) + F/S) \times \cancel{\text{Execution Time without E}}} = \frac{1}{(1 - F) + F/S}$$

Pictorial Depiction of Amdahl's Law

Enhancement E accelerates fraction F of execution time by a factor of S

Before:
Execution Time without enhancement E:



After:
Execution Time with enhancement E:

$$\text{Speedup(E)} = \frac{\text{Execution Time without enhancement E}}{\text{Execution Time with enhancement E}} = \frac{1}{(1 - F) + F/S}$$

Performance Enhancement Example

- **For the RISC machine with the following instruction mix given earlier:**

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%

CPI = 2.2

- **If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement:**

Fraction enhanced = F = 45% or .45

Unaffected fraction = 100% - 45% = 55% or .55

Factor of enhancement = 5/2 = 2.5

Using Amdahl's Law:

$$\text{Speedup(E)} = \frac{1}{(1 - F) + F/S} = \frac{1}{.55 + .45/2.5} = 1.37$$

An Alternative Solution Using CPU Equation

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%

CPI = 2.2

- **If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement:**

Old CPI = 2.2

New CPI = .5 x 1 + .2 x 2 + .1 x 3 + .2 x 2 = 1.6

$$\text{Speedup(E)} = \frac{\text{Original Execution Time}}{\text{New Execution Time}} = \frac{\text{Instruction count} \times \text{old CPI} \times \text{clock cycle}}{\text{Instruction count} \times \text{new CPI} \times \text{clock cycle}}$$

$$= \frac{\text{old CPI}}{\text{new CPI}} = \frac{2.2}{1.6} = 1.37$$

Which is the same speedup obtained from Amdahl's Law in the first solution.

Performance Enhancement Example

- **A program runs in 100 seconds on a machine with multiply operations responsible for 80 seconds of this time. By how much must the speed of multiplication be improved to make the program four times faster?**

$$\text{Desired speedup} = 4 = \frac{100}{\text{Execution Time with enhancement}}$$

$\rightarrow$ **Execution time with enhancement = 25 seconds**

25 seconds = (100 - 80 seconds) + 80 seconds / n

25 seconds = 20 seconds + 80 seconds / n

$\rightarrow$ **5 = 80 seconds / n**

$\rightarrow$ **n = 80/5 = 16**

Hence multiplication should be 16 times faster to get a speedup of 4.

Performance Enhancement Example

- **For the previous example with a program running in 100 seconds on a machine with multiply operations responsible for 80 seconds of this time. By how much must the speed of multiplication be improved to make the program five times faster?**

$$\text{Desired speedup} = 5 = \frac{100}{\text{Execution Time with enhancement}}$$

$\rightarrow$ **Execution time with enhancement = 20 seconds**

20 seconds = (100 - 80 seconds) + 80 seconds / n

20 seconds = 20 seconds + 80 seconds / n

$\rightarrow$ **0 = 80 seconds / n**

No amount of multiplication speed improvement can achieve this.

Extending Amdahl's Law To Multiple Enhancements

- **Suppose that enhancement E_i accelerates a fraction F_i of the execution time by a factor S_i and the remainder of the time is unaffected then:**

$$Speedup = \frac{\text{Original Execution Time}}{\left((1 - \sum_i F_i) + \sum_i \frac{F_i}{S_i}\right) X \text{Original Execution Time}}$$

$$Speedup = \frac{1}{\left((1 - \sum_i F_i) + \sum_i \frac{F_i}{S_i}\right)}$$

Note: All fractions refer to original execution time.

Amdahl's Law With Multiple Enhancements: Example

- **Three CPU performance enhancements are proposed with the following speedups and percentage of the code execution time affected:**

Speedup$_1$ = S$_1$ = 10	**Percentage$_1$ = F$_1$ = 20%**
Speedup$_2$ = S$_2$ = 15	**Percentage$_1$ = F$_2$ = 15%**
Speedup$_3$ = S$_3$ = 30	**Percentage$_1$ = F$_3$ = 10%**

- **While all three enhancements are in place in the new design, each enhancement affects a different portion of the code and only one enhancement can be used at a time.**
- **What is the resulting overall speedup?**

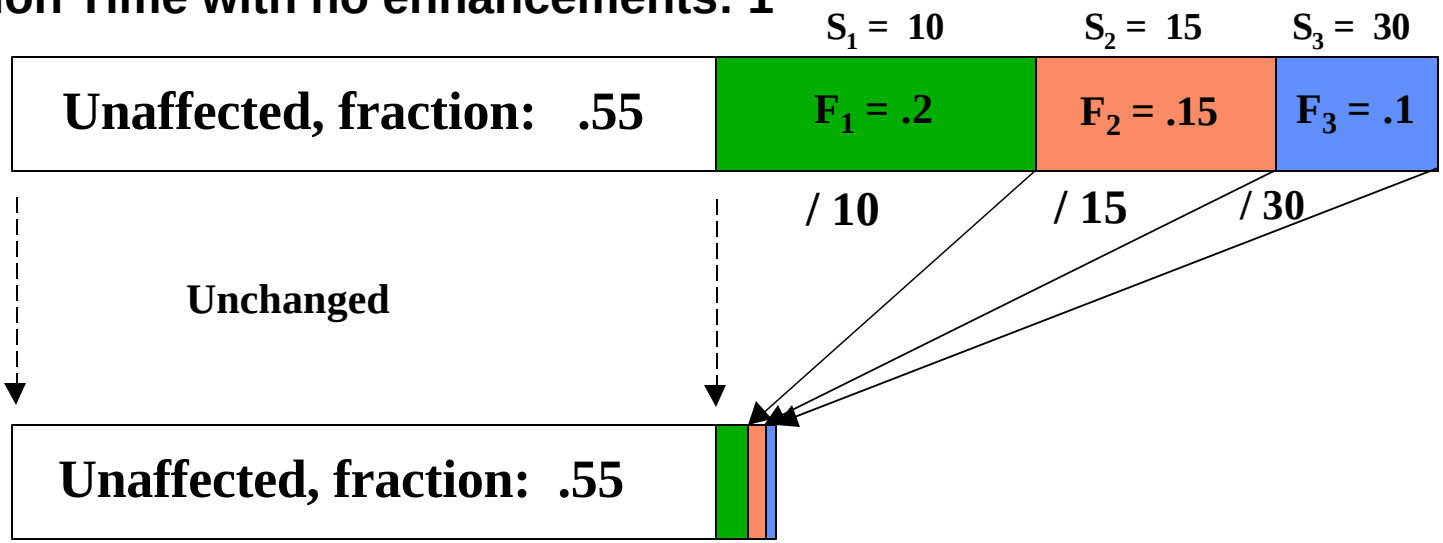
$$Speedup = \frac{1}{\left((1 - \sum_i F_i) + \sum_i \frac{F_i}{S_i}\right)}$$

- **Speedup = 1 / [(1 - .2 - .15 - .1) + .2/10 + .15/15 + .1/30)]**
  = 1 / [.55 + .0333]
  = 1 / .5833 = 1.71

Pictorial Depiction of Example

Before:

Execution Time with no enhancements: 1



After:

Execution Time with enhancements: .55 + .02 + .01 + .00333 = .5833

Speedup = 1 / .5833 = 1.71

Note: All fractions refer to original execution time.

