

Section 9

Applications Program Design

Chapter 9.1 Programming the Applications Module

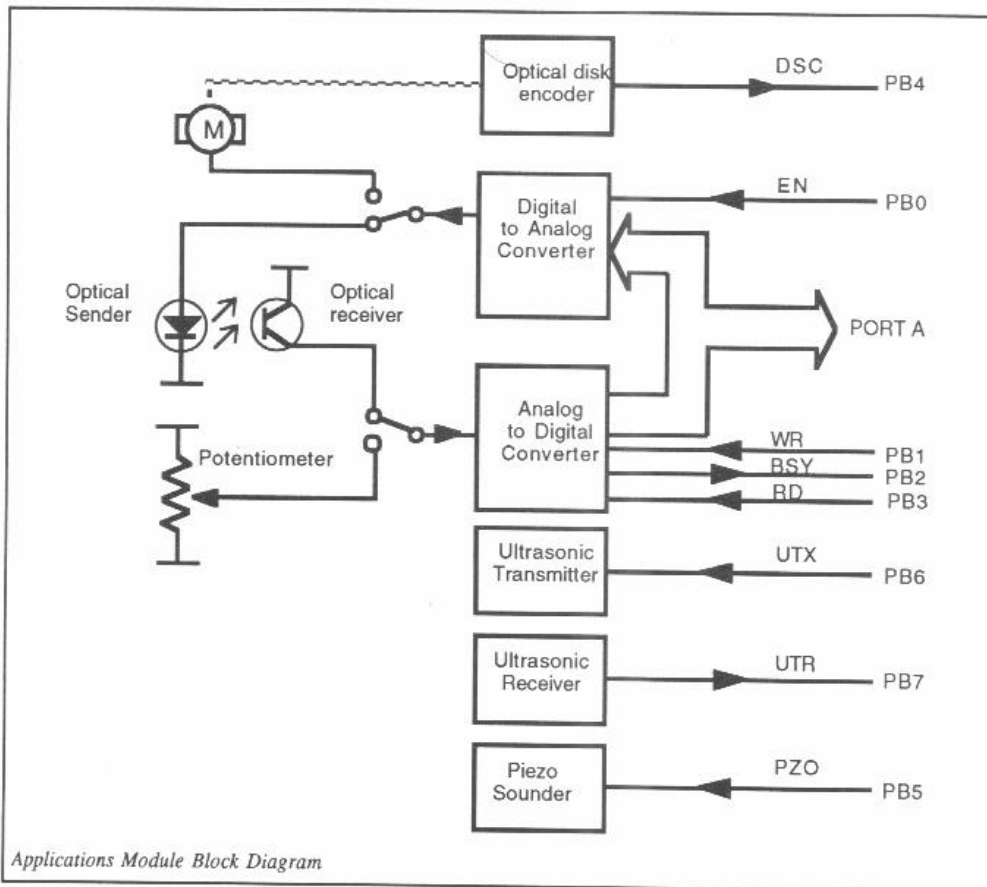
Objectives of this Chapter

Having studied this chapter you will be able to:

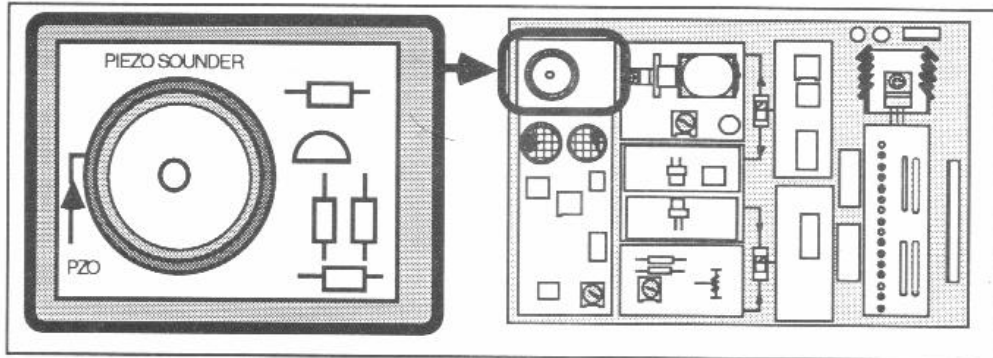
- Describe the operation of each section of the Applications Module:
 - Piezo Sounder
 - Ultrasonic Transmitter and Receiver
 - Digital to Analog Converter
 - Analog to Digital Converter
 - Optical Transmitter and Receiver
 - Optical Disc Encoder
- Write programs to control these sections of the Applications Module.

Introduction

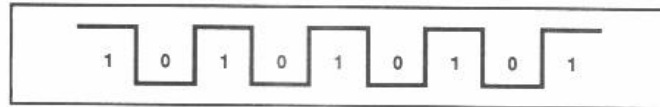
In the first chapter in this manual you learned how to run demonstration programs to control each section of the Applications Module. In other chapters you have learned how to program the microcomputer to make decisions and how to input and output data. In this chapter you will combine these skills and write programs to control each individual section on the Applications Module.



Piezo Sounder



The piezo sounder converts a TTL level waveform on Port 1, bit 5 (P15) into an audio signal of the same frequency. Changing the logic level on P15 with respect to time will generate a TTL waveform thus:



Notes:

.....

.....

.....

.....

.....

.....

.....

.....

.....

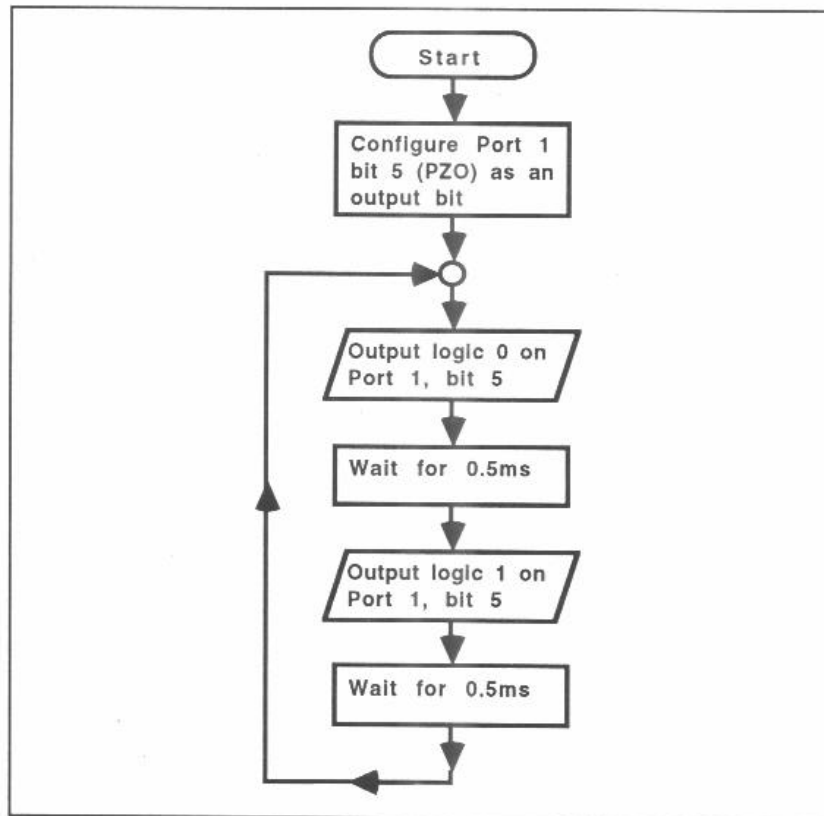
Exercise 21

Write a program which will sound the Piezo Sounder at about 1kHz.

Solution:

This problem requires a squarewave to be output to Port 1, bit 5 (P15). This is achieved by alternating the output between 0 and 1 at a frequency of 1kHz. This will require a delay of 0.5ms between changes of the output state (to give an overall period 1ms).

