UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
SPECIALIZAREA: CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

Proiect la

Structura și organizarea calculatoarelor-proiect

Coordonator științific
Ș.l.dr.ing. Alexandru BÂRLEANU

Student
Rareș-Andrei DANCĂU
1308A

Iași, 2023

Cuprins

Directive....6
Sumar al directivelor....6
Convenții ale definirilor de sintaxă....10
Directive de control al modulului....10
Sintaxă....10
Parametri....10
Descriere....11
Începerea unui modul de tip program....11
Începerea unui modul de tip bibliotecă....11
Terminarea unui modul....11
Terminarea ultimului modul....11
Asamblarea fișierelor cu mai multe module....12
Declararea modelelor de atribute runtime....12
Exemple....13
Directive de control al simbolurilor....15
Sintaxă....15
Parametri....15
Descriere....15
Exportarea simbolurilor către alte module....15
Exportarea simbolurilor cu mai multe definiții către alte module....16
Importarea simbolurilor....16
Exemplu....16
Directive de control al segmentelor....17
Sintaxă....17
Parametri....18
Descriere....19
Începerea unui segment absolut....19
Începerea unui segment relocatabil....19
Începerea unui segment stivă....19
Începerea unui segment comun....19
Setarea numărătorului pentru locația programului(PLC)....20
Alinierea unui segment....20
Exemple....21
Începerea unui segment absolut....21
Începerea unui segment relocatabil....21
Începerea unui segment stivă....22
Începerea unui segment comun....22
Alinierea unui segment....23
Operatori ai asamblorului....23
Precedența operatorilor....23
Rezumatul operatorilor....24
Operatori unari- precedența 1....24
Operatori multiplicativi aritmetici și de shiftare- precedența 3....24
Operatori aditivi aritmetici- precedența 4....24
Operatori și- precedența 5....25

Operatori sau- precedența 6....25
Operatori de comparație- precedența 7....25
Descrierea operatorilor....25
Particularizarea instrucțiunilor....39
MOV....39
Descriere....39
Operație....39
Sintaxă....40
Operanzi....40
Registrul PC(program counter)....40
Codificarea instrucțiunii pe 16 biți....40
Observații....40
LDI....40
Descriere....41
Operație....41
Sintaxă....41
Operanzi....41
Registrul PC....41
Codificarea instrucțiunii pe 16 biți....41
Observații....41
ADD....42
Descriere....42
Operație....42
Sintaxă....42
Operanzi....42
Registrul PC....42
Codificarea instrucțiunii pe 16 biți....43
Observații....43
Flag-uri afectate....43
ADC....43
Descriere....43
Operație....44
Sintaxă....44
Operanzi....44
Registrul PC....44
Codificarea instrucțiunii pe 16 biți....44
Observații....44
Flag-uri afectate....44
MOVW....45
Descriere....45
Operație....45
Sintaxă....45
Operanzi....45
Registrul PC....46
Codificarea instrucțiunii pe 16 biți....46
Observații....46
RET....46

Descriere....46
Operație....46
Sintaxă....47
Codificarea instrucțiunii pe 16 biți....47
Observații....47
RJMP....47
Descriere....47
Operație....48
Sintaxă....48
Operanzi....48
Registrul PC....48
Stivă....48
Codificarea operației pe 16 biți....48
Observații....49
RCALL....49
Descriere....49
Operație....49
Sintaxă....49
Operanzi....50
Registrul PC....50
Stivă....50
Codificarea operației pe 16 biți....50
Observații....50
SUB....51
Descriere....51
Operație....51
Sintaxă....51
Operanzi....51
Registrul PC....52
Codificarea operației pe 16 biți....52
Observații....52
Flag-uri afectate....52
Procesul creerii aplicației....53
Procesul combinării fișierelor scrise în mai multe limbaje....53
Procesul de link-editare....54
După link-editare....56
Executarea aplicației....57
Pornirea sistemului....58
Funcția main....60
Oprirea sistemului....60
Exemplu concret....62
a) Proiect cu funcția main scrisă în limbajul C care apelează mai multe funcții scrise în asamblare. 62
Realizarea proiectului pentru a obține o bibliotecă cu funcțiile scrise în asamblare....62
Proiectul principal....66
b) Proiect cu funcția main scrisă în asamblare care apelează funcții scrise în limbajul C....69
Realizarea proiectului pentru a obține o bibliotecă cu funcțiile scrise în limbajul C....69
Proiectul principal....73

Bibliografie....76
Anexa 1....76
a) Proiect cu funcția main scrisă în limbajul C care apelează mai multe funcții scrise în asamblare. 76
Realizarea proiectului pentru a obține o bibliotecă cu funcțiile scrise în asamblare....76
asm.s90....76
asm.lst....77
header.h....83
Proiectul principal....83
main.c....83
main.lst....84
main.s90....87
header.h....91
b) Proiect cu funcția main scrisă în asamblare care apelează mai multe funcții scrise în limbajul C. 91
Realizarea proiectului pentru a obține o bibliotecă cu funcțiile scrise în limbajul C....91
functie.c....91
functie.lst....91
functie.s90....95
scadere.h....98
Proiectul principal....98
asm.s90....98
asm.lst....99
scadere.h....102

Directive

Directivele sunt instrucțiuni către asamblor, care specifică acțiunea ce urmează în procesul de asamblare.

Sumar al directivelor

În tabelul următor se regăsesc rezumatele pentru directivele disponibile familiei de microcontrolere AVR.

Directivă	Descriere	Secțiune
END	Termină asamblarea ultimului modul dintr-un fișier	MC
ENDMOD	Termină asamblarea modulului curent	MC
LIBRARY	Începe un modul de tip bibliotecă	MC
MODULE	Începe un modul de tip bibliotecă	MC
NAME	Începe un modul de tip program	MC
PROGRAM	Începe un modul de tip program	MC
RTMODEL	Declară atribute de model de runtime	MC
ALIGN	aliniază numărătorul de locație prin inserarea octeților nuli	SC
ASEG	Începe un segment absolut	SC
ASEGN	Începe un segment absolut cu nume	SC
COMMON	Începe un segment comun	SC
EVEN	Aliniază registrul PC la o adresă pară	SC
ODD	Aliniază registrul PC la o adresă impară	SC
ORG	Setează valoarea numărătorului de locație	SC
RSEG	Începe un segment relocatabil	SC
STACK	Începe un segment de tip stivă	SC
EXPORT	Exportă simboluri către alte module	SyC

EXTERN	Importă un simbol extern	SyC
EXTRN	Importă un simbol extern	SyC
IMPORT	Importă un simbol extern	SyC
PUBWEAK	Exportă simboluri către alte module, permițând definiții multiple	SyC
PUBLIC	Exportă simboluri către alte module	SyC
REQUIRE	Forțează un simbol să fie referențiat	SyC
$	Include un fișier	AC
/*comentariu*/	Delimitator comentarii în stil C	AC
//	Delimitator comentarii în stil C++	AC
CASEOFF	Dezactivează sensibilitatea la capitalizare	AC
CASEON	Activează sensibilitatea la capitalizare	AC
RADIX	Setează baza standard	AC
ELSE	Asamblează instrucțiuni dacă o condiție e falsă	CA
ELSEIF	Specifică o nouă condiție într-un bloc IF...ENDIF	CA
ENDIF	Termină un bloc IF	CA
IF	Asamblează instrucțiuni dacă condiția este adevărată	CA
CFI	Specifică informațiile cadrului de apel	CFI
#define	Atribuie o valoare unei etichete	CS
#elif	Introduce o nouă condiție într-un bloc #if … #endif	CS
#endif	Încheie un bloc #if, #ifdef, #ifndef	CS
#error	Generează o eroare	CS
#if	Asamblează instrucțiuni dacă o condiție e adevărată	CS
#ifdef	Asamblează instrucțiuni dacă un simbol este definit	CS
#ifndef	Asamblează instrucțiuni dacă un simbol nu este definit	CS
#include	Include un fișier	CS
#message	Generează un mesaj la ieșirea standard.	CS
#undef	Șterge definirea unei etichete	CS
DB	Generează constante de 8 biți, incluzând șiruri de caractere	DDA
DC16	Generează constante de 16 biți, incluzând șiruri de caractere	DDA
DC24	Generează constante de 24 de biți	DDA

DC32	Generează constante de 32 de biți	DDA
DC8	Generează constante de 8 biți, incluzând șiruri de caractere	DDA
DD	Generează constante de 32 de biți	DDA
DEFINE	Definește o valoare disponibilă pentru tot fișierul	DDA
DP	Generează constante de 24 de biți	DDA
DS	Alocă spațiu de 8 biți	DDA
DS16	Alocă spațiu de 16 biți	DDA
DS24	Alocă spațiu de 24 biți	DDA
DS32	Alocă spațiu de 32 biți	DDA
DS8	Alocă spațiu de 8 biți	DDA
DW	Generează constante de 16 biți, incluzând șiruri de caractere	DDA
COL	Setează numărul de coloane pe pagină	LC
LSTCND	Controlează modul în care va apărea în fișierul listă condiționarea în asamblare	LC
LSTCOD	Controlează modul în care vor apărea în fișierul listă instrucțiunile cu mai multe linii de cod	LC
LSTEXP	Controlează modul în care vor apărea în fișierul listă liniile generate de macro-uri	LC
LSTMAC	Controlează modul în care va apărea în fișierul listă definițiile macro-urilor	LC
LSTOUT	Controlează modul de output al fișierului listă	LC
LSTPAG	Controlează formatarea ieșirii în pagini	LC
LSTREP	Controlează modul în care vor apărea în fișierul listă liniile generate de directive repetitive	LC
LSTXRF	Generează un tabel de corespondență	LC
PAGE	Generează o nouă pagină	LC
PAGSIZ	Setează numărul de linii pe pagină	LC
ENDM	Termină definirea unui macro	MP
ENDR	Termină o structură repetitivă	MP
EXITM	Iese prematur dintr-un macro	MP
LOCAL	Creează un simbol local pentru un macro	MP
MACRO	Definește un macro	MP
REPT	Asamblează instrucțiuni de un număr specificat de ori	MP

PEPTC	Repetă și substituie caractere	MP
REPTI	Repetă și substituie șiruri de caractere	MP
=	Asignează o valoare permanentă locală unui modul	VA
ALIAS	Asignează o valoare permanentă locală unui modul	VA
ASSIGN	Asignează o valoare temporară	VA
EQU	Asignează o valoare permanentă locală unui modul	VA
LIMIT	Verifică dacă o valoare este între limite	VA
SFRB	Creează etichete SFR cu acces la octeți	VA
SFRTYPE	Specifică atributele SFR	VA
SFRW	Creează etichete SFR cu acces la cuvinte	VA
VAR	Asignează o valoare temporară	VA

Legendă pentru coloana Secțiune:

MC-Control de modul
SC-Control de segment
SyC-Control de simboluri
AC-Control al asamblorului
CA-asamblare condiționată
CFI-Apelarea informației de cadru
CS-Preprocesor de stil C
DDA-Definire de date sau alocare
LC-Control al fișierului listă
MP-Procesare de macro-uri
VA-Asignare de valoare

- În cadrul proiectului au fost folosite directive de Control al modulelor, de Control al simbolurilor și de Control al segmentelor. Restul tipurilor de directive sunt definite în AVR IAR Assembler Reference Guide.

Convenții ale definirilor de sintaxă

- Parametrii apar imediat după directivă.
- Parametrii opționali sunt dispuși între paranteze pătrate.

Directive de control al modulului

Aceste directive sunt folosite pentru a marca începutul și sfârșitul modulelor programului și pentru asignarea numelor și tipurilor lor.

Directivă	Descriere
END	termină ultimul modul al unui fișier de asamblare
ENDMOD	termină asamblarea modului curent
LIBRARY	începe un modul de tip bibliotecă
MODULE	începe un modul de tip bibliotecă
NAME	începe un modul de tip program
PROGRAM	începe un modul de tip program
RTMODEL	declară atribute de model de runtime

Sintaxă

END [eticheta]
ENDMOD [eticheta]
LIBRARY simbol [(expresie)]
MODULE simbol [(expresie)]
NAME simbol [(expresie)]
PROGRAM simbol [(expresie)]
RTMODEL cheie, valoare

Parametri

- **expresie** O expresie opționala(0-255) folosită de compilatorul IAR pentru a encoda limbajul de programare, modelul de memorie și configurația procesorului.
- **cheie** Un șir de caractere ce specifică cheia.
- **eticheta** O expresie sau o etichetă ce poate fi folosită în timpul asamblării. Este ieșirea în codul obiect ca o adresă de intrare în program.
- **simbol** Numele asignat modulului, folosit de XLINK și XLIB când procesează fișiere de tip obiect.
- **valoare** Un șir de caractere ce specifică valoarea.

Descriere

Începerea unui modul de tip program

Se folosește *NAME* pentru a începe un modul de tip program și pentru a asigna un nume pentru referință viitoare de către IAR XLINK Linker™ și IAR XLIB Librarian™.

Începerea unui modul de tip bibliotecă

Se folosește *MODULE* pentru a crea module de tip bibliotecă ce conțin un număr mic de module --cum ar fi sisteme de runtime în limbajele de nivel înalt -- unde fiecare modul reprezintă o singură rutină. Cu funcționalitatea multi-modul, pot fi reduse numărul de fișiere sursă și obiect necesare.

Modulele de tip bibliotecă sunt copiate în codul link-editat dacă alte module folosesc un simbol public din modul.

Terminarea unui modul

Se folosește *ENDMOD* pentru a defini sfârșitul unui modul.

Terminarea ultimului modul

Se folosește *END* pentru a indica sfârșitul fișierului sursă. Toate liniile de după directiva *END* sunt ignorate.

Asamblarea fișierelor cu mai multe module

Entitățile din program trebuie să fie re-alocabile sau absolute și vor apărea în hărțile de încărcare XLINK, dar și în forme hexazecimale absolute ale formatelor fișierelor de ieșire. Aceste entități nu trebuie să fie definite extern.

Următoarele reguli se aplică la asamblarea fișierelor cu mai multe module:

- La începutul unui nou modul toate simbolurile create de utilizator sunt șterse, cu excepția celor create prin *DEFINE*, *#define* sau *MACRO*
- Numărătoarele de locație sunt resetate
- Modul este setat la absolut.

Directivele de control al fișierului listă rămân în efect pe durata asamblării.

- Observație: *END* trebuie să fie mereu folosit la ultimul modul și nu mai trebuie să fie linii de cod sursă(în afară de comentarii și directive de control al fișierului listă) între un *ENDMOD* și o directivă *MODULE*.
- Dacă directiva *NAME* sau *MODULE* lipsește, modulului îi va fi asignat numele fișierului sursă și atributul program.

Declararea modelelor de atribute runtime

Se folosește *RTMODEL* pentru a forța consistența între module. Toate modulele ce sunt linkeditate împreună și definesc aceeași cheie pentru atributul runtime trebuie să aibă aceeași valoare pentru valoarea cheie corespunzătoare sau cheia specială *.

Folosirea acestei chei speciale este echivalentă cu nedefinirea atributului. Poate fi utilizată pentru a spune că modulul poate colabora cu orice model de tip runtime.

Un modul poate avea mai multe definiții ale modelului de runtime.

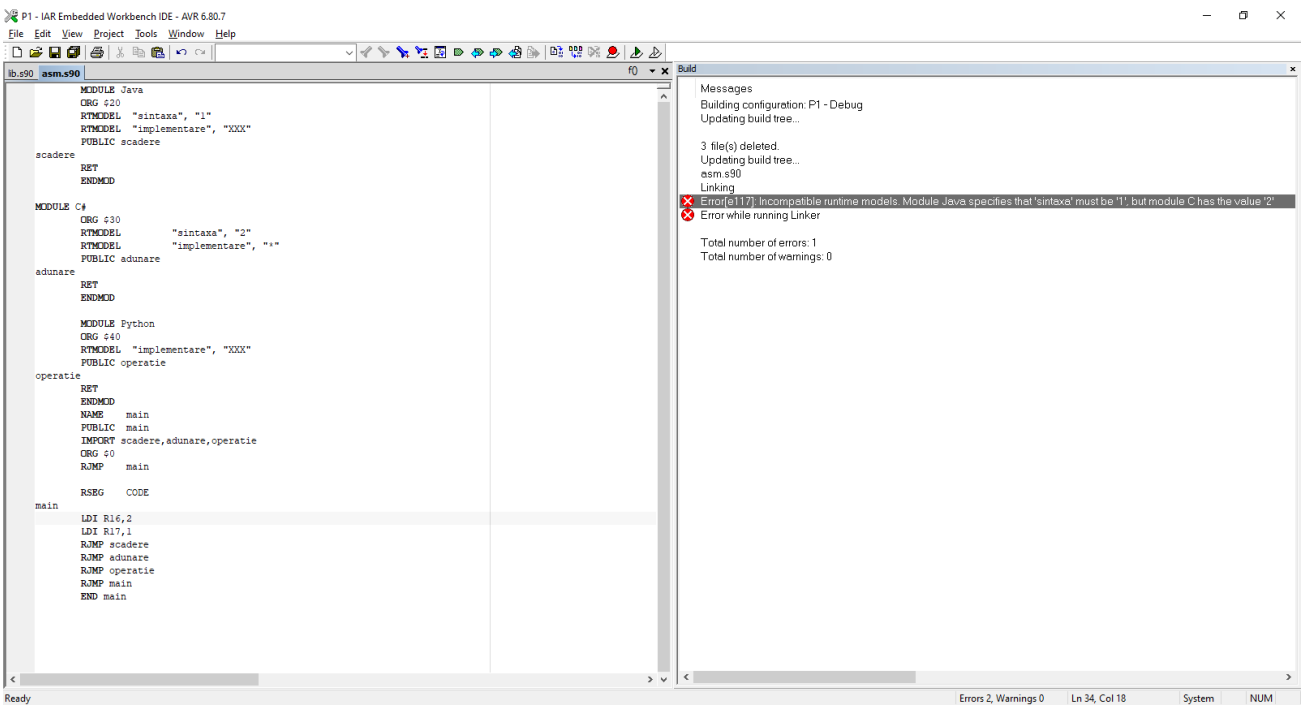
Observație: Atributele de model de runtime încep cu un dublu underscore. Pentru a nu se realiza confuzie, acest stil nu trebuie folosit la atributele definite de utilizator.

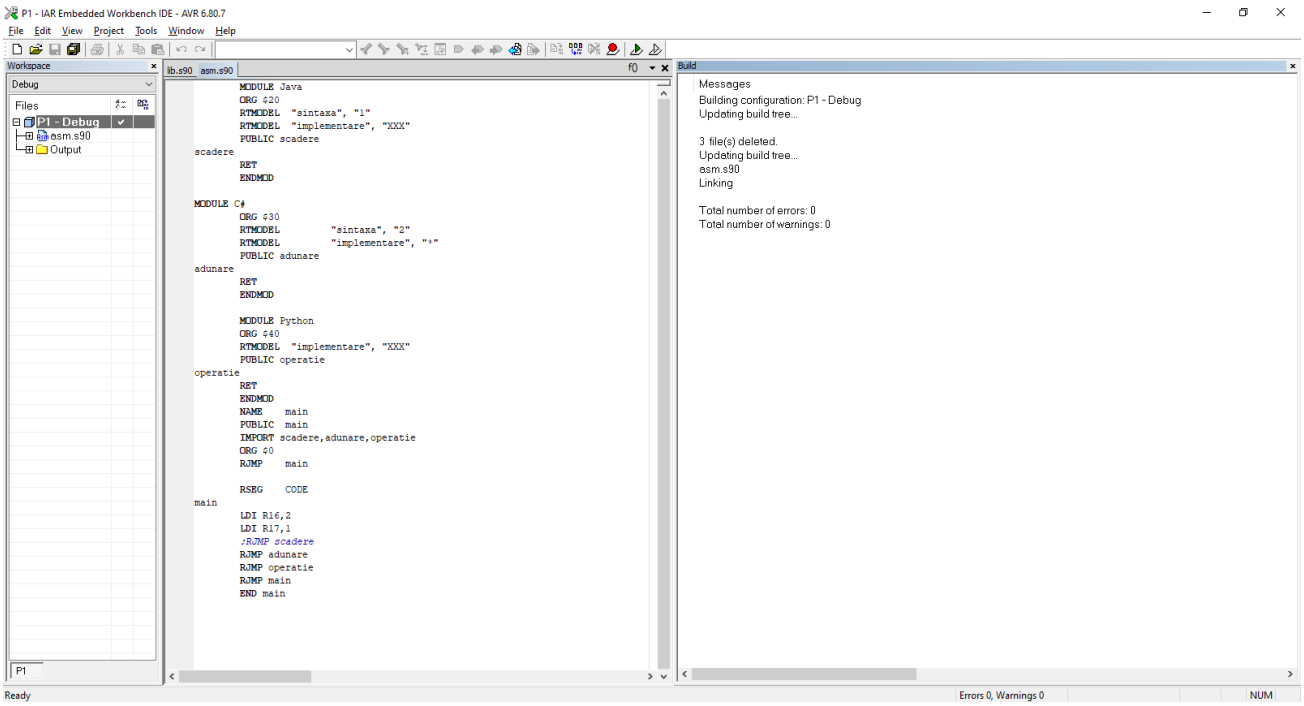
Dacă se scriu rutine în asamblare folosind cod C și se dorește controlarea consistenței modului, a se consulta AVR IAR C/EC++ Compiler Reference Guide.

