Chapter 10.2: Device Control

Microprocessors

IN / OUT · Polling · Traffic lights · INT 15h delays · Software delays · 8255 PPI · Stepper motor · LED display · 7-segment display · Thermometer · Printer · Robot · Test device

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_10_02_Device_Control.html

INTRODUCTION

Learning Outcomes

Port-based I/O

Know the forms and limits of the `IN` and `OUT` instructions.

Bit-level control

Map each bit of a port to a lamp, a magnet or a status condition.

Polling

Write loops that wait for a device's ready bit.

Timing

Compute delays in microseconds for INT 15h / AH = 86h.

emu8086 devices

Program the traffic lights, stepper motor, LED display, thermometer, printer, robot and test device.

The embedded view

Explain how a program is loaded from ROM into RAM and run.

7 Virtual Devices Attached to emu8086

Device	Port(s)	Direction	Data
Traffic lights	4	OUT	word: 12 lamp bits
Stepper motor	7	OUT / IN	bits 0–2 magnets, bit 7 ready
Robot	9, 10, 11	OUT / IN	command, data, status
Test device (simple io test)	110, 112	IN / OUT	byte (110), word (112)
Thermometer and heater	125, 127	IN / OUT	temperature, heater on/off
Printer	130	OUT	character (via INT 21h/05h)
LED display	199	OUT	word: signed number

Devices open from emu8086's *virtual devices* menu; a directive such as `#start=Traffic_Lights.exe#` at the top of the program makes the emulator start the device itself.

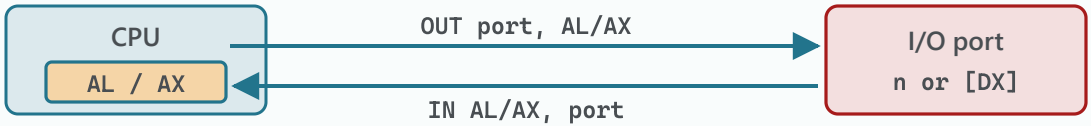
Device Control

- x86 CPUs can control a wide range of devices. The basic I/O **port numbers** vary from system to system and can be changed with suitable hardware (e.g. an address decoder).
- In an embedded system the assembled `.bin` file is burned into a **Read-Only Memory** (ROM) chip.
- At power-up the system reads the program from ROM, loads it into RAM and runs it. Many modern devices (washing machines, printers, modems…) work this way.



The IN and OUT Instructions

Form	Meaning	Port range
`IN AL, port`	read 1 byte from the port	0–255 (immediate)
`IN AX, port`	read 1 word from the port	0–255 (immediate)
`IN AL, DX` / `IN AX, DX`	port number in DX	0–65,535
`OUT port, AL` / `OUT port, AX`	write to the port	0–255 (immediate)
`OUT DX, AL` / `OUT DX, AX`	port number in DX	0–65,535



Data goes **only through AL or AX**: `OUT 4, BX` and `IN CL, 7` are invalid.

For a port above 255, first `MOV DX, 300`, then `OUT DX, AL`. All emu8086 ports in this chapter are below 255.

Port-Mapped and Memory-Mapped I/O

Port-mapped (isolated) I/O

- The 8086 has a separate 64 K I/O address space.
- Accessed only with `IN` / `OUT`; the M/IO̅ pin marks an I/O cycle.
- Uses no memory addresses.
- All of emu8086's virtual devices work this way.

Memory-mapped I/O

- Device registers live at memory addresses.
- Any MOV/ADD/CMP instruction works on them.
- Example: the PC text screen starts at B800:0000; a byte written there shows up on screen.

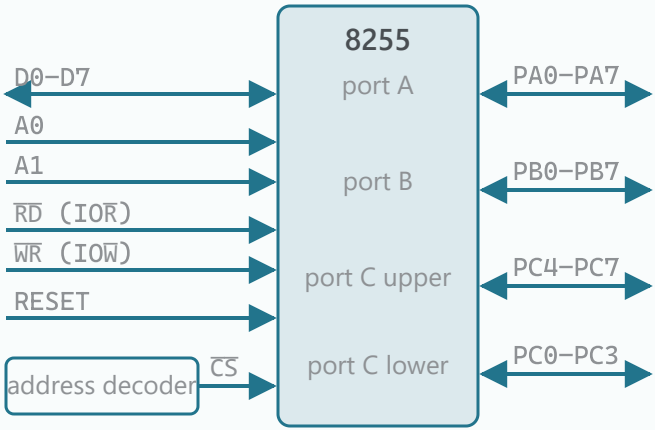


Polling: the CPU reads the device's status bit in a loop until it is “ready”. Simple, but the CPU can do nothing else while waiting. The alternative is for the device to send the CPU an **interrupt**.

The 8255 Programmable Peripheral Interface (PPI)

- Three 8-bit ports (**A**, **B** and **C**, which splits into two 4-bit halves) plus a **control register**; **A1, A0** select the register, $\overline{CS}$ comes from the address decoder.
- After RESET all ports are **inputs**; write the control word first.

A1 A0	Register	Address (base 310h)
0 0	port A	310h
0 1	port B	311h
1 0	port C	312h
1 1	control register	313h



On the 8086 even addresses move over D0–D7 (low bank); to put the registers on even addresses, wire 8255.A0 ← A1, A1 ← A2 (310h, 312h, 314h, 316h). On the 8088, A0 ← A0, A1 ← A1 is enough. Port > 255: access through `DX`.

8255 Control Word (Mode 0)

D7	D6 D5	D4	D3	D2	D1	D0
1 = mode set	group A mode 00 = mode 0	port A	PC7–PC4	group B mode 0 = mode 0	port B	PC3–PC0
			direction bits:	**1 = input**, **0 = output** (D4, D3, D1, D0)		

1 00 0 0 0 1 0b = **82h**

Wanted: A output (LEDs/relays), B input (switches), both halves of C output, all mode 0.

Step 1 / 4

```
mov dx, 313h    ; control register
mov al, 82h     ; A out, B in, C out
out dx, al
mov dx, 311h    ; port B
in  al, dx      ; read switches
mov dx, 310h    ; port A
out dx, al      ; drive the LEDs
mov dx, 313h
mov al, 0Fh     ; BSR: PC7 = 1
out dx, al
```

8255 Control Word (Mode 0)

D7	D6 D5	D4	D3	D2	D1	D0
1 = mode set	group A mode 00 = mode 0	port A	PC7–PC4	group B mode 0 = mode 0	port B	PC3–PC0
			direction bits:	**1 = input**, **0 = output** (D4, D3, D1, D0)		

D7 = 1: mode set · D6 D5 = 00 and D2 = 0: mode 0

Mode 0 is simple I/O without handshaking; modes 1 and 2 are for strobed (handshake) transfers.


```
mov dx, 313h    ; control register
mov al, 82h     ; A out, B in, C out
out dx, al
mov dx, 311h    ; port B
in  al, dx      ; read switches
mov dx, 310h    ; port A
out dx, al      ; drive the LEDs
mov dx, 313h
mov al, 0Fh     ; BSR: PC7 = 1
out dx, al
```

8255 Control Word (Mode 0)

D7	D6 D5	D4	D3	D2	D1	D0
1 = mode set	group A mode 00 = mode 0	port A	PC7–PC4	group B mode 0 = mode 0	port B	PC3–PC0
			direction bits:	**1 = input**, **0 = output** (D4, D3, D1, D0)		

D4 = 0 (A output) · D3 = 0 (C upper output)
D1 = 1 (B input) · D0 = 0 (C lower output)

Result: 1000 0010b = 82h. For A input, B output, C input you would get 1001 1001b = 99h.


```
mov dx, 313h    ; control register
mov al, 82h     ; A out, B in, C out
out dx, al
mov dx, 311h    ; port B
in  al, dx      ; read switches
mov dx, 310h    ; port A
out dx, al      ; drive the LEDs
mov dx, 313h
mov al, 0Fh     ; BSR: PC7 = 1
out dx, al
```

8255 Control Word (Mode 0)

D7	D6 D5	D4	D3	D2	D1	D0
1 = mode set	group A mode 00 = mode 0	port A	PC7–PC4	group B mode 0 = mode 0	port B	PC3–PC0
			direction bits:	**1 = input**, **0 = output** (D4, D3, D1, D0)		

If D7 = 0, it is **bit set/reset** (BSR):
D3–D1 = bit number in C, D0 = new value

0Fh (0000 1111b) → PC7 = 1, 0Eh → PC7 = 0. The other C bits are untouched.


```
mov dx, 313h    ; control register
mov al, 82h     ; A out, B in, C out
out dx, al
mov dx, 311h    ; port B
in  al, dx      ; read switches
mov dx, 310h    ; port A
out dx, al      ; drive the LEDs
mov dx, 313h
mov al, 0Fh     ; BSR: PC7 = 1
out dx, al
```

8255: Control Word, Read, Write, BSR

```
1  org 100h
▶  mov dx, 313h    ; control register
▶  mov al, 82h     ; A,C out; B in
4  out dx, al
5  mov dx, 311h    ; port B
6  in  al, dx      ; read switches
7  mov dx, 310h    ; port A
8  out dx, al      ; drive the LEDs
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```

DX: 0313h

AL: 82h

I/O log:

8255 — control = none (all inputs)

A 310h — input — (not written)

B 311h — input ← switches: 1 0 0 1 0 1 1 0

C 312h — input — (not written)

After RESET all three ports are inputs. The switches on port B are set to 1001 0110b = **96h** (port map: 311h = 96h). DX = 313h, AL = 82h.

Step 1 / 8

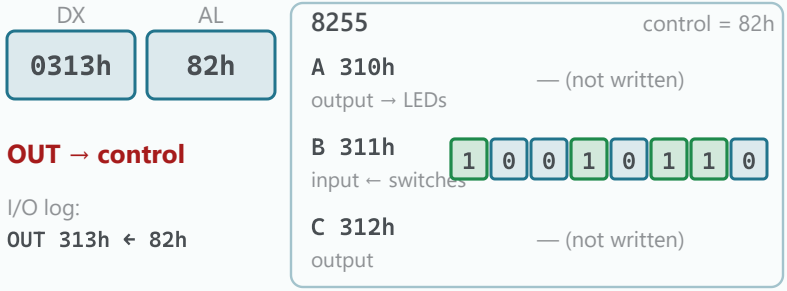
VARIABLES

DX = 0313h AL = 82h

Port values come from the interpreter's I/O log (switch input 96h is an example). A BSR word changes only the selected C bit.

8255: Control Word, Read, Write, BSR

```
1  org 100h
2  mov dx, 313h    ; control register
3  mov al, 82h     ; A,C out; B in
▶  out dx, al
5  mov dx, 311h    ; port B
6  in  al, dx      ; read switches
7  mov dx, 310h    ; port A
8  out dx, al      ; drive the LEDs
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```



OUT 313h ← **82h** = 1 00 0 0 0 1 0b: mode 0; A output, C upper output, B input, C lower output.

Step 2 / 8

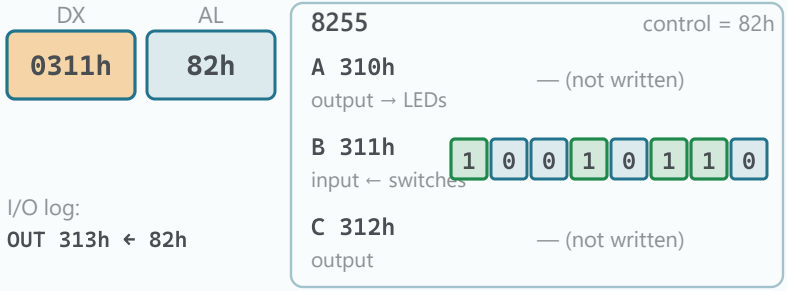
VARIABLES

DX = **0313h** AL = **82h**

Port values come from the interpreter's I/O log (switch input 96h is an example). A BSR word changes only the selected C bit.

8255: Control Word, Read, Write, BSR

```
1  org 100h
2  mov dx, 313h    ; control register
3  mov al, 82h     ; A,C out; B in
4  out dx, al
▶  mov dx, 311h    ; port B
6  in  al, dx      ; read switches
7  mov dx, 310h    ; port A
8  out dx, al      ; drive the LEDs
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```



DX = 311h: port B. Since 311h > 255, the port number goes in DX.

Step 3 / 8

VARIABLES

DX = 0311h AL = 82h

Port values come from the interpreter's I/O log (switch input 96h is an example). A BSR word changes only the selected C bit.

8255: Control Word, Read, Write, BSR

```
1  org 100h
2  mov dx, 313h    ; control register
3  mov al, 82h     ; A,C out; B in
4  out dx, al
5  mov dx, 311h    ; port B
▶  in  al, dx      ; read switches
7  mov dx, 310h    ; port A
8  out dx, al      ; drive the LEDs
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```

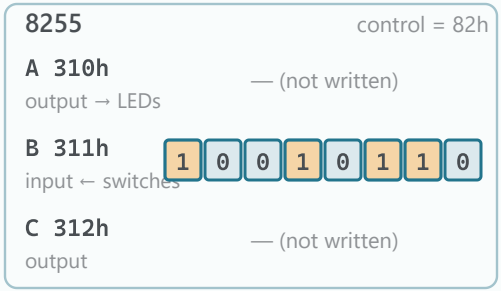
DX: 0311h

AL: 96h

IN ← port B

I/O log:

```
OUT 313h ← 82h
IN 311h → 96h
```



IN AL, DX: the switches are read → AL = **96h**.

Step 4 / 8

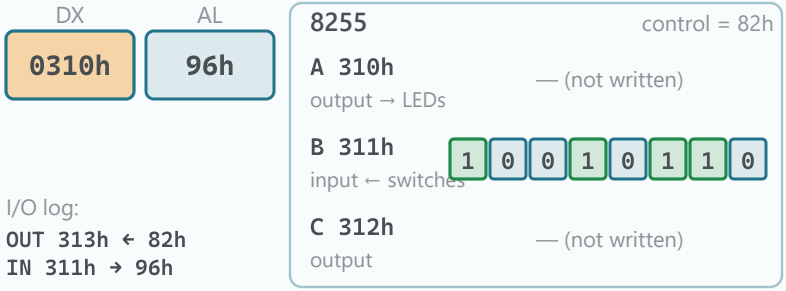
VARIABLES

DX = 0311h AL = 96h

Port values come from the interpreter's I/O log (switch input 96h is an example). A BSR word changes only the selected C bit.

8255: Control Word, Read, Write, BSR

```
1  org 100h
2  mov dx, 313h    ; control register
3  mov al, 82h     ; A,C out; B in
4  out dx, al
5  mov dx, 311h    ; port B
6  in  al, dx      ; read switches
▶  mov dx, 310h    ; port A
8  out dx, al      ; drive the LEDs
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```



DX = 310h: port A.

Step 5 / 8

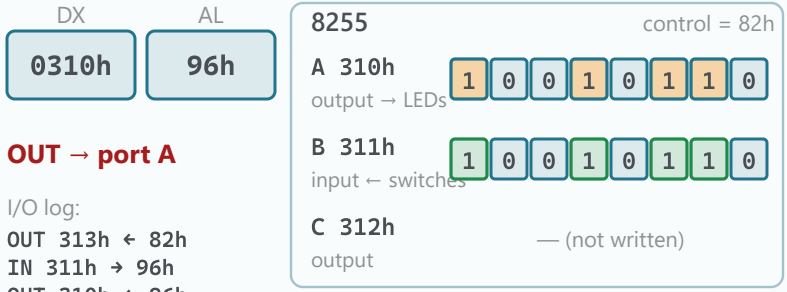
VARIABLES

DX = 0310h AL = 96h

Port values come from the interpreter's I/O log (switch input 96h is an example). A BSR word changes only the selected C bit.

8255: Control Word, Read, Write, BSR

```
1  org 100h
2  mov dx, 313h    ; control register
3  mov al, 82h     ; A,C out; B in
4  out dx, al
5  mov dx, 311h    ; port B
6  in  al, dx      ; read switches
7  mov dx, 310h    ; port A
▶  out dx, al      ; drive the LEDs
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```



OUT 310h ← 96h: port A is an output, so the LEDs mirror the switches (lit: bits 7, 4, 2, 1).

Step 6 / 8

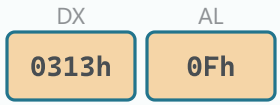
VARIABLES

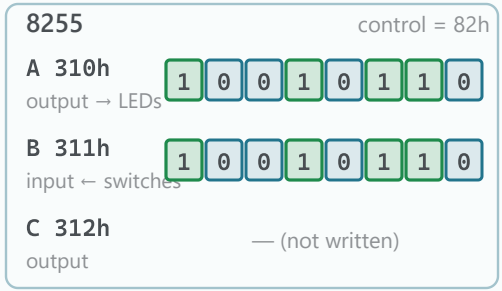
DX = **0310h** AL = **96h**

Port values come from the interpreter's I/O log (switch input 96h is an example). A BSR word changes only the selected C bit.

8255: Control Word, Read, Write, BSR

```
1  org 100h
2  mov dx, 313h    ; control register
3  mov al, 82h     ; A,C out; B in
4  out dx, al
5  mov dx, 311h    ; port B
6  in  al, dx      ; read switches
7  mov dx, 310h    ; port A
8  out dx, al      ; drive the LEDs
▶  mov dx, 313h
▶  mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```





I/O log:

```
OUT 313h ← 82h
IN 311h → 96h
OUT 310h ← 96h
```

DX = 313h, AL = **0Fh** = 0000 111 1b: D7 = 0 → not a mode word but **bit set/reset** (BSR).

Step 7 / 8

VARIABLES

`DX = 0313h` `AL = 0Fh`

Port values come from the interpreter's I/O log (switch input 96h is an example). A BSR word changes only the selected C bit.

8255: Control Word, Read, Write, BSR

```
 1  org 100h
 2  mov dx, 313h    ; control register
 3  mov al, 82h     ; A,C out; B in
 4  out dx, al
 5  mov dx, 311h    ; port B
 6  in  al, dx      ; read switches
 7  mov dx, 310h    ; port A
 8  out dx, al      ; drive the LEDs
 9  mov dx, 313h
10  mov al, 0Fh     ; BSR: PC7 = 1
▶   out dx, al
12  ret
```

DX: 0313h

AL: 0Fh

OUT → control (BSR)

I/O log:

```
IN 311h → 96h
OUT 310h ← 96h
OUT 313h ← 0Fh
```

8255 — control = 82h

A 310h (output → LEDs)	1	0	0	1	0	1	1	0
B 311h (input ← switches)	1	0	0	1	0	1	1	0
C 312h (output)	1	0	0	0	0	0	0	0

OUT 313h ← 0Fh: D3–D1 = 111 → PC7, D0 = 1 → **PC7 = 1**. The port directions (82h) stay, the other C bits are kept.

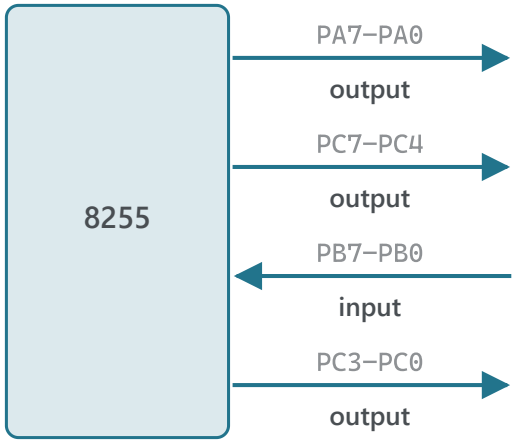
Step 8 / 8

VARIABLES

DX = 0313h AL = 0Fh

Port values come from the interpreter's I/O log (switch input 96h is an example). A BSR word changes only the selected C bit.

Interactive 8255 Control Word



Port A (PA7–PA0) — output (0)

Port C upper (PC7–PC4) — output (0)

Port B (PB7–PB0) — input (1)

Port C lower (PC3–PC0) — output (0)

Control word (D7...D0) 82 h

1	0	0	0	0	0	1	0
mode	A mode	A mode	PA	PC hi	B mode	PB	PC lo

