Nour Ahmed Mahmoud Khalil
23010913

Assembly Lab 1

Problem 1:

- **Program functionality:** This program calculates the factorial of a number (5) by using repeated multiplication. The result of the factorial calculation is stored in the AX register. The program uses CX as a loop counter to repeat the multiplication until the factorial is complete.
- **Values existing in the registers:**
    - AX (Accumulator Register):
        - Initialized to 1.
        - Multiplied repeatedly by the value in CX during the loop.
        - After the loop, AX contains 120 (5!).
    - CX (Counter Register):
        - Initialized to 5 (the value of data)
        - Decremented automatically by the LOOP instruction each iteration.
        - After the loop finishes, CX = 0.

- **Note:** The program needs a proper termination instruction for the emulator to stop safely.

Problem 2:

- **Program Objective:**
  The goal of this program is to read two bytes of input:
    1. **START** – the starting number.
    2. **RANGE** – the count of consecutive numbers to sum.

The program calculates the sum of all numbers from start up to start + range - 1. Overflow is detected and stored in a separate register.

- **Program Flow and Logic**
    1. **Initialization:**
        - AX = 0 → will store the total sum.
        - DX = 0 → will store any overflow beyond 16 bits.
        - BX = START → current number to be added.
        - CX = RANGE → loop counter.

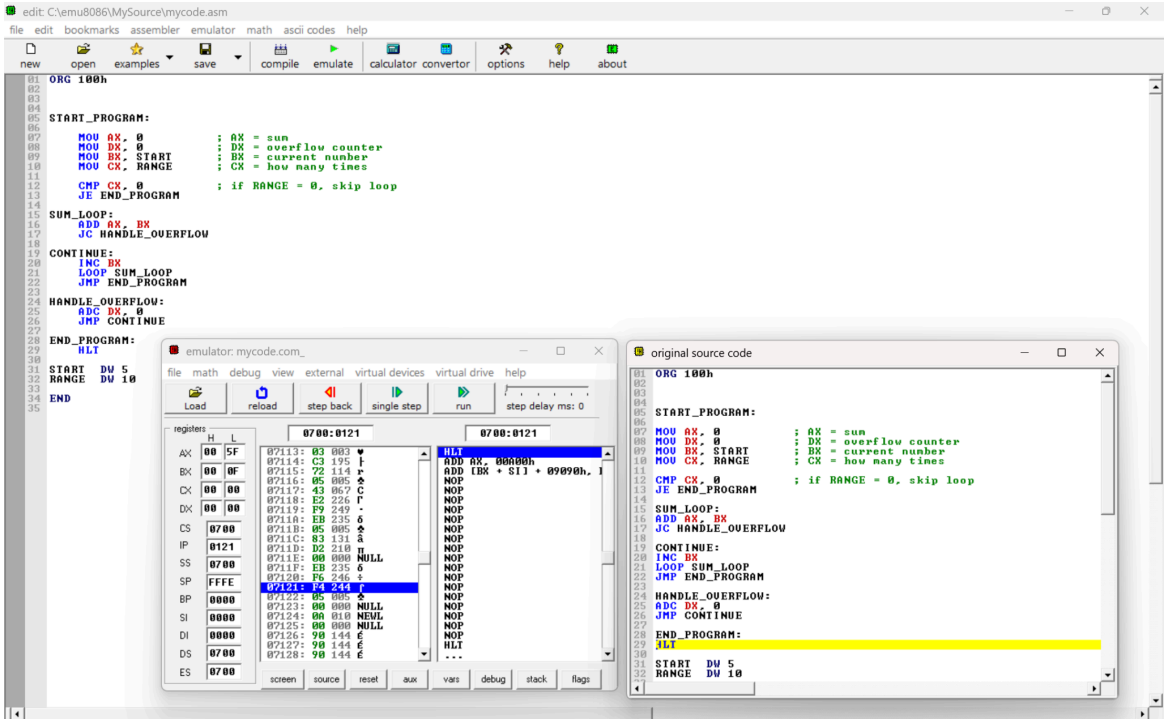
    2. **Validation:**
        - If RANGE = 0, the program jumps to END_PROGRAM immediately.

