Microprocessors and Interfacing (A1423)
Unit – I Introduction

Department of Electronics and Communication Engineering
VARDHAMAN COLLEGE OF ENGINEERING
Shamshabad, Hyderabad – 501218, India.

Pre-requisites
Digital Logic Design (A1404)
Computer Architecture and Organization (A1509)
Text Books


Reference Books

Unit - I

- Number Systems and Digital Logic Review
- Overview of Microcomputer Structure and Operation
- Evolution of Microprocessors
- 8086 Microprocessor Architecture
- Register Organization
- 8086 Pin Diagram and Signal Description
- 8086 Minimum Mode and Its Timing Diagram
- 8086 Maximum Mode and Its Timing Diagram
- Addressing Modes

Microprocessor Interface

[Diagram showing microprocessor interface with components like address decoder, memory, interface device, and I/O ports connected through address bus, data bus, and control lines.]
Evolution of Microprocessors

- Based on the Word Length (Number of binary bits in data processing)
  - 4-bit, 8-bit, 16-bit, 32-bit, and 64-bit
- 4004
- 8008
- 8080
- 8085
- 8086
- 80286
- 80386
- Pentium
- Pentium Pro

**INTEL 4004**

- Built in 1971
- 4 Bit Micro-processor
- Initial clock speed 108 KHz
- No of transistors used 2300
- Developed on P-MOS technology
INTEL 8008

1) Built in 1972
2) 8 bit Micro-processor
3) Initial clock speed of 500-800 KHz
4) No of transistors used were 3500
5) Also developed on PMOS technology

INTEL 8080

1) Built in 1974
2) 8 Bit Microprocessor
3) Initial clock speed 500-800 KHz
4) No of transistors used 4500
5) Developed using NMOS Technology
6) Drawbacks: It required 3 power supplies
INTEL 8085
1) Built in 1977
2) 5 in 8085 means +5 volt supply
3) Initial clock speed of about 3MH
4) Approx. 6500 transistors were used
5) 8 Bit μ-Processor using NMOS technology

SPECIAL FEATURES
1) Extension to support new interrupts :: Maskable (RST 7.5,RST 6.5,RST 5.5) Non-Maskable (TRAP) and externally serviced Interrupt (INTR).
2) 8085 can itself drive piezoelectric crystal directly connected to it.
3) Through 8 bit Microprocessor but it can also load 16 bit data (Through pairs BC DE HL).

INTEL 8086
1) Built in 1979
2) 16 bit Microprocessor
3) It has 20 bit address bus and 1 MB addressing space
4) Initial clock speed of 5 MHz,8 MHz and 10 MHz
5) No of transistors used were 29,000

SPECIAL FEATURES
1) Memory divided into odd and even banks
2) Can read upto 16 bits of data in one cycle
### INTEL 80286 Processor
1. It is a 16 bit x86 processor
2. Provides Memory management & protection
3. Initial clock speed 8 MHz
4. 24 bit address bus gives the capacity to access 16 MB storage
5. About 134,000 transistors were used

### INTEL 80386 Processor
1. It is a 32 bit x86 processor
2. It has 32 bit address bus and 32 bit data bus
3. Clock speeds between 12-40 MHz
4. It can access upto 4GB of physical memory
5. About 275,000 transistors were used

### Intel Pentium Processor:
1. Built in 1993
2. Clock speed 66 MHz
3. Approx. 31 Lakhs transistors were used
4. 112 million commands per second could be executed per second.

### Pentium Series Advancement:
(Pentium 2 to Pentium 4 processor)
1. Clock speed was increased from 66 MHz to 1.7 GHz
2. No of transistors were also increased from 31 Lakhs to 5.5 Crore
INTEL CORE 2 DUO AND DUAL CORE PROCESSORS:-
1) Built in 2006 and 2007 respectively
2) Clock speed varies from 1.6 GHz to 2.6 Ghz
3) In built cache memory of 2MB to 6 MB
4) Both are 64 bit microprocessors
5) Both were mainly media centred and provided HD display quality
1) Core are more powerful variants of the same processors sold as Celeron and Pentium
2) They are made mainly for multitasking
3) These processors were made available to public from January 2010
4) Clock speed varies from 1.7GHz (slowest) and 3.5GHz (fastest)

8086 Microprocessor Architecture
- The architecture of 8086 microprocessor supports/ provides
  - a 16-bit ALU
  - a set of 16-bit registers
  - segmented memory addressing capability
  - a rich instruction set
  - powerful interrupt structure
  - fetched instruction queue for overlapped fetching and execution etc.
To improve the performance by implementing the parallel processing concept the CPU of the 8086 is divided into two independent sections.

