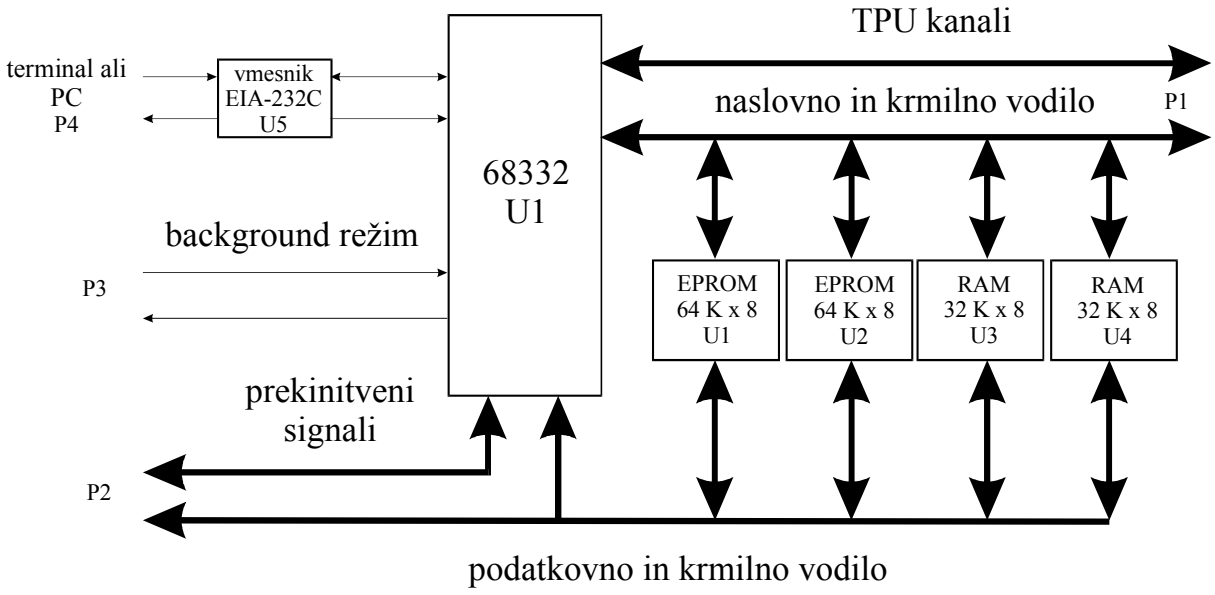
8. MC68332 V BCC IZVEDBI

V tem poglavju bomo pokazali praktično izvedbo arhitekture mikrokrmilniškega sistema z nekaterimi osnovnimi podsestavi. Takšne elementarne sisteme, ki imajo splošno namembnost ponujajo skorajda vsi proizvajalci procesorjev. V primeru MC68332 govorimo o BCC-ju (angl. Business Card Computer - računalnik velikosti vizitke), ki skupaj s PFB (angl. Platform Board) sestavlja razvojni modul (angl. Evaluation Module). Podobni sistemi so na voljo tudi pri ostalih procesorjih (npr. EVM DSP-ja TMS320F240, pogl. 11). Taki aparaturni opremi je prilagojena tudi programska oprema, ki omogoča najosnovnejše delo s sitemom.

8.1 BCC

BCC je sistem, ki na tiskanem vezju dimenzij 90 mm x 60 mm poleg mikrokrmilnika MC68332 vsebuje še nekatera dodatna vezja in elemente (slika 8.1):

- 2 x 64 K byte EPROM pomnilnik z debug monitor programom (glej poglavje 8.2),
- 2 x 32 K byte RAM pomnilnik,
- vmesnik za serijsko komunikacijo EIA-232,
- dva konektorja po 64 nožic.

