Class 2. Computer Architecture (1/30/07)

Computer Architecture

- The components that make up a computer system, and their interconnections.
- Basic components (draw schematic):
  1. Processor.
  2. Memory.
  3. I/O (input/output) devices.
  4. Communication channels (buses).

  We will discuss each of these in turn.

Processors

- Component that executes a program.
- Most PCs have only one processor (CPU)—these are "serial" or "scalar" machines.
- High-performance machines usually have many processors—these are "vector" or "parallel" machines.
- Processors execute a...

  fetch—get instruction and/or data from memory;
  decode—store instruction and/or data in register;
  execute—perform operation, storing results in memory

  ...cycle (e.g., `LD A,R1`; `LD B,R2`; `ADD R1,R2,R3`; `STORE R3,C`).
  - Instruction address held in program counter (PC).
  - PC incremented after each cycle.
- Very primitive commands! "Compilers" or "interpreters" are used to translate high-level code into such low-level operations.

Cycle

- Timing of cycle depends on internal construction and complexity of instructions.
- Quantum of time in a processor is called a "clock cycle." All tasks take an integer number of clock cycles to occur.
- The fewer the clock cycles for a task, the faster it occurs.
- NOTE: Higher clock speeds imply faster heating of components, increasing cooling requirements.

Measuring CPU performance

- Time to execute a program:

$$t = n_i \times CPI \times t_c,$$

where

$$\begin{aligned} n_i &= \text{number of instructions,} \\ CPI &= \text{cycles per instruction,} \\ t_c &= \text{time per cycle.} \end{aligned}$$

Improving CPU performance

1. Obviously, can decrease t_c. Mostly engineering problem (e.g., increase clock frequency, use better chip materials, ...).

2. Decrease CPI, e.g., by making instructions as simple as possible (RISC—Reduced Instruction Set Computer, as opposed to CISC—Complex Instruction Set Computer). Can also "pipeline" by performing different stages of fetch/decode/execute cycle at the same time, like an assembly line.

3. Decrease n_i any one processor works on:

   - Improve algorithm.
   - Distribute n_i over n_p processors, thus ideally

$$n_i' = n_i/n_p.$$

     – Actually, process of distributing work adds overhead: $n_i' = n_i/n_p + n_0$.

Defining CPU performance

MIPS—"million instructions per second": not useful due to variations in instruction length, implementation, etc.

Mflop/s—"million floating-point operations per second": measures time to complete a meaningful task, e.g., multiplying two matrices $\sim n^3$ ops.

- Computer A and B may have different MIPS but same Mflop/s.
- Often refer to "peak Mflop/s" (highest possible performance if machine only did arithmetic calculations) and "sustained Mflop/s" (effective speed over entire run).

"Benchmark"—standard performance test, e.g., `LINPACK`[1], `SPEC`[2], etc.

[1] See `http://www.netlib.org/benchmark/performance.ps`.
[2] Visit `http://www.specbench.org/`.

Memory

- Passive component that stores data or instructions, accessed by address.
- Data flows from memory ("read") or to memory ("write").
- RAM: "Random Access Memory" supports both reads and writes.
- ROM: "Read Only Memory"—no writes.

Bits & bytes

- Smallest piece of memory = 1 bit (off/on).
  - 8 bits = 1 byte.
  - 4 bytes = 1 word (on 32-bit machines).
  - 8 bytes = 1 word (on 64-bit machines).
- 1 word = number of bits used to store, e.g., single-precision floating-point number. Usually equals width of data bus.
- Typical home computers these days have $\sim$ 128–512 MB of useable RAM.
  - 1 MB = 1 megabyte or 1,048,576 (2^{20}) bytes (sometimes just 10^6).
  - 1 Mb = 1 megabit or 10^6 bits (rarely 2^{20}).

Memory performance

- Determined by access time or latency, usually 10–80 ns.[3]
  - Latency hiding: perform other operations while waiting for memory to respond.
- Would like to build all memory from fastest chips, but this is often too expensive.
- Instead, exploit "locality of reference."

Improving memory performance

- Typical applications store and access data in sequence.
- Instructions also sequentially stored in memory.
- Hence if address M accessed at time t, there is a high probability that address $M + 1$ will be accessed at time $t + 1$ (e.g., vector ops).
- Instead of building entire memory from fast chips, use "hierarchical memory":

[3]Note: DDR-SDRAM (double data rate, synchronous dynamic RAM), the newest type of memory, is speed-rated in terms "memory cycles," i.e., the time required between successive memory accesses, typically $\sim$ 10 ns or less.

  - Memory closest to processor built from fastest chips—"cache" (often more than one level).
  - Main memory built from RAM—"primary memory."
  - Additional memory buily from slowest/cheapest components (e.g., hard disks)—"secondary memory."

- Then, transfer entire blocks of memory between levels, not just individual values.

  - Block of memory transferred between cache and primary memory = "cache line."
  - Between primary and secondary memory = "page."

  How does it work?

  - If processor needs item x, and it's not in cache, request forwarded to primary memory.
  - Instead of just sending x, primary memory sends entire cache line (x, $x+1$, ...).
  - Then, when/if processor needs $x+1$ next cycle, it's already there.
  - Possible cache block replacement strategies: random, first-in-first-out (FIFO, i.e., replace block that has been in cache longest), least-recently-used (LRU).

  Hits & Misses

  - Memory request to cache which is satisfied is called a "hit."
  - Memory request which must be passed to next level is called a "miss."
  - Fraction of requests which are hits is called the "hit rate."
  - Must try to optimize hit rate ($>\sim 90\%$).

Measuring memory performance

- Define the "effective access time" as:

$$t_{\text{eff}} = (HR)t_{\text{cache}} + (1 - HR)t_{\text{pm}}$$

  where

$$\begin{aligned} t_{\text{cache}} &= \text{access time of cache,} \\ t_{\text{pm}} &= \text{access time of primary memory,} \\ HR &= \text{hit rate.} \end{aligned}$$

- E.g., $t_{\text{cache}} = 10$ ns, $t_{\text{pm}} = 100$ ns, $HR = 98\% \Rightarrow t_{\text{eff}} == 11.8$ ns, close to cache itself.

Maximizing hit rate

- Key to good performance is to design application code to maximize hit rate.
- One simple rule: always try to access memory contiguously, e.g., in array operations, fastest-changing index should correspond to successive locations in memory.

  Good Example

  - In FORTRAN:

```
DO J = 1, 1000
 DO I = 1, 1000
  A(I,J) = 0
 ENDDO
ENDDO
```

  - This references `A(1,1)`, `A(2,1)`, etc., which are stored contiguous in memory.
  - NOTE: C, unlike FORTRAN, stores 2-D array data by column, not by row, so this is a *bad* example for C!

  Bad Example

  - This version references `A(1,1)`, `A(1,2)`, ..., which are stored 1,000 elements apart. If cache < 4 KB (assuming `A` is a single-precision floating-point array), will cause memory misses:

```
DO I = 1, 1000
 DO J = 1, 1000
  A(I,J) = 0
 ENDDO
ENDDO
```

I/O Devices

- Transfer information between internal components and external world, e.g., tape drives, disks, monitors, etc.
- Performance measured by "bandwidth": volume of data per unit time that can be moved into and out of main memory (e.g., bits per second, or bps).

Communication Channels

- Connect internal components.
- Often referred to as a "bus" if just a single channel.
- More complex architectures use "switches."
  - Let any component communicate directly with any other component, but may get "blocking" or "collisions."