VOGRIN

HEX X XXXX
65536 4096 256 16 1

CVTSP str. → pack.
CVTPS

CVTPL pack. - long
CVTLP long - pack

V210

BEQL - riqured
BEQLU - unriqured

OP1 - OP2 za COND BR.

BLBC
BLBS - low bit (ϕ)

ZBIRNI JEZIK I
ZA DELTA/V

Učbenik za seminar V210

Verzija 1.0

#^A/+/ - ASCII

N! = N * (N-1)

VSEB ADR

CMPL H,#H

AP - kaže na št. argumentov za CALL

CALLG arglist, proc - v arglist = 1L št. arg.
CALLS #num, proc

Ljubljana, september 1984

POGLAVJE 1

ARHITEKTURA SISTEMOV DELTA 4780 IN DELTA 4850

Arhitektura sistemov DELTA 4780 in DELTA 4850 je na kratko opisana v knjigah "Osnove operacijskega sistema DELTA/V" in "Priročnik za operaterje na OS DELTA/V", zato bomo zdaj podrobneje opisali le centralni procesor (CPU od angleškega izraza Central Processing Unit) in organizacijo fizičnega pomnilnika.

1.1 CENTRALNI PROCESOR

Centralni procesor je enota, ki vodi in kontrolira delo računalnika. Sestavljen je iz aritmetično logične enote, ki izvaja operacije s podatki, iz splošnih registrov, ki jih lahko uporablja makro programer in iz posebnih registrov, ki jih uporablja operacijski sistem, programer pa do njih navadno nima dostopa. K procesorju spada tudi hiter vmesni pomnilnik ali "cache".

1.1.1 Splošni registri

Procesorja VAX-11/750 IN VAX-11/780 sta 32-bitna, kar pomeni, da dela procesor s celim 32-bitnim podatkom naenkrat. To pomeni, da so tudi registri in podatkovni kanali 32-bitni.

Makro programer ima za svoje namene na voljo 16 registrov, od katerih imajo nekateri še dodatne zadolžitve. Imena registrov so R0 do R11, naslednji štirje pa imajo imena AP (od argument pointer) ali kazalec podatkov, FP (ali frame pointer), SP (od stack pointer) ali kazalec sklada in PC (od program counter) ali programski števec.

Registre AP, FP, SP in PC lahko makro programer uporablja po svoje, vendar se mora zavedati, da te registre uporablja tudi operacijski sistem in sam procesor.

Kazalec argumentov AP se uporablja za prenos argumentov pri klicu podprograma in ga na enak način uporabljajo vsi višji programski jeziki. Pri klicu podprograma pa procesor sam poskrbi, da AP res kaže na pravi naslov.

Tudi FP se uporablja za delo s podprogrami. Ta register kaže kje je podatkovna struktura, ki se imenuje "call frame" in v kateri so zapisane vrednosti registrov pred klicem in informacija operacijskemu sistemu kaj naj naredi, če pride do napake pri izvajanju podprograma.

Register SP služi za organiziranje sklada, to je dinamična podatkovna struktura. Količina pomnilnika, ki ga sklad zasede je vedno ravno enaka skupni velikosti vseh podatkov v skladu. Sklad je podoben skladovnici drv ali kupu krožnikov. Če hočemo shraniti krožnik, ga položimo na vrh kupa. Zadnji dodani krožnik je na vrhu, prej dodani po so po vrsti pod njim. Ko vzamemo krožnik s kupa, vzamemo najprej tistega, ki smo ga nazadnje postavili na kup, nato vzamemo predzadnjega in tako naprej. Taka struktura nosi angleško kratico LIFO od "last in first out", kar pomeni zadnji noter, prvi ven. Kazalec sklada kaže vedno na podatek, ki smo ga nazadnje zapisali v sklad in ta podatek dobimo, ko vzamemo nekaj s sklada. Ukazi za delo s skladom sami poskrbijo, da se kazalec sklada sproti prilagaja vsem spremembam v skladu.

Za razliko od ostalih treh registrov, ki se spremenijo le, ko delamo s skladom ali pri klicu podprograma, se programski števec spremeni po vsakem ukazu. Ta register kaže vedno na naslednji ukaz in procesor sam popravi to vrednost po vsakem prebranem ukazu. Tega registra torej ne moremo uporabljati za shranjevanje svojih podatkov.

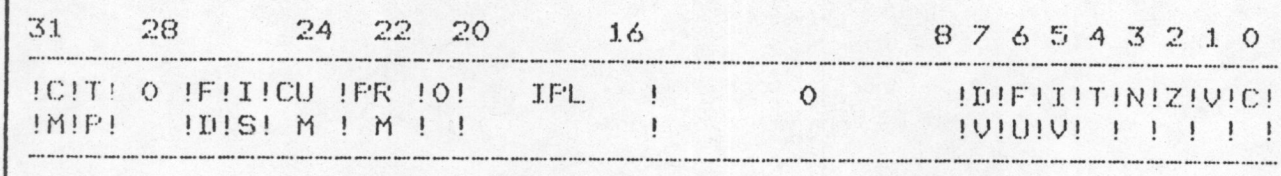
Tudi pri ostalih dvanajstih registrih se nam lahko zgodi, da vrednost v registru ni takšna, kot jo pričakujemo. Po show time show time dogovoru uporabljajo višji programski jeziki in sistemski podprogrami registra R0 in R1 za vračanje statusa, s katerim se je končalo izvajanje podprograma ali za rezultat, če kličemo podprogram kot funkcijo. Po klicu sistemskega podprograma bomo torej imeli v registru R0 drugačno vrednost kot pred klicem.

Drug primer, ki nam lahko pokvari stanje v registrih je uporaba ukazov za delo s teksti. Ti ukazi namreč uporabljajo registre R0 do R5 za shranjevanje številčev znakov v tekstih in za kazalce na tekoče znake. Pred uporabo teh ukazov moramo spraviti vsebino teh šestih registrov na varno, npr. na sklad, po uporabi pa staro vsebino vrnemo v registre.

1.1.2 Stanje procesorja

Trenutno stanje procesorja je zapisano v posebnem registru z imenom "processor status longword" ali krajše PSL. Tudi ta register ima 32 bitov, ki pa so razdeljeni na dve skupini po 16. Biti od 0 do 15 vsebujejo neprivilegirane podatke, ki so dostopni vsakemu

uporabniku in nosijo ime "processor status word" ali PSW.



Slika 1.1: Status procesorja.

Biti 0 do 3 so pogojni biti in v njih se zapišejo nekateri pogoji ob izvršitvi ukaza. Bit 0 z oznako C (Carry) pove, ali je prišlo do prenosa iz vodilnega bita. Do prenosa pride npr. pri seštevanju dveh negativnih števil, kajti negativna števila imajo v vodilnem bitu zapisano 1. Če pride do prenosa, to še ne pomeni, da je rezultat napačen zaradi prekoračitve obsega števil. Prenos uporabljamo naprimer pri delu z velikimi celimi števili (z dolžino več kot 32 bitov), ko moramo računati tako, da najprej seštejemo zadnjih 32 bitov, nato predzadnjih 32 itd. Bit 1 ali V (oVerflow) pove, če je prišlo do prekoračitve obsega števil. V ta bit se zapiše 1, če je rezultat neke operacije tako velik, da ga ne moremo zapisati v pomnilnik, ki je za podatek na voljo. Bit Z (Zero) signalizira, da je bil rezultat zadnje operacije nič in bit 3 ali N (Negative) pove, da je bil rezultat negativen.

Samo nekateri ukazi vplivajo na stanje vseh štirih pogojnih bitov, večina jih vpliva samo na enega ali dva. Pri testiranju nekega podatka naprimer ne more priti do prekoračitve. Zvemo le, če je podatek večji, enak ali manjši od nič. Pri takih ukazih ostane vsebina nekaterih pogojnih bitov nespremenjena ali pa postavijo ta bit na 0. Za vsak ukaz posebej lahko to preverite v Architecture Handbook ali v VAX11 Programming Card.

Naslednji štirje biti povedo procesorju, če naj prekine izvajanje in generira napako, ko pride do posebnih okoliščin. Bit 4 ali T (Trace) zahteva, če je postavljen na 1, da pride do prekinitve izvajanja po izvršitvi naslednjega ukaza. Ta pripomoček služi za čiščenje programa (DEBUG), ali za spremljanje izvajanja programa po posameznih korakih.

Bit IV (Integer oVerflow), to je bit 5, pove, ali naj pride do prekinitve izvajanja programa ko presežemo območje celih števil. Pogojni bit V se postavi neodvisno od stanja bita IV. Namesto da po vsakem ukazu testiramo stanje bita V, zapišemo 1 v bit IV in ko pride do prekoračitve, nas sistem sam opozori tako, da prekine izvajanje programa.

Bit 6 je FU bit (Floating Underflow) in zahteva prekinitev, če je rezultat pri delu s števili v plavajoči vejici premajhen, da bi ga lahko zapisali v prostor, ki nam je na voljo.

DV (Decimal oVerflow) ali bit 7 služi za zaznavanje prekoračitev pri delu s pakiranimi decimalnimi števili. Če je rezultat prevelik za prostor, ki smo mu ga določili, se izvajanje programa prekine.

Za ostale posebne okoliščine, kot so deljenje z nič ali prevelik rezultat pri delu s števili v plavajoči vejici ni posebnih bitov, ki bi označevali, kdaj naj pride do prekinitve.

Preostalih osem bitov v PSW mora biti postavljenih na nič in se za sedaj ne uporabljajo.

Druga polovica statusa procesorja vsebuje privilegirane podatke, služijo procesorju za kontrolo dostopa do zaščitenih podatkov in določajo, katere nujne zahteve drugih uporabnikov lahko prekinejo izvajanje tekočega programa.

Polje od bita 16 do 20 je IPL (Interrupt priority level) ali nivo prioritete prekinitev. Vsak tip prekinitve, naprimer pritisk na tipko terminala ali zaključek branja podatka z diska, ima določen svoj prioritetni nivo. Zahteva za prekinitev dela procesorja mora imeti prioriteto večjo od vrednosti, ki je zapisana v polju IPL. Biti 22 do 25 označujejo predhodni način dela (22:23) in sedanji način dela (24:25) procesorja. Ti načini so po vrsti od najmanj do najbolj privilegiranega naslednji: user, supervisor, executive in kernel.

Naslednji biti po vrsti povedo, ali dela procesor s posebnim prekinitvenim skladom (IS bit za Interrupt Stack), ali je procesor že končal izvajanje prvega dela ukaza, kar pride v poštev pri dolgih ukazih, naprimer za delo s teksti (FPD bit ali First Part Done), Bit TP (Trace Pending) koordinira prekinitve zaradi posebnih okoliščin in omogoča samo eno prekinitev naenkrat in bit CM (Compatibility Mode) ve, ali dela procesor tako, da posnema procesor tipa PDP11.

1.1.3 Posebni registri

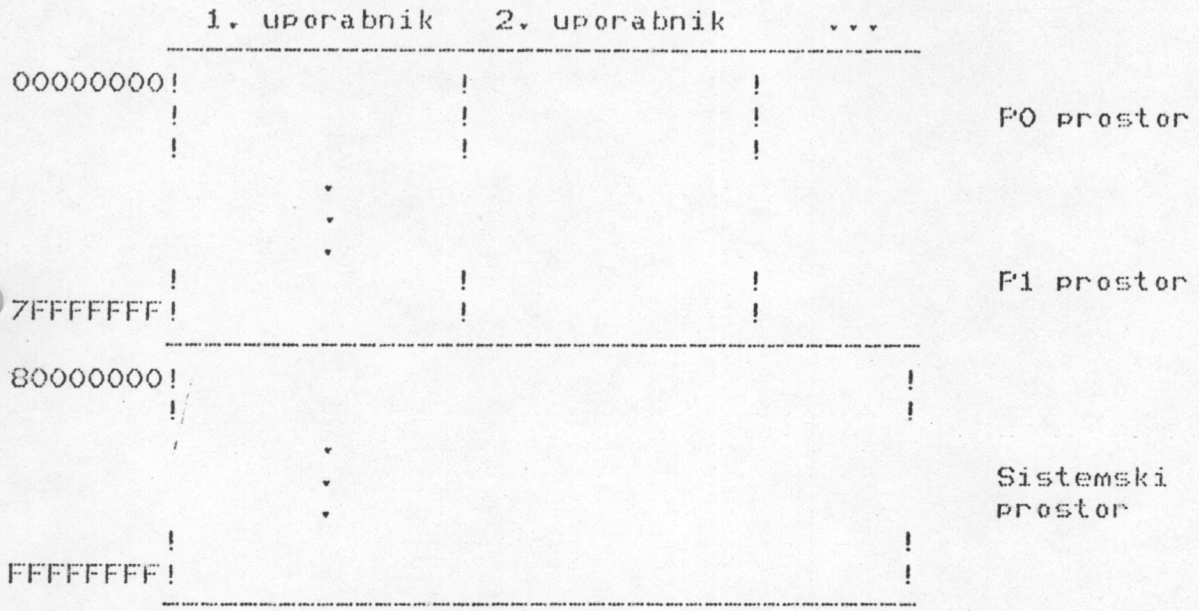
Procesor ima še vrsto posebnih registrov, ki so zaščiteni pred posegi neprivilegiranih uporabnikov, ker lahko sprememba v nekaterih od teh registrov katastrofalno vpliva na delo celotnega operacijskega sistema.

Posebni registri so naprimer štirje kazalci sklada za štiri načine dela procesorja in peti kazalec na prekinitveni sklad, potem registri, ki omogočajo delo z virtualnim pomnilnikom, ter registri za direktno branje in pisanje na konzoli, itd.

1.2 POMNILNIK

Osnovna enota pomnilnika je za procesorje tipa VAX11 byte, ki ima 8 bitov. To pomeni, da ima vsak byte svoj naslov od 0 do največjega naslova, ki se zapiše z 32 binarnimi enicami in je nekaj čez štiri milijarde. Sestavljene enote pomnilnika so beseda (word) iz dveh bytov in dolga beseda (longword) iz štirih bytov. Dolga beseda je podatek, ki ga procesor dobi z enim dostopom do pomnilnika. Pravimo, da so podatki poravnani po besedah ali po dolgih besedah, če so naslovi teh podatkov deljivi z 2 oziroma s 4. Če podatek dolžine 4 byte ni poravnan po dolgih besedah, sta potrebna dva dostopa do pomnilnika, da preberemo cel podatek. Na to moramo paziti pri optimizaciji programov.

Celoten naslovni prostor procesorja je 4 giga byte. Ta naslovni prostor je razdeljen na dva enako velika dela. Naslovi, ki imajo vodilni bit (bit številka 31) nič, spadajo v procesni prostor, naslovi z vodilnim bitom 1 pa so v sistemskem prostoru. Vsak uporabnik ima svoj procesni prostor in ti prostori se med seboj ne mešajo, vsi pa imajo skupen sistemski prostor, v katerem je naprimer sam operacijski sistem, ki je vsem uporabnikom skupen.



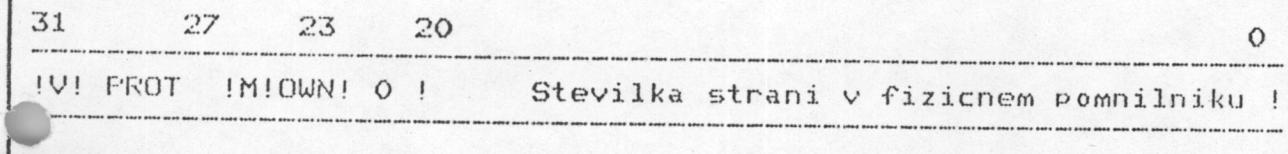
Slika 1.2: Naslovni prostor.

Oba podprostora sta razdeljena še na dva dela. Procesni prostor je razdeljen na prostor P0 z bitom 30 postavljenim na 0 in P1 z bitom 30 postavljenim na 1. V prostoru P0 so slike programov, ki jih izvaja nek uporabnik in večina podatkov, ki jih ti programi uporabljajo, v P1 pa sistem izbere prostor za sklad in za podatke, ki jih uporablja operacijski sistem in so značilni za posamezen proces. Na podoben način je razdeljen sistemski prostor, vendar se uporablja le polovica z nižjimi naslovi (bit 30 je 0).

Operacijski sistem ima poseben mehanizem za dodeljevanje fizičnega pomnilnika posameznim procesom. Angleško se imenuje "memory management", kar bi lahko prevedli kot upravljanje s pomnilnikom. Ta mehanizem, ki je sestavljen iz strojnih in programskih pripomočkov, omogoča delo z velikimi programi ali podatkovnimi strukturami, ki so prevelike za fizični pomnilnik, hkrati pa omogoča pravično delitev pomnilnika med več uporabnikov.

Fizični pomnilnik je razdeljen na strani (page) po 512 bytov. To je prav toliko, kot je velik blok na disku. Ko se nek program izvaja, so v fizičnem pomnilniku samo tiste strani, na katerih so zapisani ukazi in podatki, s katerimi program ta trenutek dela. Ostali ukazi in podatki so zapisani na disku in ko program rabi nek tak podatek, pride do napake strani. Operacijski sistem poskrbi, da se prepiše v pomnilnik pravi blok z diska in program lahko nadaljuje delo.

