**MICROPROCESSORS LAB**

|  |  |
| --- | --- |
| Sub Code : IS 512L | Credits: 0:0:1 |
|  |  |
| Total Hours : 14P | Prerequisite: IS512 |

**Course Objectives**

* Familiarize the architecture of 8086 processor, assembling language programming. Software programming using MASM tool or TASM tool.
* The student can understand the 8086 interfacing with various modules.
* The student can also understand of ARM embedded systems, industrial and real time applications by knowing the concepts of Microprocessor and Microcontrollers.

Part A

1a. Write an ALP to search for a key in an array of bytes/words using Binary Search technique.

1b. Write an ALP to count and display number of 1’s in the input given using switches of a logical controller interface.

2a. Write an ALP to sort the given array of bytes/words using Bubble Sort technique.

2b. Write an ALP to implement i) Ring Counter ii) BCD up-down Counter using logic controller interface.

3a. Write an ALP to check if the input string matches the password stored earlier. After three attempts the system should stop responding to any further transactions.

3b. Write an ALP to rotate the stepper motor clockwise/anticlockwise or in both directions by x degrees [x will be specified at the time of experiment].

4a. Write an ALP to reverse a given string and check if it is palindrome with/without using string instructions.

4b. Write an ALP to generate half rectified/full rectified wave using DAC interface.

5a. Write an ALP to find the factorial of a number using recursive procedure.

5b. Write an ALP to generate sine wave using DAC interface.

6a. Write an ALP to display ‘X’ at the center of the screen with the specific attributes [Attributes will be specified at the time of experiment].

6b. Write an ALP to store row number, column number and scan code of key pressed in Keypad interface.

7a. Write an ALP to generate first n Fibonacci numbers.

7b. Write an ALP to implement a simple calculator for add/subtract operations using Keypad interface.

8a. Write an ALP to display system time.

8b. Write an ALP to display FIRE and HELP alternatively n number of times on seven segment display interface.

9a. Write an ALP to display decimal counter to count from 00-99 at the center of the screen.

9b. Write an ALP to display n character message on seven segment display in a rolling fashion.

10a. Write an ALP to open, read and close a file. Display the contents of the file on screen.

10b. Write an ALP to move the elevator to the first request floor and bring it back to ground floor. The elevator need not respond to the intermediate requests in both directions.

**Part B using ARMSIm1.9.1 Tool**

* 1. Write a ARM program to Add two 16 bit numbers
  2. Write a ARM program to search for a key in an array using any search technique.
  3. Write a ARM program to display character from 0-9
  4. Design and develop basic calculator using ARMSim191 Tool

**Lab Evaluation**

**Continuous Internal Evaluation: 50 Marks**

Midterm test: 25 Marks

Final test: 25 Marks

Midterm and final tests include experiment and viva.

**Semester End Evaluation: 50 Marks**

Procedure Write-up – 7

Viva – 8

Conduction of experiment - 35

One Question from PART A and One from Part B. Experiments to be allotted based on lots.

**Course Outcome**

Students will be able to

C01: Analyze and apply working of 8086.

CO2: Compare the various interface techniques. Analyze and apply the working of 8255, 8279, ICs and design and develop the programs. Various interfacing models

C03: Learning the Communication Standards ARM embedded programming using ARMsim Tool and Cortex Board.

**1. a) Search a key element in a list of n numbers using the *Binary* Search *algorithm***

assume cs:code,ds:data

data segment

a db 10h,20h,30h,40h,50h ; Sorting only bytes. Try for words also.

n db n-a

key db 20h

msg1 db "key not found$"

msg2 db "key found at position: "

pos db ?,"$" ; msg2 continues till here!!!

data ends

code segment

start:

mov ax,data

mov ds,ax

mov al,0 ; low

mov dl,n

dec dl ; high = n-1

again:

cmp al,dl

ja failed

mov cl,al

add al,dl ; low + high

shr al,1 ; mid

mov ah,00h

mov si,ax

mov bl,[si] ; [mid] in bl

cmp bl,key

jae loc1 ; [mid] >= key ?

inc al ; no, low = mid+1, to search in second half

jmp again

loc1:

je success ; [mid] = key ?

dec al ; no, high = mid – 1, to search in first half

mov dl,al

mov al,cl

jmp again

failed:

lea dx,msg1 ; key not found

jmp display

success:

inc al

add al,30h ; store ASCII value at pos. guess why ????

