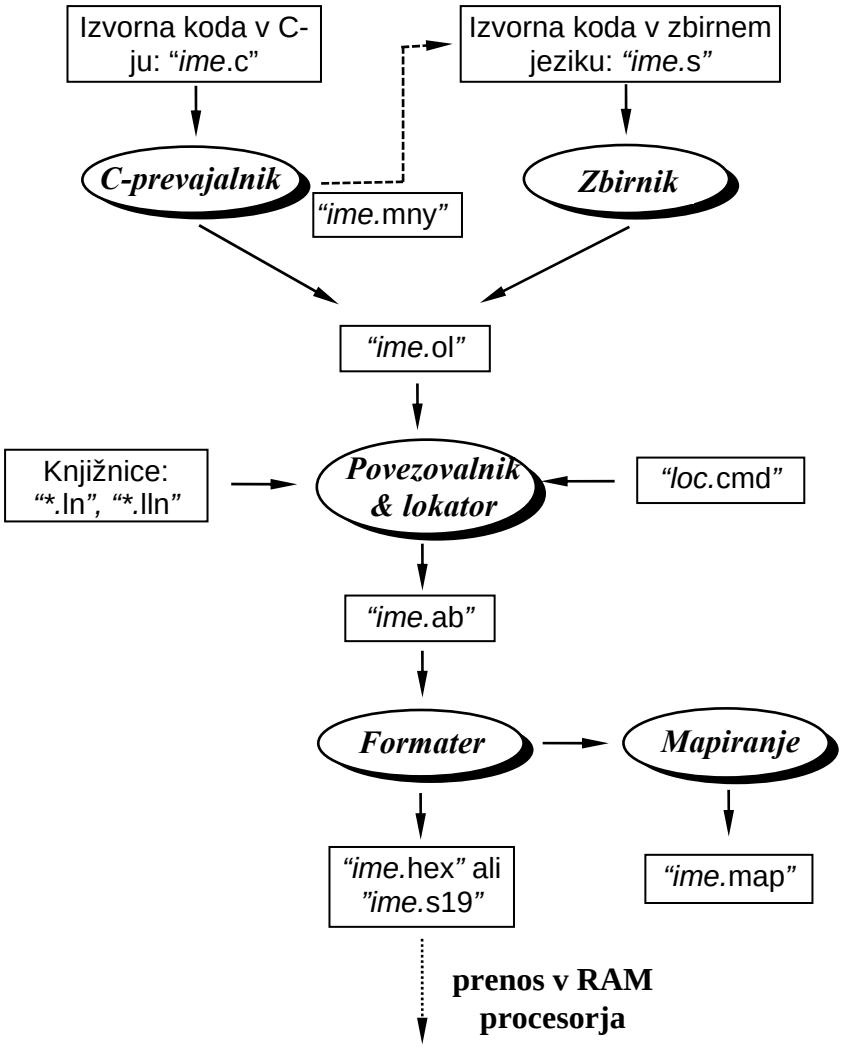
9. RAZVOJNA PROGRAMSKA OPREMA

V predhodnem poglavju smo si ogledali dva osnovna načina pisanja programov za mikroprocesorske sisteme:

- neposredno na mikrokrmilniku (on-line) in
- neodvisno od procesorja (off-line) na osebnem ali kakem drugem računalniku.

V tem poglavju bomo pokazali programska orodja, ki so na razpolago pri pisanju in analiziranju programov, ki jih pišemo v "off-line" režimu. Čeprav se sestava softverske opreme različnih proizvajalcev nekoliko razlikuje, so si koraki celotne procedure zelo podobni. Na sliki 9. 1 je prikazan del programske opreme firme "Intertools" za Motoroline mikroprocesorje serije 68XXX [33].



