Skupovi instrukcija za ARM

Vladimir Filipović
vladaf@matf.bg.ac.yu

Registri, memorija i prenos podataka kod ARM

Advanced RISC Machines (ARM) Limited je dizajnirao liniju RISC mikroprocesora i dao licencu drugim proizvođačima čipova, tako da se ovi čipovi intenzivno koriste u računarima i ugrađenim sistemima.

ARM je nastao iz Acorn Computers, koji je osnovan ranih 80-tih.

Osnovno polje korišćenja ARM procesora su jeftiniji ugrađeni sistemi, kao što su mobini telefoni, komunikacioni modemi, sistemi za upravljanje radom motora vozila, PDA itd.

Svi procesori ARM familije imaju veoma sličan skup instrukcija.

Ovde ćemo konkretno govoriti o instrukcijama ARM7 procesora.

Memorija je bajt adresibilna, koriste se 32-bitne adrese, i dužina registara procesora je 32 bita.

Operandi pri prenosu podataka u i iz memorije mogu biti dužine 8 ili 32.

Registri, memorija i prenos podataka kod ARM

Adrese reči moraju biti poravnate, tj. Moraju biti umnožci broja 4.

Podržano je i little-endian i big-endian memorijsko adresiranje. Izbor adresiranja određuje spoljni signal sa ulazne linije procesora.

Pri prenosu bajta iz memorije u registar, puni se samo najniži bajt registra, a u ostatku registra se smeštaju nule.

Memoriji se pristupa isključivo preko Load i Store instrukcija. Sve aritmetičke i logičke instrukcije operišu isključivo nad podacima koji se nalaze u registrima.

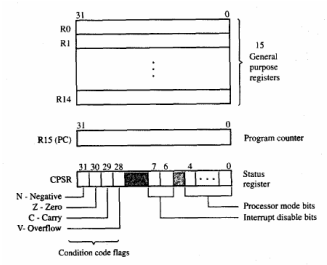
Procesor sadrži 16 32-bitnih registara označenih slovima R0 do R15, od kojih je 15 registara opšteg tipa, a šesnaesti, tj. registar R15 predstavlja brojač naredbi PC.

Registri opšteg tipa mogu da čuvaju podatke ili memorijsku adresu.

Registar CPSR predstavlja status registar, koji sadrži uslovne flegove (N,Z,C,V), Fleg onemogućivanja prekida i bitove moda procesora.

Registri, memorija i prenos podataka kod ARM

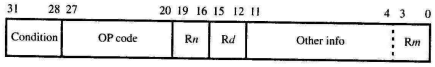
U ovom razmatranju pretpostavljamo da procesor radi u korisničkom modu i da izvršava aplikativni program.



Registri, memorija i prenos podataka kod ARM

Postoji još 15 dodatnih registara opšteg tipa, koji se nazivaju bankirani registri. Ti registri su duplikati nekog od registara R0-R14. Oni se koriste kada procesor prelazi u nadzorni ili prekidni mod izvršavanja. Dakle, u modu rada procesora koji nije korisnički, na raspolaganju su i kopije statusne reči.

Format instrukcija ARM-a je sledeći:



Instrukcija specificira uslovni kod izvršavanja (Condition), operacioni kod (OP code), dva ili tri registra (Rn,Rd i Rm), kao i neke druge informacije (Other info). Ako registar Rm nije neophodan, polje ostali podaci se ispunjava nulama do kraja.

Kod Load instrukcije, operand se prenosi iz memorije u registar označen sa četvorobitnim poljem Rd.

Instrukcije za pristup memoriji i adresni modovi kod ARM

Karakteristika koja ARM razlikuje os većine drugih procesora je da se sve instrukcije uslovno izvršavaju, zavisno od uslova koji je specificiran instrukcijom. Instrukcija se izvršava samo ukoliko tekuće stanje flegova uslovnog koda statusne reči procesora zadovoljava uslov koji je propisan bitovima b_{31-28} instrukcije koja se izvršava. Zasad pretpostavljamo da bitovi sadrže uslov b_{31-28} "uvek se izvrši".
Osnovni metod adresiranja memorijskih operanada je generisanje efektivne adrese EA dodavanjem označenog ofseta na sadržaj baznog registra Rn. Apsolutna vrednost ofseta može biti neposredna vrednost (koja je sadržana u nižih 12 bitova instrukcije) ili sadržaj nekog registra (registra koji se u zapisu instrukcije označava sa Rm, a čija oznaka se nalazi u 4 bita najmanje težine b_{3-0}). Znak ofseta je specificiran operacionim kodom (OP code).

LDR Rd,[Rn,#offset] Rd←[[Rn]+offset]

Uočavamo da je odredište na prvom mestu.

LDR Rd,[Rn,Rm] Rd←[[Rn]+[Rm]]

Instrukcije za pristup memoriji i adresni modovi kod ARM

Ofset 0 ne mora da se specificira direktno.

