Introducere

Acest îndrumător de laborator își propune, în primul rând, să prezinte cunoștințele de bază pentru programarea în limbajul de asamblare x86, specific arhitecturii de procesoare Intel x86, pornind de la trăsăturile arhitecturale ale procesorului 8086 și continuând cu procesoarele mai avansate din serie, de la 386 și Pentium, până la Intel i3, i5, i7 si i9. În al doilea rând, această lucrare prezintă bazele dezvoltării unui microsistem digital, construit în jurul microprocesorului 8086, folosind circuite de memorie și porturi de intrare / ieșire specifice familiei x86. Îndrumătorul se adresează în special studenților de la specializările Calculatoare și Tehnologia Informației (CTI) și Informatică, pentru discipina Proiectarea microsistemelor digitale (PMD).

Limbajele de asamblare sunt puternic dependente de arhitectura sistemului de calcul pe care rulează și pot fi definite ca forma simbolică de reprezentare a limbajului nativ al acelui sistem de calcul, limbaj numit "cod mașină". Orice program scris într-un limbaj de programare de nivel înalt (C, C++, Java, etc) trece prin mai multe etape înainte de a fi executat de procesor: compilare, preprocesare, link-editare și transformare în cod mașină. Limbajul de asamblare îl putem poziționa la un nivel de abstractizare situat jos, cel mai aproape de hardware, imediat înainte ca procesorul să execute codul mașină.

PMD – Lucrarea 1

Această lucrare de laborator își propune să introducă cunoștințele de bază, necesare pentru realizarea unui prim program simplu în limbajul de asamblare x86, folosind instrucțiuni aritmetice și logice.

Vor fi prezentate: setul de registre al microprocesorului 8086, structura generală a unui program în asamblare, instalarea Visual Studio Code împreună cu extenisa MASM/TASM și rularea unui prim program, precum și detalierea funcționării unui set restrâns de instrucțiuni de bază.

1.1 Setul de registre al microprocesorului 8086

Din punct de vedere al rolului pe care îl au, registrele microprocesorului 8086 se clasifică astfel:

- Registre generale, pe 16 biți: **AX** (Acumulator), **BX** (Registru de baza), **CX** (Counter), **DX** (Registru de date);
- Registre segment, pe 16 biți: **CS** (Code Segment), **DS** (Data Segment), **ES** (Extra data Segment), **SS** (Stack Segment);
- Registre index, pe 16 biți: **SI** (Source Index), **DI** (Destination Index);
- Registre indicatoare de adresă, pe 16 biți: **SP** (Stack Pointer), **BP** (Base Pointer);
- Registrul adresei instrucțiunii curente, pe 16 biți: **IP** (Instruction Pointer).

În cazul registrelor generale pe 16 biți, se pot adresa separat jumătățile lor, la nivel de octet, adică biții 15…8, respectiv 7…0, folosind denumirile *L (low), respectiv *H (high), unde * poate fi una dintre literele A, B, C, D. Astfel, registrul acumulator AX poate fi adresat la nivel de jumătate: AH reprezintă jumătatea lui superioară, în timp ce AL reprezintă jumătatea lui inferioară. În mod similar, BX este un registru de bază pe 16 biți, care este compus din două registre pe cate 8 biți: BH și BL, ș.a.m.d.

Începând cu procesorul 386, pe 32 de biți, există și variantele extinse ale registrelor de uz general, pe câte 32 de biți fiecare, denumirile începand cu litera E (“Extended”): registre generale – EAX, EBX, ECX, EDX, registre index – ESI, EDI, registre indicatoare de adresă ESP, EBP.

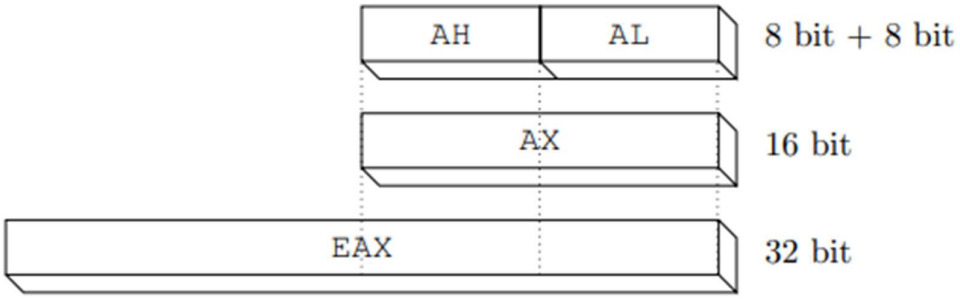

Fig. 1.1 Componentele registrului EAX (sursa: [1])