mov pos,al

lea dx,msg2

display:

mov ah,9

int 21h

mov ah,4ch

int 21h

code ends

end start

1. **b) Count no of 1’s in a logical controller input.**

assume cs:code,ds:data

data segment

pa equ 44A0h ; could be different elsewhere

pb equ 44A1h

pc equ 44A2h

cr equ 44A3h

msg db "No. of 1's : "

ones db ?,"$"

data ends

code segment

start:

mov ax,data

mov ds,ax

mov dx,cr

mov al,82h ; PA – output & PB – input

out dx,al

mov dx,pb

in al,dx ; read from PB

mov cx,8

mov ah,00

rot\_again:

ror al,1

jnc next

inc ah ; [ah] = no. of 1’s

next:

loop rot\_again

mov bl,ah ; store ASCII value of [ah]

add ah,30h

mov ones,ah

lea dx,msg

mov ah,09h

int 21h

mov al,00h

ror bl,1 ; to check odd or even no of ones

jc disp

mov al,0ffh

disp:

mov dx,pa

out dx,al

mov ah,4ch

int 21h

code ends

end start

**2. a) Sort a given set of n numbers in ascending and descending order using the *Bubble Sort algorithm***

Assume cs:code,ds:data

data segment

x db 10h,05h,03h,15h,01h ; bytes again, try urself for words

n dw n-x

data ends

code segment

start:

mov ax,data

mov ds,ax

mov bx,n

dec bx ; no. of passes required

next\_pass:

mov cx,bx ; no. of comparisons in a pass = no. of passes remaining including that pass

mov ah,00 ; a flag, guess why?

lea si,x

next\_comp:

mov al,[si]

cmp al,[si+1]

jle do\_nothing

xchg al,[si+1]

mov [si],al

mov ah,1

do\_nothing:

inc si

loop next\_comp

cmp ah, 0

je finish ; I think an efficient sort, what do you say?

dec bx

jnz next\_pass

finish:

mov ah,4ch

int 21h

code ends

end start

1. **b) Perform the following functions using the logical Controller Interface.**

**i. BCD up-down Counter ii. Ring counter**

assume cs:code,ds:data

data segment

pa equ 44A0h

pb equ 44A1h

pc equ 44A2h

cr equ 44A3h

data ends

code segment

start:

mov ax,data

mov ds,ax

mov dx,cr

mov al,82h

out dx,al

mov cx,100d

mov dx,pa

mov al,00h

next:

out dx,al

call delay

add al,01

daa ; a decimal counter, isn’t it?

loop next

mov cx,99d

next1:

sub al,01

das

out dx,al

call delay

loop next1

mov ah,4ch

int 21h

delay proc

mov si,02fffh

l2: mov di,0ffffh

l1: dec di

jnz l1

dec si

jnz l2

ret ; please don’t forget this instruction for God’s sake!!!!

delay endp

code ends

end start

2 b) ii. Ring Counter

assume cs:code,ds:data

data segment

pa equ 44A0h

pb equ 44A1h

pc equ 44A2h

cr equ 44A3h

data ends

code segment

start:

mov ax,data

mov ds,ax

mov dx,cr

mov al,82h

out dx,al

mov al,01

rpt:

mov dx,pa

out dx,al

call delay

ror al,1

push ax ; int 21h changes [al]

mov ah,06h

mov dl,0ffh

int 21h

pop ax

jz rpt ; function 06 try to sense, if a key is pressed, if yes, ZF = 0

mov ah,4ch

int 21h

delay proc

mov si,02fffh

l2: mov di,0ffffh

l1: dec di

jnz l1

dec si

jnz l2

ret

delay endp

code ends

end start

**3 a) Password Match**

assume cs:code,ds:data

data segment

password db "twilight"

new db 100 dup(?)

