Northern India Engineering College, Delhi

(GGSIP University)

SUBJECT – MICROPROCESSOR SYSTEMS SUBJECT CODE- ETCS-302

PAPER –II

Q.1 What is the difference between effective address and physical address?

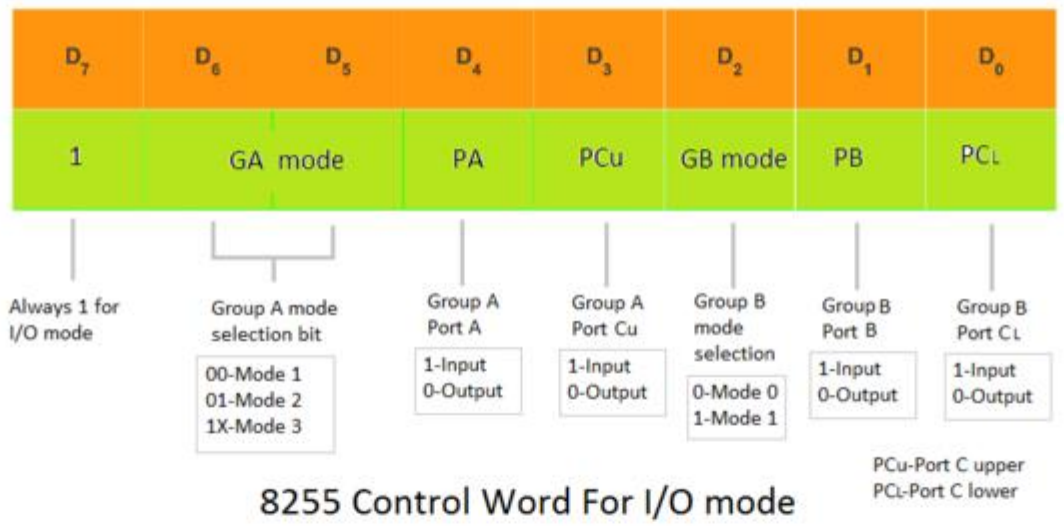
An effective address is any operand to an instruction which references memory. Effective addresses have a very simple syntax: they consist of an expression evaluating to the desired address, enclosed in square brackets. For example:

```
mov ax,[wordvar]
mov ax,[wordvar+1]
mov ax,[es:wordvar+bx]
```

In computing, a physical address (also real address, or binary address), is a memory address that is represented in the form of a binary number on the address bus circuitry in order to enable the data bus to access a *particular* storage cell of main, or a register of memory mapped I/O device.

Q2) Explain the programming of 8255 in mode 2?

Only group A can be initialized in this mode. Port A can be used for *bidirectional handshake* data transfer. This means that data can be input or output on the same eight lines (PA0 - PA7). Pins PC4 - PC7 are used as handshake lines for port A. The remaining pins of port C (PC0 - PC3) can be used as input/output lines if group B is initialized in mode 0 or as handshaking for port b if group B is initialized in mode 1.



Q3. What are the features of Pentium?

Ans. Features of Pentium:

- P5 architecture / 80586
- Introductory version: 60MHz and 66MHz, 110MIPS / 100MHz, 150MIPS
- 16KB of cache size (8KB IC, 8KB DC)

- 4GB of memory system, 64-bit data bus
- Executes up to two instructions at a time (If they don't conflict!)

Q4. Write down the addressing modes of 80386 with examples.

Ans. Addressing modes of 80386:

- Register addressing: MOV ECX, EDX
- Immediate addressing: MOV EBX, 12345678H
- Direct addressing: MOV CX, LIST
- Register indirect addressing: MOV AL, [ECX]
- Base-plus-index addressing: MOV [EAX+EBX], CL
- Register relative addressing: MOV AX, [ECX+4]
- Base relative-plus-index addressing: MOV EAX, ARRAY [EBX+ECX]
- Scaled-index addressing: MOV EDX, [EAX+4*EBX]

Q5. Differentiate between microprocessor and microcontroller?

Ans. 1. Microprocessor is a general purpose devise

2. Microcontroller is indented for a specific purpose

3. Memory ,I/O devices etc need to be interfaced with microprocessor

4. Microcontroller is having its own memory.

5. Microcontroller is a system on a chip

Q6. What is the effect of executing the instruction?

