

DOSBox + MASM Guide: From Zero to Running Your COA Programs

This guide is for you if you have never used DOSBox and do not know the basic words yet. By the end you will be able to: open DOSBox, type a program in Notepad, assemble it, link it, run it, and fix the most common errors.

Every command in this guide is a command that was actually run in DOSBox while checking your COA programs, using real MASM 6.11 and LINK 5.31. One exception is marked clearly: DEBUG (see Part 7).

Part 0: The basics, in plain words

Read this part once. Everything later uses only these words.

Computer and CPU. The CPU is the brain of the computer. It is a chip that follows instructions, one after another, very fast. Everything your program does is done by the CPU.

Register. A register is a tiny, very fast storage box inside the CPU. It holds one number. The 8086 CPU has four main boxes: AX, BX, CX, DX. AX is the main work box (it is called the accumulator). BX is often used as a helper box. CX is often used as a counter (how many times to repeat). DX is often used to point at a message. There are also SI and DI, which point at items in a list, and SP, which keeps track of a waiting stack.

ALU. The ALU is the calculator part inside the CPU. ADD, SUB, MUL, DIV, AND, OR, XOR, NOT all happen in the ALU. When a program says "add two numbers", the ALU does the adding.

Instruction. An instruction is one small order to the CPU, written in a short word. Example: MOV AX,5 means "put the number 5 in box AX". ADD AX,BX means "add the number in BX to the number in AX".

Program. A program is a list of instructions, written top to bottom. The CPU starts at the top and works down the list.

Memory. Memory is a big notebook where numbers, messages, and programs wait while the CPU works. Every place in the notebook has an address, which is just a number. RAM is the notebook the CPU can read and write quickly.

Assembly language. Assembly language is writing instructions in short English-like words (MOV, ADD, INT) instead of 0s and 1s. Humans can read it; the CPU cannot, not directly.

Assembler, and MASM. An assembler is a translator program. MASM (Microsoft Macro Assembler) is the assembler used in your lab. It reads your words and turns them into the numbers the CPU understands.

DOSBox. DOSBox is a pretend old computer (called DOS) running inside your modern Windows PC. Your lab programs were written for DOS, which modern Windows no longer runs directly, so DOSBox gives them a home. When you open DOSBox you see a black window with a prompt like Z:\>. That prompt is DOSBox waiting for you to type a command.

Segment. A segment is a chunk of memory. Your program has two main chunks: the DATA segment, where your numbers and messages live, and the CODE segment, where your instructions live. The two setup lines at the start of every program (MOV AX,@DATA and MOV DS,AX) mean "tell the CPU where the data chunk is", so it can find your numbers.

Interrupt (INT). An interrupt is a way of asking DOS or the screen system to do a standard job for you. INT 21H means "DOS, please do a job". Which job is decided by the number in box AH: AH=09H means "print a message", AH=01H means "read one typed character", AH=4CH means "exit the program". INT 10H means "screen system, please do a job": AH=00H sets the screen mode, AH=0CH lights up one dot

(pixel).

Mount. Mounting is telling DOSBox: "this Windows folder will pretend to be drive C:". After mount c c:\8086, DOSBox's C: drive IS your folder C:\8086, so it can see your files.

The four files you will meet. A .ASM file is your words (what you type in Notepad). An .OBJ file is the translated version, made by MASM, but not yet runnable on its own. An .EXE file is the finished runnable program, made by LINK. An .LST file is a listing: your code with line numbers, plus any error messages. It is your best friend when something goes wrong.

