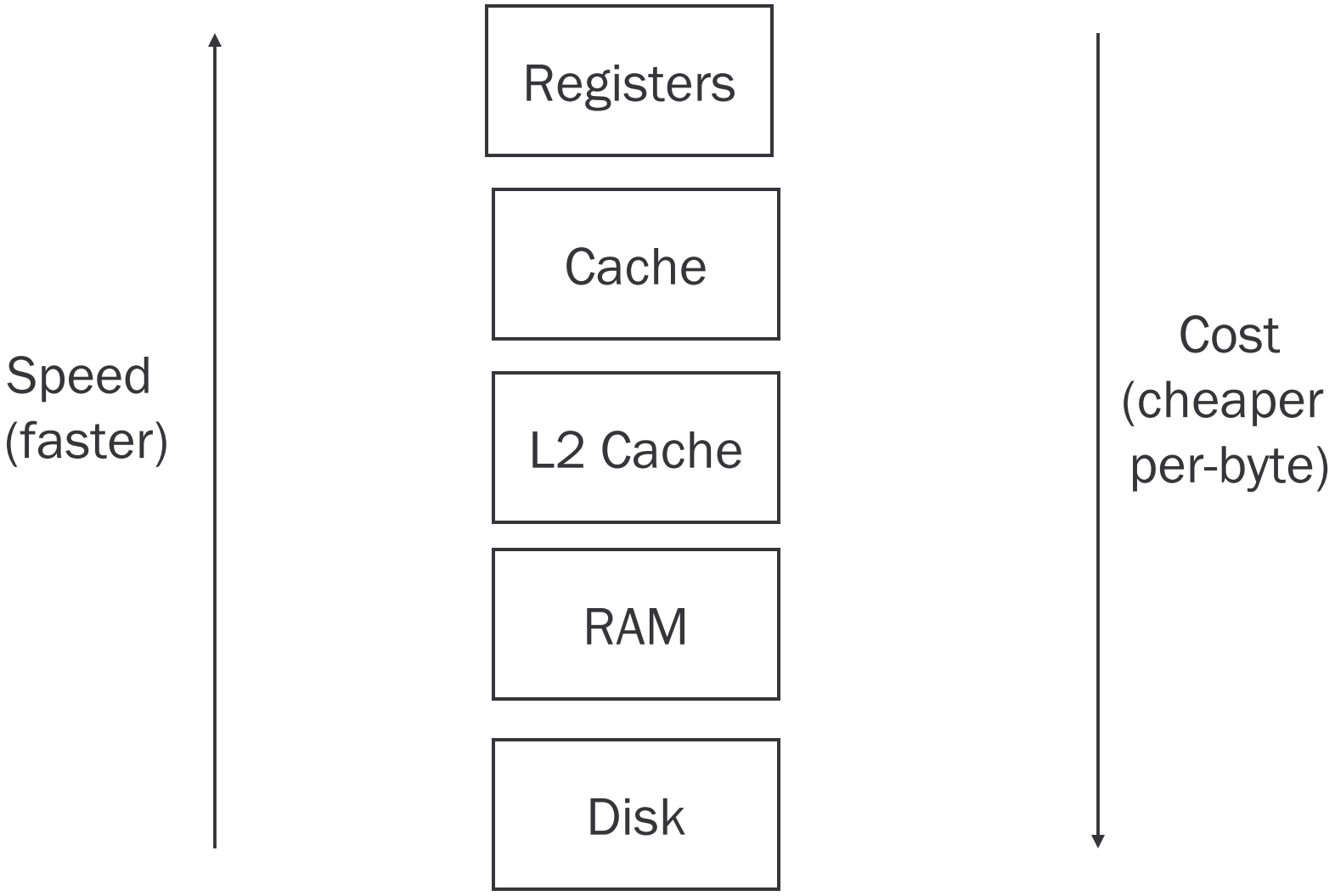
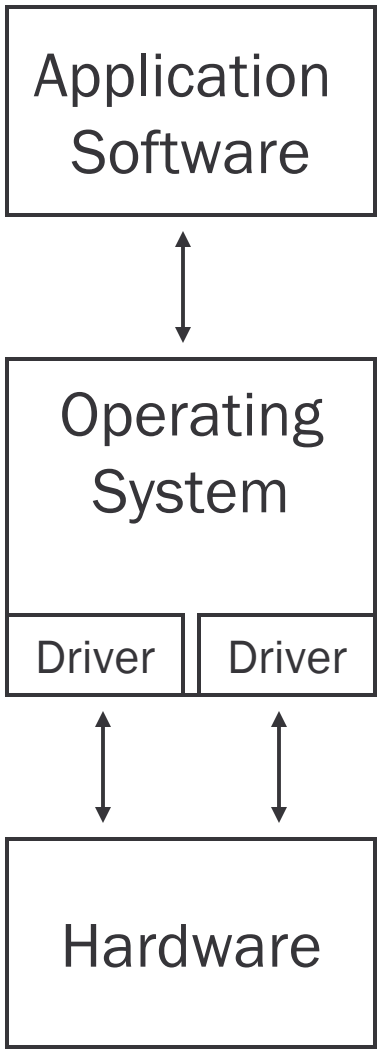
Computer Architecture