1.2 Rolul registrelor de segment

Spațiul total de memorie al microprocesorului 8086 este de 1 MB și poate fi impărțit în mai multe segmente. La un moment dat, microprocesorul poate lucra cu 4 segmente: unul de cod, unul de stivă și două de date. Adresele de început ale celor patru segmente sunt salvate în registrele de segment, fiecare pe câte 16 biți: CS, pentru segmentul de cod – în care se află instrucțiunile, DS pentru segmentul de date, SS pentru segmentul de stivă și ES pentru segmentul de date suplimentare [2].

1.3 Instalarea Visual Studio Code si a extensiei MASM/TASM

Ultima versiune a IDE-ului Visual Studio Code pentru sistemul de operare Windows poate fi instalată de aici: https://code.visualstudio.com/

Odată lansat în execuție Visual Studio Code, în meniul vertical din partea stângă accesați butonul „Extensions (CTRL+Shift+X)” și în fereastra de căutare scrieți textul „masm”. Instalați extensia MASM/TASM, exact ca în figura 1.2.

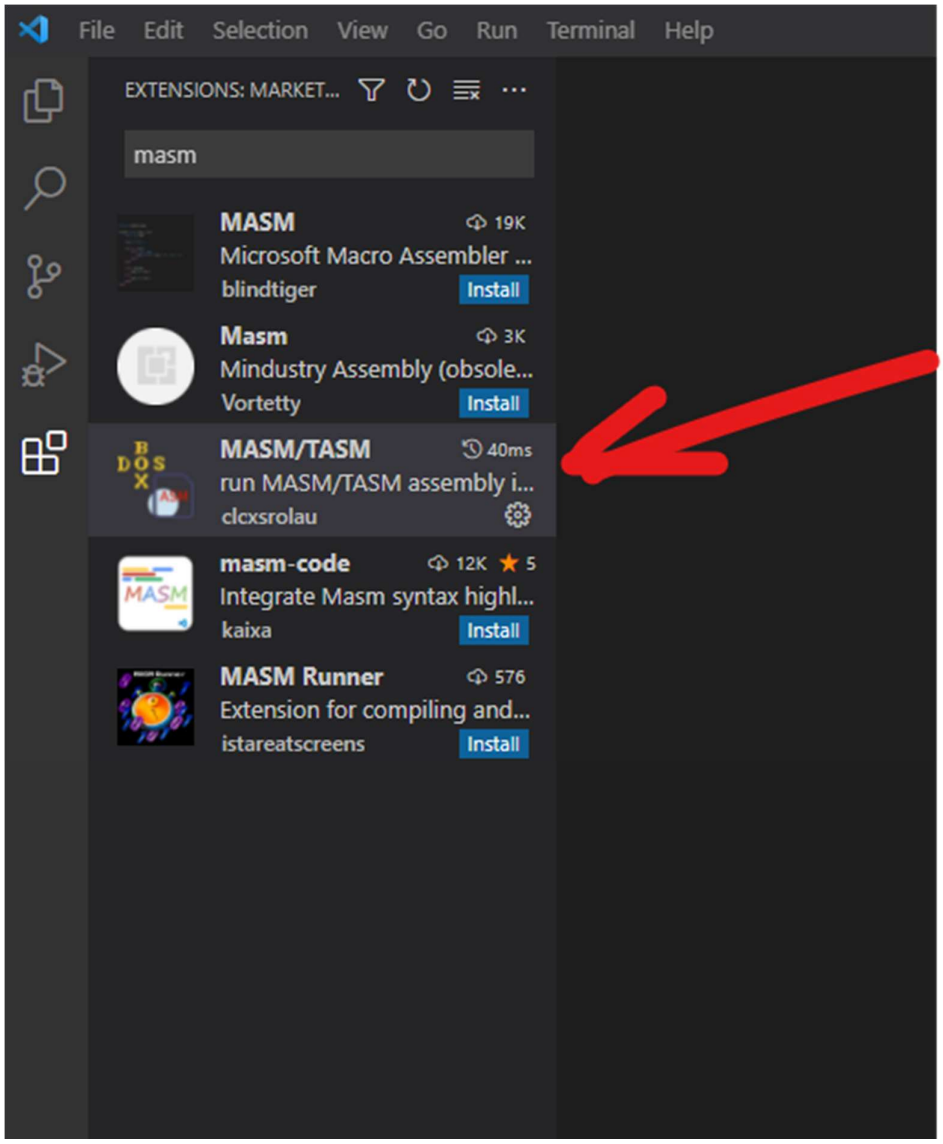

Fig. 1.2 Extensia MASM/TASM in Visual Studio Code

1.4 Crearea unui fișier nou cu cod assembly x86

Pentru crearea unui nou fișier cu cod sursă în limbajul de asamblare x86, accesați meniul superior File -> New File în Visual Studio Code și de pe prima linie a fișierului nou accesați opțiunea „Select a language” pentru a selecta limbajul „assembly(DOS)” din lista de limbaje disponibile (vezi fig. 1.3.)

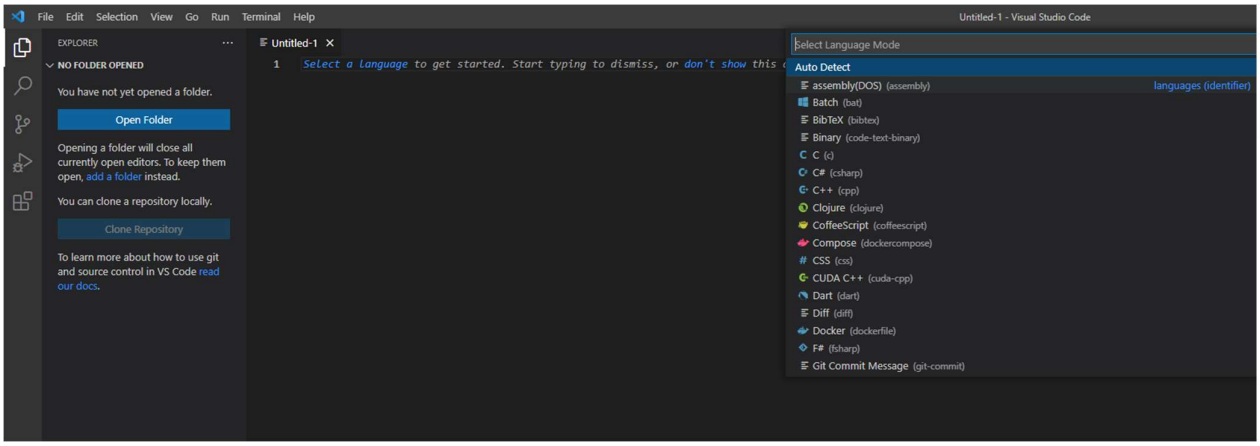

Fig. 1.3 Selecția limbajului "assembly(DOS)"

