UNIVERSITY OF TEXAS ARLINGTON

Computer Organization & Assembly Language Programming (CSE 2312)

Lecture 3

Taylor Johnson

Summary from Last Time

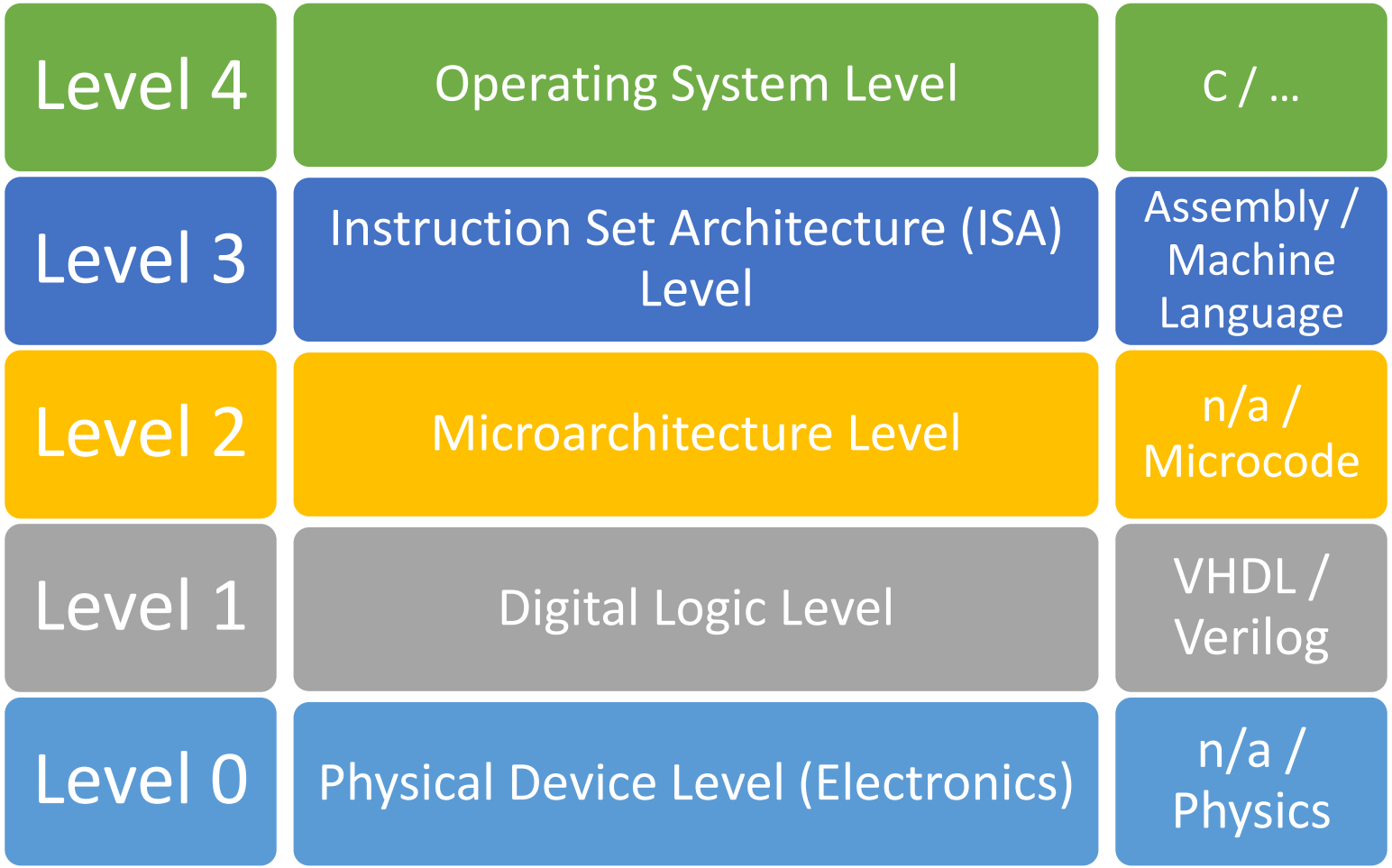
- Binary to decimal, decimal to binary, ASCII
- Structured computers
  - Multilevel computers and architectures
  - Abstraction layers

Announcements and Outline

- Quiz 1 on Blackboard site (due 11:59PM Friday)
  - Review binary arithmetic, Boolean operations, and representing numbers in binary
- Homework 1 on course website
  - Read chapter 1

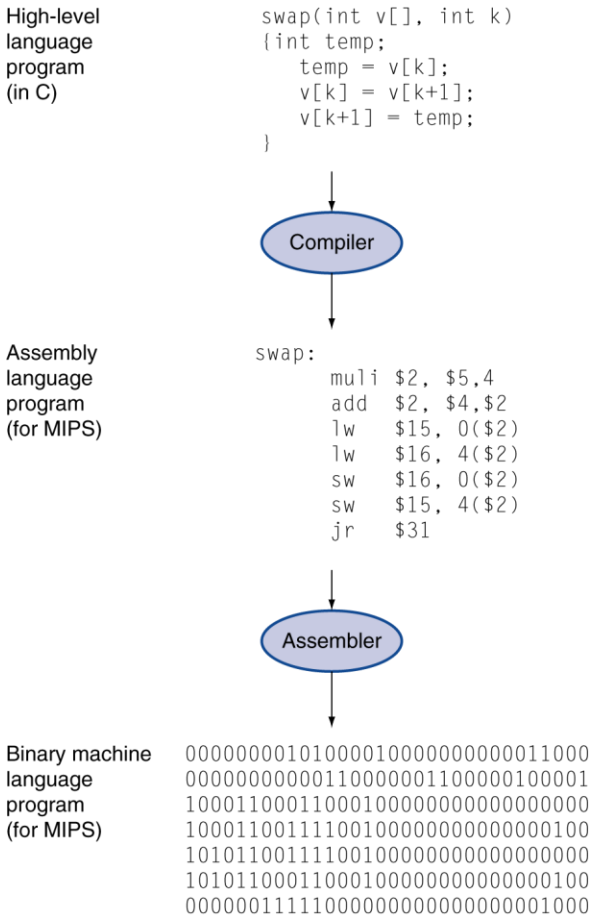
- Review from last time
  - Structured computers
- Performance metrics

Review: Multilevel Architectures



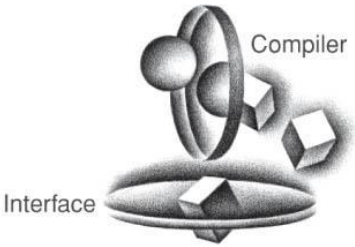
Review: Levels of Program Code

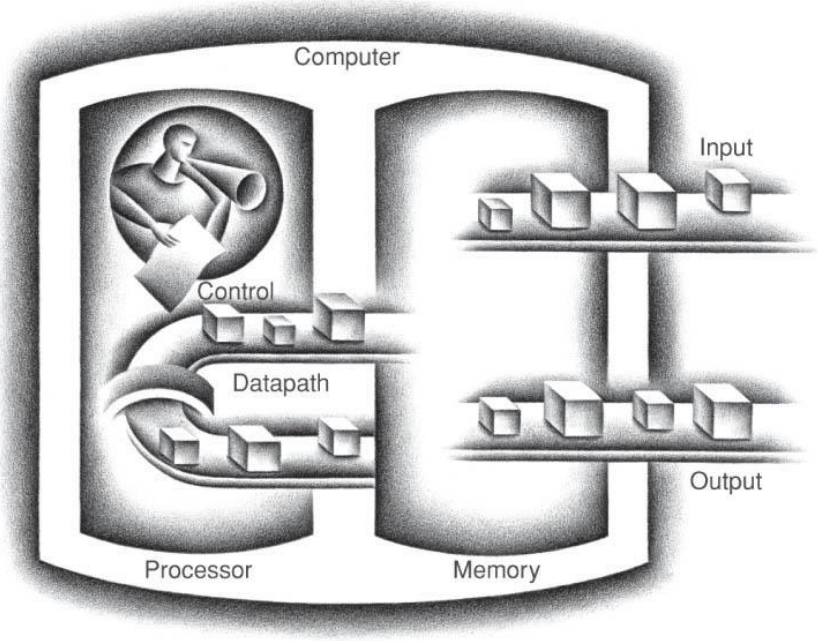
- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