Slika 9. 1: Pomožna programska oprema za delo z mikroprocesorjem

9.1 Prevajanje programa

Ne glede na to, ali se odločimo za pisanje programa v jeziku C ali v zbirniku, je postopek zelo podoben. V prvi fazi napišemo na osebnem računalniku t.i. "*izvorni program*" ali "*izvorno kodo*" (angl. source code). Tak program pišemo v navadnem urejevalniku besedil in ga tudi shranimo v obliki navadnega teksta (ASCII tekst)[1]. Izvorni program mora ustrezati vsem leksikografskim pravilom, ki jih C ali zbirnik poznata. Slika 9. 2 kaže primer zelo enostavnega programa v jeziku C.

```
/* Primer enostavnega izvornega programa v C jeziku */

int a=1,b,c,d;          /* deklaracija in inicializacija
                           spremenljivke */

main()                  /* glavni program */
{
        c = a + b;      /* sestevanje spremenljivk */

        d = 7 * b;      /* mnozenje neinicialialzirane
                        spremenljivke s konstanto*/

}                       /* konec programa */
```

Slika 9. 2: Primer izvornega programa v C jeziku

Naslednji korak je prevajanje v "*objektno kodo*" (angl. object code; datoteka s podaljškom ".ol"). Če je izvorni program pisan v C jeziku, ga prevajamo s pomočjo *C prevajalnika* (angl. compiler; od tod tudi žargonski izraz "kompajliranje"), iz zbirnega jezika pa ga prevajamo s pomočjo *zbirnika - asemblerja* (angl. assembling). Objektna koda je sestavljena iz kod CPU ukazov, ki smo jih obravnavali v poglavju 4. Poleg tega se v njej nahajajo še nekateri dodatni podatki, ki so potrebni pri nadaljnji obdelavi objektne kode. Objektno kod za izvorni program s slike 9. 2 kaže slika 9. 3. Sestavljena je iz segmentov, katerih pomen je nezahtevnemu programerju nerazumljiv. Z ozirom na to, da je to le vmesni korak, se na tem mestu ne bomo ubadali z razlago posameznih delov datoteke.

Na sliki smo posebej označili kodo ukaza `MOVE.L D0,D1`. Ta je del skupine ukazov v zbirnem jeziku, ki ustreza C ukazu "`c = a + b`;" (glej sliko 9. 4 in pogl. 9.1.1). V tej fazi mu še ni dodeljena konkretna lokacija v pomnilniku.

[1] Večina orodij je predvidena za delo na preprostih operacijskih sistemih (praviloma MS-DOS), kar omogoča uporabo tudi na manj zmogljivih osebnih računalnikih. Najlažje in "najelegantnejše" je delo s čim preprostejšimi urejevalniki teksta, npr. "Edit", itd.

```
.id "target" "68020"
.id "translator" "as68k.exe"
.id "date" "Nov 18 1999  17:37:23"
.id "as68k.exe sid" "@(#)as68k.PL 3.69.1.1"
.seg {PSCT} %1 1 1 {}
.seg {idata} %2 1 1 {data}
.seg {udata} %4 1 1 {data}
.seg {S_main} %8 1 1 {code}
.len %1 0
.org %1 0
.len %2 4
.org %2 0
.defg {_a} %3 %2 #0 +
'00000001'
.len %4 12
.org %4 0
.defg {_b} %5 %4 #0 +
0:w 2
.defg {_c} %6 %4 #4 +
0:w 2
.defg {_d} %7 %4 #8 +
0:w 2
.len %8 28
.org %8 0
.ext {__main}%9
.defg {_main} %10 %8 #0 +
'222d'
0 %2 + {data}%11 -:I
'd2ad'
0 %4 + %11 -:I
'2b41'
4 %4 + %11 -:I
'202d'
0 %4 + %11 -:I
'2200'
'e788'
'9081'
'2b40'
8 %4 + %11 -:I
'4e75'
.group {data}%11 1 65535 %2 %4
```

koda ukaza
`MOVE.L D1,D0`

Slika 9. 3: Objektna koda

Pri prevajanju imamo na voljo različne opcije. Zanimiva je opcija optimiranja. Program lahko namreč optimiramo z ozirom na pomnilniški prostor ali čas. To pa pomeni, da ga lahko poskušamo prevesti v najkrajšo možno obliko (s tem prihranimo na pomnilniškem prostoru) ali pa poskušamo dobiti program, ki se bo izvajal v čim krajšem možnem času. V nasprotju s pričakovanji najkrajša objektna koda ni nujno tudi najhitrejša, o čemer se bomo prepričali v naslednjem poglavju.