Part 1: Install DOSBox on Windows

1. Go to dosbox.com, open Downloads, and install the Windows version (Next, Next, Finish, as usual).
 2. On your C: drive, make a folder named 8086. So you have C:\8086.
 3. Put masm.exe and link.exe inside C:\8086. In the college lab these two files are usually already in the folder. If they are missing on your PC, ask your teacher for the MASM folder. Do not download random masm.exe files from the internet.
 4. Note: I checked your programs in DOSBox on a Linux machine, so the folder path there was different, but every command below is exactly the same on Windows DOSBox.
-

Part 2: Write the program in Notepad

1. Open Notepad (Start menu, type Notepad).
 2. Type the program, one instruction per line. Spelling matters: MASM reads exactly what you type.
 3. Save it with File, then Save As. Choose a short file name: at most 8 letters, no spaces. Good names: alu.asm, booth.asm, great.asm, prog.asm. Bad names: my program.asm, alu program.asm.
 4. Very important: in the Save box, change "Save as type" to "All files (*.*)". If you leave it as "Text Documents", Windows secretly saves your file as alu.asm.txt, and MASM will say it cannot find alu.asm. This is the most common beginner trap.
 5. Save the file inside C:\8086.
-

Part 3: Run it in DOSBox (the exact sequence)

Open DOSBox. You see a black window with Z:\>. Type each line, pressing Enter after each one:

```
mount c c:\8086
c:
masm alu.asm;
link alu.obj;
alu.exe
```

What each line does:

- mount c c:\8086 - your folder becomes DOSBox's C: drive. - c: - move to the C: drive, so DOSBox looks in your folder. - masm alu.asm; - MASM translates your words into alu.obj. The ; at the end means "use the default answers, do not ask me questions". Wait for it to finish. If it prints error lines, stop here and fix them (see Part 8). If it finishes with no error lines, you have alu.obj. - link alu.obj; - LINK turns alu.obj into alu.exe, the runnable program. The ; again means default answers. - alu.exe - your program runs and prints its output on the screen.

Two useful extras:

- To keep the output so you can read it slowly, run it like this: alu.exe > out.txt and then open out.txt in Notepad. - To get the listing file with line numbers and errors, assemble like this: masm alu.asm,alu.obj,alu.lst; Then open alu.lst in Notepad. Every error line tells you the exact line number to fix. - To leave DOSBox, type exit.
-

Part 4: Exam day - paper first, then PC (read this first)

This is how the performance works in your college:

1. FIRST, you write the experiment on paper. 2. You show the paper to the teacher. Only after the teacher says ok are you allowed to start the PC, type the program in Notepad, and run it in DOSBox.

Yes, it is double work, and yes, it is annoying. The reason is simple: the paper proves you wrote it yourself and did not copy it from a screen.

One relief: the teacher does NOT check every line of your paper code in detail. The paper is your ticket to the PC. It must be good enough to get approved, and a good structure gets approved.

The paper strategy

1. At the top, write the aim in one line, then the algorithm steps in plain words. Example for the greatest-number program: "Take the first number as MAX. Compare each next number with MAX. Keep the bigger one. At the end, print MAX." 2. Then write the shape: the DATA part, the CODE part, the setup lines, the main logic, the print, the exit, and END. The shape is the same for every program (see Part 5). 3. Wherever you forget the exact instruction, write the step in plain words as a comment. A comment is a line starting with ;, and MASM ignores it. Example: ; compare MAX with next number. Never leave a whole section blank. A full structure with plain-words steps gets approved; a blank page does not. 4. If you fix something on the PC later and time allows, correct it on paper too, so paper and PC match.

The PC strategy

1. Type the skeleton first: the shape plus your plain-words comments. Assemble it. It should say 0 errors, because comments cannot break anything. 2. Then turn ONE comment into real code, and assemble again. If it still says 0 errors, do the next comment. 3. If an error appears, it came from the lines you just added. That is a small area to search, not the whole program. 4. Do not type the whole program and assemble only at the very end. One error at a time is easy. Ten errors at once is not.

What to memorize

Memorize the SHAPE of each program (setup, main logic, print, exit), not every single line. All 9 programs below have the same shape. Only the middle (the main logic) changes.

Part 5: Paper skeletons for your 9 important programs

Each skeleton below is the paper version: aim, DATA, setup, plain-words steps as comments, print, exit, END. Every one of these skeletons was assembled in DOSBox and gave 0 errors, so you can type one and check that it works before you fill in the steps.

The first line, .186, is your safety line. It switches MASM to the slightly newer 80186 instruction set. Without it, five of your programs fail with error A2070 (see Part 6). Keep it as the very first line.