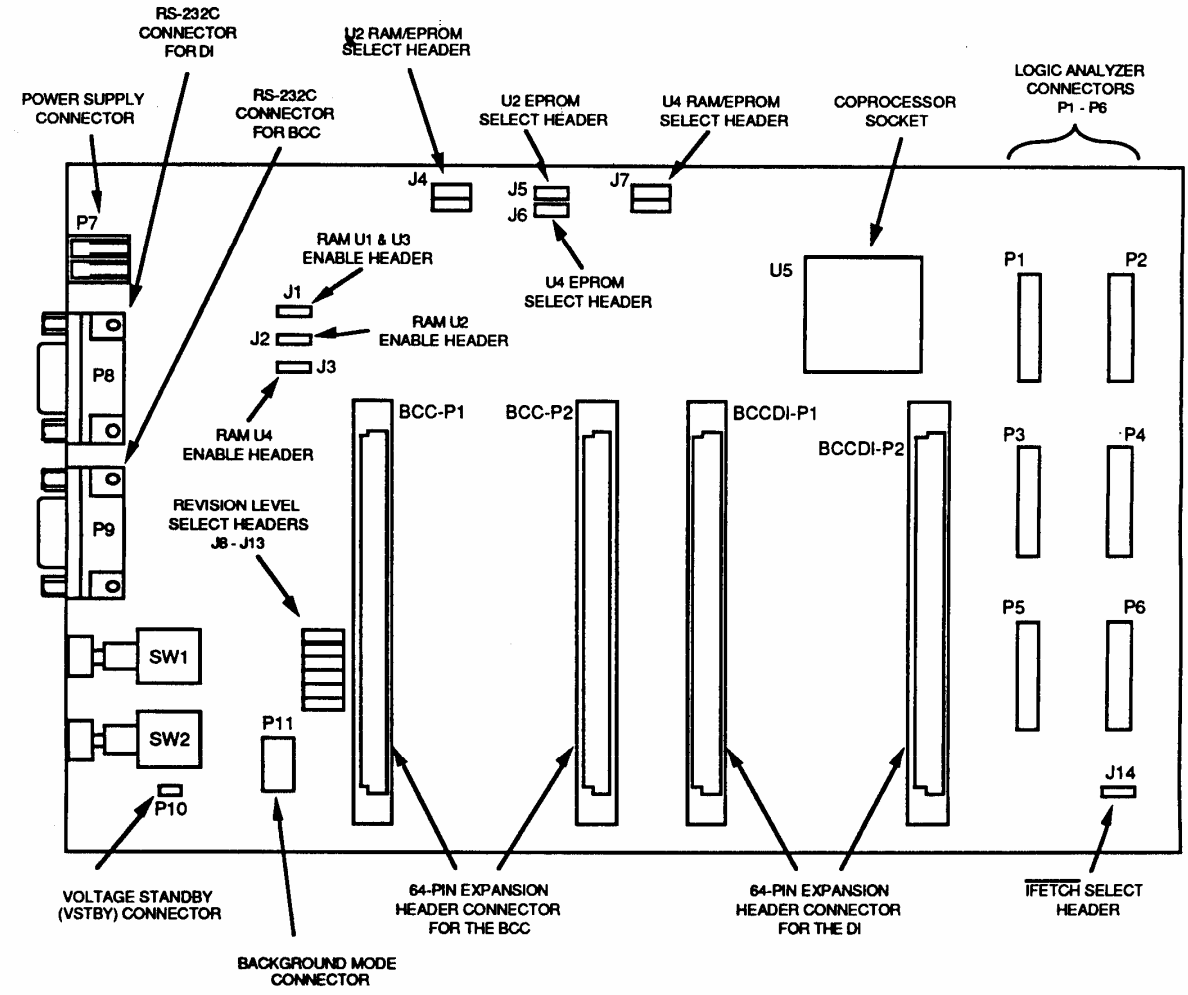


Slika 8.1: Blokovna shema BCC za MC68332

BCC lahko vgradimo v večji mikroračunalniški sistem, kar omogočata konektorja na robovih ploščice. Prek njiju so dostopne vse nožice mikrokrmilnika. Nadgraditev BCC je enostavna: na tiskanem vezju, ki vsebuje poljubne dodatne elemente, je treba predvideti konektorja, ki sta združljiva s konektorjema na BCC-ju. Iz tega sledi, da lahko BCC uporabimo za vrsto različnih nalog, potrebno je le projektirati nosilno ploščo z dodatnim okoljem (dodatni pomnilniki, A/D in D/A pretvorniki, prilagodilna vezja itd.).

