Chapter 5.2: Interrupt Functions

Microprocessors

BIOS and DOS | Calling a service | INT 10h video | The attribute byte | INT 16h keyboard | INT 21h character I/O | Strings and line input | Date and time | INT 1Ah and the BIOS data area | Mode 13h graphics | Ending a program | Example: printing binary

Sercan KÜLCÜ · Giresun University · Computer Engineering

© Sercan Külcü. All rights reserved. · Interactive HTML version of these slides:
https://sercankulcu.github.io/files/microprocessors/slides/Chapter_05_02_Interrupt_Functions.html

Learning Outcomes

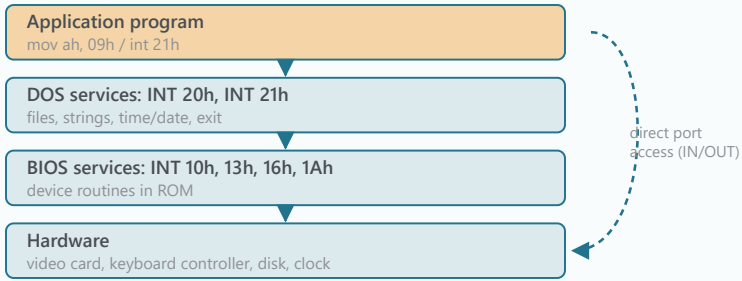
Calling services
Call BIOS and DOS services with a function number in AH and parameters in registers.

Screen output
Print characters, colored characters and strings with INT 10h and INT 21h; position the cursor.

Keyboard input
Read keys with INT 16h and INT 21h, check for a key without waiting, read a line.

Clock and exit
Read and print the date and time; end a program correctly with RET, INT 20h or INT 21h/4Ch.

BIOS and DOS Services



- **BIOS** (Basic Input/Output System): routines in ROM on the motherboard that drive the hardware directly. They work without an operating system and fill the interrupt vector table at boot.
- **DOS** sits on top of the BIOS and provides file, string, clock and program management. For its own work it usually calls the BIOS.
- Both layers are reached through **software interrupts** (`INT n`); the caller does not need to know the service's address.

The same job can be done at different layers: a character on screen via **INT 21h/02h** (DOS), **INT 10h/0Eh** (BIOS), or by writing straight into **B800h** video memory. Higher layers are more portable; lower layers are faster and more flexible.

The Service Call Pattern

1. **AH** ← service number (8 bits: at most 256 services)
2. Other parameters: AL, BX, CX, DX, DS:DX, ES:BP …
3. `INT n` (10h, 16h, 21h …)
4. Results come back in registers: AL, AX, CX, DX, ZF/CF …

```
mov ah, 02h     ; service: print char
mov dl, 'A'     ; parameter
int 21h         ; call DOS
```

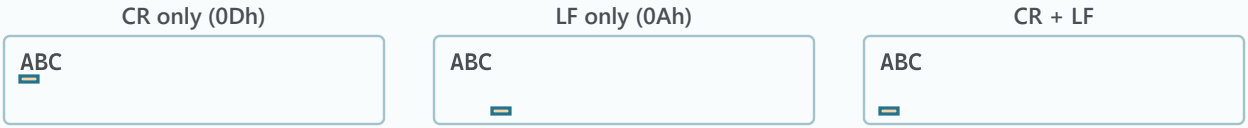
Interrupt	Provider	Purpose
`INT 10h`	BIOS	video: mode, cursor, characters, strings
`INT 16h`	BIOS	keyboard
`INT 1Ah`	BIOS	tick counter, real-time clock
`INT 13h`	BIOS	disk
`INT 20h`	DOS	terminate program (COM)
`INT 21h`	DOS	hundreds of services: I/O, files, clock, exit

A service may change the registers it uses (INT 21h/02h and 09h change AL, for example); keep values you need later **in another register** or on the stack. Some services also use AL as a subfunction (INT 10h, AX = 4F00h: VESA).

ASCII on the Screen and Keyboard

Code	Name	Effect (0Eh / 02h)
`07h`	BEL	beep
`08h`	BS	cursor back one
`09h`	TAB	tab (DOS)
`0Ah`	LF	down one line
`0Dh`	CR	start of line (Enter key)
`1Bh`	ESC	Esc key

Character	ASCII	Relation
space	`20h`	
'0' … '9'	`30h–39h`	digit = code − 30h
'A' … 'Z'	`41h–5Ah`	bit 5 = 0
'a' … 'z'	`61h–7Ah`	bit 5 = 1 (+20h)
'$'	`24h`	end of string for service 09h



A new line takes **two** characters: `0Dh`, `0Ah` (CR LF). LF alone moves the cursor down but not back to the start of the line. Turkish letters (ç, ğ, ı, ö, ş, ü) are not in 7-bit ASCII; how they show depends on the code page (CP857).

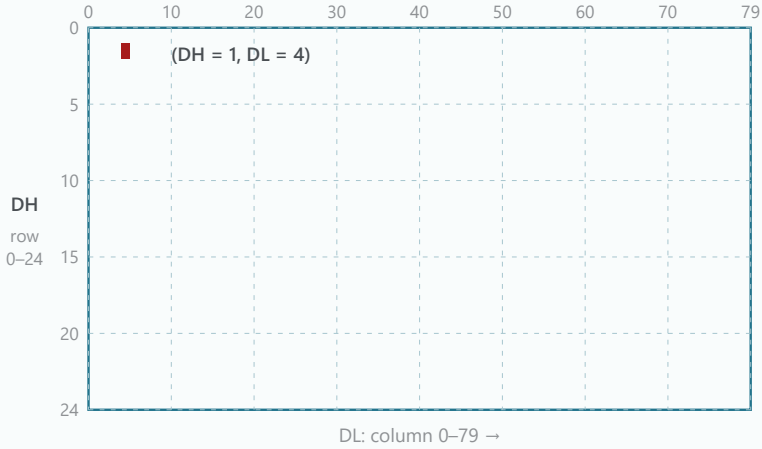
INT 10h: BIOS Video Services

AH	Service	Input	Output / note
`00h`	set video mode	AL = mode (03h: 80×25 color text)	clears the screen
`01h`	cursor shape	CH = start, CL = end scan line	CH = 20h → cursor hidden
`02h`	set cursor position	BH = page, DH = row, DL = column	
`03h`	read cursor position	BH = page	DH, DL = position; CH, CL = shape
`06h`	scroll window up	AL = lines (0 = clear), BH = attribute, CH,CL – DH,DL = corners	clears screen/window
`09h`	write char + attribute	AL = char, BH = page, BL = attribute, CX = count	cursor does **not** move
`0Ah`	write char only	AL, BH, CX	keeps the existing attribute
`0Eh`	teletype output	AL = char, BH = page	cursor moves; CR, LF, BS, BEL are obeyed
`13h`	write string	AL = mode, BH, BL, CX = length, DH,DL, **ES:BP** = string	AT and later BIOS

AH = 00h: Video Mode and the Text Screen

AL	Mode	Memory
`01h`	40×25 text, 16 colors	B800h
`03h`	**80×25 text, 16 colors**	B800h
`07h`	80×25 monochrome	B000h
`04h`	320×200, 4 colors	B800h
`13h`	320×200, 256 colors	A000h

```
mov ah, 00h
mov al, 03h   ; or mov ax, 0003h
int 10h       ; clean 80×25 screen
```



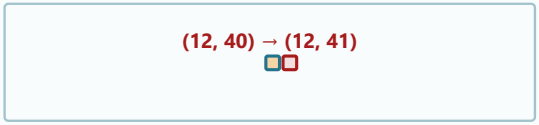
The emu8086 screen is in 80×25 text mode by default; graphics modes have limited support.

Cursor Position: AH = 02h / 03h

- Row in **DH** (0–24), column in **DL** (0–79); the top-left corner is (0, 0).
- **BH** = display page. 80×25 mode has 8 pages (0–7); page 0 is the default.
- Both in one instruction: `mov dx, 0C28h` → DH = 12, DL = 40 (the middle of the screen).
- AH = 03h reads the position: DH, DL on return, plus the cursor shape in CH, CL.
- Characters written through INT 21h also start at this cursor position.

```
; move the cursor to the middle
mov ah, 02h
mov bh, 0
mov dx, 0C28h   ; row 12, column 40
int 10h

; move the cursor one right
mov ah, 03h
mov bh, 0
int 10h         ; DH, DL = position
inc dl
mov ah, 02h
int 10h
```



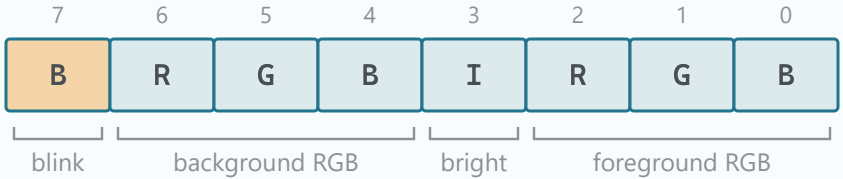
`DH = 0Ch = 12 (row), DL = 28h = 40 (column)`

read with 03h → inc dl → set with 02h: (12, 41)

offset (12 × 80 + 40) × 2 = 07D0h

In video memory, the offset of cell (row, column) = **(row × 80 + column) × 2**. For example (1, 4) → 84 × 2 = **00A8h**; the character is at the even address, the attribute at the odd one.

The Attribute Byte



Example: 1Eh = 0001 1110b → yellow (bright brown) on blue

0 black	1 blue	2 green	3 cyan
4 red	5 magenta	6 brown	7 light gray
8 dark gray	9 light blue	A light green	B light cyan
C light red	D light magenta	E yellow	F white

- Attribute = background × 16 + foreground. `07h` is the default (light gray on black), `1Fh` white on blue, `4Eh` yellow on red.
- If bit 7 = 1 the character blinks (or, depending on the BIOS setting, the background is bright): `8Fh`.

INT 10H VIDEO

Attribute Explorer

Foreground E yellow Background 4 red ☐ blink Character A



0 1 0 0 1 1 1 0

BL = 4Eh

`mov ah, 09h` · `mov al, 'A'` · `mov bh, 0` · `mov bl, 4Eh` · `mov cx, 3` · `int 10h`

The chosen colors produce the attribute byte and the INT 10h/09h call.

Writing a Character: AH = 09h or 0Eh?

Feature	AH = 09h	AH = 0Eh (teletype)
Inputs	AL = char, BH = page, BL = attribute, CX = count	AL = char, BH = page
Color	**yes** (BL)	not in text mode (the cell keeps its color)
Cursor	**does not move**	moves, wraps to the next line
CR, LF, BS, BEL	printed as symbols (♪, ◘ …)	obeyed as controls
Scrolling	none	scrolls the screen up at the last line
Typical use	colored boxes, frames, one character CX times	plain text output



A common pattern for colored text: write the character in color with **09h** (CX = 1), then move the cursor one right with **02h**; or use **13h** for a whole string.

Code Trace: Mode, Cursor, 09h and 0Eh

```
1  org 100h
▶  mov ax, 0003h     ; AH=00h, AL=03h
3  int 10h           ; clears screen
4  mov ah, 02h       ; set cursor
5  mov bh, 0         ; page 0
6  mov dx, 0104h     ; DH=1, DL=4
7  int 10h
8  mov ah, 09h       ; char + attribute
9  mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

AH = 00h (set mode), AL = 03h (80×25 16-color text). One instruction loads both registers.

Step 1 / 17

VARIABLES

AX = **0003h** BX = **0000h** CX = **0000h** DX = **0000h**

OUTPUT

Code Trace: Mode, Cursor, 09h and 0Eh

```
1  org 100h
2  mov ax, 0003h     ; AH=00h, AL=03h
▶  int 10h           ; clears screen
4  mov ah, 02h       ; set cursor
5  mov bh, 0         ; page 0
6  mov dx, 0104h     ; DH=1, DL=4
7  int 10h
8  mov ah, 09h       ; char + attribute
9  mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

`int 10h` : the mode is set, the **screen is cleared** and the cursor goes to (0,0).

Step 2 / 17

VARIABLES

AX = **0003h** BX = **0000h** CX = **0000h** DX = **0000h**

OUTPUT

Code Trace: Mode, Cursor, 09h and 0Eh

```
1  org 100h
2  mov ax, 0003h     ; AH=00h, AL=03h
3  int 10h           ; clears screen
▶  mov ah, 02h       ; set cursor
5  mov bh, 0         ; page 0
6  mov dx, 0104h     ; DH=1, DL=4
7  int 10h
8  mov ah, 09h       ; char + attribute
9  mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

AH = 02h: set the cursor position.

Step 3 / 17

VARIABLES

AX = 0203h BX = 0000h CX = 0000h DX = 0000h

OUTPUT

Code Trace: Mode, Cursor, 09h and 0Eh

```
1  org 100h
2  mov ax, 0003h     ; AH=00h, AL=03h
3  int 10h           ; clears screen
4  mov ah, 02h       ; set cursor
▶  mov bh, 0         ; page 0
6  mov dx, 0104h     ; DH=1, DL=4
7  int 10h
8  mov ah, 09h       ; char + attribute
9  mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

BH = display page (0–7 in text mode; normally 0).

Step 4 / 17

VARIABLES

AX = **0203h** BX = **0000h** CX = **0000h** DX = **0000h**

OUTPUT



Code Trace: Mode, Cursor, 09h and 0Eh

```
1  org 100h
2  mov ax, 0003h     ; AH=00h, AL=03h
3  int 10h           ; clears screen
4  mov ah, 02h       ; set cursor
5  mov bh, 0         ; page 0
▶  mov dx, 0104h     ; DH=1, DL=4
7  int 10h
8  mov ah, 09h       ; char + attribute
9  mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

DX = 0104h → **DH = 1** (row), **DL = 4** (column). The top-left corner is (0,0).

Step 5 / 17

VARIABLES

AX = **0203h** BX = **0000h** CX = **0000h**

DX = **0104h**

OUTPUT

Code Trace: Mode, Cursor, 09h and 0Eh

```
1  org 100h
2  mov ax, 0003h     ; AH=00h, AL=03h
3  int 10h           ; clears screen
4  mov ah, 02h       ; set cursor
5  mov bh, 0         ; page 0
6  mov dx, 0104h     ; DH=1, DL=4
▶  int 10h
8  mov ah, 09h       ; char + attribute
9  mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

The cursor moves to row 1, column 4. Nothing on the screen yet.

Step 6 / 17

VARIABLES

AX = **0203h** BX = **0000h** CX = **0000h** DX = **0104h**

OUTPUT

Code Trace: Mode, Cursor, 09h and 0Eh

```
1  org 100h
2  mov ax, 0003h     ; AH=00h, AL=03h
3  int 10h           ; clears screen
4  mov ah, 02h       ; set cursor
5  mov bh, 0         ; page 0
6  mov dx, 0104h     ; DH=1, DL=4
7  int 10h
▶  mov ah, 09h       ; char + attribute
9  mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

AH = 09h: write AL at the cursor CX times with attribute BL.

Step 7 / 17

VARIABLES

AX = 0903h BX = 0000h CX = 0000h DX = 0104h

OUTPUT

Code Trace: Mode, Cursor, 09h and 0Eh

```
 1 org 100h
 2 mov ax, 0003h     ; AH=00h, AL=03h
 3 int 10h           ; clears screen
 4 mov ah, 02h       ; set cursor
 5 mov bh, 0         ; page 0
 6 mov dx, 0104h     ; DH=1, DL=4
 7 int 10h
 8 mov ah, 09h       ; char + attribute
▶ mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

The character to write: AL = '*' (2Ah).

Step 8 / 17

VARIABLES

AX = **092Ah** BX = **0000h** CX = **0000h** DX = **0104h**

OUTPUT

Code Trace: Mode, Cursor, 09h and 0Eh

```
 1 org 100h
 2 mov ax, 0003h     ; AH=00h, AL=03h
 3 int 10h           ; clears screen
 4 mov ah, 02h       ; set cursor
 5 mov bh, 0         ; page 0
 6 mov dx, 0104h     ; DH=1, DL=4
 7 int 10h
 8 mov ah, 09h       ; char + attribute
 9 mov al, '*'
▶mov bl, 4Eh        ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

BL = 4Eh: background 4 (red), foreground E (yellow).

Step 9 / 17

VARIABLES

AX = **092Ah** BX = **004Eh** CX = **0000h** DX = **0104h**

OUTPUT

Code Trace: Mode, Cursor, 09h and 0Eh

```
 1 org 100h
 2 mov ax, 0003h     ; AH=00h, AL=03h
 3 int 10h           ; clears screen
 4 mov ah, 02h       ; set cursor
 5 mov bh, 0         ; page 0
 6 mov dx, 0104h     ; DH=1, DL=4
 7 int 10h
 8 mov ah, 09h       ; char + attribute
 9 mov al, '*'
10 mov bl, 4Eh       ; yellow on red
▶ mov cx, 3          ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

Repeat count CX = 3.

Step 10 / 17

VARIABLES

AX = **092Ah** BX = **004Eh** CX = **0003h** DX = **0104h**

OUTPUT

Code Trace: Mode, Cursor, 09h and 0Eh

```
 1 org 100h
 2 mov ax, 0003h     ; AH=00h, AL=03h
 3 int 10h           ; clears screen
 4 mov ah, 02h       ; set cursor
 5 mov bh, 0         ; page 0
 6 mov dx, 0104h     ; DH=1, DL=4
 7 int 10h
 8 mov ah, 09h       ; char + attribute
 9 mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
▶ int 10h            ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

*** appears, but 09h **does not move the cursor**: it is still at (1,4).

Step 11 / 17

VARIABLES

AX = **092Ah** BX = **004Eh** CX = **0003h** DX = **0104h**

OUTPUT

```
    ***
```

Code Trace: Mode, Cursor, 09h and 0Eh

```
 1 org 100h
 2 mov ax, 0003h     ; AH=00h, AL=03h
 3 int 10h           ; clears screen
 4 mov ah, 02h       ; set cursor
 5 mov bh, 0         ; page 0
 6 mov dx, 0104h     ; DH=1, DL=4
 7 int 10h
 8 mov ah, 09h       ; char + attribute
 9 mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
▶ mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

AH = 0Eh: teletype output.

Step 12 / 17

VARIABLES

AX = **0E2Ah** BX = **004Eh** CX = **0003h** DX = **0104h**

OUTPUT

```
    ***
```

Code Trace: Mode, Cursor, 09h and 0Eh

```
 1 org 100h
 2 mov ax, 0003h     ; AH=00h, AL=03h
 3 int 10h           ; clears screen
 4 mov ah, 02h       ; set cursor
 5 mov bh, 0         ; page 0
 6 mov dx, 0104h     ; DH=1, DL=4
 7 int 10h
 8 mov ah, 09h       ; char + attribute
 9 mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
▶ mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

AL = 'A'.

Step 13 / 17

VARIABLES

AX = **0E41h** BX = **004Eh** CX = **0003h** DX = **0104h**

OUTPUT

```
    ***
```

Code Trace: Mode, Cursor, 09h and 0Eh

```
1  org 100h
2  mov ax, 0003h     ; AH=00h, AL=03h
3  int 10h           ; clears screen
4  mov ah, 02h       ; set cursor
5  mov bh, 0         ; page 0
6  mov dx, 0104h     ; DH=1, DL=4
7  int 10h
8  mov ah, 09h       ; char + attribute
9  mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
▶ int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
18 ret
```

'A' is written at the cursor (1,4), **over** the first star; the cursor moves to (1,5).

Step 14 / 17

VARIABLES

AX = **0E41h** BX = **004Eh** CX = **0003h** DX = **0104h**

OUTPUT

```
    A**
```

Code Trace: Mode, Cursor, 09h and 0Eh

```
1  org 100h
2  mov ax, 0003h     ; AH=00h, AL=03h
3  int 10h           ; clears screen
4  mov ah, 02h       ; set cursor
5  mov bh, 0         ; page 0
6  mov dx, 0104h     ; DH=1, DL=4
7  int 10h
8  mov ah, 09h       ; char + attribute
9  mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
▶ mov al, 'B'
17 int 10h
18 ret
```

AL = 'B'. AH is still 0Eh.

Step 15 / 17

VARIABLES

AX = **0E42h** BX = **004Eh** CX = **0003h** DX = **0104h**

OUTPUT

```
    A**
```

Code Trace: Mode, Cursor, 09h and 0Eh

```
1  org 100h
2  mov ax, 0003h     ; AH=00h, AL=03h
3  int 10h           ; clears screen
4  mov ah, 02h       ; set cursor
5  mov bh, 0         ; page 0
6  mov dx, 0104h     ; DH=1, DL=4
7  int 10h
8  mov ah, 09h       ; char + attribute
9  mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
▶ int 10h
18 ret
```

'B' goes to (1,5): the screen shows **AB***. In text mode 0Eh takes no attribute; the cells keep their 4Eh color.

Step 16 / 17

VARIABLES

AX = **0E42h** BX = **004Eh** CX = **0003h** DX = **0104h**

OUTPUT

```
AB*
```

Code Trace: Mode, Cursor, 09h and 0Eh

```
1  org 100h
2  mov ax, 0003h     ; AH=00h, AL=03h
3  int 10h           ; clears screen
4  mov ah, 02h       ; set cursor
5  mov bh, 0         ; page 0
6  mov dx, 0104h     ; DH=1, DL=4
7  int 10h
8  mov ah, 09h       ; char + attribute
9  mov al, '*'
10 mov bl, 4Eh       ; yellow on red
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor stays!
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
15 int 10h           ; cursor moves
16 mov al, 'B'
17 int 10h
▶  ret
```

09h does not move the cursor; after writing, advance it separately with AH = 02h.

Step 17 / 17

VARIABLES

AX = **0E42h** BX = **004Eh** CX = **0003h** DX = **0104h**

OUTPUT

```
AB*
```

INT 10H VIDEO

Demo: From Text to INT 10h Code

For each character of the text, the `MOV AL` / `INT 10h` pair and its ASCII code are generated.

Text Hello

```
MOV AH, 0Eh

MOV AL, 'H' ; 72 = 48h
INT 10h
MOV AL, 'e' ; 101 = 65h
INT 10h
MOV AL, 'l' ; 108 = 6Ch
INT 10h
MOV AL, 'l' ; 108 = 6Ch
INT 10h
MOV AL, 'o' ; 111 = 6Fh
INT 10h
RET
```

```
Hello_
```

12 instructions, 5 interrupt calls

Code Trace: A Colored String with AH = 13h

```
1 org 100h
▶ mov ax, 1301h     ; AH=13h, AL=01h
3 mov bx, 001Eh     ; BH=0, BL=1Eh
4 mov cx, 7         ; string length
5 mov dx, 0205h     ; row 2, col 5
6 lea bp, msg       ; ES:BP → string
7 int 10h
8 ret
9 msg db 'Welcome'
```

AH = 13h (write string). AL = 01h: every character gets attribute BL, and the **cursor moves** after writing.

Step 1 / 7

VARIABLES

AX = **1301h** BX = **0000h** CX = **0000h** DX = **0000h** BP = **0000h**

OUTPUT

Code Trace: A Colored String with AH = 13h

```
1 org 100h
2 mov ax, 1301h     ; AH=13h, AL=01h
▶ mov bx, 001Eh     ; BH=0, BL=1Eh
4 mov cx, 7         ; string length
5 mov dx, 0205h     ; row 2, col 5
6 lea bp, msg       ; ES:BP → string
7 int 10h
8 ret
9 msg db 'Welcome'
```

BH = 0 (page), BL = 1Eh: yellow (E) on blue (1).

Step 2 / 7

VARIABLES

AX = **1301h** BX = **001Eh** CX = **0000h**
DX = **0000h** BP = **0000h**

OUTPUT

Code Trace: A Colored String with AH = 13h

```
1 org 100h
2 mov ax, 1301h     ; AH=13h, AL=01h
3 mov bx, 001Eh     ; BH=0, BL=1Eh
▶ mov cx, 7         ; string length
5 mov dx, 0205h     ; row 2, col 5
6 lea bp, msg       ; ES:BP → string
7 int 10h
8 ret
9 msg db 'Welcome'
```

CX = number of characters. There is **no** terminator such as '$'.

Step 3 / 7

VARIABLES

AX = **1301h** BX = **001Eh** CX = **0007h**

DX = **0000h** BP = **0000h**

OUTPUT

Code Trace: A Colored String with AH = 13h

```
1 org 100h
2 mov ax, 1301h     ; AH=13h, AL=01h
3 mov bx, 001Eh     ; BH=0, BL=1Eh
4 mov cx, 7         ; string length
▶ mov dx, 0205h     ; row 2, col 5
6 lea bp, msg       ; ES:BP → string
7 int 10h
8 ret
9 msg db 'Welcome'
```

DH = 2, DL = 5: where writing starts.

Step 4 / 7

VARIABLES

AX = **1301h** BX = **001Eh** CX = **0007h**

DX = **0205h** BP = **0000h**

OUTPUT



Code Trace: A Colored String with AH = 13h

```
1 org 100h
2 mov ax, 1301h     ; AH=13h, AL=01h
3 mov bx, 001Eh     ; BH=0, BL=1Eh
4 mov cx, 7         ; string length
5 mov dx, 0205h     ; row 2, col 5
▶ lea bp, msg       ; ES:BP → string
7 int 10h
8 ret
9 msg db 'Welcome'
```

The string's address must be in **ES:BP** (not DS:DX!). In a COM program ES = DS: BP = 0112h.

Step 5 / 7

VARIABLES

AX = **1301h** BX = **001Eh** CX = **0007h**

DX = **0205h** BP = **0112h**

OUTPUT

Code Trace: A Colored String with AH = 13h

```
1 org 100h
2 mov ax, 1301h     ; AH=13h, AL=01h
3 mov bx, 001Eh     ; BH=0, BL=1Eh
4 mov cx, 7         ; string length
5 mov dx, 0205h     ; row 2, col 5
6 lea bp, msg       ; ES:BP → string
▶ int 10h
8 ret
9 msg db 'Welcome'
```

Welcome is written from row 2, column 5; the cursor goes to (2,12).

Step 6 / 7

VARIABLES

AX = **1301h** BX = **001Eh** CX = **0007h**

DX = **0205h** BP = **0112h**

OUTPUT

```
Welcome
```

Code Trace: A Colored String with AH = 13h

```
1 org 100h
2 mov ax, 1301h     ; AH=13h, AL=01h
3 mov bx, 001Eh     ; BH=0, BL=1Eh
4 mov cx, 7         ; string length
5 mov dx, 0205h     ; row 2, col 5
6 lea bp, msg       ; ES:BP → string
7 int 10h
▶ ret
9 msg db 'Welcome'
```

With AL = 00h the string would still be written but the cursor would stay put. With AL = 02h/03h the string is made of character, attribute pairs.

Step 7 / 7

VARIABLES

AX = **1301h** BX = **001Eh** CX = **0007h**
DX = **0205h** BP = **0112h**

OUTPUT

```
Welcome
```

INT 10H VIDEO

AH = 06h: Scrolling a Window and Clearing the Screen

Register	Meaning
`AL`	number of lines to scroll; **0 = clear the window**
`BH`	attribute for the blanked lines
`CH`, `CL`	top-left corner (row, column)
`DH`, `DL`	bottom-right corner (row, column)

- AH = 07h does the same, scrolling down.
- The cursor stays put; after clearing, move it to (0,0) with 02h.

