Lecture Four
The processor and Memory

INPUT/OUTPUT

Input / Output (I/O) devices provide the means by which the computer system can interact with the outside world. Computers use I/O devices (also called peripheral devices) for two major purposes:

1. To communicate with the outside world and,
2. Store data.

Devices that are used to communicate like, printer, keyboard, modem, Devices that are used to store data like disk drive. I/O devices are connected to the system bus through **I/O controller** (interface) – which acts as interface between the system bus and I/O devices.

There are two main reasons for using I/O controllers

1. I/O devices exhibit different characteristics and if these devices are connected directly, the CPU would have to understand and respond appropriately to each I/O device. This would cause the CPU to spend a lot of time interacting with I/O devices and spend less time executing user programs.
2. The amount of electrical power used to send signals on the system bus is very low. This means that the cable connecting the I/O device has to be very short (a few centimeters at most). I/O controllers typically contain driver hardware to send current over long cable that connects I/O devices. See Figure5.

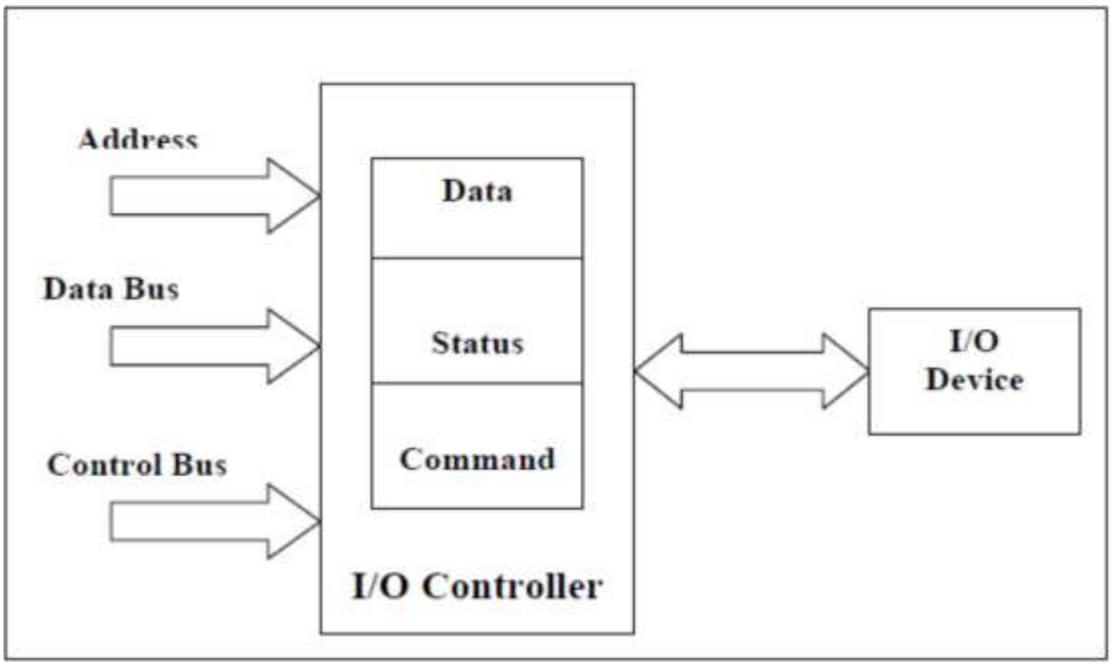
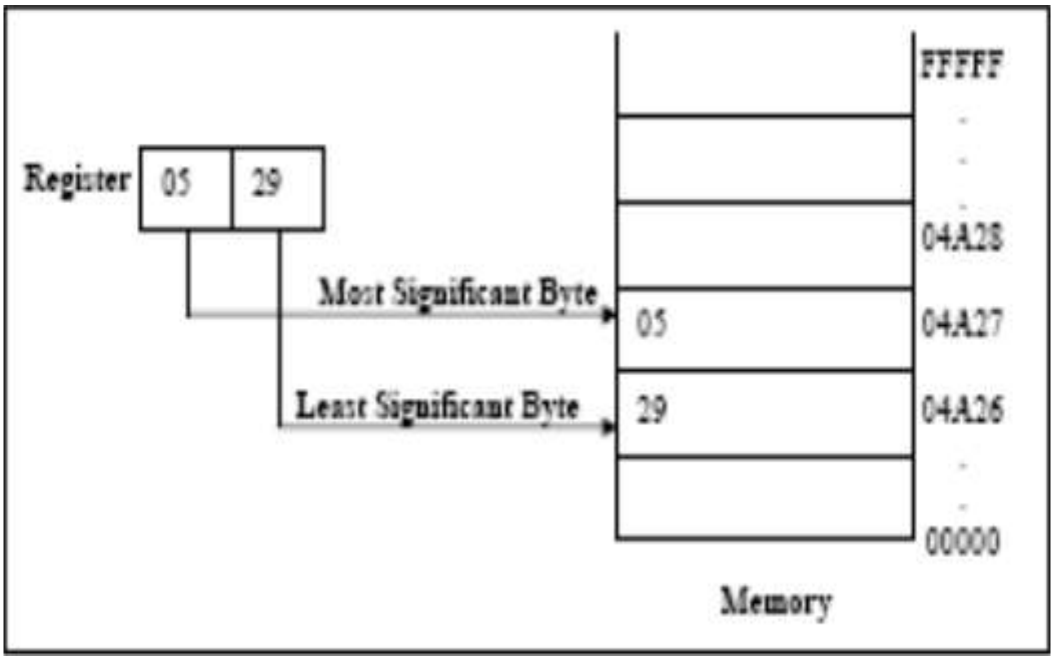


