Universitatea Tehnică „Gheorghe Asachi” din Iași

Facultatea de Automatică și Calculatoare

Domeniul: Calculatoare și Tehnologia Informației

Proiect structura și organizarea calculatoarelor

Avramescu Andrei

gr. 1312B

Prof. coord. . Bârleanu Alexandru-Theodor-Cristian

Descriere Proiect: Integrarea limbajelor ASM și C pe ATmega16

Proiectul constă în realizarea unui program corect în IAR Embedded Workbench IDE pentru un microcontroler ATmega16 utilizând un fișier principal scris în assembly și două funcții definite în limbajul C. Funcțiile din C vor fi apelate din codul assembly prin mecanismul de apel al subrutinelor (CALL), iar stiva va ajuta la gestionarea parametrilor.

Componente program:

1) Programul principal în ASM:

- Inițializează parametrii și apelează funcții externe.
- Folosește registrele interne ale microcontroller-ului pentru manipularea datelor și stocarea rezultatelor.
- Apelează funcția calc, care calculează suma a două variabile, și funcția par, care verifică dacă rezultatul este par.

2) Fișierul de funcții în C:

- Definirea funcției calc, care efectuează o operație de adunare între două numere de tip unsigned short.
- Definirea funcției par, care returnează restul împărțirii rezultatului la 2, determinând astfel paritatea.

Descriere generală IAR Embedded Workbench for AVR:

IAR Embedded Workbench for AVR este un mediu integrat de dezvoltare (IDE) avansat, conceput pentru programarea, depanarea și optimizarea aplicațiilor embedded care rulează pe microcontrolere AVR produse de Microchip. Este un instrument puternic utilizat pe scară largă în industrie pentru dezvoltarea de aplicații de înaltă performanță în domeniul sistemelor embedded, oferind suport pentru limbaje precum C, C++ și Assembly.

Descriere Generală - Microcontroler ATmega16:

***ATmega16** este un microcontroler pe 8 biți produs de Microchip Technology (anterior Atmel). Face parte din familia AVR și este conceput pentru aplicații embedded care necesită un echilibru între performanță, consum redus de energie și flexibilitate. ATmega16 este utilizat pe scară largă datorită funcționalităților sale versatile, compatibilității cu limbaje de programare precum C și Assembly și suportului extins pentru periferice integrate.*

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?

   - ***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.*
   - ***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.*
   - ***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.*
   - *Activarea opțiunii **Output Assembler File** generează fișierul asamblor (cu extensia .asm), care conține codul asamblor generat de compilator din codul sursă.*
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.*
- *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.*
- *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.*
- *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.*

