LUCRAREA DE LABORATOR 1
Elaborare programelor în limbajul de asamblare MASM în Visual Studio

1.1 SCOPUL LUCRARII

Lucrarea urmărește familiarizarea studenților cu regiştrii microprocesoarelor pe 16, 32 şi 64 de biți, cu mediul de dezvoltare a programelor Visual Studio. Se prezintă setările mediului Visual Studio necesare pentru a elabora, rula şi depăna aplicaţiile elaborate în limbajul de asamblare MASM utilizând regiştri pe 16, 32 şi 64 de biţi.

1.2 Regiştrii de bază utilizaţi în limbajul MASM

Fiecare program la execuţie obţine anumite resurse ale microprocesorului. Aceste resurse (regiştri) sunt necesare pentru executarea şi păstrarea în memorie a instrucţiunilor şi datelor programului, a informaţiei despre starea curentă a programului şi a microprocesorului.

Microprocesoarele pe 32 de biţi funcţionează în diferite moduri, ce determină mecanismele de protecţie şi de adresare a memoriei: modul real 8086 (pe 16 de biţi), modul virtual 8086 (V8086), modul protejat pe 32 de biţi (inclusiv protejat pe 16 de biţi). Modul de funcţionare a microprocesorului este impus de sistemul de operare (SO) în conformitate cu modul definit de aplicaţii (task-uri).

În microprocesoarele pe 64 de biţi au fost introduse noi moduri de funcţionare:

- Modul pe 64 de biţi (64-bit mode) – acest mod susţine adresarea virtuală pe 64 de biţi şi extensiile regiştrilor pe 64 de biţi. În acest mod este folosit numai *modelul plat de memorie* (un segment comun pentru cod, date şi stivă).

- Modul de compatibilitate (compatibility mode) permite SO să execute aplicaţii pe 32 şi 16 de biţi. Pentru aplicaţii microprocesorul reprezintă un microprocesor pe 32 de biţi cu toate atributele modului protejat, cu mecanismele de segmentare şi paginare.

Sistemele Windows pe 64 biţi nu susţin modul real pe 16 de biţi (simularea lui). În acest caz este necesară utilizarea DosBox-ului.

Microprocesoarele pe 64 de biţi reprezintă seturi de regiştri disponibili programatorilor.

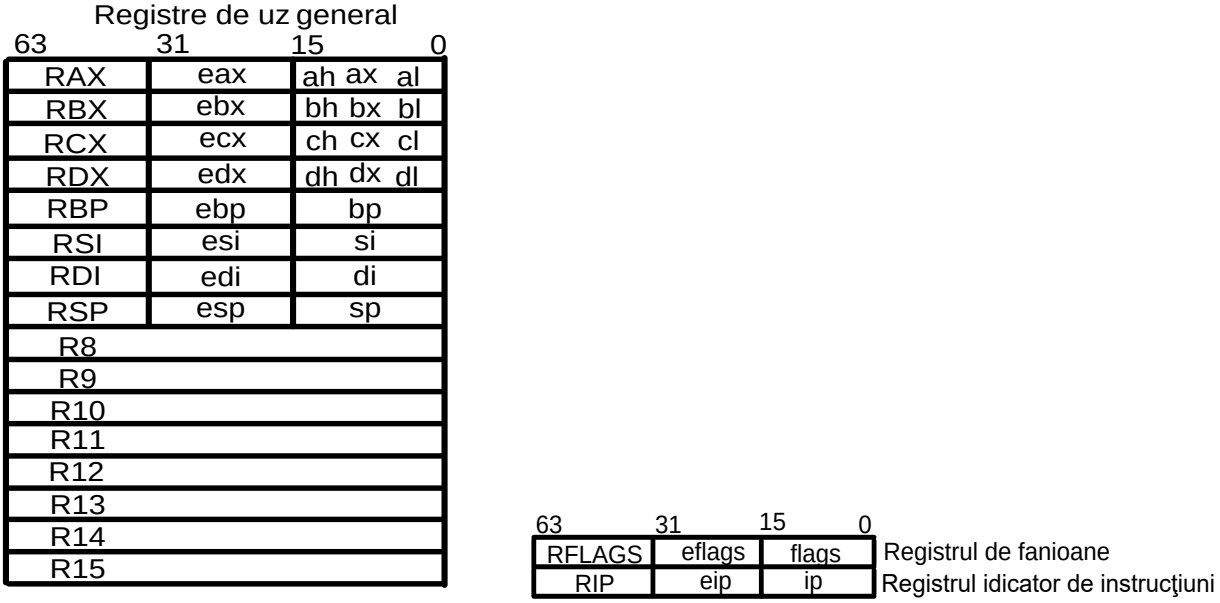


Figura 1.1

În figura 1.1 sunt prezentați regiştrii de uz general. Microprocesoarele pe 64 de biţi conţin şi alte seturi de regiştri care vor fi discutate la orele de curs.

Regiştrii pe 64 de biţi sunt indicați cu prefixul R (REX). Adresarea la fiecare din 16 regiştri se petrece ca la un registru pe 64-, 32-, 16- sau 8 biţi (se folosesc numai biţii inferiori). Structura şi destinaţia regiştrilor va fi detaliată în subcapitolele următoare, în descrierea modurilor de compatibilitate.

1.2.1 Modul de compatibilitate pe 16 de biţi

Pentru modul de compatibilitate pe 16 de biţi (modul 8086), sunt utilizați numai părţile inferioare pe 16 biţi (figura 1.2) ai regiştrilor microprocesoarelor pe 64 (32) de biţi, setul de regiştri este următorul:

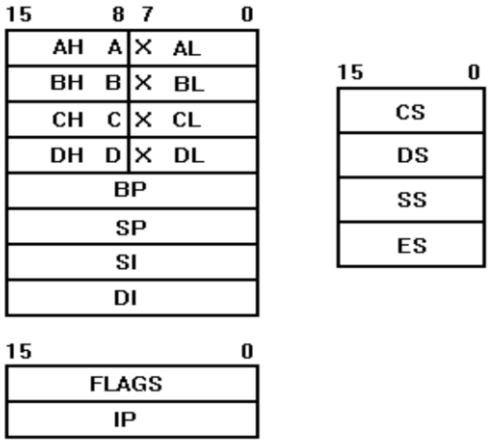


Figura 1.2

Toţi regiştrii sunt de 16 biţi. O serie de regiştri (AX, BX, CX, DX) sunt disponibili şi la nivel de octet, părţile mai semnificative fiind AH, BH, CH şi DH, iar cele mai puţin semnificative, AL, BL, CL şi DL. Denumirile regiştrilor sunt:

- AX - registru acumulator
- BX - registru de bază general
- CX - registru contor
- DX - registru de date
- BP - registru de bază pentru stivă (base pointer)
- SP - registru indicator de stivă (stack pointer)
- SI - registru index sursă
- DI - registru index destinaţie

Registrul notat FLAGS cuprinde flagurile (biţi indicatori) procesorului, sau bistabililor de condiţie, iar registrul IP (instruction pointer) este registrul de instrucțiuni.

Regiştrii de date (AX, BX, CX, DX) pot fi folosiţi pentru memorarea datelor, însă unele din aceştea sunt folosiţi special pentru alte scopuri. De exemplu, registrul CX este folosit implicit ca contor în instrucțiunea LOOP de ciclare, care îl modifică şi-i testează valoarea, registrul DX este folosit în instrucțiunile de împărțire si înmulțire, iar registrul BX este folosit pentru adresarea datelor (operanzilor).

În modul 8086 unui program scris în limbaj de asamblare se alocă patru zone de lucru în memorie: o zonă pentru date, o zonă pentru instrucțiuni, o zonă specială pentru stivă şi o zonă suplimentară pentru date. Fiecare din aceste zone pot avea până la 64K octeți şi poartă denumirea de segment. Astfel există segmentul de date (Data Segment), segmentul de instrucțiuni, cod (Code Segment), segmentul de stivă (Stack Segment) si segmentul suplimentar de date (Extra Segment).

Este posibil ca cele 4 segmente sa fie suprapuse total sau parțial. Adresele de început ale acestor segmente se află în 4 regiştri segment:

Denumirile regiştrilor de segment sunt:

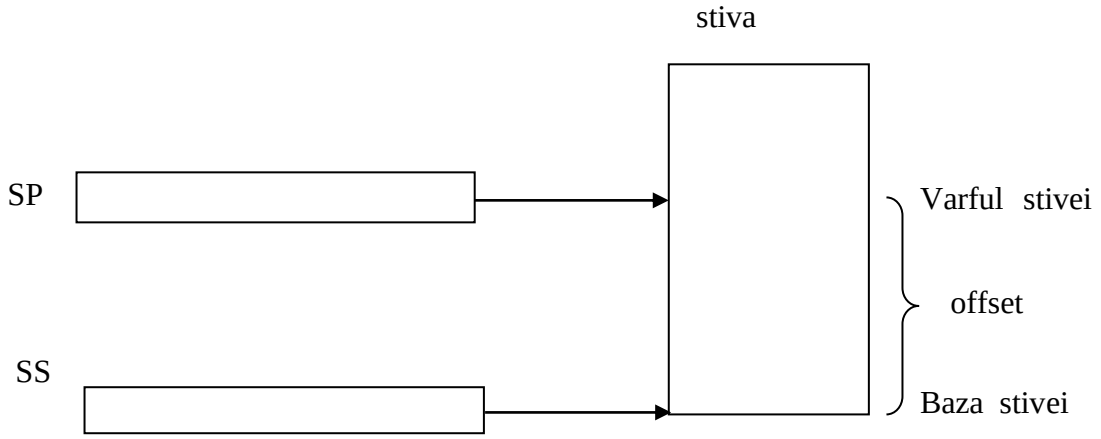
- CS - registru de segment de cod (code segment)
- DS - registru de segment de date (data segment)
- SS - registru de segment de stivă (stack segment)
- ES - registru de segment de date suplimentar (extra segment)

Se observă că denumirile regiştrilor de segment corespund zonelor principale ale unui program executabil. Astfel, perechea de regiştri (CS:IP) va indica totdeauna adresa următoarei instrucţiuni care se va executa, iar perechea (SS:SP) indică totdeauna adresa vârfului stivei. Regiştrii DS şi ES conţin adresele segmentelor de date şi sunt folosite pentru a accesa date.

Dacă segmentul de date începe de la locația de memorie 1234h atunci DS va conține valoarea 1234h. Există instrucțiuni pentru încărcarea adreselor de memorie în regiştrii segment.

Regiştrii pointer (SP si BP) se folosesc pentru calculul offset-ului (distanţei faţă de începutul unui segment) din cadrul segmentului. Cei doi regiştri pointer sunt: pointerul de stiva SP (Stack Pointer) si pointerul de baza (Base Pointer). SP si BP sunt regiştri de 16 biti.

