CS1101
Introduction to Programming

Instructors: Krishna Sivalingam, V Krishna Nandivada, Rajsekar M,

Co-ordinator: Madhu Mutyam

Course Outline

- Introduction to Computing
- Programming (in C)
- Exercises and examples from the mathematical area of Numerical Methods
- Problem solving using computers

Evaluation

- Two Quizzes – 30
- Programming Assignments – 25
- End of Semester Exam – 45
- Attendance – taken in the lab and in lectures

Class Hours

- Class meets 3 times a week (E2 slot)
  - Monday 2.55 – 3.45 PM
  - Tuesday 1.00 – 1.50 PM
  - Wednesday 4.55 – 5.45 PM
- Venue
  - CRC 102 / CRC 103

Programming Assignments

- Class split into batches; One batch per weekday
- Time: 7:30 – 9:30 PM
- Venue: Departmental Computing Facility (DCF) in CSE Dept.

Policies

- Strictly no cell phone usage in class
  - Keep your cell phone turned off
    - Not even in silent mode
  - No SMS, chat etc.
- Cell phones will be confiscated if there are violations
  - Returned only after 2 weeks
- Repeat violations
  - Students will be sent to the Dean (Acad)

What is this CS110 about?

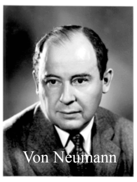
- Computer and its components
- Computing
- Programming Languages
- Problem Solving and Limitations of a Computer

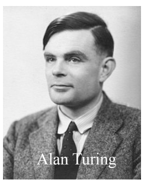
Common uses of a Computer

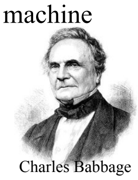
- As a tool for storing and retrieving information
  - Extracting and storing information regarding students entering IIT
- As a tool for providing services to customers
  - Billing, banking, reservation
- As a calculator capable of user-defined operations
  - Designing electrical circuit layouts
  - Designing structures
  - Non-destructive testing and simulation

What is a Computer?

- A computer is a *programmable machine*
- Its behavior is controlled by a *program*
- Programs reside in the *memory* of the machine
  - *"The stored program concept"*



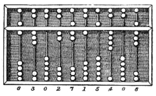




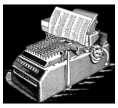
Charles Babbage

Early Computing Hardware

The Slide rule



The Chinese Abacus

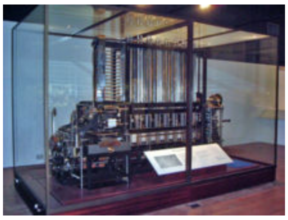


The gear replaced the beads in early mechanical calculators

"History of computing hardware"
From Wikipedia, the free encyclopedia

The Difference Engine : ~1850's

Part of Babbage's difference engine, assembled after his death by Babbage's son, using parts found in his laboratory.



The London Science Museum's replica Difference Engine, built from Babbage's design.

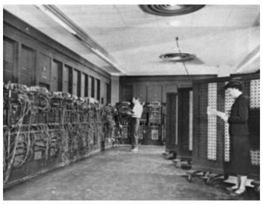
An automatic, mechanical calculator designed to tabulate polynomial functions

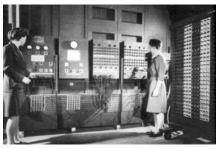
The First Programmer

Augusta Ada King, Countess of Lovelace (December 10, 1815 – November 27, 1852), born **Augusta Ada Byron**, is mainly known for having written a description of Charles Babbage's early mechanical general-purpose computer, the analytical engine.

The programming language ADA is named after her.

