

Computer Organization & Assembly Language Programming (CSE 2312)

Lecture 2

Taylor Johnson

Summary from Last Time

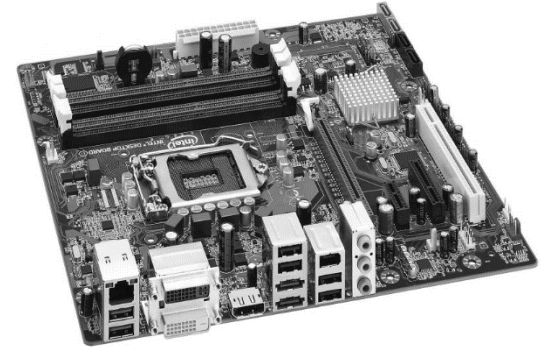
- This course aims to answer the question: **how do computers compute?**
- Complex and fundamental question
 - Organization of computer
 - Multilevel architectures
- Assembly programming
 - QEMU, ARM, gcc tools (as), and gdb (GNU debugger)

Announcements and Outline

- Quiz 1 on Blackboard site (due by end of week)
 - Review binary arithmetic, Boolean operations, and representing numbers in binary
- Homework 1 on course website
 - Read chapter 1

- Review from last time
- Binary review
- Structured computers
- Performance metrics

Review: Digital Computers

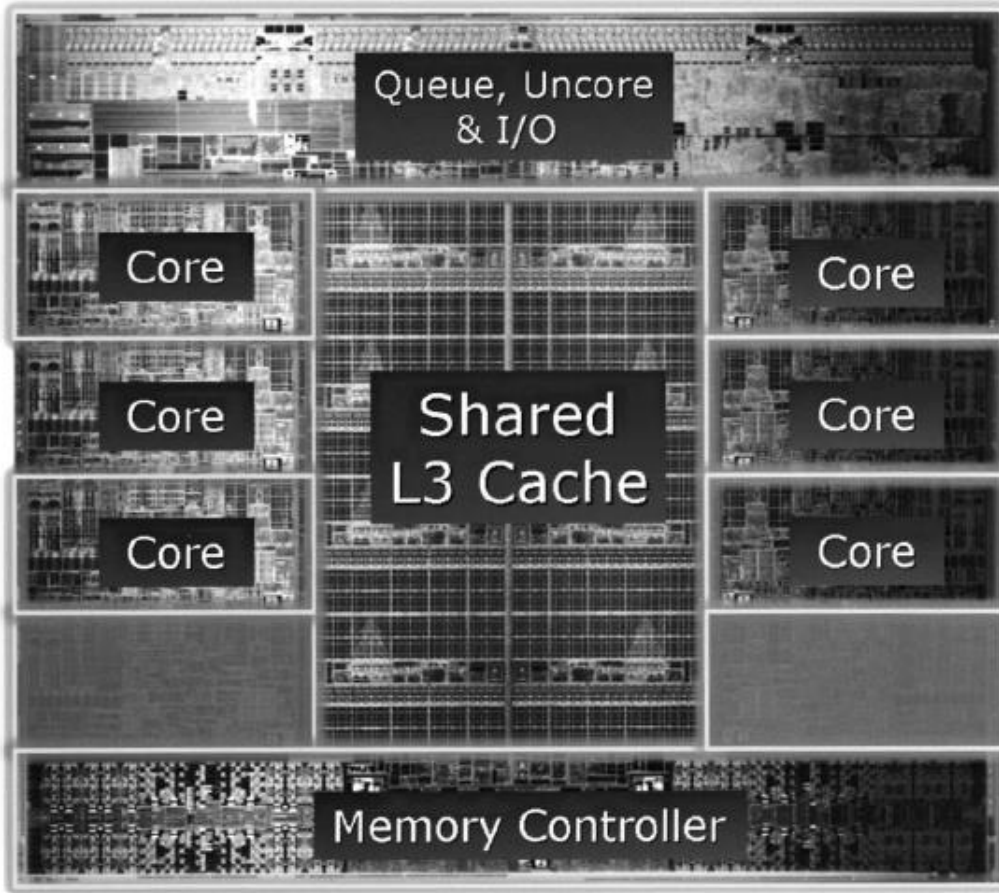


- Machine for carrying out instructions
 - Program = sequence of instructions
- Instructions = primitive operations
 - Add numbers
 - Check if a number is zero
 - Copy data between different memory locations (addresses)
 - Represented as machine language (binary numbers of a certain length)

opcode dest src0 src1

- Example: $\overline{00}$ $\overline{10}$ $\overline{01}$ $\overline{00}$ on an 8-bit computer may mean:
 - Take numbers in registers 0 and 1 (special memory locations inside the processor) and add them together, putting their sum into register 2
 - That is, to this computer, 00100100 means $r2 = r1 + r0$
 - In assembly, this could be written: `add r2 r1 r0`
- Question: for this same computer, what does $\overline{00000000}$ mean?
 - `add r0 r0 r0`, that is: $r0 = r0 + r0$

Review: Computer Examples: Our PC/Laptop



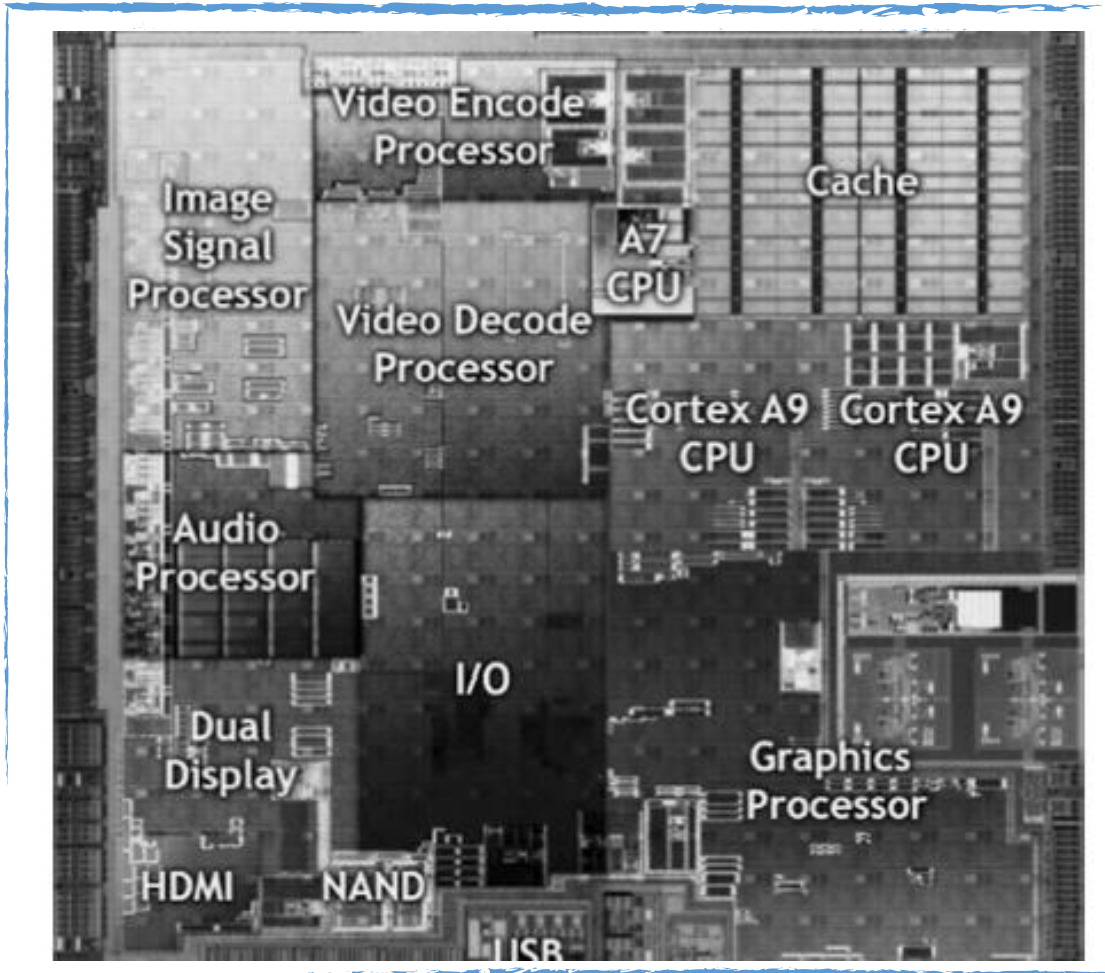
Intel Core i7-3960X die

The die is 21 by 21 mm and has 2.27 billion transistors

ISA: x86-64

© 2011 Intel Corporation

Review: Computer Examples: Our Phone



Nvidia Tegra 2 system on a chip (SoC)