Figure: Block diagram of a generic I/O device interface.

Addressing Data in Memory

Depending on the model, the processor can access one or more bytes of memory at a time. Consider the Hexa value (0529H) which requires two bytes. It consist of high order (most significant) byte 05 and a low order (least significant) byte 29. The processor store the data in memory in reverse byte sequence i.e. the low order byte in the low memory address and the high order byte in the high memory address. For example, the processor transfer the value 0529H from a register into memory addresses 04A26 H and 04A27H like this:

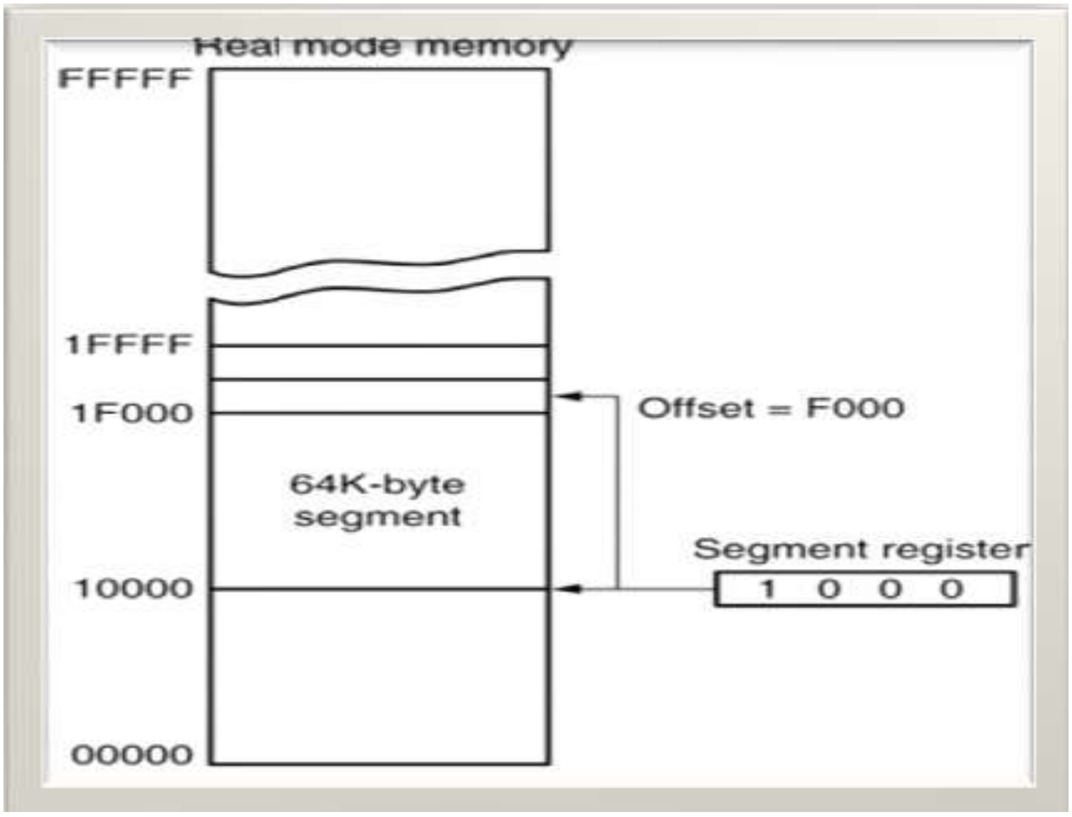


Memory addressing schemes:

1. An Absolute Address, such as 04A26H, is a 20 bit value that directly references a specific location.
2. A Segment Offset Address combines the starting address of a segment with an offset value.

