Interfacing I/O devices in 8086

Input and output ports are interfaced to microprocessor to occupy memory or I/O space of the system. I/O devices interfaced to occupy memory addresses are referred to as *memory mapped I/O*, and I/O devices interfaced to occupy I/O addresses referred to as *I/O mapped I/O*

Memory mapped I/O

In the memory mapped I/O, the I/O devices are treated as memory locations and referenced by addresses of memory locations. The decoding logic for memory mapped I/O uses memory read ($\overline{\text{MEMR}}$) and memory write ($\overline{\text{MEMW}}$) signals. Any instruction that transfers data between processor and memory is used to transfer data between the processor and the I/O devices.

I/O mapped I/O

In the I/O mapped I/O, which is also referred to as isolated I/O, the I/O devices are referenced by 8-bit or 16-bit addresses and the IN or OUT instructions are used for transferring data between the processor and I/O devices. The decoding logic for I/O mapped I/O uses I/O read ($\overline{\text{IOR}}$) and I/O write ($\overline{\text{IOW}}$) signals. Techniques of interfacing I/O devices in I/O mapped I/O are described in the following sections.

Parallel I/O methods

Microprocessor communicates with peripherals in parallel or serial I/O. All the bits of 8-bit or 16-bit data word are transferred on 8 or 16 parallel lines at the same instant in parallel I/O method. The data is sent as stream of bits on a single line in serial I/O

There are various parallel I/O methods

1. Simple I/O
2. Strobe I/O
3. Handshake I/O
4. DMA controlled I/O

Simple I/O

In simple I/O method, the microprocessor transfers data byte assuming that the peripheral is always ready to send or receive byte. Figure below illustrates the simple I/O method. Switches and LEDs connected to input and output ports are always ready to send or receive data bytes to or from the microprocessor. An I/O read operation simply enables the input port and transfers the data information of the switches from the port to data bus. Similarly, I/O write operation enables the output port and transfers the data byte to the port.

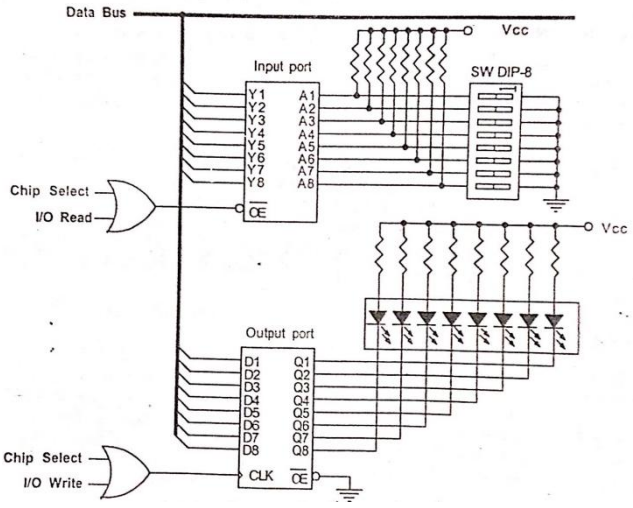


Figure 1 Simple I/O

PREPARED BY
SHAHADAT HUSSAIN PARVEZ

Strobe I/O

Simple I/O technique is not suitable for slower devices because they do not send or receive data at the speed of μP.

In strobe I/O μP transfers data only when the peripheral is ready.

Slow devices generate a control signal called *strobe* along with valid address. The readiness of the peripheral is communicated to the μP in 2 ways, by polling or by interrupt.

Polling: In polling, strobe line is connected to an input port line. μP Keeps reading (polling) and checks for readiness of the peripheral. When μP determines the valid data is present, it effects the data transfer.

Interrupt: In this technique the strobe line is connected to the interrupt input of the μP. μP is interrupted when the peripheral is ready. The μP then takes necessary actions for data transfer

The figure below shows both Polling and Interrupt.

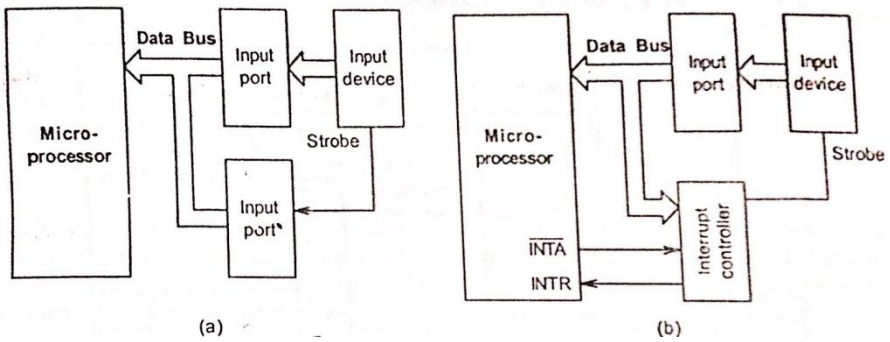


Figure 2 Strobe I/O (a) Polling (b) Interrupt

Handshake I/O

In handshake I/O the μP and the peripheral exchanges a handshake signals to indicate the readiness of the peripheral and synchronize the timing of data transfer. An interface device assists the exchange of data and handshake signal. The figure below illustrates the handshake I/O techniques for input and output operation.

