


*CS 3889*

*Performance Metrics  
advanced module*



A.R. Hurson  
323 Computer Science Building,  
Missouri S&T  
hurson@mst.edu

# Performance Metrics



## ◆ Outline

- ★ Amdahl's law
- ★ Green computing
- ★ CPU Time
- ★ Formulation of CPU time in terms of Instruction count, clock cycle time, and number of clock cycles per instruction
- ★ Formulation of CPU time in terms of Instruction count, clock cycle time, number of clock cycles per instruction, and role of different components in a simple computer organization
- ★ How to improve performance?

# Performance Metrics



- ◆ You are expected to be familiar with:
  - ★ Major components of a computer,
  - ★ Flow of operations and control in a simple computer,
  - ★ Some performance metrics

# Performance Metrics



## ◆ Who has the fastest

- ★ The first Top500 list was created in 1993.
- ★ In 2012, Top500 list was dominated by IBM Blue Gene/Q with 4 systems in the top 10. The largest of which at Lawrence Livermore National Laboratory with more than 16 Petaflops sustained performance.

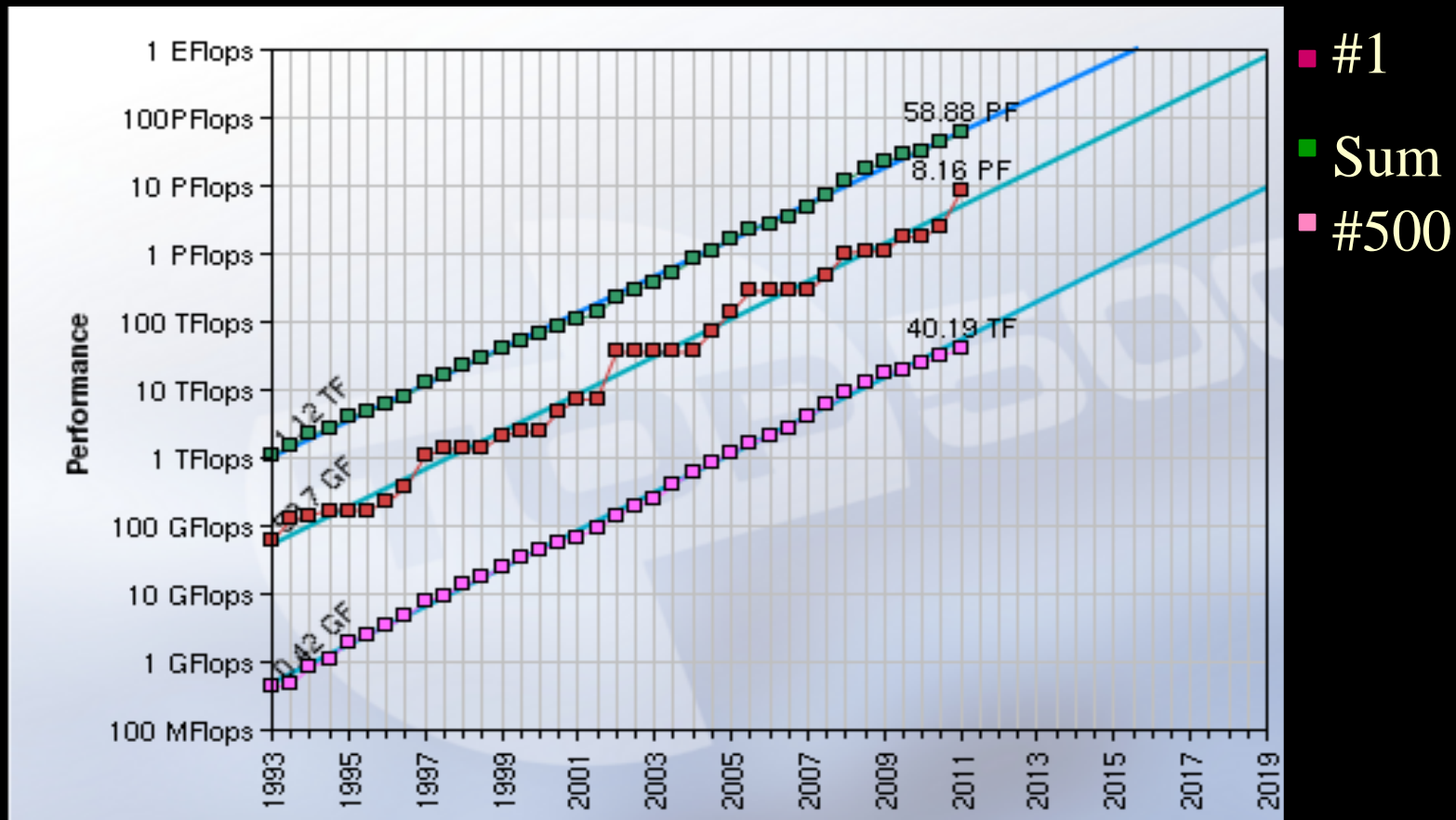
# Performance Metrics

## The Top10 supercomputers (as of 2011)

Rank	Site	Computer/Year Vendor	Country	Cores	R <sub>max</sub> (Pflops)	R <sub>peak</sub> (Pflops)	Power (MW)
1	RIKEN Advanced Institute for Computational Science	K computer, SPARC64 VIIIfx 2.0GHz,/ 2011 Fujitsu	Japan	548,352	8.162	8.774	9.899
2	National Supercomputing Center in Tianjin	Tianhe-1A - NUDT / 2010 NUDT	China	186,368	2.566	4.701	4.040
3	DOE/SC/Oak Ridge National Laboratory	Jaguar - Cray XT5-2.6 GHz / 2009 Cray Inc.	USA	224,162	1.759	2.331	6.951
4	National Supercomputing Centre in Shenzhen	Nebulae - Dawning TC3600 Blade/ 2010 Dawning	China	120,640	1.271	2.984	2.580
5	GSIC Center, Tokyo Institute of Technology	TSUBAME 2.0/2010 NEC/HP	Japan	73,278	1.192	2.288	1.399
6	DOE/NNSA/LANL/SNL	Cielo - Cray XE6 8-core 2.4 GHz /2011Cray Inc.	USA	142,272	1.110	1.365	3.980
7	NASA/Ames Research Center/NAS	Pleiades - 2.93 Ghz,/ 2011 SGI	USA	111,104	1.088	1.315	4.102
8	DOE/SC/LBNL/NERSC	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	USA	153,408	1.054	1.289	2.910
9	Commissariat a l'Energie Atomique (CEA)	Tera-100 - Bull bullx super-node S6010/S6030 / 2010 Bull SA	France	138,368	1.050	1.255	4.590
10	DOE/NNSA/LANL	Roadrunner - 3.2 Ghz /2009 IBM	USA	122,400	1.042	1.376	2.346

