CMSC 313
COMPUTER ORGANIZATION
&
ASSEMBLY LANGUAGE
PROGRAMMING

LECTURE 01, SPRING 2013

TOPICS TODAY

- Course overview
- Levels of machines
- Machine models: von Neumann & System Bus
- Fetch-Execute Cycle
- Base Conversion

COURSE OVERVIEW

Computer Organization & Assembly Language Programming

Instructor.

Prof. Richard Chang, chang@umbc.edu, 410-455-3093.

Office Hours: Tuesday & Thursday 11:30am–12:30pm, ITE 326.

Teaching Assistant.

Roshan Ghumare, roshang1@umbc.edu

Office Hours: TBA

Time and Place.

Section 01: Tu - Th 10:00am – 11:15am, ITE 229.

Section 02: Tu - Th 1:00pm – 2:15pm, ITE 229.

Textbook.

- *Essentials of Computer Organization and Architecture*, third edition, by Linda Null & Julia Lobur. Jones & Bartlett Learning, 2010. ISBN: 1449600069.
- *Assembly Language Step-by-Step: Programming with Linux*, third edition, by Jeff Duntemann. Wiley, 2009. ISBN: 0470497025.

Web Page. http://umbc.edu/~chang/cs313/

Catalog Description. This course introduces the student to the low-level abstraction of a computer system from a programmer's point of view, with an emphasis on low-level programming. Topics include data representation, assembly language programming, C programming, the process of compiling and linking, low-level memory management, exceptional control flow, and basic processor architecture.

Prerequisites. You should have mastered the material covered in the following courses: CMSC 202 Computer Science II and CMSC 203 Discrete Structures. You need the programming experience from CMSC202. Additional experience from CMSC341 Data Structures would also be helpful. You must also be familiar with and be able to work with truth tables, Boolean algebra and modular arithmetic.

Objectives. The purpose of this course is to introduce computer science majors to computing systems below that of a high-level programming language. The material covered can be broadly separated into the categories of assembly language programming, C programming and digital logic. These topics prepare the students to take CMSC411 Computer Architecture and CMSC421 Operating Systems which are required courses for the computer science major.

Under the heading of assembly language programming students will be introduced to the i386 instruction set, low-level programming, the Linux memory model, as well as the internal workings of compilers, assemblers and linkers.

C programming topics will concentrate on dynamic memory allocation.

Topics under computer organization include digital logic design (combinational circuits, sequential circuits, finite state machines) and basic computer architecture (system bus, memory hierarchy and input/output devices).

Grading. Your final grade will be based upon 5 homework assignments (15% total) and 8 programming assignments (40% total). There will also be a midterm exam (20%) and a final exam (25%). However, if some homework or programming assignments are canceled and not made up, the proportion of your grade from homework, projects and exams will remain the same. For example, if a programming assignment is canceled, then each programming assignment would be worth 5.714% (instead of 5%). That keeps programming assignments at 40% of your final grade.

Your final letter grade is based on the standard formula:

$$0 \le F < 60, \quad 60 \le D < 70, \quad 70 \le C < 80, \quad 80 \le B < 90, \quad 90 \le A \le 100$$

Depending upon the final distribution of grades in the class, there may be a curve in your favor, but under no circumstances will grades be curved downward.

Grades are given for work done *during* the semester; incomplete grades will only be given for medical illness or other dire circumstances.

Due Dates. There will be a homework assignment or programming assignment due every week of class (except the week after Spring Break). Written homework assignments are due at the beginning of lecture. Programming assignments and logic simulations are submitted online and are due at 11:59pm of the due date.

Late Assignments. Assignments turned in one day late (either submitted online or in person) will incur a 5% penalty. Assignments turned in two days late will be penalized 10%. Those three days late, 15%. For example, for a programming project due on Tuesday at 11:59pm:

Submitted:	**Penalty:**
Tuesday 11:59pm	0%
Wednesday 11:59pm	5%
Thursday 11:59pm	10%
Friday 11:59pm	15%
after Saturday 12:01am	100%

Late assignments will not be accepted after 3 days. However, each student may submit one assignment (of any kind) up to one week late without penalty using his/her *one time late pass*.

Academic Integrity. You are allowed to discuss the homework assignments with other students. However, circuit simulation exercises and programming projects must be completed by individual effort. (See the Academic Integrity Policy handout.) Furthermore, you must write up your homework ***independently***. This means you should only have the textbooks and your own notes in front of you when you write up your homework — not your friend's notes, your friend's homework or other reference material. You should not have a copy of someone else's homework or project ***under any circumstance***. For example, you should not let someone turn in your homework.

The UMBC Undergraduate Student Academic Conduct Policy is available at:

`http://www.umbc.edu/undergrad_ed/ai/documents/ACC2011.pdf`

Exams. The midterm exam has been scheduled for Thursday, March 14. The final exam is on Tuesday, May 21. For Section 01 (TuTh 10am), the time of the final exam is 10:30am – 12:30pm. For Section 02 (TuTh 1pm), the final exam is 1pm – 3pm.

The following schedule outlines the material to be covered during the semester and specifies the corresponding sections in the textbooks: *Essentials of Computer Organization and Architecture (CO&A)*, by Null & Lobur and *Assembly Language Step-by-Step (ALSbS)*, by Dunteman.