ISA: ARM

© 2011 Nvidia Corporation

Review: Computer Examples: Our Server

Oracle (Sun) SPARC T4
ISA: SPARC

Intel Xeon 7500
ISA: x86-64



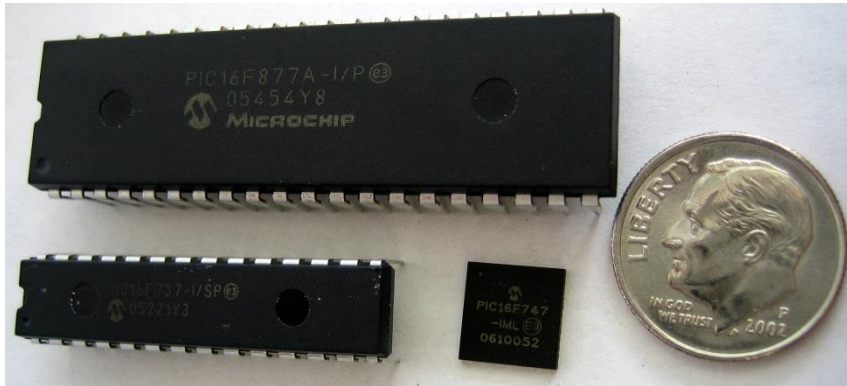
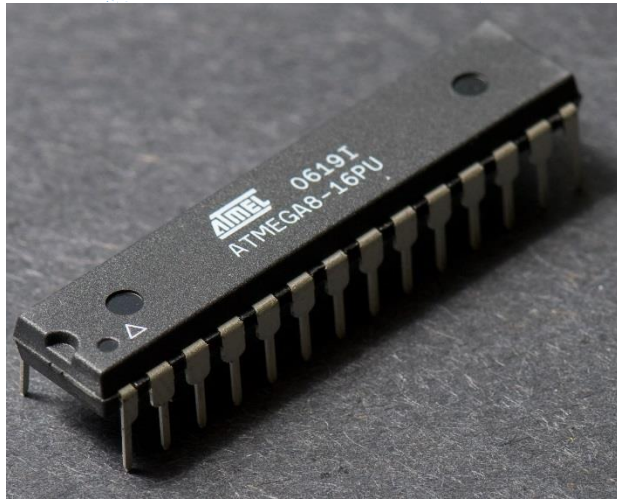
Review: Computer Examples: Our Car

Atmel ATmega

Microchip PIC

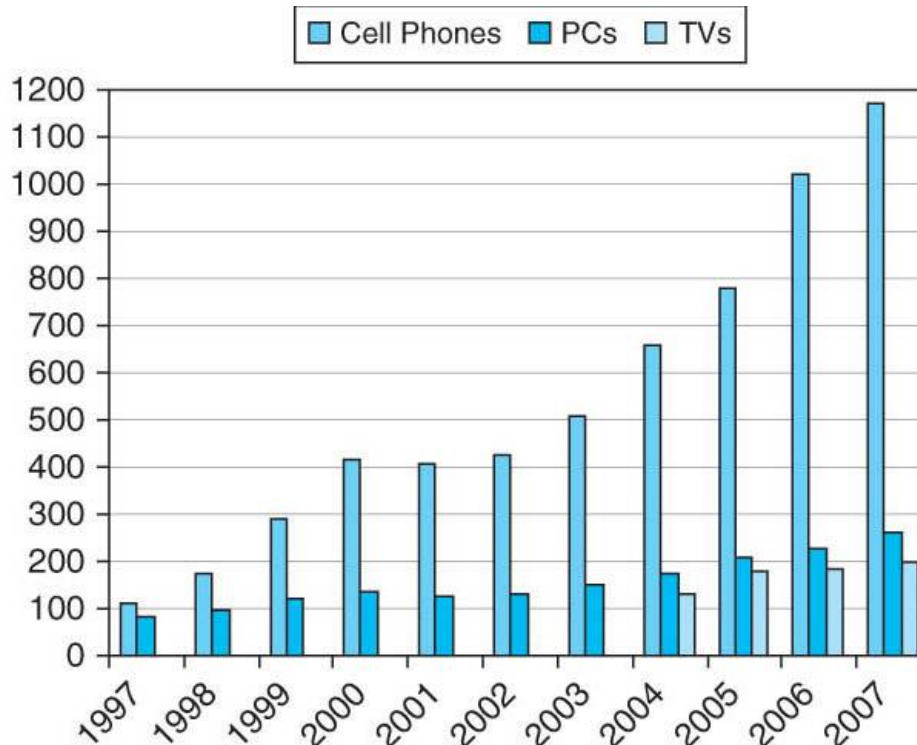
Intel 8051

Many others from TI,
Cypress, etc.



Review: Computer Examples: Others?

- What other examples can you come up with?
 - Moral: everywhere and in everything



Announcements and Outline

- Quiz 1 on Blackboard site (due by end of week)
- Homework 1 on course website

- Review from last time
- Binary review
- Structured computers
- Performance metrics

Numerical Systems and Place Values

- Integer numbers

$$N = (a_n a_{n-1} \dots a_1 a_0)_R = a_n R^n + a_{n-1} R^{n-1} + \dots + a_1 R + a_0$$

- Fractional numbers

$$F = (.a_{-1} a_{-2} \dots a_{-m})_R = a_{-1} R^{-1} + a_{-2} R^{-2} + \dots + a_{-m} R^{-m}$$

- Decimal (Base-10) Numbers ($R = 10$)

$$953.78_{10} = 9 \times 10^2 + 5 \times 10^1 + 3 \times 10^0 + 7 \times 10^{-1} + 8 \times 10^{-2}$$

- Binary (Base-2) Numbers ($R = 2$)

$$\begin{aligned} 1011.11_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} \\ &= 8 + 0 + 2 + 1 + \frac{1}{2} + \frac{1}{4} \\ &= 11.75_{10} \end{aligned}$$