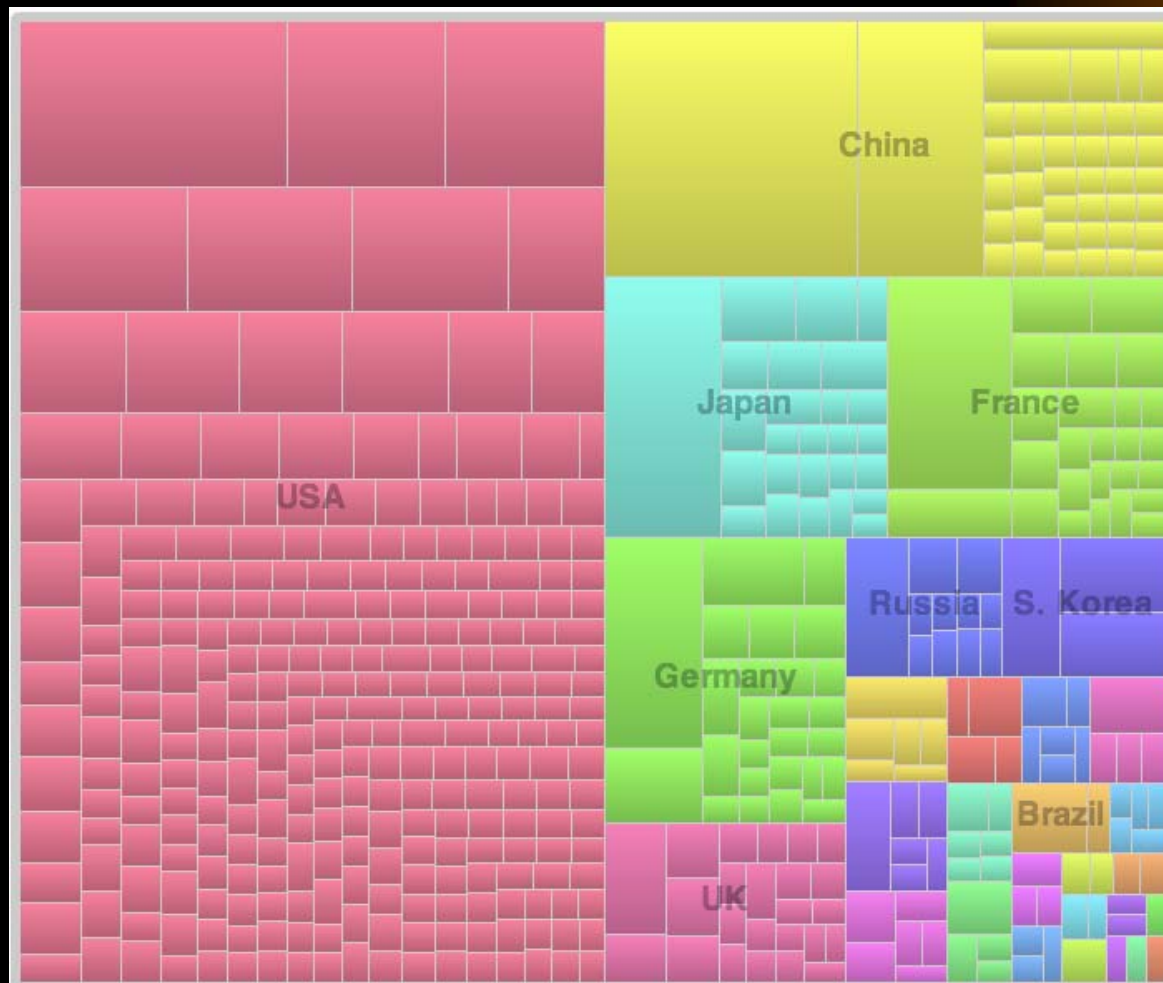
# Performance Metrics

Trend in Supercomputer technology (as of 2011)



