Univerza v Mariboru,

Fakulteta za elektrotehniko, računalništvo in informatiko

INŠTITUT ZA ROBOTIKO

Delo z razvojnim orodjem eZ430-F2013

Navodila za delo

Maribor, september, 2007

Avtor: Miran Rodič

Podatki o avtorju

Avtor: dr. Miran Rodič, dipl. inž. el.

Naslov: Inštitut za robotiko
Fakulteta za elektrotehniko, računalništvo in informatiko
Univerza v Mariboru
Smetanova ulica 17, SI-2000 Maribor, Slovenija, EU

Tel.: (+386 2) 220 7308

Fax: (+386 2) 220 7315

e-mail: miran.rodic@uni-mb.si

www: http://www.ro.feri.uni-mb.si/~miranro/

Kazalo

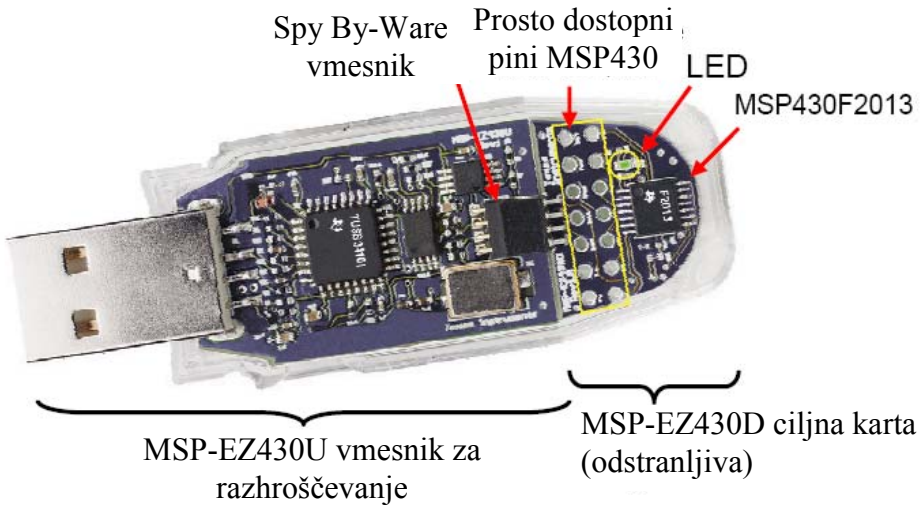
1. Uvod

Modul eZ430-F2013 je namenjen uporabi za razvoj aplikacij, lahko pa ga uporabimo tudi v pedagoške namene. Zgrajen je z uporabo mikrokrmilnika MSP430F2013.

V nadaljevanju bo opisan modul, delo s programskim orodjem in ukazi MSP430F2013 v zbirnem jeziku.

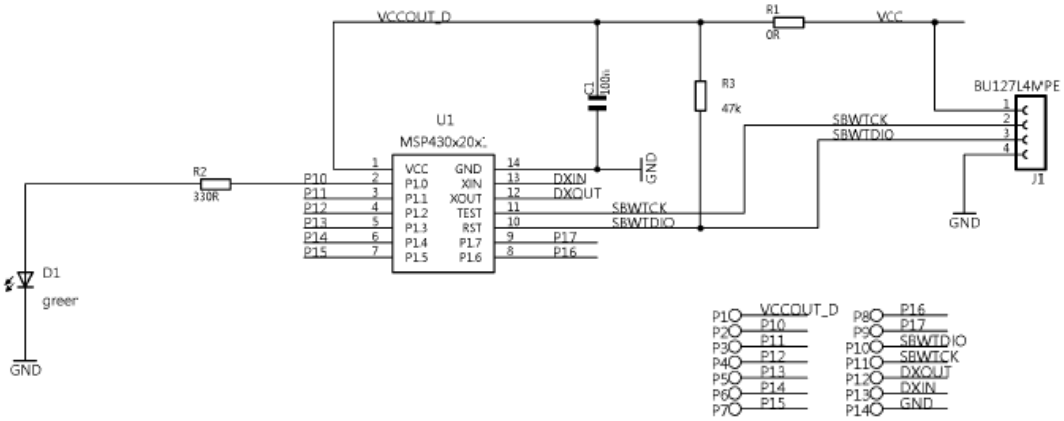
2. Opis strojne opreme

Orodje eZ430-F2013 je na videz podobno USB pomnilniku, vendar ga v ta namen ni mogoče uporabiti. Vsebuje vmesnik za razhroščevanje in ciljni sistem.



Slika 1: Razvojno orodje

Ciljni sistem lahko deluje samostojno, seveda pa je potrebno v tem primeru zagotoviti napajanje.



Slika 2: Ciljna karta MSP-EZ430D

3. Delo s programskim orodjem

Za delo z razvojnim okoljem eZ430-F2013 uporabljamo programsko orodje IAR Embedded Workbench IDE za mikrokrmilnike družine 430 proizvajalca Texas Instruments. Orodje je prosto dostopno na internetu, omejena verzija je na voljo brezplačno. Uporabimo ga lahko v okolju Windows (2000 in XP).
Ta dokument je napisan za verzijo 3.41. Ikone in oblika zaslona so lahko v novejših verzijah nekoliko drugačni.
Poleg klasičnih elementov, ki jih pričakujemo od IDE orodja:

- urejevalnik (editor),
- project manager,
- prevajalnik (compiler),
- povezovalnik (linker),
- razhroščevalnik (debugger) in
- pomoč (help),

imamo na voljo še nekaj dodatkov, ki nam olajšajo pisanje, prevajanje in preizkušanje programske opreme. Najpomembnejši so:

- kontekstno barvanje besedila (z barvami posameznih delov kode je prikazano, ali gre za komentar, spremenljivko, ...),
- čarovniki za hitro izdelavo programske opreme ... in
- orodja za iskanje in nadomeščanje (Find and replace facilities).

Bistvene dobre lastnosti orodja so:

- cena (prosto dostopno preko interneta, z uporabo ne kršimo avtorskih pravic) in
- preprosta uporaba.

Opisane bodo le neatere osnovne funkcije, ki jih potrebujemo za uporabo programskega orodja na predavanjih in vajah.
Delo s programskim orodjem bo opisano po korakih.

3.1 Instalacija programa

Program bo na vajah že instaliran, če ga želite uporabljati tudi doma, si ga naložite s priložene zgoščenke ali spletnih strani:

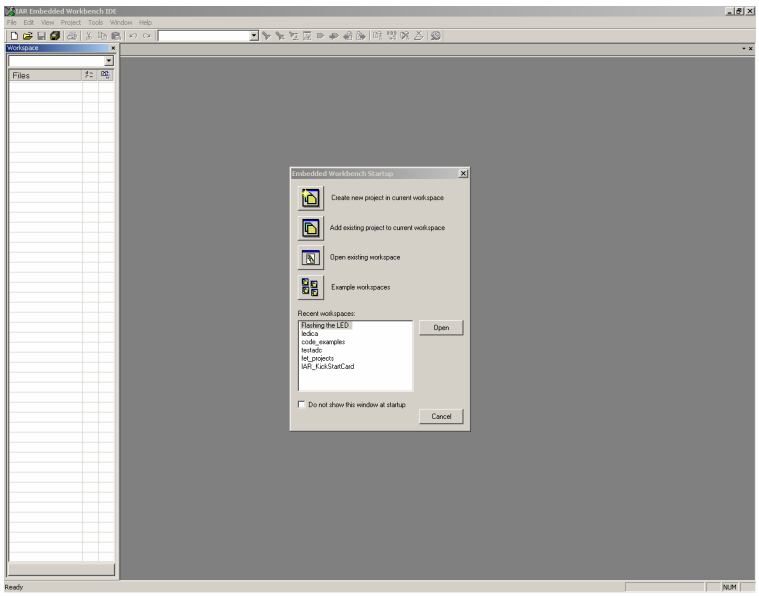
http://focus.ti.com/docs/toolsw/folders/print/iar-kickstart.html.

Ko program za instalacijo naložite na svoj računalnik, ga poženete in sledite navodilom.

3.2 Zagon programskega orodja

Razvojno okolje poženemo z izbiro ***Start ➔ Programs ➔ IAR Systems ➔ IAR Embedded Workbench Kickstart for MSP430 V3 ➔ IAR Embedded Workbench***.

Odpre se programsko okolje, izgled kaže Slika 3.



