CMSC 313 Lecture 01

- Course Overview
  - Prerequisites, Syllabus, Grading, Project Policy, etc.
- Levels of machines: from electrons to C++
- Machine Models: von Neumann vs. System Bus
- Fetch-Execute Cycle
- Review of base conversion
- Representing negative numbers

Course Syllabus

We will follow two textbooks: *Principles of Computer Architecture,* by Murdocca and Heuring, and *Linux Assembly Language Programming,* by Neveln. The following schedule outlines the material to be covered during the semester and specifies the corresponding sections in each textbook.

Date	Topic	M&H	Neveln	Assign	Due
Th 09/02	Introduction & Overview	1.1-1.8	1.1-1.6		
Tu 09/07	Data Representation I	2.1-2.2, 3.1-3.3	2.4-2.7, 3.6-3.8	HW1	
Th 09/09	Data Representation II				
Tu 09/14	i386 Assembly Language I		3.10-3.13, 4.1-4.8	HW2	HW1
Th 09/16	i386 Assembly Language II		6.1-6.5	Proj1	
Tu 09/21	i386 Assembly Language III				HW2
Th 09/23	i386 Assembly Language IV			Proj2	Proj1
Tu 09/28	Examples				
Th 09/30	Machine Language		5.1-5.7	Proj3	Proj2
Tu 10/05	Compiling, Assembling & Linking	5.1-5.3			
Th 10/07	Subroutines		7.1-7.4		
Tu 10/12	The Stack & C Functions				
Th 10/14	Linux Memory Model	7.7	8.1-8.8	Proj4	Proj3
Tu 10/19	Interrupts & System Calls		9.1-9.8		
Th 10/21	Cache Memory	7.6			Proj4
Tu 10/26	**Midterm Exam**				
Th 10/28	Introduction to Digital Logic	A.1-A.3	3.1-3.3	DigSim1	
Tu 11/02	Transistors & Logic Gates	A.4-A.7			
Th 11/04	Circuits for Addition	3.5		HW3	DigSim1
Tu 11/09	Combinational Logic Components	A.10			
Th 11/11	Circuit Simplification I	B.1-B.2		HW4	HW3
Tu 11/16	Flip Flops I	A.11			
Th 11/18	Flip Flops II			DigSim2	HW4
Tu 11/23	Finite State Machines	A.12-A.13			
Th 11/25	*Thanksgiving break*				
Tu 11/30	Circuit Simplification II	B.3		HW5	DigSim2
Th 12/02	Finite State Machine Design				
Tu 12/07	Registers & Memory	A.14-15, 7.1-7.5		DigSim3	HW5
Th 12/09	I/O	8.1-8.3			
Tu 12/14	TBA				DigSim3
Tu 12/21	**Final Exam 10:30am-12:30pm**				

Course Description

Instructor:	Prof. Richard Chang	**TA:**	Mithila Patwardhan
Office:	ITE 326	Office:	ITE 349
hrs:	Tu&Thu 11:30a-12:30p	hrs:	Mon&Wed 2:30-3:30p
Phone:	410-455-3093	Phone:	410-455-8933
Email:	chang@umbc.edu	Email:	mithila1@umbc.edu

Time and Place. Tuesday & Thursday 10:00am – 11:30am, ITE 227.

Textbooks.

- *Principles of Computer Architecture,* Murdocca and Heuring, Prentice-Hall, 2000.
- *Linux Assembly Language Programming,* Neveln, Prentice-Hall, 2000.

Course Web Page: <http://www.csee.umbc.edu/~chang/cs313.f04/>

Prerequisites. You should have mastered the material covered in the following courses: CMSC 202 Computer Science II and CMSC 203 Discrete Structures. In particular, we will assume that you have had extensive programming experience in C/C++. Also, you must 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 and computer organization. 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. 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). A secondary goal of this course is to prepare computer science majors for CMSC 411 Computer Architecture.

Grading. Your final grade will be based upon 5 homework assignments (20% total), 3 short programming assignments (12% total), 1 long programming assignment (8%), 3 circuit simulation exercises (12% total), a midterm exam (24%) and a final exam (24%). Your grade is given for work done *during* the semester; incomplete grades will only be given for medical illness or other such dire circumstances.

Lecture Policy. You are expected to attend all lectures. You are responsible for all material covered in the lecture as well as those in the assigned reading. However, this subject cannot be learned simply by listening to the lectures and reading the book. In order to master the material, you need to spend time outside the classroom on the programming assignments, simulation exercises and homework assignments.

