CSE341: Microprocessor LAB
Assembly Language
emu8086 Emulator Problem-Solution

#Use of General Purpose Registers (AX, BX, CX, DX)
#Use of MOV, ADD, SUB, MUL, DIV, NEG, DEC, INC operations

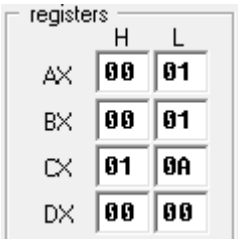
Task 01
Take input in the register AX, and then move it to BX using the MOV instruction.

Solution

```
.model small
.stack 100h
.data

.code
main proc
    mov ax, @data
    mov ds, ax

    mov ax, 1
    mov bx, ax
```

```
01 .model small            ; Defines the memory model as 'small', meaning both code and data fit within 64KB each.
02 .stack 100h             ; Reserves 256 bytes (100h) for the stack.
03 .data                   ; Start of the data segment.
04
05 .code                   ; Start of the code segment.
06 main proc               ; Beginning of the main procedure (entry point of the program).
07     mov ax, @data       ; Load the address of the data segment into AX.
08     mov ds, ax          ; Move the value in AX into DS to initialize the data segment register.
09
10     mov ax, 1           ; Load the value 1 into the AX register.
11     mov bx, ax          ; Copy the value of AX into BX (now BX = 1).
12
```

Result:



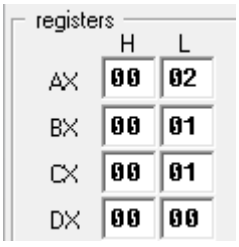
Task 02
Swap two numbers, using a maximum of 3 registers.
Hint: Use the MOV instruction.

Solution

```
.model small
.stack 100h
.data

.code
main proc
   mov ax, @data
   mov ds, ax

   mov ax, 1
   mov bx, 2
   mov cx, ax
   mov ax, bx
   mov bx, cx
```

```
01 .model small
02 .stack 100h
03 .data
04
05 .code
06 main proc
07     mov ax, @data
08     mov ds, ax
09
10     mov ax, 1        ; Loads the value 1 into register AX.
11     mov bx, 2        ; Loads the value 2 into register BX.
12     mov cx, ax       ; Copies the value in AX (which is 1) into CX.
13     mov ax, bx       ; Copies the value in BX (which is 2) into AX.
14     mov bx, cx       ; Copies the value in CX (which is 1) into BX.
```

Result



Task 03

Add two numbers using two registers.

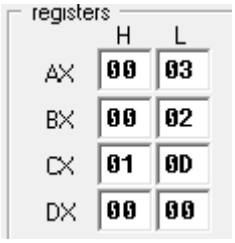
Solution

```
.model small
.stack 100h
.data
```

```
.code
main proc
   mov ax,@data
   mov ds,ax

   mov ax,1
   mov bx,2

   ;adding ax with bx
   add ax,bx  ;ax = ax + bx
```

```
01 .model small
02 .stack 100h
03 .data
04
05 .code
06 main proc
07     mov ax, @data
08     mov ds, ax
09
10     mov ax, 1        ; Loads the value 1 into the AX register.
11     mov bx, 2        ; Loads the value 2 into the BX register.
12
13     ; Adding AX with BX
14     add ax, bx       ; Adds the value in BX (2) to the value in AX (1).
15                      ; Now AX = AX + BX = 1 + 2 = 3
16
```

Result



Task 04

Subtract two numbers using two registers. Do you always get the correct answer?
What happens when you subtract larger number from the smaller one?

Solution

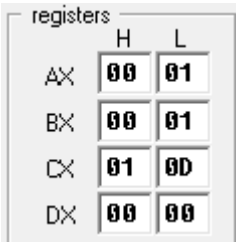
#Subtracting small number from a larger number:

```
.model small
.stack 100h
.data
.code
main proc
   mov ax ,@data
   mov ds, ax
```

```
mov ax, 1
mov bx, 2

;subtracting ax (small) from bx (large)
sub bx, ax  ;bx = bx - ax = 2 - 1 = 1
```

```
01  .model small
02  .stack 100h
03  .data
04  .code
05  main proc
06      mov ax ,@data
07      mov ds, ax
08
09      mov ax, 1
10      mov bx, 2
11
12      ;subtracting ax (small) from bx (large)
13      sub bx, ax  ;bx = bx - ax = 2 - 1 = 1
14
```

