An Introduction to Programming through C++

Abhiram G. Ranade

Lecture Sequence 1.2

Reading: Ch. 2: A bird's eye view

(Some slides of Prof. Ranade have been modified and some portions of video accordingly re-recorded to compress material – Kameswari Chebrolu)

A basic question

- How is a computer able to do so many things?
  - Search for information
  - Predict weather
  - Process pictures and say what is in them
  - Play chess..
- Goal of this lecture sequence: provide high level answers
  - Most real life problems can be viewed as mathematical problems on numbers
  - A computer is good at solving math problems
- High level answers will give good background for later discussion.

Outline

- How to express real life problems as numerical problems.
  - Picture processing
  - Processing text/language
- Algorithms and Programs
  - Enable us to tell a computer what operations to perform
- How a computer does the required operations
  - Digital circuits
  - How numbers are represented
  - Parts of a computer
  - Machine language program, compilation.

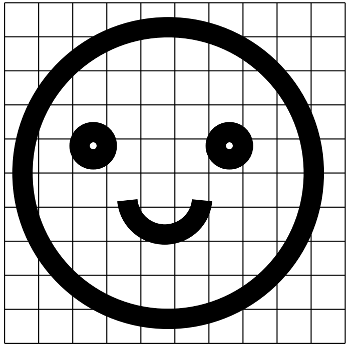
“What is in this picture?”

https://en.wikipedia.org/wiki/File:Jackson%27s_Chameleon_2_edit1.jpg

How to represent black and white pictures using numbers

- Suppose picture is 10cm x 10cm.
- Break it up into 0.1 mm x 0.1 mm squares
- 1000 x 1000 squares. 1 million "pixel"s
- If square is mostly white, represent it as 0.
- If square is mostly black, represent it as 1.
- Picture = 1 million numbers!

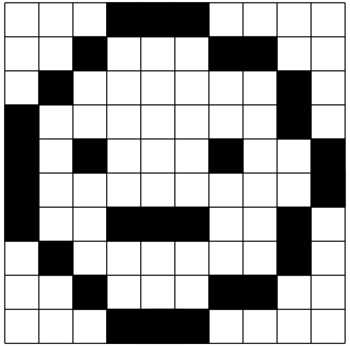
Picture, Representation, Reconstruction



(a)

$$\begin{matrix}
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0
\end{matrix}$$

(b)



(c)

Remarks

- Better representation if picture divided into more cells.
- Pictures with different “gray levels”: use numbers 0, 0.1, …, 1.0 to represent level of darkness rather than just 0, 1.
- Pictures with colours: picture = 3 sequences
  - sequence for red component,
  - sequence for blue component,
  - sequence for green component
- Add up the colours to get the actual colour.

Computer vision/Image processing

Input: sequence P of 1 million numbers (0 or 1)

- Representing a 10cm x 10cm black and white picture,
- Given in left to right, top to bottom order.

A very simple image processing problem:

- Is there a vertical line in the picture?

Expressing the problem mathematically: What property does the sequence need to have if it is to contain a vertical line?

- All 0s, except for 1s in consecutive rows of some column
- Going down a column = move 1000 positions in the sequence
- 1s in positions i, i+1000, i+2000, i+3000, i+4000,… for some i
- ”Is there a vertical line?” = “Does sequence P satisfy above property?”

One way of solving the problem:

- Try all values of i..

Does the picture contain a chameleon?

- In principle, same as asking whether the picture contains a single vertical line:
  - Identify a set of properties that the sequence of numbers representing the picture must satisfy if the picture contains a chameleon.
  - Identify computations that can check if the given number sequence satisfies the required properties.
- In practice requires enormous ingenuity
- Main concern of the deep subject “Computer Vision”

Language/text using numbers

- Define a code for representing letters.
- Commonly used code: ASCII
  - (American Standard Code for Information Interchange)
- Letter ‘a’ = 97 in ASCII, ‘b’ = 98, …
- Uppercase letters, symbols, digits also have codes. Code also for space character.
- Words = sequences of ASCII codes of letters in the word.
- ‘computer’ = 99, 111,109,112,117,116,101,114.
- Sentences/paragraphs = larger sequences.
- Does the word “computer” occur in a paragraph?
  - Does a certain sequence of numbers occur inside another sequence of numbers?