Decimal to Binary Conversion

- Division method: divide by two, take remainder until zero

$$(35)_{10} = (x)_2$$

$$35 / 2 = 17, \text{ remainder } 1$$

$$17 / 2 = 8, \text{ remainder } 1$$

$$8 / 2 = 4, \text{ remainder } 0$$

$$4 / 2 = 2, \text{ remainder } 0$$

$$2 / 2 = 1, \text{ remainder } 0$$

$$1 / 2 = 0, \text{ remainder } 1$$

Result from bottom up: 100011

- Trick: divide by two and taking remainder = odd/even

Decimal to Binary Conversion

- Subtraction method: subtract largest power of 2 until 0; if negative, 0; else 1

$$(23)_{10} = (x)_2$$

$$23 - 2^4 = 23 - 16 = 7 \quad 1$$

$$7 - 2^3 = 7 - 8 = -1 \quad 0$$

$$7 - 2^2 = 7 - 4 = 3 \quad 1$$

$$3 - 2^1 = 3 - 2 = 1 \quad 1$$

$$1 - 2^0 = 1 - 1 = 0 \quad 1$$

Result (read from top): 10111

Binary to Decimal Conversion

- Exponent summation: sum powers of two

$$\begin{aligned}(1011)_2 &= (x)_{10} \\ &= 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 \\ &= 8 + 0 + 2 + 1 \\ &= 11\end{aligned}$$

ASCII Encoding of Text

Binary	Glyph	Binary	Glyph	Binary	Glyph	Binary	Glyph
100 0001	<u>A</u>	100 1110	<u>N</u>	110 0001	<u>a</u>	110 1110	<u>n</u>
100 0010	<u>B</u>	100 1111	<u>O</u>	110 0010	<u>b</u>	110 1111	<u>o</u>
100 0011	<u>C</u>	101 0000	<u>P</u>	110 0011	<u>c</u>	111 0000	<u>p</u>
100 0100	<u>D</u>	101 0001	<u>Q</u>	110 0100	<u>d</u>	111 0001	<u>q</u>
100 0101	<u>E</u>	101 0010	<u>R</u>	110 0101	<u>e</u>	111 0010	<u>r</u>
100 0110	<u>F</u>	101 0011	<u>S</u>	110 0110	<u>f</u>	111 0011	<u>s</u>
100 0111	<u>G</u>	101 0100	<u>T</u>	110 0111	<u>g</u>	111 0100	<u>t</u>
100 1000	<u>H</u>	101 0101	<u>U</u>	110 1000	<u>h</u>	111 0101	<u>u</u>
100 1001	<u>I</u>	101 0110	<u>V</u>	110 1001	<u>i</u>	111 0110	<u>v</u>
100 1010	<u>J</u>	101 0111	<u>W</u>	110 1010	<u>j</u>	111 0111	<u>w</u>
100 1011	<u>K</u>	101 1000	<u>X</u>	110 1011	<u>k</u>	111 1000	<u>x</u>
100 1100	<u>L</u>	101 1001	<u>Y</u>	110 1100	<u>l</u>	111 1001	<u>y</u>
100 1101	<u>M</u>	101 1010	<u>Z</u>	110 1101	<u>m</u>	111 1010	<u>z</u>

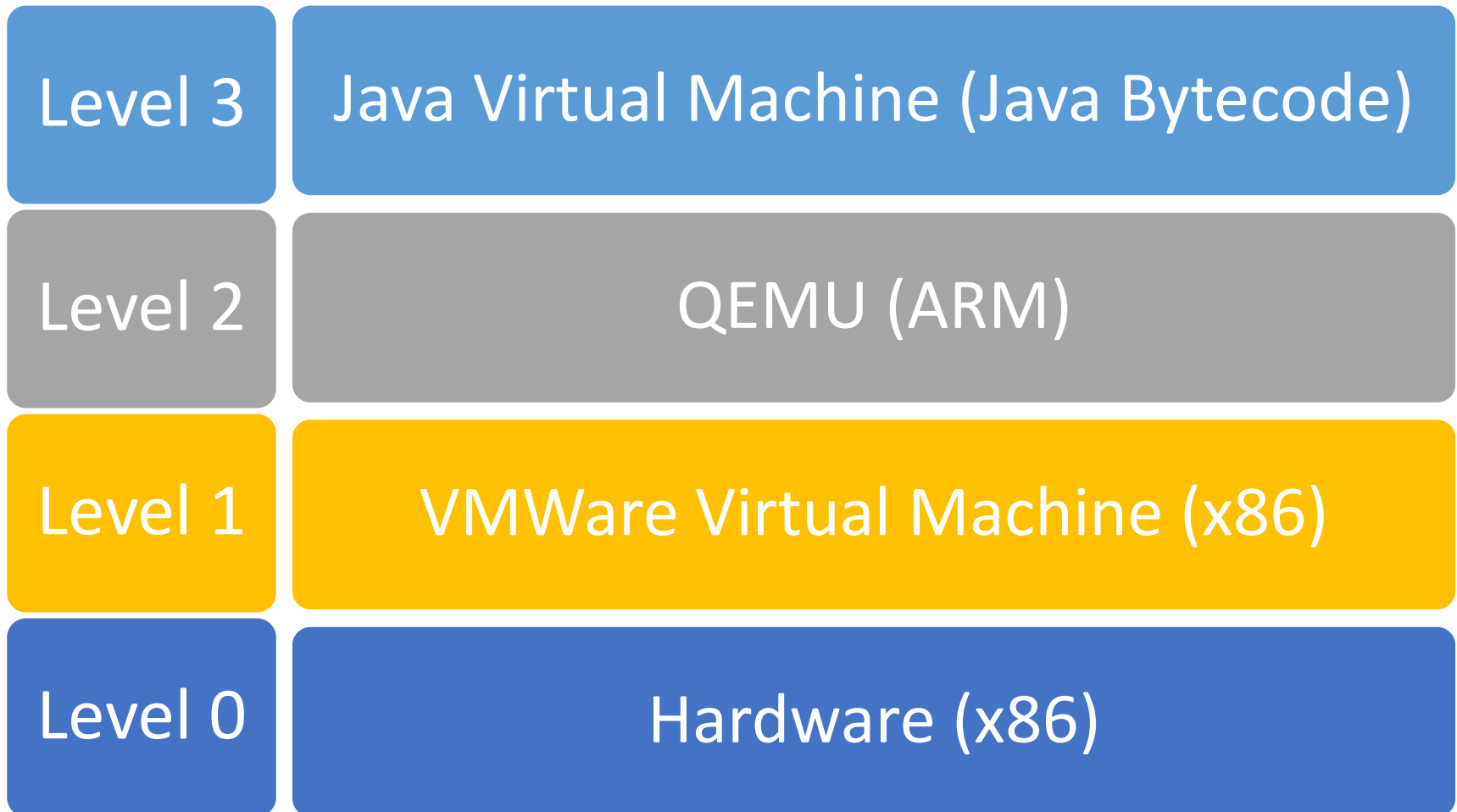
CSE = 100 0011 101 0011 100 0101

Announcements and Outline

- Quiz 1 on Blackboard site (due by end of week)
- Homework 1 on course website

- Review from last time
- Binary review
- Structured computers
- Performance metrics

Multilevel Computers

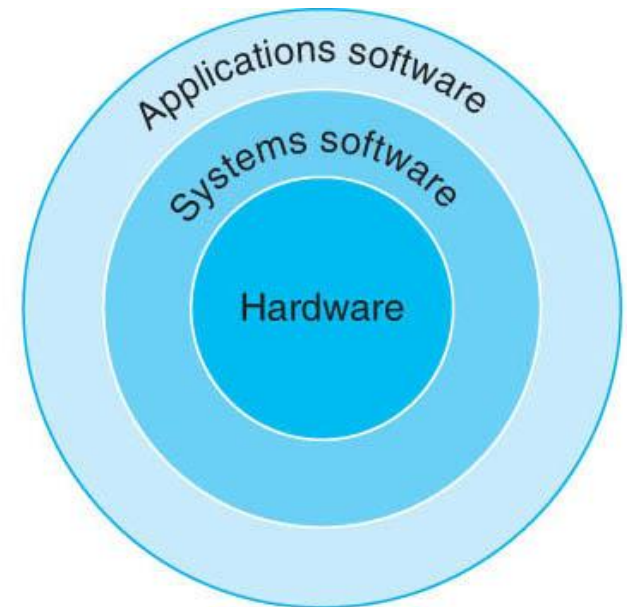


Multilevel Architectures

Level 4	Operating System Level	C / ...
Level 3	Instruction Set Architecture (ISA) Level	Assembly / Machine Language
Level 2	Microarchitecture Level	n/a / Microcode
Level 1	Digital Logic Level	VHDL / Verilog
Level 0	Physical Device Level (Electronics)	n/a / Physics