Podatki o tem katere strani so v pomnilniku in katere na disku, so zapisani v tabeli strani (page table). Podatki v taki tabeli so dolge besede. Vsakim 512 bytom naslovnega prostora, ki ga proces uporablja, ustreza en tak podatek. Naslovom od 0 do 511 ustreza podatek z zaporedno številko 0, naslovom 512 do 1023 podatek z zaporedno številko 1 itd. Podatki o straneh so razdeljeni na polja. Biti od 0 do 20 povedo številko strani v fizičnem pomnilniku, biti 21 do 25 so rezervirani, bit 26 pove, ali je bila ta stran spremenjena, biti od 27 do 30 opisujejo zaščito strani, vodilni bit pa pove, ali je ta stran v fizičnem pomnilniku.



Slika 1.3: Podatek iz tabele strani.

Poglejmo, kako operacijski sistem določi fizični naslov podatka z naslovom 1034. Binarno zapisano je ta naslov 10000001010. Zadnjih devet bitov pove, kje je ta naslov znotraj strani, v našem primeru je to enajsti byte (naslovi gredo od 0, naslov 10 označuje 11. podatek). Če zadnjih devet bitov odrežemo, nam ostane zaporedna številka te strani, v tem primeru je to 2. Operacijski sistem bo torej posledal podatek 2 v tabeli strani in če je vodilni bit 1, kar pomeni, da je stran v fizičnem pomnilniku, bo polje bitov od 0 do 20 uporabil kot številko strani v fizičnem pomnilniku. Če je ta podatek 7, torej dvojiško 111, je fizični naslov našega podatka 111000001010 ali 3594.

POGLAVJE 2

TIPI PODATKOV

Podatek zapišemo v pomnilnik vedno v enaki obliki in sicer kot zaporedje ničel in enic, torej kot neko binarno število. Isti podatek lahko procesor bere na različne načine glede na ukaz, ki ta podatek uporablja. Podatek dolžine 32 bitov je lahko celo število, število s pomično vejico, tekst in še kaj. Ogledali si bomo, kako procesor tolmači različne tipe podatkov in katere tipe sploh pozna.

2.1 ŠTEVILA

Osnovni tip podatkov so cela števila. Zapišemo jih seveda binarno, velikost celih števil pa je omejena s številom bitov, ki so za zapis števila na voljo. Celo število zapišemo navadno v dolgo besedo, se pravi v 32 bitov dolgo polje, in velikost števila je do nekaj čez štiri milijarde. Zahtevamo lahko, da se za zapis števila porabi le ena beseda ali samo byte, območje takih števil pa je seveda primerno manjše, za en byte je to od 0 do 255. Navadno potrebujemo tudi negativna števila. Po dogovoru je negativno število tisto, ki ima zapisano 1 v vodilnem bitu. Zapis za negativno število dobimo tako, da zapišemo absolutno vrednost tega števila in izračunamo dvojiški komplement - zamenjamo vse enice z ničlami in obratno in prištejemo ena. Območje celih števil z velikostjo en byte je torej -128 do 127.

Zelo velika ali zelo majhna števila zapišemo v obliki s plavajočo vejico, torej kot neka realna števila, za katera zapišemo mantiso in eksponent v točno določena polja. Osnovna oblika števila s plavajočo vejico ima dolžino dolge besede. En bit je namenjen predznaku, 8 bitov predstavlja eksponent in ostalih 23 mantiso. Območje, ki ga pokrijemo s temi števili je med 0.29E-38 in 1.7E38. Natančna so približno na 7 decimalnih mest. Za večjo natančnost in za večje območje imamo na voljo še tri oblike zapisa teh števil, dve obliki zasedeta po 8 bytov, ena pa celo 16.

Naslednji tip števil so pakirana decimalna števila, kot jih uporablja na primer COBOL. Po dve decimalni števili lahko zapišemo v en byte, za vsako številko porabimo 4 bite. Ta števila so v pomnilniku

vedno zapisana s predznakom, tako da dvomestno število ne zaseda le enega byta, to je dvakrat po 4 bite, ampak dva byta, ker potrebuje trikrat po 4 bite. Pri tem smo zavrgli nekaj prostora v pomnilniku, saj lahko s štirimi biti zapišemo števila od 0 do 15, mi pa uporabljamo samo številke od 0 do 9.

Na ta način lahko zapišemo 31 mestna cela števila in neposredno računamo z njimi, kar je prednost pred binarno predstavitvijo z dolgo besedo, kjer nimamo niti polnih deset mest.

Števila lahko zapišemo tudi kot niz znakov, torej kot tekst, vendar s tako zapisanimi podatki ne moremo računati, obstajajo pa ukazi za pretvarjanje takih števil v druge oblike.

Dodatne podatke o zapisu števil in natančen opis kako se zapišejo posamezna polja dobite v 4. poglavju knjige VAX Architecture Handbook.

2.2 TEKSTI

Teksti so v računalnikovem pomnilniku predstavljeni kot zaporedja ASCII kod za posamezne znake. Za vsak znak porabimo po en byte. Pri ukazih za delo s teksti moramo povedati kje je tekst zapisan in dolžino teksta. Dolžina teksta je navadno zapisana v eni besedi, tako da lahko delamo s teksti dolgimi do 65535 znakov.

Ko zapisujemo tekst v pomnilnik, moramo paziti na to, da je prvi znak teksta na najnižjem naslovu, drugi znak ima za ena večji naslov in tako naprej.

V prejšnjem razdelku smo govorili o številih, ki jih zapišemo z nizom desetiških znakov. V resnici so tako zapisana števila navadni teksti in z njimi ne moremo računati.

2.3 NIZ BITOV

Povsem nov tip podatka je niz bitov. V pomnilniku lahko izberemo poljubno zaporedje do 32 bitov ne glede na meje med byti. Za zaporedje bitov vzamemo na primer 3 vodilne bite iz enega byta, vse bite iz naslednjega in 5 bitov iz tretjega in tako dobimo polje 16 bitov, ki se razteza preko treh bytov.

Niz bitov določimo s tremi podatki: baza je naslov byta, od katerega bomo šteli bite, odmik je število bitov, ki jih bomo preskočili in dolžina pove kako dolg je ta niz. Baza je lahko naslov, katerekakoli byta, odmik je dolga beseda, kar pomeni, da lahko preskočimo do 2 giga bita naprej ali nazaj in dolžina je celo število med 0 in 32 vključno.

Baza je lahko tudi ime nekega registra. Če sega niz bitov preko vodilnega bita tega registra, se nadaljuje v registru z za ena večjo zaporedno številko. Ko je baza R0, se torej niz bitov nadaljuje v R1. Ko delamo z registri, mora biti odmik vedno manjši od 32, kar pomeni, da se niz bitov začne v registru, ki ga vzamemo za bazo.

2.4 DVOJNO POVEZANE VRSTE

Poseben primer podatkov so dvojno povezane vrste. Člen take vrste je sestavljen iz kazalca na naslednji člen v vrsti, kazalca na predhodni člen in iz samega podatka (to je lahko tudi kazalec na podatek, ki je zapisan na drugem mestu).

Ko je vrsta prazna, ima samo en člen, ki ga imenujemo glava vrste. Glava ima samo dva kazalca, ki kažeta na prvi in zadnji člen v vrsti. V prazni vrsti sta oba kazalca naslov, kjer je zapisana glava.

Ko dodamo v prazno vrsto nek člen, kažeta oba kazalca glave na ta člen, saj je ta hkrati prvi in zadnji. Vedno ko dodajamo nove člene, ali ko spreminjamo njihov vrstni red, spreminjamo le vrednosti kazalcev in ne prenašamo celotnih podatkov po pomnilniku. Vrste se zato največ uporabljajo za opis podatkov, ki jih dosti vstavljamo in brišemo ter spreminjamo njihov vrstni red. Vrste dosti uporablja operacijski sistem za razporejanje virov, za katere se poteguje več uporabnikov, naprimer za dodeljevanje procesorskega časa, ali pa vrsta za izpis na tiskalniku.

2.5 ZAPISOVANJE PODATKOV V POMNILNIK

MAKRO programer ima na voljo vrsto navodil prevajalniku, s katerimi zahteva, da se na določene naslove zapišejo različni podatki. Vsa taka navodila se začnejo s piko, ime pa določa vrsto navodila. Če želimo zapisati celo število 123 v dolgo besedo, napišemo:

```
.LONG   123
```

Podobne ukaze imamo tudi za ostale oblike podatkov, nekateri od njih so zbrani v razpredelnici:

ukaz	parametri	pomen
.LONG	123,777	zapiše v pomnilnik dve dolgi besedi z vrednostima 123 in 777
.WORD	123	zapiše besedo z vrednostjo 123
.BYTE	7	zapiše byte z vrednostjo 7
.FLOAT	1.123	zapiše v dolgo besedo realno število 1.123
.D_FLOATING	1.123	zapiše v dve dolgi besedi število

```
                                        z dvojno natančnostjo
.ASCII   /To je tekst./                 zapiše v pomnilnik 12 znakov teksta
.ASCID   /To je tekst./                 naredi opisnik in zapiše v
                                        pomnilnik določeni tekst
.BLKB    16                             rezervira v pomnilniku 16 zaporednih
                                        bytov in jih zapolni z ničlami
.BLKW    1                              rezervira eno besedo
.BLKL    6                              rezervira šest dolgih besed
```

<LF> = 10

<CR> = 13

.BLKB
W
L
Q
O - octa
A - address
F
D
G
H

.BYTE
.LONG 2W
.WORD
.QUAD 4W
.OCTA 8W

.ASCII
.ASCIZ
.ASCIC
.ASCID

POGLAVJE 3

NAČINI NASLAVLJANJA IN FORMAT UKAZA

3.1 OBLIKA MAKRO PROGRAMA

Preden začnemo govoriti o ukazih, si oglejmo še kako pišemo makro program. Oblika programa ni obvezna, dobro pa je, da se držimo priporočil, saj je tako program preglednejši.

Vrstica je razdeljena na polja. Prvo polje se začne v prvem stolpcu in vanj pišemo oznake vrstic. To je simbol, ki se mu priredi vrednost naslova v pomnilniku, na katerem bo zapisan ukaz iz te vrste. Oznako vrstice obvezno zaključimo z dvopičjem, ki pa ni del imena. Če je oznaka vrstice tako dolga, da sega v drugo polje, lahko vso vrstico zamaknemo v desno, ali pa pišemo oznako v svoji vrsti ukaz pa v naslednji.

Drugo polje se začne v devetem stolpcu. Navadno pridemo v drugo polje s tabulatorjem. V to polje pišemo ime ukaza ali navodilo prevajalniku. Nekaj navodil prevajalniku je opisanih v poglavju o tipih podatkov. V to polje bi naprimer napisali .WORD ali ukaz kot je CLRB.

V tretje polje, ki se začne v 17. stolpcu ali za dvema tabulatorjema, pišemo podatke za ukaz. Če je podatkov več, jih ločimo z vejico.

Četrto polje od 41. stolpca ali za petimi tabulatorji je za komentar. Začetek komentarja zaznamujemo s podpičjem. Krajši komentarji so navadno v isti vrstici kot ukaz, daljše pa raje pišemo v svoji vrstici.

Lepo je, če program začnemo z navodilom .TITLE in imenom programa, na začetku ukazov, torej tam, kjer se začne izvajati naš program, pa določimo vstopno točko z .ENTRY. Na koncu datoteke s programom moramo pisati .END in ime, ki smo ga navedli pri .ENTRY.

Primer makro programa:

```
        .title  Program za sestevanje dveh celih stevil.
;-------------------------------------------------------------------------
;
;       Program sesteje dve celi stevili, ki ju ima
;       zapisani na naslovih PRVO in DRUGO, rezultat
;       pa zapise na naslov REZULTAT.
;
;-------------------------------------------------------------------------
PRVO:   .long   5                       ; Prostor za prvo stevilo.
DRUGO:  .long   8                       ; Prostor za drugo stevilo.
REZULTAT:
        .blkl   1                       ; Rezervira blok pomnilnika
                                        ; z dolzino ene dolge besede.
; Ime vstopne tocke je SESTEJ, obvezen parameter pa je se
; maska, v kateri zapisemo imena registrov, v katerih ne bi
; radi izgubili vrednosti.
        .entry  SESTEJ  ^m<R3,R4>
        addl3   prvo,drugo,rezultat
        movl    #1,r0                   ; Program lahko koncamo z
        ret                             ; ukazom ret (povratek iz
                                        ; podprograma), v R0 pa
                                        ; zapisemo status - 1 je uspeh.
        .end    SESTEJ                  ; Konec programa z imenom
                                        ; vstopne tocke.
```

3.2 FORMAT UKAZA

Strojni ukaz je na procesorjih VAX11 sestavljen iz same kode ukaza, ki zaseda en ali dva byta in iz kod, ki povejo, kje so podatki za ta ukaz. Podatek je opisan z registrom, ali pa je vnešen neposredno. Byte s kodo podatka je razdeljen na dve polji s po štirimi biti. Biti 0 do 3 povejo številko registra, ki ga uporabljamo za določanje podatka. S temi štirimi biti lahko zapišemo 16 različnih števil, kar je dovolj za vseh 16 splošnih registrov. Preostali štirje biti določajo način naslavljanja, torej kako bomo določeni register uporabili. Za načine naslavljanja imamo 16 možnosti, vendar so nekatere kode le variante istega načina, tako da imamo le devet različnih načinov.

```
-----------
!11000001!      koda ukaza ADDL3
-----------
!00000010!      opis prvega operanda (#2)
-----------
!11001111!      opis drugega operanda
-----------     z relativnim naslavljanjem
!11111110!
-----------     odmik
!00011010!
-----------
!01010001!      opis tretjega operanda (R1)
-----------
```

Slika 3.1: Slika ukaza v pomnilniku.

Ko procesor izvrši nek ukaz, prebere naslednjega. Procesor torej prebere byte z naslova, ki je zapisan v programskem števcu in takoj poveča programski števec za 1. Ko raztolmači kodo ukaza, se odloči, ali je naslednji byte nadaljevanje tega ukaza. To je takrat, ko je koda ukaza dolga dva byta, ali ko je potreben za izvršitev ukaza še kakšen podatek. Ko procesor pričakuje podatek, prebere še en byte in spet poveča vrednost programskega števca za 1. Iz načina naslavljanja ugotovi, koliko dodatnih bytov določa ta podatek in ko jih bere sproti popravlja vrednost v programskem števcu. Ta postopek ponavlja, dokler ne prebere vseh podatkov in takrat kaže programski števec na kodo naslednjega ukaza.

3.3 UPORABA REGISTROV

Registre lahko uporabljamo na različne načine. Najpreprosteje je, če uporabimo register kot akumulator, torej da zapišemo podatek v register. Na ta način smo začasno shranili nek podatek in nam ni bilo treba za to iskati prostega pomnilnika, hkrati pa je ta podatek hitro dostopen, saj je register del procesorja in si pri naslednjem iskanju tega podatka prihranimo dostop do pomnilnika.

Namesto podatka lahko v register zapišemo naslov podatka. Zdaj je register kazalec ("pointer") na podatek. Z nekaterimi načini naslavljanja lahko avtomatično spreminjamo vrednost v registru s tem, da dostopamo do podatka. Taka uporaba je zelo učinkovita pri delu s tabelami in s skladom.

V register lahko zapišemo tudi odmik podatka od začetka neke strukture, začetek te strukture pa opišemo na poljuben način. Takemu naslavljanju pravimo indeksno, register pa uporabljamo kot indeksni register.

Poseben pomen ima register PC ali programski števec. Tega lahko uporabljamo le kot kazalec na naslednji ukaz. Vsak podatek, ki ga zapišemo v ta register bo procesor tako tolmačil. Pri uporabi programskega števca za naslavljanje je učinek drugačen kot pri ostalih registrih.

3.4 NAČINI NASLAVLJANJA

3.4.1 Registrski način (5)

MOVW R1,R2

Pri tem načinu uporabljamo register kot akumulator, to je začasna shramba za podatek. Na ta način ne moremo uporabljati programskega števca. Če je podatek prevelik, da bi ga zapisali v en register, naprimer število s plavajočo vejico z dvojno natančnostjo, uporabi procesor še naslednji register. Na ta način ne smemo uporabljati kazalca sklada, ker se s tem implicitno vključi programski števec.

Kot primer zapišimo registrsko naslavljanje z ukazom CLRW, ki zapiše vrednost 0 v eno besedo:

```
CLRW    R3          ; Zapiše 0 v bite 0 do 15 v
                    ; registru R3.
```

MOVL R6,R9

3.4.2 Posredni registrski način (6)

REG. DEFFERED

Register je pri tem načinu kazalec na podatek. Tudi tega načina ne moremo uporabiti s programskim števcem. Primer:

```
CLRW    (R3)        ; Zapiše 0 v dva byta na naslovu,
                    ; ki je zapisan v registru R3.
```

ADDW2 R5,(R8)

AUTOINCR

3.4.3 Prištevalni način (8) - +1(B) +2(W) +4(L) +8(Q)

Prištevalni način se loči od posrednega registrskega le po tem, da se po dostopu do podatka vrednost v registru poveča za število bytov, ki jih zavzema ta podatek. Na ta način delamo s podatki, ki so zapisani v pomnilniku na zaporednih lokacijah. Ko opravimo s prvim podatkom, je v registru že naslov naslednjega.

