Lab Manual – Multitasking

Activity 1: Assemble and run the code given blow (example 11.1 that we did in class):

```
; Example 11.1 - elementary multitasking of two threads

[org 0x0100]
jmp start

                ; ax,bx,ip,cs,flags storage area
pcb:    dw 0, 0, 0, 0, 0 ; task0 regs[cs:pcb + 0]
                dw 0, 0, 0, 0, 0 ; task1 regs start at [cs:pcb + 10]
                dw 0, 0, 0, 0, 0 ; task2 regs start at [cs:pcb + 20]

current:db 0 ; index of current task
chars:          db '\|/-' ; shapes to form a bar

;--------------------------------------------------------------------------
; one task to be multitasked
;--------------------------------------------------------------------------
taskone:        mov al, [chars+bx]              ; read the next shape
                        mov [es:0], al                  ; write at top left of screen
                        inc bx                                  ; increment to next shape
                        and bx, 3                               ; taking modulus by 4
                        jmp taskone                             ; infinite task

;--------------------------------------------------------------------------
; second task to be multitasked
;--------------------------------------------------------------------------
tasktwo:        mov al, [chars+bx]              ; read the next shape...0
                        mov [es:158], al                ; write at top right of screen
                        inc bx                                  ; increment to next shape
                        and bx, 3                               ; taking modulus by 4
                        jmp tasktwo                             ; infinite task

;--------------------------------------------------------------------------
; timer interrupt service routine
;--------------------------------------------------------------------------
timer:          push ax
                        push bx
```

```
                    mov bl, [cs:current]            ; read index of current task ... bl
= 0
                    mov ax, 10                      ; space used by
one task
                    mul bl                          ; multiply to get
start of task.. 10x0 = 0
                    mov bx, ax                      ; load start of
task in bx..... bx = 0

                    pop ax                          ; read original
value of bx
                    mov [cs:pcb+bx+2], ax           ; space for current task's BX

                    pop ax                          ; read original
value of ax
                    mov [cs:pcb+bx+0], ax           ; space for current task's AX

                    pop ax                          ; read original
value of ip
                    mov [cs:pcb+bx+4], ax           ; space for current task

                    pop ax                          ; read original
value of cs
                    mov [cs:pcb+bx+6], ax           ; space for current task

                    pop ax                          ; read original
value of flags
                    mov [cs:pcb+bx+8], ax           ; space for current task

                    inc byte [cs:current]           ; update current task index...1
                    cmp byte [cs:current], 3        ; is task index out of range
                    jne skipreset                   ; no, proceed
                    mov byte [cs:current], 0        ; yes, reset to task 0

skipreset:      mov bl, [cs:current]                ; read index of current task
                    mov ax, 10                      ; space used by
one task
                    mul bl                          ; multiply to get
start of task
```

```
                        mov bx, ax                                  ; load start of
task in bx... 10

                        mov al, 0x20
                        out 0x20, al                                ; send EOI to PIC

                        push word [cs:pcb+bx+8]                     ; flags of new task...
pcb+10+8
                        push word [cs:pcb+bx+6]                     ; cs of new task ...
pcb+10+6
                        push word [cs:pcb+bx+4]                     ; ip of new task...
pcb+10+4
                        mov ax, [cs:pcb+bx+0]                   ; ax of new task...pcb+10+0
                        mov bx, [cs:pcb+bx+2]                   ; bx of new task...pcb+10+2

                        iret                                        ; return to new
task

;---------------------------------------------------------------------
start:
mov ax, 1100
out 0x40, al
mov al, ah
out 0x40, al


mov word [pcb+10+4], taskone            ; initialize ip
                        mov [pcb+10+6], cs                          ; initialize cs
                        mov word [pcb+10+8], 0x0200                 ; initialize flags

                        mov word [pcb+20+4], tasktwo            ; initialize ip
                        mov [pcb+20+6], cs                          ; initialize cs
                        mov word [pcb+20+8], 0x0200                 ; initialize flags

                        mov word [current], 0                       ; set current
task index
                        xor ax, ax
                        mov es, ax
; point es to IVT base
```

```
                    cli
                    mov word [es:8*4], timer
                    mov [es:8*4+2], cs                    ;   hook
timer interrupt
                    mov ax, 0xb800
                    mov es, ax
; point es to video base
                    xor bx, bx
; initialize bx for tasks, bx=0
                    sti

                    jmp $
; infinite loop ... Task 0
```

