**Module 1 Lessone 3**

**Computer Components**

**Chapter Objectives**

At the global and processor levels of computer design, we are concerned about four component “primitives”

– CPUs

» ALUs

» Control units

– Memories

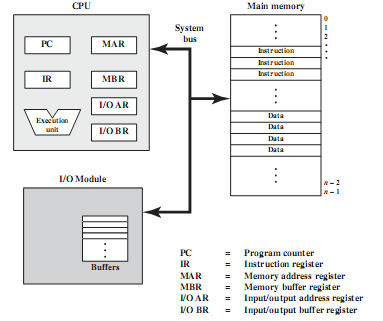
– I/O devices

– Interconnection structures

Knowledge of these components and their operation (interaction) offers insight into system bottlenecks, alternate pathways, magnitude of system failures, and opportunities for performance enhancement

**2.1 Computer Components: Top-Level View**

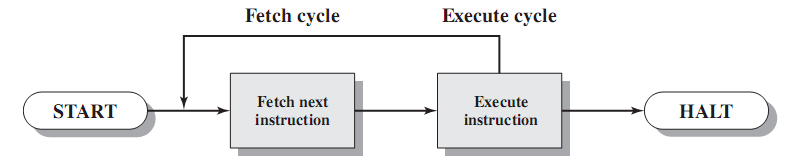
Figure 2.1 illustrates these top-level components and suggests the interactions among them. The CPU exchanges data with memory. For this purpose, it typically makes use of two internal (to the CPU) registers: a memory address register (MAR), which specifies the address in memory for the next read or write, and a memory buffer register (MBR), which contains the data to be written into memory or receives the data read from memory. Similarly, an I/O addresses register (I/OAR) specifies a particular I/O device. An I/O buffer (I/OBR) register is used for the exchange of data between an I/O module and the CPU. A memory module consists of a set of locations, defined by sequentially numbered addresses. Each location contains a binary number that can be interpreted as either an instruction or data. An I/O module transfers data from external devices to CPU and memory, and vice versa. It contains internal buffers for temporarily holding these data until they can be sent on.



**Figure 2.1 Computer Components: Top-Level View**

**2.2 Instruction Fetch and Execute**

At the beginning of each instruction cycle, the processor fetches an instruction from memory. In a typical processor, a register called the program counter (PC) holds the address of the instruction to be fetched next. Unless told otherwise, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence (i.e., the instruction located at the next higher memory address). So, for example, consider a computer in which each instruction occupies one 16-bit word of memory. Assume that the program counter is set to location 300.The processor will next fetch the instruction at location 300. On succeeding instruction cycles, it will fetch instructions from locations 301, 302, 303, and so on. This sequence may be altered, as explained presently.



**Figure 2.2 Basic Instruction Cycle**

The fetched instruction is loaded into a register in the processor known as the instruction register (IR). The instruction contains bits that specify the action the processor is to take. The processor interprets the instruction and performs the required action. In general, these actions fall into four categories:

• Processor-memory: Data may be transferred from processor to memory or from memory to processor.

• Processor-I/O: Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.

• Data processing: The processor may perform some arithmetic or logic operation on data.

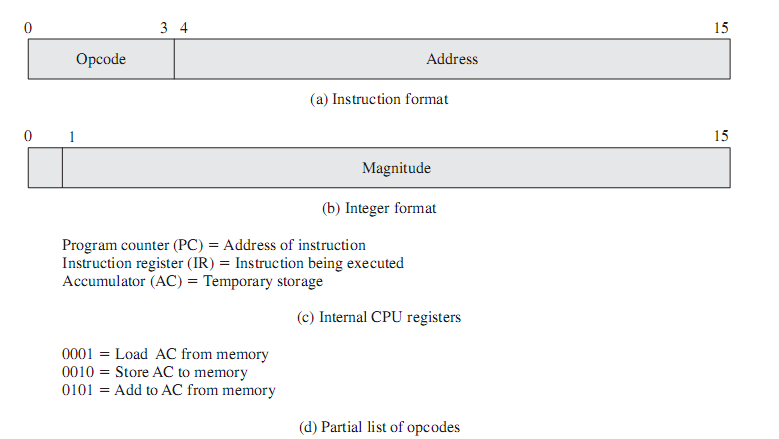
• Control: An instruction may specify that the sequence of execution be altered.