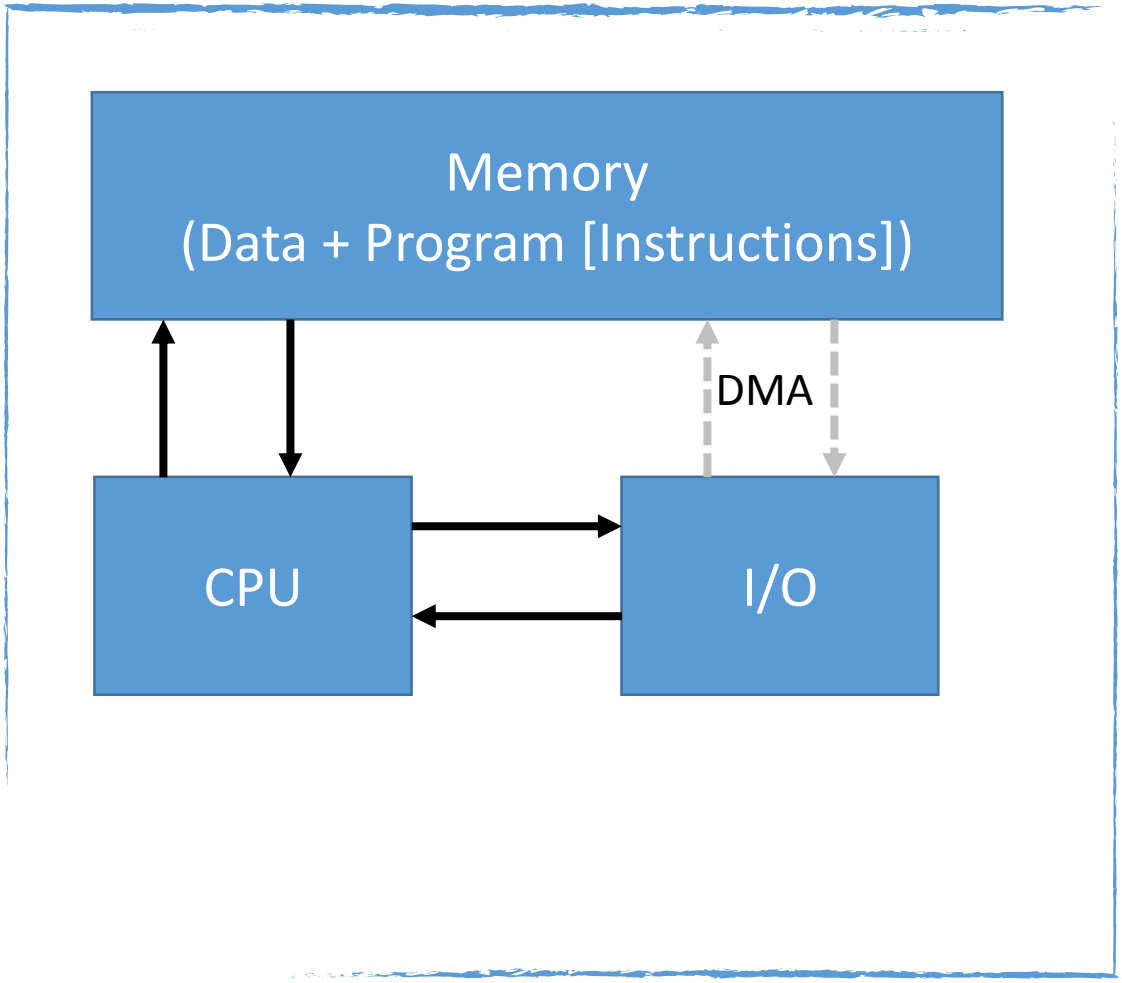
Review: Computer Organization Overview

- CPU
  - Executes instructions
- Memory
  - Stores programs and data
- Buses
  - Transfers data
- Storage
  - Permanent
- I/O devices
  - Input: keypad, mouse, touch
  - Output: printer, screen
  - Both (input and output), such as:
    - USB, network, Wifi, touch screen, hard drive



