QMP 7.1 D/F

Channabasaveshwara Institute of Technolog
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(**NAAC Accredited & ISO 9001:2015 Certified Institution)**
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Computer Science & Engineering

MICROPROCESSOR AND MICROCONTROLLER LABORATORY

[As per Choice Based Credit System (CBCS) scheme]

(Academic year 2017 -2018)

15CSL48

B.E - IV Semester

Lab Manual

Name :____

USN :____

Batch : __________________ Section : _____________

Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(**NAAC Accredited & ISO 9001:2015 Certified Institution)**
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Partnering in Academic Excellence

Department of Computer Science & Engineering

MICROPROCESSOR AND MICROCONTROLLER LABORATORY (15CSL48)

Version 1.1

February 2018

Prepared by:

Mr.Chetan Balaji
Associate Professor

Reviewed and Approved by:

Dr.Shantala C P
Professor& Head,

Dept. of CSE

Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(**NAAC Accredited & ISO 9001:2015 Certified Institution)**
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

SYLLABUS

Laboratory Code: 15CSL48 | **IA Marks 20**
Number of Lecture Hours/Week 01I + 02P | **Exam Marks 80**
Total Number of Lecture Hours 40 | **Exam Hours 03**

CREDITS – 02

A. Laboratory Session-1: Write-up on Microprocessors, 8086 Functional block diagram, Pin diagram and description. The same information is also taught in theory class; this helps the students to understand better.

B. Laboratory Session-2: Write-up on Instruction group, Timing diagrams, etc. The same information is also taught in theory class; this helps the students to understand better. Note: These TWO Laboratory sessions are used to fill the gap between theory classes and practical sessions. Both sessions are evaluated as lab experiments for 20 marks.

NOTE:

1. Develop and execute the following programs using 8086 Assembly Language. Any suitable assembler like MASM/TASM/8086 kit or any equivalent software may be used.
2. Program should have suitable comments.
3. The board layout and the circuit diagram of the interface are to be provided to the student during the examination.
4. Software Required: Open source ARM Development platform, KEIL IDE and Proteus for simulation

SOFT WARE PROGRAMS-PART A

1. Design and develop an assembly language program to search a key element "X" in a list of 'n' 16-bit numbers. Adopt Binary search algorithm in your program for searching.

2. Design and develop an assembly program to sort a given set of 'n' 16-bit numbers in ascending order. Adopt Bubble Sort algorithm to sort given elements.

3. Develop an assembly language program to reverse a given string and verify whether it is a palindrome or not. Display the appropriate message.

4. Develop an assembly language program to compute nCr using recursive procedure. Assume that 'n' and 'r' are non-negative integers.

5. Design and develop an assembly language program to read the current time and Date from the system and display it in the standard format on the screen.

6. To write and simulate ARM assembly language programs for data transfer, arithmetic and logical operations (Demonstrate with the help of a suitable program).

7. To write and simulate C Programs for ARM microprocessor using KEIL (Demonstrate with the help of a suitable program).

HARD WARE PROGRAMS-PART B

8. a. Design and develop an assembly program to demonstrate BCD Up-Down Counter (00-99) on the Logic Controller Interface.

   b. Design and develop an assembly program to read the status of two 8-bit inputs (X & Y) from the Logic Controller Interface and display X*Y.

9. Design and develop an assembly program to display messages "FIRE" and "HELP" alternately with flickering effects on a 7-segment display interface for a suitable period of time. Ensure a flashing rate that makes it easy to read both the messages (Examiner does not specify these delay values nor is it necessary for the student to compute these values).

10. Design and develop an assembly program to drive a Stepper Motor interface and rotate the motor in specified direction (clockwise or counter-clockwise) by N steps (Direction and N are specified by the examiner). Introduce suitable delay between successive steps. (Any arbitrary value for the delay may be assumed by the student).

11. Design and develop an assembly language program to

a. Generate the Sine Wave using DAC interface (The output of the DAC is to be displayed on the CRO).

b. Generate a Half Rectified Sine waveform using the DAC interface. (The output of the DAC is to be displayed on the CRO).

12. To interface LCD with ARM processor-- ARM7TDMI/LPC2148. Write and execute programs in C language for displaying text messages and numbers on LCD.

13. To interface Stepper motor with ARM processor-- ARM7TDMI/LPC2148. Write a program to rotate stepper motor.

<u>Study Experiments:</u>

1. Interfacing of temperature sensor with ARM freedom board (or any other ARM microprocessor board) and display temperature on LCD.

2. To design ARM cortex based automatic number plate recognition system.

3. To design ARM based power saving system.

INDEX SHEET

Sl. No	Name of the Experiment	Date			Manual Marks (Max . 20)	Record Marks (Max. 10)	Signature (Student)	Signature (Faculty)
		Conduction	Repetition	Submission of Record				
01	Laboratory Session-1:							
02	Laboratory Session-2:							
03	Search a 16 bit Number							
04	Sorting the 16 bit numbers							
05	Palindrome							
06	NcR							
07	Display System Time							
08	ARM programming							
09	C Programs for ARM microprocessor							
10	BCD Up-Down Counter							
11	7-segment display							
12	Stepper Motor interface							
13	DAC interface							
14	Interface LCD with ARM processor							
15	Interface Stepper motor with ARM processor							
Average								

Note:

- **If the student fails to attend the regular lab, the experiment has to be completed in the same week. Then the manual/observation and record will be evaluated for 50% of maximum marks.**

Course Objectives

This course will enable students to

- To provide practical exposure to the students on microprocessors, design and coding knowledge on 80x86 family/ARM.
- To give the knowledge and practical exposure on connectivity and execute of interfacing devices with 8086/ARM kit like LED displays, Keyboards, DAC/ADC, and various other devices

Course Outcomes

After studying this course, students will be able to

- Learn 80x86 instruction sets and gins the knowledge of how assembly language works.
- Design and implement programs written in 80x86 assembly language.
- Know functioning of hardware devices and interfacing them to x 86 families.
- Choose processors for various kinds of applications.

Graduate Attributes

- Engineering Knowledge
- Problem Analysis
- Modern Tool Usage
- Conduct Investigations of Complex Problems
- Design/Development of Solutions

General Instructions

- All laboratory experiments (all 7 + 6 nos) are to be included for practical examination.
- Students are allowed to pick one experiment from each of the lot.
- PART –A: Procedure + Conduction + Viva: 10 + 25 +05 (40)
- PART –B: Procedure + Conduction + Viva: 10 + 25 +05 (40)
- Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.
- Students should maintain an observation book along with Manual and record.
- Observation book will be evaluated for 20 Marks and Manual for 10 Marks and final IA for 10 Marks.
- Students should complete the observation book which should include the logic and tracing of the respective program and should get it evaluated before departing from the lab.
- They should produce the lab record Next week which should include lab set programs with comments and necessary Board Lay out and Circuit diagram if any.
- If in case the student is unable to attend the regular batch, He / She should take prior permission from the concerned faculty and try to attend the next batch.

CONTENTS

MASM COMMANDS:

C :/>cdfoldername

C:/foldername>edit filename.asm

After this command executed in command prompt an editor window will open. Program should be typed in this window and saved. The program structure is given below.

Structure of Program:

```
.model tiny/small/medium/large

.Stack <some number>

.data
        ; Initialize data
        ; which is used in program.
.code
        ; Program logic goes here.
        ;
end
```

To run the program, the following steps have to be followed:

C:/foldername>masm filename.asm

After this command is executed in command prompt if there are no errors in program regarding to syntax the assembler will generates an object module as discuss above.

C:/foldername>link filename.obj

After verifying the program for correct syntax and the generated object files should be linked together. For this the above link command should be executed and it will give an EXE file if the model directive is small as discuss above.

C:/foldername>debug filename.exe

After generating EXE file by the assembler it’s the time to check the output. For this the above command is used and the execution of the program can be done in different ways. It is as shown below:

_ g ; complete execution of program in single step.

_ t ; Stepwise execution.

_d ds: starting address or ending address ; To see data in memory locations

_p ; Used to execute interrupt or procedure during stepwise execution of program

_ q ; To quit the execution.

Laboratory Session-1

Write-up on Microprocessors, 8086 Functional block diagram, Pin diagram and description.

Description:

The microprocessors functions as the CPU in the stored program model of the digital computer. Its job is to generate all system timing signals and synchronize the transfer of data between memory, I/O, and itself. It accomplishes this task via the three-bus system architecture previously discussed.

The microprocessor also has a S/W function. It must recognize, decode, and execute program instructions fetched from the memory unit. This requires an Arithmetic-Logic Unit (ALU) within the CPU to perform arithmetic and logical (AND, OR, NOT, compare, etc) functions.

The 8086 CPU is organized as two separate processors, called the Bus Interface Unit (BIU) and the Execution Unit (EU). The BIU provides H/W functions, including generation of the memory and I/O addresses for the transfer of data between the outside world -outside the CPU, that is- and the EU.

The EU receives program instruction codes and data from the BIU, executes these instructions, and store the results in the general registers. By passing the data back to the BIU, data can also be stored in a memory location or written to an output device. Note that the EU has no connection to the system buses. It receives and outputs all its data thru the BIU.

FETCH AND EXECUTE

Although the 8086/88 still functions as a stored program computer, organization of the CPU into a separate BIU and EU allows the fetch and execute cycles to overlap. To see this, consider what happens when the 8086 or 8088 is first started.

1. The BIU outputs the contents of the instruction pointer register (IP) onto the address bus, causing the selected byte or word to be read into the BIU.

2. Register IP is incremented by 1 to prepare for the next instruction fetch.

3. Once inside the BIU, the instruction is passed to the queue. This is a first-in, first-out storage register sometimes likened to a "pipeline".

4. Assuming that the queue is initially empty, the EU immediately draws this instruction from the queue and begins execution.

5. While the EU is executing this instruction, the BIU proceeds to fetch a new instruction. Depending on the execution time of the first instruction, the BIU may fill the queue with several new instructions before the EU is ready to draw its next instruction.

The BIU is programmed to fetch a new instruction whenever the queue has room for one (with the 8088) or two (with the 8086) additional bytes. The advantage of this pipelined architecture is that the EU can execute instructions almost continually instead of having to wait for the BIU to fetch a new instruction.

There are three conditions that will cause the EU to enter a "wait" mode. The first occurs when an instruction requires access to a memory location not in the queue. The BIU must suspend fetching instructions and output the address of this memory location. After waiting for the memory access, the EU can resume executing instruction codes from the queue (and the BIU can resume filling the queue).

The second condition occurs when the instruction to be executed is a "jump" instruction. In this case control is to be transferred to a new (nonsequential) address. The queue, however, assumes that instructions will always be executed in sequence and thus will be holding the "wrong" instruction codes. The EU must wait while the instruction at the jump address is fetched. Note that any bytes presently in the queue must be discarded (they are overwritten).

One other condition can cause the BIU to suspend fetching instructions. This occurs during execution of instructions that are slow to execute. For example, the instruction AAM (ASCII Adjust for Multiplication) requires 83 clock cycles to complete. At four cycles per instruction fetch, the queue will be completely filled during the execution of this single instruction. The BIU will thus have to wait for the EU to pull over one or two bytes from the queue before resuming the fetch cycle.

A subtle advantage to the pipelined architecture should be mentioned. Because the next several instructions are usually in the queue, the BIU can access memory at a somewhat "leisurely" pace. This means that slow-memory parts can be used without affecting overall system performance.

Fig 1.0 shows the block diagram of 8086 microprocessor.

Fig 1.1 shows the Pin diagram of 8086 microprocessor.

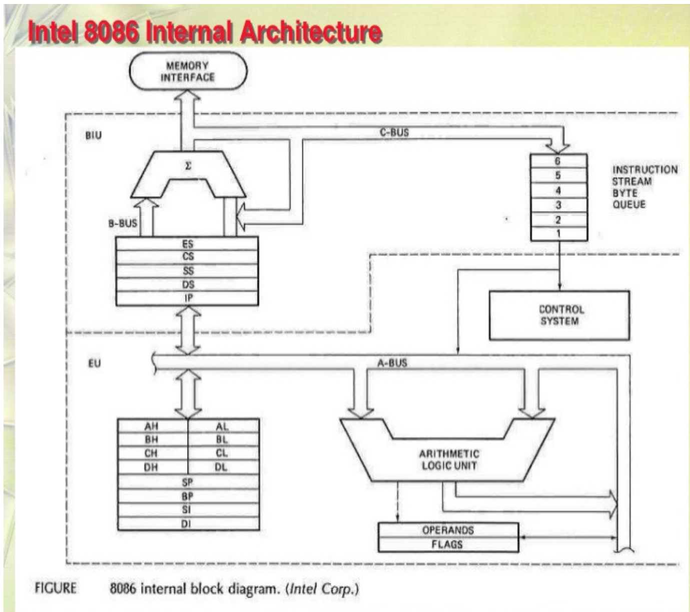


Fig 1.0

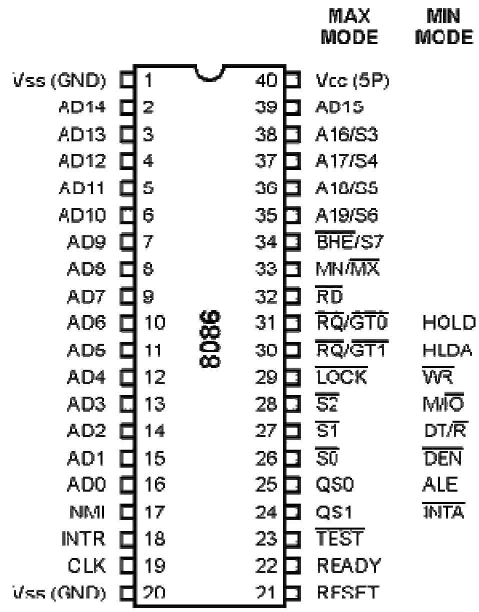


Fig 1.1

Laboratory Session-2

Write-up on Instruction group, Timing diagrams of 8086.

Instruction Set of 8086 is shown in Fig 1.2. The detailed explanation will be taught in the lab.

8086 Instructions

AAA	CMPSB	IRET	JNAE	JP	LOOPZ	PUSHF	SBB
AAD	CMPSW	JA	JNB	JPE	MOV	RCL	SCASB
AAM	CWD	JAE	JNBE	JPO	MOVSB	RCR	SCASW
AAS	DAA	JB	JNC	JS	MOVSW	REP	SHL
ADC	DAS	JBE	JNE	JZ	MUL	REPE	SHR
ADD	DEC	JC	JNG	LAHF	NEG	REPNE	STC
AND	DIV	JCXZ	JNGE	LDS	NOP	REPNZ	STD
CALL	HLT	JE	JNL	LEA	NOT	REPZ	STI
CBW	IDIV	JG	JNLE	LES	OR	RET	STOSB
CLC	IMUL	JGE	JNO	LODSB	OUT	RETF	STOSW
CLD	IN	JL	JNP	LODSW	POP	ROL	SUB
CLI	INC	JLE	JNS	LOOP	POPA	ROR	TEST
CMC	INT	JMP	JNZ	LOOPE	POPF	SAHF	XCHG
CMP	INTO	JNA	JO	LOOPNE	PUSH	SAL	XLATB
				LOOPNZ	PUSHA	SAR	XOR

Fig 1.2

8086 can perform two operations viz. Read and Write.

8086 can operate in two modes viz. Minimum mode and Maximum mode.

Fig 1.3 shows the timing diagram of 8086 Read operation in Minimum mode.

Fig 1.4 shows the timing diagram of 8086 Read operation in Minimum mode.