Za primer naj bo v registru R5 vrednost 100.

```
CLRW    (R5)+       ; Zapiše 0 na naslova 100 in 101
                    ; in poveča vrednost v R5 na 102.
CLRL    (R5)+       ; Zapiše 0 na naslove 102, 103
                    ; 104 in 105, v R5 pa zapiše 106.
```

MOVB (R5)+,(R7)+

AUTOINC. DEFF.

3.4.4 Posredni prištevalni način (9)

Pri posrednem prištevalnem načinu uvedemo še eno posrednost. Že prištevalni način je posreden, ker je v registru naslov podatka in ne sam podatek, zdaj pa je v registru naslov, na katerem najdemo naslov podatka. Po obdelavi podatka se vrednost v registru poveča za 4, ker je naslov vedno dolg 4 byte in register po ukazu kaže na naslednji naslov.

Na naslovu 100 imejmo tabelo naslovov, na katerih se začnejo neki teksti. Če želimo v vsakem tekstu brisati prvi znak, bomo v register zapisali vrednost 100 in uporabili posredni prištevalni način naslavljanja. Zapišimo 100 v R4 in nato

```
CLRB    @(R4)+          ; Briše prvi znak v prvem tekstu.
CLRB    @(R4)+          ; Briše prvi znak v drugem tekstu.
  itd.
```

TABELA NASLOVOV - L

AUTODECREM

3.4.5 Odštevalni način (7)

Ta način je ekvivalenten prištevalnemu, le da se vrednost v registru zmanjša PREDEN uporabimo vsebino registra kot naslov podatka. Kombinacijo prištevalnega in odštevalnega načina uporabljamo za delo s skladom.

Organizirajmo svoj sklad, ki se začne na naslovu 1000 in se širi proti 0. Kot kazalec sklada bomo uporabili register R7. V ta namen postavimo v R7 vrednost 1000 in s tem smo inicializirali sklad. Podatke pišemo v sklad in jemljemo iz njega z ukazom MOVx; x je lahko B za byte, W za besedo ali L za dolgo besedo.

```
MOVL    R0,-(R7)        ; Zapiše na sklad vsebino R0.
MOVB    R1,-(R7)        ; Zapiše na sklad en byte iz R1.
MOVB    (R7)+,R2        ; Vzame s sklada en byte in ga
                        ; zapiše v R2.
```

CLRW -(R5)

LITERAL

3.4.6 Literal

Če je podatek majhno celo ali realno število, lahko ta podatek zapišemo kot literal. Pri tem se v byte, ki opisuje način naslavljanja, zapiše prav to število. Procesor prepozna ta način po prvih dveh bitih (6 in 7), ki morata biti 0. Za zapis podatka nam ostane še šest bitov od 0 do 5. Z njimi lahko zapišemo cela števila med 0 in 63 ali 64 različnih realnih števil, ki so zbrana v tabeli 5-2 v VAX Architecture Handbook.

```
MOVL    #7,R4           ; Zapiše število 7 v R4.
```

MOVW #25,R11

```
        MOVF    #2.25,R5        ; Zapiše realno število 2.25 v R5.
```

3.4.7 Naslavljanje z odmikom (A)

Podatek lahko naslovimo tudi tako, da uporabimo vsebino registra kot naslov, torej kot pri posrednem registrskem naslavljanju, temu pa prištejemo še neko vrednost - odmik od naslova. Posredni registrski način lahko torej sledamo kot obliko naslavljanja z odmikom 0.

Odmik lahko zapišemo kot število ali kot simbol, ki smo mu na drugem mestu priredili vrednost. Za odmike, ki se večkrat ponavljajo je pregledneje na začetku programa definirati simbol z nazornim imenom in kasneje uporabljati za odmik ta simbol namesto ustreznega števila. Tudi to pripomore k čitljivosti programa.

Če imamo v neki podatkovni strukturi od 12. do 15. byta zapisano starost, naslov te podatkovne strukture pa je v R6, lahko dobimo podatek o starosti na naslednji način:

```
STAROST = 12
        MOVL    STAROST(R6),R2  ; Zapiše starost v register R2.
        MOVL    12(R6),R2       ; Je pri izvajanju programa
                                ; popolnoma enakovredno prvi obliki.
        MOVL    L^12(R6),R2     ; Za odmik rezervira dolgo besedo,
                                ; program bo za 3 byte daljši.
```

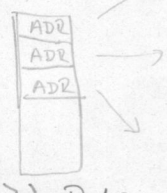
Odmik je lahko različno velik in ga lahko včasih zapišemo v en byte, včasih pa rabimo celo besedo ali celo dolgo besedo. Če sami nič ne povemo o velikosti prostora, kamor bomo zapisali odmik, izbere prevajalnik najkrajši možni prostor.

3.4.8 Posredno naslavljanje z odmikom (B)

Obstaja tudi posredna oblika naslavljanja z odmikom, pri kateri je to, kar smo pri neposrednem naslavljanju imeli za operand, naslov operanda.

Vrnimo se na primer, ko smo v tekstih brisali prvi znak. Zdaj bi radi samo v tretjem tekstu brisali prvi znak. Kazalec na začetek tabele naslovov teh tekstov naj bo v registru R3.

```
        CLRB    @2*4(R3)        ; Prištejemo naslovu v R3 2*4 byte
                                ; in s tem preskočimo prva dva
                                ; podatka (dolge besede), tretjega
                                ; pa uporabimo kot naslov.
```



Tudi pri posrednem naslavljanju z odmikom imamo tri različne kode za različno velike odmike in prevajalnik sam izbere najkrajšo možnost, če eksplicitno ne zahtevamo drugače.

3.4.9 Indeksno naslavljanje

Večino od obravnavanih načinov naslavljanja lahko še dodatno indeksiramo, kar pomeni, da dodamo končnemu naslovu za operand še neko vrednost, ki jo izračunamo iz podatka v indeksnem registru. Kot indeksni register lahko uporabimo katerikoli splošni register razen programskega števca. Če delamo z bytom, je dodana vrednost kar vsebina registra, če delamo z dalšjim podatkom, pa se vsebina indeksnega registra pomnoži z dolžino podatka v bytih.

Namesto z odmikom lahko dobimo starost iz zgornjega primera tudi z indeksnim naslavljanjem. Starost je četrta dolga beseda v podatkovni strukturi, zato moramo tri preskočiti. Naslov podatkovne strukture je še vedno v R6.

```
MOVL    #3,R5           ; Določimo vrednost indeksnega
                        ; registra R5.
MOVL    (R6)[R5],R2     ; Vsebina R5 pomnožena s 4 se
                        ; doda naslovu, ki je v R6 in to
                        ; je pravi naslov operanda.
```

Pri uporabi indeksnega naslavljanja je še nekaj omejitev. Indeksirati ne moremo registrskega naslavljanja in tudi ne naslavljanja z literali. Registrom ne ustrezajo naslovi v pomnilniku, da bi te naslove povečali za določeno število. Druga omejitev je pri naslavljanjih, ki spremenijo vrednost registra, naprimer prištevalno naslavljanje. V takem primeru ne moremo uporabiti istega registra za določanje naslovă in za indeksni register hkrati. Pri ostalih naslavljanjih lahko uporabimo isti register tudi kot indeksni in pišemo na primer:

```
CLRL    NASLOV(R7)[R7]
```

3.5 NASLAVLJANJE S PROGRAMSKIM ŠTEVCEM

Če uporabimo kot splošni register programski števec, so nekateri načini naslavljanja nesmiselni, ostali pa dobijo drug pomen. Ogledali si bomo štiri načine, ki se pogosto uporabljajo in nosijo zaradi drugačnega pomena svoja imena.

3.5.1 Takojšnje naslavljanje (8F)

IMMEDIATE

To je prištevalni način s programskim števcem kot splošnim registrom. Podatek je v tem primeru zapisan takoj za bytom z opisom načina naslavljanja. Na ta podatek kaže PC, ko procesor prebere način naslavljanja. Ko prebere podatek, popravi vrednost programskega števca in ta kaže po dostopu do podatka naslednji ukaz ali specifikacijo naslednjega operanda.

Sintaksa tega načina naslavljanja je enaka kot za literal, vendar so literali lahko le majhna števila, pri takojšnjem naslavljanju pa lahko imamo tudi do 16 bytov dolge podatke.

```
        MOVL    #12,R2          ; Takojšnje naslavljanje, vendar
                                ; bo prevajalnik naredil literal.
        MOVL    I^#12,R2        ; Zahtevamo takojšnje naslavljanje.
        MOVD    #5.64,R2        ; Takojšnje naslavljanje, podatek
                                ; zasede 8 bytov.
```

3.5.2 Absolutno naslavljanje

Pri posrednem prištevalnem naslavljanju s programskim števcem določimo absolutno vrednost naslova, s katerega bi radi dobili podatek. Tega načina navadno ne uporabljamo, ker s tem zahtevamo točno določene naslove za naše podatke. Navadno prepustimo povezovalniku, da določi v kateri del naslovnega prostora bo postavil posamezne dele kode in podatkov. Problemi nastopijo, če imamo več podprogramov, ki želijo vsi shraniti svoje podatke na iste naslove.

Za primer zapišimo vrednost 10 na naslov 12345:

```
        MOVL    #10,@#12345
```

MOVL @#VEC,R1

3.5.3 Relativno naslavljanje

RELATIVE

Če pri naslavljanju z odmikom uporabimo programski števec, bo to odmik od trenutnega naslova ali relativni odmik sledе na ukaz, ki ta podatek zahteva. Če povezovalnik prestavi del programa na druge naslove, ostanejo relativne razdalje med ukazi in podatki nespremenjene, program ni odvisen od tega, na katerih naslovih se izvaja.

Pri relativnem naslavljanju ne pišemo števila, ki pomeni odmik v bytih, ampak kar oznako vrstice, v kateri je definiran prostor za podatek. Prevajalnik in povezovalnik izračunata odmike in postavita v ukaze prave vrednosti. Kot pri naslavljanu z odmikom imamo tudi pri relativnem naslavljanju tri možnosti za specifikacijo odmika.

Prevajalnik sam izbere najkrajšo možnost, lahko pa sami zahtevamo daljšo varianto.

```
PRVI:   .LONG   123             ; Prostor za prvi podatek.
DRUGI:  .BLKL   1               ; Prazen prostor za drugi podatek.
          .
          .
        MOVL    PRVI,DRUGI      ; Prepiše prvi podatek na mesto
                                ; drugega.
```

3.5.4 Posredno relativno naslavljanje

REL. DEFF.

Razlika med neposrednim in posrednim relativnim naslavljanjem je samo v tem, da je pri posrednem naslavljanju na naslovu, čigar relativno oddaljenost od ukaza imamo podano, zapisan naslov operanda in ne sam operand.

```
TABELA: .LONG   PRVI            ; Naslov prvega podatka.
        .LONG   DRUGI           ; Naslov drugega podatka
          .
          .
PRVI:   .LONG   123             ; Prvi podatek
          .
          .
        CLRL    @TABELA         ; Briše podatek 123 na naslovu
                                ; PRVI.
```

Tudi pri naslavljanju s programskim števcem lahko uporabljamo indeksiranje, vendar ne pri takojšnjem naslavljanju, to je pri prištevalnem načinu. Za ostale tri so pravila enaka kot pri naslavljanju s splošnimi registri.

POGLAVJE 4

OSNOVNI UKAZI

4.1 UPORABA PROGRAMSKE KARTICE

V tem poglavju si bomo ogledali nekaj osnovnih ukazov, pred tem pa bomo pogledali, kako se uporablja programska kartica. Za primer vzemimo ukaz MOVL. V VAX11 Programming Card imamo za ta ukaz naslednjo vrstico:

OP	MNEMONIC	DESCRIPTION	ARGUMENTS	COND. CODES N	Z	V	C
D0	MOVL	Move long	src.rl,dst.wl	*	*	0	-

V stolpcu OP je strojna koda za ta ukaz zapisana heksadecimalno. V naslednjem stolpcu je ime ukaza, ki je sestavljeno iz generičnega imena MOV in iz znaka, ki pove, kakšen podatek premikamo. Za premikanje besede bi tako imeli ukaz MOVW. Sledi kratek opis ukaza in argumenti. Opis argumentov je iz dveh delov. Prvi del je mnemonično ime, zgoraj je to SRC za "source" ali izvor, izvorno polje in DST za "destination" ali cilj, namembno polje. Za piko je opisano, kako je ta argument uporabljen. Prvi znak pove način dostopa in je:

- a Naslov tega operanda je podatek, s katerim delamo.
- b To ni operand, temveč odmik za programski skok.
- m Operand preberemo in vanj pišemo.
- r Operand samo preberemo.
- v Isto kot a, razen pri registrskem naslavljanju. Takrat pomeni naslovljeni register in naslednji.
- w Ta operand zapisujemo.

Drugi znak pove tip podatka. Pri ukazu MOVL premikamo dolge besede (longword), zato je drugi znak l. Za različne tipe so znaki naslednji:

- b byte
- d število s plavajočo vejico z dvojno natančnostjo
- f število s plavajočo vejico

l	dolga beseda
q	quadword - podatek iz 8 bytov
v	polje bitov - le pri ukazih za delo s polji bitov
w	beseda
x	isti tip podatka, kot je prvi znak za operand v imenu ukaza
y	isti tip podatka, kot je drugi znak za operand v imenu ukaza
*	več dolgih besed - le pri posebnih ukazih

V zadnjem polju je zapisano, kako ukaz postavi pogojne bite N, Z, V in C. Znaki v tem polju so:

*	ukaz postavi pogojni bit v odvisnosti od rezultata
-	ukaz pogojnega bita ne spremeni
0	ukaz vedno briše pogojni bit
1	ukaz vedno postavi bit na 1

4.2 UKAZI ZA DELO S ŠTEVILI

4.2.1 Aritmetični ukazi

V naboru ukazov so ukazi za vse štiri osnovne aritmetične operacije, seštevanje, odštevanje, množenje in deljenje in to za različne tipe operandov, tudi za števila s pomično vejico v različnih natančnostih.

Imena ukazov so sestavljena iz generičnega imena ADD, SUB, MUL ali DIV, sledi črka, ki pove tip operandov, in število 2 ali 3, ki pove število operandov ukaza. Z ukazom DIVL3 naprimer zahtevamo, da procesor deli drugi operand s prvim in rezultat zapiše v tretjega. Vsi trije operandi so dolge besede. Ukazi z dvemi operandi zapišejo rezultat v drug operand, torej delimo na mestu. Ti dve obliki ustrezata v FORTRANu ali BASICu ukazoma C = A/B in A = A/B.

Oglejmo si še opis operandov v programski kartici. Pri ukazih s tremi operandi je način dostopa do prvih dveh R, operanda torej samo beremo, dostop do tretjega pa je W, v ta operand zapišemo rezultat. Če imamo dva operanda, prvega beremo, dostop je R, drugega pa najprej preberemo, nato pa vanj zapišemo rezultat, način dostopa je torej M.

Pogojni bit C, ki beleži prenos iz vodilnega bita, se pri večini aritmetičnih ukazov briše, nekateri, kot naprimer ADDL2, pa ga postavijo v odvisnosti od rezultata. Ostale pogojne bite nastavijo vse aritmetične operacije. V bit pomeni prekoračitev obsega števil, s katerimi delamo, naprimer pri seštevanju dveh besed dobimo rezultat večji od 32767 in ga ne moremo zapisati v besedo. N in Z bit sporočita, če je rezultat negativen oziroma nič.

Za množenje in deljenje imamo še posebne ukaze. EMUL je ukaz za razširjeno množenje. Rezultat množenja dveh dolgih besed zapiše v quadword. Ukaz ima štiri operande, prva dva množimo med seboj, tretjega pa prištejemo, preden zapišemo rezultat v četrtega z dvojno velikostjo. Ta ukaz nam pride prav, če delamo z zelo velikimi celimi števili in želimo simulirati aritmetične ukaze za daljše podatke, naprimer za 64 bitna cela števila.

Za natančnejše množenje realnih števil uporabljamo ukaz EMODx. To je množenje dveh realnih števil, od katerih je eno še podaljšano z dodatnim poljem 8 do 15 bitov odvisno od tipa realnega števila. Rezultat se zapiše v dveh delih. Posebej se zapiše celi del rezultata kot dolga beseda, ostanek pa se zapiše kot realno število.

Tudi za deljenje imamo razširjeni ukaz EDIV. Deljenec je zdaj četvorna beseda (quadword) deljitelj je dolga beseda (longword) kvocient in ostanek pa se zapišeta ločeno vsak v svojo dolgo besedo. Ukaz nudi samo to možnost za operande.

4.2.2 Ukazi za prenašanje in pretvarjanje števil

Števila prenašamo z enega mesta na drugega z ukazom MOVx. Izvir in cilj sta lahko v pomnilniku ali v registru. Za prenašanje podatkov različnih tipov zapišemo kot x B za byte, W za besedo (word) itd. Ni treba paziti, de števila s premično vejico prenašamo z ukazom MOVF, cela števila pa z MOVL. Ta dva ukaza sta namreč ekvivalentna. Ukaz za prenašanje števila ne tolmači, temveč samo prekopira določeno število bytov.

Prenašanje podatkov lahko kombiniramo z različnimi dodatnimi operacijami. Z ukazom MNEGx naprimer prenesemo število in mu hkrati spremenimo predznak. Pri tem ukazu pa moramo razlikovati med prenašanjem števila s premično vejico in prenašanjem celega števila, saj imata ta dva tipa različno zapisan predznak.

