stala v odvisnosti od temperature. Pri aplikacijah, kjer se uporablja tudi drugi kristal nizke frekvence, lahko možnost boot-clock frekvence 32 kHz, ki se uporablja pri zagonu ob priključitvi napajanja, odpravi potrebo po dodatnem drugem kristalu.

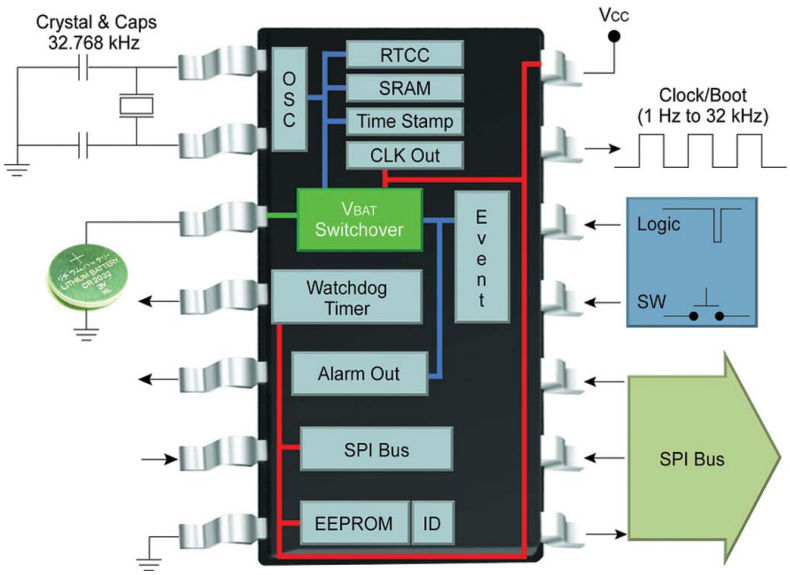
Hčerinska ploščica MCP795XX SPI RTCC PICtail ™ Plus Daughter Board (AC164147) bo za 45 $, naprodaj v februarju 2012. Hčerinska ploščica je združljiva z Microchip Explorer 16 Development Board (DM240001) in PIC18 Explorer Board (DM183032).

MCP795WXX/BXX RTCC vezja so na voljo v SOIC 14-pinskih in TSSOP ohišjih.

www.microchip.com

MICROCHIP

SPI Real-Time Clock/Calendar Block Diagram



Pokukajmo v notranjost programa Bascom-AVR

Avtor: mag. Vladimir Mitrović
E-pošta: vladimir.mitrovic@podravka.hr

Ste se kdaj vprašali, kakšen je videti vaš program, ko ga Bascom-AVR prevede v obliko, ki je razumljiva mikrokontrolerju? Tega ni tako težko izvedeti - samo odpreti morate .hex datoteko in pogledati nize heksadecimalnih števil, ki so zapisane v njih. Videti je nekoliko nečitljivo, kaj pravite? Težko je verjeti, da bi lahko ti „nizi naključnih števil“ komurkoli karkoli pomenili. Ampak ne, ta števila niso naključna in da, ta števila vodijo mikrokontroler, da deluje točno tako, kot ste zapisali v svojem programu.

Obstaja preprosto orodje, s katerim lahko pretvorite heksadecimalno zapisani objektni program v popolnoma razumljiv simbolični jezik, asembler (zbirnik). Orodje se imenuje disasembler: gre za računalniški program, ki prebere *.hex datoteko, v kateri je zapisan program, ki ga „razumejo“ (v našem primeru) AVR mikrokontrolerjem in ga izpiše v obliki ukazov zbirnika (asemblerja). Če bi iste ukaze v zbirniku spet prevedli z AVR zbirnikom, bi dobili enako objektno kodo, s katero smo začeli. No, mi ga bomo uporabljali na drugačen način - disasemblirani program je namreč lažje brati in razumeti kot niz heksadecimalnih števil.

Za disasembliranje običajno uporabljam Revava disasembler. To je brezplačni program, ki si ga lahko naložite z različnih internetnih strani, lahko v obliki C++ izvorne koda ali kot že preveden (.exe) program. V C++ obliki programa boste zraven najpogosteje dobili tudi navodila kako ga prevesti. Nameščanje ni potrebno: enostavno shranite program revava.exe v mapo, kjer so tudi ostali vaši Bascom-AVR programi (bolj natančno, njihove .hex datoteke) in že je pripravljen za uporabo. Uporabite lahko tudi druge podobne programe, vendar se bo njihov način uporabe in njihove izhodne datoteke delno razlikovale od primerov, ki jih bomo analizirali tukaj.
Slika 1 prikazuje primer disasembliranja objektnega programa, ki ga vsebuje datoteka Asm_1.hex v zbirniku. Objektni program iz tega primera je rezultat prevajanja „praznega“ Bascom-AVR programa, v katerem je zapisan le End ukaz. Kljub temu je Bascom-AVR ustvaril nekaj programske kode. Revava se izvaja znotraj DOS-ovega okna:

```
>revava Asm_1.hex,_
-o Asm_1.txt -e
```

Asm_1.hex:
» ime vhodne .hex datoteke
» -o: sledi ime izhodne datoteke
Asm_1.txt:
» naziv izhodne datoteke
-e:
» uporablja Intelov način zapisa

