Q2 (a) What are the salient features of INTEL's-8086 microprocessor? Explain with an example how 20 bit physical address is calculated?

Answer

40- Pin IC, 16-bit MP i.e. ALU is of 16 bit wide.

- 20- Address lines (multiplexed with 16- date lines) hence addressing capacity is 2^{20}= 1 M byte. 2^{20}=1048576 bytes= 1MB i.e. 00000 to FFFFF,
- Operates on single +5V supply at 5MHz clock (8086-2- 8 MHZ and 8086-1 10 MHZ)
- Can transfer data either byte wise (8 bit) or word wise (16 bit),
- Virtual memory addressing is possible,
- External clock generator (8284) is required,
- 16-bit data bus (data bus is multiplexed with address bus)
- The 8086 can read a 16-bit word at an even address in one operation and at an odd address in two operations. The 8088 needs two operations in either case.
- The least significant byte of a word on an 8086 family microprocessor is at the lower address
- Architecture is divided in to two units B I U + E U
- Contains Micro-programmed control
- 4 pointers; SP, BP, SI & DI
- 16-Bit flag register (Control + Status)
- 6 byte instruction queue (Pipe lining)
- Memory is segmented & contains 2 banks
- Can operate in two modes Maximum mode & minimum mode
- Two Interrupts – NMI & INTR

Q2 (b) Mention the I/O addressing modes available in INTEL's-8086. Explain with an example for each.

Answer

I/O addressing modes available in INTEL's-8086: 2 types

(i) Fixed (Direct) port Addressing: Here, port address in the range of 00 to FF is provided in the instn itself. IN and OUT instns in this mode are allowed to use only AL or AH.

Ex: IN AL,00h

(ii) Variable (Indirect) port Addressing: Here, the port will be given in directly in a reg/ reg. pair. However, only DX can contain part address.

Ex1: IN AL,DX Ex2: IN AX,DX

Q3 (a) Correct the following instructions if necessary and indicate its addressing mode.

(i) MOV BL, AX **(ii) OUT DX, AL**

(iii) ROL AX, 04 **(iv) DIV BX, CX**

Answer

a) Pin functions of INTEL's-8086:

(i) READY: This is an I/P pin, used to insert WAIT states into the timing of MP. It helps in the synchronization of slower peripherals with MP. When at zero MP enters wait state allowing peripheral an extra T-cycle.

(i) MN/$\overline{MX}$: This is an I/P pin, this pin is to select the mode of operation for 8086;

if MN/$\overline{MX}$ = 0 – maximum mode (multiprocessor mode)

MN/$\overline{MX}$ = 1 – minimum mode (uni-processor mode)

(iii) $\overline{RQ}$/GT: This is an I/O pin. This pin is to provide control of busses to an outside device. Both request and grant are available on the same line. Request /Grant: pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ/GT_0}$ having higher priority than $\overline{RQ/GT_1}$.

(iv) $\overline{BHE}$: This pin helps in accessing the ODD (higher memory) memory bank. When zero, processor can access 2 bytes in one cycle. Bus High Enable / Status: During T_1 the Bus High Enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins D_{15}-D_8. Eight bit oriented devices tied to the upper half of the bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus.