Registrul SP reţine adresa efectivă (offset-ul) a vârfului stivei. Adresa fizică a vârfului stivei SS:SP este dată de perechea de regiştri SS si SP, registrul SS conține adresa de început al segmentului de stiva iar SP conține offset-ul din acest registru (adică distanţa in octeți de la începutul registrului de stiva):



Registrul BP este folosit la calculul offset-ului din interiorul unui segment. De exemplu poate fi folosit ca pointer al unei stive proprii, dar nu a procesorului. Este folosit în principal pentru adresarea bazată indexată a datelor.

Regiştrii de index, de 16 biti sunt: SI (Source Index) si DI (Destination Index). Regiştrii de index sunt folosiţi pentru accesul la elementele unui tablou sau a unei tabele. Aceşti regiştri sunt folosiţi îndeosebi în prelucrarea șirurilor de caractere.

Registrul de instrucțiuni (Instruction Pointer) conține offsetul curent în segmentul de cod. Adică adresa efectivă a următoarei instrucțiuni de executat din segmentul de cod curent. După executarea instrucțiunii curente, microprocesorul preia din IP adresa următoarei instrucțiuni de executat şi incrementează corespunzător valoarea lui IP, cu numărul de octeți ai codului instrucțiunii ce va fi executată. Uneori acest registru se numește numărător de program.

Registrul de flaguri (fanioane) (bistabili de condiţie) al modului 8086 are configuraţia din figura 1.3. O serie de flaguri sunt flaguri de stare: acestea sunt poziţionate la 0 sau la 1 ca urmare a unor operaţii aritmetice sau logice, conțin informații despre ultima instrucțiune executată. Celelalte flaguri controlează anumite operații ale procesorului.

Din cei 16 biți ai registrului sunt folosiți 9 biți: 0, 2, 4, 6 – 11.

Aproape toate instrucțiunile limbajului de asamblare afectează biții de stare.

Semnificația fagurilor este următoarea:

– CF (Carry Flag, bistabil de transport) - semnifică un transport sau un împrumut din/în bitul cel mai semnificativ al rezultatului, de exemplu la operaţii de adunare sau de scădere.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	OF	DF	IF	TF	SF	ZF	-	AF	-	PF	-	CF

Figura 1.3 - Registrul de flaguri al procesorului 8086

– PF (Parity Flag, flag de paritate) - este poziţionat în aşa fel încât numărul de biţi egali cu 1 din octetul cel mai puţin semnificativ al rezultatului, împreună cu flagul PF, să fie impar; altfel formulat, suma modulo 2 (XOR) a tuturor biţilor din octetul c.m.p.s. şi a lui PF să fie 1.

– AF (Auxiliarry Carry Flag, bistabil de transport auxiliar) - indică un transport sau un împrumut din/în bitul 4 al rezultatului.

– ZF (Zero Flag, bistabil de zero) - este poziţionat la 1 dacă rezultatul operaţiei este 0.

– SF (Sign Flag, bistabil de semn) - este poziţionat la 1 dacă b.c.m.s. al rezultatului (bitul de semn) este 1.

– OF (Overflow Flag, bistabil de depăşire) - este poziţionat la 1 dacă operaţia a condus la o depăşire de domeniu a rezultatului (la operaţii cu sau fără semn).

– TF (Trap Flag, bistabil de urmărire) - dacă este poziţionat la 1, se forţează o întrerupere, pe un nivel predefinit, la execuţia fiecărei instrucţiuni; acest fapt este util în programele de depanare, în care este posibilă rularea pas cu pas a unui program.

– IF (Interrupt Flag, bistabil de întreruperi) - dacă este poziţionat la 1, procesorul ia în consideraţie întreruperile hardware externe; altfel, acestea sunt ignorate.

– DF (Direction Flag, bistabil de direcţie) - precizează sensul (crescător sau descrescător) de variaţie a adreselor la operaţiile cu şiruri de octeţi sau de cuvinte.

Flagurile CF, PF, AF, ZF, SF şi OF sunt numite flaguri de stare (aritmetice). Flagurile TF, IF şi DF sunt numite flaguri de control.

1.2.2 Modul de compatibilitate pe 32 de biţi (microprocesor pe 32 de biţi)

Pentru modul de compatibilitate pe 32 de biţi , sunt utilizate numai părţile inferioare de 32 biţi (figura 1.4) ai regiştrilor microprocesoarelor pe 64 de biţi şi setul de regiştri utilizat în lucrările de laborator este următorul:

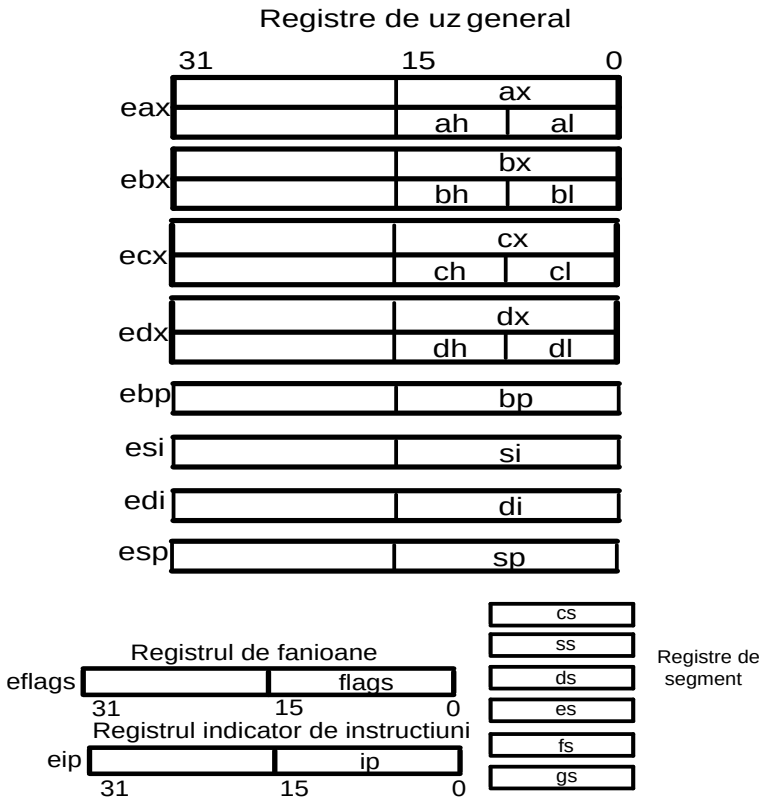


Figura 1.4

Microprocesoarele pe 32 de biţi sunt compatibile ca arhitectură cu cele pe 16 de biţi, prin aceea ca regiştrii pe 16 de biţi se regăsesc ca subregiştri ai regiştrilor de 32 de biți. Pentru accesarea regiştrilor de 32 de biţi a fost adăugat un set de instrucţiuni.

Regiştrii din figură, de exemplu al, ah, ax , indică regiştri pe 8 şi 16 de biţi ai registrului extins eax pe 32 de biţi (prefix „e” (Extended)).

Regiştrii generali ax, bx, cx, dx, si, di, bp şi sp pe 16 de biţi fac parte din regiştrii generali de 32 de biţi ai microprocesoarelor de 32 de biţi extinşi: eax, ebx, ecx, edx, esi, edi, ebp si esp. Primii 16 biți din aceşti regiştri sunt regiştri generali ai microprocesoarelor de 16 biti.

Analog regiştrii IP si FLAGS pe 16 de biţi sunt extinşi la 32 de biţi in cazul regiştrilor EIP şi ELAGS de 32 de biţi. Registrul FLAGS se regăsește în primii 16 biți ai registrului EFLAGS.

Regiştrii segment au fost păstraţi pe 16 de biţi, dar s-au adăugat doi noi regiştri FS şi GS.

Pe lângă aceşti regiştri, microprocesoarele de 32 (64) de biți dispun de alţi regiştri de control, de gestionare a adresei, de depanare şi de test, care diferă de la un tip de procesor la altul fiind folosite în principal de programele de sistem.

Semnificația regiştrilor segment în cazul microprocesoarelor de 32 de biţi a fost modificată, ele sunt folosite ca selectoare de segment. În acest caz ele nu indică o adresă de segment, ci un descriptor de segment care precizează adresa de bază a segmentului, dimensiunea acestuia şi drepturile de acces asociate acestuia. Astfel adresa de bază poate fi specificată pe 32 de biţi iar dimensiunea unui segment să fie de până la 4 GB.

1.3 Configurarea Visual Studio pentru aplicații pe 32 de biți

Instalați Visual Studio Express 2015 for Windows Desktop. Puteți verifica dacă Microsoft Assembler este instalat prin localizarea fișierului **ml.exe** în directorul de instalare Visual Studio, calea **C: \ Program Files (x86)\ Microsoft Visual Studio 14.0 \ vc \ bin.**

Instalaţi fişierul **Irvine_7th_Edition.msi** care conţine proiecte, o mulţime de exemple şi biblioteci necesare elaborării programelor. Implicit el se instalează în folderul **C:\Irvine**. Dar puteţi crea şi proiecte noi, cum este indicat în ANEXA A.

Este util rularea aplicaţiei pe 32(64) biţi fără depănare. Pentru aceasta este necesar să adăugăm o nouă comandă pentru meniul **Debug**: **Start Without Debugging.** Procedura este următoarea:

- Din meniul **Tools**, , selectați **Customize**.
- Selectați butonul **Commands**.
- Selectați **Menu** bar (radio button).
- Faceți clic pe butonul **Add Command**.
- Selectați **Debug** din **Categories** list.
- Selectați **Start Without Debugging** în caseta din dreapta.
- Executaţi clic pe butonul **OK.**
- Executați clic pe butonul **Close.**

Setarea Tab Size în 5

Selectaţi **Options** din meniul **Tools**. Selectaţi **Text Editor**, Selectaţi **All Languages**, şi selectaţi **Tabs**: Setaţi **Tab Size** şi **Indent Size** to 5 (figura 1.5). Clic **OK.**

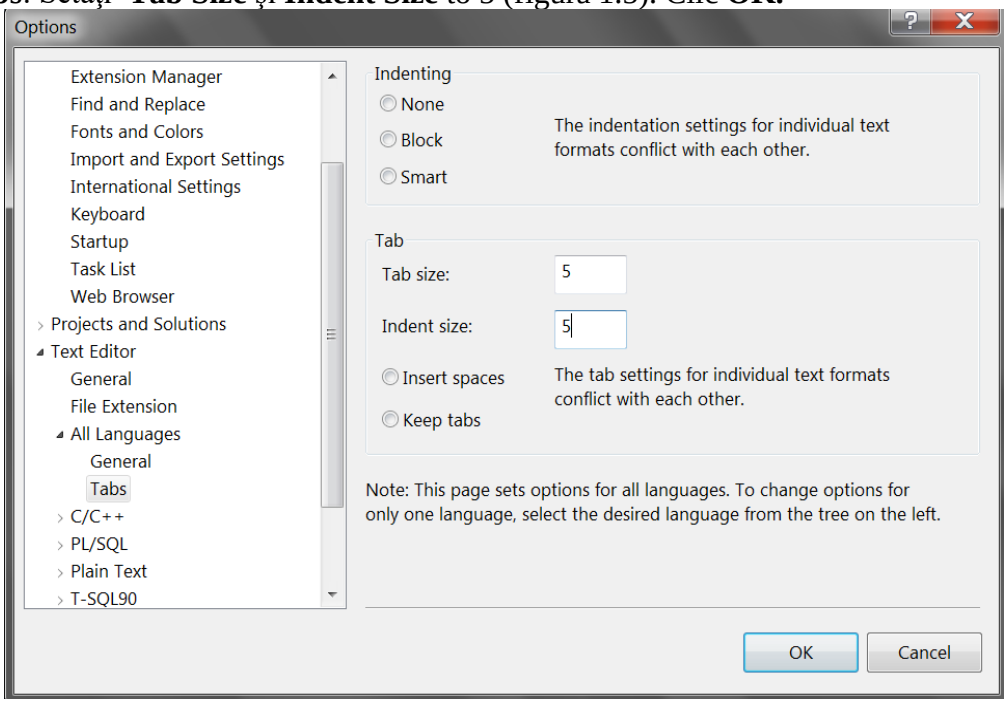


Figura 1.5

Dacă aţi instalat pe disc C:\Irvine, în Visual Studio selectaţi **File> Open Project** şi selectaţi calea spre fişierul C:\Irvine (locul instalării Irvine_7th_Edition.msi):

C:\Irvine > Examples > Project32 > Project.sln

În fereastra **Solution Explorer** va apărea **Project** (figura 1.6).

