CS1100: Introduction to Programming

Apr-Jun 2021 Trimester

V. Krishna Nandivada

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 – 20 marks each
- End semester – 20 marks.
- Lab: 40 marks.
- Attendance – taken in the lab and in lectures

Class Hours

- Class meets 5 times a week.
  - Monday: 2pm to 3.15 pm (1.5LH)
  - Tue: 3.25pm to 4.40pm (1.5LH)
  - Thu: 10.00am to 10.50am (1LH)
  - Fri: 9.00am to 9.50am (1LH)
- Lab
  - Thu and Fri: 2 – 4.40pm.

Policies

- Online classes
  – You are the only one monitoring you.
  – You are responsible for your learning.
- Be attentive in the class.
  - Make it interactive to gain “points”.
- Be honest in the exams and lab.
  - Violators will be sent to DISCO.

What is this course about?

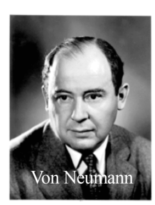
- Computer and its components
- Computing
- Programming Languages
- Problem Solving and Limitations of a Computer

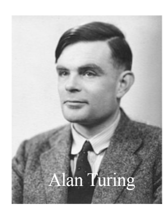
Common uses of a Computer

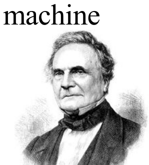
- As a tool for storing and retrieving information
  – Extracting and storing information regarding students entering IIT
- As a tool for providing services to customers
  – Billing, banking, reservation
- As a calculator capable of user-defined operations
  – Designing electrical circuit layouts
  –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

What is a Computer?

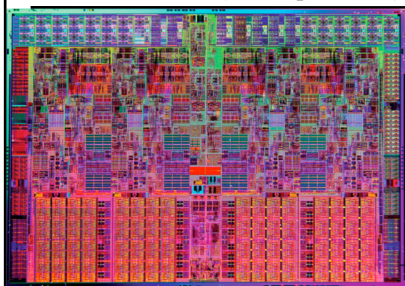
- A computer is a *programmable machine*
- Its behavior is controlled by a *program*
- Programs reside in the *memory* of the machine
  – *“The stored program concept”*

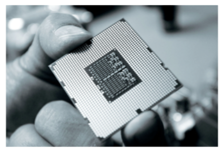





Charles Babbage

Core i7 Processor (desktop version)



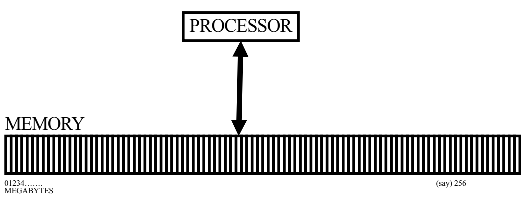


2008-19: Intel Core i7 Processor
Clock speed: 2.9 – 4.2 GHz
No. of Transistors: 1-8Billion
Technology: 45-10nm CMOS
Area: 100-485 mm^2

Memory storage on a Computer: Hierarchy

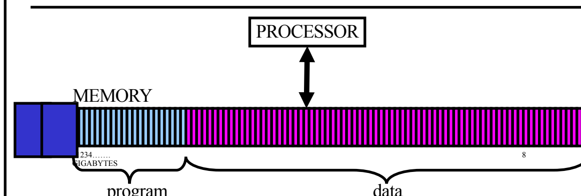
1. Registers (Small no. of registers in CPU) – fastest memory, since it is close to CPU
2. Cache (faster memory, small capacity, e.g. 12MB)
3. Main Memory (RAM) – slower than cache, a few nanonseconds to read a byte; limited capacity (e.g. 16GB … 1TB)
4. Secondary Memory – slower than RAM; but very large capacity (e.g. 512 GB disk, 4 TB disk, etc.)

The Computing Machine



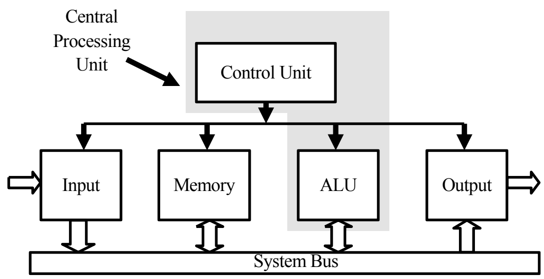
The computer consists of a *processor* and *memory*. The memory can be thought of as a series of *locations* to store information.

