```
[org 0x100] jmp start count: dd 0 check_death: db 0 num: dw 260 w_key: db 'w' s_key: db 's'
a_key: db 'a' d_key: db 'd' e_key: db 'e' p_key: db 'p' curr_move: db 0 game_over_msg: db
'GAME 0VER' score_msg: db 'SCORE' is_eat: db 0 score: dw 0 snake_heading: db "SNAKE
GAME",0 names: db "Ali Ahmad (21F-9207) Hafiza Haiqa (21F-9107)",0 press_s: db "Press 'S'
to continue",0 instruction1: db "W", 0 instruction2: db "A S D", 0 instruction0: db "CONTROLLER
INSTRUCTIONS", 0 instruction3: db "PLAYING INSTRUCTIONS :", 0 instruction4: db "1) Snake
will die if it collides with any of the boundary", 0 instruction5: db "2) Snake will die if it collides
with itself", 0 food_color: db 1 ;--------------> SCREEN CLEARING SUBROUTINE clrscr: push ax
push es push di mov ax, 0xb800 mov es, ax mov di, 0 clr: mov word [es:di], 0x0720 add di, 2
cmp di, 4000 jne clr pop di pop es pop ax ret ;--------------> STRING PRINTING SUBROUTINE
printstr: push bp mov bp, sp push es push ax push cx push si push di push ds pop es ; load ds
in es mov di, [bp+4] ; point di to string mov cx, 0xffff ; load maximum number in cx xor al, al ;
load a zero in al repne scasb ; find zero in the string mov ax, 0xffff ; load maximum number in ax
sub ax, cx ; find change in cx dec ax ; exclude null from length jz exit ; no printing if string is
empty mov cx, ax ; load string length in cx mov ax, 0xb800 mov es, ax ; point es to video base
mov al, 80 ; load al with columns per row mul byte [bp+8] ; multiply with y position add ax,
[bp+10] ; add x position shl ax, 1 ; turn into byte offset mov di,ax ; point di to required location
mov si, [bp+4] ; point si to string mov ah, [bp+6] ; load attribute in ah cld ; auto increment mode
nextchar: lodsb ; load next char in al stosw ; print char/attribute pair loop nextchar ; repeat for
the whole string exit: pop di pop si pop cx pop ax pop es pop bp ret 8 ;--------------> BANNER
PRINTING SUBROUTINE print_banner: push 30 push 6 mov ax, 0x0A push ax mov ax,
instruction0 push ax call printstr push 41 push 8 mov ax, 0x0A push ax mov ax, instruction1
push ax call printstr push 39 push 10 mov ax, 0x0A push ax mov ax, instruction2 push ax call
printstr push 0 push 17 mov ax, 0x02 push ax mov ax, instruction3 push ax call printstr push 0
push 18 mov ax, 0x03 push ax mov ax, instruction4 push ax call printstr push 0 push 19 mov ax,
0x03 push ax mov ax, instruction5 push ax call printstr mov cx, 40 looping_delay_banner: call
delay loop looping_delay_banner call clear_scr push 35 push 5 mov ax, 0x0A push ax mov ax,
snake_heading push ax call printstr push 12 push 10 mov ax, 0x0A push ax mov ax, names
push ax call printstr push 29 push 20 mov ax, 0x0A push ax mov ax, press_s push ax call
printstr ret ;--------------> SETUP THE BEEP ISR beep_setup: push es push ax xor ax, ax mov
es, ax ;Save the original ISR mov ax, WORD [es: TIMER_INT * 4] mov WORD
[cs:original_timer_isr], ax mov ax, WORD [es: TIMER_INT * 4 + 2] mov WORD
[cs:original_timer_isr + 2], ax ;Setup the new ISR cli mov ax, beep_isr mov WORD [es:
TIMER_INT * 4], ax mov ax, cs mov WORD [es: TIMER_INT * 4 + 2], ax sti pop ax pop es ret ;--
------------> TEAR DOWN THE BEEP ISR beep_teardown: push es push ax call beep_stop xor
ax, ax mov es, ax ;Restore the old ISR cli mov ax, WORD [cs:original_timer_isr] mov WORD
[es: TIMER_INT * 4], ax mov ax, WORD [cs:original_timer_isr + 2] mov WORD [es: TIMER_INT
* 4 + 2], ax sti pop ax pop es ret ;--------------> BEEP ISR beep_isr: cmp BYTE
[cs:sound_playing], 0 je _bi_end cmp WORD [cs:sound_counter], 0 je _bi_stop dec WORD
[cs:sound_counter] jmp _bi_end _bi_stop: call beep_stop _bi_end: ;Chain jmp FAR
