UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA

ARHITEKTURA ARM

JAN BRATINA

2016
ZAKLJUČNA NALOGA
BRATINA

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Arhitektura ARM

(ARM architecture)

Ime in priimek: Jan Bratina
Študijski program: Računalništvo in informatika
Mentor: izr. prof. dr. Peter Korošec

Koper, september 2016

Ključna dokumentacijska informacija

Ime in PRIIMEK: Jan BRATINA

Naslov zaključne naloge: Arhitektura ARM

Kraj: Koper

Leto: 2016

Število listov: 51 Število slik: 11 Število tabel: 3

Število referenc: 21

Mentor: izr. prof. dr. Peter Korošec

Ključne besede: ARM, arhitektura, procesor, jedro, register, ukaz

Izvleček:

V tej zaključni nalogi je predstavljena arhitektura ARM. Procesorji, ki temeljijo na arhitekturi ARM, so prisotni vse okrog nas, vendar se tega premalo zavedamo. V nalogi spoznamo arhitekturo ARM, osnovne komponente procesorja ter razširitvene možnosti arhitekture ARM. V uvodu najprej spoznamo, kaj so vgrajeni sistemi. Nato sledi poglavje o zgodovini arhitekture ARM ter njenem razvoju. V naslednjem poglavju so predstavljene osnovne lastnosti arhitekture ARM. Da dobimo občutek, kakšni ukazi se izvajajo v ARM procesorjih, sem v naslednje poglavje vključil primere ukazov v zbirnem jeziku ter predstavil razširitve osnovnega nabora ukazov ARM. V nalogi so predstavljeni tudi sistemi na čipu ter komponente, ki jih ARM vključuje. V zadnjem poglavju pa predstavim najnovejšo verzijo arhitekture ARM, ki je prinesla večje spremembe. Cilj naloge je približati arhitekturo ARM ljudem.

Key words documentation

Name and SURNAME: Jan BRATINA

Title of the final project paper: ARM architecture

Place: Koper

Year: 2016

Number of pages: 51 Number of figures: 11 Number of tables: 3

Number of references: 21

Mentor: Assoc. Prof. Peter Korošec, PhD

Keywords: ARM, architecture, processor, core, register, instruction

Abstract:

This final thesis presents the ARM architecture. Processors based on the ARM architecture are all around us but we are not even aware of them. In this thesis, we get to know the ARM architecture, and basic processor components and extensions for the ARM architecture. At the beginning of this work, embedded systems are introduced, and they are followed by a chapter about the history and development of the ARM architecture. In the following section, its basic features are presented. There are also a few examples of assembler instructions included in the thesis, just to give a brief insight into what instructions in a microprocessor look like. Next, the system on the chip and other extension components are discussed. In the last section, the latest version of the ARM architecture is examined. The aim of this thesis is to familiarise people with the ARM architecture.

ZAHVALA

Zahvaljujem se mentorju izr. prof. dr. Petru Koroščcu za vso pomoč, potrpljenje ter zelo hitro odzivnost. Prav tako se zahvaljujem punci, celotni družini ter prijateljem za vso podporo, pomoč ter spodbujanje med študijem. Zahvaljujem se tudi podjetju Mahle Letrika za finančno podporo tekom študija.

KAZALO VSEBINE

KAZALO PREGLEDNIC

KAZALO SLIK

SEZNAM KRATIC

ARM	Napredni stroj RISC (angl. Advanced RISC Machine)
RISC	Računalnik s skrčenim naborom ukazov (angl. Reduced Instruction Set Computer)
CISC	Računalnik s širokim naborom ukazov (angl. Complex Instruction Set Computer)
POP	Procesorski paket za optimizacijo (angl. Processor Optimization Pack)
SIMD	Ena operacija nad več podatki (angl. Single Instruction Multiple Data)
ALU	Aritmetična logična enota (angl. Arithmetic Logic Unit)
VFP	Vektorska plavajoča vejica (angl. Vector Floating Point)
CPSR	Trenutni statusni register (angl. Current Program Status Register)
SPSR	Shranjen statusni register (angl. Saved Program Status Register)
FIQ	Hitra zahteva po prekinitvi (angl. Fast Interrupt Request)
IRQ	Zahteva po prekinitvi (angl. Interrupt Request)
PC	Programski števec (angl. Program Counter)
LR	Povezovalni register (angl. Link Register)
SP	Kazalec na sklad (angl. Stack pointer)
SoC	Sistem na čipu (angl. System-on-Chip)
LCD	Zaslon s tekočimi kristali (angl. Liquid Crystal Display)
RTOS	Operacijski sistem, ki se izvaja v realnem času (angl. Real-Time Operating System)
CPU	Centralni procesor (angl. Central Processing Unit)
GPU	Grafični procesor (angl. Graphics Processing Unit)
RAM	Bralno-pisalni pomnilnik (angl. Random Access Memory)
ROM	Bralni pomnilnik (angl. Read Only Memory)
UART	Univerzalni asinhroni sprejemalnik/oddajnik (angl. Univesal Asynchronus receiver/transmitter)
DSP	Procesor za obdelavo digitalnih signalov (angl. Digital Signal Processor)
GPS	Globalni pozicijski sistem (angl. Global Positioning System)
GPIO	Splošno namenski vhodno izhodni priključki (angl. General Purpose Input Output)
MMU	Enota za upravljanje s pomnilnikom (angl. Memory Management Unit)
AMBA	Napredna arhitektura vodil v mikrokrmilnikih (angl. Advanced Microcontroller Bus Architecture)
ASB	Napredno sistemsko vodilo (angl. Advanced System Bus)
APB	Napredno vodilo za zunanje naprave (angl. Advanced Peripheral Bus)
AHB	Visoko zmogljivo vodilo (angl. Advanced High-performance Bus)
AXI	Napreden razširljiv vmesnik (angl. Advanced eXtensible Interface)

ATB	Napredno vodilo za sledenje (angl. Advanced Trace Bus)
ACE	Usklajen napreden razširljiv vmesnik (angl. AXI Coherency Extension)

1 UVOD

Elektronske naprave so del našega vsakdana. V današnjem času se veliko napravam dodaja pridevnik pametna, na primer pametni telefon, pametna ura, pametni hladilnik, pametna televizija itn. Tem pametnim elektronskim sistemom pravimo tudi vgrajeni sistemi (angl. Embedded Systems). Vgrajen sistem je kombinacija strojne in programske opreme za opravljanje neke točno določene naloge oziroma procesa [11]. Prav zato, ker so vgrajeni sistemi namenjeni točno določeni nalogi, sta tako strojna kot tudi programska oprema optimizirani za reševanje tega problema, kar posledično prinese manjšo porabo energije, veliko zanesljivost ter nižjo ceno.

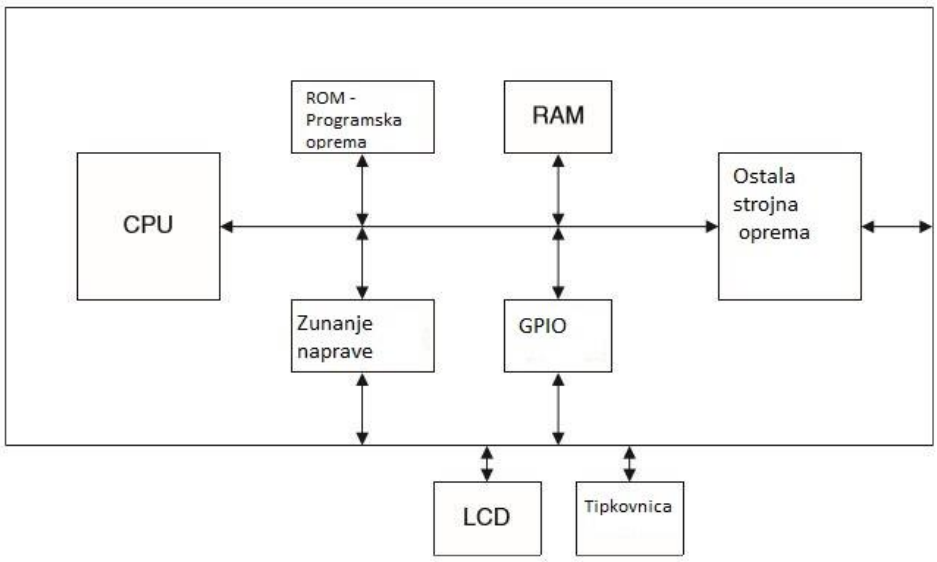
Srce vsakega vgrajenega sistema je mikroprocesor. V preprostejših vgrajenih sistemih imamo male 8-bitne mikroprocesorje, v zahtevnejših, kot je na primer mobilni telefon, pa uporabljamo že 64-bitne mikroprocesorje. Poleg mikroprocesorja imamo v vgrajenih sistemih tudi različne vrste pomnilnika, kot so bralno-pisalni pomnilnik RAM, bralni pomnilnik ROM ter bliskovni pomnilnik. Za komunikacijo med napravami se najpogosteje uporabljata protokola UART in I2C. Pomembna lastnost vgrajenih sistemov so tudi vhodno-izhodni priključki GPIO, na katere lahko priklopimo razne zunanje naprave, kot so na primer LCD zasloni, kar je vidno na sliki 1. Na tej sliki je prikazan osnovni vgrajeni sistem s centralnim procesorjem, bralno-pisalnim pomnilnikom, bralnim pomnilnikom, na katerem je naložena programska oprema ter ostala strojna oprema. Poleg tega sta na ta sistem prek GPIO-priključkov priključeni dve zunanji napravi in sicer LCD-zaslon ter tipkovnica. Vse komponente so med seboj povezane z vodili.

Moderni vgrajeni sistemi so sposobni poganjati različne operacijske sisteme, tako enostavne kot tudi kompleksne, med katere spadajo operacijski sistemi Linux in operacijski sistemi, ki se izvajajo v realnem času (RTOS). Operacijski sistemi v vgrajenih sistemih predstavljajo vmesni sloj med strojno opremo ter gonilniki in med aplikacijami. To omogoča razvoj aplikacij, ne da bi vedeli, kako natančno stvari delujejo na strojnem nivoju. Aplikacije so programska oprema na najvišjem nivoju, s katerimi običajni uporabnik tudi upravlja.