Flowchart:

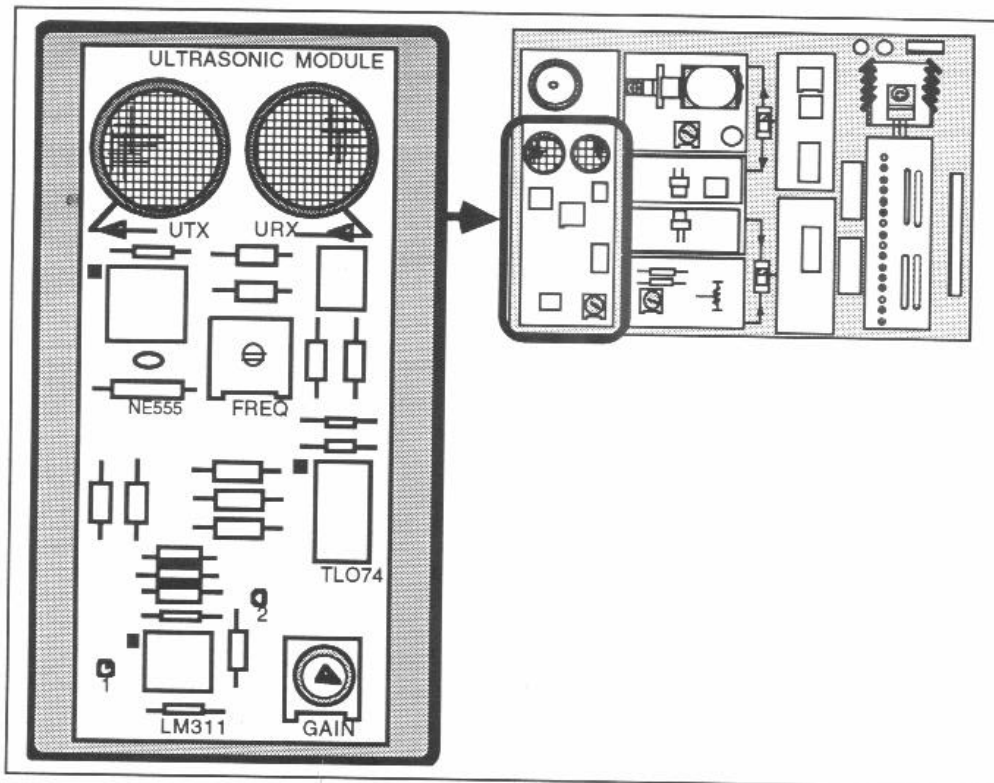


Source Program:

```
ORG 0100H ;Defines the start address for
;object code as 0080:0100H
UPORTCTL1 EQU 088H ;Defines address of Port
;Control Register
UPORT1 EQU 090H ;Defines address of Port 1
VAL1 EQU 0FAH ;Defines "VAL1" AS FAH
MOV AL,20H ;Configures Port 1, bit 5 (P15)
OUT UPORT1CTL,AL ;as an output
OUTPUT: MOV AL,00H ;Outputs a logic "0" on Port 1,
;bit 5 (P15)
OUT UPORT1,AL
DELY1: MOV CX,0FAH ;wait for 0.5ms
LOOP DELY1
MOV AL,20H ;Outputs a logic "1" on Port 1,
;bit 5 (P15)
OUT UPORT1,AL
DELY2: MOV CX,0FAH ;Wait for 0.5ms
LOOP DELY2
JMP OUTPUT ;Jump back to output again
```

Use this source program to produce an object program and transfer to PAT. Try changing the delay value ("VAL1") and observe the effect.

Ultrasonic Transmitter and Receiver



The Ultrasonic Transmitter is driven by a 40kHz oscillator within the Ultrasonic Module. The transmitter is switched on/off by the state of P16 (labelled UTX):

P16 = 1 Transmitter ON
P16 = 0 Transmitter OFF

The Ultrasonic Receiver will detect the 40kHz ultrasound signal and pass an indication to P17 (labelled URX) thus:

No 40kHz Detected: P17 = 1
40kHz Detected: P17 = 40kHz TTL Squarewave

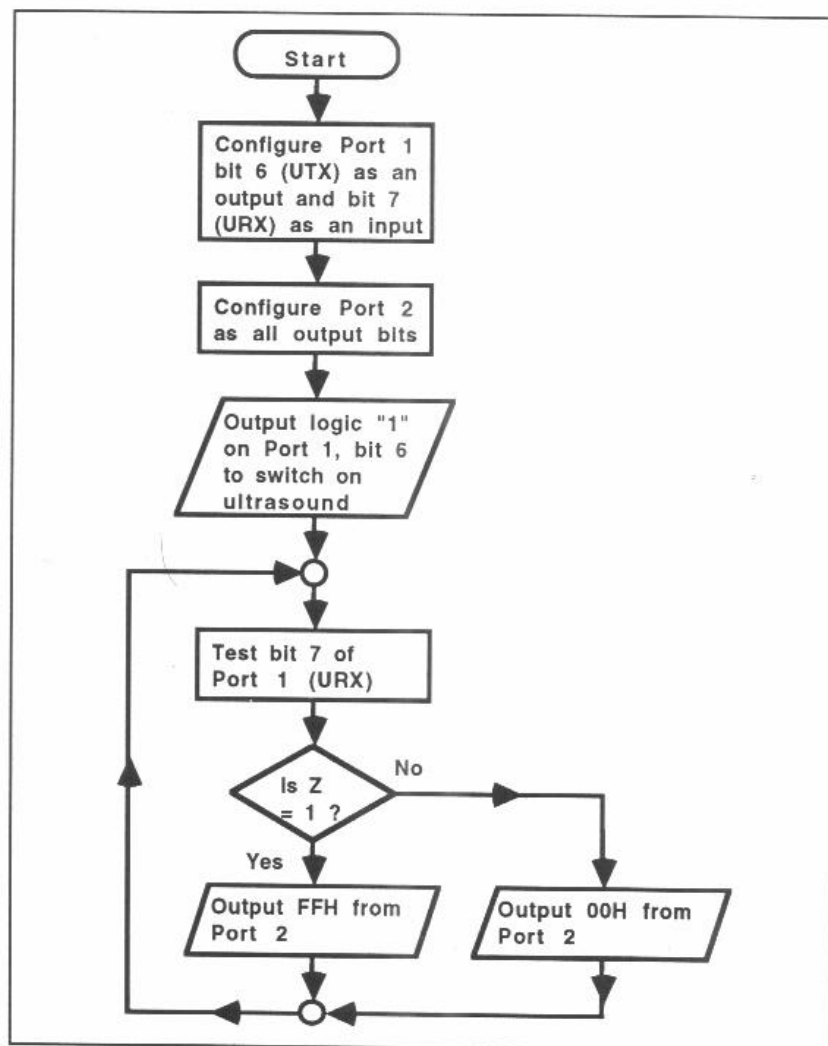
The Transmitter and Receiver can be used together to detect reflections from an object placed directly above the module.

Exercise 22

Write a program which will use the Ultrasonic Units within the Applications Module to act as a proximity detector. When an object is placed directly above the Ultrasonic Unit, all of the Port 2 monitor LEDs should be lit.

Solution:

Flowchart:



Source Program:

```

                                ;Defines the start address for
                                ;object code as 0080:0200H
ORG 0200H

UMODEREG EQU 086H                ;Defines address of Mode
                                ;Register
UPORTCTL1 EQU 088H                ;Defines address of Port
                                ;Control Register
UPORT1 EQU 090H                  ;Defines address of Port 1
UPORT2 EQU 092H                  ;Defines address of Port 2

MOV AL,20H                       ;Configures Port 1, bit 6 (P16)
OUT UPORTCTL,AL                   ;as an output and bit 7 (P17)
                                ;as an input

MOV AL,03H                       ;Configures Port 2, as all
OUT UMODEREG,AL                   ;outputs
MOV AL,40H                       ;Outputs a logic "1" on P16
OUT UPORT1,AL                     ;(UTX) to switch on Ultrasound

DETECT: IN AL,UPORT1              ;Tests P17 (URX) for Ultrasound
        TEST AL,80H              ;Received
        JZ ALARM

MOV AL,00H                       ;No Ultrasound detected so
OUT UPORT2,AL                     ;output 00H at Port 2

ALARM: MOV AL,0FFH                ;Ultrasound detected so
        OUT UPORT2,AL           ;output FFH at Port 2

        JMP DETECT               ;Jump back to check for
                                ;Ultrasound again

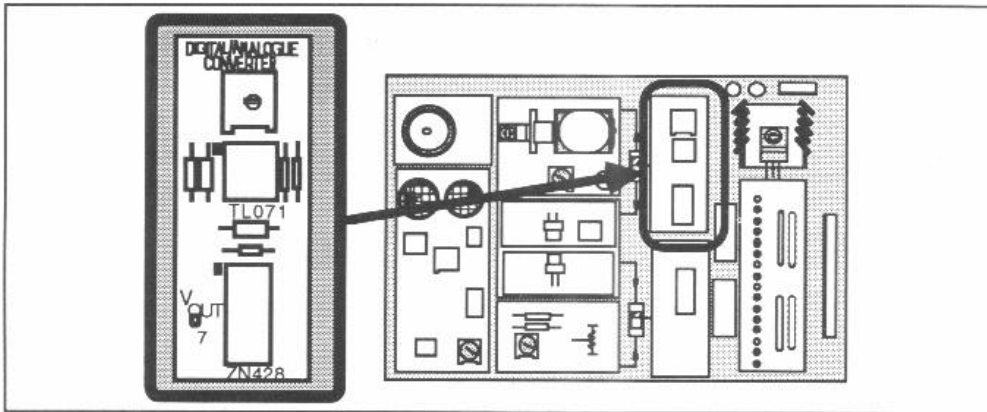
```

Use Merlin to produce an object program. Try running this program on PAT. You will need to adjust the sensitivity to avoid false triggering. This program could form the basis of an intruder alarm or automatic counter.