[cs:original_timer_isr] ;Stop beep ; beep_stop: push ax ;Stop the sound in al, 61h and al, 0fch
;Clear bit 0 (PIT to speaker) and bit 1 (Speaker enable) out 61h, al ;Disable countdown mov
BYTE [cs:sound_playing], 0 pop ax ret ;Beep ;AX = 1193180 / frequency ;BX = duration in
18.2th of sec beep_play: push ax push dx mov dx, ax mov al, 0b6h out 43h, al mov ax, dx out
42h, al mov al, ah out 42h, al ;Set the countdown mov WORD [cs:sound_counter], bx ;Start the
sound in al, 61h or al, 3h ;Set bit 0 (PIT to speaker) and bit 1 (Speaker enable) out 61h, al ;Start
the countdown mov BYTE [cs:sound_playing], 1 pop dx pop ax ret ;Keep these in the code
segment sound_playing db 0 sound_counter dw 0 original_timer_isr dd 0 TIMER_INT EQU 1ch
;--------------> EATING SUBROUTINE eat: push bp mov bp, sp sub sp, 2 push ax push bx push
cx push dx push si mov si, pos mov ax, word[len] dec ax shl ax, 1 add si, ax ;calculating the
head position of snake mov ax, word[num] mov word[bp-2], ax cmp [si], ax jne no_eat call
beep_setup ;Setup ;produces beep when eat food mov ax, 4000 ; beep frequency mov bx, 4
;time for beep call beep_play call delay call beep_teardown ;Tear down mov ax, word[num] mov
word[space], ax call print_space call print_food inc word[len] inc word[score] add si, 2 mov ax,
word[bp-2] mov word[si], ax no_eat: pop si pop dx pop cx pop bx pop ax mov sp, bp pop bp ret
;--------------> KEYBOARD INTTERUPT SUBROUTINE snake_int: push ax xor ax, ax mov ah,
0x01 int 0x16 ; call BIOS keyboard service jz no_int call movement no_int: pop ax ret ;-------------
-> RANDOM NUMBER SUBROUTINE rand_no: push ax push dx push cx rdtsc xor dx, dx mov
cx, 3998 div cx mov word[num], dx pop cx pop dx pop ax ret ;--------------> DELAY
SUBROUTINE delay: mov dword[count], 60000 looping_delay: dec dword[count] cmp
dword[count], 0 jne looping_delay ret ;--------------> CLEAR SCREEN SUBROUTINE clear_scr:
push ax push es push di mov ax, 0xb800 mov es, ax xor di, di looping_clr: mov word [es:di],
0x0720 add di, 2 cmp di, 4000 jne looping_clr pop di pop es pop ax ret ;--------------> RANDOM
FOOD GENERATING SUBROUTINE print_food: push ax push es push di push cx mov ax,
0xb800 mov es, ax call rand_no test word[num], 1 jz ok dec word[num] ok: call checking_food
mov ax, word[num] mov di, ax cmp byte[food_color], 10 jne skip_color mov byte[food_color], 1
skip_color: inc byte[food_color] mov ah, byte[food_color] mov al, 0x6F mov word[es:di], ax pop
cx pop di pop es pop ax ret ;--------------> Y-AXIS BORDER PRINTING SUBROUTINE border_y:
push bp mov bp, sp push ax push cx push es push di mov ax, 0xb800 mov es, ax mov di, [bp+4]
mov cx, 25 looping_border_y: mov word [es:di], 0x7020 add di, 160 loop looping_border_y pop
di pop es pop cx pop ax pop bp ret 2 ;--------------> X-AXIS BORDER PRINTING SUBROUTINE
border_x: push bp mov bp, sp push ax push cx push es push di mov ax, 0xb800 mov es, ax
mov di, [bp+4] mov cx, 80 looping_border_x: mov word [es:di], 0x7020 add di, 2 loop
looping_border_x pop di pop es pop cx pop ax pop bp ret 2 ;--------------> ALL BORDER
PRINTING AND CLEARING SCREEN (INITIALIZING GAME) SUBROUTINE load_border: push
ax call clear_scr push 0 call border_y push 158 call border_y push 0 call border_x push 3840
call border_x pop ax ret ;--------------> SNAKE PRINTING SUBROUTINE print_snake: push ax
push es push di push cx push bx mov ax, 0xb800 mov es, ax mov bx, pos mov cx, word[len] dec
cx looping_snake: ;snake body printing mov di, [bx] mov word[es:di], 0x0A2A add bx, 2 loop
looping_snake mov di, [bx] ;snake head printing mov word[es:di], 0x0A02 pop bx pop cx pop di
