Laborator 2

Paradigme de Programare

Introducere in Limbajul de Asamblare

Task 1: Obisnuirea cu mediul de lucru MASM 10 min sau se utilizeaza emu 8086 (atentie suporta numai asm specific 8086/8088)

La mana (se ignora pt emulatorul de 8086)

- `MASM /L FIRST.ASM`
- `LINK FIRST.OBJ`
- `FIRST.EXE`

Primul schelet

```
.286 ; se ignora pt emulatorul de 8086
.model tiny
;nu are date
.code
   org 100h
entry:
   jmp start

   ; codul de test

start:
   mov ax, 4c00h
   int 21h
end entry
```

Q: afiseaza correct? Ce mai trebuie?

afisarea unui text

```
INT 21 - DOS - PRINT STRING
        AH = 09h
        DS:DX = address of string terminated by "$"
Note:  Break checked, and INT 23h called if pressed
```

```
.MODEL Small
.STACK 100h
.DATA
MSG db 'Hello, world!$'
dym dd 10000
.CODE
start:
  mov ah, 09h
  lea dx, msg ; or mov dx, offset msg
  int 21h
  mov ax,4C00h
  int 21h
end start
```

exemplul 2

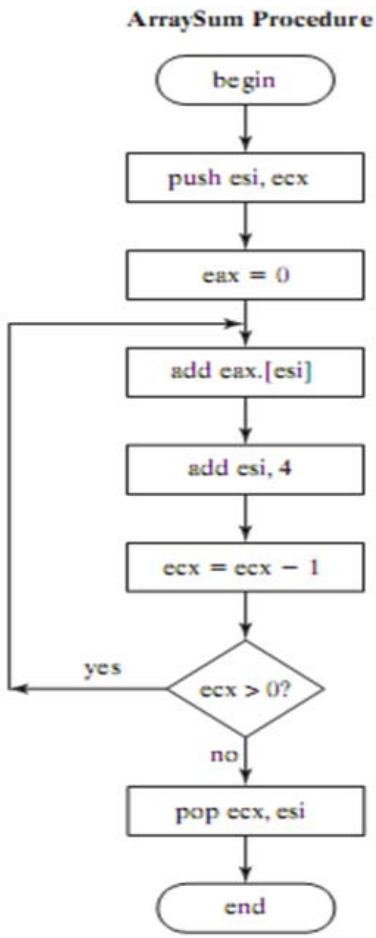
```
include emu8086.inc
ORG 100h
MOV AL, 25 ; set AL to 25.
MOV BL, 10 ; set BL to 10.
CMP AL, BL ; compare AL - BL.
JE equal ; jump if AL = BL (ZF = 1).
PUTC 'N' ; if it gets here, then AL <> BL, JMP stop ; so print 'N', and jump to stop.
equal: ; if gets here,
PUTC 'Y' ; then AL = BL, so print 'Y'.
stop:
RET ; gets here no matter what.
END
```

Task 2: Factorial pentru 286 masm (20 min)

Task 3: Sa se modifice exemplul din curs astfel incat sa sepoata face calcul lui $\sum_{i=0}^{n} i$ (20 min) (hint: faceti schema logica)

Task 4: Sa se implementeze programul conform schemei logice de mai jos (30 min)

Mode	Operand Type	Example	Comment
Register	Register	`inc bx`	This `inc`'s operand is a register
Immediate	Constant	`mov cx, 10`	This `mov`'s operands are register and immediate
Memory	Variable	`mov cx, [n]`	This `mov`'s operands are register and memory
Register Indirect	Pointer pointed by a register	`mov cx, [bx]`	This `mov`'s operands are register and register indirect
Base Relative	Pointer pointed by a register with an added index	`mov cx, [bx+1]`	This `mov`'s operands are register and base relative
Direct Indexed	Pointer pointed by an index register with an added index	`mov cx, [si+1]`	This `mov`'s operands are register and direct index
Base Indexed	Pointer pointed by a register and an index register	`mov cx, [bx+si]`	This `mov`'s operands are register and base indexed



Obs: aceasta este schema logica pentru o procedura (deci se pot ignora transferurile de parametric si folosi valori imediate. Pentru emulator trebuie reproiectata deoarece acesta nu suporta registry pe 32 de bitisori.