Due Dates. There will be homework or exercises due on most weeks. Homeworks are due at the beginning of lecture. Exercises and projects turned in via online submission are due 1 minute past 11:59pm of the due date. ***With one exception, late homework, exercises and programming assignments will not be accepted — this is to allow for timely grading and discussion of the solutions. The exception is that each student may submit one assignment (of any kind) up to one week late during the semester.***

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. 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. ***Cases of academic dishonesty will be dealt with severely.***

Exams. The exams will be closed-book and closed-notes. The date for the midterm exam is Tuesday, October 26. The final exam will cover the material from the second part of the course. The date and time of the final exam is Tuesday, December 21, 10:30am to 12:30pm. *You will not be able to take the final exam at an earlier time*.

Advising Note. This course is a replacement for CMSC 211 Assembly Language Programming and CMSC 311 Computer Organization. Students who have taken either CMSC 211 or CMSC 311 previously should not take this class. Furthermore, computer science majors who take this class must also take CMSC 411 Computer Architecture to satisfy the requirements of a BS degree in computer science. CMSC 313 by itself will not be sufficient for graduation.

Policy on Programming Projects and Exercises

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. Late assignments will not be accepted (with the one exception noted in the course description). 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 Linux operating system running on an Intel Pentium CPU. This arrangement is not compatible with other flavors of UNIX, with Linux running on non-Intel CPUs or with assemblers for Windows 95/98/2k/ME/XP/NT. 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.

Cheating

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.

All programming assignments and circuit simulation exercises must be completed by your own individual effort. You should never have a copy of someone else's program either on paper or electronically under any circumstance. Also, you should never give a copy of your program or circuit, either on paper or electronically, to another student. This also means that you cannot work on the programming assignments or circuit simulation exercises together. Cases of academic dishonesty will be dealt with severely. Egregious cases of cheating will be reported as a major infraction. In this case, you will not be allowed to drop the course. Also, a major 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.

We will be using special software to check for cheating. The software is quite sophisticated and has surprised some students in the past. We will, of course, not release the details of the internal workings of this cheat-checking software, but you are forewarned that there is no difficulty in comparing every pair of submitted projects.

Finally, you are also warned that if your program is turned in by someone else, then both you and the person who copied your program will receive a 0 for that assignment. This includes substantially similar programs. Furthermore, all parties concerned will have their prior programs checked for cheating. So, if you cheat on the last assignment, you can lose all the points from all of your assignments — even if you did all the work and just let other people copy from you.

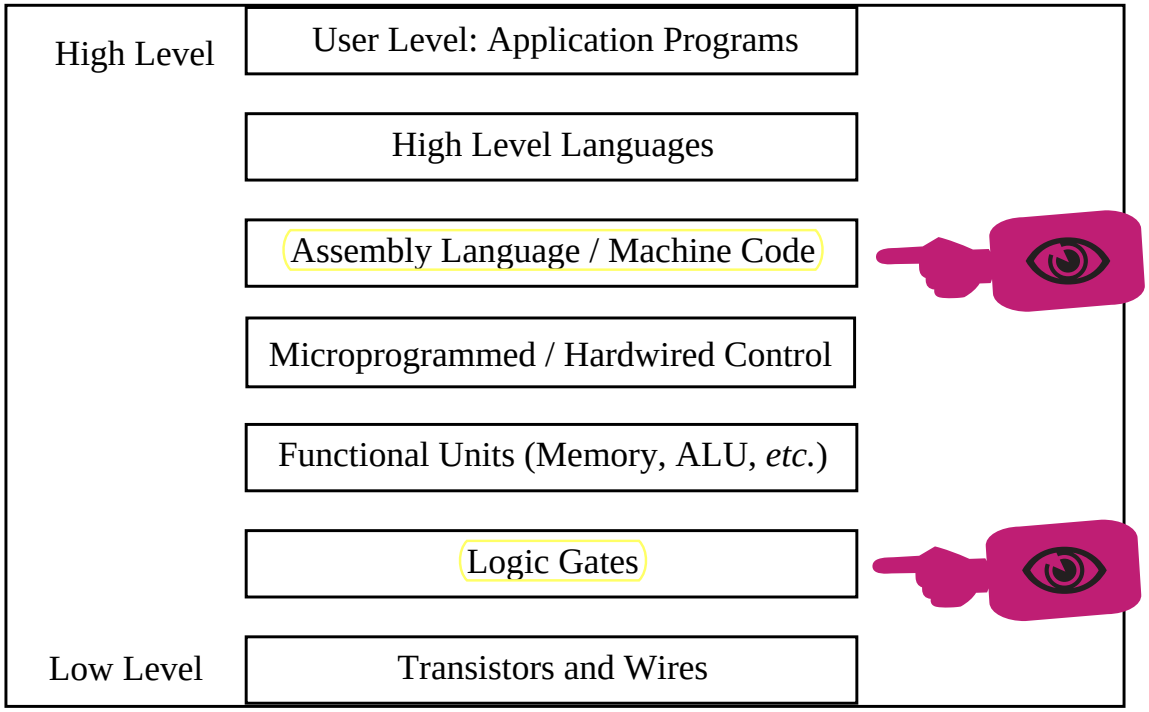
The UMBC academic integrity policy is available at http://www.umbc.edu/integrity/students.html.