msg db "not authenticated$"

data ends

code segment

start:

mov ax,data

mov ds,ax

mov es,ax

mov bh,3

wait1:

lea si,new

rpt:

mov ah,7

int 21h

cmp al,0dh

je compare

mov [si],al

inc si

jmp rpt

compare:

lea si,password

lea di, new

mov cl,8

rep cmpsb

jz next

lea dx,msg

mov ah,9

int 21h

dec bh

jnz wait1

wait2: jmp wait2

next:

mov ah,4ch

int 21h

code ends

end start

**3 b) Stepper Motor**

assume cs:code,ds:data

data segment

pa equ 44A0h

pb equ 44A1h

pc equ 44A2h

cr equ 44A3h

data ends

code segment

start:

mov ax,data

mov ds,ax

mov dx,cr

mov al,80h

out dx,al

;mov cx,64h (or) mov cx,100d(for 180 degrees rotation 180/1.8)

;mov cx,32h (or) mov cx,50d(for 90 degrees rotation 90/1.8)

mov cx,64h

mov al,77h

mov dx,pc

rot\_clock:

out dx,al

ror al,1

call delay

loop rot\_clock

mov ah,4ch

int 21h

delay proc

mov bx,02fffh

l2: mov di,0ffffh

l1: dec di

jnz l1

dec bx

jnz l2

ret

delay endp

code ends

end start

**4 a) Read an alphanumeric character and display its equivalent ASCII code at the center of the screen.**

**Reverse a given string and check whether it is a palindrome or not.**

assume cs:code,ds:data

data segment

str1 db 'madam'

n dw n – str1

str2 db 5 dup(?)

msg1 db "pallindrome$"

msg2 db "not a palindrome$"

data ends

code segment

start:

mov ax,data

mov ds,ax

mov es,ax ; please observe this !!!!!!

mov cx,n

lea si,n ; a simple trick to make si pointing to last character of main string

dec si

lea di,str2

nextchar:mov al,[si]

mov [di],al

dec si

inc di

loop nextchar

lea si,str1

lea di,str2

cld ; so that si and di will be incremented

mov cx,n

rep cmpsb

jnz unsuccess

lea dx,msg1

jmp disp

unsuccess:lea dx,msg2

disp:mov ah,9h

int 21h

mov ah,07h

int 21h

code ends

end start

**4 b) Half rectified and full rectified wave form**

**Generate a Half RectifiedSine Wave form using the DAC interface ( The output of the DAC is to be displayed on a CRO)**

assume cs:code, ds:data

data segment

sine db 0, 11d, 22d, 33d, 43d, 54d, 63d, 72d, 81d, 90d, 97d, 104d, 109d, 115d, 119d, 122d

db 125d, 126d, 127d, 126d, 122d, 119d, 115d, 109d, 104d, 97d, 90d, 81d, 72d, 63d

db 54d, 43d, 33d, 22d, 11d

pa db 44a0h

cr db 44a3h

data end

code segment

start:

mov ax, data

mov ds, ax

mov dx, cr

mov al, 80h

out dx, al

repeat: mov dx, pa

lea si, sine

mov cx, 36d

next: mov al, [si]

out dx,al

inc si

loop next

jmp repeat

code ends

end start

**5 a) Compute the  *factorial*  of a positive integer ‘n’ using recursive procedure.**

assume cs:code,ds:data

data segment

n db 5

res db ?

data ends

code segment

start:

mov ax,data

mov ds,ax

mov al,n

call fact

mov ah,4ch

int 21h

fact proc

cmp al,00

je cal

push ax

dec al

call fact

pop ax ; after first return ie. from cal label, control comes here not to main program

mul res

mov res,al

ret

cal: mov res,01

ret

fact endp

code ends

end start

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

assume cs:code, ds:data

data segment

sine db 0, 11d, 22d, 33d, 43d, 54d, 63d, 72d, 81d, 90d, 97d, 104d, 109d, 115d, 119d, 122d

db 125d, 126d, 127d, 126d, 122d, 119d, 115d, 109d, 104d, 97d, 90d, 81d, 72d, 63d

db 54d, 43d, 33d, 22d, 11d

pa db 44a0h

cr db 44a3h

data end

code segment

start:

mov ax, data

mov ds, ax

mov dx, cr

mov al, 80h

out dx, al

repeat: mov dx, pa

lea si, sine

mov cx, 36d

next: mov al, [si]

add al, 128d

out dx,al

inc si

loop next

mov cx,36d

lea si, sine

next1: mov al,128d

mov ah, [si]

sub al, ah

out dx, al

inc si

loop next1

jmp repeat

code ends

end start

**6 a) Display a character at the center of the screen.**

assume cs:code

code segment

start:

mov ah,00h ; function to set video mode

mov al,02h ; 80 x 25 gray scale, try 03

int 10h ; I am BIOS remember

mov ah,02 ; function to set cursor

mov dh,12d

mov dl,40d

int 10h

mov ah,7

int 21h

call disp

mov ah,7

int 21h

mov ah,4ch

int 21h

disp proc

mov bl,al

mov cl,4

shr al,cl

cmp al,09

jle add\_30

add al,07h

add\_30: add al,30h

mov dl,al

mov ah,2

int 21h

mov al,bl

and al,0fh

cmp al,09

jle add\_30\_1

add al,07

add\_30\_1:add al,30h

mov dl,al

mov ah,2

int 21h

ret

disp endp

code ends

end start

**6 b) Row number, Column number and Scan code of keypad**

assume cs:code,ds:data

data segment

pa equ 44A0h

pb equ 44A1h

pc equ 44A2h

cr equ 44A3h

rowval db ?

colval db ?

scode db ?

data ends

code segment

start:

mov ax,data

mov ds,ax

mov dx,cr

mov al,90h

out dx,al

try\_again:

mov bl,01h

mov bh,03h

mov cl,00h

mov ah,01h

next\_row:

mov dx,pc

mov al,bl

out dx,al

mov dx,pa

in al,dx

cmp al,00h

jne calculate

add cl,08h

inc ah

shl bl,01

dec bh

jnz next\_row

jmp try\_again

calculate:

mov rowval,ah

mov ah,00h

rot\_again:

ror al,01

jnc next

inc ah

inc cl

jmp rot\_again

next:

mov scode,cl

mov colval,ah

mov al,cl

call disp

mov ah,4ch

int 21h

disp proc

mov bl,al

mov cl,4

shr al,cl

cmp al,09

jle add\_30

add al,07

add\_30:

add al,30h

mov dl,al

mov ah,02

int 21h

mov al,bl

and al,0fh

cmp al,09

jle add\_30\_1

add al,07

add\_30\_1:

add al,30h

mov dl,al

mov ah,02

int 21h

ret

disp endp

code ends

end start

**7 a) Generate the first ‘n’ *Fibonacci* numbers**

Assume cs:code,ds:data

Data segment

Fibo db 10 dup(?)

N db 0ah ; no. of Fibonacci numbers to be genetared

Data ends

Code segment

Start:

mov ax, data

mov ds, ax

lea si, fibo

mov al, 00h ; fib(n-2)

mov [si], al

inc si

mov bl, 01h ; fib(n-1)

mov [si], bl

inc si

mov cl, n

sub cl,2 ; already two numbers generated and stored

mov ch, 00 ; so that I can use loop instruction

next:number:

add al, bl ; fib(n) = fib(n-1) + fib(n-2)

mov [si], al

inc si

xchg al, bl ; al suppose to contain fib(n-2) but it has fib(n-1), so exchange with bl

loop next\_number

mov ah,4ch

int 21h

code ends

end start

**7 b) Scan a 8 x3 keypad for a key closure and simulate ADD and SUBTRACT operations as in a calculator.**

disp\_msg macro msg

lea dx,msg

mov ah,9

int 21h

endm

disp macro ; macro to display the key pressed

lea bx,table1

xlat

mov dl,al

mov ah,2

int 21h

endm

initialize macro word

mov al,word

mov dx,44a3h

out dx,al

endm

assume ds:data,cs:code

data segment

cw db 90h

ops db 3 dup(?)

table1 db '0123456789.+-'

msg1 db "not supported$"

data ends

code segment

start:

mov ax,data

mov ds,ax

initialize cw

call readkp ; to read **first operand**

call delay ; to eliminate switch debouncing problem

mov ops,al

disp ; to display the key pressed

call readkp ; to read **operator**

call delay

mov ops+1,al

disp

call readkp ; to read **second operand**

call delay

mov ops+2,al

disp

mov dl,'=' ; to display =

mov ah,2

int 21h

cmp ops+1,0bh ; to check whether to add….

jne next ; no, then try subtract……

mov al,ops ; yes, then take one operand to al

add al,ops+2 ; add second operand

daa ; we are designing BCD calculator, isn’t it?