Exemplu de procedura

```
ORG 100h
MOV AL, 1
MOV BL, 2
CALL m2
CALL m2
CALL m2
CALL m2
RET ; return to operating system.

m2 PROC
MUL BL ; AX = AL * BL. RET ; return to caller. m2 ENDP
END
```

Sa se reia problema apeland calculul ca procedura de aceasta data

Task 5: Sa se implementeze ordonarea unui vector folosind swap din curs (20 min)

Reamintire vector

definire

```
my_arr db 5, 2, 8, 9, 1, 7, 3, 0, 4, 6
```

access element

```
mov al, [my_arr]
```

bazat-indexat

```
mov al, [my_arr+1]
mov al, [my_arr+2]
mov al, [my_arr+3]
```

bucla de procesare acces simplu

```
   :
   mov  bx, offset my_arr
   mov  cx, 10
@@myloop:
   mov  al, [bx]
   inc  bx           ; bx++
     :
     :   ; procesare
     :
   loop @@myloop
     :
```

bucla de procesare acces sbazat indexat (mai safe, nu pierd baza (din bx) ca in cazul anterior

```
   :
   mov  bx, offset my_arr
   sub  si, si
   mov  cx, 10
@@myloop:
   mov  al, [bx+si]
   inc  si           ; si++
     :
     :   ; fa ceva
     :
   loop @@myloop
     :
```

XCHG:

bubble sort clasic:

```
            :
        @@loop_j:
             mov   ah, [bx+si]  ; AH = arr_idx[i]
             mov   al, [bx+di]  ; AL = arr_idx[j]

             cmp   ah, al       ; if (arr_idx[i] <= arr_idx[j]) then no swap
```

```
        jle   @@noswap

        mov   [bx+si], al  ; else swap
        mov   [bx+di], ah

    @@no_swap:
        :
```

Buble sort cu e**xch**an**g**e

```
        :
    @@loop_j:
        mov   ah, [bx+si]  ; AH = arr_idx[i]

        cmp   ah, [bx+di]  ; if (arr_idx[i] <= arr_idx[j]) then no swap
        jle   @@noswap

        xchg  ah, [bx+di]  ; else swap

    @@no_swap:
        :
```

Tema pe ACASA : (indifferent daca este sub linux sau sub MS – la alegerea clientului) sa se implementeze o ordonare buble sort pentru nota 8 sau alt algoritm pentru nota 10

Anexa 1

Pentru cei care vor sa lucreze sub NASM (Linux)

Aveti si asamblorul pt win in dir aplicatiei

Tema ramane la fel. Pentru testarea interoperabilitatii cu C-ul

> nasm -f <format> <filename> [-o <output>] [-l listing]

Exemplu:

> nasm -f elf mytry.s -o myelf.o

myelf.o care ste in format elf format (executabil si link-abil).
Se va folosi si fisierul main.c (scris in C language) pentru a porni programul sau pentru a realiza un I/O mai usor de implementat cu utilizatorul

Pentru a compila :

> cc main.c myelf.o -o myexe.out –l mylist.lst

Se va crea un executabil myexe.out si un fisier cu informati mylist.lst.
Pentru executie:

> myexe.out

Anexa 2 – codul ASCII

Codul standard ASCII defineste 128 de coduri pentru caractere (adica reprezentarile grafice associate) (in intervalul [0, 127], din care primele 32 sunt coduri de control ((ne tiparibile) folosite in controlul tiparirii effective pentru primele imprimante) iar celelalte 96 sunt vizibile:

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Pentru a afla codul asociat se ia direct combinatia de linie si coloana. De exemplu in cazul lui A observam ca se afla pe linia 4 si coloana 1 rezulta deci un cod asociat (in HEXA) **0x41**.

In cazul in care conversia manuala intre baze este prea complicata pentru student, se poate folosi un tool existent la http://www.cplusplus.com/doc/papers/ascii.html

Pentru oameni disperati: http://frz.ir/dl/tuts/8086_Assembly.pdf