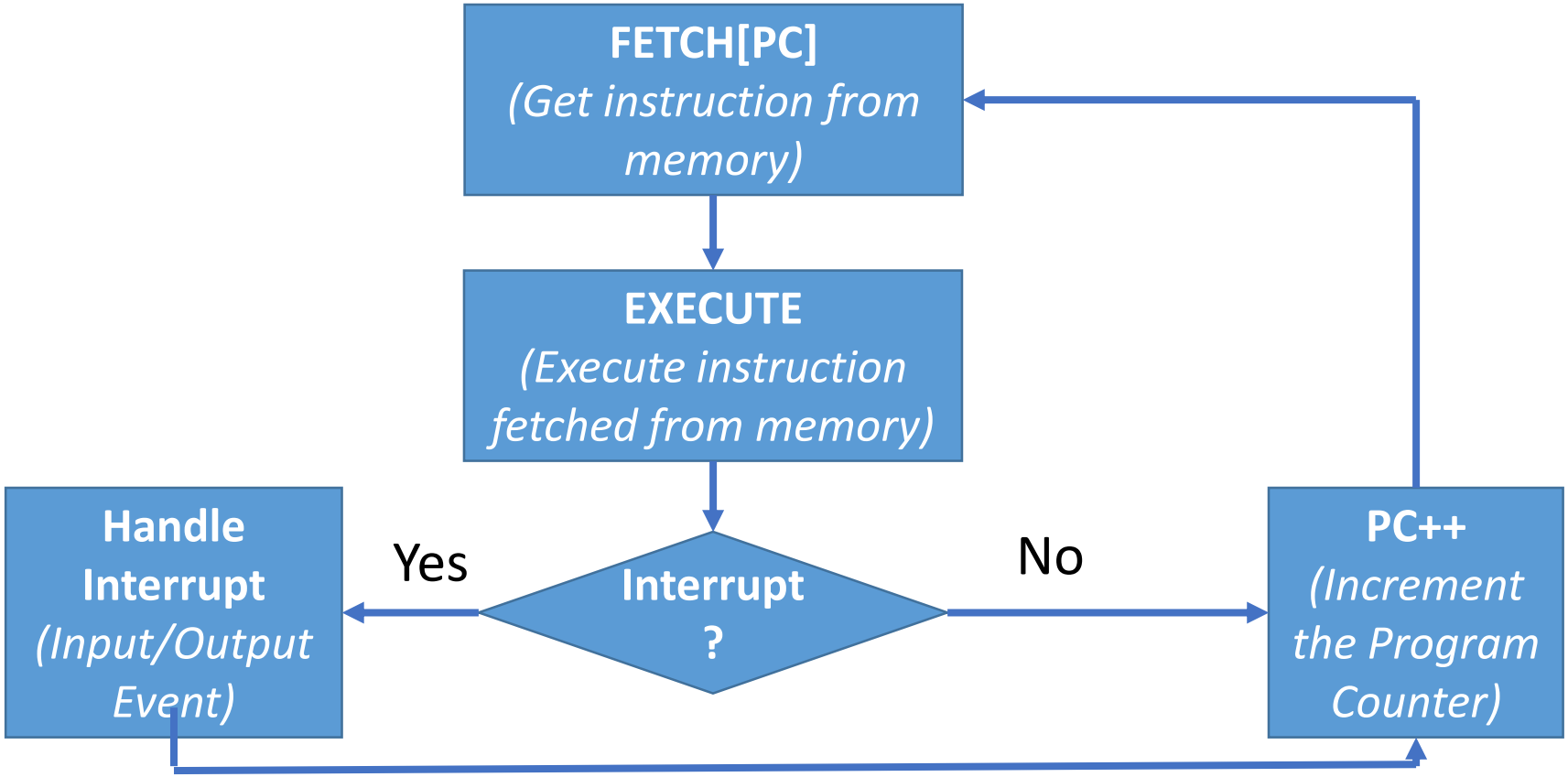


Review: Von Neumann Architecture



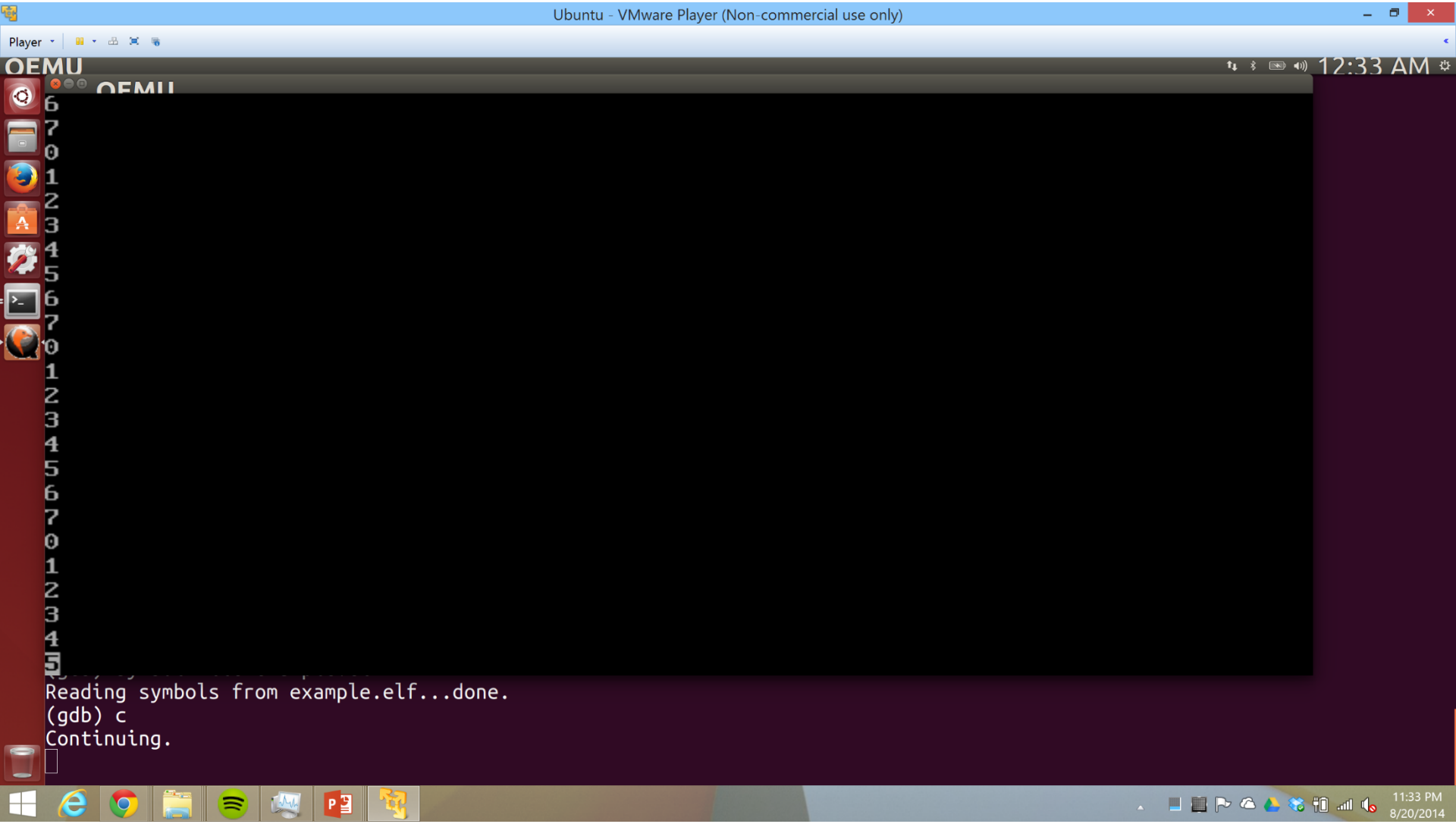
- Both data and program stored in memory
- Allows the computer to be “re-programmed”
- Input/output (I/O) goes through CPU
- I/O part is not representative of modern systems (direct memory access [DMA])
- Memory layout is representative of modern systems

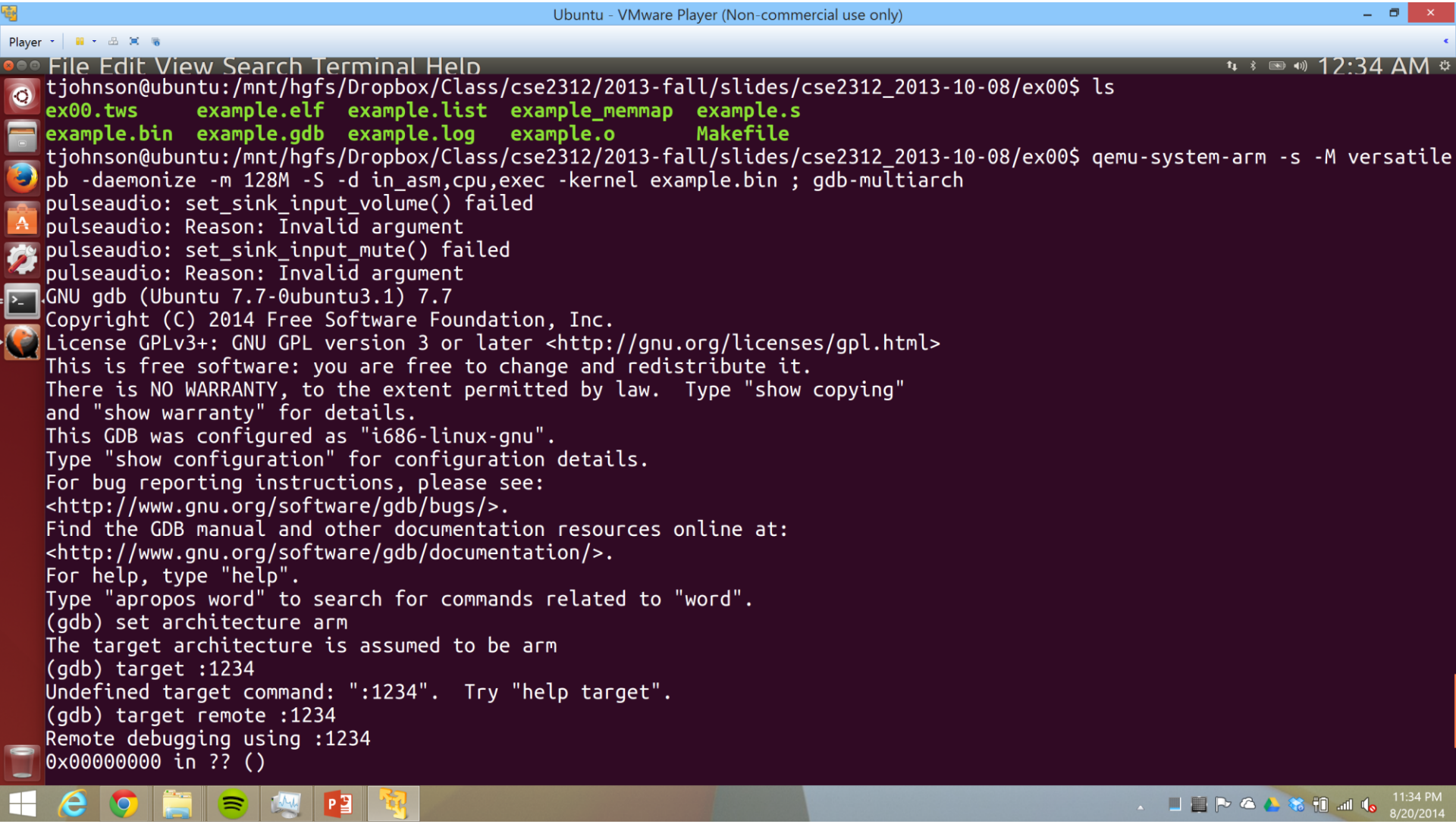
Review: Abstract Processor Execution Cycle



Demonstration

- VMWare, QEMU, and ARM ISA and gdb
- We will use QEMU and ARM later in this course
  - Particularly for programming assignments
- ARM versus x86
  - ARM is prevalent in embedded systems and handheld devices, many of which have more limited resources than your x86/x86-64 PC
  - Limited resources sometimes requires being very efficient (in space/memory or time/processing complexity)
  - Potentially greater need to interface with hardware


