CHAPTER 1 OVERVIEW

ANSWERS TO QUESTIONS

1.1 **Computer architecture** refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program. **Computer organization** refers to the operational units and their interconnections that realize the architectural specifications. Examples of architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and techniques for addressing memory. Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer and peripherals; and the memory technology used.

1.2 Computer structure refers to the way in which the components of a computer are interrelated. Computer function refers to the operation of each individual component as part of the structure.

1.3 Data processing; data storage; data movement; and control.

1.4 **Central processing unit (CPU):** Controls the operation of the computer and performs its data processing functions; often simply referred to as processor.
Main memory: Stores data.
I/O: Moves data between the computer and its external environment.
System interconnection: Some mechanism that provides for communication among CPU, main memory, and I/O. A common example of system interconnection is by means of a system bus, consisting of a number of conducting wires to which all the other components attach.

1.5 **Control unit:** Controls the operation of the CPU and hence the computer
Arithmetic and logic unit (ALU): Performs the computer's data processing functions
Registers: Provides storage internal to the CPU

CPU interconnection: Some mechanism that provides for communication among the control unit, ALU, and registers

Chapter 2 Computer Evolution and Performance

Answers to Questions

2.1 In a stored program computer, programs are represented in a form suitable for storing in memory alongside the data. The computer gets its instructions by reading them from memory, and a program can be set or altered by setting the values of a portion of memory.

2.2 A **main memory**, which stores both data and instructions: an **arithmetic and logic unit** (ALU) capable of operating on binary data; a **control unit**, which interprets the instructions in memory and causes them to be executed; and **input and output (I/O) equipment** operated by the control unit.

2.3 Gates, memory cells, and interconnections among gates and memory cells.

2.4 Moore observed that the number of transistors that could be put on a single chip was doubling every year and correctly predicted that this pace would continue into the near future.

2.5 **Similar or identical instruction set:** In many cases, the same set of machine instructions is supported on all members of the family. Thus, a program that executes on one machine will also execute on any other. **Similar or identical operating system:** The same basic operating system is available for all family members. **Increasing speed:** The rate of instruction execution increases in going from lower to higher family members. **Increasing Number of I/O ports:** In going from lower to higher family members. **Increasing memory size:** In going from lower to higher family members. **Increasing cost:** In going from lower to higher family members.

2.6 In a microprocessor, all of the components of the CPU are on a single chip.

ANSWERS TO PROBLEMS

2.1 a

Location	**Instruction/Value**	**Comments**
0	<>	Constant (N) [initialized to some value]
1	1	Constant; Integer value = 1
2	2	Constant; Integer value = 2
3	0	Variable Y (initialized to integer zero); Sum(Y)
4L	LOAD M(0	N → AC
4R	ADD M(1)	AC + 1 → AC
5L	MUL M(0)	N(N+1) → AC
5R	DIV M(2)	AC/2 → AC
6L	STOR M(3)	AC → Y; saving the Sum in variable Y
6R	JUMP M(6,20:39)	Done; HALT

b.

Location	**Instruction/Value**	**Comments**
0	<>	Constant (N) [initialized to some value]
1	1	Constant (loop counter increment)
2	1	Variable i (loop index value; current)
3	1	Variable Y = Sum of X values (Initialized to One)
4L	LOAD M(0	N → AC (the max limit)
4R	SUB M(2)	Compute N–i → AC
5L	JUMP + M(6,0:19)	Check AC > 0 ? [i < N]
5R	JUMP + M(5,20:39)	i=N; done so HALT
6L	LOAD M(2)	i<N so continue; Get loop counter i
6R	ADD M(1)	i+1 in AC
7L	STOR M(2)	AC → i
8R	ADD M(3)	i + Y in AC
8L	STOR M(3)	AC → Y
8R	JUMP M(4,0:19)	Continue at instruction located at address 4L

2.2 a.

Opcode	Operand
00000001	000000000010

b. First, the CPU must make access memory to fetch the instruction. The instruction contains the address of the data we want to load. During the execute phase accesses memory to load the data value located at that address for a total of two trips to memory.

2.3 To read a value from memory, the CPU puts the address of the value it wants into the MAR. The CPU then asserts the Read control line to memory and places the address on the address bus. Memory places the contents of the memory location passed on the data bus. This data is then transferred to the MBR. To write a value to memory, the CPU puts the address of the value it wants to write into the MAR. The CPU also places the data it wants to write into the MBR. The CPU then asserts the Write control line to memory and places the address on the address bus and the data on the data bus. Memory transfers the data on the data bus into the corresponding memory location.

2.4

Address	Contents
08A	LOAD M(0FA)
	STOR M(0FB)
08B	LOAD M(0FA)
	JUMP +M(08D)
08C	LOAD –M(0FA)
	STOR M(0FB)
08D	

This program will store the absolute value of content at memory location 0FA into memory location 0FB.