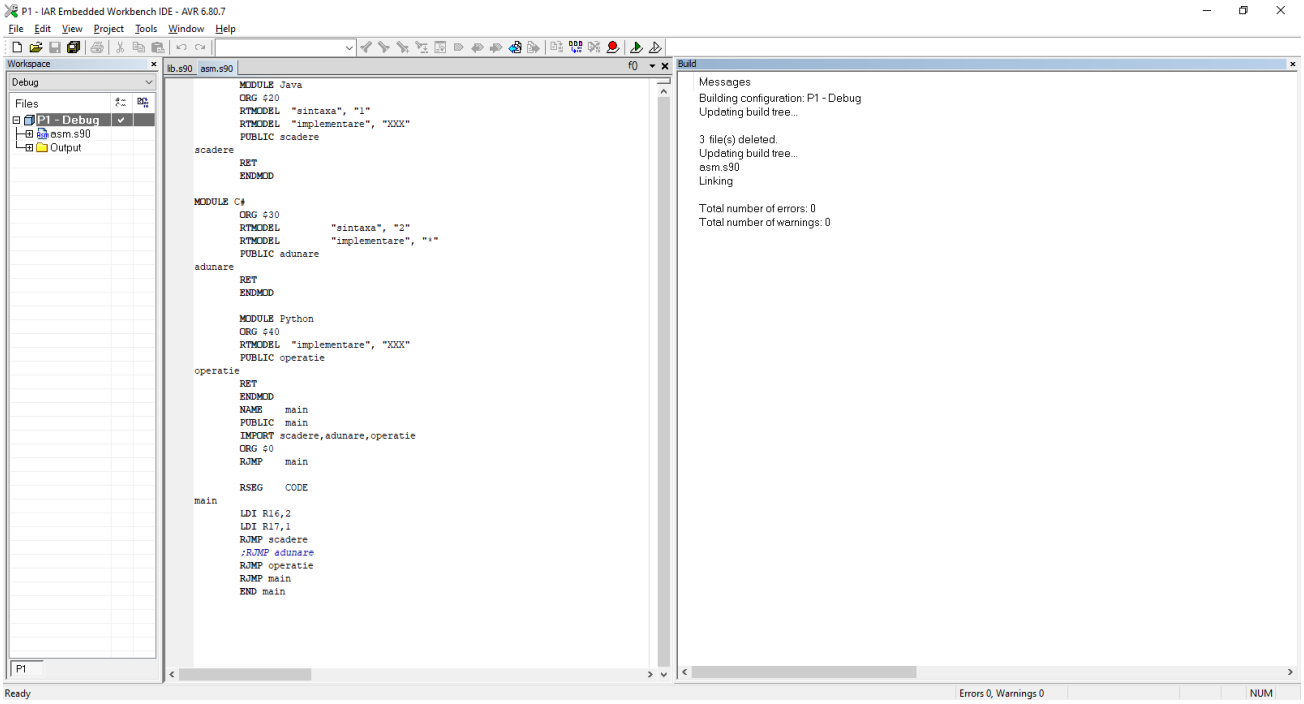
Exemple

Următorul exemplu definește trei module unde:

- Modulul Java și Modulul C# nu pot fi link-editate împreună deoarece au valori diferite ale modelului de runtime „sintaxa”.
- Modulul Java și Modulul Python pot fi link-editate împreună pentru că au aceeași definiție a modelului de runtime „implementare” și nu au conflict la definiția lui „sintaxa”.
- Modulul C# și Modulul Python pot fi link-editate împreună pentru că nu au conflicte la modele de runtime. Valoarea „*” corespunde oricărei valori a modelului de runtime.




P1 - IAR Embedded Workbench IDE - AVR 6.80.7
File Edit View Project Tools Window Help
Workspace
Debug
Files
P1 - Debug
asm.s90
Output
lib.s90 asm.s90
MODULE Java
ORG $20
RTMODEL "sintaxa", "1"
RTMODEL "implementare", "XXX"
PUBLIC scadere
scadere
RET
ENDMOD
MODULE C#
ORG $30
RTMODEL "sintaxa", "2"
RTMODEL "implementare", "*"
PUBLIC adunare
adunare
RET
ENDMOD
MODULE Python
ORG $40
RTMODEL "implementare", "XXX"
PUBLIC operatie
operatie
RET
ENDMOD
NAME main
PUBLIC main
IMPORT scadere,adunare,operatie
ORG $0
RJMP main
RSEG CODE
main
LDI R16,2
LDI R17,1
;RJMP scadere
RJMP adunare
RJMP operatie
RJMP main
END main
Build
Messages
Building configuration: P1 - Debug
Updating build tree...
3 file(s) deleted.
Updating build tree...
asm.s90
Linking
Total number of errors: 0
Total number of warnings: 0
Ready
Errors 0, Warnings 0
NUM


P1 - IAR Embedded Workbench IDE - AVR 6.80.7
File Edit View Project Tools Window Help
Workspace
Debug
Files
P1 - Debug
asm.s90
Output
lib.s90 asm.s90
MODULE Java
ORG $20
RTMODEL "sintaxa", "1"
RTMODEL "implementare", "XXX"
PUBLIC scadere
scadere
RET
ENDMOD
MODULE C#
ORG $30
RTMODEL "sintaxa", "2"
RTMODEL "implementare", "*"
PUBLIC adunare
adunare
RET
ENDMOD
MODULE Python
ORG $40
RTMODEL "implementare", "XXX"
PUBLIC operatie
operatie
RET
ENDMOD
NAME main
PUBLIC main
IMPORT scadere,adunare,operatie
ORG $0
RJMP main
RSEG CODE
main
LDI R16,2
LDI R17,1
RJMP scadere
;RJMP adunare
RJMP operatie
RJMP main
END main
Build
Messages
Building configuration: P1 - Debug
Updating build tree...
3 file(s) deleted.
Updating build tree...
asm.s90
Linking
Total number of errors: 0
Total number of warnings: 0
Ready
Errors 0, Warnings 0
NUM

Directive de control al simbolurilor

Aceste directive controlează modul în care simbolurile sunt împărțite între module.

Directivă	Descriere
EXTERN (IMPORT)	are ca efect importarea unui simbol extern.
PUBLIC (EXPORT)	face posibilă exportarea simbolurilor către alte module.
PUBWEAK	face posibilă exportarea simbolurilor către alte module, permițând mai multe definiri.
REQUIRE	forțează referențierea unui simbol.

Sintaxă

```
EXTERN simbol [,simbol] ...
PUBLIC simbol [,simbol] ...
PUBWEAK simbol [,simbol] ...
REQUIRE simbol
```

Parametri

- **simbol**- simbol ce poate fi importat sau exportat

Descriere

Exportarea simbolurilor către alte module

Se folosește *PUBLIC* pentru a face unul sau mai multe simboluri disponibile altor module. Simbolurile declarate ca *PUBLIC* își pot schimba locația sau pot fi absolute, și pot fi folosite în expresii(cu aceleași reguli ca alte simboluri).

Directiva *PUBLIC* exportă mereu valori de 32 de biți, ce ajută la folosirea constantelor globale de 32 de biți în asambloare pentru procesoare de 8 sau 16 biți. Cu operatorii LOW, HIGH, >> și <<, orice parte a acestei constante poate fi încărcată în registre de 8 sau 16 biți sau cuvinte.

Nu sunt alte restricții cu privire la numărul de simboluri declarate cu *PUBLIC* într-un modul.

Exportarea simbolurilor cu mai multe definiții către alte module

PUBWEAK este similar cu *PUBLIC*, cu excepția că permite aceluiași simbol să fie definit de mai multe ori. Doar una din definiții va fi folosită de XLINK. Dacă un modul conține o definiție a unui simbol declarată cu *PUBLIC* link-editat cu una sau mai multe module ce conțin o definire *PUBWEAK* a aceluiași simbol, XLINK va folosi definirea cu *PUBLIC*. Dacă sunt mai multe definiri cu *PUBWEAK*, XLINK o va folosi pe prima.

Un simbol definit cu *PUBWEAK* trebuie să aibă fie o etichetă în partea de segment și trebuie să fie singurul simbol definit ca *PUBLIC* sau *PUBWEAK* în partea de segment.
Observație: Modulele bibliotecă sunt linkeditate dacă există o referențiere a unui simbol al acelui modul este făcută și simbolul nu a fost deja linkeditat. În faza selecției modulelor, nu se face diferență între definirile cu *PUBLIC* sau *PUBWEAK*. Asta înseamnă că pentru a se asigura că definirea cu *PUBLIC* este selectată, ar trebui link-editată înaintea altor module sau a se verifica că există o referință la alt simbol *PUBLIC* în acel modul.

Importarea simbolurilor

Se folosește *EXTERN* pentru a importa un simbol extern.

Directiva *REQUIRE* marchează un simbol ca fiind referențiat. Este folositor dacă partea de segment ce conține simbolul trebuie să fie încărcată chiar dacă codul nu este referențiat.

Exemplu

Următorul exemplu definește o subrutină pentru a afișa un mesaj de eroare și exportă adresa de intrare err pentru a putea fi apelată de alte submodule. Definește printeaza ca o rutină externă; adresa va fi determinată la link-editare.

```
    NAME eroare
    EXTERN printeaza
    PUBLIC err
err RCALL printeaza
    DB  "** Eroare **"
    EVEN
    RET

    END
```

Directive de control al segmentelor

Directivele de control al segmentelor controlează modul în care codul și datele sunt generate.

Directivă	Descriere
ALIGN	aliniază numărătorul de locație prin inserarea de octeți nuli.
ASEG	începe un segment absolut
ASEGN	începe un segment absolut cu nume
COMMON	începe un segment comun
EVEN	aliniază registrul PC la o adresă pară
ODD	aliniază registrul PC la o adresă impară
ORG	setează valoarea numărătorului de locație
RSEG	începe un segment relocatabil
STACK	începe un segment de tip stivă

Sintaxă

ALIGN align [,value]
ASEG [start [(align)]]

ASEGN segment [:type], address
COMMON segment [:type] [(align)]
EVEN [value]
ODD [value]
ORG expr
RSEG segment [:type] [flag] [(align)]
RSEG segment [:type], address
STACK segment [:type] [(align)]

Parametri

- **address** este o adresă unde partea de segment va fi plasată.
- **align** este exponentul valorii adresei ce ar trebui aliniată, în intervalul 0, 30.
- **expr** este adresa la care se va seta numărătorul de locație.
- **flag** poate lua valorile *NOROOT*, *ROOT*, *REORDER*, *SORT*.

Dacă flag este *NOROOT*, atunci segmentul poate fi ignorat de link-editor dacă nu sunt referențiate de alte module simboluri din acest segment. În mod normal, toate părțile segmentului cu excepția codului de startup și vectorii de întreruperi ar trebui să seteze acest flag. Modul de bază e *ROOT*, ce indică că această parte de segment nu trebuie ignorată.
Dacă flag e *REORDER*, atunci link-editorul poate rearanja părțile segmentului. Pentru un segment dat, toate părțile trebuie să specifice valoarea acestui flag. Modul de bază este ca reordonarea să nu se facă.
Dacă flag e *SORT*, link-editorul poate sorta părți din segment în ordine descrescătoare a aliniamentului. Pentru un segment dat, toate părțile lui trebuie să specifice valoarea acestui flag. Modul de bază e ca sortarea să nu se facă.

- **segment** este numele segmentului.
- **start** este o adresă de început care are același efect ca folosirea directivei ORG la începutul unui segment absolut.
- **type** este tipul memoriei, cel mai des CODE sau DATA. Dar poate fi orice tip suportat de IAR XLINK Linker.
- **value** este o valoare pe 8 biți de umplutură, în mod standard este zero.

Descriere

Începerea unui segment absolut

Se folosește *ASEG* pentru a seta modul absolut al asamblării, ce este modul standard la începutul unui modul.

Dacă parametrul este omis, adresa de început a primului segment este 0, și fiecare segment continuă de la următoarea adresă a terminării segmentului anterior.

Începerea unui segment relocatabil

Se folosește *RSEG* pentru a seta modul de asamblare la relocatabil. Asamblorul menține numărătoare separate de locație(inițial setate pe zero) pentru toate segmentele, ceea ce face posibilă schimbarea între segmente și mod oricând, fără a fi necesară salvarea valorii curente a numărătorului de locație.

Până la 65536 de segmente unice, relocatabile, pot fi definite într-un singur modul.

Începerea unui segment stivă

Se folosește *STACK* pentru a aloca cod sau date alocate de la o adresă mare la mică(în contrast cu directiva *RSEG* care realizează alocare de la mic la mare).

Observație: Conținutul segmentului nu este generat invers.

Începerea unui segment comun

Se folosește *COMMON* pentru a plasa date în memorie la aceeași locație ca alte segmente *COMMON* din alte module, în același timp. În alte cuvinte, toate segmentele *COMMON* de același nume vor începe la aceeași locație de memorie și se vor acoperi unul pe altul.

Bineînțeles, segmentul *COMMON* nu ar trebui folosit pentru acoperiri de cod executabil. O aplicație tipică ar fi atunci când mai multe subrutine folosesc o locație comună, reutilizabilă în memorie pentru date.

Poate fi practică definirea vectorilor de întrerupere în segmentul *COMMON*, astfel acordând acces la datele vectorului din mai multe rutine.

Dimensiunea finală a unui segment *COMMON* este determinată de cea mai mare apariție a acestui segment. Locația în memorie este determinată de comanda
XLINK -z; A se vedea IAR Linker and Library Tools Reference Guide.

Se folosește parametrul **align** în toate directivele de mai sus pentru a alinia segmentul la adresa de început.

Setarea numărătorului pentru locația programului(PLC)

Se folosește *ORG* pentru a seta valoarea PLC al segmentul curent la valoarea unei expresii. Eticheta opțională va lua valoarea și tipul noii locații a numărătorului.

Rezultatul expresiei trebuie să fie de același tip cu segmentul curent, de exemplu nu este validă folosirea *ORG* 10 în timpul unui *RSEG*, deoarece expresia este absolută;
se folosește *ORG* $+10. Expresia nu trebuie să conțină referințe în față sau externe.

Toate PLC sunt setate pe zero la începutul unui modul în asamblare.

Alinierea unui segment

Se folosește *ALIGN* pentru a alinia PLC la un interval specificat de adrese. Expresia are ca rezultat puterea lui doi la care numărătorul de program ar trebui aliniat.

Aliniamentul este făcut relativ la începutul segmentului; în mod normal, asta înseamnă că aliniamentul segmentului trebuie să fie măcar la fel de mare ca aliniamentul directivei pentru a obține rezultatul corect.

ALIGN aliniază prin inserarea de octeți nuli/plini. Directiva *EVEN* aliniază numărătorul de program la o adresă pară(ceea ce e echivalent cu *ALIGN* 1) și directiva *ODD* aliniază numărătorul de program la o adresă impară.

Exemple

Începerea unui segment absolut

Următorul exemplu asamblează instrucțiuni de intrare într-o rutină de întrerupere în vectorii de întrerupere corespunzători folosind un segment absolut:

```
EXTERN EINT1, EINT2, RESET

ASEG INTVEC
ORG 0h

RJMP RESET
RJMP EINT1
RJMP EINT2
END
```

Începerea unui segment relocatabil

În următorul exemplu, datele ce urmează primei directive RSEG sunt plasate într-un segment relocatabil numit table; directiva ORG este folosită pentru a creea un spațiu gol de 6 octeți în table.

Codul după cea de-a doua directivă RSEG este plasat într-un segment relocatabil numit code:

```
module baza
EXTERN Table1,Table2

        RSEG TABLES
        DC16 Table1, Table2

        ORG $+6
        DC16 Table3

        RSEG CONST

Table3 DC8 1,2,4,8,16,32
        END
```

Începerea unui segment stivă

Următorul exemplu definește două segmente relocatabile numite rpnstack:

```
module liste
STACK    rpnstack
parametri        DS8        100
operatori        DS8        100
END
```

- Datele sunt alocate de la adrese mari la mici.

Începerea unui segment comun

Următorul exemplu definește două exemple comune ce conțin variabile:

```
        NAME    common1
        COMMON  data
count   DD      1
        ENDMOD

        NAME    common2
        COMMON  data
up      DB      1
        ORG     $+2
down    DB      1
        END
```

- Pentru că segmentele comune au același nume, data, variabilele up și down vor referi aceleași adrese de memorie ca primii și ultimii 4 biți ai variabilei count.

Alinierea unui segment

Acest exemplu începe un segment relocatabil, se mută la o adresă pară și adaugă niște date. După se aliniază la o limită de 64 de octeți înainte de a crea un tabel de 64 de octeți.

```
            RSEG    data ; Începe un segment relocatabil cu date
            EVEN           ; Se asigura ca este pe o adresa para
tinta       DC16    1     ; target si best vor fi la adrese pare
test        DC16    1
            ALIGN           6 ; Aliniere la o regiune de 64 octeti
rezultate   DS8     64 ; Creeaza o tabela de 64 de octeti
            END
```

Operatori ai asamblorului

Precedența operatorilor

Fiecare operator are un număr de precedență asociat ce determină ordinea în care operatorul și operanzii sunt evaluați. Numerele de precedență variază de la 1(cea mai mare precedență, evaluat primul) la 7(precedența cea mai mică, evaluat ultimul).

Următoarele reguli determină modul în care sunt evaluate expresiile:

- Operatorii cu cea mai mare precedență sunt evaluați mai întâi, apoi cei cu a doua cea mai mare și tot așa până când sunt evaluați și cei cu precedența cea mai mică.
- Operatorii cu precedență egală sunt evaluați de la stânga la dreapta în expresie.
- Parantezele pot fi folosite pentru a grupa operatorii și operanzii și pentru a controla ordinea în care expresiile sunt evaluate.

De exemplu, expresia 7/(1+(2*3)) are ca rezultat 1.

Rezumatul operatorilor

Următoarele tabele realizează rezumatul operatorilor, după prioritate. Sinonimele, dacă sunt disponibile, sunt afișate după prima formă a operatorului.

Operatori unari- precedența 1

+	Plus unar
-	Minus unar
NOT, !	NOT logic
BITNOT, ~	NOT pe biti
LOW	Octetul din partea LOW
HIGH	Octetul din partea HIGH
BYTE2	Al doilea octet
BYTE3	Al treilea octet
LWRD	Cuvantul din partea LOW
HWRD	Cuvantul din partea HIGH
DATE	Timp/data curente
SFB	Început de segment
SFE	Sfârșit de segment
SIZEOF	Dimensiune segment

Operatori multiplicativi aritmetici și de shiftare- precedența 3

*	Multiplicare
/	Împărțire întreagă
MOD, %	Modulo(rest la împărțirea întreagă)
SHR, >>	Shiftare logică la dreapta
SHL, <<	Shiftare logică la stânga

Operatori aditivi aritmetici- precedența 4

+	Adunare
-	Scădere

Operatori și- precedența 5

AND, &&	ȘI logic
BITAND, &	ȘI pe biți

Operatori sau- precedența 6

OR, ||	OR logic
BITOR, |	OR pe biți
XOR	XOR logic
BITXOR, ^	XOR pe biți

Operatori de comparație- precedența 7

EQ, =, ==	Egal
NE, <>, !=	Nu este egal
GT, >	Mai mare ca
LT,<	Mai mic ca
UGT	Mai mare ca unsigned
ULT	Mai mic ca unsigned
GE, >=	Mai mare sau egal
LE, <=	Mai mic sau egal

Descrierea operatorilor

Următoarele secțiuni vor da descrieri detaliate ale fiecărui operator de asamblare. Numărul dintre paranteze specifică prioritatea(precedența) operatorului, iar minusul imediat după operator realizează separația dintre operator și numele acestuia.

*- Multiplicare(3)

* produce rezultatul înmulțirii celor 2 operanzi. Operanzii sunt luați ca întregi cu semn pe 32 de biți și rezultatul este tot un întreg cu semn pe 32 de biți.

Exemple
2*2->4
-2*2->-4

+ -Plus unar(1)

Unar plus operator

Exemple

+3->3
3*+2->6

+ -Adunare(4)

Operatorul produce suma celor doi operanzi ce îl înconjoară. Operanzii sunt luați ca întregi cu semn pe 32 de biți și rezultatul este tot un întreg cu semn pe 32 de biți.

