IFT 307

Mr. Ibrahim Lawal

Example 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

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

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

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

Example 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?

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

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

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

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

Thus MIPSb >MIPSa.

It is interesting to note here that although MIPS has been used as a performance measure for machines, one has to be careful in using it to compare machines having different instruction sets.

This is because MIPS does not track execution time.
Consider, for example, the following measurement made on 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

$$CPI_a = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{(8 \times 1 + 4 \times 3 + 4 \times 3 + 2 \times 4) \times 10^6}{(8 + 4 + 4 + 2) \times 10^6} \cong 2.2$$

$$MIPS_a = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.2 \times 10^6} \cong 90.9$$

$$CPU_a = \frac{Instruction\ count \times CPI_a}{Clock\ rate} = \frac{18 \times 10^6 \times 2.2}{200 \times 10^6} = 0.198\,\text{s}$$

$$CPI_b = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{(10 \times 1 + 8 \times 2 + 4 \times 4 + 2 \times 4) \times 10^6}{(10 + 8 + 4 + 2) \times 10^6} = 2.1$$

$$MIPS_b = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.1 \times 10^6} = 95.2$$

$$CPU_b = \frac{Instruction\ count \times CPI_a}{Clock\ rate} = \frac{20 \times 10^6 \times 2.1}{200 \times 10^6} = 0.21\,\text{s}$$

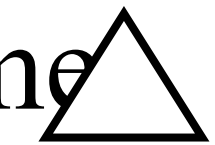
$$MIPS_b > MIPS_a \qquad \text{and} \qquad CPU_b > CPU_a$$

- The example above shows that although machine B has a higher MIPS compared to machine A, it requires longer CPU time to execute the same set of benchmark programs.

Amdahl’s law for speedup (SUo) due to enhancement. In this case, we consider speedup as a measure of how a machine performs after some enhancement relative to its original performance. The following relationship formulates Amdahl’s law.

$$SU_o = \frac{Performance\ after\ enhancement}{Performance\ before\ enhancement}$$

$$Speedup = \frac{Execution\ time\ before\ enhancement}{Execution\ time\ after\ enhancement}$$

In its given form, Amdahl's law accounts for cases whereby improvement can be applied to the instruction execution time. However, sometimes it may be possible to achieve performance enhancement for only a fraction of time  . In this case a

In this case a new formula has to be developed in order to relate the speedup, SU $\triangle$ due to an enhancement for a fraction of time $\triangle$ to the speedup due to an overall enhancement, SUo .

This relationship can be expressed as

$$SU_o = \frac{1}{(1 - \Delta) + (\Delta / SU_\Delta)}$$

- It should be noted that when $\triangle = 1$, that is, when enhancement is possible at all times, then $SUo = SU\triangle$, as expected. Consider, for example, a machine for which a speedup of 30 is possible after applying an enhancement.

If under certain conditions the enhancement was only possible for 30% of the time, what is the speedup due to this partial application of the enhancement?

$$SU_o = \frac{1}{(1-\Delta) + (\Delta/SU_\Delta)} = \frac{1}{(1-0.3) + \frac{0.3}{30}} = \frac{1}{0.7 + 0.01} = 1.4$$

It is interesting to note that the above formula can be generalized as shown below to account for the case whereby a number of different independent enhancements can

be applied separately and for different fractions of the time, $\triangle 1$, $\triangle 2$, . . . , $\triangle n$, thus leading respectively to the speedup enhancements $SU\triangle 1$, $SU\triangle 2$, . . . , $SU\triangle n$.

$$SU_{\mathrm{o}} = \frac{1}{[1 - (\Delta_1 + \Delta_2 + \cdots + \Delta_n)] + \dfrac{(\Delta_1 + \Delta_2 + \cdots + \Delta_n)}{(SU_{\Delta_1} + SU_{\Delta_2} + \cdots + SU_{\Delta_n})}}$$

$$\text{Speedup} = \frac{\text{Performance after enhancement}}{\text{Performance before enhancement}} = \frac{\text{Execution time before enhancement}}{\text{Execution time after enhancement}}$$

Suppose that a feature of the system is used during execution a fraction of the time f, before enhancement, and that the speedup of that feature after enhancement is SU_f. Then the overall speedup of the system is

$$\text{Speedup} = \frac{1}{(1 - f) + \frac{f}{SU_f}}$$

Example

A common transformation required in graphics processors is square root. Implementations of floating-point (FP) square root vary significantly in performance, especially among processors designed for graphics.

Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal is to enhance the FPSQR hardware and speed up this operation by a factor of 10.

The other alternative is just to try to make all FP instructions in the graphics processor run faster by a factor of 1.6; FP instructions are responsible for half of the execution time for the application.

The design team believes that they can make all FP instructions run 1.6 times faster with the same effort as required for the fast square root.

Compare these two design alternatives.

