Processor Level Design

The processor level design is concerned with the design of CPU, memory, BUS, and Input - outputs.

0.1 CPU or Processor level components

The CPU or processor is concerned with the instruction sets, their execution times, program control unit, CPU's communication with external devices.

The Memories are concerned with different technologies, varying cost/performanc
The memory is divided into:

- Main memory: It has properties like: fast speed, comparatively small size, and it is controlled by the CPU;
- Secondary memory: It is slow, large in size, inexpensive, and can communicate via main memory to CPU using serial/parallel access.

The figure 1 shows the inter-connection of CPU and memory via the address, data, and control buses.

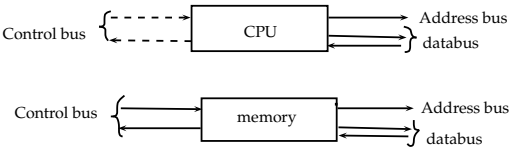

Figure 1: Interconnection of CPU and memory.

Only the the Input/output Devices can communicate with outside world through data transducers, and can be directly controlled by CPU and IOPs (input-output Processors).

The Interconnection networks consists of word level buses, often shared among the number of devices based on priorities, and have asynchronous communication with CPU through handshaking, polling, or synchronous. The communication through bus is in the form of words (many bits in parallel) and this communication is controlled by CPU and IOPs. The Information is transferred generally in words through Interconnection networks i.e., buses, between memory and IO processors. The buses provide dynamic connection between various components.

0.2 Instruction cycle

A machine language program consists sequence of instructions corresponding to the sequence of elementary operations performed by the CPU. These instructions remains in the main memory of the computer. The instructions are fetched one-by-one by the CPU and executed, i.e., the coded operations in the instructions are performed. Fetching and executing an instruction is called "Instruction cycle". Thus,

$$Instruction\ cycle = fetch\ cycle + execute\ cycle.$$

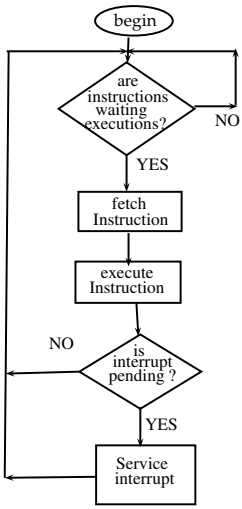

Figure 2: CPU Instruction Cycle.

The sequence of operations in an instruction cycle are given in the figure 4. Each instruction is executed in the form of number of *micro-*

operations. The duration of a micro-operation is one cpu clock-cycle, we call it t_{cpu}.

During the execution of a program, a cpu may be interrupted by the IO devices, to perform some operation with reference to the IO devices, which is generally, a data-transfer operation. To perform such an operations, a cpu need to execute a small program, called as "Interrupt service routine (ISR)". After fetching each instruction, its decoding, and executing, it checks if there is any interrupt pending to be served. If it is yes, the CPU identifies the interrupting device by certain mechanism and executes the corresponding ISR. On completion of the ISR, the CPU resumes the fetch and execute of the next instruction exactly at the place it suspended the operation. Hence, CPU needs to save the address of that instruction some where, generally in the *stack* area of the memory.

The IO devices may be treated as each having its address, independent of the memory locations. In that case the IO is called "isolated IO" or independent IO. If IO address are addresses just like the memory addresses and share the address space from the memory locations, the IO is called "memory mapped IO". The memory-mapped IO has the advantage that all the instructions which can be executed with reference to memory locations, can also be executed with reference to the IO locations. Remember that, unlike the instruction movement between memory and CPU, only the data movement takes place between the CPU and IO devices. Even if there is program file saved in a IO device, like hard disk, the movement of program from disk to memory (called loading) or from memory to disk (called saving), it behaves like data movement only. in addition to data transfer instructions, the IO processors do the data formating. as per the requirements of IO devices, like, into BCD or ASCII conversion, insertion of EOL (end of line), EOF (end of file) characters, and insertion of sectors numbers, track numbers in case of hard-disk drives.