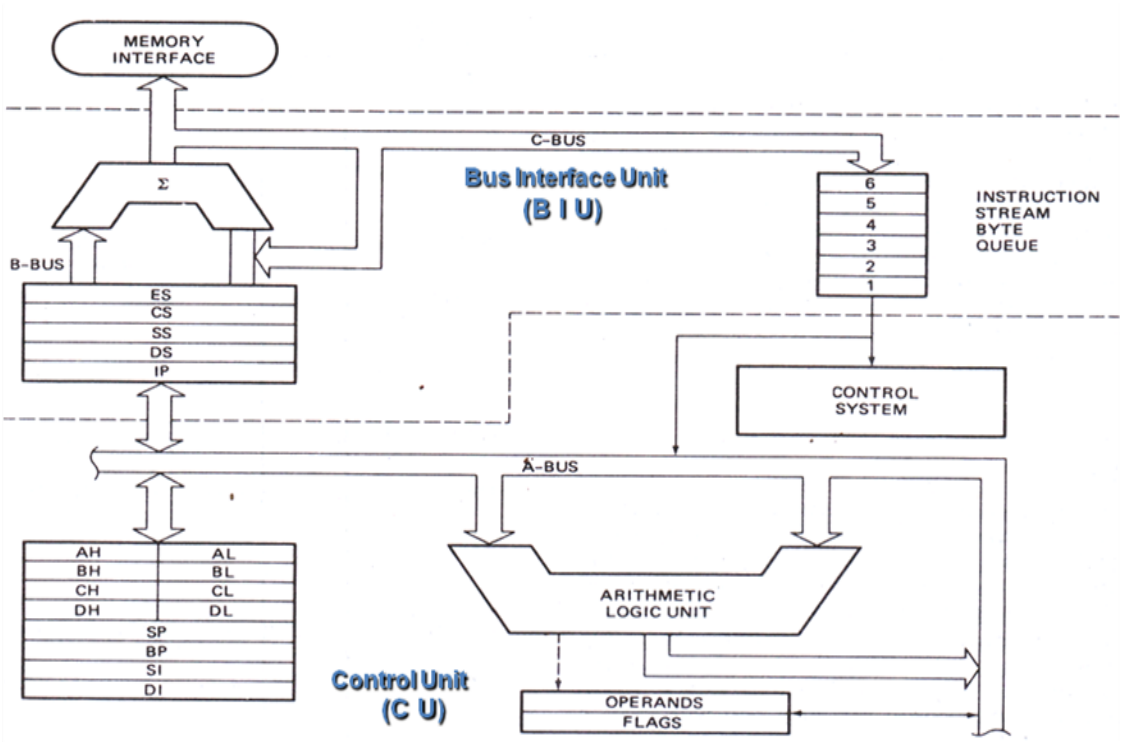
Q3 (b) Explain the following instructions of INTEL's-8086.

(i) STOSB **(ii) CMPSB**

(iii) SCASB **(iv) MOVSB**

Answer

Internal architecture of INTEL's-8086:



In order to speed-up the processing, the 8086 architecture is divided into two independent functional units; the Bus Interface unit (BIU) and the execution unit (EU).

Bus Interface Unit:

The BIU sends out addresses, fetches instructions from memory, reads data from parts and memory and writes data to parts and memory. In total the BIU manages transfer of data and addresses on the buses.

Queue: It is a group of 6 first-in-first-on (FIFO) registers. In order to speed- up the execution of program, BIU fetches six instn. Bytes ahead of time and stores them in the queue. This is called as ' pipelining'. [However, pipelining becomes waste during jump or call instr.

Segment Registers: 8086 contains four 16 bit segment registers and these hold the most

Q4 (a) Explain with examples the inter segment return and intra segment return instructions.

Answer Page Number 184-187 of Textbook I

Q4 (b) When interrupted by an external interrupt what happens to the program execution in INTEL's-8086? Explain. What is the role of IRET instruction?

Answer

Instructions of INTEL's-8086:

(i) STOSB: The STOSB instruction copies a byte from AL to a memory location in the extra segment. In effect it replaces a string element with a byte from AL. DI is used to hold the offset of the memory location in the extra segment. After the copy, DI is automatically incremented or decremented to point to the next string element in memory. If the direction flag, D is cleared, then DI will automatically be incremented by one for a byte string and if the direction flag is set, DI will be automatically decremented by one for a byte string. STOSB does not affect any flags.

EXAMPLES:

```
MOV DI, OFFSET TARGET _STRING   :   Point DI at
                                :   destination string
STOS TARGET STRING              :   Assembler uses string name to
```

determine whether string is of type byte or type word. If byte string, then string byte replaced with contents of AL. If word string, then string word replaced with contents of AX.

```
MOV DI, OFFSET TARGET STRING    :   Point DI at destination string
STOSB                           :   "B" added to STOS
```

mnemonic directly tells assembler to replace byte in string with byte from AL. STOSW would tell assembler directly to replace a word in the string with a word from AX.

(ii) CMPSB: This instruction compares a byte in one string with a byte in another string. The SI holds the offset of source string and DI holds the offset of destination string. The AF, CF, OF, PF, SF and ZF are affected by this instruction, but none of the operands are affected. The D flag decides the auto-increment (DF=0) or auto decrement (DF=1) of both SI and DI. Normally, this instruction is used with REPE or REPNE instructions to compare all elements of the strings.