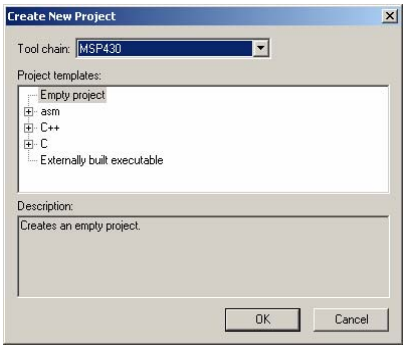
Slika 3: Programsko okolje *IAR Embedded Workbench*

3.3 Ustvarjanje novega projekta v programskem jeziku C

Nov projekt v programskem jeziku C ustvarimo tako, da v oknu ***Embedded Workbench Startup*** izberemo opcijo ***Create new project in current workspace*** s čemer poženemo čarovnika za ustvarjanje projektov.

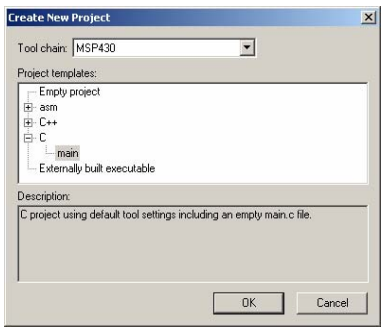
Slika 4: Ustvarjanje novega projekta v programskem jeziku C

Pokaže se okno, ki ga kaže Slika 5.



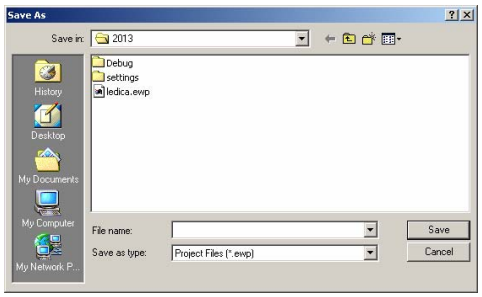
Slika 5: Uvodno okno *Create New Project*

Ker želimo ustvariti nov program v jeziku C izberemo opcijo C in kliknemo main (Slika 6).



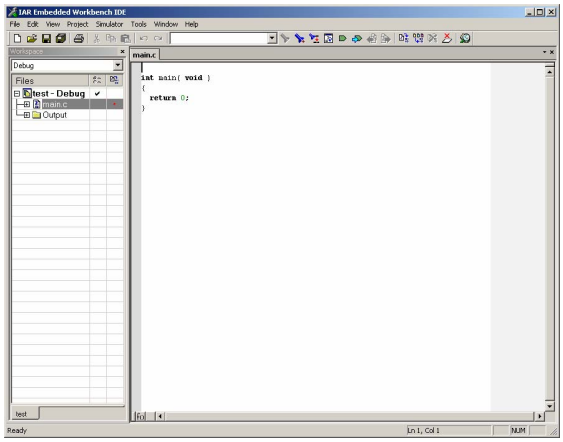
Slika 6: Izbira opcije C v oknu *Create New Project*

Kliknemo ***OK*** in odpre se okno za izbiro imena projekta in mesta, kamor ga bomo shranili.



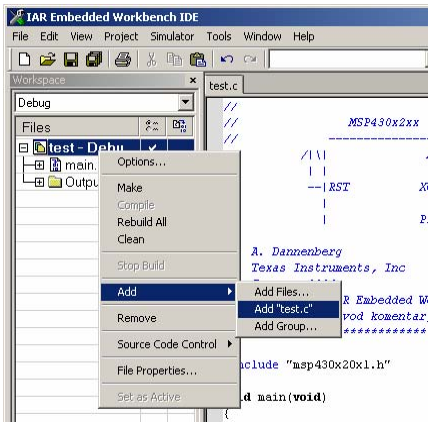
Slika 7: Izbira imena projekta in njegove lokacije na disku

Izberemo ime, npr ***test*** in kliknemo ***Save*** ter se tako pomaknemo na okno programa. Odpre se začetna datoteka ***main.c***.



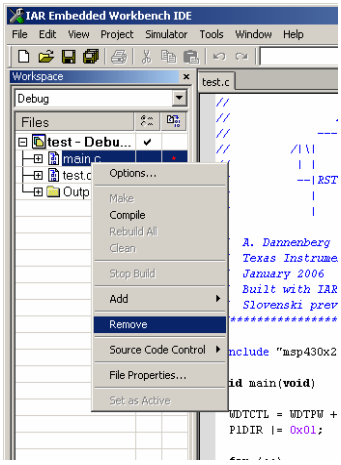
Slika 8: Okno *IAR Embedded Workbench IDE* po nastavitvi imena projekta

Zdaj lahko začnemo s pisanjem programa, lahko pa tudi naložimo drugo kdo, npr. ***test.c***, ki je priložen.



Slika 9: Dodajanje nove datoteke v projekt

V tem primeru je potrebno seveda odstraniti program ***main.c***, ki je bil ustvarjen samodejno. To naredimo tako, da z desno tipko na miški kliknemo ***main.c*** in izberemo opcijo ***Remove***.



Slika 10: Odstranjevanje datoteke *main.c* iz projekta

Odpre se okno, s katerim nam program da še eno možnost, da si premislimo.

Slika 11: Okno za potrditev odločitve o odstranitvi datoteke *main.c* iz projekta

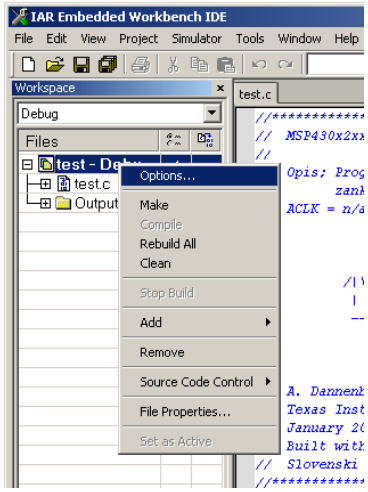
Kliknemo ***Yes*** in datoteke ***main.c*** ni več v projektu.

Vsebina datoteke ***test.c*** je:

```
//*******************************************************************************
//  MSP430x2xx Demo - Utripanje LED na pinu P1.0
//
//  Opis; Program prižiga in ugaša LED na P1.0 z uporabo xor znotraj programske
//        zanke.
//  ACLK = n/a, MCLK = SMCLK = default DCO
//
//                MSP430x2xx
//             -----------------
//         /|\|              XIN|-
//          | |                 |
//          --|RST          XOUT|-
//            |                 |
//            |             P1.0|-→LED
//
//  A. Dannenberg
//  Texas Instruments, Inc
//  January 2006
//  Built with IAR Embedded Workbench Version: 3.40A
//  Slovenski prevod komentarjev: Miran Rodic, september 2007
//*******************************************************************************

#include "msp430x20x1.h"

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;             // Zaustavi watchdog timer
  P1DIR |= 0x01;                        // Uporabi pin P1.0 kot izhod
```

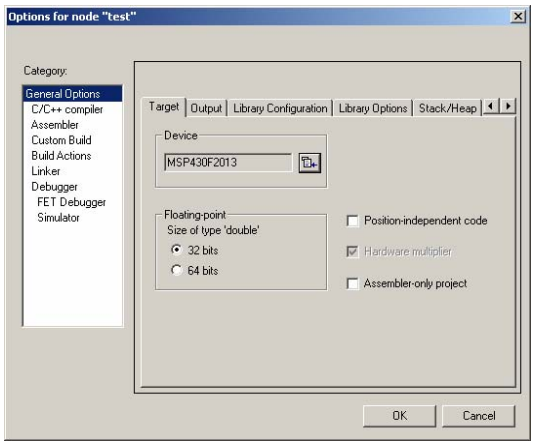
```
  for (;;)
  {
    volatile unsigned int i;            // volatile, da preprecimo optimizacijo

    P1OUT ^= 0x01;                      // Preklapljaj P1.0 z uporabo
                                        // ekskluzivnega ALI

    i = 30000;                          // Programska zakasnitev
    do i--;
    while (i != 0);
  }
}
```

Zdaj je potrebno nastaviti parametre delovanja orodja in razhroščevalnika. Z desno tipko miške kliknemo na projekt in izberemo opcijo ***Options...***



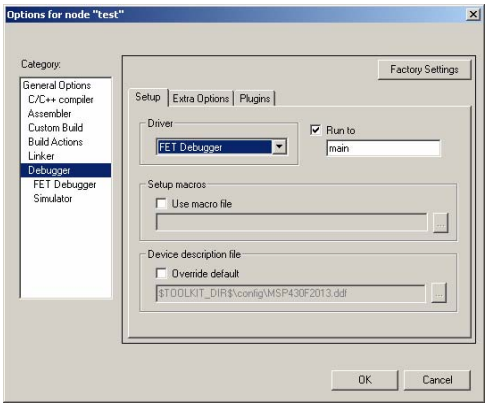
Slika 12: Nastavljenje opcij projekta

Najprej v ***General options*** v zavihku ***Target*** izberemo ciljni procesor, v našem primeru je to ***MSP430F2013***.