They are
- Bus Interface Unit (BIU) and
- Execution Unit (EU)

The BIU sends out addresses, fetches instructions, read data from ports and memory and writes data to ports and memory, i.e., the BIU handles all transfers of data and addresses on the buses required by the Execution Unit whereas the Execution Unit tells the BIU where to fetch instructions or data from, decodes the instructions and executes the instructions.

The BIU contains
- the circuit for physical address calculations
- a predecoding instruction byte queue (6 bytes long)
- four 16-bit segment registers (ES, CS, SS, DS)
- 16-bit instruction pointer (IP)

The EU contains
- control circuitry, instruction decoder and ALU
- 16-bit flag registers
- four 16-bit general purpose registers (AX, BX, CX, DX)
- 16-bit pointer registers (SP, BP) and
- 16-bit index registers (SI, DI)
Execution Unit:

Control Circuitry, Instruction Decoder and ALU:

- The EU contains control circuitry which directs internal operations.
- A decoder in the EU translates instructions fetched from memory into a series of actions which the EU carries out.
- The EU has a 16-bit arithmetic logic unit which can add, subtract, AND, OR, XOR, increment, decrement, complement, or shift binary numbers.
8086 Microprocessor Architecture
Cont…

Flag Registers:
- The 8086 16-bit flag register contents indicate the results of computations in the ALU. It also contains some flag bits to control the CPU operations.
- A flag is a flip-flop that indicates some condition produced by the execution of an instruction or controls certain operations of the EU.

General Purpose Registers:
- The registers AX, BX, CX and DX are the general purpose 16-bit registers.

8086 Microprocessor Architecture
Cont…

Pointer Registers:
- The 16-bit pointer registers usually contains offset address within the particular segments.

Index Registers:
- The 16-bit index registers are used as general purpose registers as well as for offset storage.
Bus Interface Unit:

The Queue:

- While the EU is decoding an instruction or executing an instruction which does not require use of the buses, the BIU fetches up to six instruction bytes for the following instructions.
- The BIU stores these prefetched bytes in a first-in-first-out register set called a queue.
- When the EU is ready for its next instruction, it simply reads the instruction byte(s) for the instruction from the queue in the BIU.
- This is much faster than sending out an address to the system memory and waiting for memory to send back the next instruction byte or bytes.
- Fetching the next instruction while the current instruction executes is called pipelining.

Segment Registers:

- The 8086 BIU sends out 20-bit addresses, so it can address any of $2^{20}$ or 1,048,576 bytes in memory. However, at any given time the 8086 works with only four 65,536 byte (64Kbyte) segments within this 1,048,576 byte (1 Mbyte) range.
- Four segment registers in the BIU are used to hold the upper 16 bits of the starting addresses of four memory segments that the 8086 is working with at a particular time.
- The four segment registers are the code segment (CS) register, the stack segment (SS) register, the extra segment (ES) register, and the data segment (DS) register.
Instruction Pointer:

- The instruction pointer register holds the 16-bit addresses, or offset, of the next code byte within this code segment.
Register organization of 8086

- 8086 has a powerful set of registers known as **general purpose** and **special purpose** registers.
- All of them are 16-bit registers.
General Purpose Registers

- The registers AX, BX, CX and DX are the general purpose 16-bit registers.
- The general purpose registers, can be used as either 8-bit registers or 16-bit registers.
- These registers are referred to as the accumulator register (A), the base register (B), the count register (C), and the data register (D).
- General purpose registers, can be used for holding data, variables and intermediate results temporarily or for other purposes like a counter or for storing offset address for some particular addressing modes etc.

<table>
<thead>
<tr>
<th>Register</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td>Word multiply, word divide, word I/O</td>
</tr>
<tr>
<td>AL</td>
<td>Byte multiply, byte divide, byte I/O, translate, decimal arithmetic</td>
</tr>
<tr>
<td>AH</td>
<td>Byte multiply, byte divide</td>
</tr>
<tr>
<td>BX</td>
<td>Translate</td>
</tr>
<tr>
<td>CX</td>
<td>String operations, loops</td>
</tr>
<tr>
<td>CL</td>
<td>Variable shift and rotate</td>
</tr>
<tr>
<td>DX</td>
<td>Word multiply, word divide, indirect I/O</td>
</tr>
</tbody>
</table>

_Table: Dedicated Register Functions_
Special Purpose Registers

- Segment Registers
- Pointers and Index Registers
- Flag Register

The special purpose registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes.