1.5 Structura programelor în asamblare

Pentru a scrie un nou program în limbajul de asamblare x86, puteți porni de la următorul schelet. Comentariile se scriu după simbolul "punct și virgulă", până la finalul rândului. Programele în limbaj de asamblare nu sunt case-sensitive, deci instrucțiunile pot fi scrise cu litere mari, litere mici sau o combinație între cele două.

dosseg

.model small *; prin această instrucțiune se selectează modelul de memorie folosit*

.data *; reprezintă secțiunea de declarare a datelor*

.code *; reprezintă secțiunea în care se scrie codul sursă al programului*

main proc *; proc este cuvântul cheie pentru începerea unei proceduri*

; codul sursă din main

main endp *; endp este cuvântul cheie care semnifică finalul unei proceduri*

end

O instrucțiune în limbaj de asamblare are următorul format [1]:

[<eticheta>:] [<instrucțiune / operație> [<operanzi>]] [; <comentarii>]

<eticheta> este un nume format din litere, cifre și / sau caractere speciale, care începe obligatoriu cu o literă sau cu un caracter special. Etichetele reprezintă un indicator de poziție,

ele nu influențează execuția instrucțiunilor în fața căreia sunt scrise, însă pot fi folosite pentru a ne referi la o anumită poziție din cadrul programului (indicăm o anumită linie de cod), în special în cazul instrucțiunilor de salt. În fața unei instrucțiuni se poate scrie la un moment dat cel mult un nume de etichetă urmat de :, ceea ce înseamnă că scrierea etichetelor este opțională.

