Computer Systems IV: Computer Engineering

Noah Singer, George Klees

Montgomery Blair High School Computer Team

October 12, 2017

Overview

Section 1: CPU Architecture

Terminology

- Code is stored as binary data (**machine code**)
- We consider simpler **reduced instruction set computing (RISC)** models
  - Sequence of **instructions**
  - Each instruction has a numerical **opcode** and possibly some **operands** (addresses, direct numbers, registers, etc.)
- CPUs are organized in an **architecture**
  - **Instruction set architecture (ISA)**: types and meanings of instructions (e.g. x86)
  - **Microarchitecture**: specific layout of CPU itself and implementations of instructions (e.g. Pentium III)
  - We consider the simple and widespread **reduced instruction set computing (RISC)** ISA

Common instructions

- Memory loads and stores
- Register switches (moves)
- Arithmetic and logic computations
- Comparisons
- Control flow instructions (**branches** and **jumps**)
- Stack instructions
- Empty operations (**nops**)

Speical instructions

- **Vector instructions** allow computations on many values in a large array simultaneously
- **Floating point instructions** extend CPU arithmetic to non-integers using a **floating-point unit (FPU)**
- Syscalls
- **Interrupt** management

Registers

- CPU contains a small amount of specialized memory called **registers**
- **General-purpose registers** store results of computations
  - **Load and store** instructions access memory, transferring to and from GPRs
- Various **special registers** have specific purposes for the CPU
  - **Instruction pointer** contains memory address of currently executing code
  - **Stack pointer** contains memory address of current stack
  - **Link register** contains return address

Code execution

1. CPU retrieves instruction from memory pointed to in instruction pointer (**instruction fetch**)
2. CPU decodes instruction
3. *Optional*: CPU performs the computation
4. *Optional*: CPU reads from memory and/or writes to memory
5. *Optional*: CPU writes results back into registers
6. CPU advances instruction pointer

Essential components of a modern CPU

- **Arithmetic/logic unit (ALU)** performs mathematical computations
- Memory interface containing **memory management unit (MMU)**
- **Instruction fetcher** retrieves instructions from memory, **instruction decoder** translates the opcodes into operations within the CPU
- **CPU clock** is a crystal oscillating at some **clock rate** that controls instruction execution
- **Caches** optimize data access
- **Register file** is small memory containing registers

Caching

- Memory access can take many clock cycles
- CPU often keeps one or more **caches** of memory or other resources
- When a memory read is made, either:
  1. The address is already in the cache, and it is returned immediately (**cache hit**)
  2. The address is not in the cache, so a memory access must be made, and the value is copied into the cache (**cache miss**)

Section 2: Optimization

Instruction-level parallelism

- Often, the results of instructions do not **depend** on each other, so multiple instructions can be executed somewhat simultaneously
- **Scalar processors** execute instructions sequentially, so **superscalar processors** can execute (parts of) multiple instructions at the same time by having e.g. multiple ALUs
- **Very long instruction word (VLIW)** architectures promote instruction-level parallelism in software (generated by the compiler), so that code is structured with very little dependency

Pipelining

- CPU execution is driven by master **instruction clock**
- Instructions can often be divided up into discrete **stages**
  - Example (RISC): instruction fetch, instruction decode, execute, memory access, register writeback
- Instead of executing each instruction's stages individually, instructions progress through the pipeline in stages, increasing **throughput**

Hazards

- Pipeline stages can interfere with each other, causing faults called **hazards**
- **Bubbles** can be introduced to halt all stages until one stage propagates through
- **Data hazards**: the results of one instruction depend on the results of a previous instruction still in the pipeline
- **Structural hazards**: a single component in the CPU is used twice during the same clock cycle
- **Control hazards**: one instruction changes control flow while other instructions have already began to propagate through the pipeline

Speculative execution

- Conditional branches cause the pipeline to halt until the value of the condition is known (expensive)
- CPUs employ **branch prediction** to guess the result of a conditional branch and begin executing it
- When the same condition is evaluated several times, its results can be stored and analyzed to predict a new condition
- If the wrong condition was predicted, incomplete instructions in the pipeline must be "unrolled" and "refilled" (expensive)

Out-of-order execution

- Some CPU resources are costly to use (e.g. memory, certain arithmetic)
- When instructions are independent, they can be reordered to optimize CPU resource usage
- Data dependency can be further reduced by **renaming registers**