LDR Rd,[Rn] Rd←[[Rn]]

Operacioni kod mnemonika specificira učitavanje reči iz memorije. Ako je potrebno da se učita bajt u registar, tada se koristi mnemonik LDRB. Pri izvršenju ove instrukcije, svi bajtovi registra, osim bajta najniže težine, biće postavljeni na nulu.
Instrukcije smeštanja u memoriju imaju mnemonike STR i STRB. Instrukcija STR prenosi reč-operand u memoriju. Na primer,

STR Rd,[Rn] [Rn]←[Rd]

ARM dokumentacija sve ove adresne modove označava indeksnim adresnim modovima. Adresni mod koji je korišćen u dosadašnjim primerima naziva se preindeksni adresni mod, jer se efektivna adresa operanda dobija dodavanjem ofseta na sadržaj baznog registra Rn, a sadržaj registra Rn se ne menja. Postoje još dva adresna moda: preindeksni sa povratnim upisom i postindeksni.

Instrukcije za pristup memoriji i adresni modovi kod ARM

Kod preindeksnog adresnog moda sa povratnim upisom, efektivna adresa operanda se generiše na isti način kao i u prethodnom slučaju, a po generisanju efektivne adrese, ta adresa se povratno upisuje i u registar Rn.

Kod postindeksnog adresnog moda, efektivna adresa operanda je sadržaj registra Rn. Potom se ofset dodaje na tu adresu i dobijeni rezultat se povratno upisuje u registar Rn.

Sledeća tabela daje sumarni pregled adresnih modova za ARM.

Instrukcije za pristup memoriji i adresni modovi kod ARM

Name	Assembler syntax	Addressing function
With immediate offset:		
Pre-indexed	[Rn, #offset]	EA = [Rn] + offset
Pre-indexed with writeback	[Rn, #offset]!	EA = [Rn] + offset; Rn ← [Rn] + offset
Post-indexed	[Rn], #offset	EA = [Rn]; Rn ← [Rn] + offset
With offset magnitude in Rm:		
Pre-indexed	[Rn, ± Rm, shift]	EA = [Rn] ± [Rm] shifted
Pre-indexed with writeback	[Rn, ± Rm, shift]!	EA = [Rn] ± [Rm] shifted; Rn ← [Rn] ± [Rm] shifted
Post-indexed	[Rn], ± Rm, shift	EA = [Rn]; Rn ← [Rn] ± [Rm] shifted
Relative (Pre-indexed with immediate offset)	Location	EA = Location = [PC] + offset

EA = effective address
offset = a signed number contained in the instruction
shift = direction #integer
where direction is LSL for left shift or LSR for right shift, and
integer is a 5-bit unsigned number specifying the shift amount
±Rm = the offset magnitude in register Rm can be added to or subtracted from the contents of base register Rn

Instrukcije za pristup memoriji i adresni modovi kod ARM

Uočava se da se kod ARM-a uzvičnikom označava da se radi o preindeksnom adresnom modu sa povratnim upisom, a ne o "čistom" preindeksnom modu. Postindeksni adresni mod se specificira tako što se samo jedan bazni registar uokviruje uglastim zagradama, dok ofset (specificiran bilo neposredno, bilo registarski) ostaje van uglastih zagrada. Kod preindeksnog adresnog moda se oba operanda nalaze unutar istih uglastih zagrada.
U svim ovim adresnim modovima, ofset može biti dat kao neposredna vrednost iz intervala [-4095,+4095]. Alternativno, ako se ofset specificira pomoću registra, tada treći registar Rm sadrži apsolutnu vrednost ofseta, a znak ofseta je iskazan sa simbolom + ili – koji prethodi oznaci registra u instrukciji.
Tako je efekat instrukcije

LDR R0,[R1,-R2]!

izvršenje operacije R0←[[R1]-[R2]], uz učitavanje efektivne adrese [R1]-[R2] u registar R1 (zato što se koristi mod sa povratnim upisom).
Ako se ofset zadaje pomoću registra, tada se on može skalirati stepenom dvojke, tako što će biti šiftovan ulevo ili udesno za potreban broj mesta. To šiftovanje se u naredbi zapisuje tako što se posle oznake registra Rm koji određuje ofset upiše smer pomeranja (LSL ili LSR) i broja pozicija za koje se vrši pomeranje (neposredna vrednost između 0 i 31).

Instrukcije za pristup memoriji i adresni modovi kod ARM

Na primer, instrukcija

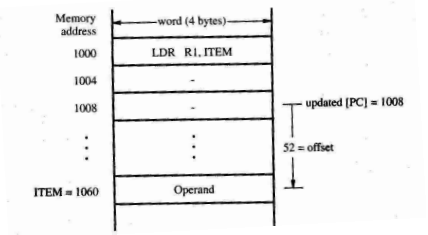
LDR R0,[R1,-R2,LSL #4]!

dovodi do R0←[[R1]-16*[R2]], uz učitavanje efektivne adrese [R1]-16*[R2] u registar R1 (zato što se koristi mod sa povratnim upisom).