<instrucțiune / operație> este mnemonica unei instrucțiuni în asamblare. Instrucțiunile pentru x86 sunt formate din 3 sau 4 litere. Exemple: ADD pentru adunare, SUB pentru scădere, LOOP pentru realizarea unei bucle de tip for.

<comentarii> - este interpretat ca și comentariu orice text care începe cu ; și continuă până la sfârșitul rândului.

Cea mai simplă instrucțiune în limbaj de asamblare este **MOV destinație, sursă**, care are ca efect copierea conținutului operandului sursă în destinație.

Ca argumente pentru instrucțiunea **MOV destinație, sursă** putem avea următoarele combinații [3]:

Reg, reg – transfer între registre pe 1, 2 sau 4 octeți (exemplu pe 2 octeți: MOV AX, CX)

Reg, const – încărcare imediată constantă într-un registru (exemplu pe 1 octet: MOV AL, S ; unde S a fost declarată în secțiunea .data a programului ca S EQU 5)

Mem, const – încărcare imediată constantă într-o adresă de memorie (exemplu: MOV [2400], S ; unde S este constanta folosită și în exemplul anterior, iar [2400] indică locația de memorie de la adresa 2400)

Reg, mem – încărcare conținut adresa de memorie într-un registru (exemplu: MOV BX, [2400])

Mem, reg – memorare conținut registru într-o locație de memorie (exemplu: MOV [2400], BX)

Atenție: Operarea directă asupra a două variabile declarate de programator sau asupra unei variabile și a unei constante este interzisă. De exemplu daca avem declarate variabilele X, Y si constanta C, oricare dintre următoarele instrucțiuni este interzisă: ~~MOV X, Y~~; ~~MOV Y, X~~; ~~MOV C, X~~; ~~MOV C, Y~~.

Un exemplu de instrucțiune care respectă formatul general explicat mai sus este:

E1: MOV AL, BH ; această instrucțiune are ca efect copierea conținutului registrului BH peste conținutul registrului AL; după execuția instrucțiunii, aceeași valoare binară se va afla atât în registrul BH, cât și în registrul AL.

Atenție: În orice instrucțiune de mutare (MOV) sau aritmetică, cei doi operanzi trebuie să aibă aceeași dimensiune. De exemplu, instrucțiunea MOV AX, CL va duce la semnalarea unei erori de compilare, pentru că dimensiunea lui AX este de 16 biți, iar a lui CL de 8 biți.

1.6 Declararea variabilelor și a constantelor

Declararea variabilelor se face în secțiunea .data a programului, în funcție de lungime, folosind identificatorii ***db – data byte, dw – data word, dd – data double-word, dq – data quad-word*** (numiți generic "size directive"), respectând următoarea sintaxă:

.data

nume_variabila db 0 *; am declarat o variabilă cu numele nume_variabila, cu lungimea de un byte (8 biți) și valoarea inițială 0*

nume1 db ? *; am declarat o variabilă cu numele nume1, cu lungimea de un byte și valoare inițială necunoscută*

nume2 dw 5 *; am declarat o variabilă cu numele nume2, cu lungimea de de doi bytes (16 biți – word) și valoarea inițială 5*

nume3 dd 117 *; variabila nume3 cu lungimea dublu-cuvânt (double-word), adică 32 biți și valoarea inițială 117 în zecimal*

nume4 dq 0 *; variabila nume4 cu lungimea quad-word, adică de 4 ori lungimea unui cuvânt, însemnând 8 bytes = 64 biți și valoarea inițială 0*

vect db 1,2,3,4,5 *; se definește un vector de 5 elemente de tip byte, elementele având valorile inițiale 1, 2, 3, 4 și 5*

Valorile numerice pot fi specificate în binar, hexazecimal sau zecimal:

A db 05h *; se declară o variabilă A pe un byte, care conține valoarea 05h, h fiind sufixul pentru hexazecimal.*