Tudi Motorola ponuja tiskano vezje, ki je aparaturna podpora BCC-ju. Imenujejo ga Platform Board - PFB. Slika 8.2 kaže skico PFB-ja. Osnovni elementi so:

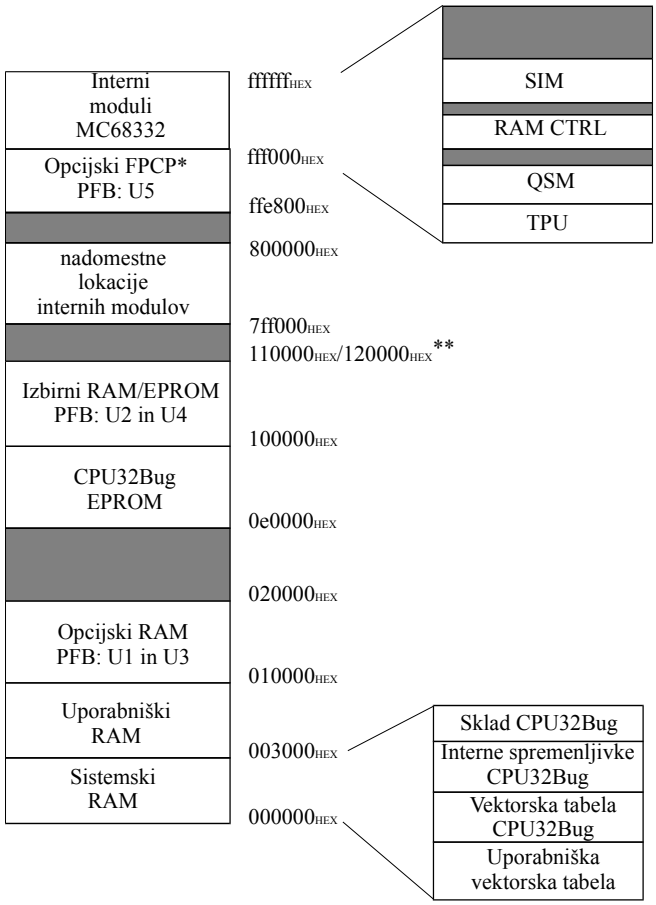
- priključne sponke za napajalno napetost 5 V z varovalko in stabilizacijskim vezjem,
- dva konektorja z devetimi nožicami za priključitev na serijski port (EIA232) osebnega računalnika (konektor TERMINAL je povezan s SCI-jem BCC-ja),
- podnožja za dodatne EPROM in RAM pomnilnike in matematični koprocesor (angl. floating point processor - FPP),
- konektorja za priključitev BCC-ja,
- konektorja za priključitev M68300DI (angl. Development Interface),
- premostitve (angl. jumper) za izbiro med različnimi arhitekturnimi inačicami (npr. izbira tipa pomnilnika - RAM ali EPROM - na določenem podnožju),
- številni posebni konektorji z dvajsetimi nožicami, ki omogočajo dostop do vseh signalov BCC-ja za priključitev logičnega analizatorja (signali so razvrščeni v različne skupine, npr. naslovno vodilo, podatkovno vodilo, TPU kanali itd.).
- Tipki za resetiranje (SW1) in prekinitev izvajanja programa (abort - SW2). Prva tipka je neposredno povezana s nožico RESET prekinitve, druga pa z IRQ7.



Slika 8.2: Skica PFB-ja

8.1.1 Pomnilniška mapa MC68332 na BCC-ju

Pomnilniki na BCC-ju ter morebitni dodatni pomnilniki na PFB-ju in matematični koprocesor se nahajajo na vnaprej določenih naslovih. S tem je del področja naslavljanja mikrokrmilnika MC68332 že rezerviran (celotno naslovljivo področje z naslovnimi nožicami A23 - A0 je 2^{24} = 16 777 216 bytov). Prikaz zasedenosti naslovljivega področja imenujemo *pomnilniška mapa* (angl. memory map, slika 8.3). Dodatne naslovljive elemente v končni aplikaciji (pomnilnike, A/D in D/A pretvornike...) lahko postavimo le na prosta področja. Pomnilniška mapa velja za najnovejšo inačico BCC-ja in se nekoliko razlikuje od starejših verzij.



