MODULE 1: ORGANIZATION AND ARCHITECTURE

UNIT 1: COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT 2: INSTRUCTION SETS CHARACTERISTICS AND FUNCTIONS

UNIT 3: TYPES OF OPERANDS

UNIT 1

1.0 INTRODUCTION

In spite of the variety and pace of change in the computer field, certain fundamental concepts apply consistently throughout. To be sure, the application of these concepts depends on the current state of technology and the price/ performance objectives of the designer.

Many computer manufacturers offer a family of computer models, all with the same architecture but with differences in organization. In a class of computers called microcomputers, the relationship between the architecture and organization is very close. Changes in technology not only influence organization but also result in the introduction of more powerful and more complex architecture. However, because a computer organization must be

designed o implement a particular architectural specification, a thorough treatment of organization requires a detailed examination of architecture as well.

2.0 OBJECTIVES

At the end of this unit you should be able to:

- Explain the operational units of a computer system.
- Outline types of operands and operations specific by machine instruction.
- Explain opcodes, operands and addressing modes

3.0 MAIN CONTENT

3.1 COMPUTER ORGANIZATION AND ARCHITECTURE

Although it is difficult to give precise definition, a consensus exists about the general area covered by it. Computer organization refers to the operational units and their interconnection that realize the architectural specification. Examples of architectural attributes include the instruction set, the number of bit used to represent various data types (e. g numbers, characters), I/O mechanism, and techniques for addressing memory. Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer peripherals and the memory technology used.

3.2 STRUCTURE AND FUNCTION

A computer is a computer system, contemporary computers contain millions of elementary electronic components.

- **Structure:** The way in which the components are interrelated.
- **Function:** The operation of each individual component as part of the structure.

In term of description, there are two choices: starting at the bottom and building up to a complete description, or beginning with a top view and

decomposing the system into its subparts. Evidence from a number of fields suggest that the top down approach is the clearest and most effective.
The approach taken is that the computer be described from the top down.

Both the structure and functioning of a computer are simple. Figure 1.1 depicts the basic functions that a computer can perform. In general terms, there are only four:

- Data processing
- Data storage
- Data movement
- Control

The computer of course, must be able to process data. The data may take a wide variety of forms, and the range of processing requirements id broad. It is also essential that a computer store data. Even if the computer is processing data on the fly (i.e data come in and get processed and the results go out immediately) the computer must temporarily store at least. Those pieces of data that are being worked on at any given moment. Files of data are stored on the computer for subsequent retrieval and update.

The computer must be able to move data between itself an outside world. The computers operating environment consist of devices that serve as either sources or destinations of data. When data are received from or delivered to a device that is directly connected to the computer, the process is known as input- output (I/O), and the device is referred to as a peripheral. When data are moved over longer distances, to or form a remote device, the process is known as data communications. Finally, there must be control of these three functions. Ultimately, this control is exercised by the individuals who provide the computer with instructions. Within the computer a control unit manages the computers resources and orchestrates the performance of its functional parts in response to those instructions.
There are four main structural components

- Understand the way in which numbers are represented (the binary format) and the algorithms used for the basic arithmetic operations (add, subtract, multiply, divide) both to integer and floating point arithmetic.

3.1 THE ARITHMETIC AND LOGIC UNIT

The arithmetic and logic unit (ALU) is that part of the computer that actually performs arithmetic and logical operations on data. All of the other elements of the computer system- Control unit, registers memory, I/0- are there mainly to bring into the ALU for it to process and then take the result back out.

An ALU and all electronic components in the computers are based on the use of simple digital logic devices that can store binary digits and perform simple Boolean logic operations.

Figure 3. 1. 1 indicates, in general terms, how the ALU is interconnected with the rest of the processor. Data are presented to the ALU in registers and the results of an operation are stored in registers. These registers are temporary storage locations within the processor that are connected by signal paths to the ALU. The ALU may also set flags as the result of an operation. For example, an overflow flag is set to 1 if the result of a computation exceeds the length of the register into which it is to be stored. The flag values are also stored in registers within the processor. The control unit provides signals that control the operation of the ALU and the movement of the data into and out of the ALU.

3.2 INTEGER REPRESENTATION

In the binary number, arbitrary numbers can be represented with just the digits zero and one the minis sign and the period or radix point.

$$-1101.0101_2 = -13.3125_{10}$$

For purposes of computer storage and processing, however w do not have the benefits of minus signs and periods. Only binary digits (0 and 1) may be used

to represent numbers. If we are limited to non negative integers, the representation is straight forward.

An 8 bit word can represent the numbers form 0 to 255, including

00000000	=	0
00000001	=	1
00101001	=	41
10000000	=	128
11111111	=	255

In general, if an n- bit sequence of binary digits is interpreted as an unsigned integer, A it value is

$$A = \sum_{2=0}^{n-1} 2i\, ai$$

In going from the first to the second equation, we require that the least significant *n* - 1 bits do not change between the two representations. Then we get to .-next to last equation, which is only true if all of the bits in positions throem *2* are 1. Therefore, the sign-extension rule works.

Fixed-point representation

Finally, we mention that the representations discussed in this section are sometime referred to as fixed point. This is because the radix point (binary point) is fixed assumed to be to the right of the rightmost digit. The programmer can use the representation for binary fractions by scaling the numbers so that the binary poor implicitly positioned at some other location.

Negation

In sign-magnitude representation, the rule for forming the negation of an integer is simple: invert the sign bit. In twos complement notation, the negation of an integer can be formed with the following rules:

Take the Boolean complement of each bit of the integer (including the sign bit). That is, set each 1 to 0 and each 0 to 1.

Treating the result as an unsigned binary integer, add 1.

(MAR) because this is only register connected to the address lines of the system bus. The second step bring in the instruction. The desired address (in the MAR) is placed on the ad c== -

The IR is now in the same state as if indirect addressing had not been use and it is ready for the execute cycle. We skip that cycle for a moment, to consider t interrupt cycle.