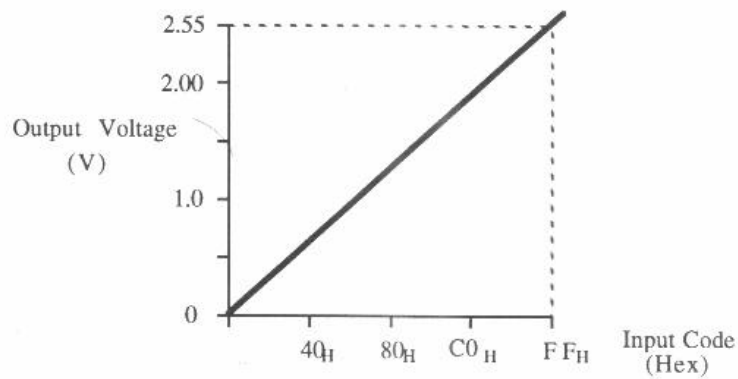
Practical Assignment

- 17 Write a program which uses the Ultrasonic Units within the Applications Module to act as a proximity detector. When an object is placed directly above the Ultrasonic Unit, the Piezo Sounder should be activated.

Digital to Analog Control

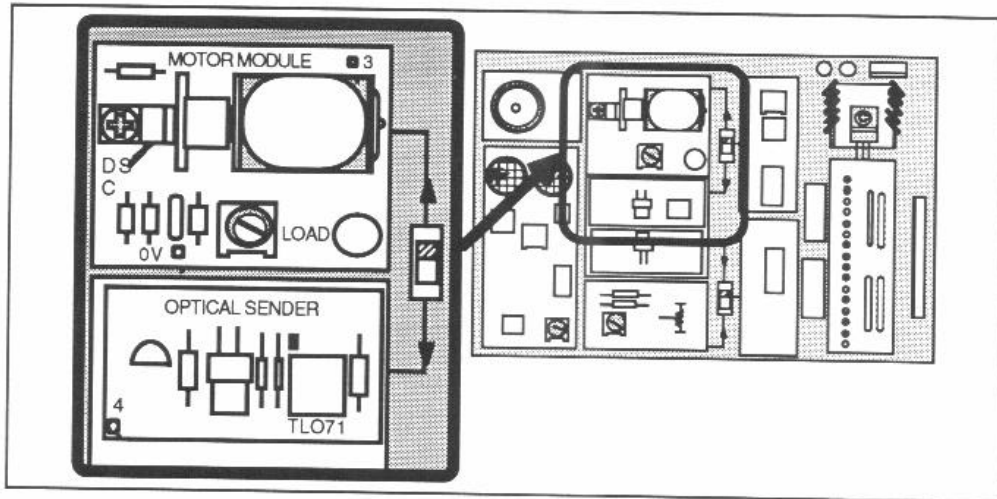


A Digital to Analog Converter (DAC) is necessary if a microprocessor-based control system is to produce an **analog** output. A DAC takes a digital value and **represents** it as a voltage level. The Applications Module DAC has an 8-bit input. The output can range from 0V to 2.55V thus:



Notice that there are $FF_H = 255_{10}$ steps between 0V and 2.55V, so each increase in 1_H gives a voltage rise 0.01V (10mV).

The upper slider switch (DAC switch) on the Applications Module allows the output of the DAC to be applied to either the Optical Sender or the DC Motor:



The following sequence is required to initiate digital to analog conversion:

- 1 Output "0" on Port 1, bit 0 (P10) to enable the DAC
- 2 Output digital data from Port 2

The voltage at the output will then be directly proportional to the input binary code.

Notes:

.....

.....

.....

.....

.....

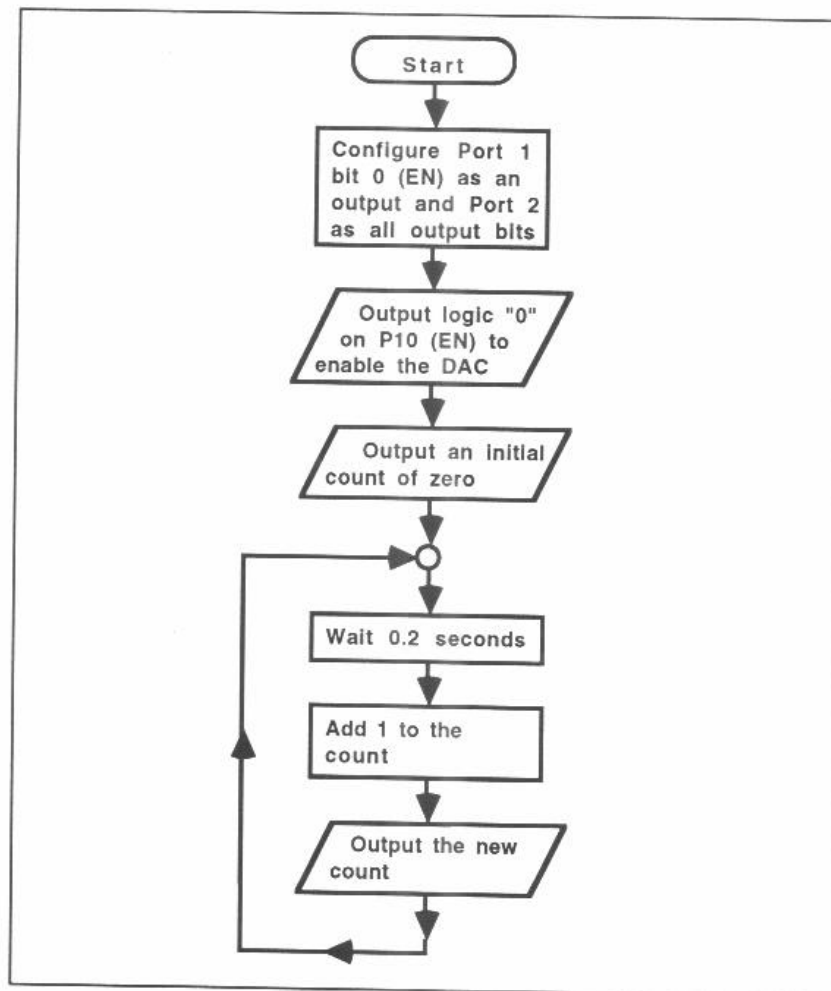
.....

Exercise 23

Write a program which will produce a slowly increasing binary count output (changing about every 0.2 seconds) at Port 2 which is passed, via the DAC to either the Optical Sender or the DC Motor.

Solution:

Flowchart:



Source Program:

```

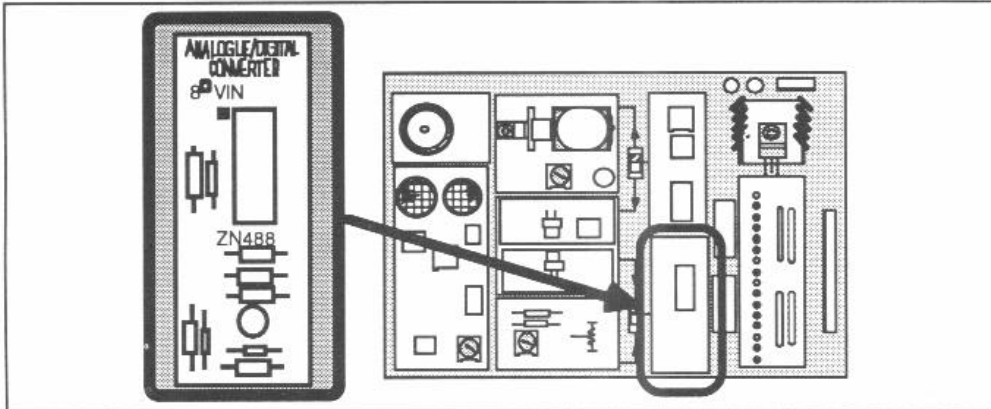
                                ;Defines the start address for
                                ;object code as 0080:0400H
                                ORG 0400H
UMODEREG EQU 086H              ;Defines address of Mode
                                ;Register
UPORTCTL1 EQU 088H            ;Defines address of Port
                                ;Control Register
UPORT1 EQU 090H               ;Defines address of Port 1
UPORT2 EQU 092H               ;Defines address of Port 2
MOV AL,01H                    ;Configures Port 1, bit 0 (P10)
OUT UPORT1CTL,AL              ;as an output
MOV AL,03H                    ;Configures Port 2, as all
OUT UMODEREG,AL               ;outputs
MOV AL,00H                    ;Outputs a logic "0" on P10
OUT UPORT1,AL                 ;(EN) to enable DAC
MOV AL,00H                    ;Outputs initial count of zero
OUTPUT: OUT UPORT2,AL
MOV CX,0FFFFH                 ;Delay of approximately 0.2 sec
DELY: LOOP DELY
INC AL                         ;Adds 1 to the count
JMP OUTPUT                    ;Jump back to output next count

```

Use Merlin to produce an object program. Try running this program on PAT. You will find that a count of about 40H is needed before motor inertia is overcome and it begins to rotate.