		Reading		Homework	
Date	**Topic**	**CO&A**	**ALSbS**	**Assign**	**Due**
Tue 01/29	Data Representation I	1.1–1.8		HW1	
Thu 01/31	Data Representation II	2.1-2.4, 2.6			
Tue 02/05	i386 Assembly Language I	4.14	Ch 3-4	HW2	HW1
Thu 02/07	i386 Assembly Language II		Ch 5-6		
Tue 02/12	i386 Assembly Language III		Ch 7-8	Proj1	HW2
Thu 02/14	i386 Assembly Language IV		Ch 9		
Tue 02/19	A Bigger Example			Proj2	Proj1
Thu 02/21	Subroutines		Ch 10		
Tue 02/26	Interrupts	7.4		Proj3	Proj2
Thu 02/28	C Programming I				
Tue 03/05	C Programming II			Proj4	Proj3
Thu 03/07	C Programming III				
Tue 03/12	C Programming IV				Proj4
Thu 03/14	**Midterm Exam**				
Tue 03/19	*Spring Break*				
Thu 03/21	*Spring Break*				
Tue 03/26	C Programming V			Proj5	
Thu 03/28	C Programming VI				
Tue 04/02	C & Assembly Language		Ch 12	Proj6	Proj5
Thu 04/04	Function Pointers				
Tue 04/09	Polymorphism in C			Proj7	Proj6
Thu 04/11	Introduction to Digital Logic	3.1–3.3			
Tue 04/16	Transistors & Logic Gates			Proj8	Proj7
Thu 04/18	Circuits for Addition	3.4-3.5			
Tue 04/23	Flip Flops	3.6		HW3	Proj8
Thu 04/25	Finite State Machines				
Tue 04/30	Finite State Machine Design			HW4	HW3
Thu 05/02	Towards a CPU				
Tue 05/07	Cache & Virtual Memory I	6.1– 6.4		HW5	HW4
Thu 05/09	Cache & Virtual Memory II	6.5			
Tue 05/14	Review				HW5
Tue 05/21	**Final Exam** (Section 01 10:30am – 12:30pm, Section 02 1pm – 3pm)				

Critical programming skills cannot be learned by attending lecture. You should budget enough time to work on the programming assignments as well. Please consult the time table given on the syllabus and plan ahead. Programs are due by midnight (1 minute after 11:59pm) of the due date. Programs will be submitted using the submit system running on the GL machines. Programs will be graded on five criteria: correctness, design, style, documentation and efficiency. So, turning in a project that merely "works" is not sufficient to receive full credit.

For this course, programming projects must be developed using the NASM assembler for the 32-bit Linux operating system running on an Intel CPU. This arrangement is not compatible with other flavors of UNIX, 64-bit Linux, Linux running on non-Intel CPUs or with assemblers for Microsoft Windows. When in doubt the UMBC machine linux.gl.umbc.edu will be the final arbiter of what constitutes a working program. You may work on your own machines running Linux, but you will have to be your own system administrator. None of the instructors, TA or support staff at OIT will be available to help you install or debug Linux.

Academic Integrity.

Read this section carefully! It describes what constitutes cheating for this course. If you have questions, ask the instructor. Ignorance will not be accepted as an excuse after the fact.

When you submit your homework and programming assignments, you are stating that the work was created by your own individual effort.

Receiving help from this class's instructor, teaching assistants or from the Computer Science Help Center does not violate this academic integrity policy.

You may also receive help from other sources. However, this help must be limited to:

- Discussions about the meaning of the assignment.
- Identifying syntax errors in your program.
- Identifying simple logic errors in your program.

The following is a non-exhaustive list of actions that clearly violate this academic integrity policy:

- Someone else is typing code in your program.
- You are cutting and pasting more than a single line of code (from a program that was not distributed by the instructor).
- You are looking at someone else's program while you are typing in your code.
- You receive someone else's program by email, hard copy, text message, instant message, ...
- You make your program available to another student in CMSC 313 directly or indirectly by email, hard copy, text message, instant message, ...

This policy recognizes that students can learn productively from many sources including from other students in the class. Thus, this policy allows small amounts of help but prohibits outright copying. Although, this leaves a gray area between "small amounts of help" and "outright copying", it is better that we live with some ambiguity than to have a clear-cut policy that deprives the students of productive learning opportunities. Students who have doubts about the propriety of an activity should consult the instructor.

Students who violate this academic integrity policy will receive a grade of 0 for that assignment. A second violation will also result in a reduction of one full letter grade in the student's final course grade.

In the case where one student copies the program of another student, **both students** are considered to have violated this policy. Here, copying includes not just programs that are verbatim copies, but also programs that are substantially similar and could not have been produced independently. Furthermore, all parties concerned will have their **prior homework and programs checked.**

Violations of this policy may be reported to the University's Academic Conduct Committee for further action. Egregious cases of cheating will be written up as a "more serious" infraction. In this case, you will not be allowed to drop the course. Also, a "more serious" infraction would appear as a permanent part of your student record and would be seen by potential employers when they ask for an official copy of your transcript.

The UMBC Undergraduate Student Academic Conduct Policy is available at:

`http://www.umbc.edu/undergrad_ed/ai/documents/ACC2011.pdf`

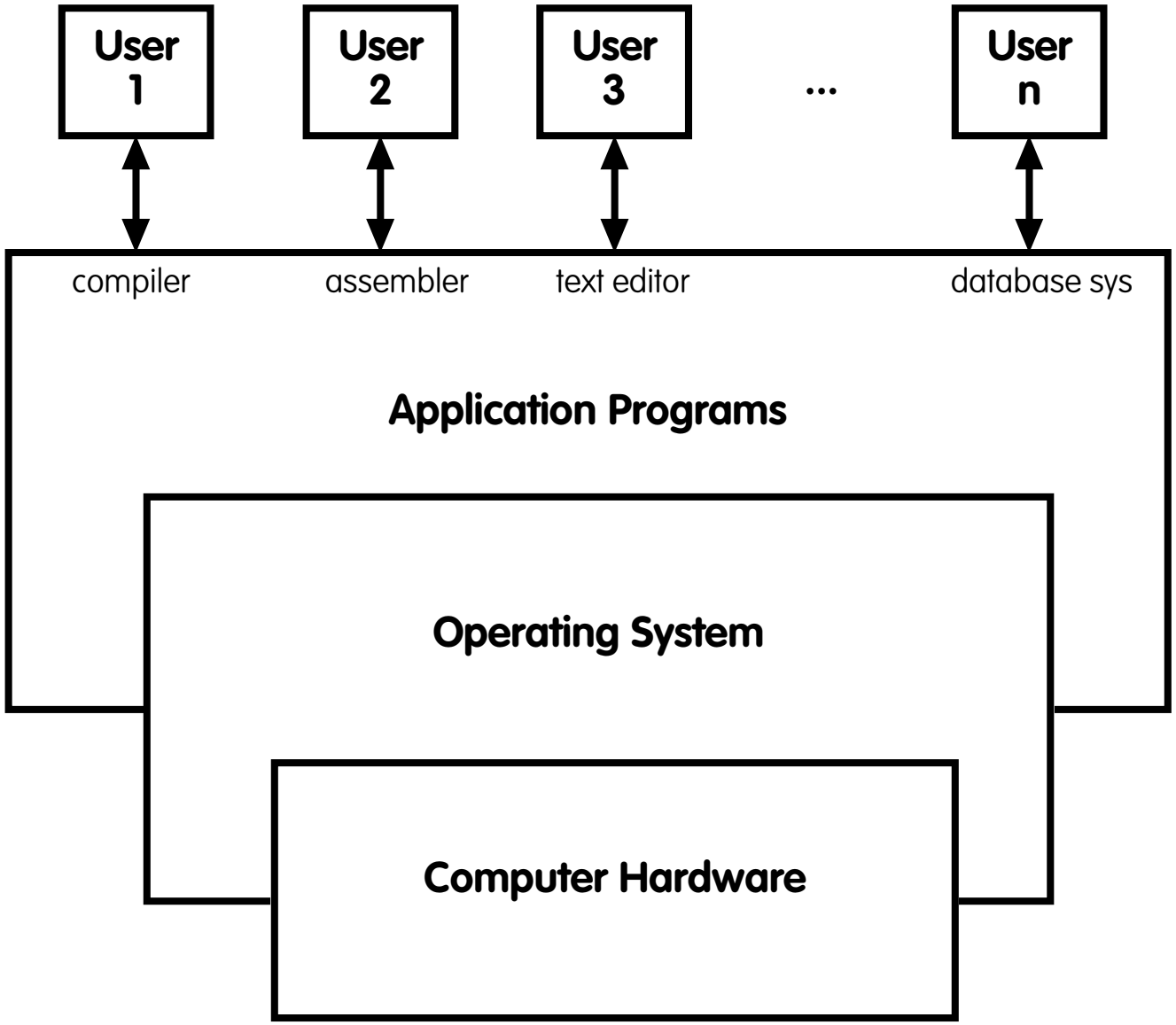
ATTENDANCE

ATTENDANCE POLICY

ATTENDANCE POLICY (CONT'D)

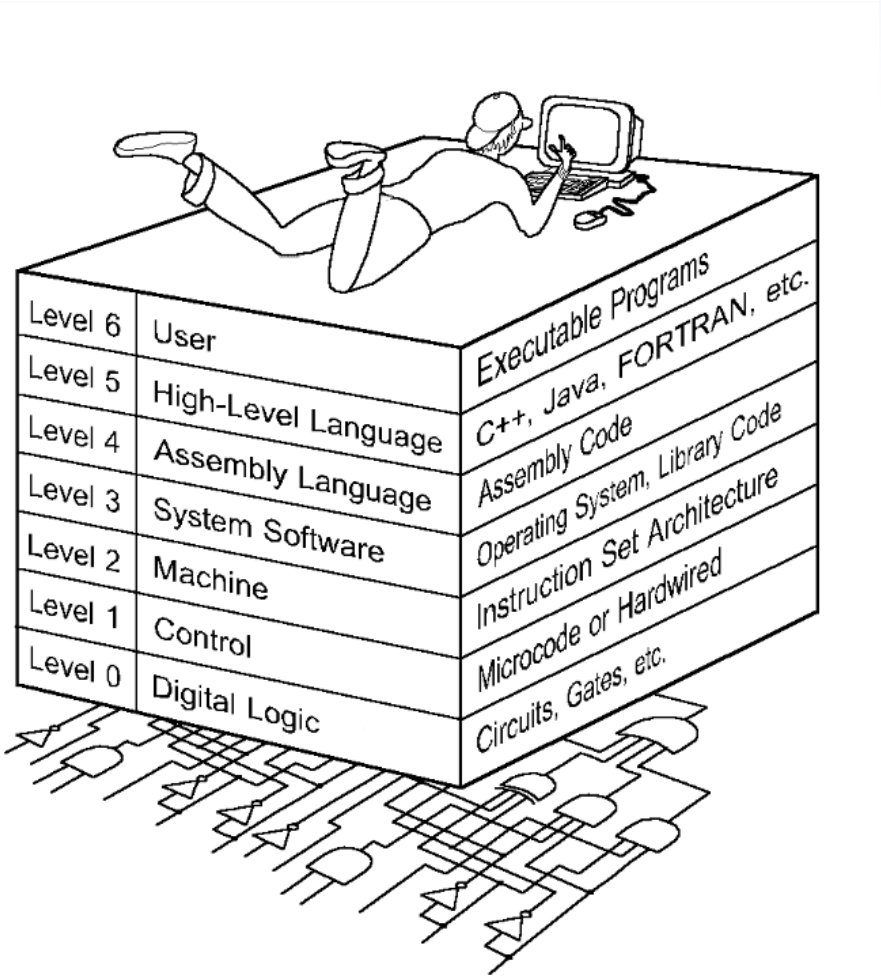
LEVELS OF MACHINES

Computer Science View of the World



1.6 The Computer Level Hierarchy

- **Each virtual machine layer is an abstraction of the level below it.**
- **The machines at each level execute their own particular instructions, calling upon machines at lower levels to perform tasks as required.**
- **Computer circuits ultimately carry out the work.**



1.6 The Computer Level Hierarchy

- Level 6: The User Level
  - Program execution and user interface level.
  - The level with which we are most familiar.
- Level 5: High-Level Language Level
  - The level with which we interact when we write programs in languages such as C, Pascal, Lisp, and Java.

1.6 The Computer Level Hierarchy

- Level 4: Assembly Language Level
  - Acts upon assembly language produced from Level 5, as well as instructions programmed directly at this level.
- Level 3: System Software Level
  - Controls executing processes on the system.
  - Protects system resources.
  - Assembly language instructions often pass through Level 3 without modification.

1.6 The Computer Level Hierarchy

- Level 2: Machine Level
  - Also known as the Instruction Set Architecture (ISA) Level.
  - Consists of instructions that are particular to the architecture of the machine.
  - Programs written in machine language need no compilers, interpreters, or assemblers.

1.6 The Computer Level Hierarchy

- Level 1: Control Level
  - A *control unit* decodes and executes instructions and moves data through the system.
  - Control units can be *microprogrammed* or *hardwired*.
  - A microprogram is a program written in a low-level language that is implemented by the hardware.
  - Hardwired control units consist of hardware that directly executes machine instructions.

1.6 The Computer Level Hierarchy

- Level 0: Digital Logic Level
  - This level is where we find digital circuits (the chips).
  - Digital circuits consist of gates and wires.
  - These components implement the mathematical logic of all other levels.