Some Definitions

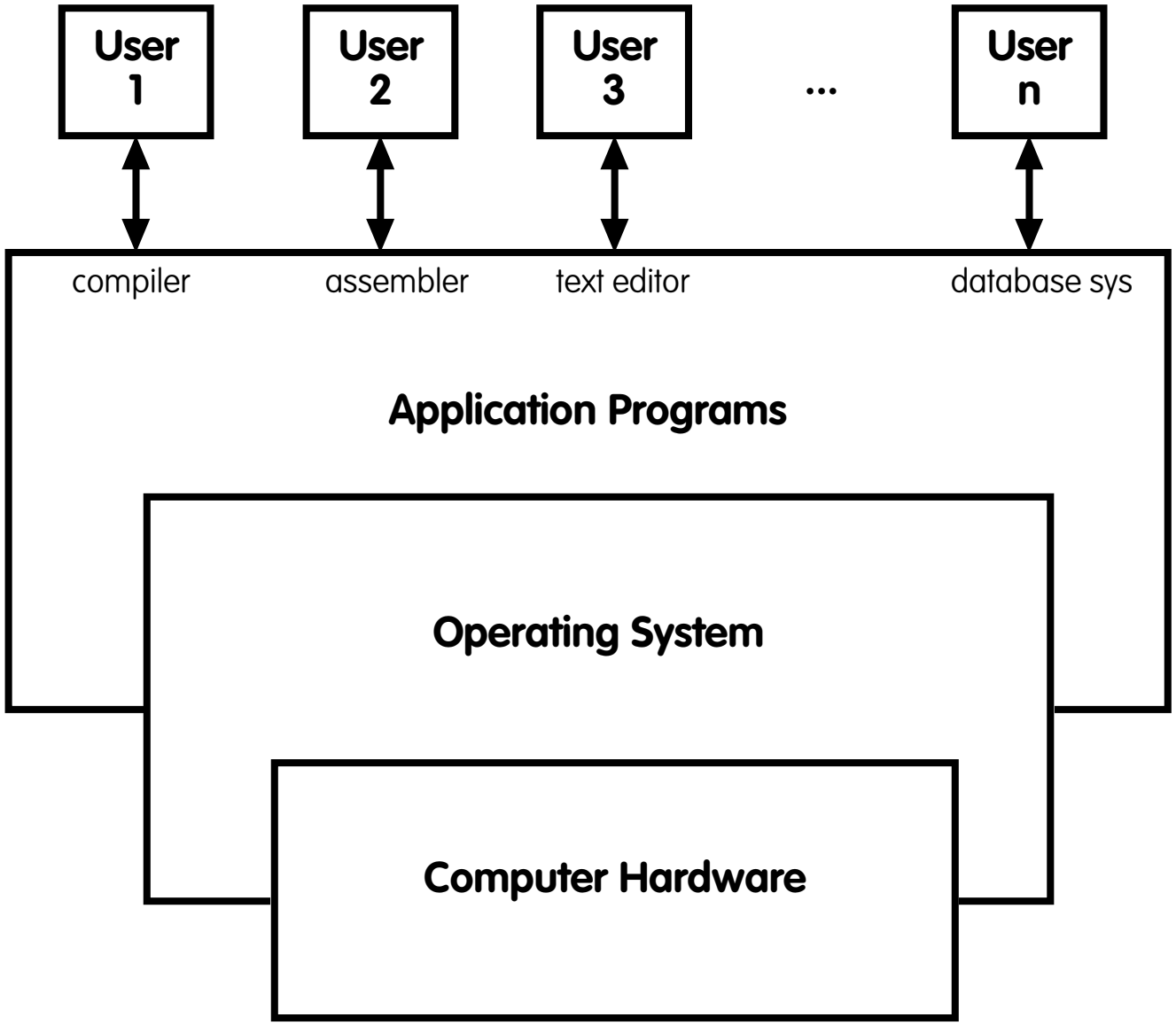
- ***Computer architecture*** **deals with the functional behavior of a computer system as viewed by a programmer (like the size of a data type – 32 bits to an integer).**
- ***Computer organization*** **deals with structural relationships that are not visible to the programmer (like clock frequency or the size of the physical memory).**
- **There is a concept of *levels* in computer architecture. The basic idea is that there are many levels at which a computer can be considered, from the highest level, where the user is running programs, to the lowest level, consisting of transistors and wires.**