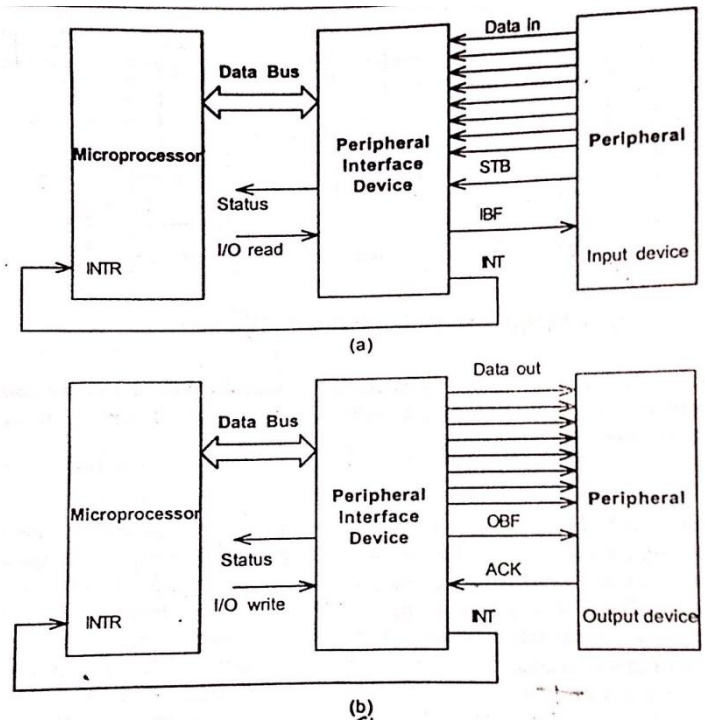


Figure 3 Handshake I/O (a) input operation (b) output operation

PREPARED BY
SHAHADAT HUSSAIN PARVEZ

Input operation: Strobe (STB) and input Buffer full (IBF) are the two handshake signals used for input operation. The peripheral sends the STB signal along with a byte to the interfacing device. The interfacing device in turn sends IBF signal to the peripheral to inform the peripheral should not send next byte till present byte is read by the microprocessor. It further interrupts the microprocessor to inform the presence of a byte with the interfacing device. The microprocessor can also poll the IBF signal and determine the presence of a byte with the device. After the data transfer is complete, the interfacing device informs the peripheral through the IBF signal that the byte transfer is complete and another byte can be sent.

Output operation: Acknowledge (ACK) and output buffer full (OBF) are the two handshake signals used for output operation. The microprocessor writes a byte into the interfacing device, which in turn informs the peripheral like a printer by OBF signal. The peripheral acknowledges the receipt of the byte by ACK signal to the device. After the byte is transferred to the peripheral, the interfacing device interrupts the processor to write the next byte into the device. The microprocessor can also poll the ACK signal and determine the readiness of the peripheral to receive the next byte.

DMA controlled I/O

DMA controlled I/O is used when the peripheral can transfer data at a higher speed than the processor. [Check previous lecture for details]

Programmable peripheral interface, 8255

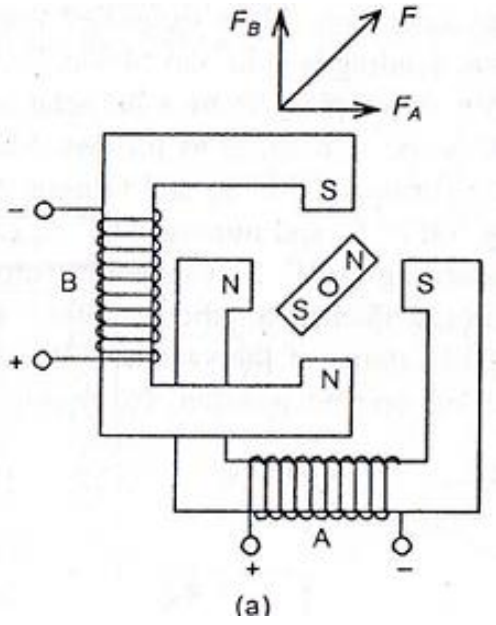
Self study

- Block diagram
- Mode of operation

Stepper motor interface

Construction of a stepper motor

A stepper motor in its simplest form consists of two stators (electromagnets) and a rotor (permanent magnet). The rotor is driven by magnetic fields produced by the stators. A voltage applied to the winding of electromagnet-A produces a magnetic field F_A and a voltage applied to the winding of electromagnet-B produces a magnetic field F_B. F_A and F_B aligns the PM in the direction of the resultant magnetic field F, as shown in Figure 4.



PREPARED BY
SHAHADAT HUSSAIN PARVEZ

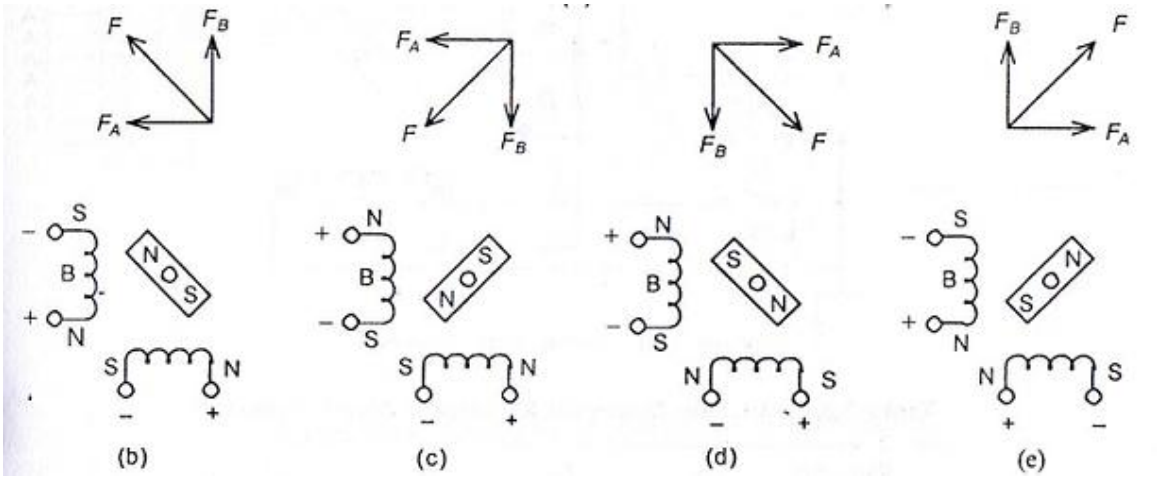


Figure 4 Stepping through action of stepper motor

Operation

The stepping through action of a stepper motor is described in figure 4. While keeping the voltage applied to winding-B as above, reversing the voltage applied to winding-A causes the resultant magnetic field to change its direction and rotates the PM by 90° in anti-clockwise direction. Then, reversing the voltage applied to winding-B causes the PM to move another 90° and again, reversing the voltage applied to A moves the PM by another 90°. Hence by simply switching the polarity of voltages applied to the windings in a proper sequence, the rotor is rotated in steps of 90°. The direction of rotation is reversed by changing the order in which the polarity is reversed In this switching scheme, if power to a coil is switched OFF before reversing the polarity, the rotor is rotated by half step. This increases the number of steps to eight for one full rotation.