ENIAC – The First Electronic Computer

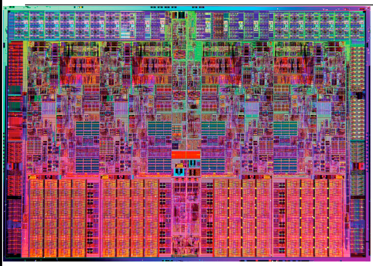


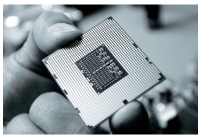


Physically, ENIAC was massive compared to modern PC standards. It contained 17,468 vacuum tubes, 7,200 crystal diodes, 1,500 relays, 70,000 resistors, 10,000 capacitors and around 5 million hand-soldered joints. It weighed 27 tons, was roughly 2.4 m by 0.9 m by 30 m, took up 167 m², and consumed 150 kW of power.

***E**lectronic **N**umerical **I**ntegrator and **C**omputer, 1946-55* (Univ. Penn.)

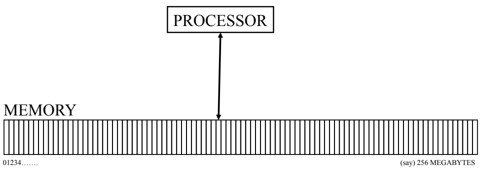
Core i7 Processor (desktop version)





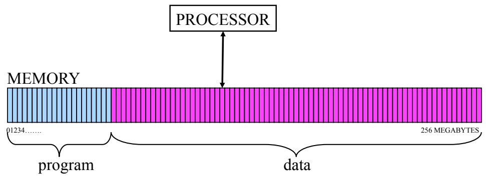
2008-15: Intel Core i7 Processor
Clock speed: >2.5 GHz
No. of Transistors: 0.731-1.3B
Technology: 45-22nm CMOS
Area: 263-181*mm*^{2}

The Computing Machine



The computer is made up of a *processor* and a *memory*. The memory can be thought of as a series of *locations* to store information.

The Computing Machine



- A *program* is a sequence of *instructions* assembled for some given task
- Most instructions operate on *data*
- Some instructions *control* the flow of the operations
- It is even possible to treat programs as data. By doing so a program could even modify itself.

Variables

- Data is represented as binary strings
  - It is a sequence of 0's and 1's (bits), of a predetermined size – "word". A *byte* is made of *8 bits*.
- Each memory location may be given a *name*.
- The name is the *variable* that refers to the data stored in that location
  - e.g. rollNo, classSize
- Variables have *types* that define the interpretation of data
  - e.g. integers (1, 14, 25649), or characters (a, f, G, H)

Instructions

- Instructions take data stored in variables as arguments
- Some instructions do some operation on the data and store it back in some variable
  - e.g. The instruction "X←X+1" on integer type says that "Take the integer stored in X, add 1 to it, and store it back in (location) X"
- Other instructions tell the processor to do something
  - e.g. "jump" to a particular instruction next, or to exit

Programs

- A program is a sequence of instructions
- Normally the processor works as follows,
  - Step A: pick next instruction in the sequence
  - Step B: get data for the instruction to operate upon
  - Step C: execute instruction on data (or "jump")
  - Step D: store results in designated location (variable)
  - Step E: go to Step A
- Such programs are known as *imperative programs*

Programming Paradigms

- *Imperative programs* are sequences of instructions. They are abstractions of how the *von Neumann machine* operates
  - Pascal, C, Fortran
- *Object Oriented Programming Systems (OOPS)* model the domain into objects and interactions between them
  - Simula, CLOS, C++, Java
- *Logic programs* use logical inference as the basis of computation
  - Prolog
- *Functional programs* take a mathematical approach of functions
  - LISP, ML, Haskell

A Limitation – Computer Arithmetic

- Number of digits that can be stored is limited
- Causes serious problems

Consider a computer that can store:
Sign, 3 digits and a decimal point
Sign and decimal point are optional

example : 212, -212, -21.2, -2.12, -.212

More Examples

- 113. + -111. = 2.00
- 2.00 + 7.51 = 9.51
- -111. + 7.51 = -103.49 (exact arithmetic)

But our computer can store only 3 digits.
So it rounds –103.49 to –103

This is a very important thing to know as a system designer. Why?