***Example of Instruction cycle:***

Figure 2.3 illustrates a partial program execution, showing the relevant portions of memory and processor registers. Instruction is adding two memory locations M[940] and M[941] then save the result in M[941].

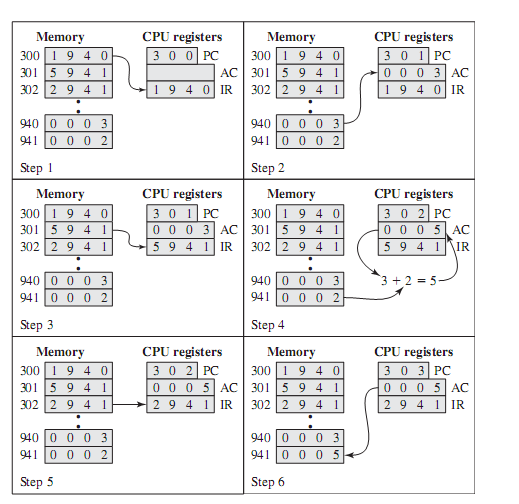
M[941]🡨M[940]+M[941]

**Figure 2.3 the program which executed in 2.5**



**Figure 2.4 Instruction format**

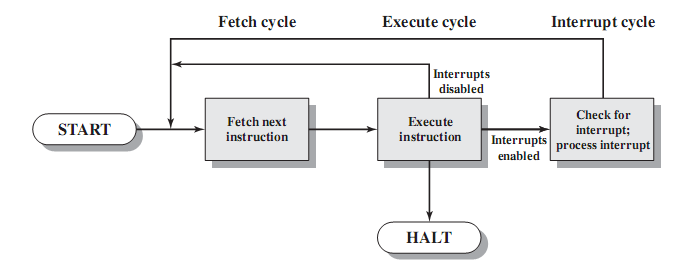
Figure 2.4 (a) Instruction format (opcode 4 bit to decode 16 instructions and address 12bit as the memory word 16 bit). (b) Number format first bit is the sign bit then the number. (c) Used registers. (d) code of instruction in binary (load 1 in Hx, store 2 Hx and add 5 in Hx).



**Figure 2.5 Example of Program Execution (contents of memory and registers in hexadecimal)**

**Interrupts**

Interrupts are provided primarily as a way to improve processing efficiency. For example, most external devices are much slower than the processor. Suppose that the processor is transferring data to a printer using the instruction cycle scheme of Figure 2.6.After each write operation, the processor must pause and remain idle until the printer catches up. The length of this pause may be on the order of many hundreds or even thousands of instruction cycles that do not involve memory. Clearly, this is a very wasteful use of the processor.

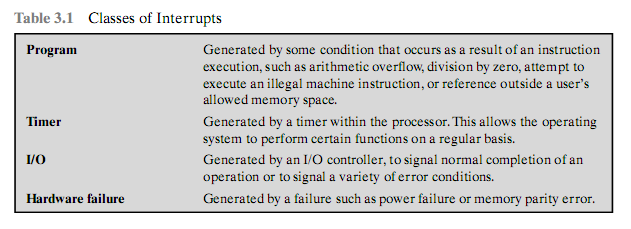


**Figure 2.6 Instruction Cycle with Interrupts**

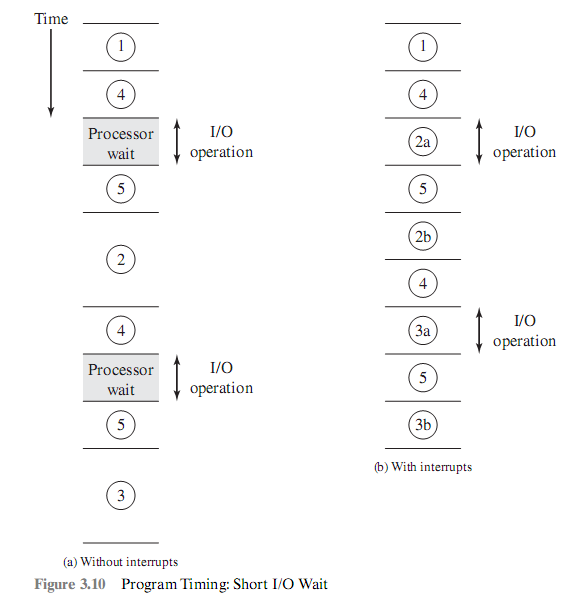
If an interrupt is pending, the processor does the following:

• It suspends execution of the current program being executed and saves its context. This means saving the address of the next instruction to be executed (current contents of the program counter) and any other data relevant to the processor’s current activity.

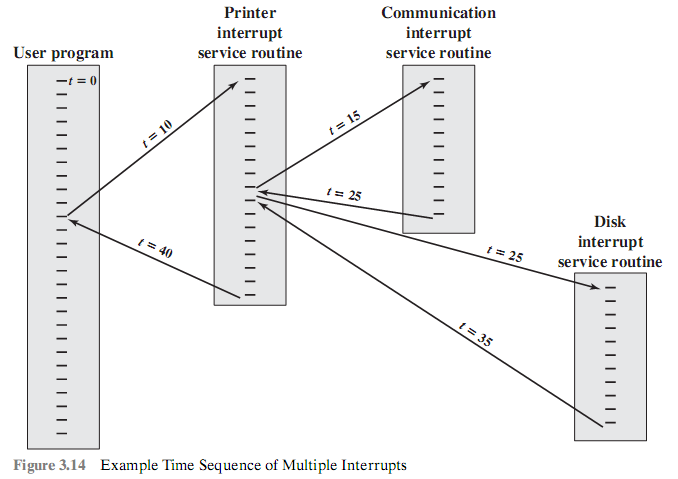
• It sets the program counter to the starting address of an interrupt handler routine.



**Table 2.1 Classes of Interrupts**



**Figure 2.7 Program Timing :Short I/O wait**



**Figure 2.8 Example Time Sequence of Multiple Interrupts**

***Interrupts are processed in an interrupt cycle*** within the overall instruction cycle

– At the end of an instruction cycle (operand storage step), check to see if any interrupts are pending

– If there aren’t any, proceed with the next instruction

– If there are

» Suspend execution of the program and save its “state”

» Jump to the interrupt service routine and resume the “normal” instruction cycle

» When the ISR is completed, restore the state of the program and resume its operation

***Multiple interrupts***

– A typical system can support several to several dozen interrupts

– How should the system respond if more than 1 interrupt occurs at the same time?

» Systems prioritize the various interrupts

» At the start of the interrupt cycle, the highest priority pending interrupt will be serviced

» Remaining interrupt requests will be serviced in turn

– What if an interrupt occurs while an ISR is being executed (a result of a previous interrupt)

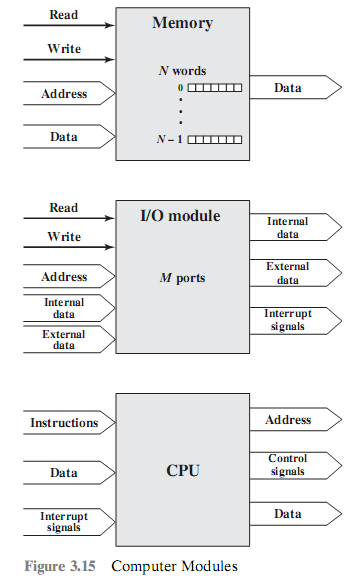
» Ignore the second interrupt (by disabling interrupts) until the ISR completes -- e.g., MC68HC11 microcontroller

» Recognize and service the interrupt only if it has a higher priority than the one currently being serviced -- e.g., 8085

**2.3 Interconnection Structures**

The collection of paths that connect the system modules together form the interconnection structure

**Memory:** Typically, a memory module will consist of N words of equal length. Each word is assigned a unique numerical address (0, 1, . . . ,N – 1).A word of data can be read from or written into the memory. The nature of the operation is indicated by read and writes control signals. The location for the operation is specified by an address.