1. ALU operations - what it does

This program makes the CPU do maths and logic on two numbers: it adds them, subtracts them, multiplies, divides, and does AND, OR, XOR, NOT, and prints each answer.

Say this in viva: "This program performs ALU operations on two numbers - addition, subtraction, multiplication, division, AND, OR, XOR and NOT - and displays every result."

```
; ALU (SKELETON)
.186
.MODEL SMALL
.STACK 100H
.DATA
```

```

M1 DB 0DH,0AH,'ADD = $'
M2 DB 0DH,0AH,'SUB = $'
M3 DB 0DH,0AH,'MUL = $'
M4 DB 0DH,0AH,'DIV = $'
M5 DB 0DH,0AH,'AND = $'
M6 DB 0DH,0AH,'OR = $'
M7 DB 0DH,0AH,'XOR = $'
M8 DB 0DH,0AH,'NOT = $'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    ; STEP 1: ADD the two numbers, print the answer in hex
    ; STEP 2: SUB the two numbers, print the answer in hex
    ; STEP 3: MUL the two numbers, print the answer in hex
    ; STEP 4: DIV the two numbers, print quotient and remainder
    ; STEP 5: AND, OR, XOR the two numbers, print each answer
    ; STEP 6: NOT one number, print the answer
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN

```

2. Booth's multiplication - what it does

This program multiplies two numbers using Booth's method. It looks at the bits of the second number two at a time, adds or subtracts, and shifts, until the answer is ready.

Say this in viva: "This program multiplies two numbers by Booth's algorithm. It checks pairs of bits, adds or subtracts the first number, shifts, and repeats 8 times to get the product."

```

; BOOTH (SKELETON)
.186
.MODEL SMALL
.STACK 100H
.DATA
    M DB 0FBH
    Q DB 03H
    AREG DB 0
    QREG DB 0
    QMINUS1 DB 0
    COUNT DB 8
    NEG_M DB 0
    MSG DB 0DH,0AH,'Booth product (hex): $'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    ; STEP 1: set A = 0, Q = 3, Q-1 = 0, count = 8, NEG_M = -M
    ; STEP 2: repeat 8 times:
    ;     look at (Q0, Q-1): 01 -> A = A + M, 10 -> A = A - M, else nothing
    ;     shift A, Q, Q-1 right together (A keeps its sign, Q shifts with SHR)
    ; STEP 3: print A then Q as hex digits (the 16-bit answer)
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN

```

3. Block transfer and exchange - what it does

This program copies a whole row of numbers from one place in memory to another place, and then swaps two rows of numbers with each other.

Say this in viva: "This program copies a block of numbers from one memory location to another, and then exchanges the contents of two blocks."

```

; BLOCK TRANSFER + EXCHANGE (SKELETON)
.186
.MODEL SMALL
.STACK 100H
.DATA
    BLOCK1 DB 10H,20H,30H,40H,50H,60H,70H,80H,90H,0A0H
    BLOCK2 DB 01H,02H,03H,04H,05H,06H,07H,08H,09H,0AH
    DEST DB 10 DUP(?)
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    ; STEP 1: copy BLOCK1 into DEST (SI = source, DI = destination,
    ;       CX = 10, REP MOVSB)
    ; STEP 2: print DEST to prove the copy worked
    ; STEP 3: exchange BLOCK1 and BLOCK2 number by number
    ; STEP 4: print BLOCK1 and BLOCK2 to prove the swap worked
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN

```

4. Flag register - what it does

This program adds two numbers and then reads the flag register: the tiny switches inside the CPU that record what happened, like carry and zero. Then it prints each switch.

Say this in viva: "This program adds two numbers and displays the flag register. The flags show carry, zero, sign, parity and overflow after the addition."

```

; FLAG REGISTER (SKELETON)
.186
.MODEL SMALL
.STACK 100H
.DATA
    MSG DB 0DH,0AH,'Flags after ADD: $'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    ; STEP 1: ADD the two numbers (example: 25H + 1BH)
    ; STEP 2: PUSHF then POP into BX (BX now holds the flag switches)
    ; STEP 3: test each flag bit one by one
    ;       (CF, PF, AF, ZF, SF, OF, DF, IF, TF)
    ; STEP 4: print each flag as 0 or 1
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN

```

5. BCD addition - what it does

This program asks you to type two two-digit numbers, adds them the way a calculator adds decimal digits (this is called BCD), and prints the sum.