0.3 CPU Architecture

The figure 3 explains the cpu architecture, comprising the ALU (arithmetic and logic unit) and program control unit (CU), along with some essential and most commonly used registers, like Address register and program counter register, data register, instruction register, and accumulator. The accumulator is source of the data (operand) and destination of result in all the arithmetic and logical operations. The program counter (PC) always holds the address of next instruction to be fetched from memory for execution. On fetching it, the instruction first of all land in the instruction register (IR) of the CPU. If an instruction operand

does not hold data but holds address of data, then this address is moved to AR to fetch the data into DR. Finally, contents of DR is added into accumulator register.

As the diagram in figure 3 indicates DR and AR of CPU have interface with the memory. The address of next instruction is maintained in the PC, but to be sent to the memory, it is loaded into the AR.

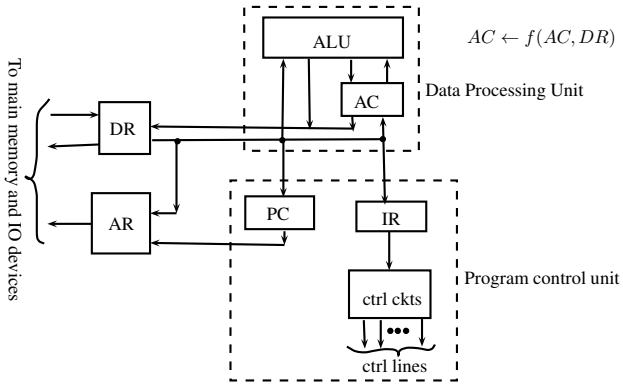

Figure 3: Basic building blocks of CPU.

One *CPU cycle*, designated as t_{cpu} is smallest micro-operation of CPU, and equal to the time of one clock cycle. Thus $1/t_{cpu}$ is maximum clock frequency at which the CPU can be operated.

Execution time for any program is proportional to number of CPU clock cycles used in that. Hence, if there are n CPU cycles spent in a program, then total execution time is $n \times t_{cpu}$ seconds.

On the similar line we define *memory cycle* t_m, which is equal to time spent between address applied to memory and data released by memory. There is a large speed gap between CPU cycle and memory cycle, and $t_m/t_{cpu} \approx 10$.

0.3.1 Instruction Cycle

Figure 4 explain in detail as what are the sequence of operations in a instruction cycle. We assume that there is an "add" instruction as the next instruction. The program counter value is loaded into the address register. Then, memory corresponding to this address is accesses and its content is loaded into the data register. This in fact is "add" instruction along with its operands. Subsequently, the opcode (operation code) of this instruction is moved to instruction register, then it is decoded. By

this time on decoding the opcode, cpu comes to know: (i) size of this instruction, (ii) whether the operands are with the instruction or they are to be fetched, (iii) what operation(s) is to be performed by this instruction.

Since, the size of this instruction has become known (say l), the program counter is incremented by l to fetch the next instruction. If it is found by decoding that it is *add* instruction (*add AC, Adr*), address of data (from address register DR) is sent to address register (Adr), and fetched operand (data) is moved into the data register. Then this data is added into the accumulator to complete the execution of the instructions (see fig. 4).

Had it been a jump instruction, the instruction would have comprised the jump address, hence this address is loaded into the program counter, to fetch the next instruction from that location rather than the next instruction address based on the program counter value.