Najpogostejši mikroprocesorji v vgrajenih sistemih so prav mikroprocesorji, ki slonijo na arhitekturi ARM. Arhitektura ARM se je s časom zelo spreminjala ter prilagajala trenutnim potrebam trga. Kot sem že prej omenil, imamo v vgrajenih sistemih različne mikroprocesorje. Tako imamo v trenutni generaciji arhitekture ARM tri različne profile, ki se delijo glede na uporabo procesorjev. Profil Cortex-A je aplikacijski profil in se uporablja v napravah, ki za svoje delovanje potrebujejo operacijski sistem, kot so pametni mobilni telefon, tablice, pametne televizije in podobno. Te naprave so sposobne poganjati

različne aplikacije in imajo zato širok nabor funkcionalnosti. V zadnjem času se procesorje iz profila Cortex-A uporablja tudi v strežniških sistemih ter ostali mrežni opremi, kjer nizka poraba energije ni zanemarljiva, saj navadno ti sistemi neprestano delujejo. Procesorji iz profila Cortex-M so najmanjši procesorji v arhitekturi ARM ter se uporabljajo v preprostih mikrokrmilnikih. Ta jedra so cenovno zelo ugodna ter zelo energijsko varčna. Navadno za svoje delovanje ne potrebujejo operacijskega sistema, saj so večinoma namenjena periodičnemu branju podatkov iz raznih zunanjih naprav. Profil Cortex-M je arhitekturno precej drugačen od profila Cortex-A, saj uporablja drugačne načine delovanja, različna sta tudi nabor registrov ter rokovanje s prekinitvami. Procesorji iz profila Cortex-R so arhitekturno bližje aplikacijskemu profilu. Uporabljajo se v sistemih, ki se izvajajo v realnem času. Kot primer uporabe lahko vzamemo sistem proti blokiranju koles (ABS) v avtomobilu, kjer moramo hitro in natančno izračunati optimalno moč zaviranja. V takih sistemih ne smemo imeti velikih zakasnitev.

V tej zaključni nalogi bom podrobneje predstavil aplikacijski profil arhitekture ARM, saj je osnova za vse ostale profile. Cilj te zaključne naloge je predstaviti ARM-procesorje, saj predstavljajo del našega vsakdana, vendar se tega premalo zavedamo. Za dobro razumevanje delovanja vgrajenega sistema je ključno, da najprej poznamo delovanje procesorja ter njegove komponente.



Slika 1: Osnovna predstavitev vgrajenega sistema [11].

V nalogi bo najprej predstavljena zgodovina podjetja ARM ter razvoj arhitekture ARM. Sledi opis osnovnih lastnosti ter gradnikov arhitekture ARM. V naslednjem poglavju je predstavljen nabor ukazov ARM z vključenimi primeri ukazov v notaciji zbirnega jezika. V tem poglavju so tudi predstavljene razširitve osnovnega nabora ukazov ARM. Sledi

poglavje, v katerem so predstavljeni sistemi na čipu ter komponente, ki jih vključuje ARM. V zadnjem poglavju pa je opisana najnovejša verzija arhitekture ARM, ki je prinesla večje spremembe v primerjavi s prejšnjimi verzijami arhitekture.

2 ZGODOVINA PODJETJA ARM

Zgodnja osemdeseta leta prejšnjega stoletja predstavljajo obdobje hitrega ter velikega razvoja računalništva. Ameriško podjetje Apple je takrat predstavilo prvi osebni računalnik s 16-bitnim procesorjem ter z revolucionarnim grafičnim operacijskim sistemom [16]. V tem času je tudi podjetje IBM zaključilo svoj računalnik IBM PC, ki je prevladoval na poslovnem področju. V Evropi je takrat prevladovalo britansko podjetje Acron Computers, ki je leta 1981 za britansko medijsko hišo BBC izdelalo poceni osebni računalnik BBC Micro, s katerim so dosegli izjemen uspeh. V ta model so vgrajevali 8-bitni procesor MOS 6502, ki pa ni veljal za zelo zmogljivega. Za svoj naslednji produkt so si zadali zmogljivejši procesor, vendar na takratnem tržišču niso našli takega, ki bi zadoščal njihovim zahtevam. Tako so začeli z razvojem nove procesorske arhitekture, ki so jo poimenovali Acron RISC Machine ali krajše ARM [21].

Idejo za razvoj te arhitekture so povzeli po projektu Berkley RISC 1 [21]. Prve simulacije projekta so bile napisane v programskem jeziku BBC Basic na njihovem računalniku BBC Micro. S projektom so začeli leta 1983, le dve leti kasneje pa so predstavili svoj prvi procesor, ki hkrati velja za prvi komercialni RISC procesor na svetu. Imel je 26-bitno naslovno vodilo z 16 registri in 32-bitno podatkovno vodilo. Uporabljal je manj kot 25.000 tranzistorjev in je bil po zmogljivostih primerljiv s takratnim Intelovim paradnim konjem 80286. Prvo verzijo arhitekture ARM imenujemo tudi ARMv1 [14].

Leta 1987 podjetje predstavi procesor ARM2 ter z njim tudi drugo generacijo arhitekture ARM, to je ARMv2. Procesor ARM2 je imel tako kot predhodnik 26-bitno naslovno vodilo s 16 registri in 32-bitno podatkovno vodilo. Nova generacija je prinesla podporo za koprocesorje. Naslednji predstavljen procesor ARM3 je ravno tako spadal v generacijo ARMv2, prinesel pa je podporo integriranemu predpomnilniku [14].

Leta 1990 podjetja Acron, Apple in VLSI Technology ustvarijo novo podjetje ARM Limited. Pri Applu so si te združitve želeli predvsem zato, ker so sledili cilju, da bi ARM-procesorje uporabljali v svojem dlančniku Newton. Novoustanovljeno podjetje je leta 1992 predstavilo novo verzijo arhitekture ARMv3 [16], ki je prinesla podporo za 32-bitno naslovno vodilo ter enoto za upravljanje s pomnilnikom (MMU, angl. Memory Management Unit). Ta generacija tudi nakazuje ARM-ov prehod k uporabi njihovih procesorjev v vgrajenih in mobilnih napravah.

Naslednja, četrta generacija procesorjev ARM, ARMv4 je bila predstavljena leta 1996 [14]. V tej generaciji pridobimo nabor ukazov Thumb, ki so podmnožica najpogosteje uporabljenih 32-bitnih ukazov ARM. Nabor ukazov Thumb je podrobneje predstavljen v

poglavju 4.8. Najvidnejši predstavnik četrte generacije je jedro ARM7TDMI, ki je bilo uporabljeno v več milijonih naprav.

V peti generaciji arhitekture ARM, ARMv5 [21], dobimo 5-stopenjski cevovod ter izboljšano sodelovanje med naborom ukazov ARM ter naborom ukazov Thumb. Peta generacija prav tako prinese podporo za nabor ukazov Jazelle [14], ki omogoča poganjanje javanske bitne kode ter podporo za digitalno procesiranje signalov (angl. Digital Signal Processing). Procesorji iz pete generacije so bili večinoma uporabljeni v mobilnih telefonih, dlančnikih, vgrajenih napravah ter tudi v mrežnih napravah, kot so modemi in usmerjevalniki.

Leta 2001 je bila predstavljena šesta generacija, ARMv6 [14]. S to generacijo pridobimo osem in večstopenjske cevovode, podporo za SIMD-operacije, varnostno razširitev TrustZone, ki omogoča delitev strojne in programske opreme na varen in normalen del sistema, ter nabor ukazov Thumb-2 [6]. Glavna novost v šesti generaciji pa je podpora večjedrnim procesorjem.

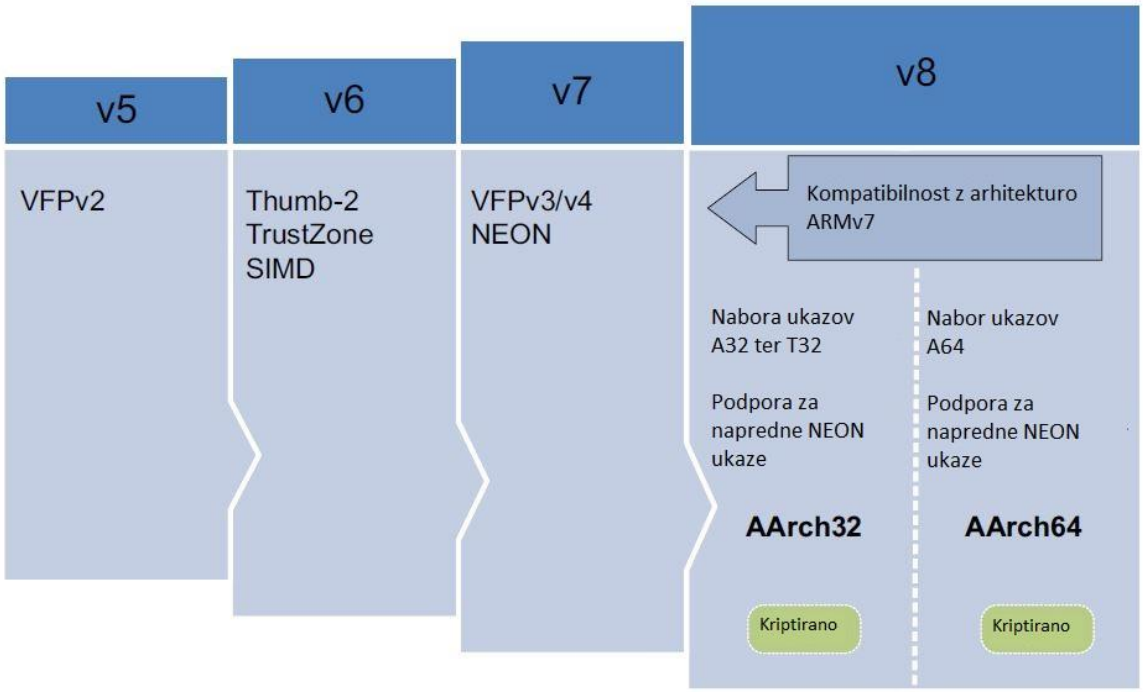
S sedmo generacijo, ARMv7, pridobimo enoto za izvajanje naprednih SIMD-ukazov, NEON [6]. V tej generaciji se prične delitev jeder na profile glede na njihovo uporabo, ki sem jih opisal že v uvodu. Zadnja generacija, ARMv8, je prinesla nov 64-bitni nabor ukazov ter hkrati ohranja združljivost z 32-bitnim naborom ukazov iz prejšnjih generacij [7].

Na sliki 2 so grafično predstavljene generacije arhitekture ARM od pete generacije pa do današnje, najnovejše, osme generacije. Pod vsako generacijo je napisana ključna pridobitev v primerjavi s prejšnjo generacijo.

ARM-procesorji so s svojo zelo nizko porabo energije primerni za uporabo v napravah, ki delujejo na akumulator. Leta 2015 je imel ARM več kot 85-% delež procesorjev v mobilnih napravah, kot so pametni telefoni, tablice in prenosni računalniki. Uporaba ARM-procesorjev pa se ne zaključi pri mobilnih napravah, saj ARM-procesorje najdemo tudi v ostalih vgrajenih napravah, ki so uporabljeni na primer v avtomobilski industriji. Na tem področju je imel ARM v letu 2015 25-% delež. V zadnjem času se uporaba ARM-procesorjev močno veča tudi v strežniških ter ostalih mrežnih sistemih, trenutno pokrivajo več kot 15 % trga [10].

