4. Measuring Performance

Before we start to examine better processor designs we need to be able to evaluate their relative **performance** characteristics, i.e., understand what “better” means. The problem is that performance can be a difficult concept to define exactly. It is often more instructive to think about the overall **quality** of a design rather than performance. Quality can combine lots of different features as dictated by different application or market areas; a **cost function** will determine how quality is evaluated by combining many **metrics**. For example, one might worry about how quickly a processor executes a given program, or how much power is used during execution. In most designs, such metrics compete for resources in the sense that no one can be satisfied without being detrimental to another; some form of trade-off or compromise must be reached, one can think of this as maximizing the cost function in some way.

For a general circuit, perhaps the most simple metric one can consider is how quickly it operates. This can be further decomposed into two standard measures.

Definition 1. The **latency** of a circuit is the total time elapsed before a given input is operated on to produce an output. The **bandwidth** or **throughput** of a circuit is the rate at which new inputs can be offered (or outputs produced).

The difference is somewhat subtle but we will see why *both* are important when we look at improving performance: for now, simply consider latency. From the basic definition, one can try to capture what is meant by performance.

Definition 2. The **performance** of design *X* is inversely proportional to latency

$$\text{PERFORMANCE}(X) = \frac{1}{\text{LATENCY}(X)}.$$

From this, we can define what is meant by the statement “design *X* is *n* times faster than design *Y*”. That is, if *n* is the speed-up offered by design *X* over design *Y*, then

$$\text{SPEED} - \text{UP}(X \text{ over } Y) = \frac{\text{LATENCY}(Y)}{\text{LATENCY}(X)}.$$

$$= \frac{\text{PERFORMANCE}(X)}{\text{PERFORMANCE}(Y)}.$$

$$= n$$

In the context of a processor rather than a general circuit, one typically uses **execution time** in place of latency; that is, the time taken to execute a given program.

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$

This means that for two computers X and Y, if the performance of X is greater than the performance of Y, we have

$$\text{Performance}_X > \text{Performance}_Y$$

$$\frac{1}{\text{Execution time}_X} > \frac{1}{\text{Execution time}_Y}$$

$$\text{Execution time}_Y > \text{Execution time}_X$$

That is, the execution time on Y is longer than that on X, if X is faster than Y.

In discussing a computer design, we often want to relate the performance of two different computers quantitatively. We will use the phrase "X is *n* times faster than Y"—or equivalently "X is *n* times as fast as Y"—to mean

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = n$$

If X is *n* times faster than Y, then the execution time on Y is *n* times longer than it is on X:

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

We know that A is n times faster than B if

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus the performance ratio is

$$\frac{15}{10} = 1.5$$

and A is therefore 1.5 times faster than B.

Estimating Execution Time

There are plenty of bad ways to evaluate the performance of a processor that are none-the-less still used in marketing literature. A prime example is the **Millions of Instructions Per Second (MIPS)** metric. Essentially the MIPS metric measures how many instructions a processor can execute each second: the assertion is that the more instructions executed each second, the higher the performance. One might therefore try to estimate the execution time of program *X* on processor *Y* using

$$\text{EXECUTION} - \text{TIME}(X,Y) = \frac{\text{INSTRUCTION} - \text{COUNT}(X)}{\text{MIPS} - \text{RATING}(Y)}$$

This is an intuitive, easy to understand measure with which one can compare the performance of processors. In fact, in the 1970s the performance of computers was quoted relative to a baseline computer; the VAX 11/780 was said to represent

1 VAX MIPS, the VAX executed one million instructions per-second. The problem is, this method of comparison is quite flawed. In particular, it should be clear that a RISC processor will typically take many instruction to execute a given program since each instruction performs a relatively simple operation; a CISC processor will typically take less instructions since each instruction performs more complex operations. Comparing one with the other is nonsense: we need a performance metric that includes the idea of amount of useful work done. That is, a RISC processor will almost always have a higher MIPS rating than a CISC alternative; regardless of the design quality. In addition, the MIPS rating is tied to the clock speed of the physical processor. In theory at least, it would be nice to evaluate the quality of a design independently from the clock speed.

The **Cycles Per-Instruction (CPI)** metric hopes to solve this problem to some extent. It measures the average number of clock cycles needed per-instruction and varies according to the processor architecture and the program being executed. The **instruction mix** for a processor tells us what ratio of different instruction types are used in the average program. For example, we might note that 40% of instructions perform arithmetic, 40% perform memory access and 20% perform branches. The process of discovering the instruction mix for a processor is called **workload characterization**; we typically use a range of benchmark programs to produce an overall picture of how often different instructions are used.

Given the instruction mix and the CPI for each instruction type, we can estimate the overall CPI for the processor. The CPI for each instruction is easily identified; typically they are quoted as part of the processor manual. Let CPI *i* denote the CPI for instruction type *i* and F*i* denote how often instruction type *i* is executed. The overall CPI is then calculated as :

$$\mathrm{CPI} = \sum_{i=0}^{n-1} F_i \, . \, CPI_i$$

Consider an example instruction mix:

Operation	F_i	CPI_i	$F_i \cdot CPI_i$
Arithmetic	40%	1	0.4
Memory	40%	4	1.6
Branch	20%	2	0.4
Total	100%		2.4

Using the overall CPI, we can then estimate how long a given program will take to execute. Firstly we can estimate the number of cycles the program will take to execute; this requires we know the number of instructions executed by the program and the overall CPI for the processor as computed above. Then, for program *X* on processor *Y* we have

CYCLES(*X*,*Y*) = INSTRUCTION-COUNT(*X*) · CPI(*Y*).