Exemple

92+19 -> 111
-2+2 -> 0
-2+-2 -> -4

- -Minus unar(1)

Operatorul produce negarea aritmetică a operandului său.

Operandul e interpretat ca număr cu semn pe 32 de biți, deci rezultatul este complementul față de 2 al negării acelui întreg.

Exemple
-3 -> -3
3*-2 -> -6
4--5 -> 9

- -Scădere(4)

Operatorul produce rezultatul scăderii operatorului din dreapta din cel din stânga.

Operanzii sunt luați ca întregi cu semn pe 32 de biți și rezultatul este tot un întreg cu semn pe 32 de biți.

Exemple

92-19 -> 73
-2-2 -> -4
-2--2 -> 0

/ -Împărțire(3)

Operatorul produce rezultatul împărțirii întregi a operatorului din stânga la cel din dreapta. Operanzii sunt luați ca întregi cu semn pe 32 de biți și rezultatul este tot un întreg cu semn pe 32 de biți.

Exemple

9/2 -> 4
-12/3 -> -4

9/2*6 -> 24

AND, &&-ȘI logic(5)

Se folosește pentru a face un ȘI între doi operanzi întregi. Dacă ambii operanzi sunt nenuli, rezultatul e 1, altfel 0.

Exemple

B’1010 && B’0011 -> 1
B’1010 && B’0101 -> 1
B’1010 && B’0000 -> 0

BITAND, &-AND pe biți(5)

Se folosește pentru a face un ȘI pe biți între cei doi operatori.

Exemple

B’1010 & B’0011 -> B’0010
B’1010 & B’0101 -> B’0000
B’1010 & B’0000 -> B’0000

BITNOT, ~ -NOT pe biți(1)

Se folosește pentru a nega un operand pe biți.

Exemplu

~ B’1010 -> B’11111111111111111111111111110101

BITOR, | -SAU pe biți(6)

Se folosește pentru a face un SAU pe biții operanzilor.

Exemple

B’1010 | B’0101 -> B’1111
B’1010 | B’0000 -> B’1010

BITXOR, ^ -XOR pe biți(6)

Se folosește pentru a face un XOR pe biții operandului.

Exemplu

B’1010 ^ B’0101 ->B’1111

BYTE2-Al doilea octet(1)

Acest operator ia un singur operand, ce este interpretat ca un întreg fără semn pe 32 de biți. Rezultatul este format din biții de la 15 la 8 ai operandului.

Exemplu

BYTE2 0x12345678 -> 0x56

BYTE3-Al treilea octet(1)

Acest operator ia un singur operand, ce este interpretat ca un întreg fără semn pe 32 de biți. Rezultatul este format din biții de la 23 la 16 ai operandului.

Exemplu

BYTE3 0x12345678 -> 0x34

DATE-Timpul/data curente

Se folosește pentru a specifica când a început asamblarea.

Acest operator ia un argument absolut și returnează:
DATE 1 Secunda curentă (0–59).
DATE 2 Minutul curent (0–59).
DATE 3 Ora curentă (0–23).
DATE 4 Ziua curentă (1–31).
DATE 5 Luna curentă (1–12).
DATE 6 Anul curent MOD 100 (1998 -> 98, 2000 -> 00, 2002 -> 02).

Exemplu

Pentru a stoca ziua asamblării
today: DC8 DATE 5, DATE 4, DATE 3

EQ, =, == -Egalitate(7)

Se evaluează la 1(adevărat) dacă cei doi operanzi sunt egali și 0 altfel.

Exemple

1=2 -> 0

2 == 2 -> 1
'ABC' = 'ABCD' -> 0

GE, >=- Mai mare sau egal(7)

Se evaluează la 1 dacă operandul din stânga este mai mare sau egal în mod numeric decât cel din dreapta, altfel 0.

Exemple

1 >= 2 -> 0
2 >= 1 -> 1
1 >= 1 -> 1

GT, > -Mai mare ca(7)

Se evaluează la 1 dacă operandul din stânga este mai mare în mod numeric decât cel din dreapta, altfel 0.

Exemple

-1 > 1 -> 0
2>1 -> 1
1>1 -> 0

HIGH-Octetul HIGH(1)

Ia un singur operand în dreapta ce este interpretat ca un întreg pe 16 biți, fără semn. Rezultatul este format din primii săi 8 biți.

Exemplu

HIGH 0xABCD -> 0xAB

HWRD-Cuvânt HIGH(1)

Ia un singur operand în dreapta ce este interpretat ca un întreg pe 32 biți, fără semn. Rezultatul este format din biții de la 31 la 16 ai operandului.

Exemplu

HWRD 0x12345678 -> 0x1234

LE, <= -Mai mic sau egal(7)

Se evaluează la 1 dacă operandul din stânga este mai mic sau egal în mod numeric decât cel din dreapta, altfel 0.

Exemple

1 <= 2 -> 1
2 <= 1 -> 0
1 <= 1 -> 1

LOW-Octetul low(1)

Ia un singur operand în dreapta ce este interpretat ca un întreg pe 16 biți, fără semn. Rezultatul este format din ultimii săi 8 biți.

Exemplu

LOW 0xABCD -> 0xCD

LT,< -Mai mic decât(7)

Se evaluează la 1 dacă operandul din stânga este mai mic în mod numeric decât cel din dreapta, altfel 0.

Exemple

-1 < 2 -> 1
2<1 -> 0
2<2 -> 0

LWRD -Cuvânt LOW(1)

Ia un singur operand în dreapta ce este interpretat ca un întreg pe 32 biți, fără semn. Rezultatul este format din biții de la 15 la 0 ai operandului.

Exemplu

LWRD 0x12345678 -> 0x5678

MOD, % -Modulo(3)

Operatorul produce restul împărțirii întregi a operandului din stânga la cel din dreapta.

Operanzii sunt numere pe 32 de biți cu semn, și rezultatul este tot un număr pe 32 de biți cu semn.

X%Y este echivalent cu X-Y*(X/Y) folosind împărțirea întreagă.

Exemple

2%2 -> 0
12 % 7 -> 5
3%2 -> 1

NE, <>, != -Nu e egal(7)

Se evaluează cu 0 dacă operanzii sunt egali în valoare, altfel 1.

Exemple

1 <> 2 -> 1
2 <> 2 -> 0
'A' <> 'B' -> 1

NOT, !- NOT logic(1)

Se folosește ! pentru a nega o expresie logică.

Exemple

! B’0101 -> 0
! B’0000 -> 1

OR, || -OR logic(6)

Se folosește pentru a face un SAU logic între doi operatori întregi.

Exemple

B’1010 || B’0000 -> 1
B’0000 || B’0000 -> 0

SFB-Început de segment(1)

Sintaxă

SFB(segment [{+|-}offset])

Parametri

- **segment** este numele unui segment relocatabil, ce trebuie să fie definit înainte ca SFB să fie folosit.
- **offset** este un decalaj opțional de la adresa de început. Parantezele sunt opționale dacă offset lipsește.

Descriere

SFB acceptă un singur operand la dreapta. Operandul trebuie să fie numele unui segment relocatabil.

Operatorul se evaluează până la adresa absolută a primului octet al segmentului. Această evaluare are loc la runtime.

Exemplu

```
      NAME demo
      RSEG CODE
start:DC16 SFB(CODE)
```

- Chiar dacă codul de mai sus este link-editat cu mai multe module, start va fi setat la adresa primului octet din segment.

SFE-Sfârșit de segment(1)

Sintaxă

SFE(segment [{+|-}offset])

Parametri

- **segment** este numele unui segment relocatabil, ce trebuie să fie definit înainte ca SFE să fie folosit.
- **offset** este un decalaj opțional de la adresa de început. Parantezele sunt opționale dacă offset lipsește.

Descriere

SFE acceptă un singur operand la dreapta. Operandul trebuie să fie numele unui segment relocatabil. Operatorul se evaluează la adresa de început a segmentului plus dimensiunea segmentului. Această evaluare are loc la runtime.

Exemplu

```
    NAME demo
    RSEG CODE
end:DC16 SFE(CODE)
```

- Chiar dacă codul de mai sus este link-editat cu mai multe module, end va fi setat la adresa ultimului octet din segment.

Dimensiunea segmentului MY_SEGMENT poate fi calculată ca:
SFE(MY_SEGMENT)-SFB(MY_SEGMENT)

SHL, << -Shiftare logică la stânga(3)

Operatorul este folosit pentru a shifta la stânga primul operand considerat ca fiind fără semn. Numărul de shiftări este specificat de al doilea operand, interpretat ca valoare întreagă între 0 și 32.

Exemple

B’00011100 << 3 -> B’11100000
B’0000011111111111 << 5 -> B’11111111111100000
14 << 1 -> 28

SHR, >> -Shiftare logică la dreapta(3)

Operatorul este folosit pentru a shifta la dreapta primul operand considerat ca fiind fără semn. Numărul de shiftări este specificat de al doilea operand, interpretat ca valoare întreagă între 0 și 32.

Exemple

B’01110000 >> 3 -> B’00001110
B’1111111111111111 >> 20 -> 0
14 >> 1 -> 7

SIZEOF-Dimensiunea segmentului(1)

Sintaxă

SIZEOF segment

Parametri

- **segment** este numele unui segment relocatabil ce trebuie să fie definit înainte de folosirea lui SIZEOF.

Descriere

SIZEOF generează SFE-SFB pentru argumentul său, ce ar trebui să fie numele unui segment relocatabil; calculează dimensiunea în octeți a unui segment. Acest lucru se face când modulele sunt link-editate împreună.

Exemplu

```
        NAME demo
        RSEG CODE
size:DC16 SIZEOF CODE
```

- Setează size la dimensiunea segmentului CODE.

UGT-Mai mare ca unsigned(7)

UGT se evaluează la 1 dacă operandul din stânga are o valoare mai mare ca cel din dreapta. Operatorul tratează operanzii ca fiind fără semn.

Exemple

2 UGT 1 -> 1
-1 UGT 1 -> 1

ULT-Mai mic decât unsigned(7)

ULT se evaluează la 1 dacă operandul din stânga are o valoare mai mică ca cel din dreapta. Operatorul tratează operanzii ca fiind fără semn.

Exemple

1 ULT 2 -> 1
-1 ULT 2 -> 0

XOR-XOR logic(6)

Este folosit pentru a determina operația logică XOR între doi operanzi.

Exemple

B’0101 XOR B’1010 -> 0
B’0101 XOR B’0000 -> 1

Particularizarea instrucțiunilor

În acest capitol vor fi particularizate instrucțiunile, prezentându-se descrierea acestora, operația pe care o realizează, sintaxa, operanzii, efectul asupra numărătorului de program, codificarea operației, flag-urile afectate, spațiul de memorie ocupat și cât durează execuția instrucțiunii în ciclii de procesor.

MOV

Copie registrul

Descriere

Instrucțiunea face o copie a unui registru în alt registru. Registrul sursă Rr nu este schimbat, în timp ce registrul destinație Rd este încărcat cu o copie a lui Rr.

Operație

Rd<-Rr

Sintaxă

MOV Rd,Rr

Operanzi

0<=d<=31, 0<=r<=31

Registrul PC(program counter)

PC<-PC+1

Codificarea instrucțiunii pe 16 biți

0010 11rd dddd rrrr

Observații

- Instrucțiunea nu afectează niciun flag.
- Instrucțiunea are dimensiunea unui cuvânt(2 octeți)
- Instrucțiunea durează un ciclu.

LDI

Încarcă constantă

Descriere

Încarcă direct o valoare pe 8 biți fără semn în registrul dat ca parametru, dar se aplică pentru regiștrii de la R16 la R31.

Operație

Rd<-K

Sintaxă

LDI Rd,K

Operanzi

16<=d<=31, 0<=K<=255

Registrul PC

PC<-PC+1

Codificarea instrucțiunii pe 16 biți

1110 KKKK dddd KKKK

Observații

- Instrucțiunea nu afectează flag-uri.
- Instrucțiunea ocupă un cuvânt de 2 octeți.

- Instrucțiunea se execută într-un ciclu al procesorului.

ADD

Adunare fără carry

Descriere

Aduna doi registri fără flag-ul C și pune rezultatul în registrul destinație Rd.

Operație

Rd<-Rd+Rr

Sintaxă

ADD Rd,Rr

Operanzi

0<=d<=31, 0<=r<=31

Registrul PC

PC<-PC+1

Codificarea instrucțiunii pe 16 biți

0000 11rd dddd rrrr

Observații

- Instrucțiunea ocupă 1 cuvânt ca spațiu.
- Executarea instrucțiunii durează un ciclu.

Flag-uri afectate

- **H**: setat dacă exista carry de la bitul 3, altfel pus pe 0
- **S**: N xor V, pentru testarea semnului
- **V**:Setat dacă a avut loc un overflow al complementului fata de 2 al rezultatului, altfel pus pe 0
- **N**: Setat dacă cel mai semnificativ bit al rezultatului este 1, altfel este pus pe 0
- **Z**: Setat dacă rezultatul adunării e 0, altfel e pus pe 0
- **C**: Setat dacă exista carry de la cel mai semnificativ bit al rezultatului, altfel pus pe 0

ADC

Adunare folosind carry

Descriere

Adună doi regiștri și conținutul flag-ului C și pune rezultatul în registrul de destinație Rd.

Operație

Rd<-Rd+Rr+C

Sintaxă

ADC Rd,Rr

Operanzi

0<=d<=31, 0<=r<=31

Registrul PC

PC<-PC+1

Codificarea instrucțiunii pe 16 biți

0001 11rd dddd rrrr

Observații

- Instrucțiunea ocupă 1 cuvânt ca spațiu.
- Executarea instrucțiunii durează un ciclu.

Flag-uri afectate

- H: setat dacă exista carry de la bitul 3, altfel pus pe 0
- S: N xor V, pentru testarea semnului
- V:Setat dacă a avut loc un overflow al complementului fata de 2 al rezultatului, altfel pus pe 0

- N: Setat dacă cel mai semnificativ bit al rezultatului este 1, altfel este pus pe 0
- Z: Setat dacă rezultatul adunării e 0, altfel e pus pe 0
- C: Setat dacă exista carry de la cel mai semnificativ bit al rezultatului, altfel pus pe 0

MOVW

Copie registru cuvânt

Descriere

Instrucțiunea face o copie a unei perechi de regiștri într-o alta pereche de regiștri.

Perechea regiștrilor de intrare Rr+1:Rr nu este modificată, în timp ce perechea de regiștri destinație Rd+1:Rd este încărcată cu o copie a Rr+1:Rr.

Operație

Rd+1:Rd<-Rr+1:Rr

Sintaxă

MOVW Rd+1:Rd,Rr+1:Rr

Operanzi

d și r sunt numere naturale pare mai mici sau egale cu 30

Registrul PC

PC<-PC+1

Codificarea instrucțiunii pe 16 biți

0000 0001 dddd rrrr

Observații

- Instrucțiunea nu afectează niciun flag.
- Instrucțiunea are dimensiunea unui cuvânt(2 octeți)
- Instrucțiunea durează un ciclu.

RET

Descriere

Revenirea din subrutină. Adresa de returnare se află pe stivă. Pointer-ul pe stivă folosește o schema de pre-incrementare în timpul instrucțiunii RET.

Operație

a) PC(15:0)<-STIVA, pentru dispozitive cu PC pe 16 biți, 128KB memorie de program maximă

b)PC(21:0)<-STIVA, pentru dispozitive cu PC pe 22 de biți, 8MB memorie de program maximă

Sintaxă

a) RET
Fără operanzi
SP<-SP+2(2 bytes, 16 biti)

b) RET
Fără operanzi
SP<-SP+3(3 bytes 22 biti)

Codificarea instrucțiunii pe 16 biți

1001 0101 0000 1000

Observații

- Instrucțiunea nu afectează niciun flag.
- Instrucțiunea ocupă 1 cuvânt.
- Instrucțiunea durează 4 cicluri pentru dispozitive cu PC pe 16 biți și 5 pentru dispozitive cu 22 de biți .

RJMP

Salt relativ

Descriere

Salt relativ la o adresă între PC-2K+1 și PC+2K(cuvinte). Pentru microcontrolerele cu mai puțin de 4K cuvinte (8kO) memorie, această instrucțiune poate adresa toată memoria de la orice adresă. Se mai poate vedea JMP(în Atmel AVR 8-bit Instruction Set).

Operație

PC←PC+k+1

Sintaxă

RJMP k

Operanzi

-2K<=k<2K

Registrul PC

PC←PC+k+1

Stivă

Nu este afectată.

Codificarea operației pe 16 biți

1100 kkkk kkkk kkkk

Observații

- Instrucțiunea nu afectează niciun flag.
- Instrucțiunea are dimensiunea unui cuvânt(2 octeți)
- Instrucțiunea durează doi cicli.

RCALL

Apel relativ la o subrutină

Descriere

Apel relativ la o adresă între PC-2K+1 și PC+2K(cuvinte). Adresa de return(instrucțiunea de după RCALL) este stocată pe stivă. Se poate vedea CALL(în Atmel AVR 8-bit Instruction Set). Pentru microcontrolerele cu mai puțin de 4K cuvinte (8kO) memorie, această instrucțiune poate adresa toată memoria de la orice adresă. Pointer-ul pe stivă folosește o schemă de post-decrementare în timpul instrucțiunii RCALL.

Operație

a) PC←PC+k+1 pentru dispozitive cu PC pe 16 biți și memorie de program de maxim 128KO

b)PC←PC+k+1 pentru dispozitive cu PC pe 22 de biți și memorie de program de maxim 8 MO

Sintaxă

a) RCALL k

b) RCALL k

Operanzi

a) -2K<=k<2K

b) -2K<=k<2K

Registrul PC

a) PC←PC+k+1

b) PC←PC+k+1

Stivă

a) STACK←PC+1 SP← SP-2(2 octeți, 16 biți)

b) STACK←PC+1 SP← SP-3(3 octeți, 22 de biți)

Codificarea operației pe 16 biți

1101 kkkk kkkk kkkk

Observații

- Instrucțiunea nu afectează niciun flag.

- Instrucțiunea are dimensiunea unui cuvânt(2 octeți)
- Instrucțiunea durează 3 cicli pentru dispozitive cu PC de 16 biți, 4 pentru dispozitive cu PC de 22 biți
- Cicli XMEGA: 2 pentru dispozitive cu PC de 16 biți, 3 pentru dispozitive cu PC de 22 biți
- Cicli tinyAVR: 4

SUB

Scădere fără carry

Descriere

Realizează scăderea între doi regiștri și pune rezultatul în registrul destinație Rd.

Operație

Rd←Rd-Rr

Sintaxă

SUB Rd,Rr

Operanzi

0<=d<=31, 0<=r<=31

Registrul PC

PC←PC+1

Codificarea operației pe 16 biți

0001 10rd dddd rrrr

Observații

- Instrucțiunea ocupă 1 cuvânt ca spațiu.
- Executarea instrucțiunii durează un ciclu.