Brojač naredbi, tj. registar PC (odnosno R15 kod ARM-a) takođe može biti korišćen kao bazni registar. U tom slučaju, isključivo se implementira se relativni adresni mod. Asembler određuje (kao neposrednu vrednost) pomeraj, računajući ga kao označeno rastojanje između adrese operanda i vrednosti ažuriranog registra PC. Kada se računa efektivna adresa u vremenu izvršavanja instrukcije, sadržaj registra PC je već ažuriran tako da adresira dve reči (8 bajtova) dalje u od instrukcije koja sadrži relativni adresni mod. Razlog za to je protočno izvršavanje instrukcija kod ARM-a.

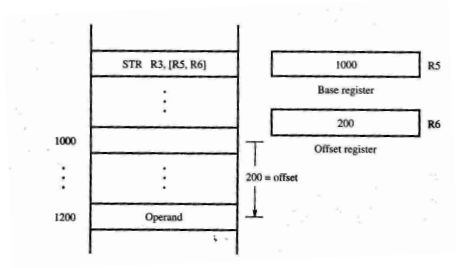
Instrukcije za pristup memoriji i adresni modovi kod ARM

Primer relativnog adresnog moda. Kod ARM-a nema apsolutnog moda. Dakle, kad god se adresa da na ovaj način koristi se relativni adresni mod, precisnije preindeksni adresni mod sa neposrednim ofsetom, gde je PC bazni registar. Operand mora biti do 4095 bajtova udaljen od ažuriranog PC. Ako to nije slučaj, asembler prijavljuje grešku.



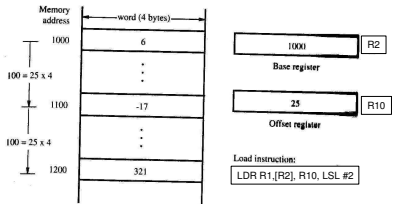
Instrukcije za pristup memoriji i adresni modovi kod ARM

Primer preindeksnog adresnog moda sa ofsetom u registru R6 i bazom R5. Instrukcija STR smešta sadržaj registra R3 na lokaciju 1200.



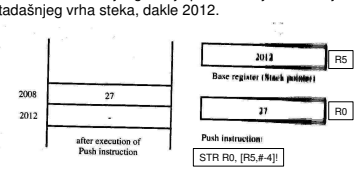
Instrukcije za pristup memoriji i adresni modovi kod ARM

Primer postindeksnog adresnog moda. Uočavamo prva tri člana liste od 25 brojeva, koji su na rastojanju od 25 reči, počev od memorijske adrese 1000. Oni mogu predstavljati elemente prve vrste matrice dimenzije 25x25 koja se čuva po kolonama – tada se elementi prve vrste nalaze na adresama 1100,1200,1300,...,3400 a elementi prve kolone na adresama 1000, 1004, 1008,.... 1096. Korišćenjem ove instrukcije u petlji sukcesivno bi se pristupalo elementima vrste date matrice.



Instrukcije za pristup memoriji i adresni modovi kod ARM

Primer preindeksnog adresnog moda sa povratnim upisom. Na ovaj način je realizovano guranje na stek vrednosti registra R0. Registar R5 je korišćen kao pokazivač na vrh steka. Taj registar je pre izvršenja instrukcije sadržavao vrednost tadašnjeg vrha steka, dakle 2012.



Instrukcije za pristup memoriji i adresni modovi kod ARM

Postoje i instrukcije za čitanje i upis većeg broja operanada, tzv. Instrukcije za blokovski transfer. Njima se može pročitati ili upisati vrednost ma kog podskupa skupa opštih registara ARM-a. Kod ovih instrukcija se radi samo sa rečima i koriste se operacioni kodovi LDM (Load Multiple) i STM(Store Multiple). Memorijski operandi se nalaze u uzastopnim rečima memorije. Dostupni su svi prethodno opisani oblici preindeksiranja i postindeksiranja, sa ili bez povratnog upisa. Lista registara koji se javljaju u instrukciji mora da bude u rastućem redosledu.
Na primer, pretpostavimo da se u registru R10 nalazi vrednost 1000 i da se izvršava sledeća instrukcija:

LDMIA R10!, [R0,R1,R6,R7]

Njen efekat je prenos reči sa lokacije 1000 u R0, sa lokacije 1004 u R1, sa lokaciji 1008 u R6 i sa lokacije 1012 u R7, pri čemu se posle poslednjeg prenosa u registru R10 nalazi 1016. Sufiks IA kod mnemonika prethodne instrukcije označava “increment after” koje odgovara postindeksiranju.

Ove instrukcije se intenzivno koriste pri implementaciji potprograma.

Instrukcije za registarski prenos kod ARM

Često je potrebno preneti sadržaj jednog registra u drugi ili učitati neposrednu vrednost u registar.
Instrukcija

MOV Rd, Rm

kopira sadržaj registra Rm u registar Rd.