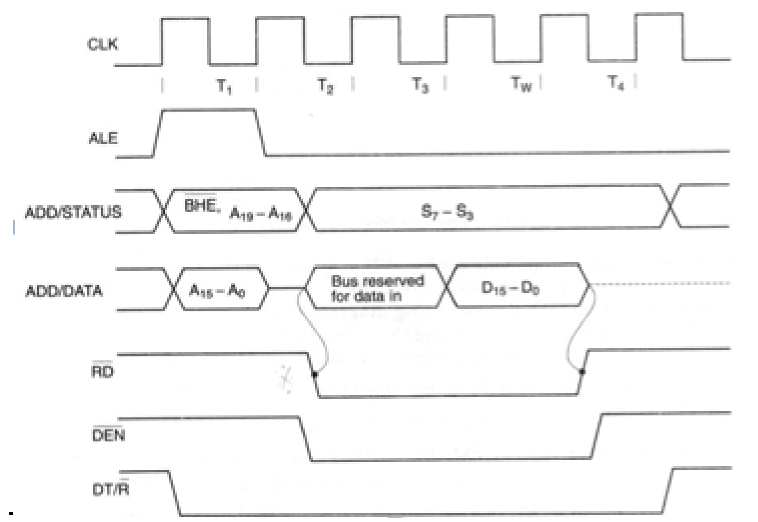


Fig 1.3

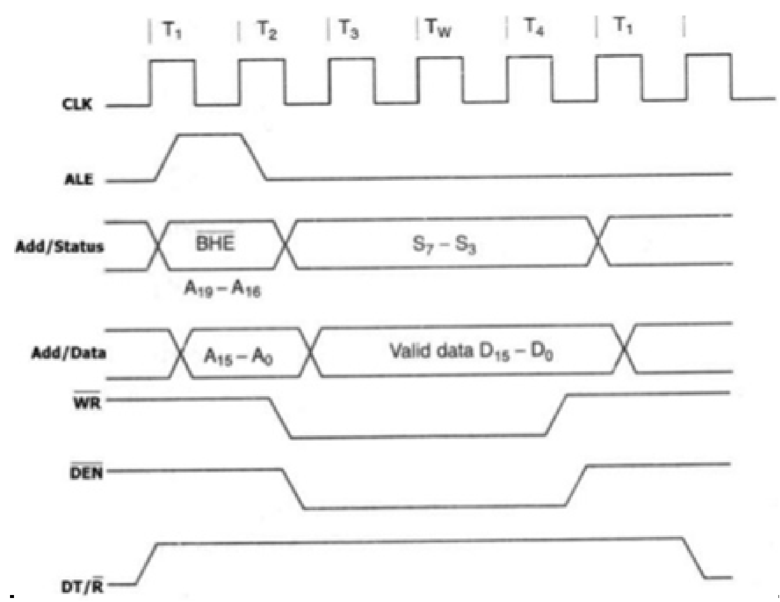


Fig 1.4

SAMPLE PROGRAMS:

1. Write an ALP to move the data between the Registers.

```
.model tiny
.data
      num1 db 50h
num2 dw 1234h

.code
Mov ax,@data
Mov ds,ax      ;DATA SEGMENT INITIALIZATION

mov al,num1
      mov ah,al
mov bh,ah
mov bl,al      ;MOVES BYTE LENGTH OF DATA FROM REG.AL TO REG.BL

mov cx,num2
      mov dx,cx
mov si,ax
mov di,si            ;MOVES WORD LENGHT OF DATA FROM REG.CX TO REG.DX

int 3                ;TERMINATES THE PROGRAM EXECUTION

end
```

2. Write and ALP to move immediate data to Registers.

```
.model tiny
.code
          mov al,10h
          mov ah,10
          mov cl,50h
          mov ch,50       ;MOVES IMMEDIATE VALUE TO 8 BIT REGISTER

          mov bx,1234h
          mov dx,1234     ;MOVES IMMEDIATE VALUE TO 16 BIT REGISTER

          mov si,4000h
          mov di,2000h

          int 3                ;TERMINATE THE PROGRAM EXECUTION
end
```

3. Write an ALP to add two numbers and to store the result in the specified destination.

```
.model small

.data
num1 db 05h
      num2 db 06h
      num3 dw 1234h
      num4 dw 0002h
sum db ?
      sum2 dw ?

.code
      mov ax,@data
mov ds,ax          ;INITIALIZES DATA SEGMENT

      mov al,num1
      mov bl,num2
add al,bl          ;ADD THE 2 BYTES
mov sum,al         ;STORES THE RESULT IN MEMORY

mov cx,num3
add cx,num4        ;ADD THE 2 WORDS
mov sum2,cx        ;STORES THE RESULT IN MEMORY

int 3              ;TERMINATE THE PROGRAM EXECUTION

align 16           ;DS STARTS FROM PAGE BOUNDARY
end
```

4. Write and ALP to multiply two 16-bit numbers and to store the result in the specified location.

```
.model small

.data
         num1 dw 1234h
         num2 dw 0ffffh
         res dw 5 dup(0)

.code
         Mov ax,@data
```

```
        Mov ds,ax      ;INITIALIZATION OF DATA SEGMENT

        mov  ax,num1
        mov  dx,num2
        mul dx           ;MULTIPLIES 2 16-BIT NUMBERS

        mov res,ax
        mov  res+2,dx    ;STORES THE IN MEMORY

        int 3             ;TERMINATE THE PROGRAM EXECUTION
        align 16         ;DS STARTS FROM PAGE BOUNDARY
end
```

5. Write an ALP to divide a 32-bit unsigned number by a 16-bit unsigned number and to store the quotient and remainder in the specified location.

```
.model small

.data
Dvd dd 12345678h
Dvr dw 0ffffh
Quot dw ?
Remd dw ?

.code
        Mov ax,@data
        Mov ds,ax      ;INITIALIZATION OF DATA SEGMENT

        Mov si,offset dvd
        Mov ax,wordptr[si]
        Mov dx,wordptr[si+2]

        Mov cx, dvr
        div cx

        mov quot ,ax
        mov remd, dx

        int 3          ;TERMINATES THE PROGRAM EXECUTION
        align 16       ;DS STARTS FROM PAGE BOUNDARY
end
```

6. Write an ALP to illustrate the operation of AAA instruction. Use Macros

```
.model small

.data
read macro          ;Start of a macro
mov ah,01h          ;read a single key stroke
int 21h
endm        ;end of macro

.code
        Mov ax,@data
        Mov ds,ax      ;INITIALIZATION OF DATA SEGMENT

        read           ;CALL MACRO READ
        mov bl,al      ;STORE THE READ KEY IN BL REGISTER

        read
        mov cl,al

        add al,bl      ;ADD AL WITH BL AND STORES THE RESULT IN AL.

        mov dl,al
        mov ah,0
        aaa            ;ADJUST THE AL VALUE TO UNPACKED BCD

        mov si,ax

        int 3          ;TERMINATES THE PROGRAM EXECUTION
end
```

SOFTWARE PROGRAMS: PART A

Program No.01. **Date:**

BINARY SEARCH

<u>AIM:</u>

Design and develop an assembly language program to search a key element "X" in a list of 'n' 16-bit numbers. Adopt Binary search algorithm in your program for searching.

```
.model small

.data                                           ;start of the data segment

        arr     dw 0111h,0112h,0113h,0114h,0115h ; 'n' elements to be searched
        len     dw ($-arr)/2
        key     equ 0116h                         ; key element to be searched
        msg1    db "found$"
        msg2    db "not found$"

.code                                             ; start of the code segment

        mov ax,@data                              ;initialization of data segment
        mov ds,ax

        mov bx,00                                 ; first data position to bx.
        mov dx,len                                ; last data position to dd.
        mov cx,key

again:  cmp bx,dx
        ja notfnd

        mov ax,bx
        add ax,dx
        shr ax,1                                  ;Get the middle element of array
        mov si,ax
        add si, si

        cmp cx,arr[si]                            ;compare the key with middle
        jae big                                   ; element of array
```

```
          dec ax
          mov dx,ax                         ;last element of new array to dx
          jmp again

big:      je found
          inc ax

          mov bx,ax
          jmp again


found:    lea dx,msg1                       ;content of the string to be displayed.
          Jmp displ

notfnd:   lea dx,msg2                       ;content of the string to be displayed.

displ  :  mov ah,09h
          int 21h
          int 3                             ; Terminates the execution
end                                         ;end of program
```

Conclusion:

This program performs a search for a key element in an array. If the search element is found it will display a message '**found**'. As the search element (key element in program) is not present in the given array it will display a message '**not found**'.

Date: **Signature of the staff**

Program No.02. **Date:**

SORTING A GIVEN SET OF 16-BIT UNSIGNED INTEGERS INTO ASCENDING ORDER

Design and develop an assembly program to sort a given set of 'n' 16-bit numbers in ascending order. Adopt Bubble Sort algorithm to sort given elements.

```
.model small
.data
    ARR DW 3333h, 4444h, 1111h, 9999h, 5555h, 2222h, 7777h, 8888h, 6666h
                                                    ; The numbers to be sorted
    LEN EQU $-ARR                                   ; Length of the array
   .code
     MOV AX, @DATA
     MOV DS, AX
    MOV CX, (LEN/2)-1          ; Get the total number of Elements In the   array

  OUTER:  LEA SI, ARR          ; Get the address of the first element of the   array


     MOV BX, 0
                               ; to have a count of number of    Comparison
      MOV SI, 00
Inner : inc bx
     MOV AX, ARR [SI]
     INC SI
     INC SI                                          ; Get the next value
     CMP AX, ARR [SI]                                ; Perform the comparison
     JBE SKIP                                        ; Skip if 1st Value is less than 2nd.
     XCHG AX, ARR [SI]                               ; Else Exchange the two values.
     MOV ARR [SI-2], AX                              ; Swap the two values.

 SKIP: CMP BX, CX                                   ; compare the total no of Comparison
    JL   INNER                                       ; Repeat if necessary.
```

```
   LOOP OUTER                          ; Inner loop is for no of iterations
      INT 3H                           ; Outer loop is for no of comparison
END START
```

Conclusion:

This program will sort the given numbers in ascending order. The sorted numbers will be stored directly in the **data Segment**. To view the data segment the following code must be used.

-d ds: 0

Date: **Signature of the staff**

Program No.03. **Date:**

Check a string for a Palindrome

Develop an assembly language program to reverse a given string and verify whether it is a palindrome or not. Display the appropriate message.

```
.model small
.data
        str1    db "alam"               ; String to be checked for palindrome
        slen    equ ($-str1)
        str2    db 40 dup(0)
        msg1    db "Palindrome$"
        msg2    db "Not Palindrome$"
.code
start:  mov     ax,@data
        mov     ds,ax
        mov     es,ax                   ; Initialize extra segment

        mov     cx,slen                 ; Length of the string
        lea     si, str1
        add     si,slen – 1             ; get the last byte of the data
        lea     di, str2

up:     mov     al,[si]
        mov     [di],al                 ; load the byte from [Si] to [Di]
        dec     si
        inc     di
        loop    up                      ; Repeat the process

        lea     si, str1
        lea     di, str2
        mov     cx,slen
        cld                             ; Clear the direction flag
repe    cmpsb                           ; compare the string bytes present in SI & DI
        jne     down                    ; Jump if the strings are not equal

        lea     dx, msg1
        jmp     down1
down:   lea     dx, msg2
```

```
down1: mov    ah, 09h
       int    21h
       int    3                     ; Terminate the program
       end    start
```

Conclusion:

This program reverse the string provided in data segment by keeping the original string as it is and compares both the strings. It will check each and every character. If all the characters are same then the given string is said to be as palindrome and it will display a message "**palindrome**" on screen otherwise the given string is not palindrome and it will display a message "**not palindrome**" on screen.

Date: **Signature of the staff**

Program No.04. **Date:**

NcR

Develop an assembly language program to compute nCr using recursive procedure. Assume that 'n' and 'r' are non-negative integers.

```
.model small
.stack   20
.data
         n       db 08h
         r       db 05h
         ncr     db ?
.code
start:   mov     ax,@data
         mov     ds,ax
         mov     ncr,00h
         mov     al,n
         mov     bl,r
         call    encer
         int     3
encer    proc
para1:   cmp     al,bl                ; compare 'n','r' for equality
         je      para8
para2:   cmp     bl,00h               ; compare 'r' with 00
         je      para8
para3:   cmp     bl,01h               ; compare 'r' with 01h
         je      para10
para4:   dec     al                   ; decrement 'n'
         cmp     bl,al
         je      para9
para5:   push    ax                   ; Push 'n' to the stack
         push    bx                   ; Push 'r' to the stack
         call    encer
para6:   pop     bx                   ; Get 'r' and 'n' from the stack
         pop     ax
         dec     bl
         push    ax
         push    bx
```

```
        call    encer
para7:  pop     bx
        pop     ax
        ret
para8:  inc     ncr
        ret                             ; Store the results
para9:  inc     ncr
para10: add     ncr,al
        ret
encer   endp
        end     start
```

Conclusion:

This program performs nCr using recursive procedure. Output is stored in data segment. To observe the output in data segment we have to search for our given **'n'** and **'r'** values as program is written to store the result after the given data in data segment.

The NcR Value for 8 and 5 is 56, But the outputwill be shown as 38 which is the Hexa value of 56.

Date: **Signature of the staff**

Program No.05. **Date:**

DISPLAY SYSTEM TIME

Design and develop an assembly language program to read the current time and Date from the system and display it in the standard format on the screen.

```
.MODEL SMALL
.DATA
msg     db "The Time  is: "
             hrs      db ?,?,' : '
             mins    db ?,?,' (hh:mm) ',10,13

        db "The Date is: "
            da  db ?,?, '/'
            mon  db ?,?, '/'
            yea  db ?,?, '(dd/mm/yy)', 10,13,'$'


.CODE
 MOV AX,@DATA
 MOV DS, AX
; Time Part
         mov   ah,2ch                   ; DOS function to read system time
         int   21h

         mov   al,ch                    ; load the hours to 'al'
         aam                            ; ASCII adjust after multiplication
         add   ax, 3030h
         mov   hrs, ah
         mov   hrs+1, al

         mov   al,cl                    ; load the seconds to 'al'
         aam
         add   ax, 3030h
         mov   mins, ah
         mov   mins+1,al
```

```
; Day Part
MOV AH, 2AH    ; To get System Date
INT 21H
MOV AL, DL     ;   Day is in DL
AAM
Add ax,3030h
mov da,Ah
mov da +1, al

MOV AL, DH      ; Month is in DH
AAM
Add ax, 3030h
MOV mon,AH
mov mon+1,al

; YEAR

ADD CX, 0F830H; To negate the effects of 16bit value,

MOV Al, cl          ; since AAM is applicable only for AL (YYYY -> YY)
aam
Add ax, 3030h

mov yea,ah
mov yea+1,al

        lea     dx,msg                  ; Display the time
        mov     ah,09h
        int     21h

int 3

end
```

Conclusion:

This program displays the present system time. Our program displays only the hours and minutes in the format HH: MM. By using the same ***DOS function*** we can also display the seconds and milliseconds.

Date: **Signature of the staff**

Program No.06. **Date:**

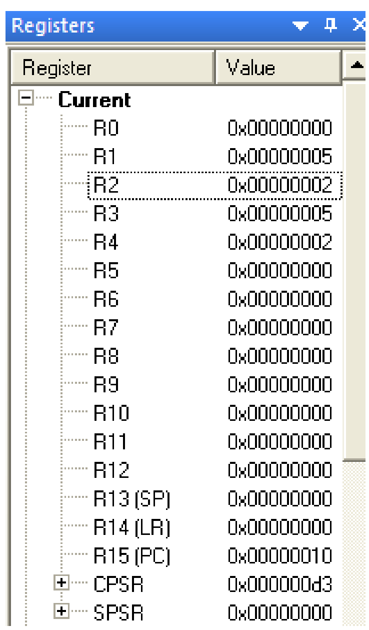
Simple ARM Programs

To write and simulate ARM assembly language programs for data transfer, arithmetic and logical operations (Demonstrate with the help of a suitable program).

1. **Data Transfer.**

The below assembly level program moves the 32 bit data from register to register.