Exercises

- What pattern of 1s and 0s would correspond to a ”+” of any size being present at the center of an otherwise white picture?
- Suppose you are given a sentence in a language you cannot understand. Would you still be able to count the number of words in the sentence? Can you express this as a question on sequences of numbers representing the ASCII codes of different characters?
- How will you represent Chess playing as a question on numbers? Start by representing a chess board with pieces on it using numbers.

Summary of what we discussed

- Questions about pictures, weather, documents can be converted to questions about properties of number sequences.
- Finding answers requires solving interesting math problems.

Outline

- ~~How to express real life problems as numerical problems.~~
  - ~~Picture processing~~
  - ~~Processing text/language~~
- How a computer solves numeric problems?
  - Recap: Problems, Algorithms and Programs
  - Digital circuits
  - How numbers are represented
  - Parts of a computer
  - Machine language program, compilation.

Problems and Algorithms

- “Solving problems” = express real life problems as numerical problems; deciding what operations to perform to calculate the required answer
- Algorithms = A precise description of the operations needed to solve a problem.

You already know many algorithms!

- Primary school algorithms contain all ingredients that are present in advanced algorithms
  - **Arithmetic operations**: “Add the least significant digit of the first number to the least significant digit of the second number”
  - **Conditional operation**: “Set carry = 1 if the previous sum was greater than 9”
  - **Repetition**: “Repeat as many times as there are digits”
- Other algorithms you know
  - determine whether a number is prime,
  - finding the greatest common divisor
  - Drawing a polygon on the screen

Programs

- Algorithms written in precise syntax/language.
- C++ is one such language.
- Other languages: Fortran, Basic, Lisp, …
- All these languages can be used to specify arithmetic operations, conditional execution, and repetition.
  - You have already seen how to specify repetition – use the `repeat` statement!

Digital circuits: building blocks of computers

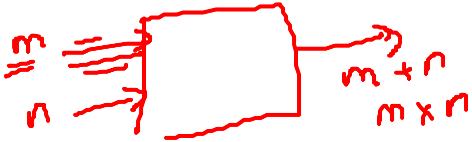
- Digital circuits: interpret electrical potential/voltage as numbers.
- Simplest convention
  - Voltage above 1 volt = number 1,   Voltage between 0 and 0.2 volt = number 0 → binary
  - Circuit designed so that voltage will never be between 0.2 and 1 volt, hence no ambiguity.
- Once you can represent 0 and 1 (binary), you can represent anything!

- Memory: Charge stored on a capacitor may also denote numbers
  - Capacitor has low charge = number 0, High charge = number 1
  - Once charge is stored on a capacitor, it persists.

- Processing: We can design circuits which perform arithmetic:
  - Circuit inputs: sets of voltages representing two numbers
  - Circuit outputs: set of voltages representing their sum or product, or quotient …

4

100 ↙ 3



Background: Binary representation

- Binary number $a_{n-1}a_{n-2}\ldots a_1a_0 . a_{-1}a_{-2}\ldots a_{-k}$
  - Example: 101.11
- Decimal value $v = \sum_i a_i 2^i$
  - $1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-2} = 5.75$
- Converting a decimal integer v to binary
  - Divide v by 2, remainder gives a_0
  - Repeat previous step with the quotient to get a_1, a_2, …
- Converting fraction f to binary
  - If $f > 0.5$, $a_{-1} = 1$
  - Similarly other bits…

Example

- Binary fraction equivalent of 0.8125_{10} base
- 0.8125 (multiply by 2) = **1**.625 = 0.625 carry **1** (MSB)
- 0.625 (multiply by 2) = **1**.25 = 0.25 carry **1** (↓)
- 0.25 (multiply by 2) = **0**.50 = 0.5 carry **0** (↓)
- 0.5 (multiply by 2) = **1**.00 = 0.0 carry **1** (LSB)
- Thus the binary equivalent of 0.8125_{10} is therefore: 0.1101_2 ← (LSB)

(Try 0.1_{10}? Infinite representation ....)