```
mov dx, 313h   ; control register
mov al, 82h    ; 10000010b
out dx, al
; Port A (DX = 310h): out dx, al
; Port C (DX = 312h): out dx, al
; Port B (DX = 311h): in  al, dx
```

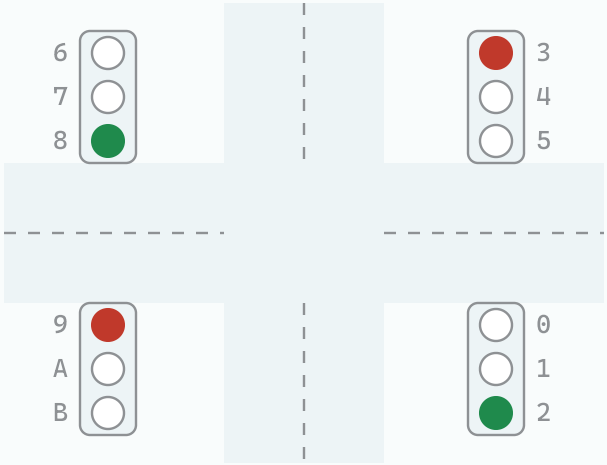
Port A **output** · Port C upper **output** · Port B **input** · Port C lower **output**

Click a port's direction to flip it, or type a value straight into the box (e.g. 9B, 80, 0F).

Traffic Lights (Port 4)

- The junction has 4 poles with 3 lamps each: **12 lamps** = **bits 0–11** of the word written to port 4.
- Bit = 1 → lamp on, 0 → off. Bits 12–15 are unused.
- On each pole (k = 0…3): bit 3k **red**, 3k+1 **amber**, 3k+2 **green**.
- Example: `0000_0011_0000_1100b` → bits 2 and 8 (green), bits 3 and 9 (red).

Pole	Bits
bottom right	0 1 2
top right	3 4 5
top left	6 7 8
bottom left	9 A B



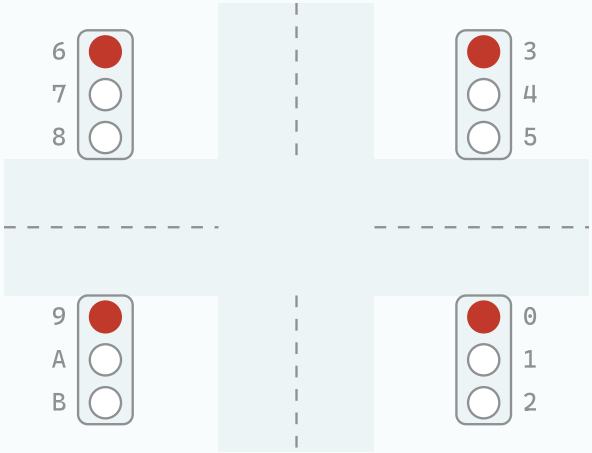
The Situation Table

```
;                   FEDC_BA98_7654_3210
situation       dw  0000_0011_0000_1100b
s1              dw  0000_0110_1001_1010b
s2              dw  0000_1000_0110_0001b
s3              dw  0000_1000_0110_0001b
s4              dw  0000_0100_1101_0011b
sit_end = $
all_red         equ 0000_0010_0100_1001b
```

Name	Hex	Lamps
`situation`	030Ch	bottom right: green; top right: red; top left: green; bottom left: red
`s1`	069Ah	bottom right: amber; top right: red + amber; top left: amber; bottom left: red + amber
`s2`	0861h	bottom right: red; top right: green; top left: red; bottom left: green
`s3`	0861h	bottom right: red; top right: green; top left: red; bottom left: green
`s4`	04D3h	bottom right: red + amber; top right: amber; top left: red + amber; bottom left: amber
`all_red`	0249h	bottom right: red; top right: red; top left: red; bottom left: red

s2 and s3 are identical, so the side road stays green for two intervals (10 s). s1 and s4 are the amber transitions (red + amber = “get ready”).

The States Step by Step



FEDC BA98 7654 3210

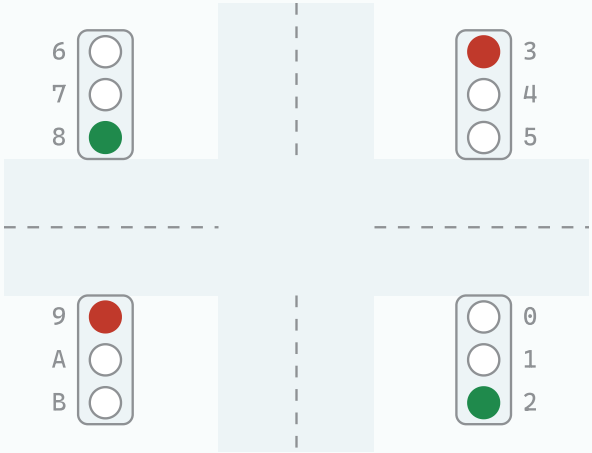
0000 0010 0100 1001

all_red = 0249h → bottom right: red; top right: red; top left: red; bottom left: red

Step 1 / 6

The program first sets every direction to red, then cycles situation → s1 → s2 → s3 → s4 → situation… at 5-second intervals.

The States Step by Step



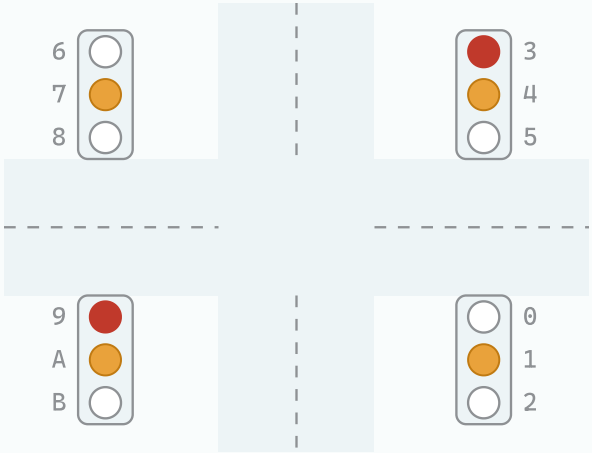
FEDC BA98 7654 3210
0000 0011 0000 1100

situation = 030Ch → bottom right: green; top right: red; top left: green; bottom left: red

Step 2 / 6

The program first sets every direction to red, then cycles situation → s1 → s2 → s3 → s4 → situation… at 5-second intervals.

The States Step by Step

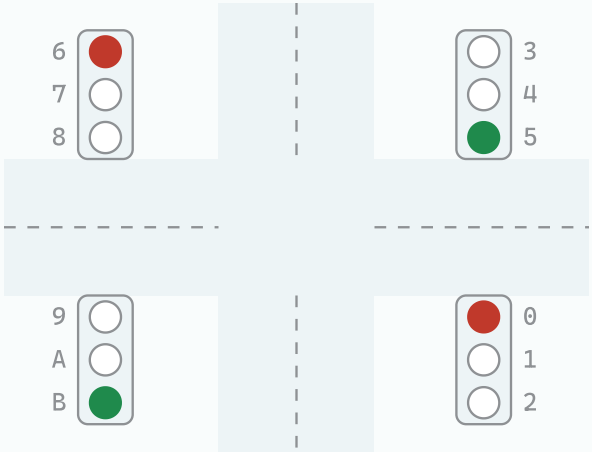


FEDC BA98 7654 3210
0000 0110 1001 1010

s1 = 069Ah → bottom right: amber; top right: red + amber; top left: amber; bottom left: red + amber

The program first sets every direction to red, then cycles situation → s1 → s2 → s3 → s4 → situation… at 5-second intervals.

The States Step by Step



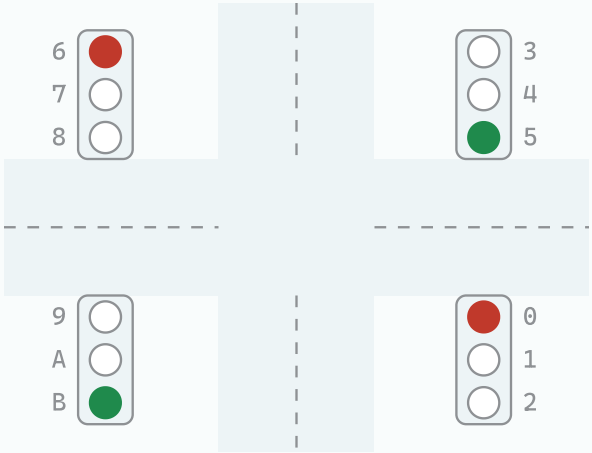
FEDC BA98 7654 3210

0000 1000 0110 0001

s2 = 0861h → bottom right: red; top right: green; top left: red; bottom left: green

The program first sets every direction to red, then cycles situation → s1 → s2 → s3 → s4 → situation… at 5-second intervals.

The States Step by Step



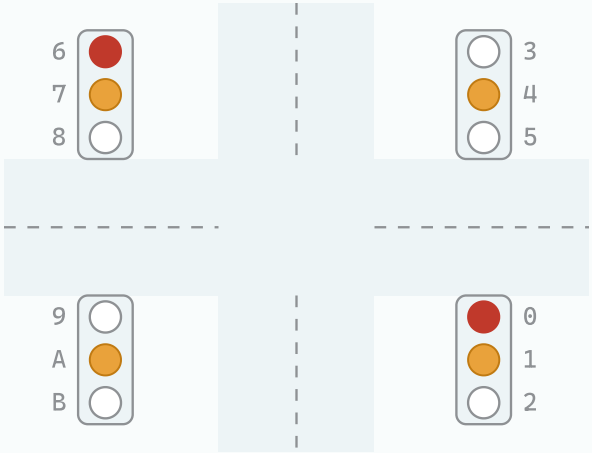
FEDC BA98 7654 3210

0000 1000 0110 0001

s3 = 0861h → bottom right: red; top right: green; top left: red; bottom left: green

The program first sets every direction to red, then cycles situation → s1 → s2 → s3 → s4 → situation… at 5-second intervals.

The States Step by Step



FEDC BA98 7654 3210

0000 0100 1101 0011

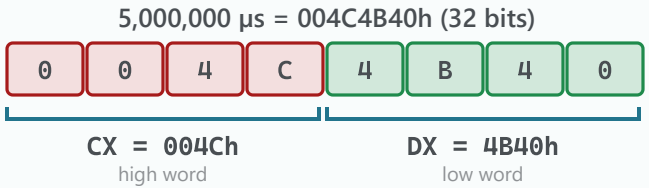
s4 = 04D3h → bottom right: red + amber; top right: amber; top left: red + amber; bottom left: amber

The program first sets every direction to red, then cycles situation → s1 → s2 → s3 → s4 → situation… at 5-second intervals.

Waiting with INT 15h / AH = 86h

```
; wait 5 seconds (5 million µs)
mov cx, 4Ch     ; 004C4B40h = 5 000 000
mov dx, 4B40h
mov ah, 86h
int 15h
```

- BIOS function **INT 15h, AH = 86h**: waits CX:DX microseconds.
- CX is the high word, DX the low word: a 32-bit delay.



5 s = 5 × 1,000,000 = **5,000,000 µs**

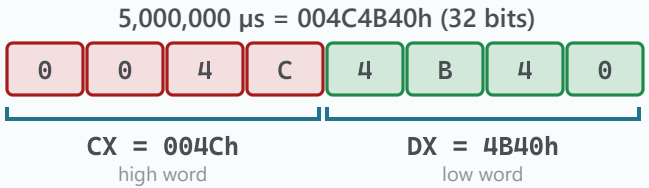
Convert the time to microseconds.

Step 1 / 4

Waiting with INT 15h / AH = 86h

```
; wait 5 seconds (5 million µs)
mov cx, 4Ch     ; 004C4B40h = 5 000 000
mov dx, 4B40h
mov ah, 86h
int 15h
```

- BIOS function **INT 15h, AH = 86h**: waits CX:DX microseconds.
- CX is the high word, DX the low word: a 32-bit delay.



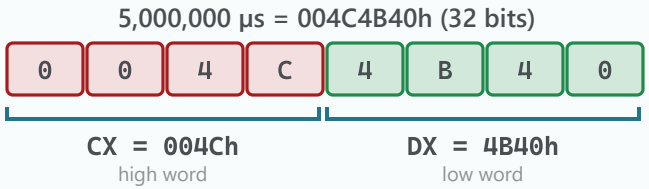
5,000,000 ÷ 16 = 312,500 rem. **0**
312,500 ÷ 16 = 19,531 rem. **4**
19,531 ÷ 16 = 1,220 rem. **11 (B)**
1,220 ÷ 16 = 76 rem. **4**
76 ÷ 16 = 4 rem. **12 (C)**
4 ÷ 16 = 0 rem. **4**

Repeated division by 16: read the remainders from last to first → **4C4B40h**.

Waiting with INT 15h / AH = 86h

```
; wait 5 seconds (5 million µs)
mov cx, 4Ch     ; 004C4B40h = 5 000 000
mov dx, 4B40h
mov ah, 86h
int 15h
```

- BIOS function **INT 15h, AH = 86h**: waits CX:DX microseconds.
- CX is the high word, DX the low word: a 32-bit delay.



`0 0 4 C` | `4 B 4 0`

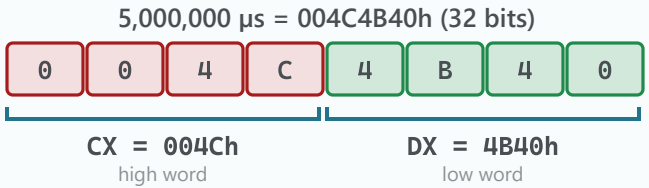
CX = **004Ch** · DX = **4B40h**

The 32-bit value 0000 0000 0100 1100 0100 1011 0100 0000b is split in two.

Waiting with INT 15h / AH = 86h

```
; wait 5 seconds (5 million µs)
mov cx, 4Ch     ; 004C4B40h = 5 000 000
mov dx, 4B40h
mov ah, 86h
int 15h
```

- BIOS function **INT 15h, AH = 86h**: waits CX:DX microseconds.
- CX is the high word, DX the low word: a 32-bit delay.



1 s → 1,000,000 = 000F4240h → CX = 000Fh, DX = 4240h

Another example: a 1-second delay.

Software Delays: LOOP and Clock Cycles

```
; ≈ 1 ms delay (4.77 MHz 8086)
mov cx, 281     ; 4 clocks
delay:
loop delay      ; 280 × 17 + 1 × 5 clocks
```

Instruction	Clocks
`MOV CX, imm`	4
`LOOP` (CX ≠ 0, jumps)	17
`LOOP` (CX = 0, falls through)	5

Total = 4 + 17(N − 1) + 5 = **17N − 8** clocks.

4.77 MHz: 1 clock ≈ 0.2096 µs

1 ms = **4,770** clocks

Time = number of clocks ÷ clock frequency.

These figures are approximate: interrupts, memory wait states and the prefetch queue can stretch the time. emu8086 doesn't model instruction timing; the loop runs at the speed of your PC and the step-delay setting. In emu8086, timed waits use INT 15h/86h.

Software Delays: LOOP and Clock Cycles

```
; ≈ 1 ms delay (4.77 MHz 8086)
mov cx, 281     ; 4 clocks
delay:
loop delay      ; 280 × 17 + 1 × 5 clocks
```

Instruction	Clocks
`MOV CX, imm`	4
`LOOP` (CX ≠ 0, jumps)	17
`LOOP` (CX = 0, falls through)	5

Total = 4 + 17(N − 1) + 5 = **17N − 8** clocks.

17N − 8 = 4,770 → N ≈ 281.06

N = **281** → 4,769 clocks ≈ 0.9998 ms

CX takes an integer, so pick the nearest value.

These figures are approximate: interrupts, memory wait states and the prefetch queue can stretch the time. emu8086 doesn't model instruction timing; the loop runs at the speed of your PC and the step-delay setting. In emu8086, timed waits use INT 15h/86h.

Software Delays: LOOP and Clock Cycles

```
; ≈ 1 ms delay (4.77 MHz 8086)
mov cx, 281     ; 4 clocks
delay:
loop delay      ; 280 × 17 + 1 × 5 clocks
```

Instruction	Clocks
`MOV CX, imm`	4
`LOOP` (CX ≠ 0, jumps)	17
`LOOP` (CX = 0, falls through)	5

Total = 4 + 17(N − 1) + 5 = **17N − 8** clocks.

5 MHz: 1 ms = 5,000 clocks → N ≈ 294.6
N = **295** → 5,007 clocks ≈ 1.0014 ms

The same N = 281 lasts only 0.954 ms at 5 MHz: the delay **depends on the clock speed**.

These figures are approximate: interrupts, memory wait states and the prefetch queue can stretch the time. emu8086 doesn't model instruction timing; the loop runs at the speed of your PC and the step-delay setting. In emu8086, timed waits use INT 15h/86h.

Software Delays: LOOP and Clock Cycles

```
; ≈ 1 ms delay (4.77 MHz 8086)
mov cx, 281     ; 4 clocks
delay:
loop delay      ; 280 × 17 + 1 × 5 clocks
```

Instruction	Clocks
`MOV CX, imm`	4
`LOOP` (CX ≠ 0, jumps)	17
`LOOP` (CX = 0, falls through)	5

Total = 4 + 17(N − 1) + 5 = **17N − 8** clocks.

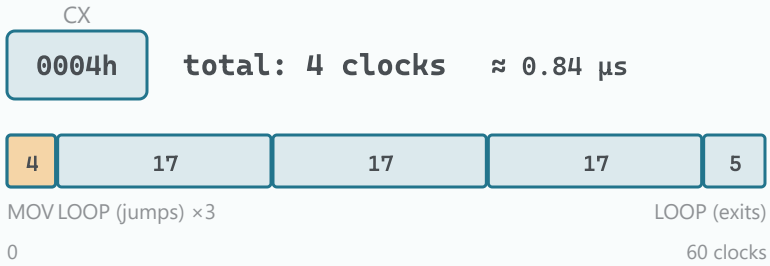
Longest: CX = 0 → 65,536 passes
17 · 65,536 − 8 = 1,114,104 clocks ≈ **233.6 ms**

Longer delays need nested loops. Called as a procedure, it also costs CALL (19) and RET (8) clocks.

These figures are approximate: interrupts, memory wait states and the prefetch queue can stretch the time. emu8086 doesn't model instruction timing; the loop runs at the speed of your PC and the step-delay setting. In emu8086, timed waits use INT 15h/86h.

Counting Clocks in a Delay Loop

```
1 org 100h
▶ mov cx, 4       ; 4 clocks
3 delay:
4 loop delay      ; 17 clocks, last one 5
5 ret
```



CX = 4. `mov cx, imm` takes 4 clocks; at 4.77 MHz, 1 clock ≈ 0.2096 µs.

Step 1 / 6

VARIABLES

`CX = 0004h` `clk = 4`

Clock counts come from 8086 instruction timings; the interpreter only checks CX and the jumps (emu8086 doesn't model timing).

Counting Clocks in a Delay Loop

```
1 org 100h
2 mov cx, 4       ; 4 clocks
3 delay:
▶ loop delay      ; 17 clocks, last
  one 5
5 ret
```



LOOP 1: CX = 3 ≠ 0 → jumps (back to itself): **17** clocks. Total 21.

Step 2 / 6

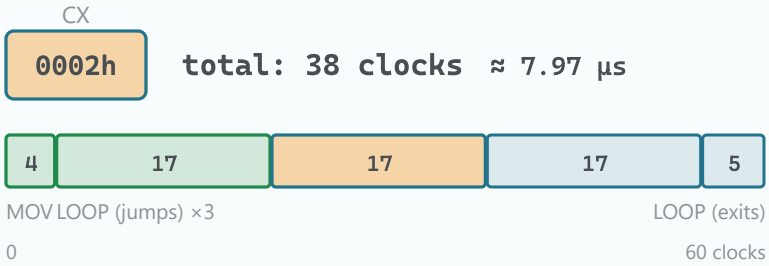
VARIABLES

`CX = 0003h` `clk = 21`

Clock counts come from 8086 instruction timings; the interpreter only checks CX and the jumps (emu8086 doesn't model timing).

Counting Clocks in a Delay Loop

```
1 org 100h
2 mov cx, 4       ; 4 clocks
3 delay:
▶ loop delay      ; 17 clocks, last one 5
5 ret
```



LOOP 2: CX = 2, again 17 clocks. Total 38.

Step 3 / 6

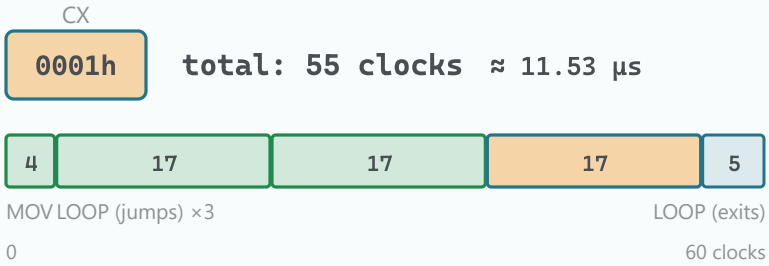
VARIABLES

`CX = 0002h` `clk = 38`

Clock counts come from 8086 instruction timings; the interpreter only checks CX and the jumps (emu8086 doesn't model timing).

Counting Clocks in a Delay Loop

```
1 org 100h
2 mov cx, 4       ; 4 clocks
3 delay:
▶ loop delay      ; 17 clocks, last one 5
5 ret
```



LOOP 3: CX = 1, 17 clocks. Total 55.

Step 4 / 6

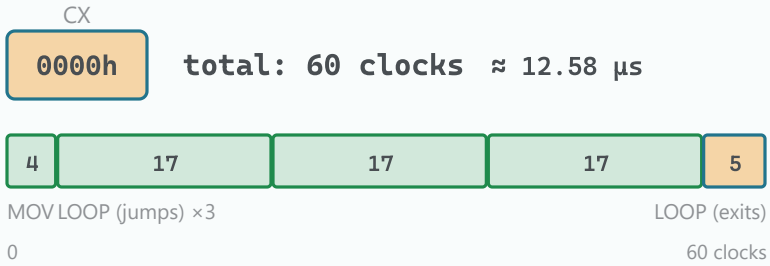
VARIABLES

`CX = 0001h` `clk = 55`

Clock counts come from 8086 instruction timings; the interpreter only checks CX and the jumps (emu8086 doesn't model timing).

Counting Clocks in a Delay Loop

