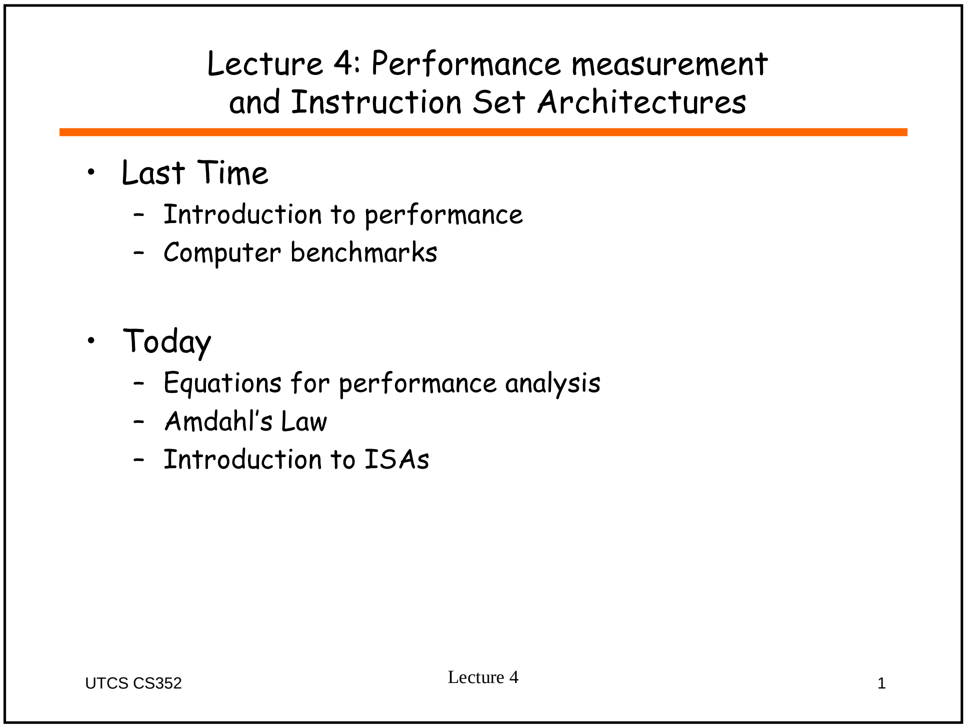
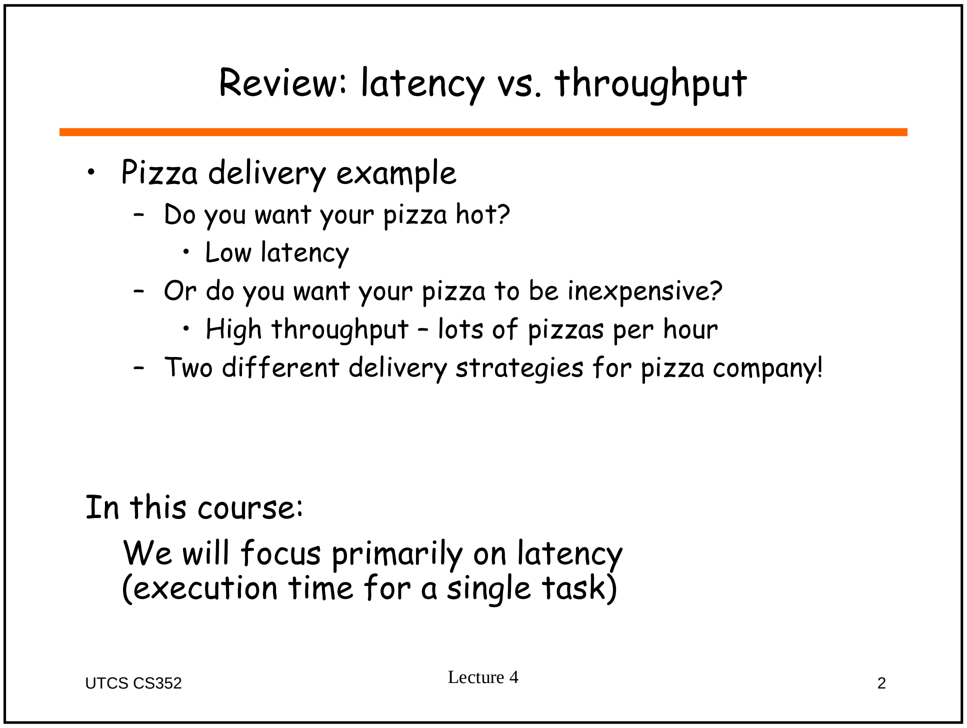


Lecture 4: Performance measurement and Instruction Set Architectures

- Last Time
  - Introduction to performance
  - Computer benchmarks

- Today
  - Equations for performance analysis
  - Amdahl's Law
  - Introduction to ISAs