Flag-uri afectate

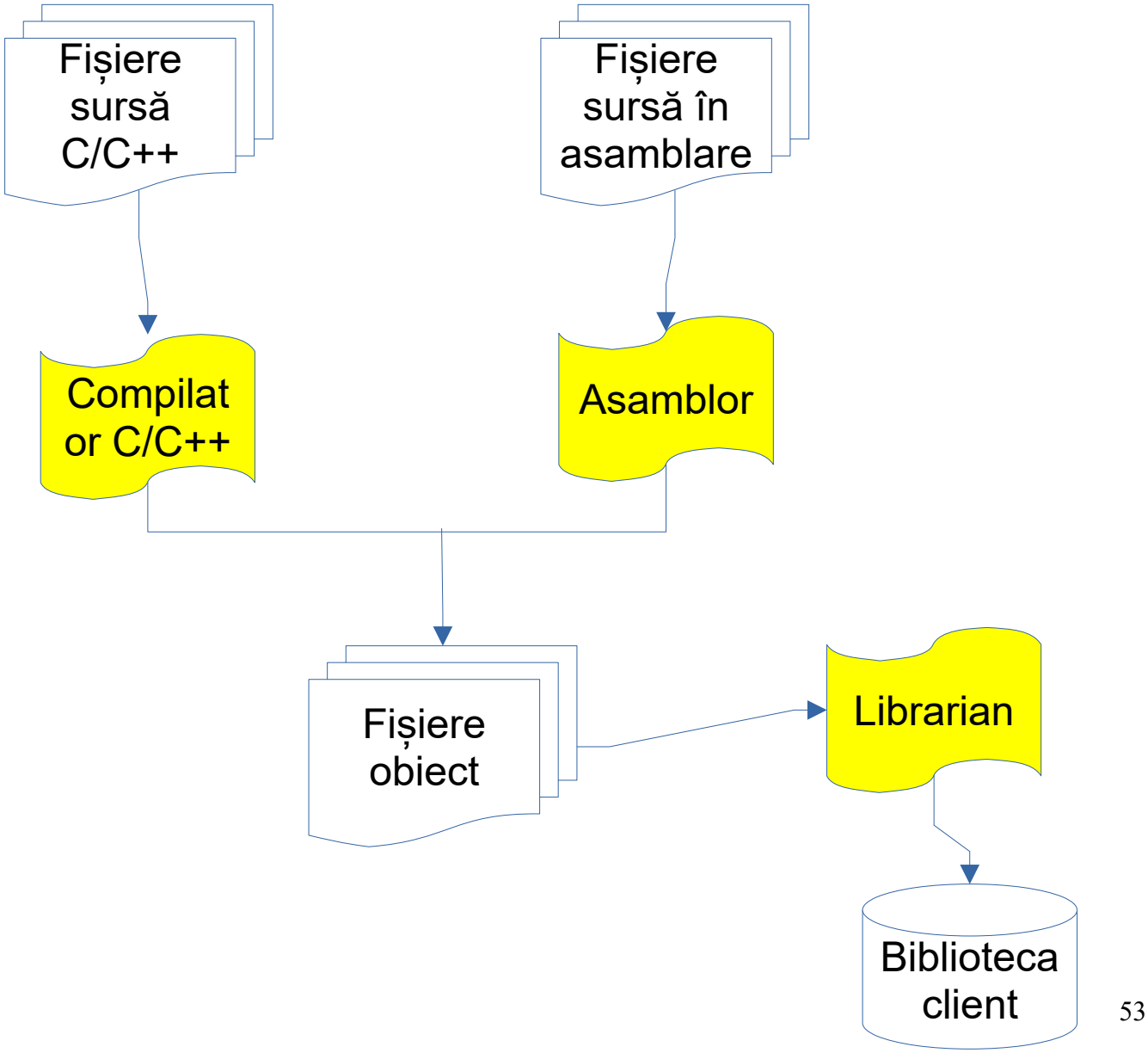
- **H**: setat dacă există împrumut de la bitul 3, altfel pus pe 0
- **S**: N xor V, pentru testarea semnului
- **V**:Setat dacă a avut loc un overflow al complementului fata de 2 al rezultatului, altfel pus pe 0
- **N**: Setat dacă cel mai semnificativ bit al rezultatului este setat, altfel este pus pe 0
- **Z**: Setat dacă rezultatul scăderii e 0, altfel e pus pe 0
- **C**: Setat dacă valoarea absolută a conținutului lui Rr e mai mare decât valoarea absolută a lui Rd, altfel pus pe 0

Procesul creerii aplicației

Această secțiune descrie modul în care se face construcția aplicației; cum diverse instrumente – compilator, asamblor și link-editor-- lucrează împreună, mergând de la cod sursă la un fișier executabil.

Procesul combinării fișierelor scrise în mai multe limbaje

Sunt instrumente în mediul de dezvoltare pentru a traduce fișierele sursă ale aplicației în fișiere obiect. Compilatorul IAR de C/C++ și Asamblorul IAR produc fișiere obiect compatibile cu formatul IAR UBROF. Procesul de traducere se desfășoară în modul următor:



După traducere, se poate alege împachetarea oricărui număr de module într-o arhivă, sau în alte cuvinte, o bibliotecă. Motivul cel mai important al folosirii unei biblioteci este că fiecare modul într-o bibliotecă este link-editat condiționat în aplicație, sau în alte cuvinte, este inclus într-o aplicație dacă este folosit direct sau indirect de un modul într-un fișier obiect. Opțional, dacă se dorește creearea unei biblioteci, se poate folosi IAR XAR Library Builder sau IAR XLIB Librarian.

Procesul de link-editare

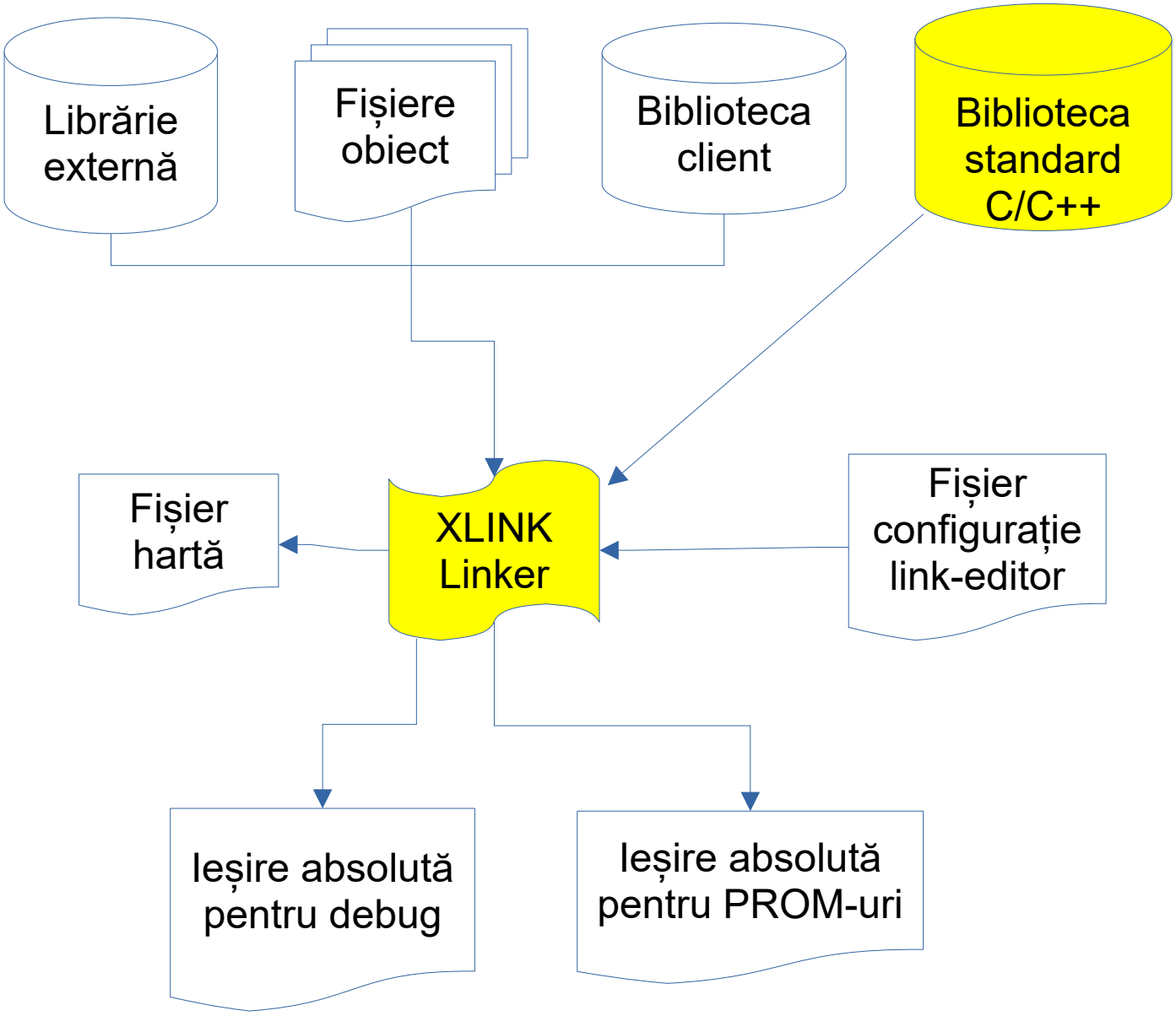
Modulele relocatabile, în fișiere obiect și librării, produse de Compilatorul IAR și asamblor nu pot fi executate direct. Pentru a deveni o aplicație executabilă, trebuie să fie link-editate.

IAR XLINK Linker este folosit pentru a construi aplicația finală. În mod normal, link-editorul are nevoie de următoarele informații ca intrări:

- Câteva fișiere obiect și posibil câteva biblioteci
- Biblioteca standard ce conține mediul de runtime și funcțiile standard ale limbajului
- O etichetă de start a programului (setată implicit)
- Fișierul de configurare al link-editorului ce descrie locul în care este stocat codul și data în memoria sistemului destinație
- Informații despre formatarea fișierului de ieșire

IAR XLINK Linker produce fișiere de ieșire după specificațiile date. Se alege formatul de ieșire ce îndeplinește cerințele. Se poate dori încărcarea ieșirii de la un debugger—ceea ce înseamnă nevoia ca fișierul de ieșire să aibă informații de debug. Alternativ, poate se dorește încărcarea ieșirii într-un programator PROM sau flash-- în acest caz este necesară ieșirea fără informațiile de debug, cum ar fi în cod hexa pentru Intel sau S-records de la Motorola. Opțiunea -F poate fi folosită pentru a specifica formatarea ieșirii.

Următorul desen arată procesul de link-editare:

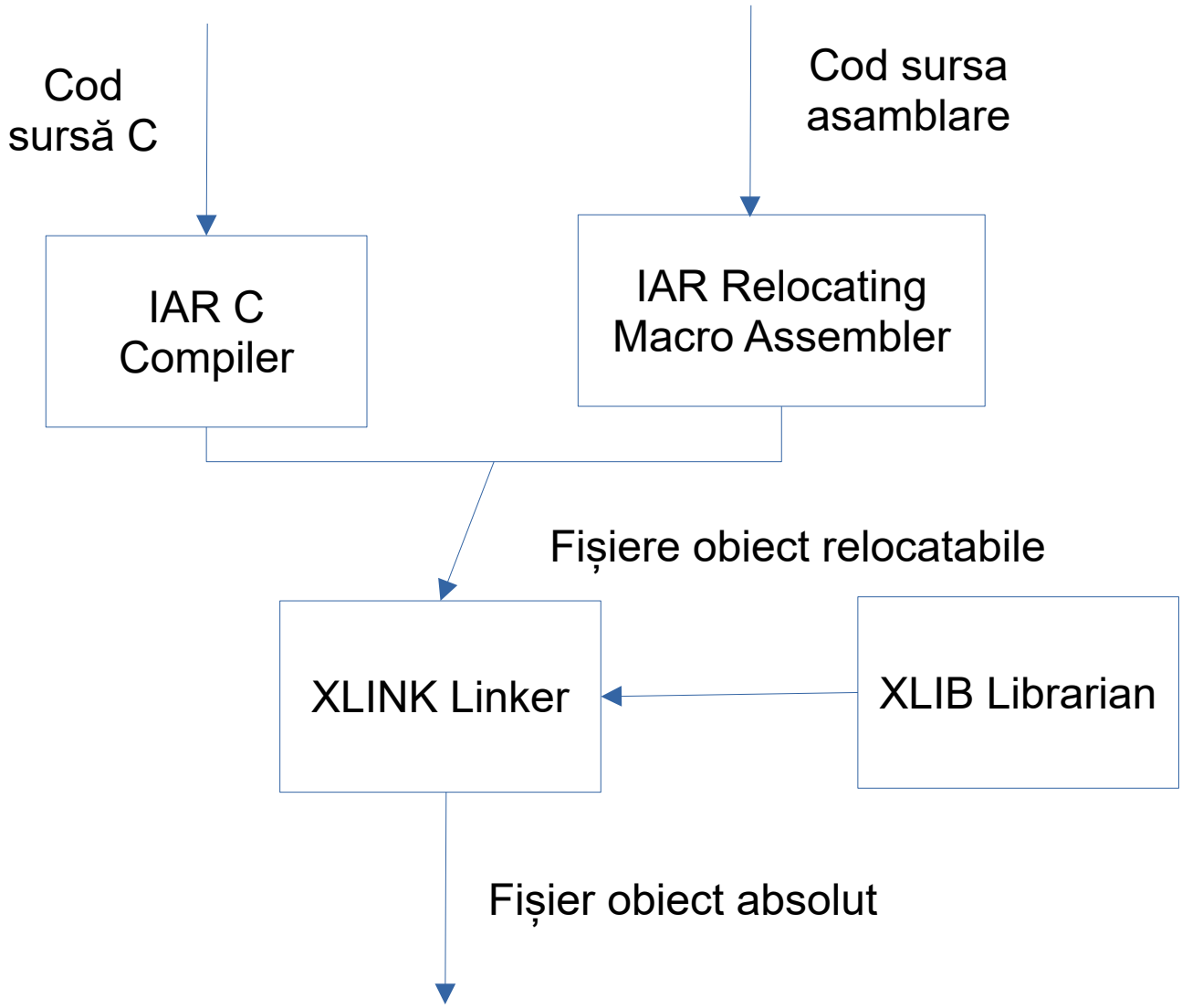


Observație: Biblioteca standard C/C++ conține rutine suport pentru compilator, și implementarea funcțiilor standard C/C++.

În timpul link-editării, link-editorul poate produce mesaje de eroare și de jurnal la ieșirea standard și la ieșirea standard de eroare. Mesajele de jurnal sunt folositoare pentru a înțelege de ce o aplicație a fost link-editată într-un mod. De exemplu, de ce a fost inclus un modul sau o secțiune a fost ștearsă.

Pentru mai multe informații despre procedura realizată de link-editor se poate accesa IAR Linker and Library Tools Reference Guide.

Procesul de link-editare mai poate fi explicat și prin următorul desen:

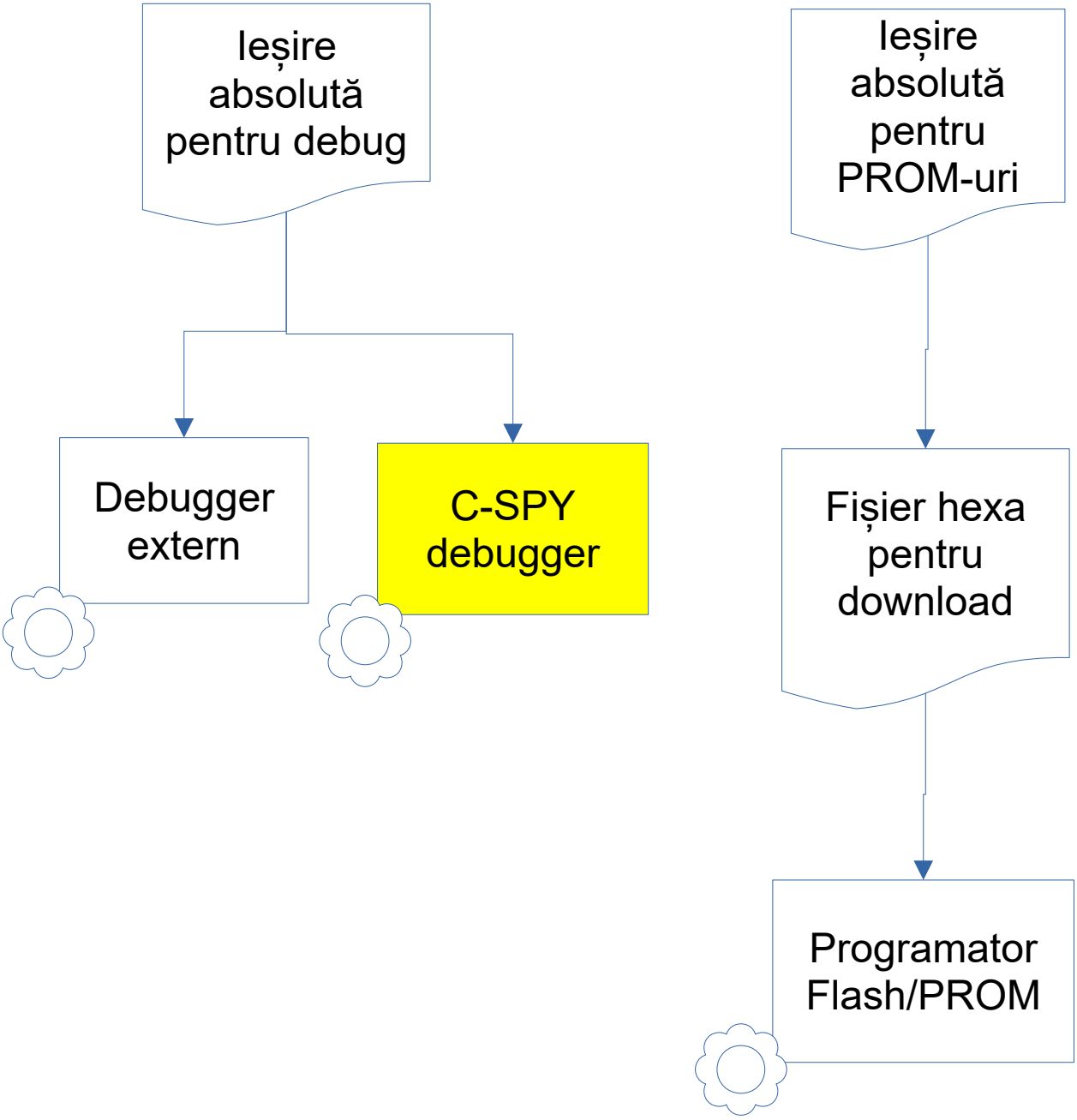


După link-editare

IAR XLINK Linker produce un fișier obiect absolut în formatul de ieșire specificat. După link-editare, fișierul executabil produs poate fi folosit pentru:

- Încărcarea în IAR C-SPY Debugger sau orice alt debugger care citește UBROF.
- Încărcarea unui flash/PROM folosind un programator.

Următoarea ilustrație arată moduri posibile în care poate fi folosit fișierul absolut de ieșire:



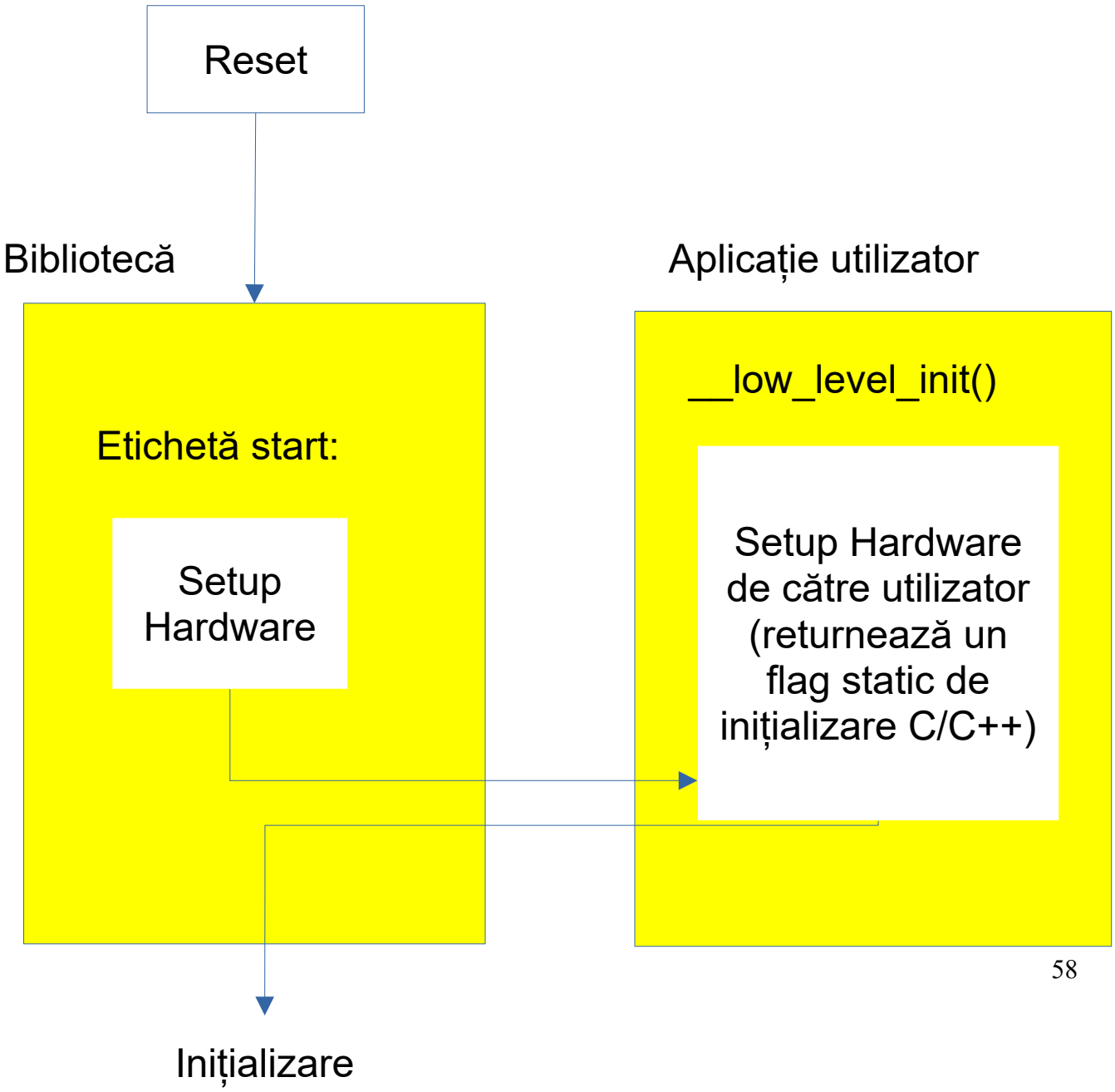
Executarea aplicației

Această secțiune descrie acțiunile executate la pornirea aplicației, în timpul execuției și la oprirea aplicației.

