CS253 Architectures II

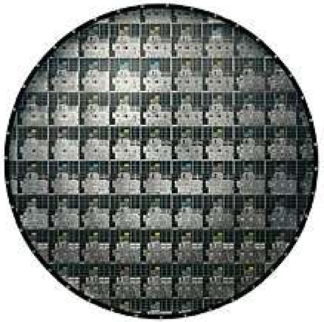
Lecture 2

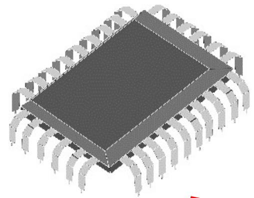
Assembly Language

Charles Markham

CS253 Aims

To go from a bucket of sand to a computer...





```
        mov bx,OFFSET msg1
back:   mov dx,[bx]   ;dl=letters

        cmp dl,'$'
        jz  done

        mov ah,02h
        int 021h

        inc bx
        jmp back

done:   nop
```

General Purpose Registers

The 8086 has eight general purpose registers. As a programmer you can think of them as eight variables. Al is the accumulator and there are more instructions associated with it than any other register. Each register is eight bits wide. However some instructions can uses registers in pairs, giving 16 bit operation.

When used as a register pair they are given the collective name AX, BX, CX, DX.

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

Very similar to the concept of unions in C.

Note registers are memory locations in the processor and are very fast to access.

The Flag Register

The 8086 keeps track of the result of certain calculations in special 16 bit flag register.

15MSB   0LSB

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

U: Undefined

OF: Overflow flag

DF: String direction flag

IF: Interrupt enable flag

TF: Single step trap flag

SF: Sign flag MSB of result

ZF: Zero flag, set if result=0

AF: BCD Carry flag

PF: Parity flag

CF: Carry flag

Conditional Flags: CF, PF, ZF, SF, OF

Control Flags: CF, PF, ZF, SF, OF

Instruction Pointer

The IP (instruction pointer) contains the offset distance (from CS register) to the next code byte that is to be fetched (the line of code that you are on). Code size is limited to 1Mbyte. It is the BIU (bus interface unit) that combine CS and IP to fetch the correct byte from memory.

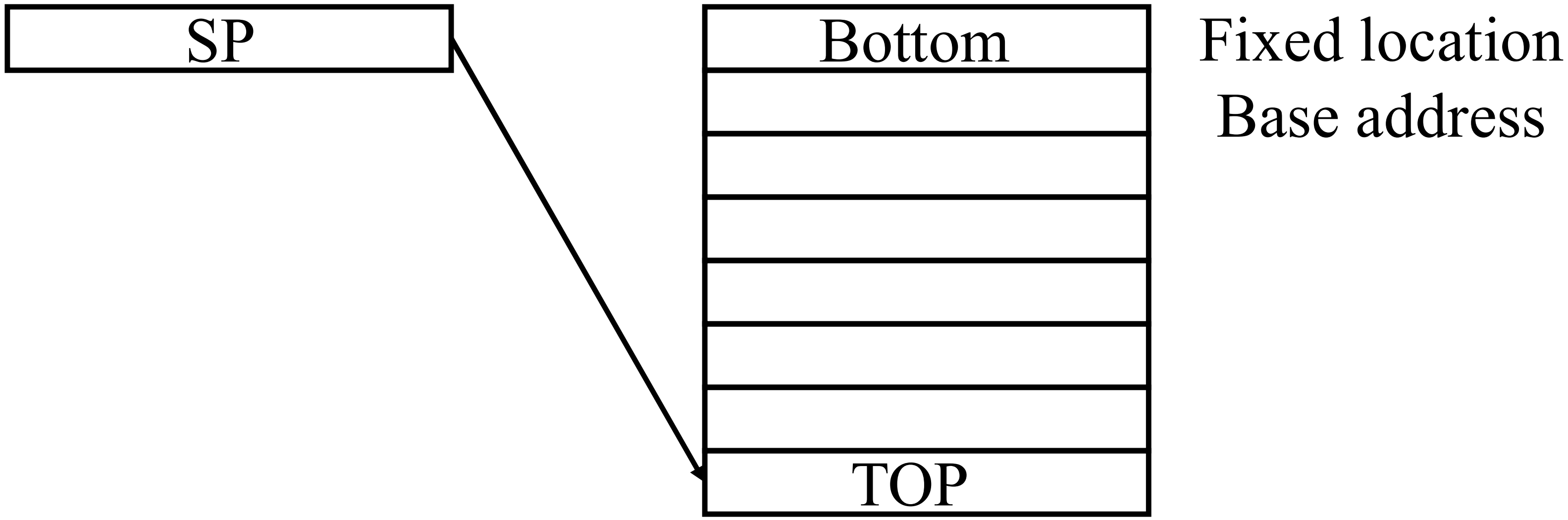
Hardwired zero

CS	3	4	8	A	0
IP+		4	2	1	4
Address in memory	3	8	A	B	4

Sometimes written: CS:IP=348A:4214 or 38AB4

The Stack

Other bits of the instruction can control which register, memory address or stack the ALU output is sent to.