Segment Registers

- 8086 addresses a segmented memory.
- The complete 1MB memory, which the 8086 addresses, is divided into 16 logical segments.
- Each segment thus contains 64KB of memory.
- There are four segment registers, viz.
  - Code Segment Register (CS)
  - Data Segment Register (DS)
  - Extra Segment Register (ES)
  - Stack Segment Register (SS)
Segment Registers Cont…

- Code segment register is used for addressing a memory location in the code segment of the memory, where the executable program is stored.
- The data segment register points to the data segment of the memory, where the data is resided.
- The extra segment also refers to a segment which essentially is another data segment of the memory. Thus, the extra segment also contains data.
- The stack segment register is used for addressing stack segment of memory, i.e., memory which is used to store stack data. The CPU uses the stack for temporarily storing important data, e.g., the contents of the CPU registers which will be required at a later stage.

Pointer Registers

**Stack Pointer Register:**

- A stack is a section of memory set aside to store addresses and data while subprogram is executing.
- The 8086 allows you to set aside an entire 64Kbyte segment as a stack.
- The data segment register points to the data segment of the memory, where the data is resided.
- The extra segment also refers to a segment which essentially is another data segment of the memory. Thus, the extra segment also contains data.
- The stack segment register is used for addressing stack segment of memory, i.e., memory which is used to store stack data. The CPU uses the stack for temporarily storing important data, e.g., the contents of the CPU registers which will be required at a later stage.
EU executes instructions that have already been fetched by the BIU. BIU and EU functions separately.

Bus Interface Unit (BIU)
BIU fetches instructions, reads data from memory and I/O ports, writes data to memory and I/O ports.

Dedicated Adder to generate 20 bit address

Four 16-bit segment registers
Code Segment (CS)
Data Segment (DS)
Stack Segment (SS)
Extra Segment (ES)
8086 Microprocessor

**Architecture**

- **Bus Interface Unit (BIU)**

**Segment Registers**

| CS | DS | SS | ES | IP |

**Internal Communication Registers**

- **8086's 1-megabyte memory is divided into segments of up to 64K bytes each.**
- **The 8086 can directly address four segments (256 K bytes within the 1 M byte of memory) at a particular time.**
- **Programs obtain access to code and data in the segments by changing the segment register content to point to the desired segments.**

---

**Code Segment Register**

- **16-bit**
  - CS contains the base or start of the current code segment; IP contains the distance or offset from this address to the next instruction byte to be fetched.
  - BIU computes the 20-bit physical address by logically shifting the contents of CS 4-bits to the left and then adding the 16-bit contents of IP.
  - That is, all instructions of a program are relative to the contents of the CS register multiplied by 16 and then offset is added provided by the IP.
### Data Segment Register

- **16-bit**
- Points to the current data segment; operands for most instructions are fetched from this segment.
- The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.

![Segment Registers Diagram](image)

### Stack Segment Register

- **16-bit**
- Points to the current stack.
- The 20-bit physical stack address is calculated from the Stack Segment (SS) and the Stack Pointer (SP) for stack instructions such as `PUSH` and `POP`.
- **In based addressing mode**, the 20-bit physical stack address is calculated from the Stack segment (SS) and the Base Pointer (BP).
**Segment Registers**

**Extra Segment Register**

- 16-bit
- Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored.
- String instructions use the ES and DI to determine the 20-bit physical address for the destination.

**Instruction Pointer**

- 16-bit
- Always points to the next instruction to be executed within the currently executing code segment.
- So, this register contains the 16-bit offset address pointing to the next instruction code within the 64Kb of the code segment area.
- Its content is automatically incremented as the execution of the next instruction takes place.
A group of First-In-First-Out (FIFO) in which up to 6 bytes of instruction code are pre fetched from the memory ahead of time.

This is done in order to speed up the execution by overlapping instruction fetch with execution.

This mechanism is known as pipelining.
Architecture

Accumulator Register (AX)

- Consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX.
- AL in this case contains the low order byte of the word, and AH contains the high-order byte.
- The I/O instructions use the AX or AL for inputting / outputting 16 or 8 bit data to or from an I/O port.
- Multiplication and Division instructions also use the AX or AL.

Execution Unit (EU)

Base Register (BX)

- Consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX.
- BL in this case contains the low-order byte of the word, and BH contains the high-order byte.
- This is the only general purpose register whose contents can be used for addressing the 8086 memory.
- All memory references utilizing this register content for addressing use DS as the default segment register.
Counter Register (CX)

- Consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX.
- When combined, CL register contains the low order byte of the word, and CH contains the high-order byte.
- Instructions such as SHIFT, ROTATE and LOOP use the contents of CX as a counter.