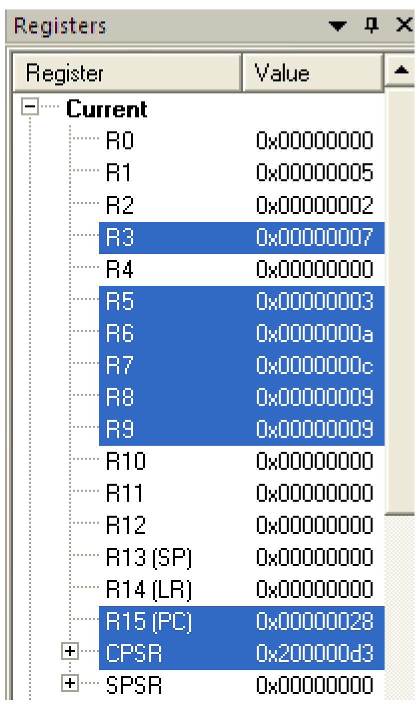
```
        area movt, code, readonly
        entry
        mov r1,#0005          ; Mov immediate 32 bit data to r1
        mov r2,#0002          ; Mov immediate 32 bit data to r1
        mov r3,r1             ; Register-Register movement
        mov r4,r2             ; Register-Register movement

stop b stop                   ; End of the program
         end
```



2. Arithmetic Operations

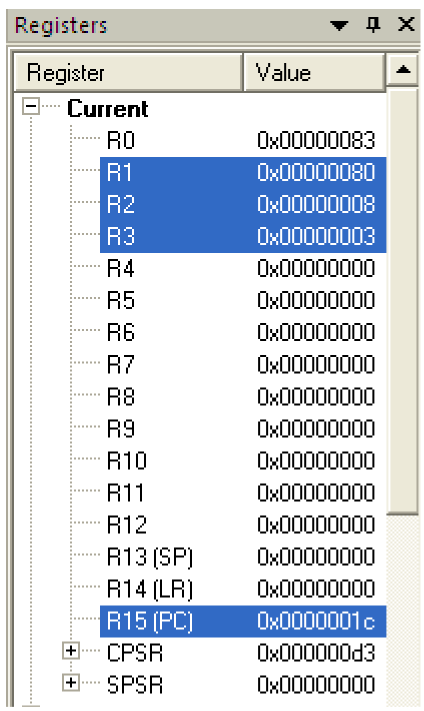
A. Addition, Subtraction and Multiplication:

```
area addt, code, readonly
entry
mov r1,#0005            ; Mov immediate 32 bit data to r1
mov r2,#0002            ; Mov immediate 32 bit data to r2
add r3,r2,r1            ; Add the contents present in r2 with the
                          contents of r1 and store in r3
sub r5,r1,r2            ; Subtract; r5 = r1-r2
mul r6,r1,r2            ; Multiply
mov r7,r6
add r7,#2               ; Add immediate data
mov r8,r7
sub r8,#3               ; Subtract immediate data
mov r9,r8

stop b stop
     end
```



3. Logical operations : To perform AND, Logical Shift operations,

```
        area dis,code,readonly
        entry
        mov r0,#0x83
        mov r1,r0
        and r1, # 0Xf0                  ; Perform Logical AND operation
        mov r2,r1
        lsr r2, #4                      ; Perform Logical right Shift operation
        mov r3, r0
        and r3, # 0X0f

stop b stop
         end
```



4. Write the similar programs and try for OR, Logical Left Shift operations.

Date: **Signature of the staff**

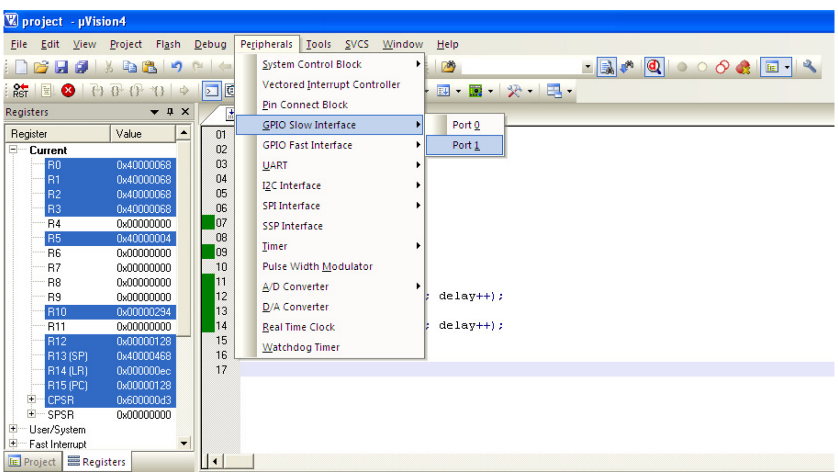
Program No.07. **Date:**

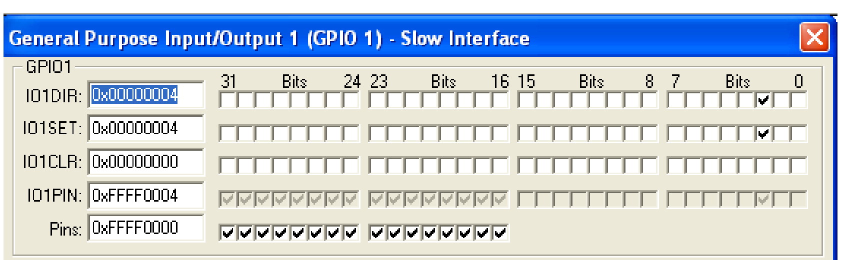
<u>'C' PROGRAMS FOR ARM PROCESSOR</u>

To write and simulate C Programs for ARM microprocessor using KEIL (Demonstrate with the help of a suitable program)

Program: To write a C program to Blink a LED /Port Pin with LPC 2148 ARM 7 Microcontroller.

```
#include <lpc214x.h>                    //Header File "x" can be wrt to controller
 unsigned int delay;

        int main(void)
              {
               IO1DIR = (4);                  // Bit No 4 (0100) will be activated

               while(1)                                  // If True
              {
                 IO1CLR = (04);                          // Clear Bit 04 of GPIO1
                for (delay=0 ;delay<5000; delay++);    // Call Delay
                IO1SET = (04);                         // Set Bit 04 of GPIO1
                for (delay=0; delay<5000; delay++);   // Call Delay
              }
            }
```





HARDWARE PROGRAMS: PART B

Program No.08. **Date:**

COUNTERS AND MULTPLYER

a. Design and develop an assembly program to demonstrate BCD Up-Down Counter (00-99) on the Logic Controller Interface.

```
.model small
.data
pa equ 0d800h
pb equ 0d801h
pc equ 0d802h
ctrlequ 0d803h

.code
mov ax, @data
mov ds, ax

mov al, 80h
mov dx, ctrl
out dx, al

mov cx, 0Ah          ; Load 10 Counts
mov al, 00h
Next:  mov dx, pa          ; Configure Port A as output port and send the of counts
out dx, al
call delay
inc al               ; Perform up counting.
loop Next

mov cx, 0Ah      ; Load 10 Counts
mov al, 09h
rpt:    mov dx, pa
out dx, al
call delay
dec al               ; Perform up counting.
Loop rpt
int 3h
```

```
delay proc

push cx
push bx

mov cx, 0ffffh
  L1:  mov bx, 8fffh
  L2:  dec bx
jnz L2
loop L1

pop bx
pop cx

ret
delay endp
end
```

Conclusion:

The program performs the up-down counter based on the input data given on logic controller read through port B. If the input is zero then it performs down counter starting from 99 down to 00 and if other than zero is the input then it performs up counter starting from 00 down to 99. And the counting will continue until a key ‘q’ is pressed in the key board, after displaying the count on logic controller every time it checks whether a key ‘q’ is pressed or not.

While observing the output of down counter or up counter if the input changes then from that point the counting will also changes. Suppose if the input is zero then it perform down counting from 99 to 00 after some time when the output is 50 then if we change the input other than zero then from that point it will start up counting that is form 50, 51, 52. and so on.

b. Design and develop an assembly program to read the status of two 8-bit inputs (X & Y) from the Logic Controller Interface and display X*Y.

```
.model small
.data
pa equ 0d800h
pb equ 0d801h
pc equ 0d802h
ctrl equ 0d803h
.code
        movax,@data
        movds,ax

        mov al,82h   ; Control word (PB as input port and PA as output port)
        mov dx, ctrl
        out dx, al

        mov dx, pb
        in al,dx              ; Read the first 8 bit number
        mov bl,al             ; Store the first number

top:    mov ah,1     ; Read a character from the key board
        int 21h
        cmp al,13    ; Compare the character with the "ENTER" key, cmp al,0dh
        jnz top
        mov dx, pb   ; Read the Second 8 bit number
        in al,dx     ; Store the first number
        mul bl       ; Multiplybl*al

        mov dx, pa
        out dx, al   ; Display the result
        int 3
        end
```

Conclusion:

The program performs the multiplication between two bytes and gives the result. First byte is read from the port B of logic controller (user has to provide) and waits for enter key to be pressed and once enter key is and it reads the Second byte and multiplies and displays the result through Port A.

Date: **Signature of the staff**

Program No.09. **Date:**

7-SEGMENT DISPLAY INTERFACE

Design and develop an assembly program to display messages “FIRE” and “HELP” alternately with flickering effects on a 7-segment display interface for a suitable period of time. Ensure a flashing rate that makes it easy to read both the messages (Examiner does not specify these delay values nor is it necessary for the student to compute these values).

```
.model small
.stack 100
.data
        pa      equ 0d800h              ; Port address
        pb      equ 0d801h
        pc      equ 0d802h
        ctrl    equ 0d803h              ; Control word address
        str1    db 8eh, 0f9h, 88h, 86h  ; Hexa values for "FIRE"
        str2    db 89h, 86h, 0c7h, 8ch  ; Hexa values for "HELP"
.code
start:  mov     ax, @data
        mov     ds, ax                  ; data segment Initialization

        mov     al, 80h                 ; control word
        mov     dx, ctrl
        out     dx, al

again:  mov     bx, offset str1
        call    display                 ; Jump to display procedure
        call    delay                   ; Jump to delay procedure
        mov     bx, offset str2
        call    display
        call    delay

        mov     ah, 06h                 ; direct console input or output
        mov     dl, 0ffh
        int     21h                     ;get character from keyboard buffer (if any)
        cmp     al, 'q'
```

```
        jne     again
        int     3                       ; Terminate the program
display proc
        mov     si, 03h                 ; To get the last byte
up1:    mov     cl, 08h
        mov     ah, [bx+si]             ; Load  the data bit to 'ah'
up:     mov     dx, pb
        rol     ah, 1                   ;Rotate each bit in the data by one
        mov     al, ah
        out     dx, al                  ; Out the first bit

        call    clock
        dec     cl
        jnz     up                      ; repeat the steps '08' times
        dec     si
        cmp     si, -1
        jne     up1
        ret                             ; return back to main program
display endp

clock   proc
        mov     dx, pc
        mov     al, 01h                 ; rising edge of clock pulse
        out     dx, al
        mov     al, 0                   ; falling edge of the clock pulse
        out     dx, al
        mov     dx, pb
        ret
clock   endp

delay   proc
        push    cx
        push    bx
        mov     cx, 0ffffh
d2:     mov     bx, 8fffh
d1:     dec     bx
        jnz     d1
        loop    d2
        pop     bx
```

```
        pop     cx
        ret
delay   endp

        end     start
```

Conclusion:

This program displays “FIRE” and “HELP” on seven segment display interface recursively one after the other with some delay till key ‘q’ is pressed on key board. It’s not going to read any data from interface device. The data which has to be displayed is provided in the program itself.

Date: **Signature of the staff**

Program No.10. **Date:**

STEPPER MOTOR INTERFACE

Design and develop an assembly program to drive a Stepper Motor interface and rotate the motor in specified direction (clockwise or counter-clockwise) by N steps (Direction and N are specified by the examiner). Introduce suitable delay between successive steps. (Any arbitrary value for the delay may be assumed by the student).

```
.model small
.data
        pa      equ 0d800h
        pb      equ 0d801h
        pc      equ 0d802h
        ctrl    equ 0d803h
        nstep   db 2                ; Initialize the number of steps
.code
start:  mov     ax, @data
        mov     ds, ax

        mov     al, 80h             ; All ports are output ports
        mov     dx, ctrl
        out     dx, al

        mov     bh, nstep
        mov     al, 88h

again1: call    step
        rol     al, 1       ; for counter-clock wise direction
                            ; Replace rol al,1 with ror al,1 for clock wise direction
        dec     bh
        jnz     again1

        int     3

step    proc
        mov     dx, pa
        out     dx, al
```

```
        push    cx
        push    bx

        mov     cx, 0ffffh
d2:     mov     bx, 8fffh
d1:     dec     bx
        jnz     d1
        loop    d2

        pop     bx
        pop     cx

        ret
step    endp
        end     start
```

Conclusion:

This program drives a stepper motor interface to rotate by 8 steps in anti-clockwise direction. After each rotation a delay is introduced to observe the rotation. After completing the rotations the execution will get stopped.

Date: **Signature of the staff**

Program No.11. **Date:**

DAC INTERFACE

Design and develop an assembly language program to

a. Generate the Sine Wave using DAC interface (The output of the DAC is to be displayed on the CRO).

b. Generate a Half Rectified Sine waveform using the DAC interface. (The output of the DAC is to be displayed on the CRO).

A. SINE WAVE

```
.model small
.data
        pa      equ 0c400h
        pb      equ 0c401h
        pc      equ 0c402h
        ctrl    equ 0c403h
        table   db 128,132,137,141,146,150,154,159,163,167,171,176,180,184,188
                db 192,196,199,203,206,210,213,217,220,223,226,229,231,234,236
                db 239,241,243,245,247,248,250,251,252,253,254,255
                db 255,254,253,252,251,250,248,247,245,243,241,239,236,234,231
                db 229,226,223,220,217,213,210,206,203,199,196,192,188,184,180
                db 176,171,167,163,159,154,150,146,141,137,132,128
                db123,119,114,110,105,101,97,93,88,84,80,76,72,68,64,60,56,52,49
                db 45,42,39,36,33,30,27,24,22,19,17,15,11,9,7,6,5,4,3,2,1,0
                db 0,1,2,3,4,5,6,7,9,11,15,17,19,22,24,27,30,33,36,39,42,45,49,52,56
                db 60,64,68,72,76,80,84,88,93,97,101,105,110,114,119,123
.code
start:  mov     ax,@data
        mov     ds,ax

        mov     al,80h                  ; All the ports are out put ports
        mov     dx,ctrl
        out     dx,al
```

```
again:   mov   bx,05h
up:      mov   cx,164               ; Load 164 values
         mov   si,00h
         mov   dx,pa

again1:  mov   al,table[si]         ; Load each value from Look-up-table to al
         out   dx,al
         inc   si
         loop  again1

         dec   bx
         cmp   bx,00
         jne   up

         mov   ah,06h               ; direct console input or output
         mov   dl,0ffh              ; Read the character from the keyboard
         int   21h
         jz    again
         int   3
         end   start
```

Conclusion:

This program generates a sine wave of having amplitude of 5V. Output will be seen in CRO. It will be continues wave. It stops execution as soon as any key is pressed from the key board.

B. Half Rectified Sine Wave:

```
.model small
.data
        pa      equ 0c400h
        pb      equ 0c401h
        pc      equ 0c402h
        ctrl    equ 0c403h
        table   db 128,132,137,141,146,150,154,159,163,167,171,176,180,184,188
                db 192,196,199,203,206,210,213,217,220,223,226,229,231,234,236
                db 239,241,243,245,247,248,250,251,252,253,254,255,254,253,252
                db 251,250,248,247,245,243,241,239,236,234,231,229,226,223,220
                db 217,213,210,206,203,199,196,192,188,184,180,176,171,167,163
                db 159,154,150,146,141,137,132,128  ; Look_up_table
.code
start:  mov     ax,@data
        mov     ds,ax

        mov     al,80h              ; All the ports are output ports
        mov     dx,ctrl
        out     dx,al

again3: mov     bx,05h
up:     mov     cx,83               ; Load 83 values
        mov     si,00

again4: mov     dx,pa
        mov     al,table[si]        ; Load each value from Look-up-table to al
        out     dx,al
        inc     si
        loop    again4

        mov     cx,83
        mov     al,128
next:   out     dx,al
        loop    next
        dec     bx
        cmp     bx,00h
```

```
        jnz     up

        mov     ah,06h          ; direct console input or output
        mov     dl,0ffh         ; Read the character from the keyboard
        int     21h
        jz      again3
        int     3               ; Terminate the program
end     start
```

Conclusion: This program generates a half - rectified sine wave of 5V amplitude. Output will be seen in CRO. It stops execution as soon as any key is pressed from the key board.