MACHINE MODELS

1.7 The von Neumann Model

- On the ENIAC, all programming was done at the digital logic level.
- Programming the computer involved moving plugs and wires.
- A different hardware configuration was needed to solve every unique problem type.

Configuring the ENIAC to solve a "simple" problem required many days labor by skilled technicians.

1.7 The von Neumann Model

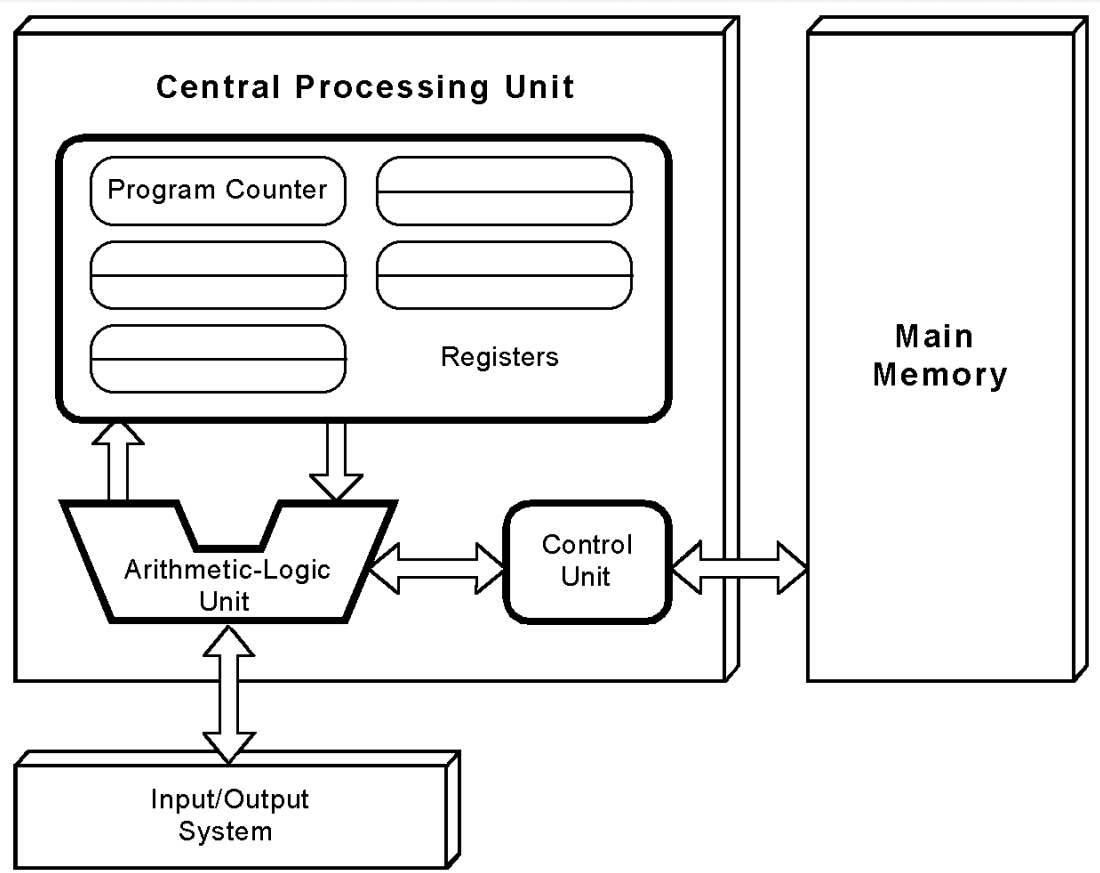
- Inventors of the ENIAC, John Mauchley and J. Presper Eckert, conceived of a computer that could store instructions in memory.
- The invention of this idea has since been ascribed to a mathematician, John von Neumann, who was a contemporary of Mauchley and Eckert.
- Stored-program computers have become known as von Neumann Architecture systems.

1.7 The von Neumann Model

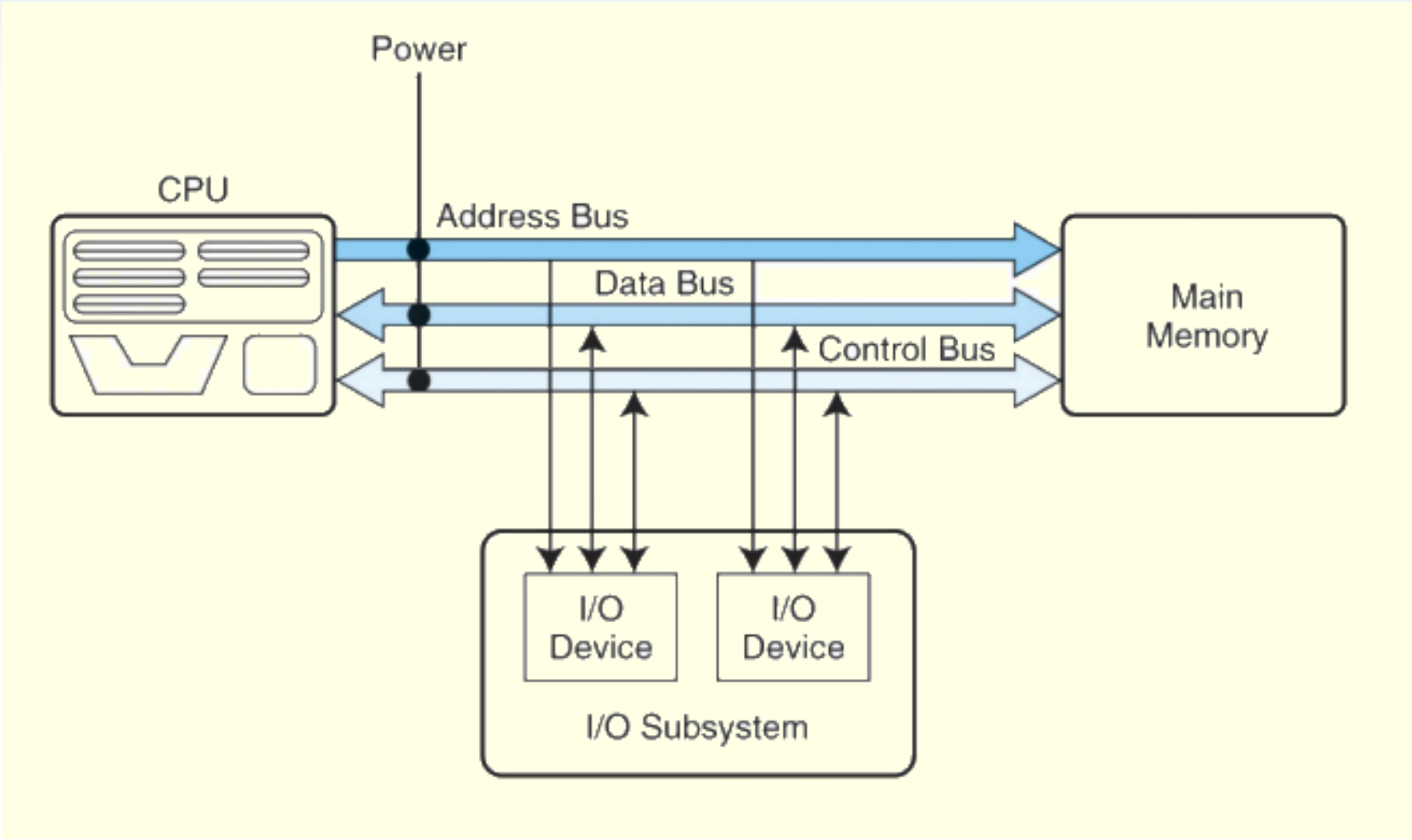
- Today's stored-program computers have the following characteristics:
  - Three hardware systems:
    - A central processing unit (CPU)
    - A main memory system
    - An I/O system
  - The capacity to carry out sequential instruction processing.
  - A single data path between the CPU and main memory.
    - This single path is known as the *von Neumann bottleneck*.