3. **Sum Loop (SUM_LOOP):**
    - Add BX (current number) to AX.
    - If carry occurs (sum exceeds 16 bits), jump to HANDLE_OVERFLOW.
    - Increment BX for the next number.
    - Decrement CX and repeat the loop until all numbers are added.
4. **Overflow Handling (HANDLE_OVERFLOW):**
    - Add the carry to DX using ADC DX, 0.
    - Continue to next number.
5. **End Program (END_PROGRAM):**
    - Halt execution (HLT instruction).
    - At this point, the sum is stored in AX, and any overflow is in DX.

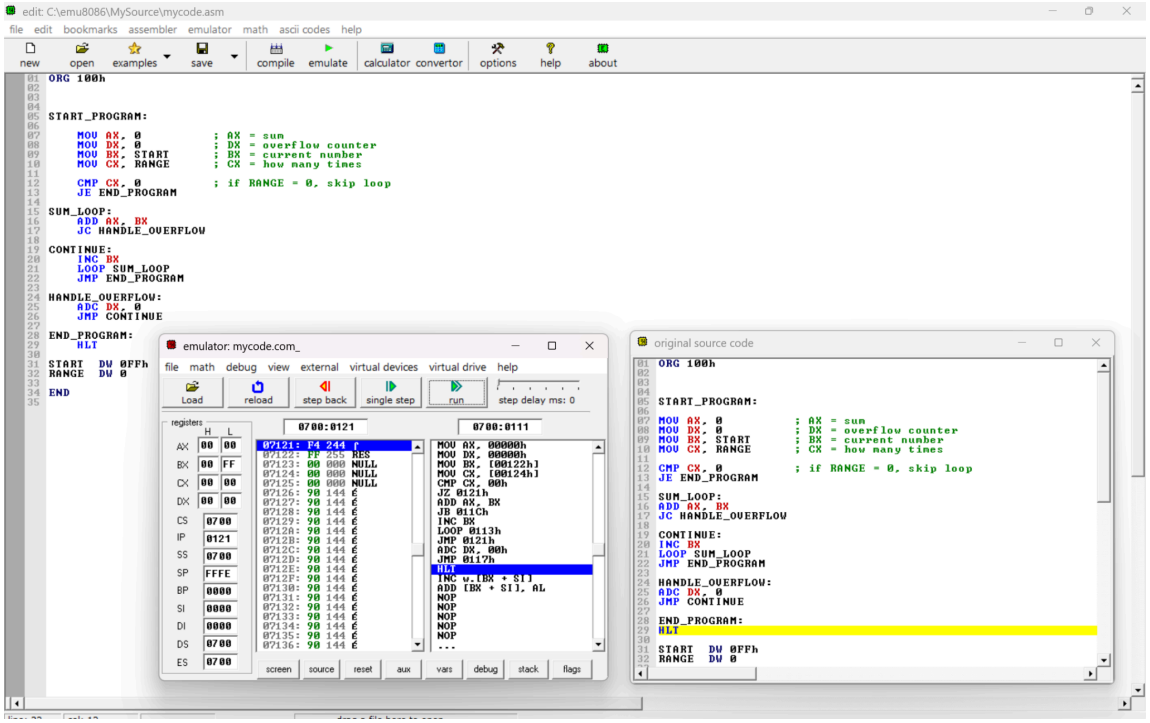
- **Code**

```
01 ORG 100h
02
03
04
05 START_PROGRAM:
06
07      MOV AX, 0           ; AX = sum
08      MOV DX, 0           ; DX = overflow counter
09      MOV BX, START       ; BX = current number
10      MOV CX, RANGE       ; CX = how many times
11
12      CMP CX, 0           ; if RANGE = 0, skip loop
13      JE END_PROGRAM
14
15 SUM_LOOP:
16      ADD AX, BX
17      JC HANDLE_OVERFLOW
18
19 CONTINUE:
20      INC BX
21      LOOP SUM_LOOP
22      JMP END_PROGRAM
23
24 HANDLE_OVERFLOW:
25      ADC DX, 0
26      JMP CONTINUE
27
28 END_PROGRAM:
29      HLT
30
31 START   DW 0FFh
32 RANGE   DW 0
33
34 END
35
```