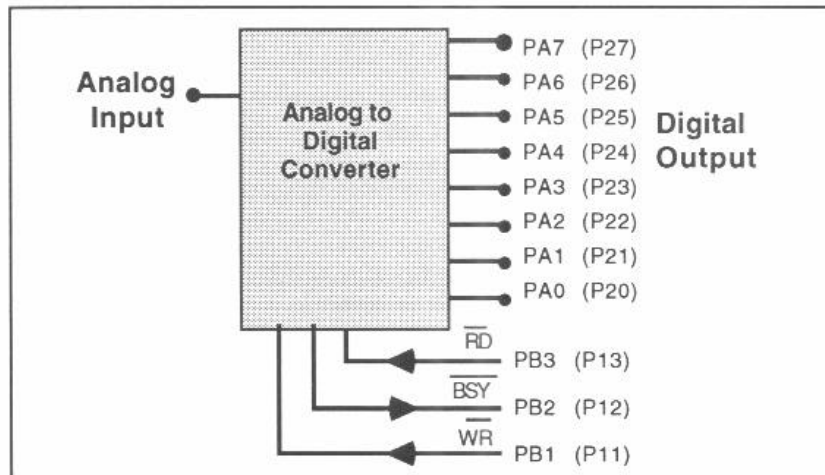
This technique can be used, for example, to slowly run a DC Motor up to its operating speed.

Analog to Digital Converter



An Analog to Digital Converter (ADC) is required where external analog inputs are to be applied to a digital system, for example a microcomputer. The ADC takes an input voltage and represents it as a binary value.

The Applications Module ADC has an 8-bit output connected to Port 2 and three control signals, connected to Port 1 thus:

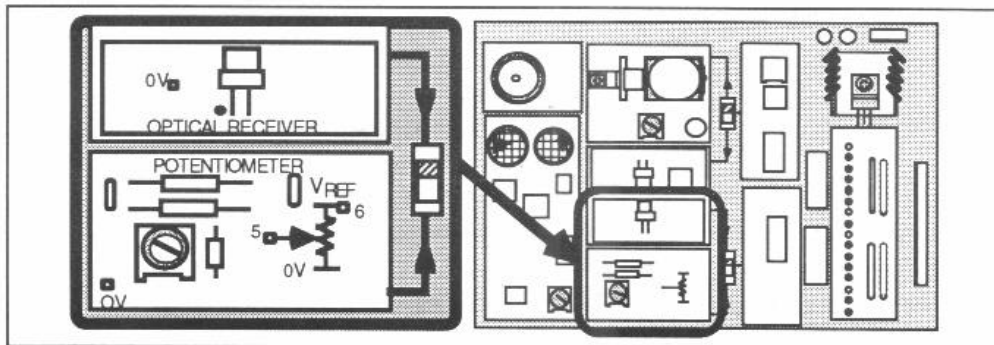


The input range is from 0V to 2.55V, like the DAC, so each step represents 10mV. The following sequence is required to perform conversion:

- 1 Output "1" on Port 1, bits 1 and 3 (P11 and P13) initially.
- 2 Output a short negative-going pulse on Port 1, bit 1 (P11) to initiate conversion. This can be done by causing an output bit to change from 1 to 0 and back to 1 again.
- 3 Monitor Port 1 bit 2 (P12) and wait for P12=1 indicating that conversion is complete
- 4 Read the digital data on Port 2
- 5 Output a "1" on Port 1 bit 3 (P13) to disable ADC outputs

The value at Port 2 will now be directly proportional to the input voltage.

The ADC can be connected to either the Potentiometer or the Optical Receiver by means of the lower slider switch (ADC switch):



Notes:

.....

.....

.....

.....

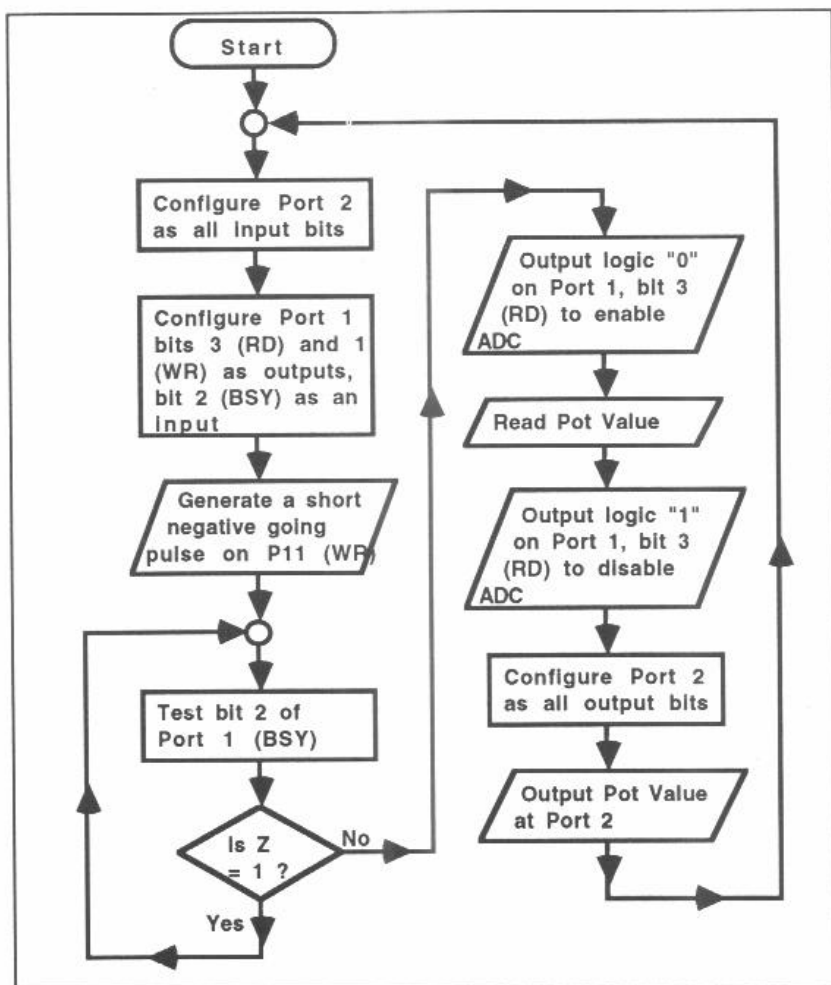
.....

Exercise 24

Write a program which will output a binary value at Port 2, dependent upon the position of the Applications Module potentiometer.

Solution

Flowchart:



Source Program:

```
ORG 0500H ;Defines the start address for
;object code as 0080:0500H

;Define MUART Addresses:
UMODEREG EQU 086H ;Defines address of Mode
;Register
UPORTCTL1 EQU 088H ;Defines address of Port
;Control Register
UPORT1 EQU 090H ;Defines address of Port 1
UPORT2 EQU 092H ;Defines address of Port 2

;Configure Ports 1 and 2 for ADC:
AGAIN: MOV AL,00H ;Configures Port 2, as all
OUT UMODEREG,AL ;inputs
MOV AL,0AH ;Configures Port 1, bits 3 and
OUT UPORT1CTL,AL ;1 as outputs, bit 2 as input

;Generate a short negative going pulse on P11 (WR):
MOV AL,02H ;Outputs a logic "1" on P11
OUT UPORT1,AL
MOV AL,00H ;Outputs a logic "0" on P11
OUT UPORT1,AL
MOV AL,02H ;Outputs a logic "1" on P11
OUT UPORT1,AL

;Wait for the ADC to complete conversion:
CPLTE: IN AL,UPORT1 ;Tests for P12 (BSY) set -
TEST AL,04H ;conversion complete
JZ CPLTE ;Wait until P12 (BSY) is set

;Enable the ADC and read the Pot Value:
MOV AL,0F7H ;Output a logic "0" on P13 (RD)
OUT UPORT1,AL ;to enable ADC output
IN AL,UPORT2 ;Read Pot Value and copy into
MOV BL,AL ;BL Register
```

continued

Source Program continued...