2.5 All data paths to/from MBR are 40 bits. All data paths to/from MAR are 12 bits. Paths to/from AC are 40 bits. Paths to/from MQ are 40 bits.

2.6 The purpose is to increase performance. When an address is presented to a memory module, there is some time delay before the read or write operation can be performed. While this is happening, an address can be presented to the other module. For a series of requests for successive words, the maximum rate is doubled.

2.7 The discrepancy can be explained by noting that other system components aside from clock speed make a big difference in overall system speed. In particular, memory systems and advances in I/O processing contribute to the performance ratio. A system is only as fast as its slowest link. In recent years, the bottlenecks have been the performance of memory modules and bus speed.

2.8 As noted in the answer to Problem 2.7, even though the Intel machine may have a faster clock speed (2.4 GHz vs. 1.2 GHz), that does not necessarily mean the system will perform faster. Different systems are not comparable on clock speed. Other factors such as the system components (memory, buses, architecture) and the instruction sets must also be taken into account. A more accurate measure is to run both systems on a benchmark. Benchmark programs exist for certain tasks, such as running office applications, performing floating-point operations, graphics operations, and so on. The systems can be compared to each other on how long they take to complete these tasks. According to Apple Computer, the G4 is comparable or better than a higher-clock speed Pentium on many benchmarks.

2.9 This representation is wasteful because to represent a single decimal digit from 0 through 9 we need to have ten tubes. If we could have an arbitrary number of these tubes ON at the same time, then those same tubes could be treated as binary bits. With ten bits, we can represent 2^{10} patterns, or 1024 patterns. For integers, these patterns could be used to represent the numbers from 0 through 1023.

2.10 *CPI* = 1.55; MIPS rate = 25.8; Execution time = 3.87 ms. Source: [HWAN93]

2.11 **a.**

$$CPI_A = \frac{\sum CPI_i \times I_i}{I_c} = \frac{(8\times1+4\times3+2\times4+4\times3)\times10^6}{(8+4+2+4)\times10^6} \approx 2.22$$

$$MIPS_A = \frac{f}{CPI_A \times 10^6} = \frac{200\times10^6}{2.22\times10^6} = 90$$

$$CPU_A = \frac{I_c \times CPI_A}{f} = \frac{18\times10^6\times2.2}{200\times10^6} = 0.2 \text{ s}$$

$$CPI_B = \frac{\sum CPI_i \times I_i}{I_c} = \frac{(10\times1+8\times2+2\times4+4\times3)\times10^6}{(10+8+2+4)\times10^6} \approx 1.92$$

$$MIPS_B = \frac{f}{CPI_B \times 10^6} = \frac{200\times10^6}{1.92\times10^6} = 104$$

$$CPU_B = \frac{I_c \times CPI_B}{f} = \frac{24\times10^6\times1.92}{200\times10^6} = 0.23 \text{ s}$$

b. Although machine B has a higher MIPS than machine A, it requires a longer CPU time to execute the same set of benchmark programs.

2.12 a. We can express the MIPs rate as: [(MIPS rate)/10^6] = I_c/T. So that: $I_c = T \times$ [(MIPS rate)/10^6]. The ratio of the instruction count of the RS/6000 to the VAX is [x × 18]/[12x × 1] = 1.5.

b. For the Vax, *CPI* = (5 MHz)/(1 MIPS) = 5.
For the RS/6000, *CPI* = 25/18 = 1.39.

2.13 From Equation (2.2), MIPS = $I_c/(T \times 10^6) = 100/T$. The MIPS values are:

	Computer A	Computer B	Computer C
Program 1	100	10	5
Program 2	0.1	1	5
Program 3	0.2	0.1	2
Program 4	1	0.125	1

	Arithmetic mean	Rank	Harmonic mean	Rank
Computer A	25.325	1	0.25	2
Computer B	2.8	3	0.21	3
Computer C	3.25	2	2.1	1

2.14 a. Normalized to R:

Benchmark	Processor		
	R	**M**	**Z**
E	1.00	1.71	3.11
F	1.00	1.19	1.19
H	1.00	0.43	0.49
I	1.00	1.11	0.60
K	1.00	2.10	2.09
Arithmetic mean	1.00	1.31	1.50

b. Normalized to M:

Benchmark	**Processor**		
	R	**M**	**Z**
E	0.59	1.00	1.82
F	0.84	1.00	1.00
H	2.32	1.00	1.13
I	0.90	1.00	0.54
K	0.48	1.00	1.00
Arithmetic mean	1.01	1.00	1.10

c. Recall that the larger the ratio, the higher the speed. Based on (a) R is the slowest machine, by a significant amount. Based on (b), M is the slowest machine, by a modest amount.

d. Normalized to R:

Benchmark	**Processor**		
	R	**M**	**Z**
E	1.00	1.71	3.11
F	1.00	1.19	1.19
H	1.00	0.43	0.49
I	1.00	1.11	0.60
K	1.00	2.10	2.09
Geometric mean	1.00	1.15	1.18

Normalized to M:

Benchmark	**Processor**		
	R	**M**	**Z**
E	0.59	1.00	1.82
F	0.84	1.00	1.00
H	2.32	1.00	1.13
I	0.90	1.00	0.54
K	0.48	1.00	1.00

Geometric mean	0.87	1.00	1.02

Using the geometric mean, R is the slowest no matter which machine is used for normalization.