Pomembno je poudariti, da podjetje ARM ne izdeluje procesorjev in vezij, temveč samo razvija procesorsko arhitekturo. Proizvodnja procesorjev je potem prepuščena njihovim poslovnim partnerjem, ki od podjetja ARM kupijo licence za izdelavo procesorjev. ARM

ponuja tri tipe licenc, in sicer licenciranje procesorja, procesorski paket za optimizacijo in licenciranje arhitekture. Pri licenciranju procesorja ARM ponudi predpisan dizajn, opis sestavnih delov ter predstavitev toka podatkov v jedru. Sama implementacija je potem prepuščena proizvajalcu. Pri licenci procesorskega paketa za optimizacijo podjetje ARM proda že končan čip z referenčnim delovanjem. Če se partnerji odločijo za nakup arhitekture, lahko arhitekturo ARM poljubno implementirajo v svoje produkte.



Slika 2: Razvoj arhitekture ARM ter ključne pridobitve pri vsaki generaciji [7].

3 LASTNOSTI ARHITEKTURE ARM

Arhitektura ARM temelji na 32-bitni procesorski arhitekturi RISC. Velika prednost arhitekture ARM je nabor različnih procesorjev za različne načine uporabe. Tako lahko za svoj produkt izberemo procesor, ki najbolj ustreza namenu našega produkta. Vsi procesorji se držijo osnovnih načel arhitekture ARM ter si delijo iste nabore ukazov. Lastnosti arhitekture ARM, predstavljene v tem poglavju, veljajo za procesorje do vključno sedme generacije arhitekture ARM.

3.1 Osnove arhitekture RISC in primerjava z arhitekturo CISC

RISC (angl. Reduced Instruction Set Computer) je eden od tipov procesorske arhitekture. Značilnost te arhitekture je, da uporablja razmeroma majhen nabor ukazov, ki se izvajajo v cevovodu. Vsi RISC ukazi se izvedejo v enem urinem ciklu, razen ukazov tipa naloži/shrani (angl. load/store). Glavna razlika od klasične arhitekture CISC (angl. Complex Instruction Set Computer) je v zapletenosti ukazov. Ukazi, ki jih izvajajo procesorji z arhitekturo CISC so kompleksnejši in se nato med izvajanjem prevedejo v več manjših mikro-operacij. Ukazi na arhitekturi CISC za izvajanje porabijo več kot en urin cikel. Nasprotno pa so ukazi na arhitekturi RISC enostavni ter enako dolgi [3]. Poznamo štiri glavne značilnosti arhitekture RISC:

- registri – registri v RISC procesorju lahko vsebujejo naslov ali podatek. Procesorji z arhitekturo RISC uporabljajo veliko število splošno namenskih registrov, vse operacije nad podatki se izvajajo prek registrov. Izjema sta ukaza naloži/shrani, s katerima lahko prenašamo podatke med registri in pomnilnikom. Procesorji z arhitekturo CISC imajo manjše število splošno namenskih registrov;
- ukaza naloži/shrani – procesor ne dostopa neposredno do pomnilnika, ampak do registrov, nad katerimi izvaja operacije. Posebna, ločena ukaza naloži/shrani se uporabljata za prenos podatkov med registri in pomnilnikom. Nasprotno ukazi na arhitekturi CISC lahko delajo neposredno s podatki v pomnilniku;
- ukazi – RISC-ukazi so optimizirani do te mere, da se izvedejo v enem urinem ciklu. RISC-prevajalnik lahko na višjem nivoju iz več preprostejših ukazov sestavi kompleksnejše ukaze, kot je na primer operacija deljenja [3]. Ukazi so fiksne dolžine, kar cevovodu omogoča, da naloži naslednji ukaz, še preden se trenutni ukaz izvede do konca. Pri arhitekturi CISC pa imamo ukaze različnih dolžin, ki za svoje izvajanje potrebujejo več urnih ciklov. Ukazi na arhitekturi CISC se ravno tako lahko izvajajo v cevovodu;
- cevovod – procesor izvaja ukaze po korakih, ki se vzporedno izvajajo v cevovodih. S tem izboljšamo učinkovitost procesorja, saj se naslednji ukaz začne izvajati, še preden se je trenutni zaključil. Za vsak korak v cevovodu se porabi en urin cikel. V

procesorjih z arhitekturo CISC pa ni nujno, da se za vsak korak v cevovodu porabi samo en urin cikel.

V začetku razvoja RISC-procesorjev je veljalo, da so enostavnejši ter da delujejo z višjo frekvenco, tradicionalni CISC-procesorji pa delujejo z nižjo frekvenco ter so kompleksnejši. V današnjem času še vedno ni jasno, katera arhitektura je boljša. Vsekakor pa si moderni procesorji izposojajo lastnosti iz obeh arhitektur.

3.2 Arhitektura ARM

ARM-procesorji temeljijo na arhitekturi RISC, ki pa je z leti razvoja prinesla nekaj sprememb v primerjavi s tradicionalno arhitekturo RISC. Uporaba ukazov fiksne dolžine 32-bitov ter uporaba ukazov naloži/shrani za manipuliranje s podatki sta dve najvidnejši lastnosti, ki jih je arhitektura ARM podedovala od arhitekture RISC. ARM-ovi načrtovalci arhitekture so se v želji po enostavnejši arhitekturi odločili za nekaj sprememb v primerjavi z arhitekturo RISC. Kot smo že v prejšnjem podpoglavju ugotovili, se RISC-ukazi izvedejo v enem urinem ciklu. Večina ARM-ukazov za obdelavo podatkov se še vedno izvede v enem urinem ciklu, medtem ko se ukazi za prenos podatkov iz pomnilnika ter obratno izvedejo v več urinih ciklih.

Prvi ARM-procesorji so temeljili na Von-Neumannovi arhitekturi, kasneje, v četrti generaciji arhitekture ARM, pa so prešli na Harvard arhitekturo. Glavna razlika med obema arhitekturama je, da ima Harvard arhitektura ločena pomnilnika za podatke ter za ukaze. Pri Von-Neumannovi arhitekturi pa imamo za ukaze ter za podatke skupni pomnilnik. Prav prehod na ločena pomnilnika omogoča, da se ukazi za dostop do pomnilnika lahko izvedejo v enem urinem ciklu. Obstajajo tudi ARM-ukazi, ki omogočajo nalaganje podatkov v več registrov naenkrat. Ti ukazi se sicer izvedejo v več kot enem urinem ciklu, vendar pa so zelo uporabni pri večjem prenosu podatkov. Za moderna ARM-jedra ne moremo reči, da izvedejo vse ukaze v enem urinem ciklu, saj so zasnovana tako, da ukaze izvedejo v najmanjšem možnem številu urinih ciklov.

Za ARM-procesorje lahko rečemo, da so v osnovi res RISC-procesorji, vendar so z leti razvoja pridobili tudi mnogo lastnosti iz arhitekture CISC. Ena izmed posebnosti arhitekture ARM je tudi nabor ukazov Thumb, ki uporablja ukaze dolžine 16-bitov, kar doprinese k varčnosti procesorja, saj pri ARM-procesorjih ni pomembna samo zmogljivost čipa, temveč tudi varčnost.

3.3 Načini delovanja

ARM-procesorji lahko delujejo v sedmih načinih delovanja, od katerih je šest načinov privilegiranih ter eden neprivilegiran. Privilegirani načini imajo polni bralno-pisalni dostop do trenutnega statusnega registra z oznako CPSR, medtem ko neprivilegirani način lahko spreminja samo pogojne zastavice v CPSR-registru. Načini delovanja v arhitekturi ARM so:

- uporabniški način (angl. USER) je edini neprivilegirani način delovanja ter je namenjen poganjanju aplikacij,
- v način hitre prekinitve (angl. FIQ) procesor vstopi, ko zazna, da je prišlo do prekinitve z visoko prioriteto,
- v način normalne prekinitve (angl. IRQ) procesor vstopi, ko zazna, da je prišlo do prekinitve z normalno prioriteto,
- v način programske prekinitve (angl. SVC) procesor vstopi, ko zazna, da je prišlo do programske prekinitve,
- v način nedefiniranega ukaza (angl. UNDEF) procesor vstopi, ko želi izvesti ukaz, ki ni podprt,
- v prekinitveni način (angl. ABORT) procesor vstopi, ko ne more dostopati do pomnilnika in
- sistemski način (angl. SYSTEM), ki je privilegiran uporabniški način, kar pomeni, da je v sistemskem načinu polni bralno-pisalni dostop do trenutnega statusnega registra CPSR.

Tabela 1: Načini delovanja ter bitna predstavitev v trenutnem statusnem registru [3].

Način delovanja	Bitna predstavitev v CPSR
USER	10000
IRQ	10010
FIQ	10001
SVC	10011
UNDEF	11011
ABORT	10111
SYSTEM	11111

V tabeli 1 so s kratico predstavljeni načini delovanja v arhitekturi ARM. Trenutni način delovanja lahko identificiramo tako, da preberemo prvih pet najmanj uteženih bitov trenutnega statusnega registra CPSR.

3.4 Registri

V arhitekturi ARM je do vključno sedme generacije 37 registrov, med katerimi imamo 30 splošno namenskih registrov, 6 statusnih registrov ter poseben register, ki se ga uporablja za programski števec. V enem izmed načinov delovanja, v katerem je trenutno procesor, je vedno na voljo 15 splošnih registrov ter programski števec. Ostali registri so med tem načinom delovanja skriti [18].

Registri od r0 do r7 so edini res splošno namenski registri, saj nimajo v nobenem načinu delovanja posebne vloge. Z izjemo uporabniškega načina FIQ so tudi registri od r8 do r12 splošno namenski. Ko je procesor v načinu hitrih prekinitev FIQ registre od r8 do r12, se zamenjajo registri od r8_fiq do r12_fiq. Ti registri lahko vsebujejo dodatne informacije, ki pospešijo izvajanje prekinitvene rutine.

Tabela 2: Pregled registrov ter njihova dostopnost v različnih načinih delovanja [18].

Načini delovanja						
	Privilegirani načini delovanja					
		Prekinitveni načini delovanja				
USER	**SYSTEM**	**SVC**	**ABORT**	**UNDEF**	**IRQ**	**FIQ**
r0	r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7	r7
r8	r8	r8	r8	r8	r8	r8_fiq
r9	r9	r9	r9	r9	r9	r9_fiq
r10	r10	r10	r10	r10	r10	r10_fiq
r11	r11	r11	r11	r11	r11	r11_fiq
r12	r12	r12	r12	r12	r12	r12_fiq
r13	r13	r13_svc	r13_abt	r13_und	r13_irq	r13_fiq
r14	r14	r14_svc	r14_abt	r14_und	r14_irq	r14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Splošno namenski registri lahko vsebujejo podatke dolžine 8, 16 ali 32 bitov. Kadar uporabljamo 32-bitni register v 8-bitnem ukazu, je v uporabi samo 8 najmanj uteženih bitov, podobno je tudi pri 16-bitnih ukazih, kjer je v uporabi 16 najmanj uteženih bitov. Registri od r13 do r15 so registri, ki se uporabljajo za določene namene: r13 je uporabljen kot kazalec na sklad, r14 je uporabljen kot povezovalni register in r15 kot programski števec.

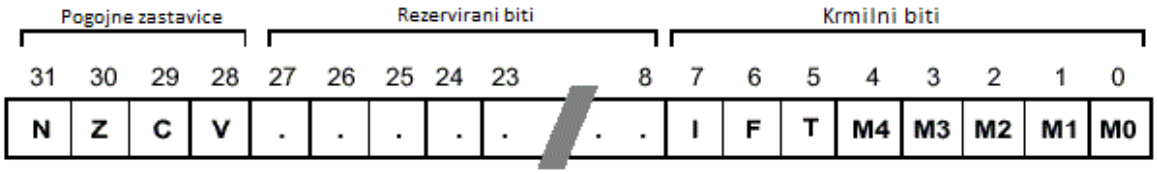
Uporabniški ter sistemski način delovanja uporablja iste registre, medtem ko ostali načini delovanja uporabljajo svoje verzije registrov r13 in r14. Ko smo na primer v prekinitvenem načinu delovanja IRQ, se namesto registrov r13 in r14 iz uporabniškega načina delovanja uporabijo registri r13_irq in r14_irq. Ti registri so v tabeli 2 označeni s sivo barvo. Do njih lahko dostopamo samo v privilegiranih načinih delovanja.

Glede na način delovanja imamo tudi enega ali dva statusna registra. Trenutni statusni register CPSR (angl. Current Program Status Register) vsebuje informacije o trenutnem stanju procesorja. V prekinitvenih načinih delovanja imamo še dodaten shranjen statusni register SPSR (angl. Save Program Status Register), ki vsebuje podatke o procesorskem stanju preden je prišlo do prekinitve.

3.4.1 Trenutni statusni register CPSR

Za nadzorovanje delovanja ter za upravljanje internih operacij je na ARM-procesorjih uporabljen trenutni statusni register (angl. Current Program Status Register). Register CPSR je 32-bitni register, predstavljen na sliki 3. Kot je vidno na tej sliki, lahko trenutni statusni register razdelimo na tri dele, in sicer na krmilne bite, na rezervirane bite ter na pogojne zastavice.

Zgornji štirje biti registra predstavljajo pogojne zastavice, ki so odvisne od rezultata zadnje operacije v aritmetični logični enoti, lahko jih tudi nastavimo kot pogoj različnim ukazom. Zastavica N se nastavi v primeru, da je bil rezultat negativen (angl. Negative), zastavica Z v primeru, ko je rezultat nič (angl. Zero), zastavica C, ko gre za prenos bita (angl. Carry), ter zastavica V, če pride do prekoračitve (angl. oVerflow). Prvih osem bitov služi za krmiljenje delovanja procesorja. Biti od 0 do 4 označujejo način delovanja procesorja in so podrobneje predstavljeni v tabeli 1, bit T označuje, ali smo v Thumb načinu delovanja, bita F in I pa označujeta prekinitvene maske za FIQ- in IRQ-prekinitve. To je generična predstavitev CPSR-registra, nekateri ARM-procesorji imajo poleg teh predstavljenih bitov še dodatne bite, kot sta recimo bit E, ki označuje vrstni red bitov, ter bit J, ki označuje, ali jedro podpira uporabo Jazelle nabora ukazov.



Slika 3: Trenutni statusni register.

Prekinitveni načini delovanja vsebujejo statusni register SPSR, ki je uporabljen za ohranitev vrednosti CPSR-registra, medtem ko se izvaja prekinitvena rutina.

3.4.2 Kazalec na sklad

Register r13 je uporabljen kot kazalec na sklad, zato ga tudi imenujemo SP-register (angl. Stack Pointer). Vsak prekinitveni način delovanja ima svojo različico registra r13, ki kaže na sklad, ki je točno določen za ta način delovanja.

V sklad shranjujemo začasne vrednosti. Navadno v sklad naložimo podatke, ki jih funkcija potrebuje za delovanje, preden vstopimo v proceduro (angl. subroutine). Ko se procedura izvede, lahko funkcija pridobi nazaj podatke iz sklada. Na ta način procedura ne spreminja vrednosti registrov in posledično vrednosti podatkov.

3.4.3 Povezovalni register

Register r14 je uporabljen kot povezovalni register, zato ga imenujemo tudi LR (angl. Link Register). Vanj shranimo povratni naslov za vrnitev iz procedure, zato moramo kopirati vrednost iz povezovalnega registra v register programskega števca [18]. To lahko naredimo na dva načina:

- z izvedbo enega od spodnjih ukazov

  ```
  MOV   PC, LR    ali
  BAL LR
  ```

  Ukaz MOV vzame vrednost iz registra LR ter jo premakne v register PC. Vsebina registra LR se med izvajanjem tega ukaza ne spremeni. Ukaz BAL se vedno izvede, saj je to nepogojni vejitveni ukaz s povezavo (vejitveni ukazi so podrobneje opisani v poglavju 4.2);
- lahko ob vstopu v proceduro kopiramo vrednost registra r14 na sklad z ukazom

  ```
  STMFD   SP!, {{registri}, LR}
  ```

  in nato ob izhodu iz procedure izvedemo ukaz za pridobivanje podatkov iz sklada

  ```
  LDMFD   SP!, {{registri}, PC}
  ```

  Ukaza STMFD ter LDMFD sta ukaza za prenos več registrov, ki uporabljata poln padajoči sklad (angl. full descending stack). Pogoj SP! pomeni, da bo za kazalec na

sklad uporabljen register SP. Za vstop v proceduro v naslednjem pogoju navedemo vse registre, ki jih potrebujemo, ter vrednost povezovalnega registra in jih shranimo na sklad. Pri izhodu iz procedure pa v naslednjem pogoju navedemo registre, ki jih bomo naložili iz sklada, ter zapišemo povratni naslov v programski števec.

Vsak prekinitveni način delovanja ima svojo različico registra r14.

3.4.4 Programski števec

Register r15 vsebuje naslov naslednjega ukaza v vrsti za izvajanje. Imenujemo ga tudi PC-register (angl. Program Counter). Programski števec nam da podatek, kateri ukaz se bo naslednji izvedel. Ker programski števec vsebuje naslov ukaza, ki se bo naslednji izvedel, mu pravimo tudi kazalec na ukaz.

Ko z ukazom pišemo na programski števec, se vrednost, ki smo jo napisali, obravnava kot naslov ukaza. Tako sistem začne z obdelavo ukaza na tem naslovu.

3.5 Izjeme in nadzor prekinitev

Izjema je dogodek, ki povzroči prekinitev normalnega izvajanja ukazov. Nadzornik prekinitev pa je program, ki se izvede, ko do prekinitve pride. Nadzornik prekinitev najprej ugotovi vzrok, zakaj je sploh do prekinitve prišlo, in nato prekinitev obdela. Izjeme v ARM-procesorjih lahko glede na vzrok razdelimo v tri skupine:

- izjeme, ki so neposredni rezultat izvajanja ukaza, na primer programske prekinitve, nedefinirani ukazi ter ukazi z nepravilnim dostopom do pomnilnika,
- izjeme, ki se zgodijo kot stranski učinek izvajanja ukaza, na primer napake med izvajanjem ukaza naloži/shrani, in
- zunanje izjeme – ponovni zagon (angl. reset), zahteva po prekinitvi IRQ ali zahteva po hitri prekinitvi FIQ [21].

Ko pride do izjeme, mora jedro shraniti naslov, ob katerem je bila prekinitev zaznana. To stori tako, da kopira vrednost iz CPSR v SPSR ter kopira vrednost programskega števca in jo shrani v povezovalni register. Nato procesor nastavi programski števec na tabelo vektorjev. Tabela vektorjev je posebno območje v pomnilniku, ki vsebuje naslove različnih nadzornikov prekinitev, kot je vidno v tabeli 3. Naslovi nadzornikov prekinitev so vedno enaki in se začnejo na naslovu 0x00000000. Jedro mora nato naložiti primeren naslov nadzornika prekinitev v programski števec. Sedaj se lahko prične prekinitvena rutina. Ko se prekinitvena rutina izvede, se moramo vrniti na mesto, kjer je do prekinitve prišlo. Jedro

tako kopira vrednost iz SPSR nazaj v CPSR ter obnovi vrednost programskega števca iz povezovalnega registra.

Tabela 3: Tabela vektorjev ter naslovi v pomnilniku [21].

Ime izjeme	Način izvajanja	Naslov
Ponovni zagon	SVC	0x00000000
Nedefiniran ukaz	UNDEF	0x00000004
Programska prekinitev	SVC	0x00000008
Poskus izvajanja ukaza iz neveljavnega naslova	ABORT	0x0000000C
Branje ali pisanje podatkov iz neveljavnega naslova	ABORT	0x00000010
/	/	0x00000014
Zahteva to prekinitvi	IRQ	0x00000018
Hitra zahteva po prekinitvi	FIQ	0x0000001C

Med izvajanjem programa lahko pride do sočasnih izjem, zato je potrebno določiti, katere izjeme imajo višjo prioriteto. Na ARM-jedrih so naslednji nivoji prioritet:

1. ponovni zagon,
2. branje ali pisanje podatkov iz neveljavnega naslova,
3. hitra zahteva po prekinitvi,
4. zahteva po prekinitvi,
5. poskus branja ukaza iz neveljavnega naslova in
6. programska prekinitev, nedefiniran ukaz.

Ponovni zagon ima najvišjo prioriteto, zato se vedno izvede pred vsemi drugimi izjemami. Ko pride do ponovnega zagona, je procesor v SVC-načinu izvajanja, nato nadzornik prekinitve inicializira sistem in ponastavi pomnilnik ter predpomnilnik.

Pri branju ali pisanju podatkov iz neveljavnega naslova pomnilnika (angl. data abort) je procesor v ABORT-načinu delovanja. Neveljaven naslov je lahko naslov izven velikosti našega pomnilnika, ali pa naslov, ki ni pravilno poravnan [18].

Izjema FIQ oziroma hitra zahteva po prekinitvi med svojim izvajanjem onemogoči ostale zahteve po prekinitvi. Procesor je med izvajanjem FIQ-izjeme v FIQ-načinu delovanja.
Izjemo IRQ lahko sproži neka zunanja naprava, ki želi pozornost procesorja. Procesor je med izvajanjem IRQ-izjeme v IRQ-načinu delovanja.

Kadar želimo prebrati ukaz na neveljavnem naslovu v pomnilniku, se sproži izjema poskus branja ukaza iz neveljavnega naslova (angl. prefetch abort). Procesor je med izvajanjem te izjeme v ABORT-načinu delovanja.

Izjemo programska prekinitev uporabnik sproži s klicem ukaza SWI. Procesor je med izvajanjem programske prekinitve v SVC-načinu izvajanja.

Izjema nedefiniran ukaz se sproži, kadar želimo izvesti ukaz, ki ga procesor ne pozna. Med izvajanjem te izjeme je procesor v UNDEF-načinu delovanja.