Integers: positive and negative

- Remember "int nsides?"
  - int give 32 capacitors (on most machines) for the variable nsides
- One of the bits can be used to indicate sign
- Sign bit = 0 (low charge/voltage) means positive number, = 1 means negative number.
- To store -25 use

31

10000000000000000000000000011001

- Leftmost bit = sign bit
- Max positive number: 2^{31} - 1

01111111111111111111111111111111

- Range stored: $-(2^{31} – 1)$ to $2^{31} – 1$.
- Actual representation used:
  - more complex. "**Two's complement**" simplest

Note the video has the brackets missing!

```
unsigned int nsides;
```

- 32 capacitors will be given, but the bit pattern in it will be interpreted as 32 bit binary number.
- 10000000000000000000000000011001 will mean $2^{31}+25$

Type	Size	Range
int	32 bits	-2,147,483,648 to +2,147,483,647
unsigned int	32 bits	0 to +4,294,967,295 $2^{32}-1$

Bits, bytes, half-words, words

- Bit = 1 binary “digit”, (one number = 0 or 1)
- byte = 8 bits
- half-word = 16 bits
- word = 32 bits
- double word = 64 bits

“one byte of memory” = memory capable of storing 8 bits = 8 capacitors.

Real numbers

- Remember "scientific notation" in decimal: significand * 10^{exponent}
  - $53 = 5.3 \times 10^{1}$
  - $0.00479 = 4.79 \times 10^{-3}$
- Same idea, but significand, exponent are in binary: significand * 2^{exponent}

  2^4 1.

  In video, extra 0 is wrong here!
  - $101.01 = 1.0101 \times 2^{2}{}_{(10)} = 1.0101 \times 2^{10}{}_{(2)}$

    ← sign
  - $0.0001101 = 1.101 \times 2^{-4}{}_{(10)} = 1.\cancel{0}101 \times 2^{1.....100}{}_{(2)}$ (2's complement)
- Store mantissa/significand and exponent separately
  - Note: need to store only bits AFTER decimal, in binary
- Actual representation: more complex. "IEEE Floating Point Standard".

- Single precision float: 32 bits
  - 1 bit for sign of significand/mantissa
  - 23 bits for mantissa itself
  - 8 bits for exponent (signed)
  - Largest number?

"signed"

- Double precision float: 64 bits
  - 1 bit for sign of significand/mantissa
  - 52 bits for mantissa itself
  - 11 bits for exponent (signed)
- Many special patterns: infinity, NaN, etc.

What we discussed

- Numbers are represented by sequence of 0s and 1s
- Can represent integers and real-numbers
  - Signed numbers are represented by two's complement
  - Real numbers are represented by IEEE 754 standard (float, double)
- As a user, you don't need to type/read binary numbers.
  - C++ will convert binary numbers to decimal system while printing
  - C++ will accept numbers typed in decimal by you and itself convert it to binary for use on the computer.
  - But you should know (roughly) what range of numbers can be stored in unsigned/signed/floating formats

float $m = 4.5678\,90\,12 \times 10^{22}$

800

9 digit

Variable Type and Ranges

Type	Size	Range	Significant Digits
int	32 bits	-2,147,483,648 to +2,147,483,647	
unsigned int	32 bits	0 to +4,294,967,295	
float	32 bits	±3.4E-38 and ±3.4E38	7
double	64 bits	±1.7E-308 and ±1.7E308	16