At the completion of the execute cycle, a test is made to determine whether any- :-_abled interrupts have occurred. If so, the interrupt cycle occurs. The nature of cycle varies greatly from one machine to another. We present a very simple sequeof events, as illustrated in Figure 12.8. We have

t_1: MBR E- (PC)

t_2: MAR F- Save Address PC F- Routine Address t_3: Memory E- (MBR)

In the first step, the contents of the PC are transferred to the MBR, so that u- can be saved for return from the interrupt. Then the MAR is loaded with the add- .at which the contents of the PC are to be saved, and the PC is loaded with the add to the MAR and PC, respectively. In any case, once this is done, the final step is to store the MBR, which contains the old value of the PC, into memory. The processor is now ready to begin the next instruction cycle.

The fetch, indirect, and interrupt cycles are simple and predictable. Each involves a small, fixed sequence of micro-operations and, in each case, the same micro-operations are repeated each time around.

This is not true of the execute cycle. Because of the variety opcodes, there are a number of different sequences of micro-operations that can occur. Let us consider several hypothetical examples.

First, consider an add instruction:

ADD R1, X

which adds the contents of the location X to register R1. The following sequence of micro-operations might occur:

We begin with the IR containing the ADD instruction. In the first step, the address portion of the IR is loaded into the MAR. Then the referenced memory

Table 15.2 Intel 8085 External Signals

Address and Data Signals

High Address (A15–A8)
The high-order 8 bits of a 16-bit address.

Address/Data (AD7–AD0)
The lower-order 8 bits of a 16-bit address or 8 bits of data. This multiplexing saves on pins.

Serial Input Data (SID)
A single-bit input to accommodate devices that transmit serially (one bit at a time).

Serial Output Data (SOD)
A single-bit output to accommodate devices that receive serially.

Timing and Control Signals

CLK (OUT)
The system clock. The CLK signal goes to peripheral chips and synchronizes their timing.

X1, X2
These signals come from an external crystal or other device to drive the internal clock generator.

Address Latch Enabled (ALE)
Occurs during the first clock state of a machine cycle and causes peripheral chips to store the address lines. This allows the address module (e.g., memory, I/O) to recognize that it is being addressed.

Status (S0, S1)
Control signals used to indicate whether a read or write operation is taking place.

IO/M
Used to enable either I/O or memory modules for read and write operations.

Read Control (RD)
Indicates that the selected memory or I/O module is to be read and that the data bus is available for data transfer.

Write Control (WR)
Indicates that data on the data bus is to be written into the selected memory or I/O location.

Memory and I/O Initiated Symbols

Hold
Requests the CPU to relinquish control and use of the external system bus. The CPU will complete execution of the instruction presently in the IR and then enter a hold state, during which no signals are inserted by the CPU to the control, address, or data buses. During the hold state, the bus may be used for DMA operations.

Hold Acknowledge (HOLDA)
This control unit output signal acknowledges the HOLD signal and indicates that the bus is now available.

READY
Used to synchronize the CPU with slower memory or I/O devices. When an addressed device asserts READY, the CPU may proceed with an input (DBIN) or output (WR) operation. Otherwise, the CPU enters a wait state until the device is ready.

(Continued)

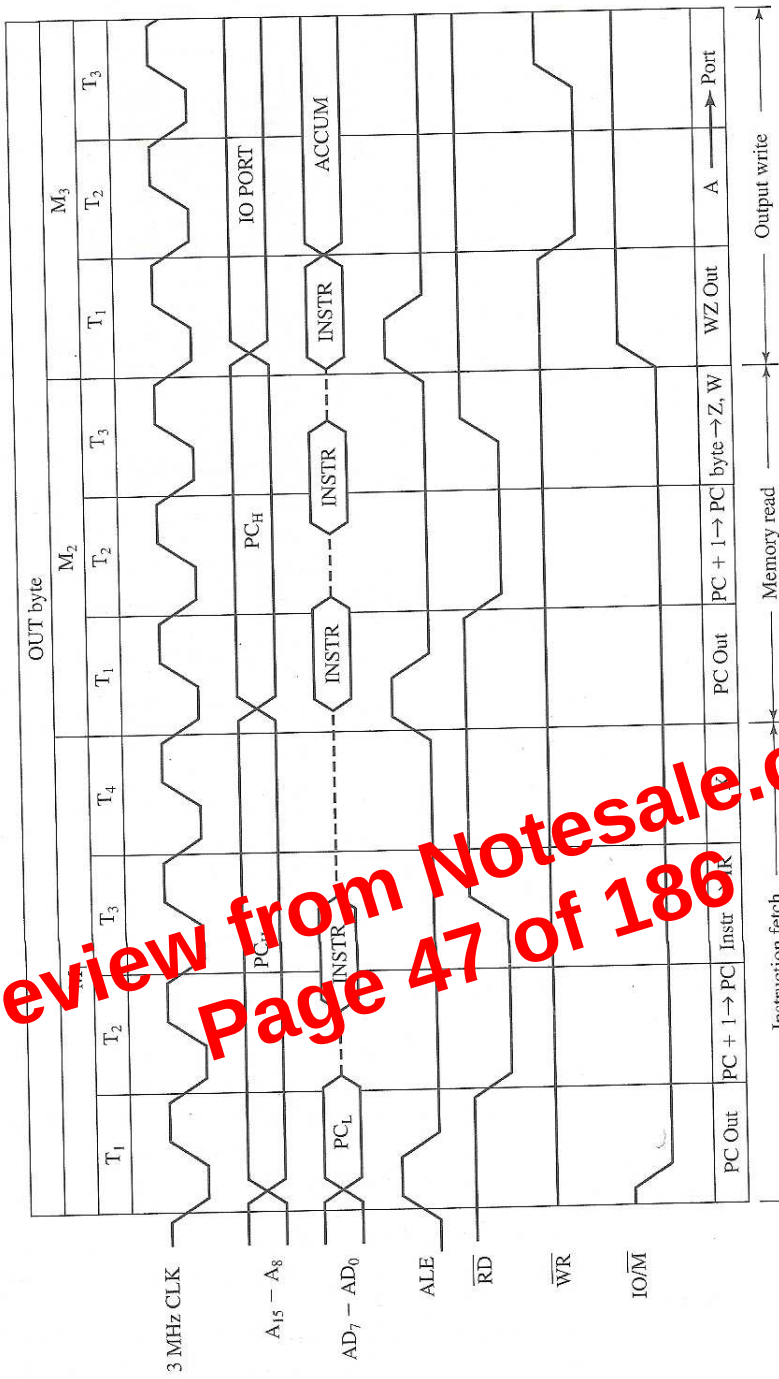


Figure 15.9 Timing Diagram for Intel 8085 OUT Instruction

addressed memory module places the contents of the addressed memory vocation on the address/data bus. The control unit sets the Read Control (RD) signal to indicate a read, but it waits until T_3 to copy the data from the bus. This gives the memory module time to put the data on the bus and for the signal levels to stabilize. The final state, T_4, is a bus *idle* state during which the processor decodes the instruction. The remaining machine cycles proceed in a similar fashion.

