**PART B**

***Qn12(a): Interfacing of 10 bit DAC***

DAC requires three elements – resistor network with appropriate weighting, switches and a reference source

The output can be a current signal or converted into a voltage signal using an operational amplifier

The time required for conversion, called settling time, is dependent on the response time of the switches and the output amplifier.

**Qn 12(b)Interface an 8 Key Keyboard to 8085 through 8255.Use port A as input.Explain the data transfer and control signals.**

Ans: The keyboard and display device can be connected to 8085 through the interfacing device 8255. The 8255 has three ports A, B and C. Here in simple I/O mode, the pushbutton keyboard is connected to Port A as an input port and seven segment LED is connected to Port B as an output port. The ports address range from FCH to FFH. When a push button key is pressed, it bounces (makes & breaks contact) a few times before it makes a firm contact. To prevent multiple readings of the same key, it is necessary to debounce the key. The hardware solution to this problem is to use a key debounce circuit and the software solution is to wait for 10 to 20ms until the key is settled and check the key again.

The display circuit is connected to the Port B of 8255A. To display a digit, it is necessary to convert it into appropriate seven segment code of the LED. The appropriate binary code can be obtained from the look table stored in memory location.

**Figure**

Refer PDF file 8255a.pdf

Keyboard

The keys K7-K0 are tied high through 10K resistors, and when a key is pressed, the corresponding line is grounded. When all keys are open and 8085 reads port A, the reading on the data bus will be FFH.

For Eg: if K7 is pressed, the output of the port A will be 0111 1111(7FH). This reading should be encoded into the binary equivalent of the digit ’7’ using software routines.

The subroutines KYCHK and KYCODE accomplish the task of checking a key pressed and encoding the key in appropriate binary format. The subroutines first check whether all keys are open. This prevents reading the same key repeatedly if someone were to hold the key for a long time. Then it checks for a key closure, debounces the key, and places the reading in accumulator. i.e when a key is pressed, it will check whether key has pressed and wait for 10-20ms for a key debounce. It also complements the accumulator contents for conversion.

The KYCODE converts (encodes) the pushed key to the binary key code. It establishes the relationship between the hardware and the number of the key. If key K7 is pressed, the reading in the accumulator will be 1000 0000. The KYCODE routine sets the register C for the count of eight and immediately decrements the count to seven. The RAL in KYCODE places the bit D7 in the CY flag, and the next instruction checks for the carry flag. IF it is set, the key K7 must be pressed, and the keycode ‘7’ is in the register C. If CY=0, the program loops back to check the next bit D6.

Seven Segment Display

A common anode seven segment LED is connected to port B through the driver 74LS244. The driver is necessary to increase the current capacity of port B. The device functions as current amplifier; whatever logic at port B will be at the output of driver. To display the number of the key pressed, a routine is necessary that will send an appropriate code to port B. The routine KYCODE supplies the binary number of the key pressed; however there is no relationship between the binary code of the digit and its seven segment code.

The look up table technique is used to find the code for the digit supplied by the KYCODE. In look-up table technique, the codes for the digits to be displayed are stored sequentially in memory. The conversion program locates the code of the digit based on its magnitude and transfers the code to the MPU to send out to a display port.

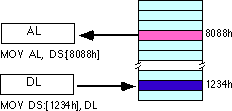
Software Approach

To reduce the cost and the chip count in a product, a matrix keyboard and scanned display are combined often. If a scanned display is used with software driven matrix keyboard, the keyboard subroutine must be coupled with the scanned display; otherwise display must go off. The program must alternate between refreshing the display and checking the keyboard to find the key pressed. The time needed to check the keyboard press is relatively short and so it doesn’t affect display.

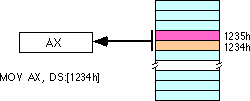
**13(a). Explain the various addressing modes used in 8086. Give examples**

### *The Displacement Only Addressing Mode*

The most common addressing mode, is the displacement-only (or direct) addressing mode. The displacement-only addressing mode consists of a 16 bit constant that specifies the address of the target location. The instruction mov al,ds:[8088h] loads the al register with a copy of the byte at memory location 8088h. Likewise, the instruction mov ds:[1234h],dl stores the value in the dl register to memory location 1234h:



The displacement-only addressing mode is perfect for accessing simple variables.