# Performance Metrics

## Countries Share



### Absolute Counts

US: 274

China: 41

Germany: 26

Japan: 26

France: 26

UK: 25

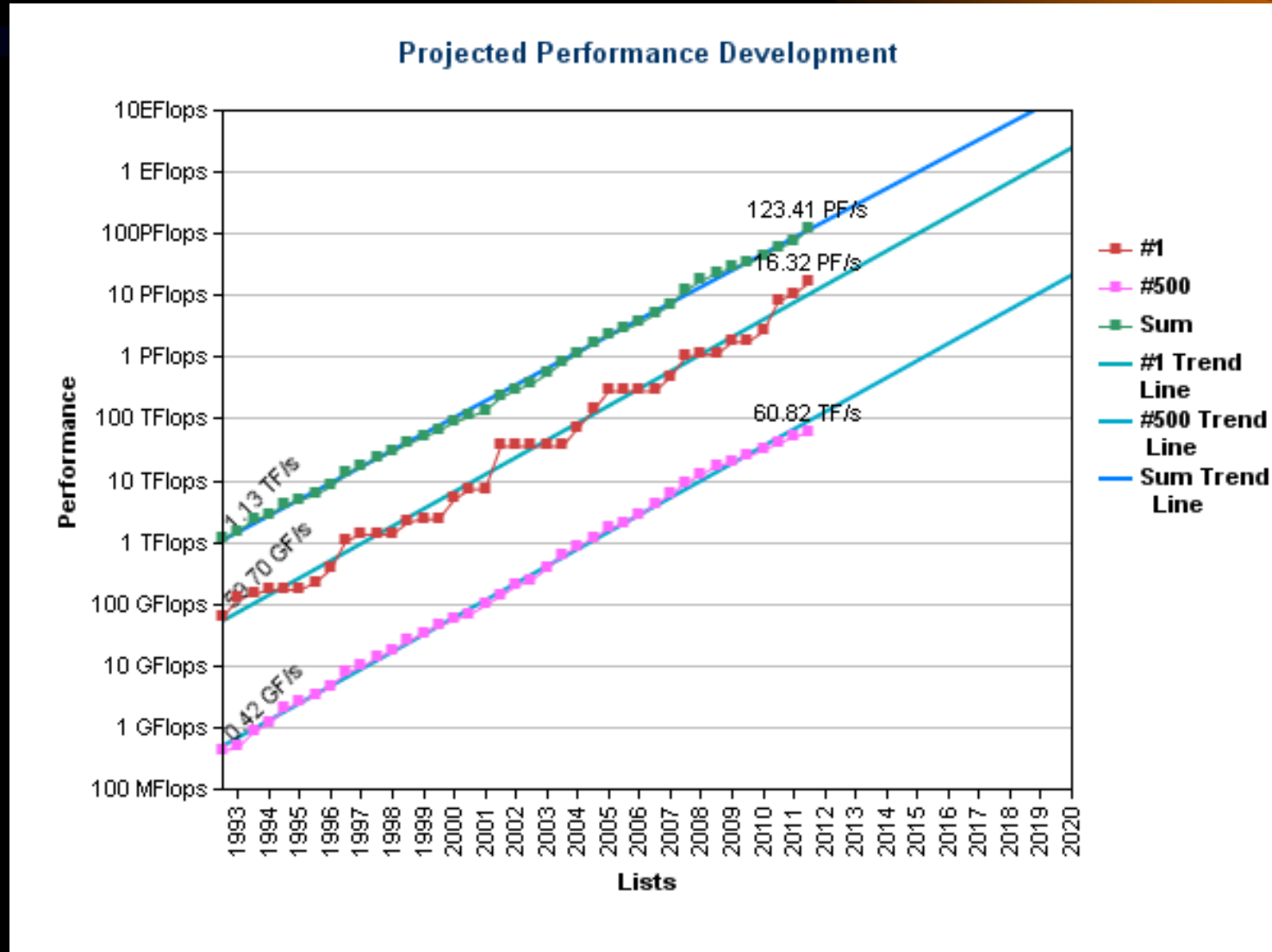
# Rapid change in Countries Share

Countries	Count	Share %	Rmax Sum (GF)	Rpeak Sum (GF)	Processor Sum
<a href="#">Australia</a>	6	1.20 %	400406	552142	40344
<a href="#">Austria</a>	2	0.40 %	188670	243386	26172
<a href="#">Belgium</a>	2	0.40 %	83840	151472	16704
<a href="#">Brazil</a>	2	0.40 %	269730	330445	37184
<a href="#">Canada</a>	8	1.60 %	640129	890598	82684
<a href="#">China</a>	61	12.20 %	7136315	14331013	881832
<a href="#">Denmark</a>	2	0.40 %	198408	260395	22218
<a href="#">Finland</a>	2	0.40 %	117858	180690	18640
<a href="#">France</a>	25	5.00 %	3180744	4100571	454928
<a href="#">Germany</a>	30	6.00 %	3242111	4181323	568952
<a href="#">India</a>	2	0.40 %	187910	242995	18128
<a href="#">Ireland</a>	1	0.20 %	40495	76608	7200
<a href="#">Israel</a>	2	0.40 %	135361	280436	23928
<a href="#">Italy</a>	5	1.00 %	471746	748248	42080
<a href="#">Japan</a>	26	5.20 %	11182236	13641290	832838
<a href="#">Korea, South</a>	4	0.80 %	950833	1126280	123384
<a href="#">Netherlands</a>	1	0.20 %	50924	64973	3456
<a href="#">Norway</a>	1	0.20 %	40590	51060	5550
<a href="#">Poland</a>	5	1.00 %	315075	448204	44274
<a href="#">Russia</a>	12	2.40 %	1341586	2290994	115120
<a href="#">Saudi Arabia</a>	4	0.80 %	359240	414841	81920
<a href="#">Singapore</a>	2	0.40 %	94073	144562	13192
<a href="#">Spain</a>	2	0.40 %	135860	197696	14160
<a href="#">Sweden</a>	5	1.00 %	489530	661642	75280
<a href="#">Switzerland</a>	4	0.80 %	317895	383373	49480
<a href="#">Taiwan</a>	2	0.40 %	220504	313570	32148
<a href="#">United Kingdom</a>	27	5.40 %	1872107	2806546	260572
<a href="#">United States</a>	255	51.00 %	25265849	36064596	3887556



# Performance Metrics

Trend in Supercomputer technology (as of June 2012)



# Performance Metrics



- ◆ Who has the fastest

- ◆ If the projection holds we can expect Exaflops system by 2019.

# Performance Metrics

## ◆ Performance Measures

- ★ **Speed up** — How much faster a task will run using the machine with enhancement relative to the original machine.

$$S = \frac{\text{Execution time on Original Machine}}{\text{Execution time on Enhanced Machine}}$$

# Performance Metrics

## ◆ Performance Measures

★ **Efficiency** — It is the ratio between speed up and number of processors involved in the process:

$$E_p = \frac{S_p}{p}$$

# Performance Metrics



## ◆ Performance Measures

- ★ Efficiency can be discussed, mainly, within the scope of concurrent system.
- ★ Efficiency indicates how effectively the hardware capability of a system has been used.
- ★ Assume we have a system that is a collection of ten similar processors. If a processor can execute a task in 10 seconds then ten processors, collectively, should execute the same task in 1 second. If not, then we can conclude that the system has not been used effectively.

# Performance Metrics



## ◆ Performance Measures

- ★ Green computing
- ★ Power consumption and power management

# Performance Metrics

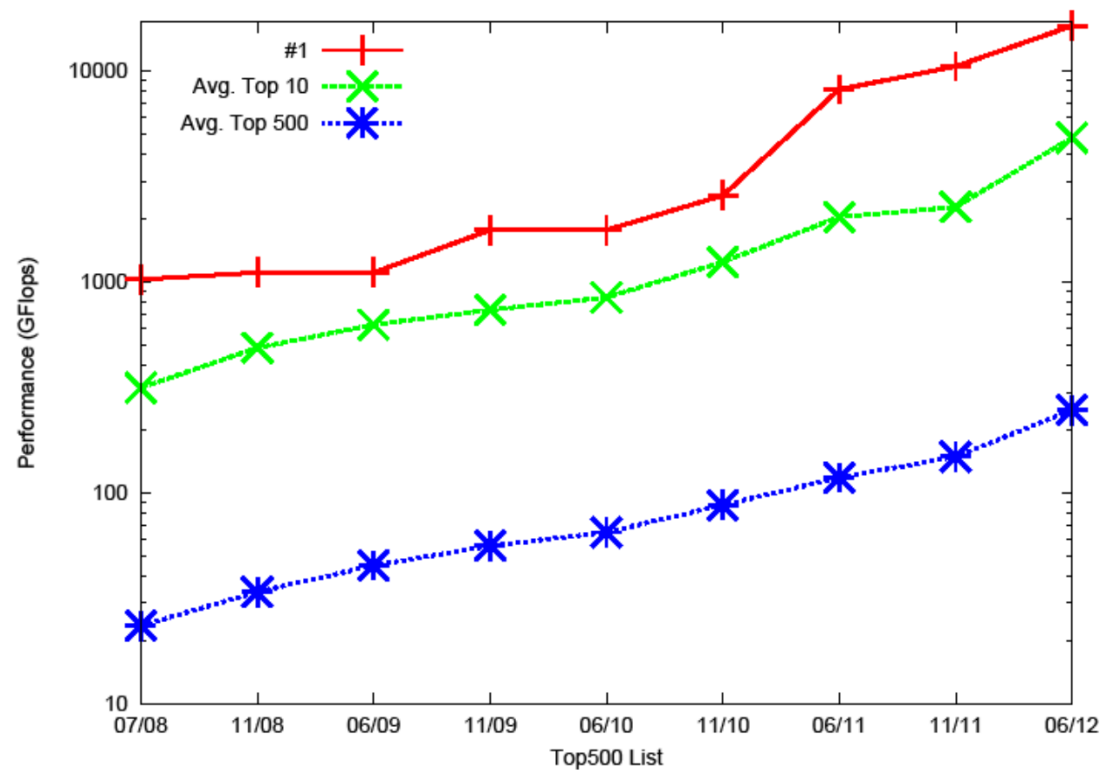


## ◆ Is one number enough?

- ★ As per our discussion, so far, performance was the major design constraint. However, the power is becoming a problem.
- ★ Power consumption became an issue with the growth of wireless technology and mobile devices. However, it is becoming even more of concern since feeding several Megawatt of power to run a supercomputer is not a trivial task and requires a great amount of supporting infrastructure

# Performance Metrics

◆ Is one number enough?