4 NABOR UKAZOV ARM TER RAZŠIRITVE

Ukazi v arhitekturi ARM delujejo nad podatki v registrih, saj lahko neposredno do pomnilnika dostopajo samo z ukazi tipa naloži/shrani. ARM-ukazi so običajno sestavljeni iz dveh ali treh operandov. Vsi ukazi so dolžine 32-bitov. Za predstavitev ukazov v tem poglavju sem uporabil notacijo zbirnega jezika ARM. Za primer lahko vzamemo ukaz ADD, ki vzame in sešteje vrednosti iz registrov r0 in r1 ter rezultat zapiše v register r2.

```
ADD   r2, r0, r1
```

V naslednjih podpoglavjih pa bomo spoznali različne razrede ARM-ukazov, in sicer ukaze za obdelavo podatkov, vejitvene ukaze, ukaze naloži/shrani, ukaze, ki sprožijo programske prekinitve, in ukaze za statusna registra. Da bi lažje razumeli, kako ukazi delujejo in kako so predstavljeni v zbirnem jeziku, bodo predstavljeni samo osnovnejši ukazi.

4.1 Ukazi za obdelavo podatkov

Najpreprostejši ARM-ukaz je MOV. Ukaz MOV premakne vrednost v ciljni register, kjer je vrednost lahko neka konstanta ali pa register. Ukaz je uporaben predvsem za premikanje podatkov med registri. V prvem primeru bo ukaz kopiral vrednost iz registra r1 v register r0, v drugem pa bo premaknil število 23 v register r0.

```
MOV   r0, r1
MOV   r0, #23
```

4.1.1 Pomikalni register

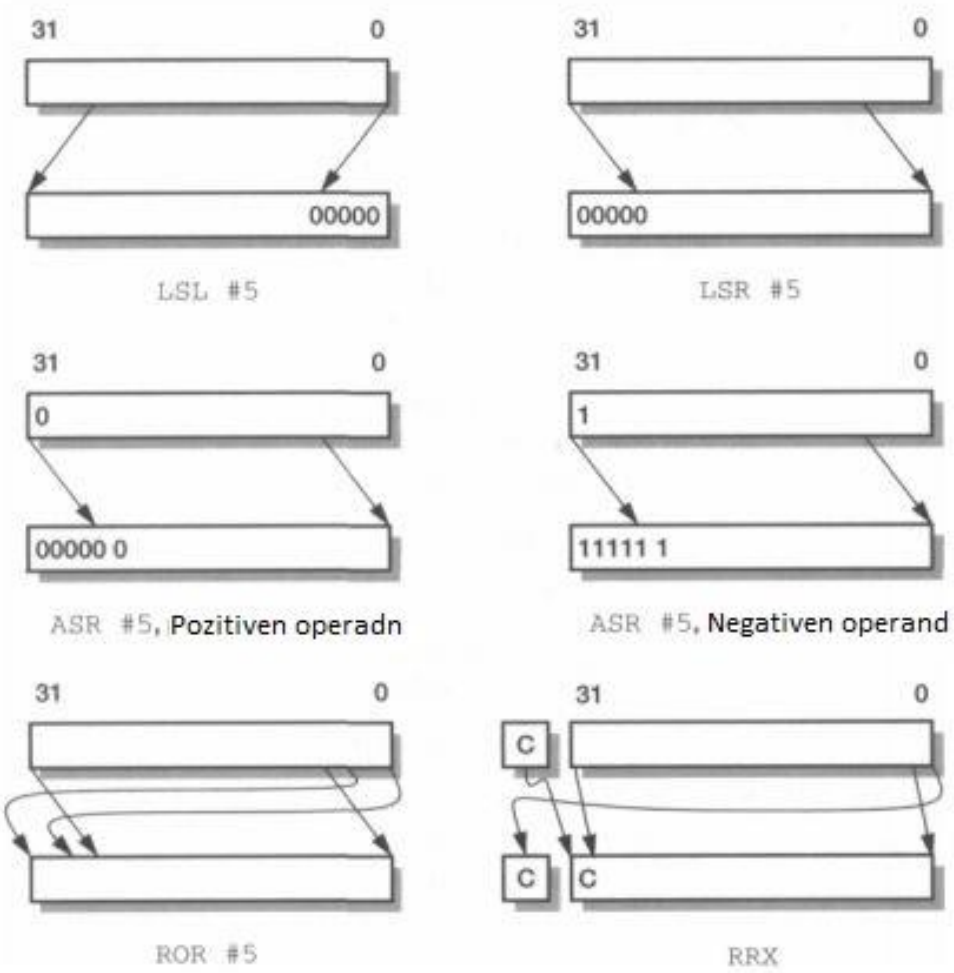
Pomikalni register (angl. Barrel Shifter) je še ena od posebnosti arhitekture ARM. Pomikalni register je strojna implementacija predprocesiranja podatkov v registrih. Tako lahko vrednost registra spremenimo, še preden pride do aritmetične logične enote. S pomikalnim registrom lahko vrednost registra premaknemo logično v levo ali desno. Premik se v pomikalnem registru izvede v času izvajanja ukaza [3].

Ukazi, ki so podprti v pomikalnem registru, so LSL, LSR, ASR, ROR ter RRX. Ukaz LSL (angl. logical shift left) logično premakne vrednost registra za 0–32 bitov v levo ter zapolni najmanj utežene bite z ničlami. Ukaz LSR (angl. logical shift right) logično premakne vrednost registra za 0–32 bitov v desno ter zapolni najbolj utežene bite z ničlami. Ukaz ASR (angl. arithmetic shift right) premakne vrednost registra v desno za 0–32 bitov. Če je uporabljen pozitiven operand, se najbolj uteženi biti zapolnijo z ničlami, če pa je uporabljen negativen operand, se najbolj uteženi biti zapolnijo z enkami. Ukaz ROR (angl. rotate right) zavrti bite v desno za 0–32 bitov, pri čemer se biti na najmanj uteženem delu registra prepišejo v najbolj utežen del registra. Ukaz RRX (angl. rotate right extended)

deluje podobno, kot če bi zavrteli bite v desno za en bit. Razlika pa je v tem, da se na bit številka 31 vpiše staro vrednost zastavice C iz trenutnega statusnega registra. Vrednost, ki je bila na najmanj uteženem bitu v registru, se zapiše v C-zastavico statusnega registra. Vsi opisani ukazi so za lažjo predstavo grafično predstavljeni na sliki 4.
Primer:

```
MOV   r0, r1, LSL #2
```

V tem primeru bo rezultat registra r0, vrednost registra r1 za 2 bita logično premaknjena v levo, pri čemer se vrednost registra r1 ne spremeni.



Slika 4: Predstavitev operacij v pomikalnem registru.

4.1.2 Aritmetične operacije

ARM-ukazi podpirajo operacije seštevanja ter odštevanja tako na predznačenih kot tudi ne-predznačenih 32-bitnih številkah. Kot primer lahko vzamemo operacijo odštevanja, ki odšteje vrednost registra r2 od vrednosti registra r1 in rezultat shrani v register r0.

```
SUB r0, r1, r2
```

Tudi pri aritmetičnih operacijah lahko uporabljamo pomikalni register in tako izboljšamo samo moč ukaza.

```
ADD r0, r1, r1, LSL #1
```

V tem primeru najprej register r1 v pomikalnem registru premaknemo logično levo za en bit, nato pa seštejemo to novo pridobljeno vrednost z vrednostjo iz r1 ter rezultat zapišemo v register r0. Ta preprost ukaz izveden v eni vrstici pomnoži vrednost registra s številom 3.

4.1.3 Logične operacije

Logične operacije lahko izvajamo nad dvema registroma. Logični ukazi, ki jih podpira arhitektura ARM, so: *logični in* (AND), *logični ali* (OR), *izključujoči ali* (EOR) ter *logični in ne* (BIC). Med logičnimi ukazi je zelo uporaben ukaz BIC, ki vzame dva registra r1 in r2, kjer vsaka binarna 1 iz registra r2 počisti isto ležeči bit v r1 ter vrednost shrani v r0, kot je vidno na primeru:

```
r1 = 11111111, r2 = 01010101
BIC   r0, r1, r2
r0 = 10101010
```

Tako kot pri aritmetičnih operacijah lahko tudi pri logičnih operacijah uporabljamo premikalni register, da si pripravimo podatke.

4.1.4 Primerjalni ukazi

S primerjalnimi ukazi posodabljamo vrednost pogojnih zastavic v CPSR-registru glede na rezultat operacije. Pogojne zastavice v CPSR-registru so predstavljene v poglavju 3.4.1. Primerjalni ukazi so uporabljeni za primerjanje ali testiranje registrov. Ti ukazi ne spreminjajo vrednosti v splošnih registrih. Rezultat teh ukazov so torej spremenjene pogojne zastavice v CPSR-registru, kar lahko uporabimo za nadaljnje izvajanje pogojnih ukazov. Tehnično gledano je ukaz primerjaj (CMP) enak ukazu odštevanja, vendar pri ukazu primerjaj rezultata nikamor ne zapišemo, temveč samo posodobimo pogojne zastavice v CPSR-registru.

4.2 Vejitveni ukazi

Vejitveni ukazi (angl. Branch instructions) spremenijo normalen potek programa. Ti ukazi omogočajo procedure, *if-else* stavke ter zanke. Sprememba poteka programa prisili programski števec, da kaže na nov naslov. Kot primer vejitvenega ukaza si poglejmo naslednji program, ki preskoči ukaza MOV in ADD ter izvede samo ukaz SUB:

```
B preskoci
MOV r0, r1
ADD r2, r3, #4
preskoci
SUB r2, r1, #5
```

Iz primera lahko vidimo, da ima vsaka vejitev svoje ime oziroma označbo. Ime, ki ga napišemo na začetek vrstice, označuje naslov, kjer se vejitev začne.

Pri ukazu vejitev s povezavo (BL) poleg vejitve tudi prepišemo vrednost povezovalnega registra. V povezovalni register vpišemo vrednost povratnega naslova, kar omogoča izvedbo procedur. Za vrnitev iz procedure moramo v programski števec naložiti vrednost iz povezovalnega registra [21].

Poznamo še dva vejitvena ukaza, in sicer BX in BLX, ki sta največkrat uporabljena za prehod v Thumb nabor ukazov ter nazaj iz njega. Med izvajanjem vejitvenega ukaza BX se bit T v registru CPSR nastavi na vrednost najmanj uteženega bita iz vejitvenega registra. Ukaz BLX deluje na podoben način kot ukaz BX, s tem da nastavi še povratni naslov v povezovalni register.

4.3 Ukazi naloži/shrani

Ukazi naloži/shrani so uporabljeni za prenos podatkov med pomnilnikom in registri. Poznamo tri tipe naloži/shrani ukazov, in sicer prenos enega registra, prenos več registrov ter zamenjava.

Ukazi za prenos enega registra se uporabljajo za prenos enega podatka v register ali iz njega. Prenašamo lahko 32-, 16- ali 8-bitne podatke. Pri uporabi ukazov naloži/shrani moramo paziti na poravnavo podatkov. Na primer ukaz STR lahko naloži 32-bitni podatek v pomnilnik na naslove, ki so večkratniki štirih zlogov.

Ukazi za prenos več registrov lahko prenesejo vsebino več registrov med pomnilnikom in jedrom v enem ukazu. Kot primer lahko vzamemo ukaza STM in LDM. Z enim ukazom STM lahko na sklad zapišemo vrednosti več registrov, povratni naslov ter posodobimo kazalec na sklad. Z enim ukazom LDM pa lahko obnovimo vrednosti več registrov iz sklada, v register PC zapišemo povratni naslov, ki smo si ga shranili z ukazom STM in posodobimo kazalec na sklad [18]. Prenos več registrov naenkrat je bolj učinkovit kot prenos enega samega registra, saj ni treba tolikokrat osvežiti sklada.

Z ukazom zamenjave zamenjamo podatke iz pomnilnika s podatki iz registrov. Ukaz zamenjave je atomarna operacija, kar drugim ukazom preprečuje branje ali pisanje na isto lokacijo, dokler se ukaz zamenjave ne izvede. V naslednjem primeru imamo v pomnilniku na naslovu 0x9000 zapisano vrednost 0x12345678. Ta vrednost se z ukazom SWP nato zapiše v r0, v pomnilnik na naslovu 0x9000 pa se zapiše vrednost registra r1, in sicer 0x11112222.

```
pomnilnik[0x9000] = 0x12345678
r0 = 0x00000000
r1 = 0x11112222
r2 = 0x00009000
SWP   r0, r1, [r2]
```

4.4 Ukaz za programsko prekinitev

Na arhitekturi ARM imamo poseben ukaz (SWI), ki povzroči programsko prekinitev. Ko se sproži programska prekinitev, se tudi zamenja procesorski način delovanja, in sicer preidemo v način SVC, ki omogoča privilegirano klicanje sistemskih klicev. Zamenjavo načina delovanja naredimo tako, da v register PC zapišemo naslov 0x00000008. Vsak sistemski klic ima določeno SWI-število, ki je odvisno od implementacije operacijskega sistema. SWI-število je tako uporabljeno za identifikacijo sistemskih klicev [3].

4.5 Ukazi za statusna registra

Nabor ukazov ARM ponuja dva ukaza za upravljanje s statusnima registroma CPSR in SPSR. Z ukazom MRS lahko premaknemo vrednosti iz statusnih registrov v neki drugi register. Za prenos podatkov v obratni smeri, iz nekega poljubnega registra v statusni register, pa uporabljamo ukaz MSR. Tako sta ukaza uporabljena za branje ali pisanje statusnih registrov. Pri tem pa moramo paziti, v katerem procesorskem načinu smo, saj v neprivilegiranem načinu lahko spreminjamo samo pogojne zastavice, ostale bite pa lahko samo beremo. Ukaza MSR in MRS sta v kombinaciji z logičnim ukazom BIC zelo uporabna za enostavno spreminjanje statusnih registrov bit po bit.

4.6 Pogojno izvajanje ukazov

Ena od posebnosti nabora ukazov ARM je, da lahko skoraj vse ukaze pogojno izvedemo. Pogojno izvajanje nadzirajo pogojne zastavice v CPSR-registru. Ukaz se normalno izvede, če so ustrezne pogojne zastavice postavljene, ko se ukaz začne izvajati. V nasprotnem primeru pa se ukaz ne izvede. Pogojni ukazi imajo svoje okrajšave, ki jih napišemo za ukaz. Če okrajšave za pogojni ukaz ni, se ukaz vedno izvede.

Kot primer lahko vzamemo naslednji ukaz:

```
MOVCS   R0, R1
```

Ukaz je sestavljen iz ukaza za premik vrednosti MOV ter pogojnega ukaza CS (angl. Carry Set). Ta sestavljen ukaz bo premaknil vrednosti registra r1 v register r0 samo, kadar je postavljena zastavica C, ki označuje prenos bita (angl. Carry). Če zastavica C v registru CPSR ni postavljena, se register r0 ne spremeni.

Ukazi za obdelavo podatkov lahko tudi spreminjajo pogojne zastavice v CPSR-registru glede na rezultat ukaza. To dosežemo tako, da za ukaz napišemo črko S. Primer:

```
MOVS   R0, #0
```

Ta ukaz bo premaknil vrednost 0 v register r0 ter ustrezno postavil pogojne zastavice. V tem primeru bo nastavil zastavico Z, počistil zastavico N, medtem pa se zastavici C in V ne bosta spremenili.

4.7 Ukazi za podporo koprocesorjem

Ena od razširitev nabora ukazov ARM je podpora za koprocesorske ukaze. Ti ukazi so dostopni samo na jedrih, ki imajo podporo za koprocesorje. Kaj so koprocesorji, je opisano v poglavju 5.7. Ukaza MRC ter MCR služita za premikanje podatkov v koprocesorske registre ter iz njih, ukaz CDP služi za obdelovanje podatkov v koprocesorju, imamo pa tudi posebna ukaza naloži/shrani, in sicer LDC ter STC. Ta ukaza omogočata nalaganje in shranjevanje večjih delov pomnilnika v koprocesor ali iz njega.

Kot primer vzemimo ukaz MRC, ki premakne vrednost registra koprocesorja CP15 v register r10. Pri vsakem koprocesorskem ukazu moramo navesti, s katerim koprocesorjem želimo delali. V našem primeru je to koprocesor CP15, ki ga v kodi navedemo kot p15. Število 0 označuje koprocesorsko operacijo, ki jo želimo izvesti. Te številke so odvisne od samega koprocesorja, in so opisane v koprocesorski dokumentaciji [6]. Registra c0 in c1 sta koprocesorska registra, katerih vrednosti želimo prenesti v register r10.

```
MRC p15, 0, r10, c0, c1
```

4.8 Thumb

Poleg nabora ukazov ARM poznamo na ARM-procesorjih še nabor ukazov Thumb. Ukazi Thumb so podmnožica najpogosteje uporabljenih 32-bitnih ukazov ARM, ki so skrajšani na 16-bitov. Zmanjšanje dolžine ukazov na 16-bitov prinese manjšo porabo pomnilnika ter posledično manjšo porabo energije. Med izvajanjem se 16-bitni Thumb ukazi v jedru

prevedejo v 32-bitne ARM-ukaze. Thumb za delovanje uporablja 12 registrov, ki so fizično na istem mestu kot registri iz ARM-nabora ukazov. To so registri od r0 do r7, katerih podatki se lahko prenašajo med programi, ki se izvajajo v ARM-načinu, ter programi, ki se izvajajo v Thumb načinu. Poleg prvih osmih registrov Thumb uporablja tudi registre od r13 do r15 ter trenutni statusni register CPSR. Thumb način ima lastna ukaza za upravljanje s skladom, in sicer ukaza PUSH in POP, ki v ARM-načinu ne obstajata. Tam ju zamenjata ukaza LDR in STR [3].

Z verzijo arhitekture ARMv7 pridobimo nov, razširjen nabor Thumb ukazov pod imenom Thumb-2, ki prinaša mešane 16- in 32-bitne ukaze. Z uporabo 32-bitnih Thumb-2 ukazov pridobimo na zmogljivosti, z uporabo 16-bitnih ukazov pa še vedno ohranjamo majhno velikost kode.

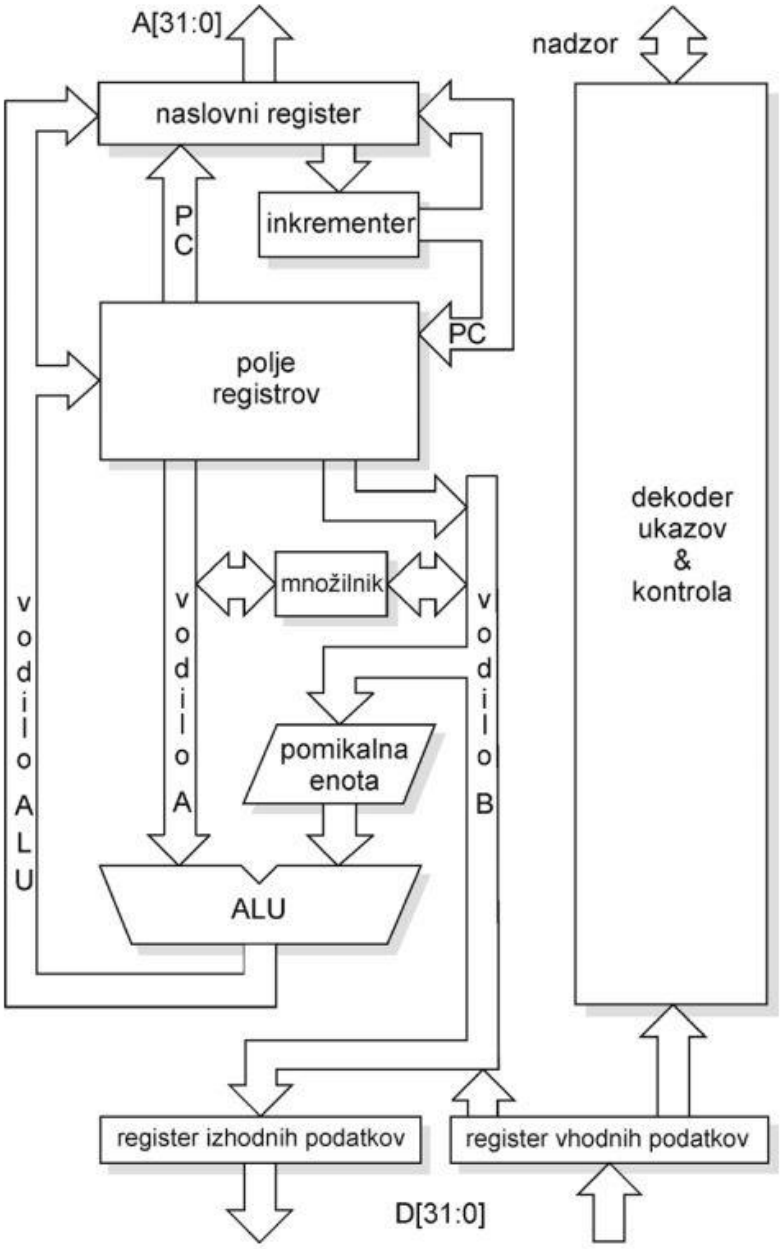
4.9 Cevovod

Cevovod je mehanizem za izvajanje ukazov. Z uporabo cevovoda pospešimo izvajanje programa, saj prevzamemo naslednji ukaz, še preden se trenutni ukaz zaključi. V različnih generacijah arhitekture ARM imamo cevovode različnih stopenj. Moderni ARM-procesorji, kot na primer cortex-A72, uporabljajo več kot 15-stopenjske cevovode, medtem ko so prvi ARM-procesorji uporabljali preproste 3-stopenjske cevovode. Pri 10- in večstopenjskih cevovodih, ki smo jih dobili s sedmo generacijo arhitekture ARM, dobimo tudi ločene cevovode za izvajanje naprednih SIMD-ukazov NEON ter dinamične cevovode za dinamično izvajanje ukazov z napovedovanjem. Razvoj ARM-cevovoda je opisan v naslednjih podpoglavjih.