We can compare these two alternatives by comparing the speedups:

$$\text{Speedup}_{\text{FPSQR}} = \frac{1}{(1-0.2)+\frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$\text{Speedup}_{\text{FP}} = \frac{1}{(1-0.5)+\frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$

Improving the performance of the FP operations overall is slightly better because of the higher frequency.

Instruction Set Architecture and Design

- The (main) memory can be modeled as an array of millions of adjacent cells, each capable of storing a binary digit (bit), having value of 1 or 0. These cells are organized in the form of groups of fixed number, say n, of cells that can be dealt with as an atomic entity.

- An entity consisting of 8 bits is called a byte.
- In many systems, the entity consisting of n bits that can be stored and retrieved in and out of the memory using one basic memory operation is called a **word** (the smallest addressable entity).
- Typical size of a word ranges from 16 to 64 bits.

- In order to be able to move a word in and out of the memory, a distinct address has to be assigned to each word.
- This address will be used to determine the location in the memory in which a given word is to be stored.
- This is called a memory **write operation.**

- Similarly, the address will be used to determine the memory location from which a word is to be retrieved from the memory.
- This is called a memory **read operation.**

- For example, the size of a typical memory of a personal computer is 256 Mbytes, that is,

- $256 \times 2^{20} = 2^{28}$ bytes.

- The number of bits, L, needed to distinctly address M words in a memory is given by $L = \log_2 M$.
- For example, if the size of the memory is 64 M (read as 64 mega words), then the number of bits in the address is $\log_2 (64 * 2^{20}) = \log_2 (2^{26}) = 26$ bits.

Alternatively, if the number of bits in the address is L, then the maximum memory size (in terms of the number of words that can be addressed using these L bits) is $M = 2^L$.