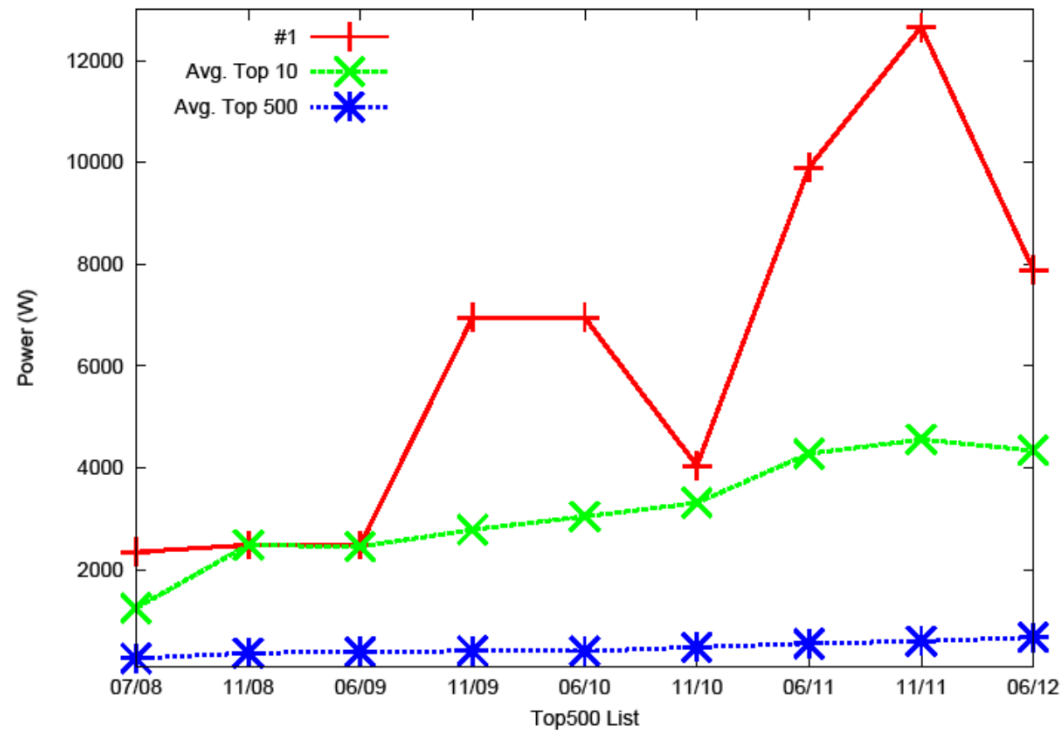


Top500 Performance



# Performance Metrics

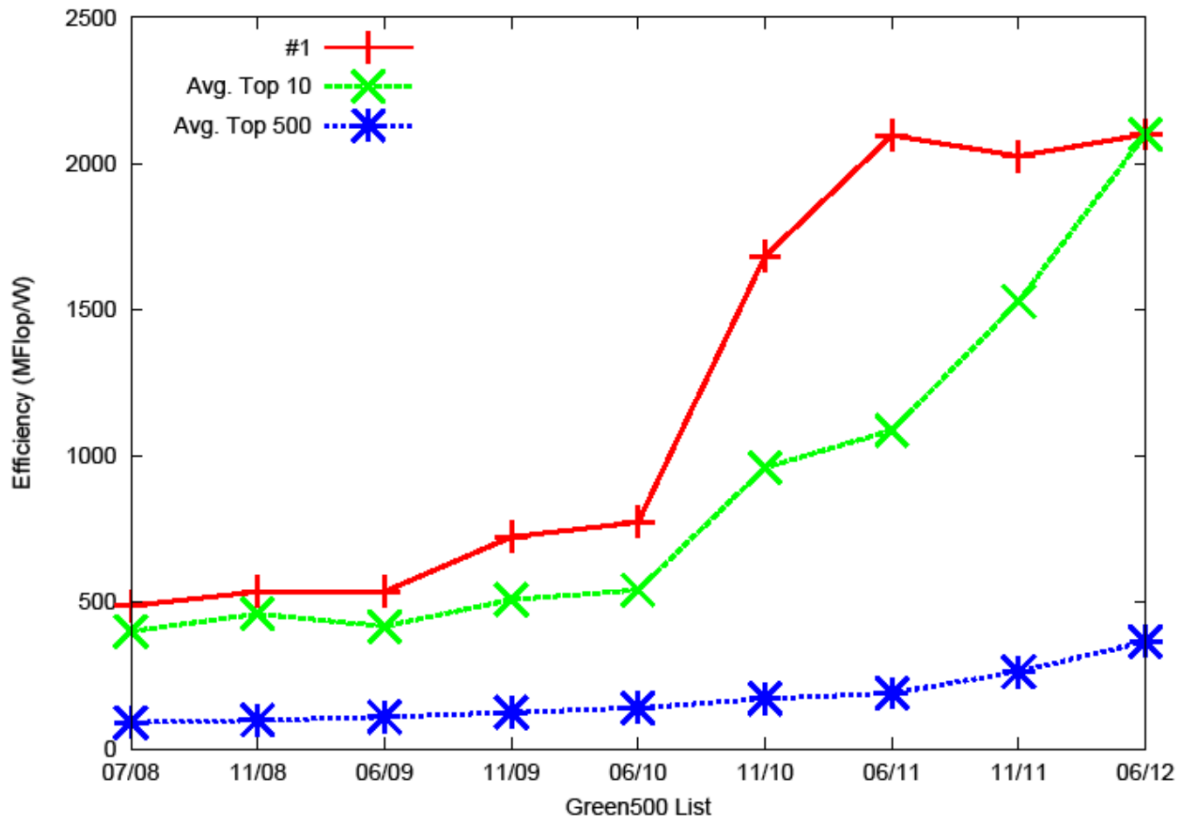
◆ Is one number enough?



Top500 Power

# Performance Metrics

◆ Is one number enough?



# Performance Metrics



- ◆ Challenges for creating Exaflops machine are:
  - ✦ Energy and Power,
  - ✦ Memory and Storage,
  - ✦ Concurrency and locality, and
  - ✦ Resiliency
- ◆ An Exaflops machine should consume at most 20 Megawatt of power which corresponds to 50 Gflop/W. To reach this goal, power efficiency needs to be increased by a factor of 25 compared to today's most power efficient system (IBM Blue Gene/Q)

# Performance Metrics



## ◆ Performance Measures

- ★ **Amdahl's law** — The performance improvement gained by improving some portion of an architecture is limited by the fraction of the time the improved portion is used — a small number of sequential operations can effectively limit the speed up of a parallel algorithm.

# Performance Metrics



## ◆ Performance Measures

- ★ **Amdahl's law** allows a quick way to calculate the speed up based on two factors — The fraction of the computation time in the original task that is affected by the enhancement, and, the improvement gained by the enhanced execution mode (speed up of the enhanced portion).

# Performance Metrics

## ◆ Performance Measures — Amdahl's law

$$Execution-time_{new} = Execution-time_{old} \times \left( (1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right)$$

$$Speedup_{overall} = \frac{Execution-time_{old}}{Execution-time_{new}} = \frac{1}{\left( (1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right)}$$

$$S \leq \frac{Execution\ time_{old}}{Execution\ time_{enhanced}} = \frac{1}{\mathbf{f} + (1 - \mathbf{f}) / \mathbf{p}}$$

Where  $f$  and  $p$  represent the unchanged portion and the speed up of the enhanced portion, respectively.

# Performance Metrics

## ◆ Performance Measures

- ★ Example — Suppose we are considering an enhancement that runs 10 times faster, but it is only usable 40% of time. What is the overall speed up?

$$S = \frac{1}{\left( .6 + \frac{.4}{10} \right)} = \frac{1}{.64} = 1.56$$

# Performance Metrics



## ◆ Performance Measures

- ★ Example — If 10% of operations, in a program, must be performed sequentially, then the maximum speed up gained is 10, no matter how many processors a parallel computer has.