Segment and offset:

Segments are special area defined in a program for containing the code, the data, and the stack. Segment Offset within a program, all memory locations within a segment are relative to the segment starting address. The distance in bytes from the segment address to another location within the segment is expressed as an offset (or displacement). A segment is an area of memory that includes up to 64K bytes as shown in the following figures. The offset address is always added to the segment starting address to locate the data. All real mode memory addresses must consist of a segment address plus an offset address. –Segment address defines the beginning address of any 64K-byte memory segment offset address selects any location within the64K byte memory segment.



The real mode memory-addressing scheme, using a segment address plus an offset. Assembly Language Program consists of three segments:

- **Code segment**: contains the program code (instructions).
- **Data segment**: used to store data (information) to be processed by the program.
- **Stack segment**: used to store information temporarily.

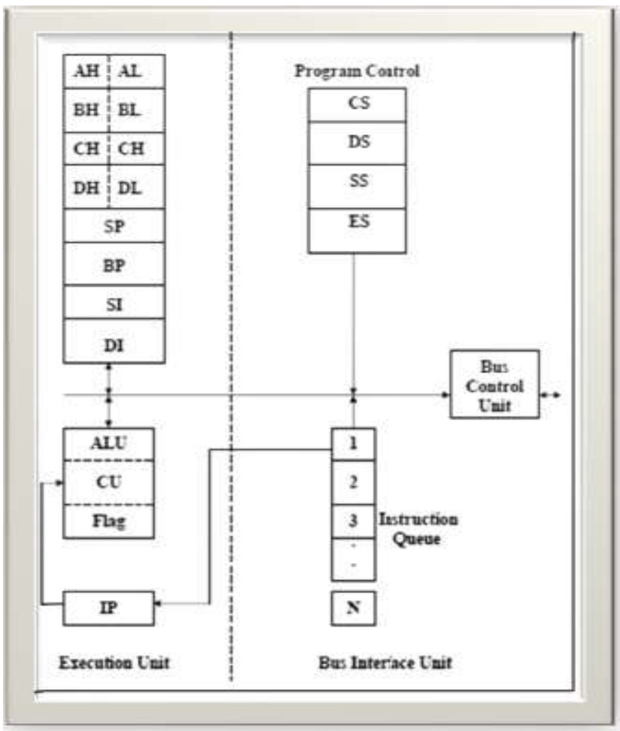
Execution Unit and Bus Interface Unit

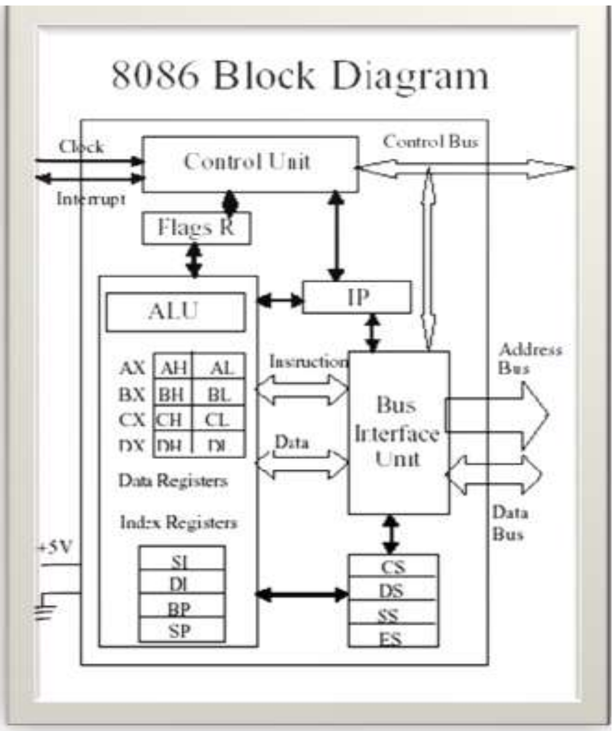
The processor is partitioned into two logical units as shown in figure:

1. Execution Unit (EU) to execute instruction and perform arithmetic and logical operations. The EU contains ALU, CU and number of registers.

2. Bus Interface Unit (BIU) to deliver the instruction and data to EU. The most important function of BIU is to manage the bus control unit, segment registers and instruction queue.

Another function of the BIU is to provide access to instructions, because the instructions for a program that is executing are kept in memory, the BIU must access instruction from memory and place them in an instruction queue, which varies in size depending on the processor. This feature enables the BIU to look ahead and prefetch instructions, so that there is always a queue of instructions ready to execute. The EU and BIU work in parallel, The top instruction is the currently executable one, and while the EU is occupied executing an instruction, the BIU fetch another instruction from memory. This fetching overlaps with execution and speeds up processing.





Execution unit and Bus interface unit.