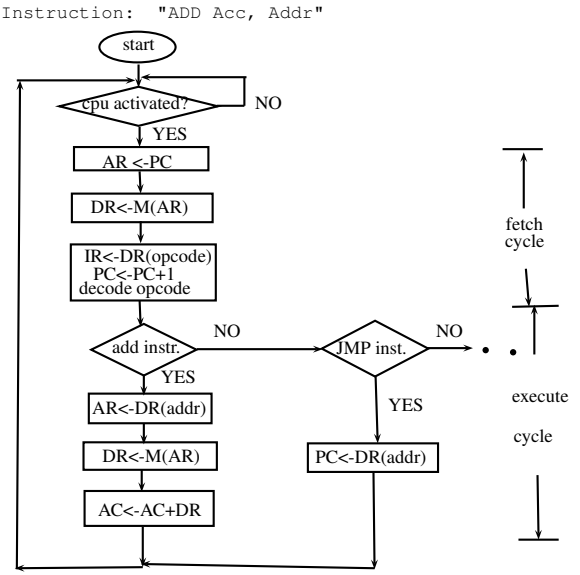

Figure 4: Instruction Cycle.

0.4 Interconnect Structures

The Interconnect structures provide communication path for communication between cpu, memory, and IO devices, so that address, data and

control signals can move long through these interconnect structures. The typical size of address lines (called address bus) is 16, 20, 32, 40, or 48 bits, which has capability to access the memory of $2^{16}, 2^{20}, 2^{32}, 2^{40}$,or$2^{48}$ words. Each word is generally equal to the size of the data bus, and is usually, 8, 12, 16, 24, 32, 40, 64 bits long. However, there is no direct relation between the size of the address bus and size of the word-length. In more powerful processors, the address and data buses are longer, typically 32-bits each.

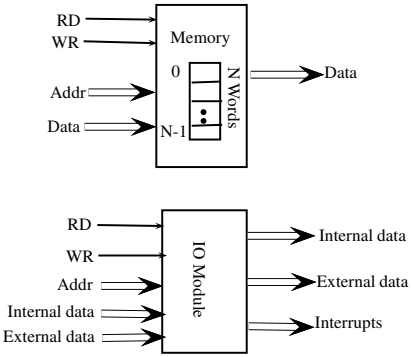

Figure 5: Interconnect Structure Interfaces for memory and IO.

The control signals are: MRD (memory read), MWR (memory write), Clock, for memory and IORD (IO read), IOWR (IO write), Transfer ACK, BUS Request, BUS Grant, Interrupt request, interrupt acknowledgment, Clock, and Reset for the IOPs, as shown in the figure 5. The figure 5 shows commonly used interfaces for memory and IO, which are to be connected to the interface structures.

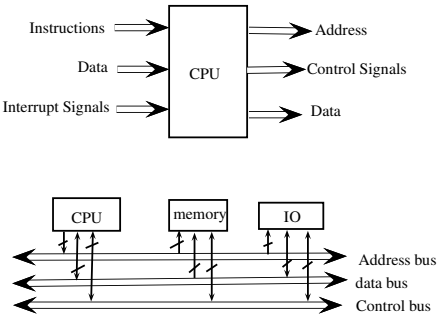

Figure 6: Common bus Interconnect Structure.

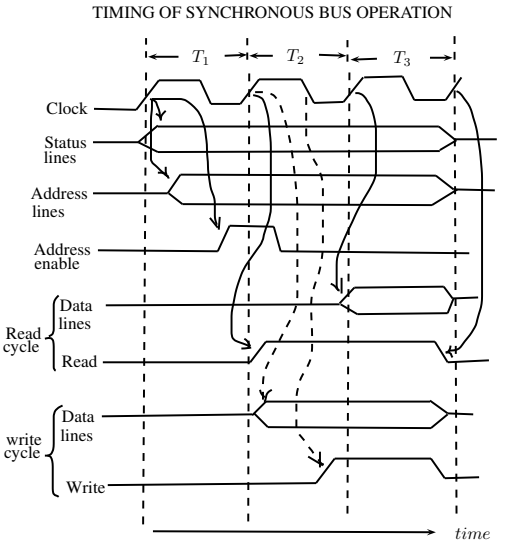

Figure 7: Synchronous bus operation

The figure 6 shows common bus as interface to connect CPU, memory and IO devices through a shared medium. Through this bus medium, only two devices can communicate to each other at any particular time, and other devices have to wait for their turn. There is some other higher level logic which grants access to the common bus, out of many contesting devices for bus, out of cpu, memory, and IO devices.