Levels of Machines

- **There are a number of levels in a computer (the exact number is open to debate), from the user level down to the transistor level.**
- **Progressing from the top level downward, the levels become less abstract as more of the internal structure of the computer becomes visible.**

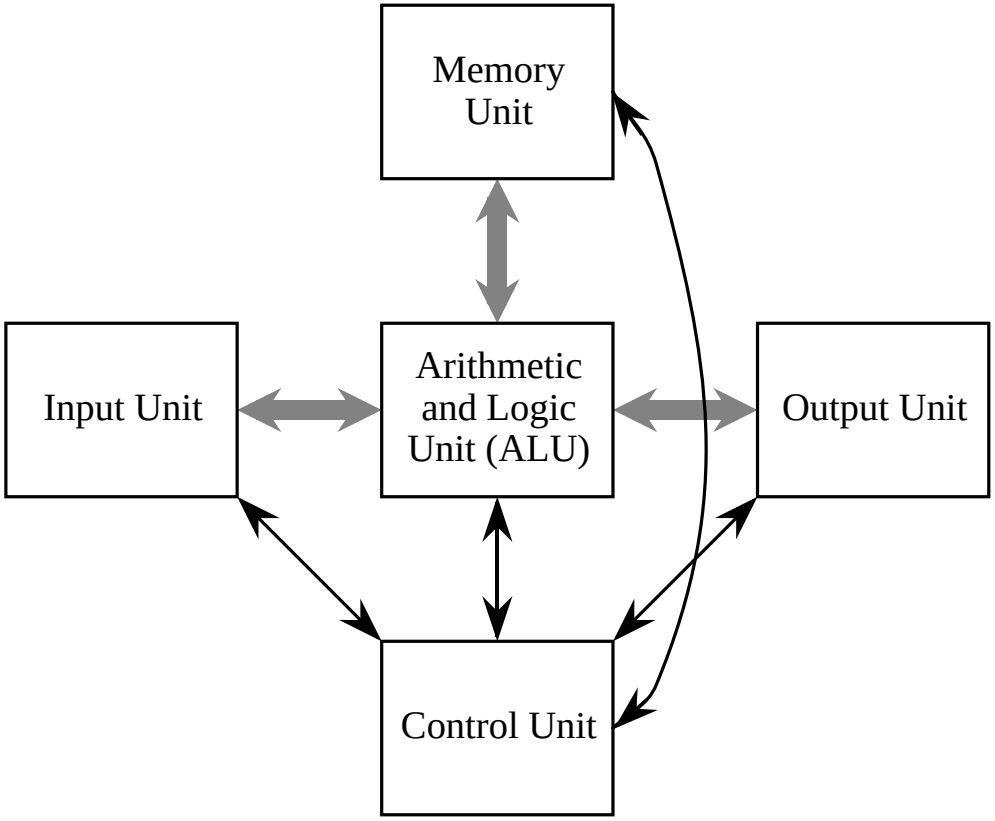


Computer Science View of the World



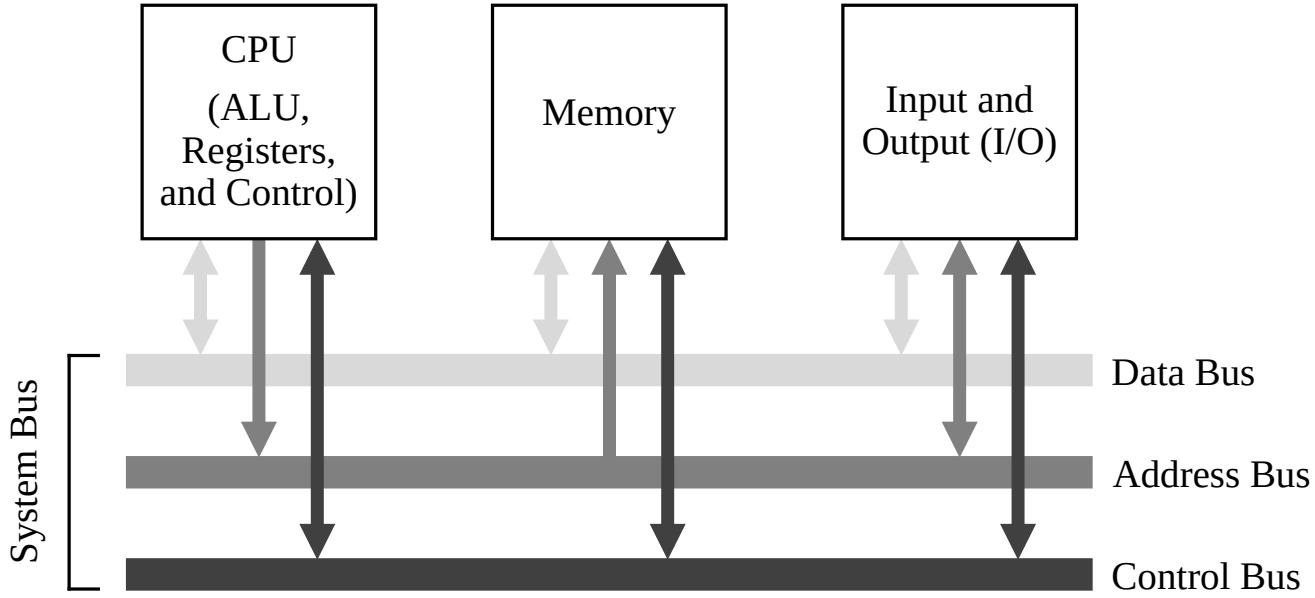
The von Neumann Model

- **The von Neumann model consists of five major components: (1) input unit; (2) output unit; (3) arithmetic logic unit; (4) memory unit; (5) control unit.**



The System Bus Model