Finally, consider a subroutine call instruction. As an example, consider a branchand-save-address instruction:

BSA X

The address of the instruction that follows the BSA instruction is saved in location X, and execution continues at location X + I. The saved address will later be uses for return. This is a straightforward technique for providing subroutine calls. The fo=lowing micro-operations suffice:

t_1: MAR E- (IR(address)) MBR ~- (PC)

t_2: PC <-- (IR(address)) Memory <-- (MBR) t_3: PC <_ (PC) + I

The address in the PC at the start of the instruction is the address of the next instruction in sequence. This is saved at the address designated in the IR. The lateeaddress is also incremented to provide the address of the instruction for the next it - struction cycle.

We have seen that each phase of the instruction cycle can be decomposed into a sequence of elementary micro-operations. In our example, there is one sequence eac= for the fetch, indirect, and interrupt cycles, and, for the execute cycle, there is one sequence of micro-operations for each opcode.

To complete the picture, we need to tie sequences of micro-operations to-gether, and this is done in Figure 15.3. We assume a new 2-bit register called

the *instruction cycle code (ICC).* The ICC designates the state of the processor in terms of which portion of the cycle it is in:

00: Fetch 01: Indirect

10: Execute 11:

Interrupt

At the end of each of the four cycles, the ICC is set appropriately. The indirect cycle is always followed by the execute cycle. The interrupt cycle is always followed by the fetch cycle (see Figure 12.4). For both the fetch and execute cycles, the next cycle depends on the state of the system.

Thus, the flowchart of Figure 15.3 defines the complete sequence of microoperations, depending only on the instruction sequence and the interrupt pattern. Of course, this is a simplified example. The flowchart for an actual processor would be more complex. In any case, we have reached the point in our discussion in whic?, the operation of the processor is defined as the performance of a sequence of microoperations. We can now consider how the control unit causes this sequence to occur.

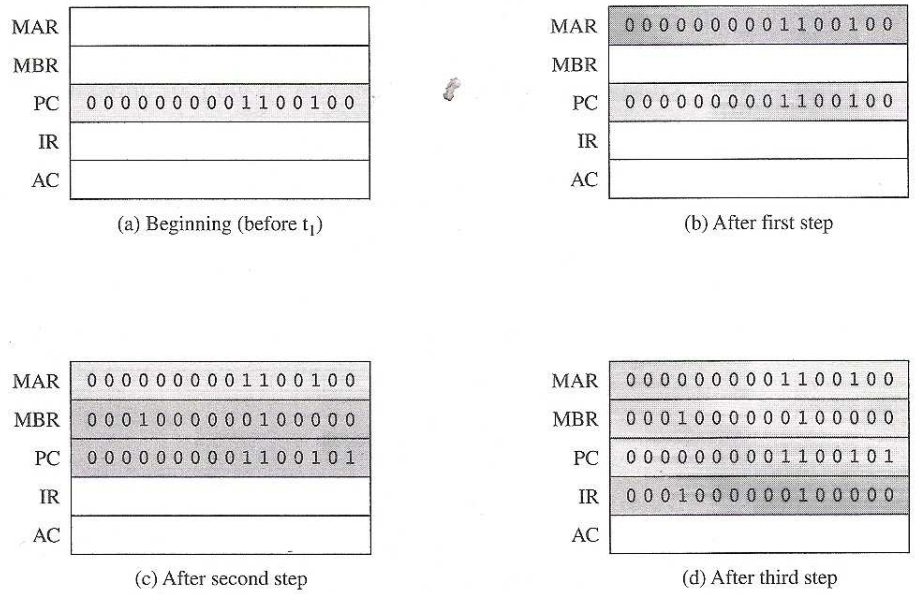
of tbp ~~r of the interrupt-processing routine. These two actions may each be - single micro-operation. However, because most processors provide multiple types and/or levels of interrupts, it may take one or more additional micro-operations obtain the Save Address and the Routine Address before they can be transfer the events of any instruction cycle can be described as a sequence of such micro operations. A simple example will be used. In the remainder of this chapter, we then show how the concept of micro-operations serves as a guide to the design of the control unit.

THE FETCH CYCLE

We begin by looking at the fetch cycle, which occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory. Four registers are involved:

- **Memory address register (MAR):** Is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.
- **Memory buffer register (MBR):** Is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from memory.
- **Program counter (PC):** Holds the address of the next instruction to be fetched.
- **Instruction register (IR):** Holds the last instruction fetched.

Let us look at the sequence of events for the fetch cycle from the point of view of its effect on the processor registers. An example appears in Figure 3.1.2. At the beginning of the fetch cycle, the address of the next instruction to be executed is in the program counter (PC); in this case, the address is 1100100. The first step is to move that address to the memory address register (MAR) because this is the only register connected to the address lines of the system bus. The second step is to bring in the instruction. The desired address (in the MAR) is placed on the address



3.2 CONTROL OF THE PROCESSOR

As a result of our analysis in the preceding section, we have decomposed the behavior or functioning of the, processor into elementary operations, called micro-operations. By reducing the operation of the processor to its most fundamental level, we are able to define exactly what it is that the control unit must cause to happen. Thus, we can define the *functional requirements* for the control unit: those functions that the control unit must perform. A definition of these functional requirements is the basis for the design and implementation of the control unit.

With the information at hand, the following three-step process leads to a characterization of the control unit:

1. Define the basic elements of the processor.
2. Describe the micro-operations that the processor performs.
3. Determine the functions that the control unit must perform to cause the micro-operations to be performed.