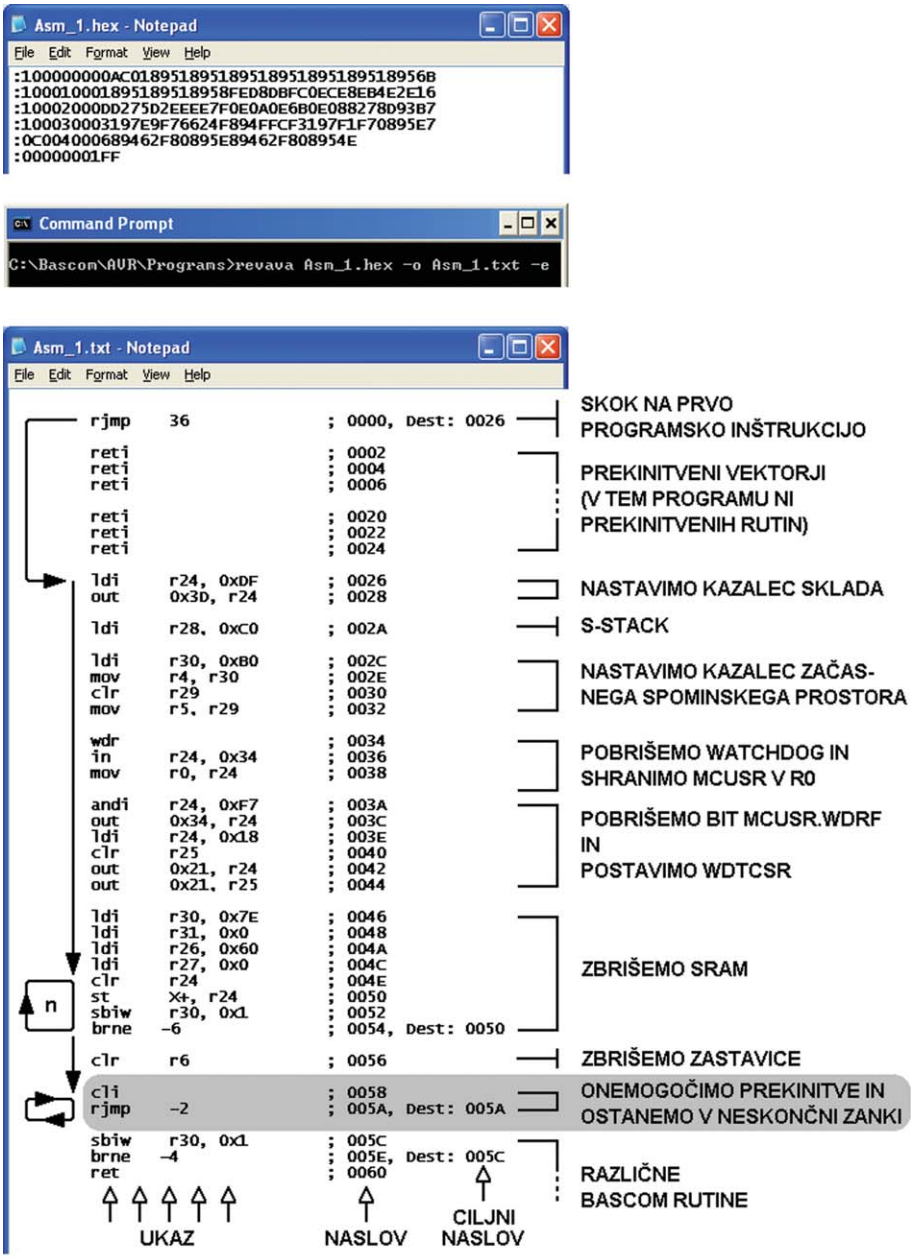
Najpogosteje uporabljam le opciji -o in -e, izpis vseh opcij pa dobite, če vpišete le „revava“, bez dodatnih opcij. Ustvarjen asemblerski program je prikazan na sliki 1 spodaj. Ta izpis je še dodatno malce „ročno obdelan“, da je bolj čitljiv. Da bi ga razumeli, je potrebno splošno predznanje o programiranju v zbirniku in poznavanje mnemonikov (asemblerskih ukazov, oznak), ki jih uporabljamo za programiranje v asemblerju za AVR mikrokontrolerje. Nekaj informacij o tem lahko najdete tudi v Bascom-AVR help-u, v poglavjih „Assembler mnemonics“ in „Mixing ASM and BASIC“. Priporočljivo je, da si dobro ogledate tudi poglavje „Language Fundamentals“.

Preden se spustimo v analizo programa, si oglejmo še samo strukturo izpisa. Revava disasembler je izpisal ukaze zbirnika kjerkoli je bilo to mogoče. Nize, ki jih ni prepoznal, pa je prikazal kot podatke (dc.W deklaracije). V vsaki vrstici se nahaja komentar (začne se za znakom »;«) v katerem je zapisan naslov, na katerem se nahaja disasembliran ukaz ali podatek. Če gre za ukaz skoka, razvejanja ali klic podprograma, je naveden tudi ciljni naslov (napisan je za „Dest:“). V primerih, ko je lahko objektna koda nastala s prevajanjem dveh ali več asemblerskih ukazov, so prikazane vse možnosti. Te alternative so prikazane kot skupina ukazov, pred katero in za katero je ena prazna vrstica. Razen prvega so vsi ostali ukazi takšne skupine opremljene s komentarji, kot kaže naslednji primer:

```
brbc   1, -6,_
     ; 0054, Dest: 0050
;brne  -6,_
     ; 0054, Dest: 0050

clr    r6          ; 0056
;eor   r6, r6      ; 0056
```

Tu ukaza BRBC in BRNE in tudi CLR ter EOR, ustvarita enako objektno kodo. Revava ne more „vedeti“, kateri ukaz je bil prvotno res napisan, zato ponudi obe možnosti (lahko jih je tudi več). Opazili boste, da sta alternativna ukaza, v našem primeru BRNE in EOR, opremljena z znakom za komentar (;), zato ne vplivata na kodo pri prevajanju. V naslovnem delu vrstice vidimo, da se nahajata na istih naslovih kot



Slika 1: Primer disasembliranja Asm_1.hex datoteke v asemblerski program

ukazi, katerim so alternative. V konkretnem programu izberemo ukaz, ki se nam zdi bolj smiseln; katerokoli izberemo, bo končni rezultat enak, saj vsi ukazi znotraj ene skupine povzročijo ustvarjanje enake objektne kode. V primeru na Sliki 1 so izbrisani vsi alternativni predlogi, da je program bolj čitljiv.

Zdaj pa poskusimo analizirati, kako je Bascom-AVR prevedel prazen program!

Prvi ukaz vsakega programa je skok na prvi „pravi izvršilni ukaz“ programa. Ta se v našem primeru nahaja na naslovu 0026 heksadecimalno (vsa števila so prikazana v heksadecimalni obliki). Med tem skokom in prvim „pravim“ ukazom se nahaja cela vrsta „prekinitvenih vektorjev“. Vsak od njih predstavlja skok na začetni naslov določene prekinitvene rutine. V tem programu prekinitve niso določene, zato je v vseh prekinitvenih vektorjih zapisan le ukaz za vrnitev iz prekinitvene rutine (RETI). Število prekinitvenih vektorjev je odvisno od mikrokontrolerja, za katerega smo napisali program (ukaz $regfile). Ta primer je napisan za ATtiny2313, ki ima skupaj 18 prekinitvenih vektorjev; da je izpis krajši, jih je na sliki 1 je prikazano le šest.

Prevajalnik Bascom-AVR ustvari nekaj inicijalizacijskih rutin, ki se izvedejo pred prvim ukazom vašega programa. Najprej se ustvarita strojni in programski sklad ter kazalec na del pomnilnika (SRAM), ki se uporablja za shranjevanje začasnih podatkov (delovno področje nekaterih ukazov, tu se hranijo tudi vmesni rezultati in podobno). Sledi vrsta ukazov, s katerimi nastavimo delovanje Watchdog časovnika. Pri tem postopku se spremeni vrednost MCUSR registra, zato Bascom-AVR njegovo začetno vrednost najprej shrani v register R0. Tako lahko kasneje med izvajanjem programa preverimo, če se je začelo izvajanje programa zaradi reseta Watchdog časovnika ali kako drugače.

Naslednja rutina, ki jo je ustvaril prevajalnik Bascom-AVR , je programska zanka, v kateri zbrišemo vsebino celotnega SRAM pomnilnika. Če želite pri zagonu programa ohraniti vsebino SRAM pomnilnika (to je lahko koristno, če je reset mikrokontrolerja povzročil Watchdog časovnik), lahko ustvarjanje te rutine za brisanje preprečite tako, da v programu napišete ukaz $noramclear. Pri tem se morate zavedati, da je vsebina SRAM brez uvodnega brisanja nedefinirana in so v njem zapisane vrednosti spremenljivk nepredvidljive. Če bi to lahko „zmedlo“ vaš program, morate začetne vrednosti vseh kritičnih spremenljivk vpisati sami na začetku programa, ali pa to prepustite Bascom-AVR prevajalniku, da to stori namesto vas.

Zadnja inicijalizacijska rutina izbriše register R6, ki ga Bascom-AVR uporablja za shranjevanje nekaterih bitnih spremenljivk za interno uporabo. Spremenljivka Err je na primer shranjena ravno v registru R6.