Osam bitova najniže težine kod instrukcije može biti korišćeno kod MOV instrukcije za prenos neposredne vrednosti u registar.
Instrukcija

MOV R0, #76

smešta neposrednu vrednost 76 u registar R0.

Kod oba tipa MOV instrukcija, izvorni operand može biti šiftovan pre nego što se smesti u odredište.

Aritmetičke instrukcije kod ARM

Osnovni oblik ovih instrukcija je: OPcode Rd, Rn, Rm

Instrukcija

ADD R0, R2, R4

izvršava sledeću operaciju

R0 ← [R2]+[R4]

Instrukcija

SUB R0, R5, R6

izvršava sledeću operaciju

R0 ← [R6]-[R5]

Operand može biti i neposredno zadat. Tako, instrukcija

ADD R0, R3, #17

izvršava sledeću operaciju

R0 ← [R3]+17

Aritmetičke instrukcije kod ARM

Drugi operand može biti šiftovan ili rotiran pre izvršenja operacije. Instrukcija

ADD R0, R1, R5, LSL #4

izvršava sledeću operaciju

R0 ← [R1]+16*[R5]

Obezbeđene su dve verzije instrukcije **množenja**. U prvoj verziji se množe sadržaji registara i niža 32 bita rezultata se upisuju u odredišni registar.

MUL R0, R1, R2

izvršava sledeću operaciju

R0 ← [R1]*[R2]

Druga verzija obezbeđuje i četvrti registar, čiji se sadržaj dodaje na dobijeni proizvod pre smeštanja u odredišni registar. Tako, instrukcija

MLA R0, R1, R2, R3

izvršava sledeću operaciju

R0 ← [R1]*[R2]+[R3]

Ovo je tzv. "pomnoži i akumuliraj" operacija, koja je često potrebna kod numeričkih algoritama za digitalnu obradu signala.

Šiftovanje i rotacija kod ARM

Još jedna karakteristika koja ARM razlikuje od najvećeg broja drugih procesora je činjenica da su šiftovanje i rotacija inkorporirani u najveći broj instrukcija. Kod skoro svih ostalih skupova instrukcija za procesor, šiftovanje se realizuje posebnim instrukcijama – a takav je slučaj i kod Motorole 68000 i kod Intel IA-32 procesora.

Uključivanjem operacija šiftovanja i rotacije u instrukciju, ako su potrebne, ARM arhitektura štedi na prostoru kodova i potencijalno može poboljšati performanse izvršavanja u odnosu na konvencionalniji dizajn procesora.

Logičke instrukcije kod ARM

Logičke operacije AND, OR, XOR i Bit-clear su realizovane instrukcijama sa operacionim kodovima AND, ORR, EOR i BIC.

Instrukcija

AND Rd, Rn, Rm

izvršava sledeću operaciju

Rd ← [Rn] AND [Rm]

Instrukcija BIC je usko povezana sa AND. Ona komplementira svaki bit Rm pre nego što izvrši konjukciju sa Rn.

Instrukcija MVN (Move Negative) komplementira bitove izvora i smešta ih u odredište Rd.

Logičke instrukcije kod ARM

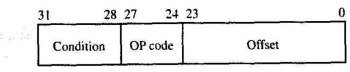
```
LDR     R0,POINTER      Load address LOC into R0.
LDRB    R1,[R0]         Load ASCII characters
LDRB    R2,[R0,#1]        into R1 and R2.
AND     R2,R2,#&F       Clear high-order 28 bits of R2.
ORR     R2,R2,R1,LSL #4 Or [R1] shifted left into [R2].
STRB    R2,PACKED       Store packed BCD digits
                          into PACKED.
```

Program za pakovanje dve četvorobitne dekadne cifre u koje se nalaze u uzastopnim memorijskim lokacijama LOC i LOC+1 i njihovo smeštanje u Lokaciju PACKED

Instrukcije grananja kod ARM

Instrukcije uslovnog grananja sadrže označeni 24-bitni ofset zapisan kao broj u potpunom komplementu. Taj broj se dodaje na vrednost ažuriranog PC i na taj način se formira ciljna adresa skoka.

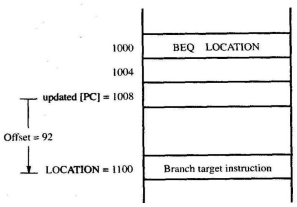
Format instrukcije grananja je prikazan na sledećem dijagramu:



Uslov koji se proverava da bi se utvrdilo da li će doći do skoka ili ne, određen je sa 4 bita najveće težine b_{31-28} u reči-instrukciji. Instrukcija grananja se izvršava na isti način kao i ma koja druga ARM instrukcija, tj. izvršava se samo ako tekuće stanje flegova status registra odgovara uslovu koji je specificiran bitovnom niskom Condition.

Instrukcije grananja kod ARM

Sledeći dijagram prikazuje primer korišćenja instrukcije grananja. Instrukcija BEQ (Branch if equal to 0) dovodi do skoka ako je Z fleg postavljen na 1.