mov cl,4

mov ah,0

shl ax,cl

shr al,cl

add ax,3030h ; hey, I remember we could use AAM, try it yourself please

mov dl,ah

mov ah,2

push ax

int 21h

pop ax

mov dl,al

mov ah,2

int 21h

jmp finish

next:

mov al,ops+1

cmp al,och ; is it to subtract the operands?

jne disp\_err ; no, then u asked me to design calculator to ADD and SUBTRACT only

mov al,ops ; if yes, then here I go….

sub al,ops+2

cmp al,09

jle noneed

neg al

mov dl,'-' ; arey yaar, my program shows negative answer also

mov ah,2

push ax

int 21h

pop ax

noneed:

add al,30h

mov dl,al

mov ah,2

int 21h

jmp finish

disp\_err:

disp\_msg msg1 ; I told I can add or subtract only

finish: mov ah,4ch

int 21h

**readkp proc ; procedure which returns scan code of the key pressed**

try\_again:

mov bl,01

mov bh,03

mov cl,00

next\_row:

mov al,bl

mov dx,44a2h

out dx,al

mov dx,44a0h

in al,dx

cmp al,00h

jne scode

add cl,08h

rol bl,1

dec bh

jnz next\_row

jmp try\_again

scode:

ror al,1

jc return

inc cl

jmp scode

return: mov al,cl

ret

readkp endp

delay proc

mov si,03fffh

l2: mov di,0ffffh

l1: dec di

jnz l1

dec si

jnz l2

ret

delay endp

code end

end start

**8 a) Read the current time from the system and display it in the standard format on the screen**

assume cs:code

code segment

start:

mov ah,2ch ;function 2C under INT 21h returns time in ch(hrs), cl(mins)

int 21h ; in hex ( seconds and milliseconds omitted)

mov al,ch

call hex\_bcd ; first convert the hrs into 24 hrs formatted bcd

call disp ; then, display it

mov dl,':'

mov ah,2

int 21h ; to display “ : “ in between HH and MM

mov al,cl ; same thing with minutes

call hex\_bcd

call disp

mov ah,4ch

int 21h

disp proc ; procedure to display 2 bcd digits

push cx

mov ah,00h

mov cx,4

shl ax,4

shr al,4

add ax,3030h

push ax

mov dl,ah

mov ah,2

int 21h

pop ax

mov ah,2

mov dl,al

int 21h

pop cx

ret

endp

hex\_bcd proc ; procedure to convert hex to bcd

push cx

mov cl,al

mov ch,0

mov al,0

next:

add al,1

daa

loop next

pop cx

ret

endp

code ends

end start

**8 b) 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).**

assume cs:code,ds:data

data segment

pa equ 44A0h

pb equ 44A1h

pc equ 44A2h

cr equ 44A3h

fire db 61h,0f5h,9fh,71h ; F I r E is reversed here

help db 31h,0e3h,61h,0d1h ; H E L P is also reversed

data ends

code segment

start:

mov ax,data

mov ds,ax

rpt:

mov cx,04h

lea si,fire

next\_char:

mov al,[si]

call disp

inc si

loop next\_char

call delay

mov cx,04h

next: mov al,[si]

call disp

inc si

loop next

mov ah,06h

mov dl,0ffh

int 21h

jz rpt

mov ah,4ch

int 21h

disp proc

push cx

mov cx,8 ; to send 8 bits one by one

next\_bit:

mov dx,pb ; through port B

out dx,al

push ax

mov al,0ffh ; since shift register is a synchronous circuit, needed a trigger ff to 00

mov dx,pc

out dx,al

mov al,00h

out dx,al

pop ax

ror al,1

loop next\_bit

pop cx

ret

disp endp

delay proc

mov bx,02fffh

l2: mov di,0ffffh

l1: dec di

jnz l1

dec bx

jnz l2

ret

delay endp

code ends

end start

**9 a) Program to simulate a Decimal Up-Counter to display 00-99**

assume cs:code

code segment

start:

mov cx,100d

mov bl,00

next\_digit: mov al,bl

aam

add ax,3030h

mov dl,ah

mov ah,2

push ax

int 21h

pop ax

mov dl,al

mov ah,2

int 21h

mov dl,0dh

mov ah,2

int 21h

call delay

inc bl

loop next\_digit

mov ah,4ch

int 21h

delay proc

mov si,02202h

l1:mov di,0ffffh

l2:dec di

jnz l2

dec si

jnz l1

ret

delay endp

code ends

end start

**9 b) Assume any suitable message of 12 characters length and display it in the *rolling* fashion on a**

**7-Segment display Interface for a suitable period of time. Ensure a flashing rate that makes it easy to read both messages. (Examiner does not specify these delay values nor is it necessary for the student to compute these values).**

