TURING MACHINES AND MODERN COMPUTERS

Abhijit Das

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

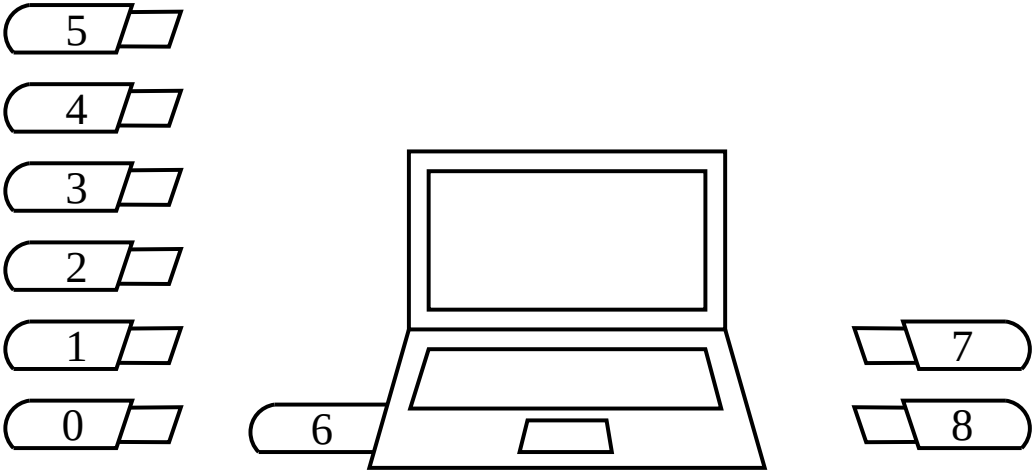
What is a Modern Computer?

- No standard well-accepted definition.
  - Laptop and desktop computers.
  - Servers, clusters, GPUs.
  - Supercomputers.
  - Computers in a network.
  - Quantum computers.
  - FPGAs and ASICs.
  - Embedded processors.
- Modern computer architectures are rather complex.
- How to make a mathematical model of such a machine?
- Let us take a laptop or desktop computer as a representative.
- We **informally** show that such a modern computer is equivalent to a Turing machine.

Simulation of a Turing Machine by a Modern Computer

- Input: A Turing machine M and an input w for M.
- Output: A C program to simulate the work of M on w.
- Seems like no big deal.
- Issues to worry about:
  - A modern computer has limited memory.
  - How to simulate the infinite tape of M?
  - What if the description of M does not fit in the computer's memory (including hard disks)?
  - What if w is so large that it does not fit in the computer's memory?
  - What if Σ or Γ for M is too large?
- Solution: Use an unlimited source of external memory (like pen drives).
- Need to update the OS to support this simulation.

Handling Infinite Memory



Handling Infinite Memory

- Number the pen drives used as $0, 1, 2, 3, \ldots$.
- Let c be the current pen drive in use.
- The pen drives storing the part of the tape to the left of the c-th drive are organized as a stack.
- The pen drives storing the used part of the tape to the right of the c-th drive are organized as another stack.
- In each of these stacks, the most recently used drive is at the top.
- If the head wants to move to the left of the first cell of the current drive, push the current drive to the right stack, pop and insert the drive at the top of the left stack.
- If the head wants to move to the right of the last cell of the current drive, push the current drive to the left stack, pop and insert the drive at the top of the right stack.
- If the right stack is empty, a new pen drive is opened.

Handling Enormous Inputs or Alphabets

- The input w for M may span across multiple pen drives.
- If Γ is very large, the entire memory of a pen drive may be insufficient to store a single symbol.
- For a 16 GB pen drive, this means $|\Gamma| > 2^{137438953472}$.
- To solve this problem, encode the symbols in binary.
- Breaks inside a tape symbol is allowed.
- There is now a reading mode to collect all the bits of an individual tape symbol and to interpret the bit collection.
- This may increase the number of states of M significantly.

Handling an Enormous M

- The input w may span across multiple pen drives.
- M should reside in the internal memory of the computer.
- The internal memory is finite, and may fail to accommodate M.
- Solution: Send M to the tape (a sequence of pen drives).
- Do not simulate M on w.
- Simulate the Universal Turing Machine U.
- U simulates M on w upon the input of $M\#w$ on the tape.
- U is a fixed machine of moderate size.
- U fits in an amount of memory independent of M and w.

Stored-Program Computers

- Memory hierarchy: Registers, Cache memory, main memory, hard drives, removable storage.
- Assume that a modern computer has infinite memory.
- We flatten the hierarchy.
- The memory is organized as an array of words (32-bit or 64-bit).
- The words are addressed as $0, 1, 2, 3, \ldots$.
- Compiled programs reside somewhere in the memory.
- Computer programs consist of instructions:

  Copy w_i to w_j.
  Add w_i and w_j, and store the result in w_k.
  Compare w_i with w_j.
  Jump to the address l.