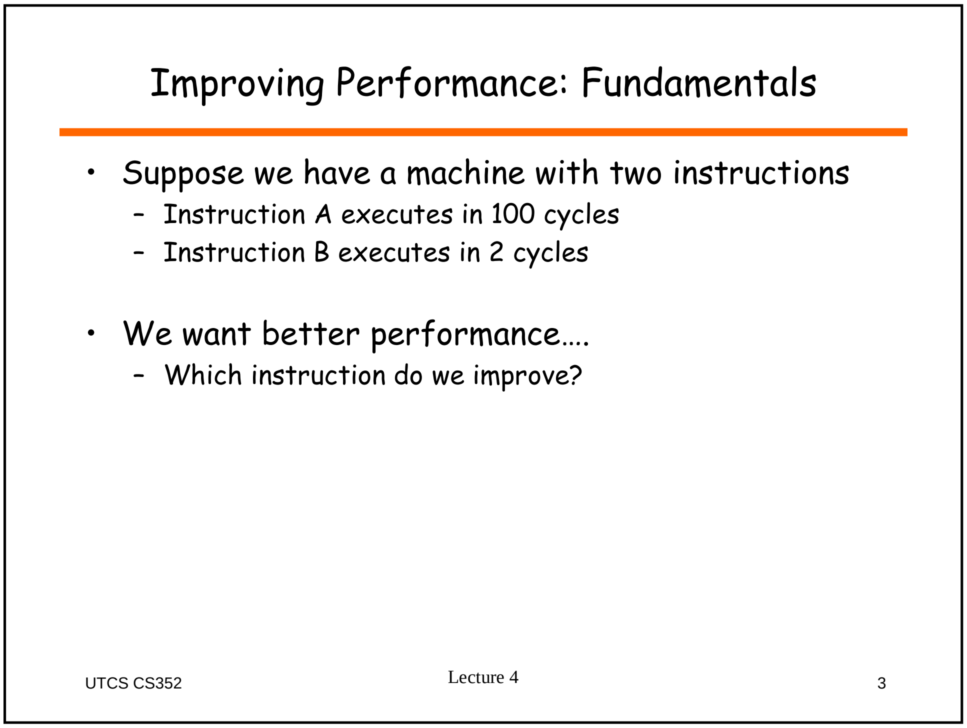


Review: latency vs. throughput

- Pizza delivery example
  - Do you want your pizza hot?
    - Low latency
  - Or do you want your pizza to be inexpensive?
    - High throughput – lots of pizzas per hour
  - Two different delivery strategies for pizza company!

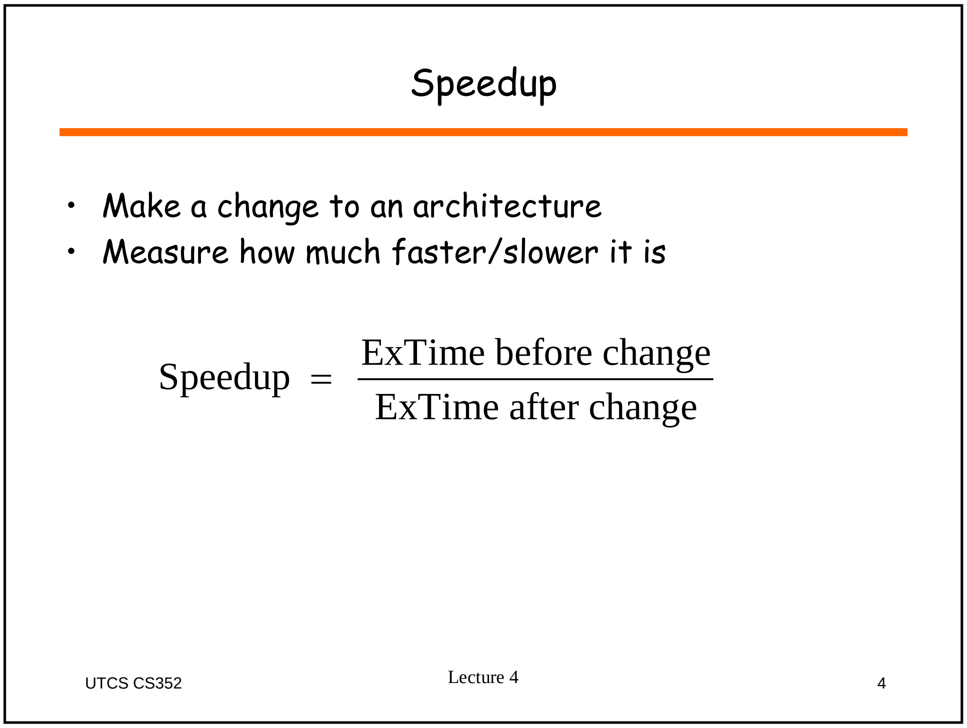
In this course:

We will focus primarily on latency (execution time for a single task)



Improving Performance: Fundamentals

- Suppose we have a machine with two instructions
  - Instruction A executes in 100 cycles
  - Instruction B executes in 2 cycles
- We want better performance….
  - Which instruction do we improve?