```
1 org 100h
2 mov cx, 4       ; 4 clocks
3 delay:
▶ loop delay      ; 17 clocks, last one 5
5 ret
```



LOOP 4: CX = 0 → falls through: only **5** clocks. Total **60** = 17 · 4 − 8 clocks.

Step 5 / 6

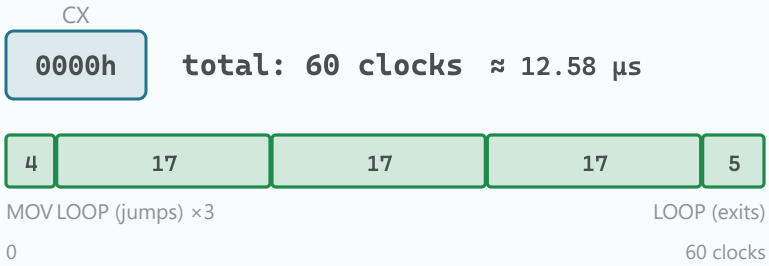
VARIABLES

`CX = 0000h` `clk = 60`

Clock counts come from 8086 instruction timings; the interpreter only checks CX and the jumps (emu8086 doesn't model timing).

Counting Clocks in a Delay Loop

```
1 org 100h
2 mov cx, 4       ; 4 clocks
3 delay:
4 loop delay      ; 17 clocks, last one 5
  ret
```



60 clocks ÷ 4.77 MHz ≈ **12.6 µs**. With CX = N the loop takes 17N − 8 clocks; for 1 ms we found N = 281.

Step 6 / 6

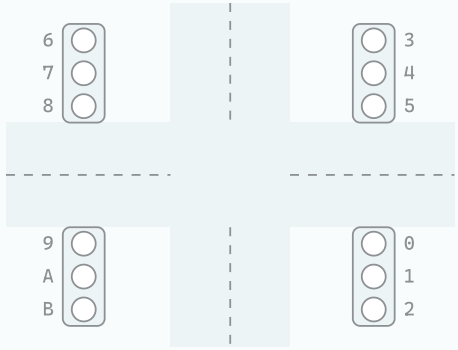
VARIABLES

`CX = 0000h` `clk = 60`

Clock counts come from 8086 instruction timings; the interpreter only checks CX and the jumps (emu8086 doesn't model timing).

The Traffic Light Program

```
#start=Traffic_Lights.exe#
mov ax, all_red
out 4, ax
mov si, offset situation
next:
mov ax, [si]
out 4, ax
mov cx, 4Ch       ; wait 5 s
mov dx, 4B40h
mov ah, 86h
int 15h
add si, 2         ; next state
cmp si, sit_end
jb  next
mov si, offset situation
jmp next
```



The table is assumed at 0124h (ORG 100h; the address depends on the assembler). sit_end = 012Eh.

Step 1 / 22

VARIABLES

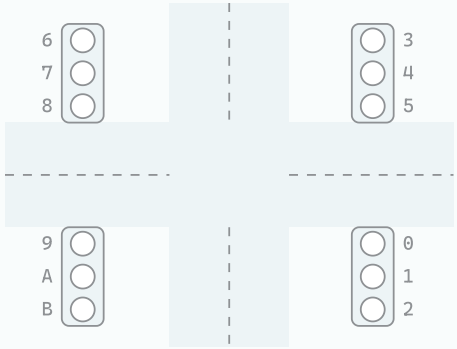
AX = **0000h** SI = **0000h** CX = **0000h**

DX = **0000h** CF = **0**

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



AX = all_red = 0249h (a constant, EQU).

Step 2 / 22

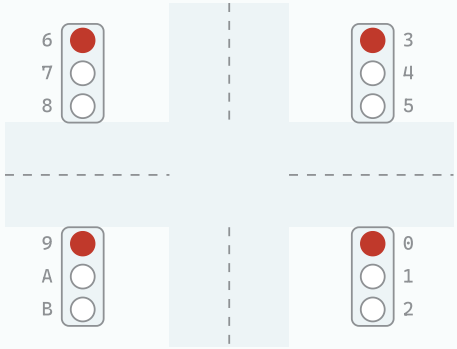
VARIABLES

AX = 0249h | SI = 0000h | CX = 0000h | DX = 0000h | CF = 0

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
▶  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0010 0100 1001

Write port 4: every direction **red**.

Step 3 / 22

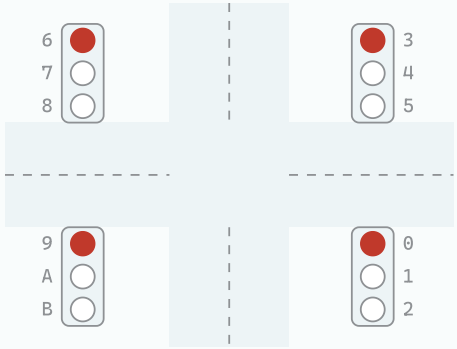
VARIABLES

AX = 0249h SI = 0000h CX = 0000h DX = 0000h CF = 0

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



SI = 0124h: start of the table (situation).

Step 4 / 22

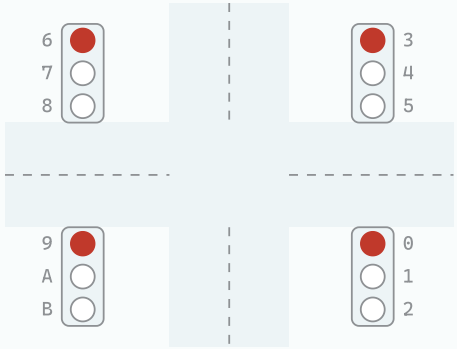
VARIABLES

AX = 0249h | SI = 0124h | CX = 0000h | DX = 0000h | CF = 0

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
▶  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



AX ← [SI] = 030Ch.

Step 5 / 22

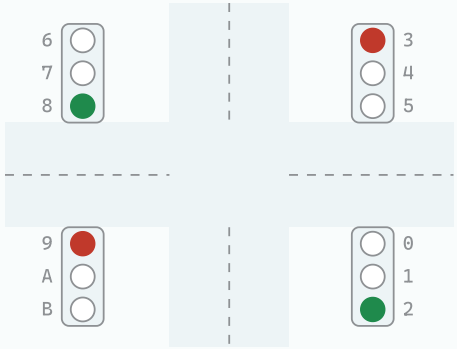
VARIABLES

AX = 030Ch | SI = 0124h | CX = 0000h | DX = 0000h | CF = 0

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
▶  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0011 0000 1100

OUT 4, AX: the lamps switch to the new state (SI = 0124h).

Step 6 / 22

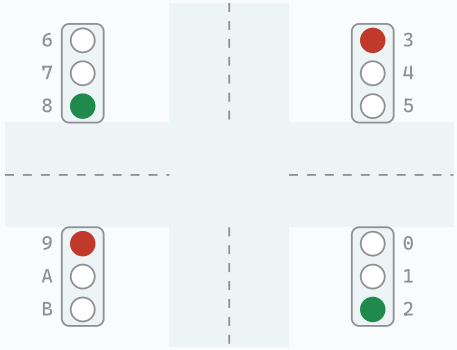
VARIABLES

AX = 030Ch | SI = 0124h | CX = 0000h | DX = 0000h | CF = 0

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
   mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



CX = 004Ch (high word of the delay).

Step 7 / 22

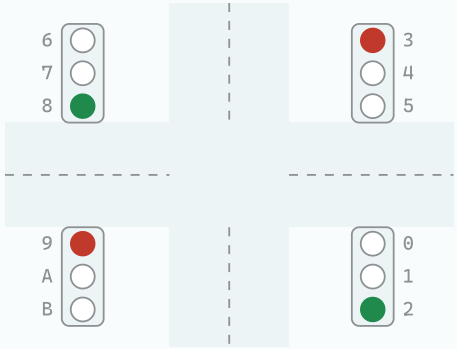
VARIABLES

AX = 030Ch | SI = 0124h | CX = 004Ch | DX = 0000h | CF = 0

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0011 0000 1100

DX = 4B40h (low word).

Step 8 / 22

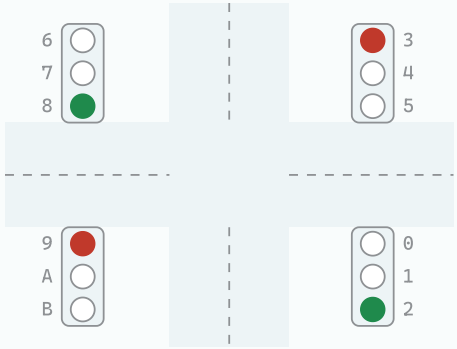
VARIABLES

AX = 030Ch | SI = 0124h | CX = 004Ch | DX = 4B40h | CF = 0

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
▶  mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



AH = 86h: BIOS wait function.

Step 9 / 22

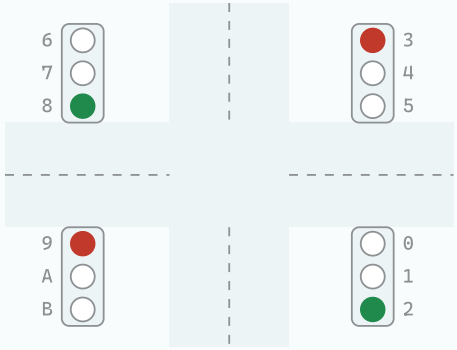
VARIABLES

AX = 860Ch | SI = 0124h | CX = 004Ch | DX = 4B40h | CF = 0

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
▶ int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



INT 15h: wait 5 seconds.

Step 10 / 22

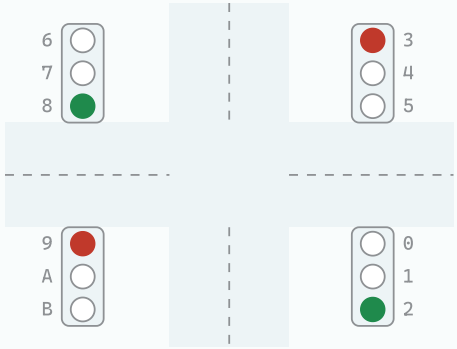
VARIABLES

AX = **860Ch** SI = **0124h** CX = **004Ch** DX = **4B40h** CF = **0**

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
▶ add si, 2          ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



SI = 0126h: next word.

Step 11 / 22

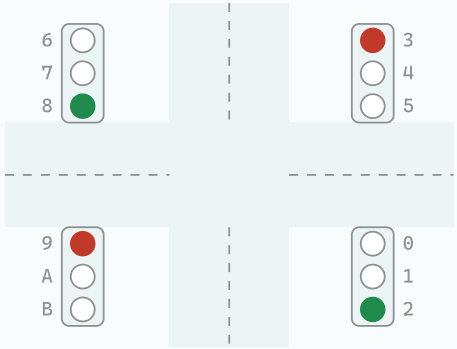
VARIABLES

AX = **860Ch** | SI = **0126h** | CX = **004Ch** | DX = **4B40h** | CF = **0**

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
   cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0011 0000 1100

CMP SI, sit_end (012Eh): SI < sit_end → CF = 1.

Step 12 / 22

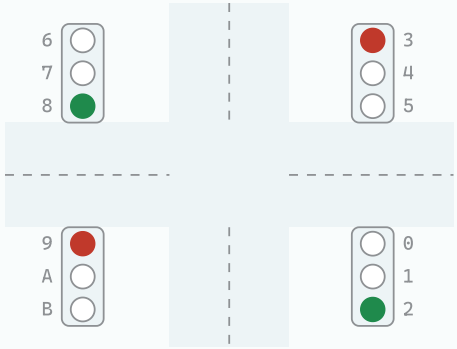
VARIABLES

AX = 860Ch | SI = 0126h | CX = 004Ch | DX = 4B40h | CF = 1

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
▶ jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0011 0000 1100

JB: CF = 1, back to next.

Step 13 / 22

VARIABLES

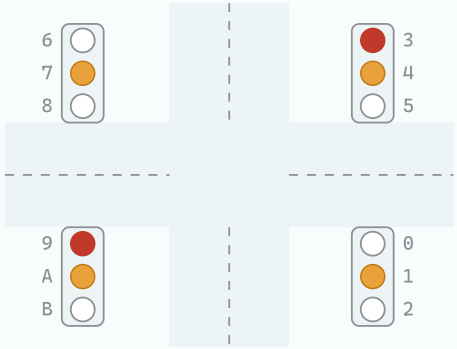
AX = **860Ch** SI = **0126h** CX = **004Ch**

DX = **4B40h** CF = **1**

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
▶  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



... wait and advance SI ... **OUT 4, AX**: SI = 0126h → port 4 = 069Ah.

Step 14 / 22

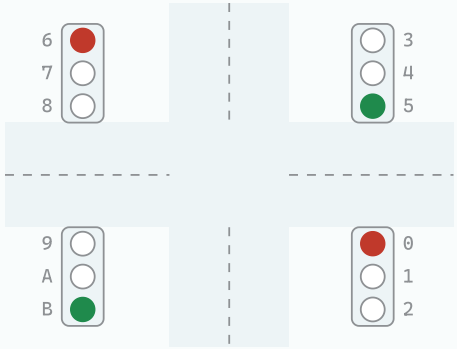
VARIABLES

AX = 069Ah | SI = 0126h | CX = 004Ch | DX = 4B40h | CF = 1

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
▶  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



... wait and advance SI ... **OUT 4, AX**: SI = 0128h → port 4 = 0861h.

Step 15 / 22

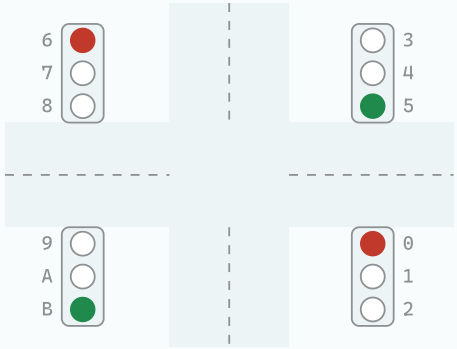
VARIABLES

AX = **0861h** SI = **0128h** CX = **004Ch** DX = **4B40h** CF = **1**

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
▶  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



… wait and advance SI … **OUT 4, AX**: SI = 012Ah → port 4 = 0861h.

Step 16 / 22

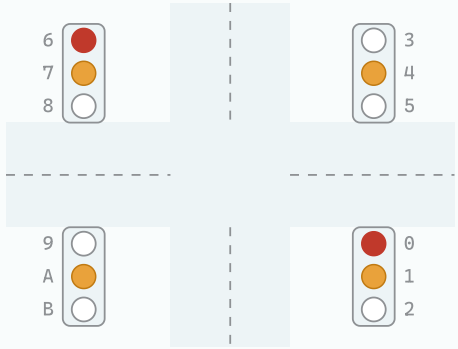
VARIABLES

AX = **0861h** | SI = **012Ah** | CX = **004Ch** | DX = **4B40h** | CF = **1**

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
▶  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



… wait and advance SI … **OUT 4, AX**: SI = 012Ch → port 4 = 04D3h.

Step 17 / 22

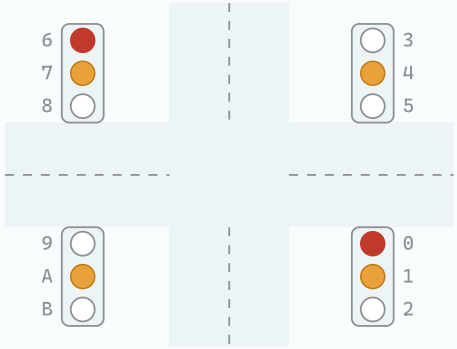
VARIABLES

AX = **04D3h** SI = **012Ch** CX = **004Ch** DX = **4B40h** CF = **1**

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
▶ cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0100 1101 0011

CMP SI, sit_end (012Eh): SI = sit_end → CF = 0.

Step 18 / 22

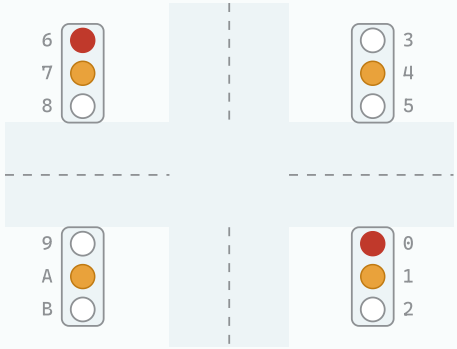
VARIABLES

AX = 86D3h | SI = 012Eh | CX = 004Ch | DX = 4B40h | CF = 0

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
▶ jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0100 1101 0011

JB: CF = 0, no jump: end of table.

Step 19 / 22

VARIABLES

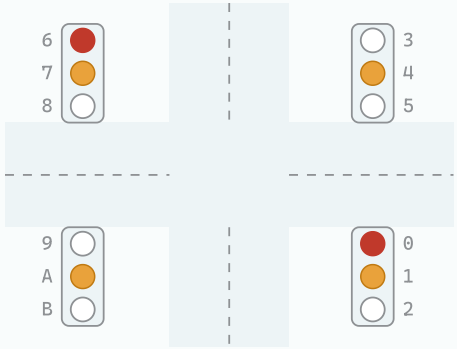
AX = **86D3h** SI = **012Eh** CX = **004Ch**

DX = **4B40h** CF = **0**

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
   mov si, offset situation
16 jmp next
```



SI = 0124h: restart the table.

Step 20 / 22

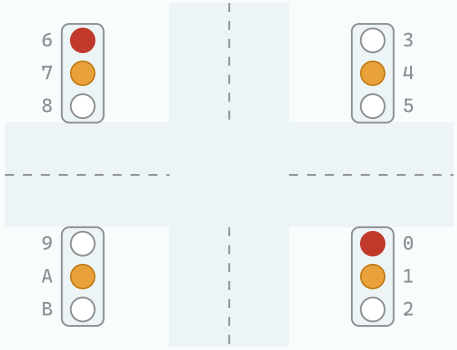
VARIABLES

AX = 86D3h | SI = 0124h | CX = 004Ch | DX = 4B40h | CF = 0

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
   jmp next
```



JMP next: endless loop.

Step 21 / 22

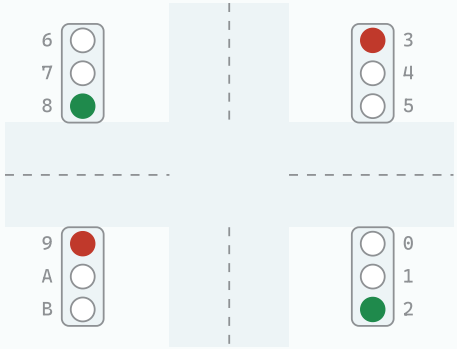
VARIABLES

AX = **86D3h** SI = **0124h** CX = **004Ch** DX = **4B40h** CF = **0**

The loop never ends: in embedded systems the main program is usually an endless loop.

The Traffic Light Program

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
▶  out 4, ax
8  mov cx, 4Ch       ; wait 5 s
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; next state
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



... wait and advance SI ... **OUT 4, AX**: SI = 0124h → port 4 = 030Ch.

Step 22 / 22

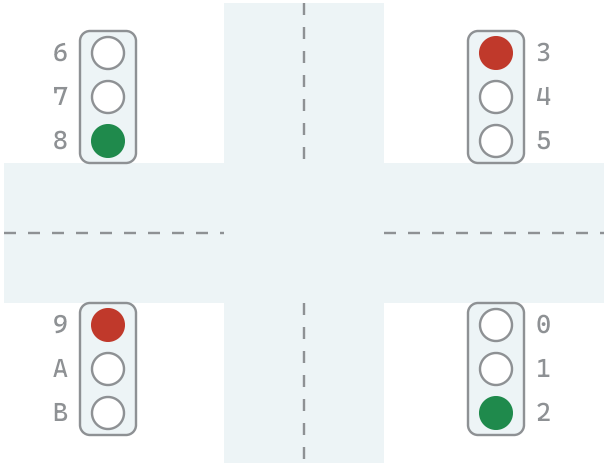
VARIABLES

AX = 030Ch | SI = 0124h | CX = 004Ch | DX = 4B40h | CF = 0

The loop never ends: in embedded systems the main program is usually an endless loop.

TRAFFIC LIGHTS

Interactive Traffic Bit Editor



Click a bit to turn its lamp on or off (F…0). Bits 12–15 are unused.

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0

all_red **situation** **s1** **s2 / s3** **s4**

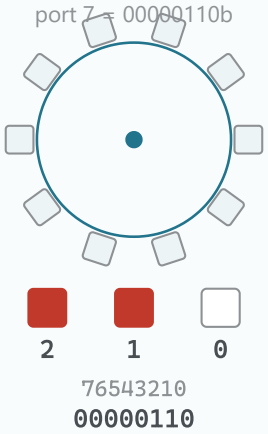
0000h