- **A refinement of the von Neumann model, the system bus model has a CPU (ALU and control), memory, and an input/output unit.**
- **Communication among components is handled by a shared pathway called the *system bus*, which is made up of the data bus, the address bus, and the control bus. There is also a power bus, and some architectures may also have a separate I/O bus.**



The Fetch-Execute Cycle

- **The steps that the control unit carries out in executing a program are:**

  (1) Fetch the next instruction to be executed from memory.

  (2) Decode the opcode.

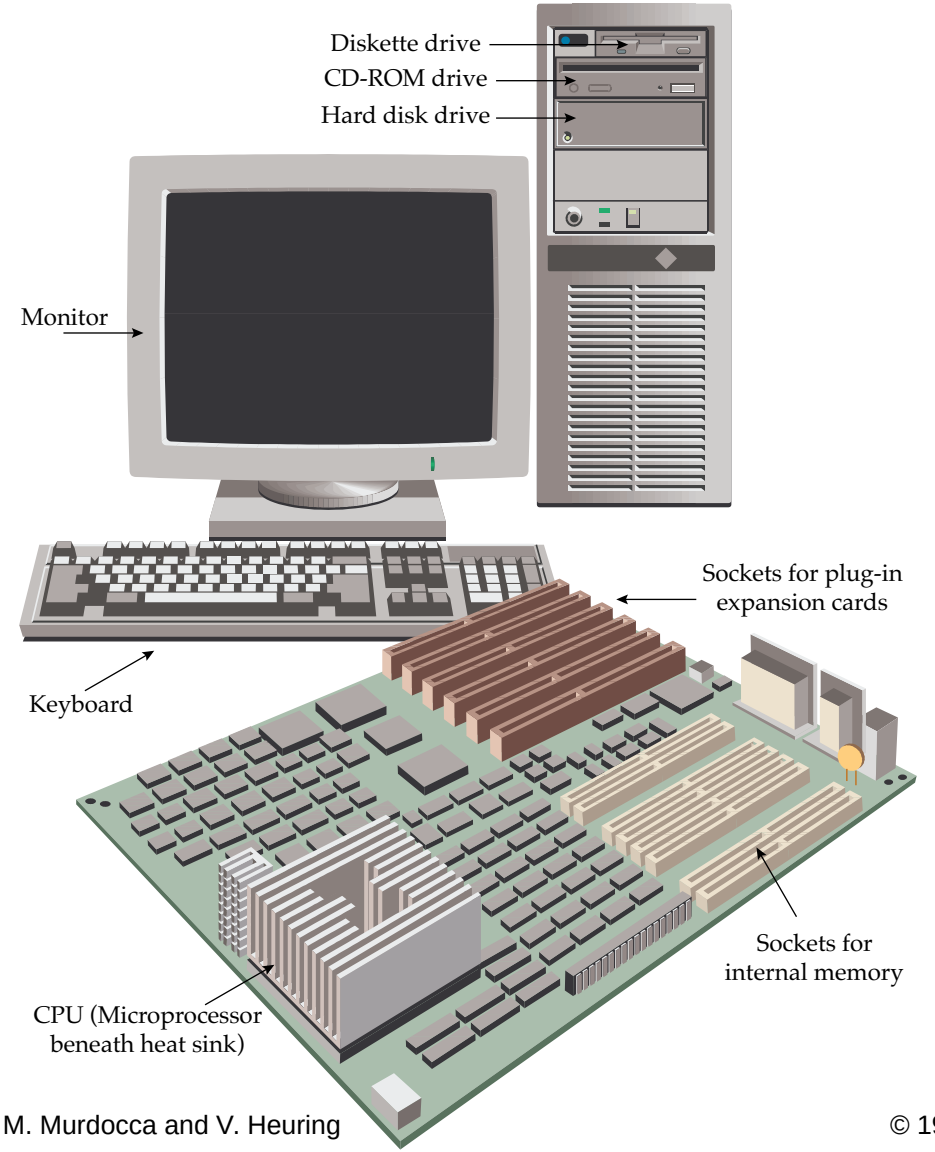
  (3) Read operand(s) from main memory, if any.