Ex.: (i)

```
MOV  SI, STRING I ;        SI      Points to STRINGI.
MOV DI, STRING II;DI       Points to STRING II.
CLD                ;       DF=0  ( auto increment)
CMPS STRING II, STRING I;
```

(ii)

```
MOV CX, 100 ; CX = 100, No. of elements

MOV SI, STRING I
MOV DI, STRING II
STD          ;     DF=1 (auto decrement)
REPE CMPSB   ;     Repeat the companion
             ;     of string bytes until
             ;     they are equal, when
             ;     not equal terminate.
```

(iii) SCASB: This instruction compares a string byte with a byte in AL. This instruction affects the flags, that it does not change either the operand in AL or the operand in the string. The string to be scanned must be in the extra segment and DI must contain the offset of the byte or the word to be compared. After SCAS executes, DI will be automatically incremented or decremented to point to the next element in the string depending on D flag. For byte strings DI will be incremented by one, if the direction flag is cleared (0) and DI will be decremented by one if the direction flag is set (I). SCAS

affects the AF, CF, OF, PF, SF and ZF. This instruction is often used with a repeat prefix to find the first occurrence of a specified byte or word in a string.

EXAMPLE:

```
:       Scan a text string of 80 characters for a carriage
:       return
        MOV AL, ODH ; Byte to be scanned for into AL,
MOV DI, OFFSET TEXT STRING  : Offset of string
                                    : to DI
MOV CX, 80  :       CX used as element counter
CLD         :       Clear DF so DI auto increments
REPNE SCAS TEXT STRING      : Compare byte in
                                    : String with byte in AL
```

NOTE: Scanning is repeated while the bytes are not equal and it is not end of the string. If a carriage return ODH is found, ZF=1 and DI will point at the next byte after the carriage return in string. If a carriage return is not found then CX=0 and ZF=0. The assembler uses the name of the string to determine whether the string is of type byte or type word.

(iv) MOVSB: This instruction copies a byte string from a set of locations in the data segment to a set of locations in the extra segment. The offset of the source byte in the data segment must be in the SI register. The offset of the destination in the extra segment must be contained in the DI register. For multiple byte moves the number of elements to be moved in put in the CX- register so that it can function as a counter. After the byte is moved SI and DI are automatically adjusted to point to the next source and the next destination. If the direction flag is 0, then SI and DI will be incremented by 1 after a byte move and if the direction flag is 1, then SI and DI will be decremented by 1 after a byte move. MOVSB affects no flags.

Example: MOVSB or MOVS STRING_DUMP, STRING_CREATE. The assembler will code the instruction for a byte move if STRING_DUMP and STRING_CREATE were declared with a DB. It will code the instruction for a word move if they were declared with a DW. Note that this reference to the source and destination strings does not load SI and DI. This must be done with separate instructions.

Q5 (a) What are the functions of the following pins of numeric co-processor-8087?

(i) RQ/GT **(ii) BUSY**

(iii) S_2 S_1 S_0 **(iv) TEST**

Answer

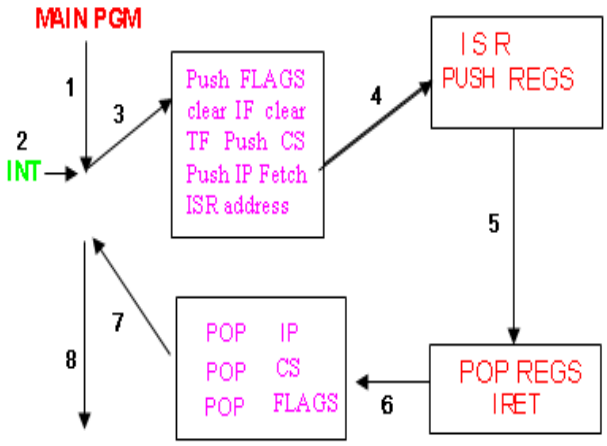
Response of INTEL's-8086 when interrupted:

At the end of each instruction cycle, 8086 checks to see if any interrupts have been requested (activated). If yes, then 8086 responds to the interrupt by stepping through the following major actions, but only after the current instruction execution completion;

- It decrements the stack pointer (SP) by two and pushes the flag register on to stacks,
- It disables the INTR interrupt i/p by clearing the IF in flag register,
- It resets the trap flag in flag register,

- It decrements the SP by two and pushes the current code segment (s) reg. contents on to stack,
- It decrements SP by two again and pushes the current instn. pointer (IP) reg contents on to stack,
- It then jumps to interrupt service routine (for jump).

An IRET instruction at the end of ISR returns the execution to the main program.



Role of IRET for ISR? : An IRET instruction at the end of ISR returns the execution to the main program. IRET is to POP the return address and the status of the previous (main) program execution. Ordinary RET instruction cannot do this.

Q5 (b) Give an example for each of the following 8087 instructions:-

(i) Arithmetic instruction
(ii) Data transfer instruction
(iii) Compare instructions
(iv) Transcendental instruction

Answer Page Number 223-246 of Textbook I

Q6 (a) Using 8086 instructions, write an assembly language program to add a series of 1 byte numbers. Write necessary comments for the same.

