CS 354 Spring 2021 (Park)

CONCEPT OF MODERN COMPUTER, HARDWARE V. SOFTWARE, PROGRAM V. PROCESS

1. Concept of modern computer conceived by Alan Turing in 1936

A "computer" (term coined by Turing) has two components: CPU and memory.

I/O devices were not a key issue then (today's computers tend to have lots of I/O devices, many operating at high speeds).

CPU has movable read/write head, interacts with memory by moving the head (left/right in sequential memory, arbitrary in random access memory) and reading/writing bits to/from current memory location.

Two types of computers invented by Turing:
a. Computers whose program (i.e., logic of what it does) is hardwired into CPU.
b. Computers whose program is given as input to a CPU as software, i.e., bits in memory.

2. ASIC computers

In the first type, what function of what a computer does is hardwired into CPU.

We can consider the example application (app) of unary addition.

Ex.: To perform 2 + 5, the input in memory is given as 110111110. We assume the head is initially positioned over the first bit. If the computer performs its task correctly, the output should be 11111110 which corresponds to 7.

Such computers where the app is hardwired (hence input to computer is just data, no software/program) are called ASIC (application specific integrated circuit) in today's terminology.

Lots of ASIC chips used today (they perform specific tasks very fast).

In CS we refer to these computers as (regular) Turing machines, in honor of their inventor.

The benefit of ASICs are speed, but a key drawback that to implement any useful function (i.e., app) we have to build special-purpose hardware which is costly and not considered a good business model in many cases.

3. Programmable computers

In 1936, Turing also conceived of programmable computers which is, by default, what we mean when we talk about computers today.

In the case of unary addition app, the program to perform unary addition is not built into hardware (i.e., CPU) but is given as input (bits in memory) called program or software.

Thus the input to CPU, inscribed in memory has two parts: program and data.

A program consists of a sequence of instructions that the programmable computer (i.e., CPU) is built to understand.

In the case of unary addition, we designed a CPU capable of understanding 7 instructions:

movL, movR (move head left/right)
read (read bit in memory to CPU register), write (opposite of read)
cmpEq (compare if register value equals a specified constant, e.g., cmpEq reg1 $1, and stores the outcome of the comparison in a special true/false register, call it TF register)
jmpT (conditional branch that jumps to specified program location if the value of TF is true/1)

goto (unconditional branch)

Hence the hardware designer builds a CPU (which contains registers reg1 and TF) that is capable of implementing these 7 instructions in a fetch-decode-execute cycle.

A software designer (app developer) writes a program which is just a sequence of instructions from these 7 types.

Our program for implementing a unary addition app had 15 lines of instructions.

To recognize Turing's contribution, these programmable computers that we use today are called "universal" Turing machines.

"Universal" means that a set of instructions (like the 7 in our version of programmable computer, i.e., CPU) when combined into a specific sequence of instructions (i.e., program) can perform the task of any ASIC computer that implements the same task/function in hardware.

It can be shown -- not difficult but requires a little care -- that just a few instructions (such as our 7 for reading/writing memory and conditional branching) suffice to make the CPU universal.

Our 7-instruction CPU would be a RISC (reduced instruction set computer) machine. Perhaps we could call it a very reduced instruction set computer (VRISC).

When we add a great deal more instructions to a CPU's capability for the software designer's convenience, then the CPU's expanded instruction set (some say bloated) may be classified as a CISC computer.

4. Notion of process

A program is just static code that's given as input (along with data) to a CPU (henceforth, by default, always universal computer).

When the CPU fetches the first instruction of the program in its fetch-decode-execute cycle, then we call the activated or running program a process.

A CPU keeps track of the location of the next instruction in memory to be executed in a special register called program counter (PC). In Intel x86 jargon, PC is referred to as instruction pointer (IP).

When a (universal) computer executes an app (i.e., process), and only when the process is completed does it execute another app, this is called a batch processing of apps.

If the batch mode computer is asked to run two apps, then the memory seen by the CPU may look like:

program1 data1 program2 data2

When a CPU is given multiple programs to run (say 2) but it switches its attention from one to the other before the processes have run to completion (their first instructions have been executed but their last instructions have not), this called concurrent processing of apps.

Who decides how much time the CPU spends on the first process before switching to the second process, and back and forth?

This is one of the chores of the operating system, a software layer that mediates the sharing of hardware resources (in this case CPU, but could be network interfaces such as WiFi and Bluetooth).

If the computer is equipped with 2 CPUs, then the two processes could run on one CPU each, thus achieving real concurrency. When there is only one CPU, then there is an illusion that two processes run concurrently -- in reality the CPU switches its attention back and forth, requiring saving of register values and other bookkeeping to keep the processes running correctly despite them sharing a single CPU -- which we call apparent concurrency.

The switching back and forth between 2 or more running apps (i.e., processes) is called context switching.

As we can surmise, context switching incurs quite a bit of overhead and is thus considered a costly operation.