Example:

The instruction LOOP START automatically decrements CX by 1 without affecting flags and will check if [CX] = 0.

If it is zero, 8086 executes the next instruction; otherwise the 8086 branches to the label START.

Data Register (DX)

- Consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.
- When combined, DL register contains the low order byte of the word, and DH contains the high-order byte.
- Used to hold the high 16-bit result (data) in 16 X 16 multiplication or the high 16-bit dividend (data) before a 32 ÷ 16 division and the 16-bit reminder after division.
Stack Pointer (SP) and Base Pointer (BP)

- SP and BP are used to access data in the stack segment.
- SP is used as an offset from the current SS during execution of instructions that involve the stack segment in the external memory.
- SP contents are automatically updated (incremented/decremented) due to execution of a POP or PUSH instruction.
- BP contains an offset address in the current SS, which is used by instructions utilizing the based addressing mode.

Source Index (SI) and Destination Index (DI)

- Used in indexed addressing.
- Instructions that process data strings use the SI and DI registers together with DS and ES respectively in order to distinguish between the source and destination addresses.
Source Index (SI) and Destination Index (DI)

- Used in indexed addressing.
- Instructions that process data strings use the SI and DI registers together with DS and ES respectively in order to distinguish between the source and destination addresses.

Generating Memory Addresses

- How can a 16-bit microprocessor generate 20-bit memory addresses?

1. Left shift 4 bits
2. Offset
   - 16-bit register
   - 0000
3. Addr1 + 0FF
   - Addr1
   - Offset
4. FFFFF
   - Segment (64K)
   - Address
5. 00000
   - 1M memory space

Intel 80x86 memory address generation
Memory Segmentation

- A segment is a 64KB block of memory starting from any 16-byte boundary
  - For example: 00000, 00010, 00020, 20000, 8CE90, and E0840 are all valid segment addresses
  - The requirement of starting from 16-byte boundary is due to the 4-bit left shifting

Segment registers in BIU

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>Code Segment</td>
</tr>
<tr>
<td>DS</td>
<td>Data Segment</td>
</tr>
<tr>
<td>SS</td>
<td>Stack Segment</td>
</tr>
<tr>
<td>ES</td>
<td>Extra Segment</td>
</tr>
</tbody>
</table>

Memory Address Calculation

- Segment addresses must be stored in segment registers
- Offset is derived from the combination of pointer registers, the Instruction Pointer (IP), and immediate values

Examples

<table>
<thead>
<tr>
<th>CS</th>
<th>IP</th>
<th>Instruction address</th>
<th>DS</th>
<th>DI</th>
<th>Data address</th>
<th>SS</th>
<th>SP</th>
<th>Stack address</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>8 A 0</td>
<td>1</td>
<td>2</td>
<td>3 4</td>
<td>5</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1 4</td>
<td>0</td>
<td>0</td>
<td>2 2</td>
<td>FF</td>
<td>FE</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>A B 4</td>
<td>2</td>
<td>2</td>
<td>3 6 2</td>
<td>FF</td>
<td>FE</td>
<td>0</td>
</tr>
</tbody>
</table>
Flag Register

This flag is set, when the result of any computation is negative.

- **Sign Flag**
- **Zero Flag**
- **Parity Flag**
- **Overflow Flag**
- **Auxiliary Carry Flag**
- **Carry Flag**

**Direction Flag**
This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto-incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto-incrementing mode.

**Tarp Flag**
If this flag is set, the processor enters the single step execution mode by generating internal interrupts after the execution of each instruction.

**Interrupt Flag**
Causes the 8086 to recognize external mask interrupts; clearing IF disables these interrupts.

Execution Unit (EU)

**Carry Flag**
This flag is set when there is a carry out of MSB in case of addition or a borrow in case of subtraction.

**Overflow Flag**
This flag is set, if an overflow occurs, i.e., if the result of a signed operation is large enough to accommodate in a destination register. The result is of more than 7-bits in size in case of 8-bit signed operation and more than 15-bits in size in case of 16-bit signed operations, then the overflow will be set.

**Direction Flag**
This flag is set, if there is a carry from the lowest nibble, i.e., bit three, during addition, or borrow for the lowest nibble, i.e., bit three, during subtraction.

**Over flow Flag**
This flag is set, if there is a carry out of MSB in case of addition or a borrow in case of subtraction.

**Auxiliary Carry Flag**
This flag is set, if there is a carry out of MSB in case of addition or a borrow in case of subtraction.