Ukaz MCOMx prenese eniški komplement podatka, se pravi, da se vse ničle v izvirnem polju pretvorijo v enice in obratno.

Ukazi za pretvarjanje podatkov iz enega tipa v drugega so po delovanju podobni ukazu MOVx, le da preneseni podatek ni enakega tipa kot izvirni. Eden izmed teh ukazov ima celo ime MOVZxy, kar pomeni prenesi podatek tipa x v daljši podatek tipa y in dodatna mesta zapolni z ničlami (move zero extended). S tem ukazom lahko pretvarjamo cela števila iz krajših oblik v daljše.

Z ukazoma CVTxy in CVTRxy (convert in convert rounded) lahko pretvarjamo tudi realna števila v cela in obratno. Razlika med ukazoma je ta, da CVT odreže decimalna mesta pri pretvarjanju realnega števila v celo, CVTR pa število zaokroži.

CMP A,B B 94 N=0
         -A 6A

TST A,0 N A<0

4.2.3 Primerjanje števil

Števila primerjamo z ukazoma CMPx in TSTx. S CMPx primerjamo dva podatka enakega tipa, TSTx pa primerja podatek s celoštevilčno ali realno ničlo. Oba ukaza ne spremenita stanja v pomnilniku ali registrih, ampak postavita le pogojne bite v statusu procesorja.

4.2.4 Ukazi za delo s posameznimi biti

Testiramo ali postavljamo lahko tudi posamezne bite ali skupine bitov v nekem podatku. Z ukazom BITx testiramo izbrane bite. Ukaz ima dva podatka, prvi je maska, ki je enakega tipa kot testirani podatek. V podatku testiramo bite na mestih, kjer so v maski zapisane enice. Z drugimi besedami lahko opišemo ukaz tako, da v podatku brišemo vse bite, ki ne ustrezajo postavljenim bitom v maski, torej naredimo logično operacijo IN med masko in podatkom, nato pa testiramo dobljeno vrednost kot z ukazom TSTx. Tudi ta ukaz podatka ne spremeni, postavi pa pogojne bite v statusu procesorja.

Z ukazoma BISx in BICx zapišemo enice ali ničle v bite, ki jih določa maska. Z ukazom BISx (bit set) naredimo torej logični ALI po bitih med masko in podatkom, z BICx (bit clear) pa naredimo logični IN med komplementom maske in podatkom. Po bitih lahko izvedemo tudi operacijo ekskluzivni ALI, ki da rezultat 1 ali točno le, ko je natančo eden od operandov 1. Ime ukaza je XORx.

V podatku lahko premaknemo vse bite v levo ali desno, s tem da pri premiku v levo izgubimo vodilne bite, če uporabimo ukaz ASHX, ali pa jih prenesemo na desni konec števila z ukazom ROTL, pri premiku v desno pa izgubimo ali prenesemo najmanj pomembne bite.

4.2.5 Razni ukazi

Ničlo lahko zapišemo na nek naslov z ukazom MOVx, hitreje in s krajšo kodo pa dosežemo isto z ukazom CLRx (clear). Podobno imamo za prištevanje in odštevanje enice ukaza INCx (increment) in DECx (decrement), ki povečata ali zmanjšata podatek za 1.

Pri klicu podprogramov pogosto prenašamo parametre na sklad. Tudi to lahko naredimo z ukazom MOVx, vendar imamo za zapis dolge besede na sklad poseben ukaz PUSHL (push longword). Za zapisovanje na sklad uporabljajo izraz "push", za jemanje podatkov s sklada pa "pop".

Zanimiv je ukaz POLYx, ki izračuna vrednost polinoma v neki točki. Koeficienti polinoma in argument so realna števila. Koeficiente polinoma podamo v tabeli, kot je natančneje opisano v Architecture Handbook na strani 213 in naslednjih.

```
ARGUMENT:
        .FLOAT  1.5
STOPNJA:
        .WORD   4
KOEFICIENTI:
        .FLOAT  1
        .FLOAT  1.5
        .FLOAT  0.5
        .FLOAT  1
        .FLOAT  2
REZULTAT:
        .BLKF   1
            .
            .
        POLYF   ARGUMENT,STOPNJA,KOEFICIENTI
        MOVF    R0,REZULTAT
```

Z ukazom POLYF smo izračunali vrednost polinoma x**4 + 1.5*x**3 + 0.5*x**2 + x + 2 v točki x = 1.5. Rezultat pusti ta ukaz v registru R0 in ga moramo sami prekopirati tja, kjer ga želimo.

4.3 KONTROLNI UKAZI

Za kontrolo toka izvajanja programa imamo dva tipa ukazov. Eno so pogojni ali brezpogojni skoki na izbran naslov, drugo pa so skoki v podprogram. Podprograme bomo natančneje obravnavali v posebnem poglavju, tukaj pa bomo govorili o skokih.

Ukaz za skok na nek naslov ima dve osnovni obliki. Prva je varianta ukaza "branch" ali razvejitev, druga je ukaz "jump" ali skok. Razlika med njima je ta, da lahko pri skoku določimo naslov, kamor skačemo na enega od splošnih načinov naslavljanja, razvejimo pa lahko program samo tako, da povemo naslov, na katerega želimo prenesti kontrolo, prevajalnik pa iz tega izračuna odmik od ukaza "branch" do naslova. Program torej razvejimo tako, da podamo odmik od tekočega ukaza.

4.3.1 Programski skok

Z ukazom JMP brezpogojno prenesemo izvajanje programa na naslov, ki smo ga določili z nekim naslavljanjem. Naslov določimo tako, da bi bil argument ukaza koda, ki jo želimo izvršiti, če bi namesto ukaza JMP uporabili kak drug ukaz, naprimer INCB. Uporabljamo lahko katerikoli način naslavljanja razen takojšnjega, registrskega in naslavljanja z literalom, ki ima enake omejitve in sintakso kot takojšnje.

CASEL 2ϕ,21,#3

LISTA: .WORD ADRϕ-LISTA
.WORD ADR1 - LISTA
.WORD ADR2 - LISTA
.WORD ADR3 - LISTA

ADRϕ:
1:
2:
3:

Naslov, na katerega želimo skočiti lahko opišemo tudi z indeksiranim naslavljanjem, ali pa naredimo tabelo naslovov in med izvajanjem določimo s podatkom v nekem registru na kateri od naslovov v tabeli želimo skočiti. Podobno uporabljamo ukaz CASEx.

4.3.2 Razvejitev programa

Program razvejimo z eno izmed oblik ukaza "branch". Brezpogojni skok je BRx, x pa je B ali W in pove ali je odmik zapisan v bytu ali besedi. Za kratke skoke lahko torej uporabimo ukaz BRB, za daljše BRW, za zelo dolge skoke, pri katerih je odmik tako velik, da ga ne moremo zapisati v 16 bitov, pa moramo uporabiti ukaz JMP.

Ukazi za pogojne skoke imajo obliko Bpogoj, pogoj pa je zapisan z 2 do 4-imi znaki. Vsi pogoji testirajo stanje pogojnih bitov v statusu procesorja. Različni pogoji in njihov pomen so natančneje opisani na straneh 261 in 262 v Architecture Handbook.

Z ukazi BBx in BBxy testiramo stanje določenega bita v podatku in razvejimo izvajanje programa, če je bit postavljen pri ukazu BBS ali če je brisan pri ukazu BBC. Daljša oblika ukaza naredi isto, s tem da ob skoku tudi postavi ali briše testirani bit. Ukaz BLBx je skrajšana oblika ukaza BBx in z njim testiramo najmanj pomemben bit (low bit).

Naslednja skupina razvejitvenih ukazov nam omogoča lahko programiranje zank. Ukaz ACBx prišteva določeno vrednost, ki jo podamo kot parameter, kontrolni spremenljivki in dokler ne doseže meje se vrača na začetek zanke. Kontrolna spremenljivka je lahko celo ali realno število. ACB so kratice za "add compare and branch", kar pomeni prištej, primerjaj in razveji.

Če spreminjamo kontrolno spremenljivko za ena in primerjamo z ničlo, lahko uporabimo ukaz AOBpogoj, če povečujemo, ali SOBpogoj, če zmanjšujemo kontrolno spremenljivko. Pogoj je lahko LSS za manjše ali LEQ za manjše ali enako, če povečujemo števec in GEQ za večje ali enako ali GTR za večje, če zmanjšujemo števec. Kot pri ukazu ACB prenaša ukaz kontrolo na začetek zanke, dokler števec izpolnjuje pogoj.

POGLAVJE 5

PODPROGRAMI

Procesor VAX11 pozna dva tipa podprogramov. Navadni podprogrami, ki jih imenujejo "subroutine", se ne ločijo od podprogramov, kot smo jih vajeni pri drugih procesorjih. Drug tip podprogramov so procedure, pri katerih operacijski sistem poskrbi za vsebino registrov, ki jih uporablja klicoči program in jih podprogram ne sme spremeniti, urejeno pa je tudi prenašanje argumentov v procedure in vračanje rezultatov v klicoči program.

5.1 PODPROGRAMI

Navadne podprograme (subroutine) kličemo z ukazi JSB ali BSBx, ki so podobni ukazom JMP in BRx. Edina razlika med ukazoma JSB in JMP je v tem, da se pri ukazu JSB shrani na sklad vrednost programskega števca ob skoku, to pa je naslov ukaza, ki je za ukazom JSB. Ta naslov uporabimo pri povratku iz podprograma z ukazom RSB.

Podprograma ne začnemo z vstopno točko .ENTRY kot glavni progam, ampak definiramo le nek globalni simbol kot oznako vrstice, kjer se začne koda podprograma, naprimer:

```
; Podprogram za seštevanje dveh polj.
; Argumenti:
;             R2 število podatkov v polju
;             R3 naslov prvega polja
;             R4 naslov drugega polja
;             R5 naslov polja za rezultat.
SESTPOLJ::
        ADDL3   (R3)+,(R4)+,(R5)+       ; Sešteje en par podatkov
        SOBGTR  R2,SESTPOLJ             ; za vse pare.
        RSB
```

Pri klicu podprograma moramo sami poskrbeti za prenos argumentov. Najpogosteje jih prenašamo v registrih, lahko pa prenesemo tudi naslov tabele, v kateri so zapisani.

Podprograme lahko kličemo le iz MACRO jezika. Višji programski jeziki vedno uporabljajo procedure. Prednost podprogramov je v tem, da je prenos kontrole v podprogram dosti hitrejši kot v proceduro in če je podprograma le par ukazov, se lahko zgodi, da pri klicu procedure sam prenos kontrole traja dalj časa kot izvajanje vseh ukazov v proceduri.

Slabost podprogramov je, da jih ne moremo klicati iz višjih programskih jezikov in da ni enotnega mehanizma za prenašanje argumentov.

5.2 PROCEDURE

Edini način za uporabo podprogramov v višjih programskih jezikih so procedure. Tudi s fortransko deklaracijo SUBROUTINE definiramo proceduro. V MAKRO kličemo proceduro z ukazoma CALLS in CALLG. Pri obeh ukazih imamo dva parametra. Prvi določa argumente, drugi pa je naslov procedure. Naslov lahko določimo kot pri ukazu JMP ali JSB, torej s katerimkoli načinom naslavljanja, ki določa podatek z naslovom (ne registrski ali takojšnji način).

5.2.1 Prenos argumentov

Za prenos argumentov v procedure obstaja dogovor, ki se ga držijo vsi višji programski jeziki. Argumente zapišemo v tabelo, katere prvi podatek je število argumentov, zapisano v dolgi besedi. V naslednjih dolgih besedah so zapisani argumenti. Razlika med ukazoma CALLS in CALLG je, da pri prvem zapišemo tabelo argumentov na sklad, pri drugem pa je tabela med podatki. V obeh primerih je naslov, na katerem se začne tabela, zapisan v registru AP. To je kazalec argumentov.

N
Argument 1
.
.
Argument N

Slika 5.1:Tabela argumentov.

Argumente lahko prenašamo z vrednostjo, z naslovom ali z opisnikom. V prvem primeru zapišemo v eno dolgo besedo samo vrednost podatka, v drugem primeru zapišemo naslov, na katerem se podatek začne, v tretjem pa zapišemo naslov opisnika, ki je sestavljen iz 4 polj s skupno dolžino 8 bytov. V prvih dveh bytih opisnika je zapisana dolžina podatka v bytih, v naslednjih dveh bytih sta podatka o tipu in vrsti podatka, sledi pa dolga beseda z naslovom, na katerem se začne podatek. Natančnejši opis standarda je v dodatku C knjige Architecture Handbook.

Vsi višji programski jeziki prenašajo argumente z naslovom, če prenašajo tekste pa z opisnikom. Z ukazi, ki so v različnih programskih jezikih različni, lahko zahtevamo, da se prenašajo argumenti na katerikoli od zgoraj omenjenih treh načinov.

Za primer si oglejmo, kako pripravimo argumente in pokličemo proceduro VAJA, ki ima za prva dva argumenta števili, ki ju prenesemo z naslovom, za tretjega pa tekst, ki ga določimo z opisnikom.

```
Klic s CALLS:
        PUSHAQ  TEKST           ; Tretji argument je naslov
                                ; opisnika teksta.
        PUSHAL  STEVILO2        ; Drugi argument je naslov
                                ; števila,
        PUSHAL  STEVILO1        ; prav tako prvi argument.
; Klic procedure VAJA. Prvi podatek je število argumentov in se
; zapiše na sklad nad ostale argumente, drugi podatek je naslov
; procedure.
        CALLS   #3,VAJA
Klic s CALLG:
; Med podatki definiramo tabelo z argumenti:
PODATKI:
        .LONG   3               ; Število argumentov.
        .ADDRESS STEVILO1       ; Prvi argument - naslov števila.
        .ADDRESS STEVILO2       ; Drugi argument - naslov števila.
        .ADDRESS TEKST          ; Tretji argument - naslov
                                ; opisnika teksta.
            .
            .
```

```
; Med ukazi je klic procedure VAJA z ukazom CALLG. Prvi podatek
; je naslov tabele argumentov, ki se bo prekopiral v register AP,
; drugi podatek je naslov procedure.
        CALLG   PODATKI,VAJA
```

5.2.2 Format procedure

Proceduro začnemo tako kot glavni program z definicijo vstopne točke z ukazom .ENTRY. Ime, ki ga pišemo v tem ukazu, je ime procedure, ki ga uporabimo v ukazih CALLS ali CALLG ali pri klicu te procedure iz višjega programskega jezika. Drugi parameter pri ukazu .ENTRY je 16-bitna maska, ki pove, katere registre želimo shraniti. Bit 0 pomeni register R0, bit 1 register R1 itd. V masko zapišemo registre, ki jih uporabljamo v proceduri. Pri klicu procedure se vrednosti teh registrov zapišejo na sklad in ob izhodu iz procedure dobijo registri spet tiste vrednosti, ki so jih imeli pred klicem procedure. Klicočemu programu procedura torej ne pokvari stanja v registrih.

Proceduro končamo tako, da v register R0 ali v R0 in R1 zapišemo status, s katerim se je izvajanje procedure končalo, če uporabljamo proceduro kot funkcijo, pa v teh registrih vrnemo vrednost funkcije. Status uspešno ima simbolično ime SS$NORMAL, številska vrednost tega simbola pa je 1. Status zapišemo v register R0 z ukazom

```
        MOVL    #SS$_NORMAL,R0
```

pred ukazom RET, s katerim vrnemo kontrolo v klicoči program.

Začetek in konec procedure sta enaka kot pri glavnem programu, ki je tudi neka procedura. Kaj je glavni program in kaj podprogram določimo z imenom pri ukazu .END. To ime pove, na kateri vstopni točki se bo začel izvajati program, ko ga aktiviramo z DCL ukazom RUN. Ukaz .END mora biti na koncu vsake datoteke z makro programom ali delom makro programa, vendar pišemo ime vstopne točke le pri enem.

5.2.3 Uporaba argumentov v proceduri

Pri klicu procedure se zapiše naslov, na katerem je začetek tabele argumentov v register AP. Do podatkov pridemo torej v proceduri preko registra AP. Če želimo argumente procedure VAJA prenesti v registre tako, da bosta števili, ki sta prva argumenta, v registrih R2 in R3, naslov teksta pa v R4, bomo to naredili tako:

```
; Procedura VAJA
; Argumenti so:     začetna pozicija izbranega podteksta
;                   dolžina izbranega podteksta
```

```
;                    naslov teksta, iz katerega je podtekst.
        .ENTRY  VAJA    ^M<R2,R3,R4>    ; Vstopna točka.
        MOVL    4(AP),R2                ; Prvo število gre v R2,
        MOVL    8(AP),R3                ; drugo število gre v R3,
        MOVAL   12(AP),R4               ; naslov teksta gre v R4.
```

V registru AP je naslov začetka tabele, 4(AP) torej določa prvi podatek v tabeli. Ker je to naslov števila, mi bi pa radi samo število, moramo uporabiti posredno naslavljanje, torej @4(AP). Če želimo uporabiti kot podatek naslov spremenljivke, naprimer teksta, lahko to dosežemo tako, da naslovimo ta argument neposredno, lepše in preglednejše pa je, če naslov prenesemo z ukazom MOVAL, naslovimo pa spet sam podatek.

