Algorithms, Computer Systems, and Programming Languages

CSC121, Introduction to Computer Programming

• what is a computational process?
• what is an algorithm?
• what is a computer?
• brief overview of computer organization
• computer processing as a discrete state machine
• assembly language programming
• high-level language programming
• the landscape of computational problems

Computational Processes

• a computational process denotes a collection of rules, or a sequence of operations or transformations whereby some resulting information is derived from a set of initial or given information and conditions.

Algorithm

• an algorithm is a plan for directing a computational process.
• algorithms should satisfy these conditions
  – each of the steps must be clearly and precisely defined.
  – each step must be effective (for the agent executing the process)
  – the steps must be ordered properly
  – the process must terminate at some point
Devising an Algorithm

- Creating an algorithm is a process that involves these considerations:
  
  a) *what is the result?* (specifications for the task, process, answer, or product)
  
  b) *what do I know?* (information that is given or input into the computational process)
  
  c) *how do I derive (a) from (b)?* (modifications, transformations, steps required to produce (a))

Computer Systems (1)

- Today's generation of computer systems are automated, programmable, electronic digital processing machines.
  
  - electronic
  - digital
  - programmable
  - automated

Computer Systems (2)

- A computer system is an electronic digital data processing machine.
  
  - data—symbolic representation of information
  - process—a sequence of states during which the data are modified in prescribed ways
  - program—a set of instructions that direct the process

Computer Systems (3)

- A computer system is comprised by software that directs the processing performed by hardware.
  
  - software—programs
    - program—a set of instructions that direct a process
  - hardware—devices that make up the physical system of the computer
Hardware

- **Hardware** denotes the devices that comprise the computer system

**Hardware**
- **processor**
  - CPU
  - main memory
- **secondary memory and I/O subsystem**
  - auxiliary memory
  - input/output devices

**Processor System**

- **main memory** stores both data and instructions used for processing
- **bus** provides interconnections
- **central processing unit**: manages the instruction-execution cycle

**Instruction-Execution Cycle**

- **fetching** the next instruction
- **decoding** (or interpreting) that instruction
- **executing** what the instruction prescribes
Memory

- Information (data) is stored in uniform-sized units.
- Each unit has a unique **address** that identifies it.
- **Reading** from memory:
  - Copy data from a memory location
  - Non-destructive
- **Writing** to memory:
  - Moving and replacing data in a memory location
  - Destructive

Secondary Memory and I/O

- **Secondary memory**
  - Nonvolatile storage
  - Cheaper, mass storage
- **Input/Output devices**
  - **Input**–translates human-readable forms of information to machine-readable data
  - **Output**–translates machine-readable data to human-readable forms

Discrete State Machine

Digital computers... may be classified amongst the 'discrete state machines.' These are the machines which move by sudden jumps or clicks from one quite definite state to another. These states are sufficiently different for the possibility of confusion between them to be ignored.

Turing, "Computing Machinery and Intelligence" (1951)

Capturing a Discrete State

- The current values of any data objects specified for the process
- The status of the processor
  - Temporary values and flags stored there
- The status of input and output operations
- The address of the next executable instruction in the program
Software

• computer programs are commonly called software
• each machine instruction in a program typically corresponds to a type of operation that the computer’s processor can perform
• the set of possible processor or machine instructions is called the processor or machine language

Assembly Language Programming

• assembly language programs contain symbolic instructions representing elements of some machine language instruction
• each instruction typically has
  – a mnemonic for the operation code
  – literals or labels for the operands

Assembly Language Programming

• Some operands are literals
  – e.g.,
  ADD 3
  [ACC ← ACC + 3] where ACC denotes the Accumulator
• Some operands are labels signifying indirect (indirect codes)
  – e.g.,
  ADD LOC 1024
  [ACC ← ACC + value(LOC 1024)]

Assembly Language Programming

Labels may function as …

• constants, literals or values that may change from time to time, but are fixed or static for the current execution of the program
• variables, these denote values that are dynamic, that is, may change in the course of processing
• pointers (handles), these denote values that are the address of other locations containing values
Assignments

- Putting a data value into memory is called (algorithmically) an **assignment**

  \[ \text{ACC} \leftarrow 3 \]