Result

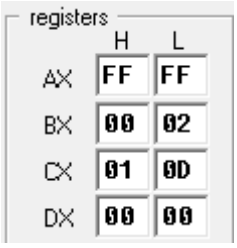


#Subtracting a lager number from a small number:

```
.model small
.stack 100h
.data

.code
main proc
    mov ax, @data
    mov ds, ax

    mov ax, 1
    mov bx, 2

    ;subtracting bx (large) from ax (small)
    sub ax, bx  ;ax = ax - bx = 1 - 2 = -1
```

```
01  .model small
02  .stack 100h
03  .data
04
05  .code
06  main proc
07      mov ax, @data
08      mov ds, ax
09
10      mov ax, 1
11      mov bx, 2
12
13      ;subtracting bx (large) from ax (small)
14      sub ax, bx   ;ax = ax - bx = 1 - 2 = -1
15
```

Result



Task 05

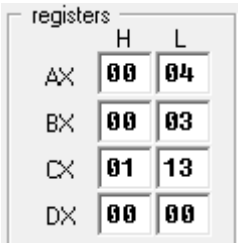
Swap two numbers using ADD/SUB instructions only.

Solution

```
.model small
.stack 100h
.data

.code
main proc
    mov ax, @data
    mov ds, ax

    mov ax, 3
    mov bx, 4

    add ax, bx   ;ax = 3 + 4 = 7
    sub bx, ax   ;bx = 4 - 7 = -3
    neg bx       ;bx = 3
    sub ax, bx   ;ax = 7 - 3 = 4
```

```
01 .model small
02 .stack 100h
03 .data
04
05 .code
06 main proc
07     mov ax, @data
08     mov ds, ax
09
10     mov ax, 3
11     mov bx, 4
12
13     add ax, bx    ;ax = 3 + 4 = 7
14     sub bx, ax    ;bx = 4 - 7 = -3
15     neg bx        ;bx = 3
16     sub ax, bx    ;ax = 7 - 3 = 4
```

Result



Task 06

Perform the following arithmetic instructions. A, B, C are three variables to be declared beforehand

1. A = B - A
2. A = -(A + 1)
3. C = A + (B + 1); use inc
4. A = B – (A – 1); use dec

Solutions

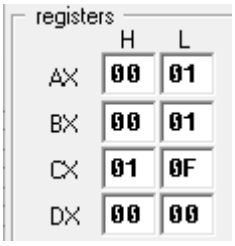
1. **A = B - A**

```
.model small
.stack 100h
.data

.code
main proc
   mov ax, @data
   mov ds, ax

   mov ax, 3  ;A = 3
```

```
mov bx, 4  ;B = 4

sub bx, ax ;B = B - A
mov ax, bx ;A = B - A
```

```
01  .model small
02  .stack 100h
03  .data
04
05  .code
06  main proc
07      mov ax, @data
08      mov ds, ax
09
10      mov ax, 3   ;A = 3
11      mov bx, 4   ;B = 4
12
13      sub bx, ax ;B = B - A
14      mov ax, bx ;A = B - A
```

Result

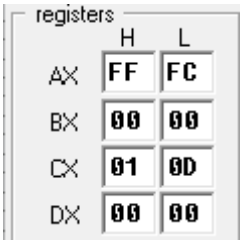


2. A = -(A + 1)

```
.model small
.stack 100h
.data

.code
main proc
   mov ax, @data
   mov ds, ax

   mov ax, 3  ;A = 3

   add ax, 1  ;A = A + 1 = 3 + 1 = 4
   neg ax     ;A = -(A + 1) = -4
```

```
01 .model small
02 .stack 100h
03 .data
04
05 .code
06 main proc
07     mov ax, @data
08     mov ds, ax
09
10     mov ax, 3   ;A = 3
11
12     add ax, 1   ;A = A + 1 = 3 + 1 = 4
13     neg ax      ;A = -(A + 1) = -4
```