Say this in viva: "This program reads two two-digit numbers from the keyboard, adds them using BCD addition with DAA adjustment, and displays the sum."

```

; BCD ADDITION (SKELETON)
.186
.MODEL SMALL
.STACK 100H
.DATA
    MSG1 DB 'Enter first 2-digit number: $'
    MSG2 DB 0DH,0AH,'Enter second 2-digit number: $'

```

```

    MSG3 DB 0DH,0AH,'Sum = $'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    ; STEP 1: read two typed digits, pack them into BH (tens in high half)
    ; STEP 2: read two typed digits, pack them into BL
    ; STEP 3: ADD BH + BL, then DAA to fix the answer into valid BCD
    ; STEP 4: print hundreds digit if there was a carry, then tens, then ones
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN

```

6. Number conversion - what it does

This program converts numbers between forms: a hex number into BCD digits, typed digits into packed BCD, and BCD back into letters (ASCII) that can be printed.

Say this in viva: "This program converts numbers between hex, BCD and ASCII forms, so the same value can be shown in different ways."

```

; NUMBER CONVERSION (SKELETON)
.186
.MODEL SMALL
.STACK 100H
.DATA
    MSG1 DB 0DH,0AH,'Hex -&gt; BCD: $'
    MSG2 DB 0DH,0AH,'ASCII -&gt; BCD: $'
    MSG3 DB 0DH,0AH,'BCD -&gt; ASCII: $'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    ; STEP 1: hex 0BH -&gt; unpacked BCD: split the two halves, print 1 then 1
    ; STEP 2: ASCII "27" -&gt; packed BCD: subtract 30H from each digit, pack them
    ; STEP 3: BCD 27H -&gt; ASCII: split the halves, add 30H to each, print 2 then 7
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN

```

7. String operations - what it does

This program works on a line of text: it copies the line, compares two lines to see if they match, and searches for one letter inside the line.

Say this in viva: "This program performs string operations: it copies a string with MOVS, compares two strings with CMPS, and searches for a character with SCAS."

```

; STRING OPERATIONS (SKELETON)
.186
.MODEL SMALL
.STACK 100H
.DATA
    SRC DB 'HELLO WORLD'
    DST DB 20 DUP(?)
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    ; STEP 1: copy SRC to DST
    ;          (SI = source, DI = destination, CX = length, REP MOVSB)
    ; STEP 2: compare SRC and DST (REPE CMPSB), print Equal or Not equal
    ; STEP 3: search for 'W' in SRC (SCASB), print where it was found

```

```

MOV AH, 4CH
INT 21H
MAIN ENDP
END MAIN

```

8. Shapes (graphics) - what it does

This program switches the screen to graphics mode and draws a rectangle and a triangle by lighting up dots, one by one.

Plain-words explanation, because this one is new: graphics mode 13H is a drawing mode of the screen (320 dots across, 200 dots down, 256 colours). INT 10H with AH=0CH lights up ONE dot (called a pixel) at position (x, y). One dot is nothing, but a loop that lights up dot after dot in a straight line makes an edge. Four edges make a rectangle. Three edges make a triangle. That is the whole shapes program: set the mode, light dots in lines, wait for a key, go back to text mode.

Say this in viva: "This program sets graphics mode and draws a rectangle and a triangle by plotting pixels with INT 10H, one dot at a time, in straight lines."

```

; SHAPES (SKELETON)
.186
.MODEL SMALL
.STACK 100H
.DATA
    MSG DB 'Press any key to exit...$'
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    ; STEP 1: set graphics mode 13H (320x200, 256 colours) with INT 10H
    ; STEP 2: draw the rectangle: 4 straight lines of dots
    ;         (top, bottom, left, right)
    ; STEP 3: draw the triangle: 3 straight lines of dots joining 3 corners
    ; STEP 4: wait for a key, go back to text mode, print the message
    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN

```

9. Greatest number in a series - what it does

This program looks through a list of numbers, keeps the biggest one seen so far, and prints it at the end. This program is NOT in your journal's 16 experiments, but it IS on the teacher's important list, so learn it too. The full working version is in the Appendix, and it was assembled and run in DOSBox: for the list 12, 45, 7, 89, 33, 67, 90, 21 it prints "Greatest number = 90".