Multiple Ways To Defining Levels

- Decomposition into these specific levels describes nicely some standard and widely used abstractions
- As you look closely, you may find that a specific level itself can be decomposed into more levels
- Examples:
 - A virtual machine
 - A compiler may go through two or three translation steps to produce the final compiled program. We can think of each of these steps as an intermediate level
 - A more sophisticated operating system (e.g., Windows) may run on top of a more simple operating system (e.g., DOS), adding extra capabilities (e.g., window management and graphical interfaces)



Virtual Machines (1)

- Some virtual machines only get implemented in software:
 - Why?
 - Examples?

Virtual Machines (1)

- Some virtual machines only get implemented in software:
 - Why? Because a hardware implementation is not cost-effective.
 - Examples? The virtual machines defined by C, Java, Python. They would be way more complex and expensive than typical hardware.
- Some virtual machines get first implemented in software, and then in hardware.
 - Why?

Virtual Machines (1)

- Some virtual machines only get implemented in software:
 - Why? Because a hardware implementation is not cost-effective.
 - Examples? The virtual machines defined by C, Java, Python. They would be way more complex and expensive than typical hardware.
- Some virtual machines get first implemented in software, and then in hardware.
 - Why? Because software implementations are much cheaper to make, and also much easier to test, debug, and modify.

Virtual Machines (2)

- Some virtual machines are used by the public in both hardware and software versions.
 - Why?

Virtual Machines (2)

- Some virtual machines are used by the public in both hardware and software versions.
 - Why? We may have a Macintosh computer, on which we want to run Windows software, or the other way around.

Compilation vs. Interpretation

- **Compilation:**

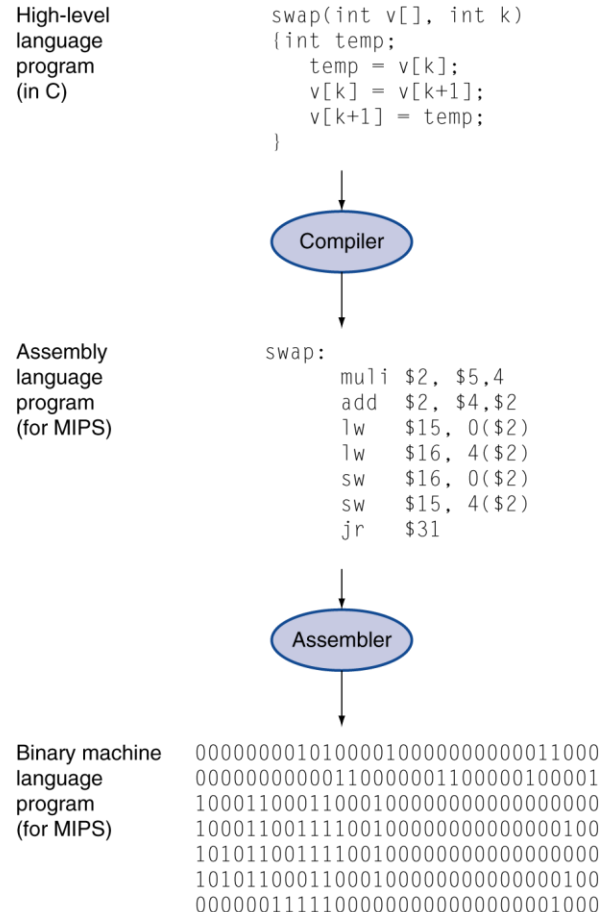
- n-level program is translated into a program at a lower level
- the program at the lower level is stored in memory, and executed
- while running, the lower-level program controls the computer

- **Interpretation:**

- An interpreter, implemented at a lower level, executes your n-level program line-by-line
- The interpreter translates each line into lower-level code, and executes that code
- The interpreter is the program that is running, not your code

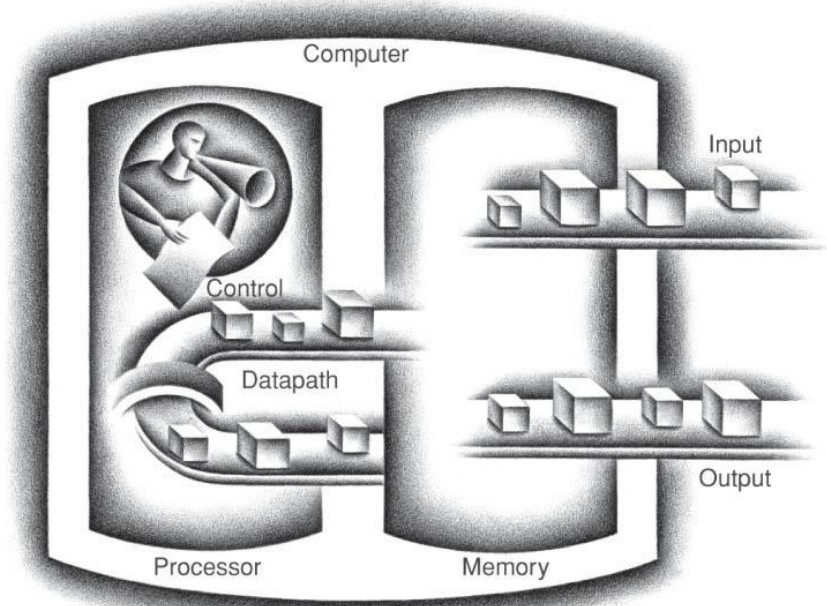
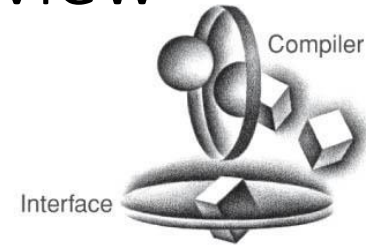
Levels of Program Code

- 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



Computer Organization Overview

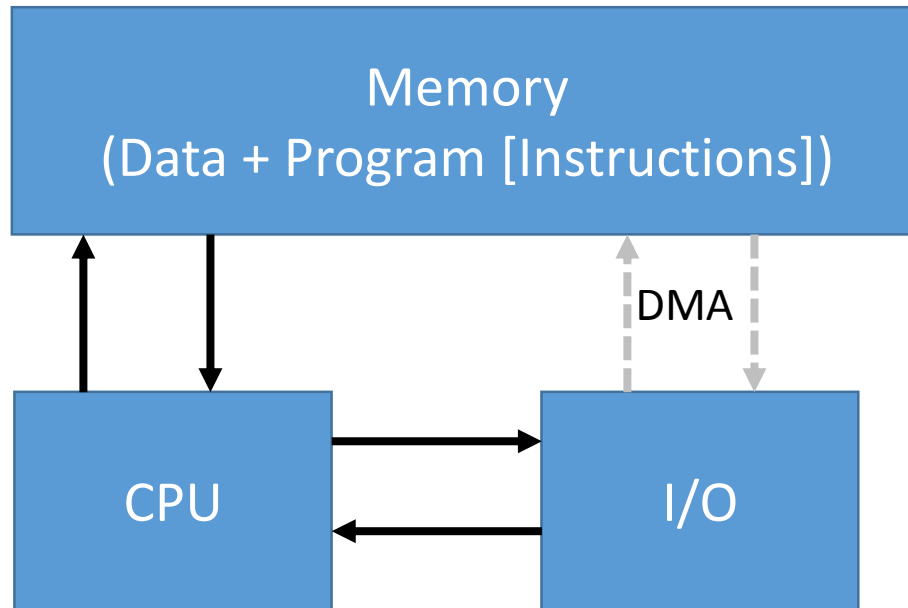
- CPU
 - Executes instructions
- Memory
 - Stores programs and data
- Buses
 - Transfers data
- Storage
 - Permanent
- I/O devices
 - Input: keypad, mouse, touch
 - Output: printer, screen
 - Both (input and output), such as:
 - USB, network, Wifi, touch screen, hard drive