```
mov ax, 0000_0011_0000_1100b   ; 030Ch
out 4, ax
```

bottom right: **green** · top right: **red** · top left: **green** · bottom left: **red**

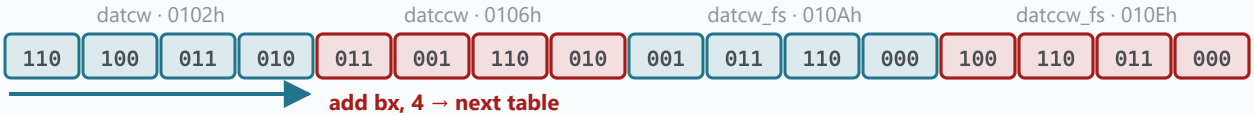
Stepper Motor (Port 7)

- The motor can turn **clockwise** (CW) or **counterclockwise** (CCW).
- **Bits 0, 1, 2** of the byte written to port 7 switch the three electromagnets.
- A **half step** is 11.25°, a **full step** 22.5°.
- Half step: two magnets on, then one magnet on, alternately.
- Full step: one, two, two magnets and an “all off” byte, in turn.
- When the motor is ready for a new command, it sets **bit 7** of the byte read from the port.



Stepper Motor Data Tables

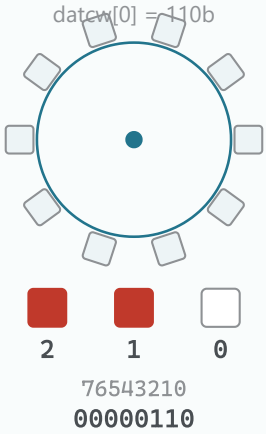
Table	Meaning	1	2	3	4
`datcw`	CW half step	110	100	011	010
`datccw`	CCW half step	011	001	110	010
`datcw_fs`	CW full step	001	011	110	000
`datccw_fs`	CCW full step	100	110	011	000



- The tables sit back to back in memory, **4 bytes** each, so `add bx, 4` moves to the next table.
- After 32 (20h) rounds the program switches tables: CW half → CCW half → CW full → CCW full → start over.
- The bits of each byte are the magnets: `110b` = magnets 2 and 1 on; `000b` = all off.
- In the original source: `datcw db 0000_0110b`, then `db 0000_0100b` … on the following lines (the underscores are just for readability).

Magnet Sequences Step by Step

- Half-step sequence (datcw): **two** → **one** → **two** → **one** magnets.
- Full-step sequence (datcw_fs): one, two, two magnets and an “all off” byte.
- After each OUT the motor is **busy** until it reaches its new position, so the program polls bit 7 before every byte.

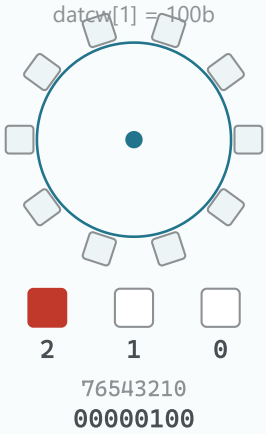


CW half step, byte 1: magnets on → 2, 1

Step 1 / 8

Magnet Sequences Step by Step

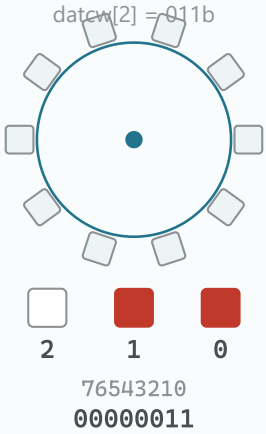
- Half-step sequence (datcw): **two** → **one** → **two** → **one** magnets.
- Full-step sequence (datcw_fs): one, two, two magnets and an “all off” byte.
- After each OUT the motor is **busy** until it reaches its new position, so the program polls bit 7 before every byte.



CW half step, byte 2: magnets on → 2

Magnet Sequences Step by Step

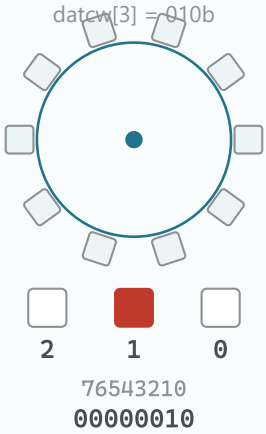
- Half-step sequence (datcw): **two** → **one** → **two** → **one** magnets.
- Full-step sequence (datcw_fs): one, two, two magnets and an “all off” byte.
- After each OUT the motor is **busy** until it reaches its new position, so the program polls bit 7 before every byte.



CW half step, byte 3: magnets on → 1, 0

Magnet Sequences Step by Step

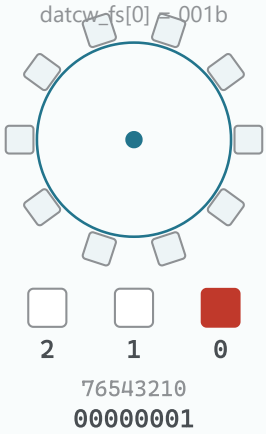
- Half-step sequence (datcw): **two** → **one** → **two** → **one** magnets.
- Full-step sequence (datcw_fs): one, two, two magnets and an “all off” byte.
- After each OUT the motor is **busy** until it reaches its new position, so the program polls bit 7 before every byte.



CW half step, byte 4: magnets on → 1

Magnet Sequences Step by Step

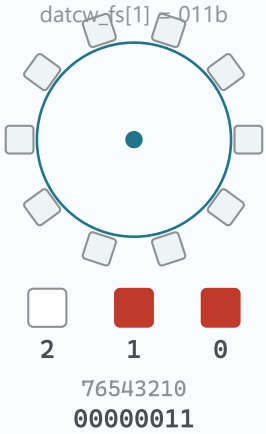
- Half-step sequence (datcw): **two** → **one** → **two** → **one** magnets.
- Full-step sequence (datcw_fs): one, two, two magnets and an “all off” byte.
- After each OUT the motor is **busy** until it reaches its new position, so the program polls bit 7 before every byte.



CW full step, byte 1: magnets on → 0

Magnet Sequences Step by Step

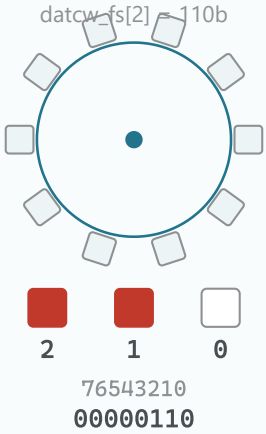
- Half-step sequence (datcw): **two** → **one** → **two** → **one** magnets.
- Full-step sequence (datcw_fs): one, two, two magnets and an “all off” byte.
- After each OUT the motor is **busy** until it reaches its new position, so the program polls bit 7 before every byte.



CW full step, byte 2: magnets on → 1, 0

Magnet Sequences Step by Step

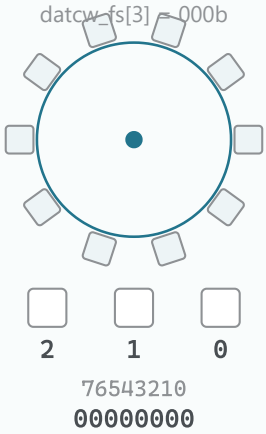
- Half-step sequence (datcw): **two** → **one** → **two** → **one** magnets.
- Full-step sequence (datcw_fs): one, two, two magnets and an “all off” byte.
- After each OUT the motor is **busy** until it reaches its new position, so the program polls bit 7 before every byte.



CW full step, byte 3: magnets on → 2, 1

Magnet Sequences Step by Step

- Half-step sequence (datcw): **two** → **one** → **two** → **one** magnets.
- Full-step sequence (datcw_fs): one, two, two magnets and an “all off” byte.
- After each OUT the motor is **busy** until it reaches its new position, so the program polls bit 7 before every byte.



CW full step, byte 4: magnets on → none

Stepper Motor Control Program

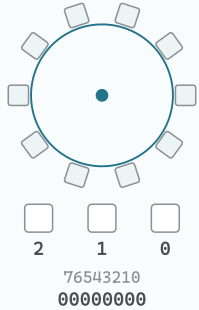
```
ORG 100h
steps_before_direction_change = 20h
jmp start
datcw     db 110b,100b,011b,010b
datccw    db 011b,001b,110b,010b
datcw_fs  db 001b,011b,110b,000b
datccw_fs db 100b,110b,011b,000b
start:
mov bx, offset datcw
mov si, 0
mov cx, 0   ; round counter
```

- **Polling** (wait): bit 7 = 0 means the motor is busy; read again.
- `[bx][si]` = [BX + SI]: BX is the table start, SI the index (0–3).
- After 4 bytes, SI = 0 and CX + 1; when CX = 20h, BX moves up 4, wrapping to the first table after the last.
- `steps_before_direction_change = 20h` is a **constant** (like EQU) and takes no memory.

```
next_step:
wait: in al, 7
test al, 10000000b
jz wait
mov al, [bx][si]
out 7, al
inc si
cmp si, 4
jb next_step
mov si, 0
inc cx
cmp cx, steps_before_direction_change
jb  next_step
mov cx, 0
add bx, 4   ; next table
cmp bx, offset datccw_fs
jbe next_step
mov bx, offset datcw
jmp next_step
```

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



ORG 100h, steps_before_direction_change = 20h.
Port 7 will return 00h (busy) first, then 80h (ready).

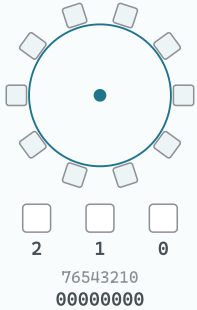
Step 1 / 44

VARIABLES

AX = 0000h | BX = 0000h | SI = 0000h | CX = 0000h | ZF = 0 | CF = 0

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



Jump over the data tables.

Step 2 / 44

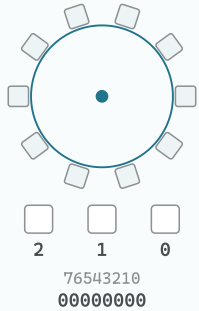
VARIABLES

AX = 0000h | BX = 0000h | SI = 0000h

CX = 0000h | ZF = 0 | CF = 0

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
▶  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



BX = 0102h: the datcw table (CW half step).

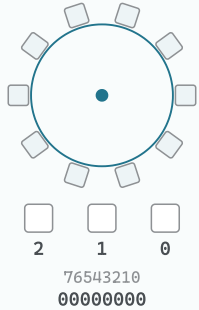
Step 3 / 44

VARIABLES

AX = 0000h | BX = 0102h | SI = 0000h | CX = 0000h | ZF = 0 | CF = 0

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
▶  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



SI = 0: first byte of the table.

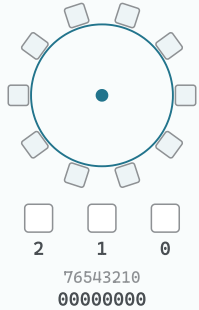
VARIABLES

AX = **0000h** | BX = **0102h** | SI = **0000h**

CX = **0000h** | ZF = **0** | CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
▶  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CX = 0: round counter.

Step 5 / 44

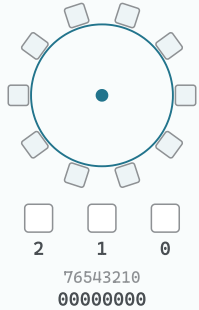
VARIABLES

AX = **0000h** | BX = **0102h** | SI = **0000h**

CX = **0000h** | ZF = **0** | CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
▶  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



IN AL, 7 → AL = 00h (bit 7 = 0, motor busy).

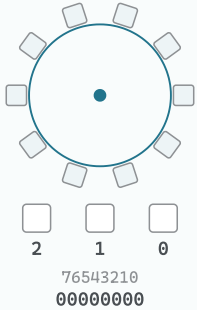
VARIABLES

AX = **0000h** | BX = **0102h** | SI = **0000h**

CX = **0000h** | ZF = **0** | CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
▶  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



TEST AL, 80h → ZF = 1.

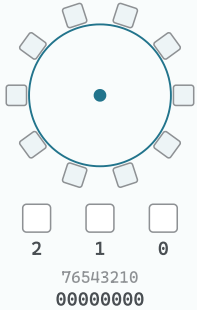
Step 7 / 44

VARIABLES

AX = 0000h | BX = 0102h | SI = 0000h | CX = 0000h | ZF = 1 | CF = 0

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
▶  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JZ: ZF = 1 → wait again.

Step 8 / 44

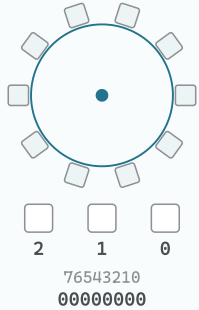
VARIABLES

AX = **0000h** BX = **0102h** SI = **0000h**

CX = **0000h** ZF = **1** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
▶  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



IN AL, 7 → AL = 80h (bit 7 = 1, ready).

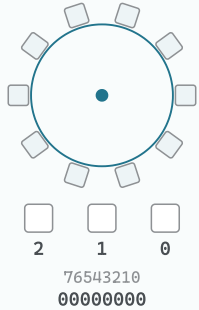
Step 9 / 44

VARIABLES

AX = 0080h | BX = 0102h | SI = 0000h | CX = 0000h | ZF = 1 | CF = 0

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
▶  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



TEST AL, 80h → ZF = 0.

Step 10 / 44

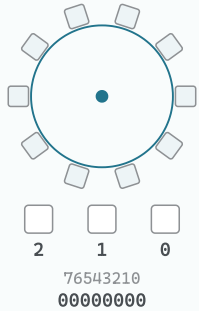
VARIABLES

AX = **0080h** | BX = **0102h** | SI = **0000h**

CX = **0000h** | ZF = **0** | CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
▶  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JZ: ZF = 0 → continue.

Step 11 / 44

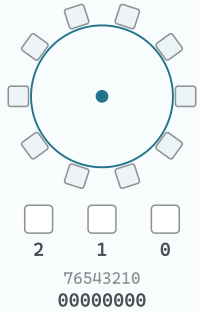
VARIABLES

AX = **0080h** BX = **0102h** SI = **0000h**

CX = **0000h** ZF = **0** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
▶  mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



AL ← [BX+SI] = [0102h] = 110b.

Step 12 / 44

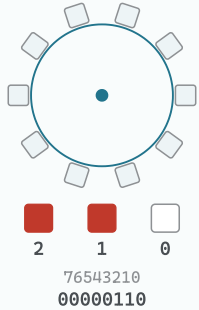
VARIABLES

AX = **0006h** | BX = **0102h** | SI = **0000h**

CX = **0000h** | ZF = **0** | CF = **0**

Stepper Motor: The First Round

```
 1 jmp start   ; tables 0102h-0111h
 2 start:
 3 mov bx, offset datcw
 4 mov si, 0
 5 mov cx, 0   ; round counter
 6 next_step:
 7 wait: in al, 7
 8 test al, 10000000b
 9 jz wait
10 mov al, [bx][si]
▶ out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



OUT 7, AL: magnets 110b → the motor turns a half step (11.25°).

Step 13 / 44

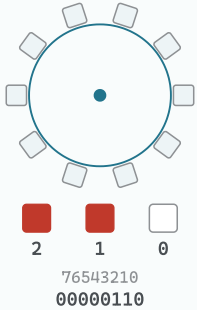
VARIABLES

AX = **0006h** BX = **0102h** SI = **0000h**

CX = **0000h** ZF = **0** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
▶ 12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



SI = 1.

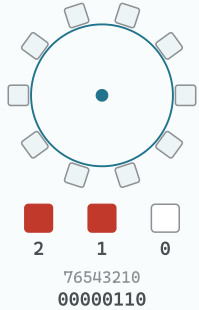
Step 14 / 44

VARIABLES

AX = 0006h | BX = 0102h | SI = 0001h | CX = 0000h | ZF = 0 | CF = 0

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
▶ cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CMP SI, 4 → CF = 1.

Step 15 / 44

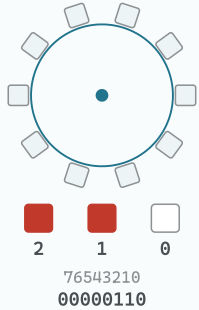
VARIABLES

AX = **0006h** BX = **0102h** SI = **0001h**

CX = **0000h** ZF = **0** CF = **1**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
▶ jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JB: SI < 4 → next_step.

Step 16 / 44

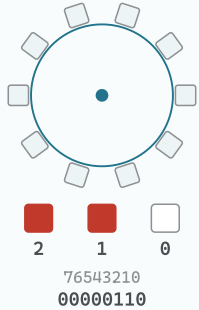
VARIABLES

AX = **0006h** | BX = **0102h** | SI = **0001h**

CX = **0000h** | ZF = **0** | CF = **1**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
▶  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



IN AL, 7 → AL = 80h (bit 7 = 1, ready).

Step 17 / 44

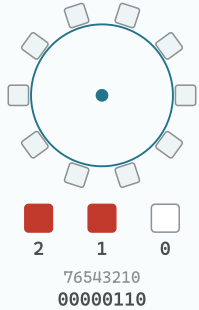
VARIABLES

AX = **0080h** | BX = **0102h** | SI = **0001h**

CX = **0000h** | ZF = **0** | CF = **1**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
▶  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



TEST AL, 80h → ZF = 0.

Step 18 / 44

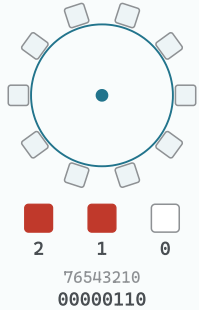
VARIABLES

AX = **0080h** BX = **0102h** SI = **0001h**

CX = **0000h** ZF = **0** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
▶  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JZ: ZF = 0 → continue.

Step 19 / 44

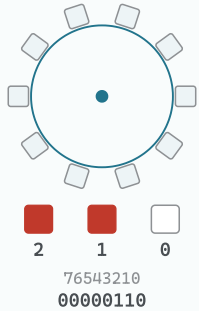
VARIABLES

AX = **0080h** BX = **0102h** SI = **0001h**

CX = **0000h** ZF = **0** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
▶  mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



AL ← [BX+SI] = [0103h] = 100b.

