

UNIVERSITATEA TEHNICĂ „GHEORGHE ASACHI” IASI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

SPECIALIZAREA CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

DISCIPLINA: STRUCTURA SI ORGANIZAREA CALCULATOARELOR

PROIECT

**Integrarea limbajelor ASM și C pe ATmega16: Implementarea unei funcții de adunare**

**Coordonator,**

**Prof. Dr. Ing. Bârleanu Alexandru-Theodor-Cristian**

**Student,**

**Petcu Andrei-Dănuț**

**Grupa 1312B**

**Iași, 2025**

**Descriere Proiect: Integrarea limbajelor ASM și C pe ATmega16**

Acest proiect demonstrează integrarea limbajelor de programare Assembly (ASM) și C pentru implementarea unei aplicații pe microcontroller-ul ATmega16. Scopul principal este utilizarea eficientă a resurselor hardware prin scrierea unei funcții principale (**main**) în limbaj ASM, care apelează funcții externe scrise în C.

Proiectul cuprinde două componente majore:

1) **Programul principal în ASM**:

a) Inițializează parametrii și apelează funcții externe.

b) Folosește registrele interne ale microcontroller-ului pentru manipularea datelor și stocarea rezultatelor.

c) Apelează funcția calc, care calculează suma a două variabile, și funcția par, care verifică dacă rezultatul este par.

**2) Fișierul de funcții în C**:

a) Definirea funcției calc, care efectuează o operație de adunare între două numere de tip unsigned short.

b) Definirea funcției par, care returnează restul împărțirii rezultatului la 2, determinând astfel paritatea.

**Funcționalitate**:

a) Programul inițializează două variabile (a și b) în ASM, le transmite funcției calc scrise în C, iar rezultatul este stocat într-un registru dedicat.

b) Verifică paritatea rezultatului folosind funcția par, demonstrând astfel colaborarea între cele două limbaje.

**Descriere generală IAR Embedded Workbench for AVR:**

**IAR Embedded Workbench for AVR** este un mediu de dezvoltare integrat (IDE) folosit pentru proiectarea aplicațiilor embedded pe microcontrolere din familia AVR, produse de Atmel (acum parte din Microchip Technology). Este o soluție completă care include un editor de cod, un compilator optimizat, un debugger puternic și instrumente de analiză a performanței, fiind ideală pentru dezvoltarea de aplicații eficiente și compacte.

Compilatorul său oferă optimizări avansate care asigură generarea de cod rapid și care utilizează minim resursele hardware disponibile. Editorul de cod integrează funcții precum evidențierea sintaxei, completarea automată și verificarea erorilor, facilitând scrierea unui cod clar și bine structurat. Debuggerul permite verificarea codului în timp real, utilizând breakpoints, vizualizarea registrelor și a memoriei pentru o analiză aprofundată.

IAR Embedded Workbench este compatibil cu majoritatea microcontrolerelor din familia AVR, inclusiv ATmega și ATTiny, oferind suport nativ pentru periferice precum GPIO, UART și ADC. Este un instrument versatil utilizat în industrie pentru dezvoltarea de sisteme embedded în domenii precum automatizări industriale, dispozitive portabile, echipamente medicale și industria auto, dar și în mediul academic pentru proiecte educaționale.

Acest mediu de dezvoltare oferă flexibilitatea utilizării limbajelor C, C++ și ASM într-un singur proiect, ceea ce permite crearea unor soluții complexe și bine optimizate. Prin integrarea sa completă, IAR Embedded Workbench reprezintă un instrument esențial pentru orice dezvoltator embedded care lucrează cu microcontrolere AVR.

**Descriere generală ATmega16:**

**ATmega16** este un microcontroller pe 8 biți, fabricat de Atmel, bazat pe arhitectura AVR. Este un dispozitiv puternic și versatil, utilizat pe scară largă în aplicații embedded datorită performanțelor sale ridicate, consumului redus de energie și setului bogat de periferice integrate.

ATmega16 dispune de un nucleu RISC avansat care poate executa majoritatea instrucțiunilor într-un singur ciclu de ceas, asigurând o execuție rapidă și eficientă a codului. Acesta are o memorie flash de 16 KB pentru stocarea programului, 1 KB de memorie SRAM pentru variabile și 512 bytes de memorie EEPROM pentru stocarea datelor non-volatile.

