Documentație SOC – Proiect

Nume student: Pintilie Justinian

Grupă: 1307B

Profesor coordonator: ș. l. dr. A. Bârleanu

Descrierea proiectului:

Proiectul este realizat în mediul de lucru IAR Embedded Workspace IDE – AVR 7.10.1. Proiectul conține:

- Programul principal (main.c) scris în limbajul C din care sunt apelate 2 funcții în limbajul Assembly:
- add_func.s90
- factorial_func.s90

Codului folosit (fișierele, cu mostre de cod**) și explicarea lui:**

main.c:

În main.c apelez cele 2 funcții create cu ajutorul keyword-ului **extern.** Acesta preia funcțiile din celălalt limbaj (în cazul nostru, Assembly) și le folosește în limbajul din fișierul în care ne aflăm, adică C.

Pentru simplitate, am folosit tipul parametrilor **byte** (care este, de fapt, unsinged char) pentru ca aceștia să „încapă" pe un octet, mai precis pe un registru. Conform documentației C/C++ Compiler, primul parametru al unei funcții va putea fi memorat pe R16, al doilea pe R20 (desi in cazul proiectului meu este memorat in R17, conform codului din Dissasembly).

add_func.s90:

Functia aduna valorile celor doi parametrii dati prin registii R16 (primul parametru), respectiv R20 (al doilea parametru) si returneaza rezultatul adunarii in registrul R16.

Apoi, in main, rezultatul din R16 este mutat (MOV) in R24.

factorial_func.s90:

Această funcție calculează factorialul unui număr unsigned char dat ca parametru. Parametrul n pentru care se va calcula factorialul este încărcat în registrul R16, apoi, recursiv, se calculează factorialul. În registrul R17 va fi produsul final, inițializat cu 1. Dupa eticheta *loop_start* observăm instrucțiunea **CPSE** care compară valorea registrilor R16 si R1, adică numarul ramas in R16 dupa decrementare (facuta ulterior) si 0. Cand R16 va deveni 0, in R17 va fi fost memorat rezultatul factorialului, care se va returna (in registrul R16, in final) in functie.

Eticheta *loop_continue* se va apela atunci cand R16 nu va fi 0, adica in toate cazurile recursive. Se va face, apoi, inmultirea in R17 si decrementarea lui R16.

cstartup.s90:

Acest fisier are un rol fundamental. Este un fisier luat din Help-ul IAR-ului: Embedded Workspace 6.0 > src > lib >cstartup.s90 De aici incepe codul, in el se fac initializarile:

- Segmentului de cod, cu un flag (NOROOT): RSEG CODE:CODE:NOROOT(1)
- Initializarea stivei de date (CSTACK) si a adreselor de returnare din stiva (RTACK):

```
RSTACK:
?SETUP_STACK:
   ;; Return address stack (RSTACK)
   LDI R16,LOW(SFE(RSTACK)-1)
   OUT 0x3D,R16
#if A90_POINTER_REG_SIZE > 1
   LDI R16,HIGH(SFE(RSTACK)-1)
   OUT 0x3E,R16
#endif

CSTACK:
   LDI Y0,LOW(SFE(CSTACK))
#if A90_POINTER_REG_SIZE > 1
#if MEMORY_MODEL == TINY_MEMORY_MODEL
   LDI Y1,0
#else
   LDI Y1,HIGH(SFE(CSTACK))
#endif
#if A90_POINTER_REG_SIZE > 2
   LDI Z0,HWRD(SFB(CSTACK))
   OUT    RAMPY,Z0
#endif
#endif
```

- inițializări ale altor segmente

De asemenea, din acest fișier se apelează main-ul. (**RCALL main**)

Acest fișier este un „Start universal” și se mai pot apela mai multe, precum constructorii unor obiecte globale, dar nu este cazul în cazul proiectului de față.

Instructiuni folosite in main si functii si explicarea lor:

MODULE = numele modulului folosit, asa fiind grupate functiile in IAR, in module

PUBLIC = arata ca modulele, variabilele si tot ce are acest keyword in fata vor fi publice si vor putea fi “vazute” de main, care e in C, dar si de catre celelalte module/fisiere, din Assembly.

RSEG = registrul segment folosit

CODE: CODE = primul e numele segmentului, al doilea e chiar tipul segmentului (aici coincid)

add_func: label de unde incepe functia.

rjmp = sare la bucata de cod furnizata ulterior (label-ul dat ca parametru)

mul = inmultire

dec = decrementare cu 1

cpse r16, r1 = Compare Skip if Equal: verifica daca r16 este 0 (valoarea din R1, intial 0). Daca este egal, ii da skip (sare peste) instructiunea imediat urmatoare, altfel o executa.

Setări folosite pentru proiect:

General Options:

-> Target: Atmega16 (deoarece aceasta este procesorul folosit)

-> Library Configuration: CLIB (deoarece aceasta e libraria folosita pentru C: "Use the legacy C runtime library.")

-> System: Enable bit definitions in I/O files. Data stack (CSTACK): 0x20 (implicit), RSTACK: 16 (implicit).

C/C++ Compiler:

-> Optimizations: None

-> List: (toate)

Linker:

-> Extra Options: Use command line options:

-Z(DATA)DATA=_..X_EXT_ROM_BASE:+4

Această comandă arata ca in segmentul de date (DATA), la adresa de baza a extensiei ROM va fi un offset de +4, adică adresa finală pentru secțiunea de date va fi adresa de bază a extensiei ROM plus 4 octeți. Adica se face o aliniere a segmentului de date la 32 de biti (4 octeti).

-> Debugger: Setup: Simulator (nu va fi legat de JTAG, ci doar simulam).

Bibliografie:

1. IAR C/C++ Compiler User Guide for Atmel® Corporation's AVR Microcontroller Family
2. AVR IAR Assembler Reference Guide for Atmel Corporation's AVR Microcontroller
3. IAR Linker and Library Tools Reference Guide

4. AVR Microcontrollers AVR Instruction Set Manual