$\pm 3.4 \times 10^{-38} = \pm 0 \quad \pm 3.4 \; 10^{38}$

23 bit

$2^{23} \rightarrow 7$ digit decimal

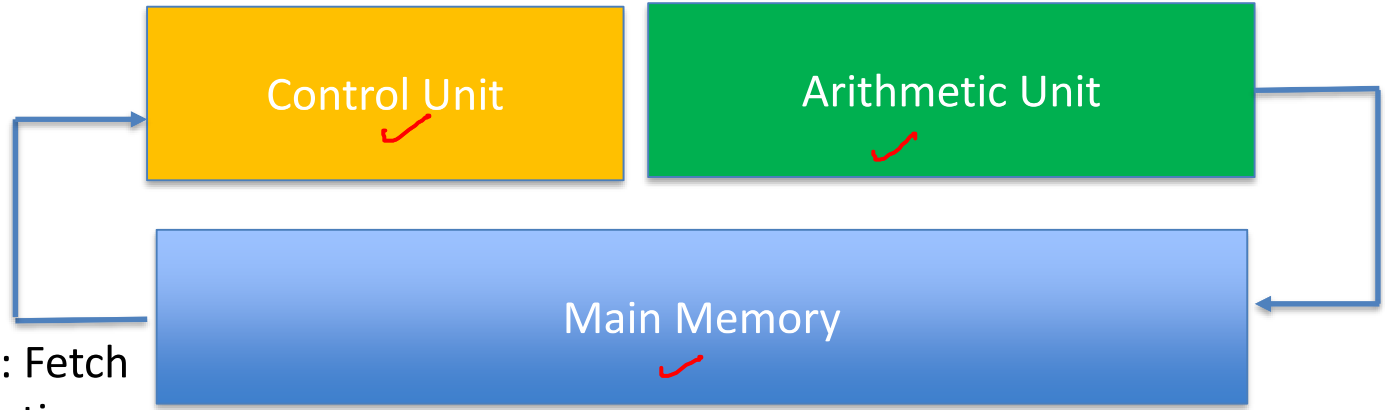
Outline

- ~~How to express real life problems as numerical problems.~~
  - ~~Picture processing~~
  - ~~Processing text/language~~
- How a computer solves numeric problems?
  - Recap: Problems, Algorithms and Programs
  - Digital circuits
  - How numbers are represented
  - Parts of a computer
  - Machine language program, compilation.

Parts of a Computer

Step2. Decode instructions into commands

Step3. Execute commands

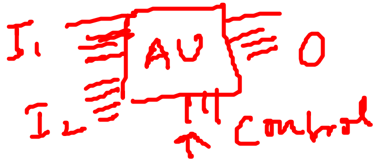


Step1: Fetch instructions from memory

Step4. Store results in memory

Various Units

- Memory: Organized in bytes (groups of 8 capacitors)
  - Memories of present day computers contain few Gigabytes, Giga = $2^{30} \approx 10^9$, billion.
  - Each byte in the memory is assigned a distinct number, or an address.
    number
    Much like houses on a street!
  - In a memory with N bytes, the addresses go from 0 to N-1
- Control Unit: Tells other parts of the computer what to do
- Arithmetic Unit:



  - Input1, Input2, Output (wires)
  - **Control** (several wires): Number appearing on the control wires will say what operation (sum, product etc) should be performed on input

C++ programs and Machine language

- On a modern computer you write a C++ program.
- A prewritten program, "compiler", translates your C++ program to a "Machine language program"
  - When you type s++ square.cpp the compiler is called upon to compile your file square.cpp.
  - It creates the "machine language program" which by default is called a.out on unix.
- When you type ./a.out : RAM → faster
  - a.out gets loaded into main memory (from hard disk)
  - Then a.out executes.

Permanent, Slower

Machine language instruction

Machine language instruction = sequence of numbers

- Possible structure of machine language instruction:
  - First number = says what operation to perform ("operation code")
  - Second and third numbers : addresses in memory from where the operands are to be taken
  - Fourth number: address in memory where the result is to be stored.