```
; clear the whole screen to blue
mov ax, 0600h   ; AH=06h, AL=0
mov bh, 1Fh     ; blue/white
mov cx, 0000h   ; (0,0)
mov dx, 184Fh   ; (24,79)
int 10h

; a 10×40 red box in the middle
mov ax, 0600h
mov bh, 4Fh
mov cx, 0714h   ; (7,20)
mov dx, 103Bh   ; (16,59)
int 10h
```

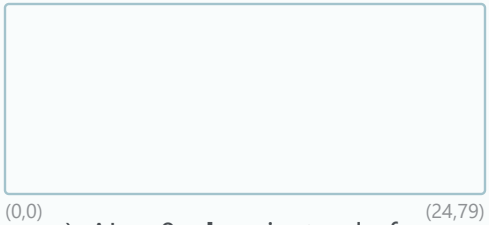
184Fh = row 24 (18h), column 79 (4Fh). Selecting mode 03h again also clears the screen, but resets the colors to the default (07h).

Clearing the Screen and a Window

```
1  org 100h
▶ mov ax, 0600h    ; AH=06h, AL=0
3  mov bh, 1Fh     ; blue/white
4  mov cx, 0000h   ; (0,0)
5  mov dx, 184Fh   ; (24,79)
6  int 10h
7  mov ax, 0600h
8  mov bh, 4Fh     ; red/white
9  mov cx, 0714h   ; (7,20)
10 mov dx, 103Bh   ; (16,59)
11 int 10h
12 ret
```

AX	BH	CX	DX
0600h	00h	0000h	0000h

CH,CL = (0, 0)
DH,DL = (0, 0)



AH = 06h (scroll window up), AL = 0: **clear** instead of scrolling.

Step 1 / 10

VARIABLES

AX = 0600h | BH = 00h | CX = 0000h | DX = 0000h

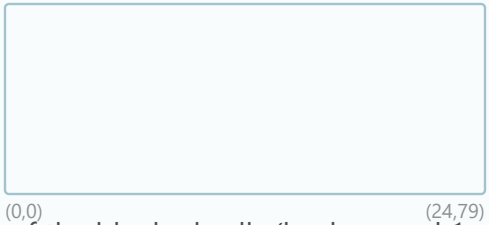
Register values come from running the code in the interpreter; the interpreter does not simulate the screen effect of 06h, so the screen is drawn from the BIOS definition.

Clearing the Screen and a Window

```
1  org 100h
2  mov ax, 0600h    ; AH=06h, AL=0
▶  mov bh, 1Fh      ; blue/white
4  mov cx, 0000h    ; (0,0)
5  mov dx, 184Fh    ; (24,79)
6  int 10h
7  mov ax, 0600h
8  mov bh, 4Fh      ; red/white
9  mov cx, 0714h    ; (7,20)
10 mov dx, 103Bh    ; (16,59)
11 int 10h
12 ret
```

AX	BH	CX	DX
0600h	1Fh	0000h	0000h

CH,CL = (0, 0)
DH,DL = (0, 0)



BH = 1Fh: the attribute of the blanked cells (background 1 blue, text F white).

Step 2 / 10

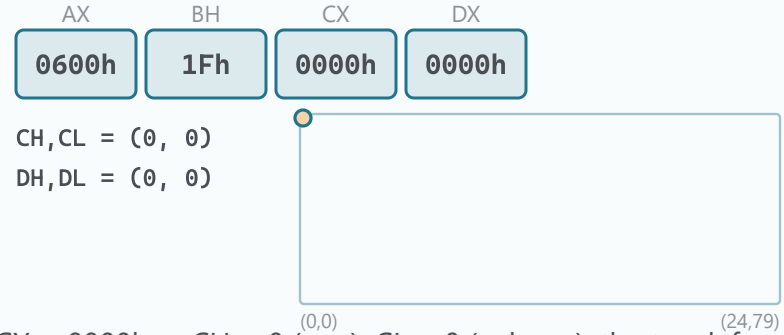
VARIABLES

AX = 0600h | BH = 1Fh | CX = 0000h | DX = 0000h

Register values come from running the code in the interpreter; the interpreter does not simulate the screen effect of 06h, so the screen is drawn from the BIOS definition.

Clearing the Screen and a Window

```
1  org 100h
2  mov ax, 0600h    ; AH=06h, AL=0
3  mov bh, 1Fh      ; blue/white
▶  mov cx, 0000h    ; (0,0)
5  mov dx, 184Fh    ; (24,79)
6  int 10h
7  mov ax, 0600h
8  mov bh, 4Fh      ; red/white
9  mov cx, 0714h    ; (7,20)
10 mov dx, 103Bh    ; (16,59)
11 int 10h
12 ret
```



CX = 0000h → CH = 0 (row), CL = 0 (column): the top-left corner.

Step 3 / 10

VARIABLES

AX = **0600h** BH = **1Fh** CX = **0000h** DX = **0000h**

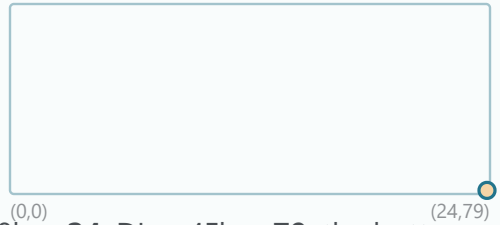
Register values come from running the code in the interpreter; the interpreter does not simulate the screen effect of 06h, so the screen is drawn from the BIOS definition.

Clearing the Screen and a Window

```
1  org 100h
2  mov ax, 0600h    ; AH=06h, AL=0
3  mov bh, 1Fh      ; blue/white
4  mov cx, 0000h    ; (0,0)
▶  mov dx, 184Fh    ; (24,79)
6  int 10h
7  mov ax, 0600h
8  mov bh, 4Fh      ; red/white
9  mov cx, 0714h    ; (7,20)
10 mov dx, 103Bh    ; (16,59)
11 int 10h
12 ret
```

AX	BH	CX	DX
0600h	1Fh	0000h	184Fh

CH,CL = (0, 0)
DH,DL = (24, 79)



DX = 184Fh → DH = 18h = **24**, DL = 4Fh = **79**: the bottom-right corner.

Step 4 / 10

VARIABLES

AX = **0600h** BH = **1Fh** CX = **0000h** DX = **184Fh**

Register values come from running the code in the interpreter; the interpreter does not simulate the screen effect of 06h, so the screen is drawn from the BIOS definition.

Clearing the Screen and a Window

```
1  org 100h
2  mov ax, 0600h    ; AH=06h, AL=0
3  mov bh, 1Fh      ; blue/white
4  mov cx, 0000h    ; (0,0)
5  mov dx, 184Fh    ; (24,79)
▶  int 10h
7  mov ax, 0600h
8  mov bh, 4Fh      ; red/white
9  mov cx, 0714h    ; (7,20)
10 mov dx, 103Bh    ; (16,59)
11 int 10h
12 ret
```

AX	BH	CX	DX
0600h	1Fh	0000h	184Fh

CH,CL = (0, 0)
DH,DL = (24, 79)



INT 10h: the whole 80 × 25 screen is cleared to blue. The cursor stays put.

Step 5 / 10

VARIABLES

AX = 0600h · BH = 1Fh · CX = 0000h · DX = 184Fh

Register values come from running the code in the interpreter; the interpreter does not simulate the screen effect of 06h, so the screen is drawn from the BIOS definition.

Clearing the Screen and a Window

```
1  org 100h
2  mov ax, 0600h    ; AH=06h, AL=0
3  mov bh, 1Fh      ; blue/white
4  mov cx, 0000h    ; (0,0)
5  mov dx, 184Fh    ; (24,79)
6  int 10h
▶  mov ax, 0600h
▶  mov bh, 4Fh      ; red/white
9  mov cx, 0714h    ; (7,20)
10 mov dx, 103Bh    ; (16,59)
11 int 10h
12 ret
```

AX	BH	CX	DX
0600h	4Fh	0000h	184Fh

CH,CL = (0, 0)
DH,DL = (24, 79)



Second call: AX = 0600h, BH = 4Fh (white on red).

Step 6 / 10

VARIABLES

AX = 0600h | BH = 4Fh | CX = 0000h | DX = 184Fh

Register values come from running the code in the interpreter; the interpreter does not simulate the screen effect of 06h, so the screen is drawn from the BIOS definition.

Clearing the Screen and a Window

```
1  org 100h
2  mov ax, 0600h    ; AH=06h, AL=0
3  mov bh, 1Fh      ; blue/white
4  mov cx, 0000h    ; (0,0)
5  mov dx, 184Fh    ; (24,79)
6  int 10h
7  mov ax, 0600h
8  mov bh, 4Fh      ; red/white
▶  mov cx, 0714h    ; (7,20)
10 mov dx, 103Bh    ; (16,59)
11 int 10h
12 ret
```

AX	BH	CX	DX
0600h	4Fh	0714h	184Fh

CH,CL = (7, 20)
DH,DL = (24, 79)



CX = 0714h → (7, 20).

Step 7 / 10

VARIABLES

AX = 0600h | BH = 4Fh | CX = 0714h | DX = 184Fh

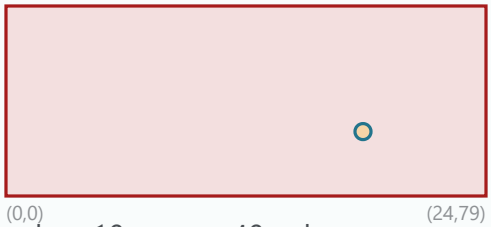
Register values come from running the code in the interpreter; the interpreter does not simulate the screen effect of 06h, so the screen is drawn from the BIOS definition.

Clearing the Screen and a Window

```
1  org 100h
2  mov ax, 0600h    ; AH=06h, AL=0
3  mov bh, 1Fh      ; blue/white
4  mov cx, 0000h    ; (0,0)
5  mov dx, 184Fh    ; (24,79)
6  int 10h
7  mov ax, 0600h
8  mov bh, 4Fh      ; red/white
9  mov cx, 0714h    ; (7,20)
▶ mov dx, 103Bh     ; (16,59)
11 int 10h
12 ret
```

AX	BH	CX	DX
0600h	4Fh	0714h	103Bh

CH,CL = (7, 20)
DH,DL = (16, 59)



DX = 103Bh → (16, 59): a box 10 rows × 40 columns.

Step 8 / 10

VARIABLES

AX = 0600h BH = 4Fh CX = 0714h DX = 103Bh

Register values come from running the code in the interpreter; the interpreter does not simulate the screen effect of 06h, so the screen is drawn from the BIOS definition.

Clearing the Screen and a Window

```
1  org 100h
2  mov ax, 0600h    ; AH=06h, AL=0
3  mov bh, 1Fh      ; blue/white
4  mov cx, 0000h    ; (0,0)
5  mov dx, 184Fh    ; (24,79)
6  int 10h
7  mov ax, 0600h
8  mov bh, 4Fh      ; red/white
9  mov cx, 0714h    ; (7,20)
10 mov dx, 103Bh    ; (16,59)
▶  int 10h
12 ret
```

AX	BH	CX	DX
0600h	4Fh	0714h	103Bh

CH,CL = (7, 20)
DH,DL = (16, 59)



INT 10h: only this window is cleared to red; the rest stays blue.

Step 9 / 10

VARIABLES

AX = 0600h | BH = 4Fh | CX = 0714h | DX = 103Bh

Register values come from running the code in the interpreter; the interpreter does not simulate the screen effect of 06h, so the screen is drawn from the BIOS definition.

Clearing the Screen and a Window

```
1  org 100h
2  mov ax, 0600h    ; AH=06h, AL=0
3  mov bh, 1Fh      ; blue/white
4  mov cx, 0000h    ; (0,0)
5  mov dx, 184Fh    ; (24,79)
6  int 10h
7  mov ax, 0600h
8  mov bh, 4Fh      ; red/white
9  mov cx, 0714h    ; (7,20)
10 mov dx, 103Bh    ; (16,59)
11 int 10h
▶  ret
```

AX	BH	CX	DX
0600h	4Fh	0714h	103Bh

CH,CL = (7, 20)
DH,DL = (16, 59)



RET: the program ends.

Step 10 / 10

VARIABLES

AX = 0600h | BH = 4Fh | CX = 0714h | DX = 103Bh

Register values come from running the code in the interpreter; the interpreter does not simulate the screen effect of 06h, so the screen is drawn from the BIOS definition.

Output Without the BIOS: B800h Video Memory

- In mode 03h the screen is held in 4000 bytes of memory starting at **B800:0000h** (80 × 25 × 2).
- Each cell is 2 bytes: **character** (even address) + **attribute** (odd address).
- The video card shows whatever is written there at once: much faster than an interrupt call.
- Written as a word it is little-endian: `4E41h` → 41h ('A') at the low address, 4Eh as the attribute.

```
; yellow-on-red 'A' at cell (1,4)
mov ax, 0B800h
mov es, ax
mov di, (1*80+4)*2  ; 00A8h
mov ax, 4E41h
mov es:[di], ax

; fill the screen with '.'
xor di, di
mov ax, 072Eh
mov cx, 2000
cld
rep stosw
```



B800:00A8 · char 'A' B800:00A9 · attribute

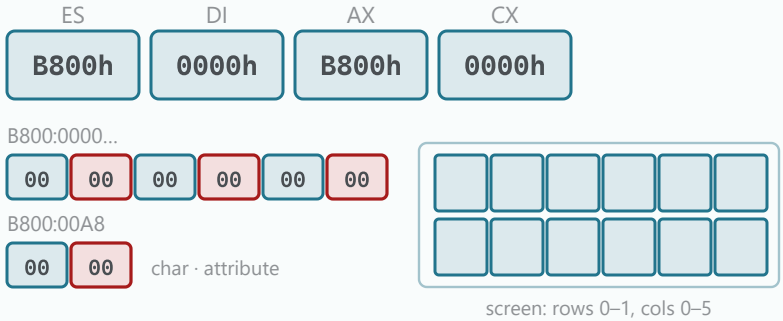
`mov es:[di], ax (AX = 4E41h)`

low byte 41h → even address (char)

Writing directly does **not move** the cursor and does not update the cursor position the BIOS keeps. emu8086 supports this memory too.

Writing Directly to Video Memory

```
1  org 100h
▶  mov ax, 0B800h
▶  mov es, ax
4  mov di, (1*80+4)*2   ; 00A8h
5  mov ax, 4E41h
6  mov es:[di], ax
7  xor di, di
8  mov ax, 072Eh
9  mov cx, 3            ; first 3 cells
10 cld
11 rep stosw
12 ret
```



ES = **B800h**: the segment of the mode 03h text screen. A segment register cannot take an immediate, so it goes through AX.

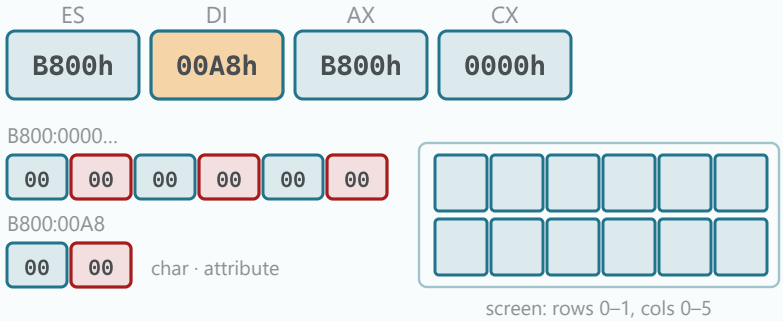
Step 1 / 9

VARIABLES

`ES = B800h` `DI = 0000h` `AX = B800h` `CX = 0000h`

Writing Directly to Video Memory

```
1  org 100h
2  mov ax, 0B800h
3  mov es, ax
▶  mov di, (1*80+4)*2  ; 00A8h
5  mov ax, 4E41h
6  mov es:[di], ax
7  xor di, di
8  mov ax, 072Eh
9  mov cx, 3           ; first 3 cells
10 cld
11 rep stosw
12 ret
```



Cell (1, 4): (1 × 80 + 4) × 2 = **168 = 00A8h**. The product is computed at assembly time.

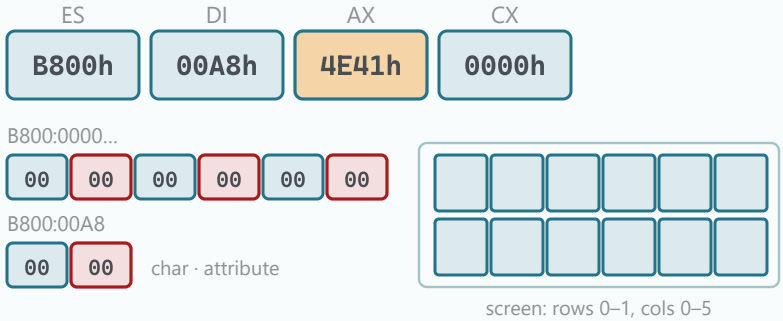
Step 2 / 9

VARIABLES

ES = B800h | DI = 00A8h | AX = B800h | CX = 0000h

Writing Directly to Video Memory

```
1  org 100h
2  mov ax, 0B800h
3  mov es, ax
4  mov di, (1*80+4)*2   ; 00A8h
▶  mov ax, 4E41h
6  mov es:[di], ax
7  xor di, di
8  mov ax, 072Eh
9  mov cx, 3            ; first 3 cells
10 cld
11 rep stosw
12 ret
```



AX = 4E41h: AL = 41h ('A', the character), AH = 4Eh (attribute: yellow on red).

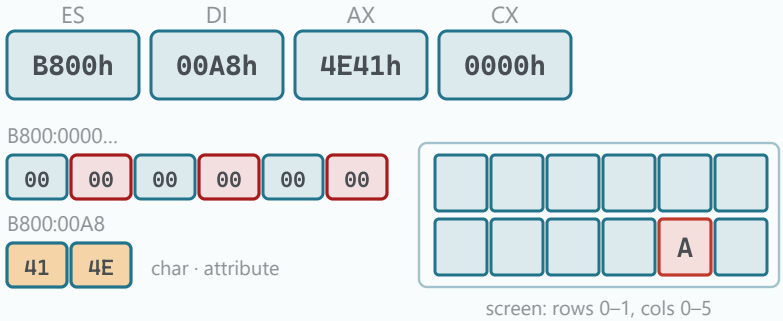
Step 3 / 9

VARIABLES

ES = B800h | DI = 00A8h | AX = 4E41h | CX = 0000h

Writing Directly to Video Memory

```
1  org 100h
2  mov ax, 0B800h
3  mov es, ax
4  mov di, (1*80+4)*2   ; 00A8h
5  mov ax, 4E41h
▶  mov es:[di], ax
7  xor di, di
8  mov ax, 072Eh
9  mov cx, 3            ; first 3 cells
10 cld
11 rep stosw
12 ret
```



MOV ES:[DI], AX: little-endian → B80A8h ← **41h**, B80A9h ← **4Eh**. The character appears at once; the cursor stays put.

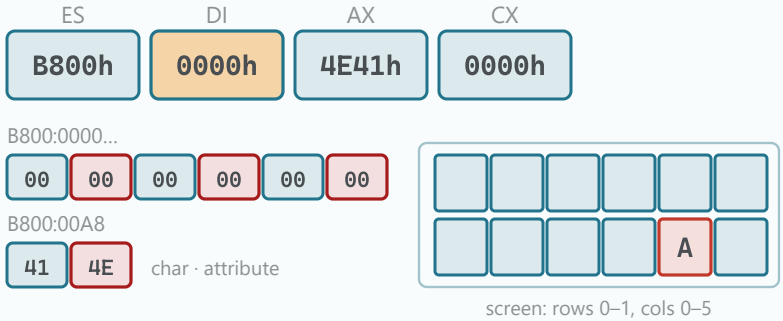
Step 4 / 9

VARIABLES

ES = **B800h** DI = **00A8h** AX = **4E41h** CX = **0000h**

Writing Directly to Video Memory

```
1  org 100h
2  mov ax, 0B800h
3  mov es, ax
4  mov di, (1*80+4)*2   ; 00A8h
5  mov ax, 4E41h
6  mov es:[di], ax
▶  xor di, di
8  mov ax, 072Eh
9  mov cx, 3            ; first 3 cells
10 cld
11 rep stosw
12 ret
```



DI = 0: the top-left corner.

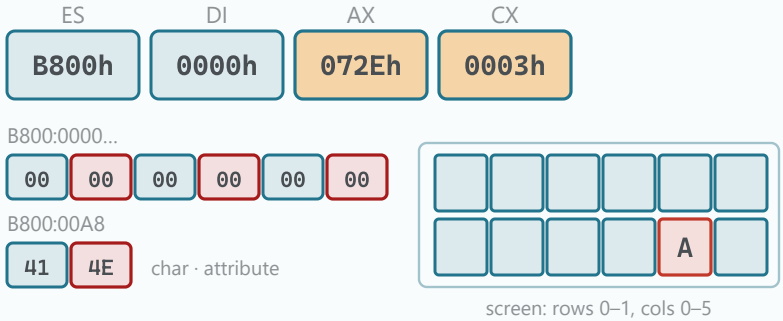
Step 5 / 9

VARIABLES

ES = B800h | DI = 0000h | AX = 4E41h | CX = 0000h

Writing Directly to Video Memory

```
1  org 100h
2  mov ax, 0B800h
3  mov es, ax
4  mov di, (1*80+4)*2   ; 00A8h
5  mov ax, 4E41h
6  mov es:[di], ax
7  xor di, di
▶  mov ax, 072Eh
▶  mov cx, 3            ; first 3 cells
10 cld
11 rep stosw
12 ret
```



AX = 072Eh ('.' + 07h light gray), CX = 3.

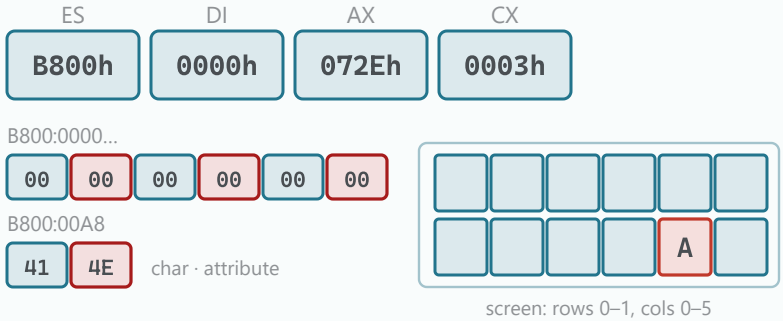
Step 6 / 9

VARIABLES

ES = B800h DI = 0000h AX = 072Eh CX = 0003h

Writing Directly to Video Memory

