Microprocessor May 18

1a. Explain Memory banks for 8086 Processor.?
The 8086 has 20-bit address bus, so it can address 2^20 or 1,048,576 addresses. Each address represents a stored byte. To make it possible to read or write a word with one bus cycle, the memory for an 8086 is set up in to 2 banks of up to 524,288 bytes each. 8086 Memory Banks. One memory bank contains all the bytes which have even addresses such as 00000h, 00002h, and 00004h etc. the data lines of this bank is connected to the lower 8 bit data lines i.e. from D0 to D7 of 8086.

The other memory bank contains all bytes which have odd addresses such as 00001h, 00003h and 00005h etc. the data lines of this bank is connected to the upper 8 bit data lines i.e. from D8 to D15 of 8086.

Address line A0 is used for enabling the memory device in the lower bank. An addressed memory device in this bank will be enabled when A0 is low, as it will be for any even address. Like address 00222h = 0000 0000 0010 0010 0010. Address lines A1 to A19 are used to select the desired memory device in the bank and hence the desired byte in the device.Address line A1 to A19 are also used to select the desired memory device in the upper bank and hence the desired byte. An additional part of enabling the upper bank memory device is handled by the BHE i.e. the bus high enable signal. This is multiplexed out from the 8086 at the same time as an address is sent out. An external latch, strobed by the ALE signal, grabs the BHE signal and holds it stable for the rest of the machine cycle like it does for the address.

So now if we read a byte from or write a byte to an even address the A0 will be low BHE will be high enabling the lower bank and disabling the upper bank.

(Leave space for diagram)

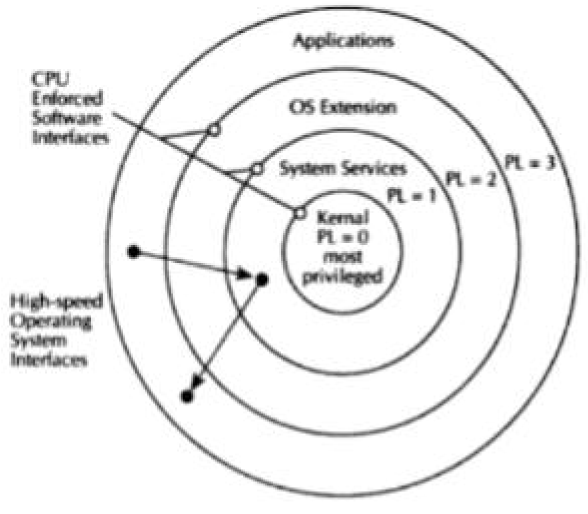
1b. Explain Multitasking and protection for 80386 processor

The selector is used to specify an index into a table defined by the operating system. The table includes the 32-bit base address of a given segment. The physical address is obtained by summing the base address obtained from the table with the offset. With the paging mechanism enabled, the 80386 provides an additional memory management mechanism. The paging feature manages large 80386 segments. The paging mechanism translates the protected linear addresses from the segmentation unit into physical addresses. Figure below shows this translation scheme.

Segmentation provides both memory management and protection. All information about the segments is stored in an 8-byte data structure called a descriptor. All the descriptors are stored in tables identified by the 80386 hardware. There are three types of tables holding 80386 descriptors: global descriptor table (GDT), local descriptor table (LDT), and interrupt descriptor table (I DT).The 80386 provides four protection levels for supporting a multitasking operating system to isolate and protect user programs from each other and the operating system.

The privilege level controls the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. The 80386 includes the protection as part of its memory management unit.

It is an extension of the user/ supervisor privilege mode used by minicomputers. Note that the user/supervisor mode is supported by the 80386 paging mechanism. The Privilege Levels (PL) is numbered 0 to 3. Level 0 is the most privileged level.