Memory Hierarchy



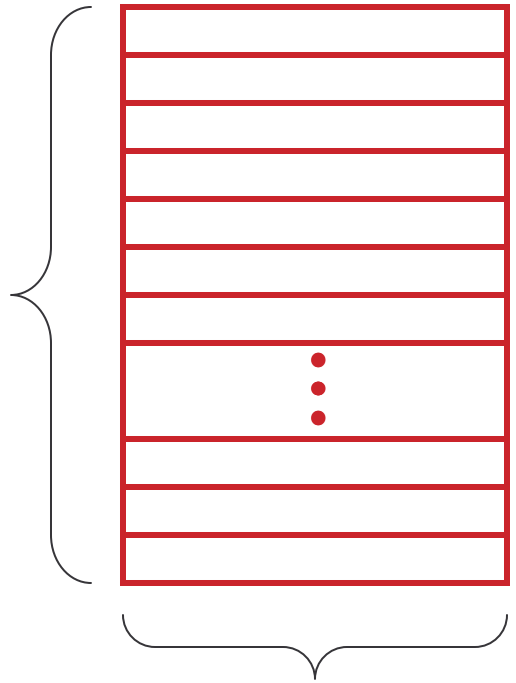
View of Computer System



Memory

To build a memory -- a logical $k \times m$ array of stored bits.

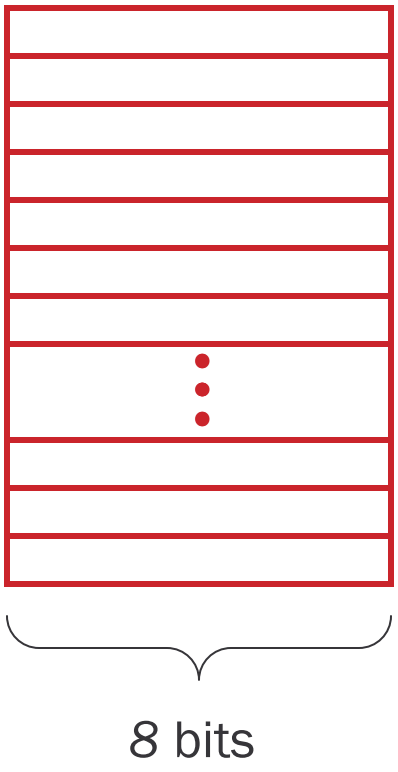
Address Space:
number of locations
(usually a power of 2)



Addressability:
number of bits per location
(e.g., byte-addressable)