Un aspect remarcabil al ATmega16 este gama largă de periferice integrate, inclusiv 32 de linii de intrare/ieșire digitale configurabile, convertor analog-digital (ADC) pe 10 biți cu 8 canale, două temporizatoare pe 8 biți și unul pe 16 biți, interfețe de comunicare UART, SPI și I2C (TWI), precum și suport pentru PWM. Acest microcontroller poate funcționa la o frecvență de până la 16 MHz, ceea ce îl face potrivit pentru o varietate de aplicații care necesită atât viteză, cât și eficiență.

Alimentarea microcontroller-ului variază între 4,5 și 5,5 V, iar consumul de energie este redus datorită modurilor de economisire a energiei, cum ar fi modul Power-down sau Idle. Aceste moduri sunt utile în aplicații unde consumul de energie este critic, cum ar fi dispozitivele portabile sau alimentate cu baterii.

ATmega16 este potrivit pentru o gamă largă de aplicații, de la control industrial și sisteme automate, până la proiecte educaționale și hobby.

Exemple de utilizări:

1) Controlul motoarelor și al mașinilor automate folosind semnale PWM (Pulse-width modulation);

2) Automatizarea luminilor bazată pe senzori de mișcare și de lumină ambientală;

3) Monitorizarea senzorilor de proximitate, fum sau temperatură;

4) Controlul motoarelor DC și servomotoarelor pentru mișcare;

**Setări proiect în IAR Embedded Workbench for AVR:**

1) Opțiunea de **Memory Model** (determină modul în care compilatorul accesează memoria microcontrolerului). Modelul **Tiny** presupune că atât codul, cât și datele (variabilele) sunt localizate într-un spațiu de memorie de **64 KB sau mai puțin**. Aceasta utilizează adrese de memorie pe 16 biți, optimizând dimensiunea codului generat și timpul de execuție. Acest model este mai puțin potrivit pentru aplicațiile mari, care necesită acces la spații de memorie mai extinse sau gestionarea unui volum mare de date.

2) Optiunea **Optimizations Level** la **None**

Compilatorul nu va aplica nicio optimizare asupra codului tău. Acesta va compila programul exact așa cum este scris, fără a încerca să îmbunătățească performanța, dimensiunea codului sau să reducă timpul de execuție.

**Ce efecte are setarea la "None" asupra programului?**

**a) Performanță mai scăzută**: Codul va fi mai puțin eficient, deoarece nu se vor face optimizări pentru a reduce numărul de instrucțiuni sau pentru a îmbunătăți viteza de execuție.

**b) Dimensiune mai mare a codului**: Fără optimizări, dimensiunea codului generat poate fi mai mare decât în cazul în care ar fi aplicate optimizări. Compilatorul nu va încerca să elimine instrucțiunile inutile sau redundante.

c) **Debugging mai simplu**: În timpul dezvoltării și depanării, este mai ușor să urmezi fluxul de execuție și să analizezi valorile variabilelor, deoarece nu vor exista schimbări care să afecteze comportamentul așteptat al codului.

3) În cadrul proiectului, am selectat toate opțiunile **Output List File** și **Output Assembler File** din setările IAR Embedded Workbench pentru a facilita analiza detaliată a procesului de compilare și a optimizării codului generat.

Prin activarea opțiunii **Output List File**, se generează un fișier care conține o listă detaliată a fiecărei instrucțiuni și a codului asamblor corespunzător, împreună cu adresele și valorile registrelor utilizate. Acest fișier oferă o perspectivă clară asupra modului în care compilatorul a tradus codul sursă în instrucțiuni mașină și este extrem de util pentru depanare, pentru înțelegerea fluxului de execuție și pentru optimizarea ulterioară a codului.

Activarea opțiunii **Output Assembler File** generează fișierul asamblor (cu extensia .asm), care conține codul asamblor generat de compilator din codul sursă. Acest fișier permite o examinare detaliată a fiecărei instrucțiuni și oferă o înțelegere mai profundă a modului în care codul sursă este transformat în instrucțiuni mașină. De asemenea, fișierul .asm poate fi utilizat pentru optimizări manuale, ajustări fine ale performanței sau adăugarea de secvențe personalizate de cod, în funcție de necesitățile proiectului.