Date: **Signature of the staff**

Program No.11. **Date:**

INTERFACE LCD WITH ARM PROCESSOR

To interface LCD with ARM processor-- ARM7TDMI/LPC2148. Write and execute programs in C language for displaying text messages and numbers on LCD

```
#include <LPC214x.h>

void cmd(unsigned char d);
void datal(unsigned char t);
void delay (int count);

int main()
{
int i;
unsigned char name[]={"AMMLUI"};

IO0DIR=0x30403C00;
delay(100);

cmd(0x02);                    //cursor home command
cmd(0x01);                    //clear display command
cmd(0x28);          //4-bit mode entry command(0x38 for 8 bit mode)
cmd(0x06);                    //entry mode command
cmd(0x0C);                    //display on cursor off command
//cmd(0xC0);                  //LCD bottom line display command

for (i=0;i<11;i++)
{
datal(name[i]);
}
while(1);
}

void cmd(unsigned char d)
{
int a=0;
a = d | 0xFFFFFF0F;
IO0CLR |= 0x00003C00;
a=a<<6;
IO0CLR = 0x20400000;
IO0SET = 0x10000000;
IO0SET = (IO0SET | 0x00003c00) &a;
delay (1000);
IO0CLR = 0x10000000;
```

```
a=0x0;
d=d<<4;
a = d | 0xFFFFFF0F;
IO0CLR |= 0x00003C00;
a=a<<6;
IO0CLR = 0x20400000;
IO0SET = 0x10000000;
IO0SET = (IO0SET | 0x00003C00)&a;
delay(1000);
IO0CLR = 0x10000000;
}
void datal(unsigned char t)
{
int b=0;
b = t|0xFFFFFF0F;
IO0CLR |= 0x00003C00;
b=b<<6;
IO0SET = 0x10400000;
IO0SET = (IO0SET | 0x00003C00)&b;
delay(1000);
IO0CLR = 0x10000000;

b=0x0;
t=t<<4;
b=t|0xFFFFFF0F;
IO0CLR |= 0x00003C00;
b=b<<6;
IO0SET = 0x10400000;
IO0SET = (IO0SET | 0x00003C00) &b;
delay(1000);
IO0CLR = 0x10000000;
}
Void delay (int count)
{
int j=0, i=0;
for (j=0;j<count;j++)
for (i=0;i<35;i++);
}
```

Date: **Signature of the staff**

Program No.12. **Date:**

INTERFACE STEPPER MOTOR WITH ARM PROCESSOR

To interface Stepper motor with ARM processor-- ARM7TDMI/LPC2148. Write a program to rotate stepper motor

```
#include <LPC214X.h>

void delay();


void delay()
{
  int i,j;
  For (i=0; i<0xff; i++)
     For (j=0; j<0x25; j++);
}


int main()
{

  IO0DIR=0x000F0000;                  ; Consider ARM port Pin from 16-19
                                      ; And set these pins
  While (1)
  {
  //while (IO0PIN & 0x00008000);
  //while (! (IO0PIN & 0x00008000));

IO0PIN=0x00010000;
delay ();
IO0PIN=0x00020000;
delay ();
IO0PIN=0x00040000;
delay ();
IO0PIN=0x00080000;
delay();


 }
}
```

; This is for Clock wise rotation

; For Anti- Clock wise Change the direction as 8,4,2,1

Date: **Signature of the staff**

STUDY EXPERIMENTS

1. Interfacing of temperature sensor with ARM freedom board (or any other ARM Microprocessor board) and display temperature on LCD.

```
#include<lpc214x.h>

#include<stdio.h>


#define vol           3.35     //Reference voltage

#define fullscale     0x3ff    //10 bit adc


//Function prototypes

void lcd_init(void);

void wr_cn(void);

void clr_disp(void);

void delay(unsigned int);

void lcd_com(void);

void wr_dn(void);

void lcd_data(void);


unsigned char temp1, pwr_on_flag=0xFF;          //unsigned char

unsigned long int temp,r=0,i=0,j=0,temp_arry[10];;

char disp[] = "WELCOME TO ALS",disp1[] = "BANGALORE";

char disp2[] = "TEMP_SENSOR",disp3[] = "INTERFACING", disp4[] = "TEMP C=";
```

```
unsigned int temp_adc=0, adc_value=0;

unsigned long sum=0;

float ana_output , adc_out , temp_out ,avg_out = 0.0 ;

char var[15],*ptr;


int main()

{

        PINSEL0 = 0X00000000;               // configure as GPIO

        PINSEL1 = 0X00040000;               //AD0.4 pin is selected (P0.25)

        IO0DIR = 0x000000FC;                //configure o/p lines for lcd


        lcd_init();                         //lcd initialization

        delay(3200);                        // delay about 1ms


          clr_disp();                       //clear display

        delay(3200);                        // delay about 1ms



        temp1 = 0x81;        //Display starting address of first line 2nd pos

        lcd_com();           //function to send command to LCD display

        ptr = disp;                                  // pointing data



while(*ptr!='\0')
  {
```

```
                temp1 = *ptr;
        lcd_data();                     // function to write data  on LCD
                ptr ++;
    }
temp1 = 0xC4;               // Display starting address of second line 5th pos
        lcd_com();
        ptr = disp1;                    // pointing data
        while(*ptr!='\0')
    {
        temp1 = *ptr;
        lcd_data();                             // function to write data  on LCD
                ptr ++;
    }

        for( i = 0 ; i < 300 ; i++ )            // delay of around a sec.
        for( j = 0 ; j < 10000 ; j++ );
        clr_disp();                             //clear display
        delay(3200);                            //delay about 1ms


        temp1 = 0x82;           //Display starting address of first line 3rd pos
        lcd_com();
```

```
        ptr = disp2;
    while(*ptr!='\0')
    {
        temp1 = *ptr;
        lcd_data();                     // function to write data  on LCD
         ptr ++;
    }
        temp1 = 0xC2;          // Display starting address of second line 1st pos
        lcd_com();

        ptr = disp3;                              // pointing data
        while(*ptr!='\0')
    {
        temp1 = *ptr;
        lcd_data();                    // function to write data  on LCD
        ptr ++;
    }



        for( i = 0 ; i < 300 ; i++ )            // delay of around sec.
        for( j = 0 ; j < 10000 ; j++ );
```

```
  clr_disp();                              //clear display

      delay(3200);                         //1ms delay


 temp1 = 0x81;         //Display starting address    of first line 2nd pos

 lcd_com();


 ptr = disp4;

 while(*ptr!='\0')

{
 temp1 = *ptr;

 lcd_data();                  // function to write data  on LCD

  ptr ++;

}


 for( i = 0 ; i < 10 ; i++ )               // delay of 1ms.

   for( j = 0 ; j < 3000 ; j++ );


  while(1)

  {

 AD0CR = 0x01200010;  //CONTROL register for ADC

 while(((temp_adc = AD0GDR) &0x80000000)  == 0x00000000);
                                           //to check the DONE bit


 adc_value = AD0GDR;                       //reading the ADC value
```

```
adc_value >>=6;                  // shift data from zero location

adc_value &= 0x000003ff;         //mask 12 bit data only

        if (pwr_on_flag==0xFF)

        {
                pwr_on_flag=0x00;

                for(i=0;i<10;i++)

                temp_arry[i]=adc_value;
                                //at 1st time add  same value for 10 times

        }

        else

        {
        for(i=9;i>0;i--)
        temp_arry[i]=temp_arry[i-1];
                                // add read data to 1st position of temp_arry

        }


        temp_arry[i]= adc_value;

        sum=0;

        for(i=0;i<10;i++)

        sum=sum+temp_arry[i];           //summing the read values

        sum=sum/10;                     //taking average of 10 value



        ana_output = ((float)sum * (float)vol)/(float)fullscale;
                                        //calculating analog voltage

        adc_out         = ana_output;
```

```
                        adc_out = (adc_out*11.2);
```
//in circuit we use the resistor voltage divider circuit , so we need to calculate actual voltage

```
                        adc_out = (adc_out/10.0);
```
// these steps find the real analog voltage corresponding to temperature.

```
                        adc_out = (adc_out-2.7315);

                        temp_out = ( adc_out*100);
```
//value corresponding to temperature

```
                sprintf(var,"%4.2f",temp_out);
```
// converting int data into ascci value

```
                        temp1 = 0x89;       //Display starting addressof first line 10th pos

                        lcd_com();

                        delay(3200);

                        ptr = var;

                        while(*ptr!='\0')

                            {
                            temp1 = *ptr;         //write ambient temperature on lcd

                            lcd_data();

                                ptr ++;
                            }
                            for ( i = 0 ; i < 300 ; i++ )       // delay of around sec.
                    for( j = 0 ; j < 10000 ; j++ );

                }
}
```
//end of main ()

// lcd initialisation routine.

```
        void lcd_init(void)
```

```
{
        temp = 0x30;                    //command to test LCD voltage levels
        wr_cn();
        delay(3200);

        temp = 0x30;                    //command to test LCD voltage levels
        wr_cn();
        delay(3200);

        temp = 0x30;                    //command to test LCD voltage levels
        wr_cn();
        delay(3200);

        temp = 0x20;            // change to 4 bit mode from default 8 bit mode
        wr_cn();
        delay(3200);
        temp1 = 0x28; // load command for lcd function setting with lcd in 4 bit mode,
        lcd_com();          // 2 line and 5x7 matrix display
        delay(3200);
        temp1 = 0x0C;    // load a command for display on, cursor on and blinking off
        lcd_com();
```

```
        delay(800);

        temp1 = 0x06;  // command for cursor increment after data dump
        lcd_com();
        delay(800);

        temp1 = 0x80;  // set the cursor to beginning of line 1
        lcd_com();
        delay(800);
}

void lcd_com(void)
{
        temp = temp1 & 0xf0;
   wr_cn();
   temp = temp1 & 0x0f;
   temp = temp << 4;
   wr_cn();
   delay(500);
}
```

// command nibble o/p routine

```
void wr_cn(void)            //write command reg
```

```
{
    IO0CLR  = 0x000000FC;           // clear the port lines.
    IO0SET      = temp;             // Assign the value to the PORT lines
    IO0CLR  = 0x00000004;           // clear bit  RS = 0
    IO0SET      = 0x00000008;       // E=1
    delay(10);
    IO0CLR  = 0x00000008;
}
```

// data nibble o/p routine

```
void wr_dn(void)                    ////write data reg
{
    IO0CLR = 0x000000FC;        // clear the port lines.
    IO0SET = temp;                  // Assign the value to the PORT lines
    IO0SET = 0x00000004;        // set bit  RS = 1
    IO0SET = 0x00000008;        // E=1
    delay(10);
    IO0CLR = 0x00000008;
}
```

// data o/p routine which also outputs high nibble first and lower nibble next

```
void lcd_data(void)
```

```
{
        temp = temp1 & 0xf0;
  temp = temp ;//<< 6;
  wr_dn();
  temp= temp1 & 0x0f;
  temp= temp << 4;
  wr_dn();
  delay(100);
}

void clr_disp(void)
{
  temp1 = 0x01;
  lcd_com();
  delay(500);
}

void delay(unsigned int r1)
{
        for(r=0;r<r1;r++);
}
```

2. To design ARM cortex based automatic number plate recognition system

3. To design ARM based power saving system

ADDITIONAL EXERIMENTS:

1. Read the status of eight input bits from the Logic Controller Interface and display FF if it is even parity bits otherwise display 00. Also display number of 1's in the input data.

```
model small
.data
        pa      equ 0d800h          ; Port address
        pb      equ 0d801h
        pc      equ 0d802h
        ctrl    equ 0d803h          ; control Register address
.code
start:  mov     ax, @data
        mov     ds, ax              ; Initialization of data segment

        mov     dx, ctrl
        mov     al, 82h             ; move the control word to 'al' register
        out     dx, al              ; move the control word to control register

        mov     dx, pb              ; Get the input data form 'pb'
        in      al, dx              ; Get the input data to AL register
        mov     bl, 00h
        mov     cx, 08              ; number of rotations
up:     rcl     al,1
        jnc     down                        ; after each rotation check for the carry flag
        inc     bl                  ; If there is a carry, increment the 'BL' register
down:   loop    up                  ; Repeat rotation for '08' times

        test    bl,01h              ; perform 'AND' operation to check for even or odd
parity
        jnz     oddp                ; If the result of the 'AND' is not zero, it is odd
parity

        mov     al,0ffh             ; If even parity display 0ffh
        jmp     next
oddp:   mov     al,00h              ; If odd parity display 00h
next:   mov     dx,pa
        out     dx,al               ; put the result to the ports

        call    delay
```

```
        mov     al, bl
        mov     dx, pa
        out     dx, al              ; Out the number of 01s present in the I/P bits
        int     3

delay   proc                        ; Delay procedure
        push    cx
        push    bx

        mov     cx, 0ffffh
d2:     mov     bx, 8fffh
d1:     dec     bx
        jnz     d1
        loop    d2

        pop     bx
        pop     cx
        ret
delay   endp
end     start
```

Conclusion:

The program reads port B of 82C55A which is an input port. If input contains an odd number of 1's (that is the number of LED's at logic 1) then the output will be 00 at port A, which is an output port, indicating input is odd parity and after some delay the number of 1's present in input will be displayed through port A on the output.

Similarly If input contains an even number of 1's (that is the number of LED's at logic 1) then the output will be FF at port A, which is an output port, indicating input is even parity and after some delay the number of 1's present in input will be displayed through port A on the output.

2. Write two ALP modules stored in two different files; one module is to read a character from the keyboard and the other one is to display a character. Use the above two modules to read a string of characters from the keyboard terminated by the carriage return and print the string on the display in the next line.

```
.model small
.data
        String  db 30 dup (?)
.code
include c:\masm\read.mac
include c:\masm\write.mac

start:  mov     ax, @data
        mov     ds, ax                  ; Initialization of data segment

        mov     si, 00h

again:  read                            ; CALL MACRO READ
        cmp     al, 0dh                 ; compare the data in 'AL' reg with enter Key
        je      down
        mov     string[si], al          ; Move  the data in 'AL' reg to destination.
        inc     si
        jmp     again

down:   mov     cx, si
        mov     si, 00h

        write   0dh                     ; '13','10' , To go to next line
        write   0ah

back:   write   string[si]              ; Call write macro to write the data
        inc     si
        loop    back                    ; Repeat the writing
        int     3                       ; Termination of the program
        end     start
```

read.mac

```
read    macro
        mov   ah, 01h            ; Dos command  to read a data from keyboard
        int   21h
        endm
```

write.mac

```
write   macro x
        mov   dl, x
        mov   ah, 02h            ; Dos command  to write a data to the O/P screen
        int   21h
        endm
```

Conclusion:

This program reads the character entered through the Key board and stores in the consecutive specified memory locations. This process repeats till the ENTER Key (carriage return) is pressed. Once the ENTER key (carriage return) is pressed the character stored in the consecutive memory locations will be displayed on the next line.

3. Scan a 8x3 keypad for key closure and to store the code of the key pressed in a memory location and display on screen. Also display row and column numbers of the key pressed.

```
.model small
.stack  100
.data
        pa      equ 0d800h
        pb      equ 0d801h
        pc      equ 0d802h
        ctrl    equ 0d803h
        ASCIICODE       db "0123456789.+-*/%ack=MRmn"       ; look up table
        str     db 13,10,"press any key on the matrix keyboard$"
        str1    db 13,10,"Press y to repeat and any key to exit $"
        msg     db 13, 10,"the code of the key pressed is :"
        key     db ?
        msg1    db 13,10,"the row is "
        row     db ?
        msg2    db 13,10,"the column is "
        col     db ?,13,10,'$'
.code
disp    macro x                         ; Display  a string
        mov     dx, offset x
        mov     ah, 09
        int     21h
        endm                            ; End of a macro

start:  mov     ax,@data
        mov     ds,ax

        mov     al,90h                  ; Port 'A' is input port
        mov     dx,ctrl
        out     dx,al

again1: disp    str
        mov     si,0h

again:  call    scan
        mov     al,bh                   ; Row number
        add     al,31h
        mov     row,al

        mov     al,ah                   ; Column number
```

```
        add     al,31h
        mov     col,al
        cmp     si,00
        je      again
        mov     cl,03
        rol     bh,cl
        mov     cl,bh
        mov     al,ah
        lea     bx,ASCIICODE            ; Address of the look up table
        add     bl,cl
        xlat                            ; Translate a byte in AL
        mov     key,al

        disp    msg
        disp    str1
        mov     ah, 01                  ; Read a string
        int     21h
        cmp     al,'y'
        je      again1
        int     3
scan    proc
        mov     cx,03
        mov     bh,0
        mov     al,80h

nxtrow: rol     al,1
        mov     bl,al
        mov     dx,pc
        out     dx,al
        mov     dx,pa
        in      al,dx
        cmp     al,0
        jne     keyid

        mov     al,bl
        inc     bh
        loop    nxtrow
        ret
keyid:  mov     si,1
        mov     cx,8
        mov     ah,0

agn:    ror     al,1
```

```
        jc      skip            ; check for the carry
        inc     ah
        loop    agn
skip:   ret                     ; Return to main program
scan    endp
        end     start
```

Conclusion:

This program reads the data from the 8*3 key interface board. It will display its value on the screen. It will also display the row number and column number of the key pressed.