1.7 The von Neumann Model

- **This is a general depiction of a von Neumann system:**
- **These computers employ a fetch-decode-execute cycle to run programs as follows . . .**

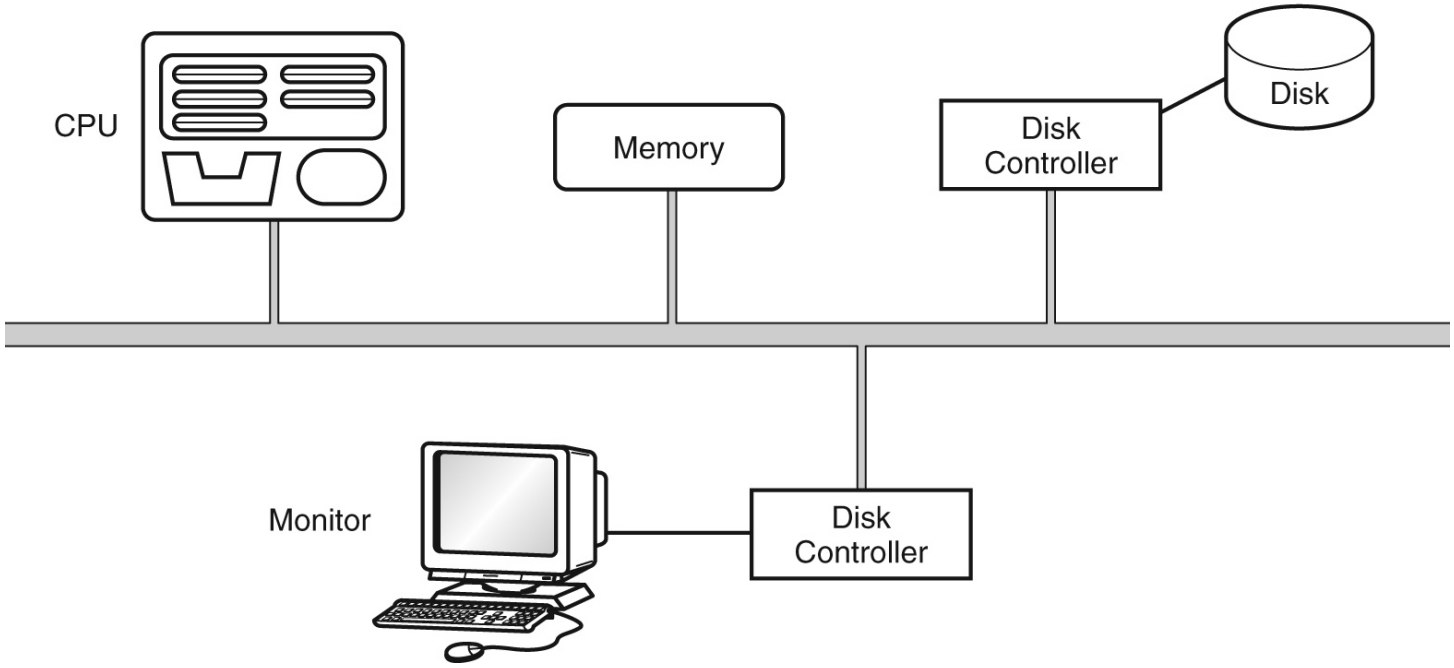


4.3 The Bus



4.3 The Bus

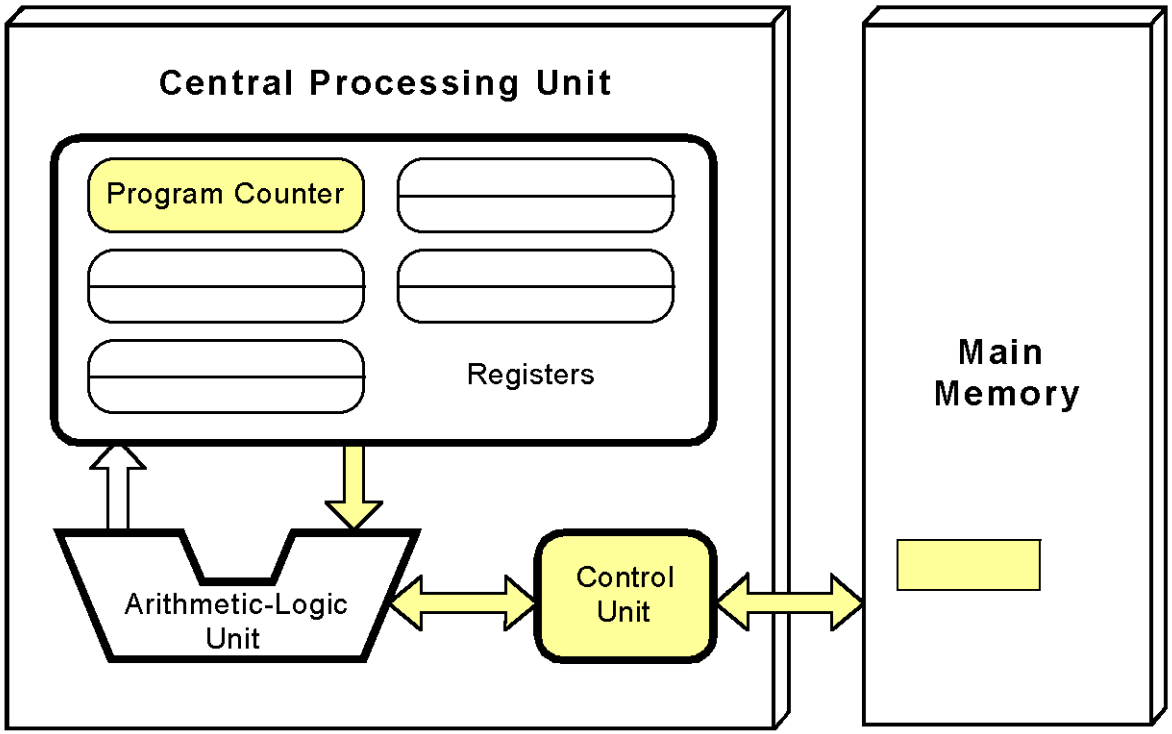
- A multipoint bus is shown below.