U trenutku kada se računa ciljno adresa skoka, sadržaj PC je već ažuriran tako da sadrži adresu lokacije dve reči iza same instrukcije skoka. Ako je BEQ bila na adresi 1000, a ciljna adresa na lokaciji 1100, tada je ofset 92.

Postavljanje uslovnih kodova za ARM

Neke instrukcije, kao što je to slučaj sa instrukcijom poređenja, koja je data sa:

CMP Rn, Rm

a koja izvršava operaciju

[Rn]-[Rm]

Ima samo jedan cilj – podešavanje flegova uslovnog koda u status registru tako da odslikava rezultat dobijen oduzimanjem.

Sa druge strane, aritmetičke i i logičke operacije utiču na flegove uslovnog koda samo u slučaju kada se to eksplicitno specificira, postavljanjem bita u OP kodu instrukcije. U asemblerskom jeziku, postavljanje bita u OP kodu se postiže dodavanjem sufiksa S na mnemonik aritmetičke ili logičke instrukcije. Tako, na primer, instrukcija

ADDS R0, R1, R2

postavlja flegove uslovnih kodova u status registru, a instrukcija

ADD R0, R1, R2

ih ne postavlja.

Program za sabiranje brojeva kod ARM

Petlja za računanje zbira brojeva ima sledeći oblik:

```
        LDR   R1,N          Load count into R1.
        LDR   R2,POINTER    Load address NUM1 into R2.
        MOV   R0,#0         Clear accumulator R0.
LOOP    LDR   R3,[R2],#4    Load next number into R3.
        ADD   R0,R0,R3      Add number into R0.
        SUBS  R1,R1,#1      Decrement loop counter R1.
        BGT   LOOP          Branch back if not done.
        STR   R0,SUM        Store sum.
```

Instrukcije čitanja i upisa su izvršene od strane prve, druge i poslednje instrukcije – i to korišćenjem relativnog adresnog moda. Stoga, adrese memorijskih lokacija N, POINTER i SUM moraju da budu unutar opsega koji se može dohvatiti ofsetom, a koji je dat relativno u odnosu na PC.
POINTER sadrži adresu prvog sabirka u nizu, N sadrži nroj elemenata koji se sabiraju, a u SUM se smešta dobijeni zbir.

Asemblerski jezik za ARM

Asemblerski jezik za ARM sadrži asemblerske direktive kojima se:
- rezerviše memorijski prostor,
- dodeljuju numeričke vrednosti labelama koje predstavljaju memorijske adrese,
- dodeljuju numeričke vrednosti konstantnim simbolima,
- definiše gde će program i blokovi podataka biti smešteni u memoriji,
- specificira kraj koda koji predstavlja tekst programa.

Direktiva AREA sa argumentima CODE ili DATA, ukazuje na početak bloka u memoriji koji sadrži bilo instrukcije programa, bilo podatke.
Direktiva ENTRY određuje da izvršavanje programa počinje od instrukcije LDR koja neposredno sledi iza te direktive.
Direktiva DCD se koristi da se postavile labele i da bi se inicijalizovali operandi – podaci.
Kod ARM-a heksadecimalne konstante imaju prefiks &, a konstante u osnovi n ($2 \le n \le 9$) se zapisuju kao n_xxx. Tako, na primer, 2_10010011 predstavlja binarnu konstantu. Dekadne konstante ne zahtevaju nikakv prefiks.
Direktiva EQU se koristi za definisanje simboličkih imena za konstante.
Direktiva RN se koristi za pridruživanje simboličkog imena registru. Imena R0 do R15, kao i PC su već unapred pridružena odgovarajućim registrima.

Asemblerski jezik za ARM

Primer za neke od ARM asemblerskih direktiva je dat sledećim kompletnim programom:

```
                    Memory address label   Operation   Addressing or data information
Assembler directives                       AREA        CODE
                                           ENTRY
Statements that                            LDR         R1,N
  generate                                 LDR         R2,POINTER
  machine                                  MOV         R0,#0
  instructions      LOOP                   LDR         R3,[R2],#4
                                           ADD         R0,R0,R3
                                           SUBS        R1,R1,#1
                                           BGT         LOOP
                                           STR         R0,SUM
Assembler directives                       AREA        DATA
                    SUM                    DCD         0
                    N                      DCD         5
                    POINTER                DCD         NUM1
                    NUM1                   DCD         3,-17,27,-12,322
                                           END
```


Pseudo instrukcije kod ARM

Pored učitavanja adrese NUM1 u registar R2 na način kako je to urađeno u prethodnom programu, postoji i alternativni način, koji je takođe obezbeđen asemblerskim jezikom ARM-a.
Naime, pseudoinstrukcija

ADR Rd, ADDRESS

učitava 32-bitnu adresu u registar Rd. Ovo nije prava mašinska instrukcija, već asembler bira pogodnu pravu mašinsku instrukciju kako bi implementirao pseudoinstrukciju.