assume cs:code,ds:data

data segment

pa equ 44A0h

pb equ 44A1h

pc equ 44A2h

cr equ 44A3h

msg db 61h,0f5h,9fh,71h,0ffh,31h,0e3h,61h,0d1h,0ffh,61h,0f5h,61h,0d1h

data ends

code segment

start:

mov ax,data

mov ds,ax

rpt:

mov cx,14d

lea si,msg

next\_char:

mov al,[si]

call disp

call delay ; delay is called in between characters here to give a rolling effect

inc si

loop next\_char

mov ah,06h

mov dl,0ffh

int 21h

jz rpt

mov ah,4ch

int 21h

disp proc

push cx

mov cx,8

next\_bit:

mov dx,pb

out dx,al

push ax

mov al,0ffh

mov dx,pc

out dx,al

mov al,00h

out dx,al

pop ax

ror al,1

loop next\_bit

pop cx

ret

disp endp

delay proc

mov bx,02fffh

l2: mov di,0ffffh

l1: dec di

jnz l1

dec bx

jnz l2

ret

delay endp

code ends

end start

**10 a) Find out whether a given sub string is present or not in a main string of characters.**

assume cs:code,ds:data

data segment

ms db "Angels and demons"

lms db lms-ms

str db "demons"

lss dw lss-str

msg1 db "substring has found$"

msg2 db "substring not found$"

data ends

code segment

start:

mov ax,data

mov ds,ax

mov es,ax

mov dl,lms

lea di,ms

search:

mov bx,di

lea si,str

mov cx,lss

cld

rep cmpsb

jz success

inc bx

mov di,bx

dec dl

jnz search

jmp failure

success:

lea dx,msg1

mov ah,09h

int 21h

jmp exit

failure:

lea dx,msg2

mov ah,09h

int 21h

exit: mov ah,4ch

int 21h

code ends

end start

**10 b)** assume cs:code,ds:data

data segment

pa equ 44A0h

pb equ 44A1h

pc equ 44A2h

cr equ 44A3h

data ends

code segment

start:

mov ax,data

mov ds,ax

mov al, 82h ; port A output, port B input

mov dx,cr

out dx,al

mov dx,pa

mov al,00h ; clear all requests, make elevator stand at first LED

out dx,al

mov al,0f0h ; enable requests

out dx,al

mov dx, pb ; to read request

scan\_again:

in al, dx

and al,0fh ; masking MS 4 bits

cmp al,0fh ; is there any request at all?

je scan\_again ; no, then please give one.

mov cl,01 ; up to this LED the elevator should move

rot\_again:

ror al,1 ; checking for floor from which the request has come ( a 0 represents request)

jc next ; If there is carry, then there is no request

jmp start\_mov ; there was no carry (ie. CF = 0), we identified the floor from which the

; request is made

next: add cl,03h

jmp rot\_again

start\_mov:

mov dx,pa

mov al,0f0 ; f in MS nibble says, don’t clear the request

next\_led:

out dx,al ; at last elevator started moving

call delay

inc al

dec cl

jnz next\_led

call delay ; wait for some time to pick passenger

call delay

dec al

and al,0fh ; now clear the request, but keep led number intact

come\_down:

out dx,al

call delay

dec al

cmp al,00h ; have I reached ground floor?

jge come\_down ; no, then come down still

mov ah,4ch ; hey, my program worked, can I go home?