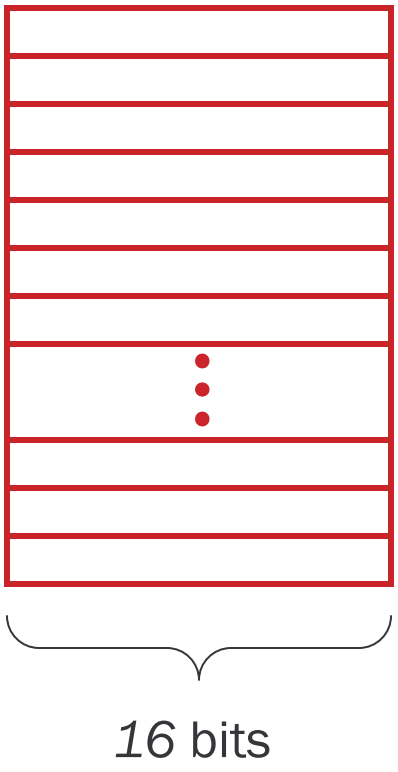
Memory

Example: Memory addresses if m=8



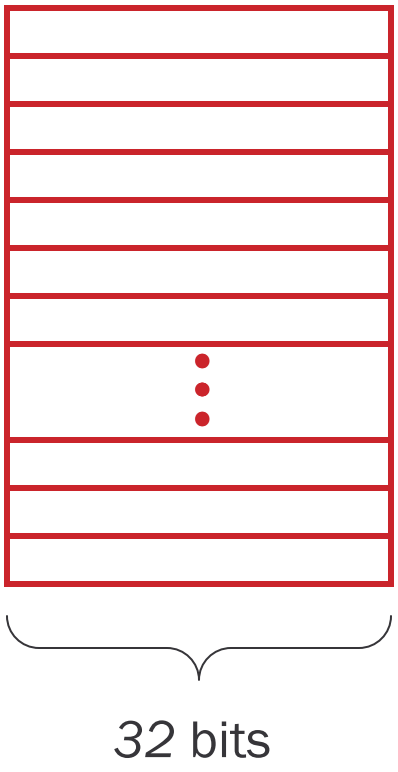
Memory

Example: Memory addresses if m=16 (byte addressable)



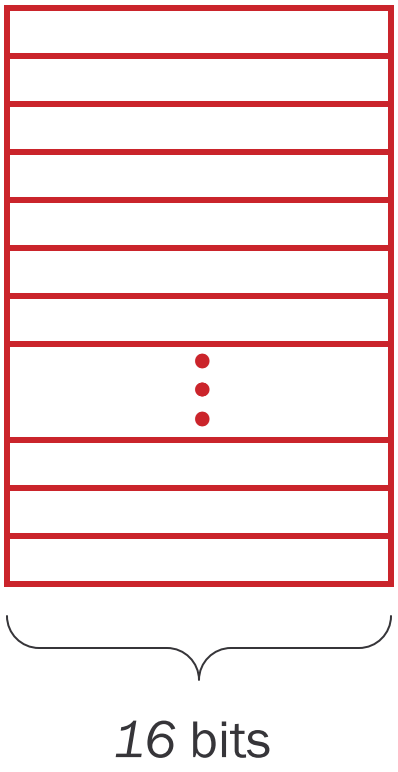
Memory

Example: Memory addresses if m=32 (byte addressable)