9.1.1 Prevajanje iz C izvornega programa v "psevdozbirnik"

Objektno kodo dobimo neposredno s prevajanjem izvornega programa iz C jezika ali zbirnika. Včasih pa si je zanimivo ogledati, kako bi izgledal program, ki smo ga napisali v C

jeziku, če bi ga pisali v zbirnem jeziku s CPU ukazi. Zato je pri večini prevajalnikov predvidena tudi ta opcija[2]. Slika 9. 4 kaže rezultat tega postopka (datoteka s podaljškom ".mny").

```
        SECTION     idata,,"data"
        XDEF  _a
_a      DC.L  1

        SECTION     udata,,"data"
        XDEF  _b
_b      DCB   2,0
        XDEF  _c
_c      DCB   2,0
        XDEF  _d
_d      DCB   2,0
*1      /* Primer enostavnega izvornega programa v C kodi */
*2
*3      int a=1,b,c,d;          /*  deklaracija  in  inicializacija
spremenljivk */
*4
*5      main()                  /* glavni program */

        SECTION     S_main,,"code"
        _bringin    __main
        XDEF  _main
_main

*6      {
*7              c = a + b;      /* sestevanje spremenljivk */
        MOVE.L      _a-data(A5),D1
        ADD.L _b-data(A5),D1
        MOVE.L      D1,_c-data(A5)
*8
*9              d = 7 * b;      /*    mnozenje    neinicializirane
spremenljivke
        MOVE.L      _b-data(A5),D0
        MOVE.L      D0,D1
        LSL.L #3,D0
        SUB.L D1,D0
        MOVE.L      D0,_d-data(A5)
*10                         s konstanto*/
*11
*12     }                /* konec programa */
        RTS
```

Slika 9. 4: Program v C-ju s slike 9. 2 v mnemotehnični obliki (zbirni jezik)

C ukazi so tukaj zapisani v obliki oštevilčenih komentarjev. Programerju je vsekakor zanimiv pogled na komentirani C ukaz s številko `*9` za množenje s konstanto. Prevajalnik je namesto CPU ukaza (npr.) `MNYS` uporabil nekaj nadomestnih ukazov; množenje s konstanto 7 smo dosegli tako, da smo originalno spremenljivko (v registrih D0 in D1) najprej pomaknili za tri bitna mesta na levo (množenje z osem) in nato od rezultata odšteli originalno vrednost.

[2] Ta korak je pri navadnem prevajanju popolnoma nepotreben in ponavadi služi le v informativne namene.

Nadomestna sekvenca se bo izvršila hitreje od navadnega množenja (podrobnejši opis je podan v pogl. 4.4).

9.1.2 Elementi zbirnega jezika

V poglavju 4 smo našteli vse ukaze CPU32 v mnemotehnični obliki, ki so osnovni del vsakega programa v zbirnem jeziku. Poleg njih vsebuje program še nekaj skupin pomožnih inštrukcij (t.i. psevdoukazov - angl. pseudo ops.)[3], ki jih uporabljamo za organizacijo uporabniškega programa in se zato ne prevajajo v objektno kodo (razen ukaza `DC`). Tukaj bomo našteli le nekaj značilnih ukazov zbirnika. Nekatere bomo uporabili v Poglavju 10.

9.1.2.1 Krmiljenje zbirnika

- `END` je ukaz zbirnika s katerim zaključimo uporabniški program. Zbirnik ignorira vse naslednje ukaze.
- Z ukazom `INCLUDE` lahko v izvorni program vključimo neko sekundarno datoteko (npr. datoteko s tabelo sinusnih vrednosti).
- Ukaz `ORG` (angl. absolute origin) spremeni vsebino programskega števca. Z njim lahko definiramo dejanski začetek uporabniškega programa.

9.1.2.2 Določanje simbolov

S temi ukazi simboličnim nazivom dodelimo konkretne vrednosti.

- `EQU` (angl. Equate Symbol Value) dodeli številčno vrednost simbolu (npr. ukaz `k1 equ $50` bo simbolu k1 dodelil vrednost 50_{HEX}.
- ukaz `SET` je podoben prejšnjemu ukazu, razlika je edino v tem, da lahko pri njem med izvajanjem programa spremenimo številsko vrednost dodeljenega simbola, kar pri ukazu `EQU` ni dovoljeno.

9.1.2.3 Definiranje podatka in dodeljevanje pomnilniškega prostora

S temi ukazi rezerviramo pomnilniški prostor, v katerem bomo vnesli podatke.