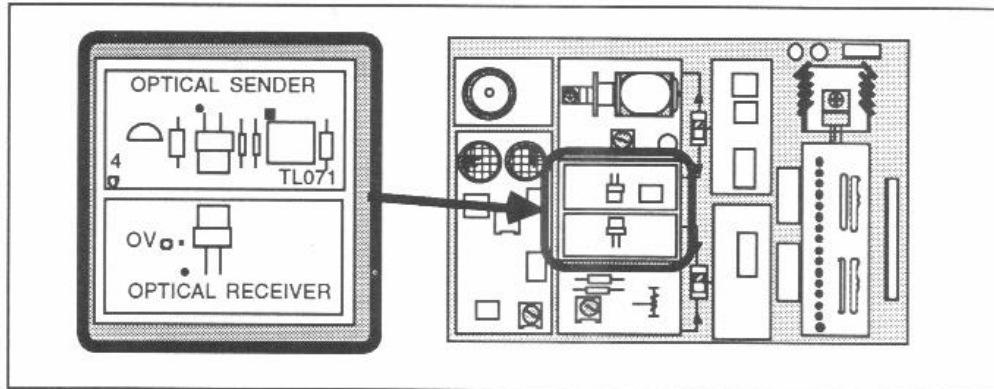
```
;Disable the ADC output and reconfigure Port 2 as all outputs so that ;the
Pot Value can be displayed:
      MOV  AL,08H                ;Output a logic "1" on P13 (RD)
      OUT  UPORT1,AL            ;to disable ADC output
      MOV  AL,03H                ;Configures Port 2, as all
      OUT  UMODEREG,AL          ;outputs
      MOV  AL,BL                 ;Get Pot Value from the BL
      OUT  UPORT2,AL            ;Register and output at Port 2

;Generate a short delay so that the output is displayed much more often
;than the input:
      MOV  CX,0700H
DELY:  LOOP DELY

;Jump back to read Pot Value again:
      JMP  AGAIN                ;Repeat forever
```

Use this source program to generate an object program. Transfer to PAT and execute. Observe the effect of changing the pot position.

This program can also act as an ambient light level indicator if the lower slider switch is moved to the upper position. The LEDs now give an indication of the intensity of light falling on the Optical Receiver.

Optical Sender and Optical Receiver

The Optical Sender and Optical Receiver units can be used in isolation, as LED and Light Detector respectively, or used together to form an Optical Link.

The output of the DAC can be switched to the Optical Sender by setting the upper slider switch to the lower position. The brightness of this LED then varies according to the code at the input to the DAC.

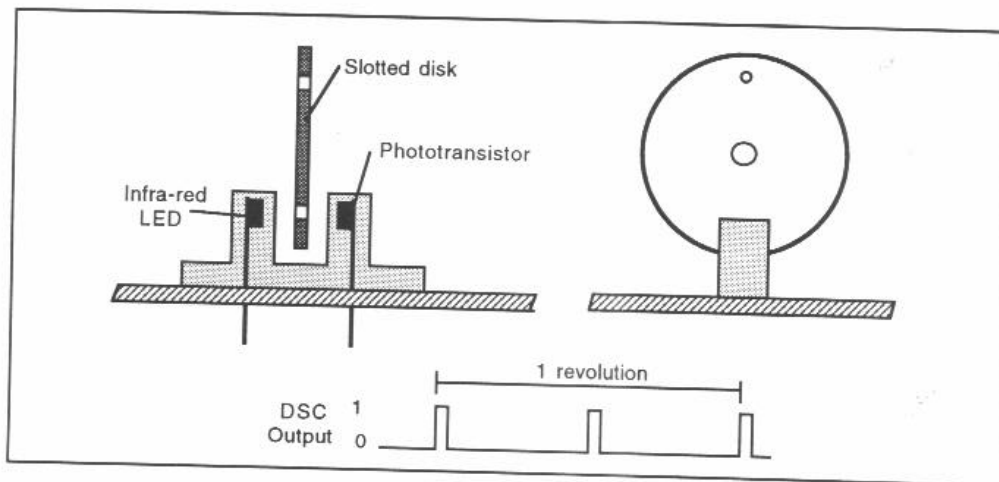
The Optical Receiver output can be switched to the input of the ADC by setting the lower slider switch to the upper position. The intensity of light falling on the Receiver can thus be converted into a binary value. The light intensity will depend upon the ambient lighting conditions and upon any light output from the Optical Sender unit.

Practical Assignment

- 18 Write a program which will sound the Piezo Sounder whenever the optical link between Optical Sender and Receiver is broken.

Optical Disc Encoder

The motor disc passes between an optical transmitter and receiver. There are two holes in the disc, each one producing a short pulse as the shaft rotates. Clearly, the number of pulses per second is a measure of the speed of rotation of the motor shaft.



Practical Assignment

- 19 Write a program which will allow the speed of the DC Motor to be varied according to the position of the Potentiometer or the level of light falling upon the Optical Receiver.

Student Assessment 11

1. For the Piezo Sounder to produce an audio frequency, a TTL signal must be applied to:
 - a Port 1, bit 5
 - b Port 1, bit 6
 - c Port 2, bit 5
 - d Port 2, bit 6
2. When the Ultrasonic Receiver detects a 40kHz ultrasound signal:
 - a P16 is set to logic 1
 - b P16 is set to logic 0
 - c P17 is set to logic 1
 - d P17 has a 40kHz squarewave
3. An increase of 01_{11} at the input of the Applications Module DAC produces a rise in output voltage of:
 - a 1mV
 - b 10mV
 - c 25.5mV
 - d 255mV
4. The signal from the Applications Module ADC which indicates that conversion is complete is:
 - a RD
 - b WR
 - c BSY
 - d EN
5. The Applications Module Optical Disc Encoder produces:
 - a 0.5 pulses per revolution
 - b 1 pulse per revolution
 - c 2 pulses per revolution
 - d 10 pulses per revolution

Chapter 9.2 Stack and Subroutines

Objectives of this Chapter

Having studied this chapter you will be able to:

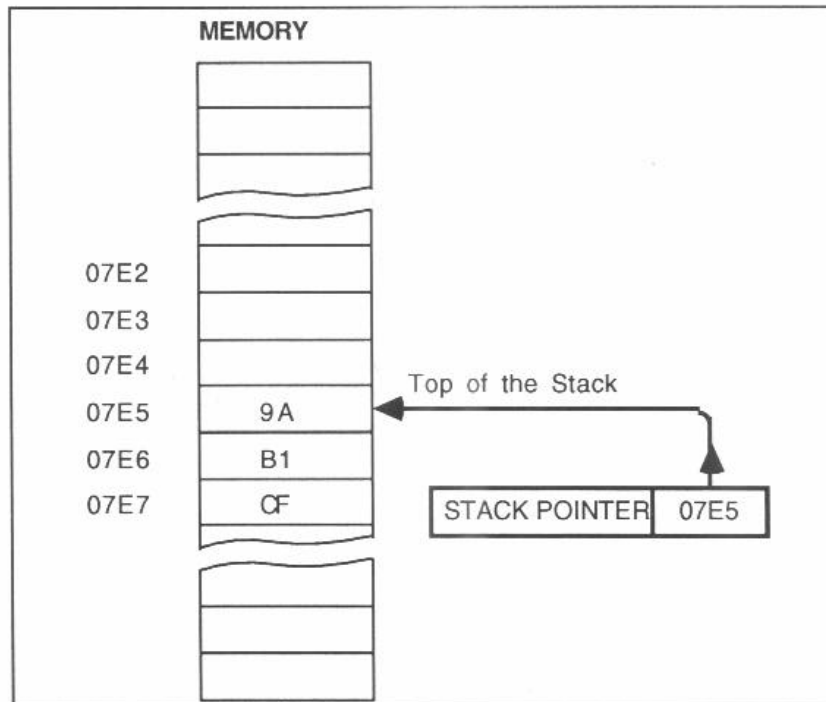
- Understand the need for a Stack
- Explain the LIFO stack mechanism
- Describe the function of the Stack Pointer
- Describe the 80286 stack save and restore instructions
- Explain the functions of a subroutine
- Describe the following 80286 instructions:
CALL
RET
PUSHA
POPA
- Explain the purpose of the INCLUDE assembler directive.
- Make use of PAT Monitor Subroutines within your own programs which will:
write characters to the display
read characters from keyboard
produce delays

The Stack

Sometimes it is necessary to save data in a temporary store. This could be a partial result in a calculation, or because a register is required for another purpose. Clearly a register or a memory location could be used for such data storage. However, the precise location of a temporary result is often relatively unimportant, provided the data can be retrieved reliably. The **stack** is a special area of memory set aside for the storage of temporary data. It provides rapid storage and retrieval of data. The stack operates rather like a pile of documents in a tray: As sheets are placed in the tray, only the last document will be immediately accessible. The **last** document placed in the tray will be the **first** one removed. In microcomputer stack terms this is called a "**Last In First Out**" or **LIFO** structure. In a LIFO stack the exact location of data is much less important than the **order** in which datawords have been saved.

Stack Pointer

A **Stack Pointer** register is used to "point" to the last stack location used:



The 80286 has both Stack Pointer and Stack Segment Registers. The Stack Segment Register defines the memory **segment** for the Stack (in the same way that the Code Segment Register defines the segment for machine code).

The PAT monitor program always sets the Stack Segment to 0F80H and the Stack Pointer to 0800H. The Stack Pointer will change as data is saved on the stack but the Stack Segment will not, unless explicitly changed by an instruction (eg "MOV SS,DS:1400H").

Stack Save and Restore Instructions

Data is **saved** on a stack by using a PUSH instruction.