The stack works on a last in first out principle. The ALU output can be directed to the stack.

Segment Registers

The 8086 has a 20 bit address bus, 2^{20}=1,048,576 byte address space.

The internal registers of the 8086 are 16 bit, 2^{16}=65536bytes.

At any moment in time the processor can only access 64K of the memory.

The segment register stores the upper 16bits of the 20bit address.

There are 64K pages of memory each separated by 16bytes.

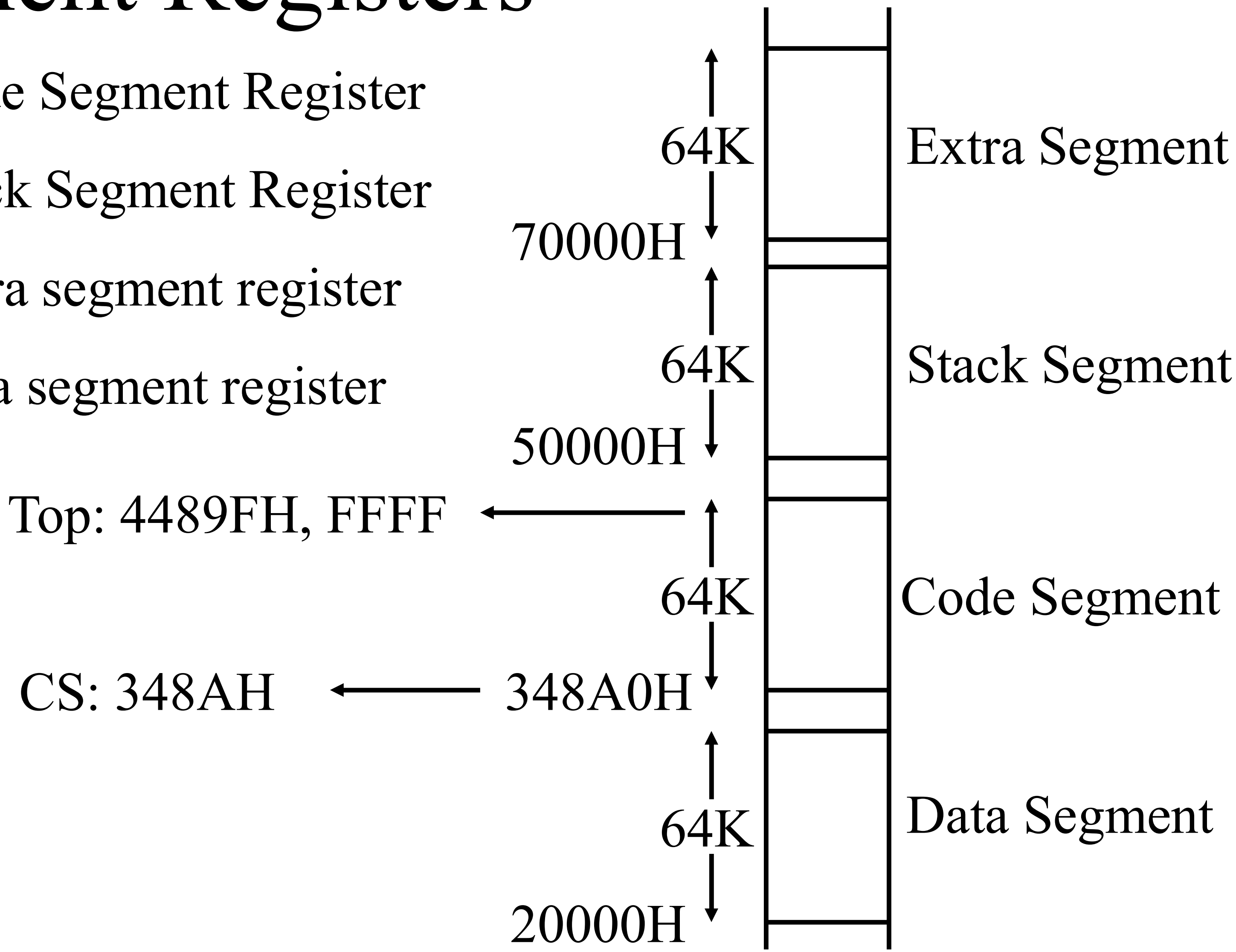
Segment Registers

CS: Code Segment Register

SS: Stack Segment Register

ES: Extra segment register

DS: Data segment register



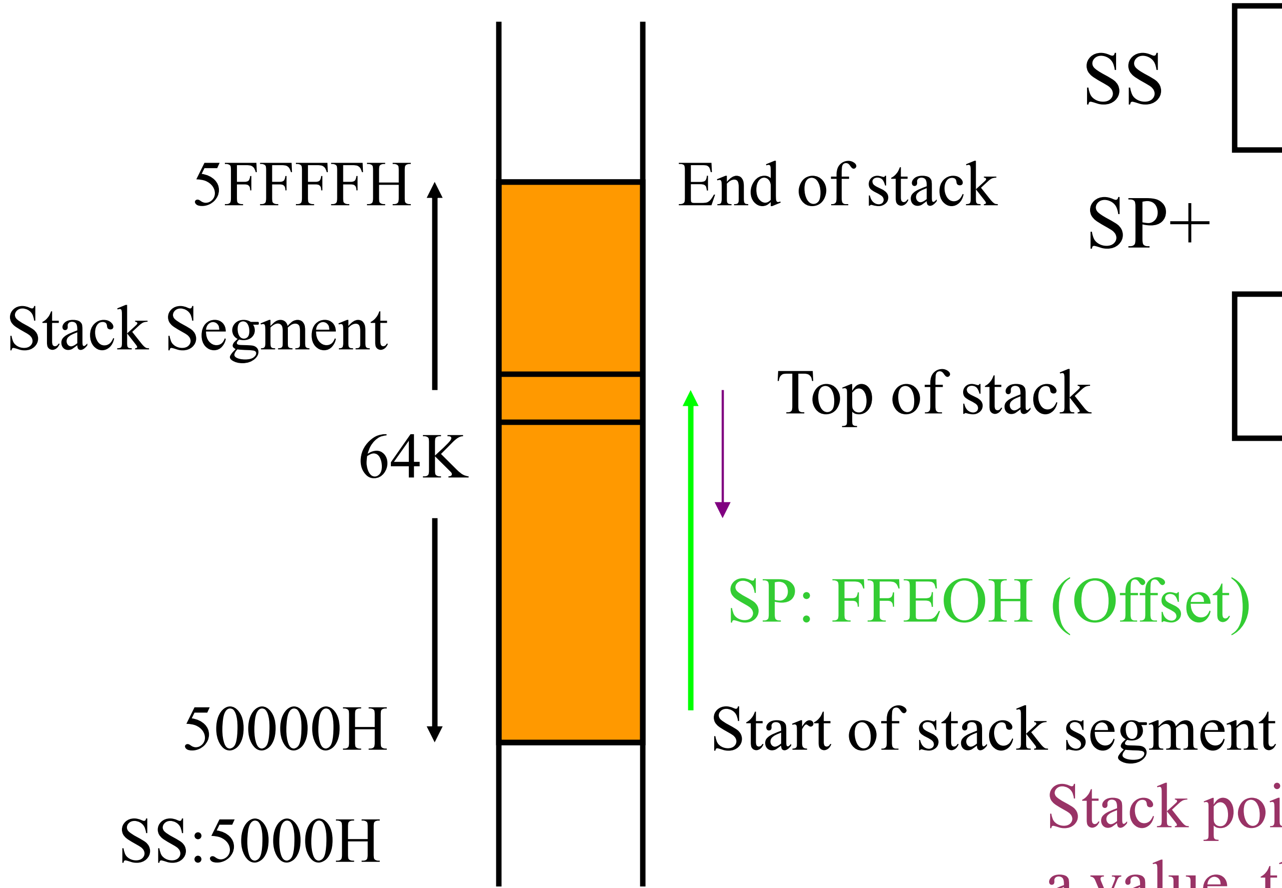
A 64K segment can be located anywhere in the 1Mbyte, the base address of each segment will have the last four bits equal to zero.