- Z ukazom `DC.x` (angl. Define Constant; `x = B, W, L` itd.) bomo v spominu rezervirali prostor določene velikosti in v njega vpisali neko konstanto. Npr.

  ```
  DC.W 10,5,7  *inicializiraj tri 16-bitne lokacije
  ```

  Zaporedna ukaza `DC.B 'E'` in `DC.B 'J'` bosta naložila dva zaporedna byta z vsebinami 45_{HEX} in $4a_{HEX}$, to je z ASCII kodo črk E in J.

[3] Načeloma velja za zbirnike vseh proizvajalcev, v konkretnem zgledu pa je uporabljen zbirnik firme Intermetrics [33].

PAZI! Asemblerskega ukaza `DC.x` ne smemo mešati z ukazom `EQU`, saj slednji ne manipulira s pomnilniškim prostorom, temveč omogoča le preglednejše pisanje programa preko zamenjave številčne vrednosti s simbolom.

- Ukaz `DS.x y` (angl. Define Storage; `x` = `B`, `W`, `L` itd., `y` je celo število) rezervira prostor v pomnilniški mapi. Ukaz je zelo pomemben pri povezovanju, saj določa področje, na katerega ni možno shranjevati ukazov, podatkov, spremenljivk itd. Primer:

```
lok     DS.W $10  *rezerviraj prostor 16(DEC) besed za□enši
z
                  *lokacijo lok.
```

9.1.3 Formatiranje izpisa

V tej skupini so ukazi, ki formatirajo izpis programa v .lst datotekah in nimajo nobenega vpliva na sam uporabniški program. Primer: po ukazu `PAGE` se bo izpis nadaljeval na naslednji strani.

9.1.4 Nadzor nad zunanjimi simboli

Osnovno pravilo pri pisanju programov je, da enakega simboličnega imena ne smemo dodeliti več spremenljivkam. V naslednjem poglavju o povezovanju bomo omenili možnost modularnega pisanja delov in njihovega združevanja v celoto. Pri tem pristopu bomo uporabljali nekatere spremenljivke, ki smo jih definirali v drugih modulih pred združitvijo. Pozorni moramo biti na dve možni napaki:

- prevajalnik izvrši svojo funkcijo brez napak le, če so vsi simboli za spremenljivke, ki jih uporabljamo v določenem modulu, na začetku tega modula tudi deklarirane,
- povezovalnik ne dovoljuje večkratnega deklariranja simbolov spremenljivk v modulih, ki jih bomo združili v eno celoto, čeprav so bili pisani ločeno.

Zato moramo vsako simbolično podano veličino definirati v enem modulu (ukaz `XDEF` - angl. external definition), v drugih modulih, ki ta simbol uporabljajo, pa na začetku programa z ukazom `XREF` (angl. external reference) prevajalniku javiti, da bo simbol definiran v drugem modulu.

9.2 Povezovanje

Povezovanje (“linkanje”, angl. linking) je zelo pomemben korak do končnega uporabnega programa. Tukaj bomo omenili le nekatere funkcije povezovanja:

- povezovanje z ostalimi objektnimi moduli,
- povezovanje s knjižnicami,
- določanje končne lokacije programa v ciljnem programu.

Datoteka "*.ol" je rezultat prevajanja le enega programa ali programskega modula. Ta modul bi lahko bil del večjega programa, ki vsebuje še druge podobne module. Povezovanje v skupno celoto opravimo s povezovalnikom.
Nekatere funkcije (poenostavljeno: ukazi) v C-ju so zelo zapletene in sestavljene iz enostavnejših ukazov. Programerju so te funkcije na voljo v t.i. knjižnicah (angl. libraries). Knjižnice so urejene po sorodnosti funkcij: npr. standardne vhodno/izhodne funkcije (izpis znakov na zaslon, branje s tipkovnice, itd.), matematične funkcije (npr. trigonometrične in logaritmične funkcije), pretvorbe formatov itd. Programer lahko tudi sam definira uporabniške knjižnice, ki jih po potrebi vključuje v program. Knjižnice moramo pri povezovanju obvezno dodati k objektni kodi izvornega programa.