By default, all displacement-only values provide offsets into the data segment. If we want to provide an offset into a different segment, we must use a segment override prefix before the address. For example, to access location 1234h in the extra segment (es) we would use an instruction of the form mov ax,es:[1234h]. Likewise, to access this location in the code segment we would use the instruction mov ax, cs:[1234h]. The ds: prefix in the previous examples is not a segment override. The CPU uses the data segment register by default. These specific examples require ds: because of MASM's syntactical limitations.

### *The Register Indirect Addressing Modes*

In 80x86, memory can be accessed indirectly through a register using the register indirect addressing modes. There are four forms of this addressing mode on the 8086, best demonstrated by the following instructions:

mov al, [bx]

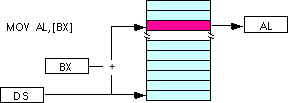
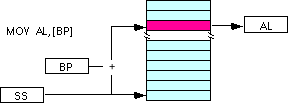
mov al, [bp]

mov al, [si]

mov al, [di]

As with the x86 [bx] addressing mode, these four addressing modes reference the byte at the offset found in the bx, bp, si, or di register, respectively. The [bx], [si], and [di] modes use the ds segment by default. The [bp] addressing mode uses the stack segment (ss) by default.

Intel refers to [bx] and [bp] as base addressing modes and bx and bp as base registers (in fact, bp stands for base pointer). Intel refers to the [si] and [di] addressing modes as indexed addressing modes (si stands for source index, di stands for destination index). However, these addressing modes are functionally equivalent. This text will call these forms register indirect modes to be consistent.

Note: the [si] and [di] addressing modes work exactly the same way, just substitute si and di for bx above.

### *Indexed Addressing Modes*

The indexed addressing modes use the following syntax:

mov al, disp[bx]

mov al, disp[bp]

mov al, disp[si]

mov al, disp[di]

If bx contains 1000h, then the instruction mov cl,20h[bx] will load cl from memory location ds:1020h. Likewise, if bp contains 2020h, mov dh,1000h[bp] will load dh from location

ss:3020.

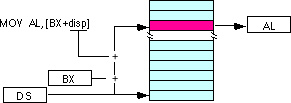
The offsets generated by these addressing modes are the sum of the constant and the specified register. The addressing modes involving bx, si, and di all use the data segment, the disp[bp] addressing mode uses the stack segment by default. As with the register indirect addressing modes, we can use the segment override prefixes to specify a different segment:

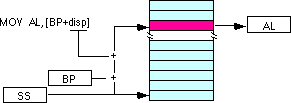
mov al, ss:disp[bx]

mov al, es:disp[bp]

mov al, cs:disp[si]

mov al, ss:disp[di]





We may substitute si or di in the figure above to obtain the [si+disp] and [di+disp] addressing modes.  
  
*Based Indexed Addressing Modes*

The based indexed addressing modes are simply combinations of the register indirect addressing modes. These addressing modes form the offset by adding together a base register (bx or bp) and an index register (si or di). The allowable forms for these addressing modes are

mov al, [bx][si]

mov al, [bx][di]

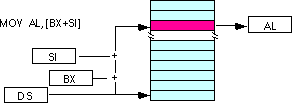
mov al, [bp][si]

mov al, [bp][di]

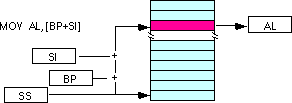
Suppose that bx contains 1000h and si contains 880h. Then the instruction

mov al,[bx][si]

would load al from location DS:1880h. Likewise, if bp contains 1598h and di contains 1004, mov ax,[bp+di] will load the 16 bits in ax from locations SS:259C and SS:259D.  
  
The addressing modes that do not involve bp use the data segment by default. Those that have bp as an operand use the stack segment by default.



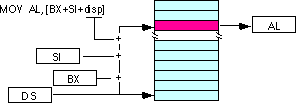
By substituting di in the figure above, we can obtain the [bx+di] addressing mode.



We substitute di in the figure above for the [bp+di] addressing mode.