Answer

Co-processor-8087 instructions:

(i) FST: Store Top of Stack to the specified operand. This instruction stores the current Top of Stack contents to the specified operand. Ex1: FST ST(7) ; ST(0) ST(7)
Ex2: FST MEM ; ST(0) ST(7)

(ii) FDIV: Divide; This instruction performs real or integer division. By default ST(0) and ST(1) will be the operands. In other cases either one operand or two operands will be explicitly mentioned. Result will be stored in ST(0) in default case.

Ex1: FDIV ; ST(0) / ST(1) ST(0)

Ex2: FDIV MEM ; ST(0) / (DS:MEM) ST(0)
Ex3: FDIV ST(2), ST(1) ; ST(2) / ST(1) ST(2)

(iii) FABS: Find absolute value. This instruction replaces the content of stack top by its absolute value (magnitude). The sign of the operand if any will be neglected.

(iv) FCHS: Change sign; This instruction changes the sign of contents of stack top and stores it into the same location. The sign of the operand will be complimented.

(v) FLDZ: Load zero; This instruction places 0.0 on to the current top of stack

Q6 (b) Explain the following assembler directives:

(i) PTR **(ii) PUBLIC**

(iii) SEGMENT

Answer

(a) Co-processor-8087 instructions:

(i) FST: Store Top of Stack to the specified operand. This instruction stores the current Top of Stack contents to the specified operand. Ex1: FST ST(7) ; ST(0) ST(7)
Ex2: FST MEM ; ST(0) ST(7)

(ii) FDIV: Divide; This instruction performs real or integer division. By default ST(0) and ST(1) will be the operands. In other cases either one operand or two operands will be explicitly mentioned. Result will be stored in ST(0) in default case.

Ex1: FDIV ; ST(0) / ST(1) ST(0)
Ex2: FDIV MEM ; ST(0) / (DS:MEM) ST(0)
Ex3: FDIV ST(2), ST(1) ; ST(2) / ST(1) ST(2)

(iii) FABS: Find absolute value. This instruction replaces the content of stack top by its absolute value (magnitude). The sign of the operand if any will be neglected.

(iv) FCHS: Change sign; This instruction changes the sign of contents of stack top and stores it into the same location. The sign of the operand will be complimented.

(v) FLDZ: Load zero; This instruction places 0.0 on to the current top of stack

(b) Assembler directives:

An assembler directive is a command/guideline to the assembler in order to order to control the process of program assembly. Assembler directive do not have any equivalent codes and hence do not find a place in Object file (appear only in Source file).

(i) GLOBAL

(ii) Global directive is used inform the assembler that the following variables are all global and available to other users also. Variables are declared in the current source, but accessible to any other program. Ex: GLOBAL DISPLAY, CUR-TEMP, etc.

(ii) EQU- Equate- EQU is used to give a name to some value or symbol. Each time the assembler finds the given name in the program it will replace the name with the value or symbol you equated with that name. Suppose, for example, you write the statement CORRECTION_FACTOR EQU 03H at the start of your program and later in the program you write the instruction statement ADD AL, CORRECTION_FACTOR. When it codes this instruction statement, the assembler will code it as if you had written the instruction ADD AL, 03H. The advantage of using EQU in this manner is that if CORRECTION_FACTOR is used 27 times in a program, and you want to change the value, all you have to do is change the EQU

statement and reassemble the program. The assembler will automatically put in the new value each time it finds the name CORRECTION_ FACTOR. If you had used 03H instead of the EQU approach, then you would have to try to find and change all 27 instructions yourself. Here are some more examples.

```
DECIMAL_ADJUST EQU DAA    :    Create clearer
                          :    mnemonic for DAA
STRING_START EQU [BX]:    Give name to [BX]
```

(iii) TITLE: This directive is used to give title to a source program. Up to 60 characters can be used following the directive. Ex: TITLE 8086 program for addition.

(iv) ENDP- End Procedure-

This directive is used along with the name of the procedure to indicate the end of a procedure to the assembler. This directive, together with the procedure, directive, PROC, is used to "bracket" a procedure. Here's an example.

```
SQUARE_ROOT PROC    :    Start of procedure
                    :    Procedure instruction
                    :    Statements
SQUARE_ROOT ENDP    :    End of procedure
```

Q7 (a) Explain DOS operating system services.

Answer Page Number 293 – 294 of Textbook I

Q7 (b) Write a 8086 program to check user entry for password.

Answer Page Number 344 – 349 of Textbook I

Q7 (c) Explain high level language services for BIOS and DOS services.