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

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
- *Decode and 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 register locations a and b and stores the result in register a*

Variables (stored in Memory)

- 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*

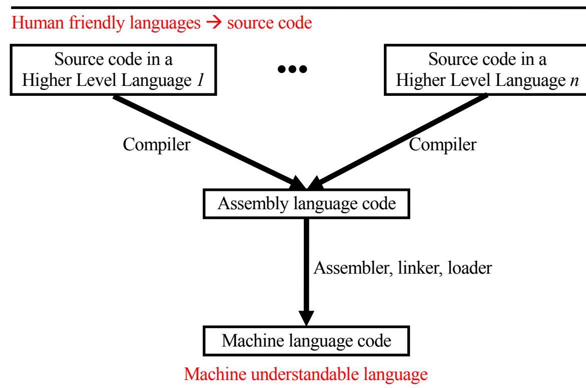
Programming Paradigms

- *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

- Father(X, Y)
- Father(X, Z)
- Father(N, M)
- Rules:
  - Sibling(A,B):- Father(J,A) and Father(J,B)

- Sibling(Y, M)? False
- Sibling(Z, Y)? True

Compilers



Assembly language

- An x86/IA-32 processor can execute the following binary instruction as expressed in machine language:

  Binary: $\underbrace{1011}_{\text{mov}}\underbrace{0000}_{\text{al,}}\ \underbrace{01100001}_{\text{061h}}$

  – Move the hexadecimal value 61 (97 decimal) into the processor register named ”al".

  – Assembly language representation is easier to remember (*mnemonic*)

From Wikipedia

Example Assembly Code (Z80 microprocessor)

LD A,5	A=5
ADD A,3	A=A+3 (A = 8)
LD B, 4	B=4
ADD A, B	A=A+B (A=12)
LD A, D	D=A

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

Number Systems

- Decimal: 0 .. 9
- Binary: 0 1
- Octal: 0 .. 7
- Hexadecimal: 0 .. 9 A B C D E F
- FEED -

Two-bit binary numbers

- 00
- 01
- 10 = 2 (base 10)
- 11 = 3 (base 10)
- N bits: 2^n numbers

3-bit binary numbers

- 000
- 001
- 010
- 011
- 100
- 101
- 110
- 111

4-bit binary numbers (Base 16: Hexadecimal)

- 0000: 0
- 0001
- 0010
- 0011
- 0100
- 0101
- 0110
- 0111: 7
- 1000: 8
- 1001: 9
- 1010: A
- 1011: B
- 1100: C
- 1101: D
- 1110: E
- 1111: F

- 789 base 10 = 7*10^2 + 8 * 10^1 + 9*10^0
- 11011 base 2 = 1*2^4 + 1 *2^3 + 0*2^2 + 1*2^1 + 1*2^0 =
  - 16 + 8 + 0 + 2 + 1 = 27
- 11011 base 10 = 11011
- 11011 base 8 = 4617
- There are 10 kind of people in the world: those who understand binary and those who dont

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: $(100111)_2$

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

$= (39)_{10}$

Steps to convert decimal to binary

98 base 10 = 1100010 base 2

```
Given X
i = 0
Loop until ( X != 0 )
    D[i] = X mod 2
   X = X / 2 ;; Quotient
   i = i + 1
 end
```


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 : $(2{,}147,483{,}647)_{10} = (2^{31} - 1)_{10}$

Signed numbers

- M = 3
- MSB is 0: positive number
- MSB is 1: negative number
- 000 : +0
- 001: 1
- 010: 2
- 011: 3
- 100 : -0
- 101: -1
- 110 :-2
- 111: -3

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!

8 bit number

What is -23 in one's complement form?