Ubuntu - VMware Player (Non-commercial use only)
Player
QEMU
12:33 AM
QEMU
6
7
0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7
0
1
2
3
4
5
Reading symbols from example.elf...done.
(gdb) c
Continuing.
11:33 PM
8/20/2014


Ubuntu - VMware Player (Non-commercial use only)
Player
File Edit View Search Terminal Help
12:34 AM
tjohnson@ubuntu:/mnt/hgfs/Dropbox/Class/cse2312/2013-fall/slides/cse2312_2013-10-08/ex00$ ls
ex00.tws example.elf example.list example_memmap example.s
example.bin example.gdb example.log example.o Makefile
tjohnson@ubuntu:/mnt/hgfs/Dropbox/Class/cse2312/2013-fall/slides/cse2312_2013-10-08/ex00$ qemu-system-arm -s -M versatile
pb -daemonize -m 128M -S -d in_asm,cpu,exec -kernel example.bin ; gdb-multiarch
pulseaudio: set_sink_input_volume() failed
pulseaudio: Reason: Invalid argument
pulseaudio: set_sink_input_mute() failed
pulseaudio: Reason: Invalid argument
GNU gdb (Ubuntu 7.7-0ubuntu3.1) 7.7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) set architecture arm
The target architecture is assumed to be arm
(gdb) target :1234
Undefined target command: ":1234". Try "help target".
(gdb) target remote :1234
Remote debugging using :1234
0x00000000 in ?? ()
11:34 PM
8/20/2014

Ubuntu - VMware Player (Non-commercial use only)

Player

File Edit View Search Terminal Help

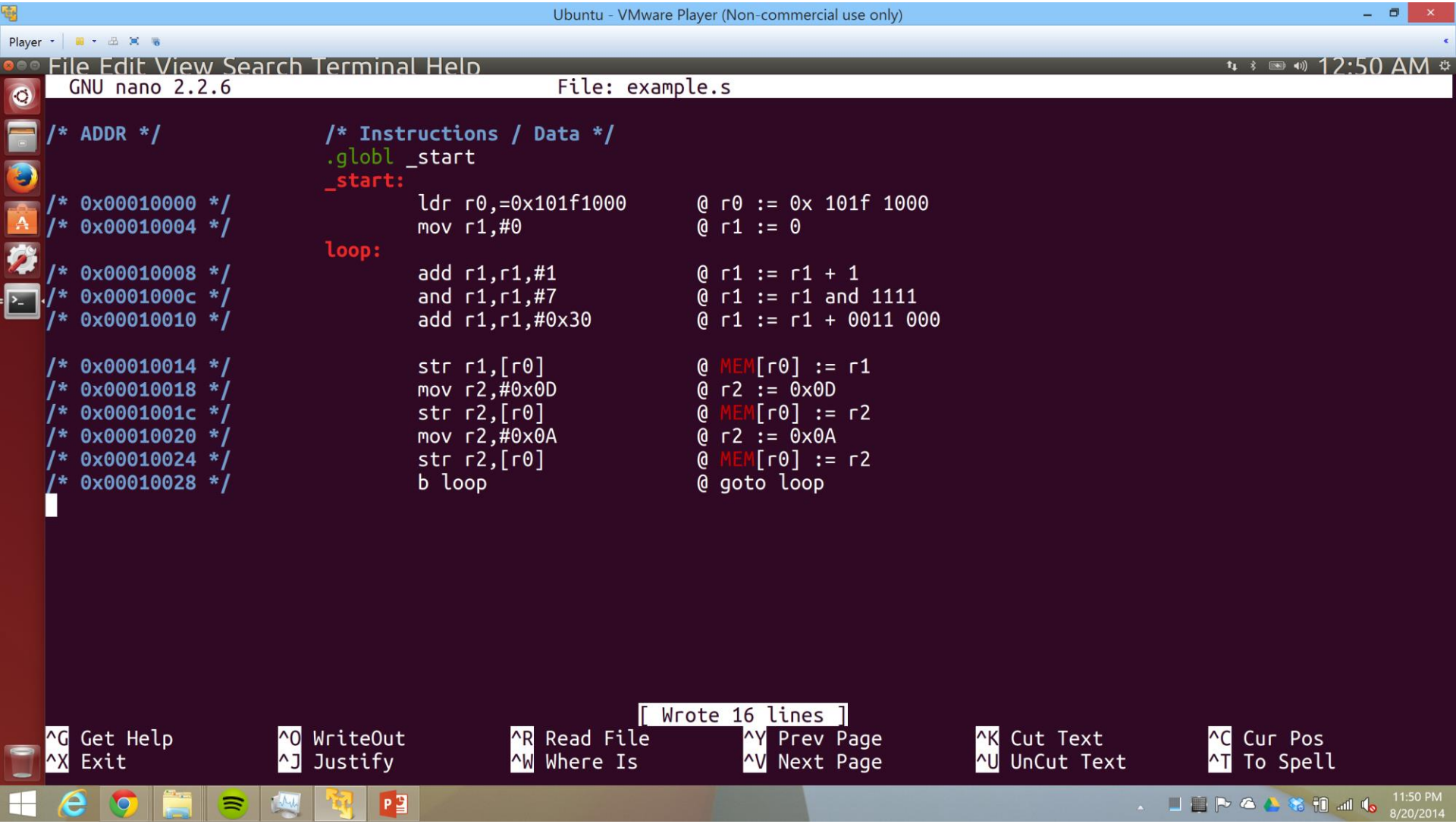
12:34 AM

```
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) set architecture arm
The target architecture is assumed to be arm
(gdb) target :1234
Undefined target command: ":1234".  Try "help target".
(gdb) target remote :1234
Remote debugging using :1234
0x00000000 in ?? ()
(gdb) symbol-file example.elf
Reading symbols from example.elf...done.
(gdb) c
Continuing.
^C
Program received signal SIGINT, Interrupt.
loop () at example.s:7
7       /* 0x00010008 */                add r1,r1,#1            @ r1 := r1 + 1
(gdb) q
A debugging session is active.

        Inferior 1 [Remote target] will be killed.

Quit anyway? (y or n) y
tjohnson@ubuntu:/mnt/hgfs/Dropbox/Class/cse2312/2013-fall/slides/cse2312_2013-10-08/ex00$
```

11:34 PM
8/20/2014


Ubuntu - VMware Player (Non-commercial use only)
Player
File Edit View Search Terminal Help
12:50 AM
GNU nano 2.2.6 File: example.s
/* ADDR */ /* Instructions / Data */
.globl _start
_start:
/* 0x00010000 */ ldr r0,=0x101f1000 @ r0 := 0x 101f 1000
/* 0x00010004 */ mov r1,#0 @ r1 := 0
loop:
/* 0x00010008 */ add r1,r1,#1 @ r1 := r1 + 1
/* 0x0001000c */ and r1,r1,#7 @ r1 := r1 and 1111
/* 0x00010010 */ add r1,r1,#0x30 @ r1 := r1 + 0011 000
/* 0x00010014 */ str r1,[r0] @ MEM[r0] := r1
/* 0x00010018 */ mov r2,#0x0D @ r2 := 0x0D
/* 0x0001001c */ str r2,[r0] @ MEM[r0] := r2
/* 0x00010020 */ mov r2,#0x0A @ r2 := 0x0A
/* 0x00010024 */ str r2,[r0] @ MEM[r0] := r2
/* 0x00010028 */ b loop @ goto loop
[Wrote 16 lines]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
11:50 PM
8/20/2014

Announcements and Outline

- Quiz 1 on Blackboard site (due 11:59PM Friday)
  - Review binary arithmetic, Boolean operations, and representing numbers in binary
- Homework 1 on course website
  - Read chapter 1

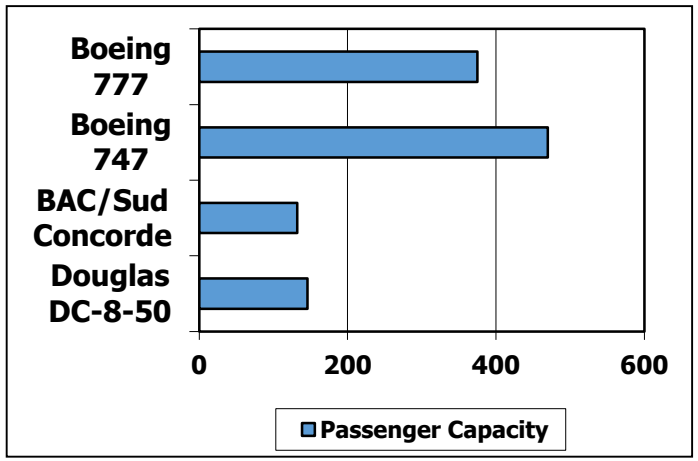
- Review from last time
  - Structured computers
- Performance metrics