Objektna koda ne vsebuje informacije o naslovih na katerih se bo nahajal končni program v RAM ali ROM pomnilniku procesnega mikroračunalnika, kar je tudi smiselno, saj ga moramo še povezati z dodatnimi moduli in knjižnicami. Končne lokacije programskih segmentov določa povezovalnik s pomočjo posebne datoteke (v našem primeru "loc.cmd", slika 9. 5).

```
MEMORY  (#10000);               -- M68332 RAM je omejen na 64 K bytov.

RESERVE (#0 TO #3000);          -- Rezerviranje spodnjega področja RAM-a za
                                -- lastne potrebe

LOCATE (init : #3000);          -- Zacetek uporabniškega programa (kode)

LOCATE({code} {} {constant} {data} {isep} {usep} {stsep} S_end_project: #3080);

                                -- Lociranje posameznih segmentov programa
```

Slika 9. 5: Primer datoteke loc.cmd

V tej preprosti datoteki smo opisali pomnilniško konfiguracijo ciljnega procesnega mikroračunalnika (velikost RAM pomnilnika 64 Kbyte = 10000_{HEX}, pri čemer je spodnje področje rezervirano). Začetek programa se nahaja na lokaciji 3000_{HEX}.

V vsakem programu lahko ločimo posamezne segmente z nekaterimi skupnimi lastnostmi. To so npr. *ukazi* (ali koda - angl. code), *konstante* (angl. constants, npr. tabelarično podani podatki), *inicializirane* ali *neinicializirane spremenljivke* itd. Prevajalnik identificira pripadnost posameznih delov izvornega programa tem segmentom in jih tudi grupira. Pri povezovanju imamo možnost izbire naslovov segmentov znotraj dostopnega pomnilniškega prostora, karstorimo z zadnjo vrstico v datoteki "loc.cmd". To je pomembno zlasti, če bomo končni program "zapekli" v (E)EPROM pomnilnik[4]. V konkretnem primeru nima niti eden izmed segmentov s slike 9. 5 vnaprej določene začetne lokacije, torej jih bo povezovalnik samostojno porazdelil po dostopnem pomnilniškem prostoru, hkrati pa bo programerju posredoval informacije o tem (glej poglavje o mapiranju).

[4] V ROM tipu pomnilnika lahko shranimo ukaze in konstante, ne pa tudi npr. neinicializiranih spremenljivk. Naslov njihovega segmenta moramo določiti v nekem RAM pomnilniku.

Končni rezultat povezovanja je datoteka s podaljškom imena ".ab" (njen del prikazuje slika 9. ***6***). Kot vidimo, so tukaj že določene končne lokacije posameznih segmentov ter njihove velikosti. Koda ukaza `MOVE.L D0,D1` je del skupine ukazov, ki se začne na naslovu 3166_{HEX} (asemblerski ukaz `.aorg %3 #3166`). Preostali del datoteke vsebuje kode ukazov in podatke, po svoji obliki pa je zelo podoben objektni kodi s slike 9. 3.

```
.id "date" "Nov 18 1999  17:33:39"
.id "target" "68020"
.id "llink sid" "@(#)llink.PL 1.109.1.1"
.seg {idata}%1 1 u {data}
.len %1 #c
.abs %1 #318a
.seg {udata}%2 1 u {data}
.len %2 #8c
.abs %2 #3196
.seg {S_main}%3 1 u {code}
.len %3 #1c
.abs %3 #316e
.seg {init}%4 1 u {code}
.len %4 #48
.abs %4 #3000
.seg {S___exit_1}%5 1 u {code}
.len %5 #28
.abs %5 #3146
.seg {S_atexit}%6 1 u {code}
.len %6 #56
.abs %6 #30f0
.seg {S_end_project}%7 1 u {endall}
.len %7 4
.abs %7 #3222
.seg {S__alloc}%8 1 u {code}
.len %8 #70
.abs %8 #3080
.group {data}%9 1 152 %1 %2
.id "translator" "llink"
.defg {ldata}%10 #98
.defg {_a}%11 #318a
.defg {_b}%12 #3196
.defg {_c}%13 #319a
.defg {_d}%14 #319e
.defg {__main}%15 #3000
.defg {_main}%16 #316e
.defg {___main1}%17 #3156
.defg {___exit1}%18 #3044
.defg {___al_init}%19 #30d2
.defg {_exit}%20 #3114
.defg {___exit_1}%21 #3146
.defg {___exit}%22 #314e
.defg {_atexit}%23 #30f0
.defg {_end_project}%24 #3222
.defg {__alloc}%25 #3080
.start #3000
.id "as68k.exe sid" "@(#)as68k.PL 3.69.1.1"
.id "mod" "test.ol"
.aorg %1 #318a
'00000001'
.aorg %2 #3196
 0:W 2
 0:W 2
 0:W 2
.aorg %3 #316e
'222d'
 0:I
'd2ad'
 #c:I
'2b41'
 #10:I
'202d'
 #c:I
'2200'
 'e788'
'9081'
'2b40'
 #14:I
'4e75'
.id "llink sid" "@(#)llink.PL 1.109.1.1"
.id "as68k.exe sid" "@(#)as68k.PL 3.69.1.1"
.id "mod" "pmn340b.ol"
```