Codul care se ocupă cu pornirea și oprirea este localizat în fișierele sursă *cstartup.s90*, *_exit.s90* și l*ow_level_init.c* localizate în directorul avr/src/lib.

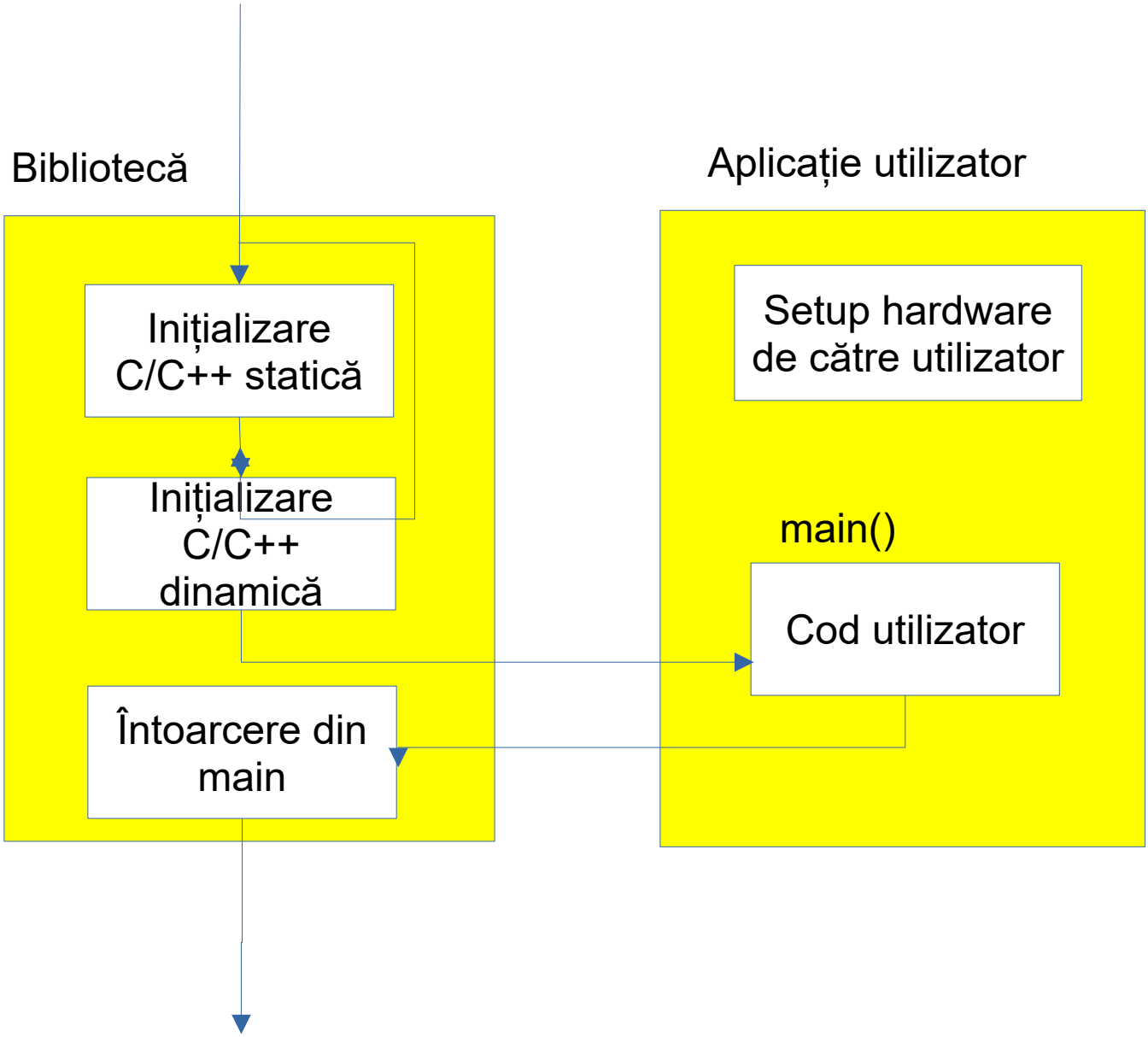
Pornirea sistemului

În timpul pornirii sistemului se execută o secvență de inițializare înainte de intrarea în funcția main. Această secvență realizează inițializări necesare pentru sistemul țintă și mediul C/C++. Pentru sistemul țintă, inițializarea arată așa:



- Când procesorul este resetat, va executa cod de la eticheta de intrare *__program_start* în codul de pornire al sistemului.
- Magistralele de date și adresă sunt activate dacă sunt necesare.
- Pointerii pe stivă sunt inițializați la sfârșitul CSTACK și RSTACK, respectiv.
- Funcția *__low_level_init* este apelată dacă a fost definită, dându-i aplicației o șansă de a face inițializări.

Inițializarea în C/C++ arată așa:



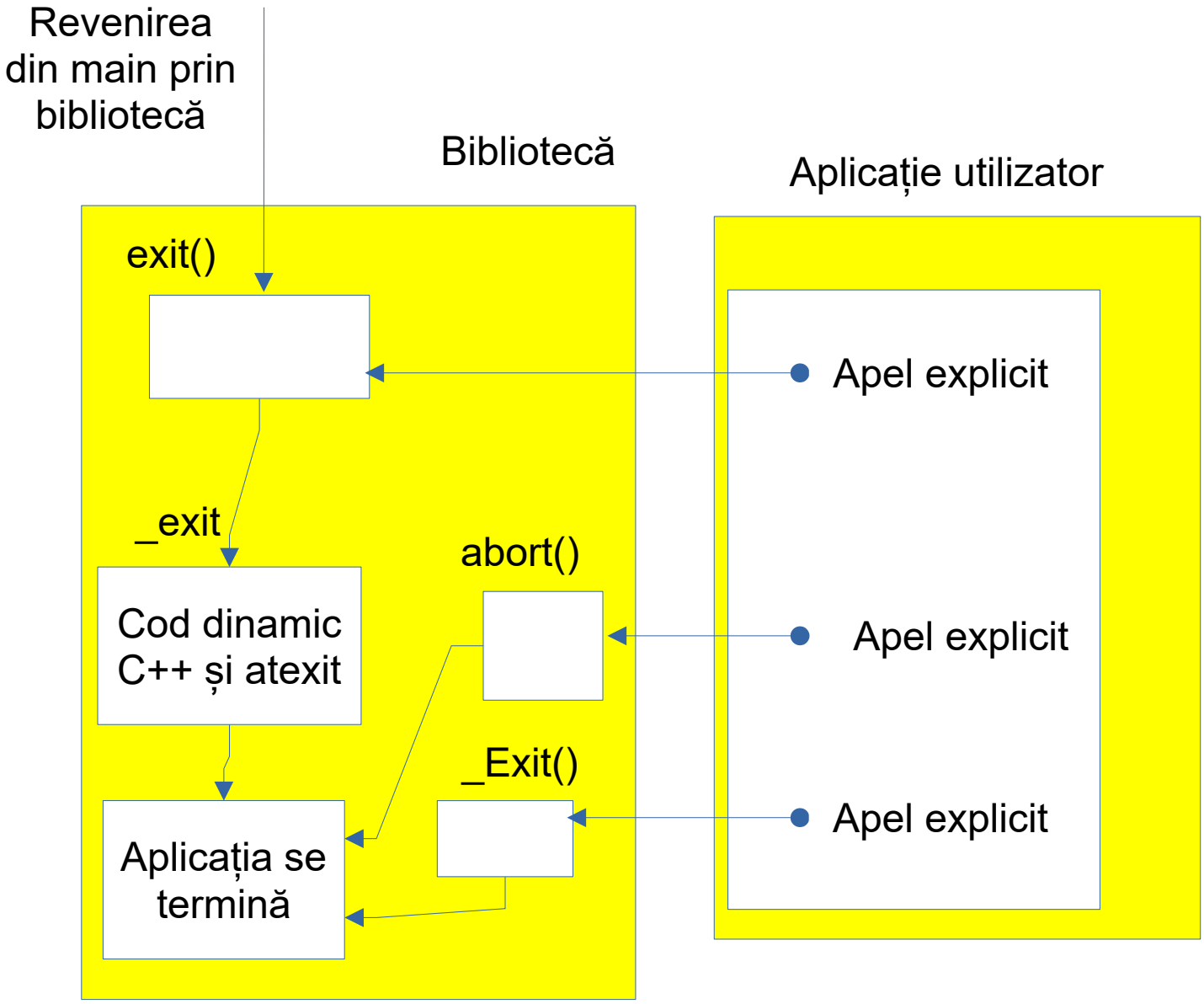
- Variabilele statice și globale sunt inițializate, cu excepția celor declarate ca *__no_init, __tinyflash, __flash, __farflash, __hugeflash* și *__eeprom*. Sunt inițializate cu valoarea zero și alte valori ale variabilelor inițializate sunt copiate din ROM în RAM. Acest pas este sărit dacă *__low_level_init* returnează zero.
- Obiectele statice C++ sunt construite
- Funcția *main* este apelată, ceea ce duce la pornirea aplicației.

Funcția main

Funcția apelată la începutul programului este numită *main*. Nu există prototip declarat pentru *main*, și singura definiție suportată pentru *main* este:
int main(void)

Oprirea sistemului

Următorul desen descrie diferitele moduri în care o aplicație se poate termina într-un mod controlat.



O aplicație se poate termina în mod normal în două căi:

- Returnarea din funcția *main*
- Apelarea funcției *exit*.

Deoarece standardul C spune că cele două metode trebuie să fie echivalente, codul de pornire al sistemului apelează funcția *exit* dacă se face return în *main*. Parametrul dat către funcția *exit* este valoarea returnată din *main*.

Funcția *exit* standard este scrisa în C. Aceasta apelează o funcție de mici dimensiuni scrisă în asamblare, numita *_exit* ce execută următoarele operații:

- apelează funcțiile ce trebuiesc apelate la închiderea aplicației. Aici intră destructorii C++ pentru variabile statice și globale și funcții înregistrate cu funcția standard *atexit*.
- Închide toate fișierele deschise
- Apelează *__exit*
- Când se ajunge la *__exit*, se oprește sistemul.

O aplicație își mai poate termina execuția apelând *abort* sau *_Exit*. Funcția *abort* doar apelează *__exit* pentru a opri sistemul, și nu realizează nicio curățare. Funcția *_Exit* este echivalentă cu funcția *abort*, cu excepția faptului că ia un argument pentru a transmite codul de ieșire.

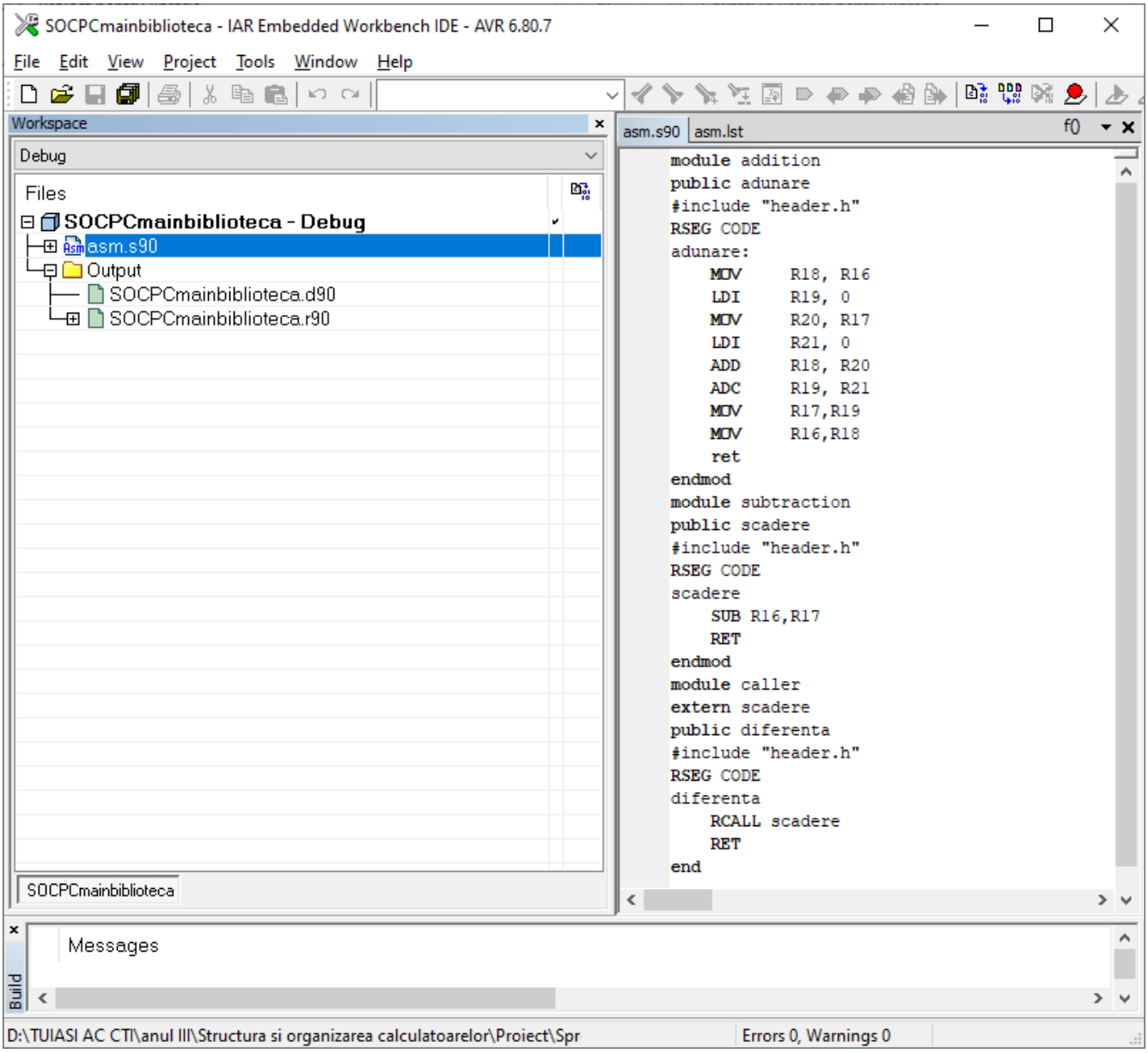
Dacă se dorește ca aplicația să facă mai multe lucruri la ieșire, cum ar fi resetarea sistemului, se poate scrie propria implementare a funcției *__exit(int)*.

Exemplu concret

a) Proiect cu funcția main scrisă în limbajul C care apelează mai multe funcții scrise în asamblare

Realizarea proiectului pentru a obține o bibliotecă cu funcțiile scrise în asamblare

În cadrul acestui proiect, toate funcțiile de asamblare fac parte din același fișier, *asm.s90*.

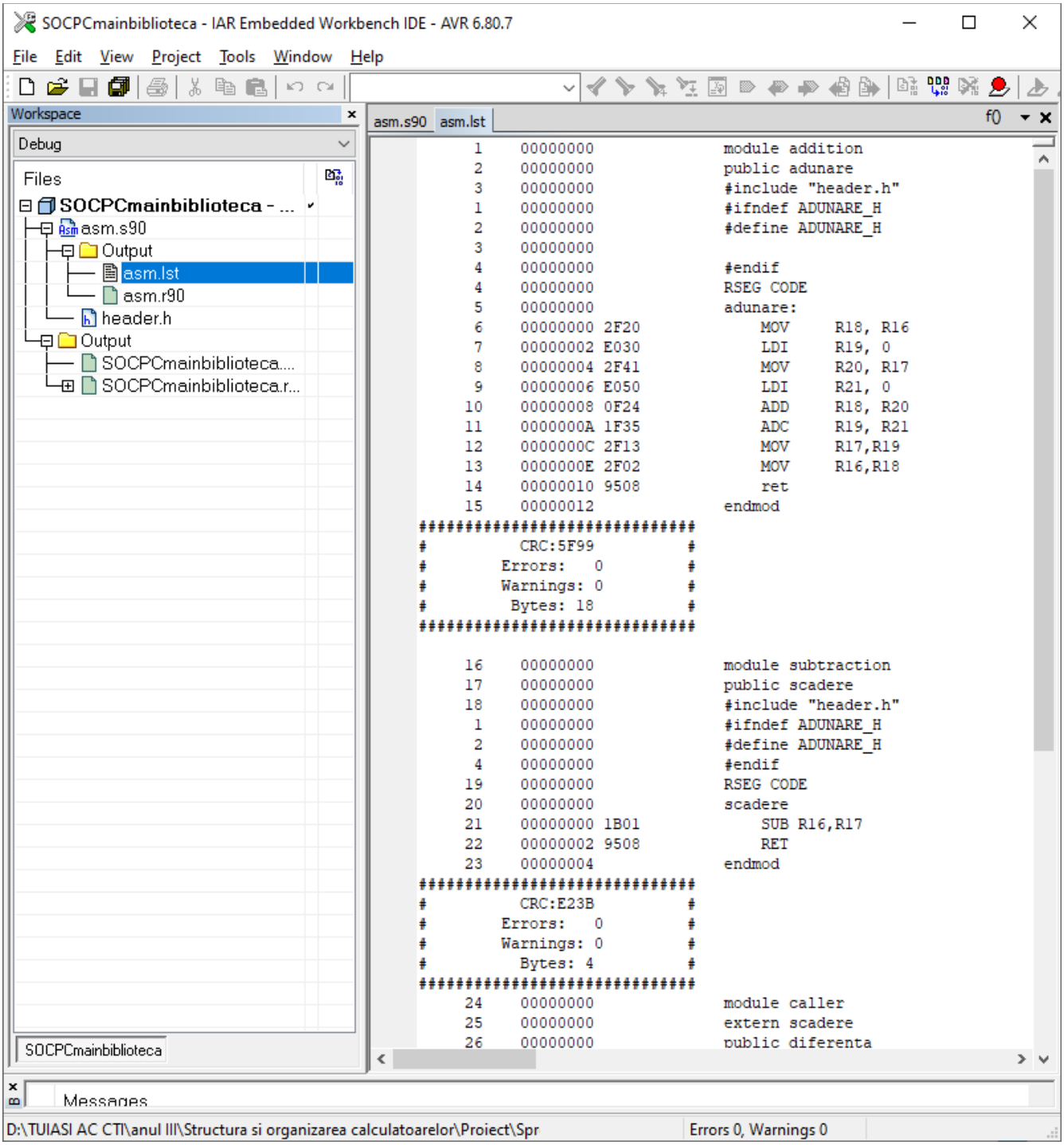


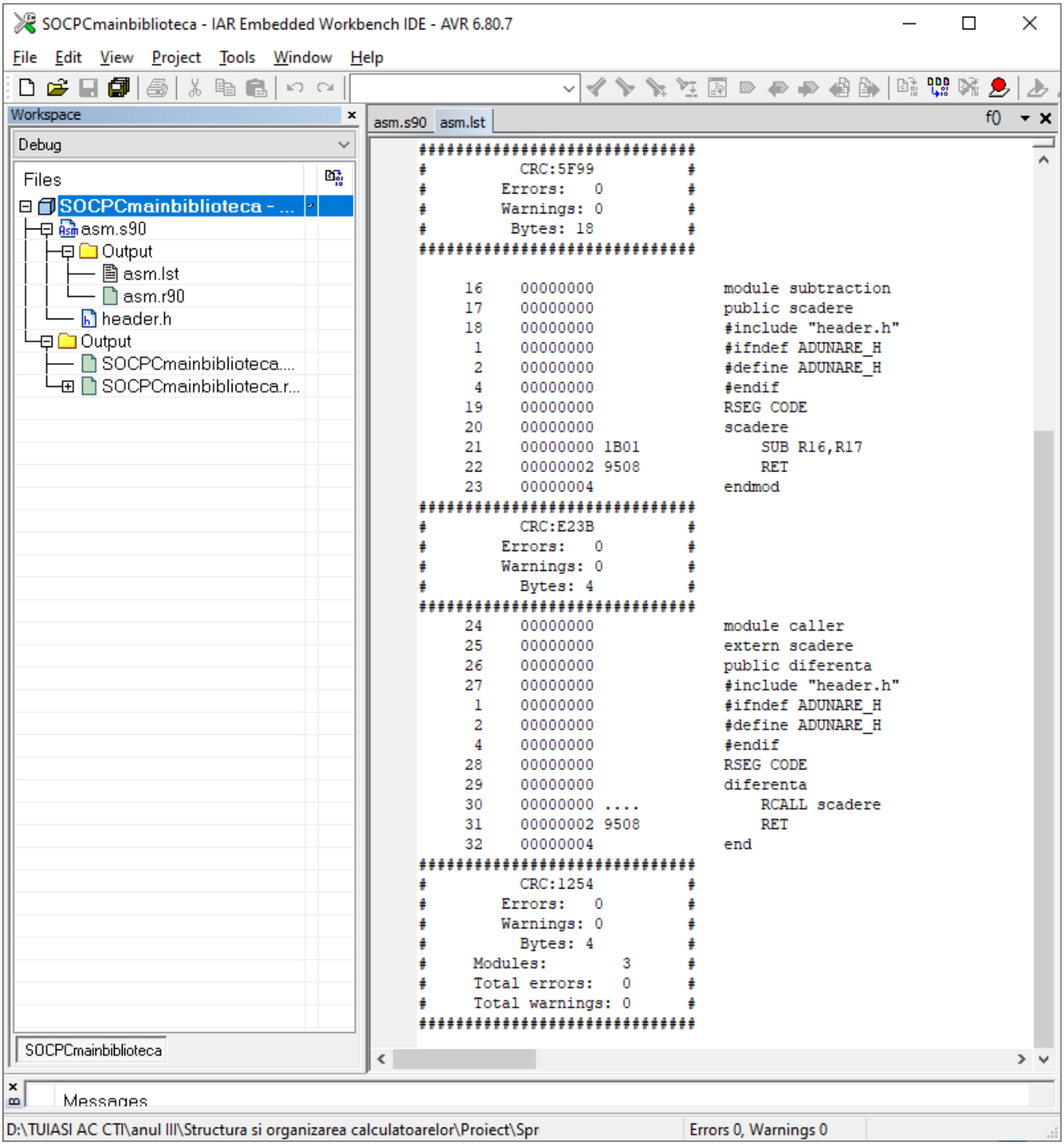
Acest fișier este disponibil în Anexa 1.