Speedup

- Make a change to an architecture
- Measure how much faster/slower it is

$$\text{Speedup} = \frac{\text{ExTime before change}}{\text{ExTime after change}}$$

Speedup when we know details about the change

- Performance improvements depend on:
  - how good is enhancement (factor **S**)
  - how often is it used (fraction **p**)
- Speedup due to enhancement E:

$$\mathrm{Speedup(E)} = \frac{\text{ExTime w/out E}}{\text{ExTime w/ E}} = \frac{\text{Perf w/ E}}{\text{Perf w/out E}}$$

$$ExTime_{new} = ExTime_{old} * \left[(1-p)+\frac{p}{S}\right]$$

$$Speedup(E) = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{(1-p)+\frac{p}{S}}$$

Example

- FP instructions improved by 2x
- But….only 10% of instructions are FP

$$ExTime_{new} = ExTime_{old} * \left(0.9+\frac{0.1}{2}\right) = 0.95 * ExTime_{old}$$

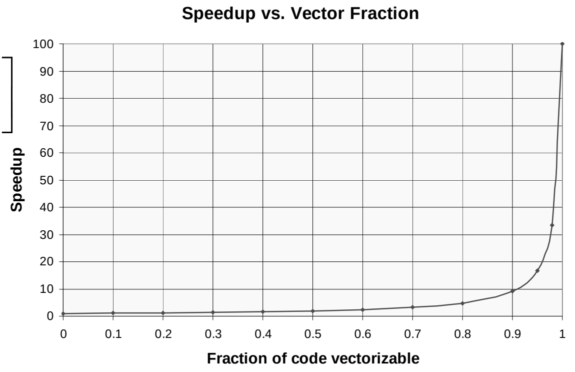
$$Speedup_{total} = \frac{1}{0.95} = 1.053$$

- Amdahl's Law:
  Speedup bounded by $\dfrac{1}{\text{fraction of time not enhanced}}$

Amdahl’s Law: Example 2

- Speed up vectorizable code by 100x

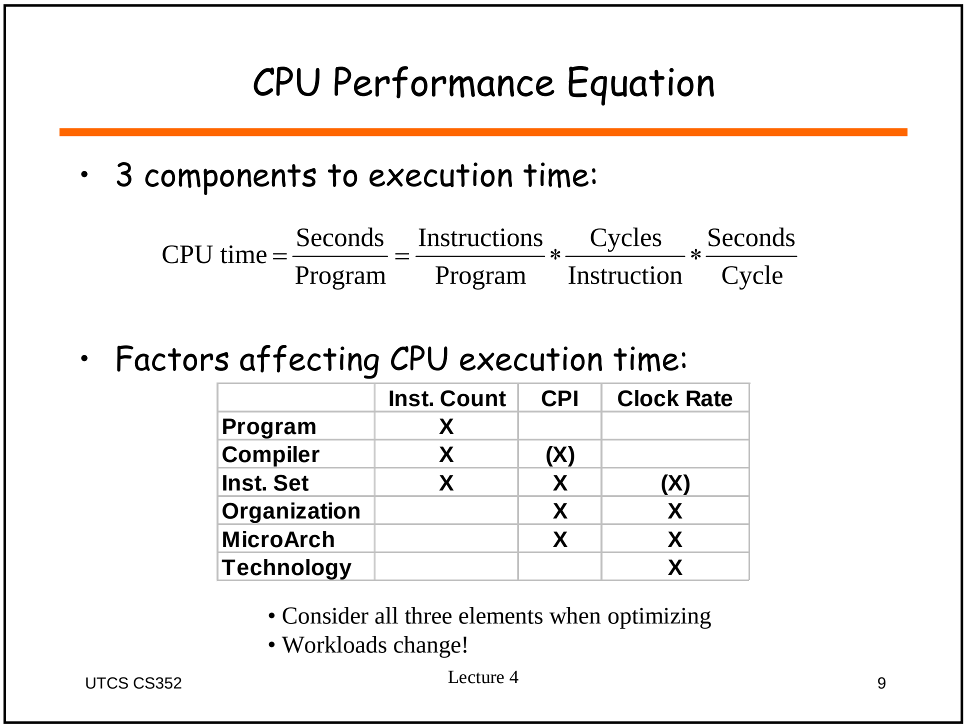
$$T_1 = T_0\left[(1-p)+\frac{p}{S}\right]$$



Amdahl’s Law: Summary message

- Make the common case fast

- Examples:
  - All instructions require instruction fetch, only fraction require data
    - ⇒ optimize instruction access first
  - Data locality (spatial, temporal), small memories faster
    - ⇒storage hierarchy: most frequent accesses to small, local memory



