```
;------------print block----------------;

printblock:
   ;[bp+4] = y2 position
   ;[bp+6] = width
   ;[bp+8] = y1 position
   ;[bp+10] = x1 position
   ;[bp+12] = color
            push bp
            mov bp,sp
            push es
            push ax
            push cx
            push dx
            push di

            mov ax,0xb800
            mov es,ax
   mov dl,0
   add dl,byte[bp+8];y1-position
   pb:
               mov al,80
               mul dl
               add ax, [bp+10] ; x1-position
               shl ax,1
               mov di,ax
               mov ax,[bp+12] ; color
               mov cx,[bp+6] ; width
```

```
            cld
            rep stosw

        add dl,1
        cmp dl, byte[bp+4] ; y2-position
        jne pb
            pop di
            pop dx
            pop cx
            pop ax
            pop es
            pop bp
            ret 10

;----------print text will automatically find msg length----------;
printtext:
    ;[bp+4] = message
    ;[bp+6] = color
    ;[bp+8] = starting y position
    ;[bp+10] = starting x position

            push bp
            mov bp,sp
            push es
            push ax
            push cx
            push si
            push di

            push ds
```

```
pop es
mov di,word[bp+4]
mov cx,0xffff
xor al,al
repne scasb
mov ax,0xffff
sub ax,cx
dec ax
jz exit

mov cx,ax
mov ax,0xb800
mov es,ax
mov al,80
mul byte [bp+8] ; y-pos
add ax, [bp+10] ; x-pos
shl ax,1
mov di,ax
mov si, word[bp+4] ; string
mov ah, byte[bp+6] ; color

cld
nextchar:
        lodsb
        stosw
        loop nextchar

exit:
        pop di
        pop si
        pop cx
```

```
                pop ax
                pop es
                pop bp
                ret 8

;----------------print num----------------;

printnum:
        ; [bp+4] = number
        ; [bp+6] = y-pos
        ; [bp+8] = x-pos
        ; [bp+10]= color

        push bp
        mov bp, sp
        push es
        push ax
        push bx
        push cx
        push dx
        push di

        mov ax, 0xb800
        mov es, ax
        mov bx, 10
        mov cx, 0
        mov al,80
        mul byte [bp+6] ; y-pos
        add ax, [bp+8] ; x-pos
        shl ax,1
        mov di,ax
```

```
        mov ax, [bp+4] ; number
        nextdigit:
                mov dx, 0
                div bx
                add dl, 0x30
                push dx
                inc cx
                cmp ax, 0
                jnz nextdigit


        nextpos:
                pop dx
                mov dh, byte[bp+10] ; color
                mov [es:di], dx
                add di, 2
                loop nextpos

                pop di
                pop dx
                pop cx
                pop bx
                pop ax
                pop es
                pop bp
                ret 8

;----------Rectangle----------;

mov si, [atr]
```

```
mov ax, [top]
mov bx, [left]
mov cx, [bottom]
mov dx, [right]
push si
push ax
push bx
push cx
push dx


printrec:
mov bp, sp
mov ax, 0xb800
mov es, ax

mov ax, [bp+6]
shl ax, 1
mov bx, 0

l1:
add ax, 160
inc bx
cmp bx,[bp+8]
jne l1

mov bx, 0
mov cx, [bp+2]
shl cx, 1

l2:
```

```
add cx, 160
inc bx
cmp bx, [bp+8]
jne l2

mov di, ax
mov ah,[bp+10]
mov al,0x2A

l3:
mov word[es:di],ax
add di,2
cmp di, cx
jbe l3

mov ax, [bp+6]
shl ax, 1
mov bx, 0

l4:
add ax, 160
inc bx
cmp bx,[bp+4]
jne l4

mov bx, 0
mov cx, [bp+2]
shl cx, 1

l5:
add cx, 160
```

```
inc bx
cmp bx, [bp+4]
jne l5

mov di, ax
mov ah,[bp+10]
mov al,0x2A

l6:
mov word[es:di],ax
add di,2
cmp di, cx
jbe l6

mov ax, [bp+6]
shl ax, 1
mov bx, 0

l7:
add ax, 160
inc bx
cmp bx,[bp+8]
jne l7
add ax, 160

mov bx, 0
mov cx, [bp+6]
shl cx, 1

l8:
add cx, 160
```

```
inc bx
cmp bx, [bp+4]
jne l8

mov di, ax
mov ah,[bp+10]
mov al,0x2A

l9:
mov word[es:di],ax
add di,160
cmp di, cx
jbe l9

mov ax, [bp+2]
shl ax, 1
mov bx, 0

l10:
add ax, 160
inc bx
cmp bx,[bp+8]
jne l10
add ax, 160

mov bx, 0
mov cx, [bp+2]
shl cx, 1

l11:
add cx, 160
```

```
inc bx
cmp bx, [bp+4]
jne l11

mov di, ax
mov ah,[bp+10]
mov al,0x2A

l12:
mov word[es:di],ax
add di,160
cmp di, cx
jbe l12

ret 10

;------------triangle------------;
[org 0x0100]


 mov ax,0xb800
 mov es,ax

 mov di,0

 nextc:
 mov word[es:di],0x0720
 add di,2
 cmp di,3838
 jne nextc
```

```
 mov si,68+160
 mov ah,7
 mov al,"*"
 mov bx,3146

 mov cx,bx
 sub cx,160


 mov dx,4

mov word[es:si],ax

l1:
 add si,158
 mov word[es:si],ax

add si,dx
 mov word[es:si],ax
sub si,dx

 add dx,4


cmp si,cx
jl l1


mov word[es:si],ax
add si,4
```

```
mov cx,si

add cx,dx

sub cx,4

l2:

mov word[es:si],ax

add si,4

cmp si,cx

jl l2
```