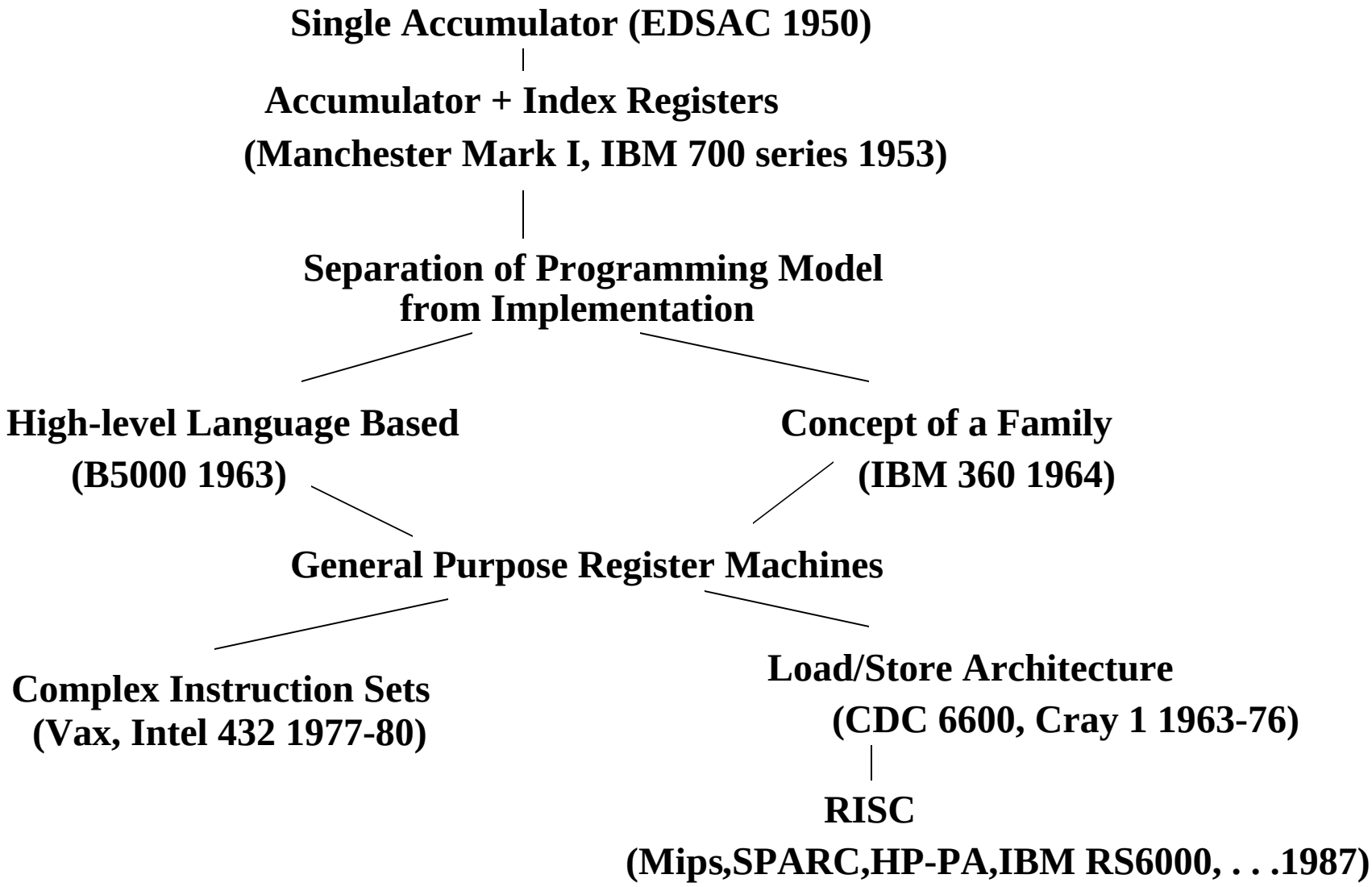
Instruction Set Architecture (ISA)

"... 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.

The instruction set architecture is concerned with:

- **Organization of programmable storage (memory & registers): Includes the amount of addressable memory and number of available registers.**
- **Data Types & Data Structures: Encodings & representations.**
- **Instruction Set: What operations are specified.**
- **Instruction formats and encoding.**
- **Modes of addressing and accessing data items and instructions**
- **Exceptional conditions.**

Evolution of Instruction Sets



Types of Instruction Set Architectures
According To Operand Addressing Fields

Memory-To-Memory Machines:

- **Operands obtained from memory and results stored back in memory by any instruction that requires operands.**
- **No local CPU registers are used in the CPU datapath.**
- **Include:**
  - **The 4 Address Machine.**
  - **The 3-address Machine.**
  - **The 2-address Machine.**

The 1-address (Accumulator) Machine:

- **A single local CPU special-purpose register (accumulator) is used as the source of one operand and as the result destination.**

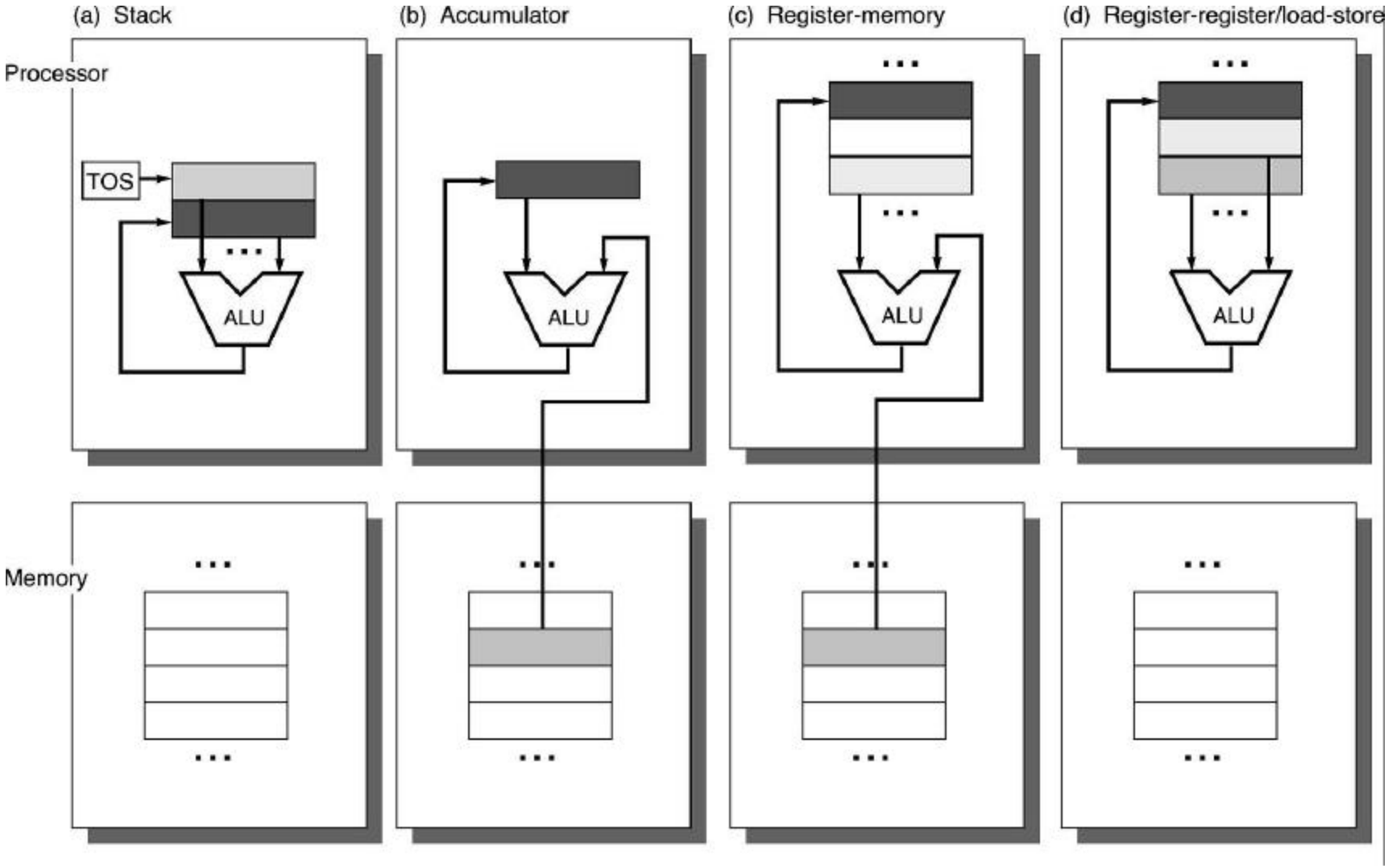
The 0-address or Stack Machine:

- **A push-down stack is used in the CPU.**

General Purpose Register (GPR) Machines:

- The CPU datapath contains several local general-purpose registers which can be used as operand sources and as result destinations.
- A large number of possible addressing modes.
- Load-Store or Register-To-Register Machines: GPR machines where only data movement instructions (loads, stores) can obtain operands from memory and store results to memory.

Operand Locations in Four ISA Classes



Code Sequence C = A + B for Four Instruction Sets

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R1, B	Load R2, B
Add	Store C	Store C, R1	Add R3,R1, R2
			Store C, R3

General-Purpose Register (GPR) Machines

- **Every machine designed after 1980 uses a load-store GPR architecture.**
- **Registers, like any other storage form internal to the CPU, are faster than memory.**
- **Registers are easier for a compiler to use.**
- **GPR architectures are divided into several types depending on two factors:**
  - **Whether an ALU instruction has two or three operands.**
  - **How many of the operands in ALU instructions may be memory addresses.**

General-Purpose Register Machines

Type	Advantages	Disadvantages
Register-register (0,3)	Simple, fixed-length instruction encoding. Simple code-generation model. Instructions take similar numbers of clocks to execute (see Ch 3).	Higher instruction count than architectures with memory references in instructions. Some instructions are short and bit encoding may be wasteful.
Register-memory (1,2)	Data can be accessed without loading first. Instruction format tends to be easy to encode and yields good density.	Operands are not equivalent since a source operand in a binary operation is destroyed. Encoding a register number and a memory address in each instruction may restrict the number of registers. Clocks per instruction varies by operand location.
Memory-memory (3,3)	Most compact. Doesn't waste registers for temporaries.	Large variation in instruction size, especially for three-operand instructions. Also, large variation in work per instruction. Memory accesses create memory bottleneck.