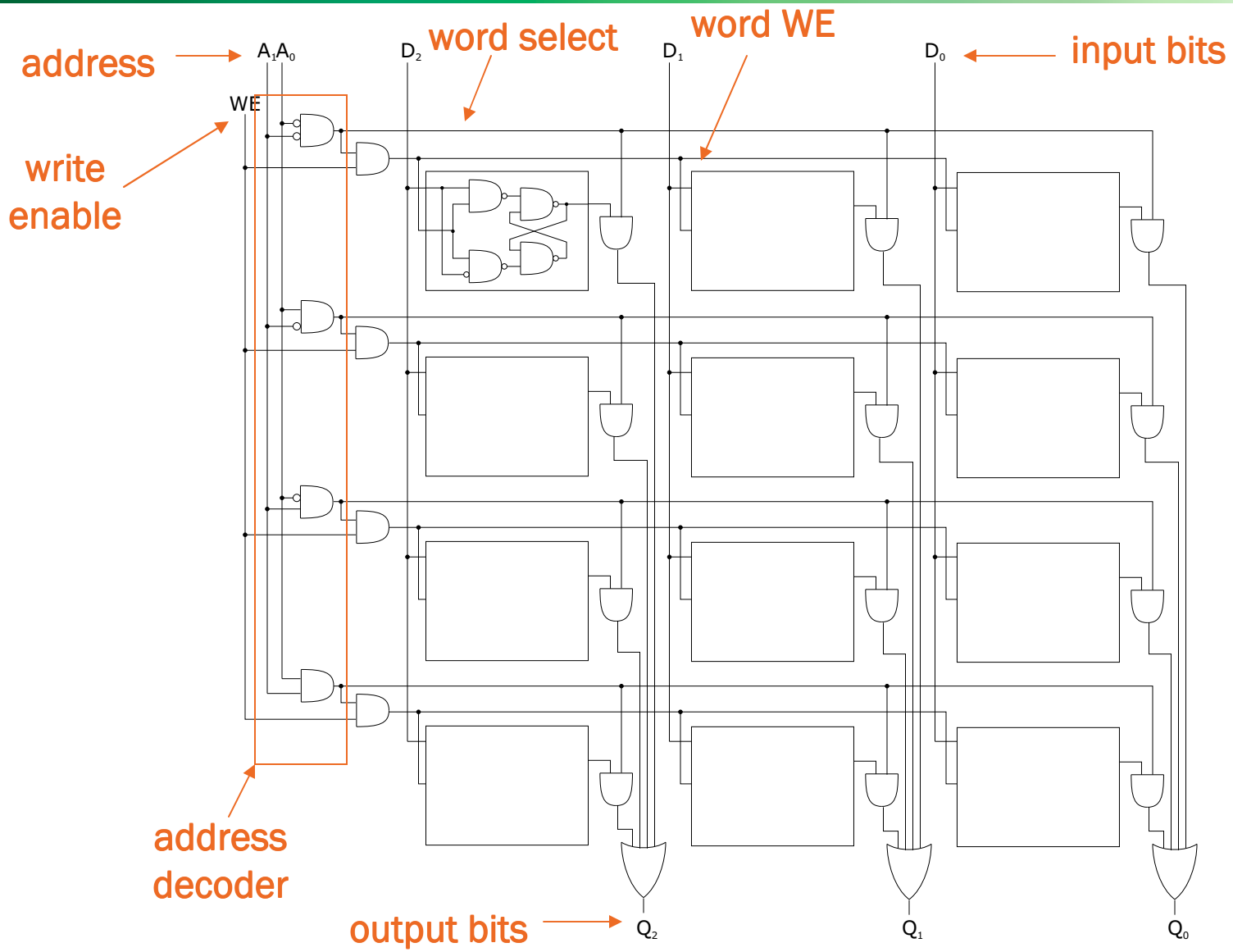


Memory

Example: Memory addresses if m=16 (location addressable) This is the model that LC-3 uses.



2^2 x 3 Memory


address
A1A0
D2
word select
D1
word WE
D0
input bits
WE
write
enable
address
decoder
output bits
Q2
Q1
Q0

More Memory Details

This is a not the way actual memory is implemented.

- fewer transistors, much more dense, relies on electrical properties

But the logical structure is very similar.

- address decoder
- word select line
- word write enable

Two basic kinds of memory (RAM = Random Access Memory)

Static RAM (SRAM)

- fast, maintains data without power

Dynamic RAM (DRAM)

- slower but denser, bit storage must be periodically refreshed

Even More Memory Details

There are other types of “non-volatile” memory devices:

- ROM
- PROM
- EPROM
- EEPROM
- Flash

Can you think of other memory devices?