Bifilar stepper motor

The above type of stepper motor requires a power supply with +V, 0 V, and -V outputs and a pair of SPDT switches for its operation. More simplified operation is possible with bifilar construction in which each stator winding is split into two and wound as shown in Figure 5. The power to the four windings is supplied from a unipolar source and controlled by four switches. The stepping through action is realized as follows. Suppose that switches S_1 and S_2 are ON and S_3 and S_4 are OFF. Turning 'OFF' S_1, and turning 'ON' S_3 cause the rotor to step through one step. Then, turning 'OFF' S_2 and turning 'ON' S_4 cause the rotor to move another step. Turning 'OFF' S_3 and turning 'ON'S_1 cause the rotor to move yet another step. Table 1 describes the switching sequence for the operation. Logic '1' represents the 'ON' status and '0'represents the 'OFF' status of the switches. It may be observed from the Table 1 that as the bits are rotated left one bit position, the motor is rotated one step clockwise. The step angle of the model is 90° . It just explains the principles of operation of a stepper motor. However, practical stepper motors have more poles on the stator and more tooth on the rotor so that the step angle can be as minimum as 0.9° .

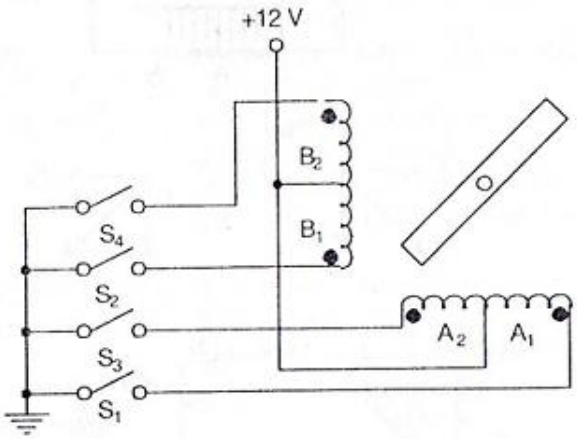


Figure 5 Bifilar stepper motor

PREPARED BY
SHAHADAT HUSSAIN PARVEZ

Table 1 Switching sequence for stepper motor operation

Step No.	S_4	S_3	S_2	S_1
—	0	0	1	1
1	0	1	1	0
2	1	1	0	0
3	1	0	0	1
4	0	0	1	1

Interfacing stepper motor to 8086

Figure 6 shows the circuit for connecting a stepper motor to an -8086 based system. The switching action is controlled through an output port. The decoder connects the output port to the system bus at port address FOH. Since the motor windings draw much more current than the current that can be supplied by an output port, current drivers are used for each winding. The drivers use NPN Darlington pair transistors. The transistor serves the function of a switch. A diode across each coil provides protection to the transistors. The decoder generates device select signal for the following input condition:

$$\overline{Y0} = A7 \times A6 \times A5 \times A4 \times \overline{A3} \times \overline{A2} \times \overline{A1} \times \overline{A0} \times \overline{IOW}$$