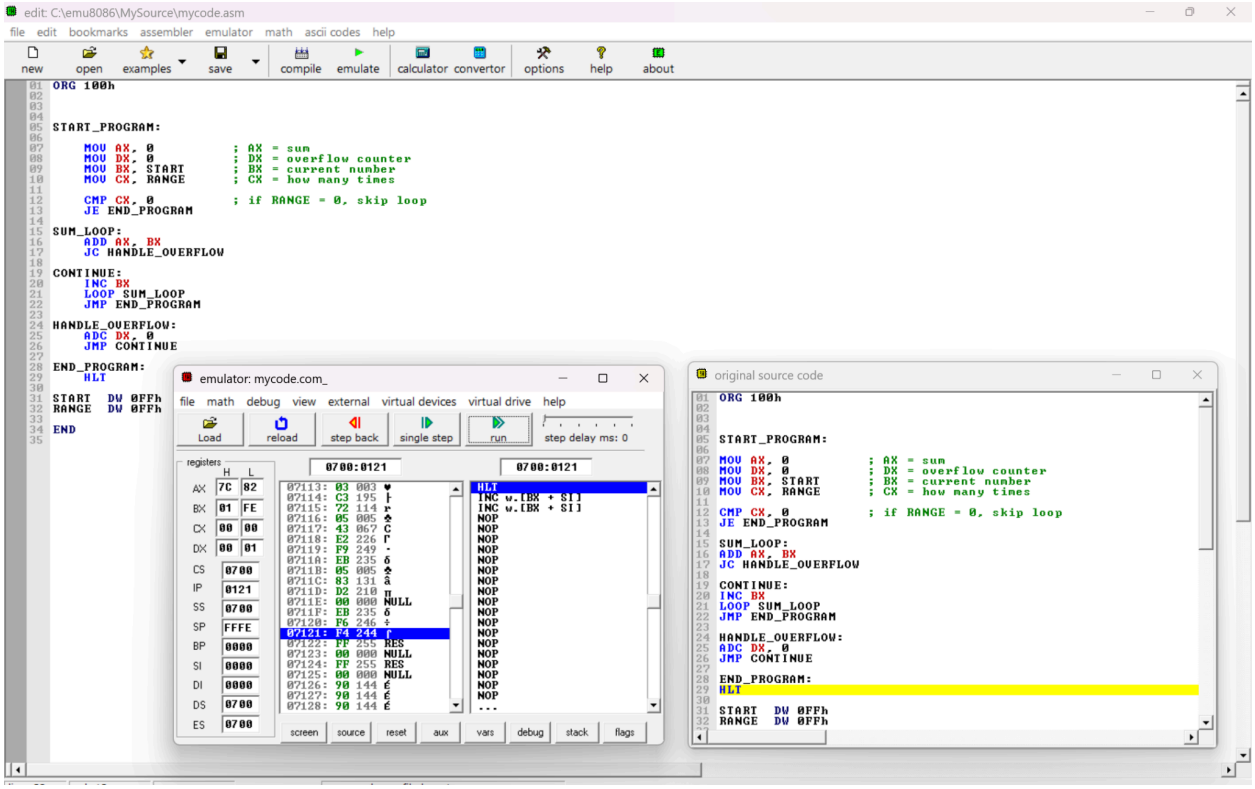
- **Test Cases:**



0 range



Overflow (use DX)



Problem 3:

- **Program Objective:**

  The program sums two multi-byte numbers (10 bytes each) stored in memory using the DB directive. The sum is stored in memory starting at address 500h. Any final carry beyond the most significant byte is stored separately.