Opombe:

- število inicijalizacijskih rutin in njihova vsebina je odvisna od mikrokontrolerja za katerega je program napisan,
- poleg $noramclear obstajajo še druge podobne „$no...“ direktive, ki vplivajo na inicijalizacijski del programa,
- s pomočjo $initmicro direktive lahko v program vnesete svoje lastne inicijalizacijske rutine. Pišete jih znotraj podprograma _INIT_MICRO, ki ga $initmicro pokliče takoj, ko je končana osnovna inicijalizacija mikrokontrolerja. Več o tem lahko najdete v Bascom-AVR help-u.

Za Bascom inicijalizacijskimi rutinami je zapisan vaš program; ta del je na sliki 1 napisan na sivi podlagi. V tem primeru je Bascom-AVR program vseboval le ukaz End, ki je preveden kot neskončna zanka, pred katero so onemogočene vse prekinitve. V „normalnem“ Bascom-AVR programu, v katerem je pred ukazom End še cela vrsta drugih izvršilnih ukazov, se bodo po prevajanju programa nahajali prav tu, pred End ukazom. Če so v Bascom-AVR po End ukazu zapisane tabele, podprogrami in/ali drugi ukazi in podatki, se bo vse to v prevedenem programu prav tako nahajalo za End ukazom. Skratka, Bascom-AVR prevajalnik spoštuje zgradbo vašega programa in svoje lastne rutine vstavlja pred ali za njim, nikoli pa znotraj njega.

V opazovanem primeru obstajajo nekatere rutine tudi za End ukazom. Te rutine se ne uporabljajo nikjer v programu in zato so tu odveč (ali pa morda jaz ne razumem njihovega smisla). Vendar tudi mnogi Bascom-AVR ukazi ustvarijo svoje lastne podprograme, ki jih prevajalnik shrani v ta prostor. Če na primer v svojem programu napišete Wait 1, se bo ta ukaz prevedel na dva dela:

- v delu programa, kjer ste uporabili Wait 1, se bodo v registre vpisale ustrezne vrednosti, potem pa poklical podprogram, ki bo res zadržal izvajanje vašega programa,
- ta podprogram se nahaja nekje na koncu programa, za vašim zadnjim ukazom.

1. analiza: Wait in Waitms

Ukaza Wait in Waitms sta idealna za začetni primer analize Bascom-AVR programov. Če želite analizirati ta dva ali katerega od ostalih Bascom-AVR ukazov, jih obkrožite z nekaj asemblerskimi NOP ukazi:

```
$crystal= 8000000
$regfile= „ATtiny2313.dat“
...
```


Smo edino podjetje v Sloveniji, kjer pod eno streho dobite svoj izdelek na ključ

vse cene so brez DDV-ja

24 URNI PROTOTIPI

PCB 49€

enoslojno ali dvoslojno tiskano vezje, do velikosti 1,5 dm2, dobava v treh tednih

MULTILAYER 99€

štirislojno tiskano vezje, do velikosti 1,5 dm2, dobava v treh tednih

SESTAVA PCB

Nudimo kompletno storitev izdelave tiskanega vezja (ES, DS, MULTI), opremljanje plošč s komponentami, izdelava mask, filmov, šablon ter prodaja laminata in elektronskega materiala.

25 letna tradicija zagotavlja najvišjo kakovost

www.luznar.com / info@luznar.com

Tel.: 04 28 18 800

Tiskana vezja

LUZNAR d.o.o.


```
...
nop
nop
nop
nop
Wait 1
nop
nop
nop
nop
Waitms 1
nop
nop
nop
nop
End
```

To nam bo v pomoč, da jih bomo v disasembliranem programu lažje izsledili, posebno takrat, kadar želimo analizirati posamezne ukaze ali logično celoto znotraj komleksnega programa. Slika 2 prikazuje, da se Wait 1 prevede kot:

```
ldi   r30, 0x1 ; 0060
ldi   r31, 0x0 ; 0062
rcall 26       ; 0064,_
                 ,Dest: 0080
```

Par registrov R30:R31 se uporabi za prenos želene zakasnitve (tukaj 1 sekunda) v podprogram, ki se začne na naslovu 0080.

Podobno se prevede tudi Waitms 1:

```
ldi   r24, 0x1 ; 006E
ldi   r25, 0x0 ; 0070
rcall 42       ; 0072,_
               ; Dest:009E
```

Waitms za prenos vrednosti zakasnitve v podprogram na naslovu 009E uporablja drug par registrov, R24:R25. Veliko Bascom-AVR ukazov se prevede na podoben način: na tistem mestu, kjer ste jih v programu napisali, bo prevajalnik v enega ali v več registrov naložil ustrezne vrednosti ali naslove spremenljivk, v katerih se te vrednosti nahajajo, potem pa bo poklical ustrezen podprogram. Kot smo že videli, so ti podprogrami shranjeni nekje na koncu prevedenega programa. Takšen pristop zagotavlja racionalno uporabo pomnilnika mikrokontrolerja, ker se (potencialno) dolg podprogram, v katerem se rutine res izvajajo, doda programu samo enkrat, ne glede na to, kolikokrat ga kasneje znotraj programa kličemo za izvajanje.

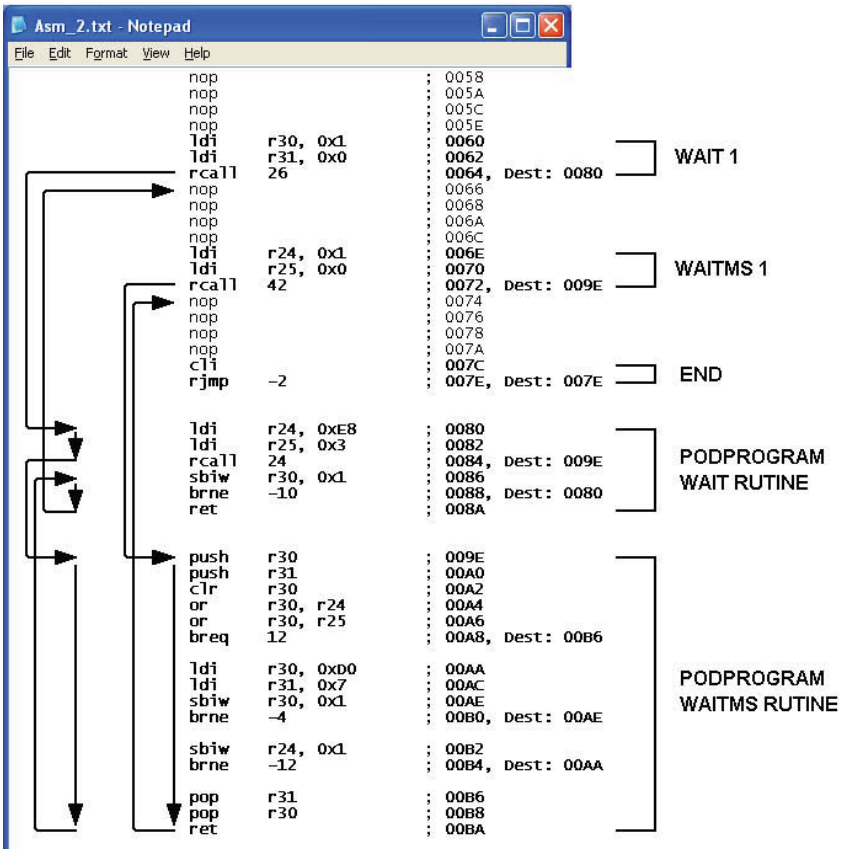
Osnovni del Waitms podprograma s slike 2 je pravzaprav programska zanka, ki se izvede tolikokrat, da dosežemo zakasnitev 1 ms:

```
ldi   r30, 0xD0 ; 00AA
ldi   r31, 0x7  ; 00AC
sbiw  r30, 0x1  ; 00AE
brne  -4        ; 00B0,_
                ; Dest:00AE
```

Potrebno število ponavljanj se določa med prevajanjem in je odvisno od frekvence oscilatorja. V tem primeru je $crystal = 8000000, zato se zanka SBIW+BRNE izvaja 0,5 µs (potrebni so štirje cikli, da se izvedeta ta dva ukaza). Iz tega sledi, da je potrebno za zakasnitev 1 ms zanko izvesti 2000 krat. 2000 je v heksadecimalnem zapisu 7D0 in ravno ta vrednost je zapisana v par registrov R30:R31.