Activity 2: In above code, timer schedules following 3 processes:

Process 0: jmp $

Process 1: Printing rotation on location [0][0] (taskone)

Process 2: Printing rotation on location [0][79] (tasktwo)

Update this program such that time schedules 5 processes:

Process 0: jmp $

Process 1: Printing rotation on location [0][0] (taskone)

Process 2: Printing rotation on location [0][79] (tasktwo)

Process 3: Printing rotation on location [20][0] (taskthree)

Process 4: Printing rotation on location [20][79] (taskfour)

Activity 3: Assemble and run following code (example 11.2 that we did in class), it takes a key from user, upon getting key it starts a new thread of infinite number printing at next line (column 70).

```
; multitasking and dynamic thread registration
[org 0x0100]
jmp start
; PCB layout:
; ax,bx,cx,dx,si,di,bp,sp,ip,cs,ds,ss,es,flags,next,dummy
; 0, 2, 4, 6, 8,10,12,14,16,18,20,22,24, 26 , 28 , 30
pcb: times 32*16 dw 0 ; space for 32 PCBs
stack: times 32*256 dw 0 ; space for 32 512 byte stacks
nextpcb: dw 1 ; index of next free pcb
current: dw 0 ; index of current pcb
lineno: dw 0 ; line number for next thread
```

```
;;;;; COPY LINES 028-071 FROM EXAMPLE 10.1 (printnum) ;;;;;
; subroutine to print a number on screen
; takes the row no, column no, and number to be printed as parameters
printnum: push bp
mov bp, sp
push es
push ax
push bx
push cx
push dx
push di
mov di, 80 ; load di with columns per row
mov ax, [bp+8] ; load ax with row number
mul di ; multiply with columns per row
mov di, ax ; save result in di
add di, [bp+6] ; add column number
shl di, 1 ; turn into byte count
add di, 8 ; to end of number location
mov ax, 0xb800
mov es, ax ; point es to video base
mov ax, [bp+4] ; load number in ax
mov bx, 16 ; use base 16 for division
mov cx, 4 ; initialize count of digits
nextdigit: mov dx, 0 ; zero upper half of dividend
div bx ; divide by 10
add dl, 0x30 ; convert digit into ascii value
cmp dl, 0x39 ; is the digit an alphabet
jbe skipalpha ; no, skip addition
add dl, 7 ; yes, make in alphabet code
skipalpha: mov dh, 0x07 ; attach normal attribute
mov [es:di], dx ; print char on screen
sub di, 2 ; to previous screen location
loop nextdigit ; if no divide it again
pop di
pop dx
pop cx
pop bx
```

```
pop ax
pop es
pop bp
ret 6



; mytask subroutine to be run as a thread
; takes line number as parameter
mytask: push bp
mov bp, sp
sub sp, 2 ; thread local variable
push ax
push bx
mov ax, [bp+4] ; load line number parameter
mov bx, 70 ; use column number 70
mov word [bp-2], 0 ; initialize local variable
printagain: push ax ; line number
push bx ; column number
push word [bp-2] ; number to be printed
call printnum ; print the number
inc word [bp-2] ; increment the local variable
jmp printagain ; infinitely print
pop bx
pop ax
mov sp, bp
pop bp
ret
; subroutine to register a new thread
; takes the segment, offset, of the thread routine and a parameter
; for the target thread subroutine
initpcb: push bp
mov bp, sp
push ax
push bx
push cx
push si
mov bx, [nextpcb] ; read next available pcb index
cmp bx, 32 ; are all PCBs used
```

```
je exit ; yes, exit
mov cl, 5
shl bx, cl ; multiply by 32 for pcb start
mov ax, [bp+8] ; read segment parameter
mov [pcb+bx+18], ax ; save in pcb space for cs
mov ax, [bp+6] ; read offset parameter
mov [pcb+bx+16], ax ; save in pcb space for ip
mov [pcb+bx+22], ds ; set stack to our segment
mov si, [nextpcb] ; read this pcb index
mov cl, 9
shl si, cl ; multiply by 512
add si, 256*2+stack ; end of stack for this thread
mov ax, [bp+4] ; read parameter for subroutine
sub si, 2 ; decrement thread stack pointer
mov [si], ax ; pushing param on thread stack
sub si, 2 ; space for return address
mov [pcb+bx+14], si ; save si in pcb space for sp
mov word [pcb+bx+26], 0x0200 ; initialize thread flags
mov ax, [pcb+28] ; read next of 0th thread in ax
mov [pcb+bx+28], ax ; set as next of new thread