MOV CX, [SOURCE_MEM]

Where SOURCE_MEM equal to 2016 is a memory location offset relative to the current data segment starting at address $1A000_{16}$

Ans. Execution of this instruction results in the following:

$$((DS)\ 0 + 20_{16}) \rightarrow (CL)$$
$$((DS)\ 0 + 20_{16} + 1_{16}) \rightarrow (CH)$$

In other words, CL is loaded with the contents held at memory address

$$1A000_{16} + 20_{16} + 1_{16} = 1A021_{16}$$

Q7. The original contents of AX, BL, word-sized memory location SUM, and carry flag

CF are 1234H, ABH, 00CDH, and 0H, respectively. Describe the results of executing the following sequence of instructions:

ADD AX, [SUM]

ADC BL, 05H

INC WORD PTR [SUM]

Ans. Executing the first instruction adds the word in the accumulator and the word in the memory location pointed to by address SUM. The result is placed in the accumulator. That is,

(AX) ← (AX) + (SUM) = 1234H + 00CDH = 1301H

The carry flag remains reset.

The second instruction adds to the lower byte of the base register (BL) the immediate operand 5H and the carry flag, which is 0H. This gives

(BL) ← (BL) + imm8 + (CF) = ABH + 5H+ 0H = B0H

Since no carry is generated CF remains reset.

The last instruction increments the contents of memory location SUM by one. That is,

(SUM) ← (SUM) + 1H = 00CDH + 1H =00CEH

Q8) Explain the concept of segmented memory clearly indicating its advantages.
There were also four 16-bit segment registers (see figure) that allowed the 8086 CPU to access one megabyte of memory in an unusual way. Rather than concatenating the segment register with the address register, as in most processors whose address space exceeded their register size, the 8086 shifted the 16-bit segment only 4 bits left before adding it to the 16-bit offset (16×segment + offset), therefore producing a 20-bit external (or effective or physical) address from the 32-bit segment:offset pair. As a result, each external address could be referred to by 212 = 4096 different segment:offset pairs. The 16-byte separation between segment bases (due to the 4-bit shift) was called a paragraph. Although considered complicated and cumbersome by many programmers, this scheme also had advantages; a small program (less than 64 kilobytes) could be loaded starting at a fixed offset (such as 0) in its own segment, avoiding the need for relocation, with at most 15 bytes of alignment waste.

Compilers for the 8086-family commonly supported two types of pointer, near and far. Near pointers were 16-bit offsets implicitly associated with the program's code and/or data segment and so could be used only within parts of a program small enough to fit in one segment. Far pointers were 32-bit segment:offset pairs resolving to 20-bit external addresses. Some compilers also supported huge pointers, which were like far pointers except that pointer arithmetic on a huge pointer treated it as a linear 20-bit pointer, while pointer arithmetic on a far pointer wrapped around within its 16-bit offset without touching the segment part of the address.

To avoid the need to specify near and far on numerous pointers, data structures, and functions, compilers also supported "memory models" which specified default pointer sizes. The tiny (max 64K), small (max 128K), compact (data > 64K), medium (code > 64K), large (code,data > 64K), and huge (individual arrays > 64K) models covered practical combinations of near, far, and huge pointers for code and data. The tiny model meant that code and data was shared in a single segment, just as in most 8-bit based processors, and could be used to build .com-files for instance. Precompiled libraries often came in several versions compiled for different memory models.

According to Morse et al., the designers actually contemplated using an 8-bit shift (instead of 4-bit), in order to create a 16 MB physical address space. However, as this would have forced segments to begin on 256 byte boundaries, and 1 MB was considered very large for a microprocessor around 1976, the idea was dismissed. Also, there were not enough pins available on a low-cost 40-pin package.[14]

In principle, the address space of the x86 series could have been extended in later processors by increasing the shift value, as long as applications obtained their segments from the operating system and did not make assumptions about the equivalence of different segment:offset pairs.[15] In practice the use of "huge" pointers and similar mechanisms was widespread and the flat 32-bit addressing made possible with the 32-bit offset registers in the 80386 eventually extended the limited addressing range in a more general way.