5.2.4 Izbira med ukazoma CALLS in CALLG

Obe obliki klica procedure imata prednosti in pomanjkljivosti. Za klic s CALLG lahko pripravimo argumente že med prevajanjem z ukazi .LONG ali .ADDRESS in med izvajanjem izvršimo le ukaz CALLG. V tem primeru je klic s CALLG hitrejši. Druga prednost je to, da lahko isto tabelo argumentov večkrat uporabimo za klice procedur s podobnimi argumenti, če spremenimo enega ali več podatkov v tabeli. S tem pa smo že izgubili prednost, da pripravimo argumente ob prevajanju.

Slabost tega načina klica procedur je, da tabele zasedajo prostor v pomnilniku tudi ko niso več potrebne. Pri klicu s CALLS se temu izognemo tako, da argumente zapišemo na sklad in ko se izvajanje procedure konča, operacijski sistem poskrbi za to, da pobere argumente s sklada.

5.3 REKURZIJA

Procedure in podprograme (subroutine) lahko uporabljamo tudi rekurzivno, to pomeni, da procedura kliče samo sebe. Pri uporabi rekurzivnih procedur moramo paziti na argumente, saj bomo pri klicu procedure s CALLG uporabljali vedno isto tabelo argumentov in tako ima procedura na voljo svoje argumente le dokler ne pokliče same sebe. Tega problema ni pri klicu procedure s CALLS, saj ima v vsaki globini procedura svojo kopijo argumentov.

Tudi če uporabljamo rekurzivno pravi podprogram, je najbolje prenašati argumente na skladu.

Preprost primer uporabe rekurzije je računanje faktoriele. Za cela števila je faktoriela (N), kar zapišemo z N!, enaka produktu vseh celih števil od 1 do N. Faktoriela števila 4 je torej 1*2*3*4. Rekurzivno definiramo faktorielo celega števila z enačbami:

$$0! = 1$$
$$N! = N * (N - 1)!$$

Proceduro za računanje faktoriele bomo napisali kot funkcijo, katere argument je število N, rezultat, ki ga vrne, pa je faktoriela tega števila. Držali se bomo zgornjih dveh definicij.

```
; Procedura FAKT izracuna faktorielo celega števila N, ki je
; edini parameter. Kličemo jo kot funkcijo z enim argumentom.
        .ENTRY  FAKT    ^M<>            ; Vstopna točka.
        TSTL    @4(AP)                  ; Testiramo vrednost N.
        BNEQ    10$                     ; _Če je N različno od 0
                                        ; računaj dalje,
        MOVL    #1,R0                   ; če je nič pa vrni v R0
        RET                             ; vrednost 1.
10$:    PUSHL   @4(AP)                  ; Zapisi N na sklad in ga
        DECL    (SP)                    ; zmanjšaj za ena.
        PUSHL   SP                      ; Zapiši na sklad naslov
                                        ; argumenta N - 1.
        CALLS   #1,FAKT                 ; Rekurzivno kliči sebe.
        MULL2   @4(AP),R0               ; Pomnoži z N faktorielo
                                        ; števila N - 1.
        RET                             ; Konča z vrednostjo N! v
                                        ; registru R0.
```

V zgornjem primeru smo zapisali podatek na sklad, da smo lahko prenesli njegov naslov v proceduro. Tega podatka nismo vzeli s sklada, vendar po povratku iz procedure tega podatka ne bo na skladu, ker ukaz RET briše s sklada "call frame" in vse, kar je bilo zapisano kasneje. Tudi argumente, ki smo jih prenesli v proceduro na skladu (z ukazom CALLS), ukaz RET ob vrnitvi iz procedure briše s sklada.

5.4 KORUTINE

Zanimiv primer uporabe pravih podprogramov so korutine. V višjem programskem jeziku jih ne moremo uporabljati, ker v korutinah prenašamo kontrolo z ukazom JSB in posebnim načinom naslavljanja.

Korutina je podprogram, ki se izvaja po delih vzporedno z nekim drugim programom. Izmenično se prenaša kontrola iz enega podprograma v drugega in nazaj.

Primer korutine je podprogram, s katerim spravimo vsebino nekaj registrov na sklad na začetku podprograma in jih na koncu vrnemo. Prednost takšne korutine je, da na začetku podprograma povemo, v katerih registrih nočemo pokvariti vrednosti in nam ni treba paziti, da bomo pred koncem podprograma vrednosti res vrnili v registre.

```
; Korutina, ki shrani vrednosti določenih registrov na sklad in
```

```
; te vrednosti vrne v registre ob koncu podprograma.
; V registru R0 pričakuje masko za registre, ki jih mora
; shraniti.
TMP:    .BLKL   1               ; Začasna shramba.
SPRAVI::
        MOVL    (SP)+,TMP       ; Na vrhu sklada mora ostati
                                ; isti podatek, spravimo ga
                                ; v TMP.
        PUSHR   R0              ; Spravi registre na sklad.
        PUSHL   R0              ; Spravi tudi masko.
        PUSHL   TMP             ; Registre smo zapisali pod
                                ; zgornji podatek na skladu.
        JSB     @(SP)+          ; Vrni kontrolo v klicajoči
                                ; podprogram.
        TSTL    (SP)+           ; Tu je vstopna točka ob koncu
                                ; podprograma. Preskoči masko, ki
                                ; je zapisana na vrhu sklada.
        POPR    -4(SP)          ; Vrni vrednosti v registre, kot
                                ; masko uporabi vrednost, ki je
                                ; bila prej na vrhu sklada.
        RSB                     ; Konec korutine.
```

Korutino kličemo tako:

```
        .
        .
        MOVL    #^M<R5,R7...R10>,R0
        JSB     SPRAVI
        .
        .
        RSB
```

V podprogramu torej zapišemo masko v register R0 in skočimo v korutino z JSB. Pri tem ukazu spravimo na sklad le naslov ukaza, ki je za JSB SPRAVI. V korutini izvršimo štiri ukaze in z JSB @(SP)+ določimo, da je naslov podprograma, ki sa kličemo, zapisan na skladu. To je naslov ukaza, ki je za klicem korutine. Ker smo uporabili prištevalni način, se ta podatek briše s sklada, na sklad pa se zapiše naslov ukaza, ki sledi JSB v korutini. Ko se podprogram konča, ne vrne kontrole v klicoči program, ampak najprej v korutino, ki vrne vrednosti s sklada v registre in šele ukaz RSB v korutini vrne kontrolo v klicoči program.

PUSHR #^M<R0,R1---R5>
POPR #^M<R0,R1---R5>
#^B<1111110---7

POGLAVJE 6

PRIMERI UKAZOV

Procesor VAX11 ima še mnogo raznovrstnih ukazov. Natančen opis vseh je v knjigi Architecture Handbook. V tem poglavju bomo natančneje obravnavali le posamezne primere ukazov iz različnih skupin.

6.1 UKAZI ZA DELO S TEKSTI

Ukazi za delo s teksti izvajajo določene operacije z zaporedjem do 65535 bytov. Taki ukazi so torej lahko zelo dolgotrajni in če bi tudi za te ukaze veljalo, da ne smemo prekiniti njihovega izvajanja, bi to lahko zelo zmanjšalo odzivnost sistema. Ti ukazi so organizirani tako, da jih lahko procesor prekine med izvajanjem, vendar mora biti stanje tega ukaza ob prekinitvi točno določeno. Trenutno stanje izvajanja je opisano z vsebino registrov R0 do R5. Registri s sodo številko se uporabljajo kot števci preostalih znakov v tekstih, registri z liho številko pa so kazalci na znak v tekstu, ki je trenutno na vrsti. Če delamo z ukazi za delo s teksti moramo torej paziti, da ne pozabimo v registrih R0 do R5 važnih podatkov. Pred ukazi za delo s teksti lahko prekopiramo vsebino teh registrov na sklad, po ukazih pa prepišemo s sklada podatke v registre R0 do R5.

Za prvi primer izberimo preprost ukaz MOVC3, ki prepiše z enega mesta na drugo zaporedje bytov. Ukaz ima tri parametre, ki so po vrsti dolžina niza, naslov izvornega polja in naslov namembnega polja. Dolžina niza določa koliko zaporednih bytov želimo prenesti. Ukaz ne interpretira vsebine, ki jo prenaša. Tako lahko s tem ukazom prenašamo kakršnokoli neprekinjeno podatkovno strukturo. Dolgo besedo lahko naprimer prenesemo z ukazom MOVL ali MOVC3. Oba naslednja ukaza preneseta dolgo besedo:

```
MOVC3   #4,IZVIR,CILJ
MOVL    IZVIR,CILJ
```

Pri zgornjem ukazu MOVC3 se spremeni tudi vsebina registrov R0 do R5. V registrih R0, R2 in R3 so ničle, to so števci preostalih znakov. V registru R1 je naslov prvega znaka za izvornim poljem, torej IZVIR + 4, v registru R3 je naslov prvega znaka za ciljnim poljem, torej CILJ + 4, v R5 pa je tudi ničla.

Nekoliko zahtevnejši je primer z ukazom MOVTUC (Move translated until character). Ta ukaz prenese zaporedje znakov, vendar jih sproti še prevaja s pomočjo tabele, ki jo sami določimo. Če naleti na znak, ki smo ga določili kot terminator, prekine prenos prevedenih znakov.

Ukaz MOVTUC ima šest parametrov. Prva dva sta dolžina in naslov izvirnega teksta, tretji je terminator ali ubežni znak, četrti je naslov tabele, s pomočjo katere prevaja tekst, zadnja dva pa sta dolžina in naslov namembnega polja. Tabela je zaporedje 256 bytov. Prevajanje poteka tako, da se ASCII koda znaka, ki je na vrsti, uporabi kot zaporedna številka byta v tabeli, vsebina tega byta pa je prevedeni znak.

Oglejmo si to na primeru:

```
TEKST:   .ASCIZ  /To je tekst!/
DOLZINA = . - TEKST
PREVOD:  .BLKB   100
TABELA:  .BYTE   32, 32, 32, ...
             .
             .
             .
         MOVTUC  #DOLZINA,TEKST,#32,TABELA,#100,PREVOD
```

Z ukazom .ASCIZ zapišemo v pomnilnik tekst, ki se konča z ASCII znakom s kodo 0. Za ciljno polje smo rezervirali 100 bytov na naslovu PREVOD. Z ukazom MOVTUC prenesemo izvorni tekst tako, da najprej uporabimo ASCII kodo črke "T", to je 84, kot zaporedno številko byta v tabeli. Podatek, ki ga dobimo na naslovu TABELA+84, primerjamo z ubežnim znakom in če je različen ga prenesemo na naslov PREVOD. Isto ponovimo za naslednje znake, dokler ne najdemo ubežnega znaka ali ne porabimo vseh vhodnih ali izhodnih znakov.

Stanje v registrih R0 do R5 je naslednje: v R0 je število znakov, ki jih nismo prenesli, vključno z znakom, ki smo ga prevedli v ubežni znak, v R1 je naslov byta za zadnjim prenesenim znakom, v R2 je nič, v R3 je naslov tabele, v R4 je število neporabljenih bytov v ciljnem polju, v R5 pa naslov naslednjega prostega byta v ciljnem polju.

Kako smo končali prenos znakov izvirnega teksta, lahko preverimo na dva načina. Le če prenesemo vse znake, je po končanem ukazu v registru R0 ničla. Če je zadnji znak povzročil prekinitev prenosa, bo ostala v R0 enica. Ugotovimo lahko tudi, če je prenos prekinil ubežni znak. V tem primeru se zapiše v pogojni bit V v statusu procesorja enica, v nasprotnem primeru je ta bit brisan.

6.2 UKAZI ZA DELO S PAKIRANIMI DECIMALNIMI ŠTEVILI

Za delo s pakiranimi decimalnimi števili imamo podobne ukaze kot za delo z binarnimi. To so osnovne aritmetične operacije – seštevanje, odštevanje, množenje in deljenje, premiki števil v levo ali desno, kar pomeni množenje ali deljenje s potenco števila deset, imamo pa tudi ukaze za pretvarjanje števil iz ene oblike v drugo, binarno v pakirano decimalno in obratno ali pakirano decimalno v nepakirano in nazaj.

Kot primer si oglejmo, kako pretvarjamo decimalna števila, ki jih preberemo z datoteke ali terminala v binarni zapis. Nimamo ukaza, ki bi neposredno pretvoril decimalno število v binarno, tako da pretvorimo najprej decimalno število v pakirano decimalno in to naprej v binarno:

```
DEC_STEVILO:
        .ASCII  /+12345/
PAK_DEC_STEV:
        .BLKB   10
BIN_STEVILO:
        .BLKL   1
          .
          .
        CVTSP   #5,DEC_STEVILO,#10,PAK_DEC_STEV
        CVTPL   #10,PAK_DEC_STEV,BIN_STEVILO
```

V ukazu CVTSP sta prva dva parametra dolžina in naslov decimalnega števila, naslednja dva pa dolžina in naslov pakiranega števila. Dolžina števila je v obeh primerih število decimalnih mest. Zgornji ukaz torej pretvori petmestno decimalno število z vodilnim predznakom na naslovu DEC_STEVILO v pakirano decimalno število, ki se zapiše v deset polovičk byta z dodatno polovičko za predznak. Z naslednjim ukazom pretvorimo pakirano decimalno število v binarno, ki se zapiše v dolgo besedo.

Kot drug primer si oglejmo premik pakiranega decimalnega števila. Z ukazom ASHP (arithmetic shift and round packed) premaknemo pakirano decimalno število za izbrano število decimalnih mest v levo ali desno. Pri premiku v desno število zaokrožimo na način, ki ga sami določimo.

Ukaz ima štiri parametre. Prvi pove, za koliko mest bomo premaknili število. Če je ta parameter pozitiven, pomeni premik v levo, če je negativen pomeni premik v desno. Druga dva parametra sta dolžina in naslov pakiranega decimalnega števila, četrti parameter pa pove, kako želimo zaokrožiti premaknjeno število. Zadnja dva parametra sta dolžina in naslov ciljnega polja.

```
PAK_DEC_STEV:
        .PACKED 123456
ZAOK_REZULTAT:
        .BLKB   10
REZULTAT:
```

```
        .BLKB   10
ZAOKROZITEV:
        .BYTE   5
          .
          .
        ASHP    #-1,#6,PAK_DEC_STEV,ZAOKROZITEV,#6,ZAOK_REZULTAT
        ASHP    #-1,#6,PAK_DEC_STEV,#0,#6,REZULTAT
```

S prvim ukazom ASHP premaknemo pakirano število za eno mesto v desno, kar pomeni deljenje z deset. Pred premikom prištejemo številu četrti parameter in obdržimo po premiku le celi del. Če je parameter, ki opisuje zaokrožitev, 5, bomo zaokrožili navzgor vsa števila, ki imajo na zadnjem izgubljenem mestu 5 ali več. Če je ta parameter 3, bomo naprimer zaokrožili navzgor števila, ki imajo na zadnjem izgubljenem mestu 7 ali več. Z drugim ukazom bomo premaknili število brez zaokroževanja, saj bomo prišteli 0.

6.3 UKAZI ZA DELO Z VRSTAMI

Za delo z vrstami imamo ukaze za vstavljanje in brisanje podatkov v vrstah. Vrste smo na kratko opisali v poglavju o podatkovnih tipih, zdaj si bomo ogledali le primer za kreiranje vrste.

```
GLAVA:  .ADDRESS GLAVA              ; Kazalec na prvi podatek vrste.
        .ADDRESS GLAVA              ; Kazalec na zadnji podatek.
; Če je vrsta prazna, kažeta oba kazalca na glavo vrste.
PODATEK1:
        .BLKL   2                   ; Dve dolgi besedi za kazalca.
        .ASCIC  /Prvi podatek./     ; Vrednost podatka.
PODATEK2:
        .BLKL   2                   ; Dve dolgi besedi za kazalca.
        .ASCIC  /Drugi podatek./
PODATEK3:
        .BLKL   2                   ; Dve dolgi besedi za kazalca.
        .ASCIC  /Tretji podatek./
          .
          .
        INSQUE  PODATEK1,GLAVA      ; Vstavimo prvi podatek za glavo.
        INSQUE  PODATEK2,GLAVA      ; Vstavimo drugi podatek za
                                    ; glavo, torej pred prvega.
        INSQUE  PODATEK3,PODATEK2   ; Tretji podatek damo za
                                    ; drugega in pred prvega.
; Vrstni red podatkov v vrsti je:
;       PODATEK2
;       PODATEK3
;       PODATEK1
```

Zgornji ukazi spremenijo le vrednosti kazalcev in ne premeščajo samih podatkov. Na naslovu GLAVA je po teh ukazih naslov PODATEK2, to je prvi podatek v vrsti, na naslovu GLAVA + 4 pa je naslov PODATEK1, to

je zadnji podatek v vrsti.

6.4 POSEBNI UKAZI

Od ostalih ukazov si oglejmo še ukaz PUSHx, ki zapiše podatek ali več podatkov na sklad in ukaz INDEX, ki izračuna naslov podatka v enodimenzionalnem polju, če podamo indeks tega podatka.