Instruction Set Architectures

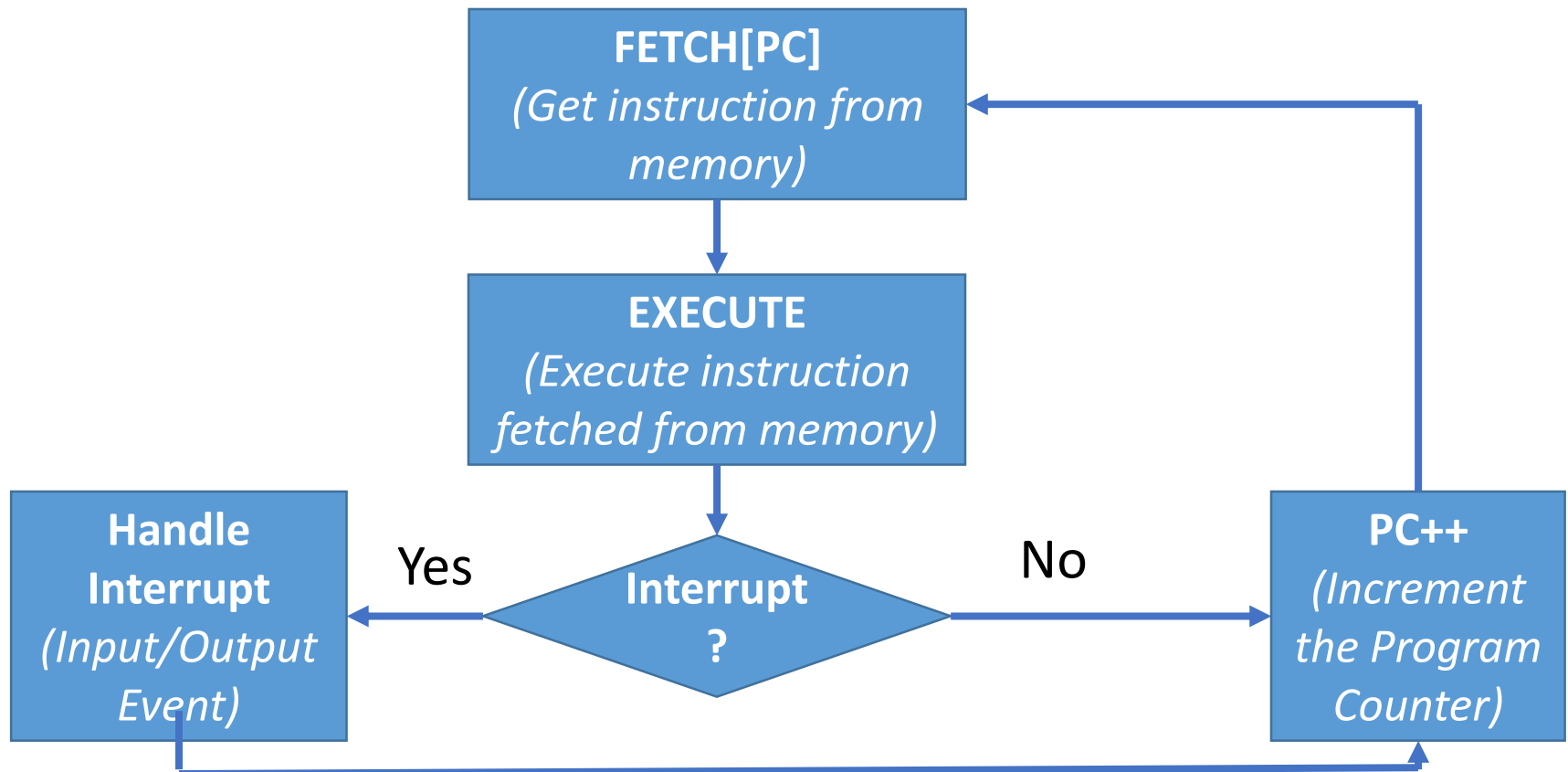
- Interface between software and hardware
- High-level language to computer instructions
 - How do we translate from a high-level language (e.g., C, Python, Java) to instructions the computer can understand?
 - Compilation (translation before execution)
 - Interpretation (translation-on-the-fly during execution)
 - What are examples of each of these?

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

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

OEMU

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.
```

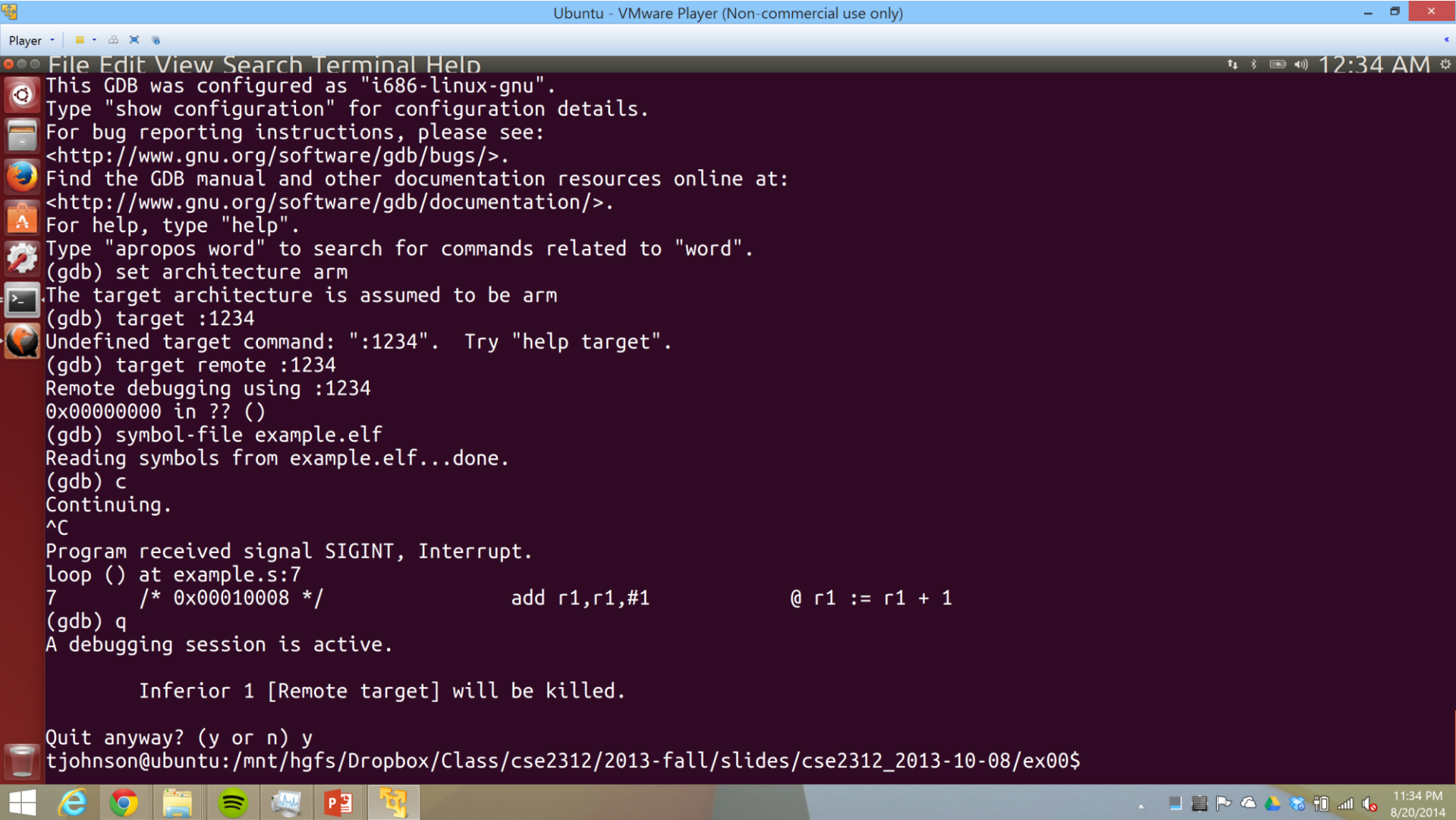
12:33 AM

Windows taskbar: e, Chrome, Spotify, PowerPoint, etc.

System tray: 11:33 PM, 8/20/2014