We have already performed steps 1 and 2. Let us summarize the results. First, the basic functional elements of the processor are the following:

- ALU
- Registers
- Internal data paths External data paths
- Control uni

Some thought should convince you that this is i complete list. The ALU is the functional essence of the computer. Registers are used to store data internal to the processor. Some registers contain status information needed to manage instruction sequencing (e.g., a program status word). Others contain data that go to or come from the ALU, memory, and I/O modules. Internal data paths are used to move data between registers and between register and ALU. External data paths link registers to memory and 1/O modules, often by means of a system bus. The control unit causes operations to happen within the processor.

output would feed back to the input. Register Z provides temporary output storage. With this arrangement, an operation to add a value from memory to the AC would have the following steps:

t_1: MAR ⟵ (IR (address))
t_2: MBR ⟵ Memory
t_3: Y ⟵ (MBR)
t_4: Z ⟵ (AC) + (Y)
t_5: AC ⟵ (Z)

Other organizations are possible, but, in general, some sort of internal bus or set of internal buses is used. The use of common data paths simplifies the interconnection layout and the control of the processor. Another practical reason for the use of an internal bus is to save space.

To illustrate some of the concepts introduced thus far in this unit, let us consider the Intel 8085. Its organization is shown in Figure 3.2.5. Several key components that may not be self-explanatory are:

- **Incremental decrementer address latch:** Logic that can add 1 to or subtract 1 from the contents of the stack pointer or program counter. This saves time by avoiding the use of the ALU for this purpose.
- **Interrupt control:** This module handles multiple levels of interrupt signals.
- **Serial I/O control:** This module interfaces to devices that communicate 1 bit at a time.

Table 15.2 describes the external signals into and out of the 8085. These are linked to the external system bus. These signals are the interface between the 8085 processor and the rest of the system (Figure 15.8).

Interrupt-Related Signals

TRAP
Restart Interrupts (RST 7.5, 6.5, 5.5)

Interrupt Request (INTR)
These five lines are used by an external device to interrupt the CPU. The CPU will not honor the request if it is in the hold state or if the interrupt is disabled. An interrupt is honored only at the completion of an instruction. The interrupts are in descending order of priority.

Interrupt Acknowledge
Acknowledges an interrupt.

CPU Initialization

RESET IN
Causes the contents of the PC to be set to zero. The CPU resumes execution at location zero.

RESET OUT
Acknowledges that the CPU has been reset. The signal can be used to reset the rest of the system.

Voltage and Ground

VCC
+5-volt power supply

VSS
Electrical ground

The control unit is identified as having two components labeled (1) instruction decoder and machine cycle encoding and (2) timing and control. A discussion of the first component is deferred until the next section. The essence of the control unit is the timing and control module. This module includes a clock and accepts as inputs the current instruction and some external control signals. Its output consists of control signals to the other components of the processor plus control signals to the external system bus.

The timing of processor operations is synchronized by the clock and controlled by the control unit with control signals. Each instruction cycle is divided into from one to five *machine* cycles; each machine cycle is in turn divided into from three to five *states*. Each state lasts one clock cycle. During a state, the processor performs one or a set of simultaneous micro-operations as determined by the control signals.

The number of machine cycles is fixed for a given instruction but varies from one instruction to accesses. Thus, the number of machine cycles for an instruction depends on the number of times the processor must communicate with external devices. For example, if an instruction consists of two 8-bit portions, then two machine cycles are required to fetch the instruction. If that instruction involves a 1-byte memory or I/O operation, then a third machine cycle is required for execution.

5.0 SUMMARY

In a parallel organization, multiple processing units cooperate to execute applications whereas a superscalar process exploits opportunities for parallel execution at the instruction level, a parallels processing organization looks for a grosser level of parallelism one that enables work to be done in parallel and cooperatively by multiple processors

6.0 TUTOR- MARKED ASSIGNMENT

1. List and briefly define types of parallel processor system.
2. List the two most common multiple processor organizations

7.0 REFERENCES/FURTHER READING

Catanzaro B. Multi processor system Architecture Mountain View CA, Sun sift pres 1994

UNIT 2: SYMMETRIC MULTI PROCESSOR

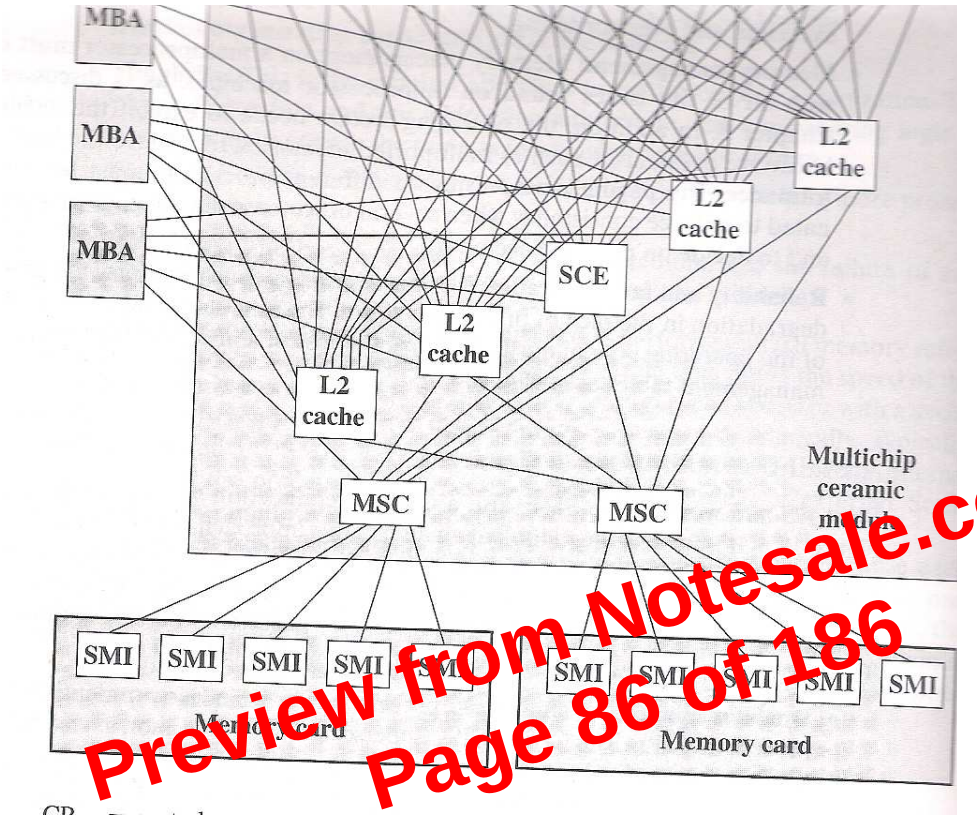
1.0 Introduction

Virtually all single user personal computers and most work stations contained a single generate purpose micro processors. As demand for performance increases and is the cost of microprocessors continues to drop. Vendors have introduced system with and SMP organization.