# Performance Metrics



## ◆ Performance Measures

- ★ Example — Assume improving the CPU by a factor of 5 costs 5 times more. Also, assume that the CPU is used 50% of time and the cost of the CPU is  $\frac{1}{3}$  of the overall cost. Is it cost efficient to improve this CPU?

# Performance Metrics

## ◆ Performance Measures

$$S = \frac{1}{.5 + \frac{.5}{5}} = \frac{1}{.6} = 1.67$$

$$\text{cost of the new machine} = \frac{2}{3} \times 1 + \frac{1}{3} \times 5 = 2.33 \text{ times of the original machine}$$

# Performance Metrics



## ◆ Performance Measures

- ★ The **CPU time (T)** is the time needed to execute a given program, excluding the time waiting for I/O or running other programs.
- ★ **CPU time** is further divided into:
  - The **user CPU time** and
  - The **system CPU time**.

# Performance Metrics

## ◆ Performance Measures

$$\begin{aligned} \text{CPU Time} &= \text{CPU Clock cycles} * \text{Clock Cycle time} \\ \text{CPU Time} &= \frac{\text{CPU Clock cycles}}{\text{Clock Rate}} \end{aligned}$$

★ The CPU time is estimated as

$$T = I_c * CPI * \tau = \sum_{i=1}^n (CPI_i * I_i) * \tau$$

# Performance Metrics



## ◆ Performance Measures

- ★ Example — It takes 10 seconds to run a program on machine *A* that has a 400 MHz clock rate.
- ★ We are intended to build a faster machine that will run this program in 6 seconds. However, machine *B* requires 1.2 times as many clock cycles as machine *A* for this program. Calculate the clock rate of machine *B*:

# Performance Metrics

## ◆ Performance Measures

$$CpuTime_A = \frac{CPUClockCycle_A}{ClockRate_A}$$

$$10 = \frac{CPUClockCycle_A}{400 * 10^6 \text{ Cycles} / \text{Second}}$$

$$CPUClockCycle_A = 4000 * 10^6$$

$$CPUTime_B = \frac{1.2 * CPUClockCycle_A}{ClockRate_B}$$

$$ClockRate_B = \frac{1.2 * 4000 * 10^6}{6} = 800 \text{ MHz}$$

# Performance Metrics

## ◆ Performance Measures

- ★ Example — Two machines are assumed: In machine *A* conditional branch is performed by a compare instruction followed by a branch instruction. Machine *B* performs conditional branch as one instruction.
- ★ On both machines, conditional branch takes two clock cycles and the rest of the instructions take 1 clock cycle. 20% of instructions are conditional branches.
- ★ Finally, clock cycle time of *A* is 25% faster than *B*'s clock cycle time. Which machine is faster?

# Performance Metrics

## ◆ Performance Measures

$$CPI_A = .8*1 + .2*2 = 1.2$$

$$t_B = t_A * 1.25$$

$$CPU_A = I_{CA} * 1.2 * t_A$$

$$CPI_B = .25*2 + .75*1 = 1.25$$

$$CPU_B = .8I_{CA} * 1.25t_A * 1.25 = I_{CA} * 1.25 * t_A$$

★ So A is faster.



# Performance Metrics

## ◆ Performance Measures

- ★ Example — Now assume that cycle time of  $B$  can be made faster and now the difference between the cycle times is 10%. Which machine is faster?

$$\text{CPU}_A = I_{CA} * 1.2 * t_A$$

$$\text{CPU}_B = .8I_{CA} * 1.1t_A * 1.25 = I_{CA} * 1.1 * t_A$$

- ★ Now  $B$  is faster.

# Performance Metrics

## ◆ Performance Measures

- ★ The execution of an instruction requires going through the instruction cycle. This involves the instruction fetch, decode, operand(s) fetch, execution, and store result(s):

$$T = I_c * CPI * \tau = I_c * (p+m*k)* \tau$$

# Performance Metrics

## ◆ Performance Measures

★ The equation

$$T = I_c * CPI * \tau = I_c * (p+m*k) * \tau$$

is the major basis for this course. We will refer to this equation through out the course.

# Performance Metrics



## ◆ Performance Measures

- ★  $P$  is the number of processor cycles needed to **decode** and **execute** the instruction,  $m$  is the number of the **memory references** needed, and  $k$  is the ratio between memory cycle time and processor cycle time, **memory latency**.

# Performance Metrics

- ◆ With respect to the *CPU* time

$$T = I_c * CPI * \tau = I_c * (p+m*k) * \tau$$

in the following sections we will study two major issues:

- ★ Design and implementation of ALU in an attempt to reduce  $P$ ,
- ★ Design and implementation of memory hierarchy in an attempt to reduce  $m$  and  $k$ .

# Performance Metrics

## ◆ Question

- ★ With respect to our earlier definition of *CPU* time, discuss how the performance can be improved?

$$T = I_c * CPI * \tau = \sum_{i=1}^n (CPI_i * I_i) * \tau$$

# Performance Metrics

- ◆ In response to this question, the *CPU* time can be reduced by reducing the  $I_C$ , *CPI*, and/or  $\tau$ .
- ◆ Note the performance improvement with respect to the  $\tau$  due to the advances in technology is beyond the scope of this discussion.

# RISC vs. CISC



## ◆ Two Design philosophies

- ★  $I_C$  can be reduced by increasing the functionality of the system — increasing the instruction set by allowing hardware support for more complex instructions.
- ★ This design pattern results in the so-called **complex instruction set computer (CISC)**.



# RISC vs. CISC



## ◆ Two Design philosophies

- ★ *CPI* can be reduced by allowing hardware support for just simple instructions.
- ★ This design pattern results in the so-called **reduced instruction set computer (RISC)**.

# RISC vs. CISC



## ◆ Two Design philosophies

- ★ In an effort to improve the performance one design philosophy **suggest complexity** and the other **suggest simplicity!**

# RISC vs. CISC



## ◆ Complex Instruction Set Computer

- ★ The introduction of the IBM System/360 family was the beginning of modern computer technology — a series of computers with different levels of performance for different prices, all running identical software (Compatibility).
- ★ As noted before, this originated the distinction between **computer architecture** and **hardware**.

# RISC vs. CISC



## ◆ Complex Instruction Set Computer

- ★ Micro-programming was the primary technological innovation behind this new marketing concept — i.e., Computer Family.
- ★ Micro-programming relied on a small control memory and was an elegant way of building the processor control unit for a large instruction set.

# RISC vs. CISC



## ◆ Complex Instruction Set Computer

- ★ The main memory of these systems were **magnetic core** memories.
- ★ The small control memories were based on a technology about 10 times faster than core memory.
- ★ The rapid growth of semiconductor memory also influenced the implementation of micro-programming at the mini and micro computer levels.

# RISC vs. CISC



## ◆ Complex Instruction Set Computer

- ★ Due to the high cost and low performance of magnetic core memory, **memory efficiency** was the dominating concern in the previous metric parameters — execution speed was proportional to the program size.
- ★ This belief led to the invention of many instruction formats that reduced program size.

# RISC vs. CISC

## ◆ Complex Instruction Set Computer

★ The rapid growth of **integrated technology**, along with the belief that **execution time is proportional to the program size**, motivated the following design principles:

- Large control memory would add little or nothing to the cost of the machine.
- Moving software functions to micro code would result in faster computer and more reliable functions.

