Evaluating Microprocessor Performance

Dinesh Sharma

EE Department
IIT Bombay, Mumbai

March 23, 2021

Desiderata

What do we need from a ‘good’ microprocessor?

- Speed
- Low Cost
- Low Power

Can we optimize all of these simultaneously?

Actual application determines which of these qualities is most important.

Cost and power are relatively easy to measure.

How do we characterize the speed of a processor?

Measuring Performance

For a given program, a processor which completes its execution in shorter time is obviously faster.

We can define a speed up factor when comparing two processors:
We can say that the processor 1 is faster than processor 2 by a factor p where

$$\text{Speed up factor } p = \frac{\text{Prog/Time1}}{\text{Prog/Time2}} = \frac{\text{Time2}}{\text{Time1}}$$

- Will this factor be the same for all programs?
- Perhaps not!
  Processor 1 may be more efficient for some instructions, while processor 2 may be more efficient for some other instructions.
  Depending on how may of either type of instructions are there in the program, the speed up will vary.

How do we measure execution time anyway?

- One way would be to measure the actual elapsed time from start to end of a program. This is also known as real time or wall time (from wall clocks).
- However, total time would include time spent in disc access, Human I-O etc. which do not depend on processor performance.
- We can also measure CPU time – this includes only the time spent by the processor while running the program.
- Most processors run at a constant clock rate. Then the CPU time will be given by No. of clock cycles $\times$ Time per clock cycle.
- The number of clock cycles will in turn depend on how many instructions are executed while running the program.
- Notice that this is different from the number of instructions in the program code – since some instructions may be run repeatedly in loops.

CPU time

If a program executes n instructions to complete its task, the CPU time will be given by:

$$\text{CPU time} = n \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Clock cycle}}$$

$$= n \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \text{Clock Period T}$$

The factors in the above expression are not constant, nor are they independent of each other.

Different instructions may require a different number of machine cycles. Some sort of average value has to be assumed over the n instructions.

For best performance, each of the above factors should be minimized.

CPU time

$$\text{CPU time} = \text{Instructions} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \text{Clock Period T}$$

- The number of instructions which need to be executed in order to run a program depends on the efficiency of the instruction set.
- Average number of clock cycles per instruction depends on the simplicity of instructions and on efficacy of hardware implementation of the digital circuits used for decoding and executing the instructions.
- The clock period T depends on technology used to implement the digital circuits, as also on what we are trying to do within a clock cycle. Thus, it depends on the VLSI process, as well as the style of instructions.

Minimization of Clock Period

Perhaps the easiest way to minimize CPU time is to minimize the clock period T.

- As processing technologies are scaled, the delay of logic gates gets scaled down. This enables the use of shorter clock periods.
- From early days of microprocessors, clock periods have been scaled down from generation to generation of processors.
- Early processors (4004, 8008 etc.) operated at clock frequencies of a few hundreds of KHz. IBM PC was introduced in 1981 using an Intel 8088 processor operating at a clock frequency of 4.77 MHz.
- Current processors typically operate with clocks of multiple GHZ. These speed were reached around 2005 and represent a speed up of several thousands over a period of about thirty years.

Saturation of Clock Frequencies

- Once clock speeds of about 3-5 GHz were reached around 2005, it became very difficult to raise the clock frequency any further due to clock jitter and interconnect delays.
- Clock periods have remained static at around 3-5 GHz for the last 15 years or so.
- Once this easy method of speeding up processors became impractical, designers of processor architecture have looked for other methods of improving processor performance.

Minimization of instructions per program

- In parallel with increases in clock frequency, data width was continually increased in early processors.
- A 32 bit number could be added in one instruction by a 32 bit processor whereas 8 bit processors would require 4 additions for the same task.
- Increasing data width reduced the number of instructions required per program. However, the number of instruction per program will reduce only if wide additions are actually needed by the program.
- 64 bit precision is considered quite adequate for most computations and increasing data width beyond this brings diminishing returns.
- Taking Intel processors as an example, the 8 bit 8085 was replaced by the 16 bit 8086, the 32 bit 80386 and the current 64 bit Pentium family processors.
- Around the same time as the saturation of clock frequencies, this straightforward method of increasing performance has reached saturation.

Choice of Instruction Set

$$\text{CPU time} = \text{Instructions} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \text{Clock Period T}$$

- Minimization of the number of instructions suggests instructions which can get a “lot done through a single instruction”.
- Examples are looping instructions like DJNZ or CJNE for 8051 and string instructions of 8086 which can do repetitive tasks through single instructions.
- However, implementation of these complex instructions requires extensive hardware, increases the number of clock cycles required per instruction and the complexity of digital circuits makes the decoding and execution slower.
- Initial attempts at optimizing processor performance used such complex instructions and these processors are known as “Complex Instruction Set Computers” or CISC.