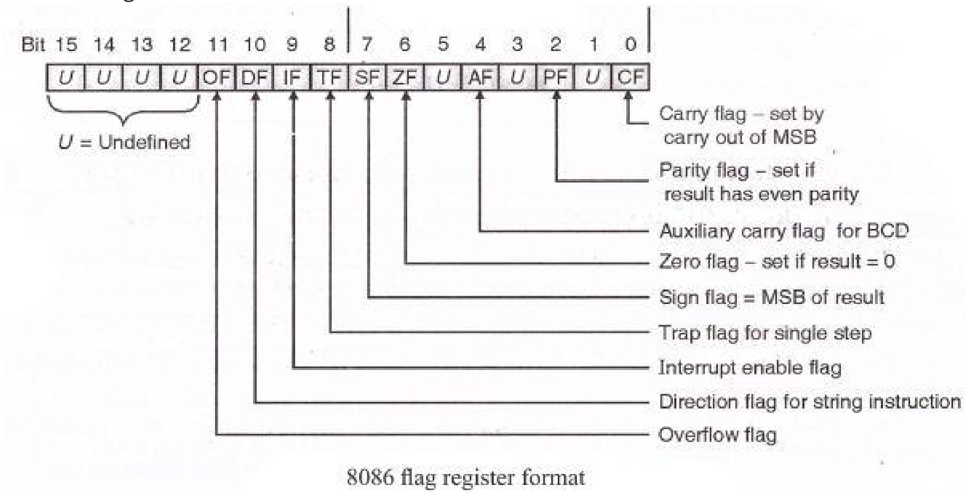

Applications
OS Extension
System Services
Kernal PL = 0 most privileged
PL = 1
PL = 2
PL = 3
CPU Enforced Software Interfaces
High-speed Operating System Interfaces

Four level hierarchical protection

1c. Explain flag register bits of 8086

1. It consists of 9 active flags out of 16. The remaining 7 flags marked ‘U’ are undefined flags.
2. These 9 flags are of two types:

- 6 Status flags
- 3Controlflags


Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
U U U U OF DF IF TF SF ZF U AF U PF U CF
U = Undefined
Carry flag – set by carry out of MSB
Parity flag – set if result has even parity
Auxiliary carry flag for BCD
Zero flag – set if result = 0
Sign flag = MSB of result
Trap flag for single step
Interrupt enable flag
Direction flag for string instruction
Overflow flag

8086 flag register format

Status flags:

Carry flag (CY)-It is set whenever there is a carry or borrow out of the MSB (most significant bit) of a result. D7 bit for an 8 bit operation and D15 bit for a 16 bit operation.

Parity flag (PF)-It is set if the result has even parity. If parity is odd, PF is reset.This flag is normally used for data transmission errors.

Auxiliary carry flag (AC)-It is set if a carry is generated out of the lower nibble.It is used only in 8 bit operations like DAA and DAS.

Zero flag (ZF)-It is set if the result is zero.

Sign flag (SF)-It is set if the MSB of the result is 1. For signed operations such a number is treated as negative.

Overflow flag (OF)-It will be set if the result of a signed operation is too large to fit in the number of bits available to represent it. It can be checked using the instruction INTO (Interrupt on Overflow).

Control flags:

Trap flag (TF)-It is used to set the trace mode i.e. start single stepping mode.Here the microprocessor is interrupted after every instruction so that the program can be debugged.

Interrupt enable flag (IF)-It is used to mask (disable) or unmask (enable) the INTR interrupt.If user sets IF flag, the CPU will recognize external interrupt requests. Clearing IF disables these interrupts.

Direction flag (DF)-If this flag is set, SI and DI are in auto-decrementing mode in string operations.

1d. Explain Virtual Mode (VM86) 80386 processor

In protected mode of operation, 80386DX provides a virtual 8086 operating environment to execute the 8086 programs

The real mode can also use to execute the 8086 programs along with the capabilities of 80386, like protection and a few additional instructions.

Once the 80386 enters the protected mode from the real mode, it cannot return back to the real mode without a reset operation.

Thus, the virtual 8086 mode of operation of 80386, offers an advantage of executing 8086 programs while in protected mode.

V86 Mode is also known as Virtual Mode of 80386.

V86 Mode is a Dynamic Mode.

It can switch repeatedly & rapidly between V86 Mode & Protected Mode.

To execute an 8086 program, the CPU enters in V86 Mode from Protected Mode.