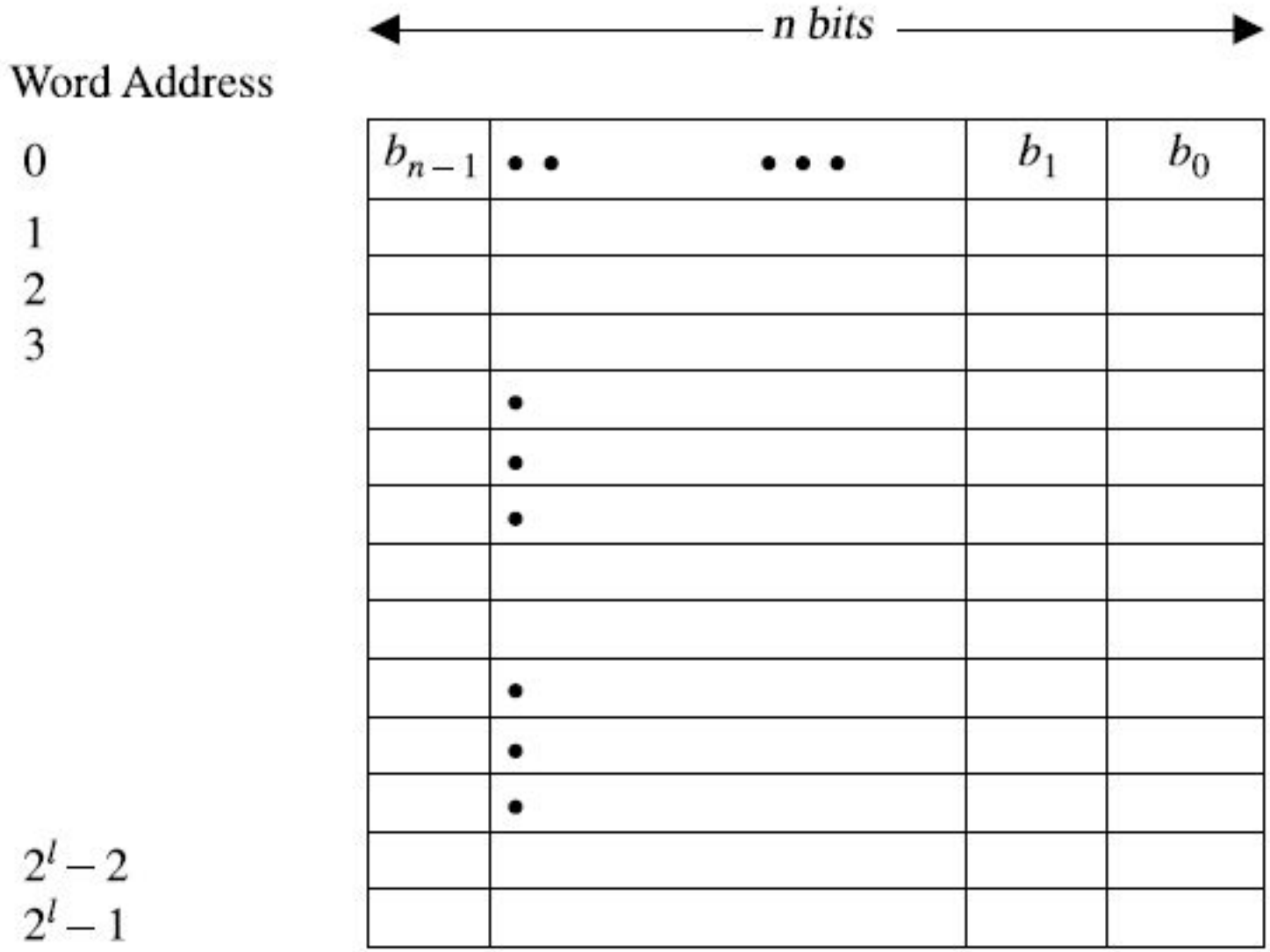


Figure 2.1 Illustration of the main memory addressing

Three basic steps are needed in order for the CPU to perform a write operation into a specified memory location:

1. The word to be stored into the memory location is first loaded by the CPU into a specified register, called the memory data register (MDR).

2. The address of the location into which the word is to be stored is loaded by the CPU into a specified register, called the memory address register (MAR).
3. A signal, called write, is issued by the CPU indicating that the word stored in the MDR is to be stored in the memory location whose address in loaded in the MAR.