Answer Page No 294 of Textbook I

Q8 (b) Write a C program to print a message, if the printer is online using BIOS and DOS services. Explain the methodology.

Answer Page Number 372 - 374 of Textbook I

Q9 (a) Explain the following for 80386 processor:-
(i) Segmentation
(iii) Paging

Answer Page Number 516 - 520 of Textbook II

Q9 (b) Draw the architecture of Pentium processor and explain the features of Pentium processor.

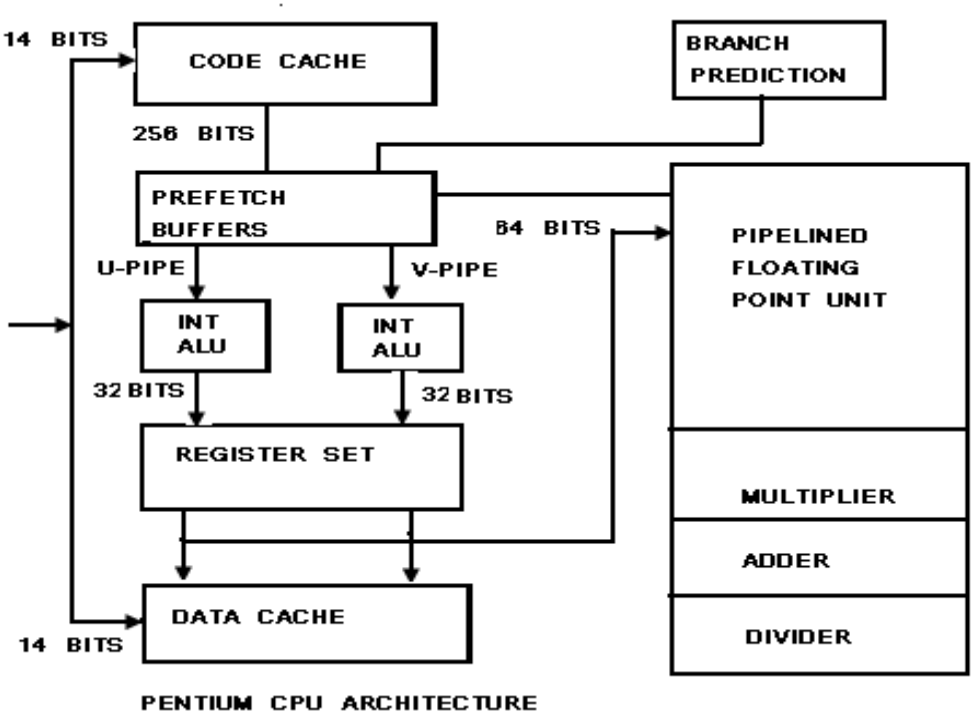
Answer

Architecture of Pentium processor:

*Super scalar (parallel) architecture, two integer pipe lines U&V

*On chip floating point unit.

*RISC performance, CISC instruction set.
*8 Kbytes of Instruction cache plus 8Kbytes of data cache on chip.
*System Memory Management mode of operation. (SMM)
*BIST-Power on self check feature is present.
*Paging unit 4 Mbytes instead of 4 Kbytes.
*Branch prediction feature using branch target buffer.
*8 General purpose floating point registers.
*Multimedia Extension (MMX) capability is present; 57 MMX instructions of the type Single Instruction Multiple Data (SIMD)
*Saturation logic.
*Pentium-II has removed the disadvantages of Pentium-I namely linear Instruction sequencing(Fetch, Decode, Execute).It uses dynamic execution technology. Speculative execution, out of turn execution & multiple branch prediction.
*Large cache (256K & 512K), 2.8 V operations.
*Pentium III used is in Image processing, Multimedia Internet applications. 0.25 micron technology is used. Over 9.5 million transistors are present. 450MHz, 500MHz &550 MHz, 1 GHz operations are possible. Dual independent bus architecture increases bandwidth and speed.
*Now we have Pentium (2.2GHz) 0.13 micron technology, 55 million transistors on a chip with applications in digital music, photography and video.
*XEON processors used in multiprocessor servers, has level three cache up to 1 Mbyte. Other uses are online transaction processing etc.,
*The latest is ITANIUM for mission critical and data intensive applications.



PENTIUM CPU ARCHITECTURE

Text Books

1. Advanced Microprocessors & IBM-PC Assembly Language Programming, K. Udaya Kumar and B.S. Umashankar, TMH, 1996

2. Advanced Microprocessors and Peripherals, A.K. Ray and K.M. Burchandi, TMH, 2000