**Zero Flag**
This flag is set, if the result of the computation or comparison performed by an instruction is zero.

**Parity Flag**
This flag is set to 1, if the lower byte of the result contains even number of 1's; for odd number of 1's set to zero.

Architecture

8086 Microprocessor

8086 registers categorized into 4 groups

<table>
<thead>
<tr>
<th>Sl.No.</th>
<th>Type</th>
<th>Register width</th>
<th>Name of register</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>General purpose register</td>
<td>16 bit</td>
<td>AX, BX, CX, DX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 bit</td>
<td>AL, AH, BL, BH, CL, CH, DL, DH</td>
</tr>
<tr>
<td>2</td>
<td>Pointer register</td>
<td>16 bit</td>
<td>SP, BP</td>
</tr>
<tr>
<td>3</td>
<td>Index register</td>
<td>16 bit</td>
<td>SI, DI</td>
</tr>
<tr>
<td>4</td>
<td>Instruction Pointer</td>
<td>16 bit</td>
<td>IP</td>
</tr>
<tr>
<td>5</td>
<td>Segment register</td>
<td>16 bit</td>
<td>CS, DS, SS, ES</td>
</tr>
<tr>
<td>6</td>
<td>Flag (PSW)</td>
<td>16 bit</td>
<td>Flag register</td>
</tr>
</tbody>
</table>
## Architecture

### Registers and Special Functions

<table>
<thead>
<tr>
<th>Register</th>
<th>Name of the Register</th>
<th>Special Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td>16-bit Accumulator</td>
<td>Stores the 16-bit results of arithmetic and logic operations</td>
</tr>
<tr>
<td>AL</td>
<td>8-bit Accumulator</td>
<td>Stores the 8-bit results of arithmetic and logic operations</td>
</tr>
<tr>
<td>BX</td>
<td>Base register</td>
<td>Used to hold base value in base addressing mode to access memory data</td>
</tr>
<tr>
<td>CX</td>
<td>Count register</td>
<td>Used to hold the count value in SHIFT, ROTATE and LOOP instructions</td>
</tr>
<tr>
<td>DX</td>
<td>Data Register</td>
<td>Used to hold data for multiplication and division operations</td>
</tr>
<tr>
<td>SP</td>
<td>Stack Pointer</td>
<td>Used to hold the offset address of top stack memory</td>
</tr>
<tr>
<td>BP</td>
<td>Base Pointer</td>
<td>Used to hold the base value in base addressing using SS register to access data from stack memory</td>
</tr>
<tr>
<td>SI</td>
<td>Source Index</td>
<td>Used to hold index value of source operand (data) for string instructions</td>
</tr>
<tr>
<td>DI</td>
<td>Data Index</td>
<td>Used to hold the index value of destination operand (data) for string operations</td>
</tr>
</tbody>
</table>

### Pins and Signals

#### 8086 Microprocessor

#### AD0-AD15 (Bidirectional)

**Address/Data bus**

These are the time multiplexed memory I/O address and data lines. Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4.

Here T1, T2, T3, T4 and Tw are the clock states of a machine cycle. Tw is a wait state.

These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

#### A16/S3, A17/S4, A18/S5, A19/S6

These are time multiplexed address and status signals S6, S3 indicates which segment register being used.

S6 gives current setting of interrupt flag and S3 is always Low.
**BHE (Active Low)/S₇ (Output)**

Bus High Enable/Status

It is used to enable data onto the most significant half of data bus, D₈-D₁₅, 8-bit device connected to upper half of the data bus use BHE (Active Low) signal. It is multiplexed with status signal S₇. S₇ is not currently used.

**MN/ MX**

**MINIMUM / MAXIMUM**

This pin signal indicates what mode the processor is to operate in, either minimum (single processor) or maximum (multiprocessor) mode.

**RD (Read) (Active Low)**

The signal is used for read (Memory or I/O read) operation. It is an output signal. It is active when low.

**TEST (Active Low)**

This input is examined by a WAIT instruction.

If the TEST input goes low, execution will continue, else, the processor remains in an idle state.

The input is synchronized internally during each clock cycle on leading edge of clock.

The signal is active low.

**READY**

This is the acknowledgement from the slow device or memory that they have completed the data transfer.

The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086.

The signal is active high.
**Common signals**

**RESET (Input)**
Causes the processor to immediately terminate its present activity.
The signal must be active HIGH for at least four clock cycles.

**CLK**
The clock input provides the basic timing for processor operation and bus control activity. It's an asymmetric square wave with 33% duty cycle.

**INTR Interrupt Request**
This is a level triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.

This signal is active high and internally synchronized.