Result

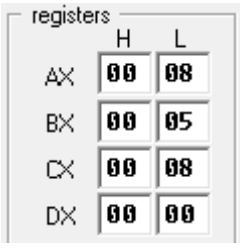


3. C = A + (B + 1); use inc

```
.model small
.stack 100h
.data

.code
main proc
   mov ax, @data
   mov ds, ax

   mov ax, 3  ;A = 3
   mov bx, 4  ;B = 4

   inc bx     ;B = B + 1 = 5
   add ax, bx ;A = A + (B + 1) = 3 + 5 = 8
   mov cx, ax ;C = A + (B + 1) = 8
```

```
01 .model small
02 .stack 100h
03 .data
04
05 .code
06 main proc
07     mov ax, @data
08     mov ds, ax
09
10     mov ax, 3   ;A = 3
11     mov bx, 4   ;B = 4
12
13     inc bx      ;B = B + 1 = 5
14     add ax, bx ;A = A + (B + 1) = 3 + 5 = 8
15     mov cx, ax ;C = A + (B + 1) = 8
```

Result

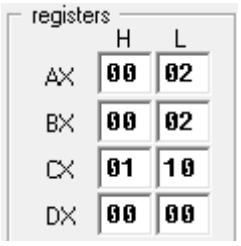


4. A = B – (A – 1); use dec

```
.model small
.stack 100h
.data

.code
main proc
   mov ax, @data
   mov ds, ax

   mov ax, 3  ;A = 3
   mov bx, 4  ;B = 4

   dec ax     ;A = A - 1 = 2
   sub bx, ax ;B = B - (A - 1) = 4 - 2 = 2
   mov ax, bx ;A = B - (A - 1) = 2
```

```
01  .model small
02  .stack 100h
03  .data
04
05  .code
06  main proc
07      mov ax, @data
08      mov ds, ax
09
10      mov ax, 3   ;A = 3
11      mov bx, 4   ;B = 4
12
13      dec ax      ;A = A - 1 = 2
14      sub bx, ax  ;B = B - (A - 1) = 4 - 2 = 2
15      mov ax, bx  ;A = B - (A - 1) = 2
```

Result



Task 07

Perform the following arithmetic operations

1. X * Y
2. X / Y
3. X * Y / Z

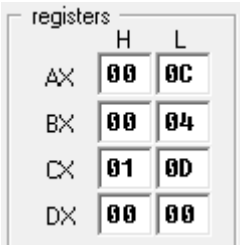
Soultions

1. **X * Y**

```
.model small
.stack 100h
.data

.code
main proc
   mov ax, @data
   mov ds, ax

   mov ax, 3  ;X = 3
   mov bx, 4  ;Y = 4

   mul bx     ;X = X * Y
```

```
01 .model small
02 .stack 100h
03 .data
04
05 .code
06 main proc
07     mov ax, @data
08     mov ds, ax
09
10     mov ax, 3   ;X = 3
11     mov bx, 4   ;Y = 4
12
13     mul bx      ;X = X * Y
```

Result

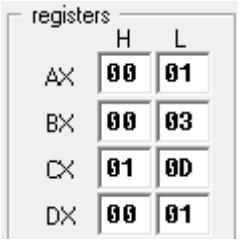


2. X / Y

```
.model small
.stack 100h
.data

.code
main proc
    mov ax, @data
    mov ds, ax

    mov ax, 4  ;X = 4
    mov bx, 3  ;Y = 3

    div bx
```

```
01 .model small
02 .stack 100h
03 .data
04
05 .code
06 main proc
07     mov ax, @data
08     mov ds, ax
09
10     mov ax, 4   ;X = 4
11     mov bx, 3   ;Y = 3
12
13     div bx      ;ax = ax / bx = 4 / 3 = 1(quotient)
14                 ;dx = 1(remainder)
15
```

Result

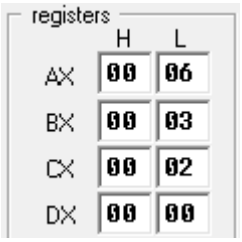


3. X * Y / Z

```
.model small
.stack 100h
.data

.code
main proc
   mov ax, @data
   mov ds, ax

   mov ax, 4  ;X = 4
   mov bx, 3  ;Y = 3
   mov cx, 2  ;Z = 2

   mul bx     ;ax = ax * bx = 4 * 3 = 12
   div cx     ;ax = ax / cx = 12 / 2 = 6(quotient)
              ;dx = 0(remainder)
```

```
01 .model small
02 .stack 100h
03 .data
04
05 .code
06 main proc
07     mov ax, @data
08     mov ds, ax
09
10     mov ax, 4   ;X = 4
11     mov bx, 3   ;Y = 3
12     mov cx, 2   ;Z = 2
13
14     mul bx      ;ax = ax * bx = 4 * 3 = 12
15     div cx      ;ax = ax / cx = 12 / 2 = 6(quotient)
16                 ;dx = 0(remainder)
17
18
```