*FPCP: matematični koprocesor (angl. Floating Point Coprocessor)
**odvisno od tipa pomnilnika

Slika 8.3: Pomnilniška mapa BCC

8.2 Program debug monitor

Večina mikrokrmilnikov vsebuje številne podsestave, ki zadostujejo vsaj za osnovno delovanje (CPU, enota za komunikacijo z nadrejenim računalnikom, pomnilnik, vhodno/izhodne enote). Kljub temu samega integriranega vezja po priključitvi napajanja ne

moremo takoj uporabljati. Mikroračunalnik (ali katerikoli drug mikroprocesorski element) se obnaša kot "tabula rasa"; nihče mu ni povedal kako naj uporablja svoje zmogljivosti. Določitev osnovnih začetnih parametrov je glavna naloga *inicializacijskega postopka*. To je del podprograma, ki se izvaja avtomatično ob vklopu ali po *resetiranju* procesorja. Aktiviranje RESET vhoda generira prekinitev najvišje prioritete, ki je ni možno maskirati. Ob tem se samodejno začne izvajati prekinitveni program, katerega naslov kaže vektor 1 (pomnilniška lokacija 00 0004_{HEX}), kazalec sklada (SP) za ta podprogram pa je zapisan v vektorju 0 (pomnilniška lokacija 00 0000_{HEX}, glej tudi poglavje 7.2.2.2).

Med tipične naloge, ki jih je treba izvršiti med inicializacijo, sodi določanje:
- parametrov serijskega podsestava za asinhronsko komunikacijo z nadrejenim računalnikom (PC) ali terminalom (hitrost prenosa, število bitov komunikacijskega paketa, tip paritete, število stop bitov itd.),
- začetnega stanja vseh registrov (npr. stanje SP-ja in PC-ja ob zaključku izvajanja inicializacije) in nožic,
- sistemskih prekinitvenih podprogramov (npr. procedura, ki se izvaja ob naslavljanju neobstoječe spominske lokacije) itd.

Vse inicializacijske funkcije morajo biti shranjene na mediju, ki ohrani svojo vsebino tudi po izklopu napajanja. V ta namen običajno uporabljamo pomnilnike tipa ROM, največkrat EPROM, ki omogoča reprogramiranje inicializacijskega podprograma. Načeloma mora uporabnik pri projektiranju mikroprocesorskega vezja postaviti pomnilnik z inicializacijskim programom na lokacijo, ki ustreza RESET vektorju, tako da se program samodejno požene ob vklopu ali resetiranju.

Problem izbire EPROM-a pri MC68332 ob resetiranju je rešen s pomočjo posebnega signala CSBOOT, ki sodi v skupino CS nožic (glej poglavje 5.5). Na njo je vezan vhod G EPROM-a z inicializacijskim programom. Po RESET signalu se izmed vseh nožic za izbiro čipa aktivira samo CSBOOT (stanje "0"). Na ta način omogoči EPROM mikrokrmilniku dostop do svojih prvih osmih bytov (RESET SP in PC). Vsebina prekinitvenega vektorja 1 (nov PC) kaže na RESET prekinitveni program, ki se nahaja na istem EPROM-u.

ABORT tipka aktivira prekinitev IRQ7, ki le prekine izvajanje uporabniškega programa in vrne nadzor debug monitorskemu programu. Pri tem pa ABORT ne reinicializira sistema, kot to stori funkcija RESET.

8.2.1 Naloge debug monitor programa

Inicializacija je le ena od nalog *debug monitor* programa CPU32Bug v EPROM-u. Program je dobil ime po dveh svojih osnovnih funkcijah:
- "**debugging**" (slov. očiščevanje, razhroščevanje) pomeni preizkušanje, odkrivanje in odpravljanje napak v programu,
- **monitor** (slov. monitor, nadzorni program) je del programa, ki vsebuje spremenljivke, procedure za dostop in inicializacijsko kodo.