Tudi sami lahko preverite trajanje te rutine, če jo zapišete v Bascom-AVR in preverite v simulatorju. Poudariti moram, da način zapisa v obliki, ki ga daje Reva-

```
$crystal = 8000000
$regfile = "ATtiny2313.dat"
nop
nop
nop
nop
Wait 1
nop
nop
nop
nop
Waitms 1
nop
nop
nop
nop
End
```



Slika 2: Analiza ukazov Wait in Waitms

va disasembler ni popolnoma združljiv z zapisom, ki ga podpira Bascom-AVR: potrebno je izbrisati vse komentarje („;“ in podatek, ki sledi) in spremeniti obliko zapisa heksadecimalnih števil. Relativno naslavljanje v inštrukcijah skokov, razvejanja in pri klicu podprogramov v obliki, ki jo daje Revava, Bascom-AVR ne zna korektno prevesti, zato je naslavljanje najbolje nadgraditi z uvedbo simboličnih naslovov (label):

```
ldi    r30,$D0
ldi    r31,$7
1ms_loop:
sbiw   r30,$1
brne   1ms_loop
```

Ta del programa se v simulatorju izvaja natančno 1,000125 ms. Registrski par R24:R25 vsebuje parameter iz Waitms ukaza. V tem primeru je to 1, za Waitms 100 pa bi bilo 100. Ta vrednost se uporablja kot števec, ki določa, kolikokrat se rutina za zakasnitev izvede, kar ima za posledico zakasnitve v razponu od 1 do 65535 ms. To so približne vrednosti; realno trajanje Waitms podprograma je nekoliko daljše, ker vsebuje še nekaj uvodnih in zaključnih ukazov, ki smo jih v tej analizi zanemarili.

Če sedaj pogledate Wait podprogram, boste opazili, da ta pravzaprav kliče

Waitms (Waitms se začne na naslovu 009E):

```
ldi   r24, 0xE8 ; 0080
ldi   r25, 0x3  ; 0082
rcall 24        ; 0084,_
                ; Dest:009E
```

Heksadecimalna vrednost 3E8 , ki je vpisana v par registrov R24:R25, ustreza decimalni vrednosti 1000, zato se bo Waitms izvedel 1000 krat in bo eno izvajanje Wait ukaza trajalo ravno toliko časa, kot je treba: približno 1 sekundo. To se množi z vrednostjo, ki je vpisana v paru registrov R30:R31, zato je lahko skupno trajanje zakasnitve, ki jo lahko izvedemo z Wait ukazom med 1 in 65535 sekundami.

Zanimivo je, kako Wait podprogram kliče Waitms, da bi izvedel zakasnitev 1 sekunde. To lahko v Bascom-AVR programu večkrat srečamo in med analizo bomo lahko videli, da se včasih posamezni deli nekaterih, na prvi pogled popolnoma različnih ukazov kličejo med seboj. Na ta način je programski pomnilnik racionalno izkoriščen, obenem pa nam lahko zelo oteži analiziranje programa.

2. analiza: Timer/Counter1 ukazi

Na sliki 3 so prikazani disasemblirani ukazi naslednjega Bascom-AVR programa:

```
$crystal= 8000000
$regfile= „ATtiny2313.dat“
```

```
$crystal = 8000000
$regfile = "ATtiny2313.dat"
nop
nop
nop
nop
Config TIMER1 = Timer , Prescale = 8 , Clear Timer = 1 , Compare A = Toggle
nop
nop
Stop TIMER1
nop
nop
Start TIMER1
nop
nop
nop
nop
End
```

Asm_3.txt - Notepad

File Edit Format View Help

```
        nop                     ; 0058
        nop                     ; 005A
        nop                     ; 005C
        nop                     ; 005E
        ldi     r24, 0x40       ; 0060
        out     0x2F, r24       ; 0062
        ldi     r24, 0xA        ; 0064
        out     0x2E, r24       ; 0066
        nop                     ; 0068
        nop                     ; 006A
        in      r24, 0x2E       ; 006C
        andi    r24, 0xF8       ; 006E
        out     0x2E, r24       ; 0070
        nop                     ; 0072
        nop                     ; 0074
        ldi     r24, 0xA        ; 0076
        out     0x2E, r24       ; 0078
        nop                     ; 007A
        nop                     ; 007C
        nop                     ; 007E
        nop                     ; 0080
```

CONFIG TIMER1, ...

STOP TIMER1

START TIMER1

Register Summary

Address	Name	Page
0x2F (0x4F)	TCCR1A	108
0x2E (0x4E)	TCCR1B	111

7	6	5	4	3	2	1	0	
COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A

7	6	5	4	3	2	1	0	
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B

Slika 3: Analiza Timer/Counter1 ukazov

AX ELEKTRONIKA
GENERALNI ZASTOPNIK ZA SLOVENIJO

AX ELEKTRONIKA D.O.O.
POT HEROJA TRTNIKA 45,
1000 LJUBLJANA
TEL.: 01 549 14 00
INTERNET: WWW.SVET-EL.SI,
E-MAIL: STIK@SVET-EL.SI

BeeProg+ JE UNIVERZALNI PROGRAMATOR NASLEDNJE GENERACIJE, KI SE MU NADGRAJUJE LE PROGRAMSKA OPREMA IN DODAJA PODPORA NOVIM KOMPONENTAM. POSODOBITVE SO BREZPLAČNE IN SPROTI SLEDIJO DOGAJANJU NA PODROČJU PROGRAMABILNIH ELEKTRONSKIH VEZIJ.

BeeProg+

```
...
...
nop
nop
nop
nop
Config Timer1 = Timer ,_
  Prescale = 8 ,_
  Clear Timer = 1 ,_
  Compare A = Toggle
nop
nop
Stop Timer1
nop
nop
Start Timer1
nop
nop
nop
nop
End
```

Ukazi, s katerimi upravljamo z delovanjem časovnika Timer1, so prevedeni v le nekaj asemblerskih ukazov in brez klicev na podprograme, zato se zdi, da bo njihova analiza enostavna. Vendar, če želimo razumeti, kaj ti ukazi pravzaprav povzročijo, moramo dobro poznati pomen posameznih registrov AVR mikrokontrolerja. Ukaz Config Timer1 na primer vpiše neke vrednosti v registre na naslovih 0x2F in 0x2E. Da bi razumeli zakaj pri tem gre, moramo najprej ugotoviti, katera sta ta dva registra. Pogledati bo treba v tehnično dokumentacijo (data-sheet) uporabljenega mikrokontrolerja. V poglavju „Register Summary“ se nahaja popis vseh I/O registrov, njihovi naslovi, struktura in številka strani, na kateri so natančno opisani. Vidimo lahko, da se na prej omenjenih naslovih nahajata registra TCCR1A in TCCR1B in da se njun natančen opis nahaja na straneh 108 in 111 (to velja za ATtiny2313 data-sheet, Rev. 2543I–AVR–04/06). V opisih je pojasnjena vloga vsakega posameznega bita teh dveh registrov.