Advantages and disadvantages of the three most common types of general-purpose register machines.

ISA Examples

Machine	Number of General Purpose Registers	Architecture	year
EDSAC	1	accumulator	1949
IBM 701	1	accumulator	1953
CDC 6600	8	load-store	1963
IBM 360	16	register-memory	1964
DEC PDP-11	8	register-memory	1970
DEC VAX	16	register-memory memory-memory	1977
Motorola 68000	16	register-memory	1980
MIPS	32	load-store	1985
SPARC	32	load-store	1987

Examples of GPR Machines

Number of memory addresses	Maximum number of operands allowed	
0	3	SPARK, MIPS PowerPC, ALPHA
1	2	Intel 80x86, Motorola 68000
2	2	VAX
3	3	VAX

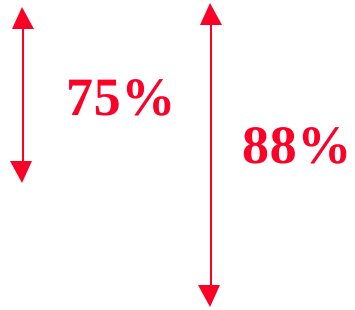
Typical Memory Addressing Modes

Addressing Mode	Sample Instruction	Meaning
Register	Add R4, R3	Regs [R4] ←Regs[R4] + Regs[R3]
Immediate	Add R4, #3	Regs[R4] ←Regs[R4] + 3
Displacement	Add R4, 10 (R1)	Regs[R4] ←Regs[R4]+Mem[10+Regs[R1]]
Indirect	Add R4, (R1)	Regs[R4] ←Regs[R4]+ Mem[Regs[R1]]
Indexed	Add R3, (R1 + R2)	Regs [R3] ←Regs[R3]+Mem[Regs[R1]+Regs[R2]]
Absolute	Add R1, (1001)	Regs[R1] ←Regs[R1] + Mem[1001]
Memory indirect	Add R1, @ (R3)	Regs[R1] ←Regs[R1] + Mem[Mem[Regs[R3]]]
Autoincrement	Add R1, (R2) +	Regs[R1] ←Regs[R1] + Mem[Regs[R2]] Regs[R2] ←Regs[R2] + d
Autodecrement	Add R1, - (R2)	Regs [R2] ←Regs[R2] -d Regs{R1] ←Regs[Regs[R1] +Mem[Regs[R2]]
Scaled	Add R1, 100 (R2) [R3]	Regs[R1] ←Regs[R1] + Mem[100+Regs[R2]+Regs[R3]*d]

Addressing Modes Usage Example

For 3 programs running on VAX ignoring direct register mode:

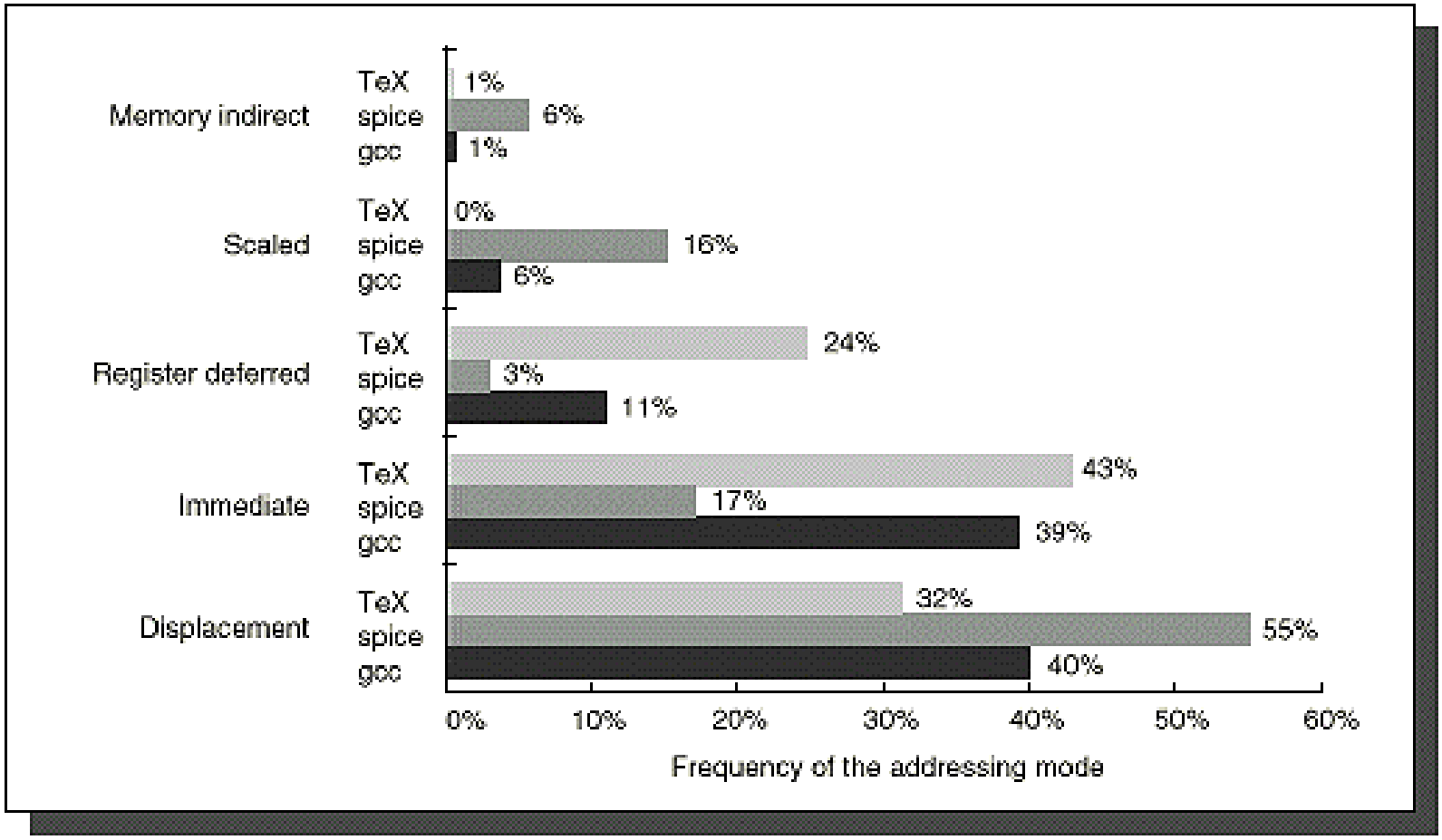
Displacement	**42% avg, 32% to 55%**
Immediate:	**33% avg, 17% to 43%**
Register deferred (indirect):	**13% avg, 3% to 24%**
Scaled:	**7% avg, 0% to 16%**
Memory indirect:	**3% avg, 1% to 6%**
Misc:	**2% avg, 0% to 3%**



75% displacement & immediate
88% displacement, immediate & register indirect.

Observation: In addition Register direct, Displacement, Immediate, Register Indirect addressing modes are important.