CPU Performance Equation

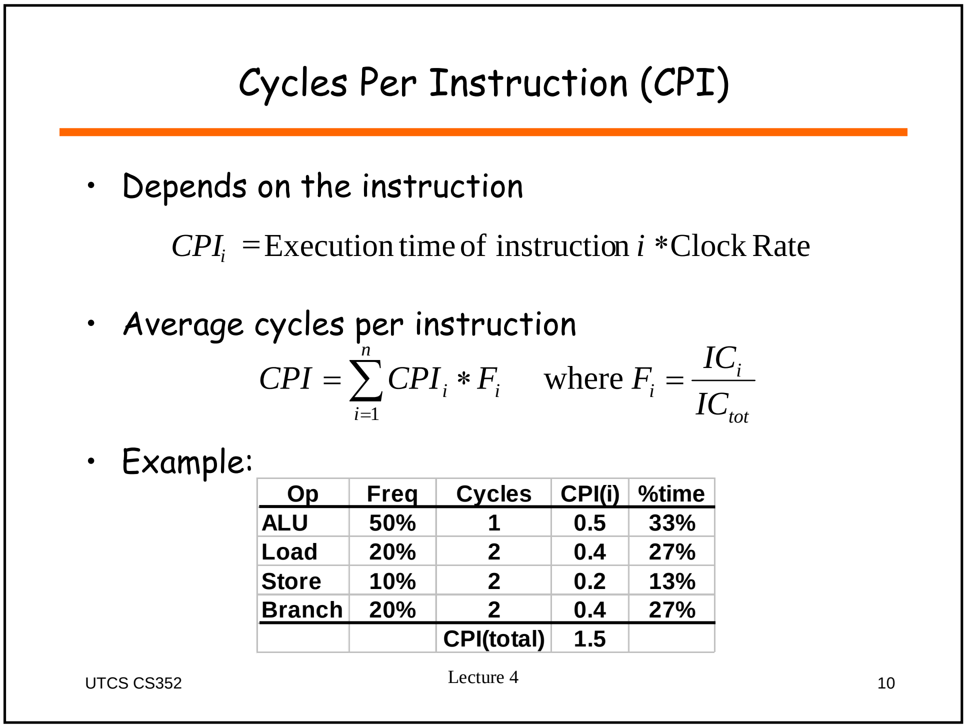
- 3 components to execution time:

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

- Factors affecting CPU execution time:

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

- Consider all three elements when optimizing
- Workloads change!



Cycles Per Instruction (CPI)

- Depends on the instruction

$$CPI_i = \text{Execution time of instruction } i * \text{Clock Rate}$$

- Average cycles per instruction

$$CPI = \sum_{i=1}^{n} CPI_i * F_i \quad \text{where } F_i = \frac{IC_i}{IC_{tot}}$$

- Example:

Op	Freq	Cycles	CPI(i)	%time
ALU	50%	1	0.5	33%
Load	20%	2	0.4	27%
Store	10%	2	0.2	13%
Branch	20%	2	0.4	27%
		CPI(total)	1.5	

Comparing and Summarizing Performance

- Fair way to summarize performance?
- Capture in a single number?

- Example: Which of the following machines is best?

	Computer A	Computer B	Computer C
Program 1	1	10	20
Program 2	1000	100	20
Total Time	1001	110	40