0.5 Synchronous Bus Operations

The bus communication needs to work synchronous to the clock, so that release and recognition of various signals are carried out with reference to the clock cycles' rising or falling edges. The figure 7 shows the synchronous bus operations which are synchronous to the CPU clock for memory Read/Write cycles. Occurrence of the events, like status lines, read, write, read-enable are determined by a clock. All devices on the bus read the clock line, and all events start at begin of the clock cycle. The status lines indicate whether the present bus cycle is is memory read cycle, memory write cycle, opcode fetch cycle, decode cycle, or execute cycle. The address values are released consequent to the completion of rising edge of the first clock. Immediately after this, the address enable

line is made true by cpu to indicate that memory / IO is now valid, so that memory or IO can pickup this address and return the content of that location. At the rising edge of second clock pulse, the data is read by the cpu (indicated by read signal as true), and later the cpu makes the read line false by lowering it.

To write the data into the memory at the address released by cpu, the cpu issues write line true after the data have been released by the cpu. The write operation is performed at the rising edge of the write control line. After some time, the cpu makes the write signal false by lowering down it.

In Synchronous operation the timing of any transition is known in advance as all the operations are in synchronous to the cpu clock.

The other type of cpu to memory communication is *Asynchronous* communication, which depends on the availability data and readiness of devices to initiate bus transition.

The disadvantage of synchronous bus operations is that cpu and memory are tide down to the cpu clock. So, even if the cpu or memory is fast for reading / writing, the next event shall wait for the completion of current clock cycle.

0.6 Asynchronous Bus Operations

The asynchronous bus operation does not make use of cpu clock, hence it can operate faster; the speed is decided solely by the speed of cpu and memory. The operation is also called in hand-shake mode communication. Occurrence of one event on bus follows the other and CPU is master for data transfer. Synchronous is simple, but tied to clock (less flexible), thus high performance devices cannot contribute. Hence, a mix of slow and fast devices can work together to have advantage of both.

For completion of events, handshake signals are exchanged between the memory and cpu to perform the data communication. Figure 8 shows the this operations.

0.7 Buses types and their Analysis

There are number of different types of buses.

1. Inside CPU (*CPU Bus or Onchip Bus*) is used to connect registers, ALU, and cache.

2. *System bus or Onboard bus* is Between Processor and main memory.

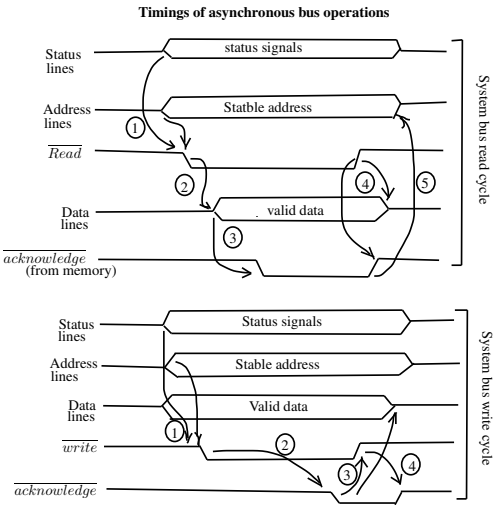

Figure 8: Hand-shake operation of data-transfer between cpu-memory.

3. *Peripheral bus* connects fast peripherals, like graphics card, LAN adapter, with the main memory for high speed data transfer as well as to connect slow devices. The bus connects from dual ported memory.

As the number of devices to be connected increases, there is need of longer bus to accommodate the large number of devices. This results to propagation delays, and coordination problems between devices. Also, more number of devices cause bottleneck for data transfer, and forces to design wider buses (32 to 64 bits) or more than one buses, so that high speed devices can be connected on one bus and slower devices on other bus.