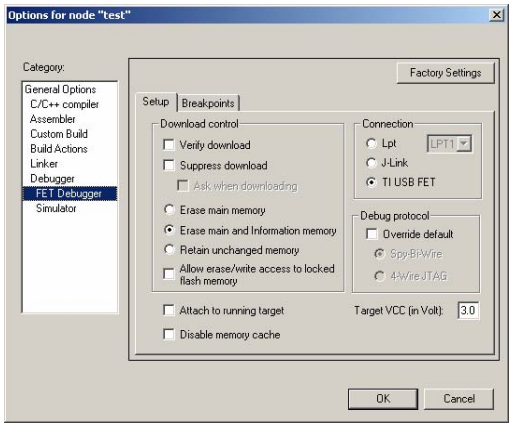
Slika 13: Izbira procesorja

Naslednji korak je nastavitev lastnosti razhoščevalnika. Pod opcijo ***Debugger*** v zavihku ***Setup*** nastavimo ***Driver*** na ***FET Debugger***.



Slika 14: Uporaba stroje opreme – *FET Debugger*

V naslednjem koraku pod opcijo ***Debugger ➔ FET Debugger*** v zavihku ***Setup*** nastavimo način priklopa naprave, pod ***Connection*** izberemo ***TI USB FET***.

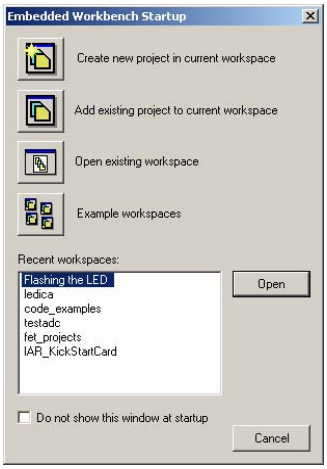


Slika 15: Izbira načina priklopa opreme

Zdaj kliknemo ***OK***. S tem je program pripravljen za prevajanje in razhroščevanje.

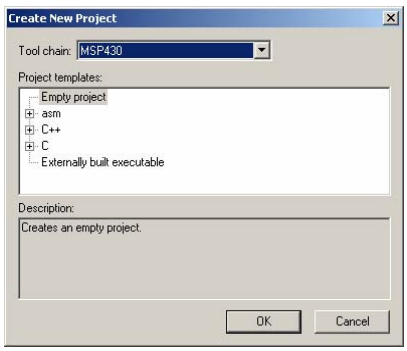
3.4 Ustvarjanje novega projekta v zbirnem jeziku

Nov projekt v zbirnem jeziku ustvarimo podobno kot projekt v programskem jeziku C. V oknu ***Embedded Workbench Startup*** izberemo opcijo ***Create new project in current workspace*** s čemer poženemo čarovnika za ustvarjanje projektov.



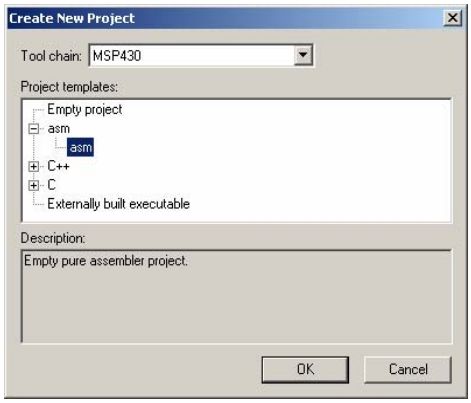
Slika 16: Ustvarjanje novega projekta v zbirnem jeziku

Pokaže se okno, ki ga kaže Slika 17.



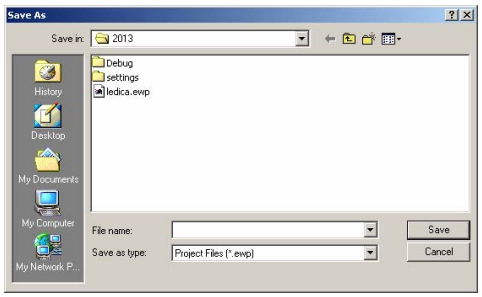
Slika 17: Uvodno okno *Create New Project*

Ker želimo ustvariti nov program v zbirnem jeziku, izberemo opcijo asm in kliknemo main (Slika 18).



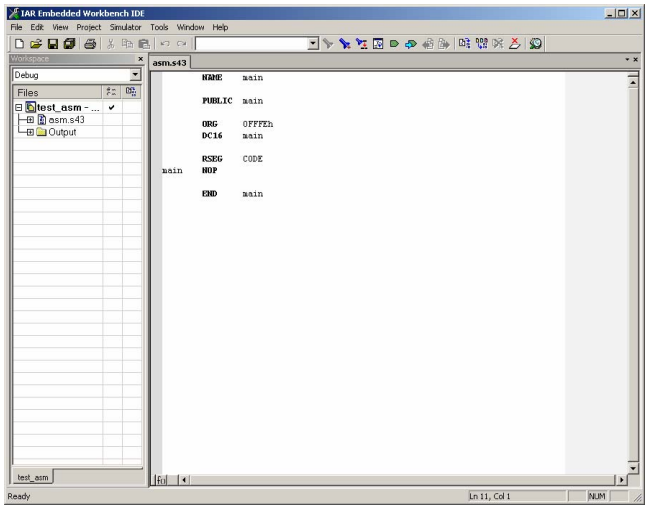
Slika 18: Izbira opcije *asm* v oknu *Create New Project*

Kliknemo ***OK*** in odpre se okno za izbiro imena projekta in mesta, kamor ga bomo shranili.



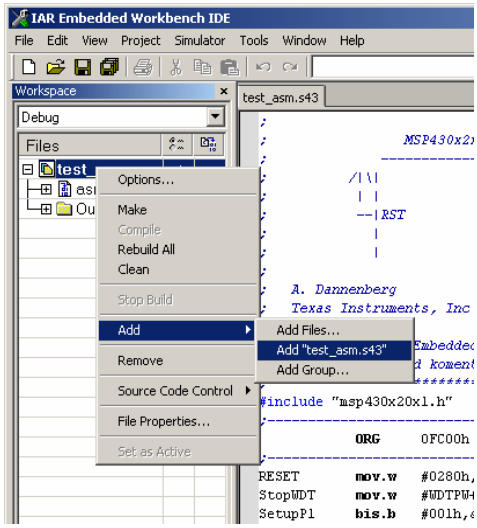
Slika 19: Izbira imena projekta in njegove lokacije na disku

Izberemo ime, npr ***test_asm*** in kliknemo ***Save*** ter se tako pomaknemo na okno programa. Odpre se začetna datoteka ***asm.s43***.



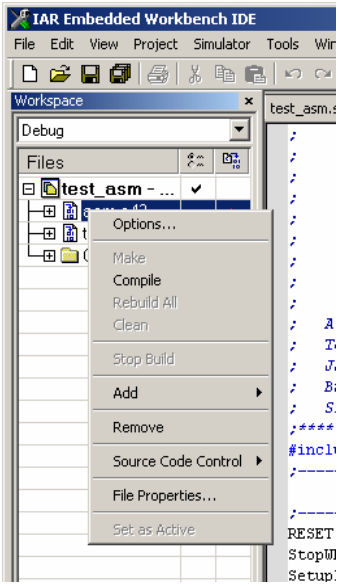
Slika 20: Okno *IAR Embedded Workbench IDE* po nastavitvi imena projekta

Zdaj lahko začnemo s pisanjem programa, lahko pa tudi naložimo drugo kdo, npr. ***test_asm.s43***, ki je priložen.



Slika 21: Dodajanje nove datoteke v projekt

V tem primeru je potrebno seveda odstraniti program ***asm.s43***, ki je bil ustvarjen samodejno. To naredimo tako, da z desno tipko na miški kliknemo ***asm.s43*** in izberemo opcijo ***Remove***.



Slika 22: Odstranjevanje datoteke *asm.s43* iz projekta

Odpre se okno, s katerim nam program da še eno možnost, da si premislimo.

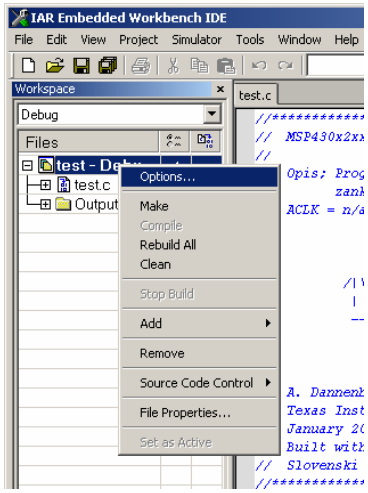
Slika 23: Okno za potrditev odločitve o odstranitvi datoteke *asm.s43* iz projekta

Kliknemo ***Yes*** in datoteke ***asm.s43*** ni več v projektu.