### *Based Indexed Plus Displacement Addressing Mode*

These addressing modes are a slight modification of the base/indexed addressing modes with the addition of an eight bit or sixteen bit constant. The following are some examples of these addressing modes:



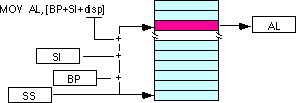
mov al, disp[bx][si]

mov al, disp[bx+di]

mov al, [bp+si+disp]

mov al, [bp][di][disp]

di may be substituted in the figure above to produce the [bx+di+disp] addressing mode.



Suppose bp contains 1000h, bx contains 2000h, si contains 120h, and di contains 5. Then mov al,10h[bx+si] loads al from address DS:2130; mov ch,125h[bp+di] loads ch from location SS:112A; and mov bx,cs:2[bx][di] loads bx from location CS:2007.

**14(b)How is is the exception processing done in 80286?Explain in detail by mentioning the various possible exception.**

***EXCEPTION PROCESSING DONE IN 80286***

Conditions that force the microprocessor out of its normal processing sequence ,are known as **exceptions**. Each processor differs in the detailed sequence of actions it performs after detecting an exception. However,in general,the following steps are performed:

**1)**save the current state of micro processor.

**2)**Determine the address of the exception handler routine.

**3)**Branch to the exception handler and resume instruction excecution.

There are many

variations of the words which are used to label an

(unmasked) FPU error condition, and also the

code which handles it. “Error”, “exception” and

“fault” are used to refer to the condition. Such a

condition results in an interrupt, if no mask or

block is in effect along the interrupt pathway. The

code which handles the interrupt can be referred

to as an error or exception or fault handler, or an

interrupt or exception service routine, etc. The

phrase “**exception handler**” has been used

consistently (as much as possible) in this

application note, for several reasons: “Exception”

is less general than interrupt (which includes

external hardware i The exception-handling routine is normally a part of

the systems software. The routine must clear (or

disable) the active exception flags in the FPU status

word before executing any FP instructions that

cannot complete execution when there is a pending

FP exception. Otherwise, the FP instruction will

trigger the FPU interrupt again, and the system will

be caught in an endless loop of nested FP

exceptions, and hang. In any event, the routine

must clear (or disable) the active exception flags in

the FPU status word after handling them, and

before IRET(D). Typical exception responses may

include:

Incrementing an exception counter for later

display or printing

Printing or displaying diagnostic information

(e.g., the FPU environment and registers)

Aborting further execution, or using the

exception pointers to build an instruction that

will run without exception and executing it

**Additional Exceptions**

The 80386 defines new exceptions that can occur even in systems designed for the 80286.

* Exception #6 -- invalid opcode   
  This exception can result from improper use of the LOCK instruction.
* Exception #14 -- page fault   
  This exception may occur in an 80286 program if the operating system enables paging. Paging can be used in a system with 80286 tasks as long as all tasks use the same page directory. Because there is no place in an 80286 TSS to store the PDBR, switching to an 80286 task does not change the value of PDBR. Tasks ported from the 80286 should be given 80386 TSSs so they can take full advantage of paging.

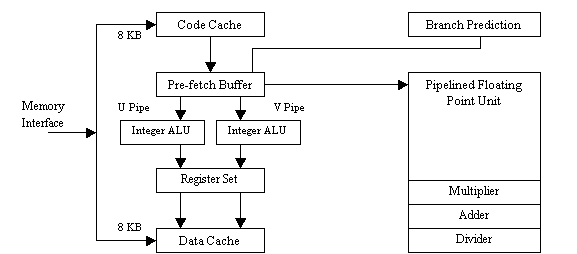
**15(a).Pentium-How execution speed is increased?**

The term ''Pentium processor'' refers to a family of microprocessors that share a common architecture and instruction set. It runs at a clock frequency of either 60 or 66 MHz and has 3.1 million transistors. Some of the features of Pentium architecture are:

* The Pentium processor has been optimized to run critical instructions in fewer clock cycles than the 80486 processor.
* Complex Instruction Set Computer (CISC) architecture with Reduced Instruction Set Computer (RISC) performance.
* 64-Bit Bus
* Upward code compatibility.
* Pentium processor uses Superscalar architecture and hence can issue multiple instructions per cycle.
* Multiple Instruction Issue (MII) capability.
* Pentium processor executes instructions in five stages. This staging, or pipelining, allows the processor to overlap multiple instructions so that it takes less time to execute two instructions in a row.
* The Pentium processor fetches the branch target instruction before it executes the branch instruction.
* The Pentium processor has two separate 8-kilobyte (KB) caches on chip, one for instructions and one for data. It allows the Pentium processor to fetch data and instructions from the cache simultaneously.
* When data is modified, only the data in the cache is changed. Memory data is changed only when the Pentium processor replaces the modified data in the cache with a different set of data.