**Figure 2.9 Computer Modules**

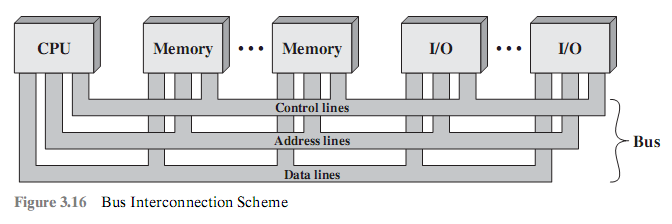
**I/O module**: From an internal (to the computer system) point of view, I/O is functionally similar to memory. There are two operations, read and write. Further, an I/O module may control more than one external device. We can refer to each of the interfaces to an external device as a port and give each a unique address (e.g., 0, 1,...,M– 1). In addition, there are external data paths for the input and output of data with an external device. Finally, an I/O module may be able to send interrupt signals to the processor.

**Processor**: The processor reads in instructions and data, writes out data after processing, and uses control signals to control the overall operation of the system. It also receives interrupt signals. The preceding list defines the data to be exchanged.

The interconnection structure must support the following types of transfers:

* ***Memory to processor***: The processor reads an instruction or a unit of data from memory.
* ***Processor to memory***: The processor writes a unit of data to memory.
* ***I/O to processor***: The processor reads data from an I/O device via an I/O module.
* ***Processor to I/O:*** The processor sends data to the I/O device.
* ***I/O to or from memory***: For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access (DMA).

**2.4 Bus Structure**



**Figure 2.10 Bus Interconnection (three bus system)**

The control lines are used to control the access to and the use of the data and address lines. Because the data and address lines are shared by all components, there must be a means of controlling their use. Control signals transmit both command and timing information among system modules. Timing signals indicate the validity of data and address information. Command signals specify operations to be performed. Typical control lines include

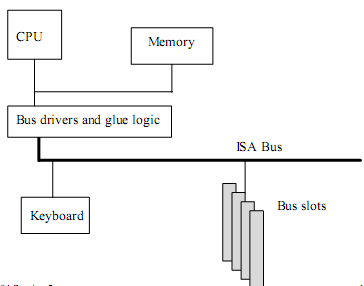
* + Memory write: Causes data on the bus to be written into the addressed location
  + Memory read: Causes data from the addressed location to be placed on the bus
  + I/O write: Causes data on the bus to be output to the addressed I/O port
  + I/O read: Causes data from the addressed I/O port to be placed on the bus
  + Transfer ACK: Indicates that data have been accepted from or placed on the bus
  + Bus request: Indicates that a module needs to gain control of the bus
  + Bus grant: Indicates that a requesting module has been granted control of the bus
  + Interrupt request: Indicates that an interrupt is pending
* Interrupt ACK:Acknowledges that the pending interrupt has been recognized
* Clock: Is used to synchronize operations
* Reset: Initializes all modules

**2.4.1 Multiple-Bus Hierarchies**

If a great number of devices are connected to the bus, performance will suffer. There are two main causes:

1. In general, the more devices attached to the bus, the greater the bus length and hence the greater the propagation delay. This delay determines the time it takes for devices to coordinate the use of the bus. When control of the bus passes from one device to another frequently, these propagation delays can noticeably affect performance.
2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus. This problem can be countered to some extent by increasing the data rate that the bus can carry and by using wider buses (e.g., increasing the data bus from 32 to 64 bits). However, because the data rates generated by attached devices (e.g., graphics and video controllers, network interfaces) are growing rapidly, this is a race that a single bus is ultimately destined to lose.