4. Program to create a file (input file) and to delete an existing file.

```
.model small
.data
        string  db "Enter the file name for the file to be created",13,10,'$'
        msg1    db 13,10,"The file cannot be created",13,10,'$'
        msg2    db 13,10,"File created successfully",13,10,'$'
        str1    db 40 dup(0)
        string1 db "Enter the file name to be deleted",13,10,'$'
        msg3    db 13,10,"The file cannot be deleted",13,10,'$'
        msg4    db 13,10,"File deleted successfully",13,10,'$'
        str2    db 40 dup(0)
.code
disp    macro x                          ; Display macro
        lea     dx, x
        mov     ah, 09h
        int     21h
        endm

start:  mov     ax,@data
        mov     ds,ax
        disp    string                   ; Display String
        mov     bx,00h

up:     mov     ah,01h                   ; Read the character from the
keyboard
        int     21h
        cmp     al,0dh
        je      exit
        mov     str1[bx],al
        inc     bx
        jmp     up

exit:   mov     str1[bx],'$'
        mov     ah,3ch                   ; Create or truncate file
        mov     cx,00h                   ;  File Attributes
        mov     dx,offset str1
        int     21h

        jc      down
        disp    msg2
        jmp     down1
```

```
down:   disp    msg1

down1:  disp    string1
        mov     bx,00h
up1:    mov     ah,01h
        int     21h
        cmp     al,0dh
        je      exit1
        mov     str2[bx],al
        inc     bx
        jmp     up1

exit1:  mov     str2[bx],'$'
        mov     ah,41h              ; delete file .
        mov     dx,offset str2
        int     21h
        jc      down2               ; CF set on error, AX = error code.

        disp    msg4                ; if successful, CF will be clear,
                                    ; and the value of AX is cleared
        jmp     down3
down2:  disp    msg3
down3:  int     3
end     start
```

5. To interface relay with ARM processor-- ARM7TDMI/LPC2148. Write a program for the on and off of a relay.

```
#include <LPC214x.H>                    /* LPC214x definitions */

#define RELAY1                  (1 << 4) // P0.4
#define KEY_CTRL_PIN      IO1PIN
#define R1ON                              (1 << 16)         //KEY1                    P1.16
#define R1OFF                             (1 << 20)         //KEY5                    P1.20

 ///////////////////// MAIN /////////////////////////////////////
int main (void)
{

  IO1DIR =        ~ (R1ON | R1OFF);

  IO0DIR = (RELAY1);

  while (1)
  {

                  if (!(KEY_CTRL_PIN & R1ON))  //R1ON key pressed

                   {
                     IO0SET = 0X00000010 ;

                   }

                  if (!(KEY_CTRL_PIN & R1OFF))//R1OFF key pressed

                   {
                  IO0CLR = 0X00000010 ;

                   }
          }
}
```

6. To interface DAC with ARM processor-- ARM7TDMI/LPC2148. Write a program to convert digital value to an analog value using DAC.

```
#include <LPC214x.H>                    /* LPC214x definitions */

//////////   Init DAC  ////////////////
Init_DAC()
{
  // Convert Port pin 0.25 to function as DAC
  PINSEL1 = 0X00080000;
  DACR = 0;
}


//////////   Write DAC   ////////////////
Write_DAC(unsigned int dacval)
{
        DACR = dacval << 6;
}
void delay(unsigned int count)
{
  int j=0,i=0;

  for(j=0;j<count;j++)
  {
    for(i=0;i<120;i++);
  }
}

//////////   MAIN ////////////////
int main (void)
{

   Init_DAC();

  while(1)
  {
        Write_DAC(00);
        delay(100);                     // change this value to change Frequency
        Write_DAC(1023);     // change this value to change Amplitude
        delay(100);                // change this value to change Frequency
  }
}
```

References:

1. The Intel Microprocessors: Eighth Edition: Bary B. Brey.

2. Microprocessors and Interfacing: Second Edition: D V Hall.

3. Advanced Microprocessors and Peripherals: A K Ray.

4. Muhammad Ali Mazidi, Janice Gillispie, Mazidi, Danny Causey, The x86 PC Assembly Language Design and Interfacing, 5th Edition, Pearson, 2013.

5. ARM system developers guide, Andrew N Sloss, Dominic Symes and Chris Wright, Elsevier,Morgan Kaufman publishers, 2008.

ANNEXURES:

Instruction Set:

Instructions	Operands	Description
MOV	**REG, memory** **memory, REG** **REG, REG** **memory, immediate** **REG, immediate** **SREG, memory** **memory, SREG** **REG, SREG** **SREG, REG**	**Copy operand2 to operand1.** The MOV instruction <u>cannot</u>: • Set the value of the CS and IP registers. • Copy value of one segment register to another segment register (should copy to general register first). • Copy immediate value to segment register (should copy to general register first). **Algorithm: operand1 = operand2** Ex: **Mov AX,BX ;Copy contents of BX to AX** **Mov si,00h ;load Si with 00h**
MUL	**REG** **Memory**	**Unsigned multiply.** Multiply the contents of REG/Memory with contents of AL register. **Algorithm:** When operand is a **byte**: AX = AL * operand. When operand is a **word**: (DX: AX) = AX * operand.
CMP	**REG, memory** **memory, REG** **REG, REG** **memory, immediate** **REG, immediate**	**Compare.** **Algorithm:*operand1 - operand2*** Result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.
JMP	**Label**	**Unconditional Jump.** Transfers control to another part of the program. *4-byte address* may be entered in this form: 1234h: 5678h, first value is a segment second value is an offset. **Algorithm:** always jump
JA	**Label**	**Jump If Above.** Short Jump if first operand is Above second operand (as set by CMP instruction). Unsigned. **Algorithm:**if (CF = 0) and (ZF = 0) then jump

JAE	**Label**	**Jump If Above Or Equal** Short Jump if first operand is Above or Equal to second operand (as set by CMP instruction). Unsigned. **Algorithm:** if CF = 0 then jump
JB	**Label**	**Jump If Below.** Short Jump if first operand is Below second operand (as set by CMP instruction). Unsigned. **Algorithm:** if CF = 1 then jump
JBE	**Label**	**Jump If Below Or Equal** Short Jump if first operand is Below second operand (as set by CMP instruction). Unsigned. **Algorithm**: if CF = 1 then jump
JC	**Label**	**Jump If Carry** Short Jump if Carry flag is set to 1. **Algorithm:** if CF = 1 then jump
JE	**Label**	**Jump If Equal.** Short Jump if first operand is Equal to second operand (as set by CMP instruction). Signed/Unsigned. **Algorithm:** if ZF = 1 then jump
JG	**Label**	**Jump If Greater** Short Jump if first operand is Greater then second operand (as set by CMP instruction). Signed. **Algorithm:** if (ZF = 0) and (SF = OF) then jump

JGE	**Label**	**Jump If Greater Or Equal.** Short Jump if first operand is Greater or Equal to second operand (as set by CMP instruction). Signed. **Algorithm:** if SF = OF then jump
JL	**Label**	**Jump If Less than.** Short Jump if first operand is Less then second operand (as set by CMP instruction). Signed. **Algorithm:** if SF <> OF then jump
JLE	**Label**	**Jump If Less Or Equal.** Short Jump if first operand is Less or Equal to second operand (as set by CMP instruction). Signed. **Algorithm:** if SF <> OF or ZF = 1 then jump
JNZ	**Label**	**Jump If Non Zero.** Short Jump if Not Zero (not equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. **Algorithm:** if ZF = 0 then jump
JZ	**Label**	**Jump If Zero.** Short Jump if Zero (equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. **Algorithm:** if ZF = 1 then jump
LEA	**REG, memory**	**Load Effective Address.** **Algorithm:** • REG = address of memory (offset)
LOOP	**Label**	**Decrease CX, jump to label if CX not zero.** **Algorithm:** • CX = CX - 1 • if CX <> 0 then

		○ jump else ○ no jump, continue
ADD	**REG, memory memory, REG REG, REG memory, immediate REG, immediate**	**Add.** **Algorithm:** operand1 = operand1 + operand2
AND	**REG, memory memory, REG REG, REG memory, immediate REG, immediate**	Logical AND between all bits of two operands. Result is stored in operand1. These rules apply: **1 AND 1 = 1; 1 AND 0 = 0** **0 AND 1 = 0; 0 AND 0 = 0**
OR	**REG, memory memory, REG REG, REG memory, immediate REG, immediate**	Logical OR between all bits of two operands. Result is stored in first operand. These rules apply: **1 OR 1 = 1; 1 OR 0 = 1** **0 OR 1 = 1; 0 OR 0 = 0**
SUB	**REG, memory memory, REG REG, REG memory, immediate REG, immediate**	**Subtract.** **Algorithm:** operand1 = operand1 - operand2

DAA	**No Operands**	**Decimal adjust After Addition.**
		Corrects the result of addition of two packed BCD values.
		Algorithm:
		if low nibble of AL> 9 or AF = 1 then:
		• AL = AL + 6
		• AF = 1
		if AL> 9Fh or CF = 1 then:
		• AL = AL + 60h
		• CF = 1

DAS	**No Operands**	**Decimal adjust After Subtraction**.
		Corrects the result of subtraction of two packed BCD values.
		
		Algorithm:
		if low nibble of AL> 9 or AF = 1 then:
		
		• AL = AL - 6
		• AF = 1
		
		if AL> 9Fh or CF = 1 then:
		
		• AL = AL - 60h
		• CF = 1

INC	**REG** **memory**	**Increment.** **Algorithm:**operand = operand + 1
DEC	**REG** **Memory**	**Decrement.** **Algorithm:**operand = operand – 1
DIV	**REG** **Memory**	**Unsigned divide.** **Algorithm:** when operand is a **byte**: AL = AX / operand AH = remainder (modulus) when operand is a **word**: AX = (DX AX) / operand DX = remainder (modulus)
SHL	**memory, immediate** **REG, immediate** **memory, CL** **REG, CL**	**Shift Left.** Shift operand1 Left. The number of shifts is set by operand2. **Algorithm:** • Shift all bits left, the bit that goes off is set to CF. • Zero bit is inserted to the right-most position.
SHR	**memory, immediate** **REG, immediate** **memory, CL** **REG, CL**	**Shift Right.** Shift operand1 Right. The number of shifts is set by operand2. **Algorithm:** • Shift all bits right, the bit that goes off is set to CF. • Zero bit is inserted to the left-most position.

ROL	**memory, immediate** **REG, immediate** **memory, CL** **REG, CL**	**Rotate Left.** Rotate operand1 left. The number of rotates is set by operand2. **Algorithm:** Shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position.
ROR	**memory, immediate** **REG, immediate** **memory, CL** **REG, CL**	**Rotate Right.** Rotate operand1 right. The number of rotates is set by operand2. **Algorithm:** Shift all bits right, the bit that goes off is set to CF and the same bit is inserted to the left-most position.
RCL	***memory, immediate*** ***REG, immediate*** ***memory, CL*** ***REG, CL***	Rotate operand1 left through Carry Flag. The number of rotates is set by operand2. **Algorithm:** Shift all bits left, the bit that goes off is set to CF and previous value of CF is inserted to the right-most position. **Example:** STC ; set carry (CF=1). MOV AL, 1Ch ; AL = 00011100b RCL AL, 1 ; AL = 00111001b, CF=0. RET C O r r OF=0 if first operand keeps original sign.
CALL	**procedure name** **label**	Transfers control to procedure, return address is (IP)pushed to stack.
RET	**No operands** **Or even immediate date**	Return from near procedure. **Algorithm:** • Pop from stack: ○ IP if <u>immediate</u> operand is present: SP = SP + operand
IN	**AL, im.byte** **AL, DX** **AX, im.byte** **AX, DX**	Input from port into **AL** or **AX**. Second operand is a port number. If required to access port number over 255 - **DX** register should be used.
OUT	**AL, im.byte** **AL, DX** **AX, DX**	Output from **AL** or **AX** to port. First operand is a port number. If required to access port number over 255 - **DX** register should be used.

POP	**REG** **SREG** **memory**	Get 16 bit value from the stack. Algorithm: Operand = SS : [SP](top of stack) SP = Sp + 2.
PUSH	**REG** **SREG** **memory**	Store 16 bit value in the stack. **Algorithm:** • SP = SP - 2 • SS:[SP] (top of the stack) = operand
XOR	**REG, memory** **memory, REG** **REG, REG** **memory, immediate** **REG, immediate**	Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand. These rules apply: **1 XOR 1 = 0; 1 XOR 0 = 1** **0 XOR 1 = 1; 0 XOR 0 = 0**
XCHG	**REG, memory** **memory, REG** **REG, REG**	Exchange values of two operands. **Algorithm:**operand1 < - > operand2
XLAT	**No Operands**	Translate byte from table. Copy value of memory byte at DS:[BX + unsigned AL] to AL register. **Algorithm:** AL = DS:[BX + unsigned AL]

AAA	**No Operands**	**ASCII Adjust after Addition.**
		Corrects result in AH and AL after addition when working with BCD values.
		
		Algorithm:
		
		if low nibble of AL> 9 or AF = 1 then:
		
		• AL = AL + 6
		• AH = AH + 1
		• AF = 1
		• CF = 1
		
		else
		
		• AF = 0
		• CF = 0
		
		in both cases:
		clear the high nibble of AL.

		Example: MOV AX, 15 ; AH = 00, AL = 0Fh AAA ; AH = 01, AL = 05

AAS	**No Operands**	**ASCII Adjust after Subtraction.**
		Corrects result in AH and AL after subtraction when working with BCD values.
		
		Algorithm:
		
		if low nibble of AL> 9 or AF = 1 then:
		
		• AL = AL - 6
		• AH = AH - 1
		• AF = 1
		• CF = 1
		
		else
		
		• AF = 0
		• CF = 0
		
		in both cases:
		clear the high nibble of AL.
		
		Example:
		MOV AX, 02FFh ; AH = 02, AL = 0FFh
		AAS ; AH = 01, AL = 09

AAM	**No Operands**	**ASCII Adjust after Multiplication.** Corrects the result of multiplication of two BCD values. **Algorithm:** • AH = AL / 10 • AL = remainder Example: MOV AL, 15 ; AL = 0Fh AAM ; AH = 01, AL = 05

INTERRUPTS:

Interrupt INT 21h:

INT 21h calls DOS functions.

Function 01h- **Read character from standard input**, result is stored in AL. If there is no character in the keyboard buffer, the function waits until any key is pressed.

Invoked by: **AH** = 01h
Returns: **AL** = character entered.

Example:

```
Mov AH, 01h
INT 21h
```

Function 02h- Write a character to standard output.
INT 21h
Invoked by: **DL** = character to write.
AH =02h
After execution AL = DL.

Example:

```
Mov AH, 02h
Mov DL, 'a'     ; Character to be displayed on screen must be stored in DL reg.
INT 21h
```

Function 02h- set cursor position.

INT 10h / AH = 2 - set cursor position.
Input:
DH = row.
DL = column.
BH = page number (0...7).

Function 03h- gets cursor position and size.

INT 10h / AH = 03h -
input:
BH = page number.
return:
DH = row.
DL = column.

CH = cursor start line.
CL = cursor bottom line.

Function 06h – Direct console for input/output. If DL = 0FFH on entry, then this function reads the console. If DL = ASCII character, then this function displays the ASCII character on the console video screen.

Invoked by: Parameters for O/P: **DL** = 0…255
Parameters for I/P: **DL** = 255.

Returns: for O/P: **AL** = **DL**.
For I/P: **ZF** set if no character available &**AL** = 0
ZF clear if character available &**AL** = character.

Example:

```
mov ah, 6
mov dl, 'a'
int 21h      ; output character.

mov ah, 6
mov dl, 255
int 21h      ; get character from keyboard buffer (if any) or set ZF=1.
```

Function 09h - **Write a string to standard output atDS: DX**.
String must be terminated by '**$**'.The string can be of any length and may contain control characters such as carriage return (0DH) and line feed (0AH).

Invoked by: **DS** = string to write.
AH = 09h

Example:

```
Mov AH, 09h
Mov DX, offset str      ; Address of the string to be displayed
INT 21h
```

Function 2Ch- Get system time.

Invoked by: **AH** =2Ch
Return: **CH** = hour. **CL** = minute. **DH** = second. **DL** = 1/100 seconds.

Example:

```
Mov AH, 2ch
INT 21h
```

Function 3Ch - Create or truncate file.

Invoked by: **CX** = file attributes:
mov cx, 0 ; normal - no attributes.
mov cx, 1 ; read-only.
mov cx, 2 ; hidden.
mov cx, 4 ; system
mov cx, 7 ; hidden, system and read-only!
mov cx, 16 ; archive
mov cx, 0BH ; Volume label
mov cx, 10H ; Subdirectory

DS: DX -> filename. ; **AH** =3Ch

Returns:
CF clear if successful, **AX** = file handle.
CF set on error **AX** = error code.

Example:
```
Mov AH, 3ch
Mov CX, 01
Mov DX, offset Filename
INT 21h
```

Function 41h - Delete file (unlink).

Invoked by: **DS: DX** -> ASCIZ filename (no wildcards, but see notes).
AH=41h
Return:
CF clear if successful, **AX** destroyed.
CF set on error **AX** = error code.

Example:
```
Mov AH, 41h
Mov DX, offset Filename
INT 21h
```

Function 4Ch – Terminate a process.

Invoked by: **AH** = 4ch
Return: returns control to the operating system.