2.0 Objectives

At the end of this unit you should be able to

- **Memory has adapter (MBA):** The MBA provides an interface to various types of I/O channels. Traffic to/from the channels goes directly to the L2 cache.
- The microprocessor in the 2990 is relatively uncommon compared with other modern processors because although. It is superscalar it executes instructions in strict architectural order



4.0 CONCLUSION

The term SMP refers to a computer hardware architecture and also to the operating system behaviour that reflects that architecture. It can be defined as a stands alone computer system with the following characteristics.

1. There are two or more similar processors of comparable capability.
2. These processors share the same main memory and I/O facilities and are interconnected by a bus or other internal connection scheme such that memory aces time is approximately the same for each process.

7.0 References/ Further reading

Milenkovic, A. “Achieving High Performance in Bus- Based shared memory multiprocessors” IEEE concurrency, July- September 2000

UNIT 3: MULTI THREADING AND CHIP MULTI PROCESSORS

CONTENT

1.0 INTRODUCTION

The most important measure of performance for a processor is the rate at which it executes instructions. This can be expressed as MIPS rate = *f* x IPC where *f is* the processor clock frequency, in MHz, and IPC (instructions per cycle) is the average number of instructions executed per cycle. Accordingly, designers have pursued the goal of increased performance on two fronts: increasing clock frequency and increasing the number of instructions executed or, more properly, the number of instructions that complete during a processor cycle.

An alternative approach, which allows for a high degree of instruction-level parallelism without increasing circuit complexity or power consumption, is called multithreading. In essence, the instruction stream is divided into several smaller streams, known as threads, such that the threads can be executed in parallel.

Operation	Cycles
AR(J) * BR(J) → T1(J)	3
AI(J) * BI(J) → T2(J)	3
T1(J) − T2(J) → CR(J)	3
AR(J) * BI(J) → T3(J)	3
AI(J) * BR(J) → T4(J)	3
T3(J) + T4(J) → CI(J)	
TOTAL	18

(a) Storage to storage

Operation	Cycles
AR(J) → V1(J)	1
BR(J) → V2(J)	1
V1(J) * V2(J) → V3(J)	1
AI(J) → V4(J)	1
BI(J) → V5(J)	1
V4(J) * V5(J) → V6(J)	1
V3(J) − V6(J) → V7(J)	1
V7(J) → CR(J)	1
V1(J) * V5(J) → V8(J)	1
V4(J) * V2(J) → V9(J)	1
V8(J) + V9(J) → V0(J)	1
V0(J) → CI(J)	
TOTAL	12

(b) Register to register

Operation	Cycles
AR(J) → V1(J)	1
V1(J) * BR(J) → V2(J)	1
AI(J) → V3(J)	1
V3(J) * BI(J) → V4(J)	1
V2(J) − V4(J) → V5(J)	1
V5(J) → CR(J)	1
V1(J) * BI(J) → V6(J)	1
V4(J) * BR(J) → V7(J)	1
V6(J) + V7(J) → V8(J)	1
V8(J) → CI(J)	
TOTAL	10

(c) Storage to register

Operation	Cycles
AR(J) → V1(J)	1
V1(J) * BR(J) → V2(J)	1
AI(J) → V3(J)	1
V2(J) − V3(J) * BI(J) → V2(J)	1
V2(J) → CR(J)	1
V1(J) * BI(J) → V4(J)	1
V4(J) + V3(J) * BR(J) → V5(J)	1
V5(J) → CI(J)	
TOTAL	8

(d) Compound instruction

Vi = Vector registers
AR, BR, AI, BI = Operands in memory
Ti = Temporary locations in memory

This approach is also used on the Cray supercomputer. An alternative approach, used on Control Data machines, is to obtain operands directly from memory. The main disadvantage of the use of vector registers is that the programmer or compiler must take them into account for good performance. For example, suppose that the length of the vector registers is K and the length of the vectors to be processed is $N > K$. In this case, a vector loop must be performed, in which the operation is performed on K elements at a time and the loop is repeated N/K times. The main advantage of the vector register approach is that the can be resumed from operation is decoupled from slower main memory and instead takes place primarily taken, in a marine with registers.

The speedup that can be achieved using registers is demonstrated in F17.20. The FORTRAN routine multiplies vector A by vector B to produce C, where each vector has a real part (AR, BR, CR) and an imaginary part (Ai. CI). The 3090 can perform one main-storage access per processor, or clock.(either read or write); has registers that can sustain two accesses for reading one for writing per cycle; and produces one result per cycle in its arithmetic. Let us assume the use of instructions that can specify two source operands result. Part a of the figure shows that, with memory-to-memory instructions iteration of the computation 'requires a total of 18 cycles.

Generalizing from the work of a number of researchers, three elements emerge that, by and large, characterize RISC architectures. First, use a large number of registers or use a compiler to optimize register usage. This is intended to optimize operand referencing. The studies just discussed show that there are several references per HLL instruction and that there is a high proportion of move (assignment) statements. This, coupled with the locality and predominance of scalar references, suggests that performance can be improved by reducing memory references at the expense of more register references. Because of the locality of these references, an expanded register set seems practical.

Second, careful attention needs to be paid to the design of instruction pipelines. Because of the high proportion of conditional branch and procedure call instructions, a straightforward instruction pipeline will be inefficient. This manifests itself as a high proportion of instructions that are prefetched but never executed.

Finally, a simplified (reduced) instruction set is indicated. This point is not as obvious as the others, but should become clearer in the ensuing discussion.