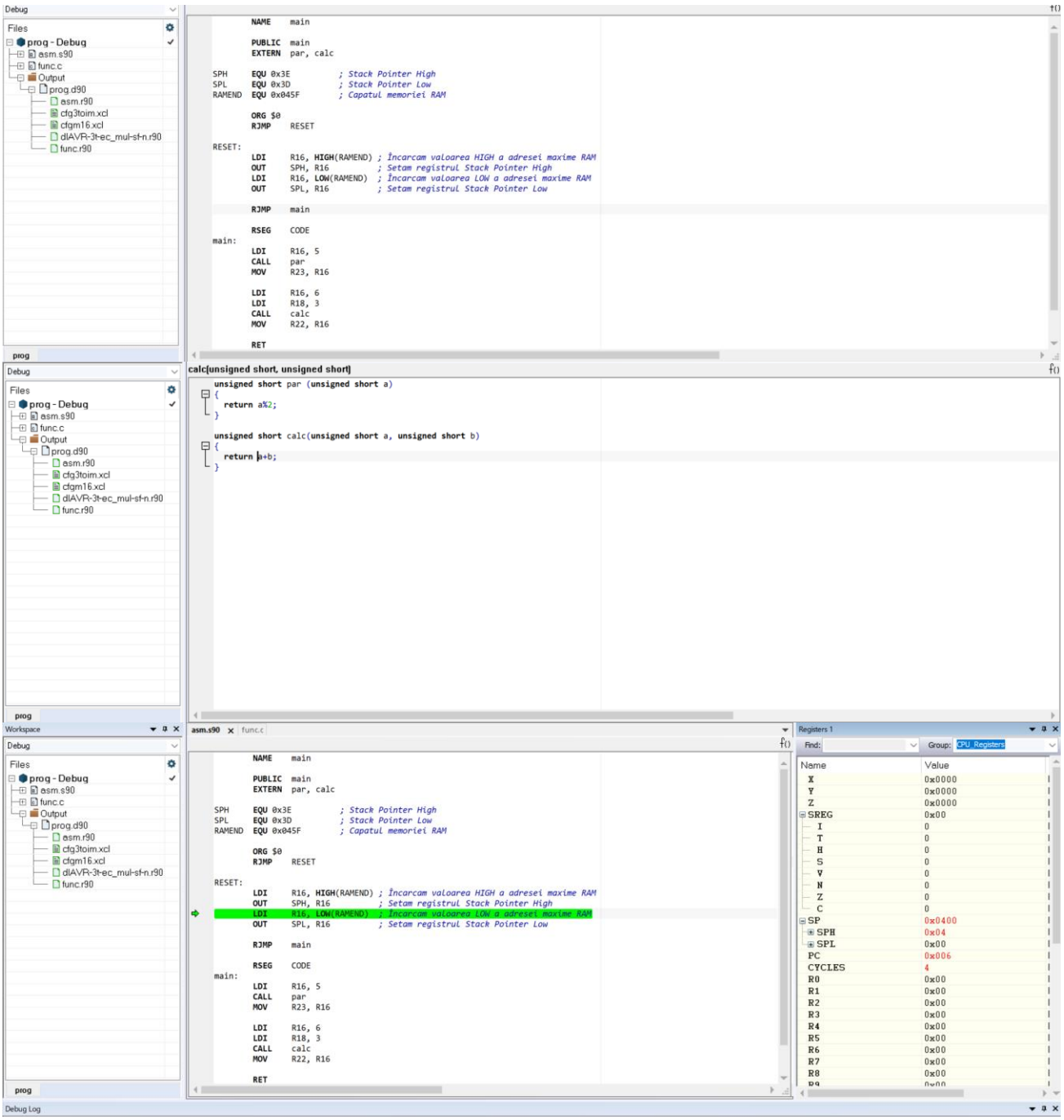
Explicatii cod :

- ***NAME main -** Setează numele modulului principal al programului. Este folosit pentru a identifica acest fișier în cadrul proiectului.*
- ***PUBLIC main -** Declară simbolul main ca fiind public, ceea ce permite altor module să îl acceseze.*
- ***EXTERN par, calc -** Declară simbolurile par și calc ca fiind externe. Acestea sunt funcții definite în alte fișiere sau module care vor fi folosite în acest program.*
- ***SPH EQU 0x3E -** Adresa registrului care controlează partea superioară (High Byte) a pointerului de stivă (Stack Pointer). (Adresa **0x3E** este rezervată pentru registrul SPH în **ATmega16**, conform documentației oficiale).*
- ***SPL EQU 0x3D -** Adresa registrului pentru partea inferioară (Low Byte)*

a pointerului de stivă. (Adresa 0x3D este rezervată pentru registrul SPL în ATmega16, conform documentației oficiale).