Funcțiile din cadrul acestui fișier sunt *adunare*, *scadere* și *diferenta*.

- Funcția *adunare* realizează adunarea a două char-uri și returnează rezultatul sub forma unui int.
- Funcția *scadere* realizează scăderea a două char-uri.
- Funcția *diferenta* apelează funcția scadere.

Conținutul principal al fișierului listă corespunzător *asm.s90* este următorul:



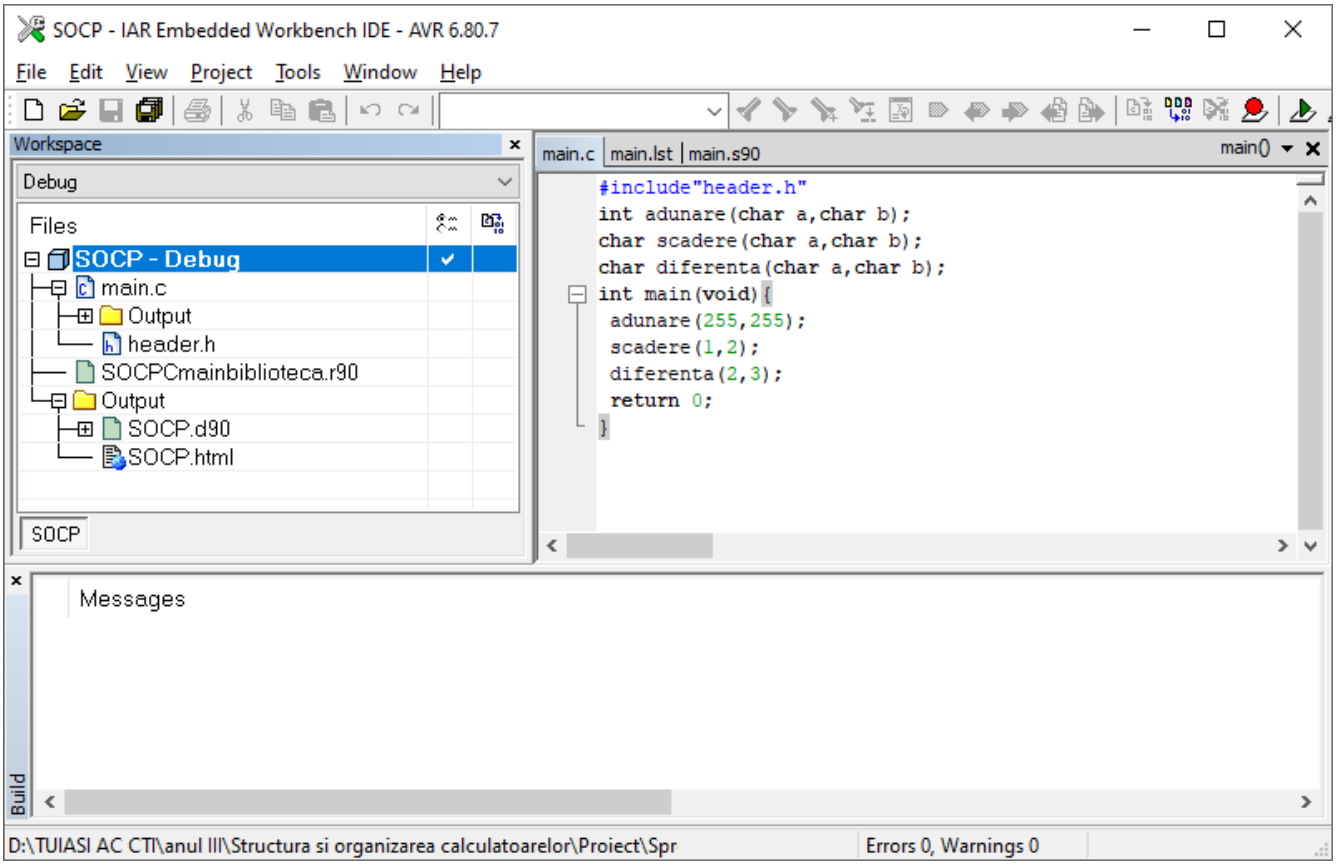


Fișierul este disponibil în varianta sa completă în Anexa 1.

În urma construirii proiectului, va rezulta un fișier de tip bibliotecă, numit *SOCPCmainbiblioteca.r90*.

Proiectul principal

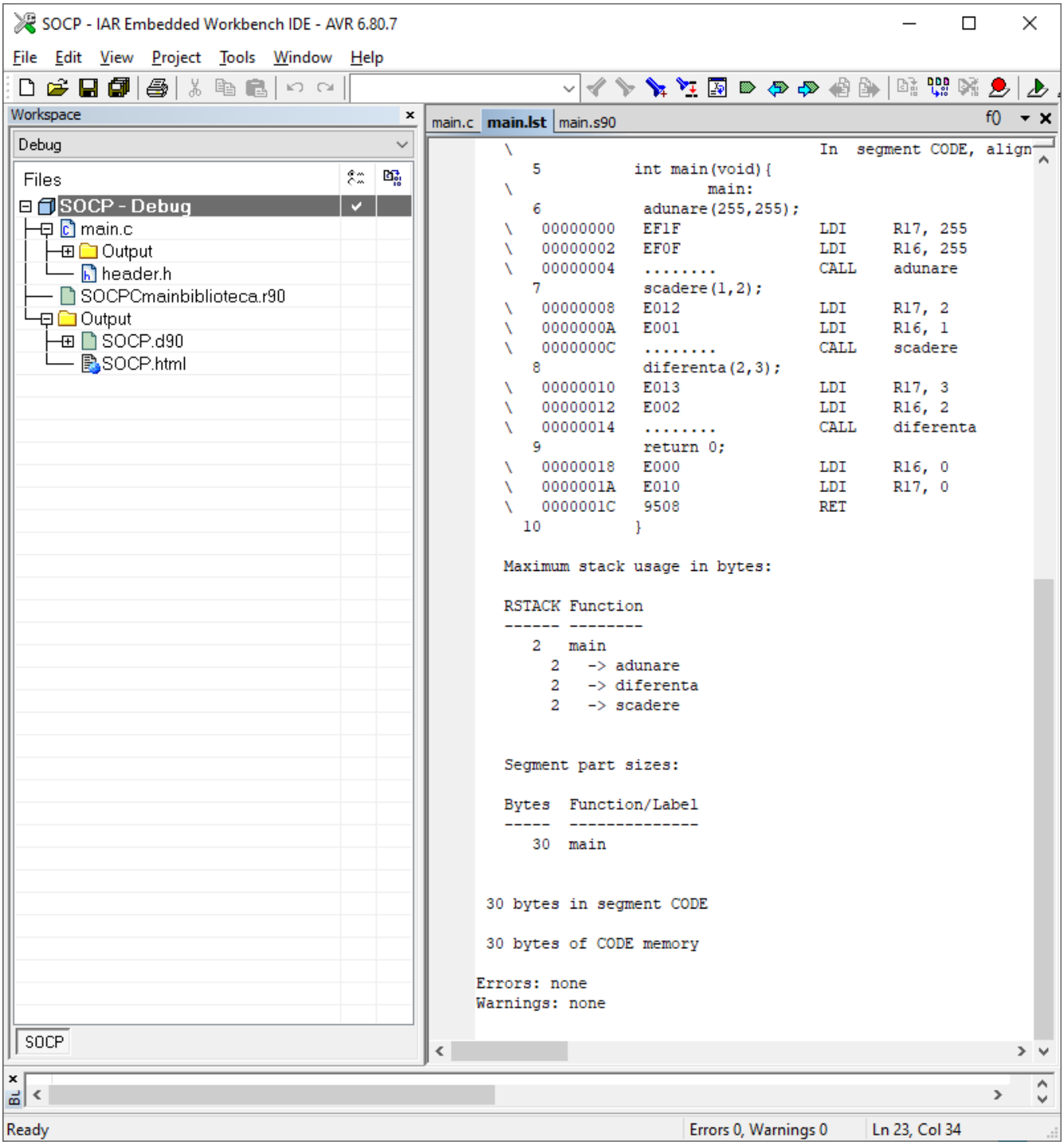
După ce s-a obținut fișierul de tip bibliotecă de mai sus, se va crea un nou proiect ce va conține funcția *main* în cadrul fișierului *main.c* și biblioteca *SOCPCmainbiblioteca.r90.*



Fișierul *main.c* este disponibil în Anexa 1.

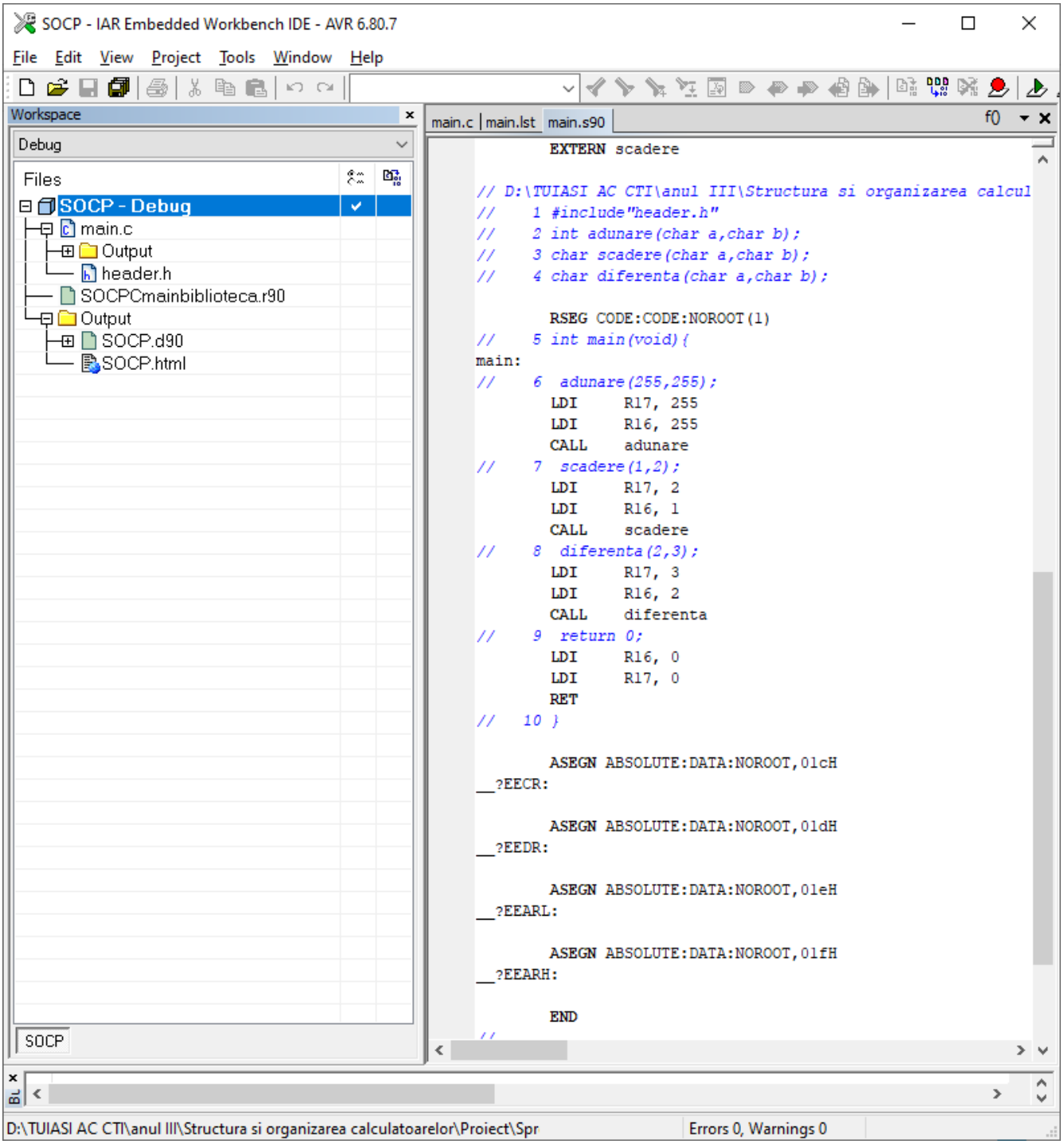
Funcția *main* apelează cele trei funcții descrise mai sus cu date de test.

Conținutul principal al fișierul listă corespunzător fișierului *main.c* este următorul:



Acest fișier este disponibil în varianta sa completă în Anexa 1.

Conținutul principal al fișierului în asamblare corespunzător fișierului *main.c* este următorul:



Acest fișier este disponibil în varianta sa completă în Anexa 1.

Tabelul de simboluri al principalelor funcții și etichete din *main*:

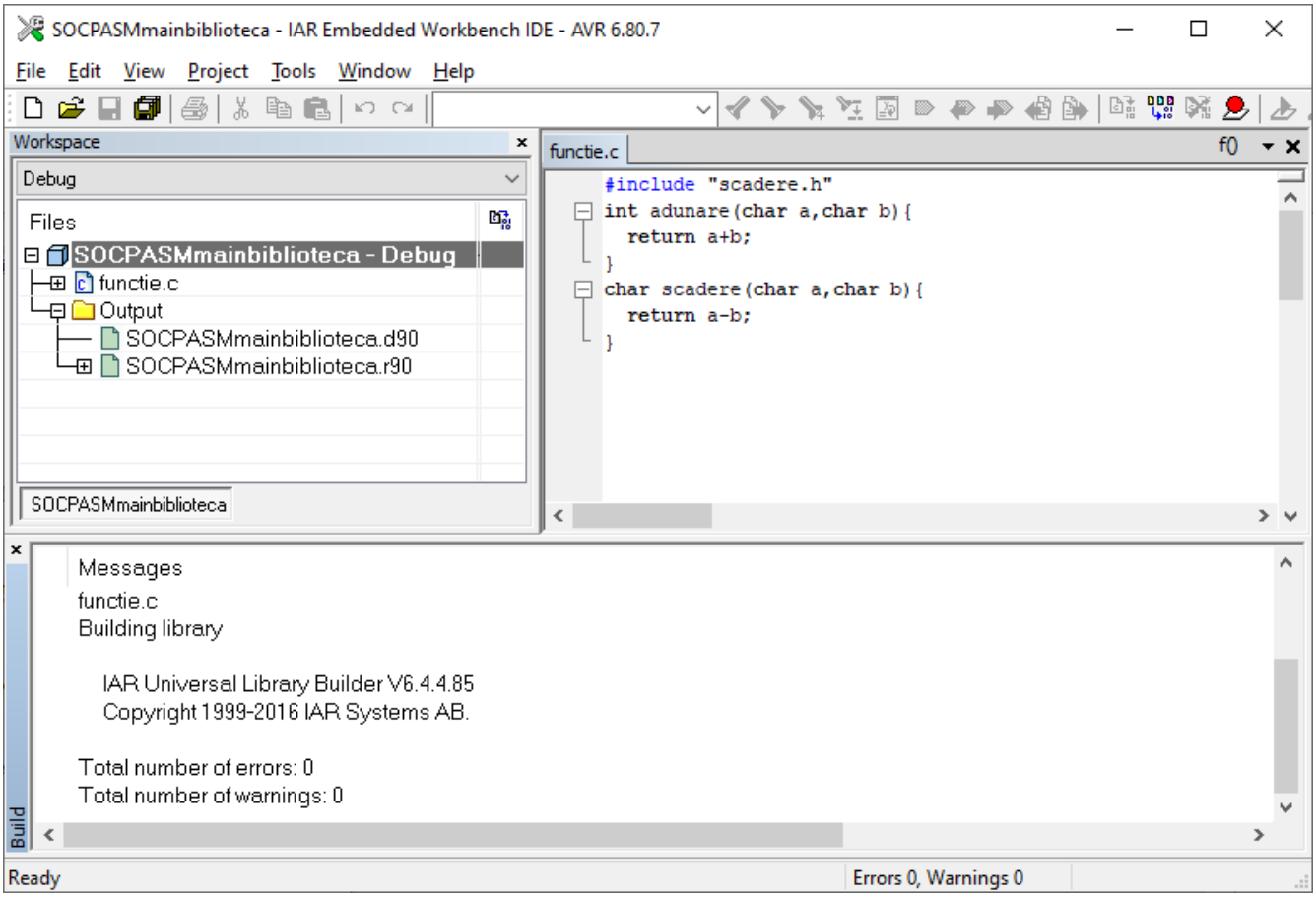
Symbol	Location	Full Name
?C_EXIT	CODE:0x00008E	?C_EXIT
?C_FUNCALL	CODE:0x00008C	?C_FUNCALL
?C_STARTUP	CODE:0x000092	?C_STARTUP
?RESET	CODE:0x000000	?RESET
?call_low_level_init	CODE:0x00009E	?call_low_level_init
?cstartup_call_main	CODE:0x0000A2	?cstartup_call_main
__program_start	CODE:0x000092	__program_start
_exit	CODE:0x00008C	_exit
adunare	CODE:0x000072	adunare
diferenta	CODE:0x000088	diferenta
exit	CODE:0x00008C	exit
main	CODE:0x000054	main
scadere	CODE:0x000084	scadere

În urma compilării și link-editării va rezulta un singur fișier executabil, numit *SOCP.d90*.

b) Proiect cu funcția main scrisă în asamblare care apelează funcții scrise în limbajul C

Realizarea proiectului pentru a obține o bibliotecă cu funcțiile scrise în limbajul C

În cadrul acestui proiect, toate funcțiile scrise în limbajul C fac parte din același fișier, *functie.c*.