4.0 CONCLUSION

Assignment statements predominate, suggesting that the simple movement of data should be optimized. There are also many IF and LOOP instructions, which suggest that the underlying sequence control mechanism needs to be optimized to permit efficient pipelining. Studies of operand reference patterns suggest that it should be possible to enhance performance by keeping a moderate number of operands in registers.

5.0 SUMMARY

The simple instruction set of a RISC lends itself to efficient pipelining because there are fewer and more predictable operations performed per instruction. Other instruction to improve pipeline efficiency.

6.0 Tutor marked assignment

1. What is a delayed branch?

7.0 REFERENCES/ FURTHER READING

Patterson, D “Reduced instruction set computers communications of the ACM, January 1985.

UNIT 2: REDUCED INSTRUCTION SET ARCHITECTURE

CONTENT

1.0 INTRODUCTION

This is the focus of this unit. The RISC architecture is a dramatic departure from the historical trend in processor architecture. An analysis of the RISC architecture brings into focus many of the important issues in computer organization and architecture.

2.0 OBJECTIVES

At the end of this unit, you should be able to understand the pitfalls in the CISC approach in companion to RISC.

3.1 Why CISC

In this section, we look at some of the general characteristics of and the motivation for a reduced instruction set architecture. Specific examples will be seen later in this chapter. We begin with a discussion of motivations for contemporary complex instruction set architectures.

We have noted the trend to richer instruction sets, which include a larger number of instructions and more complex instructions. Two principal reasons have motivated this trend: a desire to simplify compilers and a desire to improve performance. Underlying both of these reasons was the shift to HLLs on the part of programmers; architects attempted to design machines that provided better support for HLLs.

It is not the intent of this chapter to say that the CISC designers took the wrong direction. Indeed, because technology continues to evolve and because architectures exist along a spectrum rather than in two neat categories, a black-and-white assessment is unlikely ever to emerge. Thus, the comments that follow are simply meant to point out some of the potential pitfalls in the CISC approach and to provide some understanding of the motivation of the RISC adherents.

The first of the reasons cited, compiler simplification, seems obvious. The task of the compiler writer is to generate a sequence of machine instructions for each FILL statement. If there are machine instructions that resemble HLL statements, this task is simplified. This reasoning has been disputed by the RISC researchers ([HENN82]. [RAIJI83], [PATT82b]). They have found that complex machine instructions are often hard to exploit because the compiler must find those cases that exactly fit the construct. The task of optimizing the generated code to minimize code size, reduce instruction execution count, and enhance pipelining is much more difficult with a complex instruction set. As evidence of this, studies cited earlier in this chapter indicate that most of the instructions in a compiled program are the relatively simple ones.

The other major reason cited is the expectation that a CISC will yield smaller. faster programs. Let us examine both aspects of this assertion: that programs will be smaller and that they will execute faster.

There are two advantages to smaller programs. First, because the program takes up less memory, there is a savings in that resource. With memory today being so inexpensive, this potential advantage is no longer compelling. More important

	[PATT82a] 11 C Programs	[KATE83] 12 C Programs	[HEAT84] 5 C Programs
RISC I	1.0	1.0	1.0
VAX-11/780	0.8	0.67	
M68000	0.9		0.9
Z8002	1.2		1.12
PDP-11/70	0.9	0.71	

smaller programs should improve performance, and this will happen in two ways. First, fewer instructions means fewer instruction bytes to be fetched. Second, in a paging environment, smaller programs occupy fewer pages, reducing page faults.

The problem with this line of reasoning is that it is far from certain that a CISC program will be smaller than a corresponding RISC program. In many cases, the CISC program, expressed in symbolic machine language, may be *shorter (i.e.*, fewer instructions), but the number of

An OS is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware. It can be thought of as having two objectives:

- **Convenience:** An OS makes a computer more convenient to use.
- **Efficiency:** An OS allows the computer system resources to be used in an efficient manner.

Let us examine these two aspects of an OS in turn.

The operating system a as user/ computer interface

The hardware and software used in providing applications to a user can be viewed in a layered or hierarchical fashion, as depicted in Figure 8.1. The user of those applications, the end user, generally is not concerned with the computer's architecture. Thus the end user views a computer system in terms of an application. That application row can used in a programming language and is developed by an application programme. To develop an application program as a set of processor instructions that is completely responsible for controlling the computer hardware would be an overwhelmingly complex task. To ease this task, a set of systems programs is provided. Some of these programs are referred to as **utilities.** These implement frequently used functions that assist in program creation, the management of files, and the control of I/O devices. A programmer makes use of these facilities in developing an application, and the application, while it is running, invokes the utilities to perform certain functions. The most important system program is the OS. The OS masks the details of the hardware from the programmer and provides the programmer with a convenient interface for using the system. It acts as mediator, making it easier for the programmer and for application programs to access and use those facilities and services.

process some time in turn. Priority levels may also be used. Finally, there is an I/O queue for each I/O device. More than one process may request the use of the same I/O device. All processes waiting to use each device are lined up in that device's queue.

Figure 8.11 suggests how processes progress through the computer under the control of the OS. Each process request (batch job, user-defined interactive job) is placed in the long-term queue. As resources become available, a process request becomes a process and is then placed in the ready state and put in the short-term queue. The processor alternates between executing OS instructions and executing user processes. While the OS is in control, it decides which process in the short-term queue should be executed next. When the OS has finished its immediate tasks, it turns the processor over to the chosen process.

As was mentioned earlier, a process being executed may be suspended for a variety of reasons. If it is suspended because the process requests I/O, then it is placed in the appropriate I/O queue. If it is suspended because of a timeout or because the OS must attend to pressing business. then it is placed in the ready state and put into the short-term queue.

Finally, we mention that the OS also manages the I/O queues. When an I/O operation is completed, the OS removes the satisfied process from that I/O queue and places it in the short-term queue. It then selects another waiting process (if any) and signals for the I/O device to satisfy that process's request.

UNIT 3: Memory System

1.0 Introduction

2.0 Objectives

3.0 Main content

3.1 Characteristics of memory systems

3.2 The memory hierarchy

3.3 Error correction

1.0 Introduction

Computer memory is organized into a hierarchy. At the highest level (closest to the processor) are the processor registers. Next comes one or more levels of cache, When multiple levels are used, they are denoted L1, L2, and so on.Error correction techniques are commonly used in memory systems.