V tem trenutku nas ne zanima, na kakšen način bodo vrednosti, ki jih bomo vpisali v registra TCCR1A in TCCR1B, nastavila časovnik Timer1. Omenimo lahko samo to, da trije najmanj pomembni biti registra TCCR1B, namreč CS12-CS10, določajo vir taktnih impulzov, ki jih časovnik šteje. Če je njihova vsebina različna od „000“, bo časovnik začel teči in bo deloval v skladu s stanjem ostalih konfiguracijskih bitov v registrih TCCR1A in TCCR1B. Glede na to, da vsak ukaz Config Timer1 postavi CS12:CS10 v stanje, ki je različno od „000“, ga z vsakim konfiguriranjem istočasno tudi poženemo.

Naslednji ukaz, ki ga bomo analizirali, je Stop Timer1. Ta enostavno izbriše vsebino bitov CS12:CS10 in s tem ustavi časovnik, ne glede na to, kako je bil konfiguriran. Za njegov ponovni zagon je dovolj, če vrnemo bite CS12-CS10 v stanje, ki so ga imeli po konfiguraciji. Prav to naredi tudi ukaz Start Timer1, le da obnovi vrednost celotnega registra TCCR1B (s tem si je avtor programa Bascom-AVR prihranil nekaj ukazov). Ukaza Start in Stop ne spreminjata vrednosti TCCR1A registra. To dobro deluje in med izvajanjem programa lahko po mili volji ustavljate in ponovno poženete časovnike, vsaj dokler jim ne spremenite konfiguracije. Če med izvajanjem programa časovnik ponovno konfigurirate se lahko zgodi, da ga ukaz Start ne bo znal pognati po naših pričakovanjih prav zaradi tega, ker se ne ozira na vsebino registra TCCR1A. S tem problemom sem se srečal na začetku svojega programiranja z Bascom-AVR, njegov vzrok pa sem odkril ravno s pomočjo takšne analize. Da bi se izognili tem težavam, vam priporočam, da v zahtevnih programih namesto ukaza Start Timer1 ponovite ukaz Config Timer1. Seveda vse to velja tudi za ostale časovnike.

Na podoben način lahko analizirate tudi ostale ukaze, ki določajo način delovanja posameznih delov mikrokontrolerja. V to skupino spadajo na primer konfiguracija V/I (vhodno-izhodnih) priključkov (Config Pinx.y), zunanjih prekinitev (Config INTx) in Watchdoga (Config Watchdog), izbira faktorja deljenja sistemskega takta (Config Clockdiv), konfiguracija A/D pretvornika (Config ADC), omogočanje in onemogočanje prekinitev in podobno. Ti ukazi spreminjajo vsebino drugih registrov, za katere registre gre, pa lahko raziščete na enak način, kot smo to naredili v analiziranem primeru. Opomba: nekateri konfiguracijski ukazi (na primer Config Debounce ali Config 1wire) ne ustvarijo programske kode, ampak le vplivajo na to, kako se bodo prevedle posamezne Bascom-AVR rutine.

3. analiza: Naslavljanje I/O registrov in spremenljivk

Malo bolj zanimivo je raziskati, kako Bascom-AVR naslavlja V/I registre in uporabniške spremenljivke. V/I registri se v pomnilniku nahajajo takoj za registri za splošne namene (R0:R31), spremenljivke pa se shranjujejo v „interni SRAM“ , ki se začne na naslovu 60 heksadecimalno. Analizo bomo naredili z disasembliranjem naslednjega programa:

```
$crystal= 8000000
$regfile= „ATtiny2313.dat“

Dim V_byte As Byte, _
  V_word As Word

nop
nop
nop
nop
Counter0 = 100
nop
nop
Counter1 = 100
nop
nop
V_byte = 100
nop
nop
V_word = 100
nop
nop
nop
nop
End
```

V tem primeru se ista vrednost (100, oziroma 64 heksadecimalno) vpiše v 8-bitni in 16-bitni register, v spremenljivki tipa Byte in Word. Disasemblirani program je prikazan na sliki 4, poskusimo ga analizirati!

To, kar v programu Bascom-AVR imenujemo „Counter0“ , je pravzaprav števni register TCNT0 8-bitnega časovnika Timer/Counter0. Vpis vrednosti v ta register se odvija v dveh

korakih:

- vrednost se najprej vpiše v register R24 (v tem primeru gre za konstanto, vendar lahko prav tako vpišemo vrednost katere izmed spremenljivk),
- vrednost iz registra R24 se potem prenese v TCNT0 z asemblerskim ukazom OUT.

Enak postopek smo videli že v prejšnjem poglavju, le da je tam šlo za druge registre. Malo drugače je v primeru, ko želimo vpisati vrednost v Counter1. To, kar v programu Bascom-AVR imenujemo „Counter1“, se nanaša na registra TCNT1H in TCNT1L časovnika Timer/Counter1. Ta dva registra delujeta kot en sam 16-bitni števec, vendar ju mikrokontroler in asembler obravnavata ločeno. Disasemblirani program nam lepo predstavi, da je Bascom-AVR prevajalnik v registrih R25 in R24 najprej pripravil želeno vrednost 100, potem pa njuno vsebino prenese v registra TCNT1H in TCNT1L.

Zanimivi pri tem sta dve stvari. Namesto OUT, je tukaj Bascom-AVR za vpis vrednosti v V/I register uporabil asemblerski ukaz STS. Obe možnosti sta enakovredni in imata enak učinek; naj vas ne zmede dejstvo, da prevajalnik včasih uporabi eno, drugič drugo možnost. Bodite pozorni na to, da so uporabljeni različni naslovi: OUT vidi TCNT1H register znotraj naslovnega prostora, ki ga uporabljajo V/I registri (naslov = 0x2D), STS pa ga vidi kot del SRAM naslovnega prostora (naslov = 0x4D). V poglavju „Register Summary“ data-sheet-a so našteti vsi naslovi vseh registrov: da bi jih razlikovali, so SRAM naslovi zapisani z oklepaji. Edina razlika med ukazoma OUT in STS (pri takšni uporabi) je, da se OUT izvede v enem, za STS pa sta potrebna dva ciklusa. Če je za nas pomembna

```
$crystal = 8000000
$regfile = "ATtiny2313.dat"
Dim V_byte As Byte , V_word As Word

nop
nop
nop
nop
V_byte = 100
nop
nop
V_word = 100
nop
nop
COUNTER0 = 100
nop
nop
Counter1 = 100
nop
nop
nop
nop
End
```

BASCOM-AVR Report

Report | Error

Variable	Type	Address(hex)
TCNT1	Word	004C
ERR	Bit	0006
V_BYTE	Byte	0060
V_WORD	Word	0061

Asm_4.txt - Notepad

File Edit Format View Help

```
nop                     ; 005C
nop                     ; 005E
ldi     r24, 0x64       ; 0060
out     0x32, r24       ; 0062
nop                     ; 0064
nop                     ; 0066
ldi     r24, 0x64       ; 0068
ldi     r25, 0x0        ; 006A
sts     0x004D, r25     ; 006C
sts     0x004C, r24     ; 0070
nop                     ; 0074
nop                     ; 0076
ldi     r24, 0x64       ; 0078
sts     0x0060, r24     ; 007A
nop                     ; 007E
nop                     ; 0080
ldi     r24, 0x64       ; 0082
ldi     r25, 0x0        ; 0084
ldi     r26, 0x61       ; 0086
st      X+, r24         ; 0088
st      X, r25          ; 008A
nop                     ; 008C
nop                     ; 008E
```

COUNTER0 = 100

COUNTER1 = 100

V_BYTE = 100

V_WORD = 100

Address	Name		Page
0x32 (0x52)	TCNT0	Timer/Counter0 (8-bit)	81
0x2D (0x4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte	112
0x2C (0x4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte	112

Slika 4: Naslavljanje V/I registrov in spremenljivk

hitrost izvajanja programa, bo OUT seveda boljša rešitev.