B db 5 *; se declară o variabile B pe un byte, care conține valoarea 5 în zecimal, deci este egală cu variabila A de mai sus.*

C db 00000101b *; valoarea variabilei C este egală cu variabilele A și B declarate mai sus, doar ca valoarea ei a fost specificată în binar prin sufixul b*

Pentru a defini <n> elemente cu valoarea <x> se poate folosi sintaxa <n> DUP(<x>). De exemplu, un vector de 5 elemente cu toate valorile inițiale 0, se poate defini: **vect db 5 dup(0)**

Constantele se declară folosind cuvântul cheie EQU:

L EQU 10

PI EQU 3.1415

INT_DOS EQU 21h *; această constantă poate fi folosită pentru întreruperea INT 21h*

1.7 Realizarea unui prim program în limbajul de asamblare x86

Programul de mai jos calculează A+B-C pentru 3 numere pe câte 8 biți (A, B și C) citite de la tastatură și afișează rezultatul pe ecran. O întrerupere DOS este folosită pentru a citi numerele de la tastatură, precum și pentru a afișa rezultatul pe ecran : INT 21H.

Atenție: Dispozitivele periferice ale calculatorului (tastatură, ecran, imprimantă), precum și întreruperea DOS INT 21H sunt proiectate să lucreze cu șiruri de caractere codificate în ASCII. Numerele se introduc ca un șir de cifre zecimale codificate în ASCII, având codurile 30h, 31h, 32h, …, 39h pentru cifrele 0, 1, 2, …, 9. Pentru a face calcule în dispozitivul aritmetic, care lucrează în binar, șirul de cifre trebuie convertit. Conversiile zecimal ASCII – binar 2 octeți, precum și binar – zecimal ASCII nu fac scopul acestei lucrări, iar întreruperile DOS nu se mai folosesc în practică în timpuri moderne, motiv pentru care nu vom insista asupra lor și nu vom folosi întreruperea INT 21h decât cu scop demonstrativ, pe un scenariu foarte restrâns.

Ținând cont de cele de mai sus, veți observa că programul următor (Aritmetic1), funcționează corect doar dacă A, B și C sunt cifre, iar rezultatul este tot o cifră. Motivul este că în urma citirii unei cifre cu întreruperea 21h, în registrul AL se salvează codul ASCII (convertit în hexa) al acesteia, care este cu 30h mai mare decât cifra citită (de exemplu pentru cifra 5 se salvează în AL valoarea 35h). La citire se poate înlătura acest "adaos" prin scăderea valorii 30h, însă la afișarea rezultatului, daca acesta este format din mai multe cifre, acesta trebuie convertit printr-o procedură destul de complexă în asamblare.

Pentru mai multe detalii despre întreruperea INT 21h și serviciile asociate acesteia, accesați următorul link:
https://stanislavs.org/helppc/int_21.html?fbclid=IwAR3CH46mspyGDOTN2_63SkV-6NMtvKMPujmOrWwyHxsYp60X4JwgzGMsbDU

```
dosseg

.model small ; prin această instrucțiune se selectează modelul de memorie folosit

.stack 100h ;declararea segmentului stivă

.data ; reprezintă secțiunea de declarare a datelor

  A DB 0

  B DB 0

  C DB 0

  X DW 7

  n1 DB "Introduceti primul numar:$"

  n2 DB "Introduceti al doilea numar:$"
```

```
  n3 DB “Introduceti al treilea numar:$”
  result DB “Rezultatul este:$”
```

.code *; reprezintă secțiunea în care se scrie codul sursă al programului*

```
new_line proc
    MOV AH, 2
    MOV DL, 10 ; valoarea hexa 0x0A (în zecimal 10) corespunde codului ASCII al caracterului special LF – line feed, corespunzător lui ‘\n’ (new line character) din C
    INT 21H
    ret
new_line endp
```