pop es pop ax ret ;--------------> SPACE PRINTING SUBROUTINE print_space: push ax push es
push di mov ax, 0xb800 mov es, ax mov word di, [space] mov word [es:di], 0x0720 pop di pop
es pop ax ret ;--------------> RIGHT SHIFTING SUBROUTINE right_shifting: push ax push cx
push si push di mov di, pos mov si, pos mov ax, word[pos] add si, 2 mov word[space], ax mov
cx, word[len] dec cx looping_right: mov ax , word[si] mov word[di], ax add si, 2 add di, 2 loop
looping_right mov bx, pos mov cx, word[len] dec cx shl cx, 1 add bx, cx add word[bx], 2 pop di
pop si pop cx pop ax ret ;--------------> LEFT SHIFTING SUBROUTINE left_shifting: push ax
push cx push si push di mov di, pos mov si, pos mov ax, word[pos] add si, 2 mov word[space],
ax mov cx, word[len] dec cx looping_left: mov ax , word[si] mov word[di], ax add si, 2 add di, 2
loop looping_left mov bx, pos mov cx, word[len] dec cx shl cx, 1 add bx, cx sub word[bx], 2 pop
di pop si pop cx pop ax ret ;--------------> DOWN SHIFTING SUBROUTINE down_shifting: push
ax push cx push si push di mov di, pos mov si, pos mov ax, word[pos] add si, 2 mov
word[space], ax mov cx, word[len] dec cx looping_down: mov ax , word[si] mov word[di], ax add
si, 2 add di, 2 loop looping_down mov bx, pos mov cx, word[len] dec cx shl cx, 1 add bx, cx add
word[bx], 160 pop di pop si pop cx pop ax ret ;--------------> DOWN SHIFTING SUBROUTINE
up_shifting: push ax push cx push si push di mov di, pos mov si, pos mov ax, word[pos] add si,
2 mov word[space], ax mov cx, word[len] dec cx looping_up: mov ax , word[si] mov word[di], ax
add si, 2 add di, 2 loop looping_up mov bx, pos mov cx, word[len] dec cx shl cx, 1 add bx, cx
sub word[bx], 160 pop di pop si pop cx pop ax ret ;--------------> BORDER DEATH CHECKING
SUBROUTINE check_border: push ax push cx push dx push si push bx mov cx, word[len] dec
cx shl cx, 1 mov bx, pos add bx, cx mov ax, [bx] mov cx, 0 check_left: ;LEFT SIDE BORDER
CHECK cmp ax, cx je found add cx, 160 cmp cx, 3840 jne check_left mov cx, 158 check_right:
;RIGHT SIDE BORDER CHECK cmp ax, cx je found add cx, 160 cmp cx, 3998 jne check_right
mov cx, 0 check_up: ;TOP SIDE BORDER CHECK cmp ax, cx je found add cx, 2 cmp cx, 158
jne check_up mov cx, 3840 check_down: ;BOTTOM SIDE BORDER CHECK cmp ax, cx je
found add cx, 2 cmp cx, 3998 jne check_down jmp skip_checking found: ;IF SNAKE TOUCHES
BORDER, COME TO "found" AND EXIT mov byte[check_death], 1 skip_checking: pop bx ;IF
NOT TOUCH BORDER, LEAVE THE LOOP AS IT IS pop si pop dx pop cx pop ax ret ;-------------
-> BORDER FOOD CHECKING SUBROUTINE checking_food: push ax push cx mov ax,
word[num] mov cx, 0 checking_left: ;LEFT SIDE BORDER CHECK cmp ax, cx je founded add
cx, 160 cmp cx, 3840 jne checking_left mov cx, 158 checking_right: ;RIGHT SIDE BORDER
CHECK cmp ax, cx je founded add cx, 160 cmp cx, 3998 jne checking_right mov cx, 0
checking_up: ;TOP SIDE BORDER CHECK cmp ax, cx je founded add cx, 2 cmp cx, 158 jne
```

```
checking_up mov cx, 3840 checking_down: ;BOTTOM SIDE BORDER CHECK cmp ax, cx je
founded add cx, 2 cmp cx, 3998 jne checking_down jmp skip_check founded: ;IF SNAKE
TOUCHES BORDER, COME TO "found" AND EXIT call print_food skip_check: pop cx pop ax
ret ;--------------> LEFT MOVEMENT SUBROUTINE move_left: push ax mov byte[curr_move], 2
looping_left_move: mov byte[check_death], 0 call check_border cmp byte[check_death], 1 je
death_left call print_snake call left_shifting call delay call print_space call eat call self_collision
call snake_int jmp looping_left_move death_left: call game_over pop ax ret ;--------------> RIGHT
MOVEMENT SUBROUTINE move_right: mov byte[curr_move], 1 push cx push ax