Imamo dve obliki ukaza PUSH. Z ukazom PUSHL zapišemo na sklad dolgo besedo, z ukazom PUSHAx pa naslov podatka, ki je lahko byte, beseda, dolga beseda itd. V vseh teh primerih se zapiše na sklad naslov, ki je dolga beseda. Kako velik je podatek je važno le pri indeksnem naslavljanju, ker se s pomočjo dolžine izračuna naslov podatka. Če je vrednost v registru R7 10, zapiše prvi ukaz na sklad vrednost TABELA+4*10, drugi pa vrednost TABELA+2*10:

```
        PUSHAL  TABELA R7
        PUSHAW  TABELA R7
```

Druga vrsta ukaza PUSH je PUSHR. Z njim zapišemo na sklad vsebine registrov, ki jih določimo z masko. Ukaz ima en parameter in to je beseda, v kateri vsak bit pomeni en register. Če je v bitu N zapisana enica, to pomeni, da želimo shraniti na sklad vrednost registra RN.

```
MASKA:  .WORD   12              ; Maska s postavljenima bitoma 2
                                ; in 3.
            .
            .
        PUSHR   MASKA           ; Zapiše na sklad vsebino
                                ; registrov R2 in R3.
```

Z ukazom INDEX izračunamo naslov podatka. Ukaz ima šestparametrov. Prvi je indeks, naslednja dva sta spodna in zgornja meja za indeks, četrti je velikost podatka, peti je vrednost indeksa, od katerega štejemo podatke, zadnji pa je odmik od začetka polja.

```
INDEX:  .LONG   5
VELIKOST:
        .LONG   14              ; Velikost podatka je 14 bytov.
            .
            .
        INDEX   INDEX,#4,#8,VELIKOST,#-4,R4
10$:    TSTB    PODATKI(R4)+
        BNEQ    10$
```

Zgornji ukaz najprej testira, če je vrednost indeksa med 4 in 8 in ker je, prišteje vrednosti indeksa -4, kar pomeni, da je indeks prvega podatka 4, nato pa rezultat pomnoži z velikostjo podatka. Dobljeno

vrednost zapiše v register R4. Z naslednjim ukazom začnemo testirati iskani podatek po bytih.

6.5 PRIVILEGIRANI UKAZI

Nekaj ukazov je privilegiranih, kar pomeni, da jih lahko uporabljamo le v nekaterih načinih dela. Ukaz HALT npr., ki zaustavi delovanje procesorja, lahko uporabljamo le v načinu "kernel". Ukazi so zaščiteni zato, ker lahko z njimi vplivamo na delovanje celotnega sistema in z nepravilno uporabo motimo delo drugih uporabnikov.

Z ukazom MFPR (move from processor register) prekopiramo vsebino izbranega internega privilegiranega registra na nek naslov v pomnilnik. Prvi parameter ukaza je številka registra, te so opisane v knjigi Architecture Handbook na strani 166 in 167. Drugi parameter je naslov, kamor bomo prenesli vrednost registra. Z ukazom

```
MFPR    #9,P0LR
```

prekopiramo na naslov P0LR vsebino registra 9, to je register, v katerem je zapisano število strani v P0 delu virtualnega pomnilnika. Na ta način lahko torej ugotovimo velikost našega programa.

Zanimiva je tudi skupina ukazov CHMx. Ti ukazi spremenijo način dela procesorja in z njimi preidemo v bolj privilegirani način dela. Parameter tega ukaza je beseda s kodo, ki pove, katero funkcijo želimo izvršiti v drugem načinu dela. Kode s pozitivno vrednostjo pomenijo funkcije, ki so del operacijskega sistema, za kode z negativno vrednostjo pa definira funkcije uporabnik.

POGLAVJE 7

UKAZI OČIŠČEVALNIKA

V tem dodatku so zbrani najuporabnejši ukazi očiščevalnika s kratkimi opisi. Obvezni del ukaza je napisan z velikimi črkami, neobvezni del pa z majhnimi.

HELp — Pomoč.

EXIt — Izhod iz očiščevalnika (isto dosežemo s CTRL/Z).

Go — Začne izvajanje programa na tekočem naslovu in nadaljuje do prekinitvene točke ali do konca.

Step n — Izvrši naslednji ukaz ali več ukazov, če je naveden n. S kvalifikatorji določimo način izvajanja.

Examine a — Prikaže vsebino pomnilnika na naslovu a. S kvalifikatorji določimo obliko izpisa. Kot naslov lahko pišemo več naslovov ločenih z vejicami, ali interval naslovov z dvopičjem (100:120).

Deposit A=N — Zapiše na naslov A vrednost N. S kvalifikatorji določimo vrsto in velikost podatka.

SEt — Postavi karakteristike dela z očiščevalnikom ali postavi prekinitveno točko. Parametri so:

LAnguage	J	določi jezik J
MODULe	m	vključi definicije simbolov modula m
Step		določi način izvajanja po korakih
Break	A	določi prekinitveno točko A
Watch	A	zahteva prekinitev izvajanja programa, ko se spremeni vsebina na naslovu A

SHow — Prikaže karakteristike dela z očiščevalnikom. Parametri so skoraj enaki kot pri ukazu SEt.

CANcel — Prekliče vrednosti, ki smo jih določili s SEt.

DODATEK A

V dodatku so zbrani programi, ki služijo za primere na uvodnem tečaju programiranja v MACRO32 zbirnem jeziku.

```
        .title  Nacini naslavljanja.
podat:  .long   123456
st1:    .long   12
st2:    .float  2.876
st3:    .long   5555555

        .entry  START   ^m< >
        clrl    r3              ; Registrski nacin - koda 5.
                                ; Koda ukaza CLRL je D4, parameter
                                ; pa je dolocen s kodo 53 (register R3).
        movl    #512,r4         ; Takojsnje naslavljanje za prvi
                                ; parameter in registrsko za drugi.
        clrl    (r4)            ; Posredni registrski nacin - koda 6.
                                ; Brisemo podatek na naslovu 512; ta
                                ; naslov smo zapisali v R4 s prejsnjim
                                ; ukazom.
        movl    #st1,r4         ; V R4 zapisemo naslov podatka st1.
                                ; Isto bi dosegli z ukazom
                                ; MOVAL st1,r4.
        movl    #^X208,-(r4)    ; V takojsnjem naslavljanju smo zapisali
                                ; helsadecimalno stevilo. Vrednost v R4
                                ; zmansamo za 4 in novi podatek je naslov,
                                ; kamor bomo zapisali podatek.
        moval   st3,(r4)+       ; Naslov st3 zapisemo na naslov, ki je v
                                ; R4 in nato vrednost registra povecamo
                                ; za 4.
        tstw    -(r4)           ; Odstevalni nacin. Vrednost registra
                                ; zmanjsamo za 2 (beseda ima 2 byta) in
                                ; rezultat uporabimo kot naslov operanda.
        clrb    @(r4)+          ; Posredni pristevalni nacin. Vsebina R4
                                ; je naslov, na katerem najdemo naslov
                                ; operanda. Naknadno se vrednost v registru
                                ; poveca za 4 (naslov ima vednos 4 byte).
```

```
        movl    #4,r4
        clrb    st1(r4)             ; Relativni nacin - kode 10, 12, 14.
                                    ; Naslov operanda je vsota vsebine registra
                                    ; R4 in vrednosti simbola st1.
        movl    #5,@st1(r4)         ; Posredno relativno naslavljanje - kode
                                    ; 11, 13, 15. Vsota st1 + R4 je zdaj naslov,
                                    ; na katerem je zapisan naslov operanda.
;
;       Naslavljanje s programskim stevcem.
;
        movl    r2,@#516            ; Absolutni nacin - vsebino registra R2
                                    ; zapise na absolutni naslov 516.
        clrl    st2                 ; Relativno naslavljanje. Prevajalnik
                                    ; izracuna kako daleč mora skociti od
                                    ; tekocega ukaza, da pride do naslova st2
                                    ; in to vrednost uporabi kot odmik pri
                                    ; relativnem naslavljanju.
        movl    #1111111,@podat     ; Posredno relativno naslavljanje - enako
                                    ; kot prej, le da na naslovu podat dobimo
                                    ; naslov operanda.
        movl    #1,r0               ; Takojsnje naslavljanje, v registru R0
                                    ; sporocimo status, s katerim se je koncal
                                    ; program ali procedura.
        ret
        .end    start


        .title  Indeksirano naslavljanje.
;       Program ilustrira razliko med indeksiranim in relativnim
;       naslavljanjem (naslavljanje z odmikom).
tabela: .long   6                   ; Tabela ima kot prvo vrednost
        .long   1,2,3,4,5,6         ; stevilo podatkov, sledi pa
                                    ; pravo stevilo vrednosti.

        .entry  start   ^m( )
        movl    #4,r2
        movl    tabela[r2],r3       ; V R3 prepisemo 4. vrednost
                                    ; iz tabele (stevilo 4).
        movl    tabela(r2),r4       ; V R4 prepisemo vrednost z
                                    ; naslova tabela+4, to je prva
                                    ; vrednost iz tabele (stevilo 1).
        movl    #1,r0
        ret
        .end    start
```

```
        .title  Simboli.
;       Program ilustrira pomen relokatibilnih in absolutnih
;       simbolov. Relokatibilni so tisti, ki so definirani
;       relativno glede na zacetek modula. Povezovalnik takim
;       simbolom pristeje naslov, na katerega je povezal
;       zacetek modula. Ce dva relokatibilna simbola sestejemo,
;       bo povezovalnik le enkrat pristel zacetni naslov.
aa=10                                   ; Definiramo simbol aa z vrednostjo 10.
        .blkl   3
lab:    .long   5+(3*aa)                ; Na naslov lab zapisemo 35.
tabela: .long   aa/3                    ; Na naslov tabela zapisemo 3
        .long   lab+aa                  ; Sem se zapise naslov, ki je 10
                                        ; bytov za naslovom lab.
                                        ; (celostevilcni kvocient).
        .long   lab+tabela              ; Tu bi morala biti vsota naslovov,
                                        ; vendar ni, ker linker le enkrat
                                        ; relocira simbole.
        .entry  start   ^m()
        clrl    lab+tabela              ; Enaka napaka kot pri prejsnjem
                                        ; sestevanju oznak vrstic.
        $exit_s #ss$_normal
        .end    start
```

```
          .title  case
;
;         Testni programcek za uporabo CASE ukaza.
;         Namenjen je za uporabo z ociscevalnikom.
;         V registra R2 in R3 zapisujemo razlicne
;         vrednosti in opazujemo, na kateri naslov
;         prenese kontrolo ukaz CASEL.
;         Vrednost R2 - R3 pove, na katerem naslovu
;         po vrsti bomo nadaljevali. 0 pomeni prvi
;         naslov, 1 drugi ... 3 pomeni cetrti naslov.
;         V registru R0 dobimo vrednost 1 za skok na
;         naslov prvi, 2 za drugi itd.
;
;         Program prevedemo z ukazom MACRO/DEBUG CASE .
;         in ga povezemo z ukazom     LINK/DEBUG CASE.
;
          .entry  case    ^m< >
zac:      clrl    r2
          clrl    r3
;         L v ukazu CASEL pomeni, da sta prva dva operanda dolga
;         besedi (longword), odmiki, ki so zapisani za ukazom CASEx,
;         pa so vedno besede (word).
          casel   r2,r3,#3
lista:    .word   prvi-lista,drugi-lista,tretji-lista,cetrti-lista
;         Ce je razlika vrednosti R2 - R3 vecja od tretjega parametra
;         (3 v nasem primeru), se program nadaljuje za ukazom CASE
;         (preskoci tudi tabelo odmikov).
napaka:   movl    #-1,r0
          brb     zac
prvi:     movl    #1,r0
          brb     zac
drugi:    movl    #2,r0
          brb     zac
tretji:   movl    #3,r0
          brb     zac
cetrti:   movl    #4,r0
          brb     zac
          .end    case
```

```
        .title  Test fortranskih podprogramov INPUT  in OUTPUT.
outtekst:       .ascii  /Vnesi nek tekst: /
outstev:        .long   . - outtekst                    ; Dolzina teksta.
                .ascii  /To se ne izpise./
stevodgovor:    .long   intekst - odgovor               ; Dolzina odgovora.
odgovor:        .ascii  /Napisal si: /                  .
intekst:        .blkb   100                             ; 100 bytov prostora
                                                        ; za vnos teksta.
instev:         .long   0                       ; Prostor za dolzino prebranega
                                                ; teksta.


        .entry  test    ^m<>
        pushal  outtekst                        ; Naslov zacetka teksta na sklad.
        pushal  outstev                         ; Naslov dolzine niza na sklad.
        calls   #2,output                       ; Klice podprogram OUTPUT z dvema
                                                ; parametri (naslovi so na skladu).

        pushal  intekst
        pushal  instev
        calls   #2,input                        ; Isto za INPUT.
        addl2   instev,stevodgovor              ; Dolzino teksta odgovor povecamo
                                                ; za dolzino prebranega teksta.
                                                ; Prebrani tekst smo namenoma
                                                ; zapisali tik za tekstom odgovor.

        pushal  odgovor
        pushal  stevodgovor
        calls   #2,output                       ; Izpise podaljsani odgovor (skupaj
                                                ; s prebranim tekstom).

        movl    #1,r0
        ret
        .end    test
```

```
        .title  Fibonnacijeva stevila.
;
;       Podprogram izracuna n-to Fibonaccijevo stevilo Fn.
;       Ima dva parametra, prvi je n, drugi pa je vrednost
;       tega Fibonaccijevega stevila.
;       Rezultat vrne tudi v registru R0, da ga lahko uporabljamo
;       kot podprogram ali funkcijo z imenom IFIB.
;       Za Fibonaccijeva stevila velja :
;       F0 = 0
;       F1 = 1
;       Fn = Fn-1 + Fn-2.
;
        .entry  ifib    ^m<r2,r3>
        movl    4(ap),r3        ; Prvi parameter je n (zaporedno
                                ; stevilo Fibonaccijevega stevila).
        beql    20$             ; n = 0 => vrni rezultat 0.
;       Rekurzivni podprogram bomo napisali kot rutino (kontrolo
;       prenesemo z bsb ali jsb), da bo izvajanje hitrejse.
        jsb     fibonacci
5$:     cmpl    (ap),#1         ; Ali ima procedura IFIB samo en
                                ; parameter (to pomeni, da smo jo
                                ; klicali kot funkcijo)?
        beql    10$
        movl    r0,@8(ap)       ; NE - zapisi rezultat v drugi
                                ; parameter.
10$:    ret                     ; DA - rezultat je samo v R0.

20$:    clrl    r0              ; Konec za n = 0.
        brb     5$
;       Pravi podprogram (routin) za rekurzivno racunanje
;       Fibonaccijevih stevil.
;       Vhodni parameter je n v registru R3,
;       izhodni parameter je vrednost Fibonaccijevega
;       stevila v registru R0.
fibonacci:
        decl    r3              ; Ali smo ze prisli do n = 2 (Fn = 1)?
        cmpl    r3,#1
        bgtr    10$             ; NE - poklici podprogram FIBONACCI
                                ; z za 1 manjso vrednostjo v R3.
        clrl    r1              ; DA - pripravi vrednosti za F0 in F1
        movl    #1,r0           ; v registrih R1 in R0 in pojdi na
        brb     20$             ; racunanje F2 (na naslovu 20$).

10$:    jsb     fibonacci
20$:    movl    r0,r2           ; Racunanje Fibonaccijevega stevila.
        addl2   r1,r0           ; V R0 je vedno nazadnje izracunano
        movl    r2,r1           ; Fibonaccijevo stevilo, v R1 pa
                                ; predzadnje.
;       Z ukazom RSB vracamo kontrolo na ukaz, ki je za klicem
;       poprograma FIBONACCI. To ponavljamo tolikokrat, kot smo
;       klicali ta podprogram. Sele zanji RSB vrne kontrolo v
;       proceuro IFIB.
        rsb
        .end
```

```
      program test
C Testni program za macro podprogram za racunanje Fibonaccijevega
C stevila.
      type 10
10    format(1h$,'Stevilka Fibonaccijevega stevila: ')
      accept *,n
      type 20,n,ifib(n)
20    format(1h0,'Fibonaccijevo stevilo F',i2,' je ',i10)
      end
```

```
      .title  Racunanje vrednosti polinoma.
vprasanje:    .ascii  /Vnesi stopnjo polinoma: /
dolzinavpr:   .long   .-vprasanje
stopnja:      .blkl   1
vnoskoef:     .long   2                 ; Lista parametrov za klic
              .long   dolzkoef          ; procedure OUTPUT.
              .long   vprasanjekoef
vprasanjekoef:.ascii  /Vnesi koeficient polinoma: /
dolzkoef:     .long   dolzkoef-vprasanjekoef
koeficienti:  .blkl   11                ; Prostor za tabelo koeficientov.
                                        ; Najvecja stopnja polinoma je 10
                                        ; (11 koeficientov).
vprasanjearg: .ascii  /Vnesi vrednost argumenta: /
dolzarg:      .long   .-vprasanjearg
argument:     .blkl   1                 ; Prostor za argument
rezultat:     .blkl   1                 ; in za rezultat.
odgovor:      .ascii  /Rezultat je: /
dolzodg:      .long   .-odgovor
odg:          .long   2                 ; Lista parametrov za proceduro
              .long   dolzodg           ; OUTPUT.
              .long   odgovor
rez:          .long   1                 ; Lista parametrov za proceduro
              .long   rezultat          ;

      .entry  polinom ^m()
      pushal  vprasanje                 ; Pripravimo parametre za
      pushal  dolzinavpr                ; klic procedure OUTPUT.
      calls   #2,output
      pushal  stopnja                   ; Parameter za ININT (branje
      calls   #1,inint                  ; celega stevila (integer).
      addl3   #1,stopnja,r2             ; R2 = stevilo koeficientov.
      moval   koeficienti,r3            ; R3 = naslov prvega koeficienta.
```

```
;       Vnos koeficientov:
;
zanka:  calls   vnoskoef,output         ; Izpisemo vprasanje.
        pushal  (r3)+                   ; Na skladu pripravimo naslov,
                                        ; na katerega bomo zapisali
                                        ; prebrano realno stevilo (R3
                                        ; kaze na zacetek tabele).
        calls   #1,infloat              ; Preberemo stevilo.
        sobgtr  r2,zanka                ; To ponavljamo dokler ne
                                        ; zmanjsamo R2 na 0.
naslednji:
        pushal  vprasanjearg
        pushal  dolzarg
        calls   #2,output               ; Vprasa za argument
        pushal  argument
        calls   #1,infloat              ; in ga prebere.
;       Z ukazom POLYF izracunamo vrednost polinoma. F pomeni, da
;       so koeficienti in argument realna stevila (floating point).
        polyf   argument,stopnja,koeficienti
        movl    r0,rezultat             ; Rezultat ukaza POLYx je v
                                        ; registru R0.
        calls   odg,output              ; Izpisemo rezultat. Listi
        calls   rez,outfloat            ; parametrov sta ze pripravljeni.
        brw     naslednji

        $exit_s #1
        .end    polinom
```

```
        .title  vrsta
;       Program ilustrira vnasanje podatkov v vrsto in brisanje
;       iz vrste.
glava:  .address        glava           ; Glava vrste z kazalcema
        .address        glava           ; na prvi in zadnji podatek
                                        ; v vrsti.

a:      .blkl   2                       ; Prostor za 2 kazalca
                                        ; (naprej in nazaj) in
        .ascic  /To je podatek A./      ; vrednost tega podatka.

b:      .blkl   2
        .ascic  /To je podatek B./

c:      .blkl   2
        .ascic  /Konec vrste.    /

tekst:  .ascii  /Konec/

        .entry  vrsta   ^m< >
        insque  a,glava                 ; A vstavimo na prvo mesto.
        insque  b,glava                 ; B vstavimo pred A.
        insque  c,a                     ; C vstavimo za A.

        movl    glava,r6                ; Naslov prvega elementa = R6.
zanka:  movzbl  8(r6),r1                ; Prvi byte podatka je stevilo
                                        ; znakov v tekstu (ASCIC pomeni
                                        ; presteti - counted - niz).
        matchc  #5,tekst,r1,9(r6)       ; Iscemo tekst v kateremkoli
                                        ; od podatkov v vrsti.
        beql    konec                   ; Ko ga najdemo, koncamo.
        movl    (r6),r6                 ; S tem ukazom preidemo na
                                        ; naslednji podatek v vrsti
        cmpl    r6,#glava               ; in kontroliramo, ce je to
                                        ; glava vrste.
        bneq    zanka

        movl    #2,r0                   ; Tu se program konca, ce
        ret                             ; nismo nasli teksta. Vrnemo
                                        ; status 2 - napaka.

konec:  movl    #1,r0                   ; Ce smo nasli tekst, vrnemo
        ret                             ; status uspesno.
        .end    vrsta
```

```
      subroutine input(stev,tekst)
C
C     Podprogram INPUT prebere do 100 znakov s terminala (LUN 5).
C     Kot prvi parameter vrne stevilo prebranih znakov, drugi
C     parameter pa je naslov, kjer se zacne tekst.
C
      integer stev
      byte tekst(100)
      read (5,10) stev,tekst
      return
10    format (q,100a1)
      end


      subroutine output(stev,tekst)
C
C     Podprogram OUTPUT izpise do 100 znakov na terminal (LUN 5).
C     Prvi parameter je stevilo znakov, ki jih zelimo izpisati, drugi
C     parameter pa je naslov, kjer se zacne tekst.
C
      integer stev
      byte tekst(100)
      type 10, (tekst(i), i=1,stev)
      return
10    format (1h$,100a1)
      end


      subroutine inint(i)
C
C     Podprogram ININT prebere s terminala (LUN 5)
C     celo stevilo.
C
      accept *,i
      return
      end


      subroutine outint(i)
C
C     Podprogram OUTINT napise na terminal (LUN 5)
C     celo stevilo.
C
      type *,i
      return
      end
```

```
      subroutine infloat(f)
C
C     Podprogram INFLOAT prebere s terminala
C     realno stevilo.
C
      accept *,f
      return
      end


      subroutine outfloat(f)
C
C     Podprogram OUTFLOAT napise na terminal
C     realno stevilo.
C
      type *,f
      return
      end
```