Why?

Consider 113. + -111. + 7.51

To us addition is associative
(a+b)+c = a+(b+c)

(113. + -111.) + 7.51 = 2.00 + 7.51 = 9.51
113. + (-111. + 7.51) = 113. – 103. = 10.0

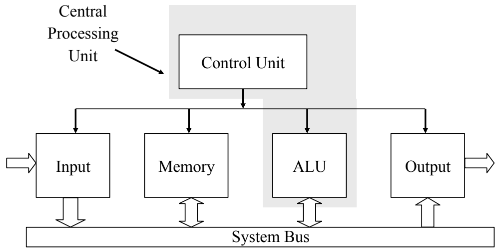
Conclusion

- Computer is fast but restricted
- So we must learn to use its speed
- And manage its restrictions

Books

- Paul Deitel and Harvey Deitel. C: How to Program.
- V. Rajaraman: Computer Programming in C
- R. G. Dromey: How to Solve It By Computer
- Kernighan and Ritchie: The C Programming Language
- Kernighan and Pike: The Unix Programming Environment

Building Blocks of a Computer



The Blocks, Their Functions

- **Input unit**
  - Takes inputs from the external world via variety of input devices – *keyboard, mouse, etc*.
- **Output Unit**
  - Sends information (after retrieving, processing) to output devices – *monitors/displays, projectors, audio devices, etc*.

More (try *more filename* on your Unix/Linux machine)

- **Memory**
  - Place where information is stored
  - ***Primary memory***
    - Electronic devices, used primarily for temporary storage
    - Characterized by their speedy response
  - ***Secondary Memory***
    - Devices for long-term storage
    - Contained well tuned mechanical components, magnetic storage media – floppies, hard disks
    - Compact Disks use optical technology

Some More (Commands are in */bin, /usr/bin*. Use *ls*)

- **System Bus**
  - Essentially a set of wires, used by the other units to communicate with each other
  - transfers data at a very high rate
- **ALU** – Arithmetic and Logic Unit
  - Processes data - add, subtract, multiply, …
  - Decides – after comparing with another value, for example

Finally (check *man cp, man mv, man ls, man –k* search string)

- **Control Unit**
  - Controls the interaction among other units
  - Knows each unit by its name, responds to requests fairly, reacts quickly on certain critical events
  - Gives up control periodically in the interest of the system

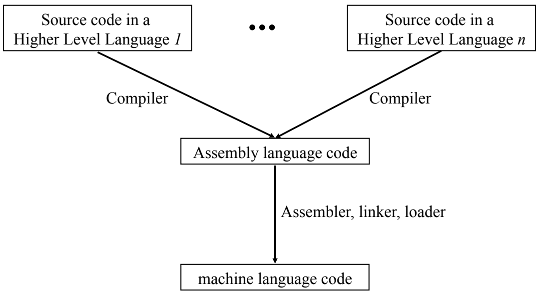
Control Unit + ALU is called the CPU

The CPU (editors *vi, emacs* used to create text)

- Can *fetch* an instruction from memory
- *Execute* the instruction
- *Store* the result in memory
- A *program* – a set of instructions
- An instruction has the following structure
  Operation operands destination
- A simple operation
  add a, b *Adds the contents of memory locations a and b and stores the result in location a*

Compilers

Human friendly languages → source code



Machine understandable language

Assembly language

- An x86/IA-32 processor can execute the following binary instruction as expressed in machine language:
  Binary: 10110000 01100001
  mov al, 061h
  – Move the hexadecimal value 61 (97 decimal) into the processor register named ”al".
  – Assembly language representation is easier to remember (*mnemonic*)

From Wikipedia

Higher Level Languages

- Higher level statement = many assembly instructions
- For example "X = Y + Z" could require the following sequence
  - Fetch the contents of Y into R1
  - Fetch the contents of Z into R2
  - Add contents of R1 and R2 and store it in R1
  - Move contents of R1 into location named X