Druga zanimivost je ta, da je podatek najprej vpisan v „višji“ register para registrov (v tem primeru je to TCNT1H) in šele za njim v „nižjega“ (TCNT1L). Vrstni red vpisa ni poljuben, ampak je določen z načinom, kako mikrokontroler zapisuje vsebino 16-bitnih registrov. Vsak 16-bitni časovnik ima po en 8-bitni pomožni register, kamor se začasno shranjuje vsebina višjega bajta med vpisom ali branjem 16-bitnega podatka. Pisanje v višji register sploh ne prenese vrednosti v ta register, ampak jo začasno zapiše v pomožni register. Šele pisanje v nižji register aktivira oba vpisa, zato se nižji register zapiše direktno, istočasno pa se vsebina iz pomožnega prenese v višji register. Tako se ves 16-bitni podatek zapiše v enem ciklu, čeprav sta ga zapisala dva ukaza. Branje vrednosti 16-bitnega registra se odvija na podoben način, vendar po obratnem vrstnem redu. Najprej moramo prebrati nižji register, ker bo mikrokontroler vsebino višjega registra istočasno prenesel v pomožni register. Branje višjega registra se potem izvede z branjem vrednosti iz pomožnega registra. Pomembno je vedeti, da prevajalnik Bascom-AVR že sam po sebi upošteva to in podobne lastnosti AVR mikrokontrolerjev.

Sedaj pa poglejmo, kako Bascom-AVR naslavlja spremenljivke. Te so shranjene v naslovnem prostoru internega SRAM-a, katerega prva lokacija se nahaja na naslovu 60 heksadecimalno. V našem primeru sta definirani dve spremenljivki:

- V_byte je 8-bitna spremenljivka na naslovu 60,
- V_word je 16-bitna spremenljivka na naslovih 61 in 62.

Naslove na katerih se nahajajo posamezne spremenljivke lahko vidimo v oknu Report programa Bascom-AVR , ki je prav tako prikazan na sliki 4. Za dostop do spremenljivk lahko Bascom-AVR uporablja neposredno ali posredno naslavljanje. V našem primeru se V_byte naslavlja neposredno z uporabo ukaza STS:

```
ldi   r24, 0x64      ; 0078
sts   0x0060, r24    ; 007A
```

Word, Integer, Long in druge spremenljivke, ki se zapišejo v več bajtih, pa običajno naslavlja posredno. V našem primeru je V_word naslovljen s posredovanjem indeksnega registra X:

```
ldi   r24, 0x64      ; 0082
ldi   r25, 0x0       ; 0084
ldi   r26, 0x61      ; 0086
st    x+, r24        ; 0088
st    x, r25         ; 008A
```

X register sestavlja par registrov R26, R27. V zgornjem primeru se za naslavljanje uporablja samo nižji bajt X-registra, R26. To je v primeru uporabe ATtiny2313 in tudi z vsemi drugimi mikrokontrolerji, ki imajo SRAM velikosti 128 bajtov ali manj. Če pišete program za mikrokontroler, ki ima večji SRAM, bo prevajalnik za indeksno naslavljanje uporabil tudi R27. Isti ukaz bi se v primeru uporabe mikrokontrolerja ATtiny4313 prevedel takole:

```
ldi   r24, 0x64      ; 0082
ldi   r25, 0x0       ; 0084
ldi   r26, 0x61      ; 0086
ldi   r27, 0x0       ; 0088
st    x+, r24        ; 008A
st    x, r25         ; 008C
```

4. analiza: Prekinitve in prekinitvene rutine

Poglejmo enostaven program, ki bo s pomočjo zunanjih prekinitev štel impulze na vhodnem priključku PD3:

```
$crystal= 8000000
$regfile= „ATtiny2313.dat“

Dim Pulse_counter As Byte

Config Pind.3 = Input
Config Int1 = Falling
On Int1 Int1_sub
Enable Interrupts
Enable Int1

Do
, delaj kar koli
Loop
End

Int1_sub:
   Incr Pulse_counter
Return
```

Ko ga prevedemo in disasembliramo, bo ta program videti tako, kot prikazuje slika 5. Prikazani so le tisti deli programa, ki so zanimivi za naše preučevanje. Na vrhu programa bomo opazili prekinitveni vektor, oziroma skok na prvi izvršilni ukaz prekinitvene rutine na naslovu 0070. Ta prekinitveni vektor je nastal s prevajanjem ukaza On Int1 Int1_sub. Čeprav vsebuje le ukaza Incr in Return, je prekinitveni podprogram preveden v dokaj obsežno kodo v zbirniku. Tisto, kar pripada ukazoma Incr in Return je le manjši del te kode in je na sliki zapisano odebeljeno. Ostalo je cela vrsta PUSH in POP ukazov, s katerima se na začetku izvajanja programa shrani na sklad, pred povratkom iz prekinitvene rutine pa obnavlja vsebina skoraj vseh registrov za splošne namene. Bascom-AVR prevajalnik to naredi iz previdnosti: ne ve, kateri registri bodo v prekinitveni rutini morda spremenjeni, zato shranjuje in obnavlja vse.

Rezultat ni samo obsežna koda, ampak je tudi izvajanje znatno daljše, kot bi bilo res potrebno: v tem primeru je za izvedbo prekinitvene rutine potrebno 116 ciklov, ob tem pa jih za ukaza Incr in Return porabimo samo 10. Seveda pa lahko vedno uporabite nosave možnost pri On Int1 ukazu in s tem onemogočite dodajanje PUSH ni POP ukazov, vendar morate v tem primeru sami skrbeti za shranjevanje in obnovitev uporabljenih registrov.