Finally we can estimate the execution time by dividing the number of cycles by the clock rate:

$$\text{EXECUTION-TIME}(X,Y) = \frac{\text{CYCLES}(X)}{\text{CLOCK-RATE}(Y)}$$

$$= \frac{\text{INSTRUCTION-COUNT}(X) \cdot \text{CPI}(Y)}{\text{CLOCK-RATE}(Y)}$$

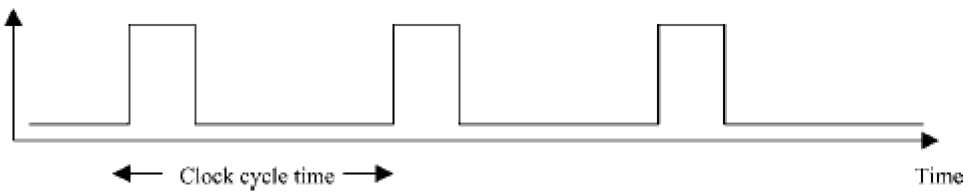
Another measure is **a <u>Million floating-point instructions per second (MFLOP)</u>**: (rate of floating-point instruction execution per unit time) has also been used as a measure for machines' performance. It is defined as:

$$MFLOPS = \frac{\text{Number of floating-point operations in a program}}{\text{Execution time} \times 10^6}$$

While MIPS measures the rate of average instructions, MFLOPS is only defined for the subset of floating-point instructions. An argument against MFLOPS is the fact that the set of floating-point operations may not be consistent across machines and therefore the actual floating-point operations will vary from machine to machine. Yet another argument is the fact that the performance of a machine for a given program as measured by MFLOPS cannot be generalized to provide a single performance metric for that machine.

In another references, we note the previous performance metrics (CPI & MIPS) as follows (as the same ideas):

- **CPI (*C*lock cycles *P*er *I*nstruction) :** Now we need to define the ***clock cycle time*** as the time between two consecutive rising edges of a periodic clock signal :



The time required to execute a job by a computer is often expressed in terms of **clock cycles**.

Also , We denote the number of CPU clock cycles for executing a job to be the ***cycle count*** (**CC**), the ***cycle time*** by **CT**, and the clock frequency by **f** = **1/CT**. The time taken by the CPU to execute a job can be expressed as:

$$CPU\ time = CC \times CT = CC/f$$

<u>It may be easier to count the number of instructions executed in a given program as compared to counting the number of CPU clock cycles needed for executing that program</u>. Therefore, the average number of **<u>*C*</u>lock cycles <u>*P*</u>er <u>*I*</u>nstruction (CPI)** has been used as an alternate performance measure. The following equation shows how to compute the CPI:

$$CPI = \frac{CPU\ clock\ cycles\ for\ the\ program}{Instruction\ count}$$

$$CPU\ time = Instruction\ count \times CPI \times Clock\ cycle\ time$$

$$= \frac{Instruction\ count \times CPI}{Clock\ rate}$$

It is known that the instruction set of a given machine consists of a number of instruction categories: ALU (simple assignment and arithmetic and logic instructions), load, store, branch, and so on. In the case that the CPI for each instruction category is known, the overall CPI can be computed as :

$$CPI = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count}$$

where Ii is the number of times an instruction of type i is executed in the program and CPI_i is the average number of clock cycles needed to execute such instruction.

Ex1: Consider computing the overall CPI for a machine A for which the following performance measures were recorded when executing a set of benchmark programs. Assume that the clock rate of the CPU is 200 MHz.

Instruction category	Percentage of occurrence	No. of cycles per instruction
ALU	38	1
Load & store	15	3
Branch	42	4
Others	5	5

Assuming the execution of 100 instructions, the overall CPI can be computed as :

$$CPI_a = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{38 \times 1 + 15 \times 3 + 42 \times 4 + 5 \times 5}{100} = 2.76$$

MIPS (million instructions-per-second): It is (the rate of instruction execution per unit time)), which is defined as:

$$MIPS = \frac{Instruction\ count}{Execution\ time \times 10^6} = \frac{Clock\ rate}{CPI \times 10^6}$$

Ex2 : Suppose that the same set of benchmark programs considered above were executed on another machine, call it machine B, for which the following measures were recorded.

Instruction category	Percentage of occurrence	No. of cycles per instruction
ALU	35	1
Load & store	30	2
Branch	15	3
Others	20	5

What is the MIPS rating for the machine considered in the previous example (machine A) and machine B assuming a clock rate of 200 MHz?

$$CPI_a = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{38 \times 1 + 15 \times 3 + 42 \times 4 + 5 \times 5}{100} = 2.76$$

$$MIPS_a = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.76 \times 10^6} = 70.24$$

$$CPI_b = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{35 \times 1 + 30 \times 2 + 20 \times 5 + 15 \times 3}{100} = 2.4$$

$$MIPS_b = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.4 \times 10^6} = 83.67$$

Home Work: compute MIPS for two different machines running a given set of benchmark programs:

Instruction category	No. of instructions (in millions)	No. of cycles per instruction
Machine (A)		
ALU	8	1
Load & store	4	3
Branch	2	4
Others	4	3
Machine (B)		
ALU	10	1
Load & store	8	2
Branch	2	4
Others	4	3