main proc *; proc este cuvântul cheie pentru începerea unei proceduri*

```
  MOV AX, @data ; instrucțiuni implicite pentru încărcarea segmentului de date
  MOV DS, AX
  ; citirea primului număr de la tastatură
  MOV AH, 9
  MOV DX, OFFSET n1
  INT 21h
  MOV AH, 1 ; serviciu de citire asociat întreruperii INT 21H – Keyboard input with echo
  INT 21H
  MOV A, AL
  CALL new_line
; citirea celui de-al doilea număr de la tastatură
  MOV AH, 9
  MOV DX, OFFSET n2
  INT 21h
  MOV AH, 1
  INT 21H
```

```
   MOV B, AL
   CALL new_line
; citirea celui de-al treilea număr de la tastatură
   MOV AH, 9
   MOV DX, OFFSET n3
   INT 21h
   MOV AH, 1
   INT 21H
   MOV C, AL
   CALL new_line
; afisarea liniei "Rezultatul este:" pe ecran
   MOV AH, 9
   MOV DX, OFFSET result
   INT 21h
; calculul sumei A + B
   MOV AL, A ; primul număr A este mutat în registrul AL pe 8 biți
   ADD AL, B ; rezultatul sumei A+B se află în registrul AL pe 8 biți
; scăderea lui C din rezultat
   SUB AL, C
; afișarea rezultatului pe ecran
   MOV AH, 2 ; serviciu de afișare asociat întreruperii INT 21H – Display output
   MOV DL, AL ; rezultatul este copiat în registrul DL
   INT 21h
   MOV AH, 4Ch
   INT 21h
main endp ; endp este cuvântul cheie care semnifică finalul unei proceduri
end main
```

După editarea programului într-un fișier nou, acesta poate fi salvat cu extensia ".asm".

Compilarea și rularea programului se realizează prin acționarea opțiunii "Debug ASM code" sau "Run ASM code" din meniul care apare la click dreapta în codul sursă (vezi fig. 1.4).