koda ukaza
MOVE.L D0,D1

Slika 9. 6: Del programa s slike 9. 2 po povezovanju (podaljšek ".ab")

9.3 Formatiranje

Vsi prej opisani postopki so imeli za cilj ustvariti končno datoteko s programom, ki ga želimo vstaviti v pomnilnik našega mikroračunalniškega sistema. Pri tem lahko program naložimo v:

- ROM (običajno EPROM) ali v
- RAM pomnilnik.

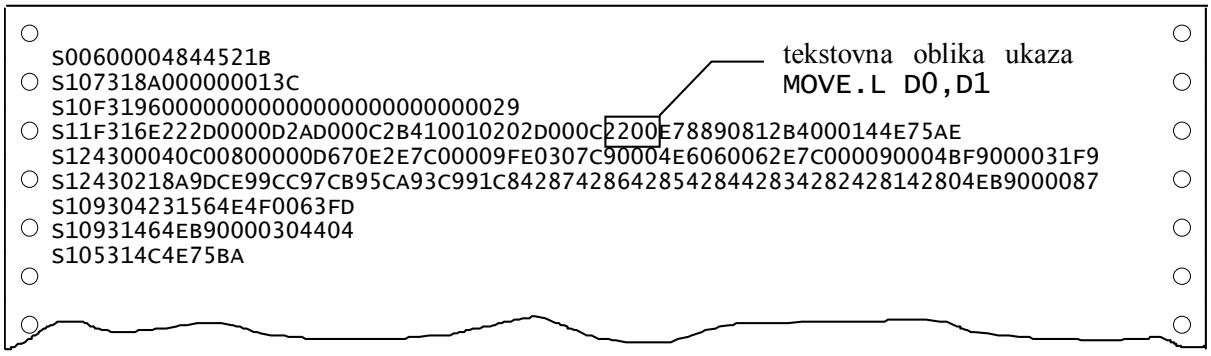
Prva varianta pride največkrat v poštev le, ko smo prepričani, da deluje program v skladu s pričakovanji in ga nimamo namena naknadno spreminjati. Takrat lahko program prilagodimo za takojšnje delovanje ob vsakokratnem vklopu mikroračunalnika.

V razvojni fazi je druga inačica veliko bolj elegantna. Ideja je v tem, da končni program z računalnika, na katerem smo program razvili (angl. host computer, najpogosteje osebni računalnik), naložimo v RAM pomnilnik procesnega mikroračunalniškega sistema in ga po potrebi spreminjamo ali ponovno naložimo.

Prenos se običajno vrši preko serijskega protokola EIA-232 za prenos ASCII znakov (glej poglavje 6.3). Datoteka iz prejšnjega podpoglavja, ki je rezultat povezovanja, ni primerna za takšen prenos, zato so proizvajalci mikroprocesorskih komponent (npr. Motorola, Texas Instrument, Hewlett Packard...) ustvarile lastne formate za prenos.

Motorola uporablja S19 format (datoteke s podaljškom ".s19" ali ".hex"), ki je prikazan na sliki 9. 7. Za pretvorbo dokončne objektne kode (podaljšek ".ab") v tekstovni S19 format skrbi posebni program, t.i. *formater* (angl. formatter).

Na sliki smo ponovno označili kodo ukaza `MOVE.L D0,D1` v tekstovni (ASCII) obliki.