- Because a multipoint bus is a shared resource, access to it is controlled through protocols, which are built into the hardware.



FETCH EXECUTE CYCLE

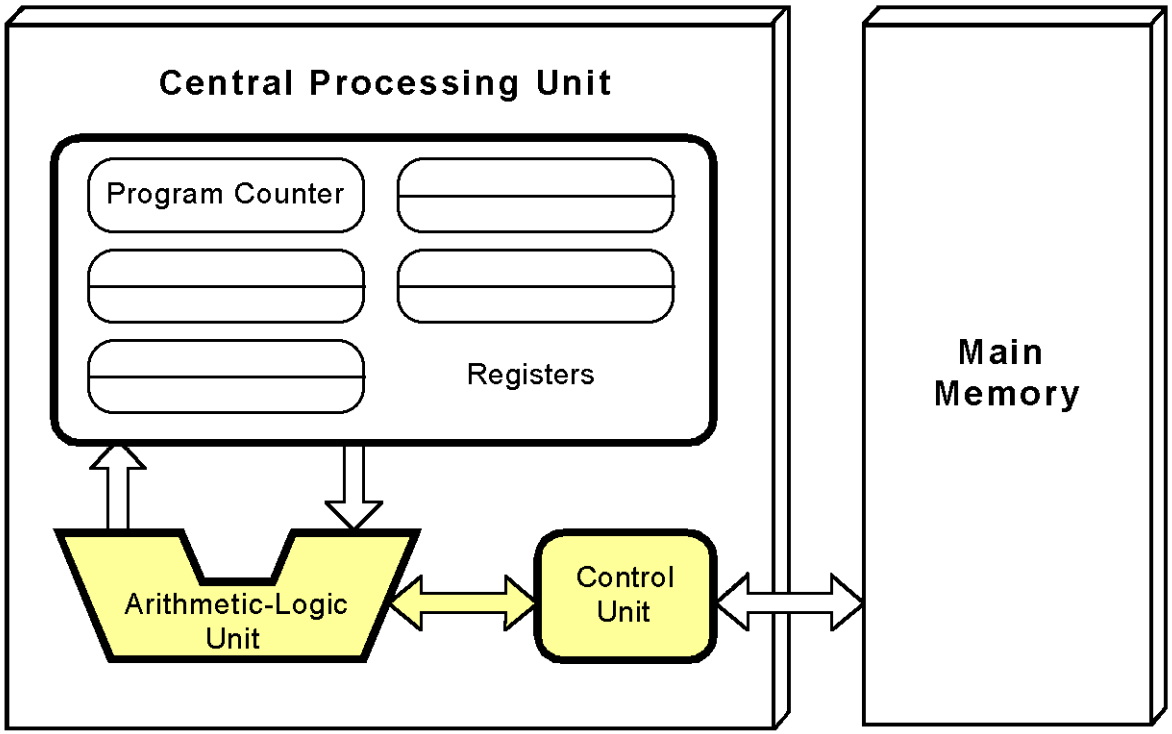
1.7 The von Neumann Model

- **The control unit fetches the next instruction from memory using the program counter to determine where the instruction is located.**