# RISC vs. CISC



## ◆ Complex Instruction Set Computer

- Architectural techniques that led to smaller programs also led to faster computers.
- **Stacks** or **memory-to-memory** architectures were superior execution models.



# RISC vs. CISC



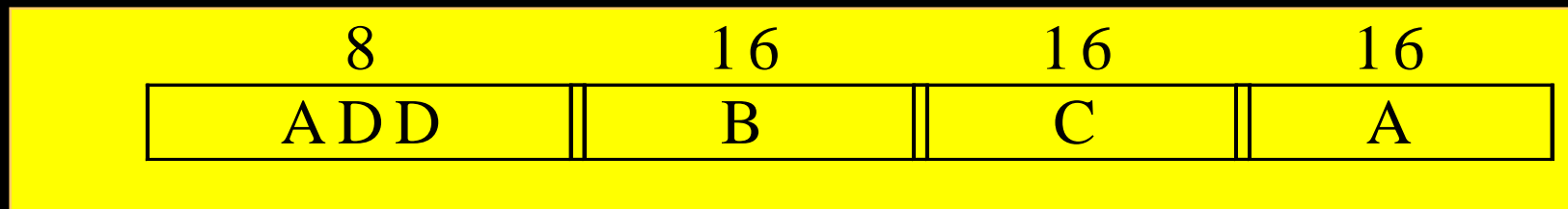
## ◆ Complex Instruction Set Computer

- ★ Let us look at the translation of  $A \Leftarrow B + C$  in three execution models:

# RISC vs. CISC

## ◆ Complex Instruction Set Computer

### ★ Memory-to-Memory organization



Instruction Length	56 bits
Data Size	96 bits (data words are 32 bits each)
Total Memory	152 bits

# RISC vs. CISC

- ◆ Complex Instruction Set Computer
  - ★ Register-to-Register Organization

8	4	16		
Load	$r_B$	B		
Load	$r_C$	C		
ADD	$r_A$	$r_B$	$r_C$	
Store	$r_A$	A		

Instruction Length    104 bits  
 Data Size                96 bits  
 Total Memory            200 bits

# RISC vs. CISC

## ◆ Complex Instruction Set Computer

### ★ Memory-to-Register Organization

8	16
Load	B
ADD	C
Store	A

Instruction Length    72 bits  
Data Size             96 bits  
Total Memory         168 bits

# RISC vs. CISC



## ◆ Complex Instruction Set Computer

★ In general, **Complex Instruction Set Computer (CISC)** supports:

- Relatively large instruction set containing some complex and time consuming instructions.
- Large number of addressing modes.
- Large number of instruction formats.

# RISC vs. CISC

## ◆ Complex Instruction Set Computer

### ★ VAX 11/780 Architectural Features

- It is a 32-bit machine.
- It has an instruction set of size 303.
- It supports different data types:
  - Integer: byte, word, long word, Quad word, octa word.
  - Floating point: 32-bit-8-bit exponent, 64-bit-8-bit exponent, 64-bit-11-bit exponent, 128-bit-15-bit exponent.
  - Packed decimal.
  - Character String.
  - Variable length bit field.

# RISC vs. CISC



## ◆ Complex Instruction Set Computer

### ★ VAX 11/780 Architectural Features

- It supports 16 different addressing modes.
- It supports several instruction formats:

op.code, {operand<sub>i</sub>}       $0 \leq i \leq 6$

# RISC vs. CISC

## ◆ Complex Instruction Set Computer

Characteristics of Some Computer Architectures				
	IBM 370/168	VAX 11/780	Dorado	iAPX-432
Year	1973	1978	1978	1982
Number of Instructions	208	303	270	222
Control Memory Size	420 Kbits	480 Kbits	136 Kbits	64 Kbits
Instruction Size	16-48	16-456	8-24	6-321
Technology	ECL MSI	TTL MSI	ECL MSI	NMOS VLSI
Execution Model	Register-Memory, Memory-Memory, Register-Register	Register-Memory, Memory-Memory, Register-Register	Stack	Stack, Memory-Memory
Cache Size	64 Kbits	64 Kbits	64 Kbits	0



# RISC vs. CISC

## ◆ Complex Instruction Set Computer

★ With the continuing growth of semiconductor memory, the architecture research community argued for **richer instruction sets**:

- Richer instruction sets would **simplify compilers**,
- Richer instruction sets would **alleviate the software crisis**,
- Richer instruction sets would **improve architectural quality**.

# RISC vs. CISC



## ◆ Complex Instruction Set Computer

- ★ To support a machine with a **complex instruction set** one is required to develop a **very complex and sophisticated control unit** to differentiate between the numerous options available in order to activate appropriate control signals.

# RISC vs. CISC



## ◆ Complex Instruction Set Computer

★ To summarize:

- **Slow access** to memory motivated an architectural design that reduced number of accesses to the main memory. This was achieved by supporting a large variety of instructions and addressing modes.
- **Complexity** of the **control unit** and **compatibility** of various architectures were supported through micro-programming.

# RISC vs. CISC



## ◆ Complex Instruction Set Computer

★ CISC Philosophy results in:

- **Complex Control Units:**

- Large design time,
- Higher probability of errors,
- Harder to locate and correct faults,
- Longer instruction cycle, and
- Longer clock cycle.

- **More complex compiler**

- **Lower hardware utilization** — creation of redundant features.

# RISC vs. CISC



## ◆ From CISC to RISC

- ★ As discussed before, computer architects were reaching some design principles (CISC), however, the implementation world was changing:
  - Semiconductor memory was replacing core memory — As a result, main memories would no longer be 10 times slower than control memories.
  - Control store ROMs were changing to control store RAMs — Since it was practically impossible to develop a large error free micro-program.

# RISC vs. CISC



## ◆ From CISC to RISC

- Cache memories were included in the architectures.
- Compilers found it difficult to help close the **semantic gap** — Attempts to close the **semantic gap** had actually introduced a **performance gap**.
- ★ As a result some computer architects got motivated to reevaluate the adapted architectural design principles.

# RISC vs. CISC

## ◆ Reduced Instruction Set Computer

- ★ Functions should be kept **simple** unless there is a very good reason to do otherwise.
- ★ Micro instructions should not be faster than simple instructions.
- ★ Simple **decoding** and **pipelined** execution are more important than program size.
- ★ **Compiler technology** should be used to simplify instructions rather than to generate complex instructions.

# RISC vs. CISC



## ◆ Reduced Instruction Set Computer

### ★ Functions should be kept simple

- The effective speed of a computer can be maximized by migrating all but the most frequently used functions into software.
- Included in hardware are only those performance features that are pointed to by dynamic studies of high level language programs.



# RISC vs. CISC



## ◆ Reduced Instruction Set Computer

### ★ Functions should be kept simple

- A resource is incorporated in the architecture only if its incorporation is justified by its frequency of use, and if its incorporation does not slow down other resources that are used more frequently.

# RISC vs. CISC



## ◆ Reduced Instruction Set Computer

- ★ Simplicity of the instruction set and addressing modes results in a small, fast and relatively easily to design decoder to analyze the instructions. This allows one to effectively develop an **instruction pipeline**.
- ★ This results in the execution of one instruction per pipeline pulse — *CPI* equal to 1.

