8086 exercises

Saturday, February 19, 2022 11:35 AM

```
; sum two matrices of bytes, and store in a word

.model small
.stack
.data
mata db 1,2,3,4, 5,6,7,8 ,9,10,11,12, 13,14,15,16
matb db 2,3,4,5, 6,7,8,9, 10,11,12,13, 14,15,16,17

dim equ 16
matc dw dim dup(?)

.code
.startup
;init to zero
xor si,si ;iterator for a and b
xor di,di ;iterator for result
xor ax,ax ;for a
xor bx,bx ;for b
xor dx,dx ;for result

cycle:
mov al,mata[si] ;db so storing in al
add dx,ax         ;adding ax bcz res is in dw and we want to store dx
mov bl,matb[si] ;same as above
add dx,bx

mov matc[di],dx ;res[i] updated

inc si            ;si inc by 1 bcz db
add di,2          ;di inc by 2 bcz dw
xor dx,dx         ;dx zero bcz new sum in each iteration

cmp si,dim        ;loop exit condition
jne cycle

;al matA[i]
;cl matB[i]
;dx ax+bx


.exit
end
```

```
variables
size: byte   elements: 16
edit   show as: signed
MATA    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
MATB    2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
MATC    3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33
```

```
;sum only negative elements in an array
.model small
.stack
.data
src db 1,2,-3,4, -5,6,7,-8, 9,10,11,12, 13,14,15,16
res dw ?
dim equ 16

.code
.startup
 xor si,si    ; src iterator
 xor ax,ax    ; sum stored here
 xor cx,cx    ; loop counter stored here

 mov ah,0xff ; higher bits negative

 mov cx,dim  ; loop 16 times

 cycle:
 cmp src[si],0  ; if element is negative
 jg  skip       ; don't add

 add al,src[si] ; else, add
 skip:
 inc si         ; get next element in src
 loop cycle

 mov res,ax     ; mov result to mem

 .exit
 end
```

```
variables
size: word   elements: 1
edit   show as: signed
SRC     1, 2, -3, 4, -5, 6, 7, -8, 9, 10, 11, 12, 13, 14, 15, 16
RES     -16
```

```
.MODEL small
.STACK
.DATA

SOURCE   DW  10, 20, 100,    10000
    DW  0,  7000,1   ,    2
    DW  9000,12345,999,30000
    DW  200,210,7,65000

MAPP   DB  16  DUP(?)

CROSS   DW  1   DUP(?)

.CODE
.STARTUP

MOV CX,016      ;to repeat loop 16 time for all elements
MOV DX,0FF00h
MOV BX,0

CYCLE:
    MOV AX,SOURCE[SI]  ;put an element of SOURCE in the AX
    AND AX,DX      ;zero all the lower 8 bits of AX

    MOV MAPP[DI],0   ;put 0 in the corresponding element of MAP
    CMP AX,0       ;compare if the element in AX is Zero, it will be zero if the
    JNE skip       ;jump if AX  is not zero
    MOV MAPP[DI],01  ;other wise it can be represented in 8 bits, mov 1 into co
    ADD BX,SOURCE[SI]  ;also add in BX the corresponding element of SOURCE
    skip:
    INC DI         ; INC DI by one (BYTES)
    ADD SI,02      ; ADD 2 to SI (WORD)
loop    CYCLE
               ;STORE    value in C variable of sum
MOV CROSS,BX

;we do not risk an overflow of CROSS because if we consider all entries to
;be 1 of MAPP and in source all entries are 255 (max value in 8 bits) then
; we get 255*16=4080 which can be easily represented in 16 bit word and we
;store it in a word.

.EXIT
END
```

```
variables
size: byte   elements: 16
edit   show as: unsigned
SOURCE  10, 20, 100, 10000, 0, 7000, 1, 2, 9000, 12345, 999, 30000, 200, 210, 7, 65000
MAPP    1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
CROSS   0
```