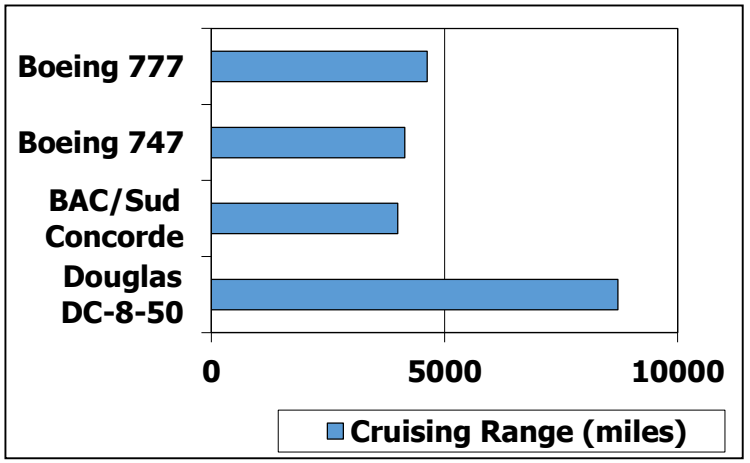
Performance Metrics

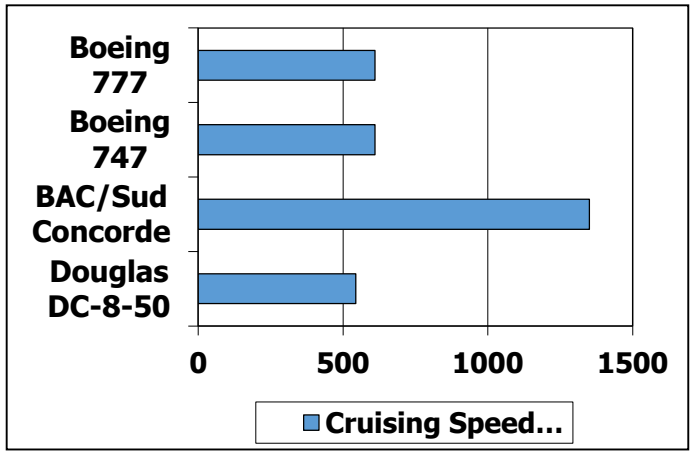
- Performance is important in computer systems
- How to ***quantitatively*** compare different computer systems?
- How to do this in general?
  - Cars: MPG, speed, acceleration, towing capability, passengers, …
  - Computer processors
    - execution time of a program (seconds)
    - instruction count (instructions executed in a program)
    - CPI: clock cycles per instruction (average number of clock cycles per instruction)
    - Clock cycle time (seconds per clock cycle)

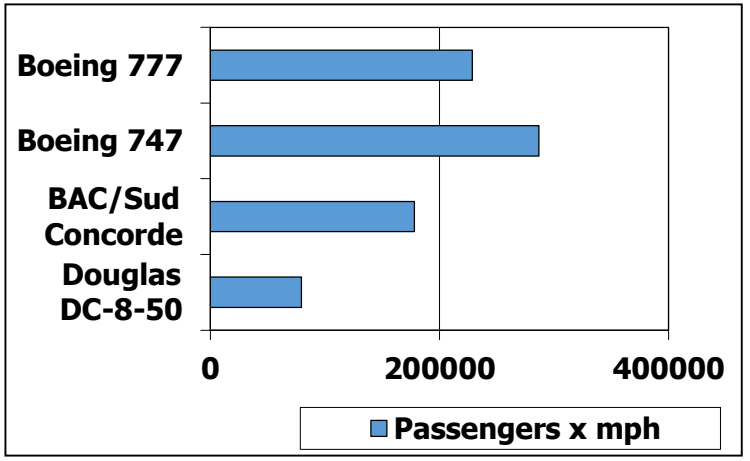
Defining Performance

- Which airplane has the best performance?









Some Units You Must Know

- Hertz (Hz): unit of frequency
- 1 Hz: once per second
- 1 Megahertz (1 MHz): one million times per second
- 1 Gigahertz (1 GHz): one billion times per second

- second: unit of time
  - 1 millisecond (1ms): a thousandth of a second.
  - 1 microsecond (1μs): a millionth of a second.
  - 1 nanosecond (1ns): a billionth of a second.
- Similarly for meters:
  - millimeter: a thousandth
  - micrometer: a millionth
  - nanometer: a billionth

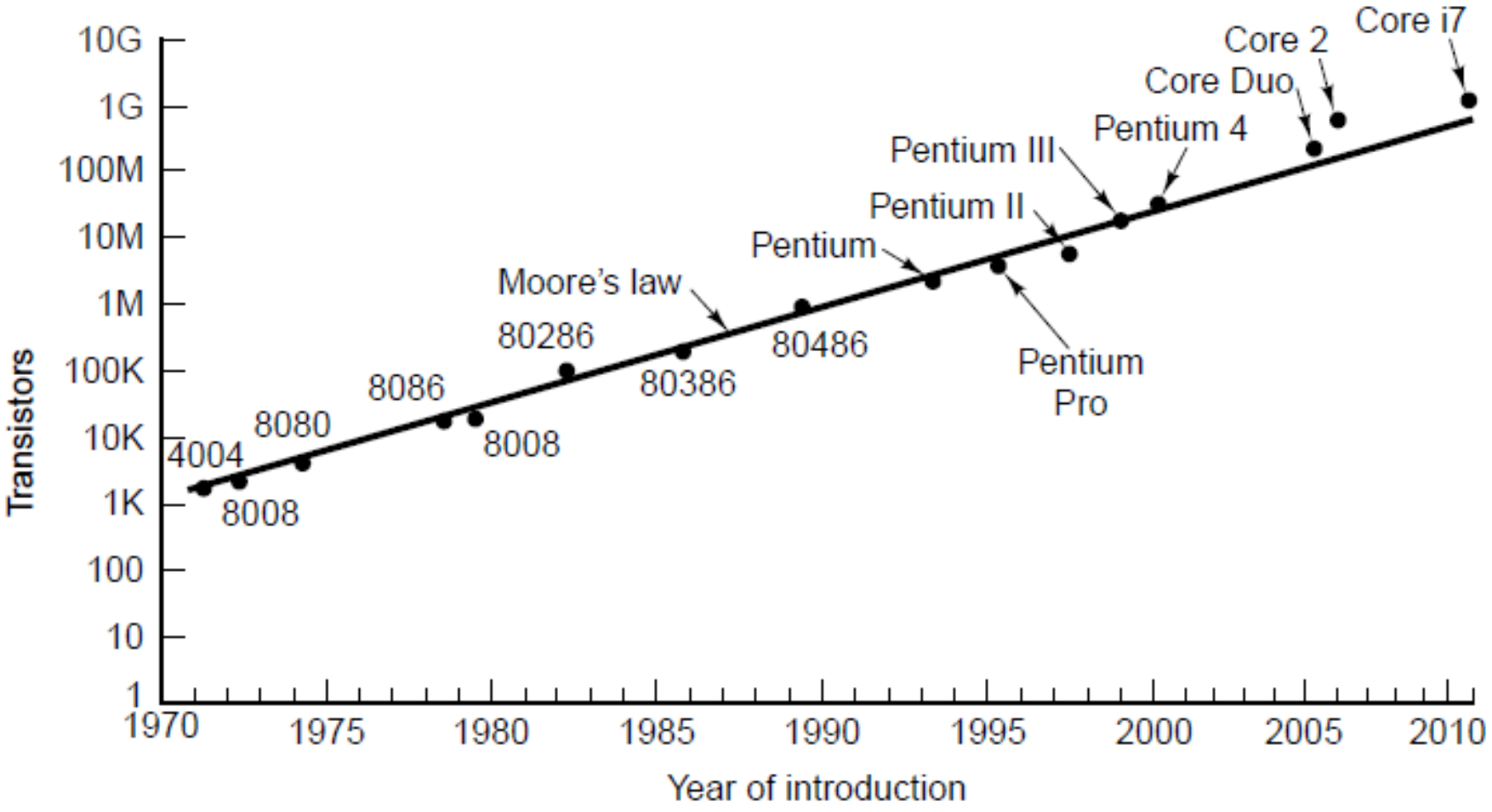
Units of Memory

- One bit (binary digit): the smallest amount of information that we can store:
  - Either a 1 or a 0
  - Sometimes refer to 1 as high/on/true, 0 as low/off/false
- One byte = 8 bits
  - Can store a number from 0 to 255
- Kilobyte (KB): $10^3 = 1000$ bytes
- Kibibyte (KiB): $2^{10} = 1024$ bytes
- Kilobit: (Kb): $10^3 = 1000$ bits (125 bytes)
- Kibibit: (Kib): $2^{10} = 1024$ bits (128 bytes)