4) Am selectat opțiunile **Generated Linker Listing**, **Segment Map**, **Symbol Listing**, și **Module Map** la nivelul linker-ului din IAR Embedded Workbench pentru a obține o viziune completă asupra procesului de legare a fișierelor obiect și pentru a facilita analiza și depanarea programului.

Prin activarea opțiunii **Generated Linker Listing**, se generează un fișier care conține un raport detaliat al legării, inclusiv informații despre secvențele de cod și date plasate în memoria microcontroler-ului. Acest fișier este util pentru a înțelege cum sunt alocate resursele de memorie și pentru a verifica corectitudinea legării fișierelor obiect.

Opțiunea **Segment Map** creează un raport detaliat cu privire la modul în care diferitele segmente de cod și date sunt plasate în memoria dispozitivului, oferind informații precise despre dimensiunea și locațiile fiecărui segment. Acest lucru este esențial pentru a identifica posibile conflicte între segmente și pentru a optimiza utilizarea memoriei.

Prin activarea opțiunii **Symbol Listing**, se generează un fișier care listează toate simbolurile (funcții, variabile, etc.) definite în proiect, împreună cu adresele și tipurile lor. Această listă poate fi extrem de utilă pentru depanare și pentru urmărirea exactă a legăturii între diferitele părți ale programului, ajutând la localizarea rapidă a erorilor legate de simboluri nedefinite sau incorect legate.

Opțiunea **Module Map** creează un raport care detaliază modul în care sunt legate modulele obiect în timpul procesului de linkare. Acesta furnizează o viziune de ansamblu asupra structurii proiectului, indicând modul în care diferitele module sunt combinate pentru a forma executabilul final.

Am ales formatul **HTML** pentru aceste rapoarte pentru a asigura o vizualizare clară și ușor de navigat a informațiilor generate.

**Explicațiile liniilor de cod:**

**I) ASM (Main):**

**1**) **NAME main**

Aceasta este eticheta care definește numele principal al programului, în acest caz, main. În limbajul Assembly, numele main este folosit pentru a marca punctul de intrare al programului.

**2)** **PUBLIC main**

Aceasta face ca eticheta main să fie accesibilă și din alte module sau fișiere, astfel încât să poată fi apelată de alte funcții externe sau din fișiere externe.

**3)** **EXTERN par, calc**

Acestea declară funcțiile externe par și calc, ceea ce înseamnă că aceste funcții sunt definite în alt loc (în fisierul C). Ele vor fi apelate mai târziu în programul principal.

**4)** **ORG $0**

Aceasta setează locația de început a programului la adresa de memorie 0. În mod tipic, adresa 0 este utilizată pentru codul de inițializare, iar ulterior programul continuă de la această locație.

**5)** **RJMP main**

Aceasta este o instrucțiune de salt relativ. Ea face ca execuția să sară la eticheta main și să înceapă execuția codului principal. Această instrucțiune este folosită pentru a sări direct la începutul funcției main.

**6)** **RSEG CODE**

Începe un segment relocabil, o zonă de memorie în care codul sau datele pot fi plasate într-o locație flexibilă în memoria dispozitivului. Practic, segmentul nu are o locație fixă, iar adresa de început a acestuia poate fi ajustată în timpul procesului de linkare, în funcție de cum este organizată memoria și de setările de configurare ale programului.

**7) main**

Aceasta etichetă marchează începutul funcției main în codul asm.

**8) LDI R16, 2**

LDI încarcă o valoare imediată în registrul R16. Aici, valoarea 2 este încărcată în R16. Așadar, R16 va avea valoarea 2 la acest punct.

**9) LDI R17, 8**

Similar, această linie încarcă valoarea 8 în registrul R17.

**10) subi R16, 255**

subi scade o valoare imediată (în acest caz, 255) din registrul R16. Astfel, R16 va conține valoarea 2 - 255 = -253. În termenii lui **2’s complement**, valoarea -253 este echivalentă cu 3 (pentru un registru pe 8 biți). Așadar, în R16 se va afla valoarea 3.

**11)** **CALL calc**

Această instrucțiune face apelul la funcția externă calc. Aici, funcția calc va aduna valorile din registrele R16 și R17, astfel încât valoarea finală va fi R16 = 3 + 8 = 11. Rezultatul va fi stocat în R16.