0.7.1 Power Loss

Buses are a significant source of power loss, especially interchip buses, which are often very wide. The standard PC memory bus includes 64 data lines and 32 address lines, and each line requires substantial drivers. A chip can expend 15 percent to 20 percent of its power on these interchip drivers. One approach to limiting this swing is to encode the address lines into a Gray code because address changes, particularly from cache refills, are often sequential, and counting in Gray code switches the least number of signals. Adapting other ideas to this problem is straight forward.

Transmitting the difference between successive address values achieves a result similar to the Gray code. Compressing the information in address lines further reduces them. These techniques are best suited to inter-chip signaling because designers can integrate the encoding into the bus controllers.

Code compression results in significant instruction-memory savings if the system stores the pro- gram in compressed form and decompresses it on the fly, typically on a cache miss. Reducing memory size translates to power savings. It also reduces code overlays - a technique still used in many digital- signal processing (DSP) systems—which are another source of power loss.

0.7.2 Traditional Bus Architecture

The figure 9 shows the traditional bus architecture, connecting all the different types of devices with different communication speeds to be connected together using the same bus.

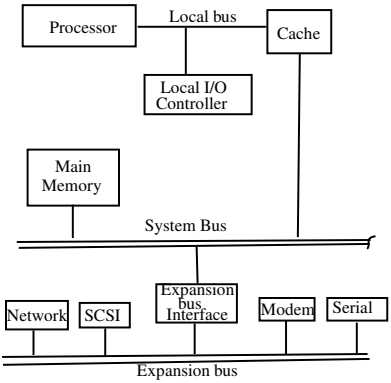

Figure 9: traditional bus architecture.

0.7.3 High performance Bus Architecture

The figure 10 shows the architecture for a high-speed bus.

There is another classifications of buses, as follows:

Bus Types:

1. *Dedicated bus:* There is separate address, data buses (it is most common architecture).

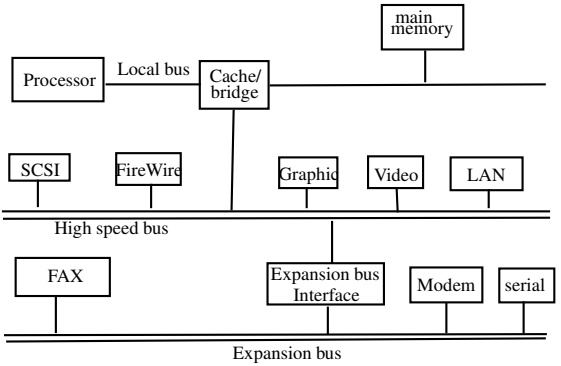

Figure 10: High-speed bus architecture.

2. *Multiplexed:* The first address is sent, after some time data is sent. This is to the number of connections in the bus. This called *time division multiplexed*. It has the disadvantage that data communication speed gets reduced, as the same bus is used for a time for address and for other time for data.

0.7.4 Multiprocessor bus

The figure 11 shows the architecture for multiprocessor bus, where number of processors and memories are connected together through a high-speed shared bus. Only one processor-memory will be using the bus at a time. It has the advantage that many processors can share number of memories together. But, there is problem called *bus contention*, which arises when more than one processors competes to access the bus.

In single-bus system *bus arbitration* is required to resolve the contention. The processor that wants to use the bus, submits a request to "arbitration logic". The arbitration logic decides based on some priority, as which processor should be granted the bus access during a certain period of time. The processor holding the control of bus during that time is called *bus master*. The Passing bus mastership is through *handshaking*, i.e., there is a bust request, and consequently the bus grant.

0.7.5 Bus Arbiter

The arbiter samples the request on rising edge of clock, and a predefined algorithm decides as which master is next to gain access to the bus. Following are the algorithm used:

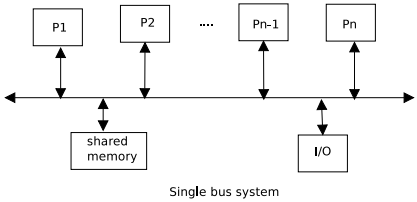

Figure 11: Multiprocessor bus.