Na primer, jedan način za implementiranje pseudoinstrukcije

ADR R2, NUM1

je kombinacija mašinske instrukcije

LDR R2, POINTER

i asemblerske direktive za deklaraciju podatka

POINTER DCD NUM1

Međutim, ovo nije jedini način.

Pseudo instrukcije kod ARM

Naime, u ovom konkretnom primeru postoji i efikasniji način, pa asembler bira taj način za implementaciju pseudoinstrukcije ADR. Kada je (kao što je to slučaj u prethodnom programu) vrednost adrese koja se učitava u opseg od 255 bajtova od trenutnog sadržaja PC (tj. od R15), tada instrukcija

ADD Rd, R15, #offset

može da se iskoristi za implementiranje pseudoinstrukcije ADR.
Ako se tako može uraditi, onda nam lokacija POINTER uopšte nije neophodna.
U prethodnom primeru, pseudoinstrukcija

ADR R2, NUM1

bi bila implementirana pomoću stvarne mašinske instrukcije

ADD R2, R15, #28

zato što je lokacija NUM1 28 bajtova iza ažuriranog PC registra u trenutku kada se izvršava ta instrukcija ADD (ovo naravno uz pretpostavku da memorijski blok za podatke neposredno sledi iza memorijskog bloka koji sadrži instrukcije).
Kako u realnom asemblerskom programu na kraju programa moraju postojati instrukcije koje vraćaju kontrolu operativnom sistemu, to se memorijski blok za podatke neće naći neposredno iza memorijskog bloka koji sadrži instrukcije – ali ovaj problem ćemo zasad ignorisati.

I/O operacije kod ARM

ARM arhitektura koristi memorijski mapiran I/O.

Pretpostavimo da bit na poziciji 3 u statusnim rečima (INSTATUS i OUTSTATUS) ulaznog i izlaznog uređaja (tastature i ekrana) sadrži kontrolni fleg, koji označavamo sa SIN i SOUT respektivno.

Nadalje, pretpostavimo da su registri sa podacima za tastaturu (DATAIN) i ekran (DATAOUT) smešteni na adresama u memoriji koje neposredno slede iza odgovarajućih statusnih registara uređaja INSTATUS+4 i OUTSTATUS+4
Na kraju, pretpostavimo da je pre početka čitanja adresa INSTATUS smeštena u registar R1, a da je pre početka upisa adresa OUTSTATUS smeštena u registar R2.

```
READWAIT  LDR   R3,[R1]
          TST   R3,#8
          BEQ   READWAIT
          LDRB  R3,[R1,#4]
```

```
WRITEWAIT LDR   R4,[R2]
          TST   R4,#8
          BEQ   WRITEWAIT
          STRB  R3,[R2,#4]
```

Tada se gornjim kodom realizuje čitanje znaka u registar R3 onda kada je pritisnut taster na tastaturi, odnosno slanje znaka iz R3 u registar DATAOUT, onda kada je ekran spreman da ga prihvati.

I/O operacije kod ARM

Prethodne dve rutine se (onako kako je to opisano u prošlom poglavlju) mogu kombinovati kako bi omogućile čitanje linije znakova sa ulaza, njihov smeštaj u memoriju i prikaz na ekranu. Sledeći segment ARM asemblerskog koda radi upravo to:

```
READ  LDR   R3,[R1]      Load [INSTATUS] and
      TST   R3,#8          wait for character.
      BEQ   READ
      LDRB  R3,[R1,#4]   Read the character and
      STRB  R3,[R0],#1     store it in memory.
ECHO  LDR   R4,[R2]      Load [OUTSTATUS] and
      TST   R4,#8          wait for display
      BEQ   ECHO           to be ready.
      STRB  R3,[R2,#4]   Send character to display.
      TEQ   R3,#CR       If not carriage return,
      BNE   READ           read more characters.
```

Pretpostavlja se da R0 sadrži adresu prvog bajta memorijskog prostora u koji se smešta pročitana linija, a R1-R4 imaju isto značenje kao u prošlim primerima.

Potprogrami kod ARM

Instrukcija BL(Branch and Link) se koristi za poziv potprograma. Oni rade na isti način kao i druge naredbe skokova, sa jednim dodatnim korakom. Povratna adresa, tj. adresa instrukcije koja sledi iza instrukcije BL, se učitava u registar R14, koji se ponaša kao link registar. Budući da potprogrami mogu biti ugnježdeni, sadržaj link registra mora da se sačuva na steku potprograma.

Potprogrami kod ARM

Sledeći program koji sabira brojeve, samo se sada koristi potprogram:

```
Calling program
        LDR     R1,N
        LDR     R2,POINTER
        BL      LISTADD
        STR     R0,SUM
        :
Subroutine
LISTADD STMFD   R13!,{R3,R14}   Save R3 and return address in R14 on
                                  stack, using R13 as the stack pointer
        MOV     R0,#0
LOOP    LDR     R3,[R2],#4
        ADD     R0,R0,R3
        SUBS    R1,R1,#1
        BGT     LOOP
        LDMFD   R13!,{R3,R15}   Restore R3 and load return address
                                  into PC (R15).
```


Potprogrami kod ARM

U prošlom primeru su parametri preneseni preko registara. Pozivajući program je prosledio potprogramu veličinu niza i adresu prvog člana niza preko registara R1 i R2, a potprogram je dobijenu sumu vratilo pozivajućem programu preko registra R0. Tokom svog rada potprogram je koristio registar R3.