Result



Task 08

Perform the following arithmetic operations

1. 236DF * AF
2. 8A32F4D5 / C9A5
3. CA92 * BAF9
4. C2A2 * ABCD / BED

Solutions

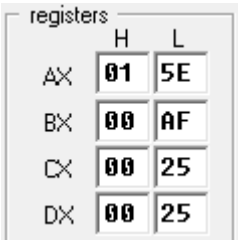
1. 236DF * AF

```
.model small
.stack 100h
.data

.code
main proc
   mov ax, @data    ; Initialize data segment
   mov ds, ax
```

```
; Load the lower and upper parts of 236DFh
mov ax, 36DFh    ; Load lower 16 bits (36DF) into AX
mov dx, 2        ; Load upper part (2) into DX

mov bx, 0AFh     ; Load AF into BX

; Perform the multiplication of lower part (36DF) with AF
mul bx           ; Multiply AX by BX, result is in DX:AX (for lower part only)

; Save result of lower multiplication
mov cx, dx       ; Store DX result of 36DF * AF in CX
mov dx, 0        ; Clear DX for the next multiplication

; Multiply the upper part (2) with AF
mov ax, 2        ; Load upper part again (2) into AX
mul bx           ; Multiply AX (2) by BX (AF)

; Add the results to form the full 32-bit result
add dx, cx       ; Add saved higher part of 36DF * AF to DX
; DX:AX now contains the final 32-bit result of 236DF * AF

; Exit
```

```
01 .model small
02 .stack 100h
03 .data
04
05 .code
06 main proc
07     mov ax, @data     ; Initialize data segment
08     mov ds, ax
09
10     ; Load the lower and upper parts of 236DFh
11     mov ax, 36DFh     ; Load lower 16 bits (36DF) into AX
12     mov dx, 2         ; Load upper part (2) into DX
13
14     mov bx, 0AFh      ; Load AF into BX
15
16     ; Perform the multiplication of lower part (36DF) with AF
17     mul bx            ; Multiply AX by BX, result is in DX:AX (for lower part only)
18
19     ; Save result of lower multiplication
20     mov cx, dx        ; Store DX result of 36DF * AF in CX
21     mov dx, 0         ; Clear DX for the next multiplication
22
23     ; Multiply the upper part (2) with AF
24     mov ax, 2         ; Load upper part again (2) into AX
25     mul bx            ; Multiply AX (2) by BX (AF)
26
27     ; Add the results to form the full 32-bit result
28     add dx, cx        ; Add saved higher part of 36DF * AF to DX
29     ; DX:AX now contains the final 32-bit result of 236DF * AF
30
31     ; Exit
```

Result

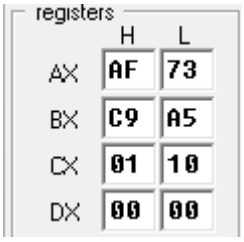


2. 8A32F4D5 / C9A5

```
.model small
.stack 100h
.data

.code
main proc
    mov ax, @data    ; Initialize data segment
    mov ds, ax

    ; Load the 32-bit dividend 8A32F4D5 into DX:AX
    mov dx, 8A32h    ; Load higher 16 bits (8A32) into DX
    mov ax, 0F4D5h   ; Load lower 16 bits (F4D5) into AX

    ; Load the divisor C9A5 into BX
    mov bx, 0C9A5h   ; Load divisor into BX

    ; Perform the division
    div bx          ; Divide DX:AX by BX
                    ; Quotient will be in AX, Remainder will be in DX

    ; Exit
```

```
01 .model small
02 .stack 100h
03 .data
04
05 .code
06 main proc
07     mov ax, @data    ; Initialize data segment
08     mov ds, ax
09
10     ; Load the 32-bit dividend 8A32F4D5 into DX:AX
11     mov dx, 8A32h    ; Load higher 16 bits (8A32) into DX
12     mov ax, 0F4D5h   ; Load lower 16 bits (F4D5) into AX
13
14     ; Load the divisor C9A5 into BX
15     mov bx, 0C9A5h   ; Load divisor into BX
16
17     ; Perform the division
18     div bx           ; Divide DX:AX by BX
19                      ; Quotient will be in AX, Remainder will be in DX
20
21     ; Exit
```