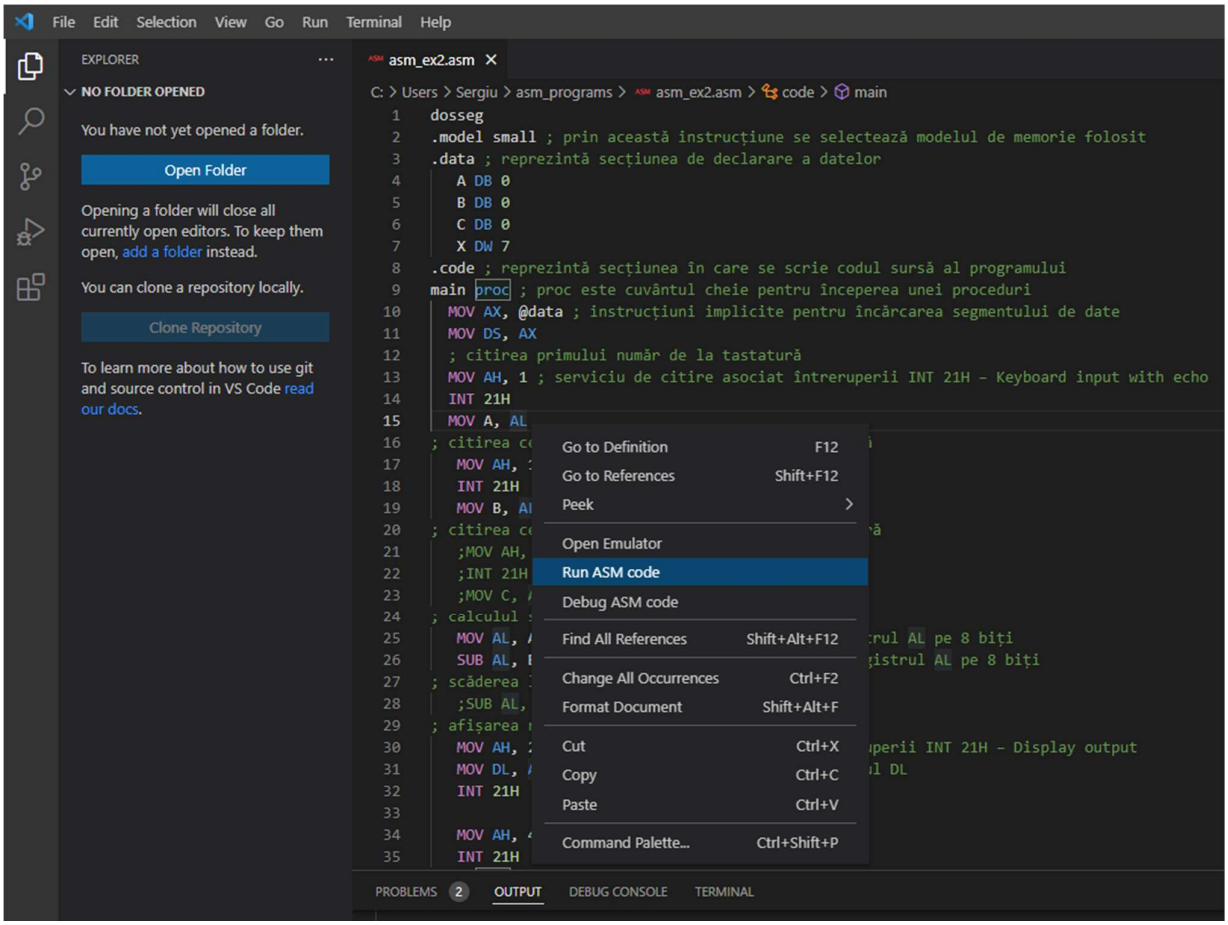

Fig 1.4 Rularea programului în VS Code

Odată ce ați selectat "Run ASM code", în partea dreapta se va deschide fereastra DOS care permite introducerea caracterelor de la tastatură și afișarea rezultatului (fig. 1.5).