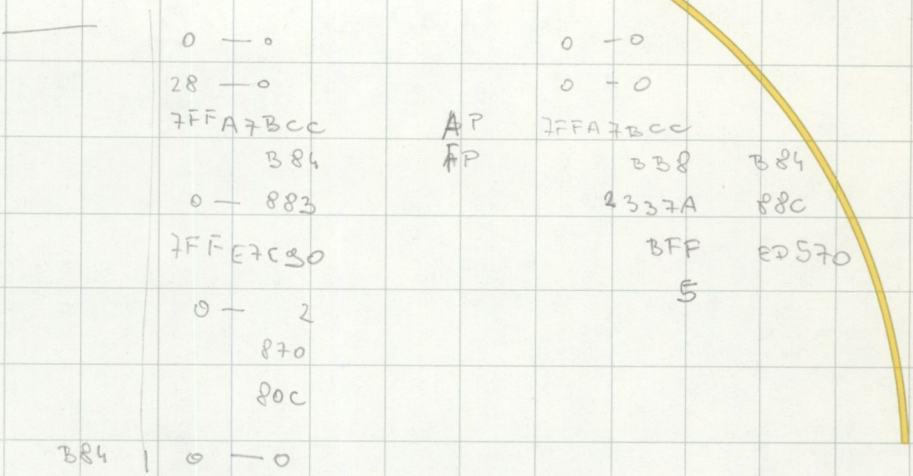
IskraDelta

PC -602

SP -7FFA7B60

AP B78

FP B60




```
.MACRO ERROR STATUS,ERR
BLBC STATUS,ERR
.ENDM

CALLS #0,ABC
ERROR R0,GRESKA  ⇒ BLBC R0,GRESKA
```

```
.MACRO ARGUMENTI N,LISTA
.LONG N
.IRP X,LISTA
.ADDRESS X
.ENDR
.ENDM ARGUMENTI

LISTA: ARGUMENTI 3,<A,B,C>   ⇒   LISTA: .LONG 3
                                        .ADDRESS A
                                        .      4   B
                                        .      4   C
LISTB: ARGUMENTI 7,<-----> 
```

```
.MACRO TEST_ERR STATUS=R0,?NASLOV
BLBS STATUS,NASLOV
MOVL STATUS,R0
RET
NASLOV: - local label 30000$:⇒
.ENDM TEST
```

RTL

- primamo bot funkcijo, status v R7
- CALLG A, G^LIB$PUT_SCREEN
  ↑ general addressing mode

DATOTEKE

1.) Redefiniramo SYS$INPUT, SYS$OUTPUT
2.) Datoteke v lastnem programskem jeziku
3.) RMS FAB - $FAB - block $READ, $WRITE
RAB - $RAB - record $GET, $PUT, $CONNECT

SYSTEM SERVICES

$IMEG_S macro
SYS$IME - bot object

TEST MACRO II

Ime i prezime:

Radna organizacija:

1. Kako se ekspandira sledeči macro:

```
        .MACRO  ERROR   STATUS=R0,?ADR
        BLBS    STATUS,ADR
        PUSHL   STATUS
        CALLS   #1,LIB$SIGNAL
ADR:    .ENDM   ERROR
```

MOVL STATUS, R0

a. ERROR

b. ERROR NAPAKA

c. ERROR TEST,GRESKA

2. Ispravi grešku u macrou ERROR (vidi 1. zadatak)!

3. Napiši macro, koji unese u memoriju sve vrednosti navedene u argumentu LISTA.

Primer:

```
ARGUMENTI <A,B,C>
```

će se ekspandirat kao:

```
.LONG   A
.LONG   B
.LONG   C
```

4. Dali se tekst ispiše?

```
. = 1000
A:      .LONG   100

        .MACRO  XX ARGUMENT
        .IF LESS A-500
        .PRINT  ; Argument je manji od 500.
        .ENDC
```


5. Napiši macro, koji pomoču direktiva .REPEAT i .BYTE

6. Postavi u definiciju macroa direktive .SAVE_PSECT, .PSECT ARGUMENTI i .RESTORE_PSECT!

```
        .MACRO  CALL    PROCEDURA,LISTA
    N=0
        .IRP    X,LISTA
    N=N+1
        .ENDR
    ARG=.
        .LONG   N
        .IRP    X,LISTA
        .LONG   X
        .ENDR
        .CALLG  ARG,PROCEDURA
        .ENDM   CALL
```