Given the 4 x 4 matrix SOURCE of words storing only positive data (represented in pure binary on 16 bits), write an 8086 assembly program, which computes the entries of another 4 x 4 matrix MAPP of bytes, according to the following very simple rule:

If SOURCE (i, j) can be represented on 8 bits only, then MAPP (i,j) = 1

Otherwise MAPP (i,j) = 0

- The same program should also compute and store to the variable CROSS (on 16 bits) the sum of all the elements of SOURCE whose corresponding entry in MAPP is equal to 1.
- In your solution, please provide the declaration of SOURCE, MAPP, and CROSS and the code, together with significant comments to the code and instructions.
- Indeed, the choice is yours about how to store the matrices in the memory.

If you have time, in order to get up to one additional point, please also clearly and shortly respond to the following questions (as "comments in the program"): do we risk an overflow for CROSS? Why? Please consider that a wrong response to these optional questions will imply a negative score up to -1.

Write your code in a file saved in the 8086 folder.

Example:

Initial matrix SOURCE

10	20	100	10000
0	7000	1	2
9000	12345	999	30000
200	210	7	65000

```
.model small
.stack

.data

src dw 'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p'
dest dw 16 dup(?)

shift dw ?

n equ 2
dim equ 16

.code
.startup

xor bx,bx        ; temp storage register init to 0
xor si,si        ; src index init to 0
mov cx,dim       ; dim for restarting the index in case of overflow

mov shift,n      ; user defined n

shl shift,3      ; n x 8 bcz single step is a shift of 4 places i.e 8 for DW

cycle:

mov bx,src[si]   ; bx=src[i]
mov di,si        ; di used for iterating over dest
add di,shift     ; adding a shift in di for rotation

cmp di,31        ; checking overflow

jle inRange      ; if no overflow, jump to line 25

sub di,32        ; else sub from 32 to restart the index from 0

inRange:

mov dest[di],bx ; dest[i+shift] = src[i]
add si,2         ; inc si to get the next element from src

loop cycle

.exit
end
```

```
variables
size: word   elements: 16
edit   show as: ascii
SRC     'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p'
DEST    'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'
SHIFT   16
```

2021.02.03

Given a 4 x 4 matrix of WORD (i.e. 16 bits single data) SOURCE write a 8086 assembly program which rotates the rows of SOURCE from up to down by 1<=n<=3 positions and stores the result in the matrix DESTINATION, with n given by the user. The choice is yours about how to store the matrices in the memory. Please add significant comments to the code and instructions. If you have time, in order to get one additional point, provide the instructions to extend the program to consider n in the range -80<=n<=+120

Example:

Initial matrix SOURCE

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

if n=3 DESTINATION becomes

E	F	G	H
I	J	K	L
M	N	O	P
A	B	C	D

```
; dst[i]=src[i]+src[i+3]

.model small
.stack
.data

src db 1,2,3, 4,5,6, 7,8,9
dim equ 9
dst dw dim dup(?)

.code
.startup

xor si,si
xor di,di
xor ax,ax
xor bx,bx
xor cx,cx

mov cx,dim
cycle:

add al,src[si]
;push si          ; save index
mov bx,si
add si,3         ; get element under it

cmp si,9
jl cont

sub si,9       ; reseting in case of overflow
cont:
add al,src[si] ; add this element

mov dst[di],ax
xor ax,ax        ; empty summing register

;pop si           ; restore index for iteration
mov si,bx
inc si
add di,2         ; next place for di

loop cycle

;computing the sum in ax and moving it to dst
.exit
end
```

```
variables
size: word   elements: 9
edit   show as: unsigned
SRC     1, 2, 3, 4, 5, 6, 7, 8, 9
DST     5, 7, 9, 11, 13, 15, 8, 10, 12
```

Given a 3 x 3 matrix of bytes SOURCE representing unsigned numbers, write a 8086 assembly program which computes (in circular buffer mode) the addition of each row element with the corresponding same column element in the row immediately below and stores the result on 16 bits in the same position of a matrix DESTINATION. The last row elements do add up with the corresponding first row elements (i.e. circular buffer mode). Please add significant comments to the code and instructions.

