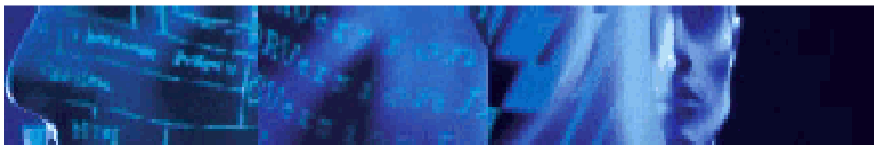
Anti-virus & Virus Technologies

YEAR 1 – SEMESTER 1

Bucharest 2010 - 2020

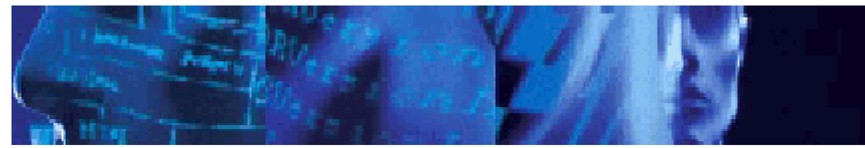
Anti-virus & Virus Technologies



Cristian Toma

IT&C Security Master

Dorobantilor Ave., No. 15-17
010572 Bucharest - Romania
http://ism.ase.ro
cristian.toma@ie.ase.ro
T +40 21 319 19 00 - 310
F +40 21 319 19 00



Catalin Boja

IT&C Security Master

Dorobantilor Ave., No. 15-17
010572 Bucharest - Romania
http://ism.ase.ro
catalin.boja@ie.ase.ro
T +40 21 319 19 00 - 310
F +40 21 319 19 00

YEAR 1 – SEMESTER 1

Bucharest 2010

Organization Form

Didactical Activities: Course 50% + Seminars 50%

Teaching Language: English

Evaluations: Final Exam (60%) + Semester activities (40% - quiz on last Sunday 15%, 15% offline assignments, 10% online activities)

Goals:

- Achieving theoretical concepts and practical principles for anti-viruses and viruses development using ASM x86, C and VBA programming languages
- Understanding the threads in other open systems than PC's operating systems
- Founding the decisions of choosing the antivirus software

AGENDA

Section I – ASM x86

- Intro in ASM x86 program structure
- Internal Representation & Data Types
- Microprocessor 8086 Registers & Real Mode Addressing
- Instructions categories & Addressing Types
- x86 Instructions Encoding
- ASM 8086 Instructions
- Fundamental Programming Control Structures
- Procedures
- Macro-definitions
- Software Interrupts – File & Console Operations in DOS
- String Operations

Section II – Viruses

- Creating COM viruses – overwriting, companion & parasitic in ASM x86
- Creating EXE viruses in ASM x86
- Creating memory resident viruses in ASM x86
- Creating boot viruses in ASM x86
- Creating source code viruses in C/C++
- Macro-viruses: Office – Word, Excel, E-mail
- Principles for building advanced and polymorphic viruses

Section III – Anti-Viruses

- Development Model & Design Principles
- Boot viruses detection and boot antivirus creation in ASM x86.
- File system viruses detection and antivirus creation in ASM x86 and C/C++.
- Memory resident viruses detection and antivirus creation for memory resident viruses
- Anti-hacker methods and techniques; Thinking the architecture of a secure system for
prevention against virus, logical bombs and Trojan horses attacks.
- Anti-virus development sample

References

Section I – ASM x86

- Microsoft MASM – Programmer's Guide, Microsoft Corporation, 1992 – English
- Vasile LUNGU – "Assembly Language Programming for Intel Processors Family", Teora Publishing House, 2005 – English
- http://www.acs.ase.ro – Teaching -> Assembler
- http://ism.ase.ro

Section II – Viruses

- Mark LUDWIG – The Giant Black Book of Computer Viruses, American Eagle Publications, 1995
- Mark LUDWIG - The Little Black Book of Computer Viruses, American Eagle Publications, 1994

Section III – Anti-Virus

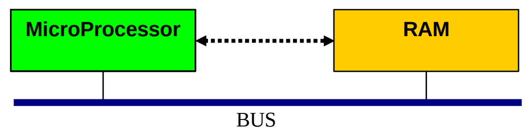
- Mark LUDWIG – The Giant Black Book of Computer Viruses, American Eagle Publications, 1995.

DAY 1

I.1 INTRO IN ASM x86 Program Structure - SAMPLE

```c
#include<stdio.h>

void main()
{
        char a = 7, b = 9;
        short int c;
        c = a+b;
}
```




```asm
.model small
.stack 16
.data
        a db 7
        b db 9
        c dw ?
.code
start:
        mov AX, @data
        mov DS, AX

        xor AX, AX
        mov AL,a
        add AL,b
        mov c,AX

        mov AX, 4C00h
        int 21h
end start
```

ASM x86 program development steps

1. Edit **type** D:\Temp\AVVT\P1ASM\P1.asm

2. Compile: D:\Temp\AVVT\P1ASM\TASM**TASM.exe** P1.asm

3. Link-edit:

D:\Temp\AVVT\ P1ASM\TASM**TLINK.exe** Ex1.obj

D:\Temp\VAVVT\ P1ASM \TASM**TLINK.exe /Tdc** Ex1.obj

4. Debugging:

D:\Temp\VAVVT\ P1ASM \TASM**TD.exe** Ex1.exe

D:\Temp\VAVVT\ P1ASM \TASM**TD.exe** Ex1.com

I.2 Internal Representation & Data Types

a) **Integer Values**
- **Unsigned**
- **Signed**
- **BCD – Binary Code Decimal**
  - **Packed**
  - **Unpacked**

b) **Real Values**
- **Fixed Point Real**
- **Floating Point Real**

c) **Alpha-Numeric – ASCII/ISO-8859-1**

I.2 Internal Representation & Data Types

a) INTEGERS

- **UNSIGNED**

DECIMAL	HEX	BINARY
32	20h	0010 0000

- **SIGNED**

DECIMAL	HEX	BINARY
-32	E0h	1110 0000

- **BCD – Binary Code Decimal**
  - *Packed*

DECIMAL	HEX	BINAR
32	32h	0011 0010

  - *Unpacked*

DECIMAL	HEX	BINARY
32	0302h	0000 0011 0000 0010

I.2 Internal Representation & Data Types

b) REAL VALUES

- **Binary Form**

Integer part
Decimal part
Sign $-1.11101*2^{101}$ Exponent

- **Mathematical Form**

$$(-1)^{S}*M*2^{(E-Bias)}$$

S – Sign Bit – is 0 for positive numbers & 1 for negative;

M – Mantis – is normalized; it has values in range [1.00…0 , 1.11…1]; first 2 formats don't represent the first bit with value 1

E – Exponent – represents the binary form of the exponent; it is used for obtaining back the decimal form of the number

Bias – helps for the negative values representation of the exponent; facilitates the comparison between real value numbers

I.2 Internal Representation & Data Types

b)REAL VALUES – cont.

- **Floating Point Simple Precision - float**

Bias = 127

31	30 ... 23	22 ... 0
Sign	Exponent	Decimal Part

- **Floating Point Double Precision - double**

Bias = 1023

63	62 ... 52	51 ... 32	31 ... 0
Sign	Exponent	Decimal Part	Decimal Part

- **Floating Point Extended Precision – long double**

79	78 ... 64	63	62 ... 32	31 ... 0
Sign	Exponent	Int	Decimal Part	Decimal Part

Bias = 16383

I.2 Internal Representation & Data Types

b)REAL VALUES - SAMPLE

-134,25

Decimal Form

Binary Form

10000110,01

Normalized Binary Form

1.000011001×2^7

$7 + 127 = 134$

Sign Bit = 1

Exponent = 10000110

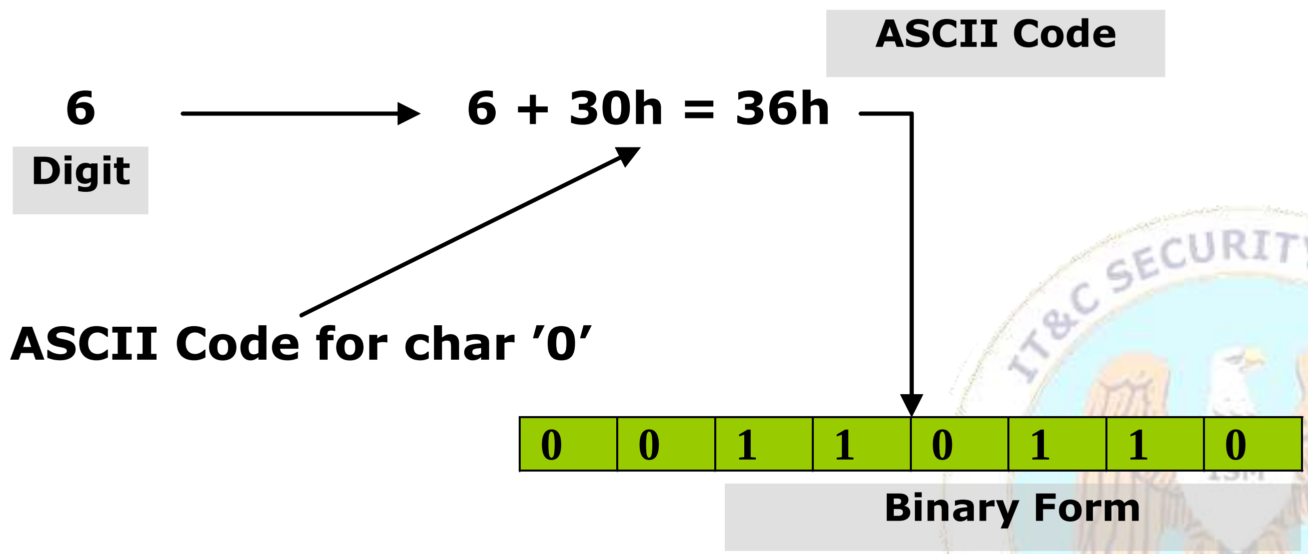
Mantis = 000011001

S	EXPONENT								MANTIS																						
1	1	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I.2 Internal Representation & Data Types

c)Alpha-Numeric – ASCII / ISO-8859-1

- **ASCII**



I.2 Internal Representation & Data Types

a) Byte (1 octet – 8 bits) – DB

b) Word (2 bytes) – DW

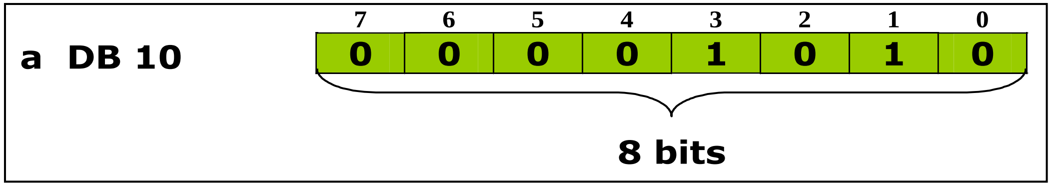
c) Double-Word (4 bytes) – DD

d) Quad-Word (8 bytes) – DQ

e) Ten-Bytes (10 bytes) - DT

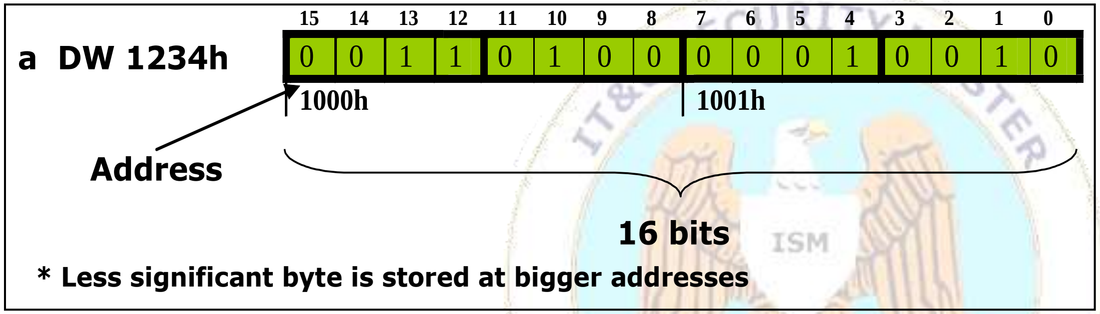
I.2 Internal Representation & Data Types

a) BYTE



- **Memory:** 1 octet – 8 bits
- **Define:** a DB 10
- **Interpretation:**
  - 8 bits Signed or Unsigned Integer
  - ASCII Char

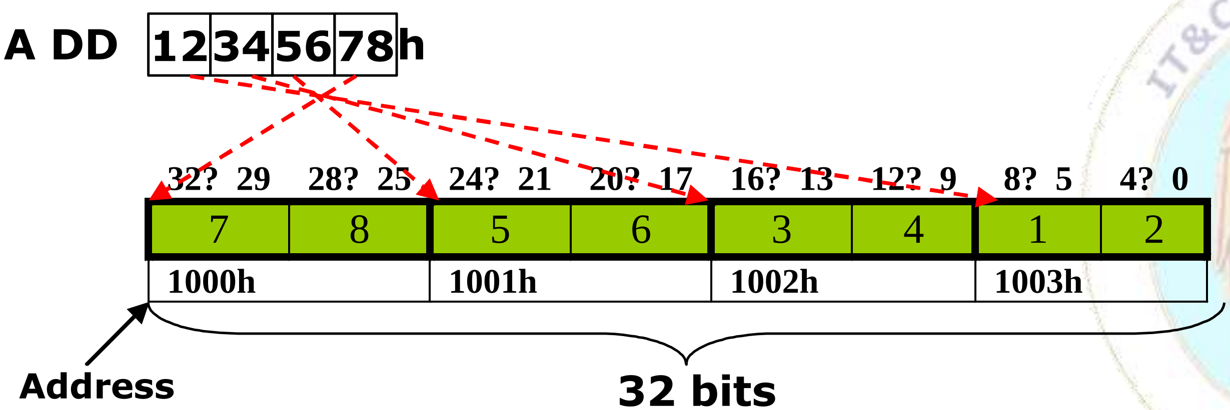
b) WORD



- **Memory:** 2 bytes – 16 bits
- **Define:** a DW 1234h
- **Interpretation:**
  - 16 bits Signed or Unsigned Integer
  - Sequence of 2 ASCII chars
  - Near Pointer – Memory Address stored in 16 bits – ONLY Offset

c)DOUBLE WORD

- **Memory:** 4 bytes
- **Define:** a DD 12345678h
- **Interpretation:**
  - 32 bits Signed or Unsigned Integer
  - Real Number Floating Point Simple Precision
  - Far Pointer – Memory Address stored in 32 bits – Segment : Offset



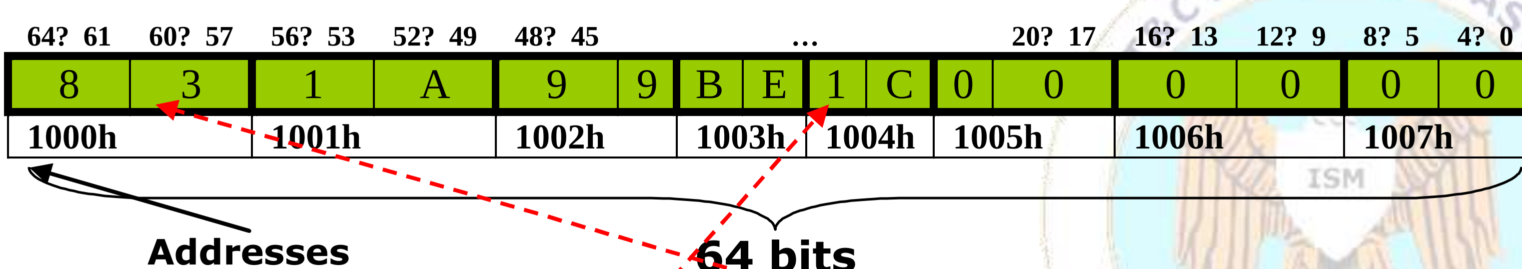
* Less significant bytes are stored at bigger addresses

I.2 Internal Representation & Data Types

d)QUAD-WORD

- **Memory:** 8 bytes
- **Define:** a DQ 123456789123
- **Interpretation:**
    - 64 bits Signed or Unsigned Integer
    - Real Number Floating Point Double Precision

a DQ 123456789123



a **has value in hex $\boxed{1C}$BE991A$\boxed{83}$h**

*** Less significant bytes are stored at bigger addresses**

I.2 Internal Representation & Data Types

e)TEN-BYTES

- **Memory:** 10 bytes
- **Define:** a DT 1547.213
- **Interpretation:**
    - 80 bits Signed or Unsigned Integer
    - Real Number Floating Point Extended Precision
    - Mathematical Coprocessor register value (ST)