Step 20 / 44

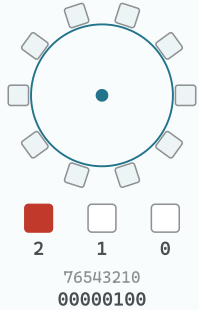
VARIABLES

AX = **0004h** | BX = **0102h** | SI = **0001h**

CX = **0000h** | ZF = **0** | CF = **0**

Stepper Motor: The First Round

```
 1 jmp start   ; tables 0102h-0111h
 2 start:
 3 mov bx, offset datcw
 4 mov si, 0
 5 mov cx, 0   ; round counter
 6 next_step:
 7 wait: in al, 7
 8 test al, 10000000b
 9 jz wait
10 mov al, [bx][si]
▶ out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



OUT 7, AL: magnets 100b → the motor turns a half step (11.25°).

Step 21 / 44

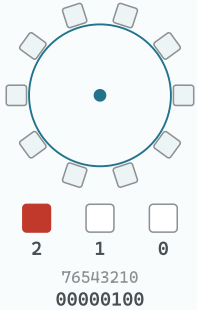
VARIABLES

AX = **0004h** BX = **0102h** SI = **0001h**

CX = **0000h** ZF = **0** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
▶ inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



SI = 2.

Step 22 / 44

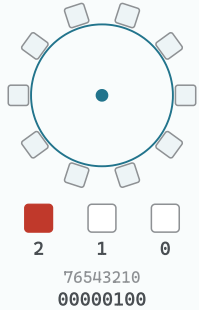
VARIABLES

AX = **0004h** BX = **0102h** SI = **0002h**

CX = **0000h** ZF = **0** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
▶ cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CMP SI, 4 → CF = 1.

Step 23 / 44

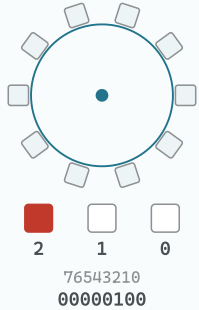
VARIABLES

AX = **0004h** BX = **0102h** SI = **0002h**

CX = **0000h** ZF = **0** CF = **1**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
▶ jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JB: SI < 4 → next_step.

Step 24 / 44

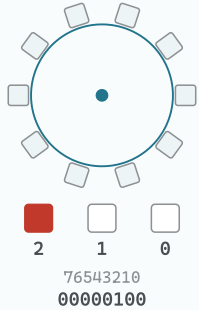
VARIABLES

AX = **0004h** BX = **0102h** SI = **0002h**

CX = **0000h** ZF = **0** CF = **1**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
▶  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



IN AL, 7 → AL = 80h (bit 7 = 1, ready).

Step 25 / 44

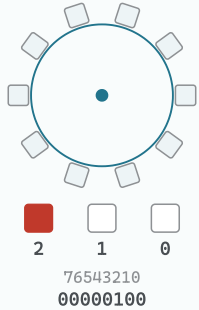
VARIABLES

AX = **0080h** | BX = **0102h** | SI = **0002h**

CX = **0000h** | ZF = **0** | CF = **1**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
▶  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



TEST AL, 80h → ZF = 0.

Step 26 / 44

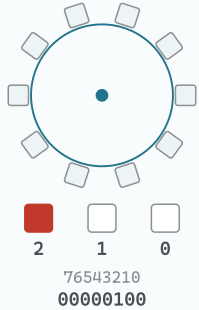
VARIABLES

AX = **0080h** BX = **0102h** SI = **0002h**

CX = **0000h** ZF = **0** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
▶  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JZ: ZF = 0 → continue.

Step 27 / 44

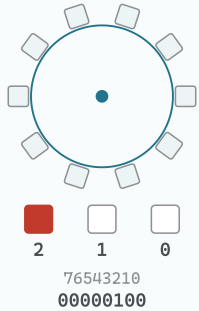
VARIABLES

AX = **0080h** BX = **0102h** SI = **0002h**

CX = **0000h** ZF = **0** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
▶  mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



AL ← [BX+SI] = [0104h] = 011b.

Step 28 / 44

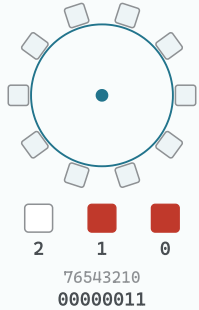
VARIABLES

AX = **0003h** | BX = **0102h** | SI = **0002h**

CX = **0000h** | ZF = **0** | CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
▶  out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



OUT 7, AL: magnets 011b → the motor turns a half step (11.25°).

Step 29 / 44

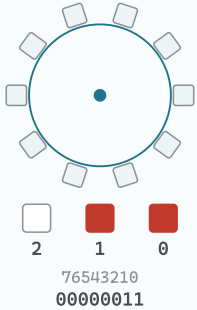
VARIABLES

AX = **0003h** BX = **0102h** SI = **0002h**

CX = **0000h** ZF = **0** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
▶ 12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



SI = 3.

Step 30 / 44

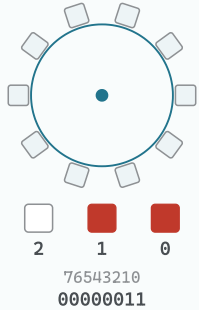
VARIABLES

AX = **0003h** | BX = **0102h** | SI = **0003h**

CX = **0000h** | ZF = **0** | CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
▶ cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CMP SI, 4 → CF = 1.

Step 31 / 44

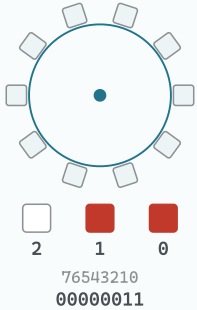
VARIABLES

AX = **0003h** BX = **0102h** SI = **0003h**

CX = **0000h** ZF = **0** CF = **1**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
▶ jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JB: SI < 4 → next_step.

Step 32 / 44

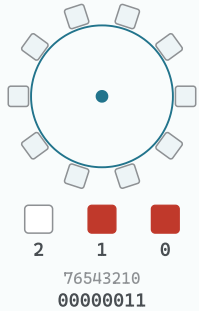
VARIABLES

AX = **0003h** BX = **0102h** SI = **0003h**

CX = **0000h** ZF = **0** CF = **1**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
▶  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



IN AL, 7 → AL = 80h (bit 7 = 1, ready).

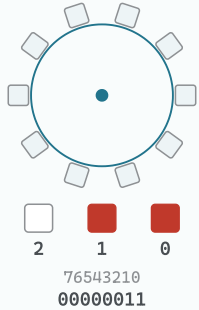
Step 33 / 44

VARIABLES

AX = 0080h | BX = 0102h | SI = 0003h | CX = 0000h | ZF = 0 | CF = 1

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
▶  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



TEST AL, 80h → ZF = 0.

Step 34 / 44

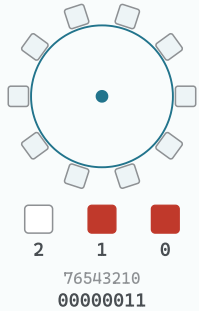
VARIABLES

AX = **0080h** | BX = **0102h** | SI = **0003h**

CX = **0000h** | ZF = **0** | CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
▶  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JZ: ZF = 0 → continue.

Step 35 / 44

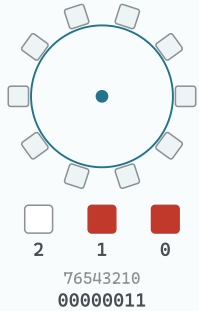
VARIABLES

AX = **0080h** BX = **0102h** SI = **0003h**

CX = **0000h** ZF = **0** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
▶  mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



AL ← [BX+SI] = [0105h] = 010b.

Step 36 / 44

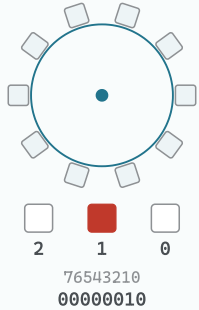
VARIABLES

AX = **0002h** | BX = **0102h** | SI = **0003h**

CX = **0000h** | ZF = **0** | CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
▶  out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



OUT 7, AL: magnets 010b → the motor turns a half step (11.25°).

Step 37 / 44

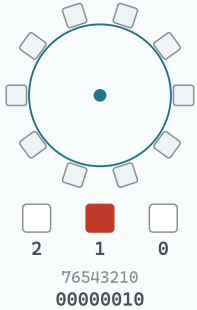
VARIABLES

AX = **0002h** BX = **0102h** SI = **0003h**

CX = **0000h** ZF = **0** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
▶ 12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



SI = 4.

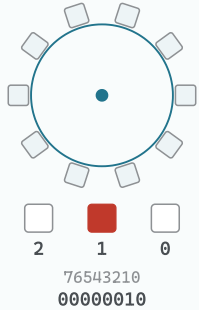
Step 38 / 44

VARIABLES

AX = 0002h | BX = 0102h | SI = 0004h | CX = 0000h | ZF = 0 | CF = 0

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
▶ cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CMP SI, 4 → CF = 0.

Step 39 / 44

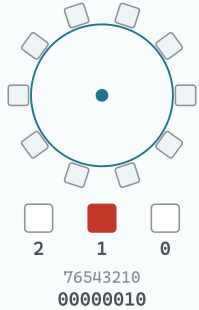
VARIABLES

AX = **0002h** | BX = **0102h** | SI = **0004h**

CX = **0000h** | ZF = **1** | CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
▶ jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JB: SI = 4: end of table.

Step 40 / 44

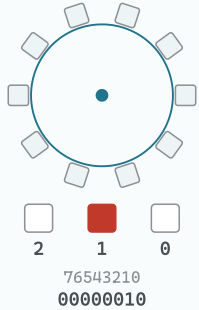
VARIABLES

AX = **0002h** BX = **0102h** SI = **0004h**

CX = **0000h** ZF = **1** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
▶  mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



SI = 0: read the table from the start.

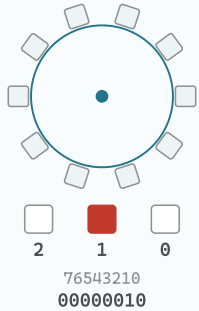
Step 41 / 44

VARIABLES

AX = **0002h** BX = **0102h** SI = **0000h**
CX = **0000h** ZF = **1** CF = **0**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
▶ 16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CX = 1: one round (4 steps) done.

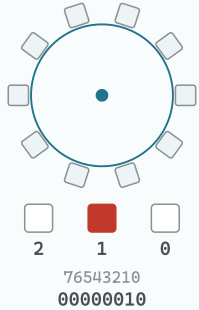
Step 42 / 44

VARIABLES

AX = 0002h | BX = 0102h | SI = 0000h | CX = 0001h | ZF = 0 | CF = 0

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
▶ cmp cx, steps_before_direction_change
18 jb  next_step
```



CMP CX, 20h → CF = 1 (not yet 32 rounds).

Step 43 / 44

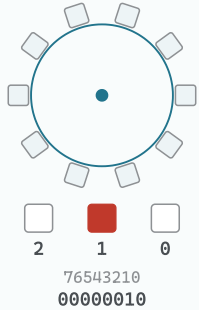
VARIABLES

AX = **0002h** BX = **0102h** SI = **0000h**

CX = **0001h** ZF = **0** CF = **1**

Stepper Motor: The First Round

```
1  jmp start   ; tables 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; round counter
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
▶  jb  next_step
```



JB next_step: carry on with the same table. After 32 rounds, `add bx, 4` moves to the next table.

Step 44 / 44

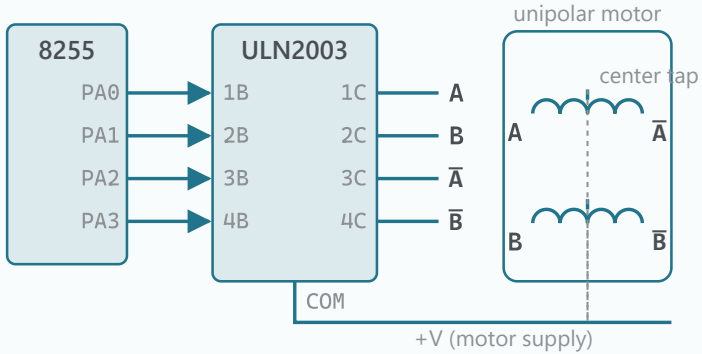
VARIABLES

AX = **0002h** BX = **0102h** SI = **0000h**

CX = **0001h** ZF = **0** CF = **1**

Real Hardware: A Unipolar Stepper and the ULN2003

- A unipolar motor has two **center-tapped** windings: four half-windings A, B, $\bar{A}$, $\bar{B}$; the center taps go to +V (5 or 6 wires).
- The winding current (hundreds of mA) is far more than a port pin can drive, so a **ULN2003** goes in between: 7 Darlington channels, up to 500 mA each.
- Input = 1 → the output pulls to ground → current flows through that half-winding.
- The back-EMF when current is cut is clamped by the ULN2003's internal diodes, with COM tied to +V.
- The motor runs open loop: there is no “ready” bit, and the delay between steps sets the speed. Too short a delay and it **misses steps**.



Real Motor: Drive Sequences

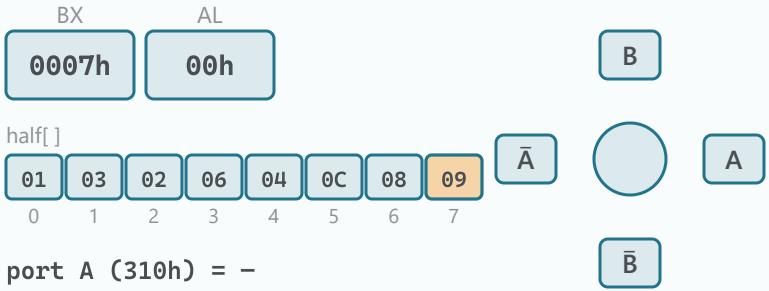
Drive	Port A sequence ($\bar{B}$ $\bar{A}$ B A = bits 3…0)
One phase (wave), full step	01h 02h 04h 08h
Two phase, full step	03h 06h 0Ch 09h
Half step	01h 03h 02h 06h 04h 0Ch 08h 09h

- With two-phase drive, two half-windings always carry current: more torque.
- Half step interleaves the two sequences: the step angle halves (e.g. 1.8° → 0.9°).
- Play the sequence **backwards** and the motor turns the other way.
- emu8086's virtual motor is a simple 3-magnet model (bits 0–2, bit 7 ready); its tables (110b, 100b, …) don't apply to a real 4-phase motor.

```
; assume port A set to output with 82h
fwd:
inc pos          ; reverse: dec pos
mov bx, pos
and bx, 7        ; index 0-7
mov al, half[bx]
mov dx, 310h     ; port A
out dx, al
call delay       ; step gap = speed
jmp fwd
delay:           ; ≈ 10 ms delay
mov cx, 2500
loop $
ret
half db 01h,03h,02h,06h
     db 04h,0Ch,08h,09h
pos  dw 0
```

Real Motor: Driving the Half-Step Sequence

```
1  org 100h
2  fwd:
▶  inc pos          ; reverse: dec pos
▶  mov bx, pos
▶  and bx, 7         ; index 0–7
6  mov al, half[bx]
7  mov dx, 310h      ; port A
8  out dx, al
9  call delay        ; step gap
10 jmp fwd
11 delay: ret        ; (delay)
12 half db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 pos  dw 6
```



Windings: bit 0 = A, bit 1 = B, bit 2 = Ā, bit 3 = B̄. With pos = 6, INC → **7**; AND BX, 7 keeps the index in 0–7 → BX = 7.

Step 1 / 8

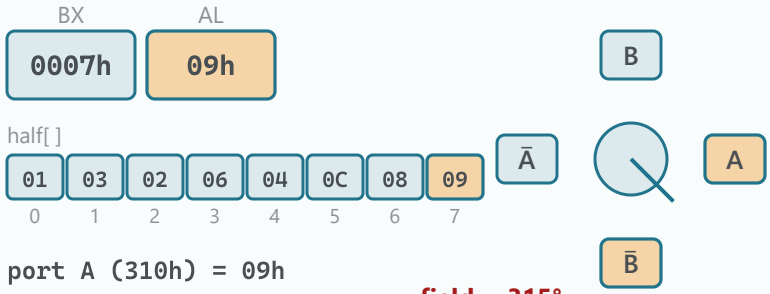
VARIABLES

`BX = 0007h` `AL = 00h` `port310h = –`

The bytes written to port 310h (09h, 01h, 03h …) come from the interpreter's I/O log; `delay` is just a RET here.

Real Motor: Driving the Half-Step Sequence

```
1  org 100h
2  fwd:
3  inc pos          ; reverse: dec pos
4  mov bx, pos
5  and bx, 7        ; index 0–7
▶  mov al, half[bx]
▶  mov dx, 310h     ; port A
▶  out dx, al
9  call delay       ; step gap
10 jmp fwd
11 delay: ret       ; (delay)
12 half db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 pos  dw 6
```



AL ← half[7] = **09h** = 1001b → windings B̄ and A: the field swings between them (≈ 315°). Written to port A.

Step 2 / 8

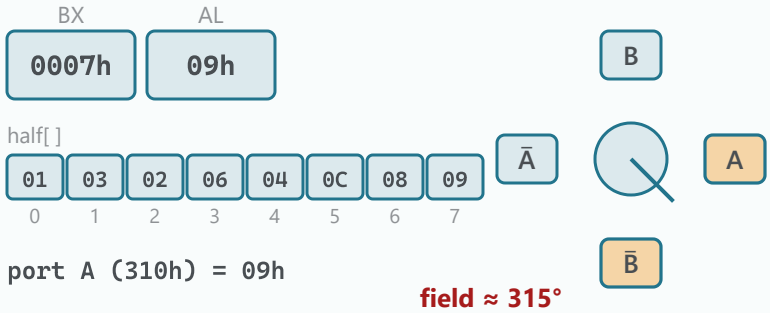
VARIABLES

`BX = 0007h` `AL = 09h` `port310h = 09h`

The bytes written to port 310h (09h, 01h, 03h …) come from the interpreter's I/O log; `delay` is just a RET here.

Real Motor: Driving the Half-Step Sequence

```
1  org 100h
2  fwd:
3  inc pos          ; reverse: dec pos
4  mov bx, pos
5  and bx, 7        ; index 0–7
6  mov al, half[bx]
7  mov dx, 310h     ; port A
8  out dx, al
▶  call delay       ; step gap
▶  jmp fwd
▶  delay: ret       ; (delay)
12 half db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 pos  dw 6
```



CALL delay / RET: the delay between steps sets the motor speed. JMP fwd.

Step 3 / 8

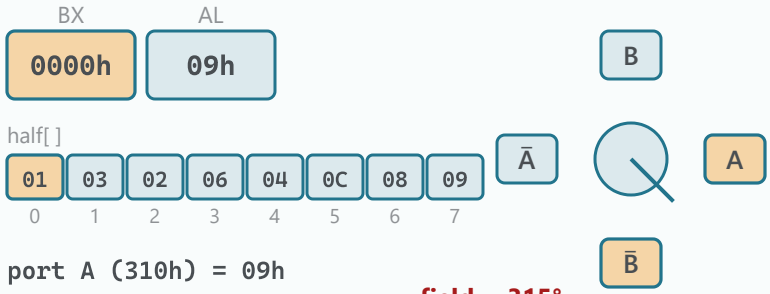
VARIABLES

`BX = 0007h` `AL = 09h` `port310h = 09h`

The bytes written to port 310h (09h, 01h, 03h …) come from the interpreter's I/O log; `delay` is just a RET here.

Real Motor: Driving the Half-Step Sequence

```
1  org 100h
2  fwd:
▶  inc pos          ; reverse: dec pos
▶  mov bx, pos
▶  and bx, 7        ; index 0-7
6  mov al, half[bx]
7  mov dx, 310h     ; port A
8  out dx, al
9  call delay       ; step gap
10 jmp fwd
11 delay: ret       ; (delay)
12 half db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 pos  dw 6
```



pos = 8 → AND 7 → BX = **0**: the sequence wraps around.

Step 4 / 8

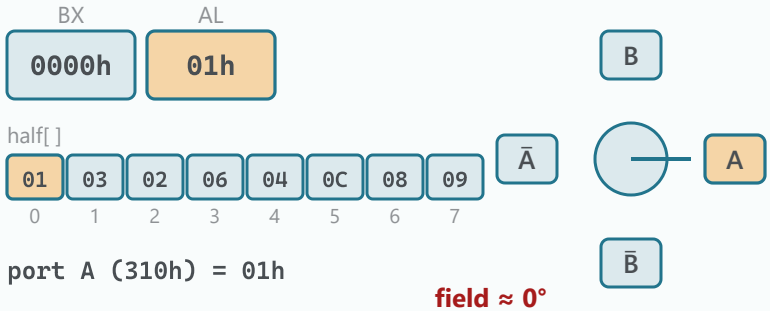
VARIABLES

`BX = 0000h` `AL = 09h` `port310h = 09h`

The bytes written to port 310h (09h, 01h, 03h …) come from the interpreter's I/O log; `delay` is just a RET here.

Real Motor: Driving the Half-Step Sequence

```
1  org 100h
2  fwd:
3  inc pos          ; reverse: dec pos
4  mov bx, pos
5  and bx, 7        ; index 0–7
▶  mov al, half[bx]
▶  mov dx, 310h     ; port A
▶  out dx, al
9  call delay       ; step gap
10 jmp fwd
11 delay: ret       ; (delay)
12 half db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 pos  dw 6
```



half[0] = **01h** → A only: the field is at 0° (a half step).

Step 5 / 8

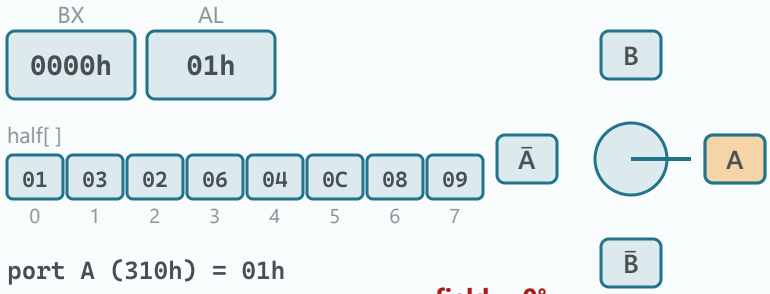
VARIABLES

`BX = 0000h` `AL = 01h` `port310h = 01h`

The bytes written to port 310h (09h, 01h, 03h …) come from the interpreter's I/O log; `delay` is just a RET here.

Real Motor: Driving the Half-Step Sequence

```
1  org 100h
2  fwd:
3  inc pos          ; reverse: dec pos
4  mov bx, pos
5  and bx, 7        ; index 0–7
6  mov al, half[bx]
7  mov dx, 310h     ; port A
8  out dx, al
▶  call delay       ; step gap
▶  jmp fwd
▶  delay: ret       ; (delay)
12 half db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 pos  dw 6
```



Wait and repeat.

Step 6 / 8

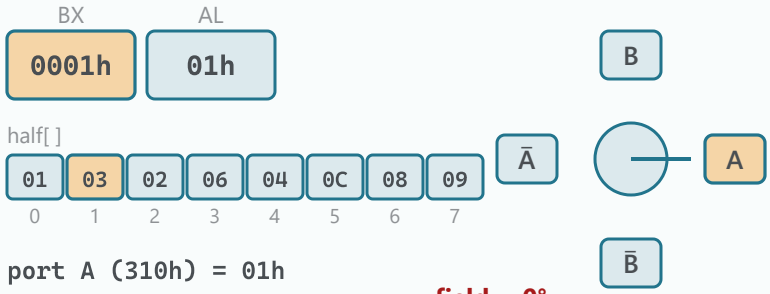
VARIABLES

`BX = 0000h` `AL = 01h` `port310h = 01h`

The bytes written to port 310h (09h, 01h, 03h …) come from the interpreter's I/O log; `delay` is just a RET here.

Real Motor: Driving the Half-Step Sequence

```
1  org 100h
2  fwd:
▶  inc pos          ; reverse: dec pos
▶  mov bx, pos
▶  and bx, 7         ; index 0-7
6  mov al, half[bx]
7  mov dx, 310h      ; port A
8  out dx, al
9  call delay        ; step gap
10 jmp fwd
11 delay: ret        ; (delay)
12 half db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 pos  dw 6
```



pos = 9 → BX = 1.