  (4) Execute the instruction and store results.

  (5) Go to step 1.

This is known as the *fetch-execute cycle*.

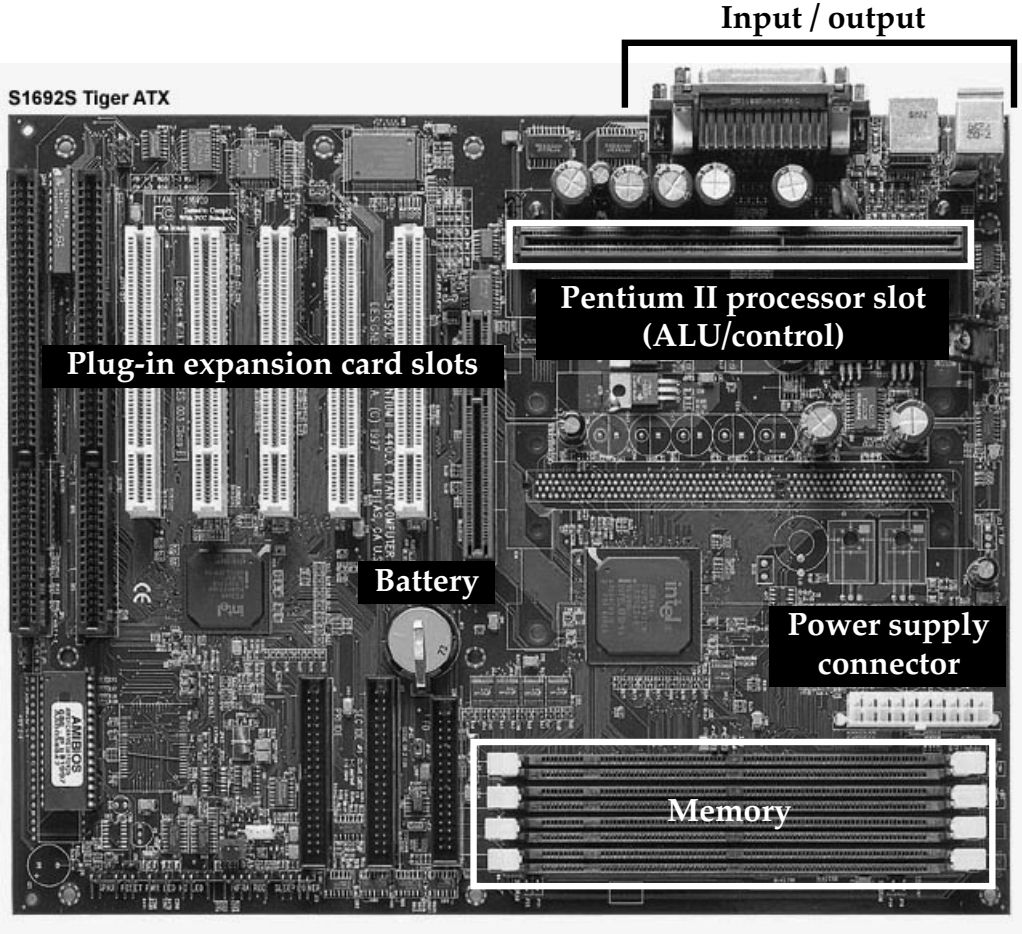
A Typical Computer System



The Motherboard

- **The five von Neumann components are visible in this example motherboard, in the context of the system bus model.**