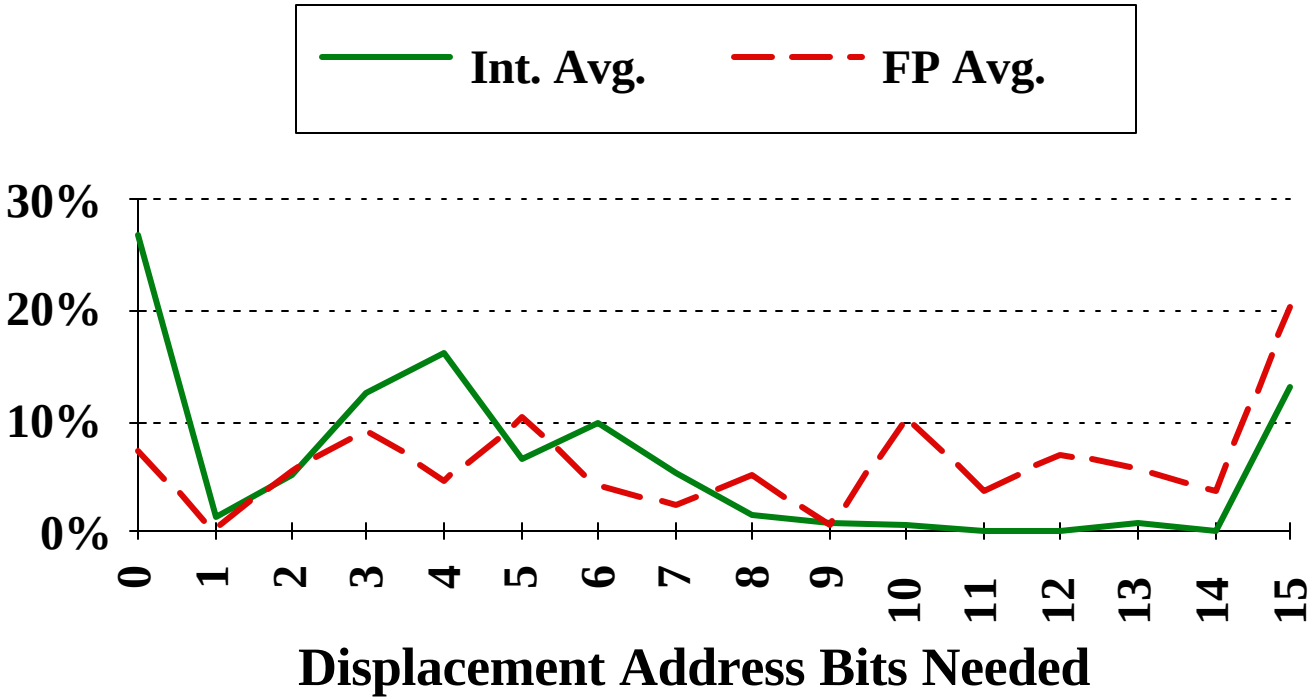
Utilization of Memory Addressing Modes



Summary of use of memory addressing modes (including immediates).

Displacement Address Size Example

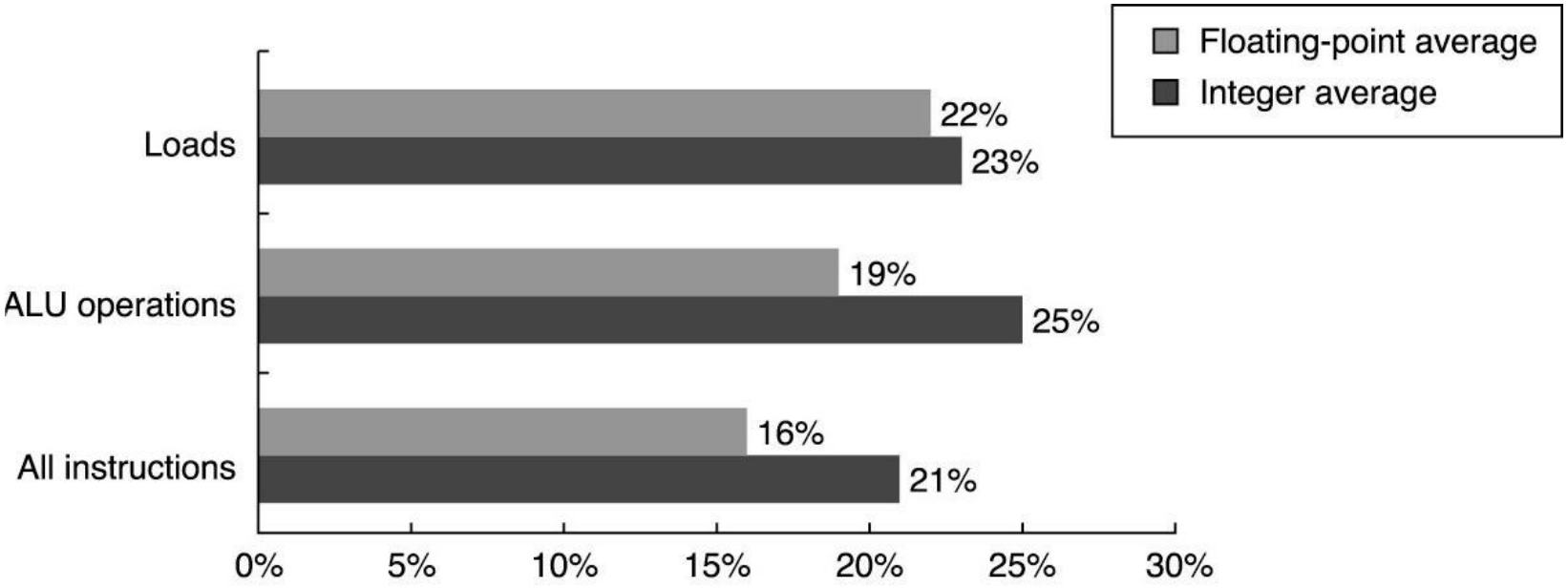
Avg. of 5 SPECint92 programs v. avg. 5 SPECfp92 programs



1% of addresses > 16-bits

12 - 16 bits of displacement needed

Immediate Addressing Mode



About one quarter of data transfers and ALU operations have an immediate operand for SPEC CPU2000 programs.

Operation Types in The Instruction Set

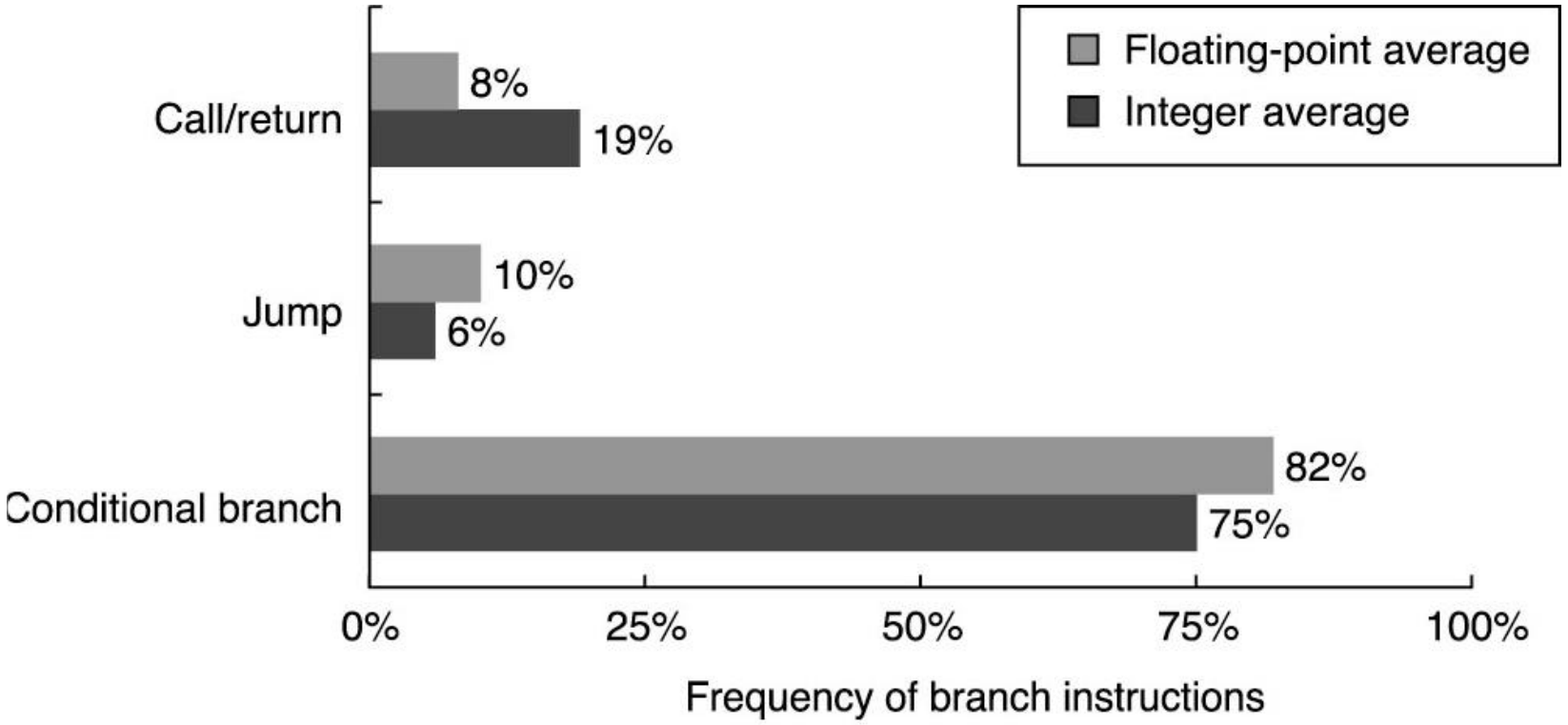
Operator Type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, or
Data transfer	Loads-stores (move on machines with memory addressing)
Control	Branch, jump, procedure call, and return, traps.
System	Operating system call, virtual memory management instructions
Floating point	Floating point operations: add, multiply.
Decimal	Decimal add, decimal multiply, decimal to character conversion
String	String move, string compare, string search
Graphics	Pixel operations, compression/ decompression operations

Instruction Usage Example: Top 10 Intel X86 Instructions

Rank	instruction	Integer Average Percent total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

Observation: Simple instructions dominate instruction usage frequency.