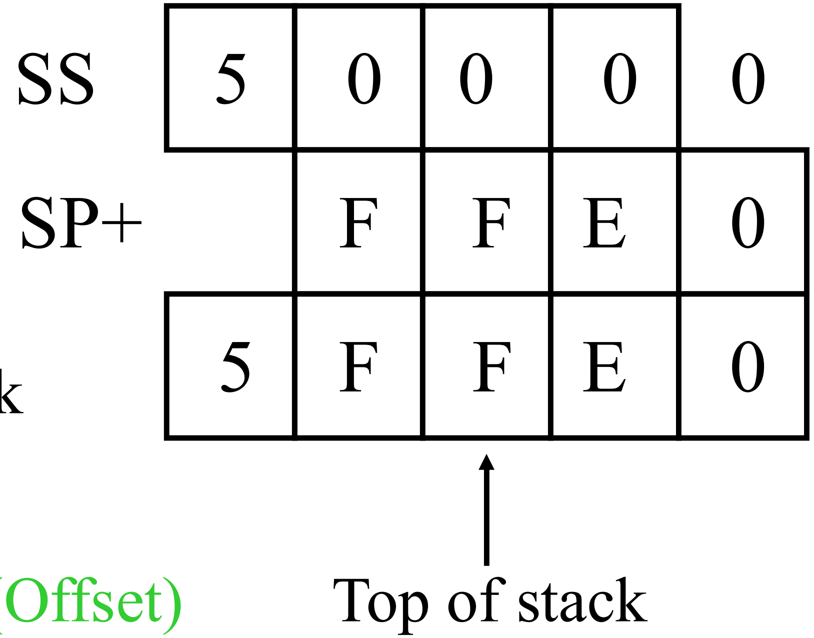
Stack segment register

A stack is a section of memory used to store addresses and data during the execution of a program. The stack segment register SS and the stack pointer work together in the same way as CS:IP. The stack pointer however points to the top of the stack.

The SI (source index), (DI) destination index and (BP) base pointer registers, are general purpose in nature. They are however used as temporary stores of the segment registers.

Stack segment register





SS:SP 5000:FFE0H = 5FFE0

Stack pointer SP is initialised to a value, this value is decreased each time something is put on the stack and increased when it is removed.

Programming Languages

Machine Language: A list of binary codes describing the instructions that are to be executed. This requires hand compiling which is a time consuming task.

Assembly Language: Each binary code can be represented by a mnemonic. Each mnemonic can be read as English but has a direct equivalent binary equivalent. This makes it much easier for the programmer to remember the codes.

High Level languages: Codes have far more functionality. They need conversion to machine language. Interpreter or Compiler.

Why look at Assembly Language?

Assembly language is very instructive for those wishing to understand the operation of hardware. It represents the actual instructions that are being executed by the microprocessor during run-time. High level compilers generate machine language represented by assembly codes.

Well written assembly language programs can be very fast. There is no overhead introduced by a high level compiler or interpreter.

Programs can be written that are very efficient on memory (e.g. Hello world, 20-30bytes).

Summary: Fast, Use small amounts of memory but unsuitable for complex problems.

Anatomy of a MASM Program

```
.MODEL   small
.STACK
.DATA

.CODE
.STARTUP

nop

.EXIT

END
```

Program size, usage

Stack 1K

Program data space

Code starts here

Program starts here at run time.