Q.9 What are 5 types of dedicated interrupts supported by 8086?
The following are the various types of interrupts:

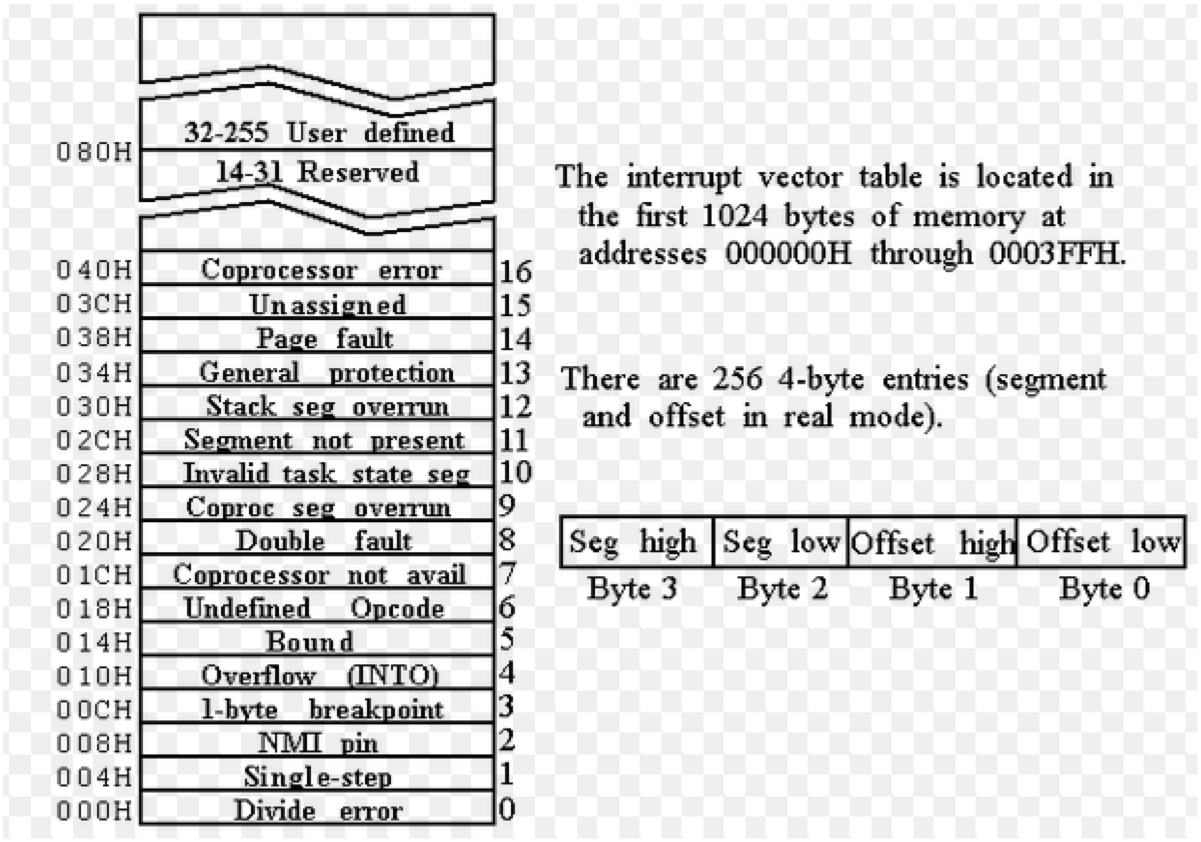
- Type 0 interrupts: This interrupt is also known as the divide by zero interrupt. For cases where the quotient becomes particularly large to be placed / adjusted an error might occur.

- Type 1 interrupts: This is also known as the single step interrupt. This type of interrupt is primarily used for debugging purposes in assembly language.

- Type 2 interrupts: also known as the non-maskable NMI interrupts. These type of interrupts are used for emergency scenarios such as power failure.

- Type 3 interrupts: These type of interrupts are also known as breakpoint interrupts. When this interrupt occurs a program would execute up to its break point.