Say this in viva: "This program finds the greatest number in a series. It takes the first number as the maximum, compares every next number with it, keeps the bigger one, and displays the maximum at the end."

```

; GREATEST (SKELETON)
.186
.MODEL SMALL
.STACK 100H
.DATA
    ARR DB 12, 45, 7, 89, 33, 67, 90, 21
    LEN EQU 8
    MSG DB 0DH, 0AH, 'Greatest number = $'
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    ; STEP 1: point SI at the list, take the first number as MAX

```

```

; STEP 2: check each next number, keep the bigger one as MAX
; STEP 3: print MAX
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN

```

Part 6: Your programs, tested for real

Every COA program from your lab set was assembled and run in real MASM 6.11 inside DOSBox before this guide was written. Here is what was found, honestly:

Five programs need the .186 first line, or they fail. ALU, number conversion, BCD addition, block transfer, and Booth all fail with error A2070: invalid instruction operands on shift lines like SHL BH,4. Why, in one line: the old 8086 chip only knows "shift by 1" or "shift by the number in CL", so a shift by 4 in one step needs the slightly newer 80186 setting, which .186 turns on. With .186 added as the very first line, all five assemble cleanly and their outputs were checked correct (for example BCD addition with inputs 12 and 34 prints Sum = 46).

Two programs ran but gave wrong answers, and both are now fixed. These matter more than assembly errors, because the teacher may run the program or ask for the answer:

- Booth's multiplication printed FFFF for -5×3 , but the correct answer is FFF1 (-15). The bug: the Q shift used SAR, and an arithmetic shift keeps Q's old sign bit, which corrupts the answer whenever A's lowest bit is 0. The fix: the Q shift must be SHR, not SAR (the A shift stays SAR). After the fix it prints FFF1. - Restoring division printed Quotient = 1, Remainder = 4 for $23 / 5$, but the correct answer is quotient 4, remainder 3. The bug: the program shifts AX (16 bits) instead of AL (8 bits), so the dividend bits never enter the remainder register. The fix: change SHL AX,1 to SHL AL,1. After the fix it prints 4 and 3.

The rest ran correctly the first time. Hello, string operations, password (correct password gives access, three wrong passwords lock the account), file operations (it really creates a file and reads it back), cursor control, flag register, and the cache program in C.

One experiment is not a program at all. Experiment 16, the 4x4 memory design, is a diagram: 2 address lines go into a 2-to-4 decoder, the decoder picks one of 4 rows, and each row has 4 one-bit chips, so one row gives one 4-bit word. For viva, describe this diagram: "The decoder picks the row, the 4 chips give the word." One note: your journal diagram shows 16 one-bit chips (4 rows of 4). If any notes say "4 chips", they are counting bigger chips instead. In viva, describe YOUR journal diagram.

Part 7: DEBUG basics (looking inside a running program)

DEBUG is a tool that lets you watch your program run, one instruction at a time, and see the numbers inside the CPU boxes.

Honest note first: plain DOSBox does NOT include DEBUG. It is a separate small file called debug.exe. In the college lab it is normally in the same folder as masm.exe. If you type DEBUG and get Illegal command, the file is missing from your folder - ask your teacher for debug.exe. The commands below are the standard lab commands; they could not be run here for exactly this reason.

```
debug alu.exe
```

You get a - prompt. Now these commands:

- -u then Enter - shows your program's instructions in readable form (this is called unassembling). - t then Enter - runs ONE instruction, then stops. Press it again for the next instruction. This is how you find a wrong answer: step through and watch the boxes change. - g then Enter - runs the whole

program to the end. - -d ds:0 then Enter - shows your data (numbers and messages) as stored in memory. - -q then Enter - quits DEBUG and goes back to DOS.

Use DEBUG when the program assembles fine but prints a wrong answer. Step with -t and watch AX, BX, CX, DX after each instruction. The instruction where a box gets a wrong number is where your bug lives.

Part 8: Common errors and fixes

error A2070: invalid instruction operands on a shift line like SHL BH,4. Fix: add .186 as the very first line of the program. Or, if you want pure 8086 code, replace one SHL BH,4 with four lines of SHL BH,1.

Illegal command: alu.exe when you try to run. This means there is no alu.exe, because assembly failed. Fix the assembly errors first, then link, then run. Never try to run a program that did not assemble.

MASM says it cannot find your file. You are probably not on C: (type c:), or you forgot mount, or your file was saved as alu.asm.txt. Re-save from Notepad with "All files (*.*)" (see Part 2).

The program runs but prints the wrong answer. This is a logic mistake, not a typing mistake. Use DEBUG -t to step through it. And check the two famous ones from Part 6: in Booth, the Q shift must be SHR; in division, the shift must be SHL AL,1.

The program runs but nothing appears. Some programs print nothing by design (the speaker program only beeps). The shapes program draws on the graphics screen, not on the text screen. First check the listing file for 0 errors.

The program seems to hang. A program waiting for a keypress will sit there until you press a key. The shapes program waits for a key at the end. Press any key. If it truly hangs, close DOSBox and open it again.

Part 9: Quick command card

Use short file names. For every program, the three commands are the same, only the name changes:

```
mount c c:\8086
c:
masm NAME.asm;
link NAME.obj;
NAME.exe
```

Your 9 important programs:

Program	File name	Special note
ALU operations	alu.asm	needs .186 first line
Booth's multiplication	booth.asm	needs .186 first line, and Q shift must be SHR
Block transfer and exchange	block.asm	needs .186 first line
Flag register	flags.asm	no special note
BCD addition	bcd.asm	needs .186 first line
Number conversion	numconv.asm	needs .186 first line
String operations	strings.asm	no special note
Shapes	shapes.asm	press a key at the end to exit
Greatest number	great.asm	full code in the Appendix

And for the other experiments: hello.asm, password.asm, fileops.asm, speaker.asm, cursor.asm, division.asm (shift must be SHL AL,1), cache.c (this one is C code, compiled with gcc, not MASM).

Appendix: Greatest number, full working version

This is the complete program, assembled and run in DOSBox. For the list 12, 45, 7, 89, 33, 67, 90, 21 it prints Greatest number = 90.

```
; GREATEST.ASM - Greatest number in a series (MASM, 8086)
; Method: take the first number as MAX, compare every next number,
; replace MAX whenever a bigger number comes, loop till the series ends.
.186
.MODEL SMALL
.STACK 100H
.DATA
    ARR DB 12, 45, 7, 89, 33, 67, 90, 21
    LEN EQU 8
    MSG DB 0DH,0AH,'Greatest number = $'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    LEA SI,ARR          ; SI points to the series
    MOV AL,[SI]         ; MAX = first number
    MOV CX,LEN-1        ; numbers still to check
    INC SI              ; move to the second number
CHK:
    CMP AL,[SI]         ; compare MAX with next number
    JAE SKIP           ; MAX already bigger or equal: keep it
    MOV AL,[SI]         ; else this number becomes the new MAX
SKIP:
    INC SI
    LOOP CHK
    ; print the result (MAX is in AL, 0-99, as two digits)
    LEA DX,MSG
    MOV AH,09H
    INT 21H
    XOR AH,AH
    MOV BL,10
    DIV BL              ; AL = tens, AH = ones
    MOV CL,AH           ; save ones
    CMP AL,0
    JE ONES_ONLY
    ADD AL,30H
    MOV DL,AL
    MOV AH,02H
    INT 21H
ONES_ONLY:
    MOV DL,CL
    ADD DL,30H
    MOV AH,02H
    INT 21H
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN
```

Good luck in the viva. Remember the shape, not every line: setup, main logic, print, exit. And on paper, never leave a section blank.