Executaţi clic dreapta pe **Project** şi selectaţi **Add> Existing Item** şi selectaţi calea spre un exemplu de program sursă, de exemplu **Colors,** din Irvine , ch5.

C:\Irvine > Examples > ch5 > 32 bit > **colors.asm** şi executaţi clic **Add** (figura 1.6).

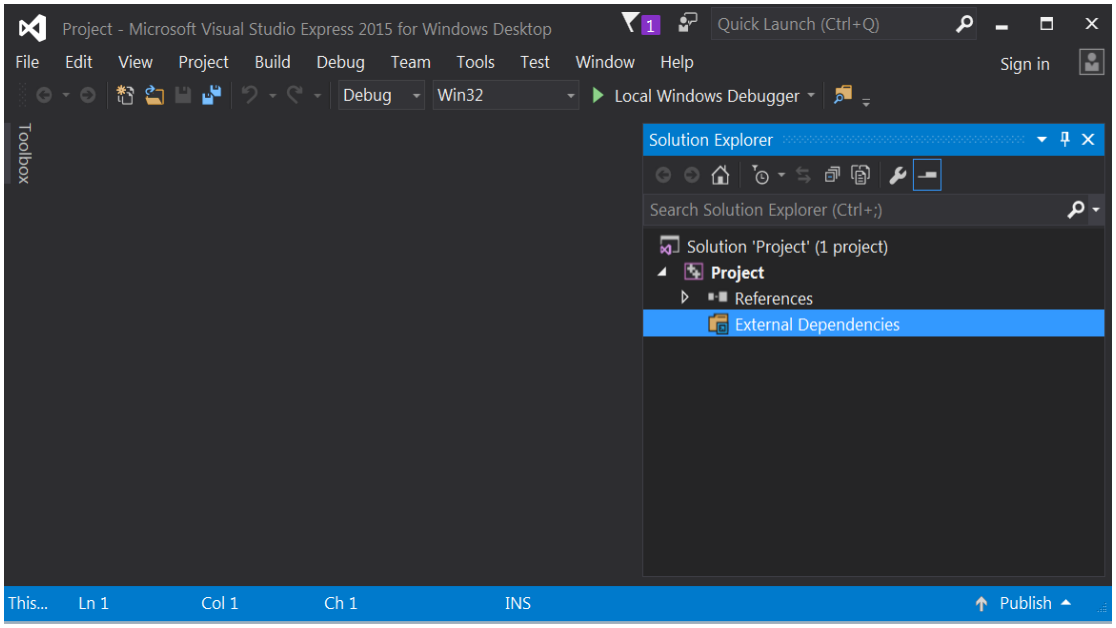


Figura 1.6

Dublu clic pe **Project** și pe **colors.asm** (figura 1.7).

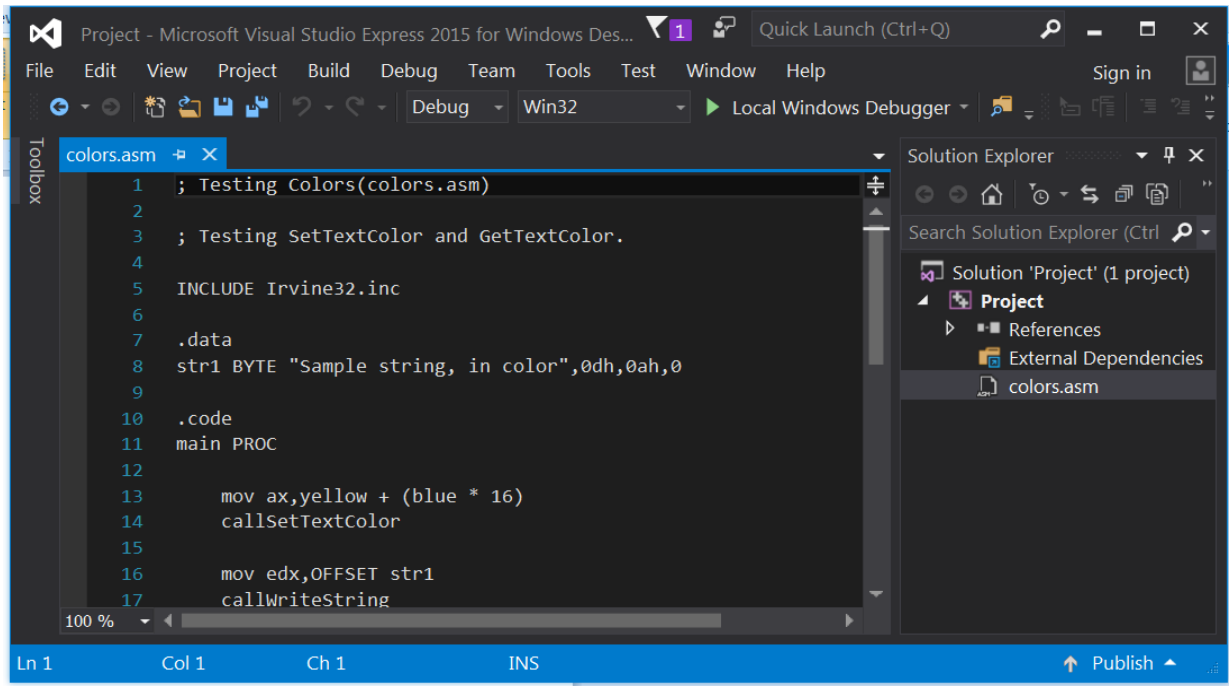


Figura 1.7

În meniul **Build**, selectați **Build Solution.** În fereastra de ieșire pentru Visual Studio, în partea de jos a ecranului, vor apărea mesajele de dezvoltare (asamblare și editarea legăturilor (link)) a programului (figura 1.8) .

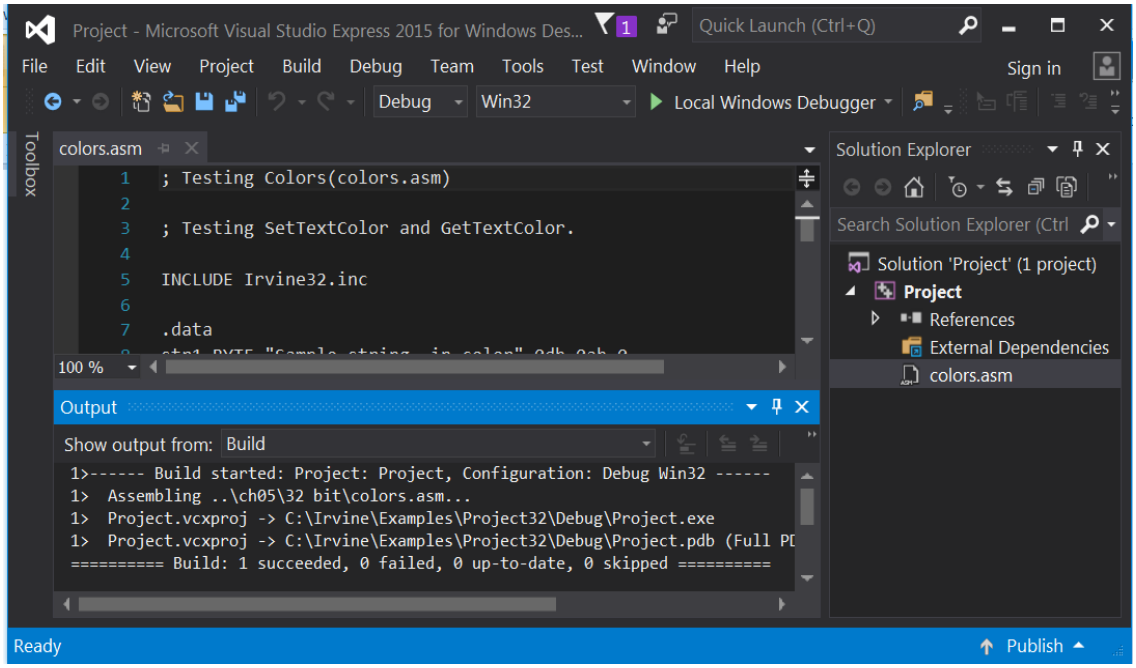


Figura 1.8

În meniul **Debug**, selectaţi **Start Without Debugging.** Va apărea o fereastră cu rezultatul execuţiei programului – afişarea unui text şi blocului de regiştri colorat (figura 1.9)**.**