Vsebina datoteke ***test_asm.s43*** je:

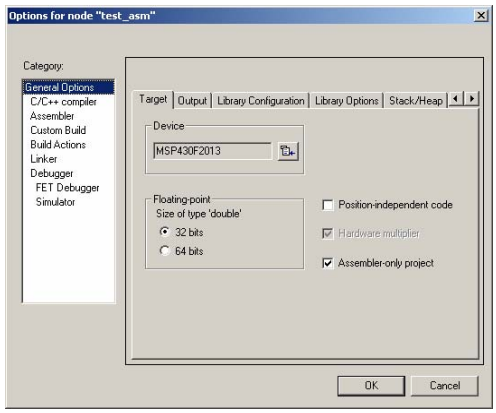
```
;*******************************************************************************
;   MSP430x1xx Demo - Utripanje LED na P1.0
;
;  Opis; Program prižiga in ugaša LED na P1.0 z uporabo xor znotraj programske
;        zanke.
;
;                MSP430x2xx
;             -----------------
;          /|\|              XIN|-
;           | |                 |
;           --|RST          XOUT|-
;             |                 |
;             |             P1.0|-→LED
;
;   A. Dannenberg
;   Texas Instruments, Inc
;   January 2006
;   Built with IAR Embedded Workbench Version: 3.40A
;   Slovenski prevod komentarjev: Miran Rodic, september 2007
;*******************************************************************************
#include "msp430x20x1.h"
;-------------------------------------------------------------------------------
            ORG     0FC00h                  ; Začetek programa (1K Flash)
;-------------------------------------------------------------------------------
RESET       mov.w   #0280h,SP               ; Postavi kazalec sklada
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Zaustavi watchdog timer
SetupP1     bis.b   #001h,&P1DIR            ; Uporabi pin P1.0 kot izhod
                                            ;
Mainloop    xor.b   #001h,&P1OUT            ; Preklapljaj P1.0 z uporabo
                                            ; ekskluzivnega ALI
Wait        mov.w   #050000,R15             ; Vrednost zakasnitve v R15
L1          dec.w   R15                     ; Zmanjsaj R15
            jnz     L1                      ; Je zakasnitev koncana?
            jmp     Mainloop                ; Ponovi
                                            ;
;-------------------------------------------------------------------------------
;           Prekinitveni vektorji
;-------------------------------------------------------------------------------
            ORG     0FFFEh                  ; MSP430 RESET Vektor
            DW      RESET                   ;
            END
```

Zdaj je potrebno nastaviti parametre delovanja orodja in razhroščevalnika. Z desno tipko miške kliknemo na projekt in izberemo opcijo ***Options...***



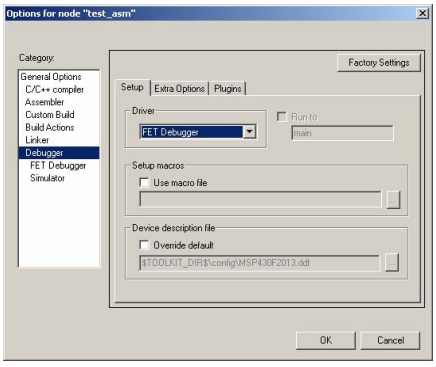
Slika 24: Nastavljenje opcij projekta

Najprej v ***General options*** v zavihku ***Target*** izberemo ciljni procesor, v našem primeru je to ***MSP430F2013***.



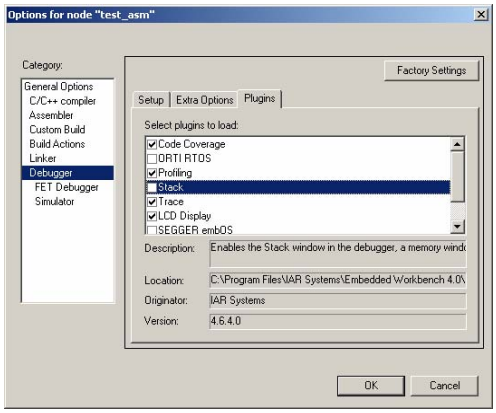
Slika 25: Izbira procesorja

Naslednji korak je nastavitev lastnosti razhoščevalnika. Pod opcijo ***Debugger*** v zavihku ***Setup*** nastavimo ***Driver*** na ***FET Debugger***.



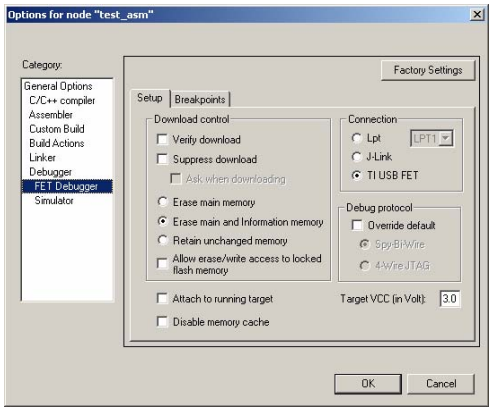
Slika 26: Uporaba stroje opreme – *FET Debugger*

Pod opcijo ***Debugger*** v zavihku ***Plugins*** izklopimo opcijo ***Stack.***



Slika 27: Uporaba stroje opreme – odstranitev opazovanja sklada iz razhroščevalnika

V naslednjem koraku pod opcijo ***Debugger ➔ FET Debugger*** nastavimo način priklopa naprave, v zavihku ***Setup*** pod ***Connection*** izberemo ***TI USB FET***.

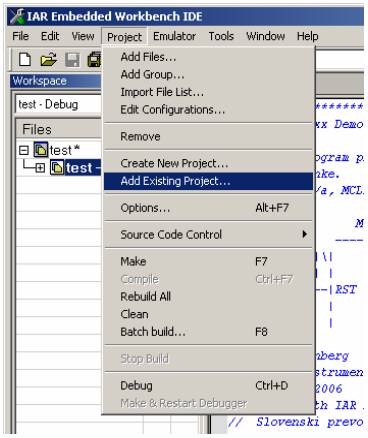


Slika 28: Izbira načina priklopa opreme

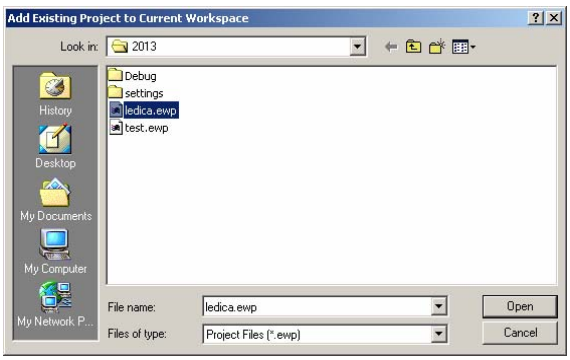
Zdaj kliknemo ***OK***. S tem je program pripravljen za prevajanje in razhroščevanje.

3.5 Odpiranje obstoječega projekta

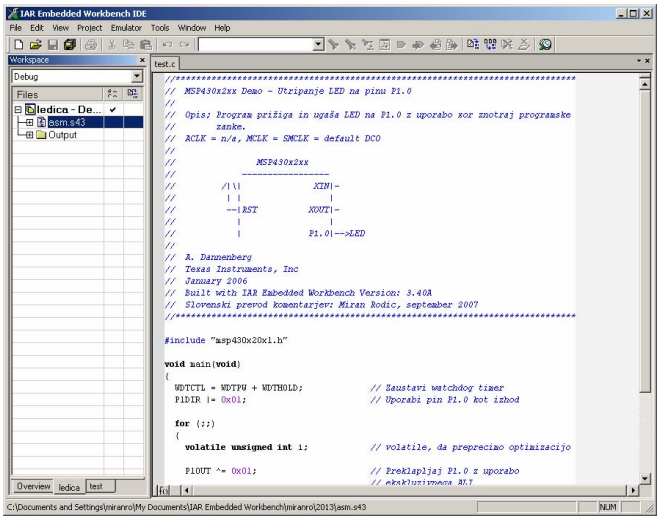
V primeru, ko želimo odpreti že obstoječi projekt, izberemo opcijo ***Project ➔ Add Existing Project ...*** (Slika 29) in izberemo obstoječi projekt (Slika 30), npr. ***ledica.ewp***. Okno programskega orodja se spremeni, kot to kaže Slika 31. Odpiranje obstoječega projekta je enako za programe v zbirnem jeziku in jeziku C.