```

Ubuntu - VMware Player (Non-commercial use only)
Player
File Edit View Search Terminal Help
tjohnson@ubuntu:/mnt/hgfs/Dropbox/Class/cse2312/2013-fall/slides/cse2312_2013-10-08/ex00$ ls
ex00.tws      example.elf  example.list  example_mmap  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 ?? ()
    
```



```

Player - [Icons]
Ubuntu - VMware Player (Non-commercial use only)
File Edit View Search Terminal Help
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$
Windows [Icons] 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
^X Exit	^J Justify	^W Where Is	^V Next Page
			^K Cut Text
			^C Cur Pos
			^U UnCut Text
			^T To Spell

11:50 PM
8/20/2014

Announcements and Outline

- Quiz 1 on Blackboard site (due by end of week)
- Homework 1 on course website

- Review from last time
- Binary review
- 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)

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 (1 μ s): 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

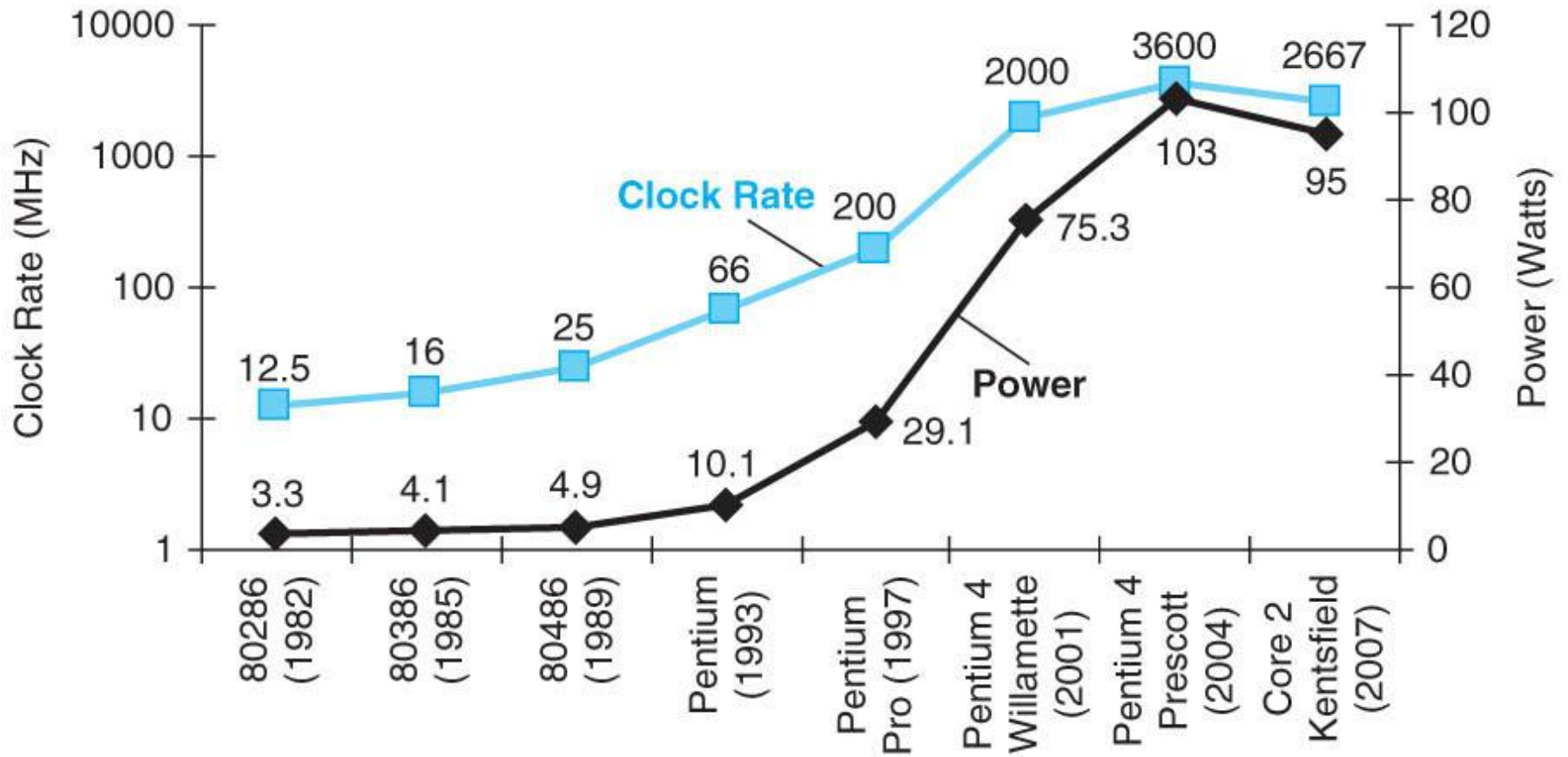
Units of Memory

- 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}	0.0000000000001	pico	10^{12}	1,000,000,000,000	tera
10^{-15}	0.0000000000000001	femto	10^{15}	1,000,000,000,000,000	peta
10^{-18}	0.0000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	exa
10^{-21}	0.00000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	zetta
10^{-24}	0.0000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	yotta