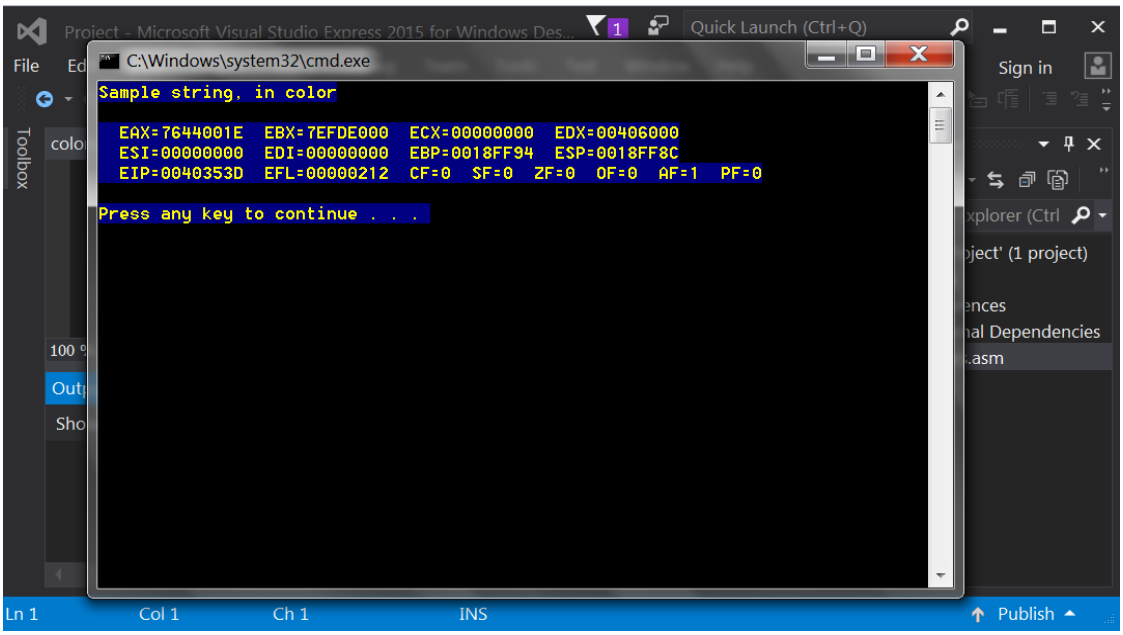


Figura 1.9

Pentru depănarea programului pas cu pas, executaţi clicuri F10 (figura 1.10), săgeata galbenă indică următorul rând din program care va fi executat. Puteţi afişa starea regiştrilor microprocesorului selectând meniul **Debug**, selectaţi **Windows** şi **Registers**. Pentru afişarea stării variabilelor selectaţi meniul **Debug**, selectaţi **Windows**, **Watch**>**Watch1**.

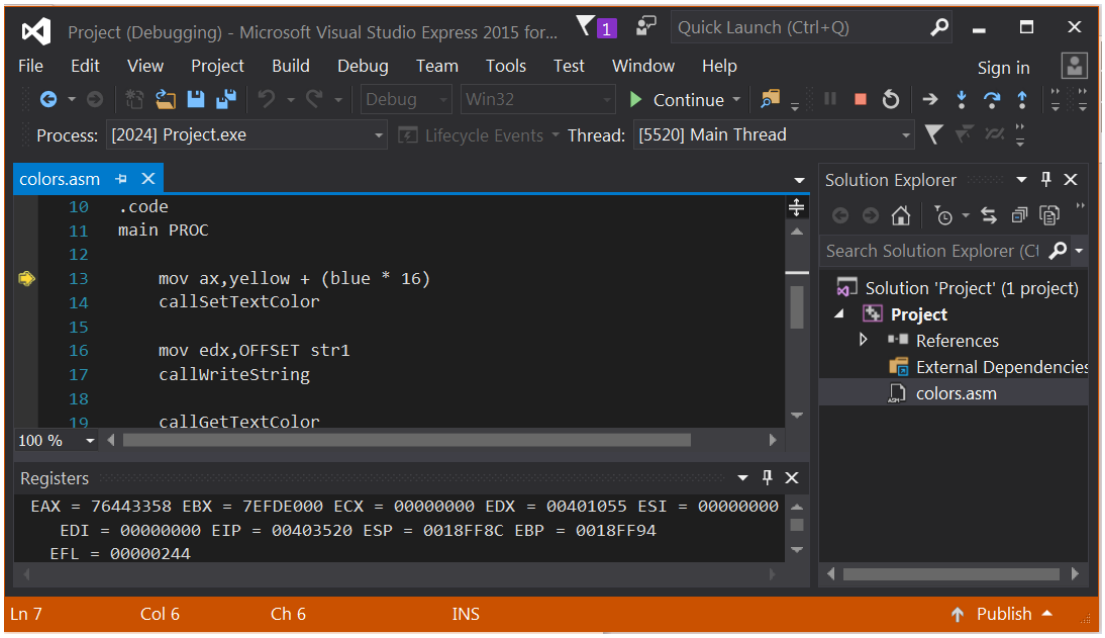


Figura 1.10

1.4 Elaborarea aplicațiilor pe 16 de biți

Pentru ca aplicaţiile pe 16 de biţi să fie executate corect, este necesar să modificaţi corect calea instalării şi versiunea Visual Studio, calea instalării fişierului **Irvine,** în fişierul **make16_vs2013.bat** (editaţi în Notepad) conform figurii 1.11. Fişierul **make16_vs2013.bat** se află în folderul Irvine.

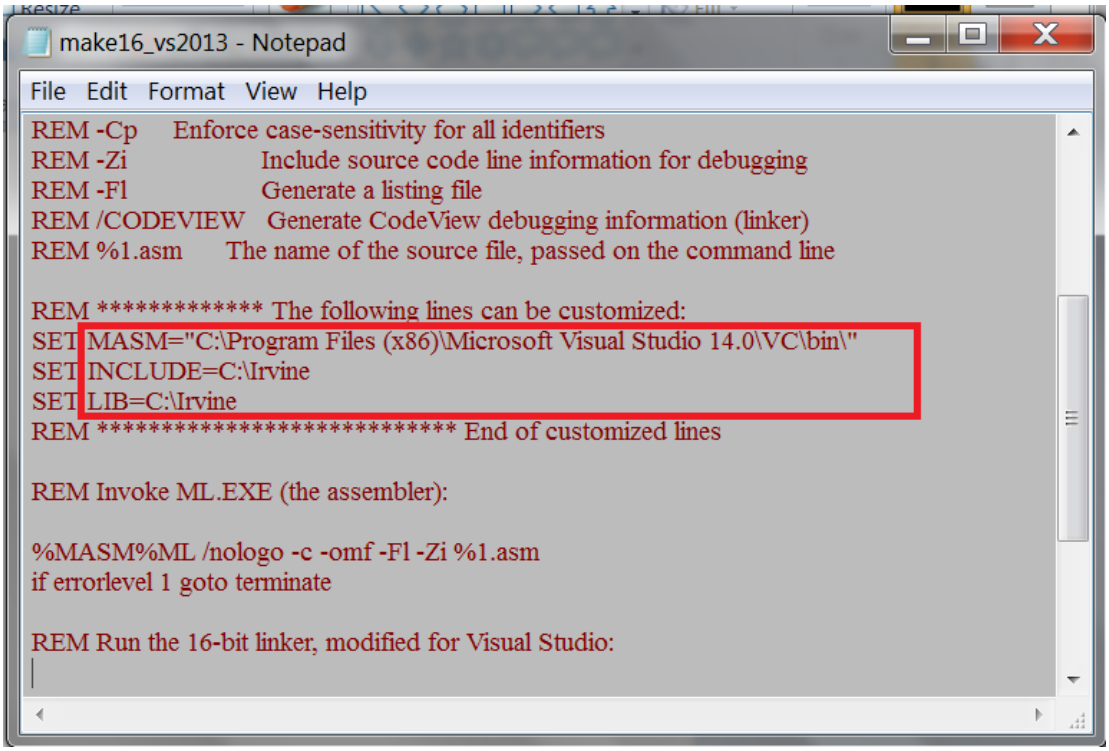


Figura 1.11

Pentru a rula aplicații de 16-biți, este necesar să adăugați două comenzi în meniul Visual Studio **Tools**. Pentru a adăuga o comandă, selectați **External Tools** din meniul **Tools**. În fereastra apărută executaţi clic pe butonul **Add** şi introduceți o nouă linie de comandă cu titlul **Build 16-bit ASM** (figura 1.12). În linia **Command - C:\Irvine\make16_vs2013.bat**, calea spre fişierului **make16_vs2013.bat**. **Arguments** şi **Initial directory** le puteţi seta executând clic pe butoanele ►(dreapta), clic **Aply**.

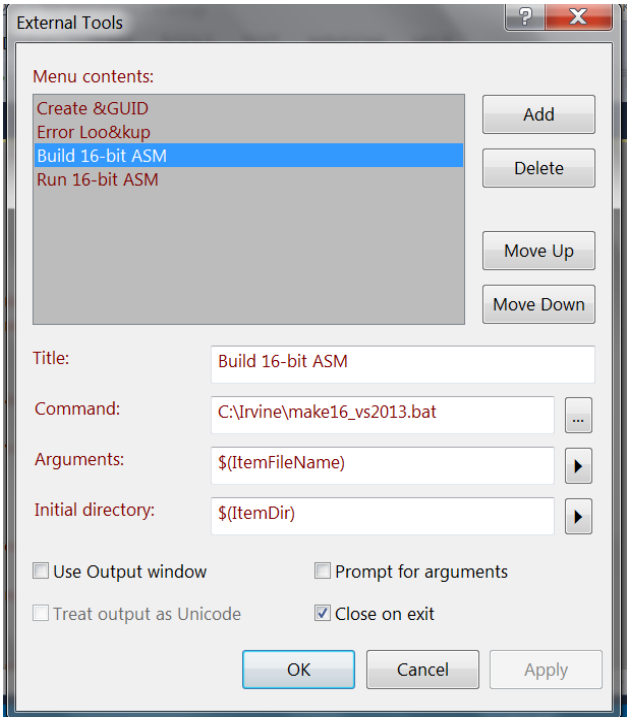


Figura 1.12

executați clic pe butonul **Add** şi introduceți o nouă linie de comandă cu titlul **Run 16-bit ASM** (figura 1.14). În linia **Command** indicați calea spre **DOSBox.exe.**. **Arguments** şi **Initial directory** le puteţi seta executând clic pe butoanele ►(dreapta), bifaţi **Prompt for arguments.** (**Close on exit** nu se bifează), clic **Aply**.

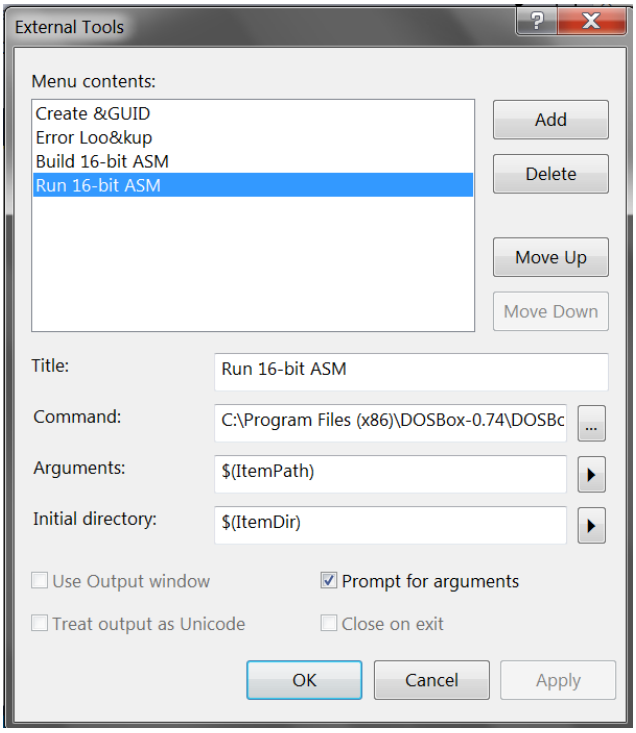


Figura 1.14

Executând clic - **Run 16-bit ASM,** în fereastra **Arguments: <u>introduceți denumirea fişierului cu extensia .exe.</u>**

Pentru ca rândurile codului să fie numerotate selectați **Tools**> **Options**> **Text Editor**> **All Languages** > şi bifaţi, în fereastra din dreapta, **Line Numbers**.