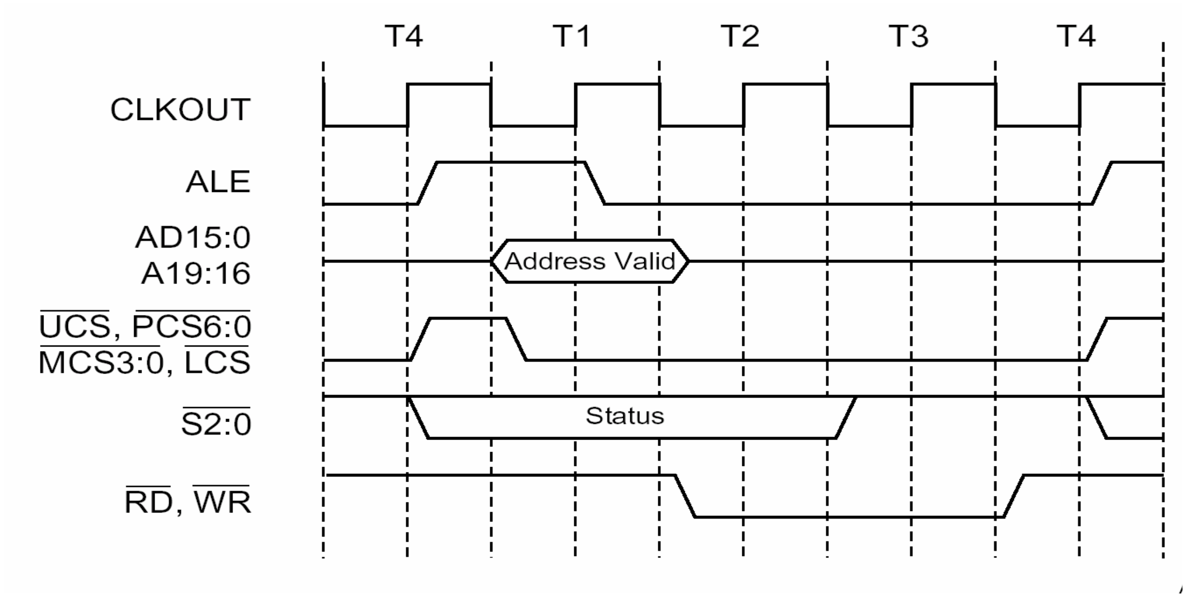
-Type 4 interrupts: Also known as overflow interrupts is generally existent after an arithmetic operation was performed.


080H
32-255 User defined
14-31 Reserved
040H Coprocessor error 16
03CH Unassigned 15
038H Page fault 14
034H General protection 13
030H Stack seg overrun 12
02CH Segment not present 11
028H Invalid task state seg 10
024H Coproc seg overrun 9
020H Double fault 8
01CH Coprocessor not avail 7
018H Undefined Opcode 6
014H Bound 5
010H Overflow (INTO) 4
00CH 1-byte breakpoint 3
008H NMI pin 2
004H Single-step 1
000H Divide error 0
The interrupt vector table is located in the first 1024 bytes of memory at addresses 000000H through 0003FFH.
There are 256 4-byte entries (segment and offset in real mode).
Seg high
Seg low
Offset high
Offset low
Byte 3
Byte 2
Byte 1
Byte 0

Q10 a). What are the features of 80186. Also, Draw the timing diagram of 80186?

Ans. The various enhancement features of 80186/88 processors are :

- Clock Generator
- Programmable Interrupt Controller
- Timers
- Programmable DMA Unit
- Programmable Chip Selection Unit
- Power Save/Power Down Feature
- Refresh Control Unit