CPU Leaves V86 Mode and enters protected mode to continue executing a native 80386 program.

The address forming mechanism in virtual 8086 mode is exactly identical with that of 8086 real mode.

In virtual mode, 8086 can address 1MB of physical memory that may be anywhere in the 4GB address space of the protected mode of 80386.

Like 80386 real mode, the addresses in virtual 8086 mode lie within 1MB of memory.

In virtual mode, the paging mechanism and protection capabilities are available at the service of the programmers.

The virtual mode allows the multiprogramming of 8086 applications.

The virtual 8086 mode executes all the programs at privilege level 3.

4a. Explain following instructions.

DAA: Adjusts the sum of two packed BCD values to create a packed BCD result. The AL register is the implied source and destination operand. The DAA instruction is only useful when it follows an ADD instruction that adds (binary addition) two 2-digit, packed BCD values and stores a byte result in the AL register. The DAA instruction then adjusts the contents of the AL register to contain the correct 2-digit, packed BCD result. If a decimal carry is detected, the CF and AF flags are set accordingly.The CF and AF flags are set if the adjustment of the value results in a decimal carry in either digit of the result (see the "Operation" section above). The SF, ZF, and PF flags are set according to the result. The OF flag is undefined.

AAA: Adjusts the sum of two unpacked BCD values to create an unpacked BCD result. The AL register is the implied source and destination operand for this instruction. The AAA instruction is only useful when it follows an ADD instruction that adds (binary addition) two unpacked BCD values and stores a byte result in the AL register. The AAA instruction then adjusts the contents of the AL register to contain the correct 1-digit unpacked BCD result.

If the addition produces a decimal carry, the AH register increments by 1, and the CF and AF flags are set. If there was no decimal carry, the CF and AF flags are cleared and the AH register is unchanged. In either case, bits 4 through 7 of the AL register are set to 0.

This instruction executes as described in compatibility mode and legacy mode. It is not valid in 64-bit mode.

XLAT: Locates a byte entry in a table in memory, using the contents of the AL register as a table index, then copies the contents of the table entry back into the AL register. The index in the AL register is treated as an unsigned integer. The XLAT and XLATB instructions get the base address of the table in memory from either the DS:EBX or the DS:BX registers (depending on the address-size attribute of the instruction, 32 or 16, respectively). (The DS segment may be overridden with a segment override prefix.)

At the assembly-code level, two forms of this instruction are allowed: the "explicit-operand" form and the "no-operand" form. The explicit-operand form (specified with the XLAT mnemonic) allows the base address of the table to be specified explicitly with a symbol. This explicit-operands form is provided to allow documentation; however, note that the documentation provided by this form can be misleading. That is, the symbol does not have to specify the correct base address. The base address is always specified by the DS:(E)BX registers, which must be loaded correctly before the XLAT instruction is executed.

The no-operands form (XLATB) provides a "short form" of the XLAT instructions. Here also the processor assumes that the DS:(E)BX registers contain the base address of the table.

In 64-bit mode, operation is similar to that in legacy or compatibility mode. AL is used to specify the table index (the operand size is fixed at 8 bits). RBX, however, is used to specify the table's base address. See the summary chart at the beginning of this section for encoding data and limits.

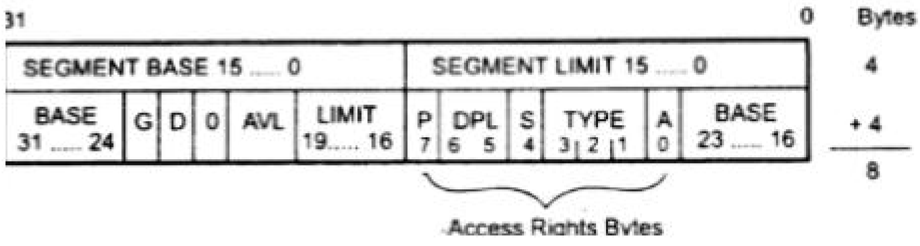
4b. Explain Segment Descriptor of 80386 Processor

5a. Explain Gate type of descriptors.