Example:

Initial matrix SOURCE

1	2	3
4	5	6
7	8	9

the following matrix DESTINATION is computed

5	7	9
11	13	15
8	10	12

```
.model small
.stack
.data
src db 1,2,3,4,5, 6,7,8,9,0, 9,8,7,6,5, 4,3,2,1,0, 7,7,7,7,7
;src dw 1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1

dim equ 25  ;DECODE THIS FFS
res dw ?
.code
.startup

xor si,si
xor di,di
xor ax,ax
xor bx,bx
xor cx,cx
xor dx,dx

mov cx,dim       ; 25 iterations
cycle:
cmp si,dx        ; check if current element in source is diagonal element
je diagonal      ; if diagonal, save this for minus

add al,src[si]   ; else, add this
jmp cont

diagonal:
add bl,src[si]   ; sum of diagonal elements
add dx,6

cont:
inc si
loop cycle

mov res,ax
sub res,bx       ; non-diag - diag = result

;cx for loop
;ax for saving the sum of non-diagonal elements
;bx for saving the sum of diagonal elements
;dx is the multiple of 6 i.e the index of next diagonal element
;si is iterating over src mat
.exit
end
```

```
variables
size: byte   elements: 25
edit   show as: unsigned
SRC     1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 7, 7, 7, 7, 7
RES     79
```

Given a 5 x 5 matrix of bytes SOURCE representing unsigned numbers, write a 8086 assembly program which computes on 16 bits the sum of all cells excluding these on the main diagonal, i.e. upper left-to-lower-right diagonal, minus the sum of all the cells of the same main diagonal.

Please add significant comments to the code and instructions.

Friendly advice: before starting to write down the code, think at a possible (very) simple algorithm! The choice of the algorithm highly influences the complexity and length of the code.

Example:

matrix SOURCE

1	2	3	4	5
6	7	8	9	0
9	8	7	6	5
4	3	2	1	0
7	7	7	7	7

all cells excluding the main diagonal:

2+3+4+5+

6+8+9+0+

9+8+6+5+

4+3+2+0+

7+7+7+7= 102

all cells on the main diagonal

1+

7+

7+

1+

7= 23

Result (on 16 bits in two's complement) = 102-23 = 79

```
.model small
.stack
.data
src db 1,2,3,4,5, 6,7,8,9,0, 9,8,7,6,5, 4,3,2,1,0, 7,7,7,7,7, 3,5,7,9,0, 8,7,
res dw 1 DUP(?)
dim equ 40

.code
.startup
;initializing
xor si,si     ; even ptr
xor ax,ax     ; even sum
xor bx,bx     ; odd sum
xor cx,cx     ; loop

mov cx,dim
cycle:
add al,src[si] ;even
inc si

add bl,src[si] ;odd
inc si

cmp si,dim

je  cont

loop cycle
cont:
mov res,ax
sub res,bx
.exit
end
```

```
variables
size: word   elements: 1
edit   show as: signed
SRC     1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 7, 7, 7, 7, 7, 3, 5, 7,
RES     19
```

Given a 8 x 5 matrix of bytes SOURCE representing unsigned numbers, write a 8086 assembly program which computes on 16 bits (two's complement) the addition of all cells with indexes (i,j) where i+j is an even value, minus all the cells whose i+j is an odd value. Please consider that i ranges from 0 to 7 and j ranges from 0 to 4.

Please add significant comments to the code and instructions.

Friendly advice: before starting to write down the code, think at a possible (very) simple algorithm! The choice of the algorithm highly influences the complexity and length of the code.

Example:

matrix SOURCE

1	2	3	4	5
6	7	8	9	0
9	8	7	6	5
4	3	2	1	0
7	7	7	7	7
3	5	7	9	0
8	7	6	5	4
9	9	9	3	2

the cells with i+j even are added up, while the cells with i+j odd are subtracted

1+3+5+7+9+9+ …

-2-4-6-8-0-….

The result will be clearly on 16 bits in two's complement.