(Fictitious) Examples of machine language instructions

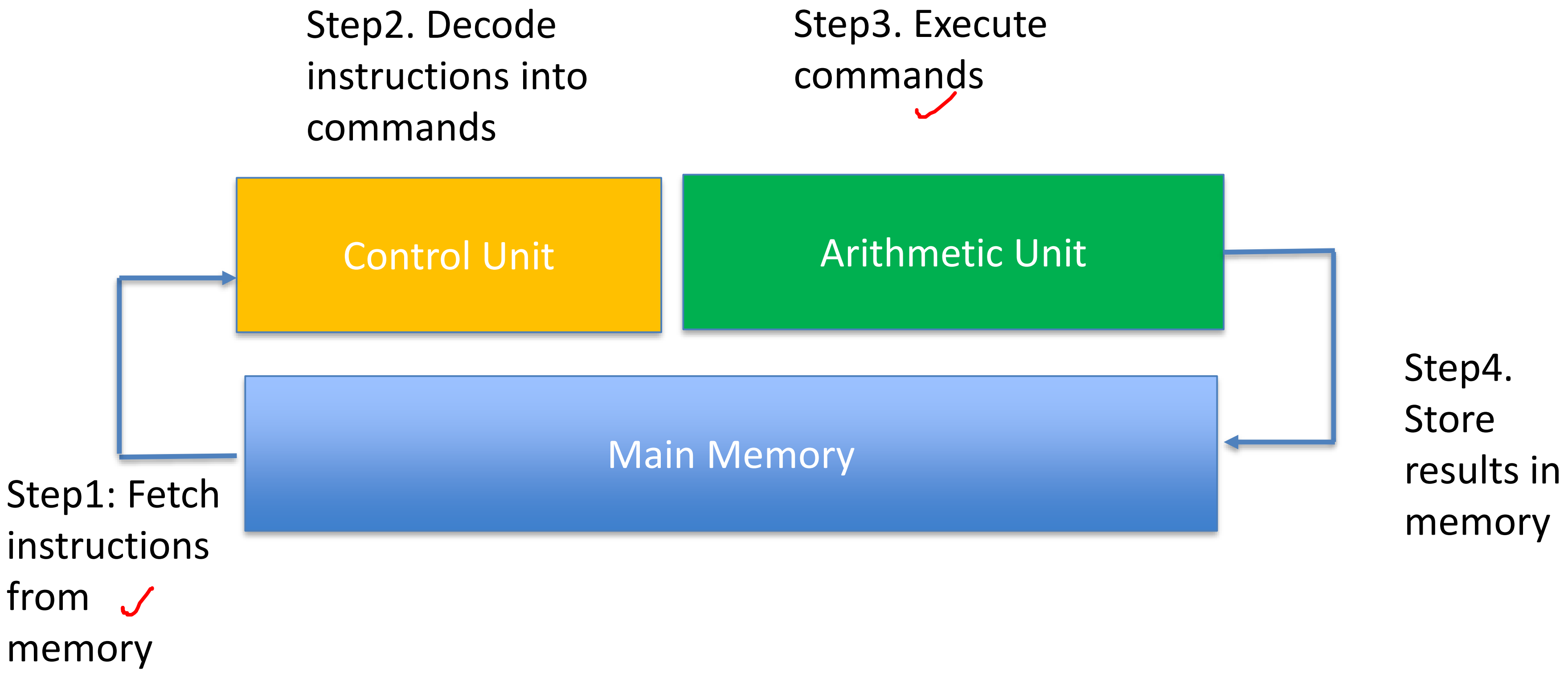
- Hypothetical instruction: 57, 100, 200, 300
- Operation code 57 might mean "multiply"
- On reading the above instruction **Control unit** does the following:
  - Tells the memory to read the words at the first two addresses and send them to the Arithmetic unit.
  - Tells the arithmetic unit to perform multiplication by sending appropriate number on its control wires. "57"
  - Moves the result from the arithmetic unit to the memory
  - Tells memory to store the received word into the word at the third address
- This instruction causes the product of the numbers stored in addresses 100, 200 to be stored in the address 300.

Machine language program (hypothetical) example

- Example: 57, 100, 100, 100, 57, 100, 100, 100
- This contains two instructions.
- Both instructions cause the word at address 100 to be multiplied by itself and the result stored back in address 100.
- After executing the first instruction, address 100 would contain the square of the number that was present before.
- The second operation would repeat the squaring operation.
- Thus this is a machine language program to compute the fourth power of a number.

Actual machine languages are more complex, will have many more instructions.

Parts of a Computer



Concluding Remarks

- In order to solve problems on a computer, they must be formulated as problems on numbers.
  - Numerical codes can represent non numerical entities
  - E.g: ASCII code for alphabets, sequence of number for pictures
- Algorithm: precise sequence of calculations needed to solve a certain problem
  - Human beings have been using algorithms well before computers were invented, for pencil paper calculations.
  - You yourself know many algorithms.
  - Computer algorithms are very similar to paper pencil algorithms
  - Express those in a programming language. (Rest of the course!)

Concluding Remarks

- Current/charge/voltage values in the computer circuits represent bits (0 or 1).
  - Numbers (integers, reals) can be represented in binary
- Circuits can be designed which take as input voltages representing numbers and produce voltages representing their product/sum/…
- Memory in a computer is organized as a sequence of bytes, each byte can be identified by its address.

Concluding Remarks

- Machine language program : sequence of machine language instructions
  - Must be present in the memory
- Control unit reads machine language instructions and interprets them
  - Decides what needs to be done
  - Sends control signals to other units and makes them do the needful
- Users write program written in high level language e.g. C++
  - User program compiled into machine language by compiler
  - Machine language program in memory is executed