Result

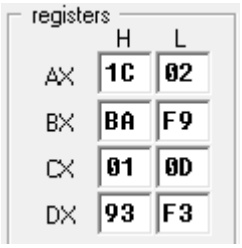


3. CA92 * BAF9

```
.model small
.stack 100h
.data

.code
main proc
    mov ax, @data    ; Initialize data segment
    mov ds, ax

    ; Load the multiplicands CA92 and BAF9
    mov ax, 0CA92h   ; Load CA92 into AX
    mov bx, 0BAF9h   ; Load BAF9 into BX

    ; Perform the multiplication
    mul bx          ; Multiply AX by BX, result is in DX:AX
                    ; DX:AX now contains the 32-bit result of CA92 * BAF9

    ; Exit
```

```
01 .model small
02 .stack 100h
03 .data
04
05 .code
06 main proc
07     mov ax, @data    ; Initialize data segment
08     mov ds, ax
09
10     ; Load the multiplicands CA92 and BAF9
11     mov ax, 0CA92h   ; Load CA92 into AX
12     mov bx, 0BAF9h   ; Load BAF9 into BX
13
14     ; Perform the multiplication
15     mul bx           ; Multiply AX by BX, result is in DX:AX
16                      ; DX:AX now contains the 32-bit result of CA92 * BAF9
17
18     ; Exit
19
```

Result

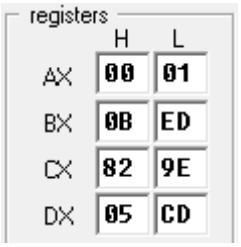


4. C2A2 * ABCD / BED

```
.model small
.stack 100h
.data

.code
main proc
    mov ax, @data        ; Initialize data segment
    mov ds, ax

    ; Step 1: Load the multiplicands C2A2h and ABCDh
    mov ax, 0C2A2h       ; Load C2A2 into AX (lower 16 bits)
    mov bx, 0ABCDh       ; Load ABCD into BX

    ; Perform the multiplication of C2A2h * ABCDh
    mul bx               ; AX = lower 16 bits of C2A2 * ABCD, DX = upper 16 bits of result

    ; Save the result of multiplication (C2A2 * ABCD) in DX:AX
    mov cx, dx           ; Store upper 16 bits of result in CX
    mov dx, 0            ; Clear DX to prepare for division

    ; Step 2: Load the divisor BEDh
    mov bx, 0BEDh        ; Load BED into BX (divisor)

    ; Step 3: Perform the division (DX:AX / BX)
    div bx               ; DX:AX (32-bit number) divided by BX
                         ; AX = quotient, DX = remainder

    ; Exit the program
```

```
.model small
.stack 100h
.data

.code
main proc
    mov ax, @data        ; Initialize data segment
    mov ds, ax

    ; Step 1: Load the multiplicands C2A2h and ABCDh
    mov ax, 0C2A2h       ; Load C2A2 into AX (lower 16 bits)
    mov bx, 0ABCDh       ; Load ABCD into BX

    ; Perform the multiplication of C2A2h * ABCDh
    mul bx               ; AX = lower 16 bits of C2A2 * ABCD, DX = upper 16 bits of result

    ; Save the result of multiplication (C2A2 * ABCD) in DX:AX
    mov cx, dx           ; Store upper 16 bits of result in CX
    mov dx, 0            ; Clear DX to prepare for division

    ; Step 2: Load the divisor BEDh
    mov bx, 0BEDh        ; Load BED into BX (divisor)

    ; Step 3: Perform the division (DX:AX / BX)
    div bx               ; DX:AX (32-bit number) divided by BX
                         ; AX = quotient, DX = remainder

    ; Exit the program
```

Result



Task 09

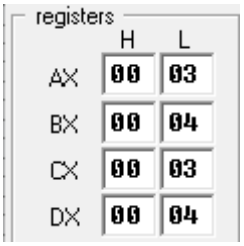
Write two examples for each combination of registers possible for the 'mov' instruction.
Hint: See the table above to see all the possible combinations.