Instructions for Control Flow



Breakdown of control flow instructions into three classes: calls or returns, jumps and conditional branches for SPEC CPU2000 programs.

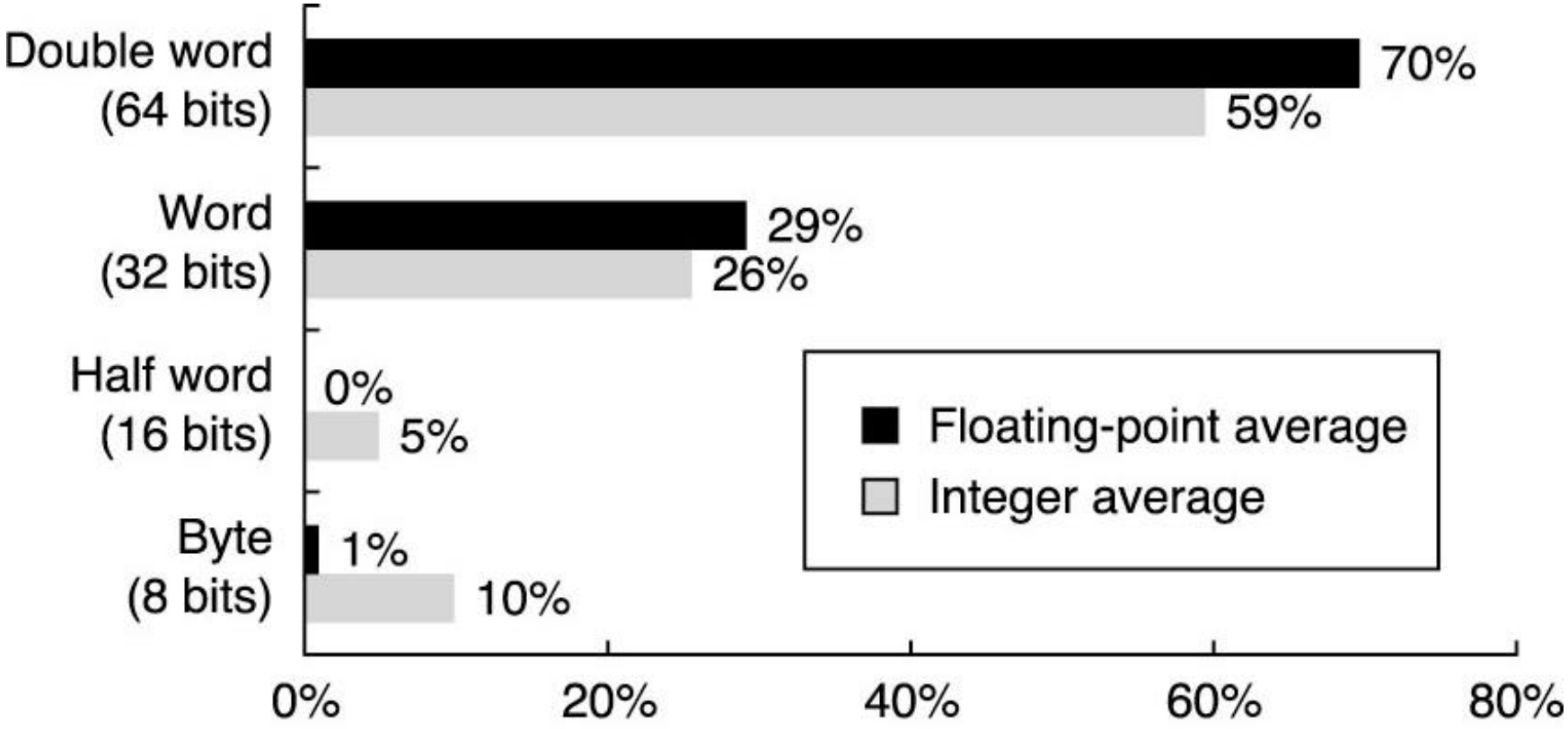
Type and Size of Operands

- Common operand types include (assuming a 64 bit CPU):

  Character (1 byte)
  Half word (16 bits)
  Word (32 bits)
  Double word (64 bits)

- IEEE standard 754: single-precision floating point (1 word), double-precision floating point (2 words).
- For business applications, some architectures support a decimal format (packed decimal, or binary coded decimal, BCD).

Type and Size of Operands



Distribution of data accesses by size for SPEC CPU2000 benchmark programs

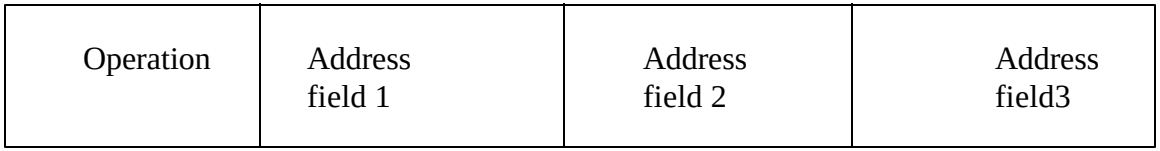
Instruction Set Encoding

Considerations affecting instruction set encoding:

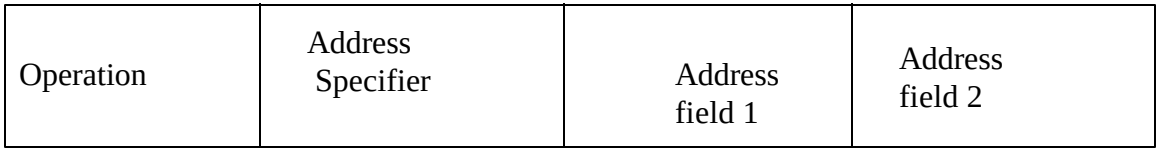
- **To have as many registers and address modes as possible.**
- **The Impact of of the size of the register and addressing mode fields on the average instruction size and on the average program.**
- **To encode instructions into lengths that will be easy to handle in the implementation. On a minimum to be a multiple of bytes.**

Three Examples of Instruction Set Encoding

Variable: VAX (1-53 bytes)



Fixed: DLX, MIPS, PowerPC, SPARC



Hybrid : IBM 360/370, Intel 80x86

Complex Instruction Set Computer (CISC)

- **Emphasizes doing more with each instruction**
- **Motivated by the high cost of memory and hard disk capacity when original CISC architectures were proposed**
  - **When M6800 was introduced: 16K RAM = $500, 40M hard disk = $ 55, 000**
  - **When MC68000 was introduced: 64K RAM = $200, 10M HD = $5,000**
- **Original CISC architectures evolved with faster more complex CPU designs but backward instruction set compatibility had to be maintained.**
- **Wide variety of addressing modes:**
    - **14 in MC68000, 25 in MC68020**
- **A number instruction modes for the location and number of operands:**
    - **The VAX has 0- through 3-address instructions.**
- **Variable-length instruction encoding.**

Example CISC ISA:
Motorola 680X0

18 addressing modes:

- Data register direct.
- Address register direct.
- Immediate.
- Absolute short.
- Absolute long.
- Address register indirect.
- Address register indirect with postincrement.
- Address register indirect with predecrement.
- Address register indirect with displacement.
- Address register indirect with index (8-bit).
- Address register indirect with index (base).
- Memory inderect postindexed.
- Memory indirect preindexed.
- Program counter indirect with index (8-bit).
- Program counter indirect with index (base).
- Program counter indirect with displacement.
- Program counter memory indirect postindexed.
- Program counter memory indirect preindexed.

Operand size:

- Range from 1 to 32 bits, 1, 2, 4, 8, 10, or 16 bytes.