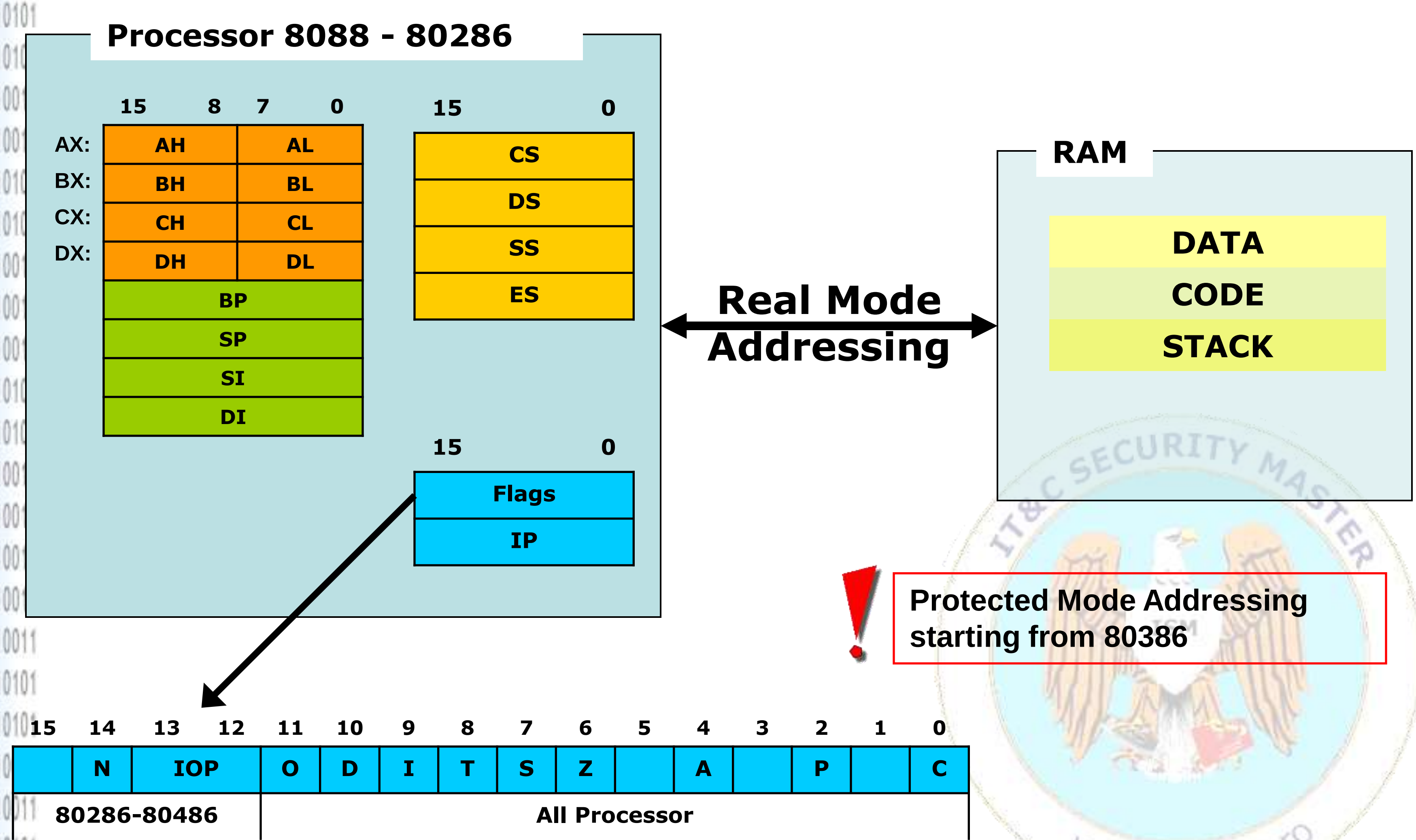
I.2 Internal Representation & Data Types

7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
9h				4h			

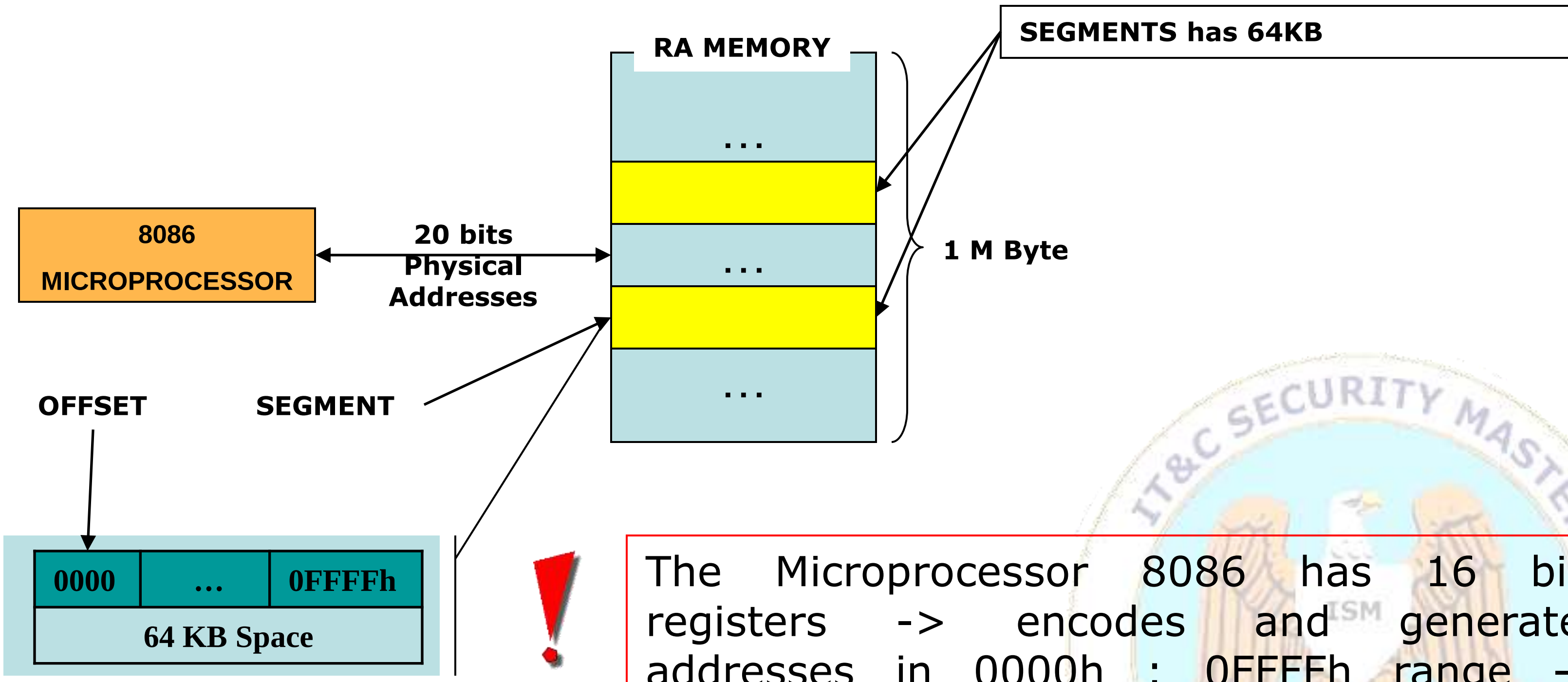
For the byte with binary value: 10010100 an application may interpret as:

- Positive decimal number: 148
- Negative decimal number: -108
- Packed BCD: 94 decimal value
- Char code: 94h – in ISO 8859-1, or ISO 8859-2?

I.3 8086 Microprocessor Registers & Real Mode Addressing



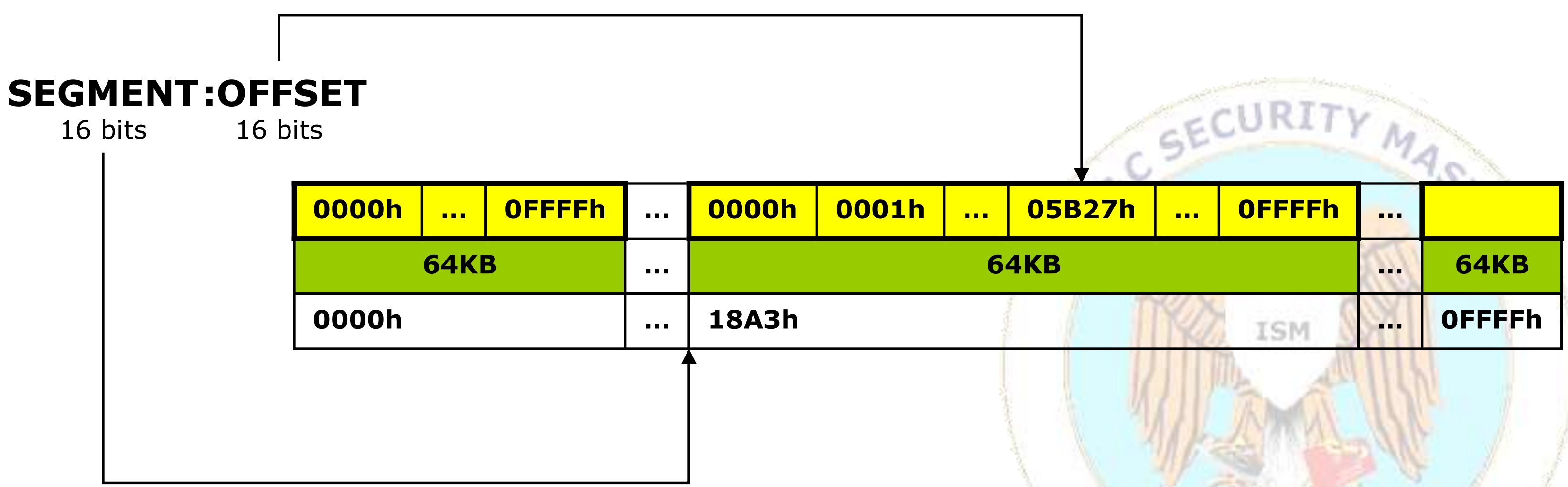
I.3 8086 Microprocessor Registers & Real Mode Addressing



The Microprocessor 8086 has 16 bits registers -> encodes and generates addresses in 0000h : 0FFFFh range -> manages memory segments of 64 KB

I.3 8086 Microprocessor Registers & Real Mode Addressing

For addressing one byte in memory there is necessary 2 items on 16 bits each:
- **Segment Address, stored in a segment register from microprocessor**
- **Offset inside the segment.**



I.3 8086 Microprocessor Registers & Real Mode Addressing

Building the physical address on 20 bits is an automated process of the microprocessor – eventually hardware & firmware:

SEGMENT : OFFSET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	0	1	0	1	0	0	0	1	1
1h				8h				Ah				3h			

← 4 bits of zero

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	0	1	0	1	0	0	0	1	1	0	0	0	0
1h				8h				Ah				3h				0h			

+

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	1	0	0	1	0	0	1	1	1
5h				Bh				2h				7h			

Physical Address

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	1	0	0	1	0	1	0	1	0	1	0	1	1	1
1h				Eh				5h				5h				7h			

I.3 8086 Microprocessor Registers & Real Mode Addressing

BASE REGISTRERS		
AX	**Accumulator Register**	Calculus & Input/Output Operations
BX	**Base Register**	Mostly used as index in various addressing types
CX	**Count Register**	Used in loop instructions
DX	**Data Register**	I/O Operations, Multiply/Divide Operations
INDEXING REGISTERS		
SI	**Source Index**	Mainly used in string operations for the source
DI	**Destination Index**	Mainly used in string operations for the destination

I.3 8086 Microprocessor Registers & Real Mode Addressing

SEGMENT REGISTERS		
CS	**Code Segment**	16 bits value for the segment address that has the active machine binary code of the program
DS	**Data Segment**	16 bits value for the segment address that has active data area for the program
SS	**Stack Segment**	16 bits value for the segment address that has active stack area for the program
ES	**Extra Segment**	16 bits value for the segment address that has active supplementary memory area for the program
INDEX REGISTRERS		
IP	**Instruction Pointer**	16 bits value for the offset within code segment – CS of the next binary code processor instruction
SP	**Stack Pointer**	16 bits value for the offset within stack segment – SS
BP	**Base Pointer**	16 bits value used as offset within the stack segment – SS for the data transfer inside/outside of the program stack

I.3 8086 Microprocessor Registers & Real Mode Addressing

16 bits FLAGS register is used for the control of the instructions execution. The Flags are bits in this register that may have the value 1 (**SET**) or 0 (**NOT SET**).

FLAGS REGISTER		
	Name	**Bit No.**
OF	**Overflow Flag**	**11**
DF	**Direction Flag**	**10**
IF	**Interrupt Flag**	**9**
TF	**Trap Flag**	**8**
SF	**Sign Flag**	**7**
ZF	**Zero Flag**	**6**
AF	**Auxiliary Carry**	**4**
PF	**Parity Flag**	**2**
CF	**Carry Flag**	**0**

I.4 Instructions Categories & Addressing Types

Instructions Categories:

a) Data Transfer
b) Arithmetic & Logic
c) Shift & Rotating
d) Unconditional & Conditional Jump
e) String Manipulation

Addressing Types:

a) Immediate
b) Direct
c) Indirect
d) Based or Indexed
e) Based and Indexed

I.4 Instructions Categories & Addressing Types

a)DATA TRANSFER INSTRUCTIONS

MOV *destination, source*

- copy the value from the source to the destination
- the source and the destination aren't:
  - both memory operands
  - FLAGS or IP register
  - different dimensions in terms of bytes
- the CS register can not be as destination

PUSH *source*

- copy the value to the stack area
- the source has the value on 16 bits
- first the value of 2 is subtracted from the value of the SP register and second the source is copied in memory addressed as SEGMENT:OFFSET – SS:SP

POP *destination*

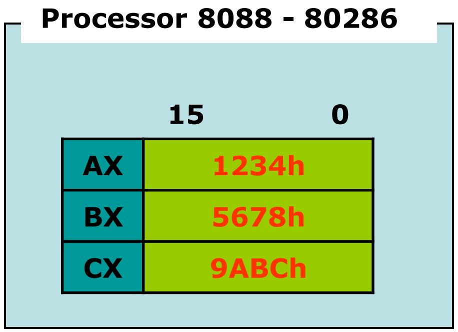
- extract the value from the stack into the destination
- the destination has 16 bits
- first the value from the memory area pointed as SEGMENT:OFFSET – SS:SP is stored in the destination, and second, the value of 2 is added to the value of the SP register

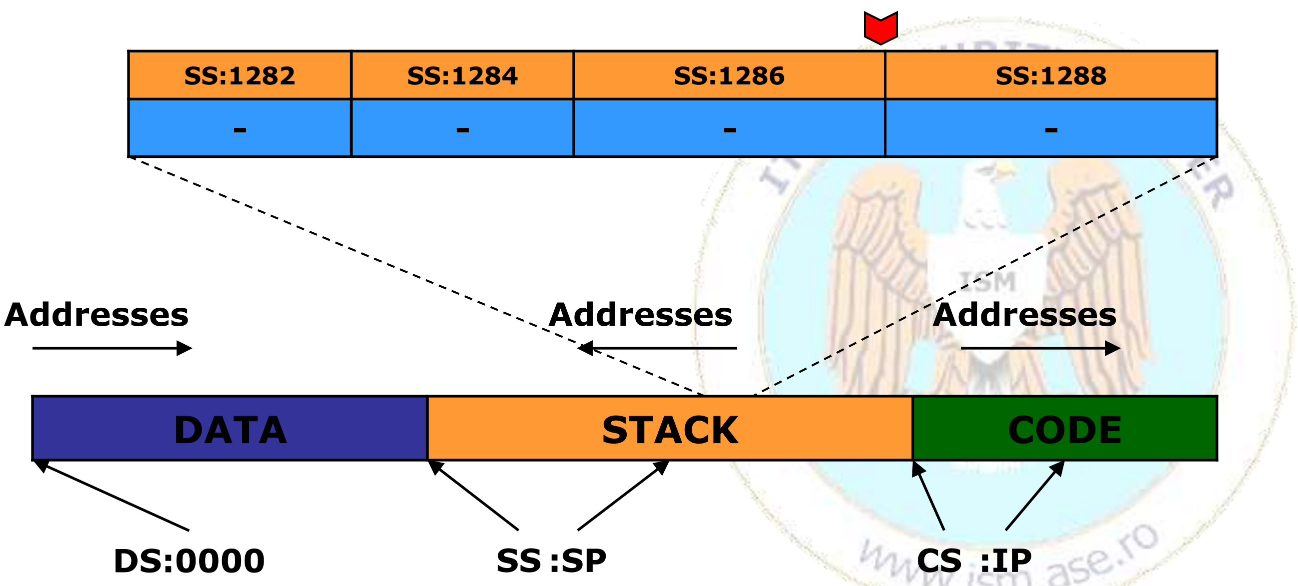
I.4 Instructions Categories & Addressing Types