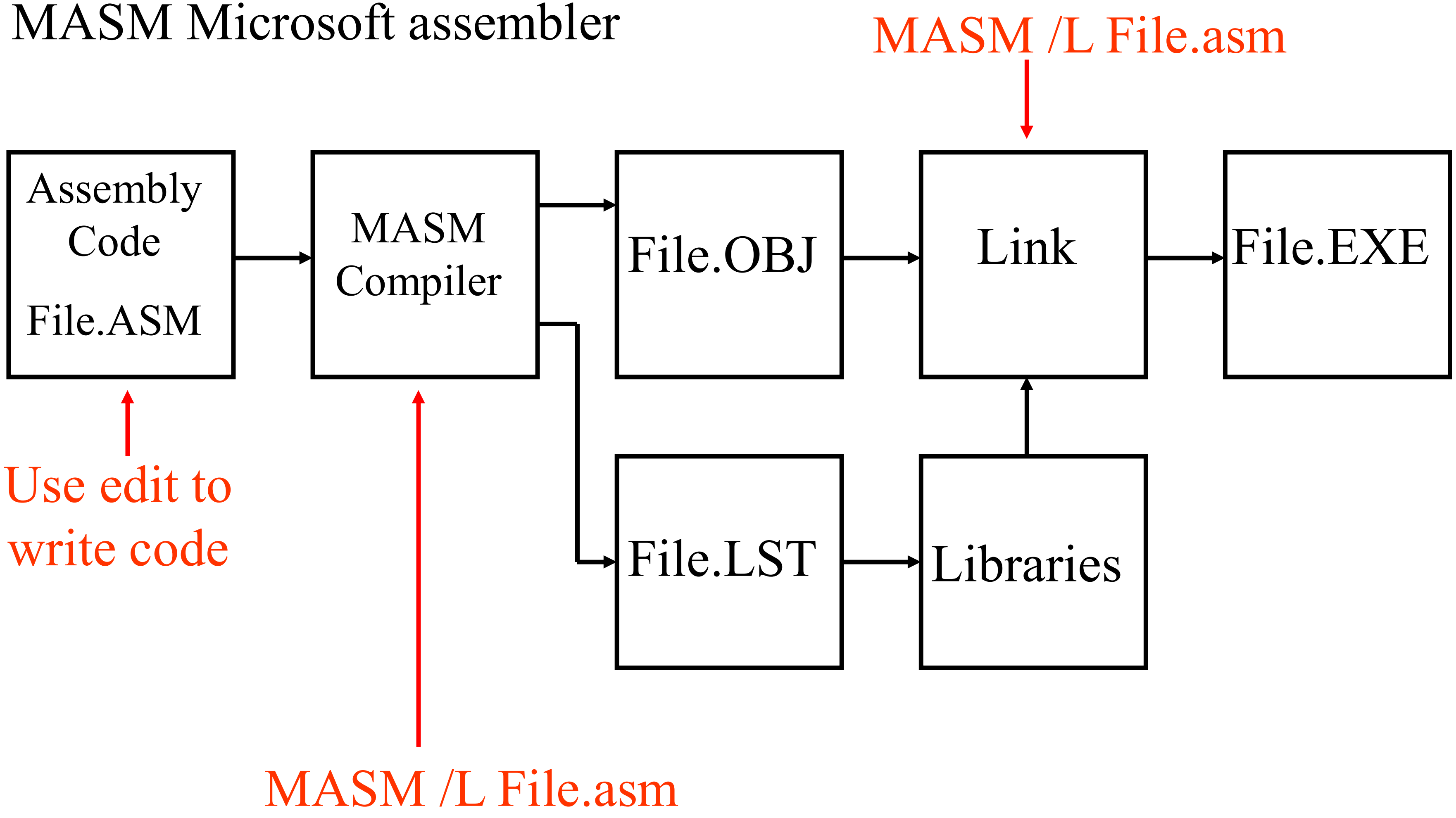
Single instruction, no operation

Program terminates here

End of everything in program.

MASM

MASM Microsoft assembler



Linking combines blocks of code and assigns segments.

After compiling

Offset address		Mnemonics, Assembly language
		.MODEL small
		.STACK
0000		.DATA
0000		.CODE
		.STARTUP
0017	90	nop
		.EXIT
		END

Binary code

Operator and operand

The 8086 processor is in its simplest form a *von Neumann* processor, that is it uses its memory to store the instructions for the program and data for the program.

The commands built into the processor have two parts, the *operator* that specifies the action to be taken and the *operand* which describes the data required to carry out the operation.

Operator

Operand

```
mov ax,568
```

Hello, world

```
                    .MODEL   medium
                    .STACK
                    .DATA

        msg1        BYTE   "Hello, world.$"

                    .CODE
                    .STARTUP

                    mov bx,OFFSET msg1
Label   back:       mov dx,[bx]   ;dl=letters

                    cmp dl,'$'          Compare dl with $
Jump if zero        jz  done

                    mov ah,02h
                    int 021h            Print character in dl

                    inc bx              bx=bx+1
                    jmp back

        done:       nop

                    .EXIT

        END
```

Compiled code

Bytes stored in data segment

Binary code

Offset address

Assembly language listing

```
                                        .MODEL   medium
                                        .STACK
0000                                    .DATA

0000   48 65 6C 6C 6F 2C   msg1         BYTE   "Hello, world.$"
       20 77 6F 72 6C 64
       2E 24

0000                                    .CODE

                                        .STARTUP

0017   BB 0000 R                        mov bx,OFFSET msg1
001A   8B 17               back:        mov dx,[bx]   ;dl=letters

001C   80 FA 24                         cmp dl,'$'
001F   74 07                            jz  done

0021   B4 02                            mov ah,02h
0023   CD 21                            int 021h

0025   43                               inc bx
0026   EB F2                            jmp back

0028   90                  done:        nop

                                        .EXIT

                                        END
```