(Source: TYAN Computer, http://www.tyan.com)



Converting Base 6 to Base 10

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

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

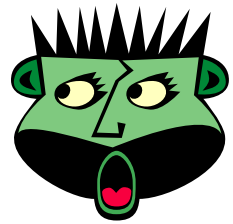
$0.45_6 = 4 \times 1/6_{10} + 5 \times 1/36_{10} = 0.805555..._{10}$

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

Converting Base 10 to Base 6

- $754.94_{10} = 3254.5\ 35012\ 35012\ 35012\ldots_6$

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



$754 \div 6 = 125$ 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)

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

$0.94 \times 6 = 5.64 \rightarrow 5$

$0.64 \times 6 = 3.84 \rightarrow 3$

$0.84 \times 6 = 5.04 \rightarrow 5$

$0.04 \times 6 = 0.24 \rightarrow 0$

$0.24 \times 6 = 1.44 \rightarrow 1$

$0.44 \times 6 = 2.64 \rightarrow 2$

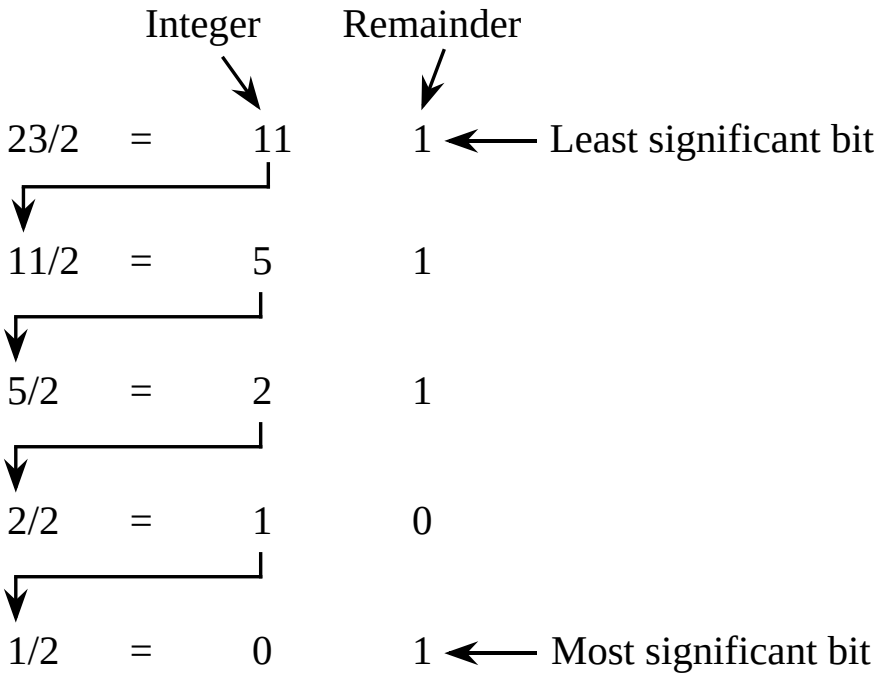
$0.64 \times 6 = 3.84 \rightarrow 3$

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

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

Base Conversion with the Remainder Method

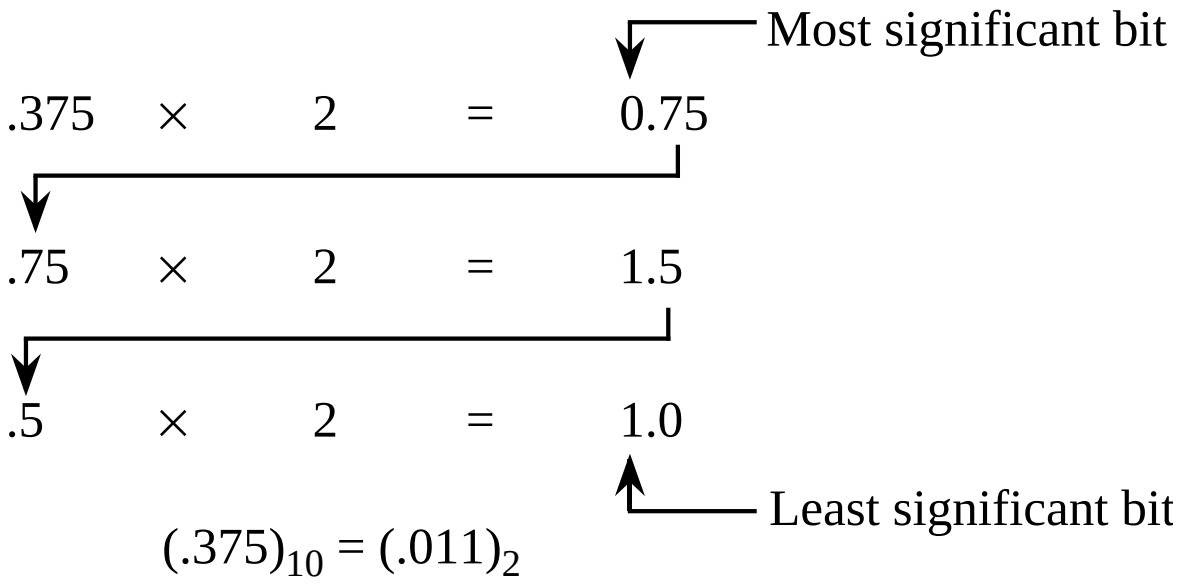
- **Example: Convert 23.375_{10} to base 2. Start by converting the integer portion:**



$$(23)_{10} = (10111)_2$$

Base Conversion with the Multiplication Method

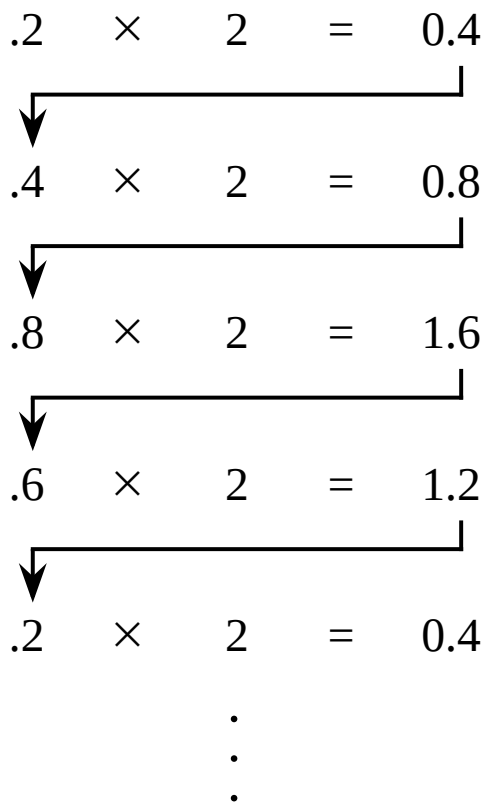