```
1  org 100h
2  mov ax, 0B800h
3  mov es, ax
4  mov di, (1*80+4)*2   ; 00A8h
5  mov ax, 4E41h
6  mov es:[di], ax
7  xor di, di
8  mov ax, 072Eh
9  mov cx, 3            ; first 3 cells
▶ cld
11 rep stosw
12 ret
```



CLD: DF = 0, DI counts up.

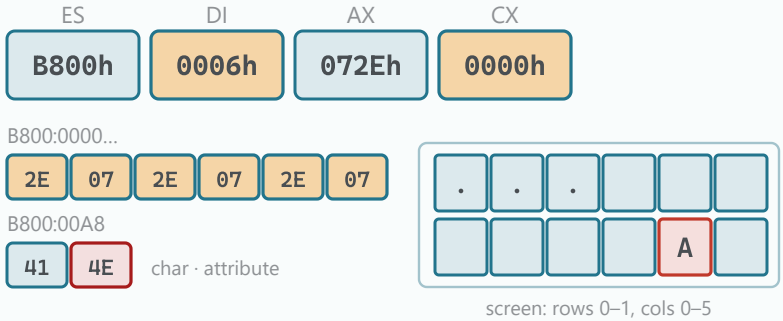
Step 7 / 9

VARIABLES

ES = B800h | DI = 0000h | AX = 072Eh | CX = 0003h

Writing Directly to Video Memory

```
1  org 100h
2  mov ax, 0B800h
3  mov es, ax
4  mov di, (1*80+4)*2   ; 00A8h
5  mov ax, 4E41h
6  mov es:[di], ax
7  xor di, di
8  mov ax, 072Eh
9  mov cx, 3            ; first 3 cells
10 cld
▶ rep stosw
12 ret
```



REP STOSW: ES:[DI] ← AX, DI += 2, CX times → the first 3 cells become '.'; DI = 0006h, CX = 0. (For the whole screen, CX = 2000.)

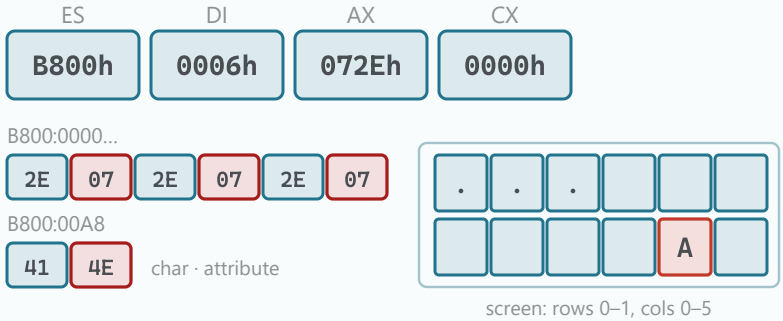
Step 8 / 9

VARIABLES

ES = B800h | DI = 0006h | AX = 072Eh | CX = 0000h

Writing Directly to Video Memory

```
1  org 100h
2  mov ax, 0B800h
3  mov es, ax
4  mov di, (1*80+4)*2  ; 00A8h
5  mov ax, 4E41h
6  mov es:[di], ax
7  xor di, di
8  mov ax, 072Eh
9  mov cx, 3           ; first 3 cells
10 cld
11 rep stosw
▶  ret
```



RET: the program ends.

Step 9 / 9

VARIABLES

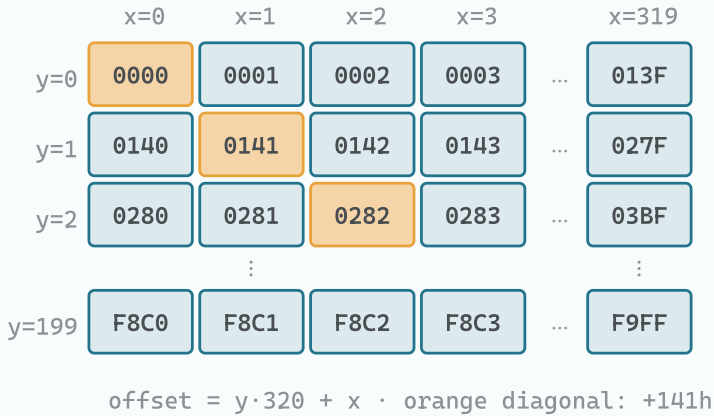
ES = B800h DI = 0006h AX = 072Eh CX = 0000h

Mode 13h: 320×200, 256 Colors

- `mov ax, 0013h` / `int 10h` : 320 × 200 graphics mode. Video memory is **A000:0000h – A000:F9FFh** (64,000 bytes).
- Each pixel is **1 byte**: a color number (palette entry). The first 16 colors match the text colors: 0Eh yellow, 0Fh white.
- 320 = **140h** bytes per row: **offset = y · 320 + x**.
- Example: (100, 50) → 50 · 320 + 100 = 16,100 = **3EE4h**.

INT 10h / AH = 0Ch: write pixel	
`AL`	color
`BH`	page (0 in mode 13h)
`CX` / `DX`	x (column 0–319) / y (row 0–199)

AH = 0Dh, with the same CX, DX, reads the pixel's color into AL.



A Diagonal Line: Through the BIOS and Through A000h

INT 10h / 0Ch: by coordinates

```
mov ax, 0013h
int 10h          ; 320×200, 256 colors
xor cx, cx       ; x = 0
xor dx, dx       ; y = 0
mov bh, 0        ; page
dot: mov ax, 0C0Eh   ; AH=0Ch, AL=yellow
int 10h
inc cx           ; one right
inc dx           ; one down
cmp dx, 200
jb  dot           ; 200 pixels
```

Why 141h? +140h (320) goes to the **same** column one row down, +1 one column right: 141h = 321. The last pixel (199, 199) is at 199 · 321 = F987h. For the anti-diagonal from the top right, add 13Fh (320 − 1).

A000h memory: by offset

```
mov ax, 0013h
int 10h
mov ax, 0A000h
mov es, ax
xor di, di       ; (0,0) → offset 0
mov al, 0Eh      ; yellow
mov cx, 200
draw: mov es:[di], al
add di, 141h     ; 320 + 1
loop draw
mov ah, 0
int 16h          ; wait for a key
mov ax, 0003h
int 10h          ; back to text mode
```

Writing to memory directly is much faster than an interrupt call per pixel.

emu8086's graphics support is limited: 0Ch draws slowly, and direct writes to A000h may not show in every version; in DOSBox they show fully.

Drawing a Diagonal into Mode 13h Memory

```
1  org 100h
▶  mov ax, 0A000h
▶  mov es, ax
4  xor di, di          ; (0,0) → offset 0
5  mov al, 0Eh         ; yellow
6  mov cx, 4           ; first 4 pixels
7  draw: mov es:[di], al
8  add di, 141h        ; 320 + 1
9  loop draw
10 ret
```

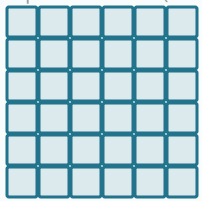
ES	DI	CX	AL
A000h	0000h	0000h	00h

DI = 0 → y = 0, x = 0

320 = 140h bytes per row

+141h = one down, one right

top left of screen (6 × 6 pixels)



ES = **A000h**: mode 13h video memory (320 × 200, 1 byte per pixel).

Step 1 / 8

VARIABLES

ES = **A000h** DI = **0000h** CX = **0000h**

Drawing a Diagonal into Mode 13h Memory

```
1  org 100h
2  mov ax, 0A000h
3  mov es, ax
▶  xor di, di          ; (0,0) → offset 0
5  mov al, 0Eh         ; yellow
6  mov cx, 4           ; first 4 pixels
7  draw: mov es:[di], al
8  add di, 141h        ; 320 + 1
9  loop draw
10 ret
```

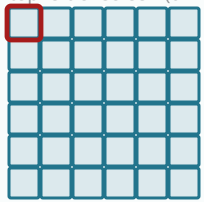
ES	DI	CX	AL
A000h	0000h	0000h	00h

DI = 0 → y = 0, x = 0

320 = 140h bytes per row

+141h = one down, one right

top left of screen (6 × 6 pixels)



DI = 0: pixel (x, y) = (0, 0), offset y · 320 + x = 0.

Step 2 / 8

VARIABLES

ES = A000h DI = 0000h CX = 0000h

Drawing a Diagonal into Mode 13h Memory

```
1  org 100h
2  mov ax, 0A000h
3  mov es, ax
4  xor di, di          ; (0,0) → offset 0
▶  mov al, 0Eh         ; yellow
▶  mov cx, 4           ; first 4 pixels
7  draw: mov es:[di], al
8  add di, 141h        ; 320 + 1
9  loop draw
10 ret
```

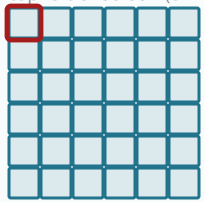
ES	DI	CX	AL
A000h	0000h	0004h	0Eh

DI = 0 → y = 0, x = 0

320 = 140h bytes per row

+141h = one down, one right

top left of screen (6 × 6 pixels)



AL = 0Eh (yellow), CX = 4: draw the first 4 pixels.

Step 3 / 8

VARIABLES

ES = A000h | DI = 0000h | CX = 0004h

Drawing a Diagonal into Mode 13h Memory

```
1  org 100h
2  mov ax, 0A000h
3  mov es, ax
4  xor di, di        ; (0,0) → offset 0
5  mov al, 0Eh       ; yellow
6  mov cx, 4         ; first 4 pixels
▶  draw: mov es:[di], al
▶  add di, 141h      ; 320 + 1
▶  loop draw
10 ret
```

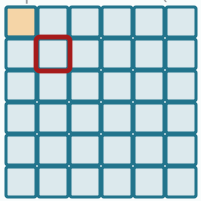
ES	DI	CX	AL
A000h	0141h	0003h	0Eh

DI = 321 → y = 1, x = 1

320 = 140h bytes per row

+141h = one down, one right

top left of screen (6 × 6 pixels)



ES:[0000h] ← 0Eh: pixel (0, 0). ADD DI, 141h → DI = 0141h (one row down + one right). LOOP: CX = 3.

Step 4 / 8

VARIABLES

ES = A000h DI = 0141h CX = 0003h

Drawing a Diagonal into Mode 13h Memory

```
1  org 100h
2  mov ax, 0A000h
3  mov es, ax
4  xor di, di        ; (0,0) → offset 0
5  mov al, 0Eh       ; yellow
6  mov cx, 4         ; first 4 pixels
▶  draw: mov es:[di], al
▶  add di, 141h      ; 320 + 1
▶  loop draw
10 ret
```

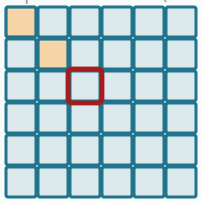
ES	DI	CX	AL
A000h	0282h	0002h	0Eh

DI = 642 → y = 2, x = 2

320 = 140h bytes per row

+141h = one down, one right

top left of screen (6 × 6 pixels)



ES:[0141h] ← 0Eh: pixel (1, 1). ADD DI, 141h → DI = 0282h (one row down + one right). LOOP: CX = 2.

Step 5 / 8

VARIABLES

ES = A000h DI = 0282h CX = 0002h

Drawing a Diagonal into Mode 13h Memory

```
1  org 100h
2  mov ax, 0A000h
3  mov es, ax
4  xor di, di        ; (0,0) → offset 0
5  mov al, 0Eh       ; yellow
6  mov cx, 4         ; first 4 pixels
▶  draw: mov es:[di], al
▶  add di, 141h      ; 320 + 1
▶  loop draw
10 ret
```

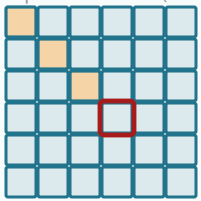
ES	DI	CX	AL
A000h	03C3h	0001h	0Eh

DI = 963 → y = 3, x = 3

320 = 140h bytes per row

+141h = one down, one right

top left of screen (6 × 6 pixels)



ES:[0282h] ← 0Eh: pixel (2, 2). ADD DI, 141h → DI = 03C3h (one row down + one right). LOOP: CX = 1.

Step 6 / 8

VARIABLES

ES = **A000h** DI = **03C3h** CX = **0001h**

Drawing a Diagonal into Mode 13h Memory

```
1  org 100h
2  mov ax, 0A000h
3  mov es, ax
4  xor di, di        ; (0,0) → offset 0
5  mov al, 0Eh       ; yellow
6  mov cx, 4         ; first 4 pixels
▶  draw: mov es:[di], al
▶  add di, 141h      ; 320 + 1
▶  loop draw
10 ret
```

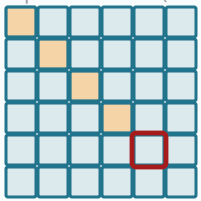
ES	DI	CX	AL
A000h	0504h	0000h	0Eh

DI = 1284 → y = 4, x = 4

320 = 140h bytes per row

+141h = one down, one right

top left of screen (6 × 6 pixels)



ES:[03C3h] ← 0Eh: pixel (3, 3). ADD DI, 141h → DI = 0504h (one row down + one right). LOOP: CX = 0. The loop ends.

Step 7 / 8

VARIABLES

ES = A000h | DI = 0504h | CX = 0000h

Drawing a Diagonal into Mode 13h Memory

```
1 org 100h
2 mov ax, 0A000h
3 mov es, ax
4 xor di, di        ; (0,0) → offset 0
5 mov al, 0Eh       ; yellow
6 mov cx, 4         ; first 4 pixels
7 draw: mov es:[di], al
8 add di, 141h      ; 320 + 1
9 loop draw
▶ ret
```

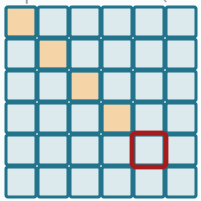
ES	DI	CX	AL
A000h	0504h	0000h	0Eh

DI = 1284 → y = 4, x = 4

320 = 140h bytes per row

+141h = one down, one right

top left of screen (6 × 6 pixels)



RET. With 200 passes the last pixel (199, 199) would be at 199 · 321 = F987h.

Step 8 / 8

VARIABLES

ES = **A000h** DI = **0504h** CX = **0000h**

INT 10H VIDEO

Pixel Address Explorer (320 × 200)



Diagonal: DI += 141h | Anti-diagonal: DI += 13Fh | Clear

(100, 50) → 50 · 320 + 100 = **3EE4h** · A000:3EE4 = physical **A3EE4h**

Pick a pixel or draw a diagonal to compute the address.

Click the screen to compute the offset y · 320 + x and the physical address of (x, y).

INT 16h: BIOS Keyboard Services

AH	Service	Output
`00h`	**wait** for a key and read it	AH = scan code, AL = ASCII; the key is removed from the buffer
`01h`	**is** a key ready? (no wait)	ZF = 1: none · ZF = 0: AX = key, **not removed** from the buffer
`02h`	shift states	AL bits: 0 right Shift, 1 left Shift, 2 Ctrl, 3 Alt, 4 Scroll, 5 Num, 6 Caps, 7 Insert

Key	AH (scan)	AL (ASCII)
a / A	1Eh	61h / 41h
1	02h	31h
Enter	1Ch	0Dh
Esc	01h	1Bh
Space	39h	20h

Key	AH (scan)	AL
↑ / ↓	48h / 50h	00h
← / →	4Bh / 4Dh	00h
F1 … F10	3Bh … 44h	00h
Backspace	0Eh	08h
Ctrl+C	2Eh	03h

The scan code is the key's **physical** position: 'a' and 'A' give the same scan code. Keys with no ASCII code (extended keys) return AL = 00h; the key is identified by AH. A `MOV AH, 0` · `INT 16h` at the end of a program waits for a key so the output can be read before the screen closes.

Keyboard Explorer: What Does INT 16h / AH = 00h Return?

click here and press a key

AH (scan code)	AL (ASCII)	AX
--	--	----

Click the field and press a key to see AH, AL and AX.

Letters, digits, arrow keys, F1–F10, Enter, Esc and Ctrl+letter are recognized. Values follow the IBM PC/AT (US) keyboard layout.

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
▶  mov ah, 0          ; wait for key
4  int 16h            ; AH=scan, AL=ASCII
5  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

AH = 00h: **wait** for a key from the keyboard buffer and take it.

Step 1 / 26

VARIABLES

AH = **00h** AL = **00h** CF = **0** ZF = **0**

OUTPUT

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0          ; wait for key
▶  int 16h            ; AH=scan, AL=ASCII
5  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

The user pressed **O** (Shift+o): AL = 4Fh (ASCII), AH = **18h** (scan code).

Step 2 / 26

VARIABLES

AH = **18h** AL = **4Fh** CF = **0** ZF = **0**

OUTPUT

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0          ; wait for key
4  int 16h            ; AH=scan, AL=ASCII
▶  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

4Fh ≠ 0Dh.

Step 3 / 26

VARIABLES

AH = **18h** AL = **4Fh** CF = **0** ZF = **0**

OUTPUT

Code Trace: Read Until Enter, Echo in Upper Case

```
 1 org 100h
 2 read:
 3 mov ah, 0          ; wait for key
 4 int 16h            ; AH=scan, AL=ASCII
 5 cmp al, 0Dh        ; Enter?
▶6 je done
 7 cmp al, 'a'
 8 jb echo
 9 cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

Not Enter; carry on.

Step 4 / 26

VARIABLES

AH = **18h** AL = **4Fh** CF = **0** ZF = **0**

OUTPUT

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0          ; wait for key
4  int 16h            ; AH=scan, AL=ASCII
5  cmp al, 0Dh        ; Enter?
6  je done
▶  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

'O' (4Fh) < 'a' (61h) → CF = 1.

Step 5 / 26

VARIABLES

AH = **18h** AL = **4Fh** CF = **1** ZF = **0**

OUTPUT

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0        ; wait for key
4  int 16h          ; AH=scan, AL=ASCII
5  cmp al, 0Dh      ; Enter?
6  je done
7  cmp al, 'a'
▶  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h      ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h          ; print it
15 jmp read
16 done:
17 ret
```

Not lower case → straight to `echo`.

Step 6 / 26

VARIABLES

AH = **18h** AL = **4Fh** CF = **1** ZF = **0**

OUTPUT

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0        ; wait for key
4  int 16h          ; AH=scan, AL=ASCII
5  cmp al, 0Dh      ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h      ; lower → upper
12 echo:
▶ mov ah, 0Eh
14 int 10h          ; print it
15 jmp read
16 done:
17 ret
```

AH = 0Eh (teletype). AL holds the character to print.

Step 7 / 26

VARIABLES

AH = 0Eh | AL = 4Fh | CF = 1 | ZF = 0

OUTPUT

Code Trace: Read Until Enter, Echo in Upper Case

```
 1 org 100h
 2 read:
 3 mov ah, 0          ; wait for key
 4 int 16h            ; AH=scan, AL=ASCII
 5 cmp al, 0Dh        ; Enter?
 6 je done
 7 cmp al, 'a'
 8 jb echo
 9 cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
▶ int 10h             ; print it
15 jmp read
16 done:
17 ret
```

O on the screen.

Step 8 / 26

VARIABLES

AH = **0Eh** AL = **4Fh** CF = **1** ZF = **0**

OUTPUT

```
O
```

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0          ; wait for key
4  int 16h            ; AH=scan, AL=ASCII
5  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
▶ jmp read
16 done:
17 ret
```

Back for the next key.

Step 9 / 26

VARIABLES

AH = **0Eh** AL = **4Fh** CF = **1** ZF = **0**

OUTPUT

```
O
```

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
▶  mov ah, 0          ; wait for key
4  int 16h            ; AH=scan, AL=ASCII
5  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

AH = 00h: **wait** for a key from the keyboard buffer and take it.

Step 10 / 26

VARIABLES

AH = 00h | AL = 4Fh | CF = 1 | ZF = 0

OUTPUT

```
O
```

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0          ; wait for key
▶  int 16h            ; AH=scan, AL=ASCII
5  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

The user pressed **k**: AL = 6Bh, AH = **25h**.

Step 11 / 26

VARIABLES

AH = **25h** AL = **6Bh** CF = **1** ZF = **0**

OUTPUT

```
0
```

Code Trace: Read Until Enter, Echo in Upper Case

```
 1 org 100h
 2 read:
 3 mov ah, 0          ; wait for key
 4 int 16h            ; AH=scan, AL=ASCII
 ▶ cmp al, 0Dh        ; Enter?
 6 je done
 7 cmp al, 'a'
 8 jb echo
 9 cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

6Bh ≠ 0Dh.

Step 12 / 26

VARIABLES

AH = **25h** AL = **6Bh** CF = **0** ZF = **0**

OUTPUT

```
0
```

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0        ; wait for key
4  int 16h          ; AH=scan, AL=ASCII
5  cmp al, 0Dh      ; Enter?
▶  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h      ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h          ; print it
15 jmp read
16 done:
17 ret
```

Carry on.

Step 13 / 26

VARIABLES

AH = **25h** AL = **6Bh** CF = **0** ZF = **0**

OUTPUT

```
0
```

Code Trace: Read Until Enter, Echo in Upper Case

```
 1 org 100h
 2 read:
 3 mov ah, 0          ; wait for key
 4 int 16h            ; AH=scan, AL=ASCII
 5 cmp al, 0Dh        ; Enter?
 6 je done
▶ cmp al, 'a'
 8 jb echo
 9 cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

'k' ≥ 'a' → CF = 0.

Step 14 / 26

VARIABLES

AH = **25h** AL = **6Bh** CF = **0** ZF = **0**

OUTPUT

```
0
```

Code Trace: Read Until Enter, Echo in Upper Case

```
 1 org 100h
 2 read:
 3 mov ah, 0          ; wait for key
 4 int 16h            ; AH=scan, AL=ASCII
 5 cmp al, 0Dh        ; Enter?
 6 je done
 7 cmp al, 'a'
▶ jb echo
 9 cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

May be lower case; carry on.

Step 15 / 26

VARIABLES

AH = **25h** AL = **6Bh** CF = **0** ZF = **0**

OUTPUT

```
0
```

Code Trace: Read Until Enter, Echo in Upper Case

```
 1 org 100h
 2 read:
 3 mov ah, 0          ; wait for key
 4 int 16h            ; AH=scan, AL=ASCII
 5 cmp al, 0Dh        ; Enter?
 6 je done
 7 cmp al, 'a'
 8 jb echo
▶ cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

6Bh − 7Ah: borrow → **CF = 1**, ZF = 0. So 'k' < 'z'.

Step 16 / 26

VARIABLES

AH = **25h** AL = **6Bh** CF = **1** ZF = **0**

OUTPUT

```
0
```

Code Trace: Read Until Enter, Echo in Upper Case

```
 1 org 100h
 2 read:
 3 mov ah, 0          ; wait for key
 4 int 16h            ; AH=scan, AL=ASCII
 5 cmp al, 0Dh        ; Enter?
 6 je done
 7 cmp al, 'a'
 8 jb echo
 9 cmp al, 'z'
▶10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

`ja` needs CF = 0 and ZF = 0; CF = 1 → no jump: lower case.

Step 17 / 26

VARIABLES

AH = **25h** AL = **6Bh** CF = **1** ZF = **0**

OUTPUT

```
0
```

Code Trace: Read Until Enter, Echo in Upper Case

```
 1 org 100h
 2 read:
 3 mov ah, 0          ; wait for key
 4 int 16h            ; AH=scan, AL=ASCII
 5 cmp al, 0Dh        ; Enter?
 6 je done
 7 cmp al, 'a'
 8 jb echo
 9 cmp al, 'z'
10 ja echo
▶ sub al, 20h         ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

Bit 5 is cleared: 6Bh − 20h = **4Bh = 'K'**.

Step 18 / 26

VARIABLES

AH = **25h** AL = **4Bh** CF = **0** ZF = **0**

OUTPUT

```
0
```

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0          ; wait for key
4  int 16h            ; AH=scan, AL=ASCII
5  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
▶ mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

AH = 0Eh (teletype). AL holds the character to print.

Step 19 / 26

VARIABLES

AH = 0Eh | AL = 4Bh | CF = 0 | ZF = 0

OUTPUT

```
0
```

Code Trace: Read Until Enter, Echo in Upper Case

```
 1 org 100h
 2 read:
 3 mov ah, 0          ; wait for key
 4 int 16h            ; AH=scan, AL=ASCII
 5 cmp al, 0Dh        ; Enter?
 6 je done
 7 cmp al, 'a'
 8 jb echo
 9 cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
▶ int 10h             ; print it
15 jmp read
16 done:
17 ret
```

K on the screen.