Means

Arithmetic mean $$\frac{1}{n}\sum_{i=1}^{n} T_i$$

Can be weighted: $a_i T_i$

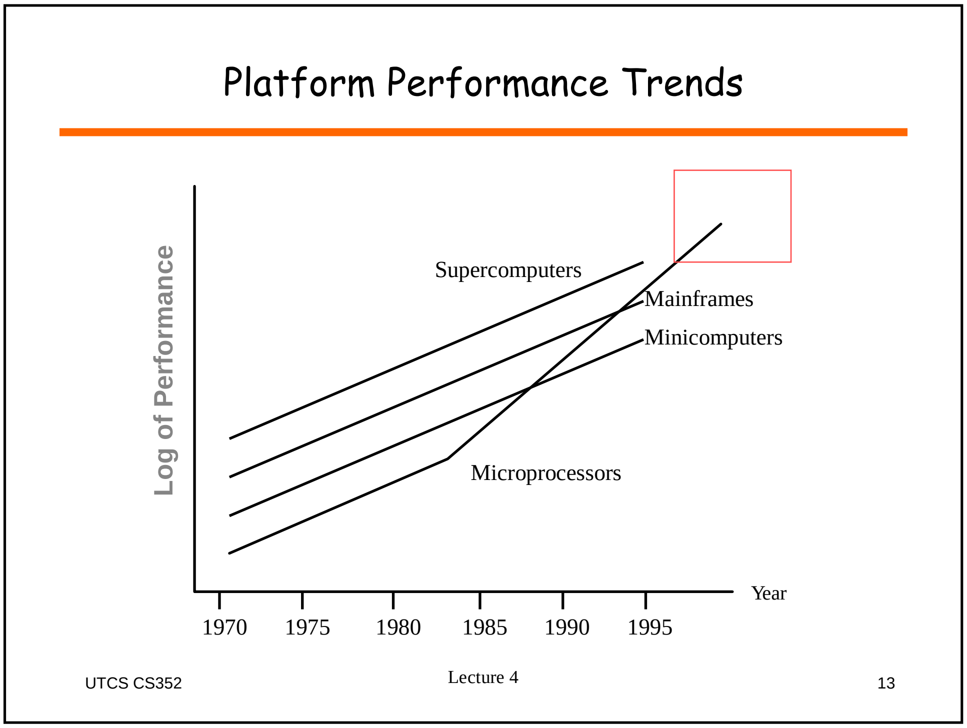
Represents total execution time

Harmonic mean $$\frac{n}{\sum_{i=1}^{n} \frac{1}{R_i}}$$

$R_i = 1/T_i$

Geometric mean $$\left(\prod_{i=1}^{n} \frac{T_i}{T_{ri}}\right)^{\frac{1}{n}}$$

Good for mean of ratios,
where the ratio is with respect to a reference.



Platform Performance Trends

Is Speed the Last Word in Performance?

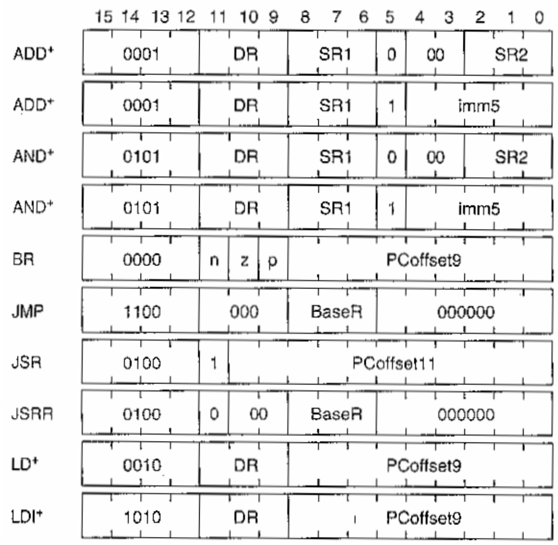
- Depends on the application!
- Cost
  - Not just processor, but other components (ie. memory)
- Power consumption
  - Trade power for performance in many applications
- Capacity
  - Many database applications are I/O bound and disk bandwidth is the precious commodity

Performance Measurement Summary

- Best benchmarks are real programs
  - Spec, TPC, Doom3
- Pitfalls still exist
  - Whole system measurement
  - Workload may not match user's
- Key concepts
  - Throughput and Latency
  - Equations: CPI, Amdahl's Law, …
- Next time
  - Instruction set architectures (ISA)
  - Read P&H 2.1 – 2.6

Instruction set architectures

LC-3 Instruction Set



	15 14 13 12	11 10 9	8 7 6	5	4 3	2 1 0
ADD+	0001	DR	SR1	0	00	SR2
ADD+	0001	DR	SR1	1	imm5	
AND+	0101	DR	SR1	0	00	SR2
AND+	0101	DR	SR1	1	imm5	
BR	0000	n z p	PCoffset9			
JMP	1100	000	BaseR	000000		
JSR	0100	1	PCoffset11			
JSRR	0100	0 00	BaseR	000000		
LD+	0010	DR	PCoffset9			
LDI+	1010	DR	PCoffset9			

⋮

LC-3 ISA – Continued

- Memory
  - 16-bit data, 16-bit addresses
- Addressing modes
  - register, immediate
  - PC+offset, base+offset, indirect PC+offset
- 8 registers
  - R7 updated on JSR, JSRR
- State
  - PC, registers, memory, condition code registers
- 16-bit instructions
  - Simple instruction set

Write programs using the ISA

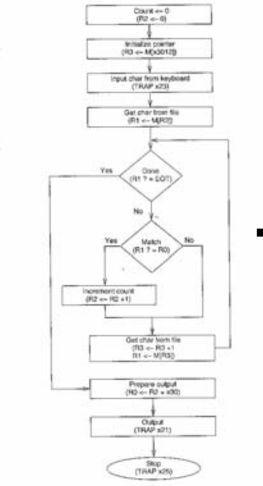
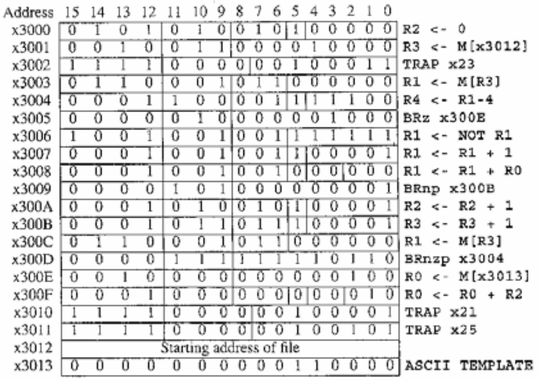