SAMPLE:

MOV AX, 1234h
MOV BX, 5678h
MOV CX, 9ABCh

PUSH AX	**(I)**
PUSH BX	**(II)**
PUSH CX	**(III)**
POP CX	**(IV)**
POP AX	**(V)**
POP BX	**(VI)**





I.4 Instructions Categories & Addressing Types

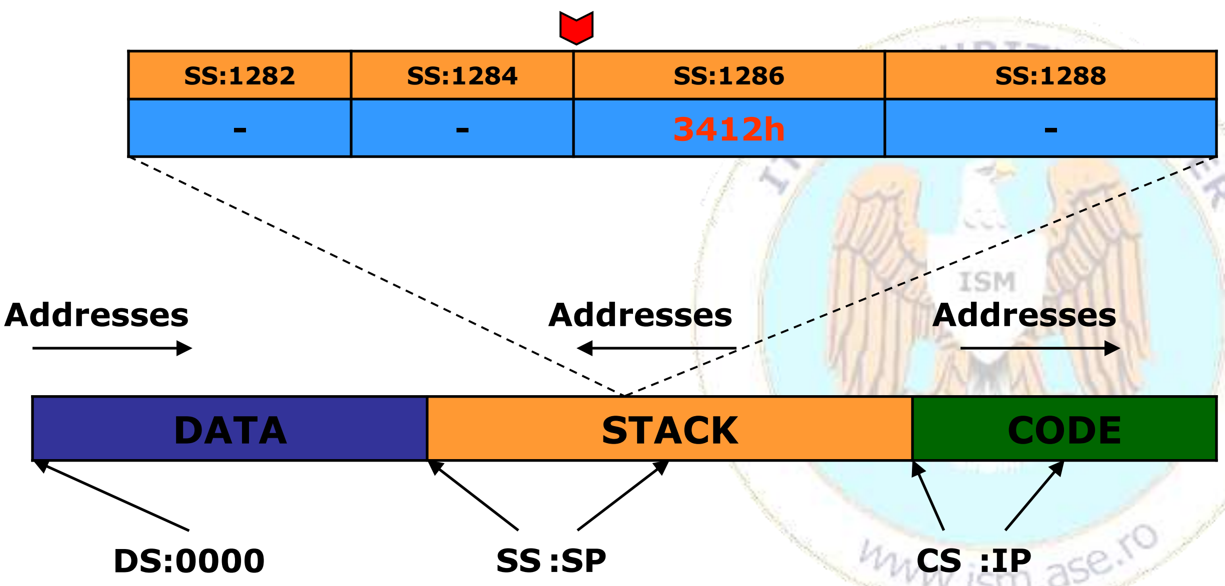
SAMPLE:

MOV AX, 1234h
MOV BX, 5678h
MOV CX, 9ABCh

PUSH AX	**(I)**
PUSH BX	**(II)**
PUSH CX	**(III)**
POP CX	**(IV)**
POP AX	**(V)**
POP BX	**(VI)**

Processor 8088 - 80286

	15 0
AX	1234h
BX	5678h
CX	9ABCh



I.4 Instructions Categories & Addressing Types

SAMPLE:

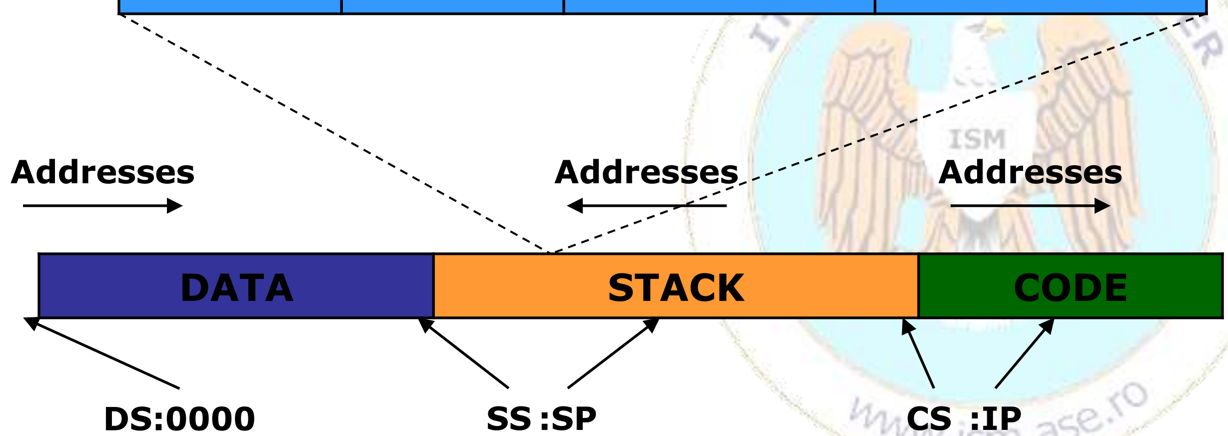
```
MOV AX, 1234h
MOV BX, 5678h
MOV CX, 9ABCh

PUSH AX        (I)
PUSH BX        (II)
PUSH CX        (III)
POP CX         (IV)
POP AX         (V)
POP BX         (VI)
```

Processor 8088 - 80286

	15 … 0
AX	1234h
BX	5678h
CX	9ABCh

SS:1282	SS:1284	SS:1286	SS:1288
BC9Ah	7856h	1234h	-



I.4 Instructions Categories & Addressing Types

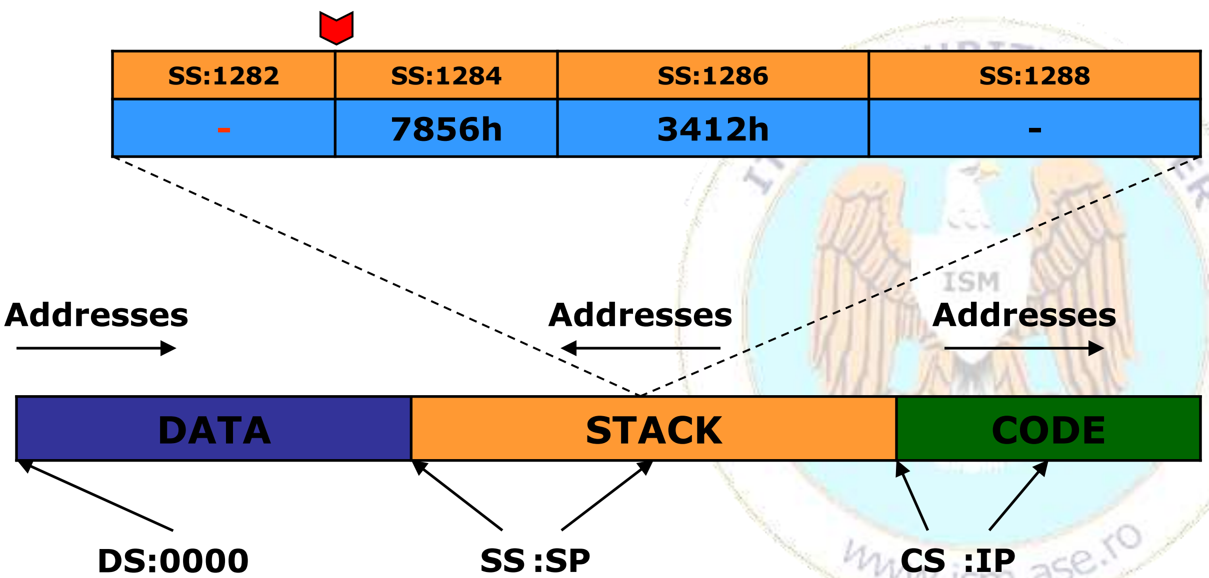
SAMPLE:

```
MOV AX, 1234h
MOV BX, 5678h
MOV CX, 9ABCh

PUSH AX        (I)
PUSH BX        (II)
PUSH CX        (III)
POP CX         (IV)
POP AX         (V)
POP BX         (VI)
```

Processor 8088 - 80286

	15 0
AX	1234h
BX	5678h
CX	9ABCh

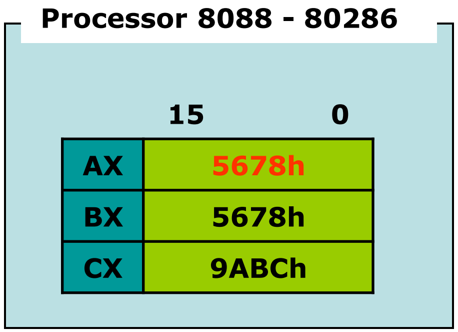


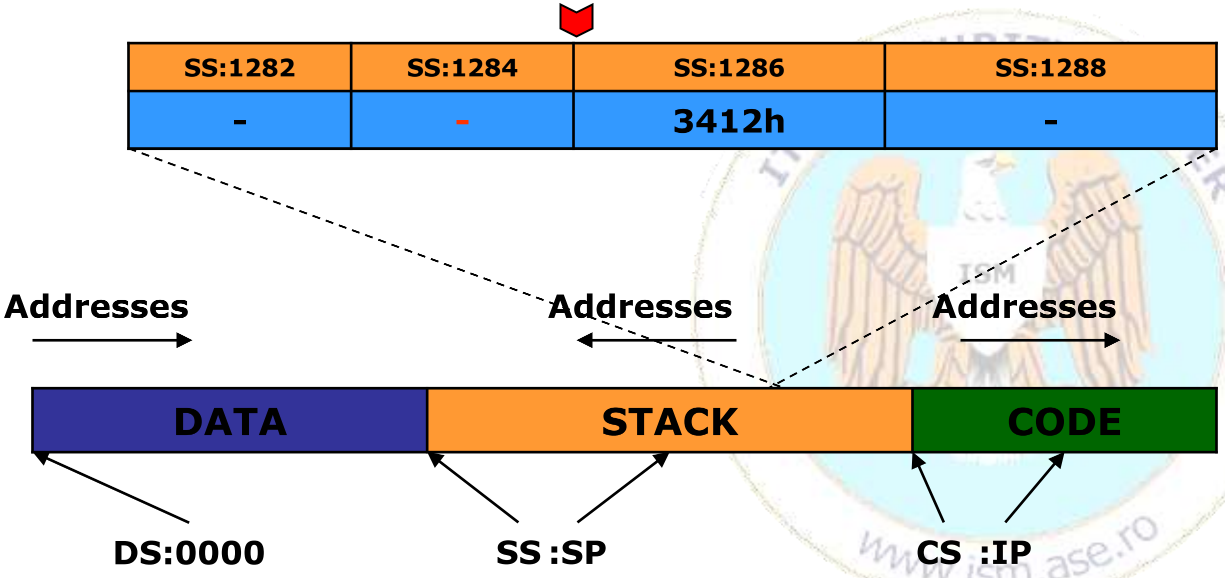
I.4 Instructions Categories & Addressing Types

SAMPLE:

MOV AX, 1234h
MOV BX, 5678h
MOV CX, 9ABCh

PUSH AX	**(I)**
PUSH BX	**(II)**
PUSH CX	**(III)**
POP CX	**(IV)**
POP AX	**(V)**
POP BX	**(VI)**



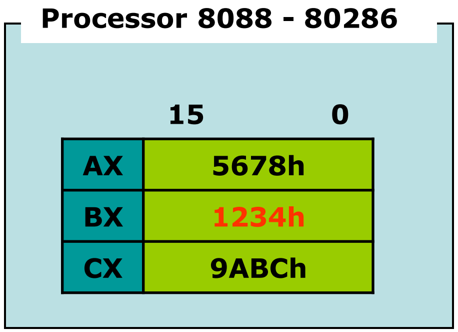


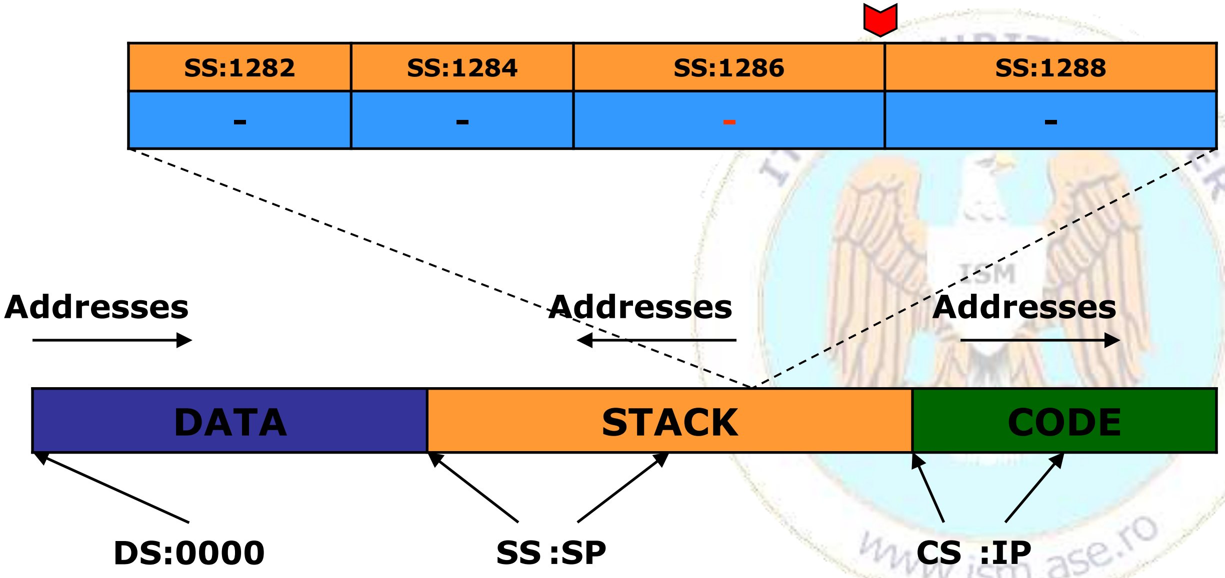
I.4 Instructions Categories & Addressing Types

SAMPLE:

MOV AX, 1234h
MOV BX, 5678h
MOV CX, 9ABCh

PUSH AX	**(I)**
PUSH BX	**(II)**
PUSH CX	**(III)**
POP CX	**(IV)**
POP AX	**(V)**
POP BX	**(VI)**





I.4 Instructions Categories & Addressing Types

a) DATA TRANSFER INSTRUCTIONS

XCHG *destination, source*

- exchange the values between the source & destination;
- the source & destination MUST not be segment registers;
- at least one of the operands MUST be microprocessor register;

IN *accumulator_register*, *source*

- **transfer 1 byte or word value from a port to the AX/AL or AH register;**
- **the source is DX register or 8 bits constant (for the ports less than 256);**

OUT *destination*, *accumulator_register*

- **transfer 1 byte or word value from the AX/AL or AH register to the destination-port;**
- **the destination port is indicated by DX value or a 8 bits constant (for the ports less than 256);**

I.4 Instructions Categories & Addressing Types

a) DATA TRANSFER INSTRUCTIONS

LEA *destination, source*

- **transfer the offset of the source in the destination;**
- **the destination MUST be microprocessor register;**
- **is equivalent to: *MOV destination, offset source*;**

LDS/LES *destination, source*

- **transfer the offset of the source in the destination; the segment address is stored in the assigned segment register from the microprocessor (DS for LDS, ES for LES)**
- **the destination MUST be microprocessor register;**

I.4 Instructions Categories & Addressing Types

a) DATA TRANSFER INSTRUCTIONS

LAHF

- transfers the flags value corresponding to the bits from 0 to 7 into AH register from the microprocessor;

SAHF

- transfers the value of AH register into the flags register corresponding to the bits from 0 to 7;

PUSHF

- puts the entire 16 bits from the flags register to the program stack;

POPF

- gets the 16 bits value from the stack – pointed by SS:SP into the flags register;

I.4 Instructions Categories & Addressing Types

a) IMMEDIATE ADDRESSING

The registers are initialized with constant values

```
MOV AX,1            ;store in AX value 1
```

b) DIRECT ADDRESSING

The name of the variable is used as operand into the instruction

```
vb dw 10
...                 ;store in AX the value of the variable vb
MOV AX,vb           ;in machine code the vb is replaced with the real offset
                    ;from the memory
```