Step 7 / 8

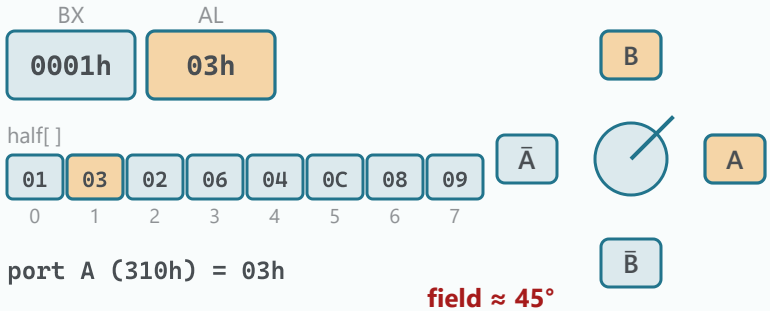
VARIABLES

`BX = 0001h` `AL = 01h` `port310h = 01h`

The bytes written to port 310h (09h, 01h, 03h …) come from the interpreter's I/O log; `delay` is just a RET here.

Real Motor: Driving the Half-Step Sequence

```
1  org 100h
2  fwd:
3  inc pos          ; reverse: dec pos
4  mov bx, pos
5  and bx, 7        ; index 0–7
▶  mov al, half[bx]
▶  mov dx, 310h     ; port A
▶  out dx, al
9  call delay       ; step gap
10 jmp fwd
11 delay: ret       ; (delay)
12 half db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 pos  dw 6
```



half[1] = **03h** → A + B: 45°. Each OUT advances the field one half step (45° winding angle); read the table backwards and the motor reverses.

Step 8 / 8

VARIABLES

`BX = 0001h` `AL = 03h` `port310h = 03h`

The bytes written to port 310h (09h, 01h, 03h …) come from the interpreter's I/O log; `delay` is just a RET here.

LED Display (Port 199)

- The **word** written to port 199 is shown as a 5-digit decimal number.
- The value is read as **signed** (two's complement): −32,768 … 32,767.
- −5678 = 10000h − 162Eh = **E9D2h**; the display shows **-5678**.
- Only `OUT 199, AX` is needed (199 ≤ 255, so an immediate port works).
- `#make_bin#`: produces a plain binary (.bin) file that doesn't depend on DOS.



port 199 (2 bytes) · AX = 0163h

Driving the LED Display

```
#start=led_display.exe#
mov ax, 1234
out 199, ax
mov ax, -5678
out 199, ax
; write the port in an endless loop:
mov ax, 0
x1:
out 199, ax
inc ax
jmp x1
hlt              ; never runs
```



port 199 (2 bytes) · AX = 0000h

The display starts at 00000.

Step 1 / 15

VARIABLES

AX = **0000h**

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 0000h

AX = 1234 (decimal) = **04D2h**.

Step 2 / 15

VARIABLES

AX = 04D2h

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 04D2h

OUT 199, AX: display shows **1234**.

Step 3 / 15

VARIABLES

AX = **04D2h**

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
▶  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 04D2h

AX = −5678 = **E9D2h** (two's complement).

Step 4 / 15

VARIABLES

AX = **E9D2h**

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
▶  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = E9D2h

OUT 199, AX: display shows **-5678**.

Step 5 / 15

VARIABLES

AX = **E9D2h**

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = E9D2h

AX = 0: the counter.

Step 6 / 15

VARIABLES

AX = 0000h

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
▶  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 0000h

OUT 199, AX: display shows **0**.

Step 7 / 15

VARIABLES

AX = **0000h**

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
9  out 199, ax
▶  inc ax
11 jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 0000h

AX = 1.

Step 8 / 15

VARIABLES

AX = 0001h

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
▶  jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 0000h

JMP x1: endless loop (the HLT below is never reached).

Step 9 / 15

VARIABLES

AX = **0001h**

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
▶  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 0001h

OUT 199, AX: display shows **1**.

Step 10 / 15

VARIABLES

AX = **0001h**

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
9  out 199, ax
▶  inc ax
11 jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 0001h

AX = 2.

Step 11 / 15

VARIABLES

AX = 0002h

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
▶  jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 0001h

JMP x1: endless loop (the HLT below is never reached).

Step 12 / 15

VARIABLES

AX = **0002h**

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
▶  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 0002h

OUT 199, AX: display shows **2**.

Step 13 / 15

VARIABLES

AX = **0002h**

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
9  out 199, ax
▶  inc ax
11 jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 0002h

AX = 3.

Step 14 / 15

VARIABLES

`AX = 0003h`

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Driving the LED Display

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; write the port in an endless loop:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
▶  jmp x1
12 hlt              ; never runs
```



port 199 (2 bytes) · AX = 0002h

JMP x1: endless loop (the HLT below is never reached); after 32767 the counter wraps to −32768.

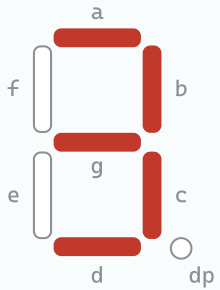
Step 15 / 15

VARIABLES

AX = **0003h**

With emu8086's step delay at 0 the numbers fly by; add an INT 15h/86h wait to the loop.

Real Hardware: The 7-Segment Display



PA7 … PA0 → dp g f e d c b a

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	0	1	1	1	1
dp	g	f	e	d	c	b	a

common cathode '3': `4Fh`

common anode '3': `B0h` (inverse)

- **Common cathode** (CC): common pin to ground; bit = 1 lights a segment.
- **Common anode** (CA): common pin at +5 V; bit = 0 lights it, so the code is the inverse of the CC code (`NOT AL`).
- Each segment gets a current-limiting resistor.
- emu8086's port 199 LED display decodes the number itself; with a real display, the program produces the code.

Digit	Lit segments	CC	CA
0	a b c d e f	3Fh	C0h
1	b c	06h	F9h
2	a b d e g	5Bh	A4h
3	a b c d g	4Fh	B0h
4	b c f g	66h	99h
5	a c d f g	6Dh	92h
6	a c d e f g	7Dh	82h
7	a b c	07h	F8h
8	a b c d e f g	7Fh	80h
9	a b c d f g	6Fh	90h

Code Table, XLAT and Multiplexed Scanning

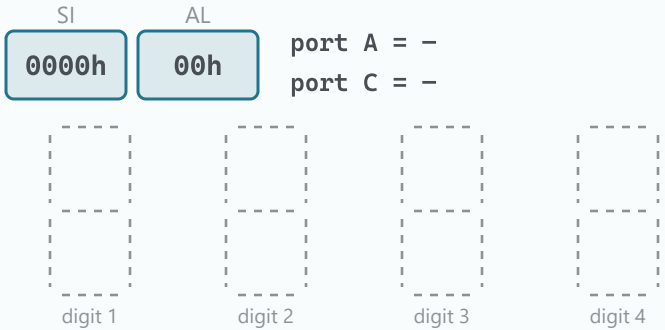
```
; digit (0–9) to segment code
mov bx, offset segtab
mov al, 3          ; digit
xlat               ; AL = [BX + AL] = 4Fh
; not al           ; if common anode
mov dx, 310h       ; 8255 port A
out dx, al
ret
segtab db 3Fh,06h,5Bh,4Fh,66h
    db 6Dh,7Dh,07h,7Fh,6Fh
```

- **Multiplexing:** the segment lines of all 4 digits share port A; each digit's common pin goes to port C through its own transistor.
- Only one digit is lit at a time; ≈ 4 ms per digit → a full pass every 16 ms ≈ **62 Hz**. The eye sees no flicker (above ≈ 50 Hz).
- Blank everything before switching digits, or the previous code shows faintly on the next digit (ghosting).

```
scan:
mov si, 0
next_dig:
mov dx, 312h       ; port C: digit select
mov al, 0
out dx, al         ; blank all digits first
mov al, digits[si]
mov bx, offset segtab   ; table on the left
xlat
mov dx, 310h       ; port A: segments
out dx, al
mov al, select[si] ; 01h, 02h, 04h, 08h
mov dx, 312h
out dx, al         ; only this digit lit
call delay_4ms
inc si
cmp si, 4
jb  next_dig
jmp scan
delay_4ms:          ; ≈ 4 ms delay
mov cx, 1000
loop $
ret
digits db 1, 2, 3, 4
select db 01h, 02h, 04h, 08h
```

Multiplexed Scanning: Showing 2026

```
 1 org 100h
 2 mov bx, offset segtab
 3 mov si, 0
 4 next_dig:
 5 mov dx, 312h  ; port C (digit)
 6 mov al, 0
 7 out dx, al    ; all blank
 8 mov al, digits[si]
 9 xlat          ; segment code
10 mov dx, 310h  ; port A (segs)
11 out dx, al
12 mov al, select[si]
13 mov dx, 312h
14 out dx, al    ; one digit lit
15 inc si
16 cmp si, 4
17 jb  next_dig
18 ret
19 segtab   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 digits db 2, 0, 2, 6
22 select db 01h, 02h, 04h, 08h
```



BX = table start, SI = 0. The number to show is **2026** (the digits array); all 4 digits share their segment lines on port A.

Step 1 / 10

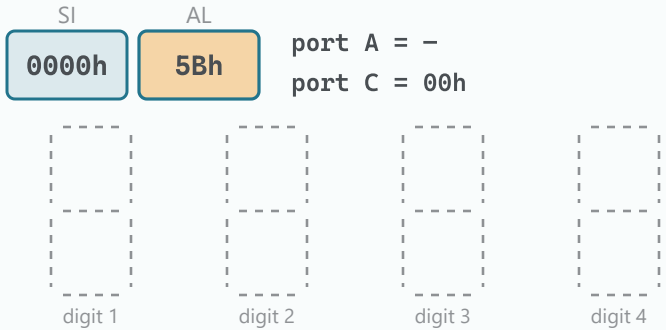
VARIABLES

`SI = 0000h` `AL = 00h` `A = –` `C = –`

Common-cathode codes; port values from the interpreter's I/O log. In practice each digit stays lit for ≈ 4 ms.

Multiplexed Scanning: Showing 2026

```
 1 org 100h
 2 mov bx, offset segtab
 3 mov si, 0
 4 next_dig:
 5 mov dx, 312h  ; port C (digit)
 6 mov al, 0
 7 out dx, al    ; all blank
 8 mov al, digits[si]
 9 xlat          ; segment code
10 mov dx, 310h  ; port A (segs)
11 out dx, al
12 mov al, select[si]
13 mov dx, 312h
14 out dx, al    ; one digit lit
15 inc si
16 cmp si, 4
17 jb  next_dig
18 ret
19 segtab   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 digits db 2, 0, 2, 6
22 select db 01h, 02h, 04h, 08h
```



Digit 1: first port C ← 00h (all blank, no ghosting). AL = digits[0] = 2; XLAT → AL = **5Bh**.

Step 2 / 10

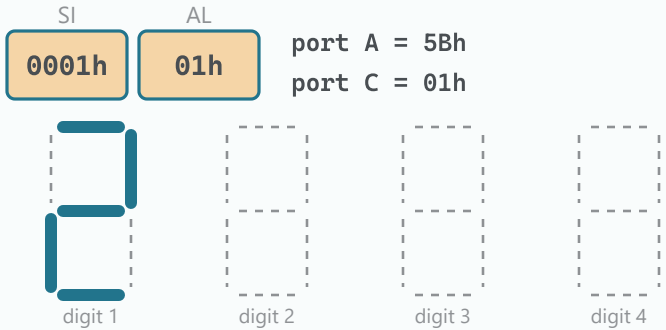
VARIABLES

`SI = 0000h` `AL = 5Bh` `A = –` `C = 00h`

Common-cathode codes; port values from the interpreter's I/O log. In practice each digit stays lit for ≈ 4 ms.

Multiplexed Scanning: Showing 2026

```
 1 org 100h
 2 mov bx, offset segtab
 3 mov si, 0
 4 next_dig:
 5 mov dx, 312h  ; port C (digit)
 6 mov al, 0
 7 out dx, al    ; all blank
 8 mov al, digits[si]
 9 xlat          ; segment code
▶ mov dx, 310h  ; port A (segs)
▶ out dx, al
▶ mov al, select[si]
▶ mov dx, 312h
▶ out dx, al    ; one digit lit
▶ inc si
▶ cmp si, 4
▶ jb  next_dig
18 ret
19 segtab   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 digits db 2, 0, 2, 6
22 select db 01h, 02h, 04h, 08h
```



Port A ← 5Bh (segments), port C ← 01h: only digit 1 is lit. INC SI, CMP, JB → next digit.

Step 3 / 10

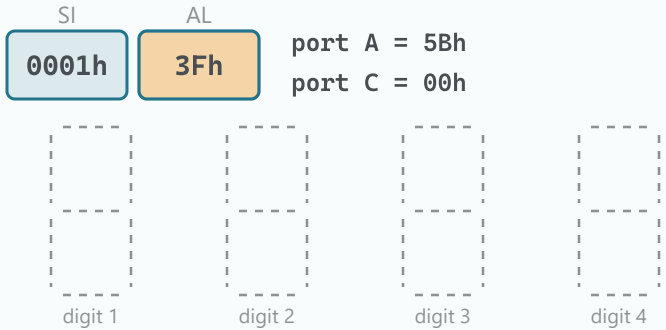
VARIABLES

`SI = 0001h` `AL = 01h` `A = 5Bh` `C = 01h`

Common-cathode codes; port values from the interpreter's I/O log. In practice each digit stays lit for ≈ 4 ms.

Multiplexed Scanning: Showing 2026

```
 1 org 100h
 2 mov bx, offset segtab
 3 mov si, 0
 4 next_dig:
 ▶ mov dx, 312h  ; port C (digit)
 ▶ mov al, 0
 ▶ out dx, al    ; all blank
 ▶ mov al, digits[si]
 ▶ xlat          ; segment code
10 mov dx, 310h  ; port A (segs)
11 out dx, al
12 mov al, select[si]
13 mov dx, 312h
14 out dx, al    ; one digit lit
15 inc si
16 cmp si, 4
17 jb  next_dig
18 ret
19 segtab   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 digits db 2, 0, 2, 6
22 select db 01h, 02h, 04h, 08h
```



Digit 2: first port C ← 00h (all blank, no ghosting). AL = digits[1] = 0; XLAT → AL = **3Fh**.

Step 4 / 10

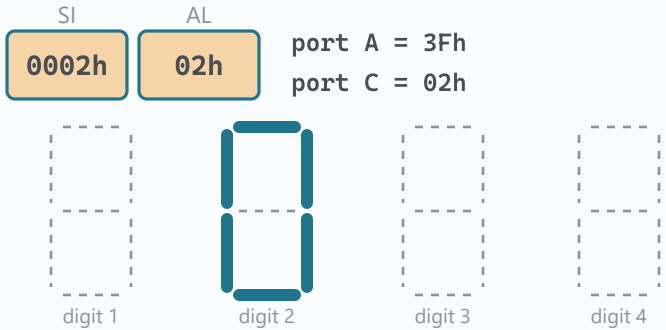
VARIABLES

`SI = 0001h` `AL = 3Fh` `A = 5Bh` `C = 00h`

Common-cathode codes; port values from the interpreter's I/O log. In practice each digit stays lit for ≈ 4 ms.

Multiplexed Scanning: Showing 2026

```
 1 org 100h
 2 mov bx, offset segtab
 3 mov si, 0
 4 next_dig:
 5 mov dx, 312h  ; port C (digit)
 6 mov al, 0
 7 out dx, al    ; all blank
 8 mov al, digits[si]
 9 xlat          ; segment code
▶ mov dx, 310h  ; port A (segs)
▶ out dx, al
▶ mov al, select[si]
▶ mov dx, 312h
▶ out dx, al    ; one digit lit
▶ inc si
▶ cmp si, 4
▶ jb  next_dig
18 ret
19 segtab   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 digits db 2, 0, 2, 6
22 select db 01h, 02h, 04h, 08h
```



Port A ← 3Fh (segments), port C ← 02h: only digit 2 is lit. INC SI, CMP, JB → next digit.

Step 5 / 10

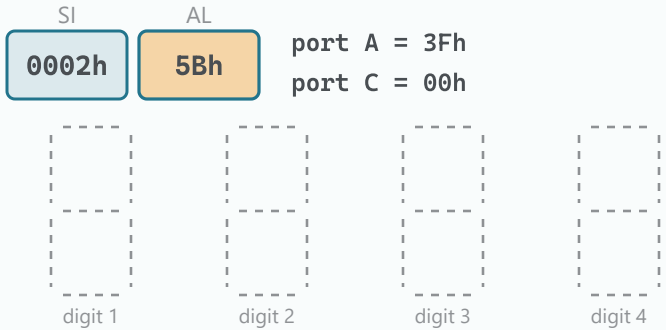
VARIABLES

`SI = 0002h` `AL = 02h` `A = 3Fh` `C = 02h`

Common-cathode codes; port values from the interpreter's I/O log. In practice each digit stays lit for ≈ 4 ms.

Multiplexed Scanning: Showing 2026

```
 1 org 100h
 2 mov bx, offset segtab
 3 mov si, 0
 4 next_dig:
 ▶ mov dx, 312h  ; port C (digit)
 ▶ mov al, 0
 ▶ out dx, al    ; all blank
 ▶ mov al, digits[si]
 ▶ xlat          ; segment code
10 mov dx, 310h  ; port A (segs)
11 out dx, al
12 mov al, select[si]
13 mov dx, 312h
14 out dx, al    ; one digit lit
15 inc si
16 cmp si, 4
17 jb  next_dig
18 ret
19 segtab   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 digits db 2, 0, 2, 6
22 select db 01h, 02h, 04h, 08h
```



Digit 3: first port C ← 00h (all blank, no ghosting). AL = digits[2] = 2; XLAT → AL = **5Bh**.

Step 6 / 10

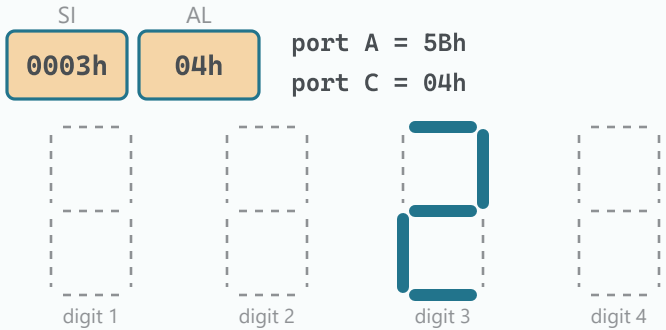
VARIABLES

`SI = 0002h` `AL = 5Bh` `A = 3Fh` `C = 00h`

Common-cathode codes; port values from the interpreter's I/O log. In practice each digit stays lit for ≈ 4 ms.

Multiplexed Scanning: Showing 2026

```
 1 org 100h
 2 mov bx, offset segtab
 3 mov si, 0
 4 next_dig:
 5 mov dx, 312h  ; port C (digit)
 6 mov al, 0
 7 out dx, al    ; all blank
 8 mov al, digits[si]
 9 xlat          ; segment code
▶ mov dx, 310h  ; port A (segs)
▶ out dx, al
▶ mov al, select[si]
▶ mov dx, 312h
▶ out dx, al    ; one digit lit
▶ inc si
▶ cmp si, 4
▶ jb  next_dig
18 ret
19 segtab   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 digits db 2, 0, 2, 6
22 select db 01h, 02h, 04h, 08h
```



Port A ← 5Bh (segments), port C ← 04h: only digit 3 is lit. INC SI, CMP, JB → next digit.

Step 7 / 10

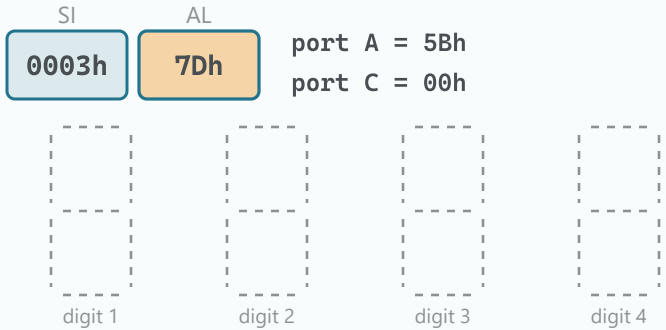
VARIABLES

`SI = 0003h` `AL = 04h` `A = 5Bh` `C = 04h`

Common-cathode codes; port values from the interpreter's I/O log. In practice each digit stays lit for ≈ 4 ms.

Multiplexed Scanning: Showing 2026

```
 1 org 100h
 2 mov bx, offset segtab
 3 mov si, 0
 4 next_dig:
 ▶ mov dx, 312h  ; port C (digit)
 ▶ mov al, 0
 ▶ out dx, al    ; all blank
 ▶ mov al, digits[si]
 ▶ xlat          ; segment code
10 mov dx, 310h  ; port A (segs)
11 out dx, al
12 mov al, select[si]
13 mov dx, 312h
14 out dx, al    ; one digit lit
15 inc si
16 cmp si, 4
17 jb  next_dig
18 ret
19 segtab   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 digits db 2, 0, 2, 6
22 select db 01h, 02h, 04h, 08h
```



Digit 4: first port C ← 00h (all blank, no ghosting). AL = digits[3] = 6; XLAT → AL = **7Dh**.

Step 8 / 10

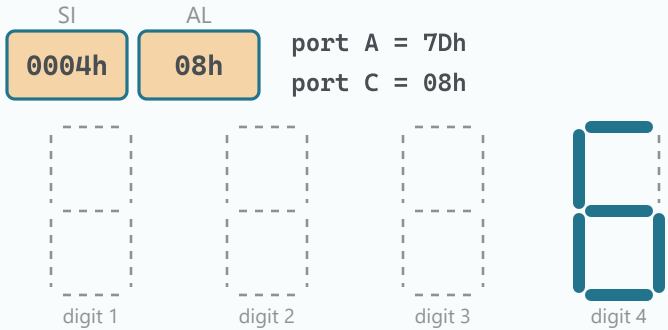
VARIABLES

`SI = 0003h` `AL = 7Dh` `A = 5Bh` `C = 00h`

Common-cathode codes; port values from the interpreter's I/O log. In practice each digit stays lit for ≈ 4 ms.

Multiplexed Scanning: Showing 2026