Instruction Encoding:

- Instructions are stored in 16-bit words.
- the smallest instruction is 2- bytes (one word).
- The longest instruction is 5 words (10 bytes) in length.

Example CISC ISA:

Intel X86, 386/486/Pentium

12 addressing modes:

- Register.
- Immediate.
- Direct.
- Base.
- Base + Displacement.
- Index + Displacement.
- Scaled Index + Displacement.
- Based Index.
- Based Scaled Index.
- Based Index + Displacement.
- Based Scaled Index + Displacement.
- Relative.

Operand sizes:

- Can be 8, 16, 32, 48, 64, or 80 bits long.
- Also supports string operations.

Instruction Encoding:

- The smallest instruction is one byte.
- The longest instruction is 12 bytes long.
- The first bytes generally contain the opcode, mode specifiers, and register fields.
- The remainder bytes are for address displacement and immediate data.

Reduced Instruction Set Computer (RISC)

- Focuses on reducing the number and complexity of instructions of the machine.
- Reduced CPI. Goal: At least one instruction per clock cycle.
- Designed with pipelining in mind.
- Fixed-length instruction encoding.
- Only load and store instructions access memory.
- Simplified addressing modes.
  - Usually limited to immediate, register indirect, register displacement, indexed.
- Delayed loads and branches.
- Instruction pre-fetch and speculative execution.
- Examples: MIPS, SPARC, PowerPC, Alpha

Example RISC ISA:

PowerPC

8 addressing modes:

- Register direct.
- Immediate.
- Register indirect.
- Register indirect with immediate index (loads and stores).
- Register indirect with register index (loads and stores).
- Absolute (jumps).
- Link register indirect (calls).
- Count register indirect (branches).

Operand sizes:

- Four operand sizes: 1, 2, 4 or 8 bytes.

Instruction Encoding:

- Instruction set has 15 different formats with many minor variations.
- All are 32 bits in length.

Example RISC ISA:

HP Precision Architecture, HP-PA

7 addressing modes:

- Register
- Immediate
- Base with displacement
- Base with scaled index and displacement
- Predecrement
- Postincrement
- PC-relative

Operand sizes:

- Five operand sizes ranging in powers of two from 1 to 16 bytes.

Instruction Encoding:

- Instruction set has 12 different formats.
- All are 32 bits in length.

Example RISC ISA:

SPARC

5 addressing modes:

- Register indirect with immediate displacement.
- Register inderect indexed by another register.
- Register direct.
- Immediate.
- PC relative.

Operand sizes:

- Four operand sizes: 1, 2, 4 or 8 bytes.

Instruction Encoding:

- Instruction set has 3 basic instruction formats with 3 minor variations.
- All are 32 bits in length.

Example RISC ISA:

Compaq Alpha AXP

4 addressing modes:

- Register direct.
- Immediate.
- Register indirect with displacement.
- PC-relative.

Operand sizes:

- Four operand sizes: 1, 2, 4 or 8 bytes.

Instruction Encoding:

- Instruction set has 7 different formats.
- All are 32 bits in length.

RISC ISA Example:

MIPS R3000 (32-bits)

Instruction Categories:

- Load/Store.
- Computational.
- Jump and Branch.
- Floating Point (using coprocessor).
- Memory Management.
- Special.

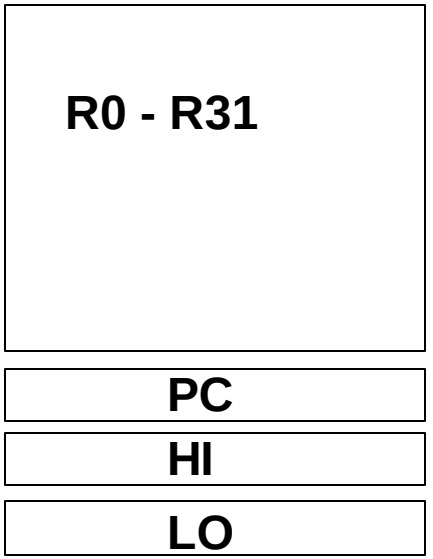
4 Addressing Modes:

- Base register + immediate offset (loads and stores).
- Register direct (arithmetic).
- Immedate (jumps).
- PC relative (branches).

Operand Sizes:

- Memory accesses in any multiple between 1 and 8 bytes.

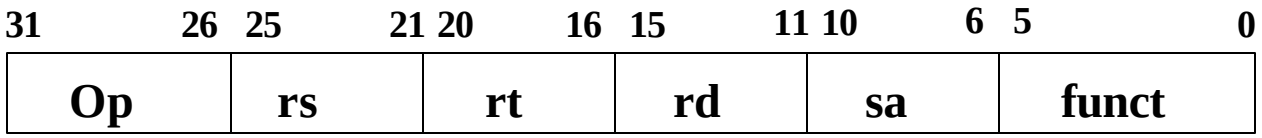
Registers



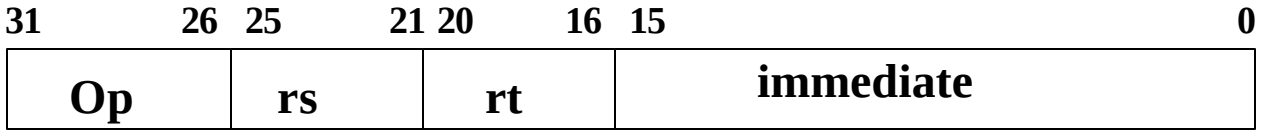
Instruction Encoding: 3 Instruction Formats, all 32 bits wide.