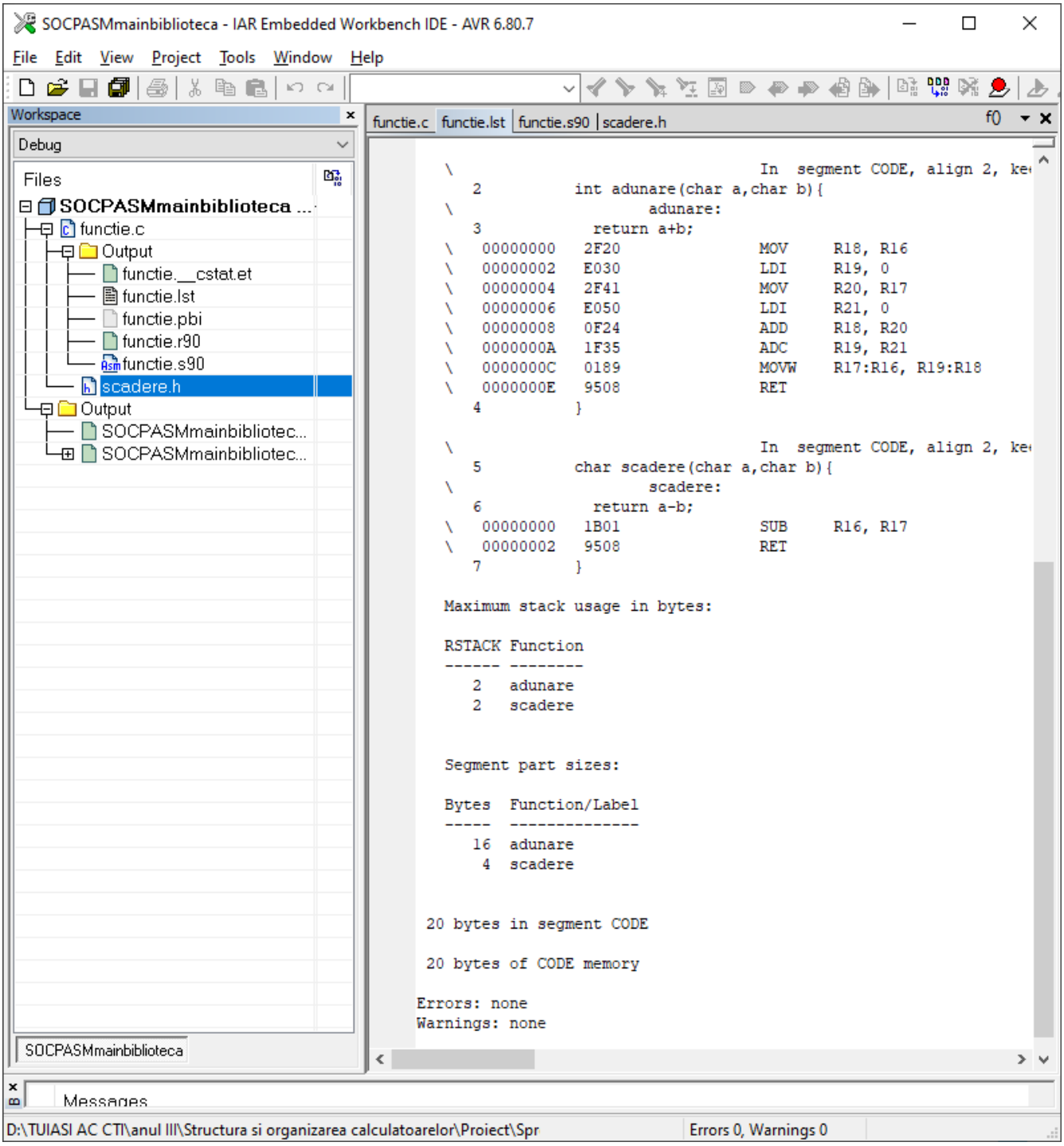


Fișierul *functie.c* este disponibil în varianta sa completă în Anexa 1.

Funcțiile din acest fișier sunt *adunare* și *scadere*.

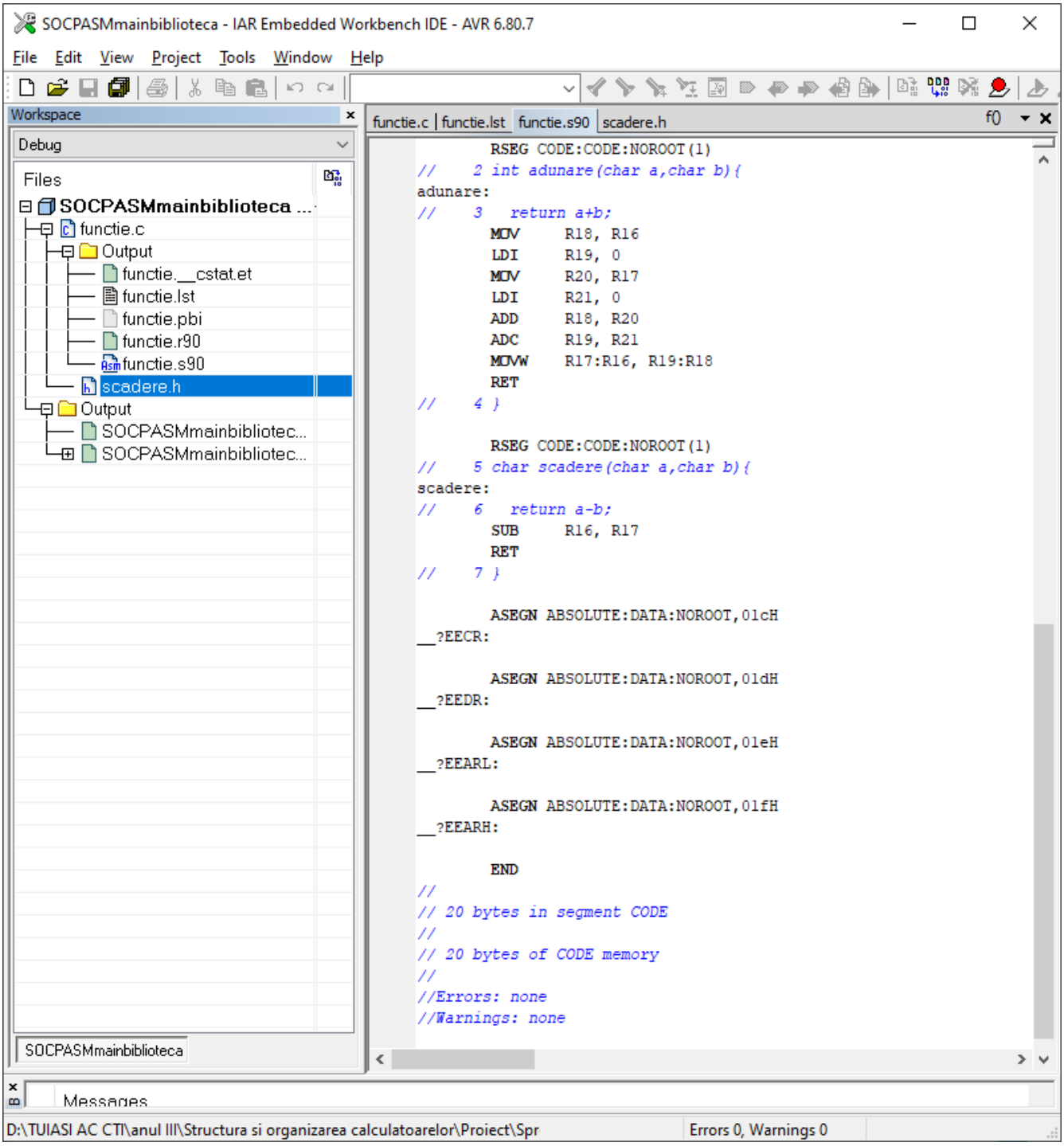
- Funcția *adunare* realizează adunarea între două char-uri și stochează rezultatul într-un int.
- Funcția *scadere* realizează scăderea celui de-al doilea parametru din primul.

Conținutul principal al fișierului listă corespunzător *functie.c* este:



Acest fișier este disponibil în varianta sa completă în Anexa 1.

Conținutul principal al fișierului de asamblare corespunzător *functie.c* este următorul:

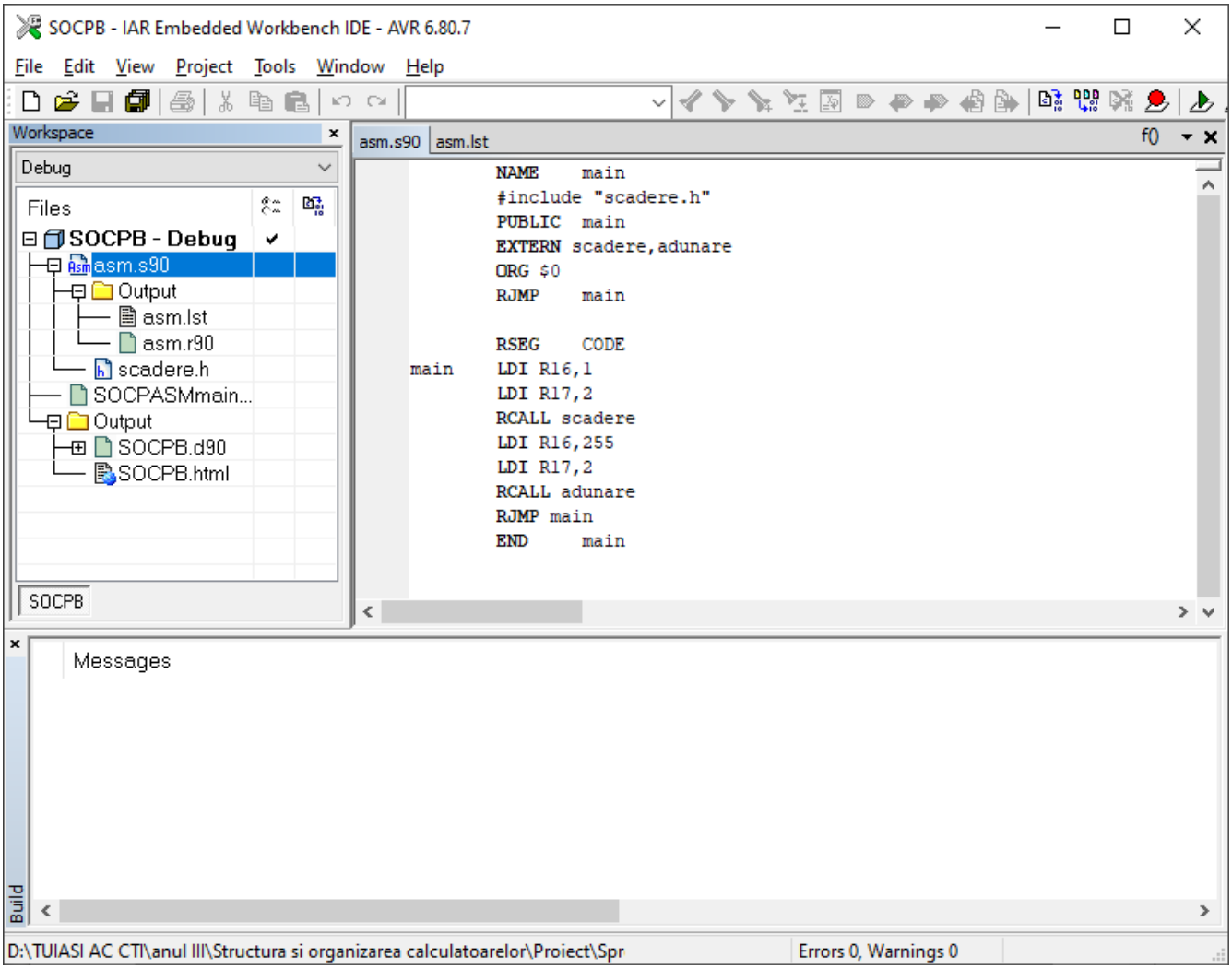


Acest fișier este disponibil în varianta sa completă în Anexa 1.

În urma construirii proiectului rezultă un fișier de tip bibliotecă, numit *SOCPASMmainbiblioteca.r90*.

Proiectul principal

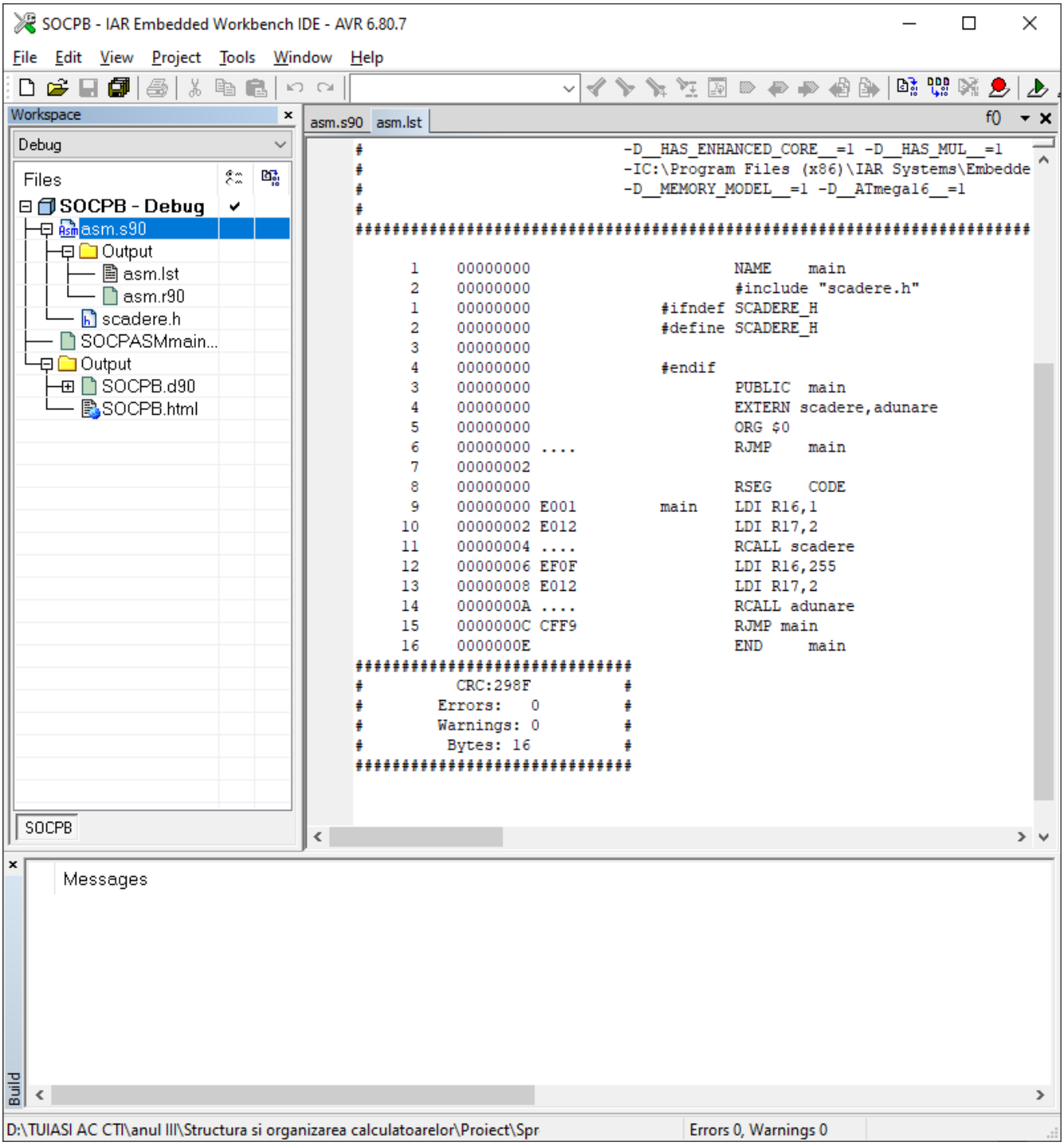
După s-a obținut fișierul de tip bibliotecă de mai sus, se creează un nou proiect ce va conține funcția *main* în cadrul fișierului *asm.s90* și biblioteca *SOCPASMmainbiblioteca.r90.*



Fișierul *asm.s90* este disponibil în Anexa 1.

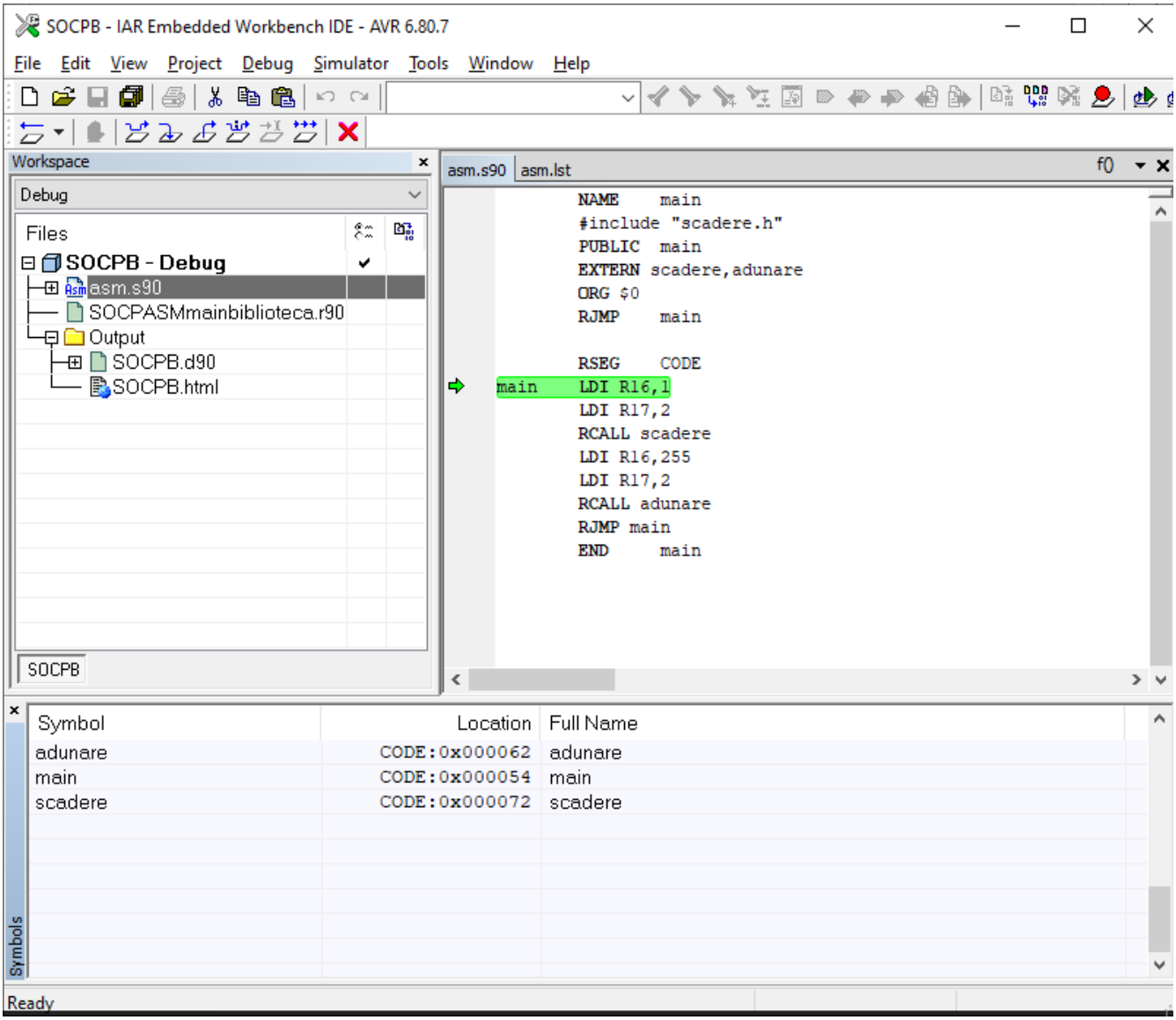
Funcția *main* apelează cele două funcții descrise mai sus cu date de test.

Conținutul principal al fișierului listă corespunzător lui *asm.s90* este următorul:



Acest fișier este disponibil în varianta sa completă în Anexa 1.

Tabelul de simboluri al principalelor funcții și etichete din *main*:



În urma compilării și link-editării rezultă un fișier executabil, numit *SOCPB.d90*.