Slika 29: Dodajanje obstoječega projekta



Slika 30: Izbira obstoječega projekta



Slika 31: Okno programskega orodja po izbiri obstoječega projekta

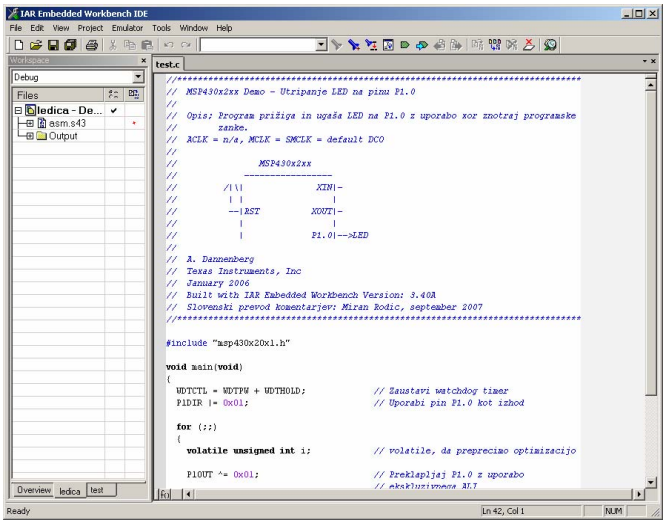
3.6 Delo z razhroščevalnikom

Razhroščevanik je orodje, ki ga uporabljamo za sledenje poteka programa. Predvsem je namenjen iskanju in odpravljanju napak.

IAR Embedded Workbench IDE ima vgrajen preprost razhroščevalnik, ki pa popolnoma zadošča našim zahtevam.

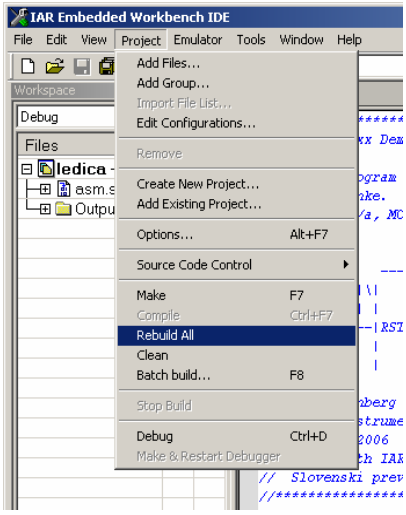
V nadaljevanju bo opisana uporaba razhroščevalnika za sledenje izvajanju preprostega programa, katerega kodo smo podali pri prikazu ustvarjanja novega projekta. Delo z razhroščevalnikom je popolnoma enako v C in zbirnem jeziku, zato bomo podali samo opis za programski jezik C.

Okno programskega orodja pred vklopom in začetkom izvajanja razhroščevalnika kaže Slika 32.



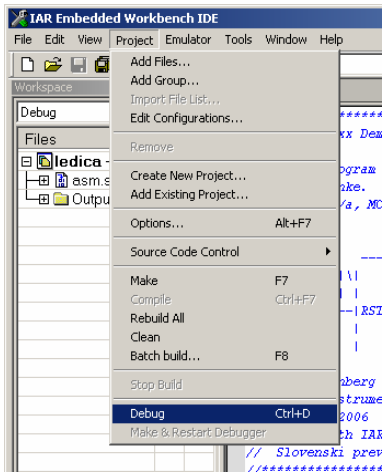
Slika 32: Okno programskega orodja pred začetkom razhroščevanja

Razhroščevanje začnemo s prevajanjem, ki ga izvedemo z uporabo izbire ***Project ➔ Rebuild All***.

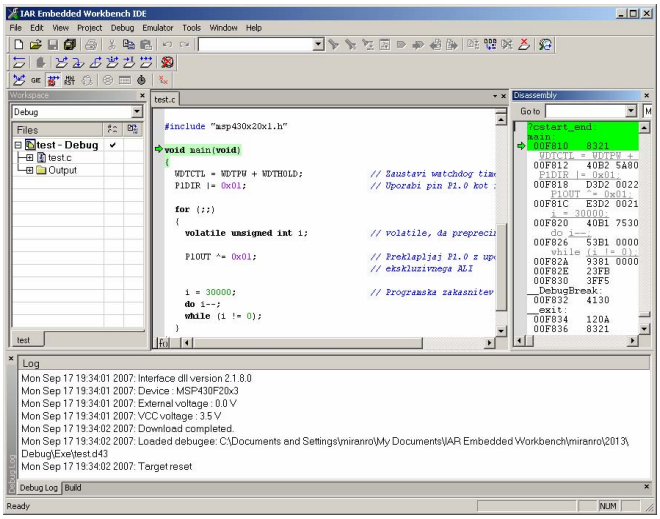


Slika 33: Prevajanje programa

Razhroščevalnik vklopimo z ukazom ***Project ➔ Debug***, tipkama ***<Ctrl>*** in ***<D>*** ali klikom na ikono  .

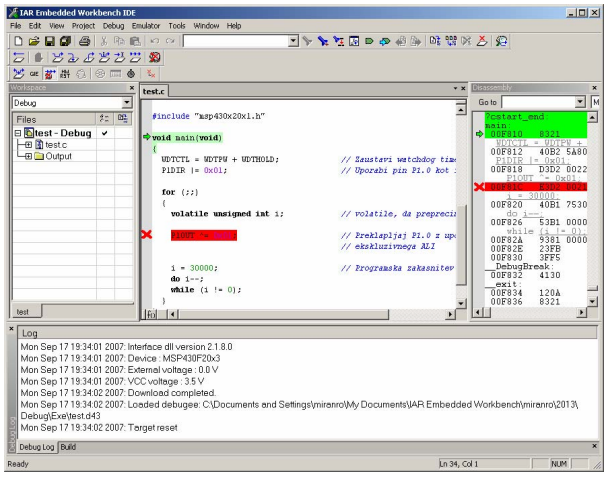


Slika 34: Vklop razhroščevalnika



Slika 35: Okno programskega orodja po zagonu razhroščevalnika

Če z levo tipko miške dvakrat kliknemo na levo stran kode (izven območja pisanja, na sivo polje), tja postavimo prekinitveno točko (Breakpoint).

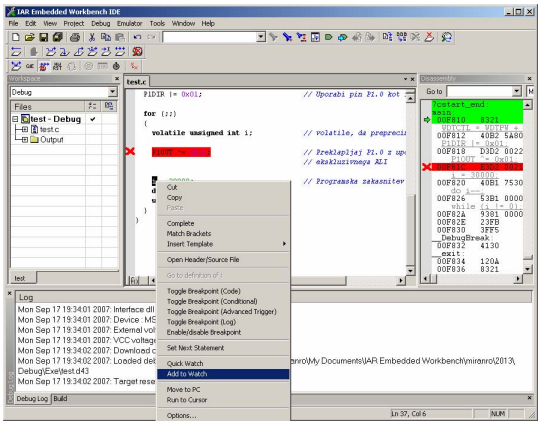


Slika 36: Izbira prekinitvene točke

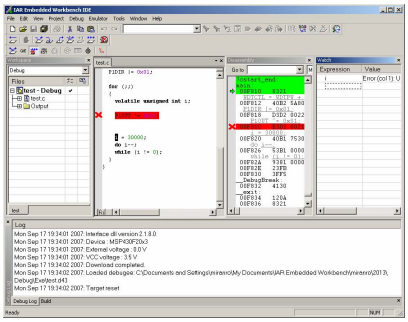
Program zaženemo z izbiro opcije ***Debug ➔ Go***, tipko ***<F5>*** ali klikom na ikono  . Ustavil se bo na mestu, kamor smo postavili prekinitveno točko.

Zelena puščica kaže na mesto, na katerem se je program zaustavil.

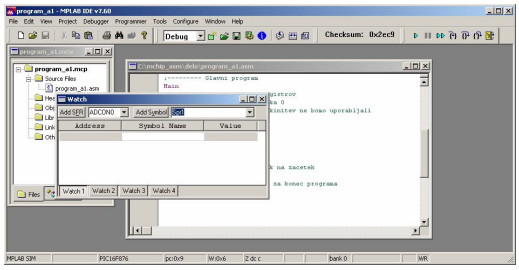
V naslednjem koraku nastavimo sledenje spremenljivki ***i***. To izvedemo tako, da z miško označimo spremenljivko in nato z desno tipko na miški izberemo ***Add to watch*** (Slika 37). Prikaže se okno (Slika 38), v katerem vidimo ime spremenljivke, ki jo želimo opazovati.