2.0 objectives

At the end of this unit, you should be able to

Table 4.1 Key Characteristics of Computer Memory Systems

Location	**Performance**
Internal (e.g. processor registers, main memory, cache)	Access time
External (e.g. optical disks, magnetic disks, tapes)	Cycle time
Capacity	Transfer rate
Number of words	**Physical Type**
Number of bytes	Semiconductor
Unit of Transfer	Magnetic
Word	Optical
Block	Magneto-optical
Access Method	**Physical Characteristics**
Sequential	Volatile/nonvolatile
Direct	Erasable/nonerasable
Random	**Organization**
Associative	Memory modules

the form of registers (e.g., see Figure 2.3). Further, as we shall see, the control unit portion of the processor may also require its own internal memory. We will defer discussion of these latter two types of internal memory to later chapters. Cache is another form of internal memory. External memory consists of peripheral storage devices, such as disk and tape, that are accessible to the processor via I/O controllers.

An obvious characteristic of memory is its capacity. For internal memory, this is typically expressed in terms of bytes (1 byte = 8 bits) or words. Common word lengths are 8, 16, and 32 bits. External memory capacity is typically expressed in terms of bytes.

A related incept is the unit of transfer. For internal memory, the unit of transfer is equal to the number of electrical lines into and out of the memory module. This may be equal to the word length, but is often larger, such as 64,128, or 256 bits. To clarify this point, consider three related concepts for internal memory:

- **Word:** The "natural" unit of organization of memory. The size of the word is typically equal to the number of bits used to represent an integer and to the instruction length. Unfortunately, there are many exceptions. For example, the **CRAY** C90 (an older model CRAY supercomputer) has a 64-bit word length but uses a 46-bit integer representation. The Intel x86 architecture has a wide variety of instruction lengths, expressed as multiples of bytes, and a word size of 32 bits.
- **Addressing units:** many systems allow addressing at the byte level. In any case, the relationship between the length in bits A of an address and the number N of addressable units is $2^A = N$.
- **Unit of transfer:** For main memory, this is the number of bits read out of or written into memory at a time. The unit of transfer need not equal a word or an addressable

processor. It is a de-.-ice for staging the movement of data between main memory and processor registers to improve performance.

The three forms of memory just described are, typically, volatile and employ miconductor technology. The use of three levels exploits the fact that semiconductor memory comes in a variety of types, which differ in speed and cost. Data are stored more permanently on external mass storage devices, of which the most com on are hard disk and removable media, such as removable magnetic disk, tape, and optical storage. External, nonvolatile memory is also referred to as secondary memory auxiliary memory. These are used to store program and data files and are usually invisible to the programmer only in terms of files and records, as opposed to individual bytes or words. Disk is also used to provide an extension to main memory own as virtual memory, which is discussed in Chapter 8.

Other forms of memory may be included in the hierarchy. For example, large 1 mainframes include a form of internal memory known as expanded storage is uses a semiconductor technology that is slower and less expensive than that of in memory. Strictly speaking, this memory does not fit into the hierarchy but is a branch: Data can be moved between main memory and expanded storage but between expanded storage and external memory. Other forms of secondary memory include optical and magneto-optical disks. Finally, additional levels can be positively added to the hierarchy in software. A portion of main memory can be used as a buffer to hold data temporarily that is to be read out to disk. Such a technique, sometimes referred to as a disk cache, [2] improves performance in two ways

- Disk writes are clustered. Instead of many small transfers of data, we have a few large transfers of data. This improves disk performance and minimize processor involvement.
- Some data destined for write-out may be referenced by a program before the next dump to disk. In that case, the data are retrieved rapidly from the software cache rather than slowly from the disk.

1.0 Conclusion

As one goes down the memory hierarchy one finds decreasing cost bit, increasing capacity, and slower access time. This unit focuses on internal memory elements.

2.0 Summary

Although seemingly simple in concept, computer memory exhibits perhaps the widest range of type, technology, organization, performance and cost of any feature of computer system

The error correction technique involves adding redundant bits that are a function of the data bit to form an error correction code. If a bits error occurs, the code will detect and usually correct the error.

3.0 Tutor Marked Assignment

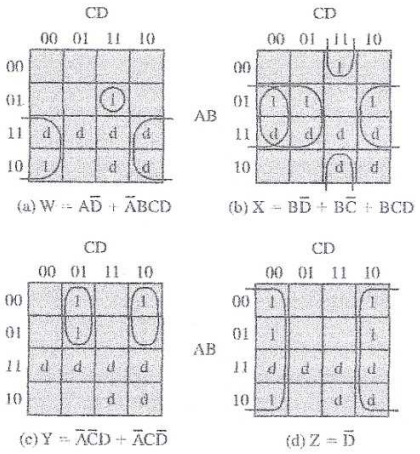
1. What are the differences among direct mapping, associative mapping and set associative mapping?
2. What is a parity bit?
3. How is the syndrome for the hamming code interpreted?

4.0 References/Further reading

1. Adamck, J. Foundation of coding New York Wiley 1991
2. Smith,a CACHE MEMORIES ACM computing surveys September 1992

UNIT 4: CACHE MEMORY

The first step is to con_ trLictt a table in which each row corresponds to one of the product terms of the expression. The terms are grouped according to the number of complemented variables. I =:at is; eve start with the term with no complements, if it exists, then all terms with one complement, and so on. Table 20.5 shows the list for our example ***expression,*** **with horizontal fines** used to indicate the grouping. For clarity,



equal the value of the selected input gate. Using this regular organization, it is easy to construct multiplexers of size 8-to-1,16-to-1, and so on.