DODATNI ZADATAK

7. Kakav je sadržaj registra R5 na kraju tog dela programa?

```
    N:      .LONG   10
            .
```

```
        CLRL    R5
ADD:    ADDL2   N,R5
        DECL    N
        .IIF    GREATER N   BRB     ADD
```

```
        .macro moj a1,a2,a3
        .narg   count
        .psect  data,wrt,noexe
nasl = .
        .long   count
        .psect  code,nowrt,exe
        .irp    arg,<a1,a2,a3>
        .iif    not_blank,arg,  pushl   arg
        .endr
        pushl   nasl
        .endm   moj
        .psect  data,wrt,noexe
a:      .long   10
b:      .long   20
c:      .long   30

        .psect  code,nowrt,exe
        .entry  m2      ^m<>

        moj     a,b,c

        moj     a,b

        moj     a
        movl    #1,r0
        ret
        .end m2
```

```
      PROGRAM TEST
C     Testni program za podprogram, ki mnozi matrike.
C     Matrike lahko imajo do 9 vrstic in 9 stolpcev.

      DIMENSION A(9,9),B(9,9),C(9,9)

C     Prebere stevilo vrstic in stolpcev za obe vhodni
C     matriki, njune dimenzije pa definira v podprogramu
C     MATR.
      TYPE 10
10    FORMAT ('$Stevilo vrstic in stevilo stolpcev prve  matrike: ')
      ACCEPT *,I,J
      TYPE 20
20    FORMAT ('$Stevilo vrstic in stevilo stolpcev druge matrike: ')
      ACCEPT  *,K,L
      CALL MATR(I,J,K,L,A,B,C)
      END

      SUBROUTINE MATR(I,J,K,L,A,B,C)
C     Podprogram sluzi za pravilno definicijo dimenzij
C     matrik. Ce bi veljala definicia iz DIMENSION stavka
C     glavnega programa, bi imeli najprej prvih J podatkov,
C     nato pa 9 - J nicel in sele za temi naslednje podatke.
      DIMENSION A(J,I),B(L,K),C(L,I)
C     FORTRAN spravlja podatke v polja tako, da se najhitreje
C     spreminja prvi indeks. Drugi programski jeziki spravijo
C     podatke v polja po vrsticah (najhitreje se spreminja
C     zadnji indeks) in ker se bomo drzali tega dogovora,
C     moramo v FORTRANu delati z zamenjanimi indeksi.
      DO 100 II=1,I
      DO 100 JJ=1,J
      TYPE 10, II,JJ
100   ACCEPT *,A(JJ,II)
      DO 200 II=1,K
      DO 200 JJ=1,L
      TYPE 20, II,JJ
200   ACCEPT *,B(JJ,II)
10    FORMAT ('$Vnesi A(',i1,',',i1,'): ')
20    FORMAT ('$Vnesi B(',i1,',',i1,'): ')
      CALL MNOZMAT(I,J,A,K,L,B,I,L,C)
      DO 300 II=1,I
      DO 300 JJ=1,L
300   TYPE 30, II,JJ,C(JJ,II)
30    FORMAT (' C(',i1,',',i1,') = ',f12.4)
      RETURN
      END
```

```
        .title  mnozenje matrik
;       Podrogram z devetimi argumenti za mnozenje matrik.
;       Argumenti so po vrsti:
;               stevilo vrstic prve matrike      24(ap)
;               stevilo stolpcev prve matrike    28(ap)
;               naslov prve matrike              12(ap)
;               stevilo vrstic druge matrike     216(ap)
;               stevilo stolpcev druge matrike   220(ap)
;               naslov druge matrike             24(ap)
;               stevilo vrstic rezultata         228(ap)
;               stevilo stolpcev rezultata       232(ap)
;               naslov rezultata                 36(ap)

        .entry  mnozmat ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
        jsb     preveri_podatke
        movl    24(ap),r2               ; Stevilo vrstic prve matrike.
        moval   236(ap),r11             ; Naslov tretje matrike - rezultat.
        clrl    r5                      ; Trenutna vrstica - zacne se z 0.
10$:    movl    220(ap),r3              ; Stevilo stolpcev druge matrike.
        clrl    r6                      ; Trenutni stolpec.
20$:    jsb     mnozi
        movl    r8,(r11)+               ; Zapisemo naslednji element rezultata
        incl    r6                      ; Naslednji stolpec.
        sobgtr  r3,20$                  ; Dokler ne porabimo vseh stolpcev.
        incl    r5                      ; Naslednja vrstica.
        sobgtr  r2,10$                  ; Dokler ne porabimo vseh vrstic.

        movl    #1,r0                   ; Uspesno koncan podprogram za
        ret                             ; mnozenje matrik.

;       Mnozenje izbrane vrstice prve matrike z izbranim stolpcem druge.
;       R5 je stevilka vrstice, R6 je stevilka stolpca, v R8 je rezultat.
mnozi:
        movl    28(ap),r4               ; Stevilo komponent enega mnozenja.
        moval   212(ap),r9              ; Naslov prve matrike.
        moval   224(ap),r10             ; Naslov druge matrike.
        mull3   28(ap),r5,r7            ; V R7 pise, koliko elementov matrike
        ashl    #2,r7,r7                ; je treba preskociti, da pridemo na
        addl2   r7,r9                   ; zacetek prave vrste.
        ashl    #2,r6,r7                ; Isto za drugo matriko za stolpec.
        addl2   r7,r10
        clrl    r8                      ; Vmesni rezultat pri mnozenju vrstice
                                        ; in stolpca.
10$:    mulf3   (r9)+,(r10),r7          ; Zmnozimo istolezne elemente
        addf2   r7,r8                   ; in jih sestevamo v r8.
        ashl    #2,220(ap),r7           ; Gremo v naslednjo vrstico druge
        addl2   r7,r10                  ; matrike.
        sobgtr  r4,10$                  ; Dokler ne zmnozimo cele vrstice s
                                        ; stolpcem.
        rsb                             ; Konec podprograma MNOZI.
                                        ; Rezultat je v R8.

;       Preverjanje podatkov.
;       Ali je pravo stevilo argumentov in ali se stevilo vrstic in
;       stolpcev ujema kot je zahtevano.
preveri_podatke:
        cmpl    (ap),#9
        bneq    napaka
        cmpl    24(ap),228(ap)          ; A(vrs) = C(vrs) ?
        bneq    napaka
        cmpl    28(ap),216(ap)          ; A(sto) = B(vrs) ?
        bneq    napaka
        cmpl    220(ap),232(ap)         ; B(sto) = C(sto) ?
```

```
        bneq    napaka
        rsb
napaka: movl    #2,r0
        ret

        .end
```

```
        .TITLE  SINHRONIZACIJA
;       PROGRAM NAREDI "COMMON EVENT FLAG CLUSTER"
;       IN POSTAVI EFN 64. CE JE TA FLAG ZE POSTAVLJEN,
;       GA JE POSTAVIL ISTI PROGRAM, KI ZDAJ CAKA NA
;       EFN 65. CE EFN 64 NI BIL POSTAVLJEN, JE TA
;       PROGRAM PRVI IN MORA CAKATI NA EFN 65.

        .PSECT  PODATKI NOEXE
IME:    .ASCID  /CAKAM_SE/
SPOROC:         .ASCID  /KONEC CAKANJA NA EFN 65!/<7>
CAKAM:          .ASCID  /CAKAM EFN 65!/
POSTAVLJAM:     .ASCID  /POSTAVLJAM EFN 65!/<7>

        .PSECT  KODA    NOWRT
        .ENTRY  C2      ČM<>
        $ASCEFC_S       EFN     = #64   -       ; NAREDI SKUPINO EVENT FLAGOV.
                        NAME    = IME
        PREVERI_STATUS                          ; CE JE PRISLO DO NAPAKE,
                                                ; PRESKOCI OSTALE UKAZE IN
                                                ; KONCA S STATUSOM TE NAPAKE.

        $SETEF_S        EFN     = #64           ; POSTAVI EFN 64 IN PREVERI,
        CMPL    #SS$_WASSET,R0                  ; CE JE BIL SETIRAN ZE PREJ.
        BEQL    POSTAVI
        PREVERI_STATUS                          ; PREVERI STATUS IN

        PUSHAQ  CAKAM                           ; SPOROCI, DA CAKA EFN 65.
        CALLS   #1,GČLIB$PUT_OUTPUT
        PREVERI_STATUS
        $WAITFR_S       EFN     = #65           ; CAKA EVENT FLAG 65.
        PREVERI_STATUS
        PUSHAQ  SPOROC                          ; IZPISE SPOROCILO O KUNCU
        CALLS   #1,GČLIB$PUT_OUTPUT             ; CAKANJA.
        PREVERI_STATUS
        BRB     KONEC

POSTAVI:
        PUSHAQ  POSTAVLJAM                      ; IZPISE SPOROCILO O
        CALLS   #1,GČLIB$PUT_OUTPUT             ; POSTAVLJANJU EFN 65.
        PREVERI_STATUS
        $SETEF_S        EFN     = #65           ; POSTAVI EFN 65 IN S TEM
                                                ; OMOGOCI PRVEMU, DA KONCA.
        PREVERI_STATUS

KONEC:  $EXIT_S #SS$_NORMAL
        .END    C2
```

```
        .TITLE PREBUJANJE
;       PROGRAM ZASPI IN SE NATO VSAKIH 5 SEKUND
;       PREBUDI, DA IZPISE TRENUTNI CAS. PO 20 BUJENJIH
;       SE NEHA IZVAJATI.

        .PSECT  PODATKI NOEXE, LONG
BIN_INTERVAL:   .BLKL   2               ; BINARNO ZAPISAN CAS BUJENJA.
BIN_CAS:        .BLKL   2               ; PROSTOR ZA SISTEMSKI CAS -BINARNO.
STEVEC:         .LONG   20              ; STEVEC PONAVLJANJ.
ASCII_INTERVAL: .ASCID  /0 00:00:05/    ; INTERVAL 5 SEKUND.
ASCII_CAS:      .ASCID  /                       /       ; 23 BYTOV ZA CAS.

        .PSECT  KODA    NOWRT
        .ENTRY  C1      ČM<>
        $GETTIM_S       TIMADR  = BIN_CAS       ; DOBI SISTEMSKI CAS ZA PRVO
                                                ; BUJENJE.
        PREVERI_STATUS
        $BINTIM_S       TIMBUF  = ASCII_INTERVAL -      ; PRETVORI INTERVAL
                        TIMADR  = BIN_INTERVAL          ; BUJENJA V BINARNO.
        PREVERI_STATUS
        $SCHDWK_S       DAYTIM  = BIN_CAS -     ; ODREDI BUJENJE OB DOLOCENEM
                        REPTIM  = BIN_INTERVAL  ; CASU IN NATO VSAKIH 5 SEK.
        PREVERI_STATUS

HIBER:  $HIBER_S                                ; ZASPI.
        $GETTIM_S       TIMADR  = BIN_CAS
        PREVERI_STATUS
        MOVW    #23,ASCII_CAS                   ; INICIALIZIRA BUFFER ZA CAS
                                                ; NA POLNO DOLZINO 23 BYTOV.
        $ASCTIM_S       TIMLEN  = ASCII_CAS -   ; PREVEDE BINARNI CAS V
                        TIMBUF  = ASCII_CAS -   ; ASCII ZNAKE.
                        TIMADR  = BIN_CAS
        PUSHAQ  ASCII_CAS               ; PRIPRAVI TEKST ZA IZPIS.
        CALLS   #1,GČLIB$PUT_OUTPUT     ; IZPISE CAS.
        DECL    STEVEC
        BEQL    10$
        BRW     HIBER
10$:    MOVL    #SS$_NORMAL,R0
        RET
        .END    C1
```

```
        .TITLE  SPREMENI MALE ZNAKE V VELIKE
        .PSECT  ZNAKI   PIC,OVR,REL,GBL,SHR,NOEXE,RD,WRT,LONG
TEKST:  .BLKB   100                 ; PRAZEN PROSTOR, KI PREKRIVA
                                    ; COMMON PODROCJE ZNAKI.

        .ENTRY  SPREM   ČM<R2,R3>
        MOVL    ž4(AP),R2           ; STEVILO ZNAKOV, KI JIH PREGLEDA.
        DECL    R2                  ; INDEKS NA ZADNJI ZNAK.
        MOVAL   TEKST,R3            ; NASLOV ZACETKA TEKSTA.
ZANKA:  CMPB    #ČA/A/,(R3)šR2ć     ; ALI JE ZNAK PRED "A"?
        BGTR    NASLEDNJI
        CMPB    #ČA/Z/,(R3)šR2ć     ; ALI JE ZNAK ZA "Z"?
        BLSS    NASLEDNJI           ; ZNAK JE MALA CRKA - ZAMENJAJ V VELIKO
        BICB    #1ž5,(R3)šR2ć       ; TAKO, DA BRISES BIT STEVILKA 5.
NASLEDNJI:
        SOBGEQ  R2,ZANKA            ; OD ZADNJEGA ZNAKA DO PRVEGA.
        RET
        .END
```

```
;       MACRO CALL KLICE PODPROGRAM S CALLG UKAZOM.
;       ARGUMENTE MU PRIPRAVI MACRO ARGUMENTI V NOVI
;       PROGRAMSKI SEKCIJI LISTA.

        .MACRO  CALL    PODPROGRAM,ARG
        .SAVE_PSECT                         ; SHRANI PROGRAMSKO SEKCIJO.
        .PSECT  LISTA   NOEXE,LONG          ; DEFINIRA NOVO SEKCIJO ZA ARGUMENTE.
LISTA = .                                   ; CE VECKRAT UPORABLJAMO MACRO CALL,
                                            ; NE MOREMO DEFINIRATI LABELE Z LISTA:.
        ARGUMENTI       <ARG>               ; NAREDI LISTO ARGUMENTOV.
        .RESTORE_PSECT                      ; VRNEMO PRVOTNO PROGRAMSKO SEKCIJO.

        CALLG   LISTA,PODPROGRAM            ; KLIC PODPROGRAMA.
        .ENDM   CALL
```

```
;       MACRO CALL KLICE PODPROGRAM S CALLG UKAZOM.
;       ARGUMENTE MU PRIPRAVI MACRO ARGUMENTI V NOVI
;       PROGRAMSKI SEKCIJI LISTA.

        .MACRO  CALL    PODPROGRAM,ARG
        .SAVE_PSECT                             ; SHRANI PROGRAMSKO SEKCIJO.
        .PSECT  LISTA   NOEXE,LONG              ; DEFINIRA NOVO SEKCIJO ZA ARGUMENTE.
LISTA = .                                       ; CE VECKRAT UPORABLJAMO MACRO CALL,
                                                ; NE MOREMO DEFINIRATI LABELE Z LISTA:.
        ARGUMENTI       <ARG>                   ; NAREDI LISTO ARGUMENTOV.
        .RESTORE_PSECT                          ; VRNEMO PRVOTNO PROGRAMSKO SEKCIJO.

        CALLG   LISTA,PODPROGRAM                ; KLIC PODPROGRAMA.
        .ENDM   CALL
```

```
        BEQL    KONEC                   ; SAMI PRESLEDKI - KONEC.
        MOVL    R1,TEKST+4              ; PRESTAVI KAZALEC TEKSTA
        MOVW    R0,TEKST                ; IN DOLZINO NIZA.
        BEQL    KONEC                   ; NI VEC ZNAKOV - KONEC.
        BRW     ISCI

KONEC:  MOVL    #1,R0
        RET
        .END    S1
```

```
        .TITLE  RAZBIJANJE TEKSTA NA BESEDE

;       MACRO ARGUMENTI PRIPRAVI LISTO ARGUMENTOV ZA KLIC
;       S CALLG (VAJA T1).
        .MACRO  ARGUMENTI       ARG
...STEV = 0
        .IRP    A,ARG
...STEV = ...STEV + 1                   ; PRESTEJE ARGUMENTE.
        .ENDR

        .LONG   ...STEV                 ; PRVI PODATEK JE STEVILO ARGUMENTOV.
        .IRP    A,ARG
        .LONG   A                       ; VPISUJE POSAMEZNE ARGUMENTE.
        .ENDR
        .ENDM   ARGUMENTI

;       MACRO CALL KLICE PODPROGRAM S CALLG UKAZOM.
;       ARGUMENTE MU PRIPRAVI MACRO ARGUMENTI V NOVI
;       PROGRAMSKI SEKCIJI LISTA (VAJA T2).
        .MACRO  CALL    PODPROGRAM,ARG
        .SAVE_PSECT                             ; SHRANI PROGRAMSKO SEKCIJO.
        .PSECT  LISTA   NOEXE,LONG              ; DEFINIRA NOVO SEKCIJO ZA ARGUMENTE.
LISTA = .                                       ; CE VECKRAT UPORABLJAMO MACRO CALL,
                                                ; NE MOREMO DEFINIRATI LABELE Z LISTA:.
        ARGUMENTI       <ARG>                   ; NAREDI LISTO ARGUMENTOV.
        .RESTORE_PSECT                          ; VRNEMO PRVOTNO PROGRAMSKO SEKCIJO.
        CALLG   LISTA,PODPROGRAM                ; KLIC PODPROGRAMA.
        .ENDM   CALL

;       MACRO DESCR DEFINIRA PRAZEN OPISNIK TEKSTA Z IZBRANO
;       DOLZINO IN PROSTOROM ZA TEKST.
        .MACRO  DESCR   DOLZINA
        .WORD   DOLZINA                 ; DOLZINA TEKSTA.
        .BYTE   DSC$K_DTYPE_T           ; TIP PODATKOV - CHARACTER CODED TEXT.
        .BYTE   DSC$K_CLASS_S           ; RAZRED PODATKOV - TEKST FIKSNE DOLZINE.
        .ADDRESS .+4                    ; NASLOV KAZE NASLEDNJI BYTE.
        .BLKB   DOLZINA                 ; PRAVA KOLICINA PRAZNIH BYTOV.
        .ENDM   DESCR

        .PSECT  PODATKI NOEXE, LONG
TEKST:  DESCR   80
PROMPT: .ASCID  /VNESI TEKST: /

        .PSECT  KODA    NOWRT
        .ENTRY  S1      ČM<>
        CALL    GČLIB$GET_COMMAND,<TEKST,PROMPT,TEKST>
        BLBS    R0,ISCI                         ; PREVERI STATUS
        PUSHR   #1
        CALLS   #1,GČLIB$SIGNAL                 ; IN JAVI NAPAKO.

ISCI:   LOCC    #ČA/ /,TEKST,ŽTEKST+4           ; POISCI NASLEDNJI PRESLEDEK.
        BNEQ    10$                             ; NI PRESLEDKA - KONEC.
        CLRL    R1                              ; ZAZNAMUJ TO V R1.
10$:    SUBW2   R0,TEKST                        ; ZMANJSAJ DOLZINO TEKSTA.
        PUSHR   #3                              ; SPRAVI REGISTRA R0 IN R1.
        CALL    GČLIB$PUT_OUTPUT,<TEKST>
        BLBS    R0,20$                          ; CE JE STATUS SOD,
        CALLS   #1,GČLIB$SIGNAL                 ; JAVI NAPAKO. NA VRHU SKLADA JE R0.
20$:    POPR    #3                              ; VRNI VREDNOSTI V R0 IN R1.
        MOVL    R1,TEKST+4                      ; PRESTAVI KAZALEC ZACETKA TEKSTA
        BEQL    KONEC                           ; (V R1 JE 0, CE SMO KONCALI)
        MOVW    R0,TEKST                        ; IN VSTAVI NOVO DOLZINO.
        BEQL    KONEC                           ; NI VEC ZNAKOV - KONEC.

        SKPC    #ČA/ /,TEKST,ŽTEKST+4
```

```
        .TITLE  PREVAJANJE ASCII ZNAKOV V EBCDIC
;       PROGRAM PREBERE ZAPIS Z ENOTE SYS$INPUT, PREVEDE
;       ASCII ZNAKE V EBCDIC KODO IN ZAPISE PREDELANI
;       ZAPIS NA ENOTO SYS$OUTPUT.

;       MACROJI ZA PREVERJANJE STATUSA IN ZA KLIC PODPROGRAMA
;       S CALL SO TUDI V KNJIZNICI MACRO.MLB NA DIREKTORIJU
;       DRA1:ŠTECAJ.KNJIZNICAĆ
        .LIBRARY        /DRA1:ŠTECAJ.KNJIZNICAĆMACRO/

        .PSECT  PODATKI NOEXE, LONG
DESCR:  .ASCID  //
BUFFER: .BLKB   132
IZPIS:  .ASCID  //
        .BLKB   132
PROMPT: .ASCID  //

        .PSECT  KODA    NOWRT
        .ENTRY  S2      ČM<>
BERI:   MOVW    #132,DESCR                      ; INICIALIZACIJA OPISNIKA ZA VNOS.
        CALL    GČLIB$GET_INPUT,<DESCR,,DESCR>
        PREVERI_STATUS  R0,KONEC                ; TESTIRA STATUS RTL PROCEDURE.
        MOVW    DESCR,IZPIS                     ; NASTAVI PRAVO DOLZINO IZHODNEGA
                                                ; TEKSTA.
        CALL    GČLIB$TRA_ASC_EBC,<DESCR,IZPIS> ; PREVAJANJE.
        PREVERI_STATUS  R0
        CALL    GČLIB$PUT_OUTPUT,<IZPIS>
        PREVERI_STATUS  R0
        BRW     BERI
KONEC:  MOVL    #1,R0
        RET
        .END    S2
```

VAJE ZA TECAJ V310 MACRO II

P1: NAPISI PODPROGRAMA BRANJE IN PISANJE V VISJEM PROGRAMSKEM JEZIKU, DA SI S TEM OSKRBIS VHODNO IZHODNE OPERACIJE ZA MACRO PROGRAME. PODPROGRAMA IMATA DVA PARAMETRA, PRVI JE STEVILO ZNAKOV V NIZU, DRUGI PA ZACETEK NIZA. OBA PARAMETRA PRENASAJ PO NASLOVU (BY REFERENCE). TESTIRAJ OBA PODPROGRAMA Z MACRO PROGRAMOM.

P2: NAPISI V MACROJU PODPROGRAM ZA MNOZENJE MATRIK. PODPROGRAM IMA 9 PARAMETROV, ZA VSAKO MATRIKO PO 3: STEVILO VRSTIC, STEVILO STOLPCEV IN ZACETEK POLJA S PODATKI. MATRIKI A Z ELEMENTI A(I,J) IN B Z ELEMENTI B(K,L) POMNOZIMO TAKO, DA VELJA ZA REZULTAT C Z ELEMENTI C(M,N):

C(M,N) = A(M,1)*B(1,N) + A(M,2)*B(2,N) + ... + A(M,A)*B(A,N).

A JE STEVILO STOLPCEV MATRIKE A, KI MORA BITI ENAKO STEVILU VRSTIC MATRIKE B, DA JE MNOZENJE MOZNO. NAJ PODPROGRAM TO PREVERI.

T1: NAPISI MACRO, KI NAREDI SEZNAM ARGUMENTOV ZA KLIC PODPROGRAMA S CALLG UKAZOM. MACRO NAJ BO DEFINIRAN TAKO, DA GA KLICEMO Z:

ARGUMENTI <ARGUMENT_1,ARGUMENT_2,...ARGUMENT_N>.

ARGUMENTI JE IME MACROJA, STEVILO ARGUMENTOV PA NI VNAPREJ DOLOCENO.

T2: UPORABI MACRO ARGUMENTI V DEFINICIJI MACROJA, S KATERIM POKLICES PODPROGRAM NA NACIN:

CALL IME_PODPROGRAMA,<ARGUMENT_1,ARGUMENT_2,...ARGUMENT_N>.

T3: NAPISI MACRO PODPROGRAM, KI V FORTRANSKEM PROGRAMU TEST1.FOR SPREMENI VSEBINO COMMON PODROCJA TAKO, DA VSE MALE CRKE SPREMENI V VELIKE, OSTALE ZNAKE PA PUSTI NESPREMENJENE. UGOTOVI, KAKSNE ATRIBUTE MORA IMETI PROGRAMSKA SEKCIJA, S KATERO DOSTOPAS DO COMMON PODROCJA.

S1: NAPISI PROGRAM, KI PREBERE NEK TEKST, GA RAZBIJE NA BESEDE IN JIH IZPISE VSAKO V SVOJI VRSTI. ZA VNOS IN IZPIS PODATKOV UPORABI PODPROGRAME IZ RUN TIME LIBRARY (RTL). PODPROGRAME KLICI Z MACROJEM CALL IZ NALOGE T2.

S2: NAPISI PROGRAM, KI BO S PRVE DATOTEKE BRAL ZNAKE ZAPISANE V ASCII KODI IN JIH ZAPISAL NA DRUGO DATOTEKO V EBCDIC KODI. UPORABI RTL. ZA DOSTOP DO DATOTEK LAHKO UPORABIS RMS MACROJE, RTL ALI PA DEFINIRAS LOGICNI IMENI SYS$INPUT IN SYS$OUTPUT KOT DATOTEKI.

C1: NAPISI PROGRAM, KI ZASPI (HIBERNATE) IN SE NATO VSAKIH 5 SEKUND ZBUDI, IZPISE TRENUTNI CAS IN SPET ZASPI. ZA IZPIS UPORABI RTL.

C2: NAPISI PROGRAM, KI CAKA, DA NEKDO POZENE ISTI PROGRAM IN NATO
IZPISE , DA JE KONEC CAKANJA IN SE KONCA. ZA IZPIS UPORABI $QIOW
SISTEMSKO ZAHTEVO. PROGRAM SINHRONIZIRAS S COMMON EVENT FLAGS.