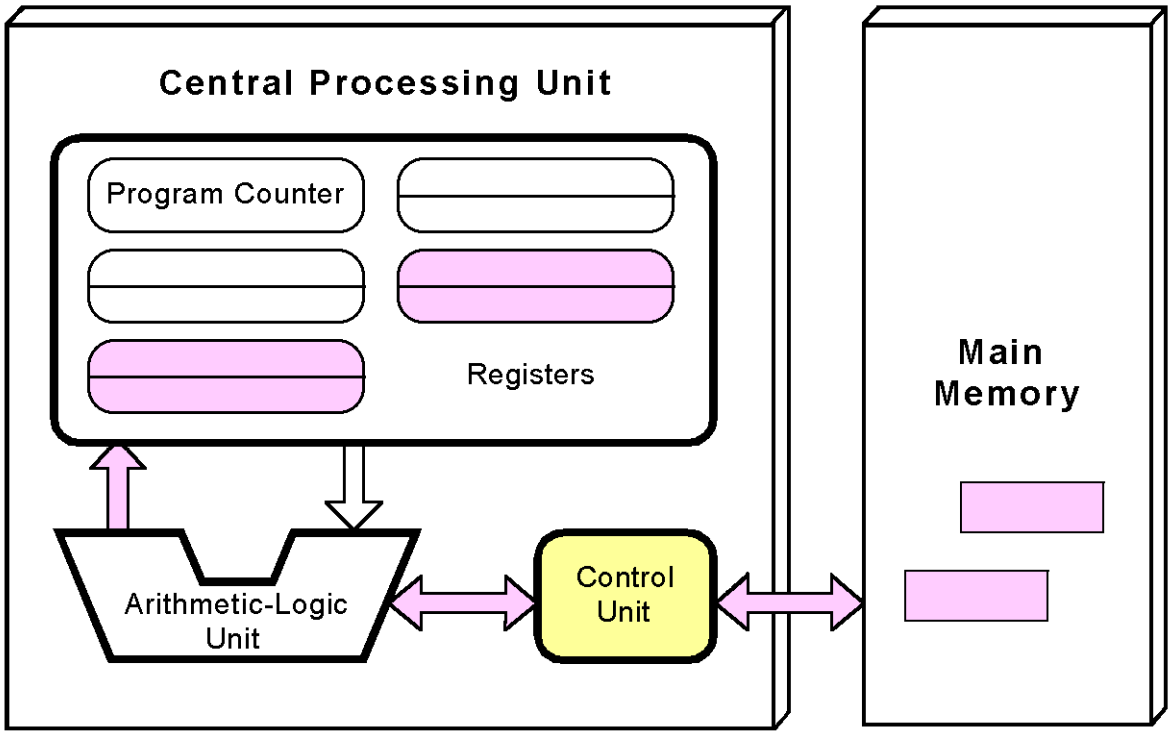
1.7 The von Neumann Model

- **The instruction is decoded into a language that the ALU can understand.**



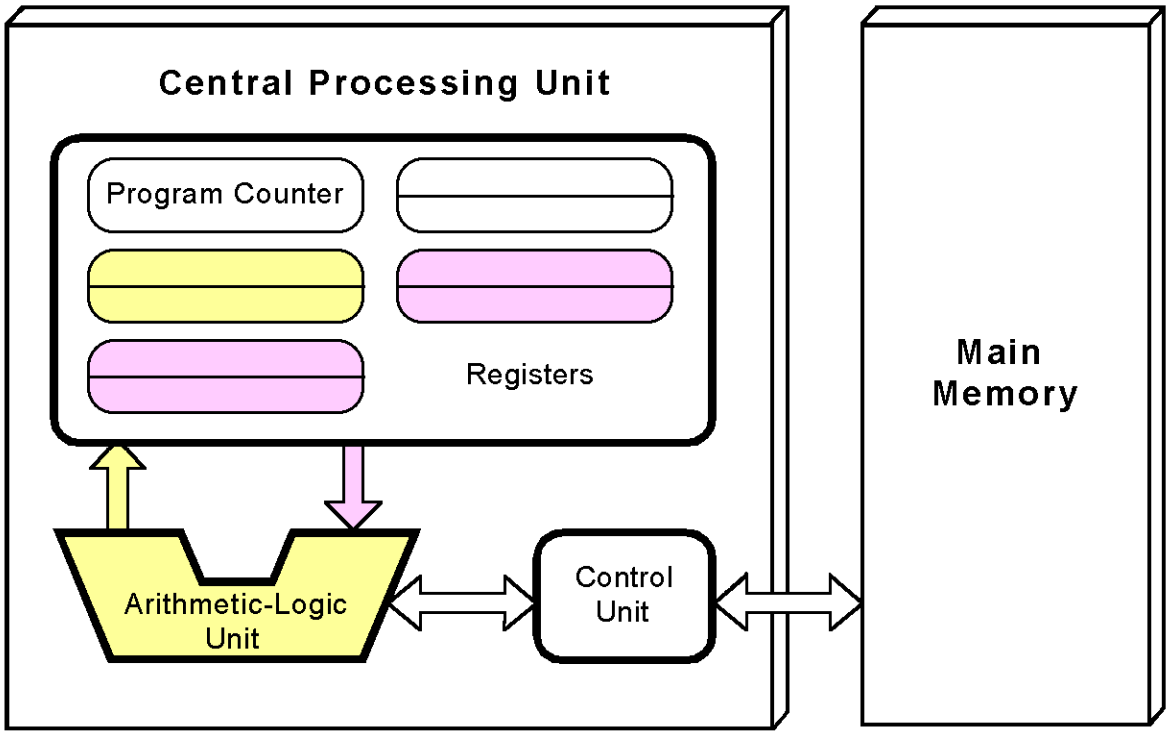
1.7 The von Neumann Model

- **Any data operands required to execute the instruction are fetched from memory and placed into registers within the CPU.**



1.7 The von Neumann Model

- **The ALU executes the instruction and places results in registers or memory.**



BASE CONVERSION

2.1 Introduction

- A *bit* is the most basic unit of information in a computer.
  - It is a state of "on" or "off" in a digital circuit.
  - Sometimes these states are "high" or "low" voltage instead of "on" or "off.."
- A *byte* is a group of eight bits.
  - A byte is the smallest possible *addressable* unit of computer storage.
  - The term, "addressable," means that a particular byte can be retrieved according to its location in memory.

2.1 Introduction

- A *word* is a contiguous group of bytes.
  - Words can be any number of bits or bytes.
  - Word sizes of 16, 32, or 64 bits are most common.
  - In a word-addressable system, a word is the smallest addressable unit of storage.
- A group of four bits is called a *nibble*.
  - Bytes, therefore, consist of two nibbles: a "high-order nibble," and a "low-order" nibble.

2.2 Positional Numbering Systems

- Bytes store numbers using the position of each bit to represent a power of 2.
  - The binary system is also called the base-2 system.
  - Our decimal system is the base-10 system. It uses powers of 10 for each position in a number.
  - Any integer quantity can be represented exactly using any base (or *radix*).

