Lecture 9

Three Components of CPU Performance

\[ \text{CPU time}_{ij} = \text{Instructions executed} \times \text{Clock cycle time} \]

Cycles Per Instruction

Instructions Executed

- Last time we saw a MIPS single-cycle datapath and control unit.
- Today, we'll explore factors that contribute to a processor's execution time, and specifically its performance using pipelining.

- We are not interested in the static instruction count, or how many lines of code are in a program.
- Instead, we care about the dynamic instruction count, or how many instructions are actually executed when the program runs.
- There are three lines of code below, but the number of instructions executed would be...

<table>
<thead>
<tr>
<th>Instruction</th>
<th>IP</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add</code></td>
<td>1</td>
<td>1,000</td>
</tr>
<tr>
<td><code>bne</code></td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Ostrich</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPI

- The average number of clock cycles per instruction, or CPI, is a function of the machine and program.
- The CPI depends on the actual instruction appearing in the program—a floating-point intensive application might have a higher CPI than an integer-based program.
- It also depends on the CPU implementation. For example, a Pentium can execute the same instructions as an older 80486, but faster.
- So far we assumed each instruction took one cycle, so we had CPI = 1.
- The CPI can be < 1 due to memory stalls and slow instructions.
- The CPI can be > 1 on machines that execute more than 1 instruction per cycle (supercycles).

Clock cycle time

- One "cycle" is the minimum time it takes the CPU to do any work.
- The clock cycle time or clock period is just the length of a cycle.
- The clock rate, or frequency, is the reciprocal of the cycle time.
- Generally, a higher frequency is better.
- Some examples illustrate some typical frequencies.
- A 500MHz processor has a cycle time of 2ns.
- A 2GHz (2000MHz) CPU has a cycle time of just 0.5ns (500ps).
Execution time, again

- The easiest way to remember this is match up the units:
  - Seconds = Instructions / Clock cycles
  - Instructions = Program
  - Clock cycle = CPI
- Make things faster by making any component smaller!
- Often easy to reduce one component by increasing another

<table>
<thead>
<tr>
<th>Program</th>
<th>Compiler</th>
<th>ISA</th>
<th>Organization</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Executed</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CPI</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Clock Cycle Time</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Example 1: ISA-compatible processors
- Let’s compare the performances two x86-based processors.
  - An IBM/Intel AMD-Duron, with a CPI of 1.4, for an MP3 compressor.
  - A Intel Pentium III with a CPI of 1.5 for the same program.
- Compatible processors implement identical instruction sets and will use the same executable files, with the same number of instructions.
- But they implement the ISA differently, which leads to different CPIs.

\[
\text{CPU time}_{\text{Duron}} = \frac{\text{Instructions}}{\text{CPI}_{\text{Duron}}} \times \text{Cycle time}_{\text{Duron}}
\]

\[
\text{CPU time}_{\text{Pentium III}} = \frac{\text{Instructions}}{\text{CPI}_{\text{Pentium III}}} \times \text{Cycle time}_{\text{Pentium III}}
\]

Example 2: Comparing across ISAs
- Intel’s Itanium (IA-64) ISA is designed facilitate executing multiple instructions per cycle. If an Itanium processor achieves an average CPI of 3.1 (3) instructions per cycle), how much faster is it than a Pentium 4 (which uses the x86 ISA) with an average CPI of 1.7?
  a) Itanium is three times faster
  b) Itanium is one third as fast
  c) Not enough information

The single-cycle design from last time

The example add from last time
- Consider the instruction add $s4, $s1, $s2.

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 01001 | 01010 | 10100 | 00000 | 00000 |

- Assume $s1 and $s2 initially contain 1 and 2 respectively.
- Executing this instruction involves several steps:
  1. The instruction word is read from the instruction memory, and the program counter is incremented by 4.
  2. The sources $s1 and $s2 are read from the register file.
  3. The values 1 and 2 are added by the ALU.
  4. The result 3 is stored back into $s4 in the register file.

How the add goes through the datapath

April 19, 2010
Performance 9

April 19, 2010
Multiprocessor datapath 11
Performance of Single-cycle Design

CPU time, \( t \) = Instructions executed / CPU cycle time.