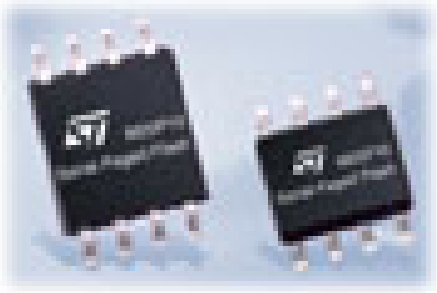
Electronics Packaging

- There are several packaging technologies available that an engineer can use to create electronic devices.
- Some are suitable for inexpensive toys but not miniature consumer products, and some are suitable for miniature consumer products but not inexpensive toys.
- These packages have metal leads that are the conductive wire that connect electricity from the outside world to the silicon inside the package.
- Leads between packages are connected with small copper traces on a printed circuit board (PCB), and the package leads are soldered to the PCB.

Examples of Electronics Packages

Dual In-line Package (DIP) Older technology, requires the metal leads to go through a hole in the printed circuit board.

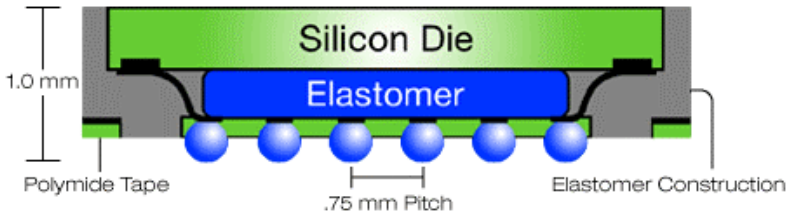
Dual Flat Pack (DFP) - A fairly recent technology, metal leads solder to the surface of the printed circuit board.



Examples of Electronics Packages

Quad Flat Pack (QFP) - like the Dual Flat Pack, except here are metal leads are on four sides.

Ball Grid Array (BGA) - The connections to the component are on the bottom of the chip, and have balls of solder on these connections.



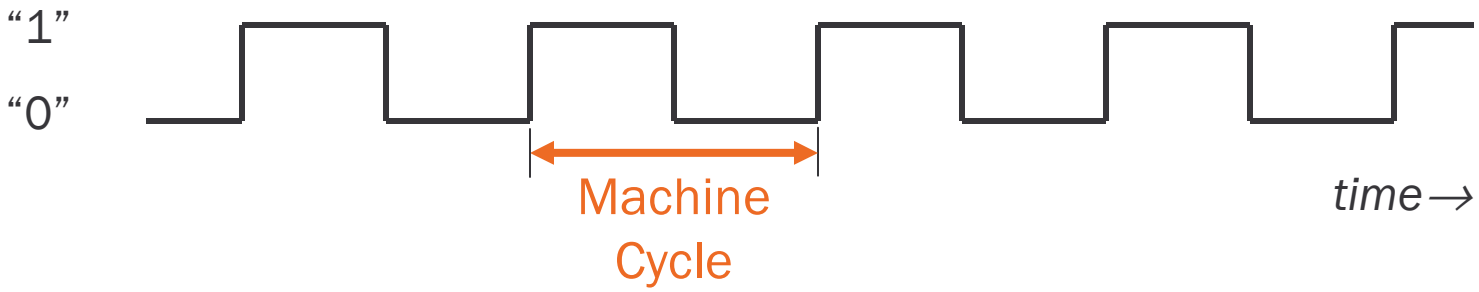
Driving Force: The Clock

The clock is a signal that keeps the control unit moving.

- At each clock "tick," control unit moves to the next machine cycle -- may be next instruction or next phase of current instruction.