```
 1 org 100h
 2 mov bx, offset segtab
 3 mov si, 0
 4 next_dig:
 5 mov dx, 312h  ; port C (digit)
 6 mov al, 0
 7 out dx, al    ; all blank
 8 mov al, digits[si]
 9 xlat          ; segment code
▶ mov dx, 310h  ; port A (segs)
▶ out dx, al
▶ mov al, select[si]
▶ mov dx, 312h
▶ out dx, al    ; one digit lit
▶ inc si
▶ cmp si, 4
▶ jb  next_dig
18 ret
19 segtab   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 digits db 2, 0, 2, 6
22 select db 01h, 02h, 04h, 08h
```



Port A ← 7Dh (segments), port C ← 08h: only digit 4 is lit. INC SI, CMP, JB → SI = 4, the loop ends.

Step 9 / 10

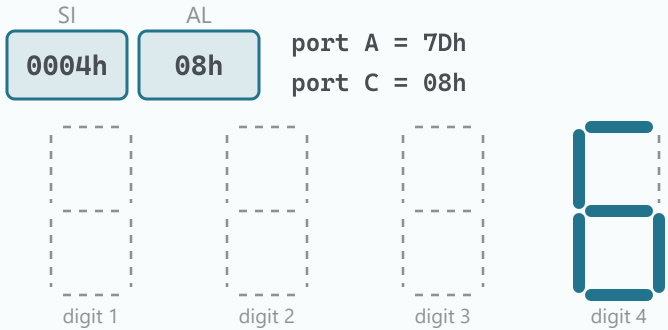
VARIABLES

`SI = 0004h` `AL = 08h` `A = 7Dh` `C = 08h`

Common-cathode codes; port values from the interpreter's I/O log. In practice each digit stays lit for ≈ 4 ms.

Multiplexed Scanning: Showing 2026

```
 1 org 100h
 2 mov bx, offset segtab
 3 mov si, 0
 4 next_dig:
 5 mov dx, 312h  ; port C (digit)
 6 mov al, 0
 7 out dx, al    ; all blank
 8 mov al, digits[si]
 9 xlat          ; segment code
10 mov dx, 310h  ; port A (segs)
11 out dx, al
12 mov al, select[si]
13 mov dx, 312h
14 out dx, al    ; one digit lit
15 inc si
16 cmp si, 4
17 jb  next_dig
▶ ret
19 segtab   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 digits db 2, 0, 2, 6
22 select db 01h, 02h, 04h, 08h
```



Each of the four digits lit once in turn. At ≈ 4 ms per digit a full pass takes 16 ms (≈ 62 Hz): the eye sees all four at once.

Step 10 / 10

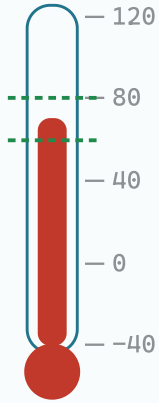
VARIABLES

`SI = 0004h` `AL = 08h` `A = 7Dh` `C = 08h`

Common-cathode codes; port values from the interpreter's I/O log. In practice each digit stays lit for ≈ 4 ms.

Thermometer and Heater (Ports 125, 127)

- `IN AL, 125` : reads the current air temperature (°C, a **signed byte**).
- `OUT 127, AL` : AL = 1 turns the heater on, AL = 0 turns it off.
- With the heater on, the temperature slowly rises; with it off, it falls.
- Goal: a **thermostat** that keeps the temperature within **60–80 °C**.
- The temperature can be negative, so comparisons use **signed** jumps (`JL` , `JLE` , `JG`).



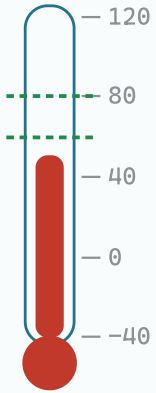
port 125: **70 °C**

heater OFF

port 127 = 0
target range: 60–80 °C

The Thermostat Program

```
; make DS equal to CS:
mov ax, cs
mov ds, ax
start:
in  al, 125
cmp al, 60
jl  low
cmp al, 80
jle ok
jg  high
low:
mov al, 1
out 127, al   ; heater on
jmp ok
high:
mov al, 0
out 127, al   ; heater off
ok:
jmp start     ; endless loop
```



port 125: 50 °C

heater OFF

port 127 = 0
target range: 60–80 °C

Port 125 will return 55, 70 and 85 °C in turn. The heater starts off.

Step 1 / 25

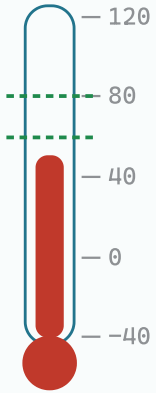
VARIABLES

AX = 0000h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
▶  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 50 °C

heater OFF

port 127 = 0
target range: 60–80 °C

AX = CS = 0700h.

Step 2 / 25

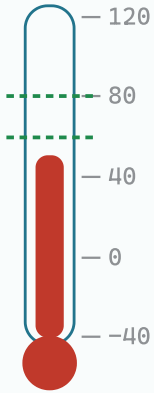
VARIABLES

AX = 0700h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; heater off
18 ok:
19 jmp start     ; endless loop
```



port 125: 50 °C

heater OFF

port 127 = 0
target range: 60–80 °C

DS = 0700h (an immediate can't go straight into a segment register).

Step 3 / 25

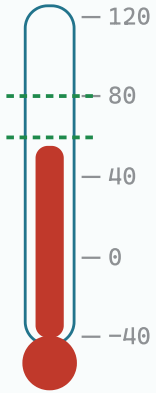
VARIABLES

AX = 0700h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
▶ in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; heater off
18 ok:
19 jmp start     ; endless loop
```



port 125: 55 °C

heater OFF

port 127 = 0
target range: 60–80 °C

IN AL, 125 → temperature **55 °C**.

Step 4 / 25

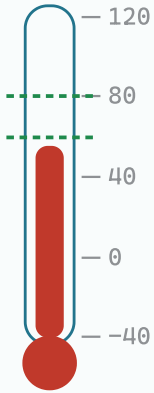
VARIABLES

AX = 0737h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 55 °C

heater OFF

port 127 = 0
target range: 60–80 °C

CMP AL, 60: 55 − 60 → SF = 1, OF = 0.

Step 5 / 25

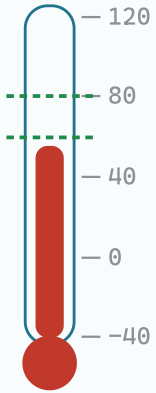
VARIABLES

AX = 0737h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
▶ 7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 55 °C

heater OFF

port 127 = 0
target range: 60–80 °C

JL (SF ≠ OF): **55 < 60** → low.

Step 6 / 25

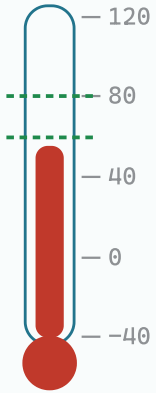
VARIABLES

AX = **0737h** DS = **0700h** ZF = **0** SF = **1** OF = **0**

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
▶ mov al, 1
13 out 127, al   ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; heater off
18 ok:
19 jmp start     ; endless loop
```



port 125: 55 °C

heater OFF

port 127 = 0
target range: 60–80 °C

AL = 1.

Step 7 / 25

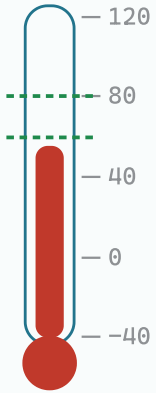
VARIABLES

AX = 0701h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
▶  out 127, al   ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; heater off
18 ok:
19 jmp start     ; endless loop
```



port 125: 55 °C

heater ON

port 127 = 1
target range: 60–80 °C

OUT 127, AL: heater **on**.

Step 8 / 25

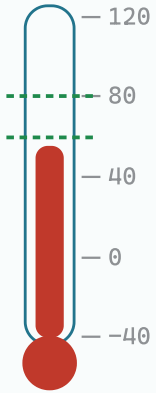
VARIABLES

AX = **0701h** DS = **0700h** ZF = **0** SF = **1** OF = **0**

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
   jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 55 °C

heater ON

port 127 = 1

target range: 60–80 °C

JMP ok.

Step 9 / 25

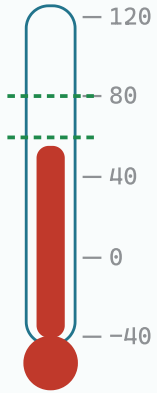
VARIABLES

AX = **0701h** DS = **0700h** ZF = **0** SF = **1** OF = **0**

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
▶  jmp start      ; endless loop
```



port 125: 55 °C

heater ON

port 127 = 1
target range: 60–80 °C

JMP start: new reading.

Step 10 / 25

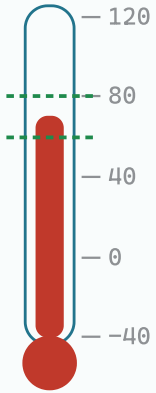
VARIABLES

AX = 0701h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
▶ in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; heater off
18 ok:
19 jmp start     ; endless loop
```



port 125: **70 °C**

heater ON

port 127 = 1
target range: 60–80 °C

IN AL, 125 → temperature **70 °C**.

Step 11 / 25

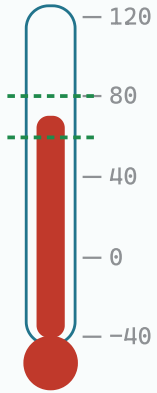
VARIABLES

AX = **0746h** | DS = **0700h** | ZF = **0** | SF = **1** | OF = **0**

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 70 °C

heater ON

port 127 = 1

target range: 60–80 °C

CMP AL, 60: 70 − 60 → SF = 0, OF = 0.

Step 12 / 25

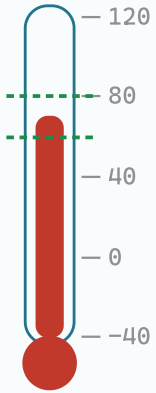
VARIABLES

AX = 0746h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
▶  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 70 °C

heater ON

port 127 = 1

target range: 60–80 °C

JL (SF ≠ OF): no jump.

Step 13 / 25

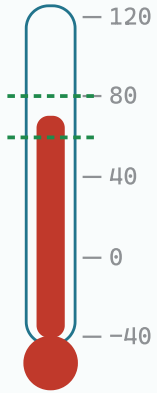
VARIABLES

AX = 0746h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
▶  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 70 °C

heater ON

port 127 = 1
target range: 60–80 °C

CMP AL, 80 → ZF = 0, SF = 1.

Step 14 / 25

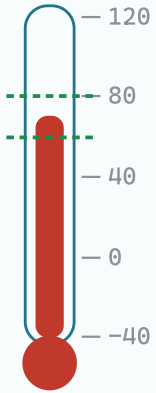
VARIABLES

AX = 0746h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
▶  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 70 °C

heater ON

port 127 = 1
target range: 60–80 °C

JLE: 60 ≤ 70 ≤ 80 → ok (leave the heater alone).

Step 15 / 25

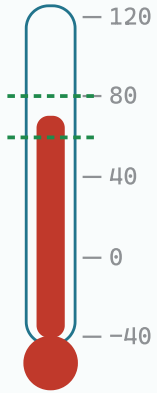
VARIABLES

AX = **0746h** | DS = **0700h** | ZF = **0** | SF = **1** | OF = **0**

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; heater off
18 ok:
▶  jmp start     ; endless loop
```



port 125: 70 °C

heater ON

port 127 = 1
target range: 60–80 °C

JMP start: new reading.

Step 16 / 25

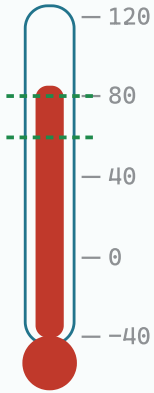
VARIABLES

AX = 0746h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
▶ in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; heater off
18 ok:
19 jmp start     ; endless loop
```



port 125: 85 °C

heater ON

port 127 = 1
target range: 60–80 °C

IN AL, 125 → temperature **85 °C**.

Step 17 / 25

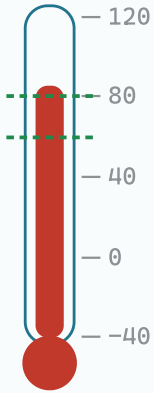
VARIABLES

AX = 0755h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 85 °C

heater ON

port 127 = 1

target range: 60–80 °C

CMP AL, 60: 85 − 60 → SF = 0, OF = 0.

Step 18 / 25

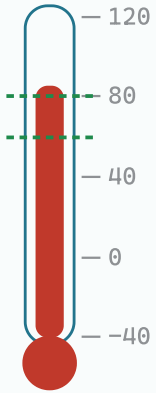
VARIABLES

AX = **0755h** | DS = **0700h** | ZF = **0** | SF = **0** | OF = **0**

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
▶  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 85 °C

heater ON

port 127 = 1

target range: 60–80 °C

JL (SF ≠ OF): no jump.

Step 19 / 25

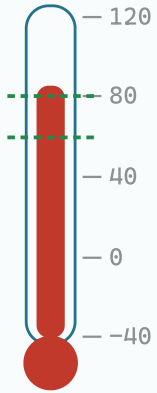
VARIABLES

AX = **0755h** DS = **0700h** ZF = **0** SF = **0** OF = **0**

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
▶  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 85 °C

heater ON

port 127 = 1

target range: 60–80 °C

CMP AL, 80 → ZF = 0, SF = 0.

Step 20 / 25

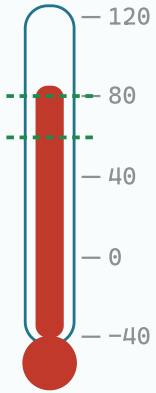
VARIABLES

AX = **0755h** DS = **0700h** ZF = **0** SF = **0** OF = **0**

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
▶  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 85 °C

heater ON

port 127 = 1

target range: 60–80 °C

JLE: no jump.

Step 21 / 25

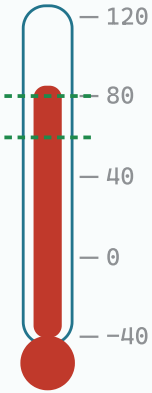
VARIABLES

AX = **0755h** DS = **0700h** ZF = **0** SF = **0** OF = **0**

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
▶  jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 85 °C

heater ON

port 127 = 1
target range: 60–80 °C

JG: 85 > 80 → high.

Step 22 / 25

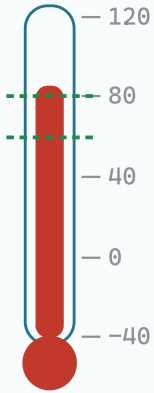
VARIABLES

AX = **0755h** | DS = **0700h** | ZF = **0** | SF = **0** | OF = **0**

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
▶ mov al, 0
17 out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 85 °C

heater ON

port 127 = 1

target range: 60–80 °C

AL = 0.

Step 23 / 25

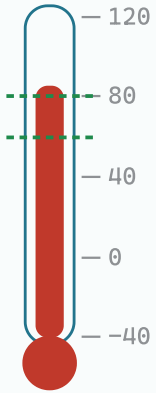
VARIABLES

AX = 0700h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al    ; heater on
14 jmp ok
15 high:
16 mov al, 0
▶  out 127, al    ; heater off
18 ok:
19 jmp start      ; endless loop
```



port 125: 85 °C

heater OFF

port 127 = 0
target range: 60–80 °C

OUT 127, AL: heater **off**.

Step 24 / 25

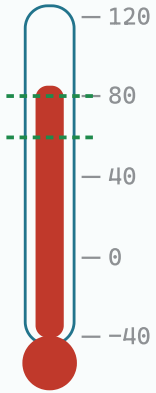
VARIABLES

AX = **0700h** DS = **0700h** ZF = **0** SF = **0** OF = **0**

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

The Thermostat Program

```
1  ; make DS equal to CS:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; heater on
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; heater off
18 ok:
▶  jmp start     ; endless loop
```



port 125: 85 °C

heater OFF

port 127 = 0
target range: 60–80 °C

JMP start: new reading.

Step 25 / 25

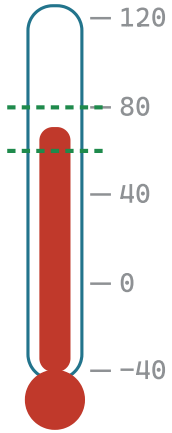
VARIABLES

AX = 0700h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` could just as well be `jmp high`: if JLE didn't jump, the temperature is already above 80.

THERMOMETER

Interactive Thermostat



port 125: **70** °C

heater OFF

port 127 = 0

Temperature: **70** °C

▶ run time

```
in al, 125
cmp al, 60
jl low
cmp al, 80
jle ok
jg high
low: mov al, 1
out 127, al
jmp ok
high: mov al, 0
out 127, al
ok: jmp start
```

60 ≤ 70 ≤ 80 → JLE taken, heater left alone (stays off).

AL = 46h

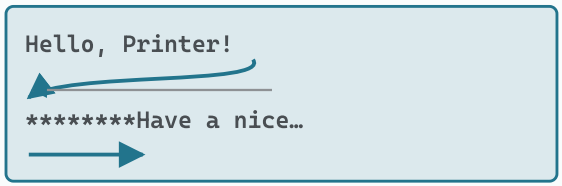
Printer (Port 130)

- The emu8086 printer sits on port 130, but instead of writing the port directly the program uses a **DOS function**:
- `INT 21h`, `AH = 05h`: sends the character in DL to the printer (PRN).
- Control characters drive the printer:

Code	Name	Effect
0Ah (10)	LF	one line down
0Dh (13)	CR	back to line start
09h (9)	HT	horizontal tab
0Ch (12)	FF	new page (form feed)

`db 13, 9`: 13 = CR (carriage return), 9 = **horizontal** tab (HT). Vertical tab (VT) is 11 = 0Bh.

`0Ah, 0Dh` is LF + CR; the usual order is CR + LF, but together they give the same result.



0Ah, 0Dh: start of next line

CR (13): back to start of this line

HT (9): jump to column 8 → overprint

FF (12): eject, new page

Printer Program: Data

```
org 100h
jmp start
msg db "Hello, Printer!", 0Ah, 0Dh
    db "***************"
    db 13, 9    ; CR and horizontal tab (HT)
    db "Have a nice printing day!"
msg_end db 0
msg2 db "press any key to eject the page.$"
```

Length

`offset msg_end - offset msg` is computed at assembly time: 15 + 2 + 15 + 2 + 25 = **59** = 3Bh bytes.

msg_end

The address just past the last character; its value (0) is not printed, it is only a **marker**.

msg2

`'$'` terminates it: INT 21h / AH = 09h prints up to the `$`.

Printer Program: The Print Loop

```
▶ start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

The output panel shows the printer paper: LF moves down a line, CR returns to the line start, TAB moves to the next 8-column stop.

Step 1 / 33

VARIABLES

CX = 0000h DX = 0000h SI = 0000h

OUTPUT

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

DL = 12 (0Ch): form feed.

Step 2 / 33

VARIABLES

CX = 0000h | DX = 000Ch | SI = 0000h

OUTPUT

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

AH = 05h: DOS printer output.

Step 3 / 33

VARIABLES

CX = 0000h | DX = 000Ch | SI = 0000h

OUTPUT

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

INT 21h: the printer starts a new page.

Step 4 / 33

VARIABLES

CX = 0000h | DX = 000Ch | SI = 0000h

OUTPUT

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
▶  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

SI = 0102h: address of msg.

Step 5 / 33

VARIABLES

CX = 0000h | DX = 000Ch | SI = 0102h

OUTPUT

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
▶  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

CX = 59 (003Bh): number of characters.

Step 6 / 33

VARIABLES

CX = 003Bh | DX = 000Ch | SI = 0102h

OUTPUT

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
▶  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

DL ← [SI] = 'H'.

Step 7 / 33

VARIABLES

CX = 003Bh | DX = 0048h | SI = 0102h

OUTPUT

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
▶  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

AH = 05h: DOS printer output.

Step 8 / 33

VARIABLES

CX = **003Bh** DX = **0048h** SI = **0102h**

OUTPUT

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
▶  int 21h
11 inc si           ; next character
12 loop print
```

INT 21h: 'H' goes to the printer.

Step 9 / 33

VARIABLES

CX = **003Bh** DX = **0048h** SI = **0102h**

OUTPUT

```
H
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
 1 start:
 2 mov dl, 12       ; form feed: new page
 3 mov ah, 5
 4 int 21h
 5 mov si, offset msg
 6 mov cx, offset msg_end - offset msg
 7 print:
 8 mov dl, [si]
 9 mov ah, 5        ; DOS printer function
10 int 21h
▶ inc si            ; next character
12 loop print
```

SI = 0103h.