2.15 **a.** Normalized to X:

Benchmark	**Processor**		
	X	**Y**	**Z**
1	1	2.0	0.5
2	1	0.5	2.0
Arithmetic mean	1	1.25	1.25
Geometric mean	1	1	1

Normalized to Y:

Benchmark	**Processor**		
	X	**Y**	**Z**
1	0.5	1	0.25
2	2.0	1	4.0
Arithmetic mean	1.25	1	2.125
Geometric mean	1	1	1

Machine Y is twice as fast as machine X for benchmark 1, but half as fast for benchmark 2. Similarly machine Z is half as fast as X for benchmark 1, but twice as fast for benchmark 2. Intuitively, these three machines have equivalent performance. However, if we normalize to X and compute the arithmetic mean of the speed metric, we find that Y and Z are 25% faster than X. Now, if we normalize to Y and compute the arithmetic mean of the speed metric, we find that X is 25% faster than Y and Z is more than twice as fast as Y. Clearly, the arithmetic mean is worthless in this context.

b. When the geometric mean is used, the three machines are shown to have equal performance when normalized to X, and also equal performance when normalized to Y. These results are much more in line with our intuition.

2.16 a. Assuming the same instruction mix means that the additional instructions for each task should be allocated proportionally among the instruction types. So we have the following table:

Instruction Type	CPI	Instruction Mix
Arithmetic and logic	1	60%
Load/store with cache hit	2	18%
Branch	4	12%
Memory reference with cache miss	12	10%

$CPI = 0.6 + (2 \times 0.18) + (4 \times 0.12) + (12 \times 0.1) = 2.64$. The CPI has increased due to the increased time for memory access.

b. $MIPS = 400/2.64 = 152$. There is a corresponding drop in the MIPS rate.

c. The speedup factor is the ratio of the execution times. Using Equation 2.2, we calculate the execution time as $T = I_c/(MIPS \times 10^6)$. For the single-processor case, $T_1 = (2 \times 10^6)/(178 \times 10^6) = 11$ ms. With 8 processors, each processor executes 1/8 of the 2 million instructions plus the 25,000 overhead instructions. For this case, the execution time for each of the 8 processors is

$$T_8 = \frac{\frac{2 \times 10^6}{8} + 0.025 \times 10^6}{152 \times 10^6} = 1.8 \text{ ms}$$

Therefore we have

$$\text{Speedup} = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{11}{1.8} = 6.11$$

d. The answer to this question depends on how we interpret Amdahl's' law. There are two inefficiencies in the parallel system. First, there are additional instructions added to coordinate between threads. Second, there is contention for memory access. The way that the problem is stated implies that none of the code is inherently serial. All of it is parallelizable, but with scheduling overhead. One could argue that the memory access conflict means that to some extent memory reference instructions are not parallelizable. But based on the information given, it is not clear how to quantify this effect in Amdahl's equation. If we assume that the fraction of code that is parallelizable is $f = 1$, then Amdahl's law reduces to $Speedup = N = 8$ for this case. Thus the actual speedup is only about 75% of the theoretical speedup.

2.17 **a.** Speedup = (time to access in main memory)/(time to access in cache) = T_2/T_1.

b. The average access time can be computed as $T = H \times T_1 + (1 - H) \times T_2$

Using Equation (2.8):

$$\text{Speedup} = \frac{\text{Execution time before enhancement}}{\text{Execution time after enhancement}} = \frac{T_2}{T} = \frac{T_2}{H \times T_1 + (1-H)T_2} = \frac{1}{(1-H) + H\dfrac{T_1}{T_2}}$$

c. $T = H \times T_1 + (1 - H) \times (T_1 + T_2) = T_1 + (1 - H) \times T_2)$

This is Equation (4.2) in Chapter 4. Now,

$$\text{Speedup} = \frac{\text{Execution time before enhancement}}{\text{Execution time after enhancement}} = \frac{T_2}{T} = \frac{T_2}{T_1 + (1-H)T_2} = \frac{1}{(1-H) + \dfrac{T_1}{T_2}}$$

In this case, the denominator is larger, so that the speedup is less.

2.18 $T_w = w/\lambda = 8/18 = 0.44$ hours