Static fixed priority Algorithm

Static fixed priority is a common scheduling mechanism on most common buses, where each master is assigned a fixed priority value. When several masters request simultaneously, the master with the highest priority will be granted. The advantage of this arbitration is its simple implement and small area cost. The static priority based architecture does not provide a means for controlling the fraction of communication bandwidth assigned to a component. If masters with high priority requests frequently, it will lead to the starvation of the ones with low priority.

Round-robin Algorithm

Time division multiplexed (TDM) scheduling divides execution time on the bus into time slots and allocates the time slots to adapters requesting use of the bus. Each time slot can span several physical transactions on the bus. A request for use of the bus might require multiple slot times to perform all required transfers. However, in this architecture, the components are provided access to the communication channel in an interleaved manager, using a two level arbitration protocol.

The first level of arbitration uses a timing wheel where each slot is statically reserved for a unique master. In a single rotation of the wheel, a master that has reserved more than one slot is potentially granted access to the channel multiple times. If the master interface associated with the current slot has an outstanding request, a single word transfer is granted, and the timing wheel is rotated by one slot.

To alleviate the problem of wasted slots, a second level of arbitration is supported. The policy is to keep track of the last master interface to be granted access via the second level of arbitration, and issue a grant to the next requesting master in a round-robin fashion

Figure 12 shows the arbitration logic, which is improved over to the simple round-robin system.

The other algorithms for bus arbitration are:

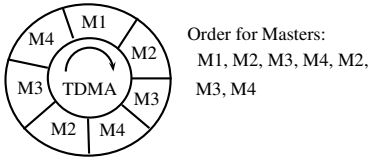

Figure 12: Arbitration logic.

Other algorithms:

- *Random priority* algorithm
- *Equal priority:* when two or more requests are made, there is equal chance of any one request being processed.
- *Static lottery bus arbiter:* Here, a probabilistic arbitration algorithm implemented in a centralized "lottery manager". The probability is fixed here.
- *Dynamic lottery bus arbiter:* The probability is dynamically changing.
- *LRU (Least Recently Used)* Algorithm: The one, given the chance long back shall be served first.

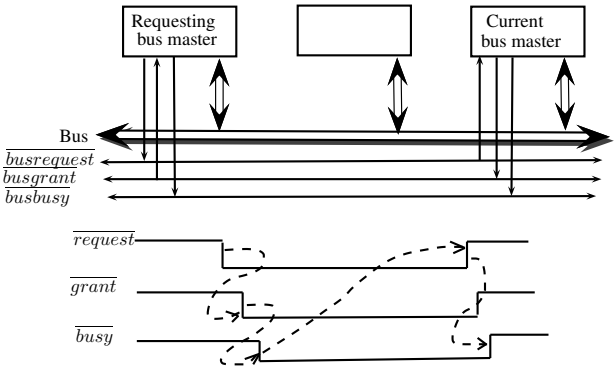

Figure 13: Bus arbitration logic.

Figure 13 shows a bus arbitration logic. The unit requiring the bus control raises a bus-request signal to arbitration-logic who in reply responds by bus-grant signal.

A Bus arbitration logic may be (1) a centralized control to grant the control over the bus, or may be (2) distributed. The centralized, though simple, has problem of single-point failure.

0.7.6 Properties of Buses

There are certain important properties is of buses:

- Bus is the most popular communication pathway among the various components of a computer system.
- Provides cost effective solution
- It provides set of shared communication links
- *Versatile:* So that new components can always be added
- It works on broadcast property
- *Disadvantage:* It is single shared communication link (no backup / standby), Bandwidth (BW) cannot increase with the increase of number of components / units connected. May some time become communication bottleneck
- Point-to-point communication links may be used for large BW requirements (but expensive solution).