Step 20 / 26

VARIABLES

AH = **0Eh** AL = **4Bh** CF = **0** ZF = **0**

OUTPUT

```
OK
```

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0          ; wait for key
4  int 16h            ; AH=scan, AL=ASCII
5  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
▶ jmp read
16 done:
17 ret
```

Back for the next key.

Step 21 / 26

VARIABLES

AH = **0Eh** AL = **4Bh** CF = **0** ZF = **0**

OUTPUT

```
OK
```

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
▶  mov ah, 0          ; wait for key
4  int 16h            ; AH=scan, AL=ASCII
5  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

AH = 00h: **wait** for a key from the keyboard buffer and take it.

Step 22 / 26

VARIABLES

AH = 00h | AL = 4Bh | CF = 0 | ZF = 0

OUTPUT

```
OK
```

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0          ; wait for key
▶  int 16h            ; AH=scan, AL=ASCII
5  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

The user pressed **Enter**: AL = 0Dh, AH = **1Ch**.

Step 23 / 26

VARIABLES

AH = **1Ch** AL = **0Dh** CF = **0** ZF = **0**

OUTPUT

```
OK
```

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0          ; wait for key
4  int 16h            ; AH=scan, AL=ASCII
▶  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

AL = 0Dh → ZF = 1.

Step 24 / 26

VARIABLES

AH = **1Ch** AL = **0Dh** CF = **0** ZF = **1**

OUTPUT

```
OK
```

Code Trace: Read Until Enter, Echo in Upper Case

```
 1 org 100h
 2 read:
 3 mov ah, 0          ; wait for key
 4 int 16h            ; AH=scan, AL=ASCII
 5 cmp al, 0Dh        ; Enter?
▶6 je done
 7 cmp al, 'a'
 8 jb echo
 9 cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
17 ret
```

Enter → `done`.

Step 25 / 26

VARIABLES

`AH = 1Ch` `AL = 0Dh` `CF = 0` `ZF = 1`

OUTPUT

```
OK
```

Code Trace: Read Until Enter, Echo in Upper Case

```
1  org 100h
2  read:
3  mov ah, 0          ; wait for key
4  int 16h            ; AH=scan, AL=ASCII
5  cmp al, 0Dh        ; Enter?
6  je done
7  cmp al, 'a'
8  jb echo
9  cmp al, 'z'
10 ja echo
11 sub al, 20h        ; lower → upper
12 echo:
13 mov ah, 0Eh
14 int 10h            ; print it
15 jmp read
16 done:
▶ ret
```

The program ends at Enter. INT 16h/00h does **not echo** the key; the program did the echoing.

Step 26 / 26

VARIABLES

AH = **1Ch** AL = **0Dh** CF = **0** ZF = **1**

OUTPUT

```
OK
```

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
▶ xor dx, dx        ; work counter
3  poll:
4  inc dx           ; background work
5  mov ah, 01h      ; key ready?
6  int 16h          ; ZF=1 → none
7  jz poll
8  mov ah, 00h      ; take the key
9  int 16h
10 ret
```

DX = 0: this counter counts the work done while waiting.

Step 1 / 16

VARIABLES

AH = **00h** AL = **00h** DX = **0000h** ZF = **1**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
5  mov ah, 01h       ; key ready?
6  int 16h           ; ZF=1 → none
7  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

Background work: DX = 0001h.

Step 2 / 16

VARIABLES

AH = 00h | AL = 00h | DX = 0001h | ZF = 0

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
▶  mov ah, 01h       ; key ready?
6  int 16h           ; ZF=1 → none
7  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

AH = 01h: ask for the key status (no wait).

Step 3 / 16

VARIABLES

AH = **01h** | AL = **00h** | DX = **0001h** | ZF = **0**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
5  mov ah, 01h       ; key ready?
▶  int 16h           ; ZF=1 → none
7  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

No key in the buffer → **ZF = 1**.

Step 4 / 16

VARIABLES

AH = **01h** | AL = **00h** | DX = **0001h** | ZF = **1**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
5  mov ah, 01h       ; key ready?
6  int 16h           ; ZF=1 → none
▶  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

ZF = 1 → back to `poll`; the program does not freeze.

Step 5 / 16

VARIABLES

AH = **01h** AL = **00h** DX = **0001h** ZF = **1**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
▶  inc dx            ; background work
5  mov ah, 01h       ; key ready?
6  int 16h           ; ZF=1 → none
7  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

Background work: DX = 0002h.

Step 6 / 16

VARIABLES

AH = **01h** AL = **00h** DX = **0002h** ZF = **0**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
▶  mov ah, 01h       ; key ready?
6  int 16h           ; ZF=1 → none
7  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

AH = 01h: ask for the key status (no wait).

Step 7 / 16

VARIABLES

AH = **01h** | AL = **00h** | DX = **0002h** | ZF = **0**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
5  mov ah, 01h       ; key ready?
▶  int 16h           ; ZF=1 → none
7  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

Still no key → ZF = 1.

Step 8 / 16

VARIABLES

AH = **01h** AL = **00h** DX = **0002h** ZF = **1**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
5  mov ah, 01h       ; key ready?
6  int 16h           ; ZF=1 → none
▶  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

Keep polling.

Step 9 / 16

VARIABLES

AH = **01h** AL = **00h** DX = **0002h** ZF = **1**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
▶  inc dx            ; background work
5  mov ah, 01h       ; key ready?
6  int 16h           ; ZF=1 → none
7  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

Background work: DX = 0003h.

Step 10 / 16

VARIABLES

AH = **01h** AL = **00h** DX = **0003h** ZF = **0**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
▶  mov ah, 01h       ; key ready?
6  int 16h           ; ZF=1 → none
7  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

AH = 01h: ask for the key status (no wait).

Step 11 / 16

VARIABLES

AH = **01h** | AL = **00h** | DX = **0003h** | ZF = **0**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
5  mov ah, 01h       ; key ready?
▶  int 16h           ; ZF=1 → none
7  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

A key arrived: **ZF = 0**, AL = 'x' (78h), AH = 2Dh. The key **stays** in the buffer.

Step 12 / 16

VARIABLES

AH = **2Dh** | AL = **78h** | DX = **0003h** | ZF = **0**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
5  mov ah, 01h       ; key ready?
6  int 16h           ; ZF=1 → none
▶  jz poll
8  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

ZF = 0 → leave the loop.

Step 13 / 16

VARIABLES

AH = **2Dh** | AL = **78h** | DX = **0003h** | ZF = **0**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
5  mov ah, 01h       ; key ready?
6  int 16h           ; ZF=1 → none
7  jz poll
▶  mov ah, 00h       ; take the key
9  int 16h
10 ret
```

AH = 00h to take the key out of the buffer.

Step 14 / 16

VARIABLES

AH = **00h** | AL = **78h** | DX = **0003h** | ZF = **0**

Code Trace: Polling Without Waiting, AH = 01h

```
1  org 100h
2  xor dx, dx        ; work counter
3  poll:
4  inc dx            ; background work
5  mov ah, 01h       ; key ready?
6  int 16h           ; ZF=1 → none
7  jz poll
8  mov ah, 00h       ; take the key
▶  int 16h
10 ret
```

The same key is read and removed from the buffer: AL = 'x', AH = 2Dh.

Step 15 / 16

VARIABLES

AH = **2Dh** | AL = **78h** | DX = **0003h** | ZF = **0**

Code Trace: Polling Without Waiting, AH = 01h

```
1 org 100h
2 xor dx, dx        ; work counter
3 poll:
4 inc dx            ; background work
5 mov ah, 01h       ; key ready?
6 int 16h           ; ZF=1 → none
7 jz poll
8 mov ah, 00h       ; take the key
9 int 16h
▶ ret
```

Games and animations use this pattern: keep running, react when a key is pressed.

Step 16 / 16

VARIABLES

AH = **2Dh** AL = **78h** DX = **0003h** ZF = **0**

INT 21h: Common DOS Services

AH	Service	Input	Output
`01h`	read char with **echo**	–	AL = char
`02h`	write char	DL = char	AL = DL
`07h` / `08h`	read char, no echo	–	AL (08h checks Ctrl+C)
`09h`	write string	DS:DX → '$'-terminated string	AL = 24h
`0Ah`	buffered line input	DS:DX → buffer	buf[1] = count, buf+2 = text
`2Ah`	get date	–	CX = year, DH = month, DL = day, AL = weekday (0 = Sun)
`2Ch`	get time	–	CH = hour, CL = min, DH = sec, DL = 1/100 s
`2Bh` / `2Dh`	set date / time	as for 2Ah / 2Ch	AL = 00h on success
`4Ch`	terminate program	AL = exit code	(does not return)

DOS services honor redirection: with `program > out.txt`, 02h/09h output goes to the file; BIOS INT 10h output always reaches the screen.

INT 10h/0Eh or INT 21h/02h?

BIOS · INT 10h, AH = 0Eh

- The character is in **AL**.
- Goes straight to the video BIOS; no DOS needed.
- Works in boot code too.
- A color can be given (BL, in graphics modes).

DOS · INT 21h, AH = 02h

- The character is in **DL**.
- A DOS call; output can be redirected (to a file, a printer).
- On return AL = DL.
- Checks for Ctrl+Break.

Both print one character and advance the cursor; the choice depends on where the program will run. They can be mixed in one program.

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
▶  mov ah, 01h       ; read + echo
3  int 21h           ; AL = char
4  mov bl, al        ; save it
5  mov ah, 02h       ; write char
6  mov dl, 0Dh       ; CR: line start
7  int 21h
8  mov dl, 0Ah       ; LF: next line
9  int 21h
10 inc bl            ; next letter
11 mov dl, bl
12 int 21h
13 ret
```

AH = 01h: wait for a character from the keyboard and **echo it**.

Step 1 / 12

VARIABLES

AL = **00h** AH = **01h** BL = **00h** DL = **00h**

OUTPUT

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
2  mov ah, 01h      ; read + echo
▶  int 21h          ; AL = char
4  mov bl, al       ; save it
5  mov ah, 02h      ; write char
6  mov dl, 0Dh      ; CR: line start
7  int 21h
8  mov dl, 0Ah      ; LF: next line
9  int 21h
10 inc bl           ; next letter
11 mov dl, bl
12 int 21h
13 ret
```

The user pressed **k**: AL = 6Bh, and 'k' appears on the screen.

Step 2 / 12

VARIABLES

AL = **6Bh** | AH = **01h** | BL = **00h** | DL = **00h**

OUTPUT

```
k
```

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
2  mov ah, 01h      ; read + echo
3  int 21h          ; AL = char
▶  mov bl, al       ; save it
5  mov ah, 02h      ; write char
6  mov dl, 0Dh      ; CR: line start
7  int 21h
8  mov dl, 0Ah      ; LF: next line
9  int 21h
10 inc bl           ; next letter
11 mov dl, bl
12 int 21h
13 ret
```

Later INT 21h calls will change AL, so it is saved in BL.

Step 3 / 12

VARIABLES

AL = **6Bh** AH = **01h** BL = **6Bh** DL = **00h**

OUTPUT

```
k
```

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
2  mov ah, 01h      ; read + echo
3  int 21h          ; AL = char
4  mov bl, al       ; save it
▶  mov ah, 02h      ; write char
6  mov dl, 0Dh      ; CR: line start
7  int 21h
8  mov dl, 0Ah      ; LF: next line
9  int 21h
10 inc bl           ; next letter
11 mov dl, bl
12 int 21h
13 ret
```

AH = 02h: write the character in DL.

Step 4 / 12

VARIABLES

AL = **6Bh** AH = **02h** BL = **6Bh** DL = **00h**

OUTPUT

```
k
```

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
2  mov ah, 01h      ; read + echo
3  int 21h          ; AL = char
4  mov bl, al       ; save it
5  mov ah, 02h      ; write char
▶  mov dl, 0Dh      ; CR: line start
7  int 21h
8  mov dl, 0Ah      ; LF: next line
9  int 21h
10 inc bl           ; next letter
11 mov dl, bl
12 int 21h
13 ret
```

DL = 0Dh (carriage return).

Step 5 / 12

VARIABLES

AL = 6Bh | AH = 02h | BL = 6Bh | DL = 0Dh

OUTPUT

```
k
```

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
2  mov ah, 01h      ; read + echo
3  int 21h          ; AL = char
4  mov bl, al       ; save it
5  mov ah, 02h      ; write char
6  mov dl, 0Dh      ; CR: line start
▶  int 21h
8  mov dl, 0Ah      ; LF: next line
9  int 21h
10 inc bl           ; next letter
11 mov dl, bl
12 int 21h
13 ret
```

The cursor returns to the start of the line (no visible change).

Step 6 / 12

VARIABLES

AL = 0Dh | AH = 02h | BL = 6Bh | DL = 0Dh

OUTPUT

```
k
```

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
2  mov ah, 01h      ; read + echo
3  int 21h          ; AL = char
4  mov bl, al       ; save it
5  mov ah, 02h      ; write char
6  mov dl, 0Dh      ; CR: line start
7  int 21h
▶  mov dl, 0Ah      ; LF: next line
9  int 21h
10 inc bl           ; next letter
11 mov dl, bl
12 int 21h
13 ret
```

DL = 0Ah (line feed).

Step 7 / 12

VARIABLES

AL = 0Dh | AH = 02h | BL = 6Bh | DL = 0Ah

OUTPUT

```
k
```

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
2  mov ah, 01h      ; read + echo
3  int 21h          ; AL = char
4  mov bl, al       ; save it
5  mov ah, 02h      ; write char
6  mov dl, 0Dh      ; CR: line start
7  int 21h
8  mov dl, 0Ah      ; LF: next line
▶  int 21h
10 inc bl           ; next letter
11 mov dl, bl
12 int 21h
13 ret
```

The cursor moves down a line. New line = **CR + LF**.

Step 8 / 12

VARIABLES

AL = **0Ah** | AH = **02h** | BL = **6Bh** | DL = **0Ah**

OUTPUT

```
k
```

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
2  mov ah, 01h       ; read + echo
3  int 21h           ; AL = char
4  mov bl, al        ; save it
5  mov ah, 02h       ; write char
6  mov dl, 0Dh       ; CR: line start
7  int 21h
8  mov dl, 0Ah       ; LF: next line
9  int 21h
▶ inc bl             ; next letter
11 mov dl, bl
12 int 21h
13 ret
```

BL = 6Bh + 1 = 6Ch = 'l'.

Step 9 / 12

VARIABLES

AL = **0Ah** | AH = **02h** | BL = **6Ch** | DL = **0Ah**

OUTPUT

```
k
```

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
2  mov ah, 01h      ; read + echo
3  int 21h          ; AL = char
4  mov bl, al       ; save it
5  mov ah, 02h      ; write char
6  mov dl, 0Dh      ; CR: line start
7  int 21h
8  mov dl, 0Ah      ; LF: next line
9  int 21h
10 inc bl           ; next letter
▶  mov dl, bl
12 int 21h
13 ret
```

DL = 'l'.

Step 10 / 12

VARIABLES

AL = **0Ah** AH = **02h** BL = **6Ch** DL = **6Ch**

OUTPUT

```
k
```

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
2  mov ah, 01h      ; read + echo
3  int 21h          ; AL = char
4  mov bl, al       ; save it
5  mov ah, 02h      ; write char
6  mov dl, 0Dh      ; CR: line start
7  int 21h
8  mov dl, 0Ah      ; LF: next line
9  int 21h
10 inc bl           ; next letter
11 mov dl, bl
▶ int 21h
13 ret
```

l is printed. After 02h, AL also holds the character written.

Step 11 / 12

VARIABLES

AL = **6Ch** AH = **02h** BL = **6Ch** DL = **6Ch**

OUTPUT

```
k
l
```

Code Trace: Read a Character, New Line, Write a Character

```
1  org 100h
2  mov ah, 01h      ; read + echo
3  int 21h          ; AL = char
4  mov bl, al       ; save it
5  mov ah, 02h      ; write char
6  mov dl, 0Dh      ; CR: line start
7  int 21h
8  mov dl, 0Ah      ; LF: next line
9  int 21h
10 inc bl           ; next letter
11 mov dl, bl
12 int 21h
▶  ret
```

The program ends.

Step 12 / 12

VARIABLES

AL = **6Ch** AH = **02h** BL = **6Ch** DL = **6Ch**

OUTPUT

```
k
l
```

Code Trace: Printing a String with AH = 09h

```
1 org 100h
▶ mov dx, offset msg1   ; DS:DX → string
3 mov ah, 09h
4 int 21h
5 mov dx, offset msg2
6 int 21h               ; AH still 09h
7 ret
8 msg1 db 'Welcome, 8086!', 0Dh, 0Ah, '$'
9 msg2 db 'Good-bye!$'
```

DX = 010Dh: the offset of the string. The segment is DS.

Step 1 / 6

VARIABLES

AX = **0000h** DX = **010Dh**

OUTPUT

Code Trace: Printing a String with AH = 09h

```
1 org 100h
2 mov dx, offset msg1   ; DS:DX → string
▶ mov ah, 09h
4 int 21h
5 mov dx, offset msg2
6 int 21h               ; AH still 09h
7 ret
8 msg1 db 'Welcome, 8086!', 0Dh, 0Ah, '$'
9 msg2 db 'Good-bye!$'
```

AH = 09h: print a '$'-terminated string.

Step 2 / 6

VARIABLES

AX = **0900h** DX = **010Dh**

OUTPUT

Code Trace: Printing a String with AH = 09h

```
1 org 100h
2 mov dx, offset msg1   ; DS:DX → string
3 mov ah, 09h
▶ int 21h
5 mov dx, offset msg2
6 int 21h               ; AH still 09h
7 ret
8 msg1 db 'Welcome, 8086!', 0Dh, 0Ah, '$'
9 msg2 db 'Good-bye!$'
```

Printing runs until '$'; 0Dh 0Ah ends the line. On return AL = 24h ('$').

Step 3 / 6

VARIABLES

AX = **0924h** DX = **010Dh**

OUTPUT

```
Welcome, 8086!
```

Code Trace: Printing a String with AH = 09h

```
1 org 100h
2 mov dx, offset msg1   ; DS:DX → string
3 mov ah, 09h
4 int 21h
▶ mov dx, offset msg2
6 int 21h               ; AH still 09h
7 ret
8 msg1 db 'Welcome, 8086!', 0Dh, 0Ah, '$'
9 msg2 db 'Good-bye!$'
```

Second string: DX = 011Eh.

Step 4 / 6

VARIABLES

AX = **0924h** DX = **011Eh**

OUTPUT

```
Welcome, 8086!
```

Code Trace: Printing a String with AH = 09h

```
1 org 100h
2 mov dx, offset msg1   ; DS:DX → string
3 mov ah, 09h
4 int 21h
5 mov dx, offset msg2
▶ int 21h               ; AH still 09h
7 ret
8 msg1 db 'Welcome, 8086!', 0Dh, 0Ah, '$'
9 msg2 db 'Good-bye!$'
```

AH has not changed, so there is no need to reload it (reloading is still a good habit).

Step 5 / 6

VARIABLES

AX = **0924h** DX = **011Eh**

OUTPUT

```
Welcome, 8086!
Good-bye!
```

Code Trace: Printing a String with AH = 09h

```
1 org 100h
2 mov dx, offset msg1   ; DS:DX → string
3 mov ah, 09h
4 int 21h
5 mov dx, offset msg2
6 int 21h               ; AH still 09h
▶ ret
8 msg1 db 'Welcome, 8086!', 0Dh, 0Ah, '$'
9 msg2 db 'Good-bye!$'
```

Two lines of output. This service cannot print the '$' character itself.

Step 6 / 6

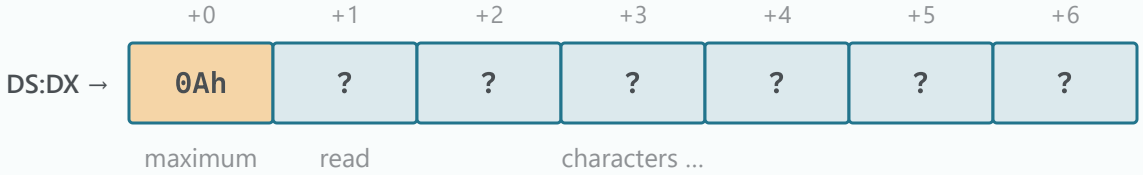
VARIABLES

AX = **0924h** DX = **011Eh**

OUTPUT

```
Welcome, 8086!
Good-bye!
```

AH = 0Ah: Buffered Line Input

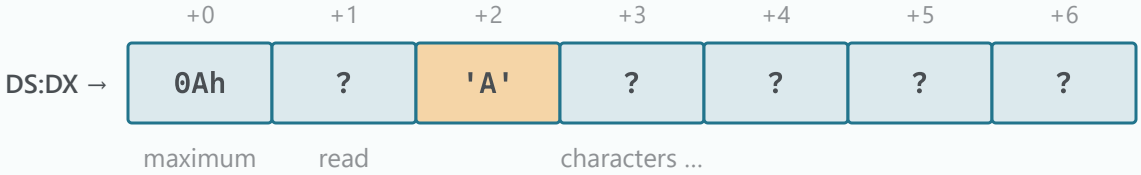


The program prepares the buffer: `buffer db 10, ?, 10 dup(?)`. Byte 0 = **at most 10** characters (Enter included).

Step 1 / 5

- DOS does the line editing itself (Backspace, Esc); the program is blocked until Enter is pressed.
- When the buffer is full, extra keys are refused with a beep. The first byte cannot be 0.

AH = 0Ah: Buffered Line Input

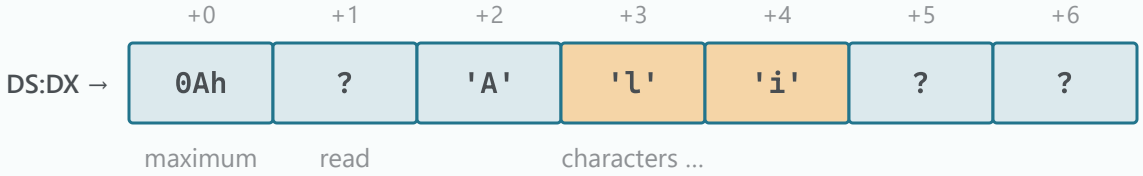


The user types **A**: the character goes to position 2 and is echoed. Backspace can correct mistakes.

Step 2 / 5

- DOS does the line editing itself (Backspace, Esc); the program is blocked until Enter is pressed.
- When the buffer is full, extra keys are refused with a beep. The first byte cannot be 0.

AH = 0Ah: Buffered Line Input

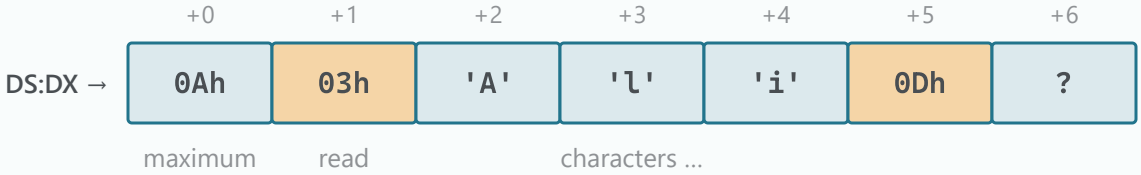


l and **i** are added too.

Step 3 / 5

- DOS does the line editing itself (Backspace, Esc); the program is blocked until Enter is pressed.
- When the buffer is full, extra keys are refused with a beep. The first byte cannot be 0.

AH = 0Ah: Buffered Line Input

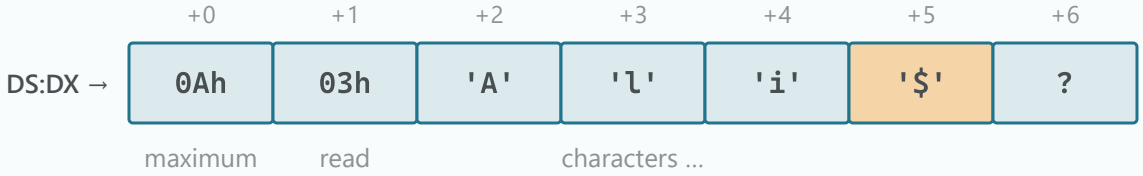


Enter: **0Dh** is stored at the end; DOS writes the count read (**3**, CR excluded) into byte 1 and returns.

Step 4 / 5

- DOS does the line editing itself (Backspace, Esc); the program is blocked until Enter is pressed.
- When the buffer is full, extra keys are refused with a beep. The first byte cannot be 0.

AH = 0Ah: Buffered Line Input



The text starts at **buffer+2**; its length is in **[buffer+1]**. It does not end with '$'; to print it with 09h, put '$' in place of the 0Dh.

Step 5 / 5

- DOS does the line editing itself (Backspace, Esc); the program is blocked until Enter is pressed.
- When the buffer is full, extra keys are refused with a beep. The first byte cannot be 0.

Code Trace: Read a Name, Say Hello

```
1  org 100h
2  mov dx, offset buffer
3  mov ah, 0Ah          ; buffered input
4  int 21h
5  mov bl, [buffer+1]   ; count read
6  mov bh, 0
7  mov byte ptr buffer[bx+2], '$'
8  mov dx, offset greet
9  mov ah, 09h
10 int 21h
11 lea dx, [buffer+2]
12 int 21h
13 ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

DS:DX points to the buffer: DX = 011Fh.

Step 1 / 12

VARIABLES

BX = **0000h** DX = **011Fh** buf = **0Ah**

buf+1 = **00h** buf+2 = **'·····'**

OUTPUT

Code Trace: Read a Name, Say Hello

```
1  org 100h
2  mov dx, offset buffer
▶  mov ah, 0Ah          ; buffered input
4  int 21h
5  mov bl, [buffer+1]   ; count read
6  mov bh, 0
7  mov byte ptr buffer[bx+2], '$'
8  mov dx, offset greet
9  mov ah, 09h
10 int 21h
11 lea dx, [buffer+2]
12 int 21h
13 ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

AH = 0Ah: read a line up to Enter.