The offset value is used for getting data direct from the data segment.

```
.data
        VB dw 10
.code
        ...
        MOV AX,DS:[0000]        ;store in AX the value 10
```

c) INDIRECT ADDRESSING

Use a base or index register for storing the address from the data segment

```
.data
        VB dw 10
.code
        ...
        MOV SI, offset VB ;init SI with vb address
        MOV AX,[SI]       ;store in AX the value of the variable vb
OR

...
        MOV BX, offset VB; init BX with vb address
        MOV AX,[BX]       ; store in AX the value of the variable vb
```

d) BASED OR INDEXED ADDRESSING

Use a base OR index register plus an offset for accessing memory areas from the data segment taking into account a certain offset regarding a fixed item/landmark

```
.data
        vector dw 10,11,12
.code
        …
        MOV SI,offset vector        ;init SI with the array address
        MOV AX,[SI+2]               ;store in AX the value 11
OR

…
        MOV BX,4                    ;init BX with the value 4
        MOV AX,vector[BX]           ;store in AX the value 12
```

I.4 Instructions Categories & Addressing Types

e) BASED AND INDEXED ADDRESSING

Use a base AND index register plus an offset for accessing memory areas from the data segment taking into account a certain offset regarding a fixed item/landmark

```
.data
        vector dw 10,11,12,13,14
.code
        …
        MOV BX,offset vector        ;init BX with the array address
        MOV SI, 2                   ;init SI with 2
        MOV AX,[BX][SI]             ;store in AX the value 11
OR

…
        MOV SI,4                    ;init SI with 4
        MOV BX,offset vector        ;init BX with the array address
        MOV AX,[BX][SI][2]          ;store in AX the value 13
```

I.4 Instructions Categories & Addressing Types

As best practice in ASM 8086 please take into account that:

- **Use only BX, SI, DI, BP between "[]" for addressing memory**

- **By default:**
  - **BX is related with Data Segment – DS:[BX],**
  - **SI is related with Data Segment – DS:[SI],**
  - **DI is related with Extra Segment – ES:[DI],**
  - **BP is related with Stack Segment – SS:[BP]**
  - **SP is related with Stack Segment – SS:[SP]**

b) ARITHMETIC & LOGIC INSTRUCTIONS

The arithmetic instructions modifies the flags register

- CF = 1 if there is transport/borrow from/into the most significant bit of the result (bit 7 for BYTE OR 15 for WORD – starting with numbering from right to left with 0);
- AF =1 if there is transport/borrow from/into bit 4 of the result (valid for BCD);
- ZF = 1 if the result of the operation is 0;
- SF = 1 if the sign bit of the result is 1 (bit 7 for BYTE OR 15 for WORD – starting with numbering from right to left with 0);
- PF = 1 if the sum modulo 2 of the least significant 8 bits from the result is 0;
- OF = 1 if the result of the operation is too small or too big to be stored in the destination;

If the sum (ADD vs. ADC) between 2 signed BYTE operands/variables is greater than 127 or is less than -127 the microprocessor sets OF – is going to be 1. If the sum is greater than 255 the microprocessor sets CF – is going to be 1.

b) ARITHMETIC & LOGIC INSTRUCTIONS

```
.data
   vb  DB 39

.code
      …
    MOV AL,26

    INC AL

    ADD AL,76

    ADD AL, vb

    MOV AH,AL

    ADD AL,AH
```

RESULT	
Signed Interpretation	Unsigned Interpretation
AL = 26	AL = 26
AL = 27	AL = 27
AL = 103	AL = 103
AL = -114 and OF=1	AL = 142
	AH = 142
	AL = 28 and CF=1

b) ARITHMETIC & LOGIC INSTRUCTIONS

```
.data
vb  DB 122

.code
        …
    MOV AL,95

    DEC AL

    SUB AL,23

    SUB AL, vb

    MOV AH,119

    SUB AL,AH
```

RESULT	
Signed Interpretation	Unsigned Interpretation
AL = 95	AL = 95
AL = 94	AL = 94
AL = 71	AL = 71
AL = -51	AL = 205 and SF=1
AH = 119	
AL = 86 and OF = 1	

b) ARITHMETIC & LOGIC INSTRUCTIONS

ADD *destination, source*

- adds to the destination operand the value of the source operand;

- the addition is with operands on 8 or 16 bits;
- the operands are considered unsigned;

- allows multiple combinations for various types of the source and destination operand (register, immediate value, memory area) BUT excludes the following:
    - the destination operand MUST NOT be CS;
    - the destination and source operands MUST NOT be the memory areas in the same time (e.g. data segment variables);
    - if the source operand is an immediate value, the destination MUST NOT be segment register (CS, DS, ES, SS);
    - the destination MUST NOT be immediate value (constant value);

- if the result is greater than the destination operand capacity then the CF is set – value 1; the value from CF could be taken using ADC instruction (Add with Carry).

I.4 Instructions Categories & Addressing Types

b) ARITHMETIC & LOGIC INSTRUCTIONS

SUB *destination, source*

- subtracts from the destination operand the value of the source operand;

- the subtraction is with operands on 8 or 16 bits;
- the operands are considered unsigned;

- allows multiple combinations for various types of the source and destination operand (register, immediate value, memory area) BUT excludes the following:
  - the destination operand MUST NOT be CS;
  - the destination and source operands MUST NOT be the memory areas in the same time (e.g. data segment variables);
  - if the source operand is an immediate value, the destination MUST NOT be segment register (CS, DS, ES, SS);
  - the destination MUST NOT be immediate value (constant value);

- if the result is greater than the destination operand capacity then the CF is set – value 1; the value from CF could be taken using SBC instruction (Subtract with Carry).

I.4 Instructions Categories & Addressing Types

b) ARITHMETIC & LOGIC INSTRUCTIONS

MUL *operand*

- multiplies an unsigned value from AL register (if the operand type is byte) OR from AX register (if the operand type is word) with the value of the specified operand;
- the result is returned in AX if the operand type is byte and in DX:AX (the most significant bytes in DX) if the operand type is word;
- the flags register is modified: OF & CF are set (with value 1) if the significant part of the result is different by 0 (DX is significant part if the operand type is word, or AH is significant part if the operand type is byte); other flags SF, ZF, AF, PF;
- the operand may be a register or a memory variable;
- if the operand should be considered signed then IMUL instruction is used;

b) ARITHMETIC & LOGIC INSTRUCTIONS

DIV *divider*

- integer division from the source operand to the divider;
- the source operand (dividend) is AX or DX:AX register, depending by divider bits length;
- the divider may be a register or a memory zone – variable from the data segment;
- the operands are considered unsigned
- for signed operands is using **IDIV** instruction

Operands Type	Dividend	Divider	Quotient	Rest
16 bits	AX	8 bits	AL	AH
32 bits	DX:AX	16 bits	AX	DX

CBW
CWD

I.4 Instructions Categories & Addressing Types

b) ARITHMETIC & LOGIC INSTRUCTIONS

NEG *operand*

- used for subtract the operand from the value 0;
- the operand must be register or memory zone – data segment variable;

NOT *operand*

used for inverting the bit values of the operand from 0 to 1 and vice versa;

AND *operand1, operand2*

used for doing logic bitwise AND between the bits of the operand1 and operand2;

OR *operand1, operand2*

used for doing logic bitwise OR between the bits of the operand1 and operand2;

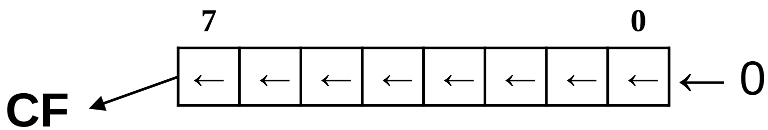
XOR *operand1, operand2*

used for doing logic bitwise XOR-eXclusively OR between the bits of the operand1 and operand2;

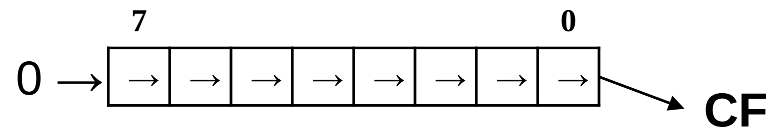
I.4 Instructions Categories & Addressing Types

c)ROTATION AND SHIFTING INSTRUCTIONS

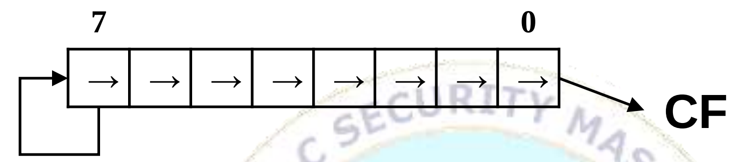
- SHL/SAL – Shift Left/Shift Arithmetic Left



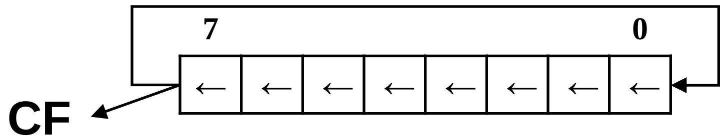
- SHR – Shift Right



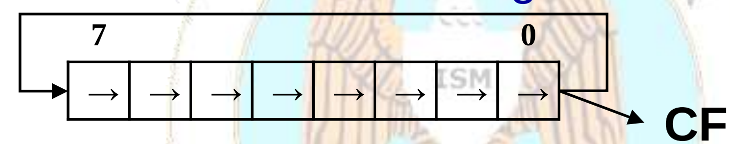
- SAR – Shift Arithmetic Right



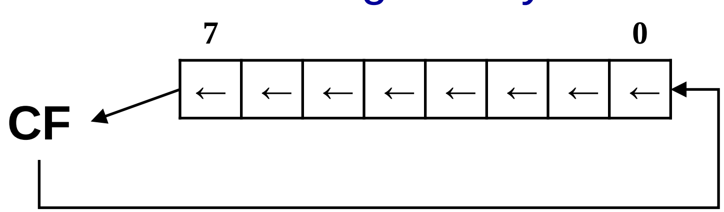
- ROL – Rotate Left



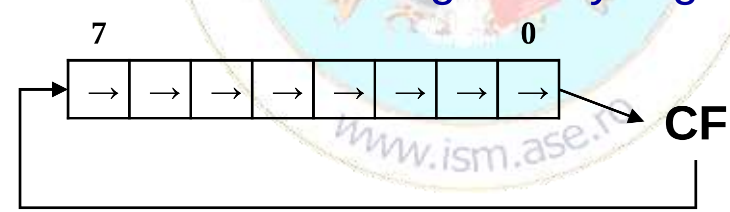
- ROR – Rotate Right



- RCL – Rotate through Carry Left



- RCR – Rotate through Carry Right



I.5 x86 Instructions Encoding

The instruction encoding produces at minim 1 byte & at maximum 6 bytes

1 or 2 bytes						0, 1 or 2 bytes		0, 1 or 2 bytes	
Instruction code	d	w	mod	reg	r/m	octet inferior	octet superior	octet inferior	octet superior
						Offset or direct address		Immediate data	

cod		**are default value**
d	**direction bit**	**if d = 0 the operation direction is memory → register (*reg*)** **if d = 1 the operation direction is *register* → *memory* OR *register* → *register* (*reg is register and r/m is register or memory*)**
w	**word/byte bit**	**if w = 1 the operands type is word** **if w = 0 the operands type is byte**
mod	**mode**	**encodes the addressing mode**
reg	**register**	**identifies the source register used in the instruction**
r/m	**register/memory**	**identifies the second operand as being register or memory area**

*** this is a NOT 100% correct summary of 8086 microprocessor encoding procedure**

I.5 x86 Instructions Encoding

The instruction encoding produces at minim 1 byte & at maximum 6 bytes

1 or 2 bytes						0, 1 or 2 bytes		0, 1 or 2 bytes	
						octet inferior	octet superior	octet inferior	octet superior
Instruction code	d	w	mod	reg	r/m	Offset or direct address		Immediate data	

reg	w =1	w = 0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

	mod				
	00	01	10	11*	
r/m				w =1	w = 0
000	DS:[BX+SI]	DS:[BX+SI+offset$_8$]	DS:[BX+SI+offset$_{16}$]	AX	AL
001	DS:[BX+DI]	DS:[BX+DI+offset$_8$]	DS:[BX+DI+offset$_{16}$]	CX	CL
010	SS:[BP+SI]	SS:[BP+SI+offset$_8$]	SS:[BP+SI+offset$_{16}$]	DX	DL
011	SS:[BP+DI]	SS:[BP+DI+offset$_8$]	SS:[BP+DI+offset$_{16}$]	BX	BL
100	DS:[SI]	DS:[SI+offset$_8$]	DS:[SI+offset$_{16}$]	SP	AH
101	DS:[DI]	DS:[DI+offset$_8$]	DS:[DI+offset$_{16}$]	BP	CH
110	SS:[BP]	SS:[BP+offset$_8$]	SS:[BP+offset$_{16}$]	SI	DH
111	DS:[BX]	DS:[BX+offset$_8$]	DS:[BX+offset$_{16}$]	DI	BH

*** this is a NOT 100% correct summary of 8086 microprocessor encoding procedure**
*** for mod = 11, the instruction has 2 register operands and r/m specifies the source register.**

I.6 ASM 8086 Instructions & Encoding

Mnemonic and Description	Instruction Code			
DATA TRANSFER				
MOV = Move:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/Memory to/from Register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate to Register	1 0 1 1 w reg	data	data if w = 1	
Memory to Accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to Memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/Memory to Segment Register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment Register to Register/Memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		
PUSH = Push:				
Register/Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		
Register	0 1 0 1 0 reg			
Segment Register	0 0 0 reg 1 1 0			
POP = Pop:				
Register/Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		
Register	0 1 0 1 1 reg			
Segment Register	0 0 0 reg 1 1 1			
XCHG = Exchange:				
Register/Memory with Register	1 0 0 0 0 1 1 w	mod reg r/m		
Register with Accumulator	1 0 0 1 0 reg			

I.6 ASM 8086 Instructions & Encoding

IN = **Input from:**		
Fixed Port	1 1 1 0 0 1 0 w	port
Variable Port	1 1 1 0 1 1 0 w	
OUT = **Output to:**		
Fixed Port	1 1 1 0 0 1 1 w	port
Variable Port	1 1 1 0 1 1 1 w	
XLAT = Translate Byte to AL	1 1 0 1 0 1 1 1	
LEA = Load EA to Register	1 0 0 0 1 1 0 1	mod reg r/m
LDS = Load Pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m
LES = Load Pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m
LAHF = Load AH with Flags	1 0 0 1 1 1 1 1	
SAHF = Store AH into Flags	1 0 0 1 1 1 1 0	
PUSHF = Push Flags	1 0 0 1 1 1 0 0	
POPF = Pop Flags	1 0 0 1 1 1 0 1	

I.6 ASM 8086 Instructions & Encoding

Mnemonic and Description	Instruction Code			
ARITHMETIC	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ADD = **Add:**				
Reg./Memory with Register to Either	0 0 0 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s: w = 01
Immediate to Accumulator	0 0 0 0 0 1 0 w	data	data if w = 1	
ADC = **Add with Carry:**				
Reg./Memory with Register to Either	0 0 0 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s: w = 01
Immediate to Accumulator	0 0 0 1 0 1 0 w	data	data if w = 1	
INC = **Increment:**				
Register/Memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m		
Register	0 1 0 0 0 reg			
AAA = ASCII Adjust for Add	0 0 1 1 0 1 1 1			
BAA = Decimal Adjust for Add	0 0 1 0 0 1 1 1			
SUB = **Subtract:**				
Reg./Memory and Register to Either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s w = 01
Immediate from Accumulator	0 0 1 0 1 1 0 w	data	data if w = 1	
SSB = **Subtract with Borrow**				
Reg./Memory and Register to Either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s w = 01
Immediate from Accumulator	0 0 0 1 1 1 w	data	data if w = 1	
DEC = **Decrement:**				
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			

I.6 ASM 8086 Instructions & Encoding

NEG = Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		
CMP = **Compare:**				
Register/Memory and Register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with Register/Memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w = 01
Immediate with Accumulator	0 0 1 1 1 1 0 w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	0 0 1 1 1 1 1 1			
DAS = Decimal Adjust for Subtract	0 0 1 0 1 1 1 1			
MUL = Multiply (Unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL = Integer Multiply (Signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM = ASCII Adjust for Multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV = Divide (Unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV = Integer Divide (Signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD = ASCII Adjust for Divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW = Convert Byte to Word	1 0 0 1 1 0 0 0			
CWD = Convert Word to Double Word	1 0 0 1 1 0 0 1			

