CS151: Computer Architecture

Professor Reinman

Thilan Tran

Winter 2021

Contents

CS151: Computer Architecture

- stack of layers between a user app and the hardware of a computer:
  1. user apps
  2. system apps
  3. Java API framework
  4. native C/C++ libraries
  5. hardware abstraction layer
  6. Linux kernel
     - all the way down to the actual silicon hardware, AKA **SoC (System on Chip)**
     - for a SoC, instead of just a single CPU, the chip contains multiple components packaged together that interact with each other:
       - eg. CPU, GPU, I/O, media, etc.
       - better integration and increased interaction since on-chip components are embedded and stacked together more tightly, vs. pin-out components
  - need to balance specificity of hardware with portability of applications
- more levels ie. layers of program code:
  1. high-level language
     - very abstracted, portable
  2. assembly language
     - textual representation of instructions
  3. hardware representation
     - encoded instructions and data in binary
     - what machine actual reads
- the **ISA (Instruction Set Architecture)** exposes a set of primitives to the layers *above* it in order to work with the silicon hardware *below* the ISA:
  - ie. interface or template for interactions between the CPU and the drivers, OS, apps above it:
    - defines instructions, calling protocols, registers, etc.
    - eg. instructions like `add`, `movsql`, etc.
      - in machine language, represented on binary
  - in this class: how do we implement the silicon *hardware* to actually implement an ISA, using the building blocks of a microprocessor?
- evaluating performance:
  - performance is one aspect of evaluating a microprocessor:
    - other areas include power management, reliability, size, etc.
    - can be evaluated on latency (delay), vs. throughput (bandwidth)
  - considering the performance equation:

$$ET = IC \times CPI \times T_C$$

    - ET is the execution time
    - IC are the instruction counts
        - *dynamic* runtime count, rather than *static* count in a program
    - CPI are the cycles per instruction:
        - due to the synchronous nature of memory in circuits
        - but different instruction *types* will have different cycle counts
        - thus to find the CPI as a *weighted average*, need to compute the percentage and cycles of instruction types that *make up* the overall IC
        - note that the CPI is a *particular* count dependent on the program and the hardware it runs on!
    - T_C is the cycle time ie. clock period:
        - at a shorter clock period, CPI may increase since there is less time for an instruction to complete
        - eg. 3.5 GHz, 4 GHz, etc. (recently topping out due to power efficiency limitations)

1. algorithm affects IC, possibly CPI
2. programming language affects IC, CPI
3. compiler affects IC, CPI
4. ISA affects IC, CPI, T_C
5. Hardware affects CPI, T_C