- The input may be stored in the memory or come from a special device.

The Fetch-Decode-Execute Cycle

- Each instruction consists of
  - An opcode (the operation)
  - Zero or more operands
  - The destination
- A counter stores the address of the instruction to execute next.
- The opcode is first fetched.
- The opcode is decoded and the requisite operands are fetched from memory.
- The operation is performed.
- The result is written back to the destination address.
- If the instruction is a jump instruction, the instruction-address counter is rewritten by the jump address.
- Otherwise the counter is incremented to the address of the next instruction.
- Initially, the counter is loaded with the address of the first instruction.

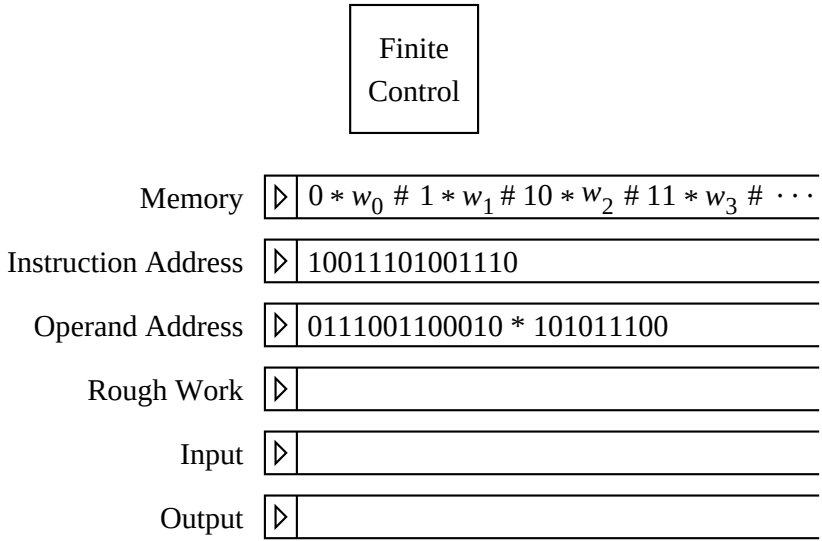
Simulation of a Modern Computer by a Turing Machine M

- M contains six tapes.
- The first tape stores the memory of the computer as

  $$\triangleright 0 * w_0 \# 1 * w_1 \# 10 * w_2 \# 11 * w_3 \# 100 * w_4 \# 101 * w_5 \# \cdots,$$

  where w_i is the content of the i-th word encoded in binary.
- The second tape stores the instruction address.
- The third tape stores the operand addresses.
- The fourth tape is used for scratch work.
- The fifth tape stores the input of the user.
- The sixth tape is used to write the outputs to the user.
- The finite control stores the information of the instruction set of the computer.

The Organization of M



Simulation of an Instruction Cycle

- Locate the word whose address is on the second tape.
- Read and remember the opcode in the finite control.
- Copy the operand addresses (if any) from the rest of the instruction to the third tape.
- Locate the operand addresses on the first tape, and copy the operands to the fourth tape.
- Apply the operation on the operands on the fourth tape.
- The result is produced in the fourth tape.
- Locate the destination address on the first tape.
- Copy the result to this location (relocation of the rest of the tape content may be needed).
- If it is a jump instruction, replace the second tape by the content of the third tape.
- Otherwise, increment the second tape by an appropriate amount.

Handling Input/Output

- Input and output are not part of the computation.
- There are specific read/write instructions.
- A read instruction scans the input from specified addresses on the fifth tape (or sequentially).
- A write instruction prints the output to specified addresses on the sixth tape (or sequentially).
- Peripheral devices handle these tapes.
- A keyboard controller can write to the fifth tape.
- A screen controller can read the sixth tape to change the graphic display.
- M may have an accept state t and a reject state r.
- Upon successful or unsuccessful termination, M goes to t or r, and halts.

Running Time of the Simulation

- Suppose that the computer takes n steps for some computation.
- Assume that the words of the computer are of fixed size (like 64 bits).
- Overflows during arithmetic operations are ignored.
- It can be calculated that the simulation by the TM requires $O(n^3)$ steps.
- If we use a single-tape Turing machine, the number of steps is $O(n^6)$.
- If n is a polynomial in the input size, the running time remains polynomial (but of a higher degree).
- The notion of polynomial-time computability is quite robust.
- Turing machines can perfectly capture this notion.

You may informally argue how a Turing machine can simulate a C program (instead of its compiled version).