The Power Wall



Moore's Law for the Intel Family

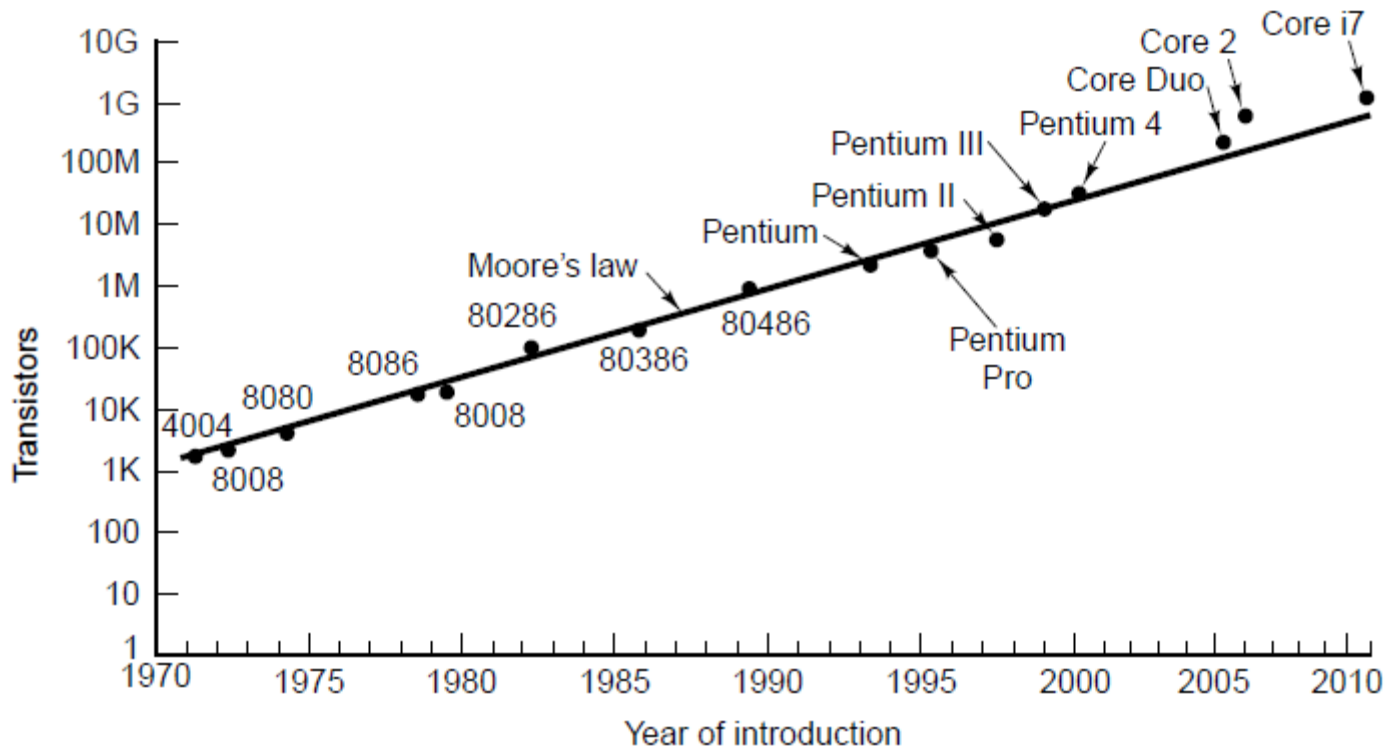


Figure 1-13. Moore's law for (Intel) CPU chips.

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.

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.

Moore's Law

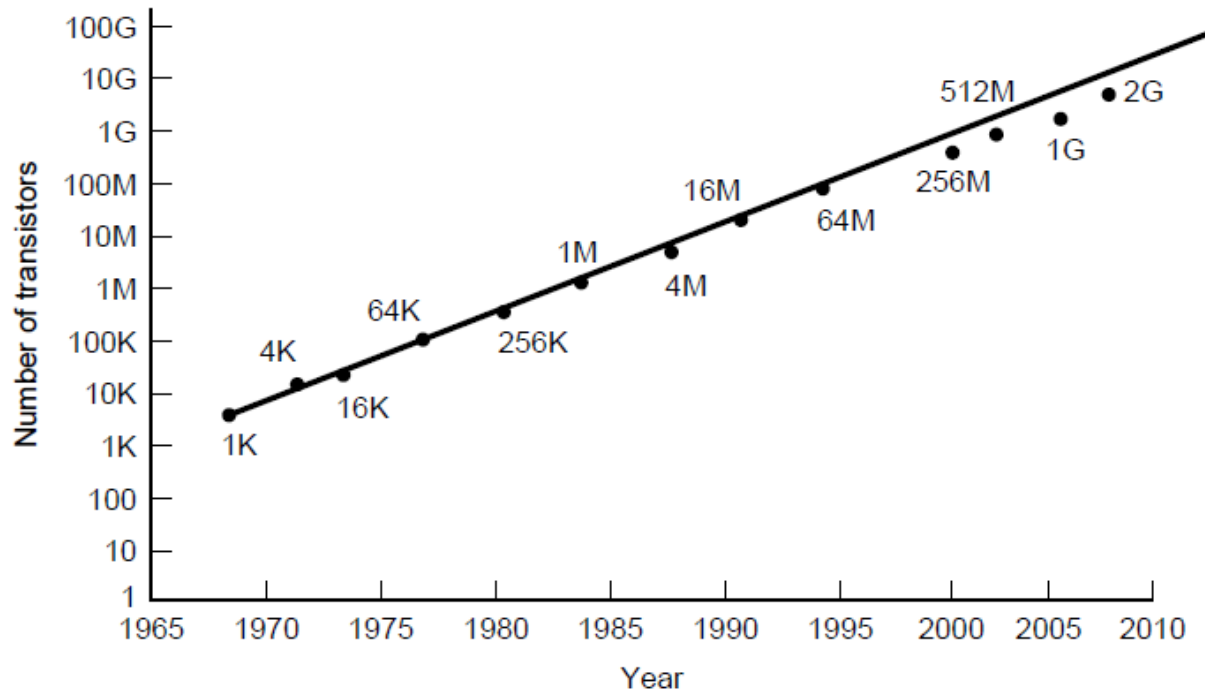
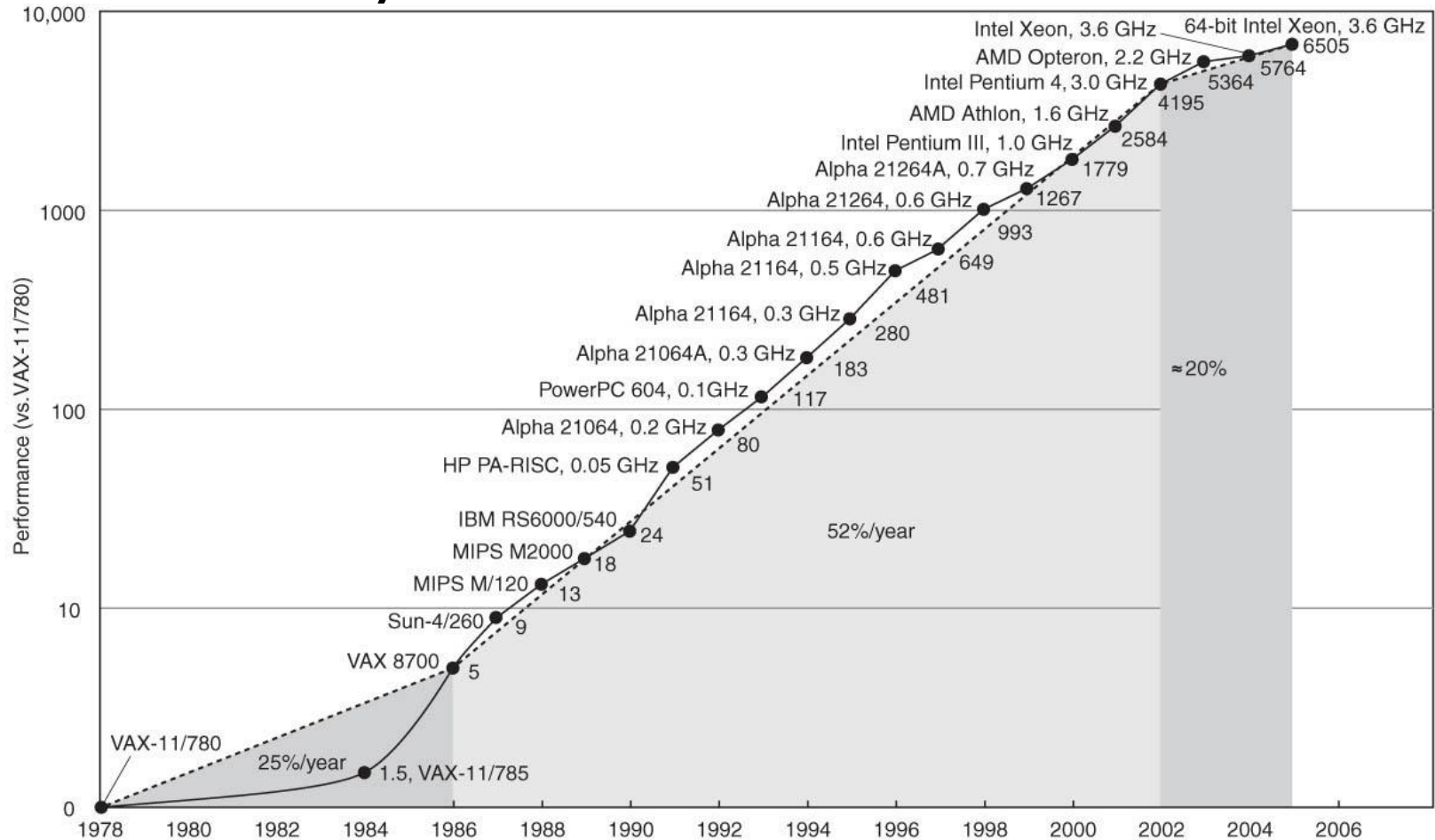