Bibliografie

- AVR IAR Assembler Reference Guide for Atmel Corporation’s AVR Microcontroller
- Atmel AVR 8-bit Instruction Set
- IAR C/C++ Compiler User Guide
- IAR Linker and Library Tools Reference Guide
- IDE Project Management and Building Guide

Anexa 1

a) Proiect cu funcția main scrisă în limbajul C care apelează mai multe funcții scrise în asamblare

Realizarea proiectului pentru a obține o bibliotecă cu funcțiile scrise în asamblare

asm.s90

```
module addition
public adunare
#include "header.h"
RSEG CODE
adunare:
    MOV     R18, R16
    LDI     R19, 0
    MOV     R20, R17
    LDI     R21, 0
    ADD     R18, R20
    ADC     R19, R21
    MOV     R17,R19
```

```
    MOV     R16,R18
    ret
endmod
module subtraction
public scadere
#include "header.h"
RSEG CODE
scadere
    SUB R16,R17
    RET
endmod
module caller
extern scadere
public diferenta
#include "header.h"
RSEG CODE
diferenta
    RCALL scadere
    RET
end
```

asm.lst

```
#############################################################################
###
#                                                           #
#    IAR Assembler V6.80.1.1057/W32 for Atmel AVR 17/Jan/2023  16:55:30     #
```

```
#     Copyright 2016 IAR Systems AB.                                    #
#                                                                       #
#           Target option =  Relative jumps do not wrap                 #
#           Source file   =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\asm.s90#
#           List file     =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\List\asm.lst#
#           Object file   =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\Obj\asm.r90#
#           Command line  =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\asm.s90 #
#                     -v3                                               #
#                     -OD:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\Obj #
#                     -s+ -w+ -r -M<>                                   #
#                     -LD:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\List #
#                     -cAOM -i -B -t8 -u_enhancedCore                   #
#                     -D__HAS_ENHANCED_CORE__=1 -D__HAS_MUL__=1         #
#                     -IC:\Program Files (x86)\IAR Systems\Embedded Workbench 6.80.8\avr\
INC\ #
#                     -D__MEMORY_MODEL__=1 -D__ATmega16__=1             #
#                                                                       #
#########################################################################
###


    1   00000000            module addition
    2   00000000            public adunare
    3   00000000            #include "header.h"
```

```
    1   00000000            #ifndef ADUNARE_H
    2   00000000            #define ADUNARE_H
    3   00000000
    4   00000000            #endif
    4   00000000            RSEG CODE
    5   00000000            adunare:
    6   00000000 2F20           MOV     R18, R16
    7   00000002 E030           LDI     R19, 0
    8   00000004 2F41           MOV     R20, R17
    9   00000006 E050           LDI     R21, 0
   10   00000008 0F24           ADD     R18, R20
   11   0000000A 1F35           ADC     R19, R21
   12   0000000C 2F13           MOV     R17,R19
   13   0000000E 2F02           MOV     R16,R18
   14   00000010 9508           ret
   15   00000012            endmod
################################
#         CRC:5F99         #
#       Errors:   0        #
#       Warnings: 0        #
#        Bytes: 18         #
################################
#################################################################################
###
#                                                                              #
#    IAR Assembler V6.80.1.1057/W32 for Atmel AVR 17/Jan/2023  16:55:30      #
#    Copyright 2016 IAR Systems AB.                                            #
```

```
#                                                                              #
#       Target option =  Relative jumps do not wrap                            #
#       Source file   =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\asm.s90#
#       List file     =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\List\asm.lst#
#       Object file   =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\Obj\asm.r90#
#       Command line  =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\asm.s90 #
#                -v3                                                           #
#                -OD:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\Obj #
#                -s+ -w+ -r -M<>                                               #
#                -LD:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\List #
#                -cAOM -i -B -t8 -u_enhancedCore                               #
#                -D__HAS_ENHANCED_CORE__=1 -D__HAS_MUL__=1                     #
#                -IC:\Program Files (x86)\IAR Systems\Embedded Workbench 6.80.8\avr\
INC\ #
#                -D__MEMORY_MODEL__=1 -D__ATmega16__=1                         #
#                                                                              #
###############################################################################
###


    16   00000000            module subtraction
    17   00000000            public scadere
    18   00000000            #include "header.h"
     1   00000000           #ifndef ADUNARE_H
```

```
      2    00000000            #define ADUNARE_H
      4    00000000            #endif
     19    00000000             RSEG CODE
     20    00000000            scadere
     21    00000000 1B01             SUB R16,R17
     22    00000002 9508            RET
     23    00000004            endmod
################################
#         CRC:E23B         #
#       Errors:   0        #
#       Warnings: 0        #
#         Bytes: 4         #
#################################################################################
##################################
#                                                  #
#    IAR Assembler V6.80.1.1057/W32 for Atmel AVR 17/Jan/2023  16:55:30      #
#    Copyright 2016 IAR Systems AB.                                  #
#                                                  #
#          Target option =  Relative jumps do not wrap                  #
#          Source file   =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\asm.s90#
#          List file     =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\List\asm.lst#
#          Object file   =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\Obj\asm.r90#
#          Command line  =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\asm.s90 #
#                    -v3                                  #
```

```
#                    -OD:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\Obj #
#                    -s+ -w+ -r -M<>                          #
#                    -LD:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Proiect pentru librarie\Debug\List #
#                    -cAOM -i -B -t8 -u_enhancedCore          #
#                    -D__HAS_ENHANCED_CORE__=1 -D__HAS_MUL__=1       #
#                    -IC:\Program Files (x86)\IAR Systems\Embedded Workbench 6.80.8\avr\
INC\ #
#                    -D__MEMORY_MODEL__=1 -D__ATmega16__=1          #
#                                                             #
###############################################################################
###


    24   00000000            module caller
    25   00000000            extern scadere
    26   00000000            public diferenta
    27   00000000            #include "header.h"
     1   00000000           #ifndef ADUNARE_H
     2   00000000           #define ADUNARE_H
     4   00000000           #endif
    28   00000000            RSEG CODE
    29   00000000            diferenta
    30   00000000 ....           RCALL scadere
    31   00000002 9508           RET
    32   00000004            end
################################
#        CRC:1254        #
```

```
#       Errors:   0        #
#       Warnings: 0        #
#         Bytes: 4         #
#    Modules:       3      #
#    Total errors:  0      #
#    Total warnings: 0     #
################################
```

header.h

```
#ifndef ADUNARE_H
#define ADUNARE_H

#endif
```

Proiectul principal

main.c

```
#include"header.h"
int adunare(char a,char b);
char scadere(char a,char b);
char diferenta(char a,char b);
int main(void){
 adunare(255,255);
 scadere(1,2);
```

```
 diferenta(2,3);
 return 0;
}
```

main.lst

```
###############################################################################
###
#
# IAR C/C++ Compiler V6.80.7.1083 for Atmel AVR           17/Jan/2023  17:44:43
# Copyright 1996-2016 IAR Systems AB.
# Standalone license - IAR Embedded Workbench for Atmel AVR
#
#    Source file  =
#        D:\TUIASI AC CTI\anul III\Structura si organizarea
#        calculatoarelor\Proiect\Spre complet\Proiect cu main\main.c
#    Command line =
#        "D:\TUIASI AC CTI\anul III\Structura si organizarea
#        calculatoarelor\Proiect\Spre complet\Proiect cu main\main.c" --cpu=m16
#        -mt -o "D:\TUIASI AC CTI\anul III\Structura si organizarea
#        calculatoarelor\Proiect\Spre complet\Proiect cu main\Debug\Obj" -lC
#        "D:\TUIASI AC CTI\anul III\Structura si organizarea
#        calculatoarelor\Proiect\Spre complet\Proiect cu main\Debug\List" -lB
#        "D:\TUIASI AC CTI\anul III\Structura si organizarea
#        calculatoarelor\Proiect\Spre complet\Proiect cu main\Debug\List"
#        --initializers_in_flash --no_cse --no_inline --no_code_motion
#        --no_cross_call --no_clustering --no_tbaa --module_name=main --debug
```

```
#       -DENABLE_BIT_DEFINITIONS -e --eeprom_size 512 --clib --library_module
#       -On
#   Locale       =  Romanian_Romania.1250
#   List file    =
#       D:\TUIASI AC CTI\anul III\Structura si organizarea
#       calculatoarelor\Proiect\Spre complet\Proiect cu
#       main\Debug\List\main.lst
#   Object file  =
#       D:\TUIASI AC CTI\anul III\Structura si organizarea
#       calculatoarelor\Proiect\Spre complet\Proiect cu
#       main\Debug\Obj\main.r90
#
################################################################################
###
```

D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\Proiect\Spre complet\
Proiect cu main\main.c

```
    1       #include"header.h"
    2       int adunare(char a,char b);
    3       char scadere(char a,char b);
    4       char diferenta(char a,char b);


  \                         In  segment CODE, align 2, keep-with-next
    5       int main(void){
  \               main:
    6        adunare(255,255);
  \  00000000   EF1F            LDI     R17, 255
```

```
\  00000002   EF0F            LDI     R16, 255
\  00000004   .....          CALL    adunare
   7          scadere(1,2);
\  00000008   E012            LDI     R17, 2
\  0000000A   E001            LDI     R16, 1
\  0000000C   .....          CALL    scadere
   8          diferenta(2,3);
\  00000010   E013            LDI     R17, 3
\  00000012   E002            LDI     R16, 2
\  00000014   .....          CALL    diferenta
   9          return 0;
\  00000018   E000            LDI     R16, 0
\  0000001A   E010            LDI     R17, 0
\  0000001C   9508            RET
  10          }
```

Maximum stack usage in bytes:

RSTACK Function

------ --------

2 main

2 -> adunare

2 -> diferenta

2 -> scadere

Segment part sizes:

```
  Bytes  Function/Label
  -----  --------------
     30  main


 30 bytes in segment CODE

 30 bytes of CODE memory

Errors: none
Warnings: none
```

main.s90

```
////////////////////////////////////////////////////////////////////////////
//
// IAR C/C++ Compiler V6.80.7.1083 for Atmel AVR          17/Jan/2023  17:44:43
// Copyright 1996-2016 IAR Systems AB.
// Standalone license - IAR Embedded Workbench for Atmel AVR
//
//    Source file  =
//        D:\TUIASI AC CTI\anul III\Structura si organizarea
//        calculatoarelor\Proiect\Spre complet\Proiect cu main\main.c
//    Command line =
//        "D:\TUIASI AC CTI\anul III\Structura si organizarea
//        calculatoarelor\Proiect\Spre complet\Proiect cu main\main.c"
//        --cpu=m16 -mt -o "D:\TUIASI AC CTI\anul III\Structura si organizarea
```

```
//       calculatoarelor\Proiect\Spre complet\Proiect cu main\Debug\Obj" -lC
//       "D:\TUIASI AC CTI\anul III\Structura si organizarea
//       calculatoarelor\Proiect\Spre complet\Proiect cu main\Debug\List" -lB
//       "D:\TUIASI AC CTI\anul III\Structura si organizarea
//       calculatoarelor\Proiect\Spre complet\Proiect cu main\Debug\List"
//       --initializers_in_flash --no_cse --no_inline --no_code_motion
//       --no_cross_call --no_clustering --no_tbaa --module_name=main --debug
//       -DENABLE_BIT_DEFINITIONS -e --eeprom_size 512 --clib --library_module
//       -On
//    Locale      =  Romanian_Romania.1250
//    List file   =
//       D:\TUIASI AC CTI\anul III\Structura si organizarea
//       calculatoarelor\Proiect\Spre complet\Proiect cu
//       main\Debug\List\main.s90
//
////////////////////////////////////////////////////////////////////////////////

        MODULE main

        RSEG CSTACK:DATA:NOROOT(0)
        RSEG RSTACK:DATA:NOROOT(0)

        PUBWEAK __?EEARH
        PUBWEAK __?EEARL
        PUBWEAK __?EECR
        PUBWEAK __?EEDR
        PUBLIC main
```

```
        EXTERN adunare
        EXTERN diferenta
        EXTERN scadere

// D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\Proiect\Spre complet\
Proiect cu main\main.c
//    1 #include"header.h"
//    2 int adunare(char a,char b);
//    3 char scadere(char a,char b);
//    4 char diferenta(char a,char b);

        RSEG CODE:CODE:NOROOT(1)
//    5 int main(void){
main:
//    6  adunare(255,255);
        LDI     R17, 255
        LDI     R16, 255
        CALL    adunare
//    7  scadere(1,2);
        LDI     R17, 2
        LDI     R16, 1
        CALL    scadere
//    8  diferenta(2,3);
        LDI     R17, 3
        LDI     R16, 2
        CALL    diferenta
```

```
//    9  return 0;
        LDI     R16, 0
        LDI     R17, 0
        RET
//   10 }

        ASEGN ABSOLUTE:DATA:NOROOT,01cH
__?EECR:

        ASEGN ABSOLUTE:DATA:NOROOT,01dH
__?EEDR:

        ASEGN ABSOLUTE:DATA:NOROOT,01eH
__?EEARL:

        ASEGN ABSOLUTE:DATA:NOROOT,01fH
__?EEARH:

        END
//
// 30 bytes in segment CODE
//
// 30 bytes of CODE memory
//
//Errors: none
//Warnings: none
```

header.h

```
#ifndef ADUNARE_H
#define ADUNARE_H

#endif
```

b) Proiect cu funcția main scrisă în asamblare care apelează mai multe funcții scrise în limbajul C

Realizarea proiectului pentru a obține o bibliotecă cu funcțiile scrise în limbajul C

functie.c

```
#include "scadere.h"
int adunare(char a,char b){
  return a+b;
}
char scadere(char a,char b){
  return a-b;
}
```

functie.lst

```
#############################################################################
###
#
# IAR C/C++ Compiler V6.80.7.1083 for Atmel AVR           17/Jan/2023  17:58:20
# Copyright 1996-2016 IAR Systems AB.
```

```
# Standalone license - IAR Embedded Workbench for Atmel AVR
#
#   Source file  =
#       D:\TUIASI AC CTI\anul III\Structura si organizarea
#       calculatoarelor\Proiect\Spre complet\Bonus\Proiect pentru
#       biblioteca\functie.c
#   Command line =
#       "D:\TUIASI AC CTI\anul III\Structura si organizarea
#       calculatoarelor\Proiect\Spre complet\Bonus\Proiect pentru
#       biblioteca\functie.c" --cpu=m16 -mt -o "D:\TUIASI AC CTI\anul
#       III\Structura si organizarea calculatoarelor\Proiect\Spre
#       complet\Bonus\Proiect pentru biblioteca\Debug\Obj" -lC "D:\TUIASI AC
#       CTI\anul III\Structura si organizarea calculatoarelor\Proiect\Spre
#       complet\Bonus\Proiect pentru biblioteca\Debug\List" -lB "D:\TUIASI AC
#       CTI\anul III\Structura si organizarea calculatoarelor\Proiect\Spre
#       complet\Bonus\Proiect pentru biblioteca\Debug\List"
#       --initializers_in_flash --no_cse --no_inline --no_code_motion
#       --no_cross_call --no_clustering --no_tbaa --debug
#       -DENABLE_BIT_DEFINITIONS -e --eeprom_size 512 --library_module -On
#   Locale       =  Romanian_Romania.1250
#   List file    =
#       D:\TUIASI AC CTI\anul III\Structura si organizarea
#       calculatoarelor\Proiect\Spre complet\Bonus\Proiect pentru
#       biblioteca\Debug\List\functie.lst
#   Object file  =
#       D:\TUIASI AC CTI\anul III\Structura si organizarea
#       calculatoarelor\Proiect\Spre complet\Bonus\Proiect pentru
```

```
#       biblioteca\Debug\Obj\functie.r90
#
##############################################################################
###
```

D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\Proiect\Spre complet\
Bonus\Proiect pentru biblioteca\functie.c

```
    1       #include "scadere.h"


  \                         In  segment CODE, align 2, keep-with-next
    2       int adunare(char a,char b){
  \               adunare:
    3         return a+b;
  \   00000000   2F20            MOV     R18, R16
  \   00000002   E030            LDI     R19, 0
  \   00000004   2F41            MOV     R20, R17
  \   00000006   E050            LDI     R21, 0
  \   00000008   0F24            ADD     R18, R20
  \   0000000A   1F35            ADC     R19, R21
  \   0000000C   0189            MOVW    R17:R16, R19:R18
  \   0000000E   9508            RET
    4       }


  \                         In  segment CODE, align 2, keep-with-next
    5       char scadere(char a,char b){
  \               scadere:
    6         return a-b;
```

```
\  00000000  1B01          SUB     R16, R17
\  00000002  9508          RET
   7         }

  Maximum stack usage in bytes:

  RSTACK Function
  ------ --------
    2   adunare
    2   scadere


  Segment part sizes:

  Bytes  Function/Label
  -----  --------------
    16  adunare
     4  scadere


 20 bytes in segment CODE

 20 bytes of CODE memory

Errors: none
Warnings: none
```

functie.s90

```
////////////////////////////////////////////////////////////////////////////////
//
// IAR C/C++ Compiler V6.80.7.1083 for Atmel AVR          17/Jan/2023  17:58:20
// Copyright 1996-2016 IAR Systems AB.
// Standalone license - IAR Embedded Workbench for Atmel AVR
//
//    Source file  =
//        D:\TUIASI AC CTI\anul III\Structura si organizarea
//        calculatoarelor\Proiect\Spre complet\Bonus\Proiect pentru
//        biblioteca\functie.c
//    Command line =
//        "D:\TUIASI AC CTI\anul III\Structura si organizarea
//        calculatoarelor\Proiect\Spre complet\Bonus\Proiect pentru
//        biblioteca\functie.c" --cpu=m16 -mt -o "D:\TUIASI AC CTI\anul
//        III\Structura si organizarea calculatoarelor\Proiect\Spre
//        complet\Bonus\Proiect pentru biblioteca\Debug\Obj" -lC "D:\TUIASI AC
//        CTI\anul III\Structura si organizarea calculatoarelor\Proiect\Spre
//        complet\Bonus\Proiect pentru biblioteca\Debug\List" -lB "D:\TUIASI AC
//        CTI\anul III\Structura si organizarea calculatoarelor\Proiect\Spre
//        complet\Bonus\Proiect pentru biblioteca\Debug\List"
//        --initializers_in_flash --no_cse --no_inline --no_code_motion
//        --no_cross_call --no_clustering --no_tbaa --debug
//        -DENABLE_BIT_DEFINITIONS -e --eeprom_size 512 --library_module -On
//    Locale       =  Romanian_Romania.1250
//    List file    =
```

```
//        D:\TUIASI AC CTI\anul III\Structura si organizarea
//        calculatoarelor\Proiect\Spre complet\Bonus\Proiect pentru
//        biblioteca\Debug\List\functie.s90
//
////////////////////////////////////////////////////////////////////////////////

        MODULE functie

        RSEG CSTACK:DATA:NOROOT(0)
        RSEG RSTACK:DATA:NOROOT(0)

        PUBWEAK __?EEARH
        PUBWEAK __?EEARL
        PUBWEAK __?EECR
        PUBWEAK __?EEDR
        PUBLIC adunare
        PUBLIC scadere

// D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\Proiect\Spre complet\
Bonus\Proiect pentru biblioteca\functie.c
//    1 #include "scadere.h"

        RSEG CODE:CODE:NOROOT(1)
//    2 int adunare(char a,char b){
adunare:
//    3   return a+b;
        MOV     R18, R16
```

```
        LDI     R19, 0
        MOV     R20, R17
        LDI     R21, 0
        ADD     R18, R20
        ADC     R19, R21
        MOVW    R17:R16, R19:R18
        RET
//  4 }

        RSEG CODE:CODE:NOROOT(1)
//  5 char scadere(char a,char b){
scadere:
//  6   return a-b;
        SUB     R16, R17
        RET
//  7 }

        ASEGN ABSOLUTE:DATA:NOROOT,01cH
__?EECR:

        ASEGN ABSOLUTE:DATA:NOROOT,01dH
__?EEDR:

        ASEGN ABSOLUTE:DATA:NOROOT,01eH
__?EEARL:

        ASEGN ABSOLUTE:DATA:NOROOT,01fH
```

```
__?EEARH:

        END
//
// 20 bytes in segment CODE
//
// 20 bytes of CODE memory
//
//Errors: none
//Warnings: none
```

scadere.h

```
#ifndef SCADERE_H
#define SCADERE_H

#endif
```

Proiectul principal

asm.s90

```
        NAME    main
        #include "scadere.h"
        PUBLIC  main
        EXTERN scadere,adunare
```

```
        ORG $0
        RJMP    main

        RSEG    CODE
main    LDI R16,1
        LDI R17,2
        RCALL scadere
        LDI R16,255
        LDI R17,2
        RCALL adunare
        RJMP main
        END     main
```

asm.lst

```
#############################################################################
###
#                                                           #
#     IAR Assembler V6.80.1.1057/W32 for Atmel AVR 17/Jan/2023  11:35:31      #
#     Copyright 2016 IAR Systems AB.                                  #
#                                                           #
#           Target option =  Relative jumps do not wrap                   #
#           Source file   =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Bonus\Proiect cu main\asm.s90#
#           List file     =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Bonus\Proiect cu main\Debug\List\asm.lst#
#           Object file   =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Bonus\Proiect cu main\Debug\Obj\asm.r90#
```

```
#        Command line  =  D:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Bonus\Proiect cu main\asm.s90 #

#                -v3                                #

#                -OD:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Bonus\Proiect cu main\Debug\Obj #

#                -s+ -w+ -r -M<>                    #

#                -LD:\TUIASI AC CTI\anul III\Structura si organizarea calculatoarelor\
Proiect\Spre complet\Bonus\Proiect cu main\Debug\List #

#                -cAOM -i -B -t8 -u_enhancedCore    #

#                -D__HAS_ENHANCED_CORE__=1 -D__HAS_MUL__=1    #

#                -IC:\Program Files (x86)\IAR Systems\Embedded Workbench 6.80.8\avr\
INC\ #

#                -D__MEMORY_MODEL__=1 -D__ATmega16__=1        #

#                                                   #

##############################################################################
###


    1   00000000                NAME    main
    2   00000000                #include "scadere.h"
    1   00000000          #ifndef SCADERE_H
    2   00000000          #define SCADERE_H
    3   00000000
    4   00000000          #endif
    3   00000000                PUBLIC  main
    4   00000000                EXTERN scadere,adunare
    5   00000000                ORG $0
    6   00000000 ....           RJMP    main
    7   00000002
```

```
    8   00000000                RSEG   CODE
    9   00000000 E001       main   LDI R16,1
   10   00000002 E012              LDI R17,2
   11   00000004 ....              RCALL scadere
   12   00000006 EF0F              LDI R16,255
   13   00000008 E012              LDI R17,2
   14   0000000A ....              RCALL adunare
   15   0000000C CFF9              RJMP main
   16   0000000E                   END    main
################################
#        CRC:298F        #
#       Errors:   0      #
#       Warnings: 0      #
#        Bytes: 16       #
################################
```

scadere.h

```
#ifndef SCADERE_H
#define SCADERE_H

#endif
```