0.7.7 Challenges

Following are the Challenges in bus systems which are faced by bus designers:

- Buses are pushed to provide higher data rates. This causes problems of:

  (a) Signal reflection, (b) Cross talk, (c) Skew of signal
- Cross talk? Skew Signal?(same signal reaches to different places at different times)
- *Bus Physics:* Electrical signal travels at finite speed (typically 5 nano sec time for one meter travel in copper wire). Clock frequency cannot be arbitrarily increased due to problem of *signal reflection.*
- May produce standing wave pattern due to reflection.

- Bus can be treated as transmission line. If Z_L is impedance of line, and Z_o of load, then *reflective index* is $\Gamma = (Z_L - Z_o)/(Z_L + Z_o)$. Γ is zero only if $Z_L = Z_0$, which is not possible as Z_0 is input impedance of active components. The non-zero reflective index will cause reflection of signals on the bus.

Exercises

1. Determine the maximum speedup of a single-bus multiprocessor system having N processors if each processor uses the bus for a fraction f of every cycle.

2. In order to enhance the CPU-Memory interactions one solution is to have an exclusive CPU-memory bus, where communication with other sub-systems is exclusively through one of the memories meant only for that purpose.

   Alternately, one may connect Bus Adapters using which other buses which accommodate the sub-systems may be developed. Discuss the advantages and disadvantages each of these systems.

3. When Bus Adapters are used for generating more buses, we can also have one such bus exclusively for memory sub-systems, one fast bus and a slow bus for appropriate types of I/O devices. Sketch such an arrangement and discuss the modes of data transfer between the main processor-memory bus and each of these different backplane buses. Also outline how communication may be established between units across the different backplane buses.

4. Develop the asynchronous interlocked two-way communication protocol between a handshaking master and a slave involved in a write cycle, and sketch the relevant timing waveforms.

5. Assuming appropriate handshake signals, indicate the series of actions involved in priority arbitration sequence, and show how they mesh in with the on-going parallel action of data transfer.

6. A computer has 64-bit instructions, having two fields: first two bytes are for opcode, and the rest is immediate operand or operand address.

   (a) What is maximum addressable memory in bytes?

   (b) How many bits are required for program counter and for IR?

7. A computer has 16-bit address and 16-bit data-lines.

   (a) What is maximum address space?

   (b) What is size of each location in bytes?

   (c) What is size of PC, AR, DR, IR?

8. Two microprocessors have 16- and 32-bit wide external data buses. Other features are same and bus cycles are identical.

   (a) If all instructions and operands are 4 bytes long, by what factor the maximum data transfer rate differ?

   (b) Repeat above, if half of the instructions and opcodes are two-bytes long.

9. For the synchronous read operation, the memory module must place the data on the bus sufficiently ahead of the falling edge of the Read signal to allow for the signal settling. The clock frequency is 20 MHz and Read signal begins to fall in the middle of the second half of T_3.

   (a) Determine the length of the memory read cycle.

   (b) When, at the latest, should memory data be placed on the bus? Allow 10 ns for settling of data lines.

10. Intel 8088 microprocessor has read bus timing like the synchronous read/write discussed in the class, but it requires 4 clock cycles. The valid data is on the bus for an amount of time that extends into the 4th cycle. Let clock is 8 Mhz.

    (a) What is the maximum data transfer rate?

    (b) Repeat above, assume the need to insert one wait state per byte transferred.

11. 8086 uses 16-bit bus that can transfer 2 bytes at a time, provided that lower byte has even address. However, the 8086 allows both even- and odd-aligned word operands. If odd aligned word is referenced, two memory cycles, each consisting of four bus cycles, are required to transfer the word. Consider an instruction on 8086 that involves two 16-bit operands. How long does it take to fetch the operands? Give range of possible answers. The clock is 4 Mhz and no wait state is present.