1.5 Dezvoltarea programelor pe 16, 32 de biţi

Exemplu de program pe 16 de biţi:

;exemplu de program pe 16 biti **INCLUDE Irvine16.inc** **.DATA** **Promt DB 'Doresti sa devii programator?(da/nu)-[y/n]$'** **Dad DB 13,10,'Vei deveni!',13,10,'$'** **Nud DB 13,10,'Vei deveni filosof!',13,10,'$'**	
.CODE	;directivă ce declară începutul segmentului de cod
main PROC	;se indică procedura cu numele *main*
mov ax,@data	;Initializarea segmentului de date **ds**
mov ds,ax	;cu adresa datelor
mov dx,OFFSET Promt	;în dx- deplasamentul (offset) sirului Promt
mov ah,9	;functia MSDOS, codul funcţiei 9 – afisarea sirului
Int 21h	; întreruperea 21h – apel la serviciul MSDOS
mov ah,1	; functia MSDOS, codul funcţiei 1 – introducerea de la tastatură(codul tastei)
Int 21h	;codul tastei in registrul **al**
cmp al,'y'	;compararea conţinutului registrului al cu codul ASCII a literei ***y***
jz IsDad	;salt condiţionat (jz-jump if zero), dacă rezultatul comparării este zero, salt la eticheta **IsDad**
cmp al,'n'	;compararea- din **al** se scade codul ASCII a literei ***n***
jz IsNud	;da, rezultatul comparării este zero, salt la eticheta **IsNud**
IsDad: mov dx,OFFSET Dad	; în dx- offsetul sirului Dad
Jmp SHORT Disp	Salt necondiţionat la eticheta ”Disp”
IsNud: mov dx,OFFSET Nud	; în dx- offsetul șirului ”Nud”
Disp: **mov ah,9**	;functia MSDOS – afisarea sirului
Int 21h	;apel la serviciul MSDOS
mov ah,1	; asteptarea unui clic

Int 21h	
Exit	; apel la procedura de iesirea din program, din fisierul **Irvine16.inc**
main ENDP	; sfârşitul procedurii **main**
END main	;finalizarea programului/ punctul de intrare in program

Se remarca urmatoarele:

1. Fişierul **Irvine16.inc** conţine un set de proceduri, macro-uri ce completează codul sursă necesare pentru dezvoltarea programelor în Visual Studio;

2.Directiva *.DATA* marchează începutul segmentului de date. In cazul nostru acest segment contine 3 şiruri de caractere, fiecărui caracter i se atribuie câte un byte (**DB**) în memorie, cu offset Prompt, Dad și Nud, valorile13,10 sunt tratate ca comenzi –sfârșitul rândului (LF), din rând nou (CR);

3. **main PROC** indică începutul procedurii main în care este inclus tot codul programului, **main ENDP** indică sfârşitul procedurii. Directiva *END main* indică sfârşitul programului (numit si **punctul de intrare in program**), tot codul plasat după această directiva va fi ignorat de asamblor;

4.Primele doua instrucțiuni inițializează registrul *ds*. Simbolul *@data* reprezintă numele segmentului creat cu directiva *.DATA* (sau adresa segmentului, adresa primului octet din acest segment);

5.Continutul variabilelor Prompt, Dad și Nud de tip șir de caractere se afișează pe ecran utilizând funcția *MSDOS* cu codul *9*. Șirurile pot fi de asemenea afișate pe ecran cu funcția *MSDOS* cu codul *40h*;

6.Întreruperea Int 21h apelează serviciul *MSDOS* (DOS Services), iar *int 10h* la serviciul BIOS, ca funcţia apelată să fie executată, codul ei este necesar să fie încărcat în registrul **al**.

Şablonul (template) unui program pe 16 de biţi este următorul:

```
;  This program
; Last update:
INCLUDE Irvine16.inc
.data

; aici definiti datele necesare
.code
main PROC
        mov ax,@data
        mov ds,ax

        ;aici plasati codul program

        exit
main ENDP
END main
```

Exemplu de program pe 32 de biţi:

```
INCLUDE Irvine32.inc
.data
Promt DB 'Doresti sa devii programator?(da/nu)-[y/n]',0
Dad DB 13,10,'Vei deveni!',13,10,0
Nud DB 13,10,'Vei deveni filosof!',13,10,0
.code
main PROC

mov edx,OFFSET Promt
call WriteString
call ReadChar
```

```
cmp al,'y'
jz IsDad
cmp al,'n'
jz IsNud

IsDad: mov edx,OFFSET Dad
call WriteString
jmp ex
IsNud: mov edx,OFFSET Nud
call WriteString

ex:
        exit
main ENDP
END main
```

Se remarca urmatoarele:

1. Se observă, că este acelaşi program dar elaborat ca o aplicație pe 32 de biţi, în care se utilizaeză procedurile din **Irvine32.inc,** pentru dezvoltarea programelor în Visual Studio;
2. Definirea şirurilor de caractere se finalizează cu zero, offset-ul şirurilor se încarcă în registrul **edx** (pe 32 de biţi).
3. Cu instrucţiunea **call** se apelează două proceduri WriteString – afişarea şirului şi ReadChar- introducerea unui caracter de la tastatură. Pentru afişarea unui şir de caractere, offset-ul şirului este necesar să fie încărcat în registrul **edx** şi apoi apelată procedura (funcţia). După apelarea procedurii ReadChar, în registrul **al** se va introduce codul tastei.

Exemplu de program pe 64 biţi:

Selectaţi În Visual Studio proiectul **C:\Irvine\Examples\Project64\Project64.sln** şi adăugați un exemplu de program din **C:\Irvine\Examples\ch3\64 bit\AddTwoSum_64.asm**.

```
; AddTwoSum_64.asm - Ch3 example.

ExitProcess proto

.data
sum qword 0

.code
main proc
        mov     rax,5
        add     rax,6
        mov   sum,rax

        mov   ecx,0
        call  ExitProcess

main endp
end
```

Se remarca urmatoarele:

1. Este un program ce adună două numere întregi şi rezultatul este salvat în variabila **sum** pe 64 biţi (qword).
2. Procedura de ieşire din program ExitProcess, şi alte proceduri utilizate, este necesar să le declarăm la începutul programului, ExitProcess proto.

1.6 MODUL DE LUCRU

1.Se vor asambla si rula exemplele de programe prezentate obţinându-se fișiere *.EXE,* ele se află în Project, cu numele Project.exe ;

2.Se vor obține și se vor analiza fișierele listing ale programelor Project.lst;

3.Se va rula sub *Debug* pas cu pas (clic F10) exemplele de programe actualizând Registers şi Watch1(pentru variabile);

1.7 Continutul referatului

Referatul va conține:
- tema și scopul lucrării,
- codurile sursă .asm (Ex. 16,32 de biţi) comentate;
- fișierele -listing (Ex. 16,32 de biţi);
- pașii rulării programelor În Debug;
- Concluzii.

Dacă utilizaţi Professional, Ultimate și Premium

Visual Studio **Professional**, edițiile **Ultimate** și **Premium** suporta mai multe limbaje de programare și tipuri de aplicații. Configurarea C ++ ca limbaj de programare se recomandă pentru programarea în limbaj de asamblare, astfel se recomandă următoarele etape:

1) Selectaţi din meniul **Tools** > **Import and Export Settings**.
2) Selectaţi butonul "Import selected environment settings".
3) Selectaţi butonul "No, just import..." .
4) Selectaţi "Visual C++" din lista **Default Settings** şi executaţi clic pe butonul **Next** .
5) Executaţi clic pe butonul **Finish** , apoi clic pe butonul **Close**.

Anexa A Crearea unui nou proiect în Visual Studio

Selectaţi din meniu File> New>Project. În fereastra apărută figura A.1 selectaţi în caseta dreapta **Visual C++> Win32,** mijloc – **Win32 Console Application,** jos – **Name** – numiţi proiectul, de exemplu, **Project32**, în **Location** – indicaţi calea, de exemplu, spre **C:\Irvine\WORK** unde veți salva

proiectul și lucrările de laborator. Clic OK.

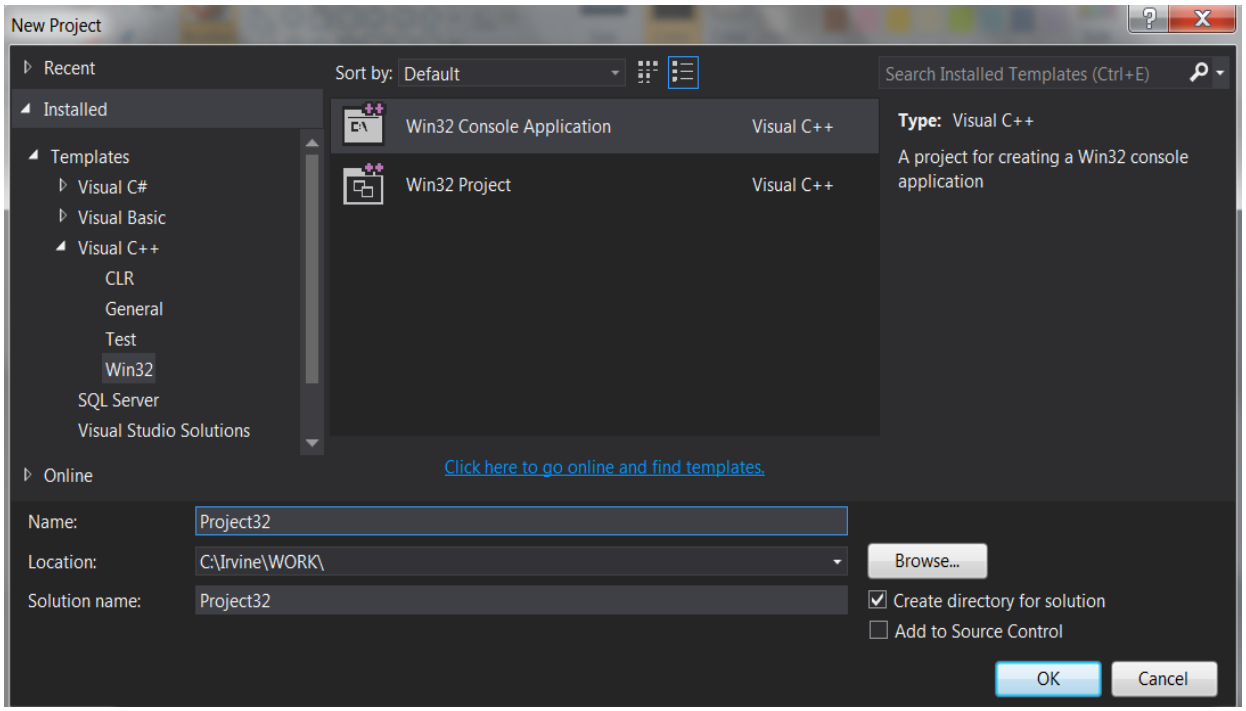


Figura A.1

În fereastra apărută executaţi clic **Application Settings,** bifaţi **Empty project (**figura A.2**).** Clic **Finish.**