Slika 37: Izbira opazovane spremenljivke



Slika 38: Okno za opazovanje spremenljivk



Slika 39: Izbira imena spremenljivke *Spr1*, ki jo želimo opazovati

Opazujemo lahko tudi registre mikrokrmilnika, za kar uporabimo opcijo ***View ➔ Register***.

S klikom na ikono  se program pomakne za eno vrstico naprej.

V nadaljevanju lahko izbiramo opcije pomikanja pri razhroščevanju:

- ***Go*** – program teče.
- ***Step into*** – pomakni se za en korak naprej, če je v tem koraku funkcija, stopi vanjo.
- ***Step over*** – pomakni se za en korak naprej, če je v tem koraku funkcija, ne stopi vanjo.
- ***Step out*** – pomakni se za en korak naprej, če je program v funkciji, stopi iz nje.
- ***Next statement*** – pomakni se na naslednji ukaz.
- ***Break*** – zaustavi izvajanje programa.

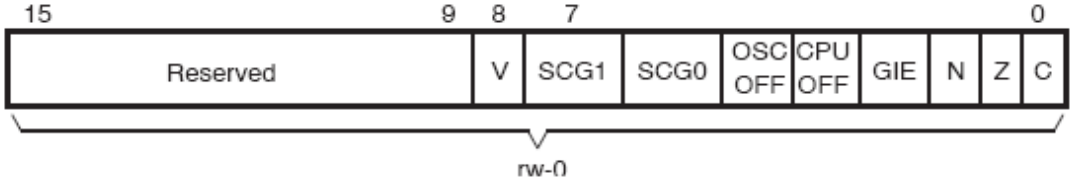
Če med izvajanjem ne želimo več opazovati vrednosti spremenljivke, se postavimo nanjo v oknu ***Watch*** in izberemo opcijo ***Remove***.

Razhroščevalnik zapustimo z izbiro ***Debug ➔ Stop Debugging*** ali klikom na ikono  .

4. Programiranje v zbirnem jeziku

Nabor ukazov zbirnega jezika MSP430F2013 zajema 27 ukazov v 7 režimih naslavljanja. Opisani bodo v nadaljevanju.

4.1 Statusni register (SR)



Slika 40: Statusni register

Bit	Opis	
V	Overflow bit. Bit za prekoračitev. Postavi se, ko rezultat aritmetične operacije prekorači obseg, določen s predznačenimi spremenljivkami.	
	Npr.:	
	ADD(.B),ADDC(.B)	V=1: Pozitivno + Pozitivno = Negativno Negativno + Negativno = Pozitivno Sicer V=0.
	SUB(.B),SUBC(.B),CMP(.B)	V=1: Pozitivno - Negativno = Negativno Negativno - Pozitivno = Pozitivno Sicer V=0.
SCG1	Generator sistemske ure št. 1. Ko je na 1, ugasne SMCLK.	
SCG0	Generator sistemske ure št. 0. Ko je na 1, ugasne DC generator DCO, če DCCLOCK ni uporabljen za MCLK ali SMCLK.	
OSCOFF	Oscillator Off. Ko je na 1, ugasne zunanji oscilator LFXT1, ko le-ta ni uporabljen za MCLK ali SMCLK	
CPUOFF	CPU off. Ko je na 1, ugasne CPE.	
GIE	General interrupt enable. Ko je na 1, omogoči maskirne prekinitve, ko je na 0, so maskirne prekinitve onemogočene.	
N	Negative bit. Negativni bit, postavi se, koje rezultat operacije negativen in izbriše, ko rezultat ni negativen.	
	Operacije .W:	N določa bit 15 rezultata.
	Operacije .B:	N določa bit 7 rezultata.
Z	Zero bit. Ničelni bit, postavljen je, ko je rezultat operacije 0, sicer je izbrisan.	
C	Carry bit. Bit za prenos, postavi se, ko je rezultat operacije dal prenos in izbriše, ko do prenosa ni prišlo.	

4.2 Pomen simbolov

src Izvorni operand (source), določen z As in S-reg.

dst	Ciljni operand (destination), določen z Ad in D-reg.
As	Biti, namenjeni izbiri režima naslavljanja za izvor (src).
S-reg	Delovni register, uporabljen za izvor (src).
Ad	Biti, namenjeni izbiri režima naslavljanja za cilj (dst).
D-reg	Delovni register, uporabljen za cilj (dst).
B/W	Operacija nad zlogom (byte) ali besedo (word): 0: operacija nad besedo. 1: operacija nad zlogom.
Rx (R0-R15)	Delovni registri
PC	Programski števec
SP	Kazalec na sklad
TOS	Vrh sklada
SR	Statusni register
WDT	Watchdog Timer
y(Rx)	Vsebina lokacije, ki jo poda (Rx + y), npr. 2(R5), indeksno naslavljanje
SPR	Spremenljivka, simbolično.
&SPR	Vsebina na naslovu, ki ga podaja spremenljivka SPR.
@Rx	Vsebina na naslovu, ki ga podaja vsebina registra Rx, indirektno naslavljanje.
#k	Konstanta, npr. #45h pomeni konstanto 45 v šestnajstiškem številskem sistemu.

4.3 Opisi ukazov

ADC.W **ADC.B**	**Prištej C**
Sintaksa:	[labela] ADC[.W] dst [labela] ADC.B dst
Operacija:	dst + C → dst
Spreminja status:	V, N, Z, C
Opis:	Vsebini dst prišteje C. Rezultat shrani v dst.

ADDC.W **ADDC.B**	**Seštej in dodaj C**
Sintaksa:	[labela] ADDC[.W] src,dst [labela] ADDC.B src,dst
Operacija:	src + dst + C → dst
Spreminja status:	V, N, Z, C
Opis:	Sešteje vsebine src, dst in C. Rezultat shrani v dst.

ADD.W **ADD.B**	**Seštej**
Sintaksa:	[labela] ADD[.W] src,dst [labela] ADD.B src,dst
Operacija:	src + dst → dst
Spreminja status:	V, N, Z, C
Opis:	Sešteje vsebini src in dst. Rezultat shrani v dst.

AND.W **AND.B**	**Logični IN**
Sintaksa:	[labela] AND[.W] src,dst [labela] AND.B src,dst
Operacija:	src & dst → dst
Spreminja status:	V, N, Z, C
Opis:	Bitni IN nad src in dst. Rezultat shrani v dst.

BIC.W **BIC.B**	**Briši bite**
Sintaksa:	[labela] BIC[.W] src,dst [labela] BIC.B src,dst
Operacija:	(~src) & dst → dst
Spreminja status:	-
Opis:	Nad negiranim src in dst izvede bitni IN. Rezultat shrani v dst.

BIS.W **BIS.B**	**Postavi bite**
Sintaksa:	[labela] BIS[.W] src,dst [labela] BIS.B src,dst
Operacija:	src | dst → dst
Spreminja status:	-
Opis:	Nad src in dst izvede bitni ALI. Rezultat shrani v dst.

BIT.W **BIT.B**	**Testiraj bite**
Sintaksa:	[labela] BIT[.W] src,dst [labela] BIT.B src,dst
Operacija:	src & dst
Spreminja status:	N, Z, V, C
Opis:	Nad src in dst izvede bitni IN. Postavi statusni register.

BR	**Vejitev**
Sintaksa:	[labela] BR dst
Operacija:	dst → PC
Spreminja status:	-
Opis:	Vejitev na lokacijo, ki jo označuje dst.

CALL	**Klic podprograma**
Sintaksa:	[labela] CALL dst
Operacija:	dst → tmp SP − 2 → SP PC → @SP tmp → PC
Spreminja status:	-
Opis:	Klic podprograma na lokaciji, ki jo označuje dst.

CLR.W **CLR.B**	**Briši**
Sintaksa:	[labela] CLR[.W] dst [labela] CLR.B dst
Operacija:	0 → dst
Spreminja status:	-
Opis:	Briše vsebino dst.

CLRC	**Briši C**
Sintaksa:	[labela] CLRC
Operacija:	0 → C
Spreminja status:	C
Opis:	Postavi bit C v statusnem registru na 0.

CLRN	**Briši N**
Sintaksa:	[labela] CLRN
Operacija:	0 → N
Spreminja status:	N
Opis:	Postavi bit N v statusnem registru na 0.

CLRZ	**Briši Z**
Sintaksa:	[labela] CLRZ
Operacija:	0 → Z
Spreminja status:	Z
Opis:	Postavi bit Z v statusnem registru na 0.