Clock generator circuit:

- Based on crystal oscillator
- Generates regular sequence of "0" and "1" logic levels
- Clock cycle (or machine cycle) -- rising edge to rising edge

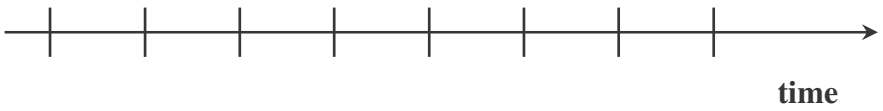


Clock Cycles

Instead of reporting execution time in seconds, we often use cycles

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

Clock "ticks" indicate when to start activities (one abstraction):



cycle time = time between ticks = seconds per cycle

clock rate (frequency) = cycles per second (1 Hz. = 1 cycle/sec)

Some Definitions

CPI = Cycles Per Instruction

CT = Cycle Time

IC = Instruction Count

CC = Clock Cycle Count

For example, a Pentium has a 233 MHz clock

2.33×10^8 clock cycles per second (MHz = 10^6)

CT = 1/clock rate

= 1/ 2.33×10^8 clock cycles/second

= 4.3×10^{-9} seconds/clock cycle

= 4.3 ns/clock cycle

More Practice

My 80486 computer runs at 66MHz. What is the cycle time?

A computer has a 2.5 ns cycle time. What is the number of cycles per second?

Run time definitions

$$\text{CPU time} = \text{clock cycle count} \times \text{Cycle Time} = CC \times CT$$

$$= \frac{\text{clock cycle count}}{\text{clock rate (Mhz)}}$$

$$\text{Cycles per instruction} = CPI = \frac{\text{clock cycle count}}{\text{Instruction Count}} = \frac{CC}{IC}$$

$$\Rightarrow CC = CPI \times IC$$

$$\therefore \boxed{\text{CPU time} = IC \times CPI \times CT}$$

"Newton's law" of microarchitecture

To improve CPU time (same as run time):

- Decrease Instruction Count (IC)
  - Good compiler
- Decrease CPI (increase "IPC", AKA *inst. level parallelism*)
  - Fancy hardware, good compiler
- Decrease CT
  - Crack designers

Examples and Practice

Program A takes 3 x10^{10} clock cycles to execute. How long does this take to run on a 100 MHz?

CPU time = CC x CT

= 3 x 10^{10} clock cycles x

10 x 10^{-9} clock cycles/second

= 3 x 10^{2} = 300 seconds

How long will this program take to run on a 233 MHz Pentium?

Example

Two implementations of the same instruction set, machine A has a clock cycle time of 1 ns, and a CPI of 2.0, machine B has a clock cycle time of 2 ns and a CPI of 1.2 for the same program. Which machine is faster, and by how much?

$$\text{CPU time}_A = I \times 2.0 \times 1 = 2.0I$$

$$\text{CPU time}_B = I \times 1.2 \times 2 = 2.4I$$

$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{\text{CPU time}_B}{\text{CPU time}_A} = \frac{2.4I}{2.0I} = 1.2$$

A is faster by a factor of 1.2

Example

Our favorite program runs in 10 seconds on computer A, which has a 400 Mhz. clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program. What clock rate should we tell the designer to target?"

Don't Panic, can easily work this out from basic principles

Now that we understand cycles

A given program will require

- some number of instructions (machine instructions)
- some number of cycles
- some number of seconds

We have a vocabulary that relates these quantities:

- cycle time (seconds per cycle)
- clock rate (cycles per second)
- CPI (cycles per instruction)

  a floating point intensive application might have a higher CPI

- MIPS (millions of instructions per second)

  this would be higher for a program using simple instructions

Performance

Performance is determined by execution time

Do any of the other variables equal performance?

- # of cycles to execute program?
- # of instructions in program?
- # of cycles per second?
- average # of cycles per instruction?
- average # of instructions per second?

Common pitfall: thinking one of the variables is indicative of performance when it really isn't.

Amdahl's Law

Execution Time After Improvement =
Execution Time Unaffected +
(Execution Time Affected / Amount of Improvement)

Example:

"Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"

How about making it 5 times faster?

Principle: Make the common case fast

Example

Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?

We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show a speedup of 3. One benchmark we are considering runs for 100 seconds with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

Remember

Performance is specific to a particular program/s

- Total execution time is a consistent summary of performance

For a given architecture performance increases come from:

- increases in clock rate (without adverse CPI affects)
- improvements in processor organization that lower CPI
- compiler enhancements that lower CPI and/or instruction count

Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance