
Osnovne informacije:

- 1983. Acorn Computers Ltd.
- Krajem 80-tih: Apple + Acorn = ARM Limited.
- Verzije arhitekture: v1-v6. Prve tri su zastarele. Imale su
  26-bitni adresni prostor. Verzija 4 predstavlja osnovu savremene
  32-bitne ARM arhitekture. Na vezbama cemo izucavati instrukcije
  ove arhitekture.
- RISC arhitektura. Uniformna duzina (32-bita) i format instrukcija.
  Relativno veliki broj registara opste namene (16).
- Load/Store arhitektura. Jednostavni (a mocni) adresni modovi.
- Moze biti little endian i big endian, zavisno od hardvera u koji se
  ugradjuje. Mi cemo na vezbama raditi na little endian platformi.

Dodatne karakteristike:

- Uslovno izvrsavanje instrukcija - instrukcija se moze izvrsiti ili
  ne u zavisnosti od ispunjenosti odredjenog uslova.
- Opciono azuriranje flegova - vecina instrukcija ne azuriraju obavezno
  flegove u statusnom registru, vec samo kada se to eksplicitno zahteva.
- Shiftovanje je opcioni deo vecine instrukcija. Ne postoje posebne
  instrukcije za shiftovanje.
- Memorijski pristup mora biti poravnat. Podaci velicine 4 bajta (reci)
  moraju biti na adresi deljivoj sa 4. Podaci velicine 2 bajta (polureci)
  moraju biti na adresi deljivoj sa 2. Instrukcije se uvek nalaze na
  adresi koja je deljiva sa 4.

Registri:

postoji 16 32-bitnih registara opste namene oznacenih sa r0-r15. Registar
r15 se oznacava i kao pc (program counter) i ima specijalnu ulogu.
Ovom registru se u vecini slucajeva moze pristupati kao i svim ostalim
registrima. Prilikom pisanja u ovaj registar vrsi se skok na upisanu
adresu. Prilikom citanja vrednosti ovog registra, dobija se adresa tekuce
instrukcije plus 8 (prefetching).

Pregled instrukcija:

Load/Store instrukcije.

Svi memorijski transferi se obavljaju instrukcijama iz ove grupe. OSTALE
INSTRUKCIJE NE MOGU IMATI MEMORIJSKE OPERANDE!!

- Ucitavanje iz memorije:

  LDR - ucitava jedan 32-bitni podatak.
  LDRB - ucitava jedan bajt, a ostatak registra je prosiren nulama.
  LDRH - ucitava 16-bitni podatak. Ostatak se prosiruje nulama.
  LDRSB - ucitava jedan bajt. Ostatak se popunjava bitom znaka.
  LDRSH - ucitava 16-bitni podatak. Ostatak se popunjava bitom znaka.

- Cuvanje u memoriju:

STR - kopira 32-bitni podatak u memoriju
STRB - kopira jedan bajt (bajt najmanje tezine) u memoriju.
STRH - kopira 16-bitni podatak (niza polovina registra) u memoriju.

ARM Load/Store adresni modovi:

- offset: bazni registar + offset koji moze biti +/- 12-bitna konstanta, ili +/- drugi registar, koji opciono moze biti shitovan za konstantan broj pozicija. Primeri:

```
ldr r0, [r1, #4]    @ ucitava rec sa adrese r1 + 4
ldr r0, [r1, #-8]   @ ucitava rec sa adrese r1 - 8
str r0, [r1, r2]    @ cuva vrednost r0 na adresu r1+r2
str r0, [r1, -r2, lsl #2]  @ cuva vrednost r0 na adresu r1 - 4*r2
```

NAPOMENA: Konstanta moze biti izmedju -4096, +4096
NAPOMENA: Kod instrukcija strh, ldrh, ldrsb i ldrsh nisu moguci shifteri, a konstanta mora biti osmobitna.
NAPOMENA: zapis [r1, #0] se krace zapisuje kao [r1] (bazno adresiranje).

- pre-indexed: isto kao i offset, samo sto se bazni registar azurira na izracunatu adresu. Npr:

```
ldr r0, [r1, #8]!      @ Ucitava rec sa adrese r1+8 a zatim registar r1
                       @ postavlja na r1+8 (kao ++r1 u C-u).
```

- post-indexed: isto kao i offset, samo sto se bazni registar azurira nakon sto se njegova originalna vrednost upotrebi kao adresa. Npr:

```
str r0, [r1], #4       @ Cuva r0 na adresu r1, a zatim r1 uvecava za 4.
                      @ (kao r1++ u C-u).
```

NAPOMENA: Moguci shifter-i su:
 -LSL: shiftovanje u levo
 -LSR: logicko siftovanje u desno
 -ASR: aritmeticko siftovanje u desno
 -ROR: rotacija u desno.
 -RRX: rotacija kroz C flag za jednu poziciju.

NAPOMENA: Instrukcije ne mogu imati apsolutnu adresu kao operand. Medjutim, u asembleru se moze navesti labela. Prilikom prevodjenja, ta labela ce biti zamenjena offset adresiranjem sa baznim registrom PC. Uslov je da labela nije predaleko u odnosu na tekucu instrukciju (+/-4K)

NAPOMENA: Instrukcija poput LEA IA-32 instrukcije ne postoji. Medjutim, postoji asemblerska pseudo-instrukcija ADR koja radi slicnu stvar, a koja se prilikom prevodjenja obicno prevodi u ADD ili SUB instrukciju. Ova instrukcija ima oblik:

adr <registar>, <labela>

Instrukcije za obradu podataka.

Ove instrukcije vrse jednostavnu obradu podataka kao sto su sabiranja i oduzimanja, zatim poredjenje i testiranje.

Instrukcije za izracunavanje:

- ADD - sabiranje dva operanda
- ADC - sabiranje sa prethodnim prenosom
- SUB - oduzimanje dva operanda
- SBC - oduzimanje sa pozajmicom
- RSB - obrnuto oduzimanje
- RSC - obrnuto oduzimanje sa pozajmicom
- AND - bitovska konjunkcija
- EOR - bitovska ekskluzivna disjunkcija
- ORR - bitovska dijunkcija

Gornje instrukcije imaju 3 operanda: prvi je odredisni registar, drugi je jedan source operand (uvek registar) a treci je drugi source operand (zadat kao shifter_operand, vidi dole). Instrukcije mogu imati sufiks S, u kom slucaju azuriraju flegove.

Instrukcije poredjenja i testiranja:

- CMP - uporedjivanje (oduzimanjem) i azuriranje flegova
- CMN - uporedjivanje (sabiranjem) i azuriranje flegova.
- TST - testiranje (bitovskom konjunkcijom) i azuriranje flegova
- TEQ - testiranje (ekskluzivnom disjunkcijom) i azuriranje flegova

Gornje instrukcije imaju dva operanda: prvi je registar, a drugi je shifter_operand. Uvek azuriraju flegove.

Instrukcije transfera medju registrima:

- MOV - premestanje iz registra u registar
- MVN - premestanje uz prethodno komplementiranje bitova

Gornje instrukcije imaju dva operanda: Prvi je registar, a drugi je shifter_operand. Mogu imati sufiks S, u kom slucaju azuriraju flegove.

shifter_operand je:
- 32-bitna konstanta (ne mogu bas sve konstante, vidi dokumentaciju)
- registar
- registar shiftovan za konstantan broj pozicija
- registar shiftovan za broj pozicija koji je dat u nekom drugom registru.

Primeri:

```
add r0, r0, #1      @ inkrementira r0
sub r0, r1, r2      @ r0 = r1 - r2
rsb r0, r1, r2, lsl #1      @ r0 = 2*r2 - r1
and r0, r1, r2, lsl r3      @ r0 = r1 & (r2 * 2^r3)

mvn r0, r0          @ negira bitove r0
rsb r0, r0, #0      @ upisuje -r0 u r0.
mov r0, r0, lsl #2  @ mnozi r0 sa 4.
```

Instrukcije mnozenja.

- MUL - ima tri operanda, i svi su regiistri: mnozi druga dva i smesta nizih 32 bita proizvoda u prvi operand. Nije bitno da li je oznaceno ili ne, jer nizi bitovi ne zavise od toga.
- MLA - ima cetiri operanda (svi su registri): mnozi druga i treci, sabira ih sa cetvrtim i nizih 32 bita te vrednosti smesta u prvi operand.

- UMULL - ima cetiri operanda. Prvi predstavlja odrediste nizih 32 bita, drugi predstavlja odrediste visih 32 bita, a treci i cetvrti predstavljaju cinioce. UMULL vrsi neoznaceno mnozenje.
- SMULL - isto kao i UMULL, samo vrsi oznaceno mnozenje.
- UMLAL - ima cetiri operanda. Prvi predstavlja nizih 32 bita operanda koji se dodaje na proizvod, kao i odrediste za nizih 32 bita rezultata. Drugi predstavlja visih 32 bita operanda koji se dodaje, kao i odrediste za visih 32 bita rezultata. Treci i cetvrti predstavljaju cinioce. Vrsi neoznaceno mnozenje.
- SMLAL - isto kao i UMLAL, samo oznaceno mnozenje.

Svi operandi instrukcija mnozenja su registri. Na pojedinim imlementacijama postoje ogranicenja da pojedini operandi ne smeju biti isti registar, tako da to treba izbegavati. Instrukcije mnozenja mogu imati sufiks S, u kom slucaju se azuriraju flegovi u statusnom registru.

NE POSTOJE INSTRUKCIJE ZA CELOBROJNO DELJENJE!!! Moze se simulirati uzastopnim oduzimanjem. Pod linux-om postoje u biblioteci definisane f-je __divsi3 i __udivsi3, kao i __modsi3 i __umodsi3 koje simuliraju date operacije nad oznacenim i neoznacenim brojevima.

Instrukcije grananja.

- B - ima jedan operand koji je labela. Labela se prevodi u relativnu adresu u odnosu na adresu PC, i mora biti u opsegu -32M - +32M. Instrukcija bezuslovno skace na datu adresu.

- BL - isto kao i B, s tim sto adresu instrukcije koja sledi nakon BL instrukcije smesta u registar r14. Ovaj registar se drugacije oznacava kao lr (link register).

Skok na proizvoljnu adresu se moze implementirati upisivanjem vrednosti zeljene adrese u PC registar.

Uslovno izvrsavanje instrukcija.

Vecina instrukcija ARM arhitekture se mogu izvrsavati uslovno. Ovo se postize dodavanjem odgovarajuceg uslovnog koda na mnemonik instrukcije. Najcesci uslovni kodovi su EQ, NE, GT, LT, GE, LE, HI, LO, HS, LS itd.

Pogledati dokumentaciju za detalje.
Primeri:

```
addgt r0, r0, r1   -  r0 = r0 + r1 samo ako je prethodno bilo vece.
subeq r1, r2, r3   -  r1 = r2 - r3 samo ako je prethodno bila jednakost.
mullts r1, r2, r3  -  r1 = r2 * r3 samo ako je prethodno bilo manje.
                      Azurira flegove nakon izvrsenja.
```

Uslovni skokovi.

Uslovni skokovi se na ARM arhitekturi implementiraju tako sto se instrukcije B i BL izvrse uslovno, dodavanjem uslovnih kodova. Npr.:

```
BGT - skace ako je vece
BEQ - skace ako je jednako
BNE - skace ako je razlicito
BLT - skace ako je manje
BHI - skace ako je vece, neoznaceno
BLGT - skace ako je vece, uz cuvanje povratne adrese u lr registru.
```

Opciono azuriranje flegova.

Instrukcije poredjenja i testiranja uvek azuriraju flegove. Ostale instrukcije to po default-u ne rade, osim kada im se doda sufiks S. Tada one azuriraju flegove na osnovu rezultata svoje operacije. Npr:

```
adds r0, r0, r1
```

azurira flegove na osnovu zbira r0+r1.

NAPOMENA: Ako instrukcija ima i uslovni kod i sufiks S, tada se uslovni kod u mnemoniku pise izmedju operacionog koda i sufiksa S (koji je uvek na kraju). Npr: addhis, subgts itd.

Instrukcije za ucitavanje i cuvanje skupa registara.

- LDM - instrukcija ucitava iz memorije vrednosti registara iz datog skupa.
- STM - instrukcija cuva u memoriju vrednosti registara iz datog skupa.

Instrukcije barataju sa skupom registara koji se zadaje izmedju zagrada { }, razdvojeni zarezima. Registri se ucitavaju (ili cuvaju) sa sukcesivnih memorijskih lokacija cija je adresa zadata baznim registrom koji se zadaje kao prvi operand. Instrukcije obavezno imaju sufiks koji odredjuje adresni mod, a moze biti jedan od sledecih (u donjoj notaciji Rb oznacava bazni registar, a k je broj registara u datom skupu):

- IA (Increment After) - Ovo znaci da se koriste memorijske lokacije Rb, Rb+4, Rb+8,...Rb+4*(k-1).
- IB (Increment Before) - Ovo znaci da se koriste memorijske lokacije Rb+4, Rb+8,...,Rb+4*k.
- DA (Decrement After) - Ovo znaci da se koriste memorijske lokacije

Rb,Rb-4,Rb-8,...,Rb-4*(k-1).
- DB (Decrement Before) - Ovo znaci da se koriste memorijske lokacije Rb-4, Rb-8, Rb-16,...,Rb-4*k

U svim slucajevima najniza navedena adresa se koristi za registar sa najmanjim brojem, dok se najvisa adresa koristi za registar sa najvisim brojem medju registrima u datom skupu. Npr, za skup {r2, r5, r7} na najnizu adresu ide r2, pa zatim r5 pa na kraju r7. Isti efekat bi bio i ako se navede {r5, r2, r7}.

Primeri:

```
ldmia r0, {r1, r2, r3} @ r1 = [r0], r2 = [r0+4], r3 = [r0+8]
ldmdb r0, {r0, r4, r2} @ r4 = [r0-4], r2 = [r0-8], r0 = [r0-12]
stmib r1, {r0, r1}     @ [r1+4] = r0, [r1+8] = r1
stmda r0, {r1, r2, r3} @ [r0] = r3, [r0-4] = r2, [r0-8] = r1
```

Ako se iza baznog registra stavi uzvicnik (!), tada se bazni registar nakon obavljene operacije azurira tako sto se uvecava za 4*k bajtova (u slucaju ia ili ib), ili se umanjuje za 4*k (u slucaju da ili db).

Primeri:

```
ldmia r0!, {r1, r2} @ r1 = [r0], r2 = [r0+4], r0 = r0+8
stmdb r0!, {r1, r2} @ [r0-4] = r2, [r0-8] = r1, r0 = r0-8
```

NAPOMENA: Kada se koristi simbol !, ne sme bazni registar biti u skupu, jer je rezultat nepredvidiv u tom slucaju.

Alternativni sufiksi (prilagodjeni stek operacijama):

- FD - Full Descending Stack
- ED - Empty Descending Stack
- FA - Full Ascending Stack
- EA - Empty Ascending Stack

Ponasanje alternativnih sufiksa, u odnosu na IA, DA, DB i IB:

```
LDMFA   =   LDMDA
LDMFD   =   LDMIA
LDMEA   =   LDMDB
LDMED   =   LDMIB
STMED   =   STMDA
STMEA   =   STMIA
STMFD   =   STMDB
STMFA   =   STMIB
```

NAPOMENA: Na nasoj platformi se koristi iskljucivo Full Descending Stack, tako da cemo pretezno koristiti instrukcije ldmfd i stmfd za rad sa stekom. Vrh steka se cuva u registru r13 (cije je alternativno ime sp).

Primeri:

```
stmfd sp!, {r0,r1}  @ Potiskuje na stek r0 i r1.
ldmfd sp!, {r0,r1}  @ Skida sa steka r0 i r1.
```

NAPOMENA: Instrukcije stm i ldm se mogu izvsavati i uslovno. U tom slucaju uslovni kod ide izmedju: ldmgtfd, stmeqia itd.

C konvencije o pozivanju funkcija:

Argumenti se prenose preko registara r0,r1,r2,r3 tim redom, s leva u desno. Ako funkcija ima vise od 4 argumenta, tada se preostali argumenti prenose preko steka, pri cemu se postavljaju na stek u obrnutom redosledu, pocev od poslednjeg. Stek uobicajeno raste prema nizim adresama.

Prema konvencijama, neki registri imaju specijalne uloge:
- r15 - pc (program counter)
- r14 - lr (link register)
- r13 - sp (stack pointer)
- r12 - ip (intra-procedure-call scratch register)
- r11 - fp (stack frame pointer)

Registri r0-r3 su tzv. "scratch" registri, tj. pripadaju pozvanoj funkciji. Registri r4-r10 pripadaju pozivajucoj funkciji. Funkcija je duzna da sacuva vrednosti registara r4-r10 kao i registara sp i fp. Ne mora da sacuva r0-r4 (koji se koriste za prenos argumenata) kao ni ip registar. Registar ip se moze koristiti i implicitno od strane linkera prilikom pozivanja "udaljenih" funkcija, tako da na ovaj registar svakako ne mozemo racunati da ce biti sacuvan nakon poziva funkcije.

Prilikom vracanja iz funkcije, po konvenciji, vrednost se smesta u r0.