Program vsebuje ukaze za:

- prikaz in spreminjanje vsebine pomnilnikov,

- določanje prekinitev točk,
- asembliranje in deasembliranje,
- samodejno preverjanje funkcionalnosti sistema ob vklopu,
- interaktivni razhroščevalnik (angl. debugger),
- uporabniški vmesnik, ki sprejema ukaze s terminala oz. osebnega računalnika.

S tem je omogočeno delo z BCC-jem v t.i. *on-line režimu*, kjer lahko v RAM pomnilnik na BCC neposredno vpisujemo ali beremo ukaze in podatke, prenašamo dele vsebine pomnilnika na druge lokacije, zaženemo program, preverjamo stanje določenih registrov, vnašamo določene *prekinitvene točke* (angl. breakpoints) itd.

8.2.1.1 Dekodiranje S19 formata

V poglavju 9 je opisan celoten postopek programiranja uporabniškega programa v "off-line" režimu, to je na osebnem računalniku, brez neposredne povezave z mikroprocesorskim sistemom. Postopek se začne z izvorno kodo (v C jeziku ali zbirniku) in konča z ASCII datoteko, ki je primerna za prenos v RAM pomnilnik mikroprocesorskega sistema (angl. downloading). Format te datoteke za Motoroline procesorje imenujemo "S19" ali "HEX". Datoteko ne sestavlja le bodoča vsebina RAM-a v ASCII obliki (kodirani ukazi in podatki), temveč tudi pomožni podatki (naslov programa, začetne lokacije memorijskih paketov, število zlogov v vrstici, nadzorna vsota - checksum itd.). Seveda mora tudi sprejemnik S19 datoteke (mikroprocesorski sistem, v katerega nalagamo datoteko, npr. BCC) ločiti med delom, ki ga je treba postaviti v RAM in "balastom", kar je ena od nalog debug monitor programa v BCC-ju.

8.2.1.2 Asembliranje in deasembliranje

"On-line" režim omogoča neposredno vpisovanje posameznih ukazov in podatkov na lokacije v RAM-u prek terminala. Vpisovanje podatkov je možno v desetiški ali dvojiški obliki, najpogosteje pa uporabljamo šestnajstiški način zapisa.

Vsebina RAM-a s CPU ukazi (spoznali smo jih v poglavju 4) se na prvi pogled sploh ne razlikuje od dela RAM-a, ki vsebuje podatke, saj so vsi zapisani v binarni kodi. Seveda gre pri ukazih za binarno strojno kodo, ki ustreza le določenemu tipu ukaza in izbranega načina naslavljanja operandov. Kodiranje ukazov s strani programerja in vpisovanje v RAM je možno, vendar je to zelo zahtevno in zamudno delo. Zato raje uporabljamo vrstični[1] *zbirnik* ali *asembler* (angl. assembler), ki je del debug monitorja. Ta program omogoča prevajanje CPU ukazov iz mnemotehnične oblike (t.i. *zbirni jezik*; assembly language) v ustrezno binarno kodo. Za čitanje ukazov rabimo *povratni zbirnik* ali *dezasembler* (angl. disassembler), ki binarno kodo ukaza prevede nazaj v mnemotehnično obliko.

[1] O vrstičnem (angl. line) zbirniku govorimo zaradi njegove lastnosti, da hkrati vnašamo oz. popravljamo le eno vrstico.

8.2.2 Debug monitorski ukazi

Funkcije debug monitor programa v EPROM-u BCC-ja izvajamo s pomočjo niza ukazov v mnemotehnični obliki. Teh ukazov ne gre v nobenem primeru mešati s CPU ukazi (Poglavje 4). Debug monitorski ukazi le pomagajo pri manipulaciji s programom, ki je sestavljen iz CPU ukazov. Slednji se po zagonu (prek debug monitorskega ukaza `GO`) izvajajo v realnem času.

V nadaljevanju bomo ob mnemoniku vsakega ukaza v oklepajih zapisali tudi dovoljeno skrajšano obliko. Ukaze lahko pišemo z velikimi ali malimi črkami. Tukaj bomo prikazali le nekatere ukaze in njihove osnovne različice.