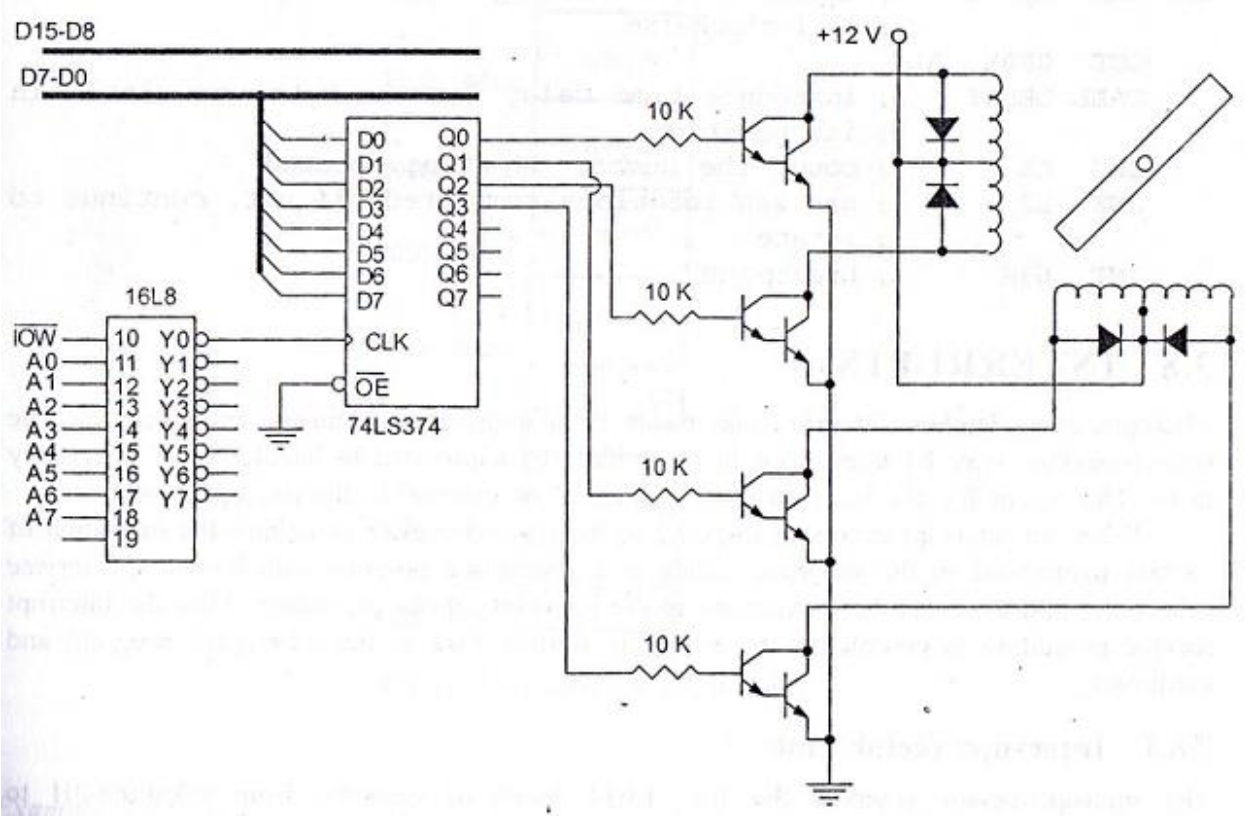


Figure 6 interfacing stepper motor with 8086 system

PREPARED BY
SHAHADAT HUSSAIN PARVEZ

Serial I/O communication

Serial communication can be two types

- Synchronous communication
  - Same clock signal used to synchronize the data transfer
  - Used for high speed
- Asynchronous communication
  - Asynchronous communication is transmission of data, generally without the use of an external clock signal, where data can be transmitted intermittently rather than in a steady stream. Any timing required to recover data from the communication symbols is encoded within the symbols.
  - The most significant aspect of asynchronous communications is that data is not transmitted at regular intervals, thus making possible variable bit rate, and that the transmitter and receiver clock generators do not have to be exactly synchronized all the time. In asynchronous transmission data is sent one byte at a time and each byte is preceded by start bit and stop bit (Data framing).

USART

A universal synchronous and asynchronous receiver-transmitter (USART) is a type of a serial interface device that can be programmed to communicate asynchronously or synchronously.

UART

A universal asynchronous receiver-transmitter (UART) is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable.

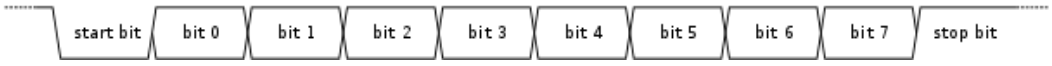
A UART usually contains the following components:

- a clock generator, usually a multiple of the bit rate to allow sampling in the middle of a bit period.
- input and output shift registers
- transmit/receive control
- read/write control logic
- transmit/receive buffers (optional)
- system data bus buffer (optional)
- First-in, first-out (FIFO) buffer memory (optional)
- Signals needed by a third party DMA controller (optional)
- Integrated bus mastering DMA controller (optional)

Data Framing

Data framing is used for enclosing data in a stream of data. The data frame is started with a start bit and ended with a stop bit. There is also parity bit added for checksum.

In most application the least significant bit is transferred first.



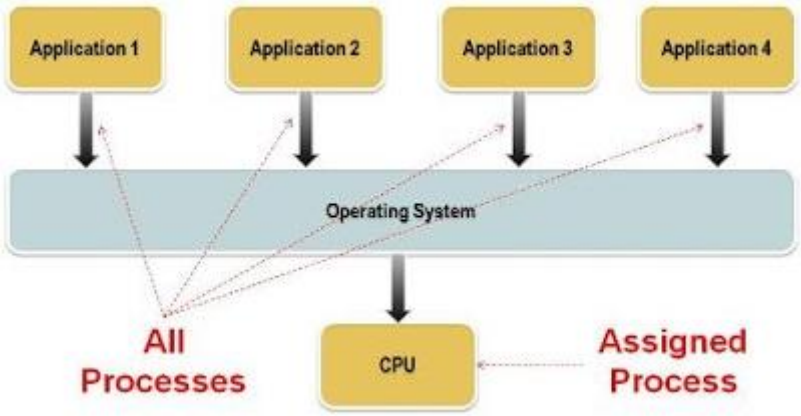
Baud Rate

In digital communications, **symbol rate**, also known as **baud rate** and modulation rate, is the number of symbol changes, waveform changes, or signaling events, across the transmission medium per time unit using a digitally modulated signal or a line code. The symbol rate is measured in baud (Bd) or symbols per second. In the case of a line code, the symbol rate is the pulse rate in pulses per second. Each symbol can represent or convey one or several bits of data. The symbol rate is related to the gross bit rate expressed in bits per second.

PREPARED BY
SHAHADAT HUSSAIN PARVEZ

Multitasking:

In computing, **multitasking** is a method where multiple tasks, also known as processes, are performed during the same period of time. The tasks share common processing resources, such as a CPU and main memory. In the case of a computer with a single CPU, only one task is said to be *running* at any point in time, meaning that the CPU is actively executing instructions for that task. Multitasking solves the problem by scheduling which task may be the one running at any given time, and when another waiting task gets a turn. The act of reassigning a CPU from one task to another one is called a context switch. When context switches occur frequently enough the illusion of parallelism is achieved. Even on computers with more than one CPU (called multiprocessor machines), multitasking allows many more tasks to be run than there are CPUs. The term "multitasking" has become an international term, as the same word is used in many other languages such as German, Italian, Dutch, Danish and Norwegian.



Operating systems may adopt one of many different scheduling strategies, which generally fall into the following categories:

- In *multiprogramming* systems, the running task keeps running until it performs an operation that requires waiting for an external event (e.g. reading from a tape) or until the computer's scheduler forcibly swaps the running task out of the CPU. Multiprogramming systems are designed to maximize CPU usage.
- In *time-sharing* systems, the running task is required to relinquish the CPU, either voluntarily or by an external event such as a hardware interrupt. Time sharing systems are designed to allow several programs to execute apparently simultaneously.
- In *real-time* systems, some waiting tasks are guaranteed to be given the CPU when an external event occurs. Real time systems are designed to control mechanical devices such as industrial robots, which require timely processing.

Multiprocessing:

Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system. The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them. There are many variations on this basic theme, and the definition of multiprocessing can vary with context, mostly as a function of how CPUs are defined (multiple cores on one die, multiple dies in one package, multiple packages in one system unit, etc.).

Multiprocessing sometimes refers to the execution of multiple concurrent software processes in a system as opposed to a single process at any one instant. However, the terms multitasking or multiprogramming are more appropriate to describe this concept, which is implemented mostly in software, whereas multiprocessing is more appropriate to describe the use of multiple hardware CPUs. A system can be both multiprocessing and multiprogramming, only one of the two, or neither of the two of them.