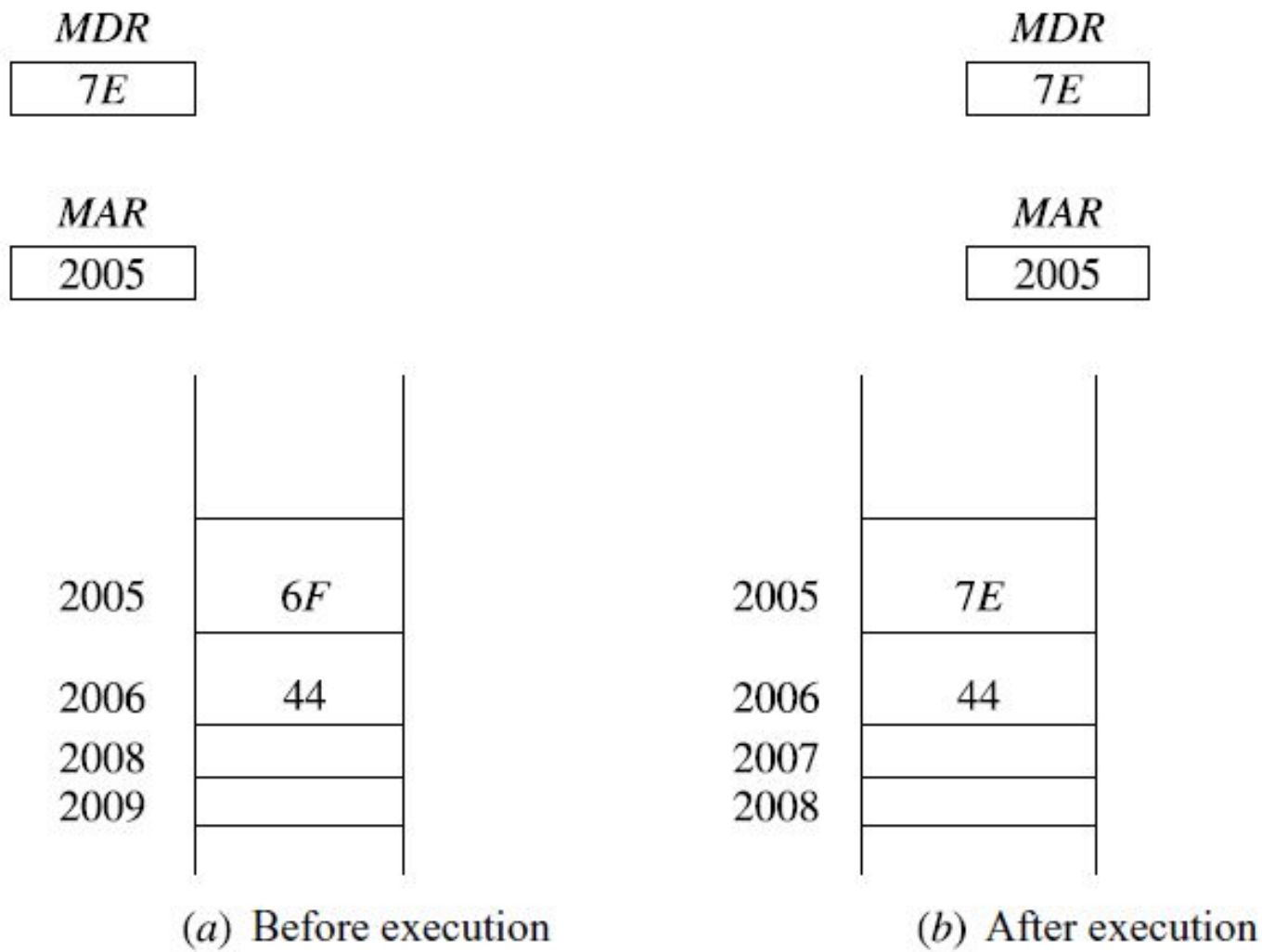


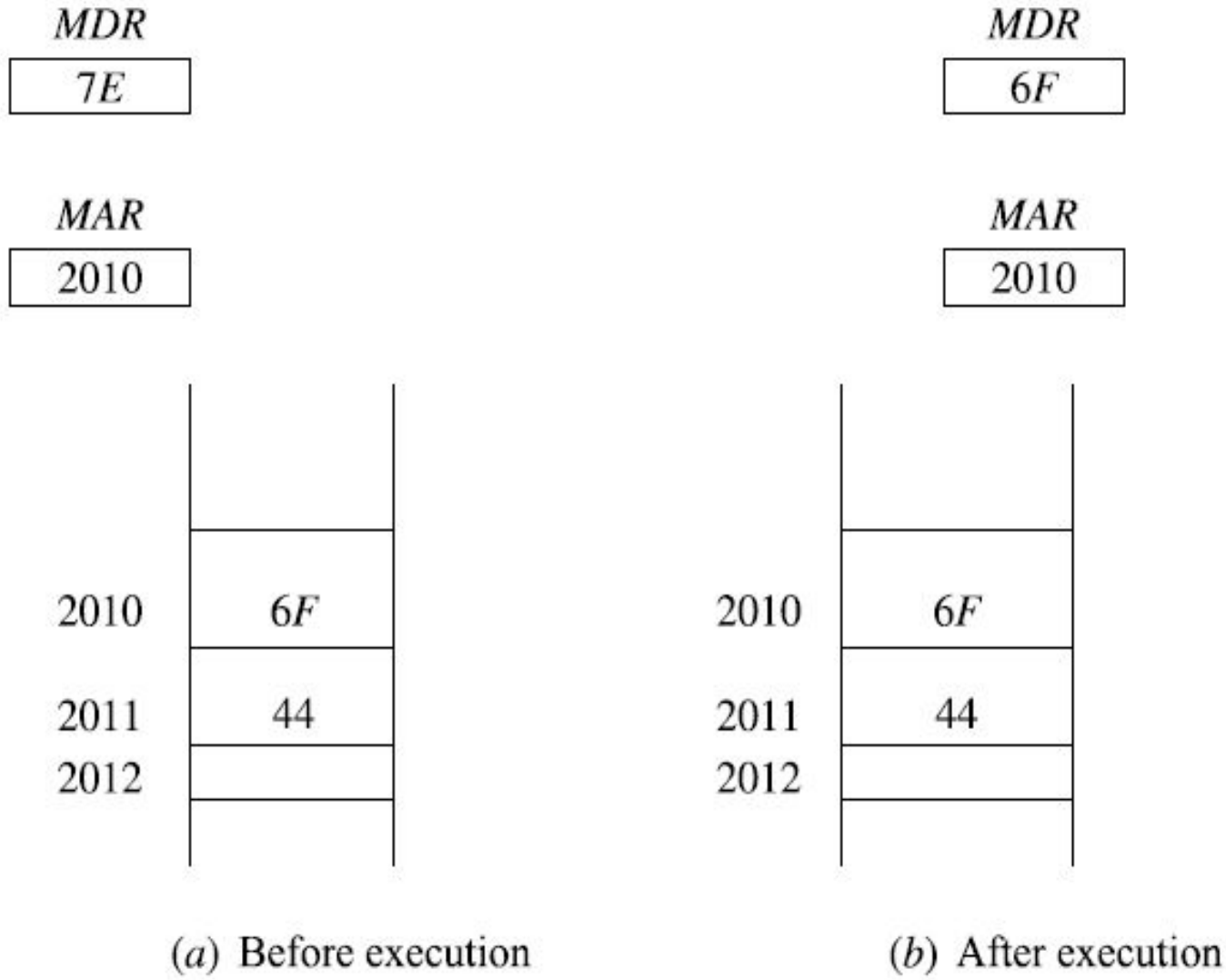
Illustration of the memory write operation

Similar to the write operation, three basic steps are needed in order to perform a memory read operation:

1. The address of the location from which the word is to be read is loaded into the MAR.

2. A signal, called read, is issued by the CPU indicating that the word whose address is in the MAR is to be read into the MDR.

3. After some time, corresponding to the memory delay in reading the specified word, the required word will be loaded by the memory into the MDR ready for use by the CPU.



(*a*) Before execution

(*b*) After execution

Illustration of the memory read operation

Thank You