Metric Units

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	yotta

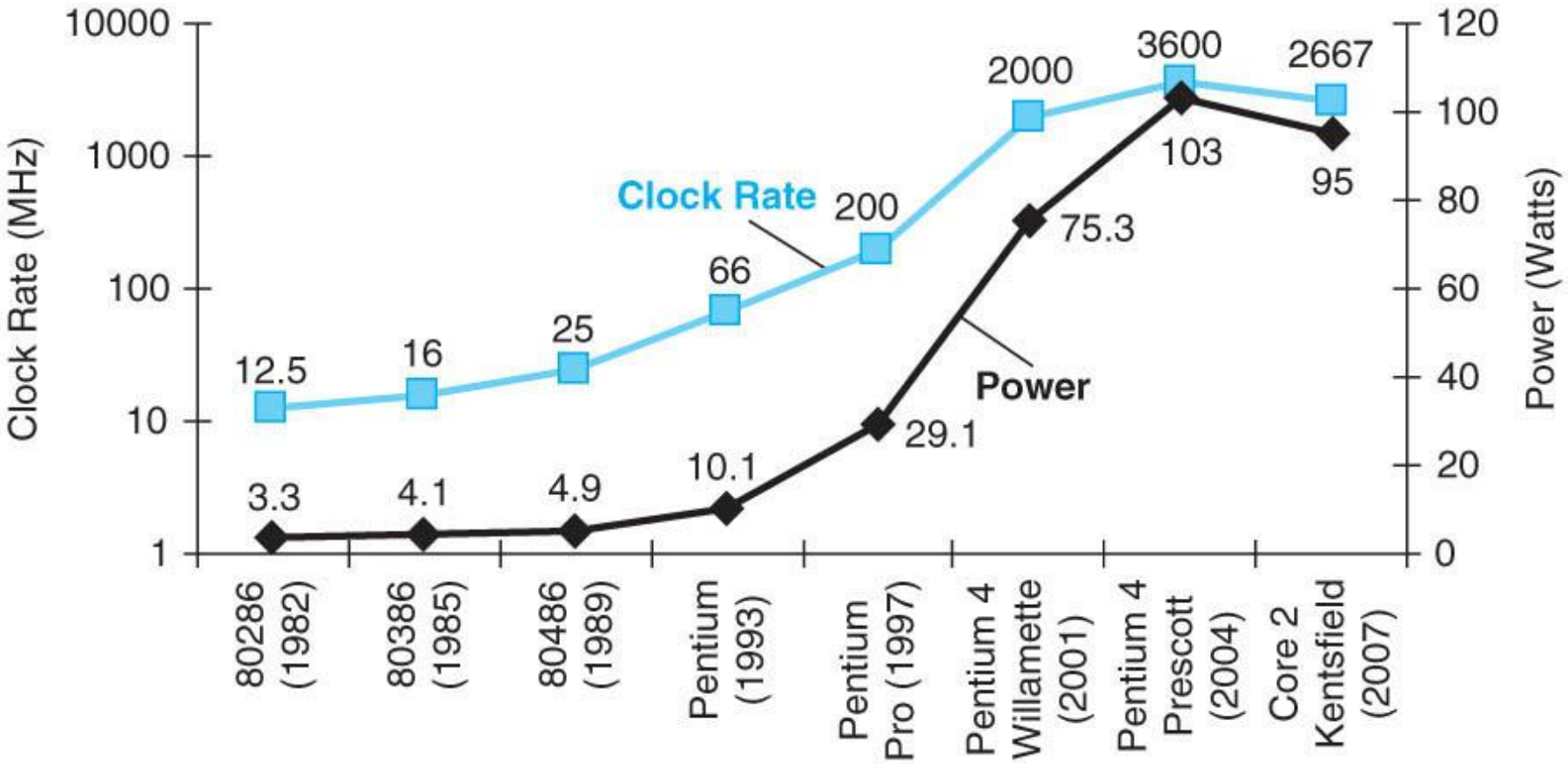
Moore's Law for the Intel Family



Moore's Law

- Not a real "law" of nature, just a practical observation that has remained surprisingly accurate for decades
- Predicts a 60% annual increase in the number of transistors per chip
- Number of transistors on a chip doubles every 18 months
- Memory capacity doubles every 2 years
- Disk capacity doubles every year

The Power Wall



Moore's Law

- These observations are more like rules of thumb
- However, they have been good predictors since the 1960's, more than half a century!
- Moore's law originally was stated in 1965
- How long will this exponential growth in hardware capabilities grow?
  - Nobody really knows
  - Expected to continue for the next few years
  - When transistors get to be the size of an atom, hard to predict if and how this growth can continue

Moore Law Example 1

- Suppose average disk capacity right now is 1TB
- Suppose disk capacity doubles each year
- What will average disk capacity be in 5 years?

Moore Law Example 1

- Suppose average disk capacity right now is 1TB.
- Suppose disk capacity doubles each year.
- What will average disk capacity be in 5 years?
- Answer: 32 TB