23: 00010111

-23: 11101000

Add these two:

0: 11111111

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

Two's complement

m = 3

011

One's complement: 100

Add 1: 100+ 1 = 101 (-3)

-1: 001 -> 110 + 1 = 111

-2: 010 -> 101 + 1 =

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

Binary addition

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10

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 το Decimal Fraction

$(10.11)_2$

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

Fractional Part $(11)_2 = 1*2\text{^}(-1) + 1 * 2\text{^}(-2) = ½ + ¼ = 0.75$

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

Decimal Fraction το Binary Fraction (1)

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

0.90625
* 2
0.8125 + integer part 1
* 2
0.625 + integer part 1
* 2
0.25 + integer part 1
* 2
0.5 + integer part 0
* 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 το Binary Fraction (2)

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

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

$(0.9)_{10} = 0.111001100110011100 . . .$

Fixed Versus Floating Point Numbers

Fixed Point: position of the radix point is 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
$1.20 \times 10^{-1} * 1.20 \times 10^{-1} = 1.44 * 10^{-2}$

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

Scientific Notation (Decimal)

$0.0000747 = 7.47 * 10^{-5}$

$31.4159265 = 3.14159265 * 10^{1}$

$9{,}700{,}000{,}000 = 9.7 * 10^{9}$

Binary

$(10.01)_2 = (1.001)_2 * 2^1$

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

Using Floating Point Notation

For any number *x*
$x = +/- (q * 2^{n})$

q – mantissa
n – exponent

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

$= (-1.001111\ 1100)_2 * 2^5$

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

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

32 bits :

First bit for sign

Next 8 bits for exponent

23 bits for mantissa

= -39. 90000152587890625

Stored value, as per IEEE 754 format: 1 10000100 00111111001100110011010
(Last bit rounded up by adding 1) 1100 0010 0001 1111 1001 1001 1001 1010

Binary Arithmetic

Half Adder

Bit 0	Bit 1	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full Adder

Carry In	Bit 0	Bit 1	Carry Out	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Number Representations for a 4-bit number

Binary Number	Sign-Mag.	One's Compl.	Two's Compl.
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7

Binary Number	Sign-Mag.	One's Compl.	Two's Compl.
1000	0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	0	-1

How to Convert a k-bit Sign-Magnitude number to decimal

- If MSB = 0, let x denote decimal value of remaining (k-1) bits
  - Decimal Number = **+x**
- If MSB = 1, let x denote decimal value of remaining (k-1) bits
  - Decimal Number = **-x**
- **0110 (Base 2) = +6 (Base 10)**
- **1010 (Base 2) = -2 (Base 10)**

How to Convert a k-bit One's Complement number to decimal

- Let y denote unsigned decimal value of all k bits
- MSB = 0
  - Decimal Number = **+y**
- MSB = 1
  - Decimal Number = **-(2^k-y-1)**
  - ALT: Flip all bits; Let z be this bit-string's value; Decimal Number = **-z**
- **0101 (Base 2) = +5 (Base 10)**
- **1101 (Base 2) = -(16-13-1) = -2 (Base 10)**

How to Convert a k-bit Two's Complement number to decimal

- Let y denote unsigned decimal value of all k bits
- MSB = 0
  - Decimal Number = **+y**
- MSB = 1
  - Decimal Number = **-(2^k-y)**
- **0101 (Base 2) = +5 (Base 10)**
- **1101 (Base 2) = -(16-13) = -3 (Base 10)**

Limits of Numbers

Number of Bits	Unsigned	Sign-Magnitude	One's Complement	Two's Complement
4	0 to 15	-7 to + 7	-7 to +7	-8 to +7
8	0 to 255	-127 to +127	-127 to +127	-128 to +127
16	0 to 65535	-32767 to +32767	-32767 to +32767	-32768 to +32767
32	0 to 2^{32} - 1	$-(2^{31} – 1)$ to $(2^{31} – 1)$	$-(2^{31} – 1)$ to $(2^{31} – 1)$	-2^{31} to $(2^{31} – 1)$
N				

- If you add two positive numbers and result is negative, overflow has occurred
- If you add two neg. numbers and result is positive, underflow has occurred

Data Representation

- **Integers – Fixed Point Numbers**

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

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

Binary System - Base 2 uses 0,1

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

Courses related to topics mentioned in Week 1

- CS2300 – Foundations of Computer Systems
- CS2600 – Computer Organization
- CS3100 – Paradigms of Programming
- CS3300 – Compiler Design
- CS3500 – Operating Systems