2.2 Positional Numbering Systems

- The decimal number 947 in powers of 10 is:

$$9 \times 10^2 + 4 \times 10^1 + 7 \times 10^0$$

- The decimal number 5836.47 in powers of 10 is:

$$5 \times 10^3 + 8 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 + 4 \times 10^{-1} + 7 \times 10^{-2}$$

2.2 Positional Numbering Systems

- The binary number 11001 in powers of 2 is:

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

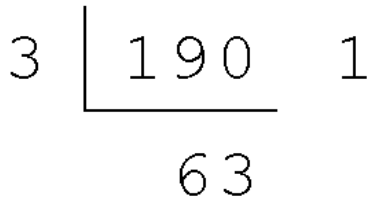
$$= 16 + 8 + 0 + 0 + 1 = 25$$

- When the radix of a number is something other than 10, the base is denoted by a subscript.
  - Sometimes, the subscript 10 is added for emphasis:

$$11001_2 = 25_{10}$$

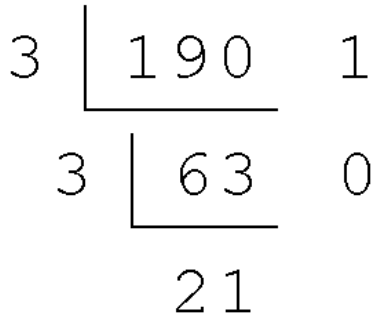
2.3 Converting Between Bases

- **Converting 190 to base 3...**
  - First we take the number that we wish to convert and divide it by the radix in which we want to express our result.
  - In this case, 3 divides 190 63 times, with a remainder of 1.
  - Record the quotient and the remainder.



2.3 Converting Between Bases

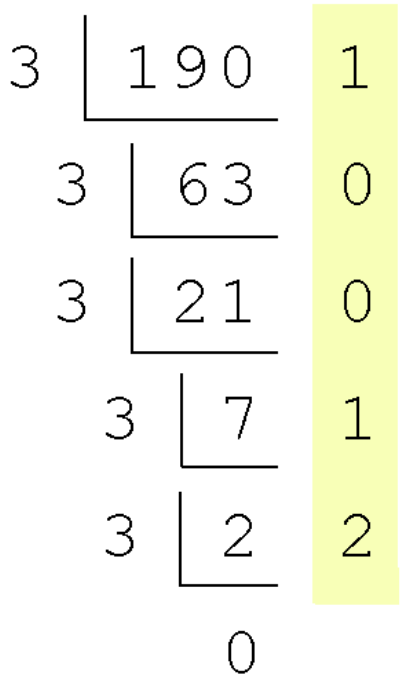
- **Converting 190 to base 3...**
  - 63 is evenly divisible by 3.
  - Our remainder is zero, and the quotient is 21.



2.3 Converting Between Bases

- **Converting 190 to base 3...**
  - Continue in this way until the quotient is zero.
  - In the final calculation, we note that 3 divides 2 zero times with a remainder of 2.
  - Our result, reading from bottom to top is:

$$190_{10} = 21001_3$$



2.3 Converting Between Bases

- **Using the multiplication method to convert the decimal 0.8125 to binary, we multiply by the radix 2.**
  - The first product carries into the units place.

```
  .8125
×    2
------
1.6250
```

2.3 Converting Between Bases

- **Converting 0.8125 to binary . . .**
  - Ignoring the value in the units place at each step, continue multiplying each fractional part by the radix.

```
  .8125
  ×   2
1.6250

  .6250
  ×   2
1.2500

  .2500
  ×   2
0.5000
```

2.3 Converting Between Bases

- **Converting 0.8125 to binary . . .**
  - You are finished when the product is zero, or until you have reached the desired number of binary places.
  - Our result, reading from top to bottom is:

    $0.8125_{10} = 0.1101_2$
  - This method also works with any base. Just use the target radix as the multiplier.

```
  .8125
  ×   2
1.6250

  .6250
  ×   2
1.2500

  .2500
  ×   2
0.5000

  .5000
  ×   2
1.0000
```

Converting Base 6 to Base 10

- $123.45_6 = ???.???_{10}$

$123_6 = 1 \times 36_{10} + 2 \times 6_{10} + 3 \times 1_{10} = 51_{10}$

$0.45_6 = 4 \times 1/6_{10} + 5 \times 1/36_{10} = 0.805555..._{10}$

$123.45_6 = 51.805555..._{10}$

Converting Base 10 to Base 6

- $754.94_{10} = 3254.5\ 35012\ 35012\ 35012..._6$

$754_{10} = 11_6 \times 244_6 + 5_6 \times 14_6 + 4_6 \times 1_6 = ???_6$

$754 \div 6 = 125$ remainder 4

$125 \div 6 = 20$ remainder 5

$20 \div 6 = 3$ remainder 2

$3 \div 6 = 0$ remainder 3

$3254_6 = 3 \times 216_{10} + 2 \times 36_{10} + 5 \times 6_{10} + 4 \times 1 = 754_{10}$

Converting Base 10 to Base 6 (cont)

- $0.94_{10} = ???.???_6$

$0.94 \times 6 = 5.64 \rightarrow 5$

$0.64 \times 6 = 3.84 \rightarrow 3$

$0.84 \times 6 = 5.04 \rightarrow 5$

$0.04 \times 6 = 0.24 \rightarrow 0$

$0.24 \times 6 = 1.44 \rightarrow 1$

$0.44 \times 6 = 2.64 \rightarrow 2$

$0.64 \times 6 = 3.84 \rightarrow 3$

$0.94_{10} = 0.5\ 35012\ 35012\ 35012..._6$

$5/6 + 3/36 + 5/216 + 0 + 1/6^5 + 2/6^6 = 0.939986282..._{10}$

2.3 Converting Between Bases

- The binary numbering system is the most important radix system for digital computers.
- However, it is difficult to read long strings of binary numbers -- and even a modestly-sized decimal number becomes a very long binary number.
  - For example: $11010100011011_2 = 13595_{10}$
- For compactness and ease of reading, binary values are usually expressed using the hexadecimal, or base-16, numbering system.

BASES

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

2.3 Converting Between Bases

- The hexadecimal numbering system uses the numerals 0 through 9 and the letters A through F.
  - The decimal number 12 is C_{16}.
  - The decimal number 26 is $1A_{16}$.
- It is easy to convert between base 16 and base 2, because $16 = 2^4$.
- Thus, to convert from binary to hexadecimal, all we need to do is group the binary digits into groups of four.

A group of four binary digits is called a hextet

2.3 Converting Between Bases

- Using groups of hextets, the binary number 11010100011011_2 ($= 13595_{10}$) in hexadecimal is:

0011	0101	0001	1011
3	5	1	B

If the number of bits is not a multiple of 4, pad on the left with zeros.

- Octal (base 8) values are derived from binary by using groups of three bits ($8 = 2^3$):

011	010	100	011	011
3	2	4	3	3

Octal was very useful when computers used six-bit words.

NEXT TIME

- Representing numbers
- Representing negative numbers
- Floating point numbers (briefly)
- Characters and strings