Optimiziranje Bascom-AVR programa z uporabo zbirnika

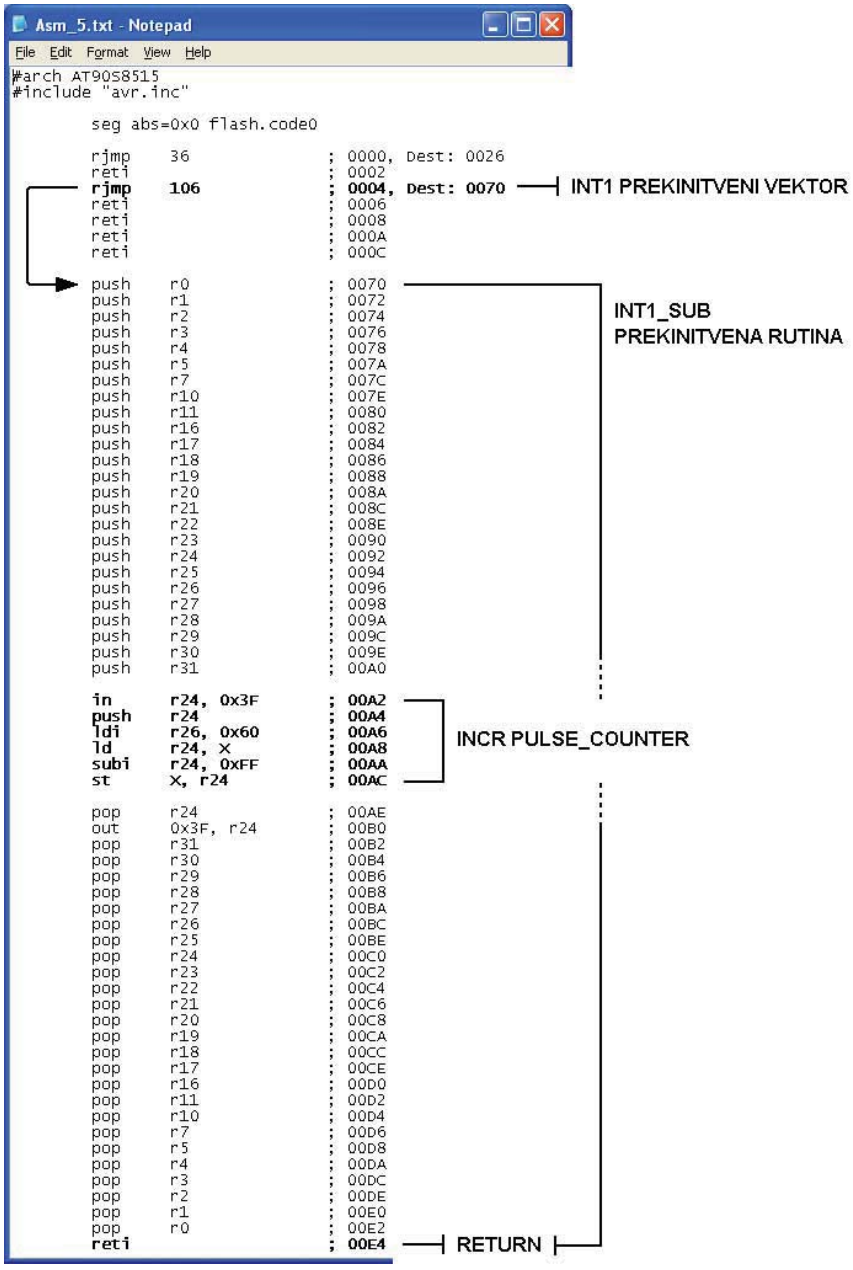
Zdaj pa smo končno že pripravljeni za nekaj izrednega: optimizirali bomo programsko kodo, ki jo je ustvaril prevajalnik programa Bascom-AVR, z zamenjavo posameznih Bascom-AVR ukazov ali celotnih delov programa z nekaj asemblerskimi ukazi! Incr ukaz iz našega primera uporablja le dva registra, R24 in R6, zato je dovolj, če shranimo le njuno vsebino. Dodatno moramo poskrbeti za statusni register SREG, na katerega vsebino vplivajo aritmetični, logični in bit-orientirani ukazi, kot tudi ukazi, s katerimi primerjamo vsebine registrov.

Seveda je potrebno SREG shraniti le v primerih, kadar bomo takšne ukaze uporabljali v prekinitveni rutini (kateri ukazi menjajo vsebino posameznih statusnih bitov - flags – si lahko ogledate poglavju Instruction Set Summary“ data-sheet-a uporabljenega mikrokontrolerja).

Naš program bomo dodatno optimizirali, če namesto posrednega uporabimo neposredno naslavljanje – takrat bo za povečanje vrednosti spremenljivke dovolj le en register, R24. Optimizirana prekinitvena rutina števca impulzov je videti takšna (prikazani so samo spremenjeni deli):

```
On Int1 Int1_sub Nosave

Int1_sub:
   push r24     'save R24
   in r24,sreg 'save SREG
   push r24
   lds r24,{pulse_counter}
           ' Incr _
             Pulse_counter
   inc r24
   sts {pulse_counter}
                'r24
pop r24
   !out sreg,r24
            ' restore SREG
   pop r24 ' restore R24
Return
```



Slika 5: Disasemblirana programska koda prekinitvene rutine

Razen tega, da je očitno krajša, je ta rešitev tudi več kot 6 krat hitrejša od izvirne programske kode Bascom-AVR – izvaja se le 19 ciklov! Opomba: pred asemblerskim OUT ukazom je napisan klicaj, ker je OUT rezervirana beseda programa Bascom-AVR-a. Več o tem lahko najdete v poglavju „Mixing ASM and BASIC“ Bascom-AVR help-a.

Podobno lahko naredimo tudi za primer, ko za štetje uporabljamo spremenljivko tipa Word. Za povečanje vrednosti spremenljivk tega tipa Bascom-AVR ustvari naslednjo programsko kodo (dobljena je z disasembliranjem) :

```
ldi   r26, 0x60 ; 00A6
rcall 6         ;00A8,_
                  Dest:00E2
...
...
ld    r30, X+       ; 00E2
ld    r31, X        ; 00E4
subi  r30, 0xFF     ; 00E6
sbci  r31, 0xFF     ; 00E8
```

```
st    X, r31          ; 00EA
st    -X, r30         ; 00EC
ret                   ; 00EE
```

Bascom-AVR za to opravilo uporablja tri registre (ali celo 4 za mikrokontrolerje s SRAM pomnilnikom, večjim od 128 B) in posredno naslavljanje, kliče pa tudi podprogram, ki mu postori nekaj dela. Veliko od tega je mogoče tudi optimizirati! Optimizirana rutina, ki uporablja neposredno naslavljanje in povečanje vrednosti spremenljivke tipa Word se izvede s pomočjo enega samega registra, kar je videti takole:

```
lds r24,{pulse_counter},_
      'Incr Pulse_counter
subi r24,$ff
sts {pulse_counter},r24
lds r24,{pulse_counter+1}
sbci r24,$ff
sts {pulse_counter+1},r24
```

Če vas bega, zakaj se za povečanje vrednosti registra uporabljajo ukazi (inštrukcije) za odštevanje, si natančno oglejte nabor ukazov AVR mikrokontrolerja: v njem ni ukaza seštevanja, s katerim bi lahko istočasno sešteli vsebino registra, konstanto in carry bit. SBCI zna ravno to in če je vrednost konstante, ki jo odštevamo FF heksadecimalno, dobimo enak učinek, kot bi vrednosti registra dodali 1. Morda ta rutina ni videti bistveno krajša od prvotne, vendar je znotraj prekinitvene rutine bolj ekonomična, ker je potrebno shraniti le vrednosti dveh registrov, R24 i SREG. Celotne optimizirane prekinitvene rutine in njihovi prevodi so prikazani na sliki 6.

```
Dim Pulse_counter As Byte
On INT1 Int1_sub Nosave

Int1_sub:
   push r24
   in r24,SREG
   push r24
   lds r24,{pulse_counter}
   inc r24
   sts {pulse_counter},r24
   pop r24
   !OUT SREG,r24
   pop r24
Return
```

```
Dim Pulse_counter As Word
On INT1 Int1_sub Nosave

Int1_sub:
   push r24
   in r24,SREG
   push r24
   lds r24,{pulse_counter}
   subi r24,$ff
   sts {pulse_counter},r24
   lds r24,{pulse_counter+1}
   sbci r24,$ff
   sts {pulse_counter+1},r24
   pop r24
   !OUT SREG,r24
   pop r24
Return
```

