How does hardware execute software?

1 Timeline

- Analytical engine
- Punch Cards
- Cloud Computing
- DNA Computing

2 ISA

The ISA is the interface between the hardware and software

- The view the programmer has of hardware
- Includes everything programmers need to know to program the processor

The ISA includes

- Organisation/structure of programmable storage
- Instruction sets and formats
- Modes of addressing and accessing data items within memory

A primary component of the ISA is its **assembly language**

- High level language is compiled into assembly language
- Assembly language instructions are very low level and go on to be implemented as machine code

3 Our processor

We'll take a much simplified look at the MIPS ISA (Each one individual for brand/processor)
Let us assume that:

- 4Gb memory has 2^{32} memory locations
- Words 4 bytes long
- When we request memory, the contents of the locations m,m+1,m+2 and m+3 are returned
- There are 32-bit registers names $zero (which always holds 0), $s1, $s2...,$s7
- Every assembly language is 32 bits wide such as: - insert from slides
  - Load some value from memory, for use (lw) in brackets means take the value out, $s1 is the location to store the data [lw $s1, ($s2]
  - Addition $s1 takes the value of $s2 plus the number x
  - Branch to move around in memory `beq $s1, $s2, $s3`
  - Jump to a particular memory location
- We can substitute different registers in these instructions (register ≈ accumulator in VN architecture)

4 A simple searching program

- Program finds the maximum value given in the array
- A C compiler converts it into assembly language

5 Assembly language in action

- Set program counter to 0
- Use fetch-decode-fetch-execute method
- Initially PC0, lw s1,(s2). Look at 44, fetch data from there
- Iterate program counter
- PC4, fetch memory, add 1 to s3 and store back
- PC8, look at registers s3 and s4. Update PC to 40 if equal, do nothing if else
- PC12, add 4 to s2 and store back in s2
- PC16, load data from s2 and store in s5
- PC20, if s1 is less than s5, set s6 to 1
- PC24, Has found new bigger number
- PC28
- PC32
- PC36, jump to memory cell 8

6 Assembly to machine code

- MIPS to machine code
- 3 types of ins
  - Data transfers
  - Registers
  - Jumps
- MIPS has 32 registers
- Registers encoded to 5 bits

6.1 I type instructions

An I type instruction has the following format:
Add diagram

- Take 1 input register that can vary as their inputs
- Each instruction has a different op code
- 5 bits for each of source and destination
- 16 bits for address - tricks used to address 32 bit addresses

Types:

- Add
- Branch
- Load memory location

6.2 R type instruction

Two input instructions
Examples:

- Addition
- Set less than

Structure

- op code all zeros
- Last 6 bits determine function
- shift for logical operations

6.3 J type operation

Jumps

- 6 bit op code
- Address (26) says where to jump to

7 Overview

1. Start with source code
2. Do compilation
3. Turn to assembly
4. Machine code
5. Voltages

- Machine code sometimes written in hexadecimal to make it easier to read and write

Operating Systems

8 Intro

- Operating system manages the hardware for you
  - Provides interface between applications and hardware
- Abstracts the hardware for applications
  - Deals with system calls
  - Provides data security
- OS **kernel**
  - Main part of the OS
  - Loaded at boot time
  - Has total control of the CPU
- Main functions
  - Virtualization - Hides complexity away, for looking at disks etc
  - Starts/Stops programs, allocating and deallocating memory, suspend execution
  - Handles IO - interrupts
  - Maintains file system - access restrictions
  - Deals with networking and provides security
  - Facilitates error handling and recovery

9 IO

IO devices

- Hard disks
- GPU

Bus:

- Lines of communication (wires)
- Cheap and versatile but very slow in comparison to the CPU, leading to a bottleneck

OS helps alleviate the bottleneck Interrupts:

- CPU instructs device and continues with other tasks
- When device finishes it raises an interrupt on a bus time

10 Processes

Process - A program in execution

- Not a program on the disk
- Multiple processes (maybe from the same program)

Process = threads + address space (allocated memory)

- **Thread** - A sequence of instructions in a sequential execution context
- **Address space** - Memory locations a process can R/W to/from

Muliple threads or processes need to

- Communicate and synchronize

Mutual exclusion

- Two threads want to do the same thing
  - Thread 1 completes before thread 2 starts
  - Thread 1 and thread 2 interleave (run at the same time)
    - Thread 1 reads value of c and adds 1
    - Thread 2 reads value of c (still 0) and adds 1
    - Thread 1 writes value to memory
    - Thread 2 writes value to memory
    - In this method c=1, however with the previous method c was 2
  - Each thread needs to acquire the **critical section** of access to c (can't read a data until other thread has checked back in)

11 Virtual memory

- Many processes means a lot of RAM is required, but often there is not enough
- Virtual memory
  - Uses all the available memory to run processes
    - Pages memory out to the hard disk
      - Keeps a longer address list, as can now use storage, this is slower, but necessary
      - Where allocated depends on priority
      -
- Each sole process things it has
  - Sole access to CPU
  - Sole access to its address space

  But
  - It time shares CPU access
  - Its physical access might reside on disk

  More
  - Multiple processes share CPU and memory, time divided amongst processes

12 Process life cycle

- New: process being created
- Ready: not on CPU, but ready to run (put in queue)
- Running: process executing on CPU
- Blocked: Waiting on an event (sit in blocked que)
- exit: process finished

State transitions:

- admit: Add to queue
- dispatch: Scheduler gives cpu to runnable process
- timeout/yield - running process gives up cpu
- event-wait: process waiting for
- event: event occurs; wake up process
- release: process terminates; release resources

13 Process control block

- Each process has unique id and state in the FSM they are, also extra info with scheduling, program counter, other CPU registers, management info (where the data was stored), various other information
- Links to prev/next control blocks

14 Context swiching

- What do when switching processes
- Sto state A
- Restore state B
- Run
- Sto B
- Loa A