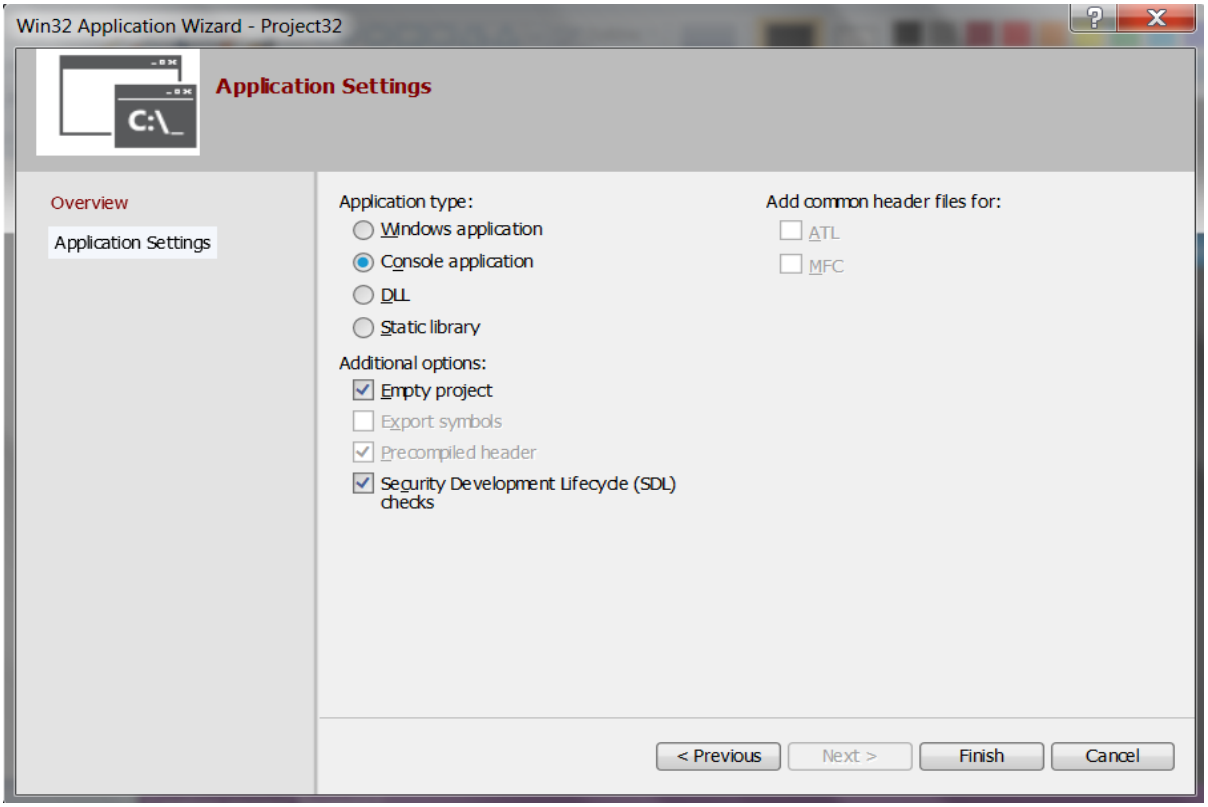


Figura A.2

În fereastra **Solution Explorer (dreapta)**, eliminaţi fișierele **Header Files, Resource Files, and Source Files, e**le nu sunt necesare. În aceeași fereastră, executaţi clic dreapta pe numele de proiect - **Project32** și selectați selectați **Build Dependencies > Build Customizations**. Va apărea o nouă fereastră (figura A.3), bifați caseta de selectare **MASM** și executaţi clic pe butonul **OK**.

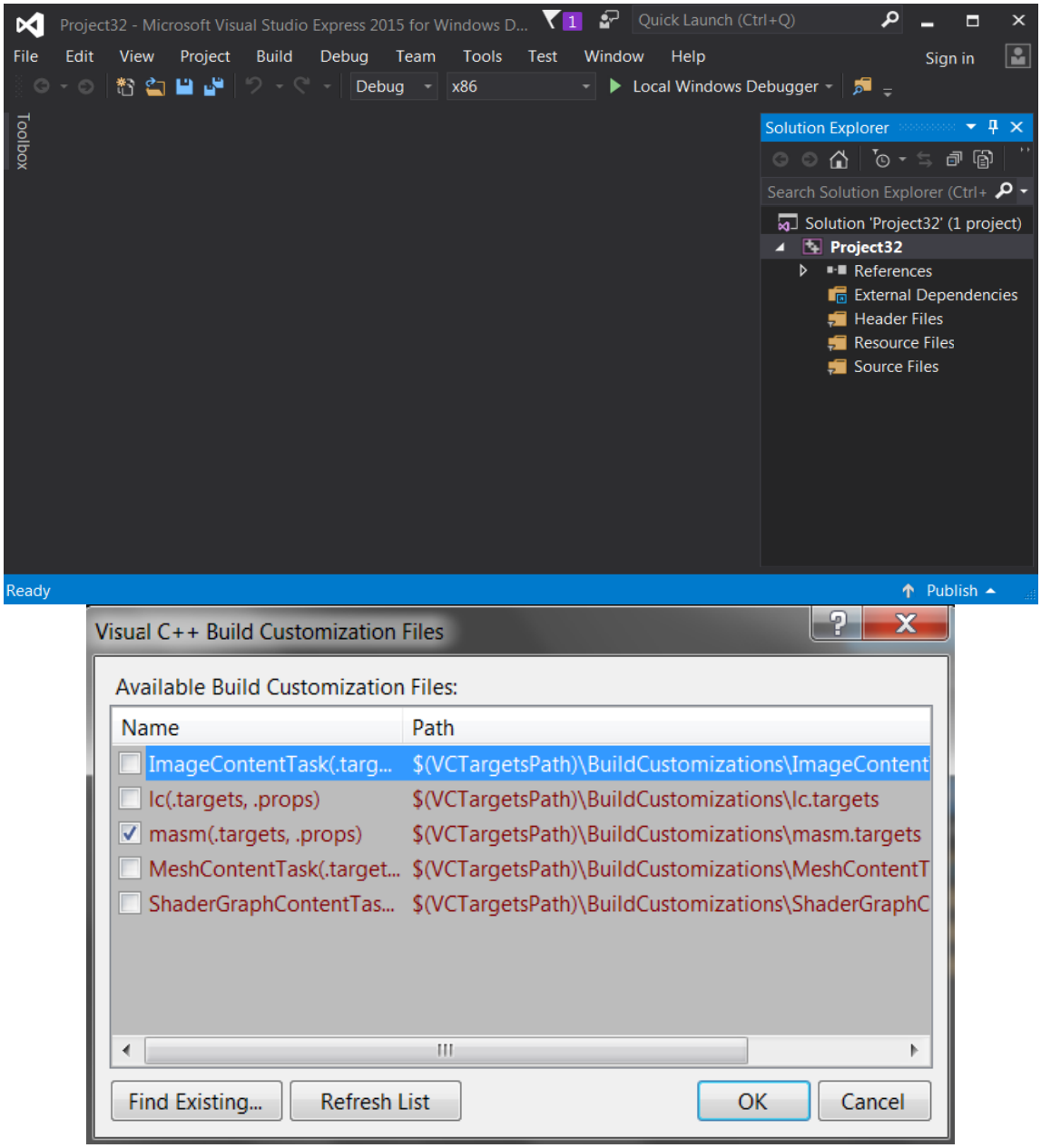


Figura A.3

Clic dreapta **Project32,** selectati **Properties**. Clic pe **Configuration Properties**. În **Configuration Properties** găsiți intrarea **Linker**.

Selectaţi **General** > **Additional Library Directories** şi introduceţi linia **c:\Irvine** în exemplul nostru, astfel linker-ul va găsi fișierul bibliotecă Irvine32.lib (figura A.4). Clic **Apply.**

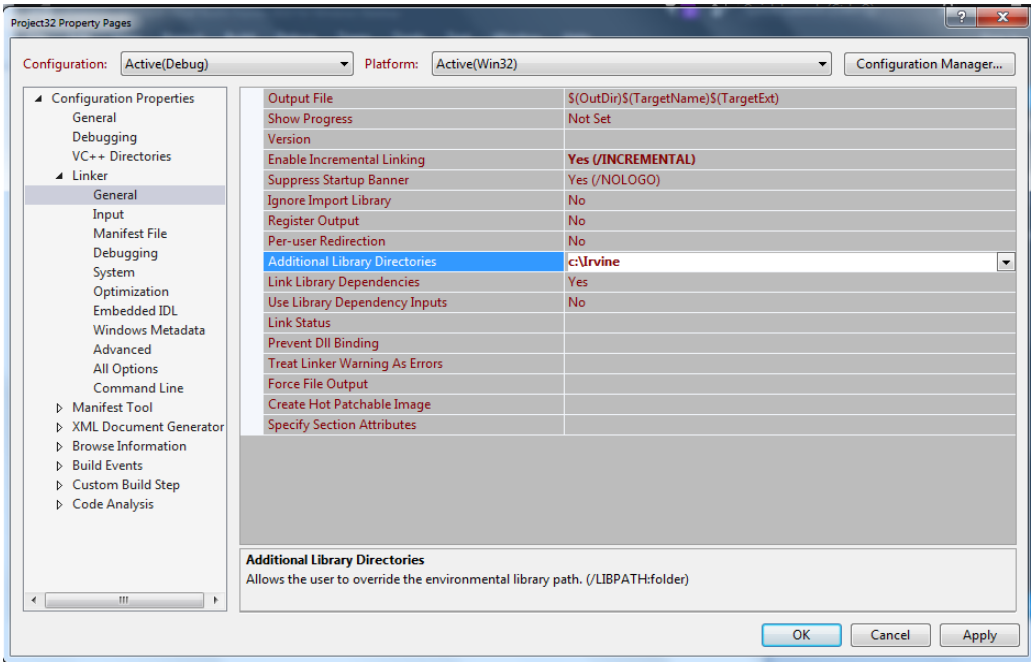


Figura A.4

Tot în **Linker** selectați **Input>Additional Dependencies** şi adăugaţi fişierul **irvine32.lib;** (figura A.5).
Numele fișierelor să fie separate prin punct și virgulă. Clic **Apply**.