- In Python (and most programming languages), it is written in this form

  \[ \text{name} = \text{value} \]

  where name is a symbolic label and value expresses some data value

---

Conditional Processing

is the capability of the processor to choose between alternative actions based on evaluating current data

---

Conditional Processing

...is made possible at the processor level by branch instructions

- **unconditional branch** (jump)—change the address of the next instruction to the one indicated in the branch instruction

- **conditional branch** (jump)—if some condition is true, then change the address of the next instruction to the one indicated in the branch instruction

---

Conditional Processing

...is made possible at the processor level by branch instructions

---
Subroutines

library program units that can be linked with the main program

call subroutine

Assembly Language Programming

```
main:    subu  $sp, $sp, 32
         sw   $ra, 20($sp)
         sw   $sp, 16($sp)
addiu   $sp, $sp, 28
li      $v0, 4
la      $a0, ar
syscall
li      $a0, 10
jal     fact
addiu   $a0, 3v0, $zero
li      $v0, 1
syscall
lw      $s1, 2v0($sp)
lw      $s1, 16($sp)
addiu   $sp, $sp, 32
jr      $ra
```

Review: Assembly Language Programming

- instructions have a low-level of abstraction because they model the actual operations of the native processor
- instructions represent (nearly) a 1–1 relation with those of the corresponding machine language
- assembly language programs are not portable
- assembly languages are considered low-level programming languages
High-Level Programming Languages

- High-Level Language (HLL) instructions are more abstract
  - HLL instructions express higher-order operations
- single HLL instructions substitute for a sequence of low-level machine language instructions. In short, there is a 1–many relation between HLL instructions and machine language instructions
- HLL programs are portable to other processors
  - provided a suitable means of translation

Advantages of High-Level Programming Languages

- programmer is relieved of many implementation details in performing tasks
- HLLs are designed to be more compatible with the way we think and solve problems
- HLL programs can be adapted more easily for a variety of computer systems and modified more easily for different functions
- HLL programs are more concise

Translating HLL Programs

- interpreters—decode and execute programs one instruction at a time
  - translation on the fly
  - emulates the way the processor functions
- compilers—analyze and translate the entire HLL source program to produce an executable machine language version
  - like a prepared text translation

Computational Problems: The Landscape

- Some computational problems have solutions, but these solutions cannot be computed effectively because we lack the practical means to do so
  - some problems may be solvable when we have faster processors or greater memory capacities
  - for others, we will never have the resources to effectively compute them with current solutions or algorithms
- Example: the traveling salesperson problem
Computational Problems: The Landscape

• The brute-force solution for the traveling salesperson problem (TSP) has a factorial growth rate
• Computer scientists often describe the behavior of algorithms using a notation called “Big O”
  – this denotes the upper bound for the worst-case performance of an algorithm
  – TSP = O(N!), for N cities (or nodes)

Computational Problems: The Landscape

• Suppose that we had hardware that was capable of analyzing one million individual paths or trips per second.
  – N = 10 solution < 4 seconds
  – N = 100 solution = approximately 10^49 centuries

• Problems with algorithms like for TSP are said to be intractable

Computational Problems: The Landscape

• there is another class of problems that are not solvable
  • example: The Halting Problem
    – Suppose that we created a program that could analyze any other program.
    – Specifically, analyze an input program with a specific set of input data
    – Our program must be able to determine whether the input program will halt—i.e., complete its task
    – An effective solution has two possibilities: "Yes", the input program terminates or "No", the input program never terminates or halts

Computational Problems: The Landscape

• Alan Turing demonstrated that the Halting Problem was unsolvable in 1936
  • headed the group at Bletchley Park that cracked the Enigma Machine in WWII
  • developed the ACE computer in 1946
  • introduced what is now called the "Turing Test" to determine whether a machine could be considered intelligent
### Computational Problems: The Landscape

- **Undecidable problems**: problems for which there can be no effective algorithm

- **Intractable problems**: these are decidable but have no practical algorithmic solutions

- **Tractable problems**: these are both decidable and have algorithms that are or would be practical to implement

---

### Review

- what is a **computational process**?
- what is an **algorithm**?
- what is a **computer**?
- brief overview of **computer organization**
- computer processing as a discrete state machine
- **Assembly language programming**
- **High-level language programming**
- the landscape of **computational problems**