(both the questions have same answer)

1. n protected mode, memory management unit (MMU) uses the segment selector to access a descriptor for the desired segment in a table of descriptors in memory.
2. Segment descriptor is a special structure which describes the segment. Exactly one segment descriptor must be defined for each segment of the memory.
3. Descriptors are eight type quantities which contain attributes about a given region of linear address space (i.e. a segment).
4. These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment.
5. Fig. below shows the general format of a descriptor. As shown in Fig. segment descriptor has following fields.

6. Base: It contains the 32-bit base address for a segment. Thus defines the location of the segment within the 4 gigabyte linear address space. The 80386 concatenates the three fragments of the base address to form a single 32-bit address.
7. Limit: It defines the size of the segment. The 80386 concatenates the two fragments of the limit field to form a 20 bit value. The 80386 interprets this 20-bit value in two ways, depending on the setting of the granularity bit (G) :
    1. If G bit 0: In units of one byte, to define a limit of up to 1 M byte (220)
        1. If G bit 1: In units of 4 kilobytes, to define a limit of up to 4 gigabytes.
8. Granularity Bit: It specifies the units with which the limit field is interpreted. When bit is 0, the limit is interpreted in units of one byte; otherwise limit is interpreted in units of 4 Kbytes.
9. D (Default size): When this bit is cleared, operands contained within this segment are assured to be 16 bits in size. When it is set, operands are assumed to be 32-bits.



5b. Data Cache architecture for Pentium.

1. Despite the potential advantages of a unified cache which is used in the 80486 processor, the Pentium microprocessor uses separate code and data caches.
2. The reason is that the superscalar design and branch prediction demand more bandwidth than a unified cache.
3. First, efficient branch prediction requires that the destination of a branch be accessed simultaneously with data references of previous instructions executing in the pipeline.
4. Second, the parallel execution of data memory references requires simultaneous accesses for loads and stores.
5. Third, in the context of the overall Pentium microprocessor design, handling self-modifying code for separate code and data caches is only marginally more complex than for a unified cache.
6. The data and instruction caches on the Pentium processor are each 8 KB, two-way associative designs with 32 byte lines. Each cache has a dedicated translation lookaside Buffer (TLB) to translate linear addresses to physical addresses.
7. The caches can be enabled or disabled by software or hardware. The Pentium microprocessor implements the data cache to supports dual accesses by the U-pipe and V-pipe to provide additional bandwidth and simplify compiler instruction scheduling algorithms.
8. The data cache is write back or write through configured on a line-by-line basis and follows the MESI protocol. The data cache tags arc triple ported to support two data transfers and an inquire cycle in the same clock.
9. The code cache is an inherent write protected cache. The code cache tags of the Pentium processor are also triple ported to support snooping and split-line accesses.
10. The data path, however, is single ported with eight way interleaving of 32-bit-wide banks. When a bank conflict occurs, the U-pipe assumes software or hardware.

11. The Pentium microprocessor implements the data cache to supports dual accesses by the U-pipe and V-pipe to provide additional bandwidth and simplify compiler instruction scheduling algorithms.
12. The data cache is write back or write through configured on a line-by-line basis and follows the MESI protocol.
13. The data cache tags are triple ported to support two data transfers and an inquire cycle in the same clock.
14. The code cache is an inherently write protected cache.
15. The code cache tags of the Pentium processor are also triple ported to support snooping and split-line accesses.
16. The data path, however, is single ported with eight way interleaving of 32-bit-wide banks.
17. When a bank conflict occurs, the U-pipe assumes priority, and the V-pipe stalls for a clock cycle.
18. The bank conflict logic also serves to eliminate data dependencies between parallel memory references to a single location.
19. For memory references to double-precision floating-point data, the processor accesses consecutive banks in parallel, forming a single 64-bit path.
20. Translation lookaside buffers (TLB): Besides general-purpose caches, X86 processors include caches called Translation Lookaside Buffers (TLB) to speed up linear address translation. When a linear address is used for the first time, the corresponding physical address is computed through slow accesses to the page tables in RAM. The physical address is then stored in a TLB entry so that further references to the same linear address are quickly translated. When the CR_3 control register is modified, the hardware automatically invalidates all entries of the TLB.