Data Representation

- **Integers – Fixed Point Numbers**

Decimal System - Base 10 uses 0,1,2,…,9

$(396)_{10} = (6 \times 10^0) + (9 \times 10^1) + (3 \times 10^2) = (396)_{10}$

Binary System - Base 2 uses 0,1

$(11001)_2 = (1 \times 2^0)+(0 \times 2^1)+(0 \times 2^2)+(1 \times 2^3)+(1 \times 2^4)$
$= (25)_{10}$

Decimal to Binary Conversion

Convert $(39)_{10}$ to binary form

Base = 2

2	39	
2	19	+ Remainder 1
2	9	+ Remainder 1
2	4	+ Remainder 1
2	2	+ Remainder 0
2	1	+ Remainder 0
	0	+ Remainder 1

$39 = 2*19 + 1$
$= 2*(2*9 +1) + 1$
$= 2^2*9 + 2^1*1 + 1$
$= 2^2*(2*4+1) + 2^1*1 + 1$
$= 2^3*4+2^2*1+ 2^1*1 + 1$
$= 2^3*(2*2+0)+2^2*1+ 2^1*1 + 1$
$= 2^4*2+2^3*0+ 2^2*1+ 2^1*1 + 1$
$= 2^4*(2*1+0) + \ldots$
$= 2^5*1+2^4*0+2^3*0+2^2*1+ 2^1*1+ 1$

Put the remainders in reverse order

$(100111)_2 = (1 \times 2^0)+(1 \times 2^1)+(1 \times 2^2)+(0 \times 2^3)+(0 \times 2^4)+(1 \times 2^5)$

$= (39)_{10}$

Largest number that can be stored in m-digits

base - 10 : $(99999\ldots9) = 10^m - 1$

base - 2 : $(11111\ldots1) = 2^m - 1$

m = 3 $(999) = 10^3 - 1$

$(111) = 2^3 - 1$

Limitation: Memory cells consist of 8 bits (1 byte) multiples, each position containing 1 binary digit

Sign - Magnitude Notation

Common cell lengths for integers : k = 16 or 32 or 64 bits

First bit is used for a sign

0 – positive number

1 – negative number

The remaining bits are used to store the binary magnitude of the number.

Limit of 16 bit cell : $(32{,}767)_{10} = (2^{15} - 1)_{10}$

Limit of 32 bit cell : $(2{,}147{,}483{,}647)_{10} = (2^{31} - 1)_{10}$

One's Complement Notation

In the one's complement method, the negative of integer *n* is represented as the bit complement of binary *n*

E.g. : One's Complement of $(3)_{10}$ in a 3 - bit cell

complement of 011 : 100

-3 is represented as = $(100)_2$

Arithmetic requires care:

2 + (-3) = 010 + 100 = 110 – ok

But, 3 + (-2) = 011 + 101 = 000 and carry of 1

need to add back the carry to get 001!

NOT WIDELY USED

000 : 0
001 : +1
010 : +2
011 : +3
100 : -3
101 : -2
110 : -1
111 : -0

Zero has two representations!

Two's Complement Notation

In the two's complement method, the negative of integer *n* in a *k* - bit cell is represented as $2^k - n$

Two's Complement of $n = (2^k - n)$

E.g. : Two's Complement of $(3)_{10}$ in a 3 - bit cell

-3 is represented as $(2^3 - 3)_{10} = (5)_{10} = (101)_2$

Arithmetic requires no special care:

2 + (-3) = 010 + 101 = 111 – ok

3 + (-2) = 011 + 110 = 001 and carry of 1

we can ignore the carry!

WIDELY USED METHOD for –ve numbers

000 : 0
001 : +1
010 : +2
011 : +3
100 : -4 (8 – 4)
101 : -3 (8 – 3)
110 : -2 (8 – 2)
111 : -1 (8 – 1)

Two's Complement Notation

The Two's Complement notation admits one more negative number than the sign - magnitude notation.