The Pentium processor has two primary operating modes -

1. **Real-Address Mode** - This mode provides the programming environment of the Intel 8086 processor, with a few extensions. Reset initialization places the processor in real mode where, with a single instruction, it can switch to protected mode
2. **Protected Mode** - In this mode all instructions and architectural features are available, providing the highest performance and capability. This is the recommended mode that all new applications and operating systems should target.



**Architecture of Pentium**

The main factors responsible for increased execution speed are:

a) Cache structure:

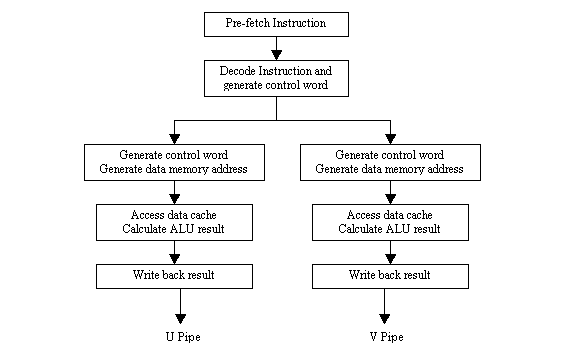
Pentium contains two 8KB cache memories- 8KB instruction cache and an 8KB data cache. The instruction cache store only instructions and data cache stores the data used by instructions.

b) Superscalar Architecture :

A salient feature of Pentium is its superscalar, super pipelined architecture. It has two integer pipelines U and V where each one is a 4 stage pipeline. This enhances the speed of integer arithmetic of Pentium. Also it has an on chip floating point unit which eliminates the need for other coprocessors for floating point operations. This therefore increases the performance of the processor.

The Pentium is organized with three execution units. One executes floating point units and the other two (called U-pipe and V-pipe) execute integer instructions. This means it is possible to execute three instructions simultaneously. This results in 40 percent of improvement in execution speed. Each of these pipelines has 5 stages. The Pentium's basic integer pipeline is five stages long, with the stages broken down as follows:

1. **Pre-fetch/Fetch** : Instructions are fetched from the instruction cache and aligned in pre-fetch buffers for decoding.
2. **Decode1** : Instructions are decoded into the Pentium's internal instruction format that is control word. Branch prediction also takes place at this stage.
3. **Decode2** : Decodes it for final execution and also generate the address for data reference.
4. **Execute** : The integer hardware executes the instruction.
5. **Write-back** : The results of the computation are written back to the register file.

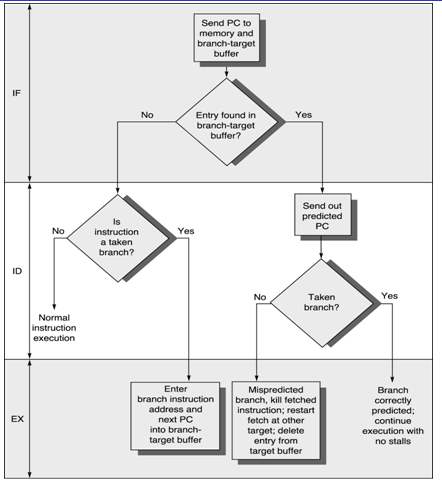


**Pentium pipeline stages**

c) Branch prediction logic

Another feature of Pentium is branch prediction logic. Here, the most likely set of instructions to be executed are predetermined, and the pipelines are kept full accordingly. It reduces the time required for a branch caused by internal delays. When a branch is encountered, the microprocessor begins prefetch instruction at the branch address. The instructions are loaded into the instruction cache, so that to execute in one clocking period.