PREPARED BY
SHAHADAT HUSSAIN PARVEZ

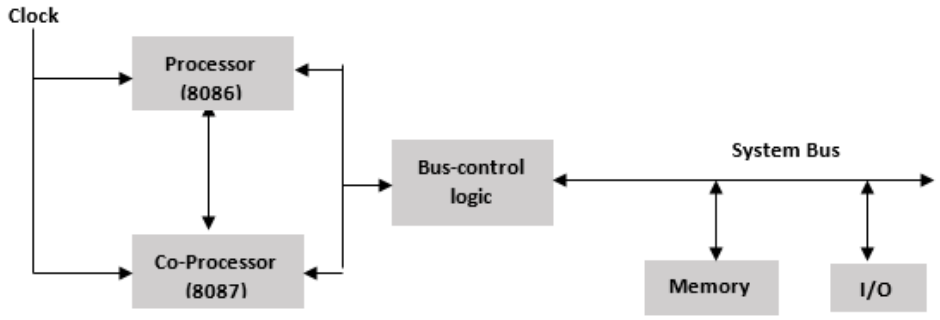
Multiprocessor means a multiple set of processors that executes instructions simultaneously. There are three basic multiprocessor configurations.

- Coprocessor configuration
- Closely coupled configuration
- Loosely coupled configuration

Coprocessor Configuration

A Coprocessor is a specially designed circuit on microprocessor chip which can perform the same task very quickly, which the microprocessor performs. It reduces the work load of the main processor. The coprocessor shares the same memory, IO system, bus, control logic and clock generator. The coprocessor handles specialized tasks like mathematical calculations, graphical display on screen, etc.

The 8086 and 8088 can perform most of the operations but their instruction set is not able to perform complex mathematical operations, so in these cases the microprocessor requires the math coprocessor like Intel 8087 math coprocessor, which can easily perform these operations very quickly.



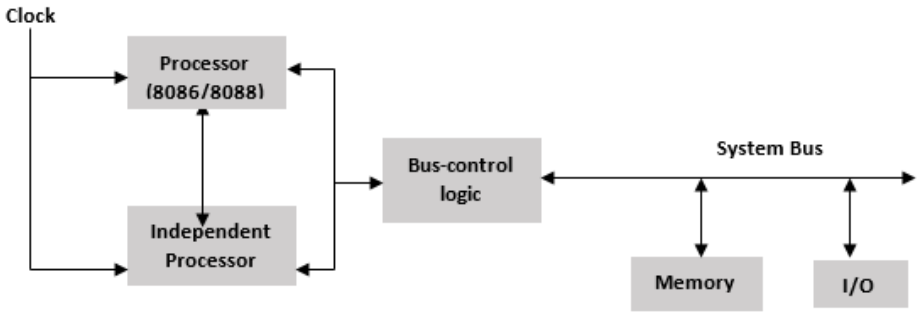
How is the coprocessor and the processor connected?

- The coprocessor and the processor is connected via TEST, RQ-/GT- and QS_0 & QS_1 signals.
- The TEST signal is connected to BUSY pin of coprocessor and the remaining 3 pins are connected to the coprocessor's 3 pins of the same name.
- TEST signal takes care of the coprocessor's activity, i.e. the coprocessor is busy or idle.
- The RT-/GT-is used for bus arbitration.
- The coprocessor uses QS_0 & QS_1 to track the status of the queue of the host processor.

Closely Coupled Configuration

Closely coupled configuration is similar to the coprocessor configuration, i.e. both share the same memory, I/O system bus, control logic, and control generator with the host processor. However, the coprocessor and the host processor fetches and executes their own instructions. The system bus is controlled by the coprocessor and the host processor independently.

Block Diagram of Closely Coupled Configuration



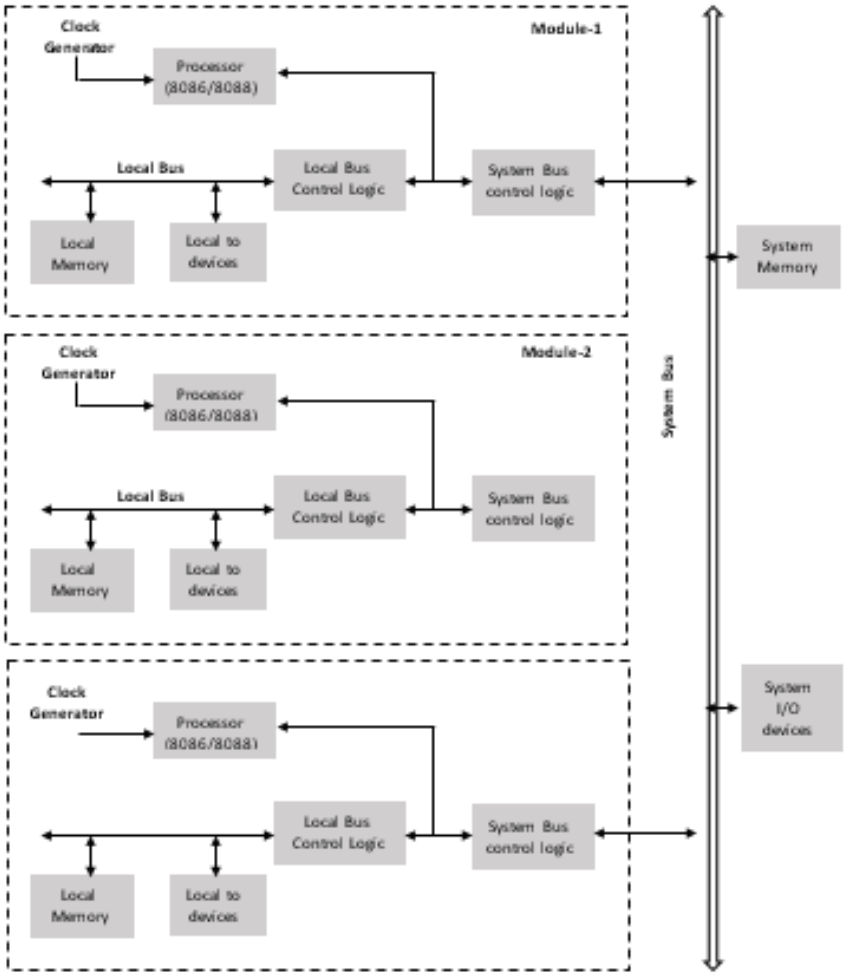
How is the processor and the independent processor connected?