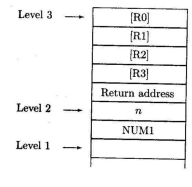
Zato se sadržaj svih prethodno pobrojanih registara , kao i sadržaj link registra R14 smeštaju na stek, korišćenjem instrukcije STMFD. Sufiks FD kod ove instrukcije ukazuje da stek raste prema nižim adresama i da se pokazivač na stek, tj. registar R13 dekrementira pre guranja reči na stek.

Instrukcija LDMFD restauriše sadržaj registra R3 i skida sa steka sačuvanu adresu povratka u registar PC (R15), čime se automatski izvrši povratak u pozivajući program.

Potprogrami kod ARM

Ovaj ARM program sabira brojeve, pri čemu se parametri prenose preko steka:

```
(Assume top of stack is at level 1 below.)
Calling program
        LDR     R0,POINTER      Push NUM1
        STR     R0,[R13,#-4]!     on stack.
        LDR     R0,N            Push n
        STR     R0,[R13,#-4]!     on stack.
        BL      LISTADD
        LDR     R0,[R13,#4]     Move the sum into
        STR     R0,SUM            memory location SUM.
        ADD     R13,R13,#8      Remove parameters from stack.
        :
Subroutine
LISTADD STMFD   R13!,{R0-R3,R14} Save registers.
        LDR     R1,[R13,#20]    Load parameters
        LDR     R2,[R13,#24]      from stack.
        MOV     R0,#0
LOOP    LDR     R3,[R2],#4
        ADD     R0,R0,R3
        SUBS    R1,R1,#1
        BGT     LOOP
        STR     R0,[R13,#24]    Place sum on stack.
        LDMFD   R13!,{R0-R3,R15} Restore registers and return.
```



Vrh steka u različitim trenucima

Potprogrami kod ARM

Ovde su parametri preneseni preko steka. Parametri NUM1 i N su gurnuti na stek pomoću prve četiri instrukcije pozivajućeg programa. Pretpostavljamo da se NUM1 sadrži u memorijskoj lokaciji POINTER. Registri R0 do R3 se unutar potprograma koriste na isti način kao i u prethodnom slučaju. Stoga će njihov sadržaj, zajedno sa adresom povratka (tj. sadržajem registra R14), biti sačuvan na steku prvom instrukcijom potprograma.

Po postavljanju svih parametara na stek izvršava se instrukcija BL, i vrh steka je na nivou 2. Po izvršenju prve instrukcije potprograma, kada vrednost svih registara bude sačuvana na steku, vrh steka je na nivou 3.

Sledeće dve instrukcije potprograma učitavaju parametre u registre R1 i R2 respektivno, koristeći ofsete od 20 i 24 bajta u odnosu na trenutni vrh steka, tj. Na nivo 3.

Kada se uma sračuna u R0, ona se ubacuje u stek instrukcijom STR, tako da "pregazi" vrednost NUM1.

Potprogrami kod ARM

Ovaj ARM program koristi ugnježdene potprograme:

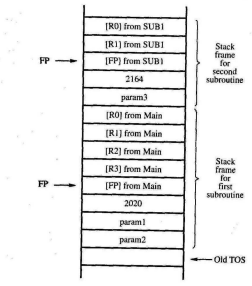
```
Memory
location        Instructions                Comments

Calling program
                :
2000            LDR   R10,PARAM2    Place parameters on stack.
2004            STR   R10,[SP,#-4]!
2008            LDR   R10,PARAM1
2012            STR   R10,[SP,#-4]!
2016            BL    SUB1
2020            LDR   R10,[SP]        Store SUB1 result.
2024            STR   R10,RESULT
2028            ADD   SP,SP,#8        Remove parameters from stack.
2032            next instruction
                :
First subroutine
2100  SUB1      STMFD SP!,{R0-R3,FP,LR}  Save registers.
2104            ADD   FP,SP,#16          Load frame pointer.
2108            LDR   R0,[FP,#8]         Load parameters.
2112            LDR   R1,[FP,#12]
                :
                LDR   R2,PARAM3     Place parameter on stack.
                STR   R2,[SP,#-4]!
2160            BL    SUB2
2164            LDR   R2,[SP],#4    Pop SUB2 result into R2.
                :
                STR   R3,[FP,#8]    Place result on stack.
                LDMFD SP!,{R0-R3,FP,PC}  Restore registers and return.

Second subroutine
3000  SUB2      STMFD SP!,{R0,R1,FP,LR}  Save registers.
                ADD   FP,SP,#8           Load frame pointer.
                LDR   R0,[FP,#8]         Load parameter.
                :
                STR   R1,[FP,#8]         Place result on stack.
                LDMFD SP!,{R0,R1,FP,PC}  Restore registers and return.
```


Potprogrami kod ARM

Stek okvir koji odgovara prethodnom programu je:



Kao pokazivač na stek-okvir koristi se registar R12.