To get back *n*, read off the sign from the MSB

If –ve, to get magnitude, complement the cell and add 1 to it!

E.g.: 101 → 010 → 011 = $(-3)_{10}$

000 : 0
001 : +1
010 : +2
011 : +3
100 : -4
101 : -3
110 : -2
111 : -1

Numbers with Fractions

Integer Part + Fractional Part

Decimal System - base 10
235 . 7846

Binary System - base 2

10011 . 11101 = $(19.90625)_{10}$

Fractional Part $(0.7846)_{10} = \frac{7}{10} + \frac{8}{10^2} + \frac{4}{10^3} + \frac{6}{10^4}$

Fractional Part $(0.11101)_2 = \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{0}{2^4} + \frac{1}{2^5} = 0.90625$

Binary Fraction → Decimal Fraction

$(10.11)_2$

Integer Part $(10)_2 = 1*2^1 + 0*2^0 = 2$

Fractional Part $(11)_2 =$

Decimal Fraction = $(2.75)_{10}$

Decimal Fraction → Binary Fraction (1)

Convert $(0.90625)_{10}$ to binary fraction

0.90625 × 2		
0.8125	+ integer part	1
0.8125 × 2		
0.625	+ integer part	1
0.625 × 2		
0.25	+ integer part	1
0.25 × 2		
0.5	+ integer part	0
0.5 × 2		
0	+ integer part	1

$0.90625 = ½(1+0.8125)$
$= ½(1+ ½(1+0.625))$
$= ½(1+ ½(1+ ½(1+0.25)))$
$= ½(1+½(1+ ½(1+½(0+0.5))))$
$= ½(1+½(1+½(1+½(0+½(1+0.0)))))$
$= ½+1/2^2+1/2^3+0/2^4+1/2^5$
$= (0.11101)_2$

Thus, $(0.90625)_{10} = (0.11101)_2$

Decimal Fraction → Binary Fraction (2)

Convert $(0.9)_{10}$ to binary fraction

0.9 × 2		
0.8	+ integer part	1
0.8 × 2		
0.6	+ integer part	1
0.6 × 2		
0.2	+ integer part	1
0.2 × 2		
0.4	+ integer part	0
0.4 × 2		
0.8	+ integer part	0

For some fractions, we do not get 0.0 at any stage! These fractions require an infinite number of bits! Cannot be represented exactly!

Repetition

$(0.9)_{10} = 0.11100110011001100... = 0.1\overline{1100}$

Fixed Versus Floating Point Numbers

Fixed Point: position of the radix point fixed and is same for all numbers

E.g.: With 3 digits after decimal point:
0.120 * 0.120 = 0.014
A digit is lost!!

Floating point numbers: radix point can float
$1.20 \times 10^{-1} * 1.20 \times 10^{-1} = 1.44 \times 10^{-2}$

Floating point system allows a much wider range of values to be represented

46

Scientific Notation (Decimal)

$0.0000747 = 7.47 \times 10^{-5}$

$31.4159265 = 3.14159265 \times 10^{1}$

$9{,}700{,}000{,}000 = 9.7 \times 10^{9}$

Binary

$(10.01)_2 = (1.001)_2 \times 2^{1}$

$(0.110)_2 = (1.10)_2 \times 2^{-1}$

47

Using Floating Point Notation

For any number *x*

$x \approx \pm q \times 2^{n}$

q – mantissa
n – exponent

$(-39.9)_{10} = (-100111.1\ 1100)_2$

$= (-1.001111\ 1100)_2 \times 2^{5}$

Decimal Value of stored number $(-39.9)_{10}$

$= (-1.\underbrace{001111\ 1100\ 1100\ 1100\ 11001}_{23\ \text{bit}}) \times 2^{5}$

32 bits :
First bit for sign
Next 8 bits for exponent
23 bits for mantissa
= -39. 90000152587890625

48