4.9.1 3-stopenjski cevovod

Najpreprostejša implementacija cevovoda je 3-stopenjski cevovod z naslednjimi stopnjami: *prevzemi* prevzame ukaz iz pomnilnika ter ga postavi v cevovod ukazov, *dekodiraj* identificira ukaz, ki se bo izvedel, ter *izvedi*, ki ukaz izvede ter njegov rezultat vpiše nazaj v registre. Cevovod je najučinkovitejši pri izvajanju zaporedne kode, saj je tako lahko vedno napolnjen. V primeru, ko pride do vejitve, pa se cevovod izprazni in ga je potrebno ponovno napolniti z novimi ukazi. Pri izvajanju preprostih ukazov za obdelavo podatkov cevovod omogoča, da se ukaz zaključi v enem urinem ciklu. Pri ukazih tipa naloži/shrani rezultat ukaza ni vpisan neposredno v register, ampak je postavljen na naslovno vodilo. Nato v naslednji stopnji izvajanja prevzamemo rezultat iz naslovnega vodila ter dostopamo do pomnilnika. S tem pa povzročimo zastoj cevovoda, saj ne moremo naložiti naslednjega ukaza, medtem ko dostopamo do pomnilnika. To lahko rešimo z dvema ločenima pomnilnikoma za podatke ter ukaze.

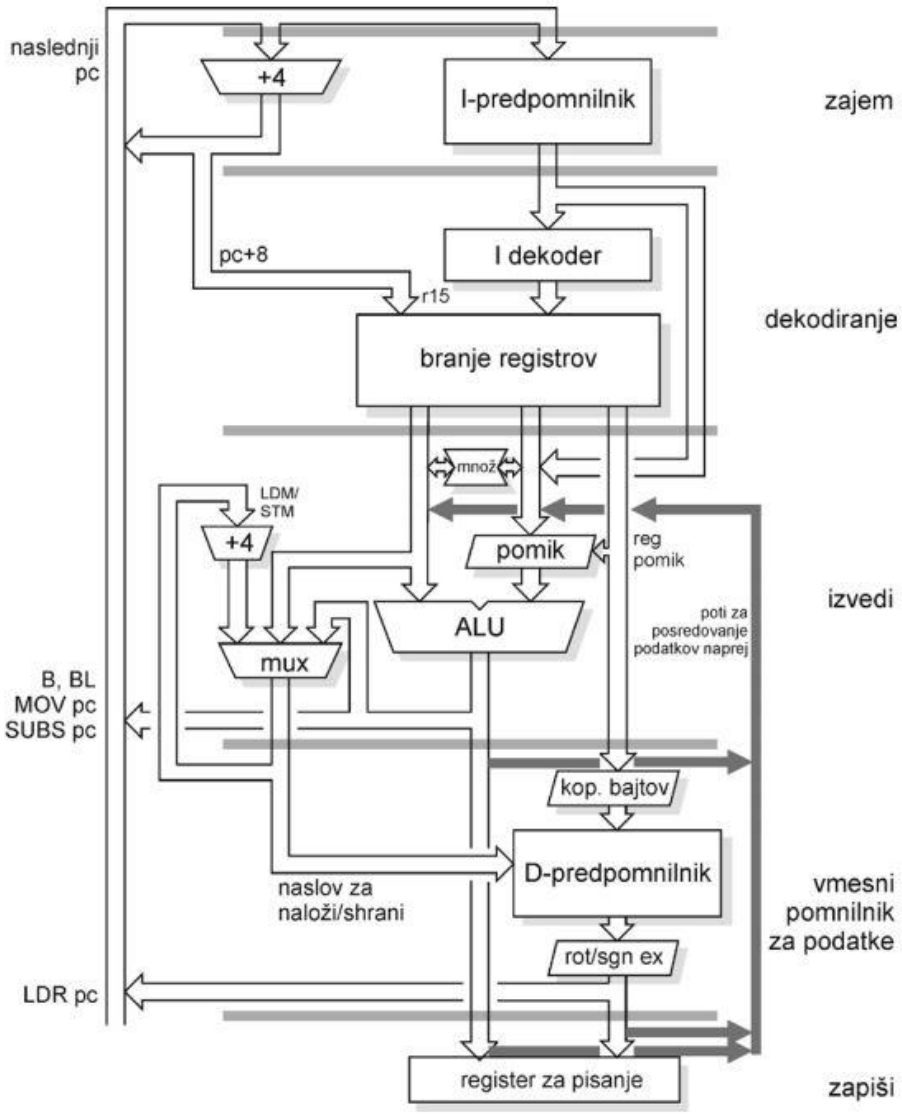
Na sliki 5 je predstavljeno osnovno ARM-jedro s 3-stopenjskim cevovodom. Preko registra vhodnih podatkov dobimo podatke, ki jih dekodiramo ter postavimo na vodilo B. Vrednost, ki je na tem vodilu, se nato v pomikalni enoti po potrebi premakne, nato pa se v aritmetični logični enoti združi z vrednostjo na vodilu A. Rezultat operacije v aritmetični logični enoti je nato zapisan v nek končni register. V tem trenutku je vrednost programskega števca v naslovnem registru, nato se premakne v inkrementer, ki povečano vrednost zapiše v register programskega števca. Morebitne izhodne podatke nato po vodilu B pripeljemo do registra izhodnih podatkov.



Slika 5: Shema preprostega 3 stopenjskega cevovoda [14].

4.9.2 5-stopenjski cevovod

Z uporabo ločenih pomnilnikov za ukaze ter za podatke smo v četrti generaciji arhitekture ARM pridobili možnost istočasnega nalaganja novih ukazov ter dostopa do pomnilnika. S tem tudi odpravimo prej omenjeno težavo zastajanja cevovoda zaradi sočasnega dostopa do pomnilnika. Pri 5-stopenjskem cevovodu je stopnja *izvedi* razdeljena na tri dele, in sicer v prvem delu se izvedejo aritmetične operacije, v drugem delu se izvaja dostop do pomnilnika in v tretjem delu se rezultati zapišejo nazaj v registre. Kadar izvajamo ukaze za obdelavo podatkov, je drugi del v mirovanju. Pri 5-stopenjskem cevovodu pa se pojavi nova težava, saj moramo zaustaviti delovanje cevovoda, če imamo medsebojno odvisne podatke v različnih stopnjah cevovoda [20]. Do tega pride, kadar nek ukaz potrebuje rezultat prejšnjega, nezaključenega ukaza.



Slika 6: Organizacija 5-stopenjskega cevovoda na procesorju ARM9TDMI [17].

Način, kako rešiti problem medsebojne odvisnosti podatkov, brez da bi pri tem povzročali zastoje v cevovodu, je uvedba poti za posredovanje podatkov naprej, kar je tudi vidno na sliki 6. Te poti omogočajo odpošiljanje rezultatov takoj, ko so rezultati razpoložljivi. Kot je tudi vidno na sliki 6, 5-stopenjski ARM-cevovod omogoča dostop do razpoložljivih rezultatov na treh različnih mestih.

4.9.3 6-stopenjski cevovod

Pri 6-stopenjskem cevovodu se stopnja *dekodiraj* razcepi na dve stopnji, in sicer *preberi register* in *dekodiraj*. V stopnji *preberi register* pripravimo register, ki ga bomo uporabljali, medtem ko se funkcija stopnje *dekodiraj* ne spremeni. Pri jedru ARM10 smo dobili tudi širša, 64-bitna podatkovna ter ukazna vodila, kar omogoča prenos dveh 32-bitnih registrov na enkrat. S to generacijo pridobimo tudi enoto za napovedovanje vejitev (angl. Branch prediction), ki poizkuša ugotoviti izid vejitve. Če izid pravilno napove, program teče nemoteno naprej, v nasprotnem primeru pa je potrebna popolna izpraznitev cevovoda [21, 20]. V tej generaciji imamo tudi nov seštevalnik za ukaze pomnoži in akumuliraj (angl. multiply accumulate). Nov seštevalnik je dodan v podatkovno stopnjo cevovoda, ker ukazi za množenje za svoje izvajanje ne potrebujejo dostopa do pomnilnika.