Ukaz za pomoč: `HELP (HE)`

Ukaz `HELP` izpiše vse debug monitor ukaze s kratkim opisom njihovega pomena:

```
BC        Block Compare
BF        Block Fill
BM        Block Move
BR        Breakpoint Insert
NOBR      Breakpoint Delete
BS        Block Search
BV        Block Verify
DC        Data Conversion and Expression Evaluation
DU        Dump S-Records
GD        Go Direct (no breakpoints)
GN        Go and Stop after Next Instruction
GO        Go to Target Code
G         "Alias" for previous command
GT        Go and Insert Temporary Breakpoint
HE        Help Facility
LO        Load S-Records
MA        Macro Define/Display
NOMA      Macro Delete
MAE       Macro Edit
MAL       Enable Macro Expansion Listing
NOMAL     Disable Macro Expansion Listing
MD        Memory Display
MM        Memory Modify
M         "Alias" for previous command
MS        Memory Set
OF        Offset Registers
PA        Printer Attach
NOPA      Printer Detach
PF        Port Format
RD        Register Display
RESET     Warm/Cold Reset
RM        Register Modify
RS        Register Set
SD        Switch Directory
T         Trace Instruction
TC        Trace on Change of Flow
TM        Transparent Mode
TT        Trace to Temporary Breakpoint
VE        Verify S-Records
```

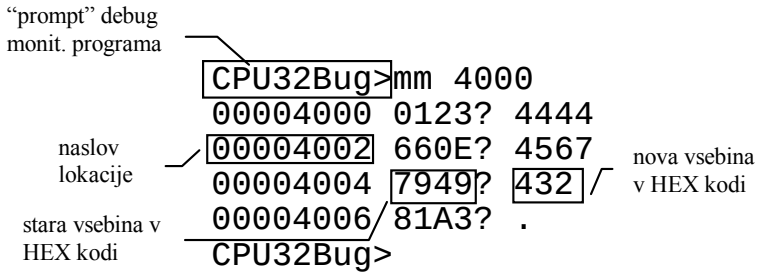
Ukaz za vpis na pomnilniško lokacijo: `MM`

Vsebina pomnilniške lokacije je lahko podatek ali CPU ukaz. Oba lahko zapišemo v šestnajstiški kodi. Pri ukazih je to njihova kodirani zapis, pri podatkih pa številčna vrednost. Po drugi strani je CPU ukaze elegantneje vpisati v mnemotehnični obliki.

Oba načina vpisa dosežemo z debug monitorskim ukazom `MM` (angl. memory modify), ki mu sledi naslov besede v RAM-u. Npr. ukaz

```
MM 4000
```

bo izpisal vsebino besede z naslovom 4000_{HEX} (vsi naslovi so po definiciji zapisani v šestnajstiški obliki) ter omogočil vpis nove vrednosti v šestnajstiški kodi. Po zapisu nove vsebine in pritisku na tipko CR (ENTER) se izpiše vsebina naslednje lokacije in omogoči njena sprememba itd. Režim vpisa zapustimo tako, da vpišemo piko in CR (ENTER):



Pri vpisovanju ukazov raje uporabimo opcijo `di` (angl. disassembler) za podpičjem, ki dezasemblira vsebino določenega naslova ter omogoči vpis novega ukaza v mnemotehnični obliki[2]:

```
CPU32Bug>mm 4000;di
00004000 4E71                nop
00004002 D280                add.l       d0,d1
00004004 04328lA3 0411       SUBI.B      #$A3,$11(A2,D0.W*4)? .
CPU32Bug>
```

Seveda je deasembliranje spominske lokacije smiselno le, če se na njej dejansko nahaja nek ukaz. V nasprotnem primeru bo rezultat dekodiranja ukaz, ki ni bil predviden na tej lokaciji. Debug monitor je vsebino lokacij od 4000 do 4008 iz prvega primera dekodiral kot ukaz

```
SUBI.B      #$A3,$11(A2,D0.W*4).
```

[2] Takoj po vpisu novega ukaza se stara vsebina lokacije zbriše. Zato po pritisku na CR (ENTER) stara vsebina ni več vidna.