I.6 ASM 8086 Instructions & Encoding

Mnemonic and Description	Instruction Code			
LOGIC	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
NOT = Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m		
SHL/SAL = Shift Logical/Arithmetic Left	1 1 0 1 0 0 v w	mod 1 0 0 r/m		
SHR = Shift Logical Right	1 1 0 1 0 0 v w	mod 1 0 1 r/m		
SAR = Shift Arithmetic Right	1 1 0 1 0 0 v w	mod 1 1 1 r/m		
ROL = Rotate Left	1 1 0 1 0 0 v w	mod 0 0 0 r/m		
ROR = Rotate Right	1 1 0 1 0 0 v w	mod 0 0 1 r/m		
RCL = Rotate Through Carry Flag Left	1 1 0 1 0 0 v w	mod 0 1 0 r/m		
RCR = Rotate Through Carry Right	1 1 0 1 0 0 v w	mod 0 1 1 r/m		
AND = **And:**				
Reg./Memory and Register to Either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 0 0 1 0 w	data	data if w = 1	
TEST = **And Function to Flags, No Result:**				
Register/Memory and Register	1 0 0 0 0 1 0 w	mod reg r/m		
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate Data and Accumulator	1 0 1 0 1 0 0 w	data	data if w = 1	
OR = **Or:**				
Reg./Memory and Register to Either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1
Immediate to Accumulator	0 0 0 0 1 1 0 w	data	data if w = 1	

I.6 ASM 8086 Instructions & Encoding

XOR = Exclusive or:

Reg./Memory and Register to Either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 1 0 1 0 w	data	data if w = 1	

STRING MANIPULATION

REP = Repeat	1 1 1 1 0 0 1 z
MOVS = Move Byte/Word	1 0 1 0 0 1 0 w
CMPS = Compare Byte/Word	1 0 1 0 0 1 1 w
SCAS = Scan Byte/Word	1 0 1 0 1 1 1 w
LODS = Load Byte/Wd to AL/AX	1 0 1 0 1 1 0 w
STOS = Stor Byte/Wd from AL/A	1 0 1 0 1 0 1 w

CONTROL TRANSFER

CALL = Call:

Direct within Segment	1 1 1 0 1 0 0 0	disp-low	disp-high
Indirect within Segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m	
Direct Intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	

I.6 ASM 8086 Instructions & Encoding

Mnemonic and Description	Instruction Code		
JMP = Unconditional Jump:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct within Segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct within Segment-Short	1 1 1 0 1 0 1 1	disp	
Indirect within Segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	
Direct Intersegment	1 1 1 0 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	
RET = Return from CALL:			
Within Segment	1 1 0 0 0 0 1 1		
Within Seg Adding Immed to SP	1 1 0 0 0 0 1 0	data-low	data-high
Intersegment	1 1 0 0 1 0 1 1		
Intersegment Adding Immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high
JE/JZ = Jump on Equal/Zero	0 1 1 1 0 1 0 0	disp	
JL/JNGE = Jump on Less/Not Greater or Equal	0 1 1 1 1 1 0 0	disp	
JLE/JNG = Jump on Less or Equal/ Not Greater	0 1 1 1 1 1 1 0	disp	

I.6 ASM 8086 Instructions & Encoding

JB/JNAE = Jump on Below/Not Above or Equal	0 1 1 1 0 0 1 0	disp
JBE/JNA = Jump on Below or Equal/ Not Above	0 1 1 1 0 1 1 0	disp
JP/JPE = Jump on Parity/Parity Even	0 1 1 1 1 0 1 0	disp
JO = Jump on Overflow	0 1 1 1 0 0 0 0	disp
JS = Jump on Sign	0 1 1 1 1 0 0 0	disp
JNE/JNZ = Jump on Not Equal/Not Zero	0 1 1 1 0 1 0 1	disp
JNL/JGE = Jump on Not Less/Greater or Equal	0 1 1 1 1 1 0 1	disp
JNLE/JG = Jump on Not Less or Equal/ Greater	0 1 1 1 1 1 1 1	disp
JNB/JAE = Jump on Not Below/Above or Equal	0 1 1 1 0 0 1 1	disp
JNBE/JA = Jump on Not Below or Equal/Above	0 1 1 1 0 1 1 1	disp
JNP/JPO = Jump on Not Par/Par Odd	0 1 1 1 1 0 1 1	disp
JNO = Jump on Not Overflow	0 1 1 1 0 0 0 1	disp
JNS = Jump on Not Sign	0 1 1 1 1 0 0 1	disp
LOOP = Loop CX Times	1 1 1 0 0 0 1 0	disp
LOOPZ/LOOPE = Loop While Zero/Equal	1 1 1 0 0 0 0 1	disp
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	1 1 1 0 0 0 0 0	disp
JCXZ = Jump on CX Zero	1 1 1 0 0 0 1 1	disp
INT = **Interrupt**		
Type Specified	1 1 0 0 1 1 0 1	type
Type 3	1 1 0 0 1 1 0 0	
INTO = Interrupt on Overflow	1 1 0 0 1 1 1 0	
IRET = Interrupt Return	1 1 0 0 1 1 1 1	

I.6 ASM 8086 Instructions & Encoding

Mnemonic and Description	Instruction Code	
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
PROCESSOR CONTROL		
CLC = Clear Carry	1 1 1 1 1 0 0 0	
CMC = Complement Carry	1 1 1 1 0 1 0 1	
STC = Set Carry	1 1 1 1 1 0 0 1	
CLD = Clear Direction	1 1 1 1 1 1 0 0	
STD = Set Direction	1 1 1 1 1 1 0 1	
CLI = Clear Interrupt	1 1 1 1 1 0 1 0	
STI = Set Interrupt	1 1 1 1 1 0 1 1	
HLT = Halt	1 1 1 1 0 1 0 0	
WAIT = Wait	1 0 0 1 1 0 1 1	
ESC = Escape (to External Device)	1 1 0 1 1 x x x	mod x x x r/m
LOCK = Bus Lock Prefix	1 1 1 1 0 0 0 0	

I.6 ASM 8086 Instructions & Encoding

NOTES:

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive;
Less = less positive (more negative) signed values
if d = 1 then "to" reg; if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction
if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
if mod = 10 then DISP = disp-high; disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)
*except if mod = 00 and r/m = 110 then EA = disp-high; disp-low.

if s w = 01 then 16 bits of immediate data form the operand
if s w = 11 then an immediate data byte is sign extended to form the 16-bit operand
if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with ZF FLAG

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:
FLAGS = X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

I.6 ASM 8086 Instructions & Encoding

SAMPLE:

8B D8 mov BX, AX;

1000 1011 11 011 000

dw mod reg r/m

1000 1011 11 000 011

8B C3 mov AX, BX;

I.7 The Fundamental Programming Control Structures

a) CMP Instruction

- Compares 2 values using a virtual subtract operation;
- The operands are not modifying their values;
- The flags from the flags register are set in order to reflect the result of the imaginary subtraction operation.

Flags Bits Affected:

- **OF (overflow)**
- **SF (sign)**
- **ZF (zero)**
- **PF (parity)**
- **CF (carry)**

Operands types

- **register / register**
- **register / memory**
- **register / immediate**
- **memory / register**

I.7 The Fundamental Programming Control Structures

Sample for CMP instruction

AX = 10, BX = -12

Flags Status:

- CMP AX, BX
  - AX – BX = +22
- SF=0, CF=1, OF=0, ZF=0

- CMP BX, AX
  - BX – AX = -22
- SF=1, CF=0, OF=0, ZF=0

- CMP AX, AX
  - AX – AX = 0
- SF=0, CF=0, OF=0, ZF=1

b) Conditional Jumps

Operation	When both operands are unsigned	When at least one operand is signed
<> or !=	JNE or JNZ	JNE or JNZ
= or ==	JE or JZ	JE or JZ
>=	JNB	JNL
>	JNBE or JA	JNLE or JG
<=	JNA	JNG
<	JNAE	JNGE or JL

c) Control Structures in Programming

- IF-THEN
- IF-THEN-ELSE
- REPEAT
- DO-WHILE
- WHILE-DO
- SWITCH-CASE

In x86 ASM these control structures in programming are implemented using CMP or TEST Instruction & "JUMPs"

I.7 The Fundamental Programming Control Structures

The Programming Control Structure *IF-THEN*

C/C++/Java

```
if (op1 == op2)
{
   < expression 1 >
   < expression 2 >
}
```

ASSEMBLER

```
    cmp op1, op2
    jne false
    <expression 1>
    <expression 2>
false:
    <program code>
```

I.7 The Fundamental Programming Control Structures

The Programming Control Structure *IF-THEN-ELSE*

C/C++/Java

```
if (op1 > op2)
{
    < expression 1 >
    < expression 2 >
}
else
{
    < expression 3 >
    < expression 4 >
}
```

ASSEMBLER

```
      cmp op1, op2
      jle else
      < expression 1 >
      < expression 2 >
      jmp done
else:
      < expression 3 >
      < expression 4 >
done:
```

I.7 The Fundamental Programming Control Structures

The Programming Control Structure *REPEAT*

C/C++/Java

```
for (init; op1<op2; iteration)
{
   <expression 1>
   <expression 2>
}
```

ASSEMBLER

```
    <init>
repeat:
    cmp op1,op2
    jae final
    <expression 1>
    <expression 2>
    <iteration>
    jmp repeat
final:
```

The Programming Control Structure *DO-WHILE*

C/C++/Java

```
do
{
   <expression 1>
   <expression 2>
} while (op1 != op2);
```

ASSEMBLER

```
do:
   <expression 1>
   <expression 2>
   cmp op1,op2
jnz do
```

The Programming Control Structure *DO-WHILE – ASM x86 loop instruction*

C/C++/Java	ASSEMBLER

```
i=n;
do
{
    <expression 1>
    <expression 2>
    i=i-1;
} while (i>0);
```

```
    mov cx,n
repeat:
    <expression 1>
    <expression 2>
loop repeat
```

Loop instruction is a conditional (by CX) jump and it MUST be a short jump to the specified label (between -127 and +127 bytes from the current position)

The Programming Control Structure *CASE (SWITCH)*

C/C++/Java

```
switch (value) {
  case constant_value1:
     <expression 1>
     break;
  case constant_value2:
     <expression 2>
     break;
 …
 default:
      <expression>
}
```

ASSEMBLER

```
       cmp value, constant_value1
       je case1
       cmp value, constant_value2
       je case2
       …
       jmp default
case1:
       <expression 1>
       jmp final
case2:
       <expression 2>
       jmp final
…
default:
       <expression>
final:
```

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

            end START
```

Processor 8086 - 80286

	15 0
AX	
BX	
CX	
DX	
BP	
SP	
SI	
DI	
IP	
DS	

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	DF32h
BX	
CX	
DX	
BP	0000h
SP	0010h
SI	
DI	
IP	0003h
DS	

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	DF32h
BX	
CX	
DX	
BP	0000h
SP	0010h
SI	
DI	
IP	0005h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	DF32h
BX	
CX	0h
DX	
BP	0000h
SP	0010h
SI	
DI	
IP	0007h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

Stack Area

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

```
.model small
.stack 10h
.data
        n dw 4
        vector dw 23,16,35,12
        suma dw ?
.code
        START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

        end START
```

Processor 8086 - 80286

	15 … 0
AX	DF32h
BX	
CX	0004h
DX	
BP	0000h
SP	0010h
SI	
DI	
IP	000Bh
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012          REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	DF32h
BX	0002h
CX	0004h
DX	
BP	0000h
SP	0010h
SI	
DI	
IP	000Eh
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 … 0
AX	0h
BX	0002h
CX	0004h
DX	
BP	0000h
SP	0010h
SI	
DI	
IP	0010h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
.model small
.stack 10h
.data
        n dw 4
        vector dw 23,16,35,12
        suma dw ?
.code
        START:

0000  B8 0000s        mov AX,@data
0003  8E D8           mov DS,AX

0005  33 C9           xor CX,CX
0007  8B 0E 0000r     mov CX, n
000B  BB 0002r        mov BX, offset vector
000E  33 C0           xor AX,AX
0010  33 F6           xor SI,SI

0012        REPETA:

0012  03 00           add AX, [BX][SI]
0014  46              inc SI
0015  46              inc SI
0016  E2 FA           loop REPETA

0018  A3 000Ar        mov suma, AX
001B  B8 4C00         mov ax, 4C00h
001E  CD 21           int 21h

        end START
```

Processor 8086 - 80286

	15 0
AX	0h
BX	0002h
CX	0004h
DX	
BP	0000h
SP	0010h
SI	0h
DI	
IP	0012h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 — 0
AX	0017h
BX	0002h
CX	0004h
DX	
BP	0000h
SP	0010h
SI	0h
DI	
IP	0014h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	0017h
BX	0002h
CX	0004h
DX	
BP	0000h
SP	0010h
SI	0001h
DI	
IP	0015h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	0017h
BX	0002h
CX	0004h
DX	
BP	0000h
SP	0010h
SI	0002h
DI	
IP	0016h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	0017h
BX	0002h
CX	0003h
DX	
BP	0000h
SP	0010h
SI	0002h
DI	
IP	0012h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	0027h
BX	0002h
CX	0003h
DX	
BP	0000h
SP	0010h
SI	0002h
DI	
IP	0012h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
.model small
.stack 10h
.data
        n dw 4
        vector dw 23,16,35,12
        suma dw ?
.code
        START:

0000  B8 0000s        mov AX,@data
0003  8E D8           mov DS,AX

0005  33 C9           xor CX,CX
0007  8B 0E 0000r     mov CX, n
000B  BB 0002r        mov BX, offset vector
000E  33 C0           xor AX,AX
0010  33 F6           xor SI,SI

0012        REPETA:

0012  03 00           add AX, [BX][SI]
0014  46              inc SI
0015  46              inc SI
0016  E2 FA           loop REPETA

0018  A3 000Ar        mov suma, AX
001B  B8 4C00         mov ax, 4C00h
001E  CD 21           int 21h

        end START
```

Processor 8086 - 80286

	15 0
AX	0027h
BX	0002h
CX	0002h
DX	
BP	0000h
SP	0010h
SI	0004h
DI	
IP	0012h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s        mov AX,@data
0003  8E D8           mov DS,AX

0005  33 C9           xor CX,CX
0007  8B 0E 0000r     mov CX, n
000B  BB 0002r        mov BX, offset vector
000E  33 C0           xor AX,AX
0010  33 F6           xor SI,SI

0012        REPETA:

0012  03 00           add AX, [BX][SI]
0014  46              inc SI
0015  46              inc SI
0016  E2 FA           loop REPETA

0018  A3 000Ar        mov suma, AX
001B  B8 4C00         mov ax, 4C00h
001E  CD 21           int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	004Ah
BX	0002h
CX	0002h
DX	
BP	0000h
SP	0010h
SI	0004h
DI	
IP	0014h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	004Ah
BX	0002h
CX	0001h
DX	
BP	0000h
SP	0010h
SI	0006h
DI	
IP	0012h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	0056h
BX	0002h
CX	0001h
DX	
BP	0000h
SP	0010h
SI	0006h
DI	
IP	0014h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012            REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

Register	15 … 0
AX	0056h
BX	0002h
CX	0000h
DX	
BP	0000h
SP	0010h
SI	0008h
DI	
IP	0018h
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
                START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

                end START
```

Processor 8086 - 80286

	15 0
AX	0056h
BX	0002h
CX	0000h
DX	
BP	0000h
SP	0010h
SI	0008h
DI	
IP	001Bh
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	56h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

```
.model small
.stack 10h
.data
        n dw 4
        vector dw 23,16,35,12
        suma dw ?
.code
        START:

0000  B8 0000s          mov AX,@data
0003  8E D8             mov DS,AX

0005  33 C9             xor CX,CX
0007  8B 0E 0000r       mov CX, n
000B  BB 0002r          mov BX, offset vector
000E  33 C0             xor AX,AX
0010  33 F6             xor SI,SI

0012        REPETA:

0012  03 00             add AX, [BX][SI]
0014  46                inc SI
0015  46                inc SI
0016  E2 FA             loop REPETA

0018  A3 000Ar          mov suma, AX
001B  B8 4C00           mov ax, 4C00h
001E  CD 21             int 21h

        end START
```

Processor 8086 - 80286

	15 0
AX	4C00h
BX	0002h
CX	0000h
DX	
BP	0000h
SP	0010h
SI	0008h
DI	
IP	001Eh
DS	DF32h