A RISC ISA Example: MIPS

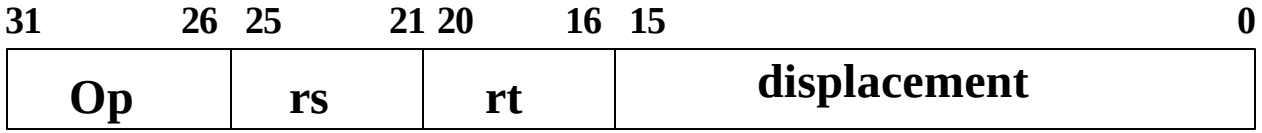
Register-Register



Register-Immediate



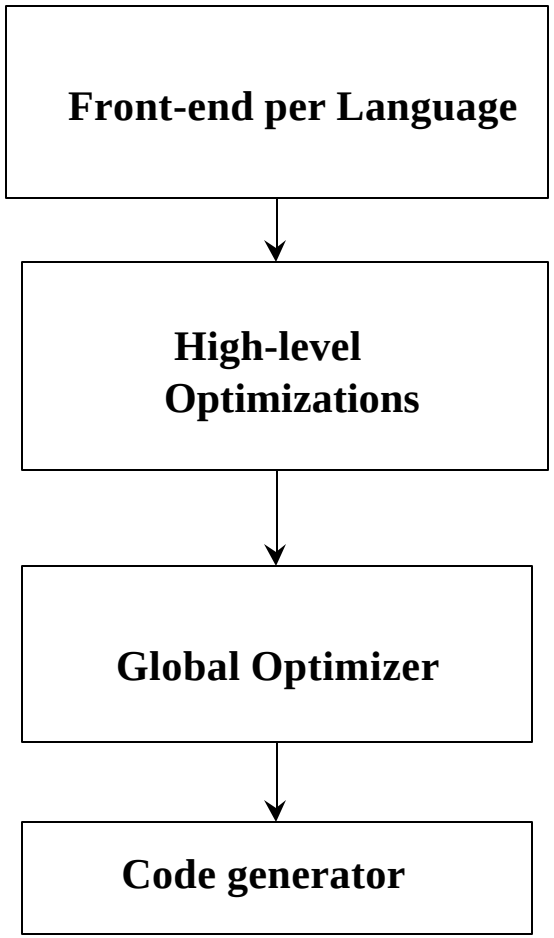
Branch



Jump / Call

The Role of Compilers

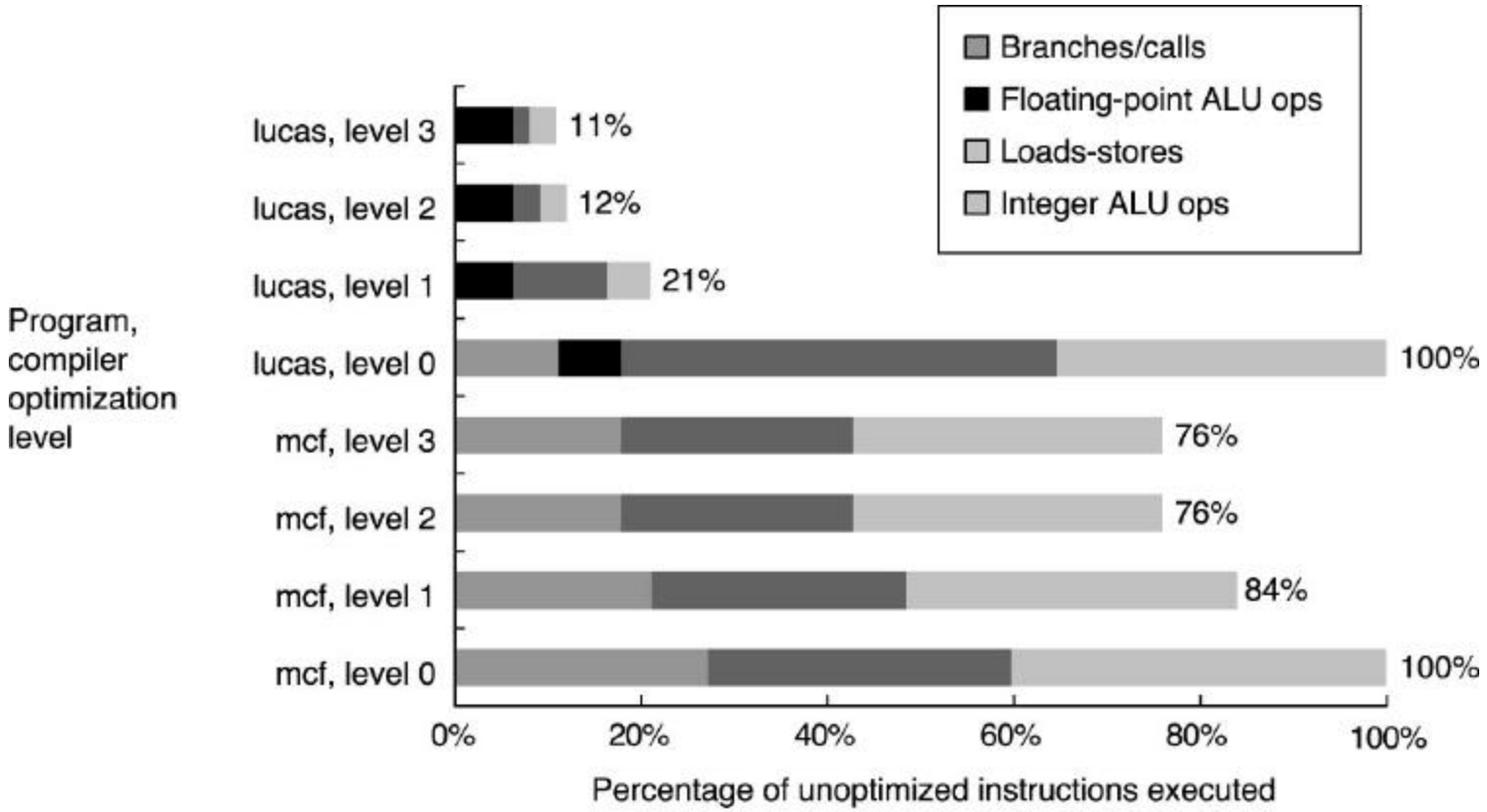
The Structure of Recent Compilers:



Optimization name	Explanation	Percentage of the total number of optimizing transforms
High-level	**At or near the source level; machine-independent**	
Procedure integration	Replace procedure call by procedure body	N.M.
Local	**Within straight-line code**	
Common subexpression elimination	Replace two instances of the same computation by single copy	18%
Constant propagation	Replace all instances of a variable that is assigned a constant with the constant	22%
Stack height reduction	Rearrange expression tree to minimize resources needed for expression evaluation	N.M.
Global	**Across a branch**	
Global common subexpression elimination	Same as local, but this version crosses branches	13%
Copy propagation	Replace all instances of a variable A that has been assigned X (i.e., $A = X$) with X	11%
Code motion	Remove code from a loop that computes same value each iteration of the loop	16%
Induction variable elimination	Simplify/eliminate array-addressing calculations within loops	2%
Machine-dependent	**Depends on machine knowledge**	
Strength reduction	Many examples, such as replace multiply by a constant with adds and shifts	N.M.
Pipeline scheduling	Reorder instructions to improve pipeline performance	N.M.
Branch offset optimization	Choose the shortest branch displacement that reaches target	N.M.

Major Types of Compiler Optimization

Compiler Optimization and Instruction Count



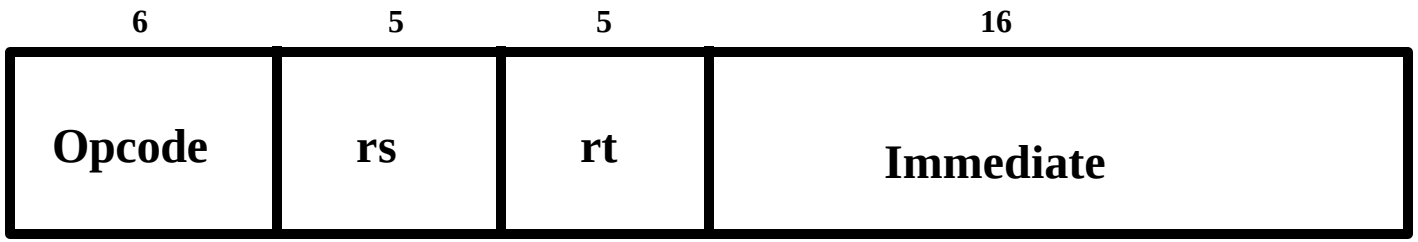
Change in instruction count for the programs lucas and mcf from SPEC2000 as compiler optimizations vary.

An Instruction Set Example: MIPS64

- **A RISC-type 64-bit instruction set architecture based on instruction set design considerations of chapter 2:**
  - **Use general-purpose registers with a load/store architecture to access memory.**
  - **Reduced number of addressing modes: displacement (offset size of 16 bits), immediate (16 bits).**
  - **Data sizes: 8 (byte), 16 (half word) , 32 (word), 64 (double word) bit integers and 32-bit or 64-bit IEEE 754 floating-point numbers.**
  - **Use fixed instruction encoding (32 bits) for performance.**
  - **32, 64-bit general-purpose integer registers GPRs, R0, …., R31. R0 always has a value of zero.**
  - **Separate 32, 64-bit floating point registers FPRs: F0, F1 … F31 When holding a 32-bit single-precision number the upper half of the FPR is not used.**

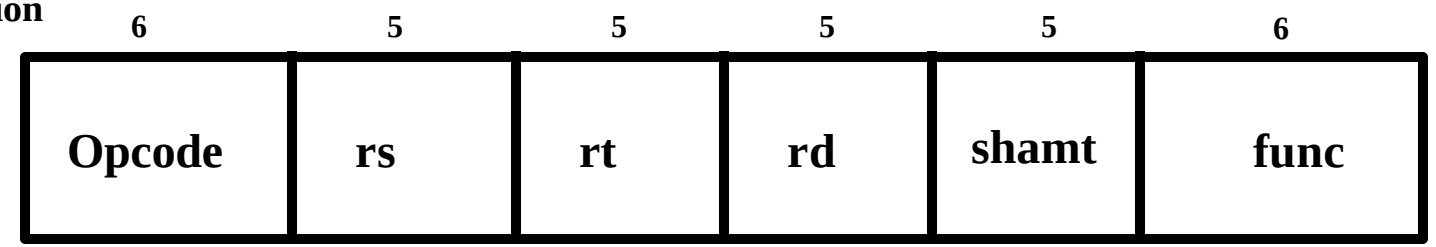
MIPS64 Instruction Format

I - type instruction



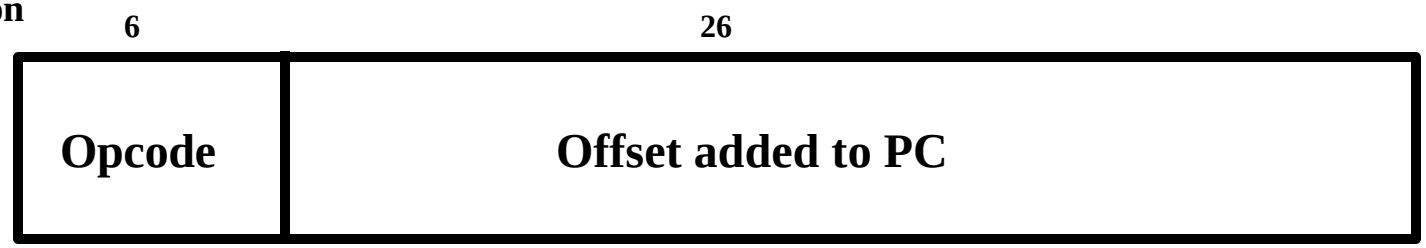
Encodes: Loads and stores of bytes, words, half words. All immediates (rd ← rs op immediate)
Conditional branch instructions (rs1 is register, rd unused)
Jump register, jump and link register (rd = 0, rs = destination, immediate = 0)

R - type instruction



Register-register ALU operations: rd ← rs func rt Function encodes the data path operation: Add, Sub .. Read/write special registers and moves.