**2.4.1.1 Examples of PC Buses**

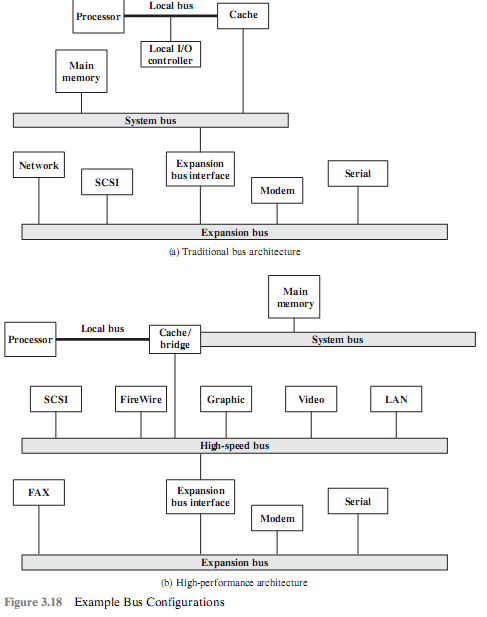


**Figure 2.11 ISA PC Bus**

ISA (Industrial Standard Architecture) as introduced in figure 2.11

– First open system bus architecture for PCs (meaning IBM-type machines)

– 8-bit and 16-bit ISA buses



**Figure 2.12Example Bus Configuration**

– 8-bit bus

» First used in the PC-XT

» 62 pins

» 4.77 MHz clock

» 20 address lines -- 1 Mword memory

» 8 data lines

» 6 interrupt lines, 2 DMA channels

– 16-bit bus

» 8-bit bus was very limiting

» 16-bit bus introduced with the PC-AT and the 80286

» Augmented the existing 8-bit bus’ 62-pin connector with a 36-pin connector

» 8.33 MHz clock

» Total of 24 address lines -- 16 MB address space

» 16 data lines

» 5 more interrupt lines and 4 more DMA channels

**Micro Channel Architecture**

– Introduction of ‘386 and then ‘486 processors put a strain on the performance of the ISA bus

» Slow to pass 32-bit data words in 2 bus operations

– IBM wanted to put ISA to rest and introduced the MCA in their PS/2 series of machines (late80s)

– Offered many improvements over the ISA

» Higher speed

» Bus arbitration

» Automatic configuration

– 16 and 32-bit implementations

– Proprietary architecture that never caught on with users

» Limited peripheral support

» Large installed base of ISA equipment and low-cost replacements

**EISA (Extended-ISA)**

– Introduced in ‘88-89 to provide enhancements to the ISA bus

– 16/32-bit data

– 24/32-bit address

– 8.33 MHz

– Backward compatible with ISA equipment

– Roughly twice the data throughput of ISA

– More interrupts and DMA channels

– Never really caught on -- viewed as a bus for “high-end” machines

**VESA Video Local Bus**

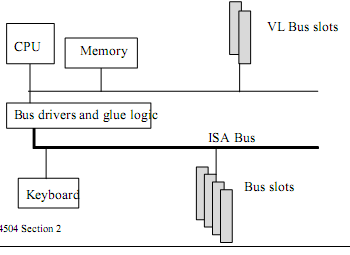
– Video Electronics Standards Assoc.

– Give video and graphics peripherals quick access to main memory

– Implemented in conjunction with ISA or EISA for support of other peripherals (2 sets of connectors on motherboard)

– 32/64-bit data, 24/32-bit address

– Speed related to speed of the processor



**Figure 2.13 VESA Video Local bus**

**PCI**

– Peripheral Component Interface bus Figure 2.14

– Introduced in late ‘92 by Intel and a consortium of manufacturers

» Effectively killed off the VL bus

– Uses a 33 MHz clock, independent from that of the processor

– 64-bit data and address lines (multiplexed)

– Supports up to 16 slots (vs. 2 for the VL bus)

– Systems also include ISA slots for compatibility

– Synchronous bus, centralized bus arbitration

**Futurebus+**

– High-performance asynchronous bus

– Introduced in the late 80s

– Architecture, processor, and technology independent

– Support:

» Parallel and arbitration protocols

» Fault tolerant and high-reliability systems

» Cache-based memory

– Has the potential to supplant other buses because of its flexibility

» Can support data bus widths of up to 256 bits

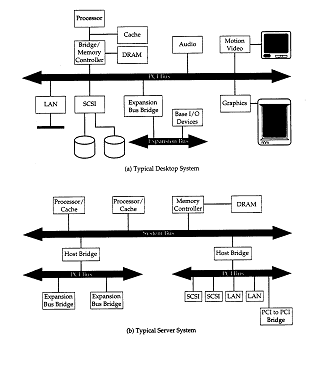
– Flexibility comes at a higher implementation cost than the PCI bus -- so would appeal to a different target user

**2.5 Bus Types**

Bus lines can be separated into ***two generic types: dedicated and multiplexed***. A dedicated bus line is permanently assigned either to one function or to a physical subset of computer components.