**12)** **MOV R22, R16**

Această linie copiază valoarea din R16 în registrul R22. Astfel, rezultatul adunării este stocat în R22 pentru utilizare ulterioară.

**13) CALL par**

Această linie face apelul la funcția externă par. Funcția par verifică dacă valoarea din R16 (care acum este 11) este pară sau impară. Aceasta va returna fie 0 (dacă este par) sau 1 (dacă este impar).

**14) NOP**

NOP este o instrucțiune "No Operation", adică nu face nimic. Este adesea folosită pentru a lăsa loc pentru instrucțiuni viitoare sau pentru a marca un loc în program fără a afecta comportamentul acestuia.

**15) END**

Aceasta marchează sfârșitul programului. Este o instrucțiune de încheiere a codului și poate indica terminarea programului sau sfârșitul secțiunii curente de cod.

**II) C (funcții):**

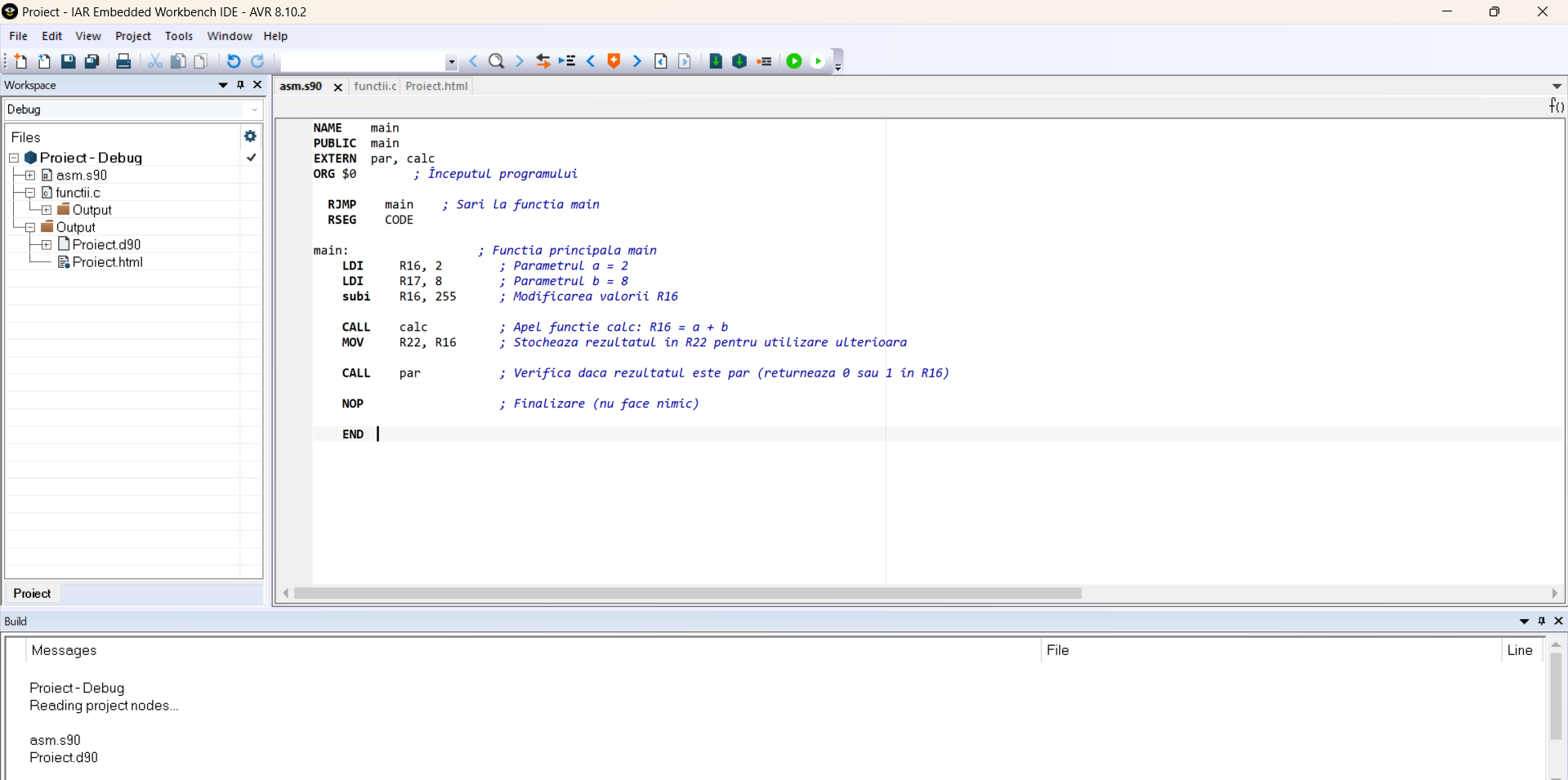
**1)** **Funcția par**:

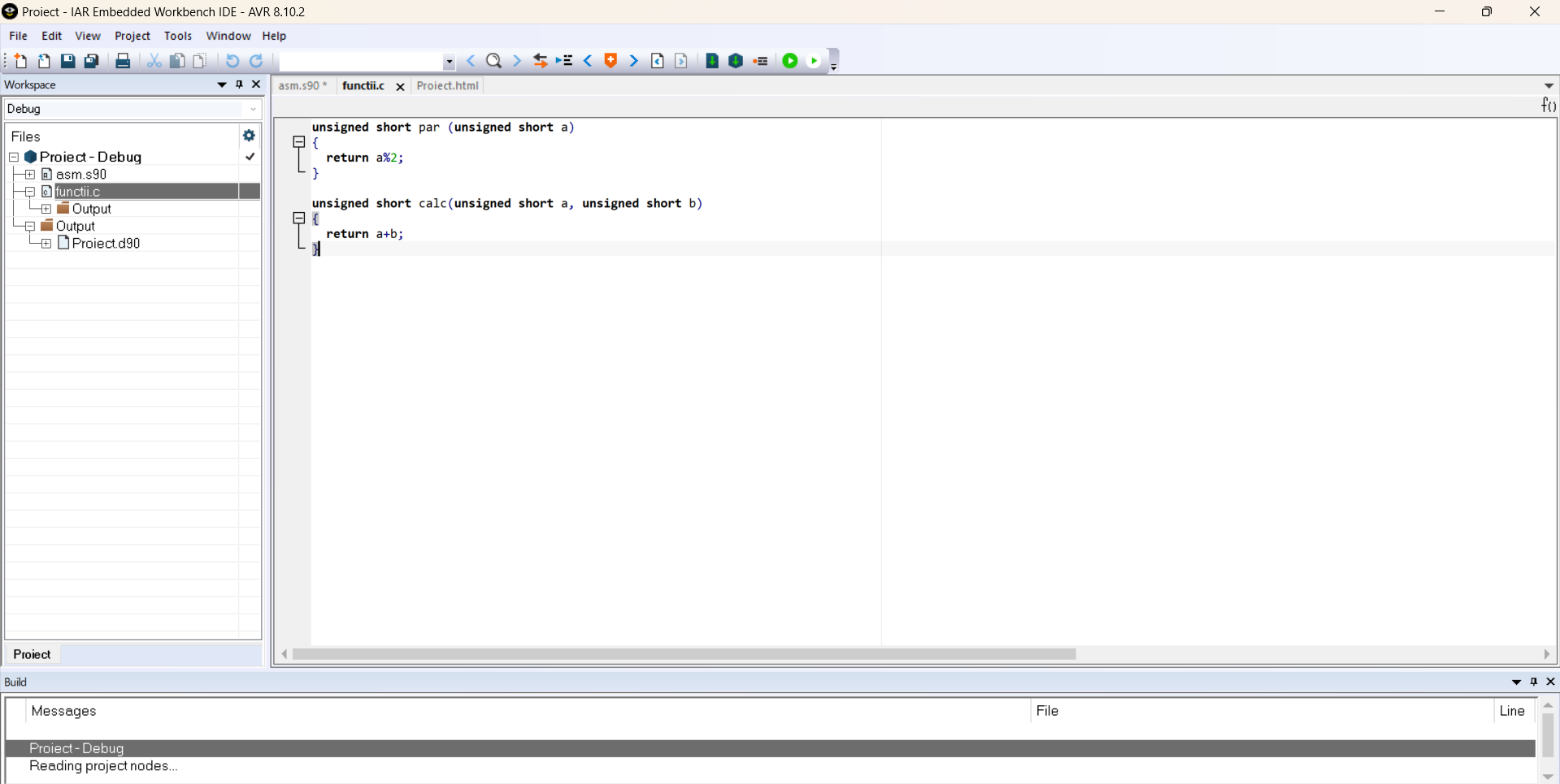
* Primește un argument a de tip unsigned short și returnează restul împărțirii lui a la 2 (adică verifică dacă a este par sau impar).
* Dacă a % 2 == 0, funcția va returna 0 (par), iar dacă a % 2 == 1, va returna 1 (impar).