Solution

```
.model small
.stack 100h
.data

.code
main proc
    mov ax, @data
    mov ds, ax

    mov ax, 3
    mov bx, 4

    mov cx, ax
    mov dx, bx
```

```
01  .model small
02  .stack 100h
03  .data
04
05  .code
06  main proc
07      mov ax, @data
08      mov ds, ax
09
10      mov ax, 3
11      mov bx, 4
12
13      mov cx, ax
14      mov dx, bx
15
```

Result



Task 10

Write two examples for each combination of registers possible for the 'add' and 'sub' instructions.
Hint: See the table above to see all the possible combinations.

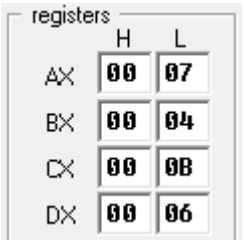
Solution

```
.model small
.stack 100h
.data

.code
main proc
    mov ax, @data
    mov ds, ax

    mov ax, 3
    mov bx, 4
    mov cx, 5
    mov dx, 6

    add ax, bx
```

add cx, dx

```
01  .model small
02  .stack 100h
03  .data
04
05  .code
06  main proc
07      mov ax, @data
08      mov ds, ax
09
10      mov ax, 3
11      mov bx, 4
12      mov cx, 5
13      mov dx, 6
14
15      add ax, bx
16      add cx, dx
17
```

Result



Task 11

Perform the following arithmetic operation: (1 + 2) * (3 – 1) / 5 + 3 + 2 – (1 * 2)

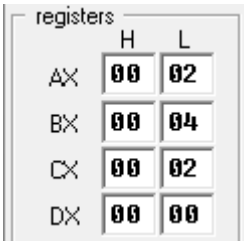
Solution

```
.model small
.stack 100h
.data

.code
main proc
    mov ax, @data
    mov ds, ax

    mov ax, 2
    inc ax        ;ax = ax + 1 = (1+2)=3
    mov bx, 3
    dec bx        ;bx = bx - 1 = (3-1)=2
```

```
mul bx        ;ax = ax * bx = (1+2)*(3-1)=6
mov bx, 5
div bx        ;ax = ax / bx = (1+2)*(3-1)/5=1
              ;mul is done before div, violating BODMAS
              ;can't use dx register to store divisor(5)
              ;since dx is used to store the remainder
mov bx, ax    ;bx = ax = (1+2)*(3-1)/5=1
mov ax, 1
mov cx, 2
mul cx        ;ax = ax * 2 = (1*2)=2
mov cx, 3
add bx, cx    ;bx = bx + cx = (1+2)*(3-1)/5+3=4
mov cx, 2
add bx, cx    ;bx = bx + cx = (1+2)*(3-1)/5+3+2=6
sub bx, ax    ;bx = bx - ax = (1+2)*(3-1)/5+3+2-(1*2)=4
```

```
01 ;(1 + 2) * (3 - 1) / 5 + 3 + 2 - (1 * 2)
02
03 .model small
04 .stack 100h
05 .data
06
07 .code
08 main proc
09     mov ax, @data
10     mov ds, ax
11
12     mov ax, 2
13     inc ax          ;ax = ax + 1 = (1+2)=3
14     mov bx, 3
15     dec bx          ;bx = bx - 1 = (3-1)=2
16     mul bx          ;ax = ax * bx = (1+2)*(3-1)=6
17     mov bx, 5
18     div bx          ;ax = ax / bx = (1+2)*(3-1)/5=1
19                     ;mul is done before div, violating BODMAS
20                     ;can't use dx register to store divisor(5)
21                     ;since dx is used to store the remainder
22     mov bx, ax      ;bx = ax = (1+2)*(3-1)/5=1
23     mov ax, 1
24     mov cx, 2
25     mul cx          ;ax = ax * 2 = (1*2)=2
26     mov cx, 3
27     add bx, cx      ;bx = bx + cx = (1+2)*(3-1)/5+3=4
28     mov cx, 2
29     add bx, cx      ;bx = bx + cx = (1+2)*(3-1)/5+3+2=6
30     sub bx, ax      ;bx = bx - ax = (1+2)*(3-1)/5+3+2-(1*2)=4
```

Result


registers
H
L
AX
00
02
BX
00
04
CX
00
02
DX
00
00