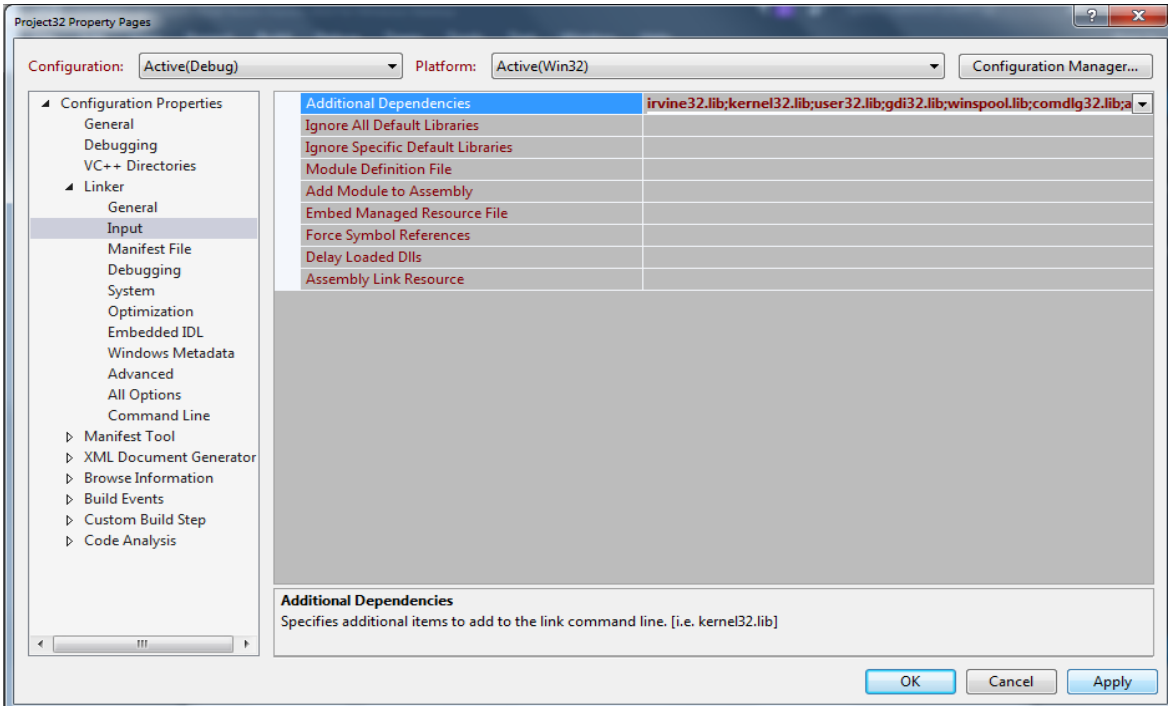


Figura A.5

Tot în **Linker** selectați **Debugging** (depanare). Verificați dacă opțiunea **Generate Debug Info** este setat **- Optimize for debugging (/DEBUG) (**figura A.6**).**

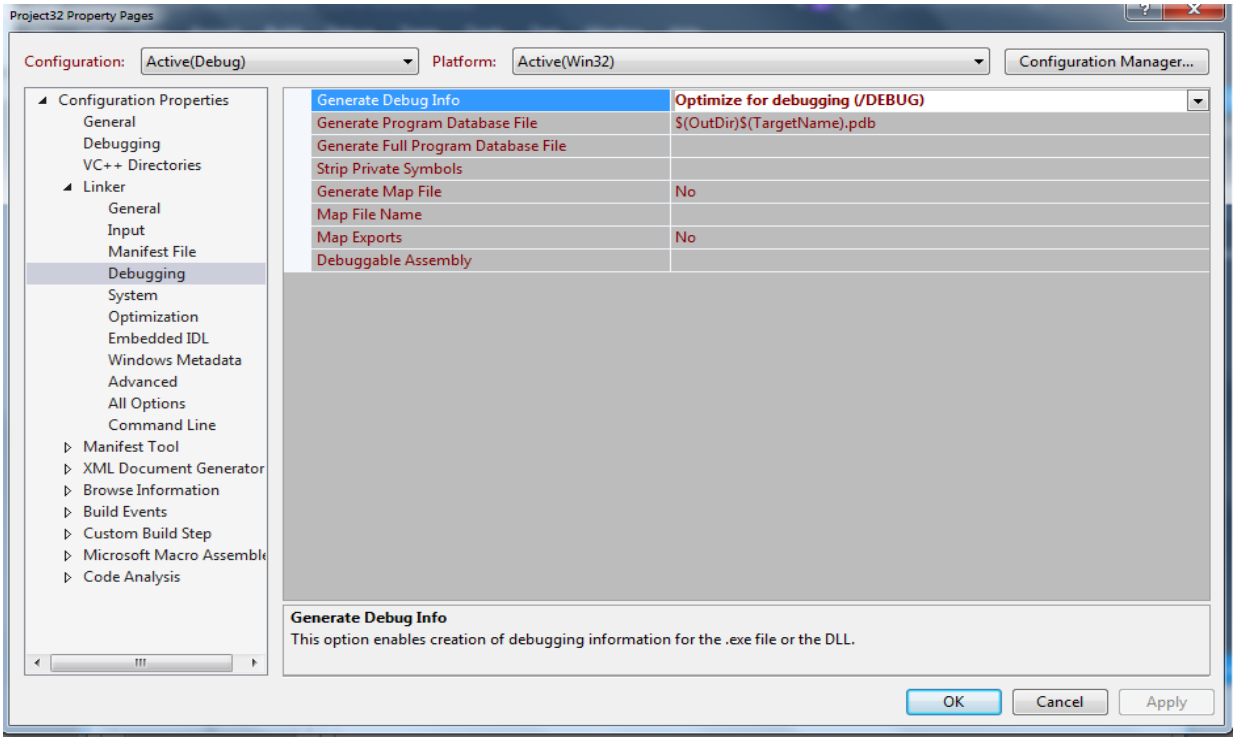


Figura A.6

Tot în **Linker** selectați **System**. Verificați dacă opțiunea **SubSystem** este setat **Console (**figura A.7**).**

Executaţi clic **Apply**.

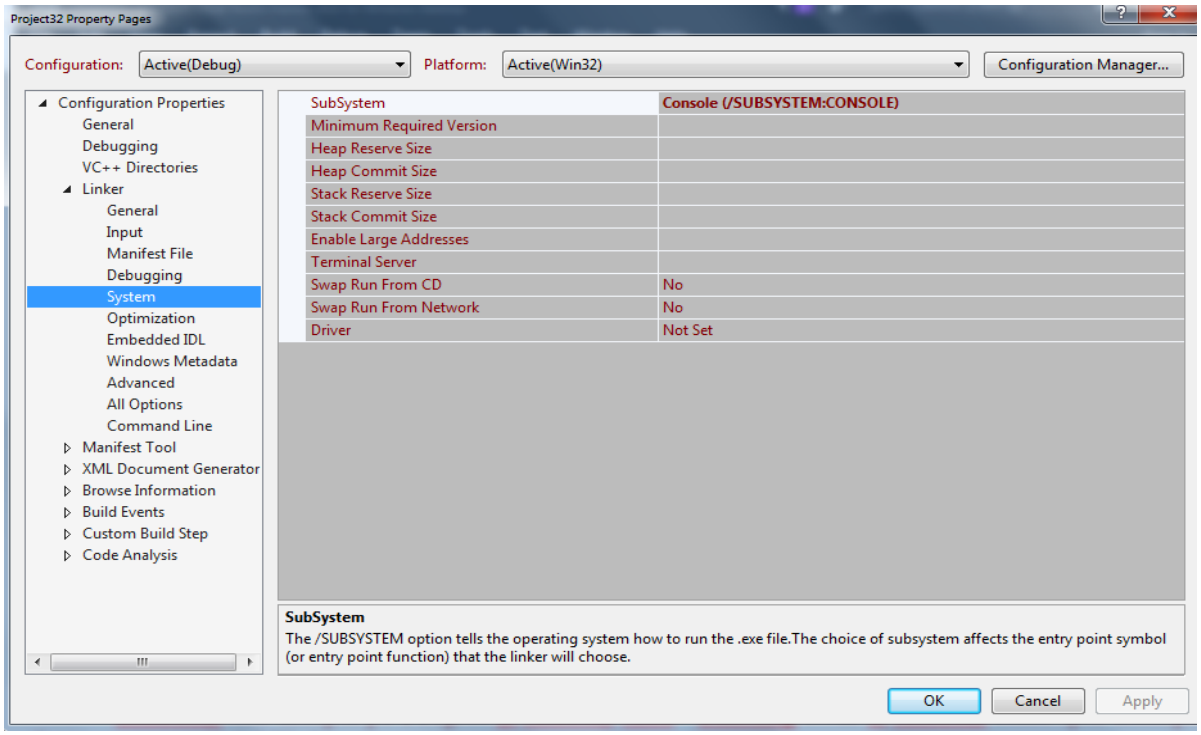
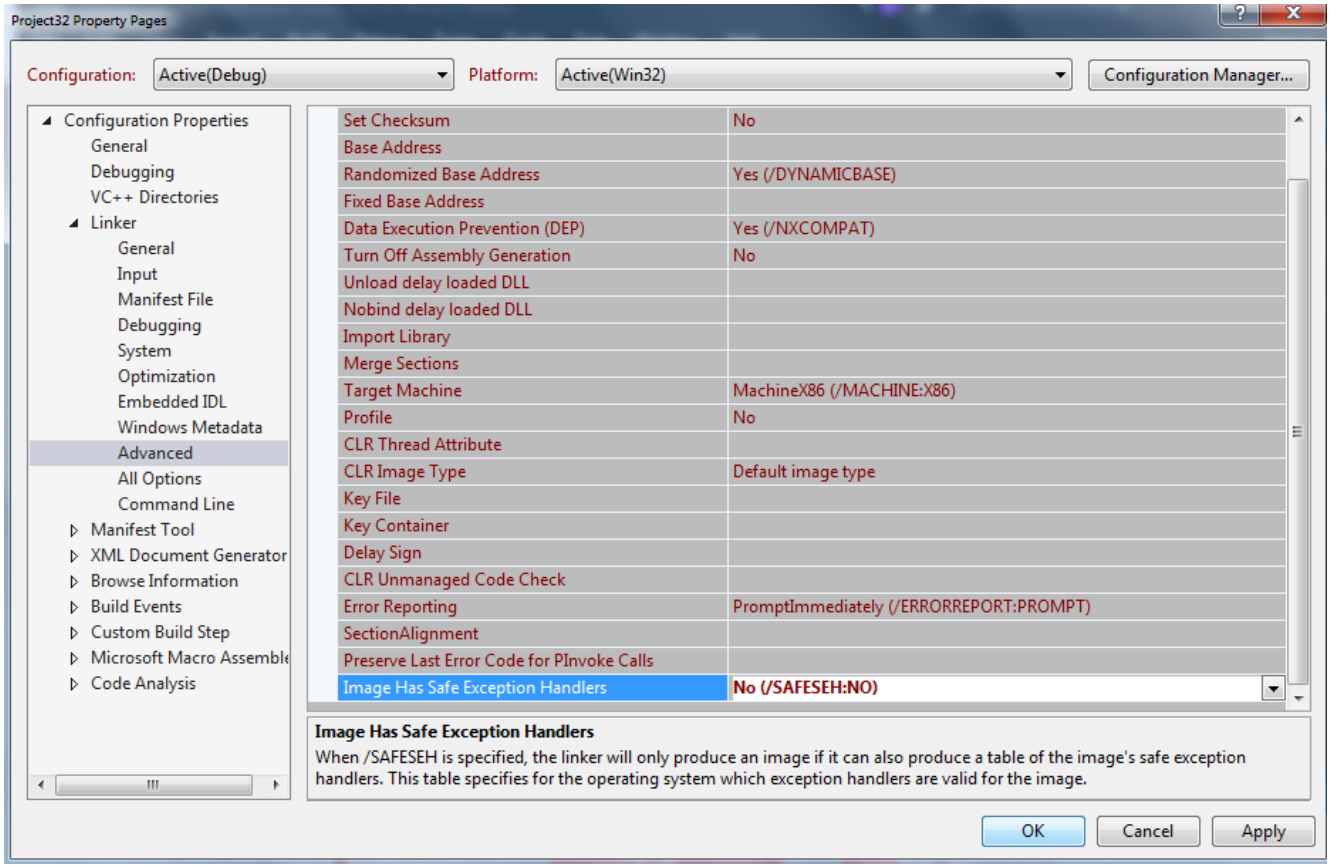