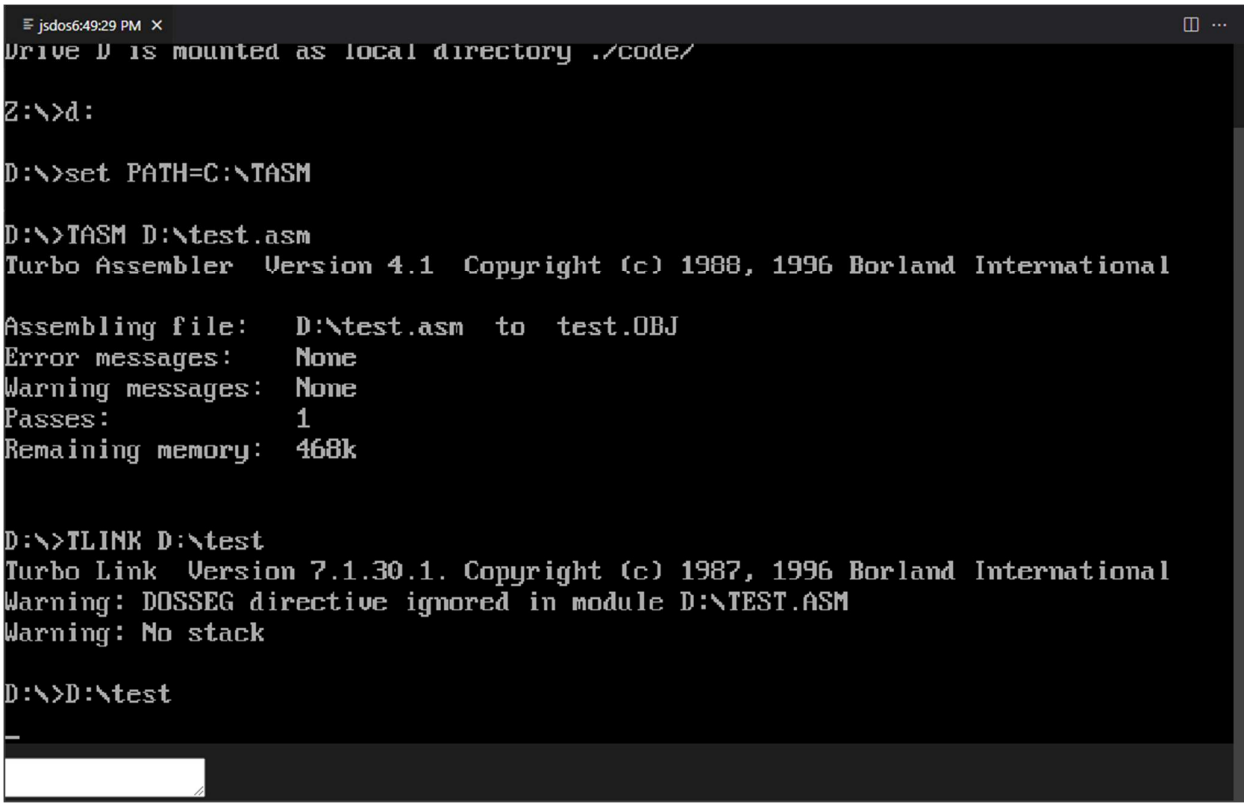

Figure 1.5 Rularea programului în DOS

1.8 Setul de instrucțiuni aritmetice și logice

Instrucțiunile aritmetice și logice pot fi executate pe 1, 2 sau 4 octeți și sunt de forma [3]:

<table>
<tr><td>Adunare</td><td>ADD</td><td rowspan="10">dest, sursa</td><td colspan="2"></td></tr>
<tr><td>Adunare cu carry</td><td>ADC</td><td colspan="2"></td></tr>
<tr><td>Scadere</td><td>SUB</td><td>Reg, reg</td><td rowspan="6">Lungime 1, 2 sau 4 octeți</td></tr>
<tr><td>Scadere cu imprumut (borrow)</td><td>SBB</td><td>Reg, const</td></tr>
<tr><td>SI logic pe bit</td><td>AND</td><td>Mem, const</td></tr>
<tr><td>SAU logic pe bit</td><td>OR</td><td>Reg, mem</td></tr>
<tr><td>SAU-Exclusiv</td><td>XOR</td><td>Mem, reg</td></tr>
<tr><td>Comparatie (se realizeaza prin scadere)</td><td>CMP</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Comparatie prin SI logic</td><td>TEST</td></tr>
</table>

Toate instrucțiunile aritmetice și logice poziționează indicatorii de condiție din registrul FLAGS în funcție de valoarea rezultatului. Indicatorii de condiție pot fi citiți și interpretați de către instrucțiunile de salt condiționat, aspect pe care îl vom trata ulterior.

Atenție: Instrucțiunile MOV și XCHG NU poziționează indicatorii de condiție!

1.9 Întrebări

1. Care sunt registrele de uz general ale procesorului 8086 și ce denumiri au ele?

2. Specificați numărul de biți ai fiecărui registru, dintre următoarele: AX, CL, BH, AL, DX.

3. Cum se declară variabila D pe 16 biți cu valoarea inițială 1?

4. Cum se declară o variabilă X pe 8 biți cu valoarea inițială necunoscută?

5. Pot să adun direct două variabile X și Y pe câte 8 biți fiecare, dacă ambele au fost decalarate în secțiunea .data a programului, folosind instrucțiunea ADD X, Y?

6. Dați exemplu de un registru segment pe 16 biți și explicați rolul acestuia.

7. Care este numele registrului pe 16 biți în care se salvează adresa de început din memorie a segmentului de stivă?

8. Care dintre următoarele instrucțiuni poziționează indicatorii de condiție: ADD AL, BH; MOV AL, A; SUB AL, Y?

9. În urma apelării întreruperii INT 21H cu serviciul de citire al unei taste de la tastatură, în care registru al procesorului va fi stocat codul tastei și câți biți are acest registru?

10. Considerând că folosim întreruperea INT 21H cu serviciul de afișare a unui caracter pe ecran, în care registru al procesorului trebuie sa salvăm data ce se dorește a fi afișată, înainte de a apela întreruperea? Pe câți biți este acel registru?

11. Care este acronimul utilizat pentru numele registrului care conține adresa instrucțiunii următoare celei care este executată de către microprocesor? Ce înseamnă acesta?

1.10 Probleme

1. Pornind de la programul Aritmetic1 dat ca exemplu, scrieți un nou program care afișează pe ecran primii 7 termeni din seria Fibonacci, considerând primii 3 termeni ca fiind: 0, 1, 1. Se va folosi întreruperea INT 21H doar pentru afișare, adaugând 30h la fiecare cifră pe care doriți să o afișați, înainte de apelarea întreruperii.

2. Scrieți un program în limbajul de asamblare x86, care afișează pe ecran suma primelor N numere naturale, pentru un N pe 8 biți de valoare mică, hardcodat în program (declarat în secțiunea .data ca o variabilă sau o constantă).