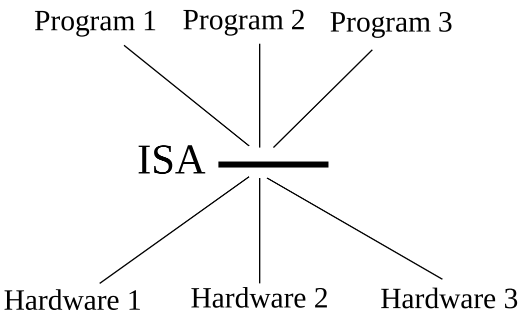
Figure 5.16 An algorithm to count occurrences of a character



Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
x3000	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	R2 <- 0
x3001	0	0	1	0	0	1	1	0	0	0	0	1	0	0	0	0	R3 <- M[x3012]
x3002	1	1	1	1	0	0	0	0	0	0	1	0	0	0	1	1	TRAP x23
x3003	0	1	1	0	0	0	1	0	1	1	0	0	0	0	0	0	R1 <- M[R3]
x3004	0	0	0	1	1	0	0	0	0	1	1	1	1	1	0	0	R4 <- R1-4
x3005	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	BRz x300E
x3006	1	0	0	1	0	0	1	0	0	1	1	1	1	1	1	1	R1 <- NOT R1
x3007	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	1	R1 <- R1 + 1
x3008	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	R1 <- R1 + R0
x3009	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	BRnp x300B
x300A	0	0	0	1	0	1	0	0	1	0	1	0	0	0	0	1	R2 <- R2 + 1
x300B	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	1	R3 <- R3 + 1
x300C	0	1	1	0	0	0	1	0	1	1	0	0	0	0	0	0	R1 <- M[R3]
x300D	0	0	0	0	1	1	1	1	1	1	1	1	0	1	1	0	BRnzp x3004
x300E	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	R0 <- M[x3013]
x300F	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	R0 <- R0 + R2
x3010	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	1	TRAP x21
x3011	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	TRAP x25
x3012	Starting address of file																
x3013	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	ASCII TEMPLATE

Figure 5.17 A machine language program that implements the algorithm of Figure 5.16

ISA is an interface (abstraction layer)



Program 1 Program 2 Program 3

ISA

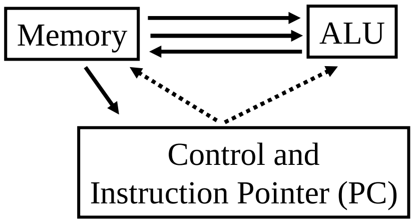
Hardware 1 Hardware 2 Hardware 3

Instruction Set Architecture is a Contract

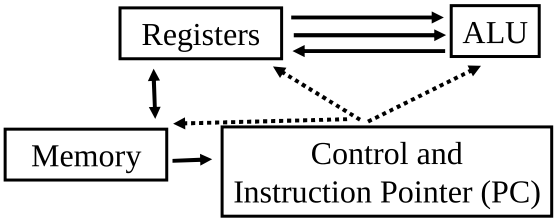
- **Contract** between programmer and the hardware
  - Defines visible state of the system
  - Defines how state changes in response to instructions
- Programmer:
  - ISA is model of how a program will execute
- Hardware Designer:
  - ISA is formal definition of the correct way to execute a program
- ISA specification
  - The binary encodings of the instruction set
  - How instructions modify state of the machine

ISA includes a model of the machine

A very simple model….