Examples:

```
PUSH AX                ;Pushes the Accumulator onto
                       ;the stack, high byte first
PUSH 7890H             ;Pushes the value 7890H onto
                       ;the stack, high byte first
```

Suppose the Stack Pointer contains 0F80:0800H and the following instruction is executed:

```
PUSH 3456H             ;Pushes the value 3456H onto
                       ;the stack
```

This instruction will:

- 1 Decrement the Stack Pointer to point to location 0F80:07FFH.
- 2 Save the **high byte** of the source (34H) in this stack location.
- 3 Decrement the Stack Pointer again to point to location 0F80:07FEH.
- 4 Save the **low byte** of the source (56H) in this stack location.

So, following this program section:

```
location 0F80:07FFH contains 34H
location 0F80:07FEH contains 56H
Stack Pointer = 0F80:07FEH
```

Data is **restored** from the stack by using a POP instruction.

Examples:

```
POP AX                ;Restores the Accumulator from  
                    ;the stack, low byte first  
POP DS:6700H         ;Restores the contents of  
                    ;location 6700H within DS, low  
                    ;byte first
```

If the Stack Pointer contained 0F80:07EC_H and the following instruction is executed:

```
POP AX                ;Restores the Accumulator from  
                    ;the stack
```

This instruction will:

- 1 Increment the Stack Pointer to point to location 0F80:07ED_H.
- 2 Restore the **low byte** of the Accumulator
- 3 Increment the Stack Pointer again to point to location 0F80:07EE_H.
- 4 Save the **high byte** of the Accumulator

Notes:

.....

.....

.....

.....

.....

.....

.....

.....

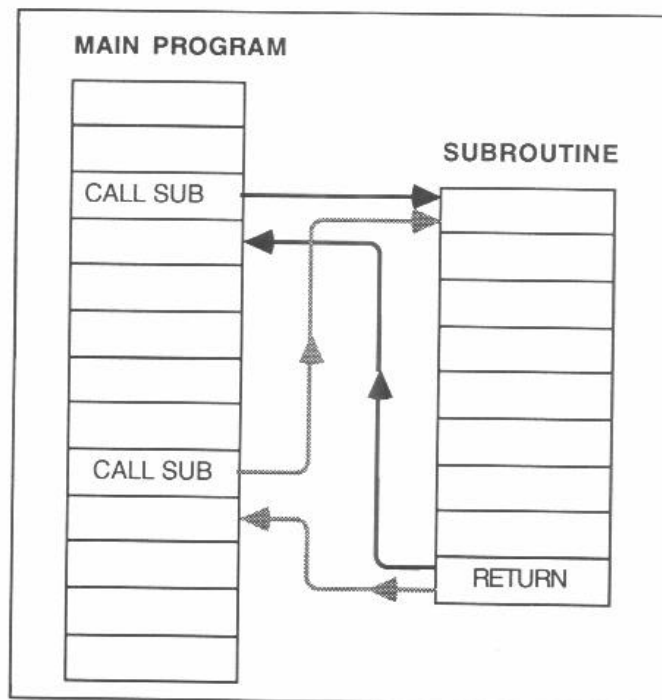
.....

.....

Subroutines

In many programs there will be sequences of instructions which are used several times within the program. For example the short time delay used in a number of the Applications Module programs in previous chapters. Such a repeated section of instructions is called a "routine". Now, rather than include the routine **every** time it is required, the microprocessor allows such sequences of object code to appear only **once** and then to be **called** upon several times within the program. A routine which can be used in this way is called a "subroutine".

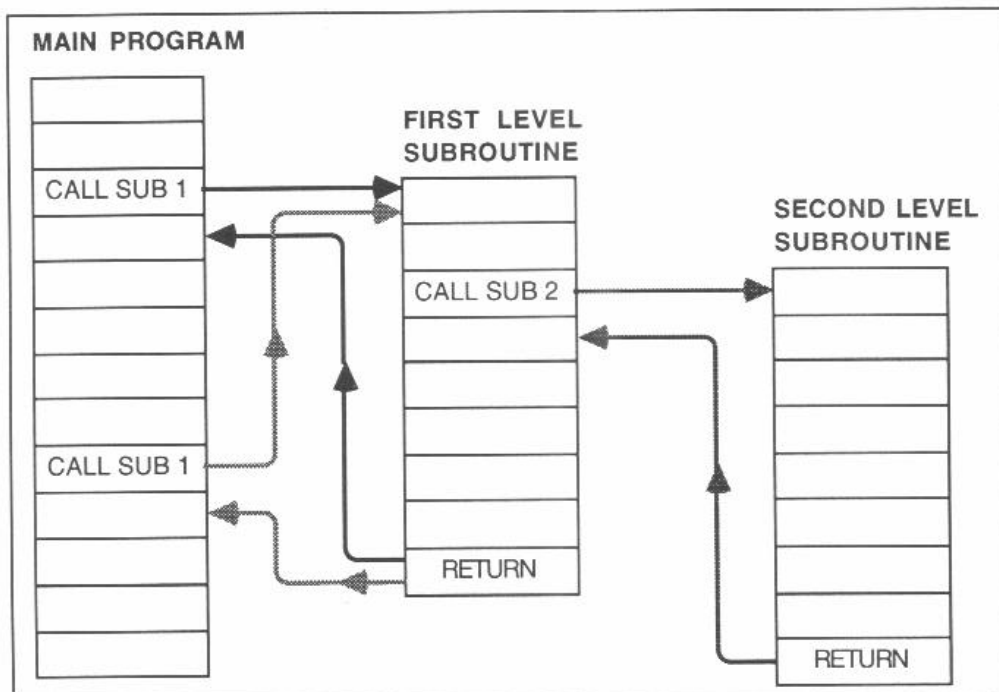
Subroutines are also often used by more than one program. Libraries of useful routines may be compiled to reduce program development time. Programs which use subroutines are much easier to develop and understand.



A **CALL** instruction transfers program execution to a subroutine.

A **RETURN (RET)** instruction restores the program counter from the point at which it left the main program.

When a subroutine is called, the return address is **automatically** saved on the **Stack**. At the end of a subroutine the return address is again **automatically** restored from the stack. This type of structure allows **multiple** levels of subroutines to be supported (sometimes called **nested** subroutines), where one subroutine calls another:



The first return address is saved on the stack and then the second. Since the stack has a LIFO action, each address will be restored as it is required (ie second return address **then** first).

Subroutines may use registers which the main program uses so it is good practice for a subroutine to save any registers which it uses.

Special Push and Pop Instructions

Although **individual** registers can be saved and subsequently restored from the stack by using a PUSH instruction, the **PUSHA** instruction will Push **all** of the General Purpose Registers (in the order: AX, CX, DX, BX, SP, BP, SI, DI).

Similarly, **all** of the General Purpose Registers can be **restored** from the stack by the **POPA** instruction. This will restore these registers in the order DI, SI, BP, SP, BX, DX, CX, AX. Notice that this is the **opposite** order to PUSHA. This is due to the LIFO nature of the Stack.

Very often it will be necessary to preserve the Flags before executing a subroutine. The **PUSHF** and **POPF** instructions can be used to save Flags and restore Flags respectively.

Notes:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Exercise 25

Modify the program for Exercise 21 to take advantage of subroutine structures.

Solution:

The source program used for Exercise 21 is shown below:

```
ORG 0100H                ;Defines the start address for
                          ;object code as 0080:0100H
UPORTCTL1 EQU 088H       ;Defines address of Port
                          ;Control Register
UPORT1 EQU 090H         ;Defines address of Port 1
VAL1 EQU 0FAH           ;Defines "VAL1" AS FAH
MOV AL,20H              ;Configures Port 1, bit 5 (P15)
OUT UPORTCTL,AL        ;as an output
OUTPUT: MOV AL,00H      ;Outputs a logic "0" on Port 1,
                          ;bit 5 (P15)
OUT UPORT1,AL
DELY1: MOV CX,0FAH      ;Wait for 0.5ms
LOOP DELY1
MOV AL,20H              ;Outputs a logic "1" on Port 1,
                          ;bit 5 (P15)
OUT UPORT1,AL
DELY2: MOV CX,0FAH      ;Wait for 0.5ms
LOOP DELY2
JMP OUTPUT              ;Jump back to output again
```

Notice that the **same** time delay has been used **twice**.