Ukaz za izpis vsebine pomnilniške lokacije: MD

Ukaz izpiše vsebine nekaj spominskih lokacij (odvisno od opcije). Hkrati za vsako lokacijo izpiše tudi pomen vsebine v ASCII kodi (če kombinacija bitov sploh ustreza veljavnemu znaku):

```
CPU32Bug>md 4000
00004000 4E71 D280 0432 81A3 0411 4504 FFFB 9C34   NqR..2.#..E..{.4
```

Lahko uporabimo tudi opcijo `di`, podobno kot pri ukazu `MM`:

```
CPU32Bug>md 4000;di
00004000 4E71                NOP
00004002 D280                ADD.L       D0,D1
00004004 043281A3 0411       SUBI.B      #$A3,$11(A2,D0.W*4)
0000400A 4504                CHK.L       D4,D2
0000400C FFFB                DC.W        $FFFB
0000400E 9C340010            SUB.B       $10(A4,D0.W),D6
00004012 84A887EF            OR.L        -$7811(A0),D2
00004016 4EB09811            JSR         $11(A0,A1.L)
CPU32Bug>
```

Hkrati z ukazi se izpišejo tudi kode ukazov (v HEX zapisu).

Ukaza za izpis in spremembo vsebine registrov: RD in RM

Ukaza sta podobna prejšnjima ukazoma. V tem primeru izpišemo (`RD` - register display) ali spremenimo (`RM` - register modify) vsebine registrov CPU-ja (programskega števca - PC, statusnega registra - SR, podatkovnih in naslovnih registrov itd.). Primer ukaza `RD`:

```
CPU32Bug>rd
PC   =00003000 SR   =2700=TR:0FF_S_7_.....   VBR  =00000000
SFC  =5=SD     DFC  =5=SD     USP  =0000FC00 SSP* =00010000
D0   =00000000 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00010000
00003000 0000F3B3            ORI.B       #$B3,D0
CPU32Bug>
```

Ukaz za zagon programa: GO (G)

Program, ki se nahaja v RAM ali ROM pomnilniku, poženemo z debug monitorskim ukazom `GO`, ki mu sledi naslov začetne lokacije. Npr. ukaz

```
GO 4000
```

bo pognal program, ki se nahaja na lokaciji 4000_{HEX}. V nadaljevanju se izvajajo CPU ukazi v realnem času. Iz tega režima se nadzor vrne debug monitorju v naslednjih primerih:

- PC kaže na lokacijo, ki ne vsebuje veljavnega ukaza,
- vsebina PC-ja se izenači s prekinitveno točko (glej naslednja ukaza),

- program se izvaja v *trace* režimu,
- izvajanje prekinemo z RESET ali ABORT prekinitvijo (tipki na PFB-ju).

Ukaz GO brez naslova začne izvajati program na lokaciji, na katero trenutno kaže PC.

Ukaz za vnos ali brisanje prekinitvene točke: BR in NOBR

Pri razvoju je včasih zelo koristno prekinjati izvajanje programa v izbranih točkah, ki jim rečemo *prekinitvene točke* (angl. breakpoints). Točke prekinitve določamo s pomočjo debug monitor ukaza BR, ki mu sledi naslov ukaza, pri katerem bo prišlo do prekinitve, npr.

```
BR 4010
```

Debug monitor omogoča definiranje do šestnajstih prekinitvenih točk. Po zagonu programa (ukaz GO) izvaja CPU ukaze v realnem času, vse dokler se vsebina PC-ja ne izenači z naslovom ene izmed prekinitvenih točk. Nato se nadzor vrne debug monitorju; hkrati se izpišejo vsebine registrov CPU. Od te točke naprej lahko program ponovno zaženemo z ukazom GO (brez definiranja naslova; glej prejšnji ukaz), do morebitne nove prekinitvene točke.

Posamezne prekinitvene točke "brišemo" z ukazom NOBR (angl. no breakpoint), kateremu sledi njen naslov. Ukaz brez naslova zbriše vse prekinitvene točke.