CMP.W **CMP.B**	**Primerjaj**
Sintaksa:	[labela] CMP[.W] src, dst [labela] CMP.B src, dst
Operacija:	dst - src
Spreminja status:	N, Z, V, C
Opis:	Primerja src in dst, postavi bite statusnega registra.

DADC.W **DADC.B**	**Prištej C desetiško**
Sintaksa:	[labela] DADC[.W] dst [labela] DADC.B dst
Operacija:	dst + C → dst (desetiško)
Spreminja status:	N, Z, V, C
Opis:	Desetiško prišteje C k dst.

DADD.W **DADD.B**	**Seštej desetiško**
Sintaksa:	[labela] DADD[.W] src, dst [labela] DADD.B src, dst
Operacija:	src + dst + C → dst (desetiško)
Spreminja status:	N, Z, V, C
Opis:	Desetiško sešteje src, dst in C. Rezultat shrani v dst.

DEC.W **DEC.B**	**Dekrement**
Sintaksa:	[labela] DEC[.W] dst [labela] DEC.B dst
Operacija:	dst - 1 → dst
Spreminja status:	N, Z, V, C
Opis:	Zmanjša dst za 1, rezultat shrani v dst.

DECD.W **DECD.B**	**Dvojni dekrement**
Sintaksa:	[labela] DECD[.W] dst [labela] DECD.B dst
Operacija:	dst - 2 → dst
Spreminja status:	N, Z, V, C
Opis:	Zmanjša dst za 2, rezultat shrani v dst.

DINT	**Onemogoči prekinitve**
Sintaksa:	[labela] DINT
Operacija:	0 → GIE
Spreminja status:	GIE
Opis:	Onemogoči klice in izvajanje prekinitvenih rutin.

EINT	**Omogoči prekinitve**
Sintaksa:	[labela] EINT
Operacija:	1 → GIE
Spreminja status:	GIE
Opis:	Omogoči klice in izvajanje prekinitvenih rutin.

INC.W **INC.B**	**Inkrement**
Sintaksa:	[labela] INC[.W] dst [labela] INC.B dst
Operacija:	dst + 1 → dst
Spreminja status:	N, Z, V, C
Opis:	Poveča dst za 1, rezultat shrani v dst.

INCD.W **INCD.B**	**Dvojni inkrement**
Sintaksa:	[labela] INCD[.W] dst [labela] INCD.B dst
Operacija:	dst + 2 → dst
Spreminja status:	N, Z, V, C
Opis:	Poveča dst za 2, rezultat shrani v dst.

INV.W **INV.B**	**Invertiraj**
Sintaksa:	[labela] INV[.W] dst [labela] INV.B dst
Operacija:	~dst → dst
Spreminja status:	N, Z, V, C
Opis:	Invertira dst, rezultat shrani v dst.

JC **JHS**	**Skoči, če je postavljen C** **Skoči, če večje ali enako**
Sintaksa:	[labela] JC labela [labela] JHS labela

Operacija:	if C == 1:
	PC + 2 * offset → PC
	if C == 0:
	izvedi naslednji ukaz

Spreminja status:	-
Opis:	Če je C na 1 (rezultat primerjave je večje ali enako), skoči na mesto, ki ga označuje labela. Sicer nadaljuj z izvajanjem naslednjega ukaza.

JEQ **JZ**	**Skoči, če enako** **Skoči, če je 0**
Sintaksa:	[labela] JEQ labela [labela] JZ labela

Operacija:	if Z == 1:
	PC + 2 * offset → PC
	if Z == 0:
	izvedi naslednji ukaz

Spreminja status:	-
Opis:	Če je Z na 1 (rezultat primerjave je enako), skoči na mesto, ki ga označuje labela. Sicer nadaljuj z izvajanjem naslednjega ukaza.

JGE	**Skoči, če večje ali enako**
Sintaksa:	[labela] JGE labela

Operacija:	If (N ^ V) == 0:
	PC + 2 * offset → PC
	If (N ^ V) == 1:
	izvedi naslednji ukaz

Spreminja status:	-
Opis:	Če je rezultat primerjave večje ali enako, skoči na mesto, ki ga označuje labela. Sicer nadaljuj z izvajanjem naslednjega ukaza. Omogoča primerjavo predznačenih celih števil.

JL	**Skoči, če manj**
Sintaksa:	[labela] JL labela

Operacija:	If (N ^ V) == 1:
	PC + 2 * offset → PC
	If (N ^ V) = 0:
	izvedi naslednji ukaz

Spreminja status:	-
Opis:	Če je rezultat primerjave manjše, skoči na mesto, ki ga označuje labela. Sicer nadaljuj z izvajanjem naslednjega ukaza. Omogoča primerjavo predznačenih celih števil.

JMP	**Brezpogojni skok**
Sintaksa:	[labela] JMP labela
Operacija:	PC + 2 * offset → PC
Spreminja status:	-
Opis:	Skoči na mesto, ki ga označuje labela.

JN	**Skoči, če je negativno**
Sintaksa:	[labela] JN labela

Operacija:	if N == 1:
	PC + 2 * offset → PC
	if N == 0:
	izvedi naslednji ukaz

Spreminja status:	-
Opis:	Če je N na 1, skoči na mesto, ki ga označuje labela. Sicer nadaljuj z izvajanjem naslednjega ukaza.

JNC	**Skoči, če ni postavljen C**
JLO	**Skoči, če manjše**
Sintaksa:	[labela] JNC labela [labela] JLO labela

Operacija:	if C == 0:
	PC + 2 * offset → PC
	if C == 1:
	izvedi naslednji ukaz

Spreminja status:	-
Opis:	Če je C na 0 (rezultat primerjave je manjše), skoči na mesto, ki ga označuje labela. Sicer nadaljuj z izvajanjem naslednjega ukaza.

JNE	**Skoči, če ni enako**
JNZ	**Skoči, če ni 0**
Sintaksa:	[labela] JNE labela [labela] JLZ labela

Operacija:	if Z == 0:
	PC + 2 * offset → PC
	if Z == 1:
	izvedi naslednji ukaz

Spreminja status:	-
Opis:	Če je Z na 0 (rezultat primerjave ni enako), skoči na mesto, ki ga označuje labela. Sicer nadaljuj z izvajanjem naslednjega ukaza.

MOV.W	**Prepiši**
MOV.B	
Sintaksa:	[labela] MOV[.W] src, dst [labela] MOV.B src, dst
Operacija:	src → dst
Spreminja status:	-
Opis:	Prepiše dst z vrednostjo src.

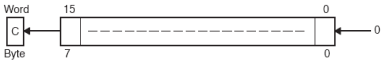
NOP	**Ni operacije**
Sintaksa:	[labela] NOP
Operacija:	-
Spreminja status:	-
Opis:	Ne naredi ničesar.

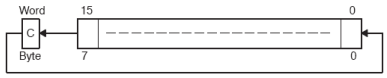
POP.W **POP.B**	**Poberi iz sklada**
Sintaksa:	[labela] POP[.W] dst [labela] POP.B dst
Operacija:	@SP → temp SP + 2 → SP temp → dst
Spreminja status:	-
Opis:	Prebere vsebino sklada v dst.

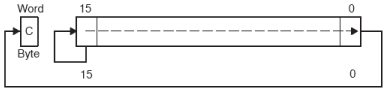
PUSH.W **PUSH.B**	**Odloži na sklad**
Sintaksa:	[labela] PUSH[.W] src [labela] PUSH.B src
Operacija:	SP − 2 → SP src → @SP
Spreminja status:	-
Opis:	Prebere vsebino sklada v dst.

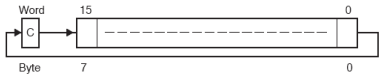
RET	**Vrnitev iz podprograma**
Sintaksa:	[labela] RET
Operacija:	@SP→ PC SP + 2 → SP
Spreminja status:	-
Opis:	Zaključi podprogram in nadaljuje izvajanje z ukazom, ki sledi ukazu za klic podprograma.

RETI	**Vrnitev iz prekinitve**
Sintaksa:	[labela] RETI
Operacija:	TOS → SR SP + 2 → SP TOS → PC SP + 2 → SP
Spreminja status:	N, Z, C, V
Opis:	Zaključi izvajanje prekinitvene rutine in nadaljuj izvajanje tam, kjer je bil izveden klic prekinitve.

RLA.W **RLA.B**	**Aritmetična rotacija v levo**
Sintaksa:	[labela] RLA[.W] dst [labela] RLA.B dst
Operacija:	
Spreminja status:	N, Z, C, V
Opis:	Pomakne bite dst za eno mesto v levo, na najnižje mesto postavi 0.