int 21h

delay proc

mov bx,02fffh

l2: mov di,0ffffh

l1: dec di

jnz l1

dec bx

jnz l2

ret

delay endp

code ends

end start

**Arm Programs:**

1. **Add two 16-bit numbers.**

.text

.global start

loop:

swi 0x203

cmp r0,#0

beq loop

clz r5,r0

rsb r5,r5,#31

mov r3,#1

mov r4,#1

mov r6,r5

row:

cmp r6,#4

addge r3,r3,#1

subge r6,r6,#4

bge row

column:

cmp r6,#0

addgt r4,r4,#1

subgt r6,r6,#1

bgt column

1. **Search for a given key**

.text

.global \_start

\_start:

mov r7,#0

mov r5,#1

mov r6,#1

waitk:

swi 0x203

cmp r0,#0

beq waitk

loop:

mov r0,r0,lsr #1

cmp r0,#0

beq print

add r7,r7,#1

add r6,r6,#1

cmp r6,#5

moveq r6,#1

addeq r5,r5,#1

b loop

print:

swi 0x206

mov r0,#1

mov r1,#1

ldr r2,=rowpos

swi 0x204

mov r2,r5

mov r0,#25

swi 0x205

mov r0,#1

mov r1,#2

ldr r2,=colpos

swi 0x204

mov r0,#25

mov r2,r6

swi 0x205

mov r0,#1

mov r1,#3

ldr r2,=presno

swi 0x204

mov r2,r7

mov r0,#25

swi 0x205

b \_start

rowpos: .asciz "Row position is :"

colpos: .asciz "Col position is "

presno: .asciz "Pressed number is :"

.end

1. **Display Characters.**

.text

.global \_start

\_start:

wait: swi 0x202

cmp r0, #0

beq wait

ldr r5, =zero

ldr r3, =F

mov r1, #0

cmp r0, #1

bgt loop2

loop1: ldrb r0, [r5]

add r1, r1, #1

cmp r1, #17

beq end

swi 0x200

add r5, r5, #1

bl count

b loop1

loop2: ldrb r0, [r3]

add r1, r1, #1

cmp r1, #17

beq end

swi 0x200

sub r3, r3, #1

bl count

b loop2

end: b wait

count: mov r4, #64000

count1: cmp r4, #0

subgt r4, r4, #1

bgt count1

swi 0x206

mov pc, lr

.data

zero: .byte 0b11111101

one: .byte 0b01110000

two: .byte 0b11011110

three: .byte 0b11111010

four: .byte 0b01110011

five: .byte 0b10111011

six: .byte 0b10111111

seven: .byte 0b11110000

eight: .byte 0b11111111

nine: .byte 0b11111011

A: .byte 0b11100111

B: .byte 0b11101111

C: .byte 0b10001101

D: .byte 0b11101101

E: .byte 0b10001111

F: .byte 0b10000111

1. **Calculator**

.text

.global start

start:

mov r0,#0

bl getdig

mov r6,r2 ; first operand

mov r0,#3

mov r1,#2

swi 0x205

mov r0,#0

bl getop

mov r0,#6

mov r1,#2

cmp r7,#2

ldrlt r2,=op1 ;r2:operator

ldreq r2,=op2

cmp r7,#3

ldreq r2,=op3

ldrgt r2,=op5

swi 0x204

mov r0,#0

bl getdig

mov r8,r2 ;r8:second operand

mov r0,#9

mov r1,#2

swi 0x205

mov r0,#0

wait:

swi 0x203

cmp r0,#0

beq wait

cmp r7,#2

addlt r9,r6,r8

beq sub

cmp r7,#3

muleq r9,r6,r8 ;r9:result

bgt division

back: mov r0,#12

mov r1,#2

ldr r2,=op4

swi 0x204

cmp r5,#1

beq printdiff

mov r0,#16

mov r2,r9

swi 0x205

swi 0x11

getdig:

swi 0x203

cmp r0,#0

beq getdig

clz r0,r0

rsb r0,r0,#31

mov r2,r0

mov pc,lr

getop:

swi 0x203

cmp r0,#0

beq getop

cmp r0,#2

moveq r7,#2

movlt r7,#1

cmp r0,#3

moveq r7,#3

movgt r7,#4

mov pc,lr

sub: cmp r8,r6

bgt rsub

sub r9,r6,r8

b back

rsub: mov r5,#1

sub r9,r8,r6

b back

printdiff: mov r0,#15

ldr r2,=op2

swi 0x204

mov r2,r9

mov r0,#16

swi 0x205

swi 0x11

division: mov r10,#0

mov r11,r6

cmpo: cmp r11,r8

blt back

sub r11,r11,r8

add r10,r10,#1

mov r9,r10

b cmpo

.data:

op1: .asciz "+"

op2: .asciz "-"

op3: .asciz "\*"

op5: .asciz "/"

op4: .asciz "="

.end

**---- BEST OF LUCK--------**