- **Program Flow**
  1. **Initialization:**
     - SI points to the last byte of NUM1 (OFFSET NUM1 + 9)
     - DI points to the last byte of NUM2 (OFFSET NUM2 + 9)
     - BX points to memory location 500h + 9 (result)
     - CX = 10 (number of bytes to sum)
     - Clear carry with CLC

  2. **Addition Loop (SUM_LOOP):**

     - Load a byte from NUM1 into AL
     - Add corresponding byte from NUM2 **with carry** using ADC
     - Store the result in memory at [BX]

- Decrement SI, DI, and BX to move to the next byte
- Repeat until all 10 bytes are processed

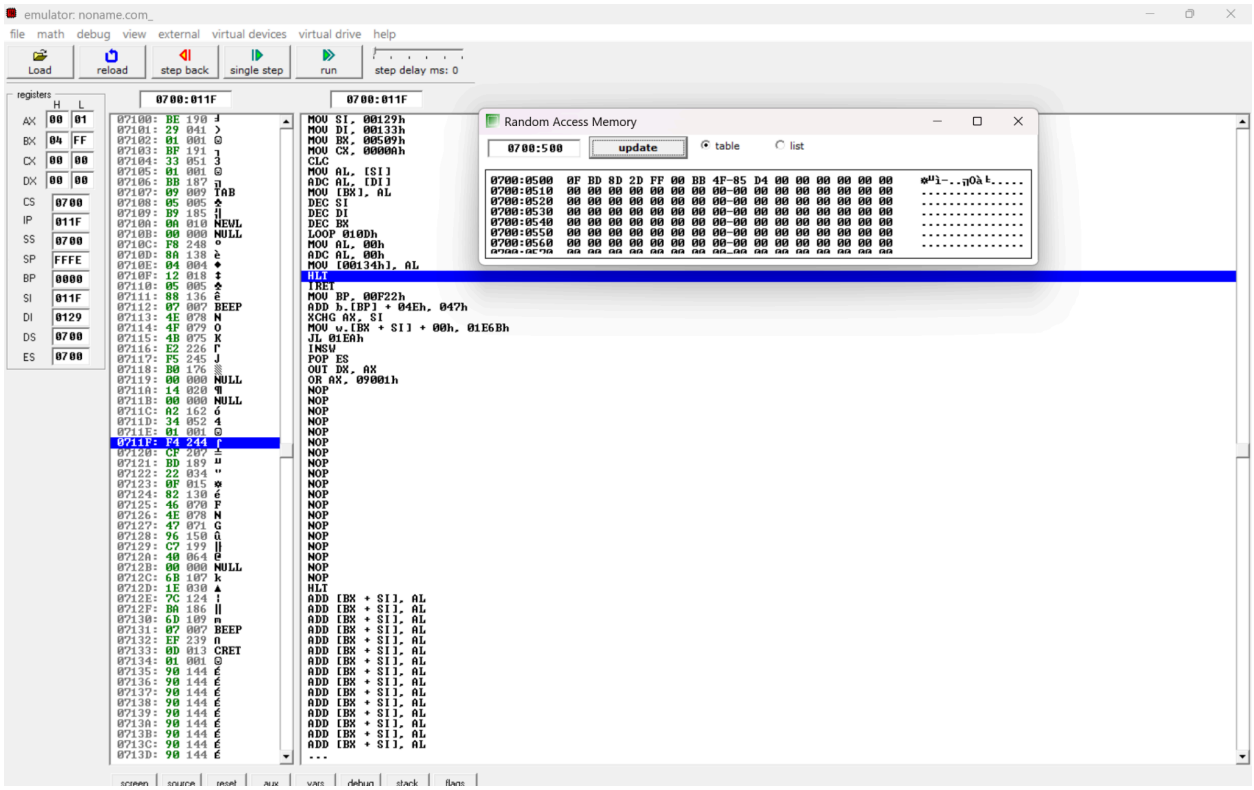
3. **Store Final Carry:**

   - Use ADC AL, 0 after the loop to capture the last carry (0 or 1)
   - Store in memory location CARRY

4. **End Program:**
   - Halt execution (HLT)