Step 2 / 12

VARIABLES

BX = **0000h** DX = **011Fh** buf = **0Ah**

buf+1 = **00h** buf+2 = '·····'

OUTPUT

Code Trace: Read a Name, Say Hello

```
1  org 100h
2  mov dx, offset buffer
3  mov ah, 0Ah          ; buffered input
▶  int 21h
5  mov bl, [buffer+1]   ; count read
6  mov bh, 0
7  mov byte ptr buffer[bx+2], '$'
8  mov dx, offset greet
9  mov ah, 09h
10 int 21h
11 lea dx, [buffer+2]
12 int 21h
13 ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

The user typed **Ali⏎**. DOS: buffer[1] = 3, buffer[2..4] = 'Ali', buffer[5] = 0Dh. Enter prints only a CR: the cursor is at the start of the line.

Step 3 / 12

VARIABLES

BX = **0000h** DX = **011Fh** buf = **0Ah**

buf+1 = **03h** buf+2 = **'Ali··'**

OUTPUT

```
Ali
```

Code Trace: Read a Name, Say Hello

```
1  org 100h
2  mov dx, offset buffer
3  mov ah, 0Ah          ; buffered input
4  int 21h
▶  mov bl, [buffer+1]   ; count read
6  mov bh, 0
7  mov byte ptr buffer[bx+2], '$'
8  mov dx, offset greet
9  mov ah, 09h
10 int 21h
11 lea dx, [buffer+2]
12 int 21h
13 ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

BL = number of characters read = **3** (CR not counted).

Step 4 / 12

VARIABLES

BX = **0003h** | DX = **011Fh** | buf = **0Ah** | buf+1 = **03h** | buf+2 = **'Ali··'**

OUTPUT

```
Ali
```

Code Trace: Read a Name, Say Hello

```
1  org 100h
2  mov dx, offset buffer
3  mov ah, 0Ah          ; buffered input
4  int 21h
5  mov bl, [buffer+1]   ; count read
▶  mov bh, 0
7  mov byte ptr buffer[bx+2], '$'
8  mov dx, offset greet
9  mov ah, 09h
10 int 21h
11 lea dx, [buffer+2]
12 int 21h
13 ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

BX = 0003h, for the address calculation.

Step 5 / 12

VARIABLES

BX = **0003h** DX = **011Fh** buf = **0Ah**

buf+1 = **03h** buf+2 = **'Ali··'**

OUTPUT

```
Ali
```

Code Trace: Read a Name, Say Hello

```
1  org 100h
2  mov dx, offset buffer
3  mov ah, 0Ah          ; buffered input
4  int 21h
5  mov bl, [buffer+1]   ; count read
6  mov bh, 0
▶  mov byte ptr buffer[bx+2], '$'
8  mov dx, offset greet
9  mov ah, 09h
10 int 21h
11 lea dx, [buffer+2]
12 int 21h
13 ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

buffer[3 + 2] = buffer[5]: **'$'** replaces the 0Dh → the string can be printed with AH = 09h.

Step 6 / 12

VARIABLES

BX = **0003h** DX = **011Fh** buf = **0Ah**

buf+1 = **03h** buf+2 = **'Ali$·'**

OUTPUT

```
Ali
```

Code Trace: Read a Name, Say Hello

```
 1 org 100h
 2 mov dx, offset buffer
 3 mov ah, 0Ah           ; buffered input
 4 int 21h
 5 mov bl, [buffer+1]    ; count read
 6 mov bh, 0
 7 mov byte ptr buffer[bx+2], '$'
▶ mov dx, offset greet
 9 mov ah, 09h
10 int 21h
11 lea dx, [buffer+2]
12 int 21h
13 ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

DX = greet.

Step 7 / 12

VARIABLES

BX = **0003h** | DX = **012Bh** | buf = **0Ah** | buf+1 = **03h** | buf+2 = **'Ali$·'**

OUTPUT

```
Ali
```

Code Trace: Read a Name, Say Hello

```
 1 org 100h
 2 mov dx, offset buffer
 3 mov ah, 0Ah           ; buffered input
 4 int 21h
 5 mov bl, [buffer+1]    ; count read
 6 mov bh, 0
 7 mov byte ptr buffer[bx+2], '$'
 8 mov dx, offset greet
▶ mov ah, 09h
10 int 21h
11 lea dx, [buffer+2]
12 int 21h
13 ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

AH = 09h.

Step 8 / 12

VARIABLES

BX = **0003h** DX = **012Bh** buf = **0Ah**

buf+1 = **03h** buf+2 = **'Ali$·'**

OUTPUT

```
Ali
```

Code Trace: Read a Name, Say Hello

```
1  org 100h
2  mov dx, offset buffer
3  mov ah, 0Ah          ; buffered input
4  int 21h
5  mov bl, [buffer+1]   ; count read
6  mov bh, 0
7  mov byte ptr buffer[bx+2], '$'
8  mov dx, offset greet
9  mov ah, 09h
▶ int 21h
11 lea dx, [buffer+2]
12 int 21h
13 ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

CR LF + 'Hello ' is printed (on a new line).

Step 9 / 12

VARIABLES

BX = **0003h** DX = **012Bh** buf = **0Ah**

buf+1 = **03h** buf+2 = **'Ali$·'**

OUTPUT

```
Ali
Hello
```

Code Trace: Read a Name, Say Hello

```
 1 org 100h
 2 mov dx, offset buffer
 3 mov ah, 0Ah           ; buffered input
 4 int 21h
 5 mov bl, [buffer+1]    ; count read
 6 mov bh, 0
 7 mov byte ptr buffer[bx+2], '$'
 8 mov dx, offset greet
 9 mov ah, 09h
10 int 21h
▶ lea dx, [buffer+2]
12 int 21h
13 ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

DX = buffer + 2: the start of the text entered (0121h).

Step 10 / 12

VARIABLES

BX = **0003h** | DX = **0121h** | buf = **0Ah** | buf+1 = **03h** | buf+2 = **'Ali$·'**

OUTPUT

```
Ali
Hello
```

Code Trace: Read a Name, Say Hello

```
1  org 100h
2  mov dx, offset buffer
3  mov ah, 0Ah          ; buffered input
4  int 21h
5  mov bl, [buffer+1]   ; count read
6  mov bh, 0
7  mov byte ptr buffer[bx+2], '$'
8  mov dx, offset greet
9  mov ah, 09h
10 int 21h
11 lea dx, [buffer+2]
▶ int 21h
13 ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

Output: **Hello Ali**.

Step 11 / 12

VARIABLES

BX = **0003h** DX = **0121h** buf = **0Ah**

buf+1 = **03h** buf+2 = **'Ali$·'**

OUTPUT

```
Ali
Hello Ali
```

Code Trace: Read a Name, Say Hello

```
 1 org 100h
 2 mov dx, offset buffer
 3 mov ah, 0Ah          ; buffered input
 4 int 21h
 5 mov bl, [buffer+1]   ; count read
 6 mov bh, 0
 7 mov byte ptr buffer[bx+2], '$'
 8 mov dx, offset greet
 9 mov ah, 09h
10 int 21h
11 lea dx, [buffer+2]
12 int 21h
▶ ret
14 buffer db 10, ?, 10 dup(?)
15 greet  db 0Dh, 0Ah, 'Hello $'
```

buffer[0] = 10: at most 9 characters + CR. Anything more is refused (DOS beeps).

Step 12 / 12

VARIABLES

BX = **0003h** DX = **0121h** buf = **0Ah**

buf+1 = **03h** buf+2 = **'Ali$·'**

OUTPUT

```
Ali
Hello Ali
```

AH = 2Ah / 2Ch: Date and Time

AH = 2Ah (date)	
CX	year (e.g. 07EAh = 2026)
DH	month (1–12)
DL	day (1–31)
AL	day of week (0 = Sunday … 6 = Saturday)

AH = 2Ch (time)	
CH	hour (0–23)
CL	minute (0–59)
DH	second (0–59)
DL	hundredths (0–99)

```
; print the day name
mov ah, 2Ah
int 21h           ; AL = 0…6
mov ah, 0
mov bl, 4
mul bl            ; AX = AL × 4
mov dx, offset days
add dx, ax
mov ah, 09h
int 21h
ret
days db 'Sun$Mon$Tue$Wed$'
     db 'Thu$Fri$Sat$'
```

The values are **binary**, not BCD: hour 14 → CH = 0Eh. To print them, use AAM or divide by 10. The BIOS INT 1Ah/02h, by contrast, returns the time in BCD.

Day Name from a Table: AL × 4

```
1  org 100h
▶  mov ah, 2Ah
3  int 21h          ; AL = 0…6
4  mov ah, 0
5  mov bl, 4
6  mul bl           ; AX = AL × 4
7  mov dx, offset days
8  add dx, ax
9  mov ah, 09h
10 int 21h
11 ret
12 days db 'Sun$Mon$Tue$Wed$'
13      db 'Thu$Fri$Sat$'
```



AH = 2Ah: the DOS date service.

Step 1 / 8

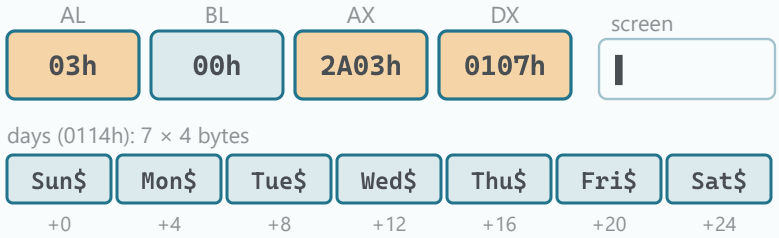
VARIABLES

AL = 00h AX = 2A00h DX = 0000h

The date and weekday come from the system clock; the trace assumes Wednesday 07.01.2026. `mov dx, offset days` wipes the date in DX; save it first if you need it.

Day Name from a Table: AL × 4

```
1  org 100h
2  mov ah, 2Ah
▶  int 21h           ; AL = 0…6
4  mov ah, 0
5  mov bl, 4
6  mul bl            ; AX = AL × 4
7  mov dx, offset days
8  add dx, ax
9  mov ah, 09h
10 int 21h
11 ret
12 days db 'Sun$Mon$Tue$Wed$'
13      db 'Thu$Fri$Sat$'
```



Sample date Wednesday, 7 January 2026: CX = 07EAh (2026), **DX = 0107h** (DH = month 1, DL = day 7), **AL = 3** (0 = Sunday).

Step 2 / 8

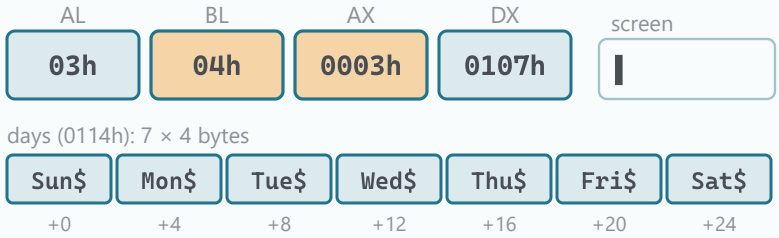
VARIABLES

AL = 03h AX = 2A03h DX = 0107h

The date and weekday come from the system clock; the trace assumes Wednesday 07.01.2026. `mov dx, offset days` wipes the date in DX; save it first if you need it.

Day Name from a Table: AL × 4

```
1  org 100h
2  mov ah, 2Ah
3  int 21h           ; AL = 0…6
▶  mov ah, 0
▶  mov bl, 4
6  mul bl            ; AX = AL × 4
7  mov dx, offset days
8  add dx, ax
9  mov ah, 09h
10 int 21h
11 ret
12 days db 'Sun$Mon$Tue$Wed$'
13      db 'Thu$Fri$Sat$'
```



AH = 0 (AX = 0003h), BL = 4: each day name is 4 bytes including '$'.

Step 3 / 8

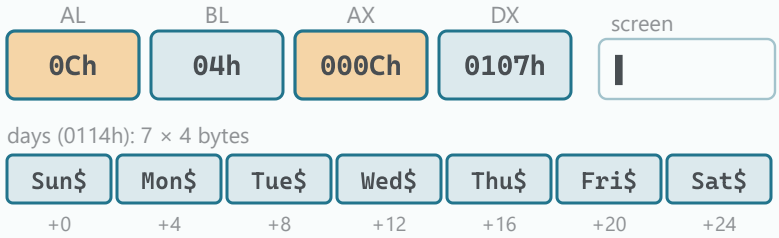
VARIABLES

AL = 03h AX = 0003h DX = 0107h

The date and weekday come from the system clock; the trace assumes Wednesday 07.01.2026. `mov dx, offset days` wipes the date in DX; save it first if you need it.

Day Name from a Table: AL × 4

```
1  org 100h
2  mov ah, 2Ah
3  int 21h           ; AL = 0…6
4  mov ah, 0
5  mov bl, 4
▶  mul bl            ; AX = AL × 4
7  mov dx, offset days
8  add dx, ax
9  mov ah, 09h
10 int 21h
11 ret
12 days db 'Sun$Mon$Tue$Wed$'
13      db 'Thu$Fri$Sat$'
```



MUL BL: AX = 3 × 4 = **12** = 000Ch: the offset within the table.

Step 4 / 8

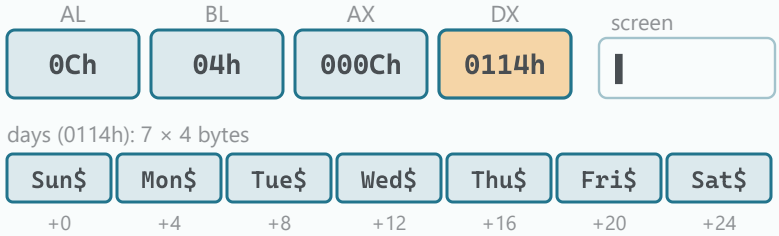
VARIABLES

AL = **0Ch** AX = **000Ch** DX = **0107h**

The date and weekday come from the system clock; the trace assumes Wednesday 07.01.2026. `mov dx, offset days` wipes the date in DX; save it first if you need it.

Day Name from a Table: AL × 4

```
1  org 100h
2  mov ah, 2Ah
3  int 21h           ; AL = 0…6
4  mov ah, 0
5  mov bl, 4
6  mul bl            ; AX = AL × 4
▶  mov dx, offset days
8  add dx, ax
9  mov ah, 09h
10 int 21h
11 ret
12 days db 'Sun$Mon$Tue$Wed$'
13      db 'Thu$Fri$Sat$'
```



DX = days = 0114h.

Step 5 / 8

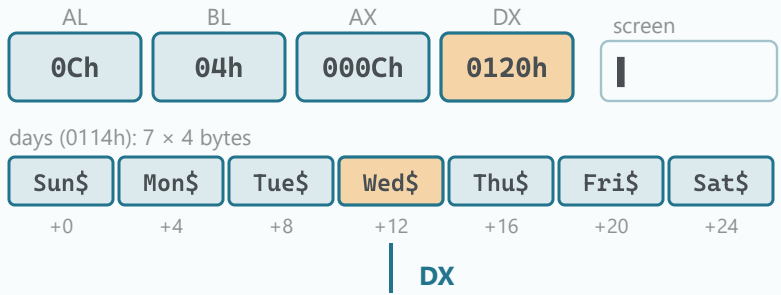
VARIABLES

AL = 0Ch AX = 000Ch DX = 0114h

The date and weekday come from the system clock; the trace assumes Wednesday 07.01.2026. `mov dx, offset days` wipes the date in DX; save it first if you need it.

Day Name from a Table: AL × 4

```
1  org 100h
2  mov ah, 2Ah
3  int 21h           ; AL = 0…6
4  mov ah, 0
5  mov bl, 4
6  mul bl            ; AX = AL × 4
7  mov dx, offset days
▶  add dx, ax
9  mov ah, 09h
10 int 21h
11 ret
12 days db 'Sun$Mon$Tue$Wed$'
13      db 'Thu$Fri$Sat$'
```



ADD DX, AX → DX = 0120h: the start of the string 'Wed$'.

Step 6 / 8

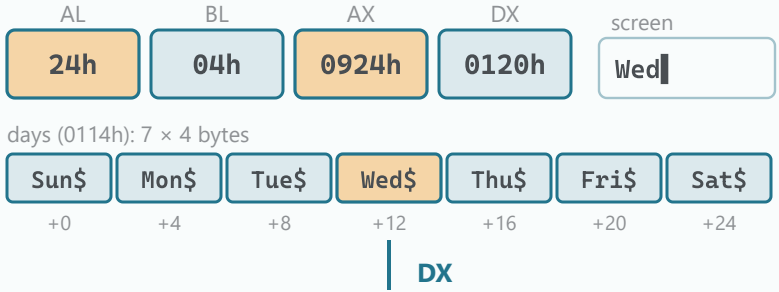
VARIABLES

AL = 0Ch AX = 000Ch DX = 0120h

The date and weekday come from the system clock; the trace assumes Wednesday 07.01.2026. `mov dx, offset days` wipes the date in DX; save it first if you need it.

Day Name from a Table: AL × 4

```
1  org 100h
2  mov ah, 2Ah
3  int 21h           ; AL = 0…6
4  mov ah, 0
5  mov bl, 4
6  mul bl            ; AX = AL × 4
7  mov dx, offset days
8  add dx, ax
▶  mov ah, 09h
▶  int 21h
11 ret
12 days db 'Sun$Mon$Tue$Wed$'
13      db 'Thu$Fri$Sat$'
```



AH = 09h, INT 21h: from DS:DX up to '$' → **Wed** on the screen.

Step 7 / 8

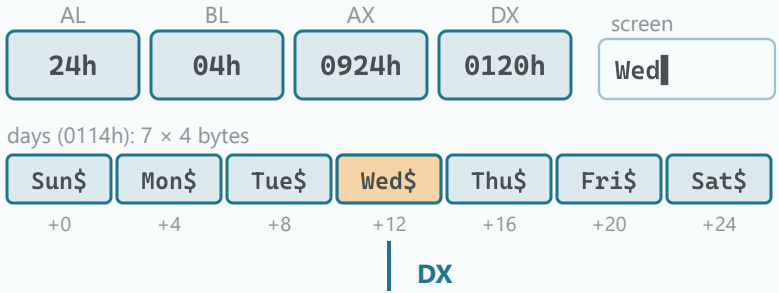
VARIABLES

AL = **24h** AX = **0924h** DX = **0120h**

The date and weekday come from the system clock; the trace assumes Wednesday 07.01.2026. `mov dx, offset days` wipes the date in DX; save it first if you need it.

Day Name from a Table: AL × 4

```
1  org 100h
2  mov ah, 2Ah
3  int 21h           ; AL = 0…6
4  mov ah, 0
5  mov bl, 4
6  mul bl            ; AX = AL × 4
7  mov dx, offset days
8  add dx, ax
9  mov ah, 09h
10 int 21h
▶  ret
12 days db 'Sun$Mon$Tue$Wed$'
13      db 'Thu$Fri$Sat$'
```



RET: the program ends.

Step 8 / 8

VARIABLES

AL = **24h** AX = **0924h** DX = **0120h**

The date and weekday come from the system clock; the trace assumes Wednesday 07.01.2026. `mov dx, offset days` wipes the date in DX; save it first if you need it.

Code Trace: Printing the Time as HH:MM

```
1  org 100h
▶  mov ah, 2Ch          ; get system time
3  int 21h              ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:              ; AL (0-99) → 2 digits
13 aam                  ; AH=tens AL=ones
14 add ax, 3030h        ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

AH = 2Ch: read the system time.

Step 1 / 28

VARIABLES

AX = **2C00h** BX = **0000h** CX = **0000h**
DX = **0000h** SP = **FFFEh**

CALL STACK

empty

OUTPUT

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
▶  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

Returned: **CH = 14** (0Eh) hours, **CL = 35** (23h) minutes, DH = 7 seconds, DL = 42 hundredths.

Step 2 / 28

VARIABLES

AX = **2C00h** BX = **0000h** CX = **0E23h** DX = **072Ah** SP = **FFFEh**

CALL STACK

empty

OUTPUT

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
▶  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

AL = hour = 0Eh.

Step 3 / 28

VARIABLES

AX = **2C0Eh** BX = **0000h** CX = **0E23h** DX = **072Ah** SP = **FFFEh**

CALL STACK

empty

OUTPUT

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
▶  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

The procedure is called: the return address is on the stack.

Step 4 / 28

VARIABLES

AX = **2C0Eh** BX = **0000h** CX = **0E23h**

DX = **072Ah** SP = **FFFCh**

CALL STACK

FFFCh: 0109h

OUTPUT

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
▶ aam                 ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

`aam`: 14 → AH = 1, AL = 4.

Step 5 / 28

VARIABLES

AX = **0104h** BX = **0000h** CX = **0E23h**
DX = **072Ah** SP = **FFFCh**

CALL STACK

FFFCh: 0109h

OUTPUT

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
▶ add ax, 3030h       ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

Both digits to ASCII: AX = 3134h.

Step 6 / 28

VARIABLES

AX = 3134h | BX = 0000h | CX = 0E23h | DX = 072Ah | SP = FFFCh

CALL STACK

FFFCh: 0109h

OUTPUT

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
▶  mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

Save in BX; INT 21h/02h changes AL.

Step 7 / 28

VARIABLES

AX = **3134h** BX = **3134h** CX = **0E23h**

DX = **072Ah** SP = **FFFCh**

CALL STACK

FFFCh: 0109h

OUTPUT

Code Trace: Printing the Time as HH:MM

```
 1 org 100h
 2 mov ah, 2Ch        ; get system time
 3 int 21h            ; CH=hour CL=min
 4 mov al, ch
 5 call print2
 6 mov dl, ':'
 7 mov ah, 02h
 8 int 21h
 9 mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
▶ mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

DL = tens digit.

Step 8 / 28

VARIABLES

AX = **3134h** BX = **3134h** CX = **0E23h**

DX = **0731h** SP = **FFFCh**

CALL STACK

FFFCh: 0109h

OUTPUT

Code Trace: Printing the Time as HH:MM

```
 1 org 100h
 2 mov ah, 2Ch        ; get system time
 3 int 21h            ; CH=hour CL=min
 4 mov al, ch
 5 call print2
 6 mov dl, ':'
 7 mov ah, 02h
 8 int 21h
 9 mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
▶ mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

AH = 02h.

Step 9 / 28

VARIABLES

AX = **0234h** BX = **3134h** CX = **0E23h**

DX = **0731h** SP = **FFFCh**

CALL STACK

FFFCh: 0109h

OUTPUT

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
▶  int 21h
19 mov dl, bl
20 int 21h
21 ret
```

'1' on the screen.

Step 10 / 28

VARIABLES

AX = **0231h** BX = **3134h** CX = **0E23h**

DX = **0731h** SP = **FFFCh**

CALL STACK

FFFCh: 0109h

OUTPUT

```
1
```

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
▶ mov dl, bl
20 int 21h
21 ret
```

DL = ones digit.

Step 11 / 28

VARIABLES

AX = **0231h** BX = **3134h** CX = **0E23h**

DX = **0734h** SP = **FFFCh**

CALL STACK

FFFCh: 0109h

OUTPUT

```
1
```

Code Trace: Printing the Time as HH:MM

```
 1 org 100h
 2 mov ah, 2Ch        ; get system time
 3 int 21h            ; CH=hour CL=min
 4 mov al, ch
 5 call print2
 6 mov dl, ':'
 7 mov ah, 02h
 8 int 21h
 9 mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
▶ int 21h
21 ret
```

'4' on the screen → **14**.

Step 12 / 28

VARIABLES

AX = **0234h** BX = **3134h** CX = **0E23h**

DX = **0734h** SP = **FFFCh**

CALL STACK

FFFCh: 0109h

OUTPUT

```
14
```

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
▶  ret
```

Return from the procedure.

Step 13 / 28

VARIABLES

AX = **0234h** BX = **3134h** CX = **0E23h**
DX = **0734h** SP = **FFFEh**

CALL STACK

empty

OUTPUT

```
14
```

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
▶  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

DL = ':'.

Step 14 / 28

VARIABLES

AX = **0234h** BX = **3134h** CX = **0E23h**

DX = **073Ah** SP = **FFFEh**

CALL STACK

empty

OUTPUT

```
14
```

Code Trace: Printing the Time as HH:MM

```
 1 org 100h
 2 mov ah, 2Ch        ; get system time
 3 int 21h            ; CH=hour CL=min
 4 mov al, ch
 5 call print2
 6 mov dl, ':'
 ▶ mov ah, 02h
 8 int 21h
 9 mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

AH = 02h.

Step 15 / 28

VARIABLES

AX = **0234h** BX = **3134h** CX = **0E23h**

DX = **073Ah** SP = **FFFEh**

CALL STACK

empty

OUTPUT

```
14
```

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
▶  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

':' on the screen.

Step 16 / 28

VARIABLES

AX = 023Ah | BX = 3134h | CX = 0E23h | DX = 073Ah | SP = FFFEh

CALL STACK

empty

OUTPUT

```
14:
```

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
▶  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

AL = minute = 23h (35). No call has touched CL.

Step 17 / 28

VARIABLES

AX = **0223h** BX = **3134h** CX = **0E23h**
DX = **073Ah** SP = **FFFEh**

CALL STACK

empty

OUTPUT

```
14:
```

Code Trace: Printing the Time as HH:MM

```
 1 org 100h
 2 mov ah, 2Ch        ; get system time
 3 int 21h            ; CH=hour CL=min
 4 mov al, ch
 5 call print2
 6 mov dl, ':'
 7 mov ah, 02h
 8 int 21h
 9 mov al, cl
▶ call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

The same procedure for the minutes.

Step 18 / 28

VARIABLES

AX = **0223h** BX = **3134h** CX = **0E23h**
DX = **073Ah** SP = **FFFCh**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:
```

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
▶  aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

35 → AH = 3, AL = 5.

Step 19 / 28

VARIABLES

AX = **0305h** BX = **3134h** CX = **0E23h**

DX = **073Ah** SP = **FFFCh**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:
```

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
▶ add ax, 3030h       ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

Both digits to ASCII: AX = 3335h.

Step 20 / 28

VARIABLES

AX = **3335h** BX = **3134h** CX = **0E23h**
DX = **073Ah** SP = **FFFCh**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:
```

Code Trace: Printing the Time as HH:MM

```
 1 org 100h
 2 mov ah, 2Ch        ; get system time
 3 int 21h            ; CH=hour CL=min
 4 mov al, ch
 5 call print2
 6 mov dl, ':'
 7 mov ah, 02h
 8 int 21h
 9 mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
▶ mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

Save in BX; INT 21h/02h changes AL.

Step 21 / 28

VARIABLES

AX = **3335h** BX = **3335h** CX = **0E23h**
DX = **073Ah** SP = **FFFCh**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:
```

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
▶ mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

DL = tens digit.

Step 22 / 28

VARIABLES

AX = **3335h** BX = **3335h** CX = **0E23h**

DX = **0733h** SP = **FFFCh**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:
```

Code Trace: Printing the Time as HH:MM

```
 1 org 100h
 2 mov ah, 2Ch        ; get system time
 3 int 21h            ; CH=hour CL=min
 4 mov al, ch
 5 call print2
 6 mov dl, ':'
 7 mov ah, 02h
 8 int 21h
 9 mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
▶ mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

AH = 02h.

Step 23 / 28

VARIABLES

AX = **0235h** BX = **3335h** CX = **0E23h**

DX = **0733h** SP = **FFFCh**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:
```

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
▶  int 21h
19 mov dl, bl
20 int 21h
21 ret
```

'3' on the screen.

Step 24 / 28

VARIABLES

AX = **0233h** BX = **3335h** CX = **0E23h** DX = **0733h** SP = **FFFCh**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:3
```

Code Trace: Printing the Time as HH:MM

```
 1 org 100h
 2 mov ah, 2Ch        ; get system time
 3 int 21h            ; CH=hour CL=min
 4 mov al, ch
 5 call print2
 6 mov dl, ':'
 7 mov ah, 02h
 8 int 21h
 9 mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
▶ mov dl, bl
20 int 21h
21 ret
```

DL = ones digit.

Step 25 / 28

VARIABLES

AX = **0233h** BX = **3335h** CX = **0E23h**

DX = **0735h** SP = **FFFCh**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:3
```

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
▶ int 21h
21 ret
```

'5' on the screen → **14:35**.

Step 26 / 28

VARIABLES

AX = **0235h** BX = **3335h** CX = **0E23h**

DX = **0735h** SP = **FFFCh**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:35
```

Code Trace: Printing the Time as HH:MM

```
 1 org 100h
 2 mov ah, 2Ch        ; get system time
 3 int 21h            ; CH=hour CL=min
 4 mov al, ch
 5 call print2
 6 mov dl, ':'
 7 mov ah, 02h
 8 int 21h
 9 mov al, cl
10 call print2
11 ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
▶ ret
```

Return from the procedure.

Step 27 / 28

VARIABLES

AX = **0235h** BX = **3335h** CX = **0E23h**

DX = **0735h** SP = **FFFEh**

CALL STACK

empty

OUTPUT

```
14:35
```

Code Trace: Printing the Time as HH:MM

```
1  org 100h
2  mov ah, 2Ch        ; get system time
3  int 21h            ; CH=hour CL=min
4  mov al, ch
5  call print2
6  mov dl, ':'
7  mov ah, 02h
8  int 21h
9  mov al, cl
10 call print2
▶ ret
12 print2:            ; AL (0-99) → 2 digits
13 aam                ; AH=tens AL=ones
14 add ax, 3030h      ; ASCII
15 mov bx, ax
16 mov dl, bh
17 mov ah, 02h
18 int 21h
19 mov dl, bl
20 int 21h
21 ret
```

RET: SP = 0000h; the INT 20h in the PSP ends the program. The screen shows **14:35**.

Step 28 / 28

VARIABLES

AX = **0235h** BX = **3335h** CX = **0E23h**
DX = **0735h** SP = **0000h**

CALL STACK

empty

OUTPUT

```
14:35
```

INT 1Ah: Tick Counter and Real-Time Clock

AH	Function	Returns
`00h`	read tick counter	CX:DX = ticks since midnight, AL = midnight-rollover flag
`02h`	read RTC time	CH = hour, CL = minute, DH = second (**BCD**), DL = daylight saving; **CF = 1**: error
`04h`	read RTC date	CH = century, CL = year, DH = month, DL = day (BCD); CF = 1: error

The RTC is the battery-backed CMOS clock (AT and later); it keeps running while the computer is off.

Packed BCD → ASCII (CL = 35h, i.e. minute 35)

`35h = 0011 0101b`

Each nibble is one decimal digit: high nibble 3, low nibble 5.

Among the interrupts emu8086 supports, INT 1Ah has only **00h** (tick counter); 02h works under real DOS or DOSBox. In emu8086, read the time with INT 21h/**2Ch**.

INT 1Ah: Tick Counter and Real-Time Clock

AH	Function	Returns
`00h`	read tick counter	CX:DX = ticks since midnight, AL = midnight-rollover flag
`02h`	read RTC time	CH = hour, CL = minute, DH = second (**BCD**), DL = daylight saving; **CF = 1**: error
`04h`	read RTC date	CH = century, CL = year, DH = month, DL = day (BCD); CF = 1: error

The RTC is the battery-backed CMOS clock (AT and later); it keeps running while the computer is off.

Packed BCD → ASCII (CL = 35h, i.e. minute 35)

`shr dl, cl` (CL = 4) → **`03h`**

The high digit is shifted right. On the 8086 a shift count above 1 must be in CL.

Among the interrupts emu8086 supports, INT 1Ah has only **00h** (tick counter); 02h works under real DOS or DOSBox. In emu8086, read the time with INT 21h/**2Ch**.

INT 1Ah: Tick Counter and Real-Time Clock

AH	Function	Returns
`00h`	read tick counter	CX:DX = ticks since midnight, AL = midnight-rollover flag
`02h`	read RTC time	CH = hour, CL = minute, DH = second (**BCD**), DL = daylight saving; **CF = 1**: error
`04h`	read RTC date	CH = century, CL = year, DH = month, DL = day (BCD); CF = 1: error

The RTC is the battery-backed CMOS clock (AT and later); it keeps running while the computer is off.

Packed BCD → ASCII (CL = 35h, i.e. minute 35)

`and dl, 0Fh` → **`05h`**

The low digit is isolated.

Step 3 / 4

Among the interrupts emu8086 supports, INT 1Ah has only **00h** (tick counter); 02h works under real DOS or DOSBox. In emu8086, read the time with INT 21h/**2Ch**.

INT 1Ah: Tick Counter and Real-Time Clock

AH	Function	Returns
`00h`	read tick counter	CX:DX = ticks since midnight, AL = midnight-rollover flag
`02h`	read RTC time	CH = hour, CL = minute, DH = second (**BCD**), DL = daylight saving; **CF = 1**: error
`04h`	read RTC date	CH = century, CL = year, DH = month, DL = day (BCD); CF = 1: error

The RTC is the battery-backed CMOS clock (AT and later); it keeps running while the computer is off.

Packed BCD → ASCII (CL = 35h, i.e. minute 35)

`+30h → '3' 33h, '5' 35h`

Each digit becomes an ASCII digit. INT 21h/2Ch returns the same minute in binary, CL = 23h, which needs AAM instead.

Among the interrupts emu8086 supports, INT 1Ah has only **00h** (tick counter); 02h works under real DOS or DOSBox. In emu8086, read the time with INT 21h/**2Ch**.

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

AH = 02h: read the BIOS real-time clock (RTC).

Step 1 / 37

VARIABLES

AX = **0200h** BX = **0000h** CX = **0000h** DX = **0000h** CF = **0**

CALL STACK

empty

OUTPUT

Code Trace: Printing the RTC Time as HH:MM:SS

```
1  org 100h
2  mov ah, 02h
3  int 1Ah           ; CH=hr CL=min DH=sec
4  jc  done          ; CF=1: clock error
5  mov bl, cl        ; save the minute
6  mov cl, 4         ; SHR count
7  mov bh, ch
8  call putbcd       ; "14"
9  mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

The result is **packed BCD**: CH = **14h** (hour 14), CL = **35h** (minute 35), DH = **07h** (second). DL = 00h (no daylight saving).

Step 2 / 37

VARIABLES

AX = **0200h** BX = **0000h** CX = **1435h** DX = **0700h** CF = **0**

CALL STACK

empty

OUTPUT

Code Trace: Printing the RTC Time as HH:MM:SS

```
1  org 100h
2  mov ah, 02h
3  int 1Ah           ; CH=hr CL=min DH=sec
4  jc  done          ; CF=1: clock error
5  mov bl, cl        ; save the minute
6  mov cl, 4         ; SHR count
7  mov bh, ch
8  call putbcd       ; "14"
9  mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

CF = 0: the read succeeded. CF = 1 would mean the clock is not running; the program would end without printing.

Step 3 / 37

VARIABLES

AX = **0200h** BX = **0000h** CX = **1435h** DX = **0700h** CF = **0**

CALL STACK

empty

OUTPUT

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

CL is about to get the shift count, so the minute is copied to BL first.

Step 4 / 37

VARIABLES

AX = **0200h** BX = **0035h** CX = **1435h** DX = **0700h** CF = **0**

CALL STACK

empty

OUTPUT

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

CL = 4: `SHR DL, CL` will shift 4 bits (the 8086 has no immediate shift count above 1).

Step 5 / 37

VARIABLES

AX = **0200h** BX = **0035h** CX = **1404h** DX = **0700h** CF = **0**

CALL STACK

empty

OUTPUT

Code Trace: Printing the RTC Time as HH:MM:SS

```
1  org 100h
2  mov ah, 02h
3  int 1Ah           ; CH=hr CL=min DH=sec
4  jc  done          ; CF=1: clock error
5  mov bl, cl        ; save the minute
6  mov cl, 4         ; SHR count
▶  mov bh, ch
8  call putbcd       ; "14"
9  mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

BH = hour = 14h. The procedure expects the value in BH; INT 21h/02h does not touch BH.

Step 6 / 37

VARIABLES

AX = **0200h** BX = **1435h** CX = **1404h** DX = **0700h** CF = **0**

CALL STACK

empty

OUTPUT

Code Trace: Printing the RTC Time as HH:MM:SS

```
1  org 100h
2  mov ah, 02h
3  int 1Ah          ; CH=hr CL=min DH=sec
4  jc  done         ; CF=1: clock error
5  mov bl, cl       ; save the minute
6  mov cl, 4        ; SHR count
7  mov bh, ch
▶  call putbcd      ; "14"
9  mov bh, bl
10 call colon       ; ":35"
11 mov bh, dh
12 call colon       ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:          ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl       ; high digit
20 add dl, 30h      ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh      ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

The procedure is called for the hour (skipping colon, straight to putbcd).

Step 7 / 37

VARIABLES

AX = **0200h** BX = **1435h** CX = **1404h** DX = **0700h** CF = **0**

CALL STACK

FFFCh: 010Fh

OUTPUT

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
▶ mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

DL = BH = 14h = 0001 0100b.

Step 8 / 37

VARIABLES

AX = **0200h** BX = **1435h** CX = **1404h**

DX = **0714h** CF = **0**

CALL STACK

FFFCh: 010Fh

OUTPUT

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
▶ shr dl, cl         ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

`shr dl, cl` : 0001 0100b → 0000 0001b. DL = **01h**, the high digit.

Step 9 / 37

VARIABLES

AX = **0200h** BX = **1435h** CX = **1404h**

DX = **0701h** CF = **0**

CALL STACK

FFFCh: 010Fh

OUTPUT

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
▶ add dl, 30h        ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

+30h: DL = 31h = **'1'**.

Step 10 / 37

VARIABLES

AX = **0200h** BX = **1435h** CX = **1404h**

DX = **0731h** CF = **0**

CALL STACK

FFFCh: 010Fh

OUTPUT

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
▶ mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

AH = 02h: write the character in DL.

Step 11 / 37

VARIABLES

AX = **0200h** BX = **1435h** CX = **1404h** DX = **0731h** CF = **0**

CALL STACK

FFFCh: 010Fh

OUTPUT

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
▶ int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

'1' on the screen. DOS copies DL into AL (AL = 31h).

Step 12 / 37

VARIABLES

AX = 0231h BX = 1435h CX = 1404h DX = 0731h CF = 0

CALL STACK

FFFCh: 010Fh

OUTPUT

```
1
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
▶ mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

DL = BH = 14h again, for the second digit.

Step 13 / 37

VARIABLES

AX = **0231h** BX = **1435h** CX = **1404h**

DX = **0714h** CF = **0**

CALL STACK

FFFCh: 010Fh

OUTPUT

```
1
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
▶ and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

`and dl, 0Fh` : the high 4 bits are cleared: 0000 0100b = **04h**.

Step 14 / 37

VARIABLES

AX = **0231h** BX = **1435h** CX = **1404h**

DX = **0704h** CF = **0**

CALL STACK

```
FFFCh: 010Fh
```

OUTPUT

```
1
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
▶ add dl, 30h
26 int 21h
27 ret
```

+30h: DL = 34h = **'4'**.

Step 15 / 37

VARIABLES

AX = **0231h** BX = **1435h** CX = **1404h**

DX = **0734h** CF = **0**

CALL STACK

FFFCh: 010Fh

OUTPUT

```
1
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
▶ int 21h
27 ret
```

'4' on the screen → **14**. AH is still 02h, so no reload is needed.

Step 16 / 37

VARIABLES

AX = **0234h** BX = **1435h** CX = **1404h** DX = **0734h** CF = **0**

CALL STACK

FFFCh: 010Fh

OUTPUT

```
14
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
   ret
```

Return from the procedure.

Step 17 / 37

VARIABLES

AX = **0234h** BX = **1435h** CX = **1404h** DX = **0734h** CF = **0**

CALL STACK

empty

OUTPUT

```
14
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
1  org 100h
2  mov ah, 02h
3  int 1Ah           ; CH=hr CL=min DH=sec
4  jc  done          ; CF=1: clock error
5  mov bl, cl        ; save the minute
6  mov cl, 4         ; SHR count
7  mov bh, ch
8  call putbcd       ; "14"
▶  mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

BH = minute = 35h (saved in BL).

Step 18 / 37

VARIABLES

AX = **0234h** BX = **3535h** CX = **1404h** DX = **0734h** CF = **0**

CALL STACK

empty

OUTPUT

```
14
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
▶10 call colon       ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

This time the **colon** entry is called: it prints ':' first, then falls into putbcd.

Step 19 / 37

VARIABLES

AX = **0234h** BX = **3535h** CX = **1404h** DX = **0734h** CF = **0**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
▶ colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

DL = ':' (3Ah).

Step 20 / 37

VARIABLES

AX = **0234h** BX = **3535h** CX = **1404h**

DX = **073Ah** CF = **0**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
▶ mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

AH = 02h.

Step 21 / 37

VARIABLES

AX = **0234h** BX = **3535h** CX = **1404h**

DX = **073Ah** CF = **0**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
▶ int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

':' on the screen → **14:**. Execution then falls through into the putbcd label.

Step 22 / 37

VARIABLES

AX = **023Ah** BX = **3535h** CX = **1404h** DX = **073Ah** CF = **0**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
▶ shr dl, cl         ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

35h → **03h** (high digit).

Step 23 / 37

VARIABLES

AX = **023Ah** BX = **3535h** CX = **1404h**

DX = **0703h** CF = **0**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
▶ int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

'3' (33h) on the screen.

Step 24 / 37

VARIABLES

AX = 0233h BX = 3535h CX = 1404h DX = 0733h CF = 0

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:3
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
▶ and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

35h AND 0Fh = **05h**.

Step 25 / 37

VARIABLES

AX = **0233h** BX = **3535h** CX = **1404h**

DX = **0705h** CF = **0**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:3
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
▶ int 21h
27 ret
```

'5' on the screen → **14:35**.

Step 26 / 37

VARIABLES

AX = **0235h** BX = **3535h** CX = **1404h**

DX = **0735h** CF = **0**

CALL STACK

FFFCh: 0114h

OUTPUT

```
14:35
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
   ret
```

Return.

Step 27 / 37

VARIABLES

AX = **0235h** BX = **3535h** CX = **1404h**

DX = **0735h** CF = **0**

CALL STACK

empty

OUTPUT

```
14:35
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
▶ mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

BH = second = 07h. INT 21h/02h left DH alone.

Step 28 / 37

VARIABLES

AX = **0235h** BX = **0735h** CX = **1404h** DX = **0735h** CF = **0**

CALL STACK

empty

OUTPUT

```
14:35
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
▶ call colon         ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

colon again, for the seconds.

Step 29 / 37

VARIABLES

AX = **0235h** BX = **0735h** CX = **1404h**

DX = **0735h** CF = **0**

CALL STACK

FFFCh: 0119h

OUTPUT

```
14:35
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
▶ colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

DL = ':'.

Step 30 / 37

VARIABLES

AX = **0235h** BX = **0735h** CX = **1404h**

DX = **073Ah** CF = **0**

CALL STACK

FFFCh: 0119h

OUTPUT

```
14:35
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
▶ int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

':' on the screen → **14:35:**.

Step 31 / 37

VARIABLES

AX = **023Ah** BX = **0735h** CX = **1404h**

DX = **073Ah** CF = **0**

CALL STACK

FFFCh: 0119h

OUTPUT

```
14:35:
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
▶ shr dl, cl         ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

07h → **00h** (high digit).

Step 32 / 37

VARIABLES

AX = **023Ah** BX = **0735h** CX = **1404h**

DX = **0700h** CF = **0**

CALL STACK

FFFCh: 0119h

OUTPUT

```
14:35:
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
▶ int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

'0' (30h) on the screen.

Step 33 / 37

VARIABLES

AX = 0230h BX = 0735h CX = 1404h DX = 0730h CF = 0

CALL STACK

FFFCh: 0119h

OUTPUT

```
14:35:0
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
▶ and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

07h AND 0Fh = **07h**.

Step 34 / 37

VARIABLES

AX = **0230h** BX = **0735h** CX = **1404h**

DX = **0707h** CF = **0**

CALL STACK

FFFCh: 0119h

OUTPUT

```
14:35:0
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
▶ int 21h
27 ret
```

'7' on the screen → **14:35:07**.

Step 35 / 37

VARIABLES

AX = **0237h** BX = **0735h** CX = **1404h**

DX = **0737h** CF = **0**

CALL STACK

FFFCh: 0119h

OUTPUT

```
14:35:07
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
13 done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
   ret
```

Return.

Step 36 / 37

VARIABLES

AX = **0237h** BX = **0735h** CX = **1404h**

DX = **0737h** CF = **0**

CALL STACK

empty

OUTPUT

```
14:35:07
```

Code Trace: Printing the RTC Time as HH:MM:SS

```
 1 org 100h
 2 mov ah, 02h
 3 int 1Ah           ; CH=hr CL=min DH=sec
 4 jc  done          ; CF=1: clock error
 5 mov bl, cl        ; save the minute
 6 mov cl, 4         ; SHR count
 7 mov bh, ch
 8 call putbcd       ; "14"
 9 mov bh, bl
10 call colon        ; ":35"
11 mov bh, dh
12 call colon        ; ":07"
▶ done: ret
14 colon: mov dl, ':'
15 mov ah, 02h
16 int 21h
17 putbcd:           ; BH = packed BCD
18 mov dl, bh
19 shr dl, cl        ; high digit
20 add dl, 30h       ; ASCII digit
21 mov ah, 02h
22 int 21h
23 mov dl, bh
24 and dl, 0Fh       ; low digit
25 add dl, 30h
26 int 21h
27 ret
```

The screen shows **14:35:07**. INT 21h/2Ch would return the same time in binary (CH = 0Eh, CL = 23h); in BCD the hex digits are the decimal digits.

Step 37 / 37

VARIABLES

AX = **0237h** BX = **0735h** CX = **1404h**

DX = **0737h** CF = **0**

CALL STACK

empty

OUTPUT

```
14:35:07
```

The BIOS Data Area (Segment 0040h)

Address	Size	Contents
`0040:0010`	word	equipment list (the value INT 11h returns)
`0040:0013`	word	base memory in KB (INT 12h)
`0040:0017`	byte	keyboard flags: bit 0 right Shift, 1 left Shift, 2 Ctrl, 3 Alt, 4 Scroll Lock, 5 Num Lock, 6 Caps Lock, 7 Insert
`0040:001A`	word	buffer **head**: offset of the next key to read
`0040:001C`	word	buffer **tail**: offset of the next free slot (head = tail → buffer empty)
`0040:001E–003D`	32 bytes	keyboard buffer: 16 words (ASCII + scan code), circular; at most 15 keys
`0040:0049`	byte	current video mode (03h, 13h, …)
`0040:004A`	word	columns on screen (80 in mode 03h)
`0040:0050`	8 words	cursor position for the 8 pages (low byte column, high byte row)
`0040:006C`	dword	**tick counter**: number of ≈ 18.2 Hz interrupts since midnight
`0040:0070`	byte	midnight-passed flag (INT 1Ah/00h returns it in AL and clears it)

Physical address = 00400h + offset: 0040:006C = 0046Ch = 0000:046C. INT 16h and INT 1Ah use this area; a program can read it directly too.

Reading the BIOS Data Area

```
; wait about 1 second
mov ax, 0040h
mov es, ax
mov bx, es:[006Ch]  ; starting tick
wait1:
mov ax, es:[006Ch]  ; current tick
sub ax, bx          ; ticks elapsed
cmp ax, 18          ; 18 ticks ≈ 0.99 s
jb  wait1

; Caps Lock on? Any key waiting?
test byte ptr es:[0017h], 40h
jnz caps_on         ; bit 6 = 1
mov ax, es:[001Ah]  ; buffer head
cmp ax, es:[001Ch]  ; buffer tail
je  buf_empty
```

1,193,182 Hz ÷ 65,536 = **18.2065 Hz**

With a divisor of 65,536 the 8253/8254 timer raises IRQ 0 at this rate; each interrupt adds one to 0040:006C.

Step 1 / 4

emu8086 does not update the whole BIOS data area as a real PC does; the tick counter and keyboard buffer examples work fully in DOSBox.

Reading the BIOS Data Area

```
; wait about 1 second
mov ax, 0040h
mov es, ax
mov bx, es:[006Ch]  ; starting tick
wait1:
mov ax, es:[006Ch]  ; current tick
sub ax, bx          ; ticks elapsed
cmp ax, 18          ; 18 ticks ≈ 0.99 s
jb  wait1

; Caps Lock on? Any key waiting?
test byte ptr es:[0017h], 40h
jnz caps_on         ; bit 6 = 1
mov ax, es:[001Ah]  ; buffer head
cmp ax, es:[001Ch]  ; buffer tail
je  buf_empty
```

1 tick = 65,536 ÷ 1,193,182 s ≈ **54.9 ms**
18 ticks ≈ 0.989 s · 5 s ≈ 91 ticks

Delays are measured in ticks; the resolution is about 55 ms.

emu8086 does not update the whole BIOS data area as a real PC does; the tick counter and keyboard buffer examples work fully in DOSBox.

Reading the BIOS Data Area

```
; wait about 1 second
mov ax, 0040h
mov es, ax
mov bx, es:[006Ch]  ; starting tick
wait1:
mov ax, es:[006Ch]  ; current tick
sub ax, bx          ; ticks elapsed
cmp ax, 18          ; 18 ticks ≈ 0.99 s
jb  wait1

; Caps Lock on? Any key waiting?
test byte ptr es:[0017h], 40h
jnz caps_on         ; bit 6 = 1
mov ax, es:[001Ah]  ; buffer head
cmp ax, es:[001Ch]  ; buffer tail
je  buf_empty
```

1800B0h = **1,573,040** ticks a day

When the counter reaches this value it resets and the 0040:0070 flag is set.

emu8086 does not update the whole BIOS data area as a real PC does; the tick counter and keyboard buffer examples work fully in DOSBox.

Reading the BIOS Data Area

```
; wait about 1 second
mov ax, 0040h
mov es, ax
mov bx, es:[006Ch]  ; starting tick
wait1:
mov ax, es:[006Ch]  ; current tick
sub ax, bx          ; ticks elapsed
cmp ax, 18          ; 18 ticks ≈ 0.99 s
jb  wait1

; Caps Lock on? Any key waiting?
test byte ptr es:[0017h], 40h
jnz caps_on         ; bit 6 = 1
mov ax, es:[001Ah]  ; buffer head
cmp ax, es:[001Ch]  ; buffer tail
je  buf_empty
```

The difference with `sub ax, bx`

Only the low word is read. Because we take a difference, the result is right even if the low word wraps from FFFFh to 0000h; only the midnight reset spoils it.

emu8086 does not update the whole BIOS data area as a real PC does; the tick counter and keyboard buffer examples work fully in DOSBox.