# RISC vs. CISC



## ◆ Reduced Instruction Set Computer

### ★ Instruction Pipelining

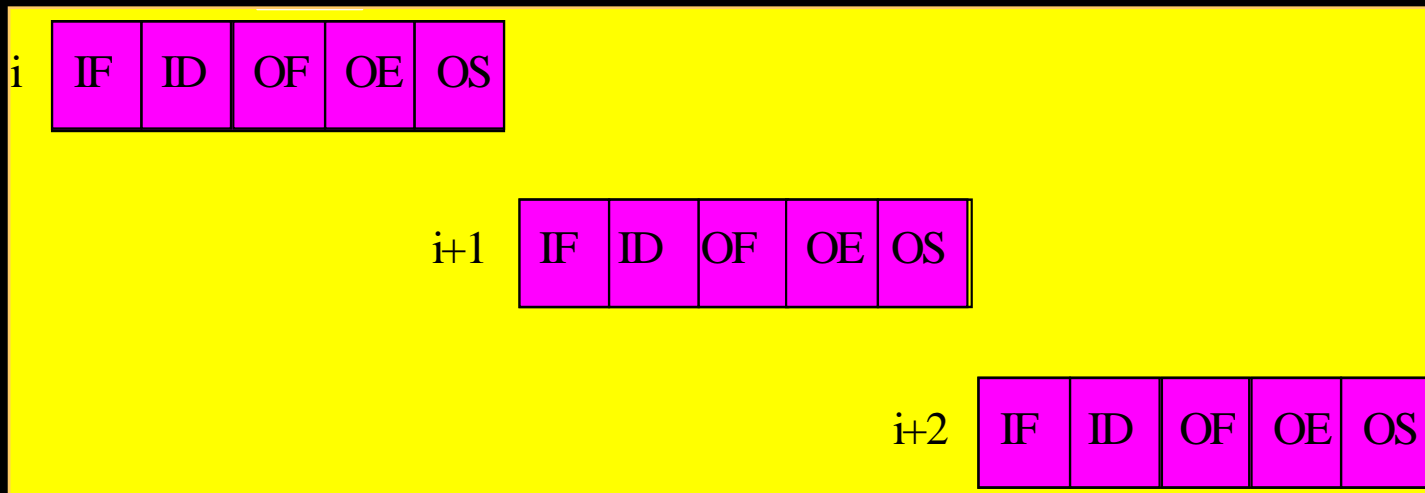
- Assume an instruction cycle can be partitioned into the following stages:

- **IF** : instruction fetch
- **OE** : operand execute
- **ID** : instruction decode
- **OS** : operand store
- **OF** : Operand fetch

# RISC vs. CISC

## ◆ Reduced Instruction Set Computer

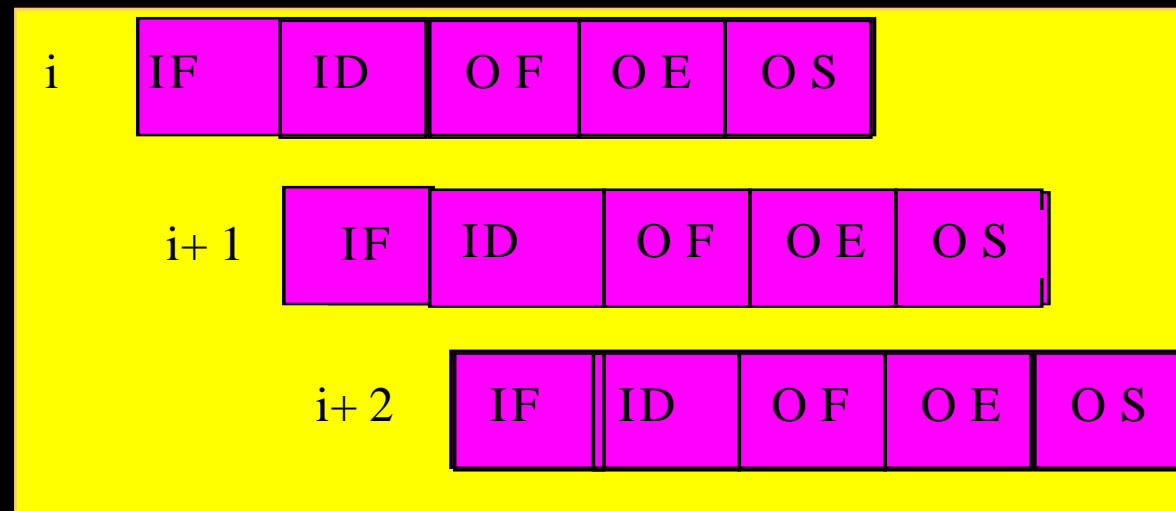
★ Non-pipelined instruction cycle:



# RISC vs. CISC

## ◆ Reduced Instruction Set Computer

★ Pipelined instruction cycle:



# RISC vs. CISC



## ◆ Reduced Instruction Set Computer

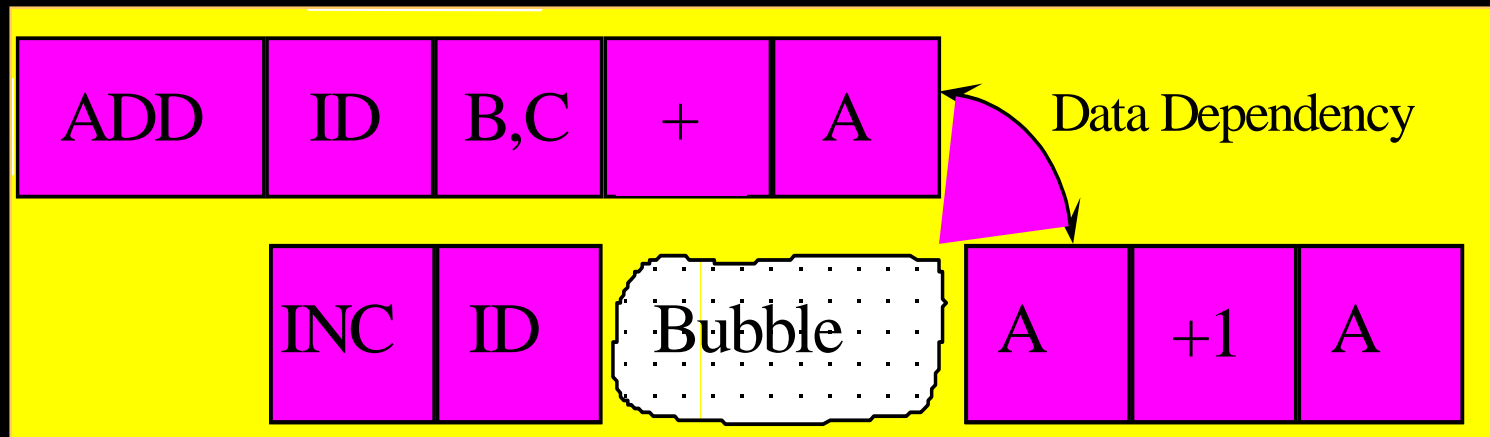
### ★ Pipelined instruction cycle:

- Naturally, instruction pipelining is not without its own problems. A concept known as **hazard** effects the performance of a pipeline organization.
- Related to our discussion, in the following two types of hazard will be discussed.