looping_right_move: mov byte[check_death], 0 call check_border cmp byte[check_death], 1 je
death_right call print_snake call right_shifting call delay call print_space call eat call
self_collision call snake_int jmp looping_right_move death_right: call game_over pop ax pop cx
ret ;--------------> UP MOVEMENT SUBROUTINE move_up: push ax mov byte[curr_move], 3
looping_up_move: mov byte[check_death], 0 call check_border cmp byte[check_death], 1 je
death_up call print_snake call up_shifting call delay call print_space call eat call self_collision
call snake_int call delay jmp looping_up_move death_up: call game_over pop ax pop cx ret ;-----
---------> DOWN MOVEMENT SUBROUTINE move_down: push ax mov byte[curr_move], 4
looping_down_move: mov byte[check_death], 0 call check_border cmp byte[check_death], 1 je
death_down call print_snake call down_shifting call delay call print_space call eat call
self_collision call snake_int call delay jmp looping_down_move death_down: call game_over
pop ax ret ;--------------> PAUSE SUBROUTINE pausing: looping_pausing: call snake_int jmp
looping_pausing ret ;--------------> OVERALL MOVEMENT CONTROLLER movement: push ax
xor ah, ah int 0x16 cmp al, [a_key] jne skip1 cmp byte[curr_move], 1 jne move_left skip1: cmp
al, [d_key] jne skip2 cmp byte[curr_move], 2 jne move_right skip2: cmp al, [w_key] jne skip3
cmp byte[curr_move], 4 jne move_up skip3: cmp al, [s_key] jne skip4 cmp byte[curr_move], 3
jne move_down skip4: cmp al, [e_key] jne skip5 call game_over skip5: cmp al, [p_key] jne skip6
call pausing skip6: pop ax ret ;--------------> SCORE PRINTING SUBROUTINE print_score: ;
subroutine to print a score at top left of screen ; takes the number to be printed as its parameter
push es push ax push bx push cx push dx push di mov ax, 0xb800 mov es, ax ; point es to
video base mov ax, word[score] ; load number in ax mov bx, 10 ; use base 10 for division mov
cx, 0 ; 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 push dx ; save ascii value on stack inc cx ;
increment count of values cmp ax, 0 ; is the quotient zero jnz nextdigit ; if no divide it again mov
di, 2162 ; point di to top left column nextpos: pop dx ; remove a digit from the stack mov dh,
0x03 ; use normal attribute mov [es:di], dx ; print char on screen add di, 2 ; move to next screen
location loop nextpos ; repeat for all digits on stack pop di pop dx pop cx pop bx pop ax pop es
ret ;--------------> SELF-COLLISION DEATH CHECK SUBROUTINE self_collision: push bx push
ax push cx mov bx, pos mov ax, word[len] dec ax shl ax, 1 add bx, ax mov ax, word[bx] mov bx,
pos mov cx, word[len] sub cx, 2 looping_self: cmp [bx], ax jne skip_self call game_over
skip_self: add bx, 2 dec cx jnz looping_self pop cx pop ax pop bx ret ;--------------> GAME OVER
SUBROUTINE game_over: call clear_scr mov ax, 0xb800 mov es, ax mov si, game_over_msg
mov di, 1990 mov cx, 9 mov ah, 0x04 looping_game_over: mov byte al, [si] mov word[es:di], ax
add di, 2 inc si loop looping_game_over mov si, score_msg mov ax, 0xb800 mov es, ax mov di,
2150 mov cx, 5 mov ah, 0x04 looping_score: mov byte al, [si] mov word[es:di], ax add di, 2 inc
si loop looping_score call print_score mov cx, 3 loop_over_delay: call delay loop
loop_over_delay mov ax, 0x4c00 int 0x21 ret ;--------------> GAME STARTING SUBROUTINE
start: call clrscr call print_banner loading: xor ax, ax mov ah, 0x01 int 0x16 ; call BIOS keyboard
service jz loading xor ax, ax int 0x16 cmp al, byte[s_key] je continue jmp loading continue: call
load_border call print_food call move_right mov ax, 0x4c00 int 0x21 ;--------------> SNAKE
RELATED DATA len: dw 3 space: dw 162 pos: dw 162, 164, 166
```