;----------Copy half screen----------;

```
start:
    mov ax, 12
    push ax
    call fun1

fun1:
    push bp
    mov bp,sp
    push ax
    push cx
    push si
    push di
    push es
    push ds

    mov ax, 80
    mul byte [bp+4]
    push ax
    shl  ax,1
    mov si,3998
    sub si,ax

    mov cx,2000
    shr ax,1
    sub cx, ax

    mov ax, 0xb800
    mov es, ax
    mov ds, ax
    mov di,3998

    std
    rep movsw

    pop ds
    pop es
    pop di
    pop si
    pop cx
    pop ax
    pop bp
    ret 2
```

;----------Pipeline formulas----------;

IDEAL CASE (DERIVATIONS)

All stages take equal amount of time T

Latch time = 0

Stages = k

Number of instructions = n

Then

Clock cycle of pipeline = T , clock cycle of non pipeline= k*T

Frequency of pipeline=1/T , Frequency of non pipeline=1/k*T

Time taken to complete n instructions without pipeline= n*k*T= n*clock cycle of non pipeline
Through put for n instructions without pipeline= n/ n*k*T

Time taken to complete n instructions with pipeline= (k+n-1)*T= (k+n-1)*(Clock Cycle of pipeline)
Through put for n instructions with pipeline= n/ (k+n-1)*T

Speedup for n instructions = n*k*T/ (k+n-1)*T = n*k/ (k+n-1)

Latency without pipelining = k*T

Latency with pipelining= k*T

NON IDEAL CASE

(DERIVATIONS)

If all stages do not take same time

Non pipeline processor takes T1 time to complete one instruction

In pipeline processor max time take by any stage is T2

Latch time=0

Stages =k

Number of instructions = n

Then

Clock cycle of pipeline processor = T2, Clock Cycle of processor without pipelining = T1 Frequency of pipeline processor =1/T2, Frequency of processor without pipelining = 1/T1 Time taken to complete n instructions without pipeline= n*T1= n*clock cycle of non pipeline Through put for n instructions without pipeline= n/ n*T1

Time taken to complete n instructions with pipeline= (k+n-1)*T2 =(k+n-1)*(Clock Cycle of pipeline)
Through put for n instructions with pipeline= n/ (k+n-1)*T2

Speedup for n instructions = n*T1/ (k+n-1)*T2

Latency without pipelining = T1

Latency with pipelining= K*T2

PIPELINE IN COMPUTER:: PERFORMANCE

Including latch time

Speedup= n* T1 / (k+(n-1))* (T2+ T3)

Where n is number of instruction, k is number of stages, T1 time require by one instruction to complete without pipeline, and T2 is max time require of all the stages, T3 is latch time (same as transfer time as explained in slide 8)

PIPELINE IN COMPUTER::

PERFORMANCE

NON IDEAL CASE WITH LATCH TIME (DERIVATIONS) If all stages do not take same time

Non pipeline processor takes T1 time to complete one instruction

In pipeline processor max time take by any stage is T2

Latch time=T3

Stages =k

Number of instructions = n

Then

Clock cycle of pipeline processor = T2+T3, Clock Cycle of processor without pipelining = T1

Frequency of pipeline processor =1/(T2+T3), Frequency of processor without pipelining = 1/(T1)

Time taken to complete n instructions without pipeline= n*T1= n*clock cycle of non pipeline

Through put for n instructions without pipeline= n/ n*T1

Time taken to complete n instructions with pipeline= (k+n-1)*(T2+T3)= (k+n-1)*(Clock Cycle of pipeline)

Through put for n instructions with pipeline= n/ (k+n-1)*(T2+T3)

Speedup for n instructions = n*T1/ (k+n-1)*(T2+T3)

Latency without pipelining = T1

Latency with pipelining= K*(T2+T3)

;----------Cache Formulas----------;

Direct Mapping:

• Index = Main Memory Address % Size of Cache

• Data is placed in cache at this index

• Tag = Main Memory Address / Size of Cache

• As stored in cache along with data

• Main Memory Address = Tag*Size of Cache + Index

The size of tag will depend of size of cache and size of RAM

• Number of bits required for tag

=Number of bits required for Ram address – number of bits of cache index

= log 2(Size of ram) - log 2(size of cache)

=log2(size of Ram/ Size of Cache)