Primer:

V RAM-u se nahaja naslednji program z začetnim ukazom na lokaciji 3000_{HEX}:

```
00003000 203C0000 0123      MOVE.L      #$123,D0
00003006 D440               ADD.W       D0,D2
00003008 02020001           ANDI.B      #$1,D2
0000300C E556               ROXL.W      #$2,D6
0000300E 4E71               NOP
00003010 00004284           ORI.B       #$84,D0
```

Sledi definiranje prekinitvene točke in zagon programa:

```
CPU32Bug>br 3010
BREAKPOINTS
00003010
CPU32Bug>g 3000
Effective address: 00003000
At Breakpoint
PC   =00003010 SR   =2704=TR:OFF_S_7_..Z..   VBR  =00000000
SFC  =5=SD     DFC  =5=SD     USP  =0000FC00 SSP* =00010000
D0   =00000123 D1   =00000000 D2   =00000101 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00010000
00003010 00004284           ORI.B       #$84,D0
CPU32Bug>
```

Ukaz za zasledovanje: T

Prekinitvene točke omogočajo trenutno prekinjanje programa le v izbranih točkah. Včasih pa je zelo koristno opazovati posledice vsakega ukaza na stanje registrov. To nam omogoča ukaz `T` (angl. trace - zasledovanje). Z njim se izvrši le CPU ukaz, na katerega kaže PC. Pritisk na tipko CR (ENTER) ponovi izvajanje debug monitorskega ukaza `T`, le da sedaj kaže PC že na naslednji CPU ukaz. Z vsakokratnim pritiskom na tipko CR se bo ukaz `T` ponovil:

```
CPU32Bug>t
PC   =00003006 SR   =2700=TR:0FF_S_7_.....   VBR  =00000000
SFC  =5=SD     DFC  =5=SD     USP  =0000FC00 SSP* =00010000
D0   =00000123 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00010000
00003006 D440                ADD.W       D0,D2
CPU32Bug>
PC   =00003008 SR   =2700=TR:0FF_S_7_.....   VBR  =00000000
SFC  =5=SD     DFC  =5=SD     USP  =0000FC00 SSP* =00010000
D0   =00000123 D1   =00000000 D2   =00000123 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00010000
00003008 02020001            ANDI.B      #$1,D2
CPU32Bug>
PC   =0000300C SR   =2700=TR:0FF_S_7_.....   VBR  =00000000
SFC  =5=SD     DFC  =5=SD     USP  =0000FC00 SSP* =00010000
D0   =00000123 D1   =00000000 D2   =00000101 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00010000
0000300C E556                R0XL.W      #$2,D6
CPU32Bug>
```

Ukaz za nalaganje datotek S19 z nadrejenega računalnika: `LO`

Ukaz `LO` (angl. load - naloži) uporabljamo za nalaganje datotek v S19 formatu, ki smo jih ustvarili v "off-line" režimu (pogl. 9) z nadrejenega računalnika v RAM pomnilnik mikrokrmilnika. Postopek je naslednji:

- z ukazom `LO` pripravimo BCC na sprejem S19 datoteke,
- s komunikacijskim programom na nadrejenem računalniku pošljemo datoteko S19 prek serijskih vrat,
- po končanem prenosu pritisnemo dvakrat tipko CR (ENTER); debug monitor prevzame nadzor.

8.3 Komunikacija med terminalom (PC-jem) in BCC-jem

Uporabnik v "on-line" režimu komunicira z debug monitorjem BCC-ja prek PC ali Apple Macintosh računalnika[3]. Pri tem morata biti izpolnjena dva pogoja (glej tudi pogl. 6.3 o SCI):

- serijska vrata (SCI) BCC-ja morajo biti konfigurirana (naloga debug monitorja) in
- računalnik mora delovati kot terminal.

Slednje dosežemo s pomočjo programov za emulacijo (posnemanje) terminala, kot so npr. ProComm, Kermit, Hyper Terminal itd., za PC kompatibilne računalnike in npr. Mac Terminal za Apple Macintosh računalnike. V terminalskem programu moramo nastaviti takšne parametre, ki ustrezajo konfiguraciji SCI porta na BCC-ju. Po definiciji veljajo naslednji parametri (izkušeni programerji lahko te parametre spremenijo z reprogramiranjem EPROM pomnilnika):

- hitrost prenosa: 9600 bps,
- 8 bitov na znak,
- pariteta: ne,
- stop biti: 1.

[3] Na istem računalniku lahko delamo tudi v "off-line" režimu, na BCC se priključimo pri prenosu S19 datoteke in zagonu ali testiranju programa.