---

**Min/Max Pins**

The 8086 microprocessor can work in two modes of operations: **Minimum mode** and **Maximum mode**.

In the **minimum mode** of operation the microprocessor do not associate with any co-processors and can not be used for multiprocessor systems.

In the **maximum mode** the 8086 can work in multi-processor or co-processor configuration.

Minimum or maximum mode operations are decided by the pin MN/ MX(Active low).

When this pin is high 8086 operates in **minimum mode** otherwise it operates in **Maximum mode**.
### Pins and Signals

#### 8086 Microprocessor

<table>
<thead>
<tr>
<th>Pins</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>AD1</td>
<td>Address 1</td>
</tr>
<tr>
<td>AD2</td>
<td>Address 2</td>
</tr>
<tr>
<td>AD3</td>
<td>Address 3</td>
</tr>
<tr>
<td>AD4</td>
<td>Address 4</td>
</tr>
<tr>
<td>AD5</td>
<td>Address 5</td>
</tr>
<tr>
<td>AD6</td>
<td>Address 6</td>
</tr>
<tr>
<td>AD7</td>
<td>Address 7</td>
</tr>
<tr>
<td>AD8</td>
<td>Address 8</td>
</tr>
<tr>
<td>AD9</td>
<td>Address 9</td>
</tr>
<tr>
<td>M/IO</td>
<td>Memory/IO</td>
</tr>
<tr>
<td>HOLD</td>
<td>Hold</td>
</tr>
<tr>
<td>HLDA</td>
<td>Hold Acknowledge</td>
</tr>
<tr>
<td>NMI</td>
<td>NMI</td>
</tr>
<tr>
<td>INTR</td>
<td>Interrupt</td>
</tr>
<tr>
<td>CLK</td>
<td>Clock</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
</tr>
</tbody>
</table>

#### Minimum mode signals

**For minimum mode operation, the MN/ MX is tied to VCC (logic high)**

8086 itself generates all the bus control signals.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT/R</td>
<td>(Data Transmit/Receive) Output signal from the processor to control the direction of data flow through the data transceivers.</td>
</tr>
<tr>
<td>DEN</td>
<td>(Data Enable) Output signal from the processor used as output enable for the transceivers.</td>
</tr>
<tr>
<td>ALE</td>
<td>(Address Latch Enable) Used to demultiplex the address and data lines using external latches.</td>
</tr>
<tr>
<td>M/IO</td>
<td>Used to differentiate memory access and I/O access. For memory reference instructions, it is high. For IN and OUT instructions, it is low.</td>
</tr>
<tr>
<td>WR</td>
<td>Write control signal; asserted low whenever processor writes data to memory or I/O port.</td>
</tr>
<tr>
<td>INTA</td>
<td>(Interrupt Acknowledge) When the interrupt request is accepted by the processor, the output is low on this line.</td>
</tr>
<tr>
<td>HOLD</td>
<td>Input signal to the processor from the bus masters as a request to grant the control of the bus. Usually used by the DMA controller to get the control of the bus.</td>
</tr>
<tr>
<td>HLDA</td>
<td>(Hold Acknowledge) Acknowledge signal by the processor to the bus master requesting the control of the bus through HOLD. The acknowledge is asserted high, when the processor accepts HOLD.</td>
</tr>
</tbody>
</table>
### Maximum mode signals

During maximum mode operation, the MN/\textit{MX} is grounded (logic low)

Pins 24 - 31 are reassigned

#### Status signals

These are decoded as shown.

<table>
<thead>
<tr>
<th>Status Signal</th>
<th>Machine Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_0 )</td>
<td>Interrupt acknowledge</td>
</tr>
<tr>
<td>( S_1 )</td>
<td>Read I/O port</td>
</tr>
<tr>
<td>( S_2 )</td>
<td>Write I/O port</td>
</tr>
<tr>
<td>( S_3 )</td>
<td>Halt</td>
</tr>
<tr>
<td>( S_4 )</td>
<td>Code access</td>
</tr>
<tr>
<td>( S_5 )</td>
<td>Read memory</td>
</tr>
<tr>
<td>( S_6 )</td>
<td>Write memory</td>
</tr>
<tr>
<td>( S_7 )</td>
<td>Passive/Active</td>
</tr>
</tbody>
</table>

#### Queue signals

\((\text{Queue Status})\) The processor provides the status of queue in these lines.

The queue status can be used by external device to track the internal status of the queue in 8086.

The output on \(QS_0\) and \(QS_1\) can be interpreted as shown in the table.