Data Area

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

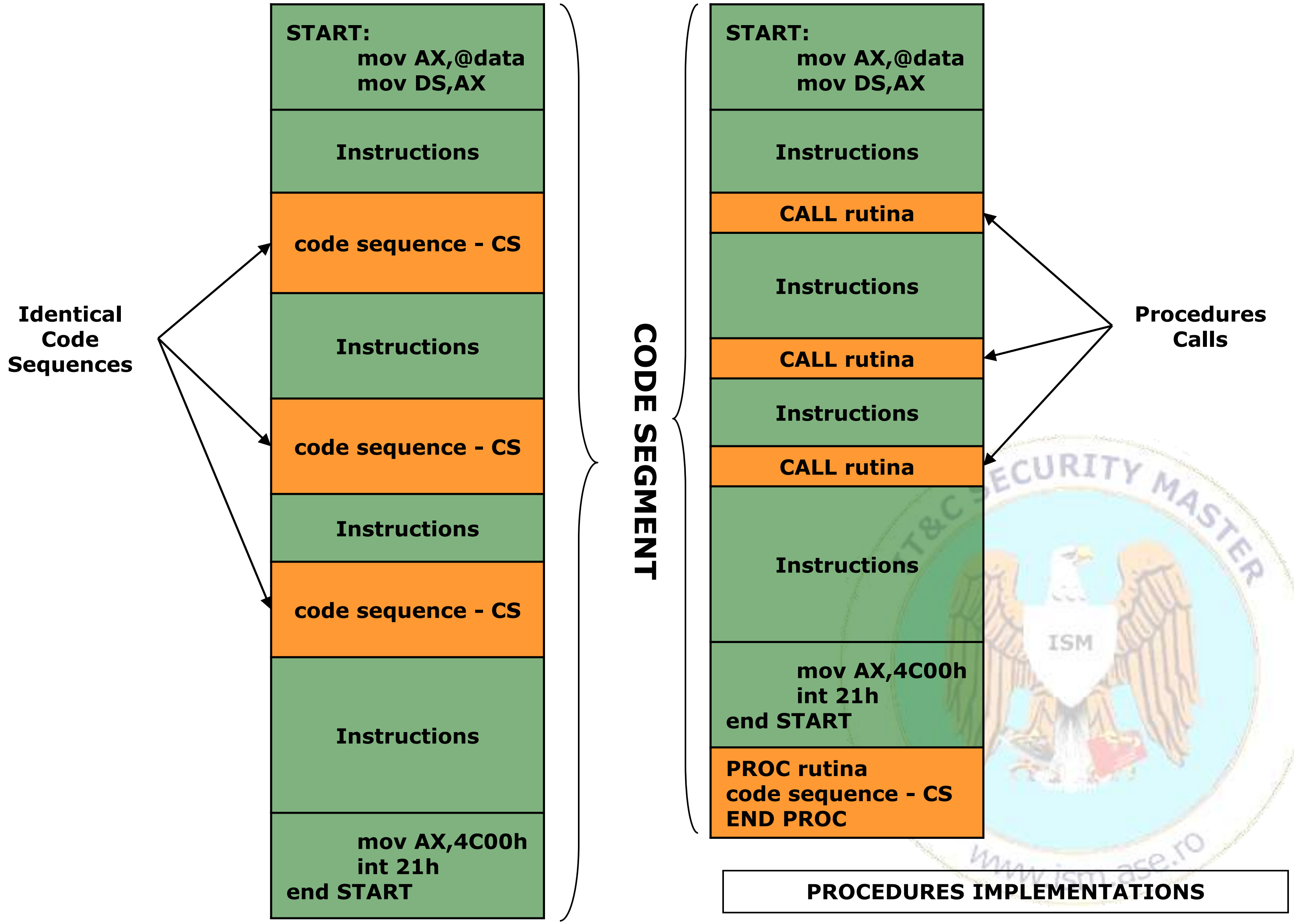
DS:09	0A	0B	0C	0D
00h	56h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Area

DAY 2

I.8 PROCEDURES



I.8 PROCEDURES

PROCEDURE SYNTAX:

Procedure_Name PROC [FAR | NEAR]

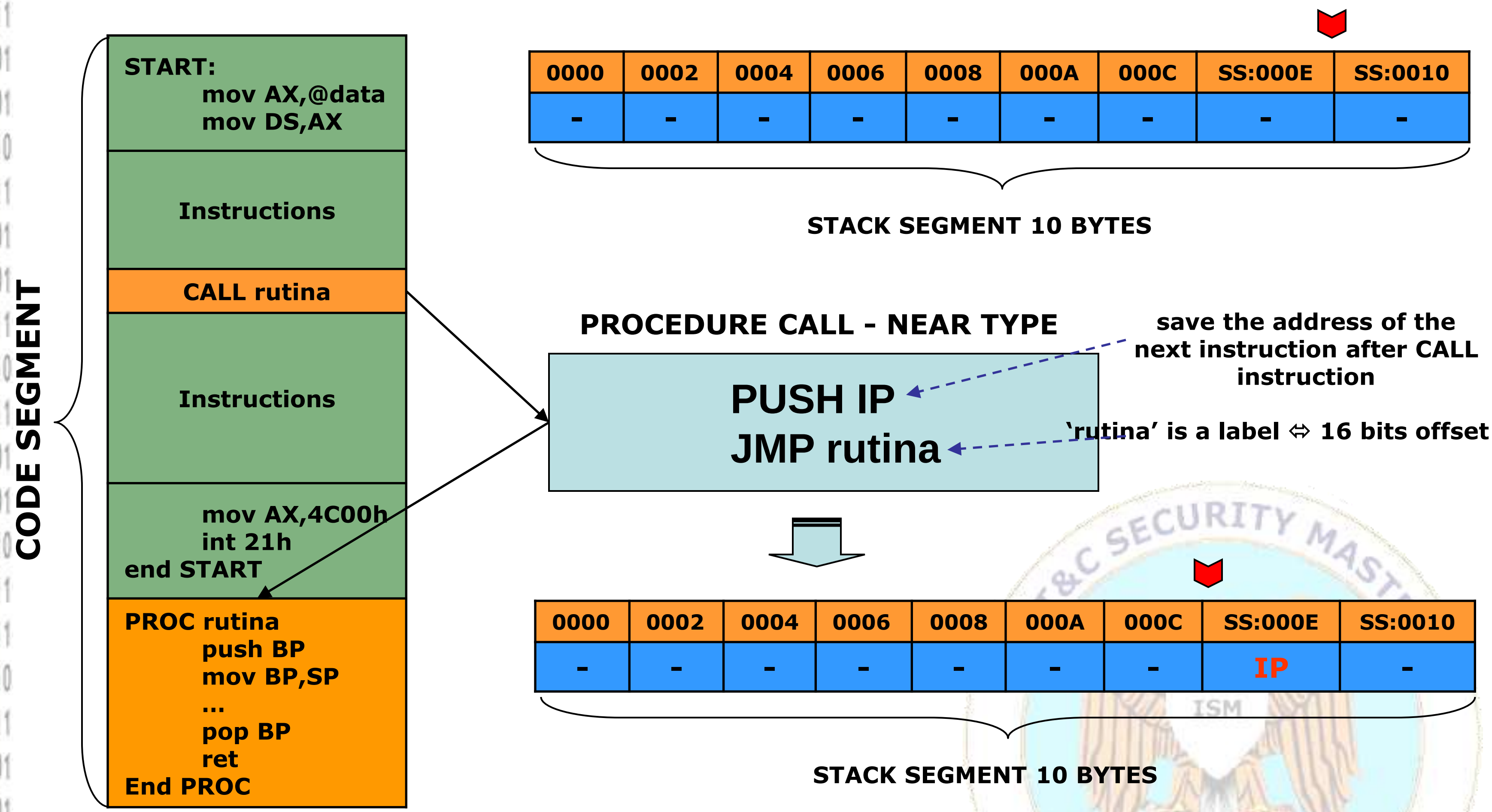
....

[Procedure_Name] ENDP

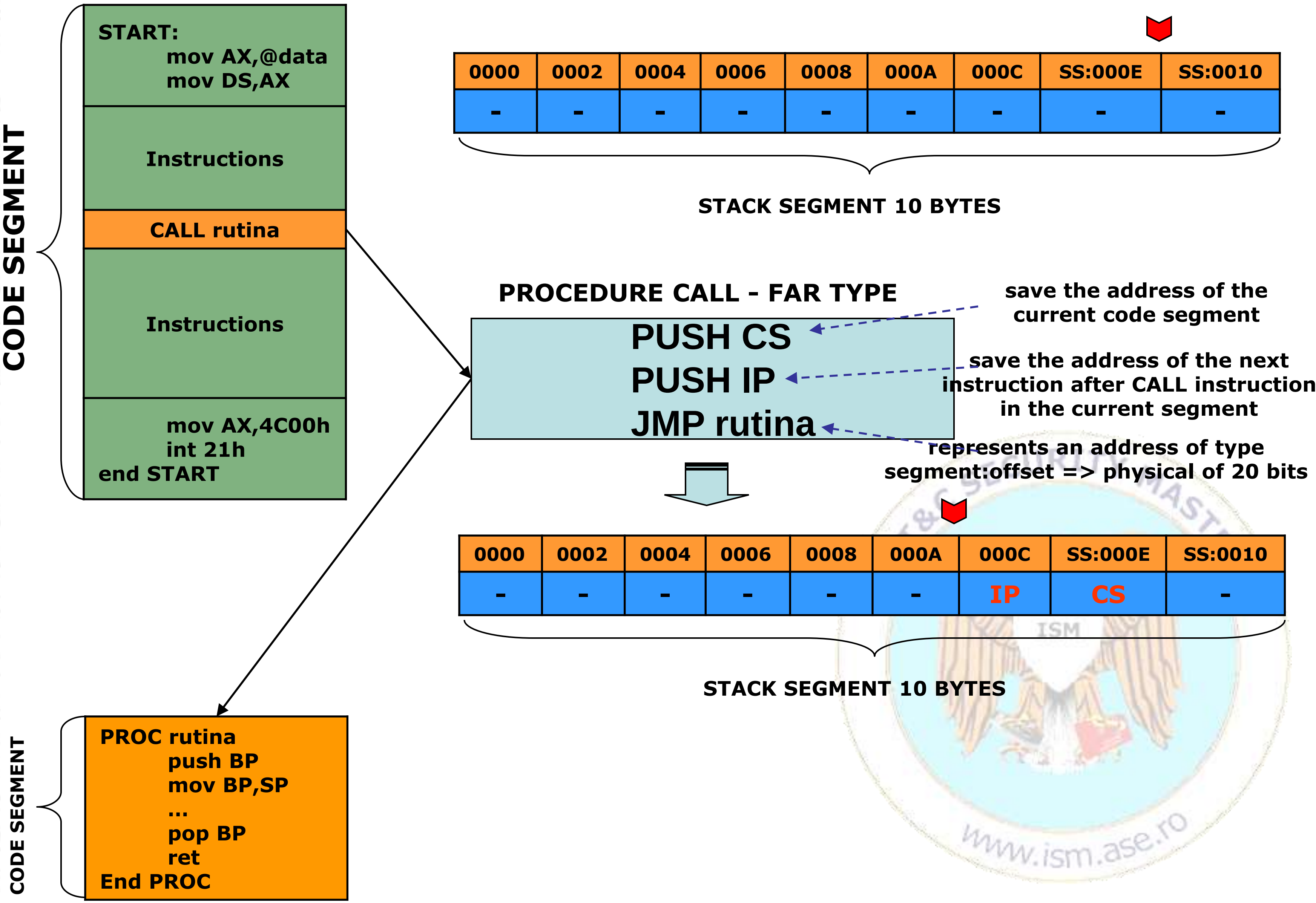
General structure of the procedure in assembler:

Procedure_Name PROC [FAR | NEAR]
Registers Saving
Processing code using the formal parameters
Saving the results
Register restoring
RET [bytes_no]
[Procedure_Name] ENDP

I.8 PROCEDURES



I.8 PROCEDURES



I.8 PROCEDURES

OPERAND TYPE	DESCRIPTION
CALL procedure_name	Procedure name is called as beig a label.
CALL label	The microprocessor considers in these situations that the label is local and the jump is NEAR => the jump must be in terms of bytes related to encoded instructions in range [-32768,32676] bytes.
CALL FAR PTR label	The label is in another segment. The instruction replaces CS and IP register values with the value of the label's segment & offset.
CALL register or variable	The value from the register or the variable is copied in IP register after the old IP value has been pushed on the stack segment. Therefore, the register or the variable represents a NEAR pointer (contains only the procedure offset)
CALL [WORD|DOUBLE] PTR variable	The value from the variable represents an offset and the jump is NEAR type. Usually the variable is represented by a index register (SI,DI) or base register (BX) – an indirect addressing type. In [BX] or [SI] or [DI] should be specified how many bytes are read (must be indicated the JUMP type – NEAR/WORD, are read 2 bytes; FAR/DOUBLE are read 4 bytes).
CALL DWORD PTR address	The value from the variable represents a segment + offset and the JUMP is FAR type. Usually the variable is represented by a index register (SI,DI) – [SI] or [DI] or base register (BX) – [BX] – an indirect addressing type.

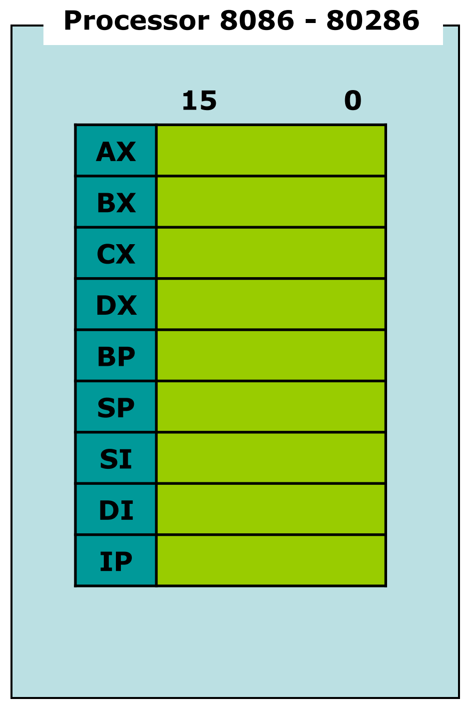
I.8 PROCEDURES

I/O Parameters Transfer to the Procedures

- **VARIABLES declared into memory - data segment;**
- **REGISTERS – *best practice – the results***
- **STACK – transfer area between the procedures and the main program – *best practice – the input parameters*;**

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```



Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Segment

```
     .model small
     .stack 10h
     .data
          n dw 4
          vector dw 23,16,35,12
          suma dw ?
     .code
0000 START:
0000      mov ax,@data
0003      mov ds,ax
0005      mov AX, offset vector
0008      push AX
0009      mov AX,n
000C      push AX
000D      mov AX, offset suma
0010      push AX

0011      CALL addv
0014      mov ax, 4C00h
0017      int 21h
0019 addv PROC NEAR
0019      push BP
001A      mov BP, SP
001C      xor CX,CX
001E      mov CX, [BP+6]
0021      mov BX, [BP+8]
0024      xor AX,AX
0026      xor SI,SI
0028 repeta:
0028      add AX, [BX][SI]
002A      inc SI
002B      inc SI
002C      loop repeta
002E      mov BX, [BP+4]
0031      mov [BX],AX
0033      pop BP
0034      ret 6
0037   addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0002
BX	
CX	
DX	
BP	0000
SP	0010
SI	
DI	
IP	0006

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```

Processor 8086 - 80286

	15 ... 0
AX	0002
BX	
CX	
DX	
BP	0000
SP	000E
SI	
DI	
IP	0009

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0004
BX	
CX	
DX	
BP	0000
SP	000E
SI	
DI	
IP	000C

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	SS:000E	SS:0010
-	-	-	-	-	-	-	0200	-

Stack Segment

```
          .model small
          .stack 10h
          .data
                 n dw 4
                 vector dw 23,16,35,12
                 suma dw ?
          .code
0000 START:
0000             mov ax,@data
0003             mov ds,ax
0005             mov AX, offset vector
0008             push AX
0009             mov AX,n
000C             push AX
000D             mov AX, offset suma
0010             push AX

0011             CALL addv
0014             mov ax, 4C00h
0017             int 21h
0019 addv PROC NEAR
0019             push BP
001A             mov BP, SP
001C             xor CX,CX
001E             mov CX, [BP+6]
0021             mov BX, [BP+8]
0024             xor AX,AX
0026             xor SI,SI
0028 repeta:
0028             add AX, [BX][SI]
002A             inc SI
002B             inc SI
002C             loop repeta
002E             mov BX, [BP+4]
0031             mov [BX],AX
0033             pop BP
0034             ret 6
0037     addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0004
BX	
CX	
DX	
BP	0000
SP	000C
SI	
DI	
IP	000D

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	-	-	-	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	000A
BX	
CX	
DX	
BP	0000
SP	000C
SI	
DI	
IP	0010

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	-	-	-	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	000A
BX	
CX	
DX	
BP	0000
SP	000A
SI	
DI	
IP	0011