- **Code**

```
01 ORG 100h
02
03 start:
04     MOV SI, OFFSET NUM1 + 9
05     MOV DI, OFFSET NUM2 + 9
06     MOV BX, 500h + 9
07     MOV CX, 10
08     CLC
09
10 SUM_LOOP:
11     MOV AL, [SI]
12     ADC AL, [DI]
13     MOV [BX], AL
14
15     DEC SI
16     DEC DI
17     DEC BX
18     LOOP SUM_LOOP
19
20     ; Store the final carry bit
21     MOV AL, 0
22     ADC AL, 0                    ; AL = carry flag (0 or 1)
23     MOV [CARRY], AL
24
25     HLT
26
27 NUM1 DB 0CFh,0BDh,22h,0Fh,82h,46h,4Eh,47h,96h,0C7h
28 NUM2 DB 040h,00h,6Bh,1Eh,7Ch,0BAh,6Dh,07h,0EFh,0Dh
29 CARRY DB 0
30
31 END
```


emulator: noname.com_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
run
step delay ms: 0
registers
AX 00 01
BX 04 FF
CX 00 00
DX 00 00
CS 0700
IP 011F
SS 0700
SP FFFE
BP 0000
SI 011F
DI 0129
DS 0700
ES 0700
0700:011F
0700:011F
07100: BE 190 ¥
07101: 29 041)
07102: 01 001 ☺
07103: BF 191 ┐
07104: 33 051 3
07105: 01 001 ☺
07106: BB 187 ╗
07107: 09 009 TAB
07108: 05 005 ♣
07109: D9 185 ╣
0710A: 0A 010 NEWL
0710B: 00 000 NULL
0710C: F8 248 °
0710D: 8A 138 è
0710E: 04 004 ♦
0710F: 12 018 ↕
07110: 05 005 ♣
07111: 88 136 ê
07112: 07 007 BEEP
07113: 4E 078 N
07114: 4F 079 O
07115: 4B 075 K
07116: E2 226 Γ
07117: F5 245 J
07118: B0 176 ░
07119: 00 000 NULL
0711A: 14 020 ¶
0711B: 00 000 NULL
0711C: A2 162 ó
0711D: 34 052 4
0711E: 01 001 ☺
0711F: F4 244 ⌠
07120: CF 207 ╧
07121: BD 189 ╜
07122: 22 034 "
07123: 0F 015 ☼
07124: 82 130 é
07125: 46 070 F
07126: 4E 078 N
07127: 47 071 G
07128: 96 150 û
07129: C7 199 ╟
0712A: 40 064 @
0712B: 00 000 NULL
0712C: 6B 107 k
0712D: 1E 030 ▲
0712E: 7C 124 |
0712F: BA 186 ║
07130: 6D 109 m
07131: 07 007 BEEP
07132: EF 239 ∩
07133: 0D 013 CRET
07134: 01 001 ☺
07135: 90 144 É
07136: 90 144 É
07137: 90 144 É
07138: 90 144 É
07139: 90 144 É
0713A: 90 144 É
0713B: 90 144 É
0713C: 90 144 É
0713D: 90 144 É
MOV SI, 00129h
MOV DI, 00133h
MOV BX, 00509h
MOV CX, 00000h
CLC
MOV AL, [SI]
ADC AL, [DI]
MOV [BX], AL
DEC SI
DEC DI
DEC BX
LOOP 010Dh
MOV AL, 00h
ADC AL, 00h
MOV [00134h], AL
HLT
IRET
MOV BP, 00F22h
ADD b.[BP] + 04Eh, 047h
XCHG AX, SI
MOV w.[BX + SI] + 00h, 01E6Bh
JL 01E8Ah
INSW
POP ES
OUT DX, AX
OR AX, 09001h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
HLT
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
Random Access Memory
0700:500
update
table
list
0700:0500 0F BD 8D 2D FF 00 BB 4F-85 D4 00 00 00 00 00 00
0700:0510 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:0520 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:0530 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:0540 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:0550 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:0560 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
screen
source
reset
aux
vars
debug
stack
flags