ENDING A PROGRAM

Three Ways to End a Program

Method	How does it work?	Where?
`ret`	In a COM program the top of the stack holds 0000h; RET returns to PSP:0000h, where `int 20h` (CD 20) sits.	COM only, with a balanced stack
`int 20h`	DOS terminates the program; CS must point to the PSP segment.	COM
`mov ax, 4C00h` `int 21h`	Terminates with the **exit code** in AL and closes open files.	COM and EXE · **recommended**

Exit code

AL = 0 means success, nonzero an error. A batch file tests it with `IF ERRORLEVEL 1`.

A common mistake

Forget the exit instruction and the processor executes the bytes after the program (the data!) as code; the system may hang.

PSP (Program Segment Prefix): the 256-byte area DOS places in front of every program; that is why COM code starts at ORG 100h.

emu8086 Templates: COM and EXE

COM (single segment)

```
org 100h
  mov dx, offset msg
  mov ah, 09h
  int 21h
  ret              ; or int 20h
msg db 'Welcome$'
```

- CS = DS = ES = SS; no setup needed.
- At most 64 KB (code + data + stack).
- The file is pure machine code (no header); execution starts at 0100h.

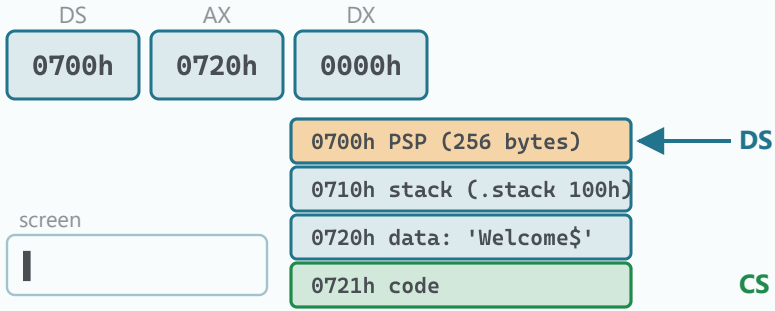
EXE (separate segments)

```
.model small
.stack 100h
.data
msg db 'Welcome$'
.code
start:
  mov ax, @data
  mov ds, ax       ; set DS!
  mov dx, offset msg
  mov ah, 09h
  int 21h
  mov ax, 4C00h
  int 21h
end start
```

An EXE file carries a header with relocation information, names its entry point with `end start`, and may be larger than 64 KB. DOS sets DS to the PSP at startup; without `mov ax, @data / mov ds, ax`, DS:DX points to the wrong place and garbage is printed.

Setting DS in an EXE Program

```
1  .model small
2  .stack 100h
3  .data
4  msg db 'Welcome$'
5  .code
6  start:
▶  mov ax, @data
8  mov ds, ax      ; set DS!
9  mov dx, offset msg
10 mov ah, 09h
11 int 21h
12 mov ax, 4C00h
13 int 21h
14 end start
```



At startup DS = ES = **0700h**: DOS points them at the PSP. CS = 0721h (code), SS = 0710h (stack). `@data` is the data segment value → AX = 0720h.

Step 1 / 6

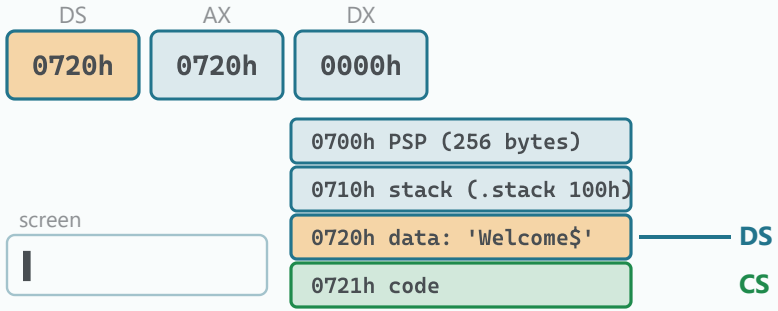
VARIABLES

DS = **0700h** AX = **0720h** DX = **0000h** CS = **0721h**

The segment values are the interpreter's layout (PSP at 0700h); under real DOS the numbers may differ, but the order and the need to set DS do not.

Setting DS in an EXE Program

```
1  .model small
2  .stack 100h
3  .data
4  msg db 'Welcome$'
5  .code
6  start:
7  mov ax, @data
▶  mov ds, ax       ; set DS!
9  mov dx, offset msg
10 mov ah, 09h
11 int 21h
12 mov ax, 4C00h
13 int 21h
14 end start
```



MOV DS, AX: DS = 0720h now points at the data segment. Without this line DS:DX would point into the PSP and print garbage.

Step 2 / 6

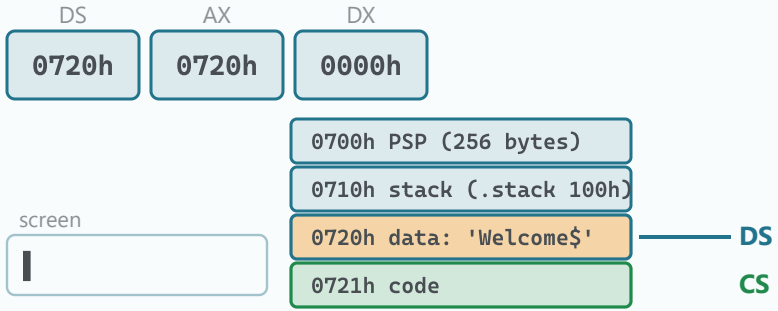
VARIABLES

DS = 0720h | AX = 0720h | DX = 0000h | CS = 0721h

The segment values are the interpreter's layout (PSP at 0700h); under real DOS the numbers may differ, but the order and the need to set DS do not.

Setting DS in an EXE Program

```
1  .model small
2  .stack 100h
3  .data
4  msg db 'Welcome$'
5  .code
6  start:
7  mov ax, @data
8  mov ds, ax      ; set DS!
▶  mov dx, offset msg
10 mov ah, 09h
11 int 21h
12 mov ax, 4C00h
13 int 21h
14 end start
```



DX = offset msg = 0000h: msg is at the start of the data segment.

Step 3 / 6

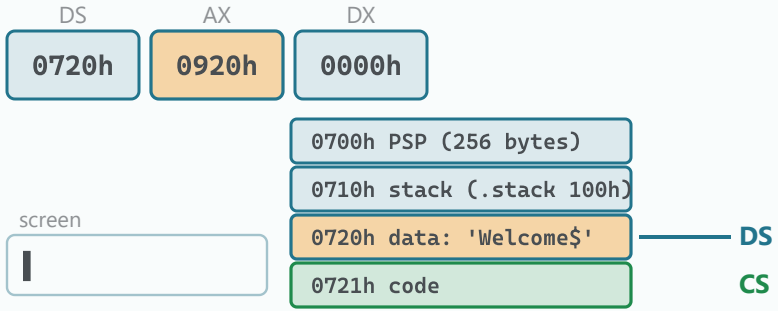
VARIABLES

DS = **0720h** AX = **0720h** DX = **0000h** CS = **0721h**

The segment values are the interpreter's layout (PSP at 0700h); under real DOS the numbers may differ, but the order and the need to set DS do not.

Setting DS in an EXE Program

```
1  .model small
2  .stack 100h
3  .data
4  msg db 'Welcome$'
5  .code
6  start:
7  mov ax, @data
8  mov ds, ax      ; set DS!
9  mov dx, offset msg
▶  mov ah, 09h
11 int 21h
12 mov ax, 4C00h
13 int 21h
14 end start
```



AH = 09h.

Step 4 / 6

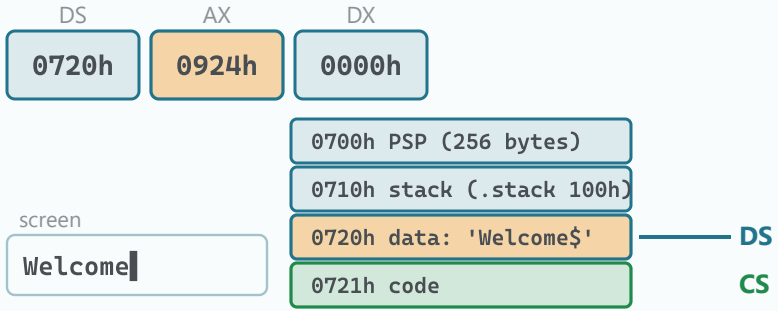
VARIABLES

DS = 0720h | AX = 0920h | DX = 0000h | CS = 0721h

The segment values are the interpreter's layout (PSP at 0700h); under real DOS the numbers may differ, but the order and the need to set DS do not.

Setting DS in an EXE Program

```
1  .model small
2  .stack 100h
3  .data
4  msg db 'Welcome$'
5  .code
6  start:
7  mov ax, @data
8  mov ds, ax      ; set DS!
9  mov dx, offset msg
10 mov ah, 09h
▶  int 21h
12 mov ax, 4C00h
13 int 21h
14 end start
```



INT 21h: from DS:DX = 0720h:0000h up to '$' → **Welcome**.

Step 5 / 6

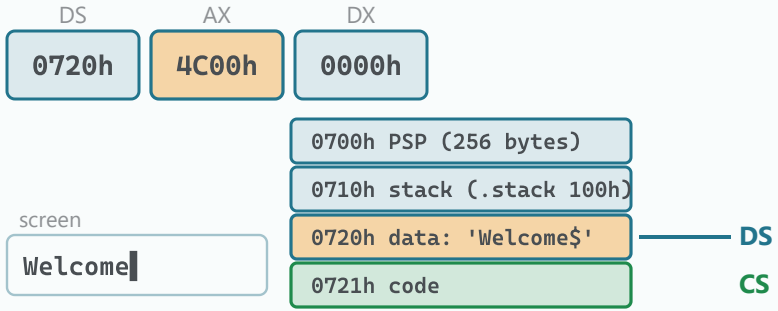
VARIABLES

DS = 0720h | AX = 0924h | DX = 0000h | CS = 0721h

The segment values are the interpreter's layout (PSP at 0700h); under real DOS the numbers may differ, but the order and the need to set DS do not.

Setting DS in an EXE Program

```
1  .model small
2  .stack 100h
3  .data
4  msg db 'Welcome$'
5  .code
6  start:
7  mov ax, @data
8  mov ds, ax      ; set DS!
9  mov dx, offset msg
10 mov ah, 09h
11 int 21h
   mov ax, 4C00h
   int 21h
14 end start
```



AX = 4C00h, INT 21h: back to DOS with exit code 0. RET cannot be used in an EXE (no PSP address on top of the stack).

Step 6 / 6

VARIABLES

DS = 0720h | AX = 4C00h | DX = 0000h | CS = 0721h

The segment values are the interpreter's layout (PSP at 0700h); under real DOS the numbers may differ, but the order and the need to set DS do not.

APPLICATION

Add, Subtract and Print the Result

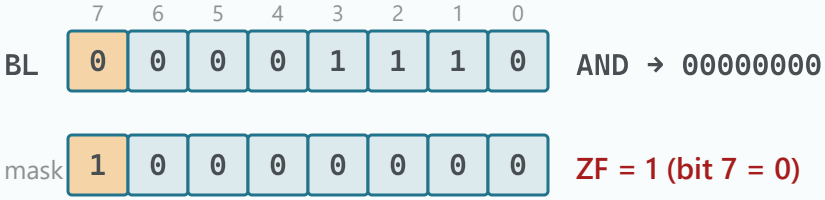
```
org 100h
mov al, 5       ; 00000101b
mov bl, 10      ; 0Ah = 00001010b
; 5 + 10 = 15 = 0Fh = 00001111b
add bl, al
; 15 - 1 = 14 = 0Eh = 00001110b
sub bl, 1
mov cx, 8       ; print result in binary
```

```
print:
mov ah, 2   ; print function
mov dl, '0'
test bl, 10000000b  ; test top bit
jz zero
mov dl, '1'
zero:
int 21h
shl bl, 1
loop print
; print the binary suffix b
mov dl, 'b'
int 21h
; wait for a key press
mov ah, 0
int 16h
ret
```

- Computes 5 + 10 − 1 = **14 = 0Eh**; the result is in BL.
- The loop runs 8 times (CX = 8): each pass tests the **most significant bit** (MSB) of BL, prints '0' or '1', and shifts BL left by one.
- Finally it prints the suffix 'b' and waits for a key. Expected output: **00001110b**.

APPLICATION

Reading Bits with TEST and SHL



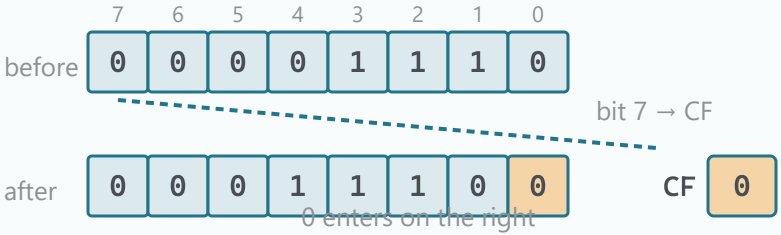
`TEST BL, 10000000b` : BL is ANDed with the mask but **not changed**. If the result is 0, bit 7 = 0 → **ZF = 1** and `JZ` jumps.

Step 1 / 3

Alternative: `SHL BL,1` and then `JC` on CF. In that order, though, the shift must come *before* printing.

APPLICATION

Reading Bits with TEST and SHL



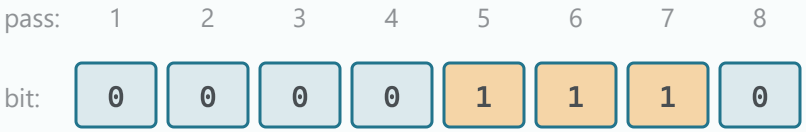
`SHL BL, 1` : every bit moves left; bit 7 goes out into **CF** and a 0 enters bit 0. The next bit moves into the tested position (bit 7).

Step 2 / 3

Alternative: `SHL BL,1` and then `JC` on CF. In that order, though, the shift must come *before* printing.

APPLICATION

Reading Bits with TEST and SHL



After 8 passes BL's bits have been printed from MSB to LSB: **00001110**, then **b**.

Step 3 / 3

Alternative: `SHL BL,1` and then `JC` on CF. In that order, though, the shift must come *before* printing.

Code Trace (1/2): The Calculation

```
1 org 100h
▶ mov al, 5        ; 00000101b
3 mov bl, 10       ; 0Ah = 00001010b
4 ; 5 + 10 = 15 = 0Fh = 00001111b
5 add bl, al
6 ; 15 - 1 = 14 = 0Eh = 00001110b
7 sub bl, 1
8 mov cx, 8        ; print result in binary
```

AL = 5 = 05h = 00000101b. MOV does not affect the flags.

Step 1 / 5

VARIABLES

AL = **05h** BL = **00h** CX = **0000h**

CF = **0** ZF = **0** SF = **0** PF = **0**

AF = **0**

Code Trace (1/2): The Calculation

```
1 org 100h
2 mov al, 5        ; 00000101b
▶ mov bl, 10       ; 0Ah = 00001010b
4 ; 5 + 10 = 15 = 0Fh = 00001111b
5 add bl, al
6 ; 15 - 1 = 14 = 0Eh = 00001110b
7 sub bl, 1
8 mov cx, 8        ; print result in binary
```

BL = 10 = 0Ah = 00001010b.

Step 2 / 5

VARIABLES

AL = **05h** | BL = **0Ah** | CX = **0000h**

CF = **0** | ZF = **0** | SF = **0** | PF = **0**

AF = **0**

Code Trace (1/2): The Calculation

```
1 org 100h
2 mov al, 5        ; 00000101b
3 mov bl, 10       ; 0Ah = 00001010b
4 ; 5 + 10 = 15 = 0Fh = 00001111b
▶ add bl, al
6 ; 15 - 1 = 14 = 0Eh = 00001110b
7 sub bl, 1
8 mov cx, 8        ; print result in binary
```

`ADD BL, AL` : BL = 0Ah + 05h = **0Fh** (15). No carry (CF = 0); the result has four 1 bits → PF = 1.

Step 3 / 5

VARIABLES

AL = **05h** | BL = **0Fh** | CX = **0000h**

CF = **0** | ZF = **0** | SF = **0** | PF = **1**

AF = **0**

Code Trace (1/2): The Calculation

```
1 org 100h
2 mov al, 5        ; 00000101b
3 mov bl, 10       ; 0Ah = 00001010b
4 ; 5 + 10 = 15 = 0Fh = 00001111b
5 add bl, al
6 ; 15 - 1 = 14 = 0Eh = 00001110b
▶ sub bl, 1
8 mov cx, 8        ; print result in binary
```

`SUB BL, 1` : BL = **0Eh** (14) = 00001110b. Three 1 bits → PF = 0. No borrow: CF = 0.

Step 4 / 5

VARIABLES

AL = **05h** | BL = **0Eh** | CX = **0000h**

CF = **0** | ZF = **0** | SF = **0** | PF = **0**

AF = **0**

Code Trace (1/2): The Calculation

```
1 org 100h
2 mov al, 5        ; 00000101b
3 mov bl, 10       ; 0Ah = 00001010b
4 ; 5 + 10 = 15 = 0Fh = 00001111b
5 add bl, al
6 ; 15 - 1 = 14 = 0Eh = 00001110b
7 sub bl, 1
▶ mov cx, 8        ; print result in binary
```

CX = 8: the `LOOP` counter, one pass per bit.

Step 5 / 5

VARIABLES

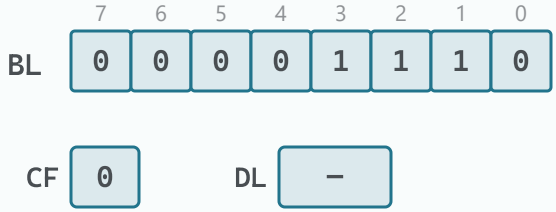
AL = **05h** BL = **0Eh** CX = **0008h**

CF = **0** ZF = **0** SF = **0** PF = **0**

AF = **0**

Code Trace (2/2): Printing in Binary

```
1  print:
▶  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



Pass 1: AH = 2 (DOS write character).

Step 1 / 26

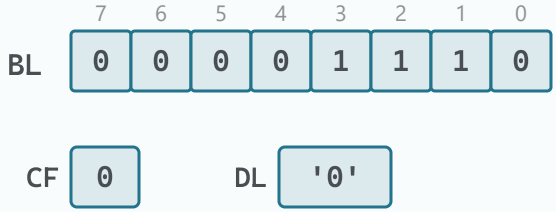
VARIABLES

AH = 02h | DL = 00h | BL = 0Eh | CX = 0008h | ZF = 0 | CF = 0

OUTPUT

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
▶  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



Default character DL = '0' (30h).

Step 2 / 26

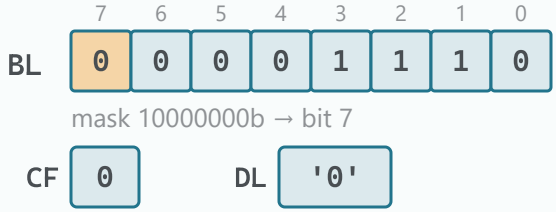
VARIABLES

AH = 02h | DL = 30h | BL = 0Eh | CX = 0008h | ZF = 0 | CF = 0

OUTPUT

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
▶  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



`TEST` : bit 7 = 0 → **ZF = 1**, CF = 0.

Step 3 / 26

VARIABLES

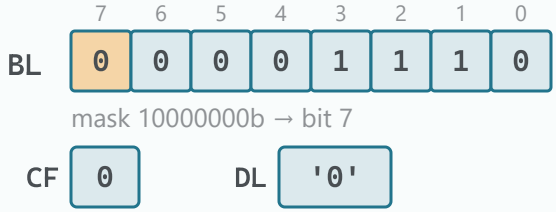
AH = **02h** DL = **30h** BL = **0Eh** CX = **0008h**

ZF = **1** CF = **0**

OUTPUT

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
▶  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



ZF = 1 → `JZ zero` **jumps**; DL stays '0'.

Step 4 / 26

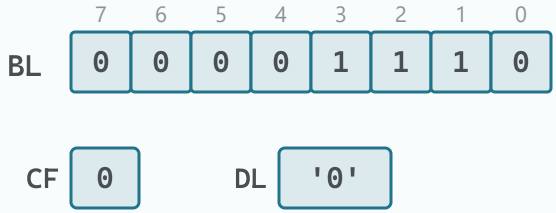
VARIABLES

AH = **02h** DL = **30h** BL = **0Eh** CX = **0008h**
ZF = **1** CF = **0**

OUTPUT

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
▶  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



`INT 21h` : the '0' in DL is printed.

Step 5 / 26

VARIABLES

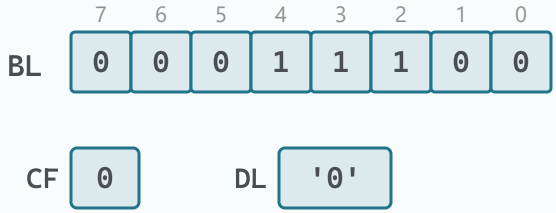
AH = **02h** DL = **30h** BL = **0Eh** CX = **0008h**
ZF = **1** CF = **0**

OUTPUT

```
0
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
▶  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



`SHL BL,1` : BL = 1Ch, the bit shifted out: CF = 0.

Step 6 / 26

VARIABLES

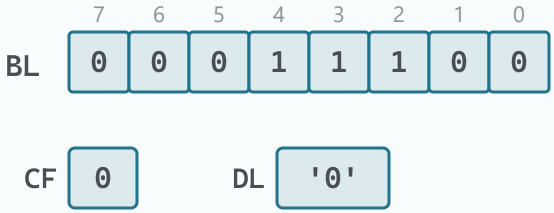
AH = 02h | DL = 30h | BL = 1Ch | CX = 0008h | ZF = 0 | CF = 0

OUTPUT

```
0
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
▶  loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



`LOOP` : CX = 7 ≠ 0 → back to `print` .

Step 7 / 26

VARIABLES

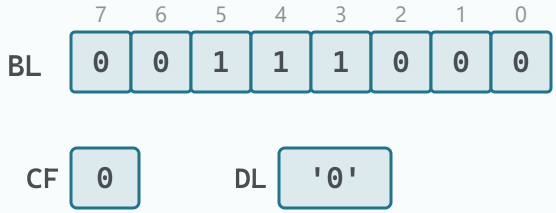
AH = **02h** DL = **30h** BL = **1Ch** CX = **0007h** ZF = **0** CF = **0**

OUTPUT

```
0
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
▶ loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



Pass 2 (summary): bit = 0, '0' printed, BL = 38h. `LOOP` : CX = 6 ≠ 0 → back to `print` .

Step 8 / 26

VARIABLES

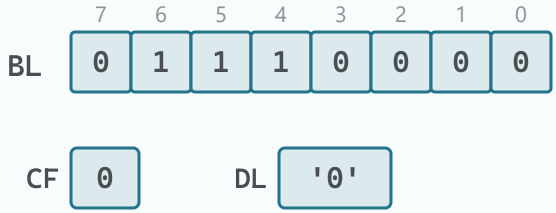
AH = **02h** DL = **30h** BL = **38h** CX = **0006h** ZF = **0** CF = **0**

OUTPUT

```
00
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
▶ 10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



Pass 3 (summary): bit = 0, '0' printed, BL = 70h. `LOOP` : CX = 5 ≠ 0 → back to `print` .

Step 9 / 26

VARIABLES

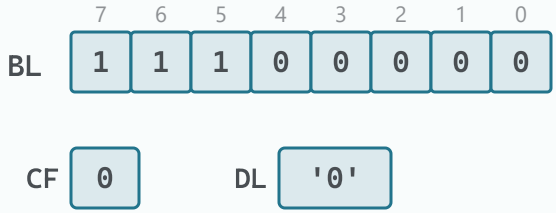
AH = **02h** DL = **30h** BL = **70h** CX = **0005h** ZF = **0** CF = **0**

OUTPUT

```
000
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
▶ 10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



Pass 4 (summary): bit = 0, '0' printed, BL = E0h. `LOOP`: CX = 4 ≠ 0 → back to `print`.

Step 10 / 26

VARIABLES

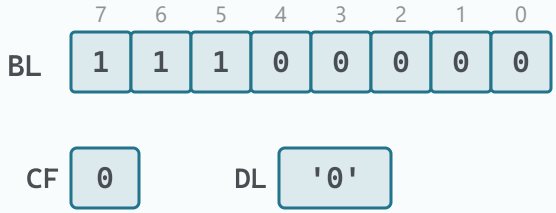
AH = **02h** | DL = **30h** | BL = **E0h** | CX = **0004h** | ZF = **0** | CF = **0**

OUTPUT

```
0000
```

Code Trace (2/2): Printing in Binary

```
1  print:
▶  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



Pass 5: AH = 2 (DOS write character).

Step 11 / 26

VARIABLES

AH = **02h** DL = **30h** BL = **E0h** CX = **0004h**

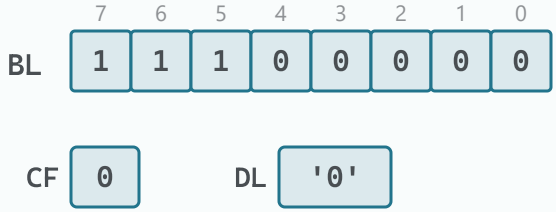
ZF = **0** CF = **0**

OUTPUT

```
0000
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
▶  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



Default character DL = '0' (30h).