These can be changed to subroutine CALLs as shown on the page opposite:

```

;Main Program:
                ORG 0100H                ;Defines the start address for
                                                ;object code as 0080:0100H
UPORTCTL1 EQU 088H                ;Defines address of Port
                                                ;Control Register
UPORT1 EQU 090H                ;Defines address of Port 1
VAL1 EQU 0FAH                ;Defines "VAL1" AS FAH
                MOV AL,20H            ;Configures Port 1, bit 5 (P15)
                OUT UPORT1CTL,AL        ;as an output
OUTPUT:        MOV AL,00H            ;Outputs a logic "0" on Port 1,
                                                ;bit 5 (P15)
                OUT UPORT1,AL
                CALL DELAY            ;Calls delay of 0.5ms
                MOV AL,20H            ;Outputs a logic "1" on Port 1,
                                                ;bit 5 (P15)
                OUT UPORT1,AL
                CALL DELAY            ;Calls delay of 0.5ms
                JMP OUTPUT            ;Jump back to output again

;Subroutine:
DELAY:        MOV CX,0FAH            ;Wait for 0.5ms
DELY:        LOOP DELY
                RET                    ;Return to Main Program

```

Use this source program to generate an object program. Transfer to PAT and verify correct operation.

Practical Assignment

- 20 Write a subroutine which will add the AX, BX, CX and DX registers, saving the result in location 0080:1000H. The previous contents of all registers and the Status Flags should be restored before returning.

PAT Monitor Subroutines

The PAT Monitor program includes a number of subroutines which are available for you to use in your own programs. These subroutines are not however called using a CALL instruction. Instead they are accessed using an "Interrupt" (INT) instruction.

We shall be examining 80286 interrupts in detail in the next chapter. For now we shall just consider one particular interrupt. This is one which you have been using almost since the beginning of this Curriculum Manual: Interrupt 28H.

Many of the programs you have seen so far ended with a return to the PAT system thus:

```
MOV  BX,00H                ;Returns to PAT System
MOV  AH,04H
INT  028H
```

The "INT 28H" instruction is used to access PAT Monitor Subroutines. In the case above, the "EXIT" subroutine is called. The actual subroutine called is defined by the contents of the AH Register. So "EXIT" is Function Number 04H.

You will also notice that the BX Register is always set to 0000H. This is because the "EXIT" Function uses the BX Register for Error Codes. If the BX Register contains a non-zero value, the corresponding error message will be displayed.

There are 20 or so other PAT System Calls available. A full list and description of the function of these can be found in Appendix 2. Notice that several Functions are associated with control of the PC screen:

WRCHAR (Function Number 0CH)

This interprets the contents of the AL Register as an ASCII code and sends the corresponding character to the screen.

CLSCR (Function Number 12H)

This clears the screen.

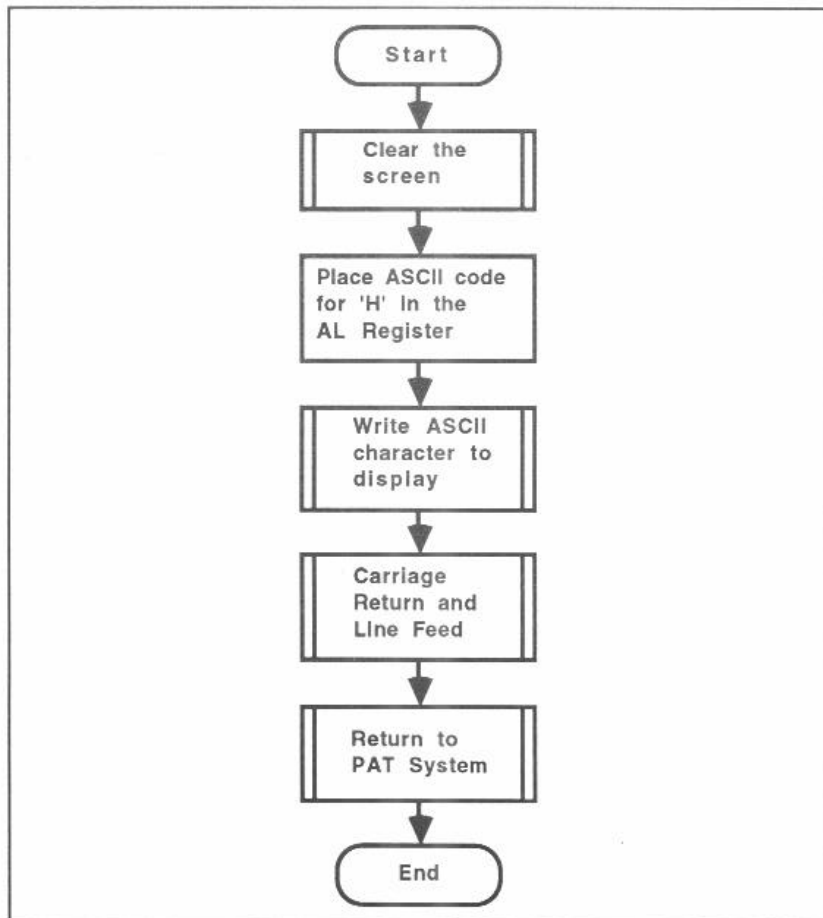
CRLF (Function Number 11H)

This outputs a "Carriage Return" and "Line Feed" to the screen.

Exercise 26

Write a program which will display the character "H " on the screen.

Solution:



Note: The ASCII code for "H" is 48_H. Other ASCII codes can be found in Appendix 3.

Source Program:

```
EXIT      EQU  004H          ;Defines Function Number for
                          ;"Exit to PAT System" Function
WRCHAR    EQU  00CH          ;Defines Function Number for
                          ;"Write One Character" Function
CRLF      EQU  011H          ;Defines Function Number for
                          ;"Carriage Return and Line
                          ;Feed" Function
CLRSCR    EQU  012H          ;Defines Function Number for
                          ;"Clear Screen" Function
          ORG  0300H          ;Defines the start address for
                          ;object code as 0080:0300H
          MOV  AH,CLRSCR      ;Clears the Screen
          INT  028H
          MOV  AL,048H        ;Places ASCII code for 'H' in
                          ;the AL Register
          MOV  AH,WRCHAR      ;Writes an ASCII character to
          INT  028H          ;the display
          MOV  AH,CRLF        ;Carriage Return and Line Feed
          INT  028H
          MOV  BX,0000H       ;Returns to PAT System
          MOV  AH,EXIT
          INT  028H
```

Use Merlin to produce source and object programs. Load this program into PAT memory and execute. You will see the letter "H" and the "PAT:" prompt on the next line.

The INCLUDE Directive

INCLUDE is an assembler directive which allows source code files on disk to be **included** within an assembly language program. Now, your 80286 Development Disk will contain a file called "PATCALLS.INC". This holds the label definitions for all System Calls. By means of the INCLUDE directive you can use all of these definitions within a program **without** having to define each one separately using an EQU directive. So, you can simply use the Function names as required.

The solution to Exercise 26 could therefore be modified thus:

```
INCLUDE    PATCALLS.INC      ;Use label definitions
                                ;from the file
                                ;PATCALLS.INC

ORG    0300H                  ;Defines the start address for
                                ;object code as 0080:0300H

MOV    AH,CLRSCR              ;Clears the Screen
INT    028H

MOV    AL,048H                ;Places ASCII code for 'H' in
                                ;the AL Register
MOV    AH,WRCHAR              ;Writes an ASCII character to
INT    028H                  ;the display
MOV    AH,CRLF                ;Carriage Return and Line Feed
INT    028H

MOV    BX,0000H               ;Returns to PAT System
MOV    AH,EXIT
INT    028H
```

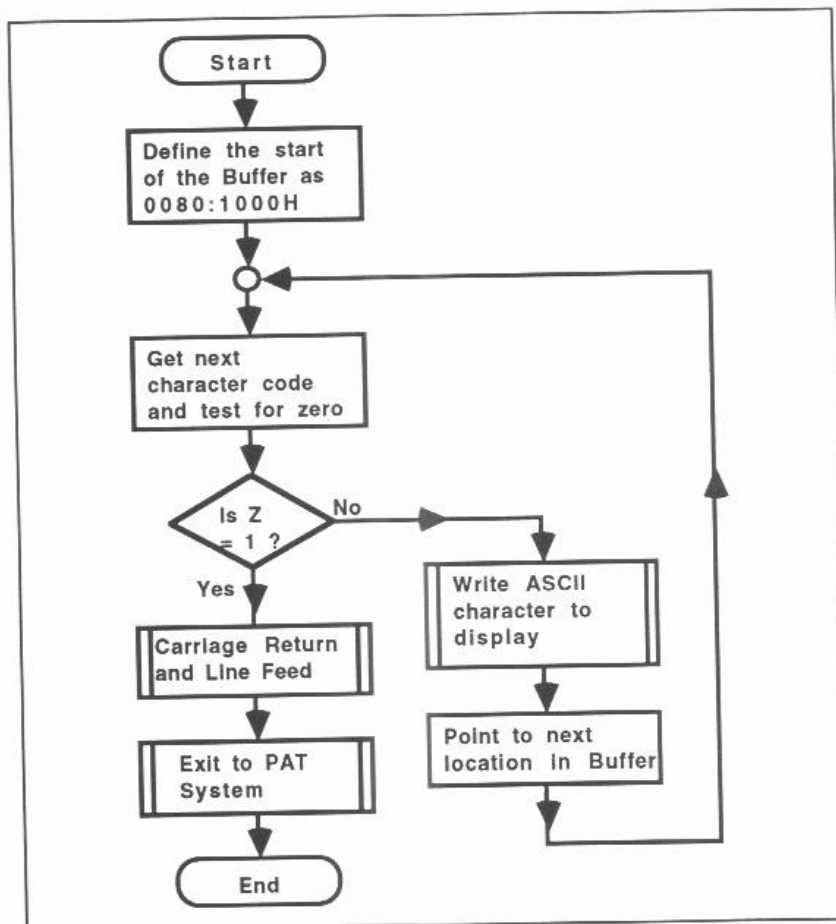
Modify your source program as above and assemble. You should find that this will assemble without error, although the labels CLRSCR, WRCHAR, CRLF and EXIT have not been explicitly defined. These names are defined by the "PATCALLS.INC" file.