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	-	-	0A00	0400	0200	-

Stack Segment

```
          .model small
          .stack 10h
          .data
                    n dw 4
                    vector dw 23,16,35,12
                    suma dw ?
          .code
0000 START:
0000                mov ax,@data
0003                mov ds,ax
0005                mov AX, offset vector
0008                push AX
0009                mov AX,n
000C                push AX
000D                mov AX, offset suma
0010                push AX

0011                CALL addv
0014                mov ax, 4C00h
0017                int 21h
0019 addv PROC NEAR
0019                push BP
001A                mov BP, SP
001C                xor CX,CX
001E                mov CX, [BP+6]
0021                mov BX, [BP+8]
0024                xor AX,AX
0026                xor SI,SI
0028 repeta:
0028                add AX, [BX][SI]
002A                inc SI
002B                inc SI
002C                loop repeta
002E                mov BX, [BP+4]
0031                mov [BX],AX
0033                pop BP
0034                ret 6
0037     addv ENDP
end START
```

Processor 8086 - 80286

	15 — 0
AX	000A
BX	
CX	
DX	
BP	0000
SP	0008
SI	
DI	
IP	0019

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	-	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037      addv ENDP
end START
```

Processor 8086 - 80286

	15 … 0
AX	000A
BX	
CX	
DX	
BP	0000
SP	0006
SI	
DI	
IP	001A

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	000A
BX	
CX	
DX	
BP	0006
SP	0006
SI	
DI	
IP	001C

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
          .model small
          .stack 10h
          .data
                  n dw 4
                  vector dw 23,16,35,12
                  suma dw ?
          .code
0000 START:
0000              mov ax,@data
0003              mov ds,ax
0005              mov AX, offset vector
0008              push AX
0009              mov AX,n
000C              push AX
000D              mov AX, offset suma
0010              push AX

0011              CALL addv
0014              mov ax, 4C00h
0017              int 21h
0019 addv PROC NEAR
0019              push BP
001A              mov BP, SP
001C              xor CX,CX
001E              mov CX, [BP+6]
0021              mov BX, [BP+8]
0024              xor AX,AX
0026              xor SI,SI
0028 repeta:
0028              add AX, [BX][SI]
002A              inc SI
002B              inc SI
002C              loop repeta
002E              mov BX, [BP+4]
0031              mov [BX],AX
0033              pop BP
0034              ret 6
0037      addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	000A
BX	
CX	0000
DX	
BP	0006
SP	0006
SI	
DI	
IP	001E

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	000A
BX	
CX	0004
DX	
BP	0006
SP	0006
SI	
DI	
IP	0021

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037     addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	000A
BX	0002
CX	0004
DX	
BP	0006
SP	0006
SI	
DI	
IP	0024

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0000
BX	0002
CX	0004
DX	
BP	0006
SP	0006
SI	
DI	
IP	0026

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

Stack Segment

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

```
     .model small
     .stack 10h
     .data
          n dw 4
          vector dw 23,16,35,12
          suma dw ?
     .code
0000 START:
0000      mov ax,@data
0003      mov ds,ax
0005      mov AX, offset vector
0008      push AX
0009      mov AX,n
000C      push AX
000D      mov AX, offset suma
0010      push AX

0011      CALL addv
0014      mov ax, 4C00h
0017      int 21h
0019 addv PROC NEAR
0019      push BP
001A      mov BP, SP
001C      xor CX,CX
001E      mov CX, [BP+6]
0021      mov BX, [BP+8]
0024      xor AX,AX
0026      xor SI,SI
0028 repeta:
0028      add AX, [BX][SI]
002A      inc SI
002B      inc SI
002C      loop repeta
002E      mov BX, [BP+4]
0031      mov [BX],AX
0033      pop BP
0034      ret 6
0037   addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0000
BX	0002
CX	0004
DX	
BP	0006
SP	0006
SI	0000
DI	
IP	0028

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
          .model small
          .stack 10h
          .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
          .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037     addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0017
BX	0002
CX	0004
DX	
BP	0006
SP	0006
SI	0000
DI	
IP	002A

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
          .model small
          .stack 10h
          .data
                 n dw 4
                 vector dw 23,16,35,12
                 suma dw ?
          .code
0000 START:
0000             mov ax,@data
0003             mov ds,ax
0005             mov AX, offset vector
0008             push AX
0009             mov AX,n
000C             push AX
000D             mov AX, offset suma
0010             push AX

0011             CALL addv
0014             mov ax, 4C00h
0017             int 21h
0019 addv PROC NEAR
0019             push BP
001A             mov BP, SP
001C             xor CX,CX
001E             mov CX, [BP+6]
0021             mov BX, [BP+8]
0024             xor AX,AX
0026             xor SI,SI
0028 repeta:
0028             add AX, [BX][SI]
002A             inc SI
002B             inc SI
002C             loop repeta
002E             mov BX, [BP+4]
0031             mov [BX],AX
0033             pop BP
0034             ret 6
0037      addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0017
BX	0002
CX	0004
DX	
BP	0006
SP	0006
SI	0002
DI	
IP	002C

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037     addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0017
BX	0002
CX	0003
DX	
BP	0006
SP	0006
SI	0002
DI	
IP	0028

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037      addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0027
BX	0002
CX	0003
DX	
BP	0006
SP	0006
SI	0002
DI	
IP	002A

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```

Processor 8086 - 80286

15 … 0	
AX	0027
BX	0002
CX	0003
DX	
BP	0006
SP	0006
SI	0004
DI	
IP	002C

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0027
BX	0002
CX	0002
DX	
BP	0006
SP	0006
SI	0004
DI	
IP	0028

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0056
BX	0002
CX	0000
DX	
BP	0006
SP	0006
SI	0004
DI	
IP	002E

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037     addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0056
BX	000A
CX	0000
DX	
BP	0006
SP	0006
SI	0004
DI	
IP	0031

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	00h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037      addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0056
BX	000A
CX	0000
DX	
BP	0006
SP	0006
SI	0004
DI	
IP	0033

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	56h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
          .model small
          .stack 10h
          .data
                  n dw 4
                  vector dw 23,16,35,12
                  suma dw ?
          .code
0000 START:
0000              mov ax,@data
0003              mov ds,ax
0005              mov AX, offset vector
0008              push AX
0009              mov AX,n
000C              push AX
000D              mov AX, offset suma
0010              push AX

0011              CALL addv
0014              mov ax, 4C00h
0017              int 21h
0019 addv PROC NEAR
0019              push BP
001A              mov BP, SP
001C              xor CX,CX
001E              mov CX, [BP+6]
0021              mov BX, [BP+8]
0024              xor AX,AX
0026              xor SI,SI
0028 repeta:
0028              add AX, [BX][SI]
002A              inc SI
002B              inc SI
002C              loop repeta
002E              mov BX, [BP+4]
0031              mov [BX],AX
0033              pop BP
0034              ret 6
0037      addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0056
BX	000A
CX	0000
DX	
BP	0000
SP	0008
SI	0004
DI	
IP	0034

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	56h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	0000	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037      addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	0056
BX	000A
CX	0000
DX	
BP	0000
SP	0010
SI	0004
DI	
IP	0014

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	56h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	-	1400	0A00	0400	0200	-

Stack Segment

```
        .model small
        .stack 10h
        .data
                n dw 4
                vector dw 23,16,35,12
                suma dw ?
        .code
0000 START:
0000            mov ax,@data
0003            mov ds,ax
0005            mov AX, offset vector
0008            push AX
0009            mov AX,n
000C            push AX
000D            mov AX, offset suma
0010            push AX

0011            CALL addv
0014            mov ax, 4C00h
0017            int 21h
0019 addv PROC NEAR
0019            push BP
001A            mov BP, SP
001C            xor CX,CX
001E            mov CX, [BP+6]
0021            mov BX, [BP+8]
0024            xor AX,AX
0026            xor SI,SI
0028 repeta:
0028            add AX, [BX][SI]
002A            inc SI
002B            inc SI
002C            loop repeta
002E            mov BX, [BP+4]
0031            mov [BX],AX
0033            pop BP
0034            ret 6
0037    addv ENDP
end START
```

Processor 8086 - 80286

	15 0
AX	4C00
BX	000A
CX	0000
DX	
BP	0000
SP	0010
SI	0004
DI	
IP	0017

Data Segment

DS:00	01	02	03	04	05	06	07	08
04h	00h	17h	00h	10h	00h	23h	00h	0Ch

DS:09	0A	0B	0C	0D
00h	56h	00h	00h	00h

0000	0002	0004	0006	0008	000A	000C	000E	SS:0010
-	-	-	-	-	-	-	-	-

Stack Segment

I.9 MACRODEFINITIONS

a) DEFINING & EXPANDING THE MACRODEFINITIONS

ASM Source Code sequence which is used for the program modularization.

SYNTAX:

```
Macro_Name MACRO
        ;Instructions
ENDM
```

The macro-definitions are NOT procedures => they are expanded in the preprocessing step which is before the assembling step – the "macro-definition call" is replaced by the source code of the instructions from the macro-definition body.

I.9 MACRODEFINITIONS

b) MACRODEFINITIONS without PARAMETERS

Saving the registers:

```
savereg MACRO
        push AX
        push BX
        push CX
        push DX
        push SI
        push DI
ENDM
```

Restoring the registers:

```
restreg MACRO
        pop DI
        pop SI
        pop DX
        pop CX
        pop BX
        pop AX
ENDM
```

Macro-definitions “Call” & Expanding:

```
Procedura1 PROC NEAR
    savereg
    …
    restreg
    RET
Procedura1 ENDP
```

```
Procedura1 PROC NEAR
    push AX
    push BX
    …
    pop BX
    pop AX
    RET
Procedura1 ENDP
```

I.9 MACRODEFINITIONS

c) MACRODEFINITIONS with PARAMETERS

```
macro_name MACRO P1, P2,…, Pn
                    ;Instructions
ENDM
```

```
add02 MACRO a, b, s
          push AX
                    mov AX, a
                    add  AX, b
                    mov s, AX
          pop   AX
ENDM
```

The expression avaluation:
e = a + b + c + d

```
add02 a, b, e
add02 e, c, e
add02 e, d, e
```

I.9 MACRODEFINITIONS

c) MACRODEFINITIONS with PARAMETERS

The one must pay attention to the modification of the SP register value which is directly related with the stack segment access – PUSH/POP instructions

```
swap01 MACRO X, Y
        push AX
        push BX
        mov BX, X
        mov AX, Y
        mov X, AX
        mov Y, BX
        pop   BX
        pop   AX
ENDM
```

```
swap01 AX, SI ; Interchange AX with SI ?
        push AX
        push BX
        mov BX, AX
        mov AX, SI
        mov AX, AX
        mov SI, BX
        pop   BX
        pop   AX
```

```
swap01 SP, DI ; Interchange SP with DI ?
        push AX
        push BX
        mov BX, SP
        mov AX, DI
mov SP, AX ; Here SP is modified
mov DI, BX  ; and the POP instructions
pop   BX    ; are compromised
pop   AX    ; (solve with XCHG instruction)
```

I.9 MACRODEFINITIONS

d) LOCAL LABELS & VARIABLES

Local Labels (in expanding step maxim 10000)

```
minim MACRO a, b, c, min
  local et1, et2
   push AX
   mov  AX, a
   cmp  AX, b
   jlz et1
   mov  AX, b
et1:
   cmp AX, c
   jlz et2
   mov AX, c
et2:
   mov min, AX
   pop AX
ENDM
```

Local variables

```
mask MACRO X
   local masca
 .data
   masca equ 00001111b
 .code
   push AX
   mov  AL, X
   and  AL, masca
   mov  X, AL
   pop AX
ENDM
```

d) LOCAL LABELS & VARIABLES

Main Program:

```
...
minim x, y, z,  min1
minim w, v, u, min2
minim   i, j, k,  min3
minim min1, min2, min3, min
...
```

MACROEXPANDING ⇔ MACRO-DEFINITION "CALL"

```
...
push AX
   mov  AX, x
   cmp  AX, y
   jlz et10000
   mov  AX, y
et10000:
   cmp AX, z
   jlz et20000
   mov AX, z
et20000:
   mov min1, AX
   pop   AX
   push AX

   mov  AX, w
   cmp  AX, v
   jlz et10001
   mov  AX, v
et10001:
   cmp AX, u
   jlz et20001
   mov AX, u
et20001:
   mov min2, AX
   pop AX
...
```

MACROEXPANDING 1

MACROEXPANDING 2

I.9 MACRODEFINITIONS

e) REPETITIVE MACRO-DEFINITIONS & SPECIFIC OPERATORS

```
REPT n
        ;instructions
ENDM
```

```
IRP formal_p, <actual_prms_list>
        ;Instructions
ENDM
```

SAMPLE

```
REPT 3
m_mesaj         ;Instructions
ENDM
```

```
m_mesaj
m_mesaj
m_mesaj
```

SAMPLE

```
IRP x, <'a', 'b', 'c'>
        db x
ENDM
```

```
db 'a'
db 'b'
db 'c'
```

I.9 MACRODEFINITIONS

f) THE DERIVATION of the MACRO-DEFINITIONS

Initial Solutions

```
add04 MACRO a1, a2, a3, a4, s
        push AX
        mov AX,  a1
        add  AX, a2
        add  AX, a3
        add  AX, a4
        mov s, AX
        pop   AX
ENDM
```

```
add02 MACRO a1, a2, sum
        push AX
        mov AX,  a1
        add  AX, a2

        mov sum, AX
        pop   AX
ENDM
```

DERIVATION

```
add04 MACRO a1, a2, a3, a4, s
        add02  a1, a2, s
        add02  s,   a3, s
        add02  s,   a4, s
ENDM
```

```
add04 MACRO a1, a2, a3, a4, s
 local t1, t2
.data
 t1 dw ?
 t2 dw ?
.code
        add02  a1, a2, t1
        add02  a3, a4, t2
        add02  t1, t2, s
ENDM
```

a) INTERRUPTS

Using the software interrupts should be for:
- execution transfer to the another memory segment (interrupts routines)
- accessing the DOS functions and ROM-BIOS in assembling languages

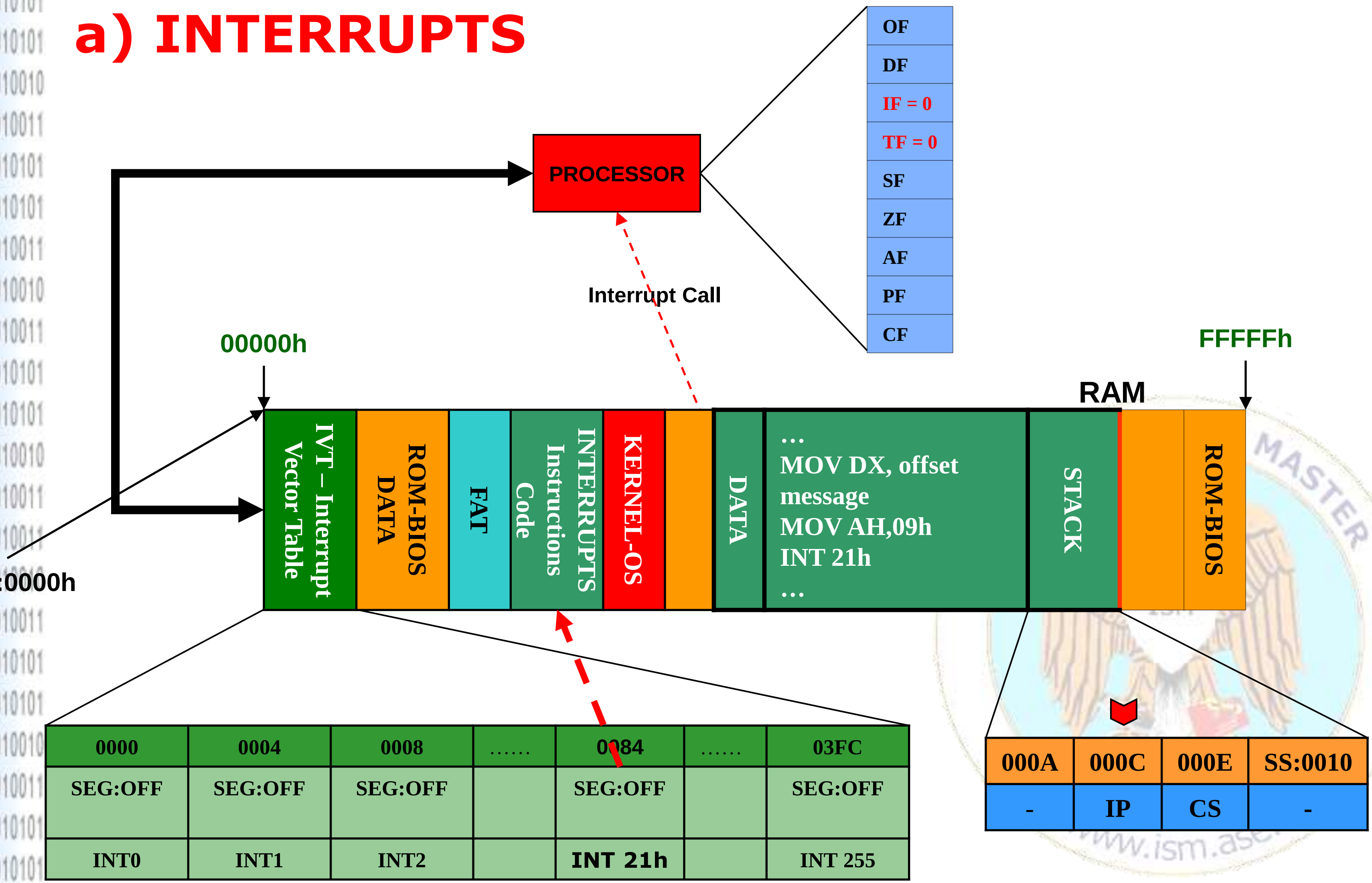
Features:
- are called using **INT** instruction followed by a value between 0 and 255;
- the interrupts routines in DOS & BIOS accept the parameter transfer though the microprocessor registers;
- most of the DOS interrupts routines receive the internal function code for execution into AH microprocessor register

```
message1        DB      "Interrupts sample !",'$'
.code
    MOV AX, @data
    MOV DS,AX
    MOV DX, offset message1
    MOV AH,09h
    INT 21h
```