Figure 2.12a shows some typical examples of I/O devices that might be attached to the expansion bus .Network connections include local area networks (LANs) such as a ***10-Mbps*** Ethernet and connections to wide area networks (WANs) such as a packet-switching network. SCSI (small computer system interface) is itself a type of bus used to support local disk drives and other peripherals. A serial port could be used to support a printer or scanner.

This traditional bus architecture is reasonably efficient but begins to break down as higher and higher performance is seen in the I/O devices. In response to these growing demands, a common approach taken by industry is to build a high speed bus that is closely integrated with the rest of the system, requiring only a bridge between the processor’s bus and the high-speed bus. This arrangement is sometimes known as mezzanine architecture. Figure 2.11b shows a typical realization of this approach. Again, there is a local bus that connects the processor to a cache controller, which is in turn connected to a system bus that supports main memory. The cache controller is integrated into a bridge, or buffering device, that connects to the high-speed bus. This bus supports connections to high-speed LANs, such as Fast Ethernet at ***100 Mbps***, video and graphics workstation controllers, as well as interface controllers to local peripheral buses, including SCSI and FireWire. The latter is a high-speed bus arrangement specifically designed to support high-capacity I/O devices. Lower-speed devices are still supported off an expansion bus, with an interface buffering traffic between the expansion bus and the high-speed bus.



**Figure 2.14 Example PCI Configurations**

**Assignment 3**

1-What general categories of functions are specified by computer instructions?

2-List and briefly define two approaches to dealing with multiple interrupts.

3-List and briefly define the possible states that define an instruction execution.

4-What is the benefit of using a multiple-bus architecture compared to single-bus architecture?

**PROBLEMS**

3.1 The hypothetical machine of Figure 3.4 also has two I/O instructions:

0011 Load AC from I/O 0111 Store AC to I/O

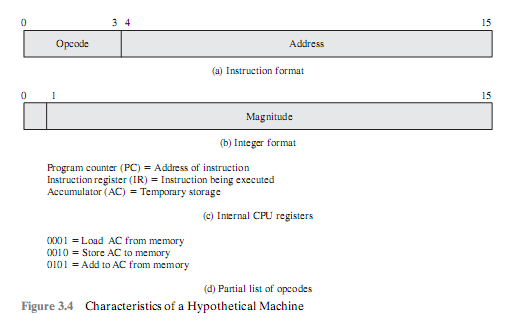
In these cases, the 12-bit address identifies a particular I/O device. Show the program execution (using the format of Figure 3.5) for the following program:

1. Load AC from device 5.

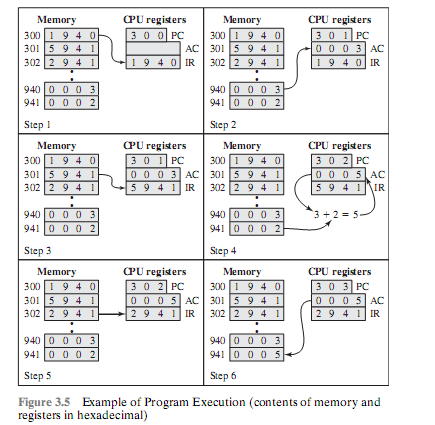
2. Add contents of memory location 940.

3. Store AC to device 6.

Assume that the next value retrieved from device 5 is 3 and that location 940 contains a value of 2.



3.2 The program execution of Figure 3.5 is described in the text using six steps. Expand this description to show the use of the MAR and MBR.



3.3 Consider a hypothetical 32-bit microprocessor having 32-bit instructions composed of two fields: the first byte contains the opcode and the remainder the immediate operand or an operand address.

a. What is the maximum directly addressable memory capacity (in bytes)?

b. Discuss the impact on the system speed if the microprocessor bus has

1. a 32-bit local address bus and a 16-bit local data bus, or

2. a 16-bit local address bus and a 16-bit local data bus.

c. How many bits are needed for the program counter and the instruction register?