- **Now, convert the fraction:**



- **Putting it all together, $23.375_{10} = 10111.011_2$.**

Nonterminating Base 2 Fraction

- **We can't always convert a terminating base 10 fraction into an equivalent terminating base 2 fraction:**



Base 2, 8, 10, 16 Number Systems

Binary (base 2)	Octal (base 8)	Decimal (base 10)	Hexadecimal (base 16)
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

- **Example: Show a column for ternary (base 3). As an extension of that, convert 14_{10} to base 3, using 3 as the divisor for the remainder method (instead of 2). Result is 112_3**

More on Base Conversions

- **Converting among power-of-2 bases is particularly simple:**

  $1011_2 = (10_2)(11_2) = 23_4$

  $23_4 = (2_4)(3_4) = (10_2)(11_2) = 1011_2$

  $101010_2 = (101_2)(010_2) = 52_8$

  $01101101_2 = (0110_2)(1101_2) = 6D_{16}$

- **How many bits should be used for each base 4, 8, *etc.*, digit? For base 2, in which $2 = 2^1$, the exponent is 1 and so one bit is used for each base 2 digit. For base 4, in which $4 = 2^2$, the exponent is 2, so so two bits are used for each base 4 digit. Likewise, for base 8 and base 16, $8 = 2^3$ and $16 = 2^4$, and so 3 bits and 4 bits are used for base 8 and base 16 digits, respectively.**

Next Time

- Representing negative numbers
- Modulo Arithmetic & Two's Complement