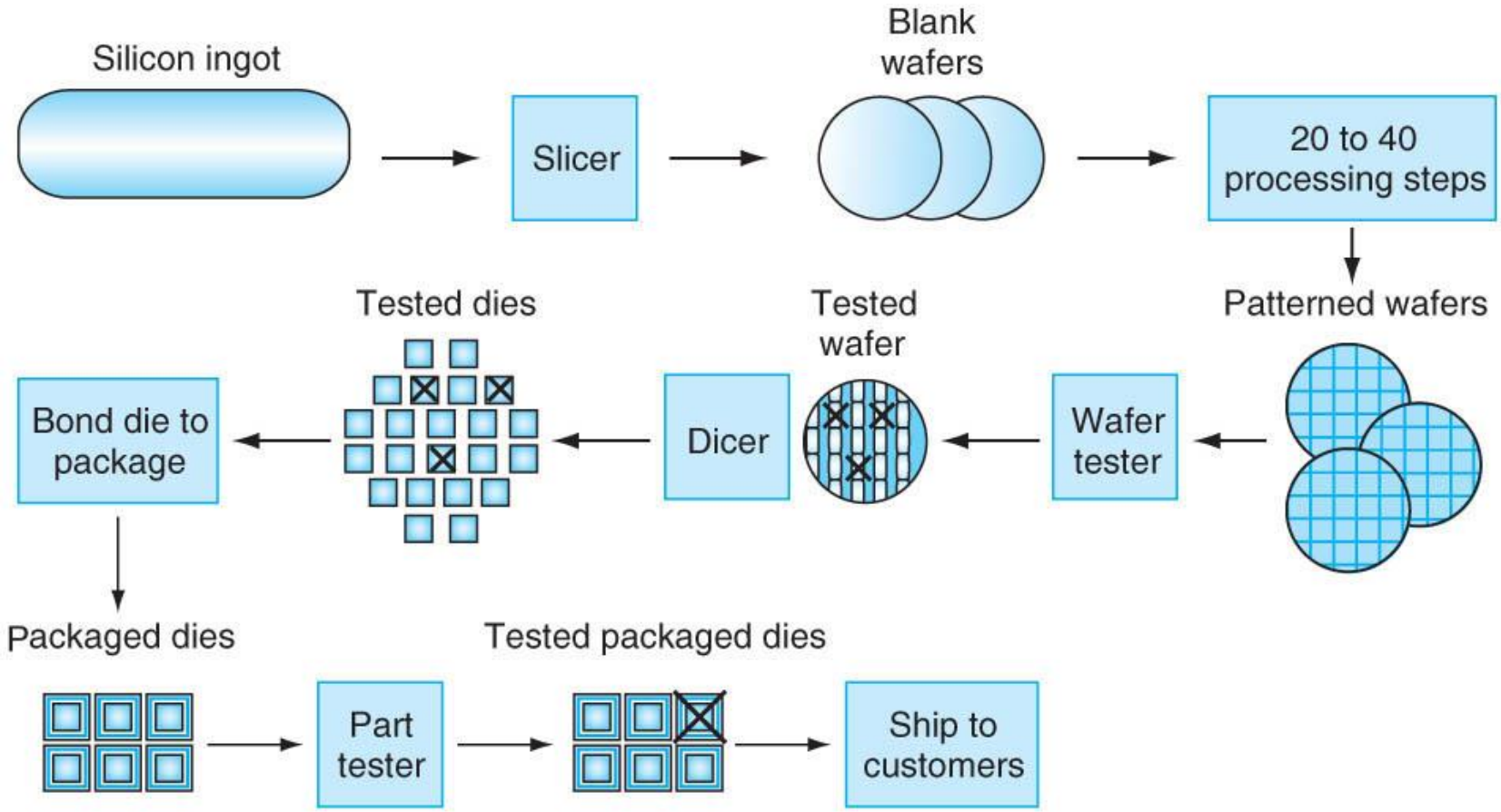
Moore Law Example 2

- Suppose average number of instructions per second in 1960 was 100,000 (this number is made up)
- Suppose average number of instructions per second in 1970 was 10,000,000 (this number is made up)
- What would be Moore's law for the average number of instructions? How often does it double?

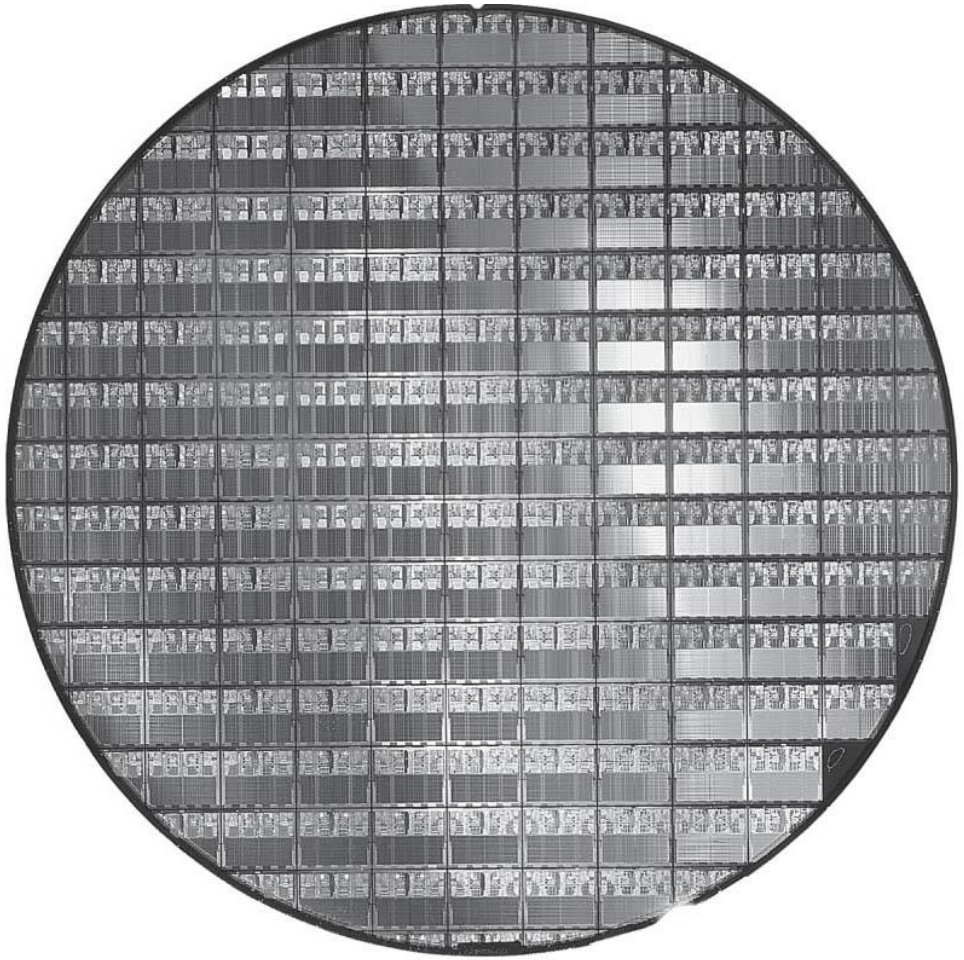
Moore Law Example 2

- Suppose average number of instructions per second in 1960 was 100,000 (this number is made up)
- Suppose average number of instructions per second in 1970 was 10,000,000 (this number is made up)
- What would be Moore's law for the average number of instructions? How often does it double?
- Answer:
  - In 10 years, this number increased by 100 times.
  - $100 = 2^{6.64}$
  - Thus, this number doubles every 10/6.64 years = about 18 months

Silicon Integrated Circuit Manufacturing Process



A 12-inch (300mm) wafer of AMD Opteron X2 chips, the predecessor of Opteron X4 chips (Courtesy AMD). The number of dies per wafer at 100% yield is 117. The several dozen partially rounded chips at the boundaries of the wafer are useless; they are included because it's easier to create the masks used to pattern the silicon. This die uses a 90-nanometer technology, which means that the smallest transistors are approximately 90 nm in size, although they are typically somewhat smaller than the actual feature size, which refers to the size of the transistors as "drawn" versus the final manufactured size.



Integrated Circuit Cost

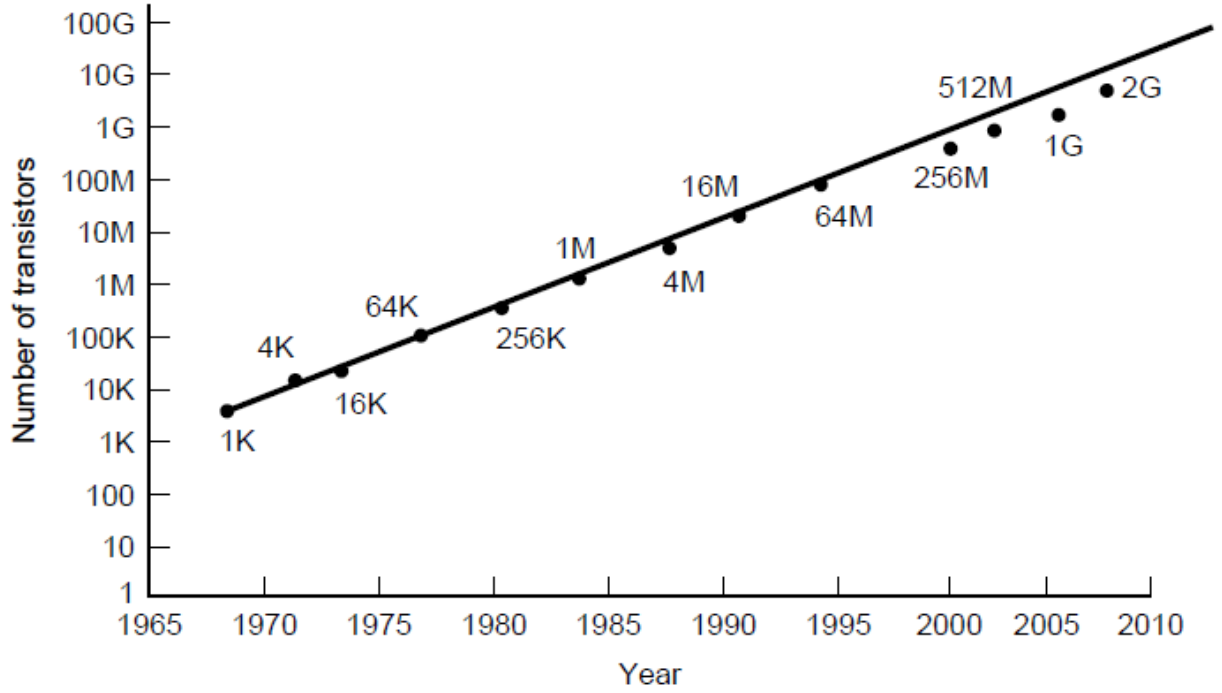
$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

$$\text{Yield} = \frac{1}{(1+(\text{Defects per area} \times \text{Die area}/2))^2}$$