The branch instruction changes the normal sequential control flow of the program and may stall the pipelined execution in Pentium system. Pentium designers implemented a branch prediction algorithm for speed up of the instruction execution. A 256 entry branch target buffer in Pentium CPU holds branch target addresses for previously executed branches. Whenever, a branch is taken, the CPU enters the branch instruction address and also destination address in the BRB. When an instruction decoded, the CPU searches the BTB to determine whether there exist any entry for the corresponding branch instruction. If there is a hit, then the CPU uses the history to decide whether the branch will be taken or not. If the CPU, based upon its previous history decides to take the branch, it fetches the instruction from the target address and decodes them. If the prediction is correct, the process continues. If the prediction is incorrect, CPU flushes the pipeline and fetches from the correct target address.

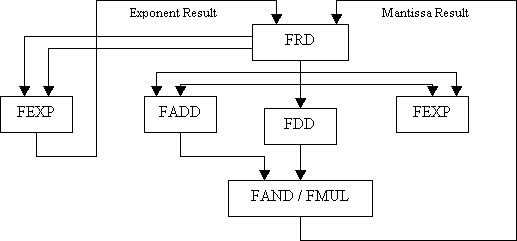


d) High performance floating Point Unit

The processor has an on-chip floating point unit, which is capable of executing floating point instructions five to ten times faster than 80486.

**Floating Point Unit** :

There are 8 general-purpose 80-bit Floating point registers. Floating point unit has 8 stages of pipelining. First five are similar to integer unit. Since the possibility of error is more in Floating Point unit (FPU) than in integer unit, additional error checking stage is there in FPU. The floating point unit is shown as below:



**Floating Point Unit**

FRD - Floating Point Rounding   
FDD - Floating Point Division   
FADD - Floating Point Addition   
FEXP - Floating Point Exponent   
FAND - Floating Point And.   
FMUL - Floating Point Multiply

**Floating point exception**

There are 5 possible floating point exception:

1. Divide by zero
2. Overflow
3. Underflow
4. Denormal operand
5. Invalid operation.

e) Performance monitoring

Processor design enables the user to monitor the performance of the processor and to optimize potential bottlenecks in code execution. User can observe and record time for internal events.

**15(b) draw neat block diagram and explain basic features of 80386. Explain branch prediction logic.**

Module 8 learning unit 18

Architecture of 80386

•The Internal Architecture of 80386 is divided into 3 sections.

•Central processing unit

•Memory management unit

•Bus interface unit

•Central processing unit is further divided into Execution unit and Instruction unit

•Execution unit has 8 General purpose and 8 Special purpose registers which are either used for handling data or calculating offset addresses.



• The multiply / divide logic implements the bit-shift-rotate algorithms to complete the operations in minimum time.

•Even 32- bit multiplications can be executed within one microsecond by the multiply / divide logic.

•The Memory management unit consists of a Segmentation unit and a Paging unit.

•Segmentation unit allows the use of two address components, viz. segment and offset for relocability and sharing of code and data.

•Segmentation unit allows segments of size 4Gbytes at max.

•The Paging unit organizes the physical memory in terms of pages of 4kbytes size each.

•Paging unit works under the control of the segmentation unit, i.e. each segment is further divided into pages. The virtual memory is also organizes in terms of segments and pages by the memory management unit.

•The Segmentation unit provides a 4 level protection mechanism for protecting and isolating the system code and data from those of the application program.

•Paging unit converts linear addresses into physical addresses.

•The control and attribute PLA checks the privileges at the page level. Each of the pages maintains the paging information of the task. The limit and attribute PLA checks segment limits and attributes at segment level to avoid invalid accesses to code and data in the memory segments.•The Bus control unit has a prioritizer to resolve the priority of the various bus requests. This controls the access of the bus. The address driver drives the bus enable and address signal A0 – A31. The pipeline and dynamic bus sizing unit handle the related control signals.

•The data buffers interface the internal data bus with the system bus.

Branch prediction

There is dynamic and static branch prediction logic on the ARM11 cores to predict conditional direct branches (some indirect branches are also predicted using the Return Stack). The dynamic branch prediction logic keeps a running history of whether a branch was taken, the branch information/history is stored in the Branch Target Address Cache (BTAC). If the branch information is not contained in the BTAC, static branch prediction is used, whereby we assume the branch will be taken if the branch is a conditional backwards branch or not taken if the branch is a conditional forwards branch.

Using a combination of dynamic and branch prediction we can typically correctly predict whether the branch will be taken about 85% of the time. If the branch prediction fails, the core pipeline will need to be refilled and the current instruction pipeline flushed.