Slika 9. 7: Del datoteke z S19 formatiranim programom

Naziv Motorolinega formata izhaja iz začetnih črk vsake vrstice (od S0 do S9), ki določajo pomen teh vrstic. Npr., vrstica S0 lahko vsebuje opisne informacije o programu, S9 ga zaključuje, vrstice z začetnicami S1, S2 in S3 pa vsebujejo podatke ali kode ukazov. Razlika med vrsticami S1, S2 in S3 je le v formatu naslova, na katere se shranjujejo podatki ali kode:

pri S1 je naslov sestavljen iz dveh bytov, pri S2 iz treh, pri S3 pa iz štirih. Oglejmo si primer zadnje vrstice:

```
S1    -   oznaka vrstice
05    -   kontrolno število (v HEX formatu), ki pove koliko bytov mu sledi v vrstici
314c  -   naslov
4e75  -   koda ukaza ali podatek
ba    -   kontrolna vsota (angl. checksum)
```

Kontrolna vsota je le podatek, ki vsebuje informacijo o pravilnosti prenosa podatkov. To je byte z najmanjšo težo enojnega komplementa vsote vseh bytov (brez uvodnih bytov S*n*).

V našem primeru:

$05_{HEX} + 31_{HEX} + 4c_{HEX} + 4e_{HEX} + 75_{HEX} = 145_{HEX}$,

$\sim 145_{HEX} = eba_{HEX}$,

kjer upoštevamo le byte z najnižjo težo ba_{HEX}.

Sprejemnik (monitorski program v procesnem mikroračunalniku) na koncu vsake vrstice izračuna kontrolno vsoto in jo primerja z zadnjim sprejetim bytom.

9.4 Mapiranje

Pri pisanju programskega modula v C jeziku ali zbirniku programer načeloma nima vpogleda v naslove, na katerih se bodo nahajali spremenljivke, procedure in konstante (razen če jih sam eksplicitno ne definira). V fazi testiranja programa v procesnem računalniku pa si pogosto želimo ogledati vsebine pomnilniških področjih, na katerih se nahajajo spremenljivke ali podatki, ali locirati določen podprogram v pomnilniku zaradi sprotnega spremljanja njegovega delovanja. *Mapiranje* (angl. mapping) je ena izmed neobveznih opcij, ki nam omogoča izpis vseh relevantnih podatkov o položaju posameznih segmentov programa v pomnilniku (angl. cross-reference). Rezultat mapiranja (datoteka z podaljškom ".map") je zelo zanimiv, saj nam veliko pove o konceptu povezovanja in upravljanja s pomnilnikom, zato jo prikažimo v celoti (slika 9. 8).

Skupina ukazov s slike 9. 2, katere del je tudi ukaz `MOVE.L D0,D1` se nahaja v sekciji `_main`, ki se nahaja na naslovu $316e_{HEX}$. Na sliki ***9. 2*** so to ukazi v glavnem programu (`main`), od C ukaza `c = a + b;` naprej.

```
Translator : llink
Target     : 68020

Global       Address

__main       00003000 (12288)
___exit1     00003044 (12356)
__alloc      00003080 (12416)
___al_init   000030d2 (12498)
_atexit      000030f0 (12528)
_exit        00003114 (12564)
___exit_1    00003146 (12614)
___exit      0000314e (12622)
___main1     00003156 (12630)
_main        0000316e (12654)
_a           0000318a (12682)
_b           00003196 (12694)
_c           0000319a (12698)
_d           0000319e (12702)
_end_project 00003222 (12834)

Group        Size Limit            Align  Member Segments

data         000098 (152)          hword  idata        udata

Segment      Address               Length            Class      Align  Combine

init         00003000 (12288)      000048 (72)       code       hword  private
S__alloc     00003080 (12416)      000070 (112)      code       hword  private
S_atexit     000030f0 (12528)      000056 (86)       code       hword  private
S___exit_1   00003146 (12614)      000028 (40)       code       hword  private
S_main       0000316e (12654)      00001c (28)       code       hword  private
idata        0000318a (12682)      00000c (12)       data       hword  private
udata        00003196 (12694)      00008c (140)      data       hword  private
S_end_project
             00003222 (12834)      000004 (4)        endall     hword  private
```

```
Statistics

Segments   : 8
Globals    : 16
Groups     : 1
Sum of class "code" segments  : 00000152 (338)
Sum of class "data" segments  : 00000098 (152)
Sum of all other segments     : 00000004 (4)

Total size of all segments    : 000001ee (494)

User Start Address = #3000
```

Slika 9. 8: Datoteka ".map"