- ***RAMEND EQU 0x045F*** *- Adresa finală a memoriei RAM pentru ATmega16 (în acest caz, 0x045F).*
- ***ORG $0 -*** *Plasează următoarele instrucțiuni la adresa 0x0000, care este vectorul de reset. Aceasta este locația în care microcontrolerul începe execuția după pornire.*
- ***RJMP RESET -*** *Realizează un salt relativ la eticheta RESET, adică programul sare la codul din secțiunea RESET.*
- ***LDI R16, HIGH(RAMEND)*** *- Încărcăm în registrul R16 valoarea octetului superior al adresei maxime RAM (HIGH returnează partea superioară a adresei).*
- ***OUT SPH, R16 -*** *Transferăm valoarea din R16 în registrul SPH, configurând partea superioară a pointerului de stivă.*
- ***LDI R16, LOW(RAMEND)*** *- Încărcăm în registrul R16 valoarea octetului inferior al adresei maxime RAM.*
- ***OUT SPL, R16 -*** *Transferăm valoarea din R16 în registrul SPL, configurând partea inferioară a pointerului de stivă.*
- ***RJMP main*** *- Realizează un salt relativ către secțiunea main, unde începe programul principal.*
- ***RSEG CODE*** *- Specifică faptul că urmează secțiunea de cod (instrucțiuni executabile).*
- ***LDI R16, 5*** *- Încărcăm valoarea 5 în registrul R16.*
- ***CALL par -*** *Apelăm funcția externă par. Valoarea din R16 este, probabil, utilizată de funcție pentru a calcula ceva.*
- ***MOV R23, R16*** *- Copiem rezultatul din R16 (modificat de funcția par) în registrul R23.*
- ***LDI R16, 2*** *- Încărcăm valoarea 2 în registrul R16.*
- ***LDI R18, 3*** *- Încărcăm valoarea 3 în registrul R18.*
- ***CALL calc*** *- Apelăm funcția externă calc. Probabil, funcția utilizează valorile din R16 și R18 pentru a calcula ceva.*
- ***MOV R22, R16*** *- Copiem rezultatul calculului (returnat în R16 de funcția calc) în registrul R22.*
- ***RET*** *- Încheie execuția funcției main și revine la apelant (în acest caz, la programul principal sau, dacă este folosit debugger-ul, termină execuția).*
- ***END main*** *- Marchează sfârșitul codului. Indică IDE-ului că execuția începe de la eticheta main.*

Imagini cod :


Debug
Files
prog - Debug
asm.s90
func.c
Output
prog.d90
asm.r90
cfg3toim.xcl
cfgm16.xcl
dlAVR-3t-ec_mul-sf-n.r90
func.r90
prog
NAME main
PUBLIC main
EXTERN par, calc
SPH EQU 0x3E ; Stack Pointer High
SPL EQU 0x3D ; Stack Pointer Low
RAMEND EQU 0x045F ; Capatul memoriei RAM
ORG $0
RJMP RESET
RESET:
LDI R16, HIGH(RAMEND) ; Încarcam valoarea HIGH a adresei maxime RAM
OUT SPH, R16 ; Setam registrul Stack Pointer High
LDI R16, LOW(RAMEND) ; Încarcam valoarea LOW a adresei maxime RAM
OUT SPL, R16 ; Setam registrul Stack Pointer Low
RJMP main
RSEG CODE
main:
LDI R16, 5
CALL par
MOV R23, R16
LDI R16, 6
LDI R18, 3
CALL calc
MOV R22, R16
RET
calc(unsigned short, unsigned short)
unsigned short par (unsigned short a)
{
return a%2;
}
unsigned short calc(unsigned short a, unsigned short b)
{
return a+b;
}
Workspace
asm.s90
func.c
Registers 1
Find:
Group:
CPU Registers
Name
Value
X 0x0000
Y 0x0000
Z 0x0000
SREG 0x00
I 0
T 0
H 0
S 0
V 0
N 0
Z 0
C 0
SP 0x0400
SPH 0x04
SPL 0x00
PC 0x006
CYCLES 4
R0 0x00
R1 0x00
R2 0x00
R3 0x00
R4 0x00
R5 0x00
R6 0x00
R7 0x00
R8 0x00
Debug Log