- Communication between the host and the independent processor is done through memory space.
- None of the instructions are used for communication, like WAIT, ESC, etc.
- The host processor manages the memory and wakes up the independent processor by sending commands to one of its ports.
- Then the independent processor accesses the memory to execute the task.
- After completion of the task, it sends an acknowledgement to the host processor by using the status signal or an interrupt request.

Loosely Coupled Configuration

Loosely coupled configuration consists of the number of modules of the microprocessor based systems, which are connected through a common system bus. Each module consists of their own clock generator, memory, I/O devices and are connected through a local bus.

Block Diagram of Loosely Coupled Configuration



Advantages

- Having more than one processor results in increased efficiency.
- Each of the processors have their own local bus to access the local memory/I/O devices. This makes it easy to achieve parallel processing.
- The system structure is flexible, i.e. the failure of one module doesn't affect the whole system failure; faulty module can be replaced later.

Arithmetic Co processor

8087 numeric data processor is also known as **Math co-processor, Numeric processor extension** and **Floating point unit**. It was the first math coprocessor designed by Intel to pair with 8086/8088 resulting in easier and faster calculation.
Once the instructions are identified by the 8086/8088 processor, then it is allotted to the 8087 co-processor for further execution.

The data types supported by 8087 are −

- Binary Integers
- Packed decimal numbers
- Real numbers
- Temporary real format

The most prominent features of 8087 numeric data processor are as follows −

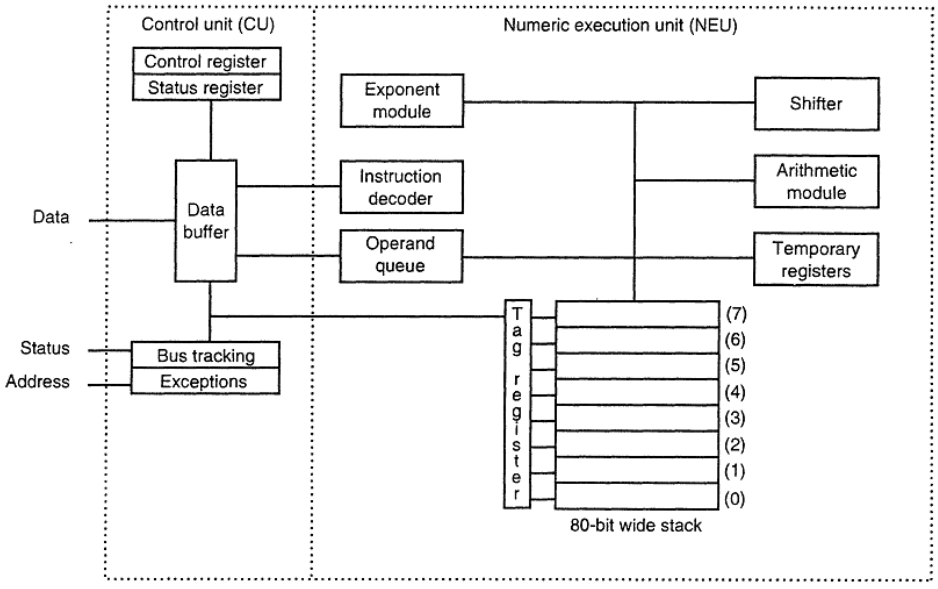
- It supports data of type integer, float, and real types ranging from 2-10 bytes.
- The processing speed is so high that it can calculate multiplication of two 64-bits real numbers in ~27 µs and can also calculate square-root in ~35 µs.
- It follows IEEE floating point standards.

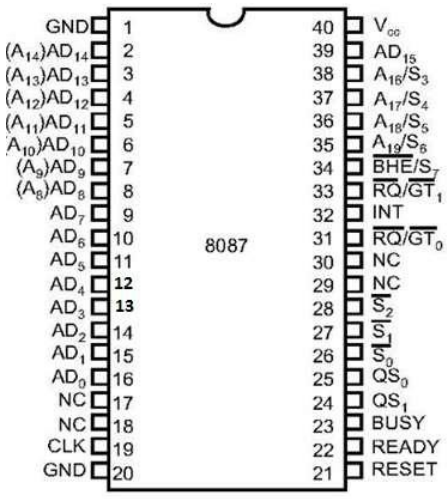
8087 Architecture

8087 Architecture is divided into two groups, i.e., **Control Unit** (CU) and **Numeric Extension Unit** (NEU).

- The **control unit** handles all the communication between the processor and the memory such as it receives and decodes instructions, reads and writes memory operands, maintains parallel queue, etc. All the coprocessor instructions are ESC instructions, i.e., they start with 'F', the coprocessor only executes the ESC instructions while other instructions are executed by the microprocessor.
- The **numeric extension unit** handles all the numeric processor instructions like arithmetic, logical, transcendental, and data transfer instructions. It has 8 register stack, which holds the operands for instructions and their results.

The architecture and pin diagram of 8087 coprocessor is as follows −





The following list provides the Pin Description of 8087 −

- **AD_0 – AD_{15}** − These are the time multiplexed address/data lines, which carry addresses during the first clock cycle and data from the second clock cycle onwards.