<table>
<thead>
<tr>
<th>Queue status</th>
<th>Queue operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ( QS_0 )</td>
<td>No operation</td>
</tr>
<tr>
<td>0 ( QS_1 )</td>
<td>First byte of an opcode from queue</td>
</tr>
<tr>
<td>1 ( QS_0 )</td>
<td>Empty the queue</td>
</tr>
<tr>
<td>1 ( QS_1 )</td>
<td>Subsequent byte from queue</td>
</tr>
</tbody>
</table>
Unit-I

- Minimum mode system operation
- Timing diagram (Write & Read Cycle)
- Maximum mode system operation
- Timing diagram (Write & Read Cycle)
Minimum Mode

- In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by connecting its MN/MX pin to logic 1.
- In this mode, all the control signal are given out by the microprocessor chip itself.
- There is a single microprocessor in the single mode system.
- The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices, chip selection logic for selecting memory or I/O devices.
- The latches are generally buffered output D-type flip-flops like 74LS373.
- They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

Minimum Mode

- Transreceivers are the bi-directional buffers and are sometimes called data amplifiers.
- They are required to separate the valid data from the time multiplexed address/data signals.
- They are controlled by two signals namely DEN and DT/R.
- The DEN signal indicates the valid data available on the data bus while DT/R indicates the direction of data, i.e., from / to the processor.
- The system contains memory(RAM or ROM), I/O devices for the communication with the processor.
- The clock generator(IC8284) generates the clock from the crystal oscillator and used as an accurate timing reference for the system.
**Minimum Mode**

- The clock generator also synchronizes some external signals with the system clock.
- Since it has 20 address lines and 16 data lines, the 8086 CPU requires three octal address latches and two octal data buffers for the complete address and data separation.
- The working of the minimum mode configuration system can be better described in terms of the timing diagrams.
- The timing diagram can be categorized into two parts. The first is the timing diagram for read cycle and second is the write cycle.
Minimum Mode (Read Cycle)

- The read cycle begins in $T_1$ with the assertion of the Address Latch Enable (ALE) signal and M/IO signal.
- During the negative going edge of this signal, the valid address is latched on the local bus.
- The BHE and $A_0$ signals address low, high or both bytes.
- From $T_1$ to $T_4$, the signal indicates a memory or I/O operation.
- At $T_2$, the address is removed from the local bus and is sent to the output. The bus is then tristated.
- The Read (RD) control signal is also activated in $T_2$.
- This signal causes the addressed device to enable its data bus drivers.

Minimum Mode (Read Cycle)

- After RD goes low, the valid data is available on the data bus.
- The addressed device will drive the READY line high.
- When the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.
- CS logic indicates chip select logic and ‘e’ and ‘o’ suffixes indicate even and odd address memory banks.
Minimum Mode (Read cycle)

A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO signal is again asserted to indicate a memory or IO operation.

In T2, after sending the address in T1, the processor sends the data to be written to the addressed location.

The data remains on the bus until the middle of T4 state.

The WR becomes active at the beginning of T2 (unlike RD is somewhat delayed in T2 to provide time for floating.)
Minimum Mode (Write cycle)

<table>
<thead>
<tr>
<th>T₁</th>
<th>T₂</th>
<th>T₃</th>
<th>T₄</th>
<th>T₅</th>
<th>T₆</th>
<th>T₇</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add/Status</td>
<td>BHE</td>
<td>S₇ – S₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A₁₉ – A₁₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add/Data</td>
<td>A₀₅ – A₀</td>
<td>Valid data D₁₅ – D₀</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DT/R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Maximum Mode

- In the maximum mode 8086 system the microprocessor 8086 is operated in maximum mode by strapping its MN/MX pin to logic 0.
- In this mode the processor drives the status signals Sₓ, Sₓ, Sₓ.
- Another chip called a bus controller derives the control signal using this status information.
- In the maximum mode there may be more than one microprocessor in the system configuration.
- The remaining components in the system are latches, transceivers, clock generator, memory and I/O devices.
Maximum Mode Cont...

Maximum Mode (Read cycle)
Addressing Modes of 8086

- Addressing mode indicates a way of locating data or operands.
- The addressing modes describe the types of operands and the way they are accessed for executing an instruction.
- According to the flow of instruction execution, the instructions may be categorized as
  1. Sequential control flow instructions
  2. Control transfer instructions
Addressing Modes of 8086 Cont..

- Sequential control flow instructions are the instructions, which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program. For example the arithmetic, logic, data transfer and processor control instructions are sequential control flow instructions.

- The control transfer instructions, on the other hand, transfer control to same predefined address or the address somehow specified in the instruction, after their execution. For example CALL, RET and JUMP instructions fall under this category.