Example:
```
Mov AH, 4Ch
```

INT 21h

Interrupt INT 10h:

INT 10h calls the BIOS functions. This interrupt often called the video services interrupt as it directly controls the video display in a system.

Function 02h - Set cursor position.

Invoked by*:* **DH** = row; **DL** = column; **BH** = page number (0...7); **AH**=02h.

Example:

```
MOV AH, 02h
MOV BH, 00
MOV DH, 06
MOV DL, 10
INT 10h
```

Function 03h – Get cursor position.

Invoked by: **BH** = page number. (In general 0)
AH = 03h

Return: **DH** = row number; **DL** = column number; **CH** = cursor start line;
CL = cursor bottom line.

Example:

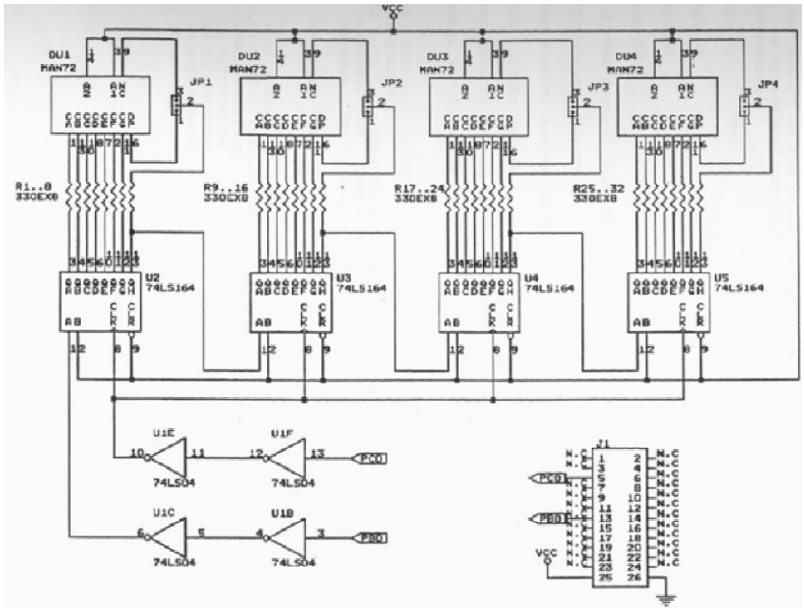
```
Mov BH, 0
Mov AH, 03h
INT 10h
```

Function 06h – Scroll up window

Invoked by: **AL** = number of lines by which to scroll. (00h = clear the entire screen.)
BH = attribute used to write blank lines at bottom of window.
CH, CL = row, column of window's upper left corner.
DH, DL = row, column of window's lower right corner.

Circuit diagrams of interfacing devices

1. Seven Segment Display



The hardware uses four shift register ICs 74164. 74164 is an 8-bit serial in-parallel out shift register with asynchronous reset and two input pins. It requires 8 clock cycles at "CLK" pin to shift the serial data from input to 8 parallel outputs. After 8 shifts, the first serial bit will be in output QH, and only now the data at output is valid. To cascade more 74164 shift register IC need to connect the last output QH to the input of second shift register.

The output is connected to the cathode of the LEDs in the 7 segment display and thus common anode displays are used. The anode is connected to $+V_{cc.}$ The last output of the first sift register is connected to input of the 2nd shift register and the last output o f 2^{nd} shift register to input of 3^{rd} and so on. Thus the shift register are serial in parallel out and they are connected to displays, in such a way that output 0A is connected to display segment 'a' and 0B to 'b' and so on up to 0H; through 330 ohm resistors.

The shifting of data bit takes place for each clock cycle. 7404 IC used provides isolation and the interface board gets 5V through port bit.
Pin 1 is used as data pin and pin 2 is used as other input to Vcc. The clock signal is generated at a port bit which will be connected to the clock of the shift register.

PB0 is used for data bit; and PC0 for clock through which a falling edge has to be sent.

The microprocessor stores the display information in a RAM. Each time a display has to be updated the microprocessor fetches all bytes one by one from RAM and outputs corresponding display codes serially that is bit by bit to display. Hexadecimal code is stores in the RAM. The code conversion from hexa to 7 segment is done just before the display is updated.

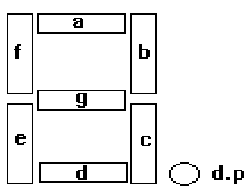
The 7 segment display is used as a numerical indicator on many types of test equipment. It is an assembly of light emitting diodeswhich can be powered individually. There are two important types of 7-segment LED display.

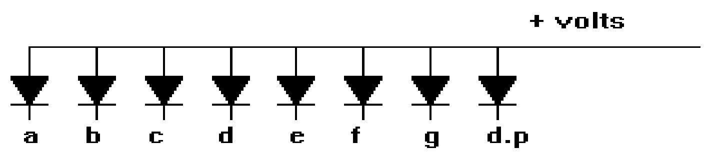
In a **common cathode** display, the cathodes of all the LEDs are joined together and the individual segments are illuminated by HIGH voltages.

In a **common anode** display, the anodes of all the LEDs are joined together and the individual segments are illuminated by connecting to a LOW voltage.

Display code

Since the outputs of shift registers are connected to cathode sides of displays, low input must be given to segments for making them glow and high inputs for making them blank. Each display has 8 segments (a, b, c, d, e, f, g, h) as shown. For displaying any character the corresponding segment must be given low inputs.





The one shown above is a common anode display since all anodes are joined together and go to the positive supply. The cathodes are connected individually to zero volts. Resistors must be placed in series with each diode to limit the current through each diode to a safe value. The **d.p** represents a decimal point.

The following table shows how to form characters: '0' means that pin is connected to ground. '1' means that pin is connected to Vcc.

	d.p	g	f	e	d	c	b	a	Hex. value
0	1	1	0	0	0	0	0	0	C0
1	1	1	1	1	1	0	0	1	F9
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99
5	1	0	0	1	0	0	1	0	92
6	1	0	0	0	0	0	1	0	82
7	1	1	1	1	1	0	0	0	F8
8	1	0	0	0	0	0	0	0	80
9	1	0	0	1	1	0	0	0	98

F	1	0	0	0	1	1	1	0	8e
I	1	1	1	1	1	0	0	1	F9
R	1	0	0	0	1	0	0	0	88
E	1	0	0	0	0	1	1	0	86

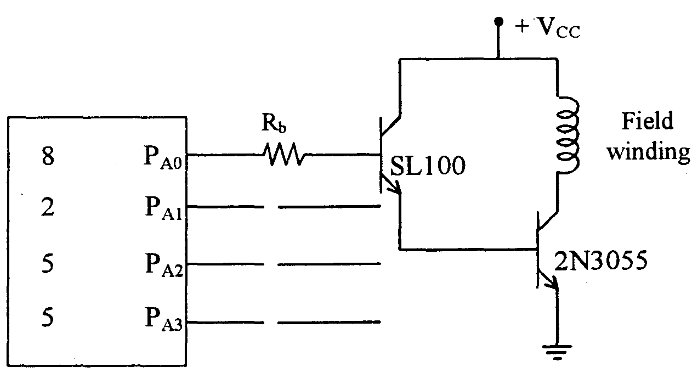
2. Stepper Motor:

A stepper motor is a widely used device that translates electrical pulses into mechanical movement. In applications such as disk drives, dot matrix printers, and robotics, the stepper motor is used for Position control.

Every stepper motor has a permanent magnet rotor (also called the shaft.) surrounded by a stator. The most common stepper motors have four common stator windings that are pairs with a center-taped common. This type of stepper motor is commonly referred to as a four-phase stepper motor.

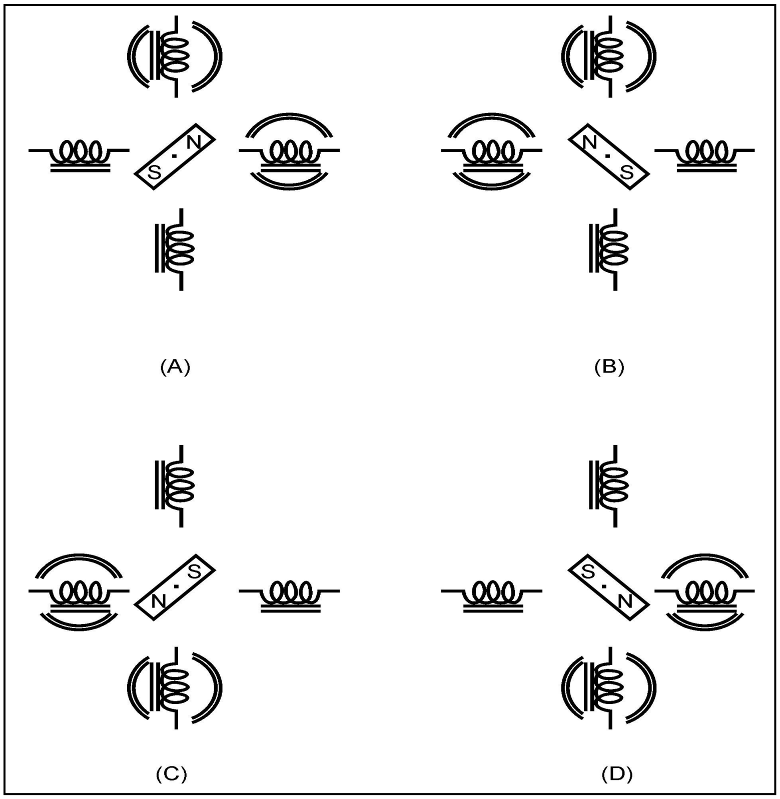
A Stepper motor is stepped from one position to the next by changing the currents through the fields in the motor. Common step sizes for stepper motors range from 0.9 degrees to 30 degrees.

82C55A is used to provide the drive signals that are used to rotate the armature of the motor in either the right-hand or left-hand direction.



The power circuit for one winding of the stepper motor is as shown in figure above. It is connected to the port A (P_{A0}) of 82C55A. Similar circuits are connected to the remaining lower bits of port A (P_{A1}, P_{A2}, P_{A3}). One winding is energized at a time. The coils are turned ON/OFF one at a time successively.

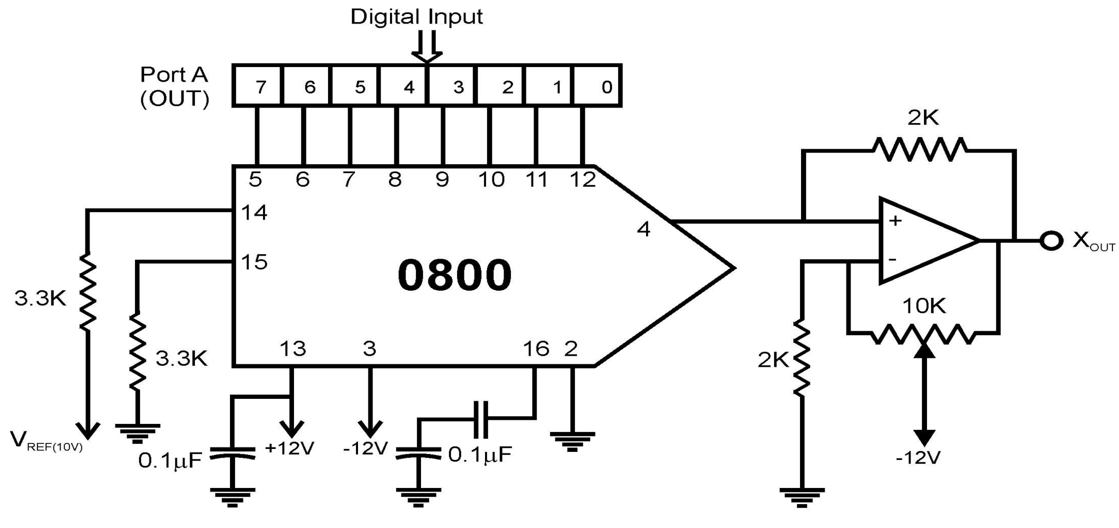
The stepper motor showing full-step operation is shown below.
(A) 45-degrees. (B) 135-degrees (C) 225-degrees (D) 315-degrees.



3. DAC INTERFACE

The pin details of DAC 0800 is given below and schematic diagram of the dual DAC interface is given below.

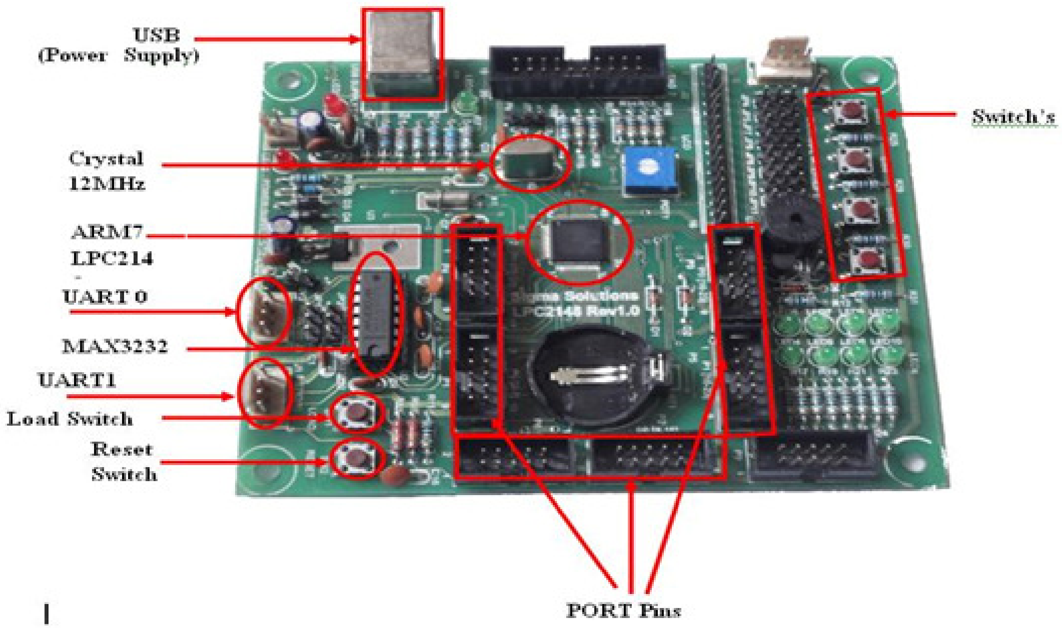
The port A and port B of 82C55A peripheral are used as output ports. The digital inputs to the DACs are porvided through these ports. The analog outputs of the DACs are connected to the inverting inputs of OP-amps 741 which acts as current to voltage converters. The outputs from the OP-amps are connected to points marked X out and Y out at which the waveforms are observed on a CRO. The power supplies of +12 and -12 are regulated for this interface.



Interfacing of DAC 0800

ARM LPC 2148 FEATURES:

- 16-bit/32-bit ARM7TDMI-S Microcontroller.
- 40 kB of on-chip static RAM and 512 kB of on-chip flash memory.
- In-System Programming/In-Application Programming (ISP/IAP) via on-chip boot loader software.
- Embedded ICE RT and Embedded Trace interfaces offer real-time debugging with the on-chip Real Monitor software and high-speed tracing of instruction execution.
- USB 2.0 Full-speed compliant device controller with 2 kB of endpoint RAM.
- Two 10-bit ADCs provide a total of 14 analog inputs
- Single 10-bit DAC provides variable analog output
- Two 32-bit timers/external event counters (with four capture and four compare channels each)
- PWM unit (six outputs)
- Watchdog Timer.
- Low power Real-Time Clock (RTC) with independent power and 32 kHz clock input.
- Multiple serial interfaces including two UARTs, two Fast I^2C-bus (400 kbit/s), SPI and SSP with buffering and variable data length capabilities.
- Vectored Interrupt Controller (VIC) with configurable priorities and vector addresses.
- 60 MHz maximum CPU clock available from programmable on-chip PLL with settling time of 100 us.
- On-chip integrated oscillator operates with an external crystal from 1 MHz to 25 MHz
- Power saving modes include Idle and Power-down.
- Individual enable/disable of peripheral functions as well as peripheral clock scaling for additional power optimization.