Step 12 / 26

VARIABLES

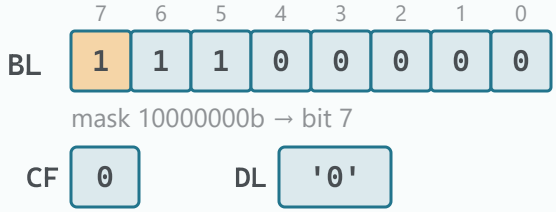
AH = **02h** DL = **30h** BL = **E0h** CX = **0004h**
ZF = **0** CF = **0**

OUTPUT

```
0000
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
▶  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



TEST : bit 7 = 1 → **ZF = 0**, CF = 0.

Step 13 / 26

VARIABLES

AH = **02h** DL = **30h** BL = **E0h** CX = **0004h**

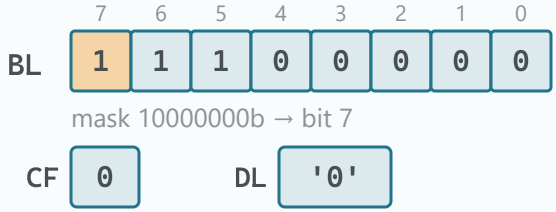
ZF = **0** CF = **0**

OUTPUT

```
0000
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
▶  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



ZF = 0 → `JZ` does not jump; the next instruction runs.

Step 14 / 26

VARIABLES

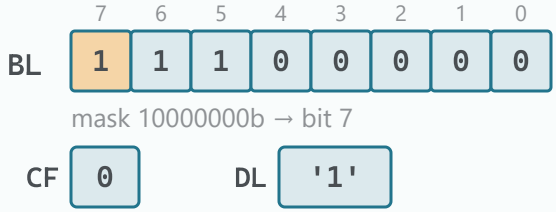
AH = **02h** DL = **30h** BL = **E0h** CX = **0004h**
ZF = **0** CF = **0**

OUTPUT

```
0000
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
▶  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



DL = '1' (31h).

Step 15 / 26

VARIABLES

AH = 02h | DL = 31h | BL = E0h | CX = 0004h | ZF = 0 | CF = 0

OUTPUT

```
0000
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
▶  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



CF `0` DL `'1'`

`INT 21h` : the '1' in DL is printed.

Step 16 / 26

VARIABLES

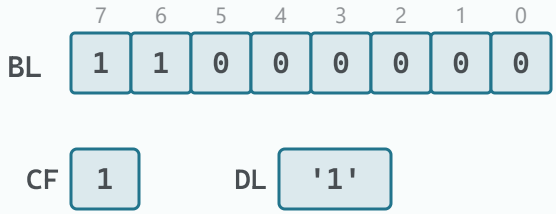
AH = **02h** | DL = **31h** | BL = **E0h** | CX = **0004h** | ZF = **0** | CF = **0**

OUTPUT

```
00001
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
▶  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



`SHL BL,1` : BL = C0h, the bit shifted out: CF = 1.

Step 17 / 26

VARIABLES

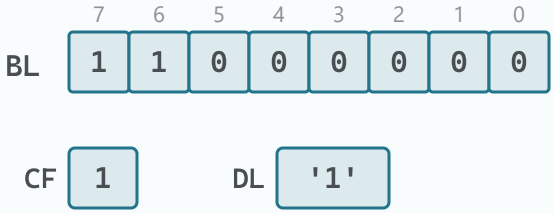
AH = **02h** | DL = **31h** | BL = **C0h** | CX = **0004h** | ZF = **0** | CF = **1**

OUTPUT

```
00001
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
▶  loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



`LOOP` : CX = 3 ≠ 0 → back to `print` .

Step 18 / 26

VARIABLES

AH = **02h** DL = **31h** BL = **C0h** CX = **0003h**

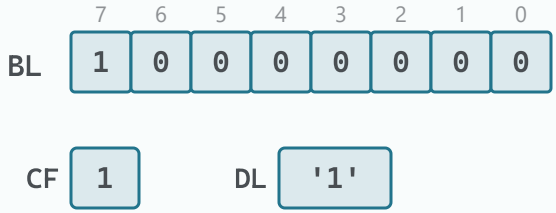
ZF = **0** CF = **1**

OUTPUT

```
00001
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
▶ loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



Pass 6 (summary): bit = 1, '1' printed, BL = 80h. `LOOP`: CX = 2 ≠ 0 → back to `print`.

Step 19 / 26

VARIABLES

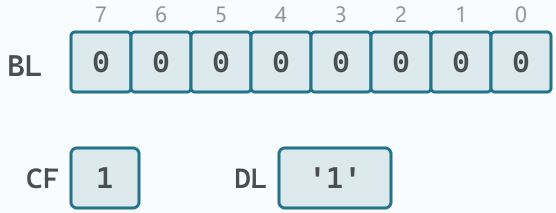
AH = **02h** | DL = **31h** | BL = **80h** | CX = **0002h** | ZF = **0** | CF = **1**

OUTPUT

```
000011
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
▶ loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



Pass 7 (summary): bit = 1, '1' printed, BL = 00h. `LOOP`: CX = 1 ≠ 0 → back to `print`.

Step 20 / 26

VARIABLES

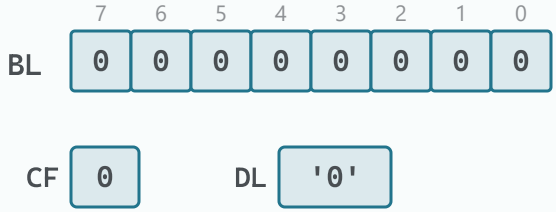
AH = **02h** DL = **31h** BL = **00h** CX = **0001h** ZF = **1** CF = **1**

OUTPUT

```
0000111
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
▶  loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



Pass 8 (summary): bit = 0, '0' printed, BL = 00h. `LOOP` : CX = 0 → the loop ends.

Step 21 / 26

VARIABLES

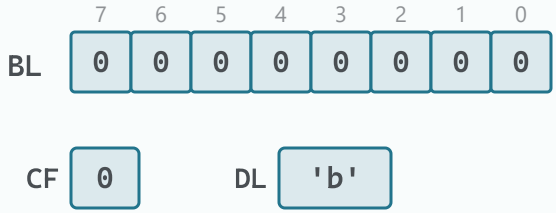
AH = 02h | DL = 30h | BL = 00h | CX = 0000h | ZF = 1 | CF = 0

OUTPUT

```
00001110
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
▶ mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



DL = 'b'. AH is still 2.

Step 22 / 26

VARIABLES

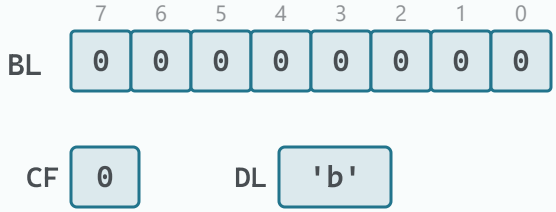
AH = 02h | DL = 62h | BL = 00h | CX = 0000h | ZF = 1 | CF = 0

OUTPUT

```
00001110
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
▶ int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
17 ret
```



`INT 21h` : the suffix 'b' is printed.

Step 23 / 26

VARIABLES

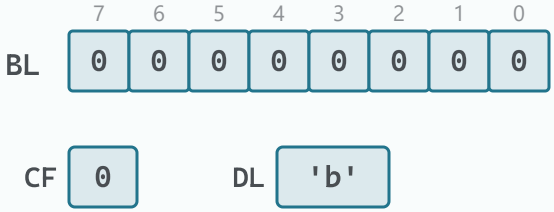
AH = **02h** DL = **62h** BL = **00h** CX = **0000h**
ZF = **1** CF = **0**

OUTPUT

```
00001110b
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
▶ mov ah, 0
16 int 16h
17 ret
```



AH = 0: wait for a key.

Step 24 / 26

VARIABLES

AH = 00h | DL = 62h | BL = 00h | CX = 0000h | ZF = 1 | CF = 0

OUTPUT

```
00001110b
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
▶  int 16h
17 ret
```





`INT 16h` : waits for a key (so the output can be read).
Enter: AH = 1Ch (scan code), AL = 0Dh.

Step 25 / 26

VARIABLES

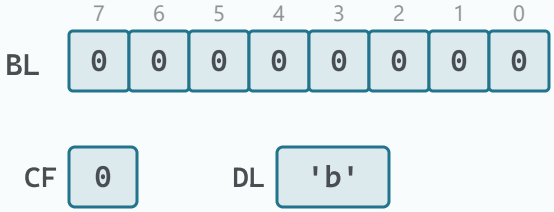
AH = **1Ch** | DL = **62h** | BL = **00h** | CX = **0000h** | ZF = **1** | CF = **0**

OUTPUT

```
00001110b
```

Code Trace (2/2): Printing in Binary

```
1  print:
2  mov ah, 2   ; print function
3  mov dl, '0'
4  test bl, 10000000b  ; test top bit
5  jz zero
6  mov dl, '1'
7  zero:
8  int 21h
9  shl bl, 1
10 loop print
11 ; print the binary suffix b
12 mov dl, 'b'
13 int 21h
14 ; wait for a key press
15 mov ah, 0
16 int 16h
▶  ret
```



`RET` : the program ends. The screen shows **00001110b**.

Step 26 / 26

VARIABLES

AH = **1Ch** | DL = **62h** | BL = **00h** | CX = **0000h** | ZF = **1** | CF = **0**

OUTPUT

```
00001110b
```

APPLICATION

Screen Output and Notes



emu8086 screen window (80×25)

- The number is already binary inside; printing it means turning each bit into a **'0'/'1' character**.
- `TEST` stores no result, so BL survives; only `SHL` changes it (on purpose).
- `LOOP` decrements CX; do not use CX for anything else inside the loop.
- `INT 21h/02h` changes AL (AL = DL); save any value kept in AL first.

Code Trace: A Mini Menu Program

```
 1 org 100h
 2 show:
 3 mov dx, offset menu
 4 mov ah, 09h
 5 int 21h
 6 mov ah, 01h       ; read choice
 7 int 21h
 8 cmp al, 'q'
 9 je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

DX = address of the menu text.

Step 1 / 22

VARIABLES

AH = **00h** AL = **00h** DX = **0121h** ZF = **0**

OUTPUT

Code Trace: A Mini Menu Program

```
1  org 100h
2  show:
3  mov dx, offset menu
▶  mov ah, 09h
5  int 21h
6  mov ah, 01h       ; read choice
7  int 21h
8  cmp al, 'q'
9  je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

AH = 09h.

Step 2 / 22

VARIABLES

AH = **09h** AL = **00h** DX = **0121h** ZF = **0**

OUTPUT

Code Trace: A Mini Menu Program

```
1  org 100h
2  show:
3  mov dx, offset menu
4  mov ah, 09h
▶  int 21h
6  mov ah, 01h       ; read choice
7  int 21h
8  cmp al, 'q'
9  je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

The menu is printed.

Step 3 / 22

VARIABLES

AH = **09h** AL = **24h** DX = **0121h** ZF = **0**

OUTPUT

```
1:Hello q:Leave
```

Code Trace: A Mini Menu Program

```
1  org 100h
2  show:
3  mov dx, offset menu
4  mov ah, 09h
5  int 21h
▶  mov ah, 01h       ; read choice
7  int 21h
8  cmp al, 'q'
9  je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

AH = 01h: read one character with echo.

Step 4 / 22

VARIABLES

AH = **01h** | AL = **24h** | DX = **0121h** | ZF = **0**

OUTPUT

```
1:Hello q:Leave
```

Code Trace: A Mini Menu Program

```
1  org 100h
2  show:
3  mov dx, offset menu
4  mov ah, 09h
5  int 21h
6  mov ah, 01h       ; read choice
▶  int 21h
8  cmp al, 'q'
9  je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

The user pressed **1**: AL = 31h (shown on screen).

Step 5 / 22

VARIABLES

AH = **01h** | AL = **31h** | DX = **0121h** | ZF = **0**

OUTPUT

```
1:Hello q:Leave 1
```

Code Trace: A Mini Menu Program

```
 1 org 100h
 2 show:
 3 mov dx, offset menu
 4 mov ah, 09h
 5 int 21h
 6 mov ah, 01h       ; read choice
 7 int 21h
 ▶ cmp al, 'q'
 9 je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

31h ≠ 'q' → ZF = 0.

Step 6 / 22

VARIABLES

AH = **01h** AL = **31h** DX = **0121h** ZF = **0**

OUTPUT

```
1:Hello q:Leave 1
```

Code Trace: A Mini Menu Program

```
1  org 100h
2  show:
3  mov dx, offset menu
4  mov ah, 09h
5  int 21h
6  mov ah, 01h       ; read choice
7  int 21h
8  cmp al, 'q'
▶  je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

Not 'q'.

Step 7 / 22

VARIABLES

AH = **01h** AL = **31h** DX = **0121h** ZF = **0**

OUTPUT

```
1:Hello q:Leave 1
```

Code Trace: A Mini Menu Program

```
 1 org 100h
 2 show:
 3 mov dx, offset menu
 4 mov ah, 09h
 5 int 21h
 6 mov ah, 01h       ; read choice
 7 int 21h
 8 cmp al, 'q'
 9 je quit
▶ cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

AL = '1' → ZF = 1.

Step 8 / 22

VARIABLES

AH = **01h** AL = **31h** DX = **0121h** ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
```

Code Trace: A Mini Menu Program

```
 1 org 100h
 2 show:
 3 mov dx, offset menu
 4 mov ah, 09h
 5 int 21h
 6 mov ah, 01h       ; read choice
 7 int 21h
 8 cmp al, 'q'
 9 je quit
10 cmp al, '1'
▶ jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

Equal → no jump (any other key would show the menu again).

Step 9 / 22

VARIABLES

AH = **01h** AL = **31h** DX = **0121h** ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
```

Code Trace: A Mini Menu Program

```
1  org 100h
2  show:
3  mov dx, offset menu
4  mov ah, 09h
5  int 21h
6  mov ah, 01h       ; read choice
7  int 21h
8  cmp al, 'q'
9  je quit
10 cmp al, '1'
11 jne show
▶ mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

DX = greet.

Step 10 / 22

VARIABLES

AH = **01h** AL = **31h** DX = **0134h** ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
```

Code Trace: A Mini Menu Program

```
 1 org 100h
 2 show:
 3 mov dx, offset menu
 4 mov ah, 09h
 5 int 21h
 6 mov ah, 01h        ; read choice
 7 int 21h
 8 cmp al, 'q'
 9 je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
▶ mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h      ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

AH = 09h.

Step 11 / 22

VARIABLES

AH = **09h** | AL = **31h** | DX = **0134h** | ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
```

Code Trace: A Mini Menu Program

```
1  org 100h
2  show:
3  mov dx, offset menu
4  mov ah, 09h
5  int 21h
6  mov ah, 01h       ; read choice
7  int 21h
8  cmp al, 'q'
9  je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
▶ 14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

Hello! is printed on a new line.

Step 12 / 22

VARIABLES

AH = 09h | AL = 24h | DX = 0134h | ZF = 1

OUTPUT

```
1:Hello q:Leave 1
Hello!
```

Code Trace: A Mini Menu Program

```
 1 org 100h
 2 show:
 3 mov dx, offset menu
 4 mov ah, 09h
 5 int 21h
 6 mov ah, 01h       ; read choice
 7 int 21h
 8 cmp al, 'q'
 9 je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
▶ jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

Back to the menu.

Step 13 / 22

VARIABLES

AH = **09h** AL = **24h** DX = **0134h** ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
Hello!
```

Code Trace: A Mini Menu Program

```
 1 org 100h
 2 show:
 3 mov dx, offset menu
 4 mov ah, 09h
 5 int 21h
 6 mov ah, 01h       ; read choice
 7 int 21h
 8 cmp al, 'q'
 9 je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

DX = address of the menu text.

Step 14 / 22

VARIABLES

AH = **09h** | AL = **24h** | DX = **0121h** | ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
Hello!
```

Code Trace: A Mini Menu Program

```
1  org 100h
2  show:
3  mov dx, offset menu
▶  mov ah, 09h
5  int 21h
6  mov ah, 01h       ; read choice
7  int 21h
8  cmp al, 'q'
9  je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

AH = 09h.

Step 15 / 22

VARIABLES

AH = **09h** AL = **24h** DX = **0121h** ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
Hello!
```

Code Trace: A Mini Menu Program

```
 1 org 100h
 2 show:
 3 mov dx, offset menu
 4 mov ah, 09h
 ▶ int 21h
 6 mov ah, 01h       ; read choice
 7 int 21h
 8 cmp al, 'q'
 9 je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

The menu is printed again.

Step 16 / 22

VARIABLES

AH = **09h** AL = **24h** DX = **0121h** ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
Hello!
1:Hello q:Leave
```

Code Trace: A Mini Menu Program

```
1  org 100h
2  show:
3  mov dx, offset menu
4  mov ah, 09h
5  int 21h
▶  mov ah, 01h       ; read choice
7  int 21h
8  cmp al, 'q'
9  je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

AH = 01h: read one character with echo.

Step 17 / 22

VARIABLES

AH = **01h** | AL = **24h** | DX = **0121h** | ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
Hello!
1:Hello q:Leave
```

Code Trace: A Mini Menu Program

```
1  org 100h
2  show:
3  mov dx, offset menu
4  mov ah, 09h
5  int 21h
6  mov ah, 01h       ; read choice
▶  int 21h
8  cmp al, 'q'
9  je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

The user pressed **q**: AL = 71h.

Step 18 / 22

VARIABLES

AH = **01h** | AL = **71h** | DX = **0121h** | ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
Hello!
1:Hello q:Leave q
```

Code Trace: A Mini Menu Program

```
 1 org 100h
 2 show:
 3 mov dx, offset menu
 4 mov ah, 09h
 5 int 21h
 6 mov ah, 01h       ; read choice
 7 int 21h
 ▶ cmp al, 'q'
 9 je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

AL = 'q' → ZF = 1.

Step 19 / 22

VARIABLES

AH = **01h** | AL = **71h** | DX = **0121h** | ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
Hello!
1:Hello q:Leave q
```

Code Trace: A Mini Menu Program

```
 1 org 100h
 2 show:
 3 mov dx, offset menu
 4 mov ah, 09h
 5 int 21h
 6 mov ah, 01h       ; read choice
 7 int 21h
 8 cmp al, 'q'
▶ je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

→ `quit` .

Step 20 / 22

VARIABLES

AH = **01h** AL = **71h** DX = **0121h** ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
Hello!
1:Hello q:Leave q
```

Code Trace: A Mini Menu Program

```
1  org 100h
2  show:
3  mov dx, offset menu
4  mov ah, 09h
5  int 21h
6  mov ah, 01h       ; read choice
7  int 21h
8  cmp al, 'q'
9  je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
▶ mov ax, 4C00h     ; exit code 0
18 int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

AH = 4Ch (terminate), AL = 00h (exit code).

Step 21 / 22

VARIABLES

AH = **4Ch** | AL = **00h** | DX = **0121h** | ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
Hello!
1:Hello q:Leave q
```

Code Trace: A Mini Menu Program

```
 1 org 100h
 2 show:
 3 mov dx, offset menu
 4 mov ah, 09h
 5 int 21h
 6 mov ah, 01h       ; read choice
 7 int 21h
 8 cmp al, 'q'
 9 je quit
10 cmp al, '1'
11 jne show
12 mov dx, offset greet
13 mov ah, 09h
14 int 21h
15 jmp show
16 quit:
17 mov ax, 4C00h     ; exit code 0
▶ int 21h
19 menu  db 0Dh, 0Ah, '1:Hello q:Leave $'
20 greet db 0Dh, 0Ah, 'Hello!$'
```

The program returns to DOS; batch files read the exit code through ERRORLEVEL.

Step 22 / 22

VARIABLES

AH = **4Ch** AL = **00h** DX = **0121h** ZF = **1**

OUTPUT

```
1:Hello q:Leave 1
Hello!
1:Hello q:Leave q
```

APPLICATION

Common Mistakes

Forgotten '$'
INT 21h/09h prints memory until it finds a '$': the screen fills with garbage.

DS not set (EXE)
DX is right but the segment is wrong: `mov ax, @data` / `mov ds, ax`.

AL gets clobbered
02h and 09h change AL; save a character you read in another register before printing.

Cursor after 09h
INT 10h/09h does not move the cursor; the next output lands on top.

DS:DX with 13h
The BIOS string service expects the address in **ES:BP**.

The 0Ah buffer
Set the first byte (capacity); the text starts at buffer+2 and does not end with '$'.

LF alone
A new line needs CR + LF (0Dh, 0Ah).

No exit
Forget RET/4Ch and the data is executed as instructions.

Lab: BIOS and DOS Services

INT 10h: cursor, 09h and 0Eh

CODE · **36 bytes**

```
 1 org 100h
 2 mov ax, 0003h     ; AH=00h,…
 3 int 10h           ; clears …
 4 mov ah, 02h       ; set cur…
 5 mov bh, 0         ; page 0
 6 mov dx, 0104h     ; DH=1, D…
 7 int 10h
 8 mov ah, 09h       ; char + …
 9 mov al, '*'
10 mov bl, 4Eh       ; yellow …
11 mov cx, 3         ; 3 times
12 int 10h           ; cursor …
13 mov ah, 0Eh       ; teletype
14 mov al, 'A'
```

REGISTERS

AX	0000	AH 00	AL 00		
BX	0000	BH 00	BL 00		
CX	0024	CH 00	CL 24		
DX	0000	DH 00	DL 00		
SI	0000	DI 0000	BP 0000		
SP	FFFE	CS 0700	IP 0100		
DS	0700	ES 0700	SS 0700		

FLAGS

O	D	I	T	S	Z	A	P	C
0	0	1	0	0	0	0	0	0

CS:IP = 0700:0100 · step 0

MEMORY **DS = 0700h**

```
0100 B8 03 00 CD 10 B4 02 B7
0108 00 BA 04 01 CD 10 B4 09
0110 B0 2A B3 4E B9 03 00 CD
```

SCREEN

Program loaded: 36 bytes, CS:IP = 0700:0100h. As in emu8086, CX initially holds the program size (0024h). The Step button executes instructions one at a time; the highlighted line is the next instruction.

Summary: Quick Reference

Task	Call	Parameters
Clear the screen	`INT 10h / 00h`	AL = 03h
Position the cursor	`INT 10h / 02h`	BH = 0, DH = row, DL = column
Colored character	`INT 10h / 09h`	AL, BH, BL = attribute, CX = count
Character (teletype)	`INT 10h / 0Eh`	AL
Colored string	`INT 10h / 13h`	AL = mode, BL, CX, DH:DL, ES:BP
Wait for / poll a key	`INT 16h / 00h, 01h`	→ AH = scan, AL = ASCII; 01h: ZF
Read / write a character	`INT 21h / 01h, 02h`	→ AL / DL
Write a string / read a line	`INT 21h / 09h, 0Ah`	DS:DX ('$' / buffer)
Date / time	`INT 21h / 2Ah, 2Ch`	→ CX DH DL AL / CH CL DH DL
RTC time (BCD)	`INT 1Ah / 02h`	→ CH CL DH, CF = 1 error
Pixel (mode 13h)	`INT 10h / 0Ch`	AL = color, CX = x, DX = y; or A000:(y·320 + x)
End the program	`INT 21h / 4Ch`, `INT 20h`, `RET`	AL = exit code

SUMMARY

Exercises

1. Using INT 10h/13h, write a program that prints "8086" in white on blue at the center of the screen (12, 38).
2. Read two digits from the user (01h) and print their sum as a single digit (sum ≤ 9).
3. Write a program that prints the scan code of each key in hex until Esc is pressed.
4. Print a line read with 0Ah backwards.
5. Print the date as DD.MM.YYYY. How would you convert the year?

▶ **Hints / answers**

Exercises (continued)

6. How would you print the '$' character with `mov ah, 09h` / `int 21h`?
7. In mode 13h, what is the offset of pixel (160, 100) in the A000h segment?
8. INT 1Ah/02h returned DH = 59h. What is printed? What value does INT 21h/2Ch return for the same second?
9. How many ticks of the 0040:006C counter make about 5 seconds?

▶ **Hints / answers**

References

- Ralf Brown, *Ralf Brown's Interrupt List* (detailed reference for INT 10h, 16h, 21h).
- IBM, *Personal Computer AT Technical Reference*, BIOS section, 1984.
- Peter Abel, *IBM PC Assembly Language and Programming*, 5th ed., Prentice Hall, 2001.
- Barry B. Brey, *The Intel Microprocessors*, 8th ed., Pearson, 2009.
- emu8086 help: *supported interrupt functions*.
- Ş. Ç. Öztürk, O. Ergin, *Intel 8086 ile Mikroişlemci Programlamaya Giriş* (in Turkish), 2009.

The End

Chapter 5.2: Interrupt Functions

Previous: Chapter 5.1: Interrupts · Next: Chapter 6: Arithmetic, Logic and Shift Instructions