Amdhal's Law

- Reduction of clock period and increasing data width were the early and obvious attempts to improve processor performance.
- This was because a faster clock speed made all instructions faster.
- Once the performance improvement due to these techniques were saturated, designers had to look for other methods of making processors faster.
- Other modifications to the processor architecture improved the performance of only a fraction of the instructions. For example, most of the instructions of the complex instruction set processors were the same as that of earlier processors and these would run no faster.
- The improvement in overall performance brought about by steps which make only a few instructions faster (for example, through parallel execution) is given by Amdhal's law.

Amdhal's Law

$$\text{CPU time} = \text{Instructions} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \text{Clock Period T}$$

Let f be the fraction of instructions which are **not** improved by an architectural modification.

Then $(1 - f)$ is the fraction for which clock cycles/instruction are reduced by some factor. Let its average value be p. Execution time will now be given by

$$\begin{aligned}
& \text{Instructions} \times f \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \text{Clock Period T} \\
+ \quad & \text{Instructions} \times \frac{(1 - f)}{p} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \text{Clock Period T} \\
= \quad & \text{Instructions} \times (f + (1 - f)/p) \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \text{Clock Period T}
\end{aligned}$$

Amdhal's Law

The time taken when a fraction $(1 - f)$ of all executed instructions has been speeded up by the factor p is given by:

$$\text{Instructions} \times (f + (1 - f)/p) \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \text{Clock Period T}$$

and so the performance improvement is given by:

$$\frac{1}{f + (1 - f)/p}$$

This is called Amdhal's law. Notice that f is the fraction of instructions which have **not** become faster.

- When $f = 1$, (no instructions are improved), performance gain is just 1 as expected.
- For $f = 0$, (all instructions are improved by the factor p), the overall improvement is by a factor p. This also makes sense.

Amdhal's Law

Performance improvement when a fraction $(1 - f)$ of all executed instructions has been speeded up by the factor p is given by:

$$\frac{1}{f + (1 - f)/p}$$

- For large p, the asymptotic value of this improvement is $1/f$. Thus, performance gains are limited by the fraction of instruction which are **not** improved.
- In Qualitative terms, what Amdhal's law tells us is that performance gain is limited by unchanged instructions.
- If a fraction f of all executed instructions sees no change in execution time, no amount of improvement for the remaining $1 - f$ fraction will produce a speedup greater than $1/f$.

Performance improvement when a fraction $(1 - f)$ of all executed instructions has been speeded up by the factor p is given by:

$$\frac{1}{f + (1 - f)/p}$$

- As the improvement factor p increases, the improved instructions account for a smaller and smaller fraction of total execution time,
- Therefore increasing p gives diminishing returns after a point.
- In fact, if improving a subset of instructions makes a larger collection of instructions slower, it might actually make the processor slower.

These considerations led to a ‘rethink’ about the efficacy of complex instruction set architectures.

CISC versus RISC

- While complex instructions do reduce the number of instructions per task, implementation of these instructions requires much more hardware and leads to slower decoders and multiplexers.
- This has an adverse effect on the average number of clock cycles required per instruction for all instructions.
- Work at Stanford and UC Berkeley in 1980s suggested that in fact, performance of a processor can be improved by far if we use simple and regular instruction sets.
- While it does increase the number of instructions required to implement the program, that is more than offset by the speed gain provided by fewer clock cycles per instruction of the simpler instruction set architecture.
- Processors of this kind are called Reduced Instruction Set Computers or RISC.

CISC versus RISC

- The Stanford suggestions for a RISC architecture resulted in the design of MIPS processor.
- The UC Berkeley design resulted in the development of SPARC architecture, which is another prominent RISC architecture.
- Commercially, perhaps the most successful RISC architecture is the one used in the ARM microprocessor. ARM is, however, not a pure RISC design and includes a few CISC instructions.
- The silicon real estate freed by discontinuation of complex instructions is used for a much larger set of registers, so that memory accesses are minimized.

RISC architecture

- Although by itself, the RISC architecture implies just a reduced and regular instruction set, which frees up silicon real estate for a larger number of registers and other useful hardware units like barrel shifters, most RISC processors have adopted other architectural features as well to improve performance.
- Most RISC processors are pipelined. In a pipelined architecture, the hardware works on several instructions at the same time. While one instruction is being fetched, the previous one is being decoded and the one before that is begin executed. This improves throughput.
- Most RISC processors also use a load-store architecture. This means that only the load and store instructions access the memory. All other instructions operate only on registers. If you need to operate on a memory operand, it should first be loaded into a register and then the operation is carried out.