Multiplexers are used in digital circuits to control signal and data routing. An example is the loading of the program counter (PC). The value to be loaded into the program counter may come from one of several different sources:

A binary counter, if the PC is to be incremented for the next instruction

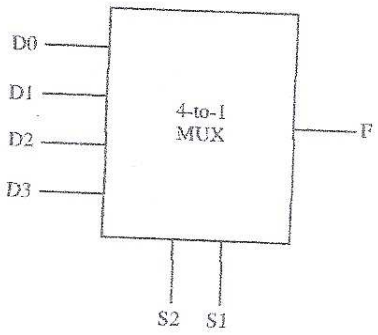


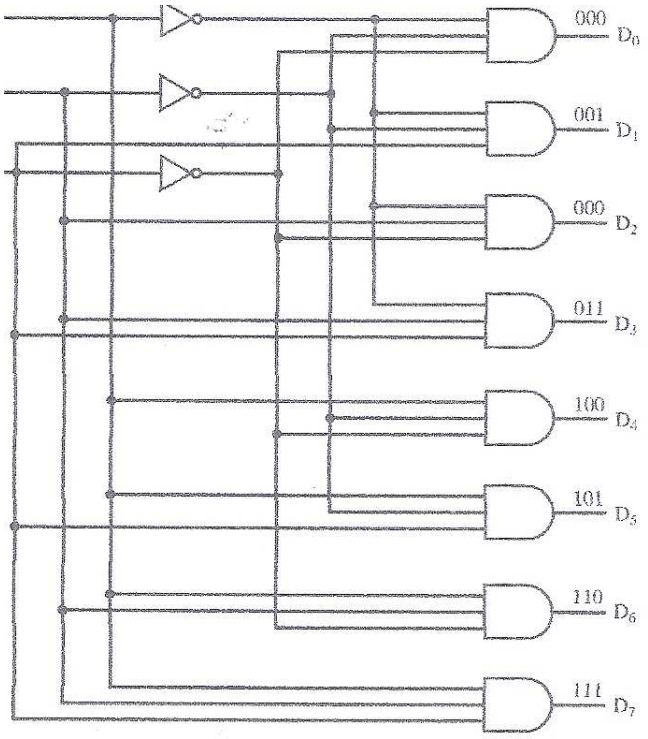
Figure 20.12 4-to-1 Multiplexer Representation

Table 20.1 4-to-1 Multiplexer

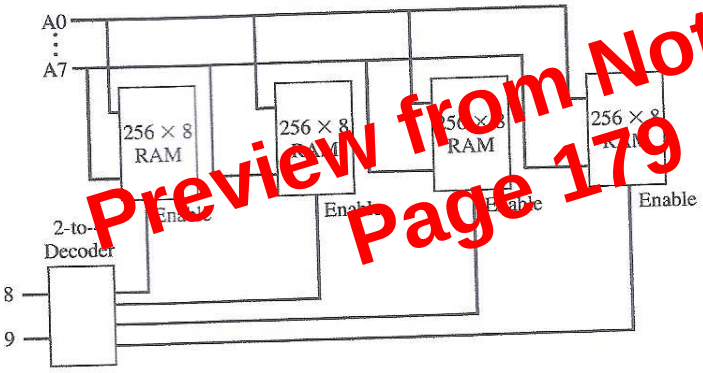
S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

- The instruction register, if a branch instruction using a direct address has just been executed
- The output of the AT U, if the branch instruction specifies the address using a displacement mode

These various inputs could be connected to the input lines of a mulltiplexer, with the PC connected to the output line. The select lines determine which value is loaded into the PC. Because the PC contains multiple bits, multiple multiplexers are used, one per bit. Figure 20.14 illustrates this for 16-bit addresses.



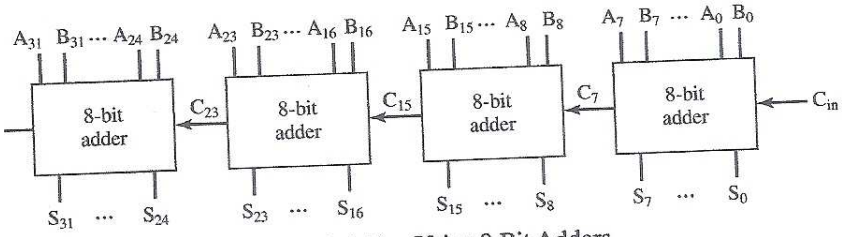
ure 20.15 Decoder with 3 Inputs and $2^3 = 8$ Outputs



ıgure 20.16 Address Decoding

Each chip requires 8 address lines, and these are supplied by the lower-order 8 bits of the address. The higher-order 2 bits of the 10-bit address are used to select one of the four RAM chips. For this purpose, a 2-to-4 decoder is used whose output enables one of the four chips, as shown in Figure 20.16.

With an additional input line, a decoder can be used as a demultiplexer. The demultiplexer performs the inverse function of a multiplexer; it connects a single input to one of several outputs. This is shown in Figure 20.17. As before, n inputs are decoded to produce a single one of 2" outputs. All of the 2" output lines are ANDed with a data input



20.21 Construction of a 32-Bit Adder Using 8-Bit Adders

Thus we have the necessary logic to implement a multiple-bit adder such as shown in Figure 20.21. Note that because the output from each adder depends on the carry from the previous adder, there is an increasing delay from the least significant to the most significant bit. Each single-bit adder experiences a certain amount of gate delay, and this gate delay accumulates. For larger adders, the accumulated delay can become unacceptably high.

If the carry values could be determined without having to ripple through all the previous stages, then each single-bit adder could function independently, and delay would not accumulate. This can be achieved with an approach known as carry *lookahead*. Let us look again at the 4-bit adder to explain this approach.

We would like to come up with an expression that specifies the carry input to any stage of the adder without reference to previous carry values. We have

$$C_o = A_O B_Q \qquad (20.4)$$

$$C_i = A_j B_j + A_j A_0 B_0 + B_I A_G B_O \qquad (20.5)$$

$$C_o = A_O B_Q \qquad (20.4)$$

$$C_i = A_j B_j + A_j A_0 B_0 + B_I A_G B_O \qquad (20.5)$$

$$C2 = A2B2 + A_2A_jB_i + A_2A_jA_0B_o + A_2B_jA_0B_o + B2ACB1 + B_2A_jA_0B_o + B2BjA0Bo$$

This process can be repeated for arbitrarily long adders. Each carry term can be expressed in SOP form as a function only of the original inputs, with no dependence on the carries. Thus, only two levels of gate delay occur regardless of the length of the adder.

For long numbers, this approach becomes excessively complicated. Evaluating the expression for the most significant bit of an n-bit adder requires an OR gate with n - 1