6a. Explain SPARC Processor with block diagram.

It is an open processor architecture.(i.e. Member companies to the SPARC community can freely produce the processor)SUN ULTRA SPARCv9 is a robust RISC architecture with-64 bit integer address and data-Superscalar implementations-Extremely fast trap handling and context switching.The presentation will look in detail to the SUN Microsystem's Ultra SPARC III v9 architecture.The processor's micro-architecture designhas six major functional units that performrelatively independently:Instruction issue unit (IIU)Floating point unit (FPU)Integer execution unit (IEU)Data cache unit (DCU)External memory unit (EMU)System interface unit (SIU)The units communicate requests and results among themselves through well-defined interface protocols, as the next figure

Instruction issue unit:

This unit feeds the execution pipelines with the instructions.It independently predicts the control flow through a program and fetches the predicted path from the memory system.Fetched instructions are staged in a queue before forwarding to the two execution units: 'integer and floating point'This unit includes:32-Kbyte, four-way associative 'Instruction cache"The instruction address translation buffer'A 16 K-entry 'branch predictor'

Pipleine and data:

Pipeline feature ParameterInstruction issue integer2 float point2 graphicsLevel-one(L1) caches Data: 64-Kbyte, 4-wayInstructions: 32-Kbyte, 4-wayPrefetch: 2-Kbyte,4-wayWrite : 2-Kbyte,4-wayLevel-two(L2) cache Unified (data and instructions)4- and 8-Mbyte,1-wayOn-chip tags;off chip data

6b Explain with block diagram PIT 8254

The Intel 8253 and 8254 are Programmable Interval Timers (PTIs) designed for microprocessors to perform timing and counting functions using three 16-bit registers. Each counter has 2 input pins, i.e. Clock & Gate, and 1 pin for "OUT" output. To operate a counter, a 16-bit count is loaded in its register. On command, it begins to decrement the count until it reaches 0, then it generates a pulse that can be used to interrupt the CPU.

The most prominent features of 8254 are as follows −

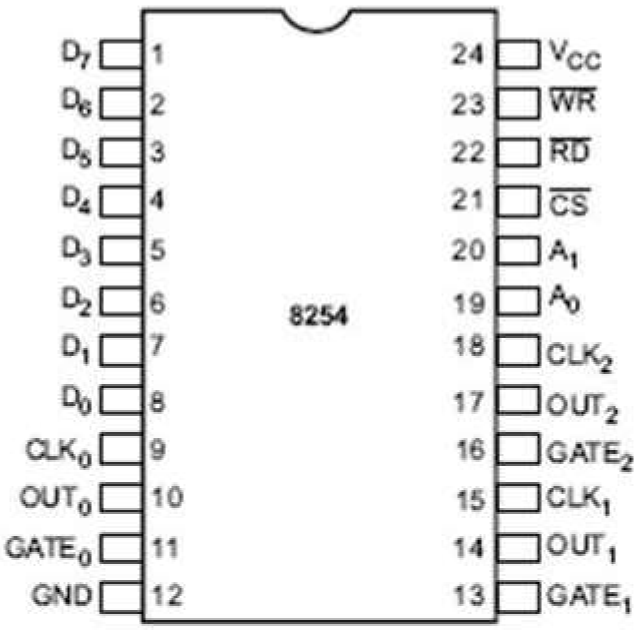
It has three independent 16-bit down counters.

It can handle inputs from DC to 10 MHz.

These three counters can be programmed for either binary or BCD count.

It is compatible with almost all microprocessors.

8254 has a powerful command called READ BACK command, which allows the user to check the count value, the programmed mode, the current mode, and the current status of the counter.


8254
D7 1
D6 2
D5 3
D4 4
D3 5
D2 6
D1 7
D0 8
CLK0 9
OUT0 10
GATE0 11
GND 12
24 VCC
23 WR
22 RD
21 CS
20 A1
19 A0
18 CLK2
17 OUT2
16 GATE2
15 CLK1
14 OUT1
13 GATE1