Bölüm 13: Soru Cevap

Mikroişlemciler

Örnek

```
jmp start:
    number dw 6
start:
    mov cx, number
    mov ax, 0
loop:
    add ax, number
    dec cx
    jnz loop
ret
```

Örnek

```
mov ax, num1        ; load first number in ax
mov bx, num2        ; load second number in bx
add ax, bx          ; accumulate sum in ax
mov bx, num3        ; load third number in bx
add ax, bx          ; accumulate sum in ax
mov num4, ax        ; store sum in num4
ret
num1 dw 5
num2 dw 10
num3 dw 15
num4 dw 0
```

Örnek

```
mov ax, num1        ; load first number in ax
mov bx, num2        ; load second number in bx
add ax, bx          ; accumulate sum in ax
mov bx, num3        ; load third number in bx
add ax, bx          ; accumulate sum in ax
mov num4, ax        ; store sum in num4
ret
num1: dw 5
num2: dw 10
num3: dw 15
num4: dw 0
```

Örnek

```
lea si, num1
mov ax, [si]      ; load first number in ax
lea si, num2
mov bx, [si]      ; load second number in bx
add ax, bx        ; accumulate sum in ax
lea si, num3
mov [si], ax      ; store sum in num4
ret
num1: dw 5
num2: dw 10
num3: dw 0
```

Örnek

```
mov si, offset num1
mov ax, [si]      ; load first number in ax
mov bx, [si+2]    ; load second number in bx
add ax, bx        ; accumulate sum in ax
mov bx, [si+4]    ; load third number in bx
add ax, bx        ; accumulate sum in ax
mov [si+6], ax    ; store sum at num1+6
ret
num1: dw 5
dw 10
dw 15
dw 0
```

Örnek

```
mov si, num1
mov ax, [si]     ; load first number in ax
mov bx, [si+2]   ; load second number in bx
add ax, bx       ; accumulate sum in ax
mov bx, [si+4]   ; load third number in bx
add ax, bx       ; accumulate sum in ax
mov [si+6], ax   ; store sum at num1+6
ret
num1: dw 5, 10, 15, 0
```

Örnek

```
mov bx, num1
mov ax, [bx]     ; load first number in ax
mov [bx+6], ax   ; store first number in result
mov ax, [bx+2]   ; load second number in ax
add [bx+6], ax   ; add second number to result
mov ax, [bx+4]   ; load third number in ax
add [bx+6], ax   ; add third number to result
ret
num1: dw 5, 10, 15, 0
```

Örnek

```
mov bx, num1
mov al, [bx]     ; load first number in al
mov dl, [bx+1]   ; load second number in bl
add al, dl       ; accumulate sum in al
mov dl, [bx+2]   ; load third number in bl
add al, dl       ; accumulate sum in al
mov [bx+3], al   ; store sum at num1+3
ret
num1: db 5, 10, 15, 0
```

Örnek

```
mov bx, num1       ; point bx to first number
mov cx, 10         ; load count of numbers in cx
mov ax, 0          ; initialize sum to zero
l1:
    add ax, [bx]   ; add number to ax
    add bx, 2      ; advance bx to next number
    sub cx, 1      ; numbers to be added reduced
    jnz l1         ; if numbers remain add next
mov [total], ax ; write back sum in memory
ret
num1: dw 10, 20, 30, 40, 50, 10, 20, 30, 40, 50
total: dw 0
```

Örnek

```
mov bx, num1        ; point bx to first number
mov cx, 10          ; load count of numbers in cx
mov ax, 0           ; initialize sum to zero
l1:
    add ax, [bx]    ; add number to ax
    add bx, 2       ; advance bx to next number
    loop l1         ; if numbers remain add next
mov [total], ax     ; write back sum in memory
ret
num1: dw 10, 20, 30, 40, 50, 10, 20, 30, 40, 50
total: dw 0
```

Örnek

```
mov bx, 0          ; initialize array index to zero
mov cx, 10         ; load count of numbers in cx
mov ax, 0          ; initialize sum to zero
l1:
    add ax, [num1+bx]   ; add number to ax
    add bx, 2           ; advance bx to next index
    loop l1             ; if numbers remain add next
mov [total], ax         ; write back sum in memory
ret
num1: dw 10, 20, 30, 40, 50, 10, 20, 30, 40, 50
total: dw 0
```

Örnek

```
mov bx, 0   ; initialize array index to zero
mov ax, 0   ; initialize sum to zero
l1:
    add ax, [num1+bx]   ; add number to ax
    add bx, 2           ; advance bx to next index
    cmp bx, 20          ; are we beyond the last index
    jne l1              ; if not add next number
mov [total], ax         ; write back sum in memory
ret
num1: dw 10, 20, 30, 40, 50, 10, 20, 30, 40, 50
total: dw 0
```

Örnek

```
jmp start              ; unconditionally jump over data
num1: dw 10, 20, 30, 40, 50, 10, 20, 30, 40, 50
total: dw 0
start:
    mov bx, 0          ; initialize array index to zero
    mov ax, 0          ; initialize sum to zero
l1:
    add ax, [num1+bx]      ; add number to ax
    add bx, 2              ; advance bx to next index
    cmp bx, 20             ; are we beyond the last index
    jne l1                 ; if not add next number
mov [total], ax            ; write back sum in memory
```

Örnek

```
jmp start
data: db 60h, 55h, 45h, 50h, 40h
swap: db 0
start:
    mov bx, 0           ; initialize array index to zero
    mov cx, 4
    mov [swap], 0       ; rest swap flag to no swaps
loop1:
    mov al, [data+bx]   ; load number in ax
    cmp al, [data+bx+1] ; compare with next number
    jbe noswap          ; no swap if already in order
```

Örnek devam

```
    mov dl, [data+bx+1]       ; load second element in dx
    mov [data+bx+1], al       ; store first number in second
    mov [data+bx], dl         ; store second number in first
    mov [swap], 1             ; flag that a swap has been done
noswap:
    add bx, 1             ; advance bx to next index
    cmp bx, 4             ; are we at last index
    jne loop1             ; if not compare next two
    cmp [swap+si], 1      ; check if a swap has been done
    je start              ; if yes make another pass
ret
```

Örnek

```
max_iterations equ 10  ; n
start:
    xor ax, ax
    mov bx, 1                   ; holds (F(n-1))
    mov cx, max_iterations      ; holds remaining iterations
loop1:
    mov dx, ax ; current (F(n))
    add ax, bx ; F(n) = F(n-1) + F(n-2)
    mov bx, dx ; Update (F(n-1)) with (F(n))
    loop loop1
ret
```

Örnek

```
jmp start
multiplicand: db 13          ; 4bit multiplicand (8bit space)
multiplier: db 5             ; 4bit multiplier
result: db 0                 ; 8bit result
start: mov cl, 4                    ; initialize bit count to four
    mov bl, [multiplicand+si]       ; load multiplicand in bl
    mov dl, [multiplier+si]         ; load multiplier in dl
checkbit: shr dl, 1          ; move right most bit in carry
    jnc skip                ; skip addition if bit is zero
    add [result+si], bl     ; accumulate result
skip: shl bl, 1              ; shift multiplicand left
    loop checkbit           ; repeat if bits left
```

SON