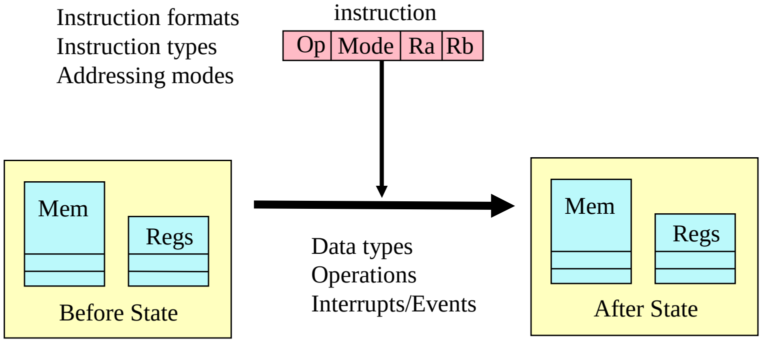


A more typical ISA machine model



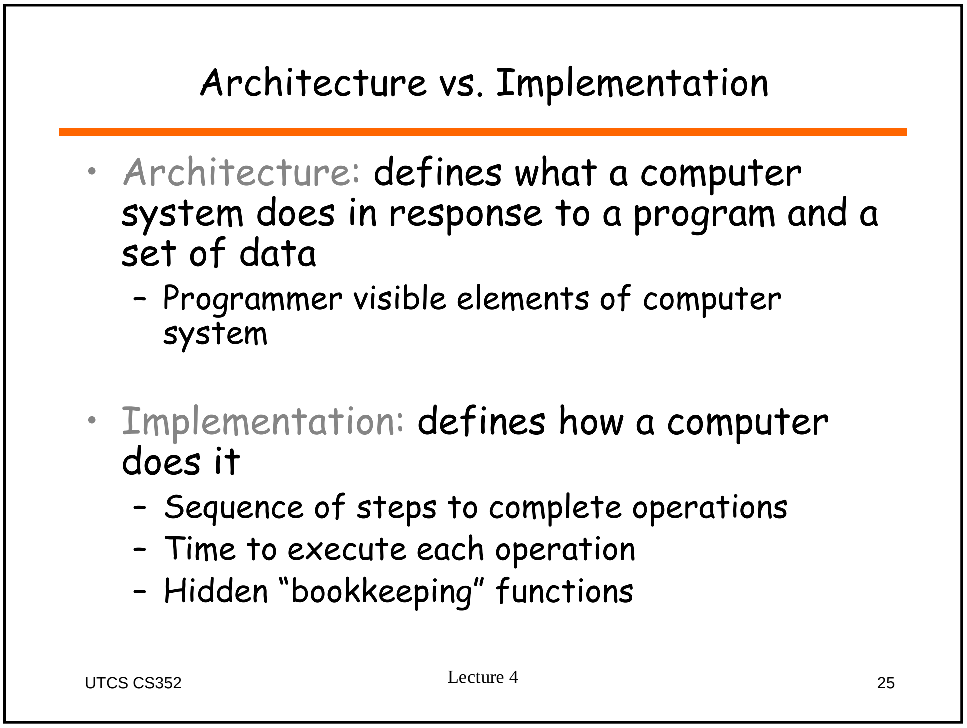
ISA Basics

Instruction formats
Instruction types
Addressing modes



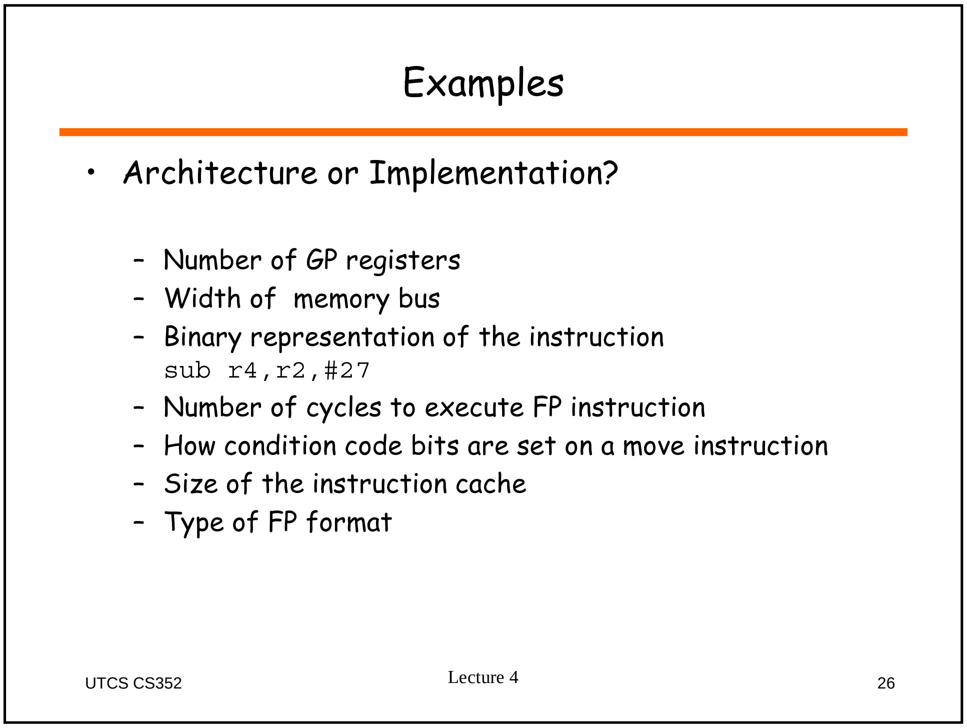
Data types
Operations
Interrupts/Events

Machine state
Memory organization
Register organization



Architecture vs. Implementation

- Architecture: defines what a computer system does in response to a program and a set of data
  - Programmer visible elements of computer system
- Implementation: defines how a computer does it
  - Sequence of steps to complete operations
  - Time to execute each operation
  - Hidden "bookkeeping" functions