LPC 2148 TECHNICAL SPECIFICATIONS:

- Microcontroller: LPC2148 with 512K on chip memory
- Crystal for LPC2148: 12Mhz
- Crystal for RTC: 32.768KHz
- 6 – 10pin Berg headers for external interfacing(GPIOs)
- No separate programmer required (Program with Flash Magic using on-chip boot loader)
- No Separate power adapter required (USB port as power source)
- 20pin(2X10) FRC JTAG connector for Programming and debugging
- 16 Pin Berg Header for LCD Interfacing
- Two RS-232 Interfaces (UART0 and UART1)
- Real-Time Clock with Battery Holder
- 1 Analog Potentiometer connected to ADC
- 4 USER Switches
- 8 USER LEDs
- Reset and Boot loader Switches
- On Board Buzzer Interface

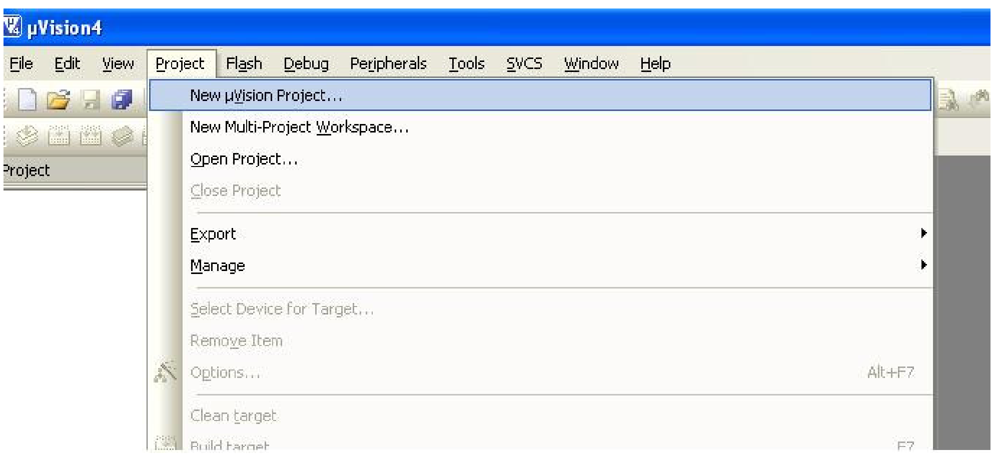
HOW TO USE KEIL µVISION4

For ARM7 (LPC2148) Step By Step

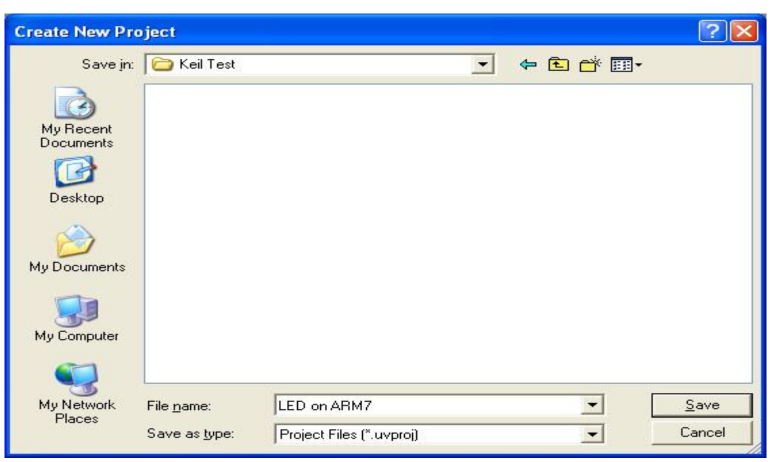
Keil is on the tool which is widely used in Industry, KEIL has tools for ARM, Cortex-M, Cortex-R, 8051, C166, and 251 processor families. In this article we are going to discuss KEIL tools for ARM. The development tools of for ARM include following...

1. µvision IDE v4
2. Compiler for ARM (armcc)
3. MicroLib (C library)
4. Assembler for ARM (armasm)
5. Linker For ARM (armLink)

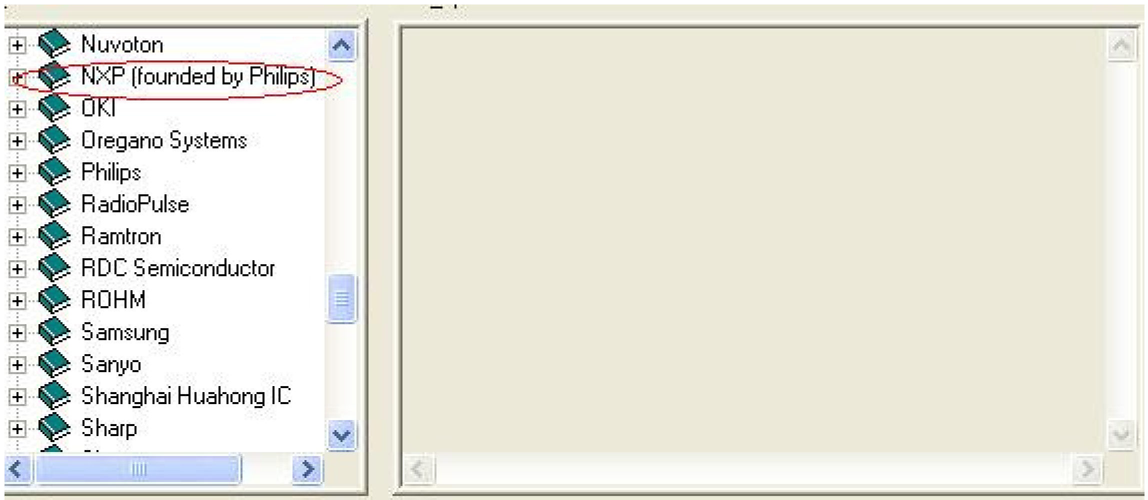
Step1: *Click for KEIL µVISION4 Icon .*
Which appearing after Installing Keil KEIL µVISION4.
This will open uvison IDE.
Step2: *Click on Project Menu, Then* ***New µVison Project****.*



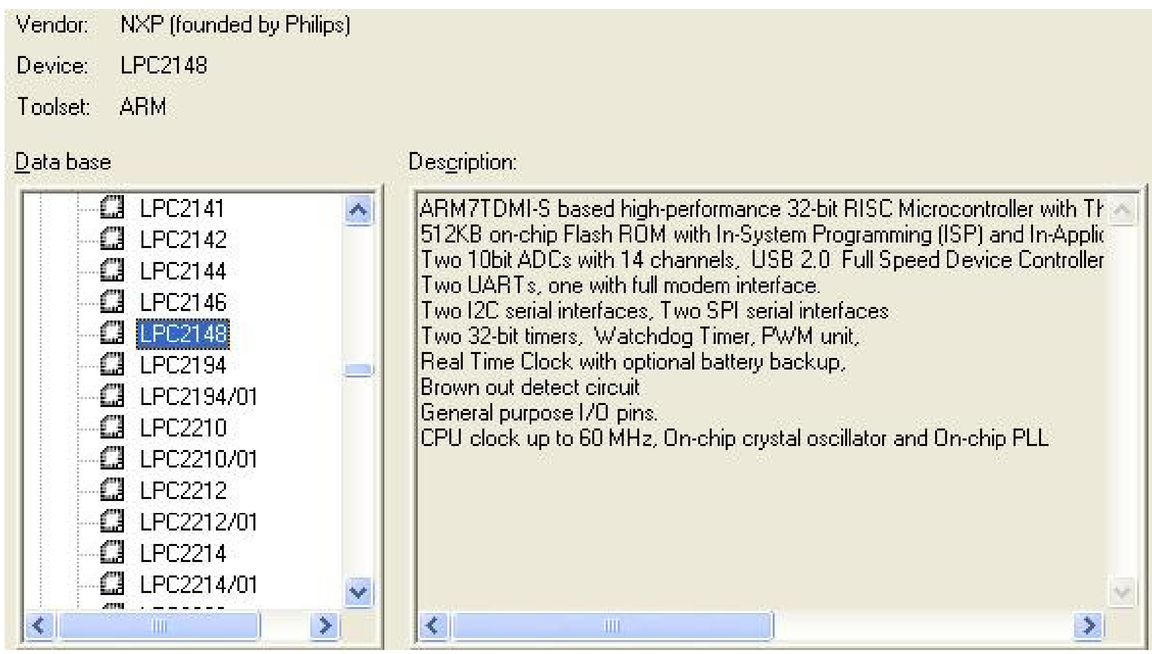
Step3: Create New Project Folder named as "**Keil Test**".



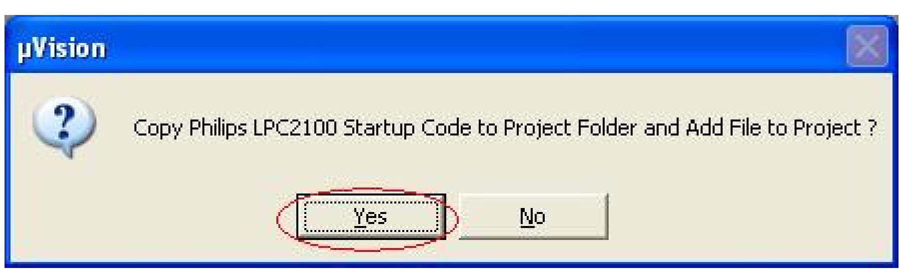
Step 4: Select Target Device



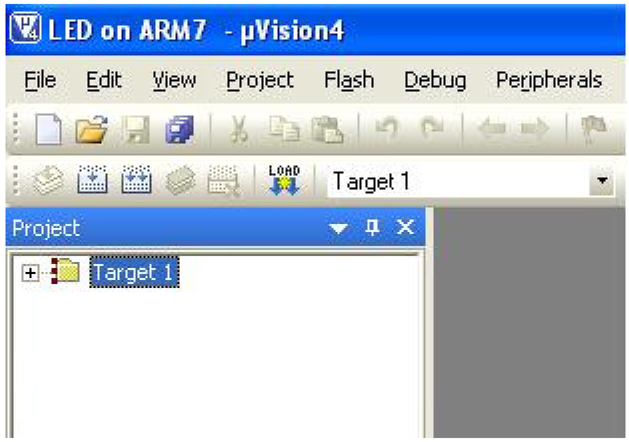
Step 5: Then select specific chip LPC2148.



Step 6: Then select specific chip i.e. LPC2148.



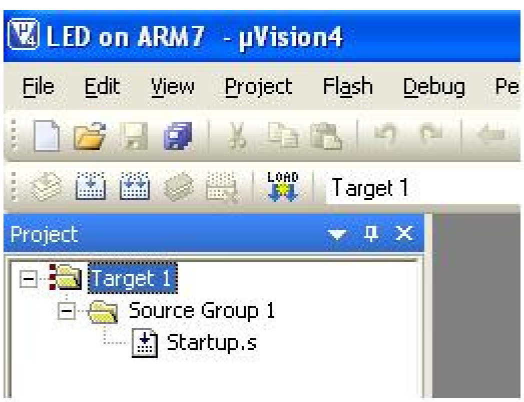
Step 7: Then you will see following window



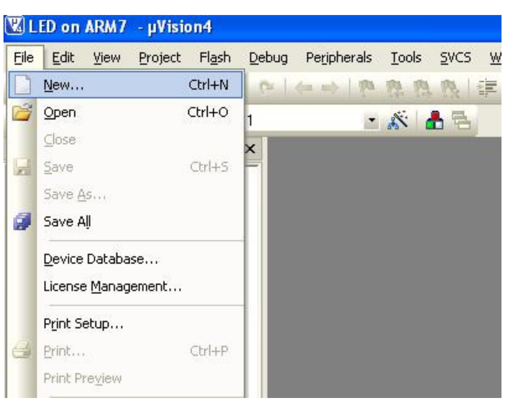
Step 8: Now you see Startup.s is already added which is necessary for running code for Keil.

__Note:__ Code wills Not Run without Startup.s

Startup.s is available in C:\Keil\ARM\Startup\Philips.

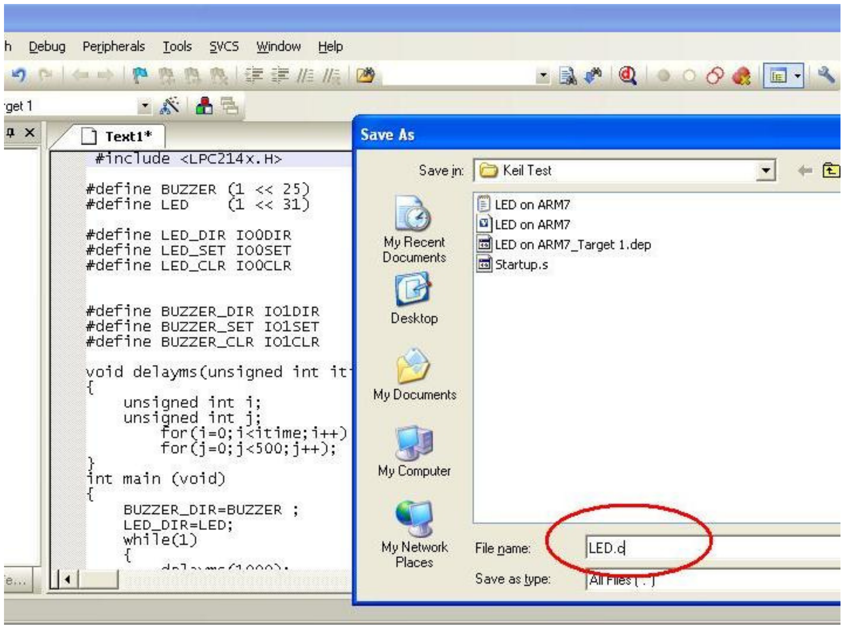


Step 9: Now Click on File Menu and Click on New.

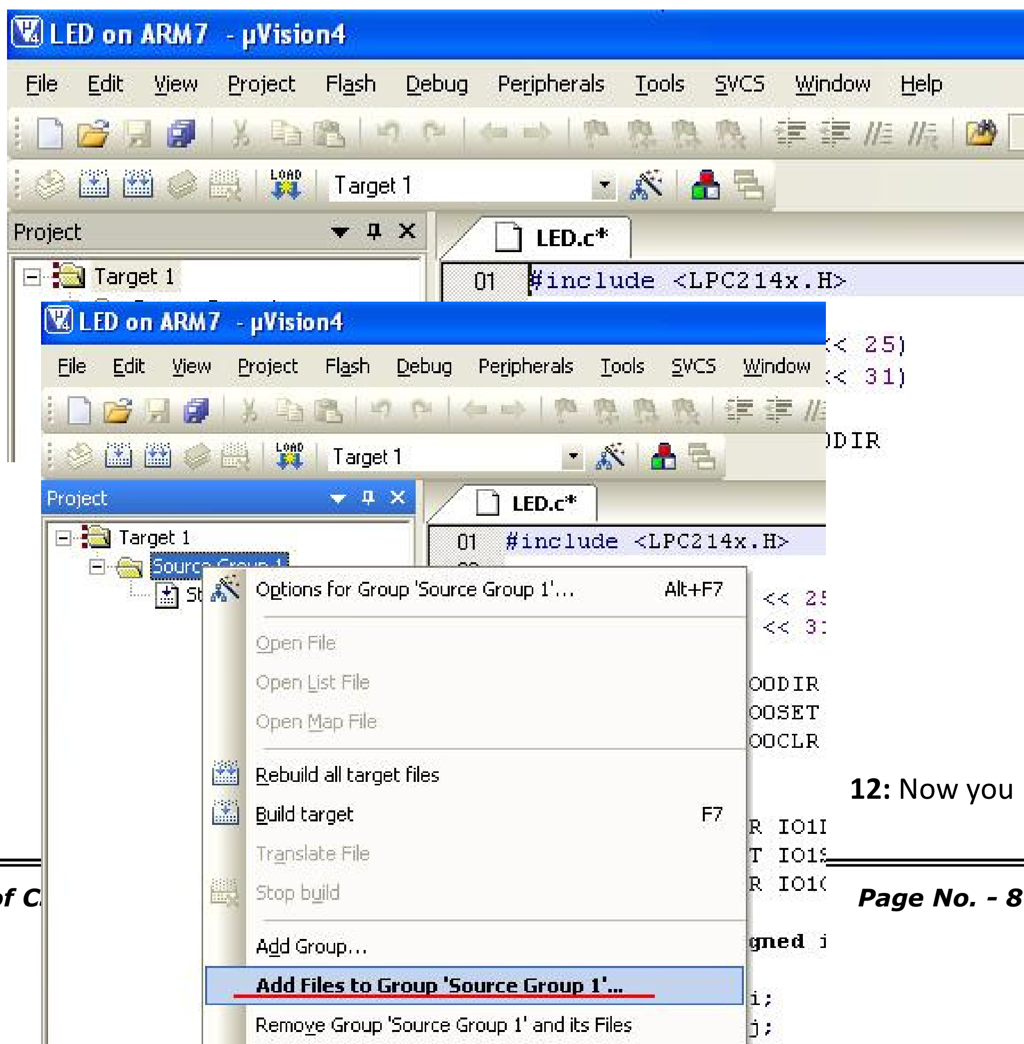


Step 10: Write Code for Blink LED in C OR ASM and FileName.c/ASM Save.

Note: Don't forget to save .c/ASM Extension.