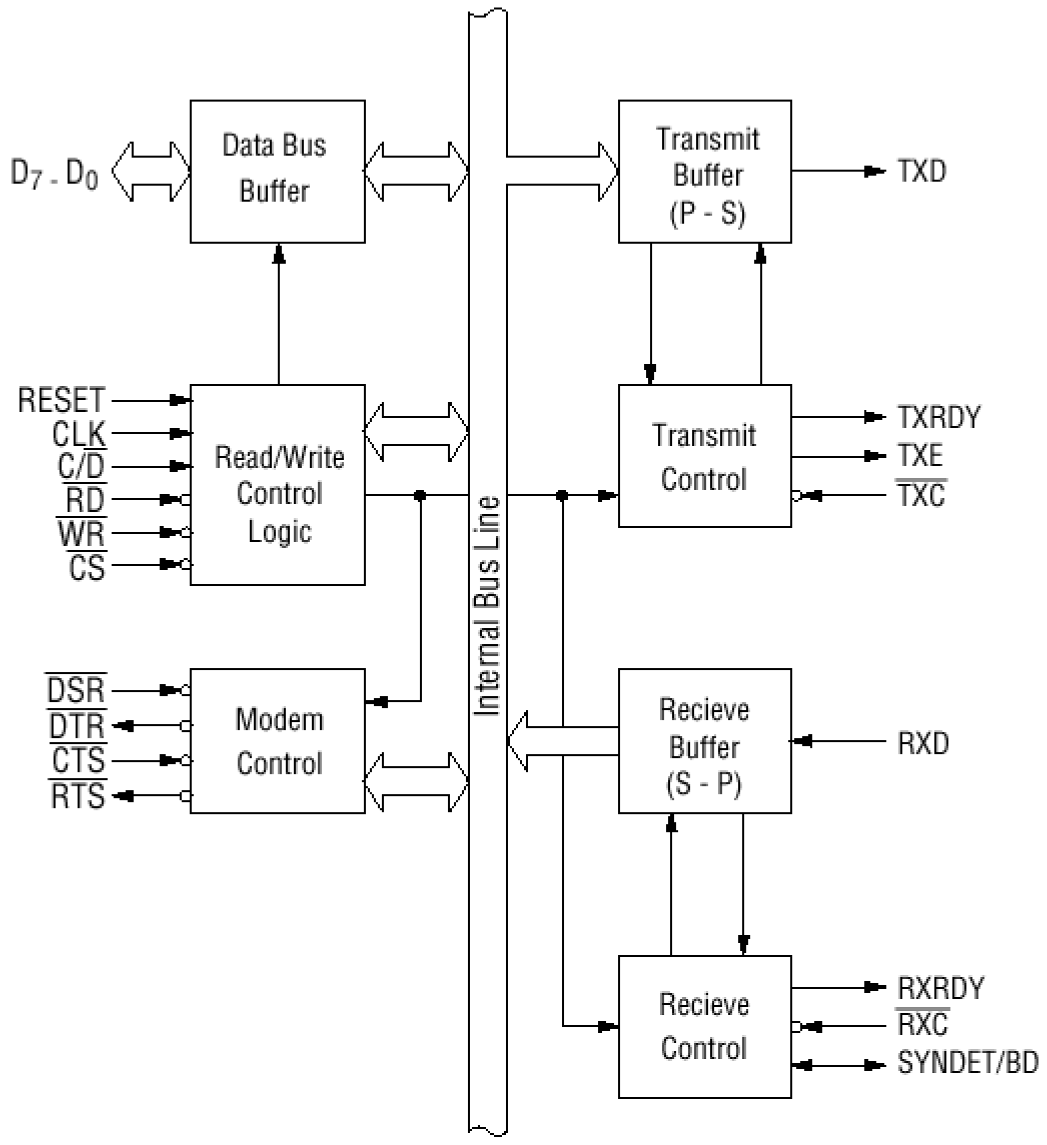

T4
T1
T2
T3
T4
CLKOUT
ALE
AD15:0
A19:16
Address Valid
$\overline{UCS}$, $\overline{PCS6:0}$
$\overline{MCS3:0}$, $\overline{LCS}$
$\overline{S2:0}$
Status
$\overline{RD}$, $\overline{WR}$

The only difference in 80186/88 vs. 8086/88 is in the generation of ALE which is asserted one-half clock cycle earlier

b) Explain the operation od 8251 with the help of block diagram .

The 8251 Universal Synchronous/Asynchronous Receiver/Transmitter packaged in a 28-pin DIP made by Intel. It is typically used for serial communication and was rated for 19.2 Kbits per second signaling rate. It includes 5 sections

1. read/write control logic
2. transmitter
3. receiver
4. data bus system
5. modem control



Q11) Explain the interrupt response sequence of 8086.

The Operation of an Interrupt sequence on the 8086 Microprocessor:

1. External interface sends an interrupt signal, to the Interrupt Request (INTR) pin, or an internal interrupt occurs.

2. The CPU finishes the present instruction (for a hardware interrupt) and sends Interrupt Acknowledge (INTA) to hardware interface.

3. The interrupt type N is sent to the Central Processor Unit (CPU) via the Data bus from the hardware interface.

4. The contents of the flag registers are pushed onto the stack.

5. Both the interrupt (IF) and (TF) flags are cleared. This disables the INTR pin and the trap or single-step feature.

6. The contents of the code segment register (CS) are pushed onto the Stack.

7. The contents of the instruction pointer (IP) are pushed onto the Stack.

8. The interrupt vector contents are fetched, from (4 x N) and then placed into the IP and from (4 x N +2) into the CS so that the next instruction executes at the interrupt service procedure addressed by the interrupt vector.

9. While returning from the interrupt-service routine by the Interrupt Return (IRET) instruction, the IP, CS and Flag registers are popped from the Stack and return to their state prior to the interrupt.