Exercise 27

Write a program which will display the message "Hello " on the screen.

Solution:

Clearly, this is an extension of the previous exercise. One possible solution would be to effectively repeat the previous program a number of times but this would be rather inelegant. A more suitable approach is to set up a **buffer** in PAT memory which contains the necessary codes and use a looped program to display each code in turn thus:



Source Program:

```
INCLUDE PATCALLS.INC ;Use label definitions from the
;file PATCALLS.INC
BUFFER EQU 1000H ;Defines start of Data Buffer
;as 0080:1000H
ORG 0400H ;Defines the start address for
;object code as 0080:0400H
MOV SI,BUFFER ;Uses SI Register to point to
;the first Buffer location
MOV AH,CLRSCR ;Clears the Screen
INT 028H
NXTCHR: MOV AL,[SI] ;Moves the next ASCII code into
;the AL Register
TEST AL,0FFH ;Checks for end of buffer
JZ DONE ;character code (00H)
MOV AH,WRCHAR ;Character code is non zero so
INT 028H ;write the ASCII character to
;the display
INC SI ;Point to next character
JMP NXTCHR ;Repeat character write
DONE: MOV AH,CRLF ;Carriage Return and Line Feed
INT 028H
MOV BX,0000H ;Returns to PAT System
MOV AH,EXIT
INT 028H
```

Note: The ASCII codes for "Hello " must be entered **before** this program is run. One way to do this is to use the "C" command from the Terminal Program thus:

```
PAT: C 3000: FF "Hello" 00:
```

Alternatively, the ASCII data can be made part of the Source Program by using the DB (Define a Byte) assembler directive thus:

```

INCLUDE  PATCALLS.INC           ;Use label definitions from the
                                ;file PATCALLS.INC
                                ;ASCII Data Buffer
ORG  1000H
DB  "Hello",00H
BUFFER EQU 1000H                ;Defines start of Data Buffer
                                ;as 0080:1000H
ORG  0400H                      ;Defines the start address for
                                ;object code as 0080:0400H
MOV  SI,BUFFER                  ;Uses SI Register to point to
                                ;the first Buffer location
MOV  AH,CLRSCR                  ;Clears the Screen
INT  028H
NXTCHR: MOV AL,[SI]              ;Moves the next ASCII code into
                                ;the AL Register
TEST AL,0FFH                   ;Checks for end of buffer
JZ  DONE                        ;character code (00H)
MOV  AH,WRCHAR                  ;Character code is non zero so
INT  028H                       ;write the ASCII character to
                                ;the display
INC  SI                          ;Point to next character
JMP  NXTCHR                     ;Repeat character write
DONE: MOV AH,CRLF                ;Carriage Return and Line Feed
INT  028H
MOV  BX,0000H                   ;Returns to PAT System
MOV  AH,EXIT
INT  028H

```

Use Merlin to produce source and object programs. Load this program into PAT memory and execute. You will see the message "Hello " and the "PAT:" prompt on the next line.

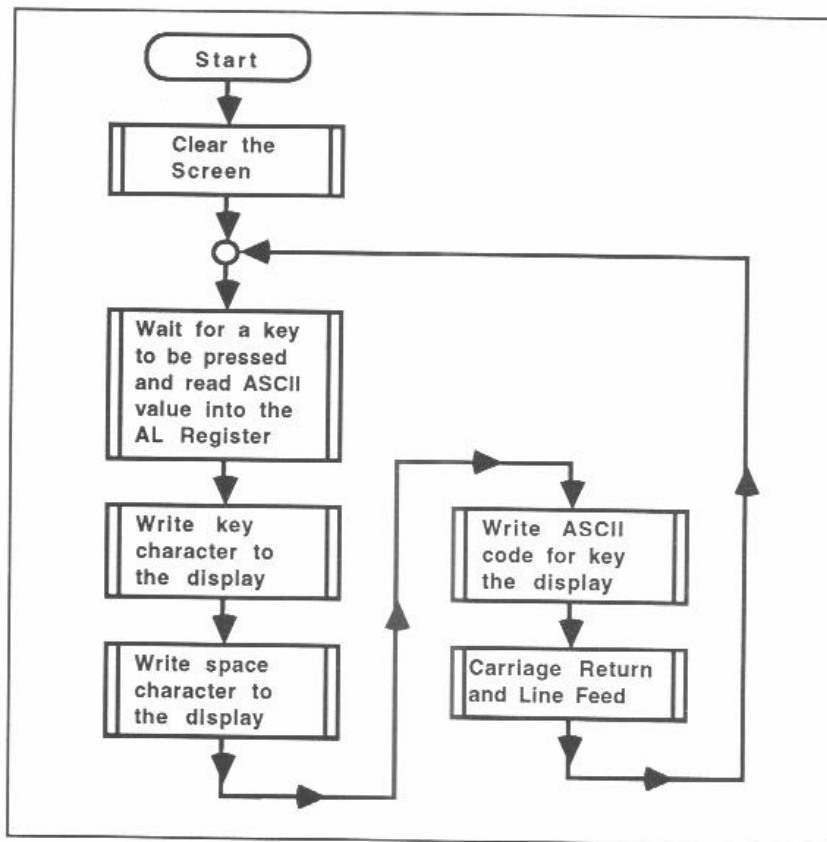
You can experiment with other words by changing the DB directive line.

Exercise 28

Write a program which will read a hexadecimal byte from the console and then write the corresponding ASCII character to the screen.

Solution:

Flowchart:



Source Program:

```
INCLUDE PATCALLS.INC          ;Use label definitions from the
                                ;file PATCALLS.INC
ORG 0500H                      ;Defines the start address for
                                ;object code as 0080:0500H
MOV AH,CLRSR                   ;Clears the Screen
INT 028H
NXTCHR: MOV AH,RDCHAR           ;Wait for a key to be pressed
INT 028H                       ;then return ASCII code to the
                                ;AL Register
MOV AH,WRCHAR                  ;Write the ASCII character to
INT 028H                       ;the display
PUSH AX                        ;Save ASCII character code
MOV AL,20H                     ;Write a space to the display
MOV AH,WRCHAR
INT 028H
POP AX                          ;Restore the ASCII character
                                ;code
MOV AH,WRBYTE                  ;Write ASCII code for character
INT 028H                       ;to the display
MOV AH,CRLF                    ;Carriage Return and Line Feed
INT 028H
JMP NXTCHR                     ;Repeat from label "NXTCHR"
```

Use this source program to generate an object program. Transfer this to the PAT and execute.

You could use this program instead of a **table** of ASCII codes.

Practical Assignments

- 21 Write a program which will produce a two-digit **hexadecimal** count, increasing at the rate of once per second.
- 22 Write a program which will add the bytes at memory locations 0080:2000_H and 0080:2100_H. The screen should display:
"Contents of 0080:2000_H + Contents of 0080:2100_H = Result"
- 23 Write a program which will sound the piezo sounder whenever the "S" key is held down.
- 24 Write a program, using PAT monitor subroutines, which will allow the speed of the DC Motor to be controlled by the "+" and "-" keys. The motor should slowly accelerate when the "+" key is pressed, hold speed constant if no keys are pressed and decelerate when the "-" key is pressed.

Notes:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Student Assessment 12

1. The last stack location used is defined by the contents of the:
 - a Data Segment Register
 - b Instruction Pointer Register
 - c Stack Pointer Register
 - d Data Index Register
2. The Stack Pointer contains 0F80:045DH. After the instruction POP AX has been executed, the Stack Pointer will contain:
 - a 0F80:045CH
 - b 0F80:045DH
 - c 0F80:045EH
 - d 0F80:045FH
3. The 80286 instruction which usually occurs at the end of a subroutine is:
 - a RST
 - b RTS
 - c RET
 - d RETN
4. The PAT monitor subroutine which allows ASCII characters to be written to the screen is:
 - a WRITE
 - b WRCHAR
 - c TOAHEX
 - d WTNMS
5. The assembler directive which allows a source code file on disk to be added to an assembly language program is:
 - a ADD
 - b INCLUDE
 - c JOIN
 - d LINK