Da bi se poboljšala čitljivost programa, korišćena su simbolička imena za registre: R12, R13, R14 i R15 su označeni sa FP, SP, LR i PC respektivno. Ta simbolička imena su dodeljena pomoću asemblerske direktive RN.

Poslednja instrukcija svakog od potprograma restauriše staru vrednost pokazivača na stek-okvir, kao i stare vrednosti registara koji su korišćeni i smesti adresu povratka sa steka u registar PC.

Primeri programa kod ARM

$$\text{Dot Product} = \sum_{i=0}^{n-1} \mathrm{A}(i) \times \mathrm{B}(i)$$

```
        ADR   R1,AVEC       R1 points to vector A.
        ADR   R2,BVEC       R2 points to vector B.
        LDR   R3,N          R3 is the loop counter.
        MOV   R0,#0         R0 accumulates the dot produ
LOOP    LDR   R4,[R1],#4    Load A component.
        LDR   R5,[R2],#4    Load B component.
        MLA   R0,R4,R5,R0   Multiply components and
                              accumulate into R0.
        SUBS  R3,R3,#1      Decrement the counter.
        BNE   LOOP          Branch back if not done.
        STR   R0,DOTPROD    Store dot product.
```


Primeri programa kod ARM

```
for  (j = n-1; j > 0; j = j - 1)
    { for ( k = j-1; k >= 0; k = k - 1 )
        { if   (LIST[k] > LIST[j])
               {  TEMP = LIST[k];
                  LIST[k] = LIST[j];
                  LIST[j] = TEMP;
               }
        }
    }
```

Sortiranje niza

```
        ADR   R4,LIST       Load list pointer register R4,
        LDR   R10,N           and initialize outer loop base
        ADD   R2,R4,R10       register R2 to LIST + n.
        ADD   R5,R4,#1      Load LIST + 1 into R5.
OUTER   LDRB  R0,[R2,#-1]!  Load LIST(j) into R0.
        MOV   R3,R2         Initialize inner loop base register
                              R3 to LIST + n - 1.
INNER   LDRB  R1,[R3,#-1]!  Load LIST(k) into R1.
        CMP   R1,R0         Compare LIST(k) to LIST(j).
        STRGTB R1,[R2]      If LIST(k) > LIST(j), interchange
        STRGTB R0,[R3]        LIST(k) and LIST(j), and
        MOVGT R0,R1           move (new) LIST(j) into R0.
        CMP   R3,R4         If k > 0, repeat
        BNE   INNER           inner loop.
        CMP   R2,R5         If j > 1, repeat
        BNE   OUTER           outer loop.
```


Primeri programa kod ARM

```
Subroutine
INSERTION  CMP    RHEAD,#0          Check if list empty.
           MOVEQ  RHEAD,RNEWREC     If empty, insert new
           MOVEQ  PC,R14              record as head.
           LDR    R0,[RHEAD]        If not empty, check if
           LDR    R1,[RNEWREC]        new record becomes
           CMP    R0,R1               new head, and
           STRGT  RHEAD,[RNEWREC,#4]  insert if yes.
           MOVGT  RHEAD,RNEWREC
           MOVGT  PC,R14
           MOV    RCURRENT,RHEAD    If new record goes after
LOOP       LDR    RNEXT,[RCURRENT,#4]  current head,
           CMP    RNEXT,#0             find where.
           STREQ  RNEWREC,[RCURRENT,#4]  New record becomes new tail.
           MOVEQ  PC,R14
           LDR    R0,[RNEXT]        Go further?
           CMP    R0,R1
           MOVLT  RCURRENT,RNEXT    Yes, then loop back.
           BLT    LOOP
           STR    RNEXT,[RNEWREC,#4]  Otherwise, insert new record
           STR    RNEWREC,[RCURRENT,#4]  between current and
           MOV    PC,R14                 next records.
```

Primeri programa kod ARM

Subroutine

```
DELETION  LDR     R0,[RHEAD]              Check if record to be
          CMP     R0,RIDNUM                 deleted is the head.
          LDREQ   RHEAD,[RHEAD,#4]        If yes, delete
          MOVEQ   PC,R14                    and return.
          MOV     RCURRENT,RHEAD          Otherwise, continue search.
LOOP      LDR     RNEXT,[RCURRENT,#4]     Is next record the one
          LDR     R0,[RNEXT]              to be deleted?
          CMP     R0,RIDNUM
          LDREQ   R0,[RNEXT,#4]           If yes, delete
          STREQ   R0,[RCURRENT,#4]          and return.
          MOVEQ   PC,R14
          MOV     RCURRENT,RNEXT          Otherwise, loop back
          B       LOOP                      to continue search.
```


Zadaci

-
-
-