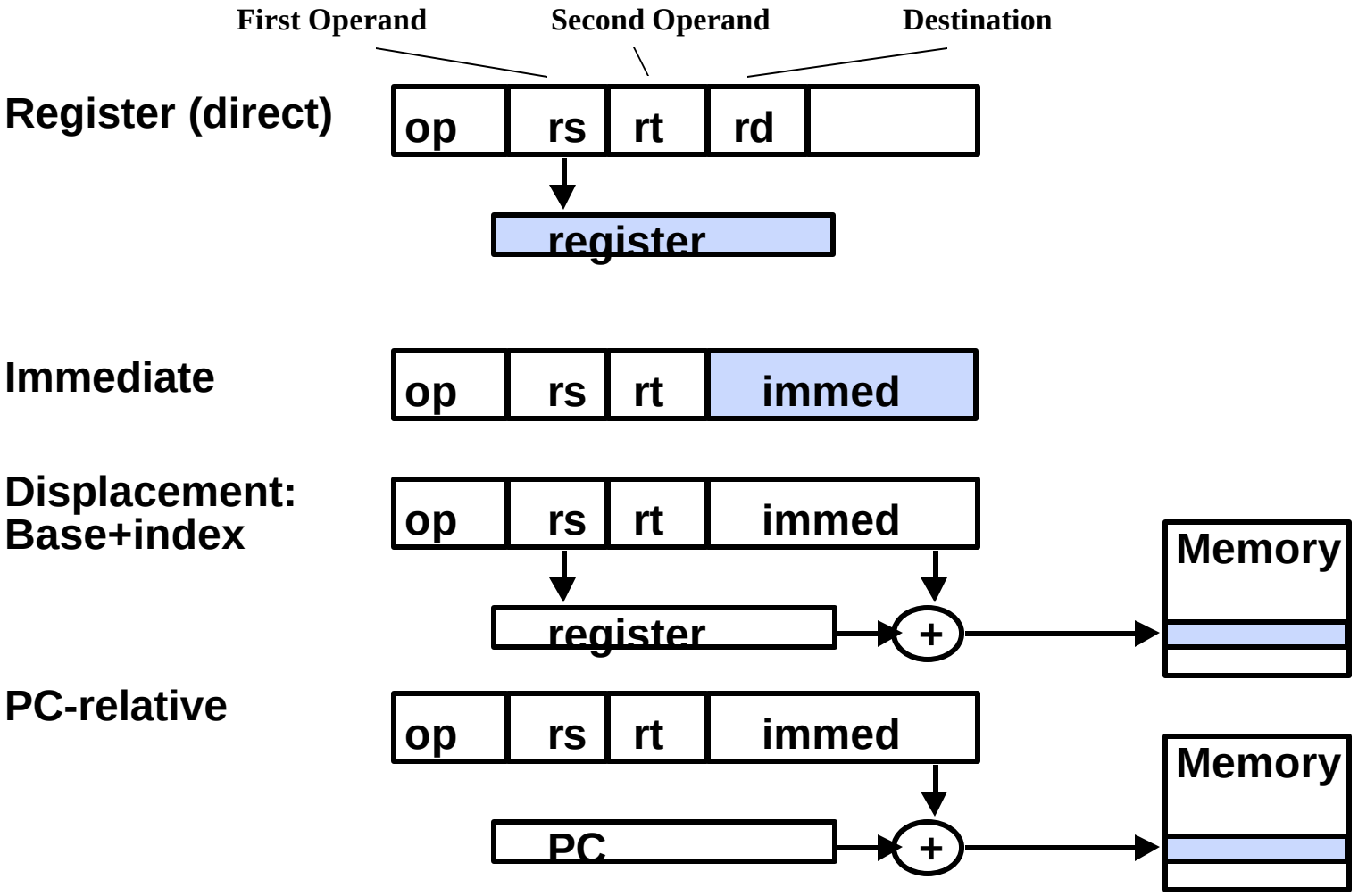
J - Type instruction



Jump and jump and link. Trap and return from exception

MIPS Addressing Modes/Instruction Formats

- All instructions 32 bits wide



MIPS64 Instructions: Load and Store

LD R1,30(R2)	Load double word	Regs[R1] $\leftarrow_{64}$ Mem[30+Regs[R2]]
LW R1, 60(R2)	Load word	Regs[R1] $\leftarrow_{64}$ (Mem[60+Regs[R2]]$_0$)^{32} ## Mem[60+Regs[R2]]
LB R1, 40(R3)	Load byte	Regs[R1] $\leftarrow_{64}$ (Mem[40+Regs[R3]]$_0$)^{56} ## Mem[40+Regs[R3]]
LBU R1, 40(R3)	Load byte unsigned	Regs[R1] $\leftarrow_{64}$ 0^{56} ## Mem[40+Regs[R3]]
LH R1, 40(R3)	Load half word	Regs[R1] $\leftarrow_{64}$ (Mem[40+Regs[R3]]$_0$)^{48} ## Mem[40 + Regs[R3]] # # Mem [41+Regs[R3]]
L.S F0, 50(R3)	Load FP single	Regs[F0] $\leftarrow_{64}$ Mem[50+Regs[R3]] ## 0^{32}
L.D F0, 50(R2)	Load FP double	Regs[F0] $\leftarrow_{64}$ Mem[50+Regs[R2]]
SD R3,500(R4)	Store double word	Mem [500+Regs[R4]] $\leftarrow_{64}$ Reg[R3]
SW R3,500(R4)	Store word	Mem [500+Regs[R4]] $\leftarrow_{32}$ Reg[R3]
S.S F0, 40(R3)	Store FP single	Mem [40, Regs[R3]] $\leftarrow_{32}$ Regs[F0] $_{0\ldots31}$
S.D F0,40(R3)	Store FP double	Mem[40+Regs[R3]] $\leftarrow$-$_{64}$ Regs[F0]
SH R3, 502(R2)	Store half	Mem[502+Regs[R2]] $\leftarrow_{16}$ Regs[R3]$_{48\ldots63}$
SB R2, 41(R3)	Store byte	Mem[41 + Regs[R3]] $\leftarrow_{8}$ Regs[R2] $_{56\ldots63}$

MIPS64 Instructions: Arithmetic/Logical

DADDU R1, R2, R3 Add unsigned Regs[R1] ← Regs[R2] + Regs[R3]

DADDI R1, R2, #3 Add immediate Regs[R1] ← Regs[R2] + 3

LUI R1, #42 Load upper immediate Regs[R1] ← 0^{32} ##42 ## 0^{16}

DSLL R1, R2, #5 Shift left logical Regs[R1] ← Regs [R2] <<5

DSLT R1, R2, R3 Set less than if (regs[R2] < Regs[R3])
Regs [R1] ← 1 else Regs[R1] ← 0

MIPS64 Instructions: Control-Flow

J name	Jump	$PC_{36..63} \leftarrow$ name
JAL name	Jump and link	Regs[31] $\leftarrow$ PC+4; $PC_{36..63} \leftarrow$ name; $((PC+4)-2^{27}) \leq$ name $< ((PC+4)+2^{27})$
JALR R2	Jump and link register	Regs[R31] $\leftarrow$ PC+4; PC $\leftarrow$ Regs[R2]
JR R3	Jump register	PC $\leftarrow$ Regs[R3]
BEQZ R4, name	Branch equal zero	if (Regs[R4] ==0) PC $\leftarrow$ name; $((PC+4)-2^{17}) \leq$ name $< ((PC+4)+2^{17}$
BNEZ R4, Name	Branch not equal zero	if (Regs[R4] != 0) PC $\leftarrow$ name $((PC+4)-2^{17}) \leq$ name $< ((PC+4)+2^{17}$
MOVZ R1,R2,R3	Conditional move if zero	if (Regs[R3] ==0) Regs[R1] $\leftarrow$ Regs[R2]