PREPARED BY
SHAHADAT HUSSAIN PARVEZ

- **A_{19} / S_6 – A_{16}/S** − These lines are the time multiplexed address/status lines. It functions in a similar way to the corresponding pins of 8086. The S_6, S_4 and S3 are permanently high, while the S_5 is permanently low.
- **BHE/S_7** − During the first clock cycle, the $\overline{BHE}$/S_7 is used to enable data on to the higher byte of the 8086 data bus and after that works as status line S_7.
- **QS_1, QS_0** − These are queue status input signals which provides the status of instruction queue, their conditions as shown in the following table −

QS_0	QS_1	Status
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty the queue
1	1	Subsequent byte from the queue

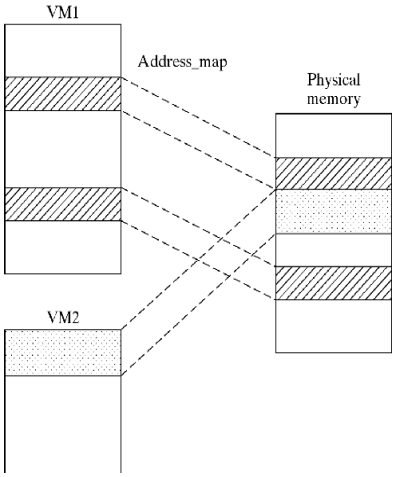
- **INT** − It is an interrupt signal, which changes to high when an unmasked exception has been received during the execution.
- **BUSY** − It is an output signal, when it is high it indicates a busy state to the CPU.
- **READY** − It is an input signal used to inform the coprocessor whether the bus is ready to receive data or not.
- **RESET** − It is an input signal used to reject the internal activities of the coprocessor and prepare it for further execution whenever required by the CPU.
- **CLK** − The CLK input provides the basic timings for the processor operation.
- **VCC** − It is a power supply signal, which requires +5V supply for the operation of the circuit.
- **S_0, S_1, S_2** − These are the status signals that provide the status of the operation which is used by the Bus Controller 8087 to generate memory and I/O control signals. These signals are active during the fourth clock cycle.

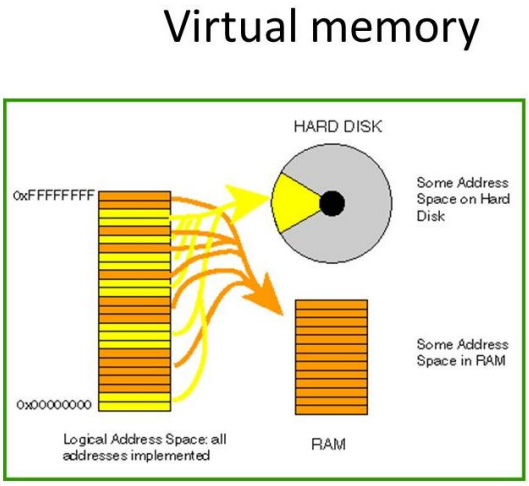
S_2	S_1	S_0	Queue Status
0	X	X	Unused
1	0	0	Unused
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

- **RQ/GT_1 & RQ/GT_0** − These are the **Request/Grant** signals used by the 8087 processors to gain control of the bus from the host processor 8086/8088 for operand transfers.

PREPARED BY
SHAHADAT HUSSAIN PARVEZ

Virtual memory:



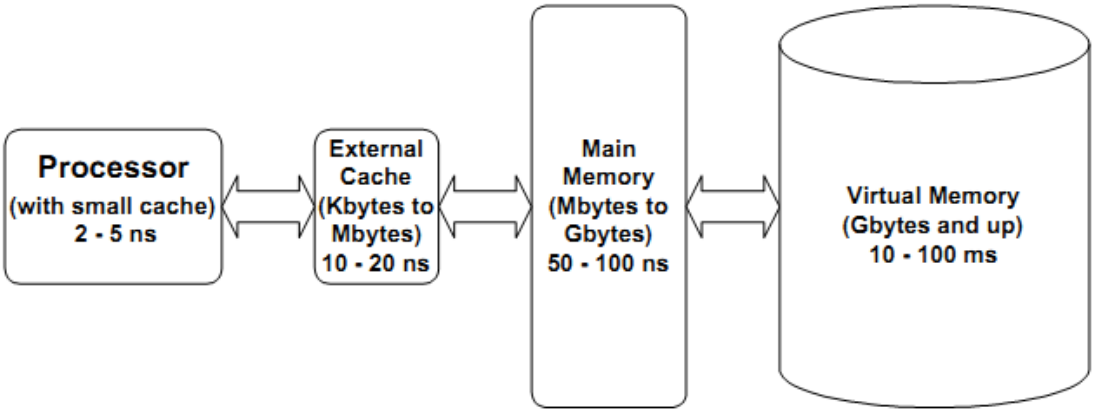


Virtual memory systems

- Virtual memory is another level in the memory hierarchy.
- All general-purpose processors today employ virtual memory.
- For each process, the system creates the **illusion** of **large contiguous** memory space(s)
- Relevant portions of Virtual Memory (VM) are loaded **automatically** and **transparently**
- **Address Map** translates Virtual Addresses to Physical Addresses

In our quest to make the fastest possible computer system, we sometimes must acquiesce to the programmers and add function that slows the system down.

- We add cache memory to a computer system solely for the purpose of speeding up the processing rate.
- We add virtual memory to a computer system as a convenience for the programmer. Virtual memory slows down the processing rate.



Multiple address spaces

Virtual memory allows multiple large linear address spaces to concurrently exist in a single real memory by breaking each address space into fixed sized pages, and keeping only a subset of the pages in real memory. The remaining pages of each virtual memory space may or may not be defined and allocated space in the system. If they are defined, they are allocated space in a backing store, usually fixed disk.