The datapath and the clock

1. On a positive clock edge, the PC is updated with a new address.
2. A new instruction can then be loaded from memory. The control unit sets the datapath signals appropriately so that:
   - Registers are read.
   - ALU output is generated.
   - Data memory is read or written, and
   - Branch target addresses are computed.
3. Several things happen on the next positive clock edge:
   - The register file is updated for arithmetic or logic instructions.
   - Data memory is written for a sw instruction.
   - The PC is updated to point to the next instruction.

   In a single-cycle datapath, everything in Step 2 must complete within one clock cycle, before the next positive clock edge.

How long is that clock cycle?

The slowest instruction...

- If all instructions must complete within one clock cycle, then the cycle time has to be large enough to accommodate the slowest instruction.
- For example, (l=50, l=65) is the slowest instruction needing \( \mu s \).
- Assuming the circuit latencies below.

The slowest instruction...

- If all instructions must complete within one clock cycle, then the cycle time has to be large enough to accommodate the slowest instruction.
- For example, (l=50, l=65) needs \( \mu s \), assuming the delays shown here.

- Reading the instruction memory
- Reading the base register \( \mu s \)
- Computing memory address \( \mu s \)
- Reading the data memory
- Storing data back to \( \mu s \)

Edge-triggered state elements

- In an instruction like add $t1, $t1, $t2, how do we know $t1 is not updated until after its original value is read?
- We'll assume that our state elements are positive edge triggered, and are updated only on the positive edge of a clock signal.
- The register file and data memory have explicit write control signals, RegWrite and MemWrite. These units can be written to only if the control signal is asserted and there is a positive clock edge.
- In a single-cycle machine the PC is updated on each clock cycle, so we don't bother to give it an explicit write control signal.
...determines the clock cycle time

- If we make the cycle time 8ns then every instruction will take 8ns, even if they don’t need that much time.
- For example, the instruction `add $a4, $a1, $a2` really needs just 6ns.

![Diagram showing the cycle time of an instruction](image)

It gets worse...

- We’ve made very optimistic assumptions about memory latency:
  - Load memory accesses on modern machines is ~50ns.
  - For comparison, an ALU on an AMD Opteron takes ~0.3ns.
- Our worst case cycle (loads/stores) includes 2 memory accesses:
  - A modern single cycle implementation would be stuck at ~180Hz.
  - Caches will improve common case access time, not worst case.
- Tying frequency to worst case path violates first law of performance!
  - “Make the common case fast!” (we’ll revisit this often)

How bad is this?

- With these same component delays, a `sw` instruction would need 7ns, and `beq` would need just 5ns.
- Let’s consider the gsc instruction mix from p. 189 of the textbook.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>48%</td>
</tr>
<tr>
<td>Loads</td>
<td>22%</td>
</tr>
<tr>
<td>Stores</td>
<td>11%</td>
</tr>
<tr>
<td>Branches</td>
<td>19%</td>
</tr>
</tbody>
</table>

- With a single-cycle datapath, each instruction would require 8ns.
- But if we could execute instructions as fast as possible, the average time per instruction for gsc would be:
  \[(48\% \times 6\text{ns}) + (22\% \times 8\text{ns}) + (11\% \times 7\text{ns}) + (19\% \times 5\text{ns}) = 6.26\text{ns}\]
- The single-cycle datapath is about 1.26 times slower!

Summary

- Performance is one of the most important criteria in judging systems.
  - Here we’ll focus on Execution time.
- Our main performance equation explains how performance depends on several factors related to both hardware and software.
  \[\text{CPU time}_{\text{CPU}} = \text{instructions executed, } \text{CPI}_{\text{CPU}} \times \text{clock cycle time},\]
- It can be hard to measure these factors in real life, but this is a useful guide for comparing systems and designs.
  - A single-cycle CPU has two main disadvantages.
    - The cycle time is limited by the worst case latency.
    - It isn’t efficiently using its hardware.
- Next time, we’ll see how this can be rectified with pipelining.