# RISC vs. CISC

## ◆ Reduced Instruction Set Computer

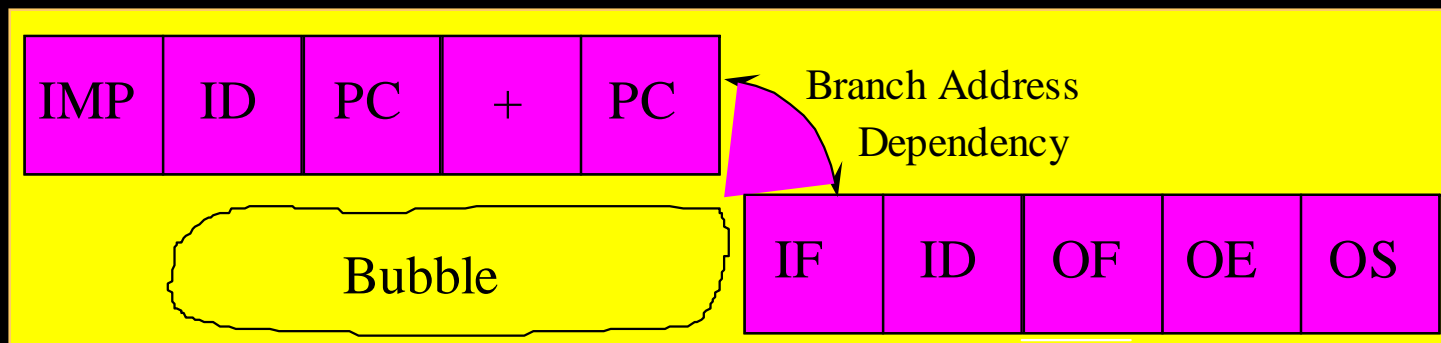
- ★ Pipelined instruction cycle — Pipelined with data interlock (Data Dependence Hazard)



# RISC vs. CISC

## ◆ Reduced Instruction Set Computer

- ★ Pipelined instruction cycle — Pipelined with branch interlock (Control Dependence Hazard)





# RISC vs. CISC



## ◆ Reduced Instruction Set Computer

- ★ **Delayed Branch** — To get the advantage of the instruction pipeline, it would be necessary to insert a NO-OP operation — in the worst case every branch would take several NO-OP instructions.
- ★ A better approach is to find some independent instructions and switch the order of the instructions in the program.

# RISC vs. CISC

## ◆ Reduced Instruction Set Computer

Address	Traditional Machine	Delayed Branch	Optimized Delayed Branch
100	Load X,A	Load X,A	Load X,A
101	ADD 1,A	ADD 1,A	JMP 105
102	JMP 105	JMP 106	ADD 1,A
103	ADD A,B	NO-OP	ADD A,B
104	SUB C,B	ADD A,B	SUB C,B
105	Store A,Z	SUB C,B	Store A,Z
106		Store A,Z	

★ The branch instruction is not data dependent on the *ADD* at address 101, so **switching** the *JMP* and *ADD* results an equivalent result.

# RISC vs. CISC



## ◆ Reduced Instruction Set Computer

### ★ Common RISC Features

- Operations are register-to-register with only LOAD and STORE instructions to access memory.
- The operations and addressing modes are reduced.
- Instruction formats are simple and do not cross word boundaries.
- RISC branches avoid pipeline penalties.

# RISC vs. CISC

## ◆ Reduced Instruction Set Computer

Architectural Features of Some Earlier RISC Machines			
	IBM 801	RISC1	MIPS
Year	1980	1982	1983
Number of Instructions	120	39	55
Control Memory Size	0	0	0
Instruction Length	32	32	32
Technology	ECL MSI	NMOS, VLSI	NMOS, VLSI
Execution Model	Register-Register	Register-Register	Register-Register

# RISC vs. CISC



## ◆ CISC Characteristics

- ★ Instruction set usually larger than 100,
- ★ Number of addressing modes supported is usually larger than 4,
- ★ Number of instruction formats supported is usually larger than 4,
- ★ Most instructions require multiple cycles for execution,

# RISC vs. CISC



## ◆ CISC Characteristics

- ★ Support of memory-to-memory model of execution,
- ★ Existence of special purpose registers,
- ★ Micro-programmed control unit, and
- ★ Machine instructions at a relatively high level, which is close to the level of high level language statements.

# RISC vs. CISC



## ◆ RISC Characteristics

- ★ Most instructions require single cycle for execution,
- ★ Memory is accessed just through LOAD and STORE instructions,
- ★ Hardwired control unit,
- ★ Supports relatively few instruction formats and addressing modes,

# RISC vs. CISC



## ◆ RISC Characteristics

- ★ Fixed instruction length format,
- ★ Highly pipelined instruction cycle,
- ★ Large number of on chip registers,
- ★ Instruction set is targeted for a specific application, and
- ★ Use of co-processor for complex operations requiring hardware support.



# RISC vs. CISC

## ◆ RISC vs. CISC

- ★ Back to our original question — based on the equation

$$T = I_c * CPI * \tau = I_c * (p+m*k) * \tau$$

what is a better design philosophy, RISC or CISC?

# RISC vs. CISC



## ◆ Reduced Instruction Set Computer

- ★ As noted before, a RISC concept in the best case would allow a CPI equal to 1. Is it possible to break this barrier?
- ★ Reducing the clock cycle time and hence increasing the frequency is one way to improve the performance. Address the shortcoming(s) of this approach.

# RISC vs. CISC



## ◆ Questions

- ★ True or False: shorter length instructions imply faster processor (why)?
- ★ Length of the operation code affects the length of the instructions. Define two schemes which allows one to reduce the length of the op-code.
- ★ Name and explain different factors which affect the length and format of an instruction.

# RISC vs. CISC



## ◆ Problem

- ★ Within the scope of RISC, use **delay branch** technique to reduce pipeline penalties (instruction format — op-code, destination, source1, source2): Justify your answer.
- ★ Assuming new value of PC is determined in ID stage and instructions in the block are independent of  $R_4$ ,  $R_5$ , and  $R_6$ , then we have:

# RISC vs. CISC

## ◆ Problem

### Sequence of Instructions Before

ADD R<sub>1</sub>, R<sub>2</sub>, R<sub>3</sub>  
IF R<sub>2</sub> = 0 Then  
[Redacted]  
←

SUB R<sub>4</sub>, R<sub>5</sub>, R<sub>6</sub>  
ADD R<sub>1</sub>, R<sub>2</sub>, R<sub>3</sub>  
IF R<sub>1</sub> = 0 Then  
[Redacted]  
←

ADD R<sub>1</sub>, R<sub>2</sub>, R<sub>3</sub>  
IF R<sub>1</sub> = 0 Then  
[Redacted]  
SUB R<sub>4</sub>, R<sub>5</sub>, R<sub>6</sub>  
←

### Sequence of Instructions After

IF R<sub>2</sub> = 0 Then  
ADD R<sub>1</sub>, R<sub>2</sub>, R<sub>3</sub>  
[Redacted]  
←

ADD R<sub>1</sub>, R<sub>2</sub>, R<sub>3</sub>  
IF R<sub>1</sub> = 0 Then  
SUB R<sub>4</sub>, R<sub>5</sub>, R<sub>6</sub>  
[Redacted]  
←

ADD R<sub>1</sub>, R<sub>2</sub>, R<sub>3</sub>  
IF R<sub>1</sub> = 0 Then  
SUB R<sub>4</sub>, R<sub>5</sub>, R<sub>6</sub>  
[Redacted]  
←