Figura A.7

Tot în **Linker** selectați **Advanced.** Setaţi opţiunea - **Image Has Safe Exception Handlers,** în - **No (/SAFESEH:NO).** Executaţi clic **OK.**



Executaţi clic dreapta pe **Project32** şi selectaţi **Add> Existing Item** şi selectaţi calea spre exemplul de program sursă **Colors.asm**

C:\Irvine\Examples\ch05\32 bit\Colors şi executaţi clic **Add** (figura A.8)

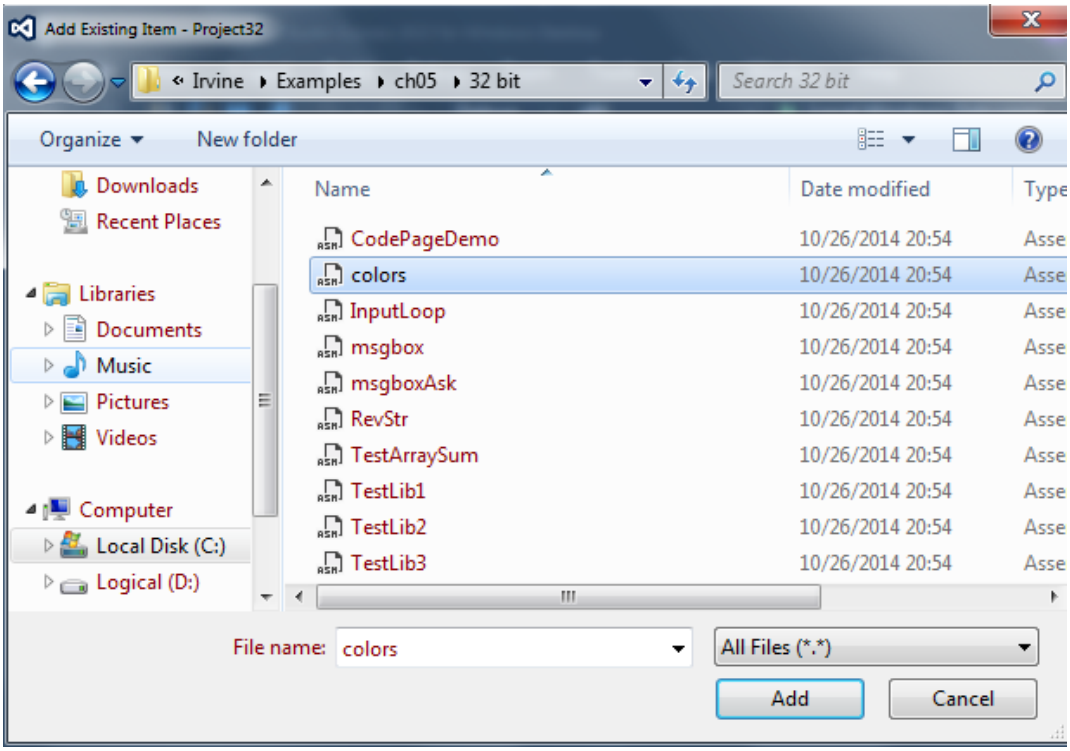


Figura A.8

În fereastra **Solution Explorer** va apărea **Colors.asm.**

Executaţi clic dreapta pe **Project32** şi selectaţi **Properties** > **Configuration properties** > **Microsoft Macro Assembler**> **General**

Modificaţi **Include Paths** cu calea spre fişierul **C:\irvine**. Aceasta indică calea spre fişierele cu

extensia "**.inc**" (figura A.9). Clic **Apply.**

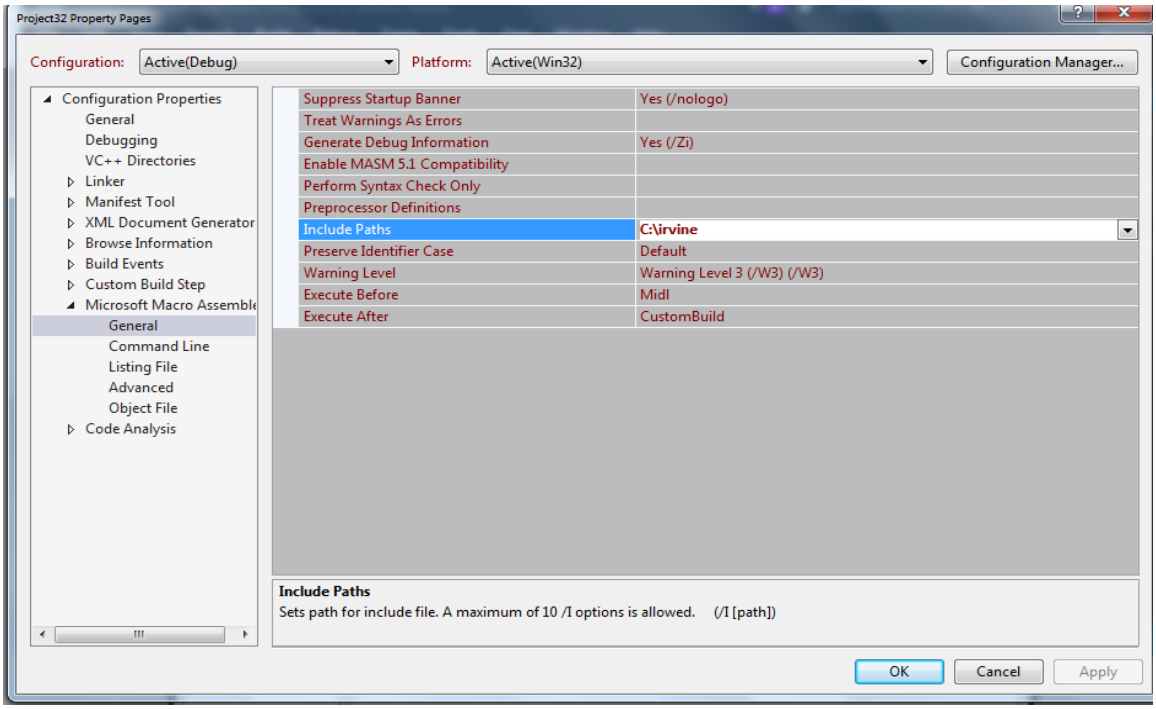


Figura A.9

Selectaţi **Listing File** din listă şi în linia **Assembled Code Listing File** (figura A.10), adăugaţi **$(ProjectName).lst**. Clic **Apply, OK.**

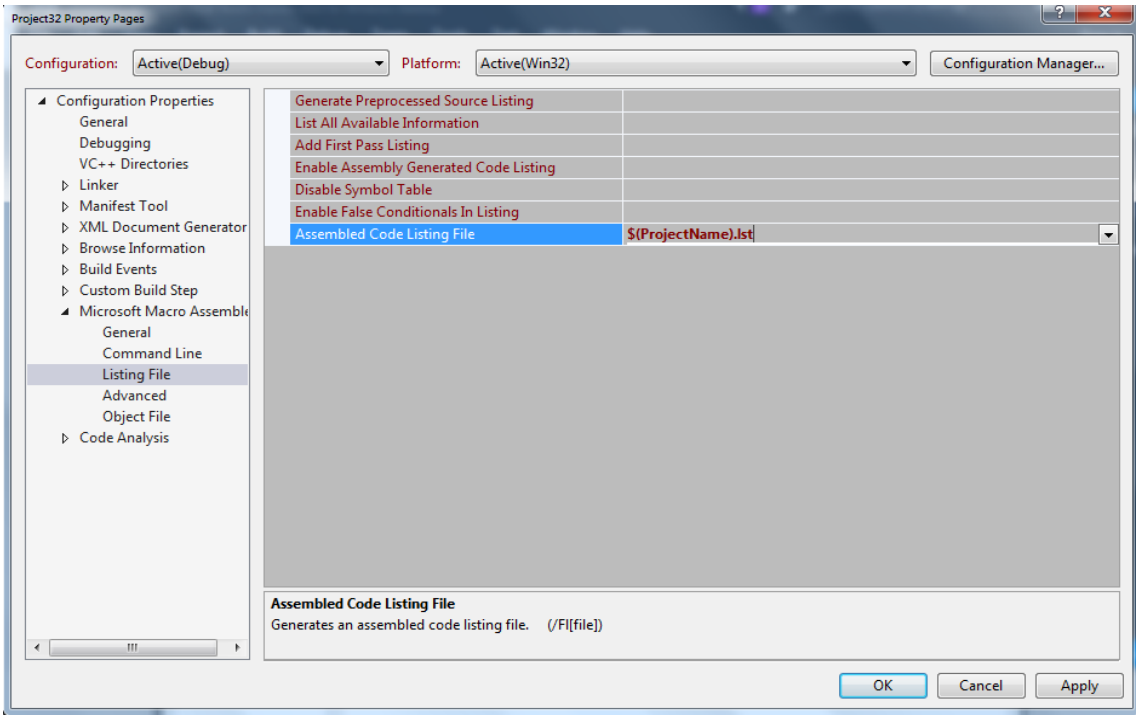


Figura A.10