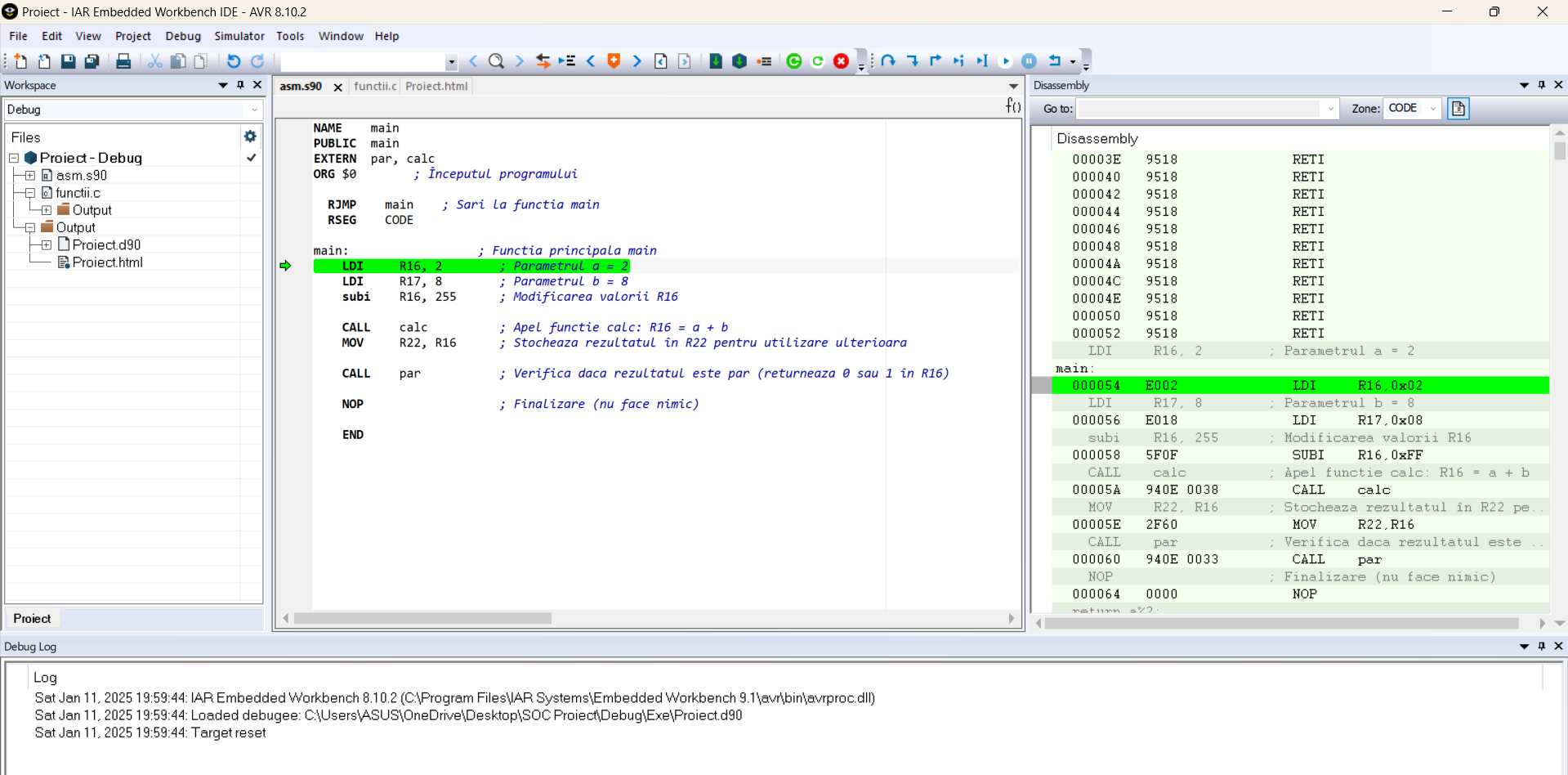
**2)** **Funcția calc**:

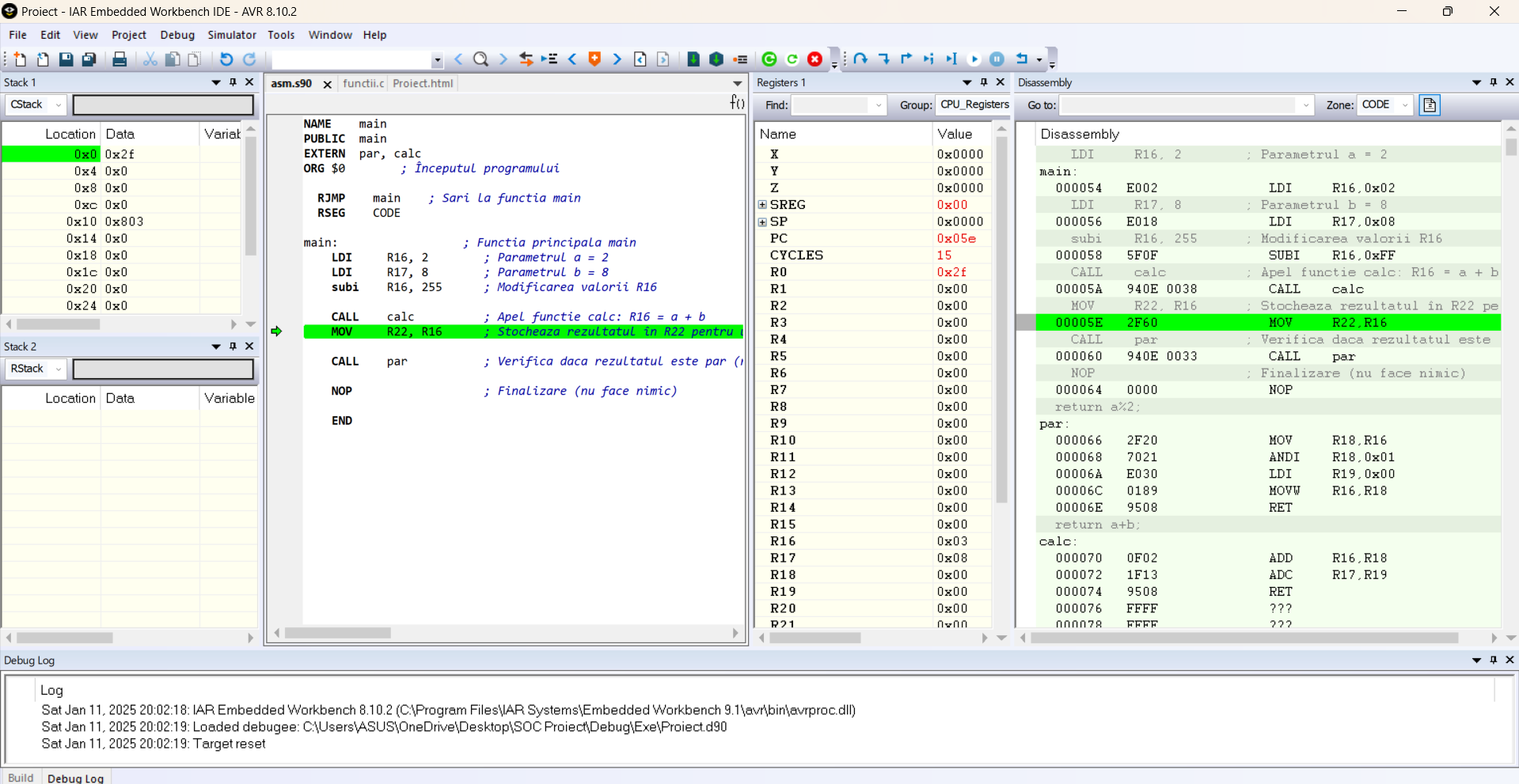
* Primește două argumente, a și b de tip unsigned short, și returnează suma acestora (a + b).

**Imagini proiect:**

****

****

****

****