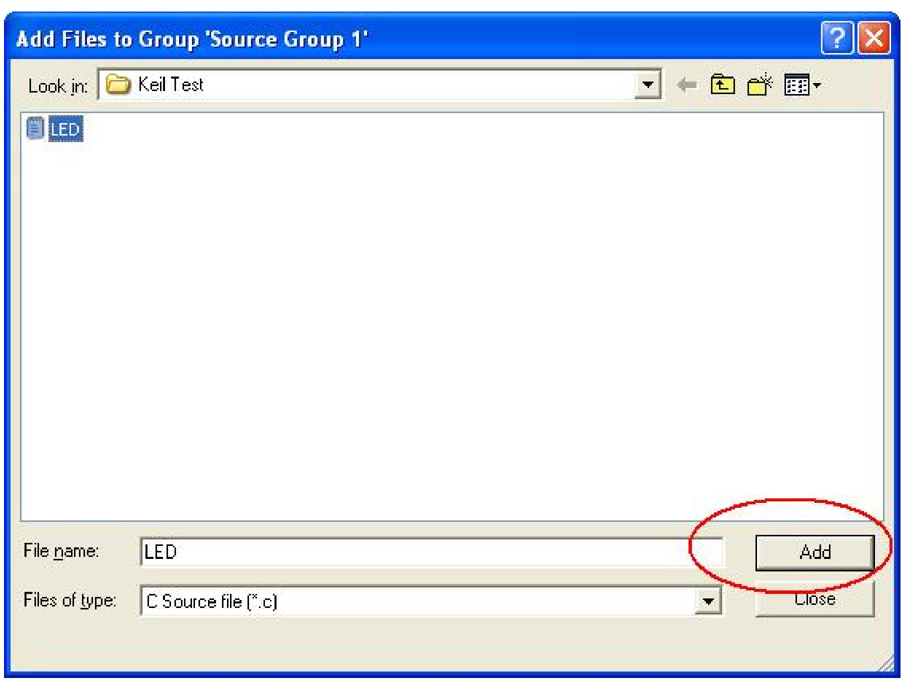
Step 11: Now you Window in C Syntax.



Step **12:** Now you

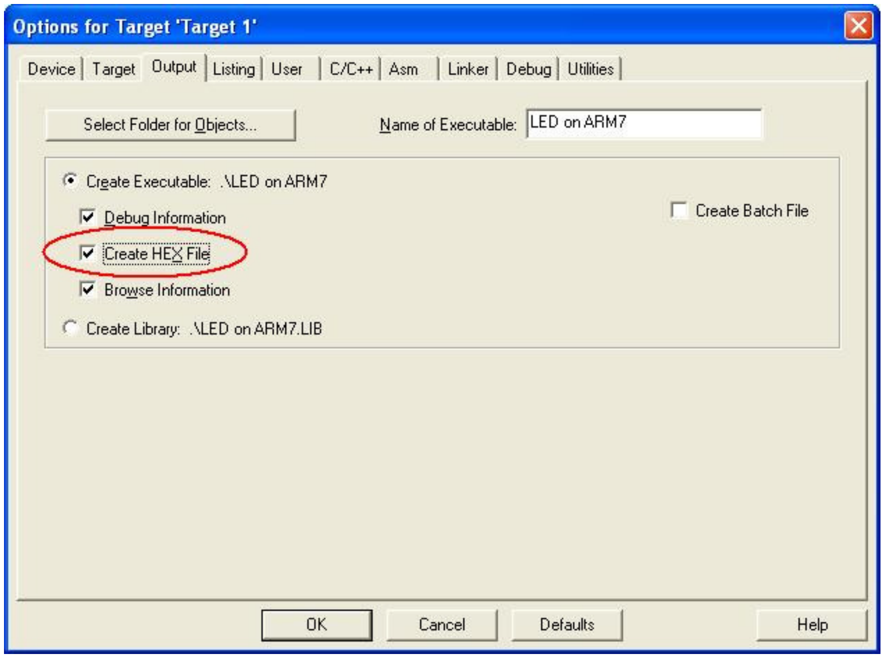
add LED.c file by adding Source Group 1 Add files to Group 'Source Group 1'.

Step 13: Add LED.C file.

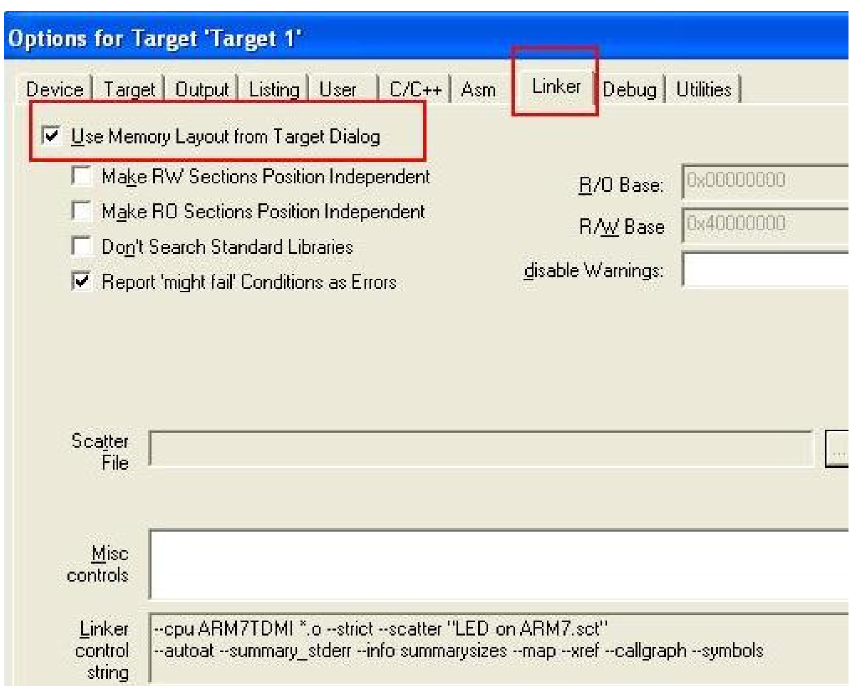


Step 14: Now Click on Options for Target 'Target 1'.

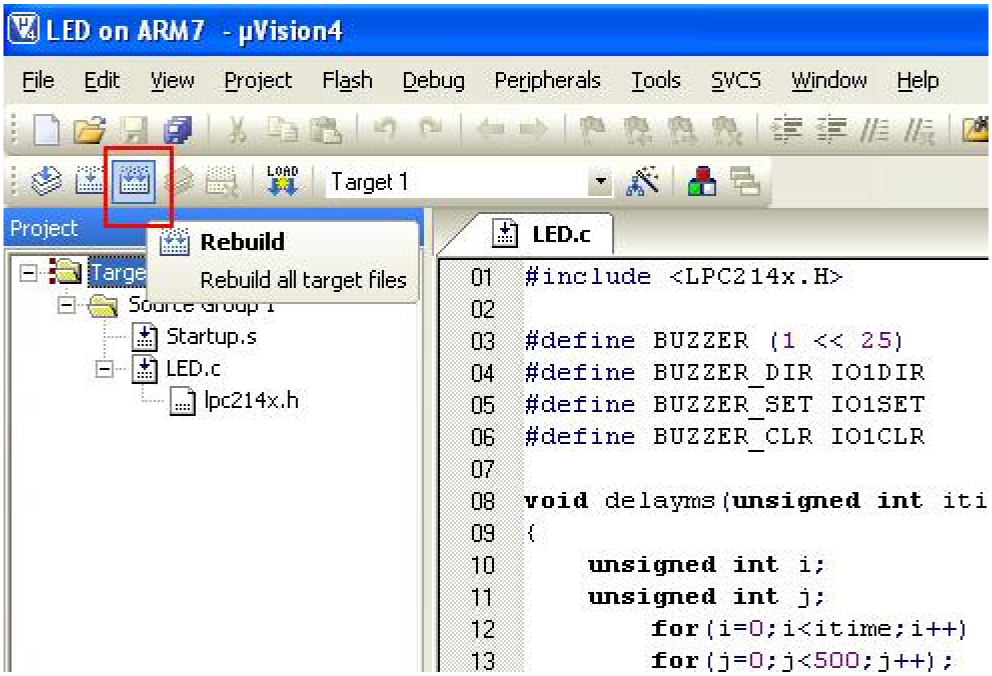
Step 15: Go to Options for Target 'Target 1'. Click on Check Box Create HEX File.



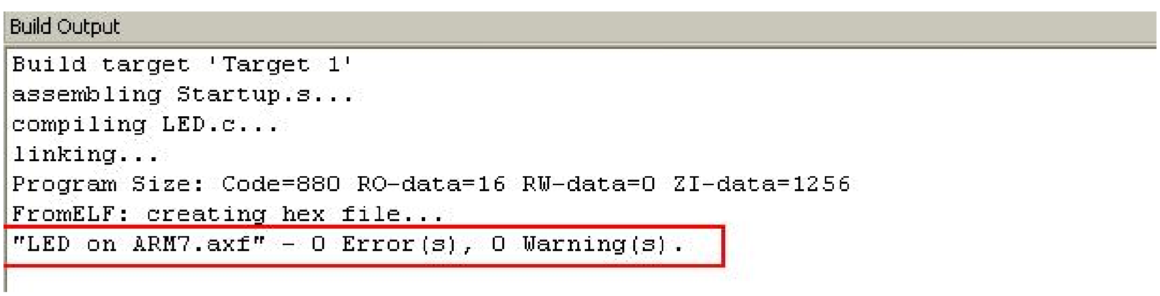
Step 16: Then go to Linker. Click on Use Memory Layout for Target Dialog.



Step 17: Then Click on Rebuild All Target Files



Step 17: Now you see 0 Error(s), 0 Warning (s). Then Hex File will create in Specific Folder. Now to download it for you target hardware.



Viva Questions and Answers

1. What is a Microprocessor?

ANS: Microprocessor is a program-controlled device, which fetches the instructions from memory, decodes and executes the instructions. Most Micro Processor are single- chip devices.

2. What is the difference between 8086 and 8088?

ANS: The BIU in 8088 is 8-bit data bus & 16- bit in 8086.Instruction queue is 4 byte long in 8088and 6 byte in 8086.

3. what are the functional units in 8086?

ANS: 8086 has two independent functional units because of that the processor speed is more. The Bus interface unit and Exectuion unit are the two functional units.

4. What are the flags in 8086?

ANS: In 8086 Carry flag, Parity flag, Auxiliary carry flag, Zero flag, Overflow flag, Trace flag, Interrupt flag, Direction flag, and Sign flag.

5. What is the Maximum clock frequency in 8086?

ANS: 5 Mhz is the Maximum clock frequency in 8086.

6. What are the various segment registers in 8086?

ANS: Code, Data, Stack, Extra Segment registers in 8086.

7. Logic calculations are done in which type of registers?

ANS: Accumulator is the register in which Arithmetic and Logic calculations are done.

8. How 8086 is faster than 8085?

ANS: Because of pipelining concept. 8086 BIU fetches the next instruction when EU busy in executing the anoter instruction.

9. What does EU do?

ANS: Execution Unit receives program instruction codes and data from BIU, executes these instructions and store the result in general registers.

10. Which Segment is used to store interrupt and subroutine return address registers?

ANS: Stack Segment in segment register is used to store interrupt and subroutine return address registers.

11. What does microprocessor speed depend on?

ANS: The processing speed depends on DATA BUS WIDTH.

12. What is the size of data bus and address bus in 8086?

ANS: 8086 has 16-bit data bus and 20- bit address bus.

13. What is the maximun memory addressing capability of 8086?

ANS: The maximum memory capability of 8086 is 1MB.

14. What is flag?

ANS: Flag is a flip-flop used to store the information about the status of a processor and the status of the instruction executed most recently.

15. Which Flags can be set or reset by the programmer and also used to control the operation of the processor?

ANS: Trace Flag, Interrupt Flag, Direction Flag.

16. In how many modes 8086 can be opertaed and how?

ANS: 8086 can be opertaed in 2 modes. They are Minimum mode if MN/MX pin is active high and Maximum mode if MN/MX pin is ground.

17. Whatis the difference between min mode and max mode of 8086?

ANS: Minimum mode operation is the least expensive way to operate the 8086 microprocessor because all the control signals for the memory and I/O are generated by the micro processor. In Maximum mode some of the control signals must be externally generatred. This requires the addition of an external bus controller. It used only when the system contains external coprocessors such as 8087 arithmetic coprocessor.

18. Which bus controller used in maximum mode of 8086?

ANS: 8288 bus controller is used to provide the signals eliminated from the 8086 by the maximum mode operation.

19. What is stack?

ANS: Stack is a portion of RAM used for saving the content of Program Counter and general purpose registers.

20. Which Stack is used in 8086?

ANS:FIFO (First In First Out) stack is used in 8086.In this type of Stack the first stored information is retrieved first.

21. What is the position of the Stack Pointer after the PUSH instruction?

ANS: The address line is 02 less than the earlier value.

22. What is the position of the Stack Pointer after the POP instruction?

ANS: The address line is 02 greater than the earlier value.

23. What is interrupt?

ANS: Interrupt is a signal send by external device to the processor so as to request the processor to perform a particular work.

24. What are the various interrupts in 8086?

ANS: Maskable interrupts, Non-Maskable interrupts.

25. What is meant by Maskable interrupts?

ANS: An interrupt that can be turned off by the programmer is known as Maskable interrupt.

26. What is Non-Maskable interrupts?

ANS: An interrupt which can be never be turned off (ie.disabled) is known as Non-Maskable interrupt.

27. Which interrupts are generally used for critical events?

ANS: Non-Maskable interrupts are used in critical events. Such as Power failure, Emergency, Shut off etc.,

28. Give example for Non-Maskable interrupts?

ANS: Trap is known as Non-Maskable interrupts, which is used in emergency condition.

29. Give examples for Maskable interrupts?

ANS*:* RST 7.5, RST6.5, RST5.5 are Maskable interrupts. When RST5.5 interrupt is received the processor saves the contents of the PC register into stack and branches to 2Ch (hexadecimal) address.

When RST6.5 interrupt is received the processor saves the contents of the PC register into stack and branches to 34h (hexadecimal) address.

When RST7.5 interrupt is received the processor saves the contents of the PC register into stack and branches to 3Ch (hexadecimal) address.

30. What is SIM and RIM instructions?

ANS:SIM is Set Interrupt Mask. Used to mask the hardware interrupts. RIM is Read Interrupt Mask. Used to check whether the interrupt is Masked or not.

31.What is macro?

ANS: Macro is a set of instructions that perform a task and all the isntructions defined in it is inserted in the program at the point of usage.

32. What is the difference between Macro and Procedure?

ANS: A procedure is accessed via a CALL instruction and a macro will inserted in the program at the point of execution.

33. What is meant by LATCH?

ANS: Latch is a D- type flip-flop used as a temporary storage device controlled by a timing signal, which can store 0 or 1. The primary function of a Latch is data storage. It is used in output devices such as LED, to hold the data for display

34. What is a compiler?

ANS: Compiler is used to translate the high-level language program into machine code at a time. It doesn.t require special instruction to store in a memory, it stores automatically. The Execution time is less compared to Interpreter.

35. What is the disadvantage of microprocessor?

ANS: It has limitations on the size of data. Most Microprocessor does not support floating-point operations.

36. What is the 82C55Adevice?

ANS: The 8255A/82C55A interfaces peripheral I/O devices to the microcomputer system bus. It is programmable by the system software. It has a 3-state bi-directional 8-bit buffer which interfaces the 8255A/82C55A to the system data bus.

37. What kind of input/output interface dose a PPI implement?

ANS: It provides a parallel interface, which includes features such as single-bit, 4-bit, and byte-wide input and output ports; level-sensitive inputs; latched outputs; strobed inputs or outputs; and strobed bidirectional input/outputs.

38. How many I/O lines are available on the 82C55A?

ANS: 82C55A has a total of 24 I/O lines.

39. Describes the mode 0, mode 1, and mode 2 operations of the 82C55A?

ANS: MODE 0: Simple I/O mode. In this mode, any of the ports A, B, and C can be programmed as input or output. In this mode, all the bits are out or in.

MODE 1: Ports A and B can be used as input or output ports with handshaking capabilities. Handshaking signals are provided by the bits of port C.

MODE 2: Port A can be used as a bidirectional I/O port with handshaking capabilities whose signals are provided by port C. Port B can be used either in simple I/O mode or handshaking mode 1.

40. What is the mode and I/O configuration for ports A, B, and C of an 82C55A after its control register is loaded with 82H?

ANS: If control register is loaded with 82H, then the port B is configured as an input port, port A and port C are configured as output ports and in mode 0.