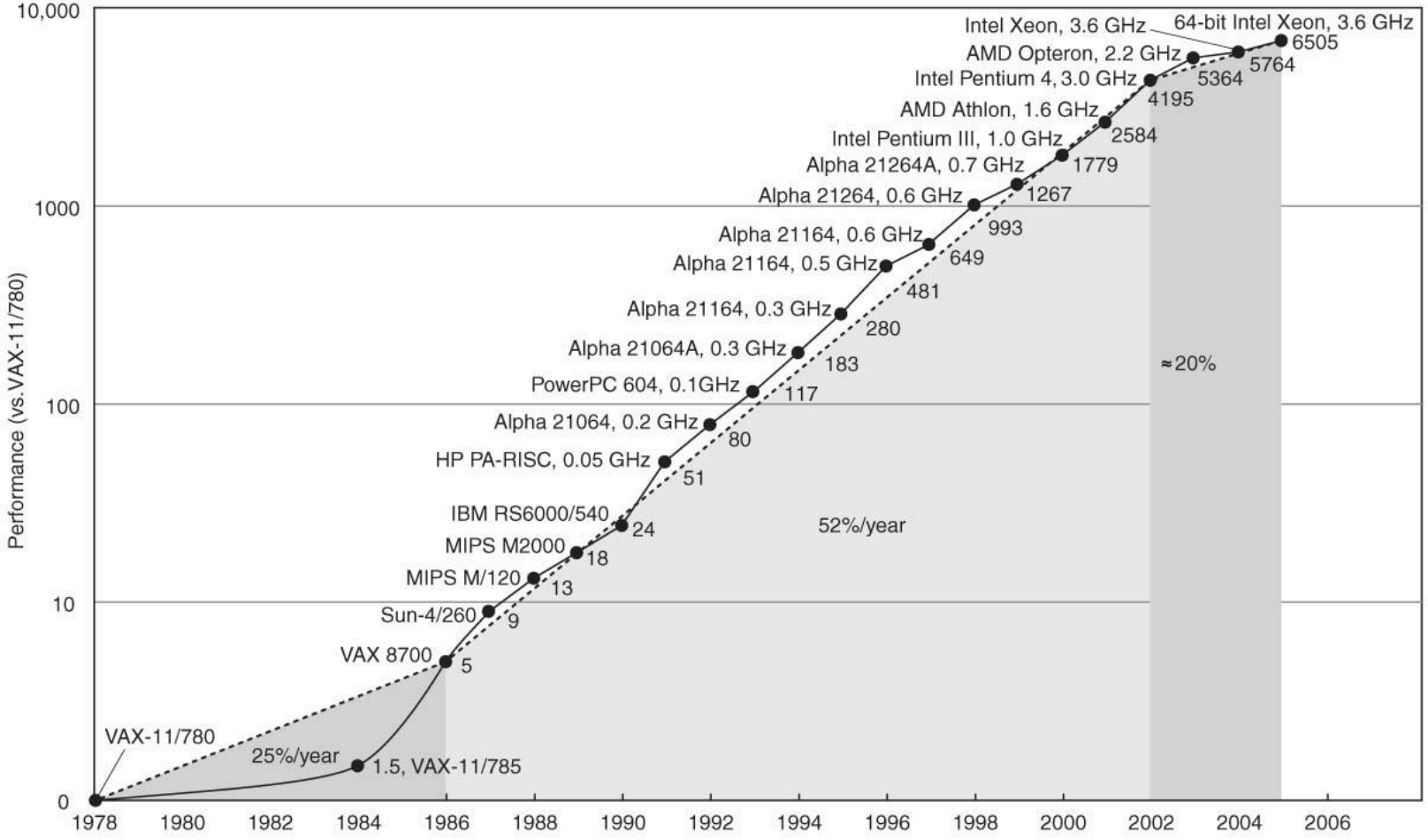
- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

Moore's Law



Moore’s law predicts a 60 percent annual increase in the number of transistors that can be put on a chip. The data points given above and below the line are memory sizes, in bits.

Growth in processor performance since the mid-1980 (relative to VAX 11/780 on SPECint benchmarks)



Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

Relative Performance

- Define Performance = 1/Execution Time
- "X is *n* time faster than Y"

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

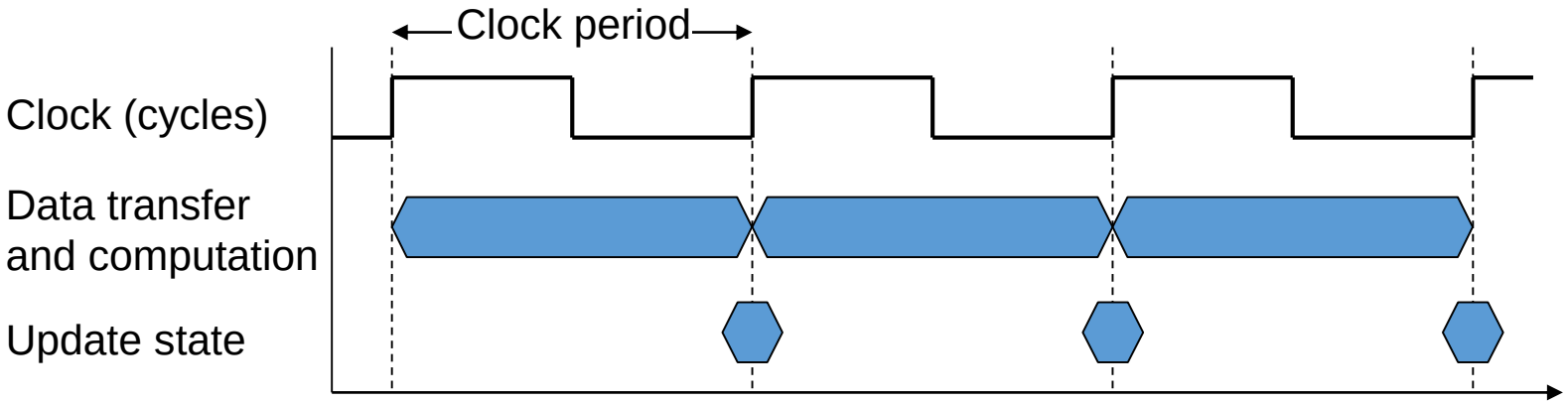
- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program = number of instructions in program
  - Determined by program, ISA and compiler
- Average cycles per instruction (CPI) = number of cycles to execute an instruction (on average)
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

…by this much

CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

Performance Summary

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

SPEC CPU Benchmark

- Programs used to measure performance
  - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, …
- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

SPECint2006 / CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECrati
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

  - CPI varies between programs on a given CPU

Technological and Economic Forces

- Improvements in hardware creates opportunities for new applications
- New applications attract new businesses
- New businesses drive competition
- Competition drives improvements in hardware

Technological and Economic Forces

- "Software is a gas. It expands to fill the container holding it."

  - Nathan Myhrvold, former Microsoft executive.

- Software expands with additional features, to exploit new hardware capabilities.
- Software expansion creates need for better hardware.

Chapter 1 Summary

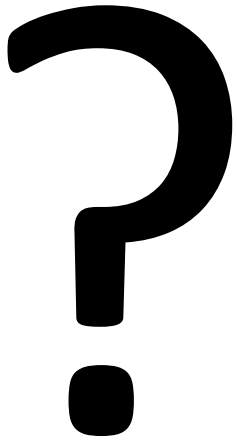
- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance

Summary

- Reviewed structured computers
- Basic performance metrics, units

- Quiz 1 on Blackboard
- Homework 1

Questions?



SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

SPECpower_ssj2008 for Xeon X5650

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
Σssj_ops/Σpower =		2,490

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

  - Can't be done!
- Corollary: make the common case fast

Fallacy: Low Power at Idle

- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load