mov ax, [nextpcb] ; read new thread index
mov [pcb+28], ax ; set as next of 0th thread
inc word [nextpcb] ; this pcb is now used
exit: pop si
pop cx
pop bx
pop ax
pop bp
ret 6
; timer interrupt service routine
timer: push ds
push bx
push cs
pop ds ; initialize ds to data segment
mov bx, [current] ; read index of current in bx
shl bx, 1
shl bx, 1
shl bx, 1
shl bx, 1
```

```
shl bx, 1 ; multiply by 32 for pcb start
mov [pcb+bx+0], ax ; save ax in current pcb
mov [pcb+bx+4], cx ; save cx in current pcb
mov [pcb+bx+6], dx ; save dx in current pcb
mov [pcb+bx+8], si ; save si in current pcb
mov [pcb+bx+10], di ; save di in current pcb
mov [pcb+bx+12], bp ; save bp in current pcb
mov [pcb+bx+24], es ; save es in current pcb
pop ax ; read original bx from stack
mov [pcb+bx+2], ax ; save bx in current pcb
pop ax ; read original ds from stack
mov [pcb+bx+20], ax ; save ds in current pcb
pop ax ; read original ip from stack
mov [pcb+bx+16], ax ; save ip in current pcb
pop ax ; read original cs from stack
mov [pcb+bx+18], ax ; save cs in current pcb
pop ax ; read original flags from stack
mov [pcb+bx+26], ax ; save cs in current pcb
mov [pcb+bx+22], ss ; save ss in current pcb
mov [pcb+bx+14], sp ; save sp in current pcb
mov bx, [pcb+bx+28] ; read next pcb of this pcb
mov [current], bx ; update current to new pcb
mov cl, 5
shl bx, cl ; multiply by 32 for pcb start
mov cx, [pcb+bx+4] ; read cx of new process
mov dx, [pcb+bx+6] ; read dx of new process
mov si, [pcb+bx+8] ; read si of new process
mov di, [pcb+bx+10] ; read diof new process
mov bp, [pcb+bx+12] ; read bp of new process
mov es, [pcb+bx+24] ; read es of new process
mov ss, [pcb+bx+22] ; read ss of new process
mov sp, [pcb+bx+14] ; read sp of new process
push word [pcb+bx+26] ; push flags of new process
push word [pcb+bx+18] ; push cs of new process
push word [pcb+bx+16] ; push ip of new process
push word [pcb+bx+20] ; push ds of new process
mov al, 0x20
out 0x20, al ; send EOI to PIC
mov ax, [pcb+bx+0] ; read ax of new process
```

```
mov bx, [pcb+bx+2] ; read bx of new process
pop ds ; read ds of new process
iret ; return to new process
start: xor ax, ax
mov es, ax ; point es to IVT base
cli
mov word [es:8*4], timer
mov [es:8*4+2], cs ; hook timer interrupt
sti
nextkey: xor ah, ah ; service 0 – get keystroke
int 0x16 ; bios keyboard services
push cs ; use current code segment
mov ax, mytask
push ax ; use mytask as offset
push word [lineno] ; thread parameter
call initpcb ; register the thread
inc word [lineno] ; update line number
jmp nextkey ; wait for next keypress
```

Activity 4: Update above program such that it supports 8 processes only.

Activity 5: Above program prints next thread's number on next line same column (i.e. 70th). Update this program such that new process prints number in next line with difference of 10 columns. i.e.

1st process: Number printing at 0th row, 70th column

2nd process: Number printing at 1st row, 60th column

3rd process: Number printing at 2nd row, 50th column and so on

Activity 5:

Write a program that starts a new thread of MovingStar if user presses any key, providing max 16 threads of MovingStar running simultaneously.

Practice Problem

Part 1: Write a function fallingStar that takes column number as parameter and prints a star moving in that column. For example, if colNo is 80, your function will print a star in column 80, falling from row 0 to row 24 (with some delay). After reaching row 24, it will again appear on 1st row and start falling again, in an infinite loop.

Part 2: Write a program that starts a new thread of falling star if user presses a key. Each thread will start with a difference of 5 columns i.e.

1st thread: star falling on column 80 (row 0 to 24 in infinite loop)

2nd thread: star falling on column 75 (row 0 to 24 in infinite loop)

3rd thread: start falling on column 70 and so on.