RLC.W **RLC.B**	**Rotacija v levo skozi C**
Sintaksa:	[labela] RLC[.W] dst [labela] RLC.B dst
Operacija:	
Spreminja status:	N, Z, C, V
Opis:	Pomakne bite dst za eno mesto v levo, na najnižje mesto postavi vsebino C.

RRA.W **RRA.B**	**Aritmetična rotacija v desno**
Sintaksa:	[labela] RRA[.W] dst [labela] RRA.B dst
Operacija:	
Spreminja status:	N, Z, C, V
Opis:	Pomakne bite dst za eno mesto v desno, na najvišjem mestu ostane vrednost bita nespremenjena.

RRC.W **RRC.B**	**Rotacija v desno skozi C**
Sintaksa:	[labela] RRC[.W] dst [labela] RRC.B dst
Operacija:	
Spreminja status:	N, Z, C, V
Opis:	Pomakne bite dst za eno mesto v desno, na najvišje mesto postavi vsebino C.

SBC.W **SBC.B**	**Odštej z izposojo**
Sintaksa:	[labela] SBC[.W] dst [labela] SBC.B dst
Operacija:	dst + 0FFFFh + C → dst dst + 0FFh + C → dst
Spreminja status:	N, Z, C, V
Opis:	Prišteje vsebino C vrednosti dst, ki je predhodno zmanjšana za 1.

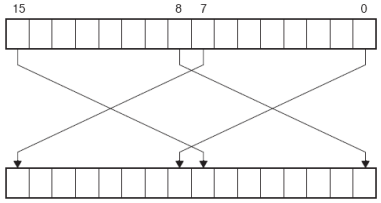
SETC	**Postavi C**
Sintaksa:	[labela] SETC
Operacija:	1 → C
Spreminja status:	C
Opis:	Postavi bit C v statusnem registru na 1.

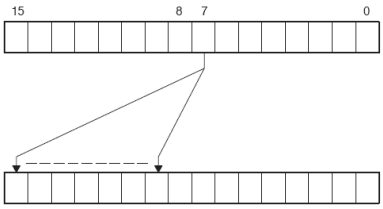
SETN	**Postavi N**
Sintaksa:	[labela] SETN
Operacija:	1 → N
Spreminja status:	N
Opis:	Postavi bit N v statusnem registru na 1.

SETZ	**Postavi Z**
Sintaksa:	[labela] SETZ
Operacija:	1 → Z
Spreminja status:	Z
Opis:	Postavi bit Z v statusnem registru na 1.

SUB.W **SUB.B**	**Odštej**
Sintaksa:	[labela] SUB[.W] src,dst [labela] SUB.B src,dst
Operacija:	dst + ~src + 1 → dst [(dst − src → dst)]
Spreminja status:	V, N, Z, C
Opis:	Odšteje vsebino src od dst. Rezultat shrani v dst.

SUBC.W **SUBC.B**	**Odštej z izposojo**
Sintaksa:	[labela] SUBC[.W] src,dst [labela] SUBC.B src,dst
Operacija:	dst + ~src + C → dst [(dst − src − 1 + C → dst)]
Spreminja status:	V, N, Z, C
Opis:	Odšteje vsebino src od dst in upošteva C. Rezultat shrani v dst.

SWPB	**Zamenjaj zloga**
Sintaksa:	[labela] SWPB dst
Operacija:	
Spreminja status:	-
Opis:	Zamenja vsebini zgornjega in spodnjega zloga v dst. Rezultat shrani v dst.

SXT	**Razširi predznak**
Sintaksa:	[labela] SXT dst
Operacija:	bit7 → bit8, ..., bit15 
Spreminja status:	-
Opis:	Razširi predznak iz zloga na besedo. Rezultat shrani v dst.

TST.W **TST.B**	**Testiraj**
Sintaksa:	[labela] TST[.W] dst [labela] TST.B dst
Operacija:	dst + 0FFFFh + 1 dst + 0FFh + 1
Spreminja status:	V, N, Z, C
Opis:	Primerja dst z 0, rezultat vpliva na bite statusnega registra, dst pa se ne spremeni.

XOR.W **XOR.B**	**Logični ekskluzivni ALI**
Sintaksa:	[labela] XOR[.W] src,dst [labela] XOR.B src,dst
Operacija:	src ^ dst → dst
Spreminja status:	V, N, Z, C
Opis:	Bitni ekskluzivni ALI nad src in dst. Rezultat shrani v dst.

4.4 Naslavljanja

Uporabljamo 7 tipov naslavljanj:

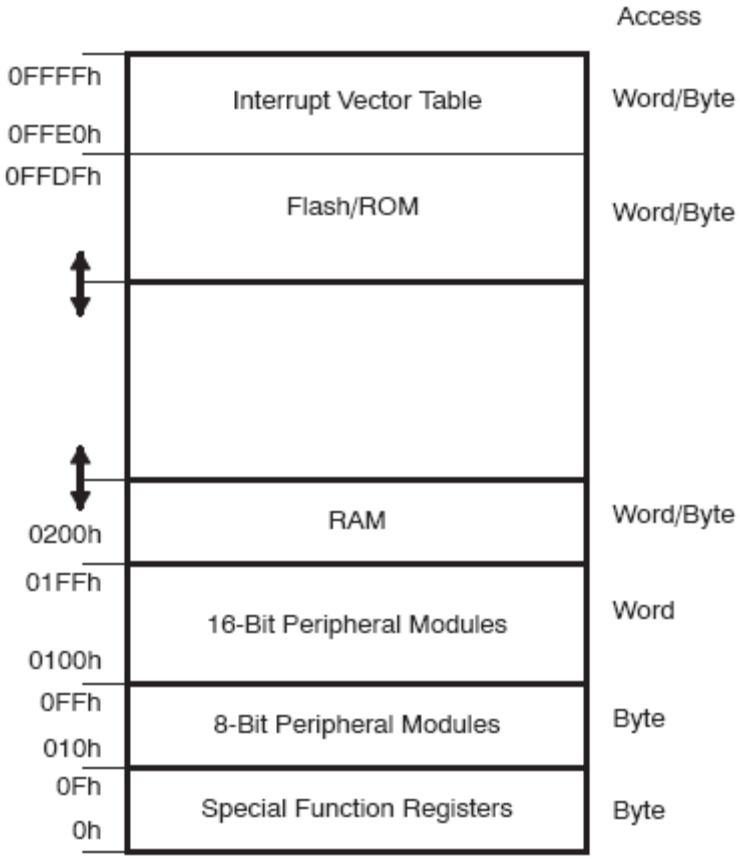
Naslavljanje	(ang)	Sintaksa	Opis
Registersko	Register mode	Rn	Operandi so registri. Primer: `MOV R10, R11`
Indeksno	Indexed mode	X(Rn)	Na operand kaže (Rn + X). X je shranjen v naslednji besedi. Primer: `MOV 2(R5),6(R6)`

Simbolno	Symbolic mode	ADDR	Na operand kaže (PC + X). X je shranjen v naslednji besedi.
			Primer:
			`MOV SPR1,SPR2`
			`;Src. addr. SPR1 = 0F016h`
			`;Dest. addr. SPR2 = 01114h`

Absolutno	Absolute mode	&ADDR	Beseda, ki sledi instrukciji, vsebuje absolutni naslov. X je shranjen v naslednji besedi.
			Primer:
			`MOV &SPR1,&SPR2`
			`;Src. addr. SPR1 = 0F016h`
			`;Dest. addr. SPR2 = 01114h`

Indirektno registersko	Indirect register mode	@Rn	Delovni register Rn je uporabljen kot kazalec na operand. Primer: `MOV.B @R10,0(R11)`
Indirektno z avtoinkrementom	Indirect autoincrement	@Rn+	Delovni register Rn je uporabljen kot kazalec na operand. Po uporabi se vsebina Rn poveča za 1 za instrukcije .B in za 2 za instrukcije .W. Primer: `MOV @R10+,0(R11)`
Trenutno	Immediate mode	#N	Beseda, ki sledi instrukciji, vsebuje konstanto N. Primer: `MOV #45h,SPR1`

4.5 Organizacija pomnilnika



Slika 41: Pomnilnik MSP430F2013

5. Literatura

[1] *eZ430-F2013 User's Guide (Rev. B), http://focus.ti.com/docs/toolsw/folders/print/iar-kickstart.html*

[2] FET User's Guide (IAR Workbench Ver 3.x) (Rev. D), http://www-s.ti.com/sc/techlit/slau138

[3] MSP430F2xx Family User's Guide (Rev. B), http://www-s.ti.com/sc/techlit/slau144

6. Varstvo avtorskih pravic