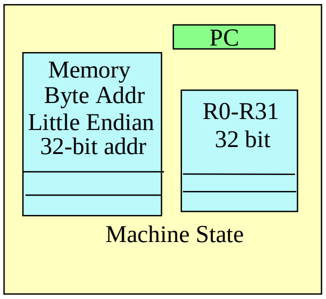


Examples

- Architecture or Implementation?
  - Number of GP registers
  - Width of memory bus
  - Binary representation of the instruction `sub r4,r2,#27`
  - Number of cycles to execute FP instruction
  - How condition code bits are set on a move instruction
  - Size of the instruction cache
  - Type of FP format

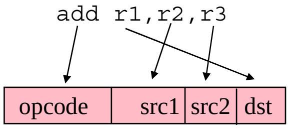
Machine State

- Registers
  - Size/Type
    - Program Counter (= IP)
    - accumulators
    - index registers
    - general registers
    - control registers
- Memory
  - Visible hierarchy (if any)
  - Addressibility
    - byte, word, bit
    - byte order (endian-ness)
    - maximum size
  - protection/relocation



Components of Instructions

- Operations (opcodes)
- Number of operands
- Operand specifiers



- Instruction encodings
- Instruction classes
  - ALU ops (add, sub, shift)
  - Branch (beq, bne, etc.)
  - Memory (ld/st)

Operand Number

• No Operands	HALT NOP	
• 1 operand	NOT R4 JMP _L1	R4 ⇐ R4
• 2 operands	ADD R1, R2 LDI R3, #1234	R1 ⇐ R1 + R2
• 3 operands	ADD R1, R2, R3	R1 ⇐ R2 + R3
• > 3 operands	MADD R4,R1,R2,R3	R4 ⇐R1+(R2*R3)

Effect of Operand Number

```
E = (C+D)*(C-D)
```

Assign

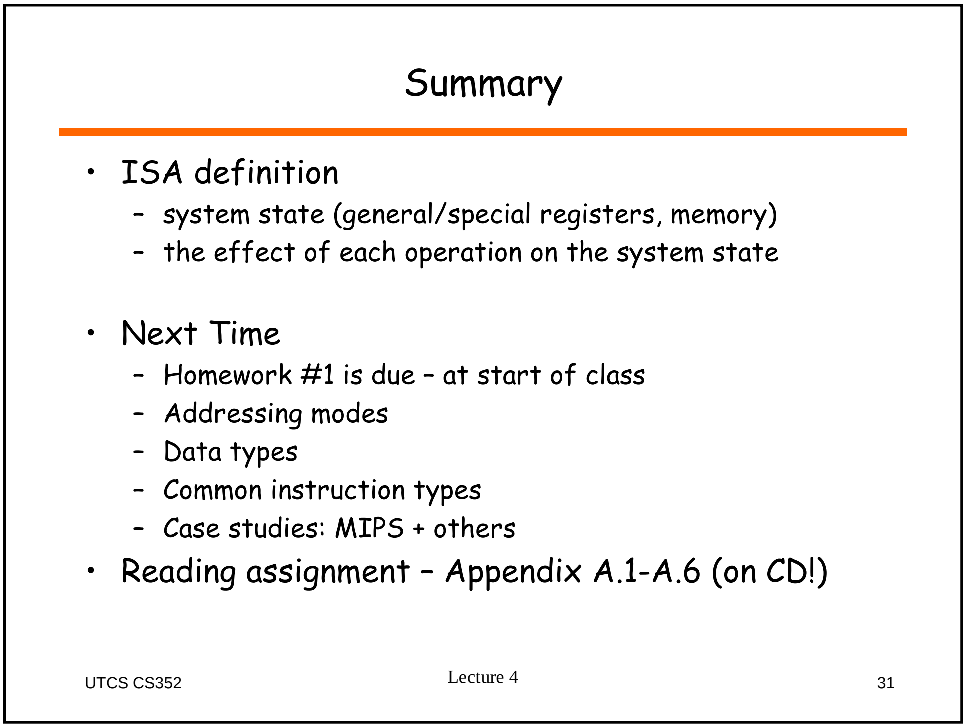
```
C ⇒ r1
D ⇒ r2
E ⇒ r3
```

3 operand machine

```
add  r3,r1,r2
sub  r4,r1,r2
mult r3,r4,r3
```

2 operand machine

```
mov  r3,r1
add  r3,r2
sub  r2,r1
mult r3,r2
```



Summary

- ISA definition
  - system state (general/special registers, memory)
  - the effect of each operation on the system state
- Next Time
  - Homework #1 is due – at start of class
  - Addressing modes
  - Data types
  - Common instruction types
  - Case studies: MIPS + others
- Reading assignment – Appendix A.1-A.6 (on CD!)