Step 10 / 33

VARIABLES

CX = 003Bh | DX = 0048h | SI = 0103h

OUTPUT

```
H
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
   loop print
```

LOOP: CX = 58 → continue.

Step 11 / 33

VARIABLES

CX = 003Ah | DX = 0048h | SI = 0103h

OUTPUT

```
H
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
▶  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

DL ← [SI] = 'e'.

Step 12 / 33

VARIABLES

CX = 003Ah | DX = 0065h | SI = 0103h

OUTPUT

```
H
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
▶  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

AH = 05h: DOS printer output.

Step 13 / 33

VARIABLES

CX = **003Ah** DX = **0065h** SI = **0103h**

OUTPUT

```
H
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
▶  int 21h
11 inc si           ; next character
12 loop print
```

INT 21h: 'e' goes to the printer.

Step 14 / 33

VARIABLES

CX = 003Ah | DX = 0065h | SI = 0103h

OUTPUT

```
He
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
 1 start:
 2 mov dl, 12       ; form feed: new page
 3 mov ah, 5
 4 int 21h
 5 mov si, offset msg
 6 mov cx, offset msg_end - offset msg
 7 print:
 8 mov dl, [si]
 9 mov ah, 5        ; DOS printer function
10 int 21h
▶ inc si            ; next character
12 loop print
```

SI = 0104h.

Step 15 / 33

VARIABLES

CX = **003Ah** DX = **0065h** SI = **0104h**

OUTPUT

```
He
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
   loop print
```

LOOP: CX = 57 → continue.

Step 16 / 33

VARIABLES

CX = 0039h | DX = 0065h | SI = 0104h

OUTPUT

```
He
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
▶  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

DL ← [SI] = 'l'.

Step 17 / 33

VARIABLES

CX = 0039h | DX = 006Ch | SI = 0104h

OUTPUT

```
He
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
▶  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

AH = 05h: DOS printer output.

Step 18 / 33

VARIABLES

CX = **0039h** DX = **006Ch** SI = **0104h**

OUTPUT

```
He
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
▶  int 21h
11 inc si           ; next character
12 loop print
```

INT 21h: 'l' goes to the printer.

Step 19 / 33

VARIABLES

CX = 0039h | DX = 006Ch | SI = 0104h

OUTPUT

```
Hel
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
▶  inc si           ; next character
12 loop print
```

SI = 0105h.

Step 20 / 33

VARIABLES

CX = **0039h** DX = **006Ch** SI = **0105h**

OUTPUT

```
Hel
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
   loop print
```

LOOP: CX = 56 → continue.

Step 21 / 33

VARIABLES

CX = 0038h | DX = 006Ch | SI = 0105h

OUTPUT

```
Hel
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
▶  int 21h
11 inc si           ; next character
12 loop print
```

... INT 21h: 'l' printed (CX = 56).

Step 22 / 33

VARIABLES

CX = **0038h** DX = **006Ch** SI = **0105h**

OUTPUT

```
Hell
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
▶  int 21h
11 inc si           ; next character
12 loop print
```

... INT 21h: 't' printed (CX = 48).

Step 23 / 33

VARIABLES

CX = 0030h | DX = 0074h | SI = 010Dh

OUTPUT

```
Hello, Print
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
▶  int 21h
11 inc si           ; next character
12 loop print
```

... INT 21h: '*' printed (CX = 40).

Step 24 / 33

VARIABLES

CX = 0028h DX = 002Ah SI = 0115h

OUTPUT

```
Hello, Printer!
***
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
▶  int 21h
11 inc si           ; next character
12 loop print
```

... INT 21h: '*' printed (CX = 32).

Step 25 / 33

VARIABLES

CX = 0020h DX = 002Ah SI = 011Dh

OUTPUT

```
Hello, Printer!
**********
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
▶  int 21h
11 inc si           ; next character
12 loop print
```

... INT 21h: 'a' printed (CX = 24).

Step 26 / 33

VARIABLES

CX = 0018h DX = 0061h SI = 0125h

OUTPUT

```
Hello, Printer!
********Ha*****
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
▶  int 21h
11 inc si           ; next character
12 loop print
```

... INT 21h: 'c' printed (CX = 16).

Step 27 / 33

VARIABLES

CX = 0010h DX = 0063h SI = 012Dh

OUTPUT

```
Hello, Printer!
********Have a nic
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
▶  int 21h
11 inc si           ; next character
12 loop print
```

... INT 21h: 'i' printed (CX = 8).

Step 28 / 33

VARIABLES

CX = 0008h DX = 0069h SI = 0135h

OUTPUT

```
Hello, Printer!
********Have a nice printi
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
▶  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

DL ← [SI] = '!'.

Step 29 / 33

VARIABLES

CX = 0001h DX = 0021h SI = 013Ch

OUTPUT

```
Hello, Printer!
********Have a nice printing day
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
▶  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
12 loop print
```

AH = 05h: DOS printer output.

Step 30 / 33

VARIABLES

CX = **0001h** DX = **0021h** SI = **013Ch**

OUTPUT

```
Hello, Printer!
********Have a nice printing day
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
▶  int 21h
11 inc si           ; next character
12 loop print
```

... INT 21h: '!' printed (CX = 1).

Step 31 / 33

VARIABLES

CX = **0001h** DX = **0021h** SI = **013Ch**

OUTPUT

```
Hello, Printer!
********Have a nice printing day!
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
▶  inc si           ; next character
12 loop print
```

SI = 013Dh.

Step 32 / 33

VARIABLES

CX = **0001h** DX = **0021h** SI = **013Dh**

OUTPUT

```
Hello, Printer!
********Have a nice printing day!
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Printer Program: The Print Loop

```
1  start:
2  mov dl, 12       ; form feed: new page
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS printer function
10 int 21h
11 inc si           ; next character
▶  loop print
```

LOOP: CX = 0 → **all 59 characters done**.

Step 33 / 33

VARIABLES

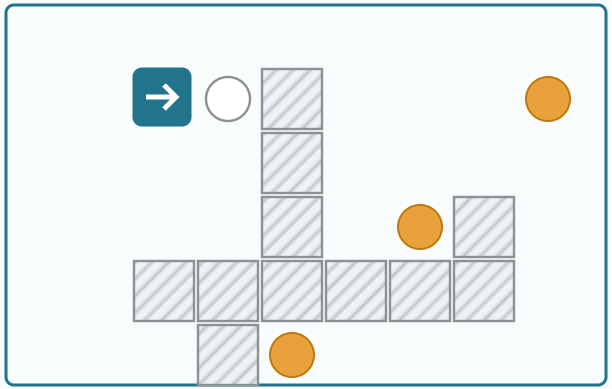
CX = 0000h | DX = 0021h | SI = 013Dh

OUTPUT

```
Hello, Printer!
********Have a nice printing day!
```

CR (13) without LF makes the third line print **over** the stars (shown as overwriting in the panel). After the loop, `mov dx, offset msg2` / `mov ah, 9` / `int 21h` shows a message; `int 16h` (AH = 0) waits for a key; a final form feed ejects the page.

Robot (Ports 9, 10, 11)



```
command (9): 00000001
data (10): 00000000
status (11): 00000000
```

lamp on

lamp off

wall

- The robot moves around a grid world; it can detect walls and lamps and switch lamps on and off.
- Port **9** is the command (OUT), port **10** data (IN), port **11** status (IN). You edit the world with the toolbox in the device window.

Robot Commands and Status Bits

Command (port 9)	Effect
0	do nothing
1	move forward
2	turn left
3	turn right
4	examine what's ahead
5	switch lamp on
6	switch lamp off

Data (port 10)	Ahead
255 (FFh)	wall
0	empty
7	lamp on
8	lamp off

Status (port 11)	If 1
bit 0	new data in the data register
bit 1	robot busy (command running)
bit 2	error in last command

Robot Program: Find the Lamp and Switch It On

```
#start=robot.exe#
main_loop: CALL wait_ready
       MOV  AL, 4        ; examine
       OUT  9, AL
       CALL wait_data
       IN   AL, 10
       CMP  AL, 0FFh     ; wall?
       JE   turn
       CMP  AL, 8        ; lamp off?
       JNE  advance
       CALL wait_ready
       MOV  AL, 5        ; lamp on
       OUT  9, AL
       JMP  main_loop
turn:   MOV  AL, 3        ; turn right
       JMP  send
advance: MOV AL, 1        ; forward
send: PUSH AX          ; keep command
       CALL wait_ready
       POP  AX
       OUT  9, AL
       JMP  main_loop
```

- Before every command, wait until the **busy** bit (bit 1) is zero.
- After an examine (4), wait for the **data ready** bit (bit 0), then read port 10.
- Wall: turn right; unlit lamp: switch it on; otherwise move forward.
- The wait routines are called with `CALL`: the return address is on the stack.

Robot: Wait Routines and Pitfalls

```
wait_ready PROC
b1:     IN   AL, 11
        TEST AL, 010b     ; busy?
        JNZ  b1
        RET
wait_ready ENDP

wait_data PROC
v1:     IN   AL, 11
        TEST AL, 001b     ; new data?
        JZ   v1
        RET
wait_data ENDP
```

Preserve AL

`wait_ready` overwrites AL with IN. On the `send` path AL holds the command, so `PUSH AX` before the call and `POP AX` after; otherwise the status byte would be sent to the port.

Error bit

A “forward” into a wall sets the error bit (bit 2); read and check the status.

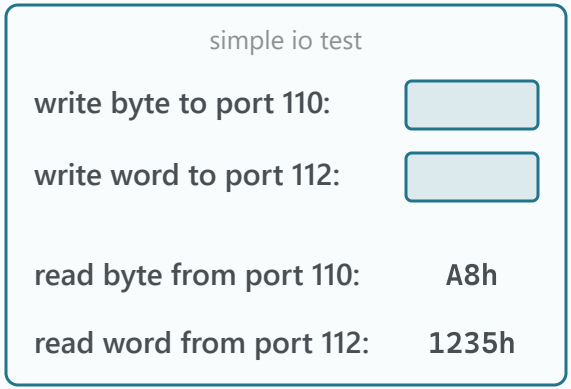
The cost of polling

The CPU is fully tied up in the wait loop; real systems prefer interrupts.

Simple I/O Test Device (Ports 110, 112)

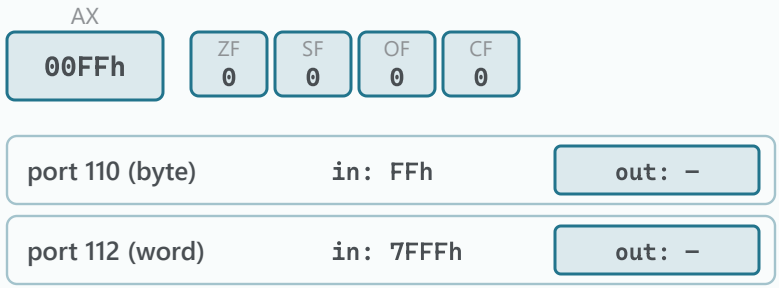
- Port **110**: a **byte**; port **112**: a **word**.
- Values typed into the window become program input (use the `h` suffix for hex); what the program writes shows up in the window.
- Input ports are refreshed every 100 ms.
- Ideal for trying out and debugging IN/OUT.

```
IN   AL, 110    ; e.g. reads A7h
INC  AL
OUT  110, AL    ; writes A8h
IN   AX, 112    ; e.g. 1234h
ADD  AX, 1
OUT  112, AX    ; 1235h
```



Test Device: Byte and Word I/O

```
1 org 100h
▶ in   al, 110      ; read byte
3 inc  al
4 out  110, al      ; write byte
5 in   ax, 112      ; read word
6 add  ax, 1
7 out  112, ax      ; write word
8 ret
```



Values entered in the test device window: port 110 = **FFh**, port 112 = **7FFFh**. IN AL, 110 → AL = FFh.

Step 1 / 7

VARIABLES

`AX = 00FFh` `ZF = 0` `SF = 0` `OF = 0` `CF = 0`

Input values were fed into the interpreter's port map; outputs were read from the I/O log.

Test Device: Byte and Word I/O

```
1 org 100h
2 in   al, 110      ; read byte
▶ inc  al
4 out  110, al      ; write byte
5 in   ax, 112      ; read word
6 add  ax, 1
7 out  112, ax      ; write word
8 ret
```

AX

0000h	ZF	SF	OF	CF
	1	0	0	0

port 110 (byte)	in: FFh	out: –
port 112 (word)	in: 7FFFh	out: –

INC AL: FFh + 1 = **00h** (wraps in 8 bits). ZF = 1; INC **leaves CF unchanged**.

Step 2 / 7

VARIABLES

AX = 0000h · ZF = 1 · SF = 0 · OF = 0 · CF = 0

Input values were fed into the interpreter's port map; outputs were read from the I/O log.

Test Device: Byte and Word I/O

```
1 org 100h
2 in   al, 110      ; read byte
3 inc  al
4 out  110, al      ; write byte
5 in   ax, 112      ; read word
6 add  ax, 1
7 out  112, ax      ; write word
8 ret
```

AX

0000h	ZF	SF	OF	CF
	1	0	0	0

port 110 (byte)	in: FFh	out: 00h
port 112 (word)	in: 7FFFh	out: –

OUT 110, AL: the window shows port 110 = 00h.

Step 3 / 7

VARIABLES

AX = 0000h · ZF = 1 · SF = 0 · OF = 0 · CF = 0

Input values were fed into the interpreter's port map; outputs were read from the I/O log.

Test Device: Byte and Word I/O

```
1 org 100h
2 in   al, 110      ; read byte
3 inc  al
4 out  110, al      ; write byte
▶ in   ax, 112      ; read word
6 add  ax, 1
7 out  112, ax      ; write word
8 ret
```

AX

7FFFh

ZF	SF	OF	CF
1	0	0	0

port 110 (byte)	in: FFh	out: 00h
port 112 (word)	in: 7FFFh	out: –

IN AX, 112: 16-bit read → AX = **7FFFh** (+32,767).

Step 4 / 7

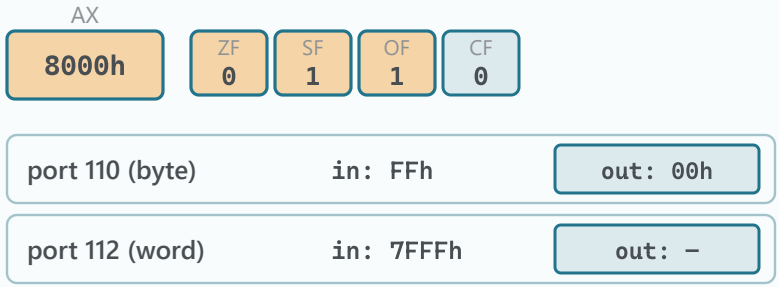
VARIABLES

AX = **7FFFh** ZF = **1** SF = **0** OF = **0** CF = **0**

Input values were fed into the interpreter's port map; outputs were read from the I/O log.

Test Device: Byte and Word I/O

```
1 org 100h
2 in   al, 110      ; read byte
3 inc  al
4 out  110, al      ; write byte
5 in   ax, 112      ; read word
▶ add  ax, 1
7 out  112, ax      ; write word
8 ret
```



ADD AX, 1 → **8000h**: two positives gave a negative sum → OF = 1, SF = 1; no unsigned carry, CF = 0.

Step 5 / 7

VARIABLES

`AX = 8000h` `ZF = 0` `SF = 1` `OF = 1` `CF = 0`

Input values were fed into the interpreter's port map; outputs were read from the I/O log.

Test Device: Byte and Word I/O

```
1 org 100h
2 in   al, 110      ; read byte
3 inc  al
4 out  110, al      ; write byte
5 in   ax, 112      ; read word
6 add  ax, 1
▶ out  112, ax      ; write word
8 ret
```

AX

8000h

ZF	SF	OF	CF
0	1	1	0

port 110 (byte)	in: FFh	out: 00h
port 112 (word)	in: 7FFFh	out: 8000h

OUT 112, AX: the window shows port 112 = 8000h.

Step 6 / 7

VARIABLES

AX = **8000h** ZF = **0** SF = **1** OF = **1** CF = **0**

Input values were fed into the interpreter's port map; outputs were read from the I/O log.

Test Device: Byte and Word I/O

```
1 org 100h
2 in   al, 110      ; read byte
3 inc  al
4 out  110, al      ; write byte
5 in   ax, 112      ; read word
6 add  ax, 1
7 out  112, ax      ; write word
▶ ret
```

AX

8000h

ZF	SF	OF	CF
0	1	1	0

port 110 (byte)	in: FFh	out: 00h
port 112 (word)	in: 7FFFh	out: 8000h

RET: the program ends.

Step 7 / 7

VARIABLES

AX = **8000h** ZF = **0** SF = **1** OF = **1** CF = **0**

Input values were fed into the interpreter's port map; outputs were read from the I/O log.

Lab: Port Devices

Traffic lights (port 4)

CODE · **46 bytes**

```
1  #start=Traffic_Lights.exe#
2  org 100h
3  all_red equ 0000_0010_0100…
4  mov ax, all_red
5  out 4, ax
6  mov si, offset situation
7  next:
8  mov ax, [si]
9  out 4, ax
10 mov cx, 4Ch       ; wait 5 s
11 mov dx, 4B40h
12 mov ah, 86h
13 int 15h
14 add si, 2         ; next st…
```

REGISTERS

AX	0000	AH	00	AL	00
BX	0000	BH	00	BL	00
CX	002E	CH	00	CL	2E
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

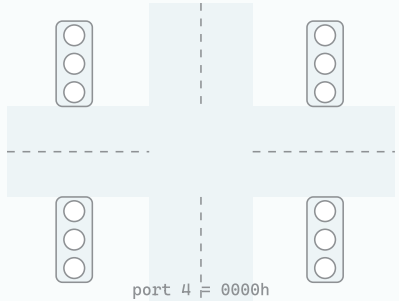
O	D	I	T	S	Z	A	P	C
0	0	1	0	0	0	0	0	0

CS:IP = 0700:0100 · step 0

MEMORY **DS = 0700h**

```
0120 24 01 EB E4 0C 03 9A 06
0128 61 08 61 08 D3 04 00 00
0130 00 00 00 00 00 00 00 00
```

DEVICES



no port access yet

Program loaded: 46 bytes, CS:IP = 0700:0100h. The clock is virtual: in Step and Run modes it advances only through INT 15h waits; in Play mode it runs in real time (scaled by the speed setting). The Step button executes instructions one at a time; the highlighted line is the next instruction.

Summary

Device	Port	Key idea
Traffic lights	4	One bit per lamp; a situation table + INT 15h/86h waits.
Stepper motor	7	Bits 0–2 magnets, bit 7 ready; half step 11.25°, full step 22.5°.
LED display	199	A word, shown as signed decimal.
Thermometer	125, 127	Read + signed compare + heater on/off.
Printer	130	INT 21h/05h; control characters LF, CR, HT, FF.
Robot	9, 10, 11	Command, data, status; poll the busy/data bits.
Test device	110, 112	Byte and word I/O experiments.
8255 PPI (real)	310h–313h	The control word (e.g. 82h) sets the port directions; port > 255 → DX.

The common pattern: read status → test a bit → if ready, write/read data → loop. IN/OUT only through AL/AX, and through DX for ports > 255.

SUMMARY

Exercises

1. What port 4 value lights only the amber lamps in every direction?
2. What should CX and DX be for a 2.5-second wait?
3. Which hex value in AX shows −1 on the LED display?
4. Change the thermostat to a 20–25 °C range. What would happen at −10 °C if you used `JB` ?
5. Which lines of the stepper program must change to turn the motor only clockwise, in full steps?

▶ **Answers**

Exercises (continued)

6. Write the shortest program that prints “Hello” on the printer and ejects the page.
7. On an 8 MHz 8086, what N gives about 1 ms with `mov cx, N` / `loop`?
8. What 8255 control word makes A input, B output and all of C input (mode 0)?
9. On a common-anode display, what are the codes for the digit '5' and the letter 'H' (order dp g f e d c b a)?

▶ **Answers**

References

- emu8086 help: *I/O ports and hardware interrupts*, *Virtual devices* and the sample programs (traffic_lights.asm, stepper_motor.asm, led_display_test.asm, thermometer.asm, printer.asm, robot.asm).
- Intel, *The 8086 Family User's Manual*, 1979 — IN/OUT, the I/O address space and instruction timings.
- Intel, *8255A Programmable Peripheral Interface* data sheet; Texas Instruments, *ULN2003A* data sheet.
- M. Bodur, *Mikroişlemcilere Giriş: Assembler ile Yazılım ve Arayüz*, EMO, 2014 (in Turkish).
- R. Brown, *Ralf Brown's Interrupt List* — INT 15h/86h, INT 21h/05h, INT 16h/00h.
- B. B. Brey, *The Intel Microprocessors*, 8th ed., Pearson, 2009 — Chapter 11 (basic I/O interface).

The End

Chapter 10.2: Device Control

Previous: Chapter 10.1: 8086 Pins · Next: Chapter 11: Instruction Encoding and String Instructions