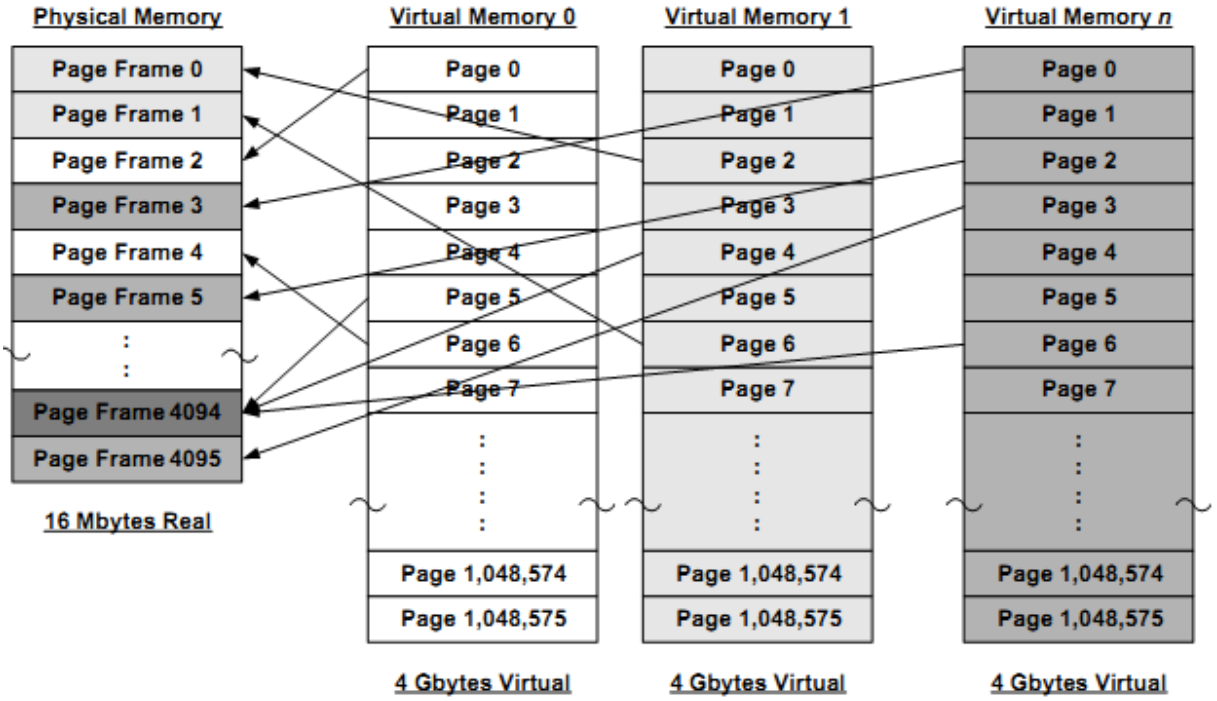
The job of the computer system architect is to minimize the overhead imposed by the virtual memory design. The subject of virtual memory is covered in detail in CSE 503, Computer Architecture.

What's the problem with Virtual Memory?

- Cache memory deals with physical memory addresses.
- The processor actually generates virtual memory addresses.
- Before we can begin our search of the cache, we must translate the virtual address into a physical address.

Example Virtual Memory System

- 16 Megabytes of real memory (24-bit address)
- 4 Gigabytes of virtual memory per address space (32-bit address).
- 4K page size.



Mapping of the virtual addresses to physical addresses

Most virtual memory systems use a two-level lookup to translate virtual addresses into real addresses.

The virtual address is divided into:

- A segment table entry (STE) number
- A page table entry (PTE) number
- A byte displacement into the page

The system uses a single segment table pointer to locate (in real memory) the segment table that is currently in use.

Each segment table entry contains:

- Whether or not the corresponding page table is valid (defined).
- Whether or not it is resident in real memory or paged out.
- The address of the page table (real or disk) if it is valid.

Each page table entry contains:

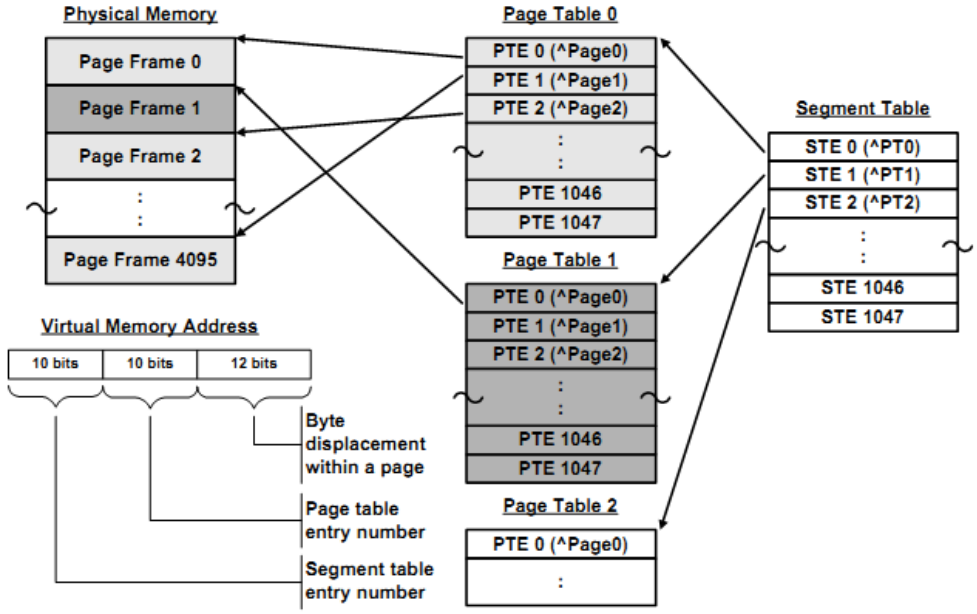
- Whether or not the page is valid (defined).
- Whether or not it is resident in real memory or paged out.
- The address of the page (real or disk) if it is valid.

PREPARED BY
SHAHADAT HUSSAIN PARVEZ

Example virtual address translation

The following diagram shows the translation of a 32-bit virtual address into a 24-bit real address using a two-level lookup system.

- 1K possible page tables
- 1K pages per page table
- 4K page size



Steps to translate virtual to real

- Index into the segment table using the segment table address and the STE number in the virtual address.
- If the page table is not valid, generate an address exception interrupt.
- If the page table is valid but paged out, generate a page fault interrupt.
- Index into the corresponding page table using the page table address in the STE and the PTE number in the virtual address.
- If the page is not valid, generate an address exception interrupt.
- If the page is valid but paged out, generate a page fault interrupt.
- Concatenate the page frame address in the PTE and the byte displacement in the virtual address to form the real address.

We are now ready to see if the data at the real address is in the cache!

BUT, we just had to access the system memory TWICE in order to translate the virtual address into the real address so we can get the data the processor really wants.

Every access to memory is now three accesses to memory:

- Access memory to get the segment table entry.
- Access memory to get the page table entry.
- Access memory to read or write the data requested by the processor.

This does not bode well for the performance of our computer system.