Display the message to the console – using interrupt 21h with 09 function

I.10 Software Interrupts – File & Console Operations in DOS

a) INTERRUPTS



a) INTERRUPTS

For interrupt execution the microprocessor:

• search the interrupts routines in IVT – Interrupt Vectors Table; IVT is at the beginning of the RAM memory (segment 0, offset 0) and contains the far pointers (segment:offset) to the interrupt code handler; the position in the IVT defines the interrupt number (**INT 21h** ⇔ 0000h:(0021*4)h);

• clears (put them to 0) the flags TF (trap flag) & IF (interrupt enable flag) from the flags register;

• save on the stack segment the flags register (ZF, PF, CF, etc)

• save on the stack segment IP and CS registers for returning to the main program after interrupt execution;

• JUMP to the interrupt routine;

• executes the interrupt routine until it reach IRET;

• restore IP and CS

• restore the flags register (flag – ZF, PF, CF, etc)

• sets (put them to 1) the flags TF (trap flag) & IF (interrupt enable flag)

b) FILES in DOS

A file is identified through:
- **name** (char string), for opening and creation the file; OR
- **handler** (unsigned value on 16 bits which identifies in unique mode the file on disk and another resources), for closing, reading, writing, seeking

DEFAULT ASSIGNED HANDLERS FOR STANDARD DEVICES:

Value	Name	Description
0	stdin	Standard Input
1	stdout	Standard Output
2	stderr	Standard Output for Errors Messages
3	stdaux	Auxiliary Device
4	stdprn	Standard Printer

b) FILES in DOS

MAIN DOS O.S. FUNCTIONS – of INT 21h – FOR FILES OPERATIONS:

Function (AH)	Operation
3Ch	Create File
3Dh	Open File
3Eh	Close File
3Fh	Read File
40h	Write File
41h	Delete File
42h	Seek into the File
56h	Rename File

Input Parameters:	Registers
- Function Code	3Ch → AH
- File Name	DX
- File Attribute	CX
Output Parameters:	
- handler	AX
- operation result	Set/Clear CF – Carry Flag

At the creation time, the file type may be:
•read-only (1)
•hidden (2)
•system (4)
•normal (0)

b) FILES in DOS

```
.data
handler          dw      ?               ;file handler
fAttribute       dw      ?               ;attribute for file creation

res              db      ?               ;validation variable for the file operations
fileName         db      'fisier.dat',0  ;file name


.code
CrearteFile MACRO fileName, fAttribute, handler, res
local error1, final
        push AX
        push CX
        push DX
        mov AH,3Ch              ;function code for file creation
        mov CX,fAttribute       ;set file attribute
        lea DX, fileName        ;load in DX the offset of the assigned char string file name
        INT 21h                 ;call interrupt 21h
        jc error1               ;if the operation fails CF is set – value 1
                                        ;check out the operation success
        mov handler,AX          ; if the file has been created then init the handler
        mov res,0               ; if everything OK init res with 0
        jmp final               ;JUMP to the end of the macro-definition
error1:                                 ;in case of error
        mov handler,-1          ;init the handler with negative value 0xFF 0xFF
        mov res,AX              ;store in res the error code (is different than 0)
final:
        pop DX
        pop CX
        pop AX
ENDM
```

b) FILES in DOS

OPEN FILE

Input Parameters:	Registers:
- Function Code	3Dh → AH
- File Name	DX
- Access Type	AL
Output Parameters:	
- Handler	AX
- Operation Result	Set/Clear CF – Carry Flag

WRITE into the FILE

Input Parameters:	Registers:
- Function Code	40h → AH
- File Handler	BX
- Pointer to the buffer that contains the data in RAM for writing into the file	DX
- Bytes number to be written into the file	CX
Output Parameters:	
- The number of bytes that have been written with success into the file	AX
- Operation Result	Set/Clear CF – Carry Flag

b) FILES in DOS

READ from the FILE

Input Parameters:	Registers:
- Function Code	**3Fh → AH**
- File Handler	**BX**
- pointer to the memory segment buffer which will contain the data from file	**DX**
- Bytes number to be read from the file	**CX**
Output Parameters:	
- The number of bytes that have been read with success from the file	**AX**
- Operation Result	Set/Clear CF – Carry Flag

SEEK/Positioning in the FILE

Input Parameters:	Registers:
- Function Code	**42h → AH**
- File Handler	**BX**
- Inside file reference (0-SEEK_SET; 1-SEEK_CURR; 2-SEEK_END)	**AL**
- The bytes number as offset related to the inside file reference (DWORD)	**inferior word → DX** **superior word → CX**
Output Parameters:	
- the new position in file (DWORD)	**inferior word → AX** **superior word → DX**
- Operation Result	Set/Clear CF – Carry Flag

I.11 Char Strings Operations

I.11.1 Char Strings Processing Steps

1. Init DS:SI with SRC – Source String offset
2. Init ES:DI with DST – Destination String offset
3. Init CX with the length of the processed strings. Should be the minim length between the source string – SRC and destination string – DST.
4. DF – Direction flag
   - 0 (CLD) – the strings are processed from the left to the right
   - 1 (STD) – the strings are processed from the right to the left

I.11 Char Strings Operations

I.11.2 Char Strings Mnemonics

1. movsb & movsw
2. cmpsb & cmpsw
3. lodsb & lodsw
4. stosb & stosw
5. scasb & scasw

I.11 Char Strings Operations

I.11.2 Char Strings Mnemonics

1. Move the source string to the destination string

- **movsb**
  - DF=0 (CLD)
  - ((DI))=((SI))
  - (SI)=(SI)+1
  - (DI)=(DI)+1

  - DF=1 (STD)
  - ((DI))=((SI))
  - (SI)=(SI)-1
  - (DI)=(DI)-1

- **movsw**
  - DF=0 (CLD)
  - ((DI))=((SI))
  - (SI)=(SI)+2
  - (DI)=(DI)+2

  - DF=1 (STD)
  - ((DI))=((SI))
  - (SI)=(SI)-2
  - (DI)=(DI)-2

I.11 Char Strings Operations

I.11.2 Char Strings Mnemonics

2. Compares through temporary subtracting dst vs. src

- **cmpsb**
  - DF=0 (CLD)
    - ((DI))-((SI))
    - (SI)=(SI)+1
    - (DI)=(DI)+1
  - DF=1 (STD)
    - ((DI))-((SI))
    - (SI)=(SI)-1
    - (DI)=(DI)-1
- **cmpsw**
  - DF=0 (CLD)
    - ((DI))-((SI))
    - (SI)=(SI)+2
    - (DI)=(DI)+2
  - DF=1 (STD)
    - ((DI))-((SI))
    - (SI)=(SI)-2
    - (DI)=(DI)-2

I.11 Char Strings Operations

I.11.2 Char Strings Mnemonics

3. Loading src in accumulator register

- lodsb
  - DF=0 (CLD)
  - (AL)=((SI))
  - (SI)=(SI)+1

  - DF=1 (STD)
  - (AL)=((SI))
  - (SI)=(SI)-1

- lodsw
  - DF=0 (CLD)
  - (AX)=((SI))
  - (SI)=(SI)+2

  - DF=1 (STD)
  - (AX)=((SI))
  - (SI)=(SI)-2

I.11 Char Strings Operations

I.11.2 Char Strings Mnemonics

4. Storing the accumulator register in dst

- **stosb**
  - DF=0 (CLD)
  - ((DI))=(AL)
  - (DI)=(DI)+1

  - DF=1 (STD)
  - ((DI))=(AL)
  - (DI)=(DI)-1

- **stosw**
  - DF=0 (CLD)
  - ((DI))=(AX)
  - (DI)=(DI)+2

  - DF=1 (STD)
  - ((DI))=(AX)
  - (DI)=(DI)-2

I.11 Char Strings Operations

I.11.2 Char Strings Mnemonics

5. Scanning string area through temporary subtracting of src from the accumulator register

- **scasb**
  - DF=0 (CLD)
    - (AL)-((SI))
    - (SI)=(SI)+1
  - DF=1 (STD)
    - (AL)-((SI))
    - (SI)=(SI)-1
- **scasw**
  - DF=0 (CLD)
    - (AX)-((SI))
    - (SI)=(SI)+2
  - DF=1 (STD)
    - (AX)-((SI))
    - (SI)=(SI)-2

I.11 Char Strings Operations

I.11.3 Samples

1. Copy a source char string into a destination char string
2. Compare if 2 char strings are the same – procedure

I.11 Char Strings Operations

I.11.3 Char Strings Samples

1. Copy a source char string into a destination char string
2. Compare if 2 char strings are the same – procedure

```
rep_label:
  mov AL, [SI]
  mov [DI], AL
  INC SI
  INC DI
loop rep_label
```

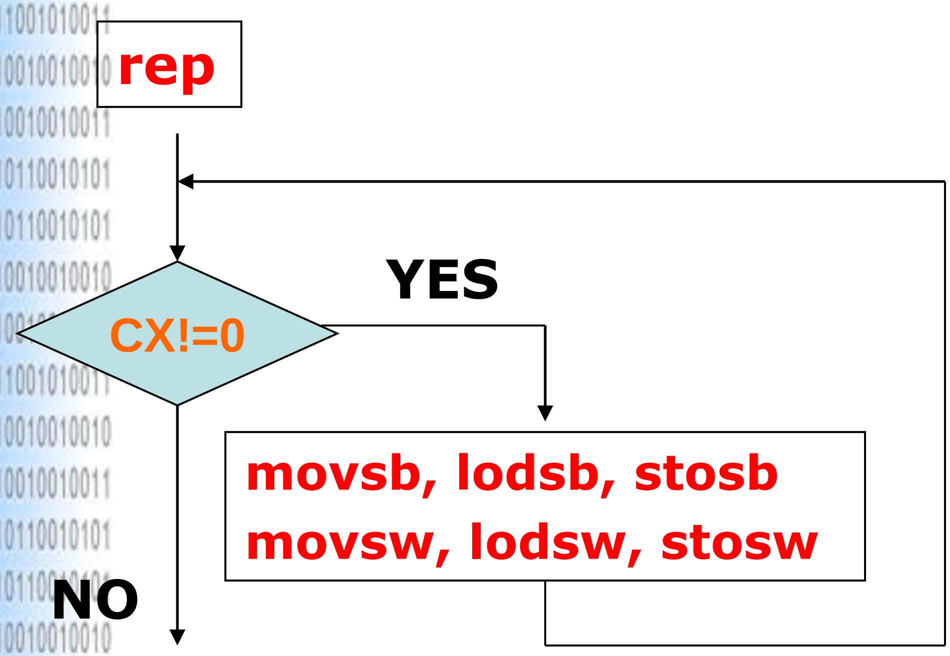
movsb

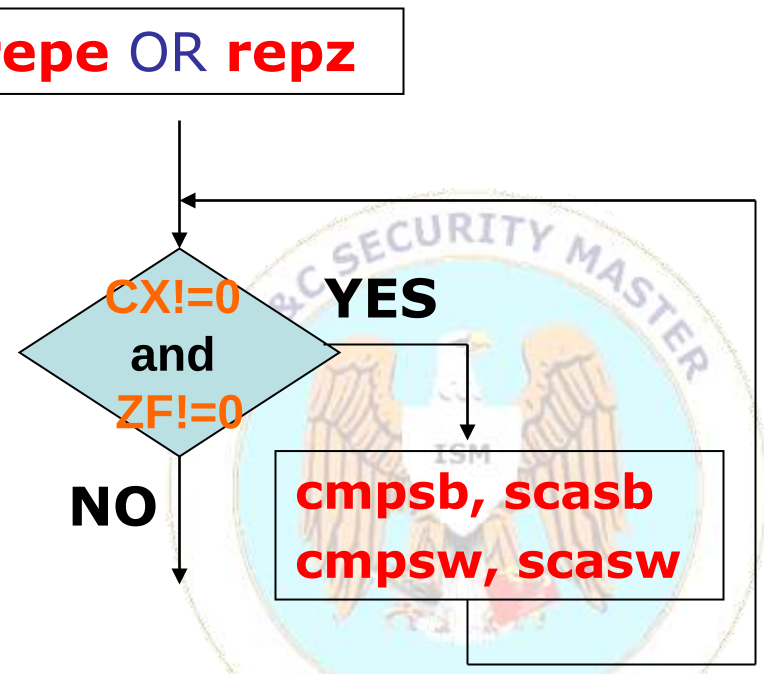
rep movsb

I.11 Char Strings Operations

I.11.3 Char Strings Samples

1. Copy a source char string into a destination char string
2. Compare if 2 char strings are the same – procedure





I.11 Char Strings Operations

I.11.3 Char Strings Sample

1. Copy a source char string into a destination char string

```
.model large

exit_dos MACRO
 mov ax,4c00h
 int 21h
ENDM

DATEprog SEGMENT
 src db 'Sirul meu sursa$'
 dimsrc dw $-src
 dst db '111111111111111$'
 dimdst dw $-dst
DATEprog ENDS
```

```
;no stack
SEGProg SEGMENT
 ASSUME DS:DATEProg,ES:DATEProg,CS:SEGProg
start:
   mov AX, SEG src
   mov DS,AX
   mov ES,AX
   cld
   mov SI, offset src
   mov DI, offset dst
   mov cx, dimsrc
   rep movsb
; ciclu:
   ;;movsb
   ;lodsb
   ;stosb
; loop ciclu
  exit_dos

SEGProg ENDS
end start
```

I.11 Char Strings Operations

I.11.3 Char Strings Sample

2. Compare if 2 char strings are the same – procedure

```
.model large

exit_dos MACRO
 mov ax,4c00h
 int 21h
ENDM

DATEprog SEGMENT
 src db 'Sirul meu sursa$',0
 dimsrc dw $-src
 dst db 'Sirul1111111111$',0
 dimdst dw $-dst
DATEprog ENDS

Stiva SEGMENT
   dw 10 dup(?)
   varf label word
Stiva ENDS
```

```
SEGProg SEGMENT
 ASSUME DS:DATEProg,ES:DATEProg,SS:Stiva,CS:SEGProg
start:
   mov AX, SEG src
   mov DS,AX
   mov ES,AX
   mov ax, Stiva
   mov ss,ax
   mov sp,varf
;FAR procedure with the result in DX=position of the first difference
;int compare1(char* s, char* d, short int length);
  mov ax, dimsrc
  push ax
  mov ax, offset dst
  push ax
  mov ax, offset src
  push ax
  ;PROCEDURE CALL
  CALL FAR PTR compare1
  exit_dos
SEGProg ENDS
**HERE is the procedures segment
end start
```

I.11 Char Strings Operations

I.11.3 Char Strings Sample

2. Compare if 2 char strings are the same – procedure

```
;**continue from the previous slide
Procedures SEGMENT
 ASSUME CS:Procedures

 compare1 PROC FAR
   push bp
   mov bp, sp
   ;draw the stack segment
   ;BP;IP;CS;adr-offset src;adr-offset dst;dimsrc
   ;     ;[bp+4]          ;[bp+8]        ;[bp+10]

   cld
   mov SI, [bp+6]
   mov DI, [bp+8]
   mov cx, [bp+10]

i_index:
     cmpsb
     jnz not_eq
loop i_index
  eqq:
   mov DX,0
   jmp sfarsitproc
  not_eq:
   mov DX,CX
  sfarsitproc:
   pop bp
   ret 6h
 compare1 ENDP
Procedures ENDS
```

DAY 3