Asm_5.txt - Notepad

File Edit Format View Help

```
#arch AT90S8515
#include "avr.inc"

        seg abs=0x0 flash.code0

        rjmp    36              ; 0000, Dest: 0026
        reti                    ; 0002
        rjmp    106             ; 0004, Dest: 0070   <- INT1 PREKINITVENI VEKTOR
        reti                    ; 0006
        reti                    ; 0008

        push    r24             ; 0070
        in      r24, 0x3F       ; 0072
        push    r24             ; 0074
        lds     r24, 0x0060     ; 0076
        inc     r24             ; 007A
        sts     0x0060, r24     ; 007C
        pop     r24             ; 0080
        out     0x3F, r24       ; 0082
        pop     r24             ; 0084
        reti                    ; 0086
```

INT1_SUB PREKINITVENA RUTINA (BYTE)

Asm_5.txt - Notepad

File Edit Format View Help

```
#arch AT90S8515
#include "avr.inc"

        seg abs=0x0 flash.code0

        rjmp    36              ; 0000, Dest: 0026
        reti                    ; 0002
        rjmp    106             ; 0004, Dest: 0070   <- INT1 PREKINITVENI VEKTOR
        reti                    ; 0006
        reti                    ; 0008

        push    r24             ; 0070
        in      r24, 0x3F       ; 0072
        push    r24             ; 0074
        lds     r24, 0x0060     ; 0076
        subi    r24, 0xFF       ; 007A
        sts     0x0060, r24     ; 007C
        lds     r24, 0x0061     ; 0080
        sbci    r24, 0xFF       ; 0084
        sts     0x0061, r24     ; 0086
        pop     r24             ; 008A
        out     0x3F, r24       ; 008C
        pop     r24             ; 008E
        reti                    ; 0090
```

INT1_SUB PREKINITVENA RUTINA (WORD)

Slika 6: Optimizirana programska koda prekinitvenih rutin

Prekinitvene rutine so idealne za optimiziranje: vedno želimo, da so čim krajše in čim hitrejše. Optimizirate jih lahko z uporabo naslednjega recepta: » disasemblirajte Bascom-AVR program,

AX ELEKTRONIKA
GENERALNI ZASTOPNIK ZA SLOVENIJO

AX ELEKTRONIKA D.O.O.
POT HEROJA TRTNIKA 45,
1000 LJUBLJANA
TEL.: 01 549 14 00
INTERNET: WWW.SVET-EL.SI,
E-MAIL: STIK@SVET-EL.SI

SMARTPROG2 JE VSESTRANSKI 40-PINSKI PROGRAMATOR Z ISP IZBIRO IN Z USB POVEZAVO NA PC, KVALITETEN HARDVER JE NADPOLNJEN Z VRHUNSKIM PROGRAMOM, KI KOT STANDARDE PODPIRA VSE MS WINDOWS OPERACIJSKE SISTEME (OD WIN.95 DO WIN7). PROGRAMIRA VEČ KOT 24100 RAZLIČNIH MIKROKONTROLERJEV.

SmartProg2
universal 40-pindrive programmer

- poiščite del programa, ki pripada prekinitveni rutini (to ni težko, obkrožen je z veliko PUSH-i in POP-i),
- če imate v svojem programu več kot eno prekinitveno rutine, poglejte prekinitvene vektorje na začetku programa, da bi ugotovili, na katerem naslovu se začne prekinitvena rutina, ki jo želite optimizirati
- ugotovite, kateri registri se uporabljajo v prekinitveni rutini in v podprogramih, če jih ta kliče,
- dodajte nosave možnost pri On interrupt ukazu,
- dodajte PUSH ukaz za vsak uporabljeni register (ne pozabite na SREG!) čisto na začetku prekinitvene rutine,
- dodajte POP ukaze za iste registre (vendar v obratnem vrstnem redu) preden se vrnete iz prekinitvene rutine,
- ponovno prevedite program.

Takšna sprememba programa ne zahteva poznavanja programiranja v asemblerju; dovolj je, da prepoznamo ukaze, ki uporabljajo registre in napisati PUSH in POP ukaze za te registre. Primeri na sliki 6 prikazujejo, kako to naredimo za R24 in SREG (ne pozabite na tega zadnjega, saj ga v večini ukazov ne vidimo, lahko pa prav ti ukazi spremenijo njegovo vsebino). V naslednjem koraku optimizacije lahko zamenjate kritične Bascom-AVR rutine z ustreznimi asemblerskimi deli programa. Prednost tega postopka je, da bo morda vaša koda bolj elegantna in krajša od originalne in da ne bo več odvisna od uporabljene različice Bascom-AVR prevajalnika (način prevajanja posameznih Bascom-AVR ukazov se lahko spremeni, če uporabimo drugo različico prevajalnika). To je lahko koristno ne le v prekinitvenih rutinah, ampak vedno, kadar želite imeti popoln nadzor nad ustvarjeno programsko kodo. Seveda takšno poseganje v program zahteva dobro poznavanje arhitekture uporabljenega mikrokontrolerja in programiranja v asemblerju.

Še več informacij lahko najdete na naslednjih virih:

- Bascom-AVR help, poglavja „Language Fundamentals“, „Assembler mnemonics“ in „Mixing ASM and BASIC“,
- AVR instruction set si lahko naložite s spletnega naslova www.atmel.com;
- data-sheet uporabljenih mikrokontrolerjev si tudi lahko naložite s spletnega naslova www.atmel.com.

www.svet-el.si

Elektronika za začetnike - Programiranje mikrokontrolerjev III (27)

Avtor: Bojan Kovač
E-pošta: bojan@svet-el.si

Mikrokontroler je računalnik v malem, saj ima poleg programskega in delovnega pomnilnika ter EEPROMA lahko vgrajeno še celo vrsto internih perifernih »naprav«, s katerimi komunicira interno, rezultate pa uporabi kot spremenljivke v katerem od podprogramov ali pa jih posreduje kot rezultat na svoje izhodne priključke.

ELEKTRONIKA ZA ZAČETNIKE

Prek svojih vhodov je sposoben brati tudi zunanjo vhodno periferijo, prek izhodov krmiliti izhodno periferijo, v notranjosti pa izvajati logične in računske operacije. Zdaj že vemo, da različna stanja na vhodih prožijo izvajanje različnih programov, njihove končne rešitve pa mikrokontroler »objavi« na svojih izhodih. Poleg internih, vgrajenih perifernih naprav imamo lahko tudi zunanje (eksterne), ki jih mikrokontrolerju dodamo na ploščico tiskanega vezja (TIV) in z njimi potem komunicira prek vhodno/izhodnih priključkov.

Notranje periferne naprave

Nabor notranje periferije posameznega mikrokontrolerja je odvisen od tega, za kakšen je namen je bil izdelan. Vedeti je treba tudi to, da ima posamezen mikrokontroler lahko veliko izpeljank, ki se razlikujejo v glavnem le po naboru vgrajene periferije. Kaj vse spada k internim perifernim napravam? Poglejmo na sliko 1 in vse nam bo takoj jasno! Naj omenim le še to, da podobno velja za mikrokontrolerje vseh proizvajalcev, ne le družine AVR proizvajalca Atmel, ki jo obravnavamo v članku.