Addressing Modes of 8086 Cont..

The addressing modes for sequential control transfer instructions are:

- **Immediate**: In this type of addressing, immediate data is a part of instruction and appears in the form of successive byte or bytes.

- **Ex**: MOV AX, 0005H

  In the above example, 0005H is the immediate data. The immediate data may be 8-bit or 16-bit in size.
Addressing Modes of 8086 Cont..

- **Direct:** In the direct addressing mode a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

  - Ex: MOV AX, [5000H]

  Here, data resides in a memory location in the data segment, whose effective address may be completed using 5000H as the offset address and content of DS as segment address. The effective address here, is 10H * DS + 5000H.

- **Register:** In register addressing mode, the data is stored in a register and is referred using the particular register. All the registers, except IP, may be used in this mode.

  - Ex: MOV BX, AX
Addressing Modes of 8086 Cont..

- **Register Indirect**: Sometimes, the address of the memory location, which contains data or operand, is determined in an indirect way, using the offset register. This mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI registers. The default segment is either DS or ES. The data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.

  - **Ex**: MOV AX, [BX]
    Here, data is present in a memory location in DS whose offset address is in BX. The effective address of the data is given as 10H * DS + [BX].

- **Indexed**: In this addressing mode, offset of the operand is stored in one of the index registers. DS and ES are the default segments for index registers, SI and DI respectively. This is a special case of register indirect addressing mode.

  - **Ex**: MOV AX, [SI]
    Here, data is available at an offset address stored in SI in DS. The effective address, in this case, is computed as 10H * DS + [SI].
Addressing Modes of 8086 Cont..

- **Register Relative**: In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment.

  - Ex: MOV AX, 50H [BX]

Here, the effective address is given as 10H *DS+50H+ [BX]

Addressing Modes of 8086 Cont..

- **Based Indexed**: The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

  - Ex: MOV AX, [BX][SI]

Here, BX is the base register and SI is the index register the effective address is computed as 10H * DS + [BX] + [SI].
Addressing Modes of 8086 Cont..

- **Relative Based Indexed:** The effective address is formed by adding an 8 or 16-bit displacement with the sum of the contents of any one of the base register (BX or BP) and any one of the index register, in a default segment.

- **Ex:** MOV AX, 50H [BX] [SI]
  
  Here, 50H is an immediate displacement, BX is base register and SI is an index register the effective address of data is computed as
  
  \[ 10H \cdot DS + [BX] + [SI] + 50H \]

Addressing Modes of 8086 Cont..

- **For control transfer instructions,** the addressing modes depend upon whether the destination is within the same segment or different one.

- **Basically,** there are two addressing modes for the control transfer instructions, intersegment addressing and intra-segment addressing modes.
Addressing Modes of 8086 Cont..

Addressing modes for Control Transfer Instructions

- **Intra-segment Direct Mode**: In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfer instruction lies and appears directly in the instruction as an immediate displacement value. In this addressing mode, the displacement is computed relative to the content of the instruction pointer IP.

- The effective address to which the control will be transferred is given by the sum of 8 or 16-bit displacement and current content of IP. In the case of jump instruction, if the signed displacement (d) is of 8-bits (i.e. –128<d<+127) we term it as short jump and if it is of 16-bits (i.e.-32,768<d<+32,767) it is termed as long jump.

- Ex. JMP SHORT LABEL;
Addressing Modes of 8086 Cont..

- **Intra-segment Indirect Mode:** In this mode, the displacement to which the control is to be transferred, is in the same segment in which the control transfer instruction lies, but it is passed to the instruction indirectly. Here, the branch address is found as the content of a register or a memory location. This addressing mode may be used in unconditional branch instructions.
  - **JMP [BX]:**
    Jump to effective address stored in BX

Addressing Modes of 8086 Cont..

- **Intersegment Direct:** In this mode, the address to which the control is to be transferred is in a different segment. This addressing mode provides a means of branching from one code segment to another code segment. Here, the CS and IP of the destination address are specified directly in the instruction.
  - **JMP 5000H : 2000H**
    Jump to effective address 2000H in segment 5000H
Addressing Modes of 8086 Cont..

- **Intersegment Indirect**: In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly, i.e. contents of a memory block containing four bytes, i.e. IP (LSB), IP(MSB), CS(LSB) and CS (MSB) sequentially. The starting address of the memory block may be referred using any of the addressing modes, except immediate mode.

- **JMP [2000H]**
  
  Jump to an address in the other segment at effective address 2000H in DS, that points to the memory block.