3.4 Consider a hypothetical microprocessor generating a 16-bit address (for example, assume that the program counter and the address registers are 16 bits wide) and having a 16-bit data bus.

a. What is the maximum memory address space that the processor can access directly if it is connected to a “16-bit memory”?

b. What is the maximum memory address space that the processor can access directly if it is connected to an “8-bit memory”?

c. What architectural features will allow this microprocessor to access a separate “I/O space”?

d. If an input and an output instruction can specify an 8-bit I/O port number, how many 8-bit I/O ports can the microprocessor support? How many 16-bit I/O ports? Explain.

3.5 Consider a 32-bit microprocessor, with a 16-bit external data bus, driven by an 8-MHz input clock. Assume that this microprocessor has a bus cycle whose minimum duration equals four input clock cycles. What is the maximum data transfer rate across the bus that this microprocessor can sustain, in bytes/s? To increase its performance, would it be better to make its external data bus 32 bits or to double the external clock frequency supplied to the microprocessor? State any other assumptions you make, and explain. Hint: Determine the number of bytes that can be transferred per bus cycle.

3.6 Consider a computer system that contains an I/O module controlling a simple key-board/printer teletype. The following registers are contained in the processor and connected directly to the system bus:

INPR: Input Register, 8 bits

OUTR: Output Register, 8 bits

FGI: Input Flag, 1 bit

FGO: Output Flag, 1 bit

IEN: Interrupt Enable, 1 bit

Keystroke input from the teletype and printer output to the teletype are controlled by the I/O module. The teletype is able to encode an alphanumeric symbol to an 8-bit word and decode an 8-bit word into an alphanumeric symbol.

a. Describe how the processor, using the first four registers listed in this problem, can achieve I/O with the teletype.

b. Describe how the function can be performed more efficiently by also employing IEN.

3.7 Consider two microprocessors having 8- and 16-bit-wide external data buses, respectively. The two processors are identical otherwise and their bus cycles take just as long.

a. Suppose all instructions and operands are two bytes long. By what factor do the maximum data transfer rates differ?

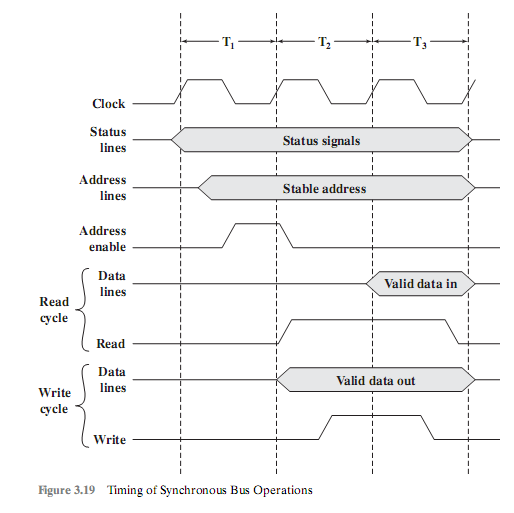
b. Repeat assuming that half of the operands and instructions are one byte long.

3.12 Consider a microprocessor that has a memory read timing as shown in Figure 3.19.

After some analysis, a designer determines that the memory falls short of providing read data on time by about 180 ns.

a. How many wait states (clock cycles) need to be inserted for proper system operation if the bus clocking rate is 8 MHz?

b. To enforce the wait states, a Ready status line is employed. Once the processor has issued a Read command, it must wait until the Ready line is asserted before attempting to read data. At what time interval must we keep the Ready line low in order to force the processor to insert the required number of wait states?



**quiz 3**

1- The main parts of computer are:

a)2 parts b) three parts c) four parts d)five parts

2- The control unit is part of

a)main memory b)processor c)input units d) output units

3- the instruction is transferred to control signal by

a)main memory b)Instruction register c)ALU d) control unit

4- the register stored the address of executed instruction is

a)DR b)AR C)PC d)IR

5-the register store the operant is

a)DR b)AR C)PC d)IR

6- the register store the address to access the main memory is

a)DR b)AR C)PC d)IR

Answer (1-c) (2-b) (3-d) (4-c) (5-a) (6-b)