Figure 1-8. Moore's law predicts a 60 percent annual increase in the number of transistors that can be put on a chip. The data points given above and below the line are memory sizes, in bits.

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



SPECint2006 Benchmarks on AMD Opteron X4

Description	Name	Instruction Count $\times 10^9$	CPI	Clock cycle time (seconds $\times 10^9$)	Execution Time (seconds)	Reference Time (seconds)	SPE Ratio
Interpreted string processing	perl	2,118	0.75	0.4	637	9,770	15.3
Block-sorting compression	bzip2	2,389	0.85	0.4	817	9,650	11.8
GNU C compiler	gcc	1,050	1.72	0.4	724	8,050	11.1
Combinatorial optimization	mcf	336	10.00	0.4	1,345	9,120	6.8
Go game (AI)	go	1,658	1.09	0.4	721	10,490	14.6
Search gene sequence	hmmer	2,783	0.80	0.4	890	9,330	10.5
Chess game (AI)	sjeng	2,176	0.96	0.4	837	12,100	14.5
Quantum computer simulation	libquantum	1,623	1.61	0.4	1,047	20,720	19.8
Video compression	h264a vc	3,102	0.80	0.4	993	22,130	22.3
Discrete event simulation library	omnetpp	587	2.94	0.4	690	6,250	9.1
Games/path finding	astar	1,082	1.79	0.4	773	7,020	9.1
XML parsing	xalanbmk	1,058	2.70	0.4	1,143	6,900	6.0
Geometric Mean							11.7

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

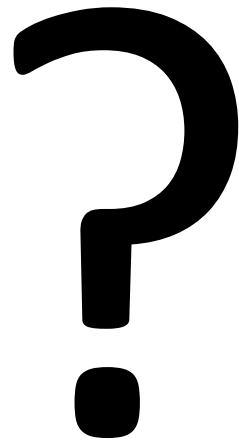
- "Software is a gas. It expands to fill the container holding it."
 - Nathan Myhrvold, former Microsoft executive.
- Software expands with additional features, to exploit new hardware capabilities.
- Software expansion creates need for better hardware.

Summary

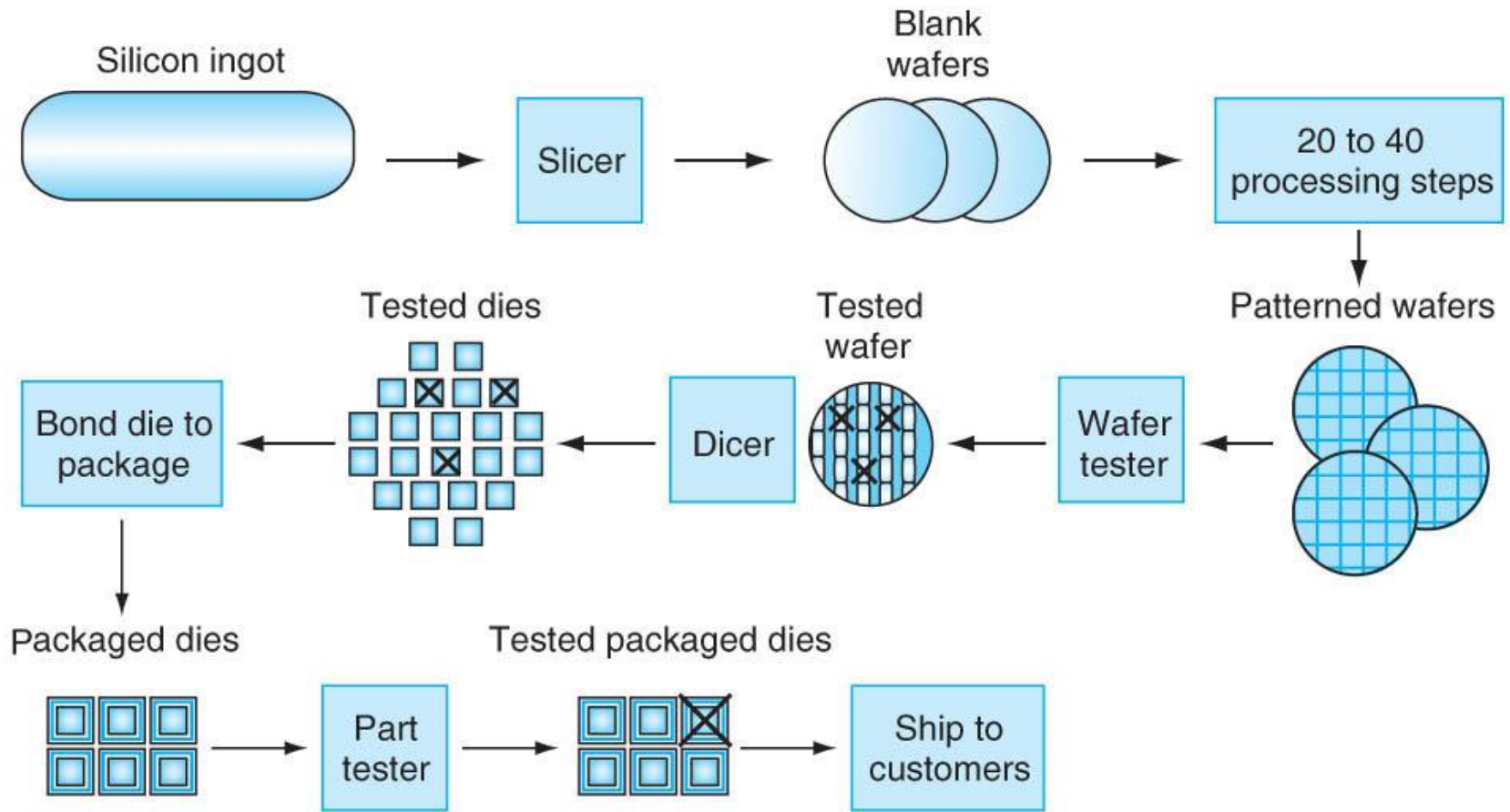
- Structured computers
- Binary and decimal numbers, ASCII
- Basic performance metrics, units

- Quiz 1 on Blackboard
- Homework 1

Questions?



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.