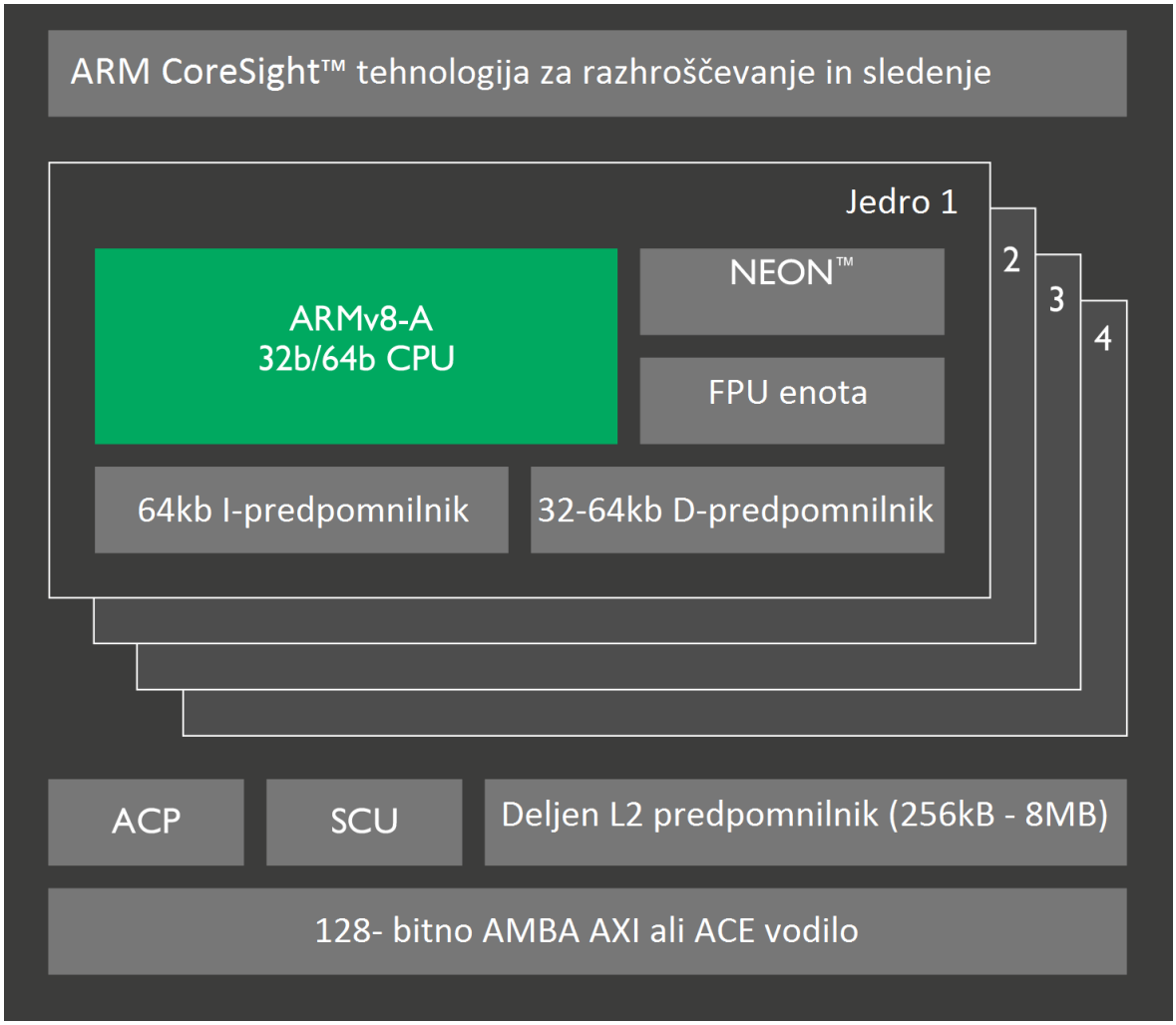
4.9.4 8-stopenjski cevovod

Pri naslednji generaciji jeder ARM je premikalna operacija (angl. shift operation) prestavljena v ločeno stopnjo cevovoda. Prav tako ločimo ukazne in podatkovne dostope do pomnilnika na dve ločeni stopnji cevovoda.
Pri tej generaciji tudi pridobimo možnost izvajanja ukazov izven reda (angl. out-of-order execution), saj je izvajanje ukazov razdeljeno na tri različne cevovode, ki se lahko izvajajo sočasno.

5 ARM SISTEM NA ČIPU

5.1 Sistem na čipu (SoC)

Za delovanje osebnega računalnika potrebujemo veliko različnih čipov, od centralnega procesorja (CPU), do grafičnega procesorja (GPU) in notranjega pomnilnika (RAM) ter še bi lahko naštevali. Vsaka od teh komponent zavzame veliko prostora ter porabi veliko energije, kar pa sta dve lastnosti, ki ne govorita v prid mobilnih ter ostalih energijsko varčnih naprav. Sistem na čipu (angl. System-on-chip, SoC) je integrirano vezje, ki združuje vse računalniške komponente v en čip. Poleg procesorja na vezju najdemo grafični pospeševalnik, razne vrste pomnilnika, mrežno podporo, GPS, DSP ter ostale komponente. Taka sestava čipa omogoča fizično manjšo velikost ter manjšo porabo energije, kar je ključnega pomena za prenosne naprave. Sistemi na čipu pa se ne uporabljajo samo v mobilnih napravah, saj že velika večina procesorjev za osebne računalnike vsebuje na istem čipu kot je procesor tudi grafični pospeševalnik.



Slika 7: Procesor ARM Cortex-A73.

Na sliki 7 je blokovno predstavljen najnovejši ter najzmogljivejši procesor Cortex-A73. To je 4-jedrni procesor iz najnovejše verzije arhitekture ARMv8. Vsako jedro ima ločena predpomnilnika za ukaze ter za podatke. Večji L2-predpomnilnik pa je deljen med vsa štiri jedra. Za skladnost predpomnilnika skrbita enoti ACP ter SCU, kar zagotavlja, da se podatki v nekem bloku v predpomnilniku ne razlikujejo od podatkov v istem bloku v glavnem pomnilniku. Vse komponente so povezane s 128-bitnim AMBA AXI ali ACE vodilom. Ti dve vodili sta opisani v naslednjem podpoglavju.

5.2 Vodilo AMBA

Vodilo AMBA (angl. Advanced Microcontroller Bus Architecture) je standard za vodila na čipu. Prva AMBA-vodila so bila napredna sistemska vodila na čipu ASB (angl. Advanced System Bus) ter napredna vodila za povezovanje z zunanjimi napravami APB (angl. Advanced Peripheral Bus) [19]. Kasneje so v verziji AMBA 2 dodali visoko zmogljivo vodilo AHB (angl. Advance High Performace Bus), ki se največ uporablja v družini Cortex-M [19]. Za doseganje še višjih zmogljivosti pri medsebojnem povezovanju naprav so v tretji generaciji AMBA 3 [19] predstavili vodilo AXI (angl. Advanced eXtensible Interface). Prav tako s tretjo generacijo dobimo vodilo ATB (angl. Advanced Trace Bus), ki se uporablja pri odkrivanju napak. Vodilo ACE (angl. AXI Coherency Extension), ki je bilo predstavljeno v četrti generaciji AMBA 4 [19], omogoča dostop do pomnilnika več procesorjem, ki delujejo na istem sistemu. To vodilo predstavlja pomemben člen v big.LITTLE tehnologiji. Ta tehnologija združuje manj zmogljiva jedra z zelo zmogljivi jedri v en večjedrni sistem.

5.3 Tipi pomnilnikov

Moderni večjedrni ARM-procesorji podpirajo neurejeno izvajanje ukazov, kar pomeni, da lahko začnemo z izvajanjem novega ukaza tudi, če se prejšnji ukaz še ni zaključil. Pomembna pridobitev je tudi predpomnilnik, v katerega lahko začasno shranimo podatke ali ukaze. Na arhitekturi ARM delimo pomnilnik na tri različne tipe, ki imajo svoja pravila za dostop. Način dostopa do različnih blokov v pomnilniku se določi v enoti za zaščito pomnilnika (angl. Memory Protection Unit).

5.3.1 Normalen pomnilnik

V normalnem pomnilniku hranimo podatke ter ukaze. Bralno-pisalni pomnilnik RAM ter bralni pomnilnik ROM sta definirana kot normalna pomnilnika. Normalen pomnilnik dovoljuje špekulativne dostope do dodatnih lokacij v pomnilniku, ne dovoljuje pa špekulativnih pisanj v pomnilnik. Pri špekulativnem dostopu do pomnilnika uporabljamo

dinamično predvidevanje skokov. Ker pa predvidevanje ni vedno pravilno, imenujemo tak dostop špekulativen. Večkratne dostope do pomnilnika lahko jedro združi v manj dostopov z večjo velikostjo podatkov. Tako lahko pisanje štirih bajtov združimo v en 32-bitni dostop do pomnilnika. Do normalnega pomnilnika lahko dostopamo tudi z neporavnanimi dostopi. Nekatere dele normalnega pomnilnika lahko uporabimo za predpomnilnik, kar lahko dosežemo samo z enoto za upravljanje s pomnilnikom (angl. Memory Management Unit) [6].

5.3.2 Pomnilnik naprave ter strogo urejen pomnilnik

Pomnilnik naprave ter strogo urejen pomnilnik sta enako urejena ter imata enaka pravila za dostopanje. Uporabljena sta za bloke pomnilnika, ki se uporabljajo za pomnilniško preslikavo zunanjih naprav (angl. memory mapped peripherals) [6]. Dostopi do teh dveh pomnilnikov so atomarni in ne morejo biti prekinjeni. Nedovoljeni so špekulativni dostopi, predpomnjenje ter neporavnani dostopi. Program mora točno določiti, koliko dostopov do pomnilnika bo potreboval. Dostopov jedro ne more združevati kot v primeru normalnega pomnilnika. Razlika med pomnilnikom naprave ter strogo urejenim pomnilnikom je ta, da lahko pomnilnik naprave omogoča uporabo pisalnega medpomnilnika.

5.4 Predpomnilnik

Predpomnilnik je majhen, hiter pomnilnik med jedrom in glavnim pomnilnikom. V njem shranjujemo kopije podatkov iz glavnega pomnilnika. Do podatkov v predpomnilniku lahko dostopamo precej hitreje kot do podatkov v pomnilniku, kar je tudi glavna prednost predpomnilnika. Ko jedro bere iz nekega naslova, najprej pogleda, če so podatki na tem naslovu že v predpomnilniku. Nato namesto počasnejšega dostopa do glavnega pomnilnika raje uporabi podatke, ki so že shranjeni v predpomnilniku. Posledično to prinese tudi nižjo porabo energije. Pri klasični Von-Neumannovi arhitekturi je en predpomnilnik za podatke in ukaze. Pri harvardski arhitekturi pa imamo ločena predpomnilnika za ukaze (I-predpomnilnik) ter za podatke (D-predpomnilnik).

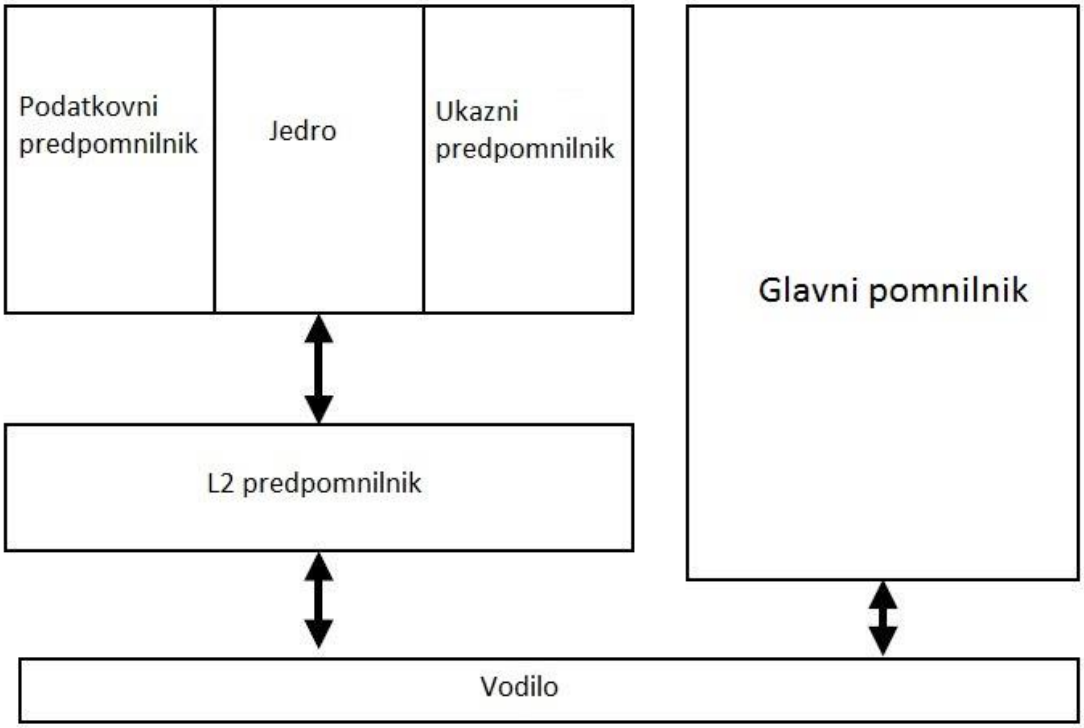
Novejši ARM-procesorji imajo predpomnilnik v dveh ali več nivojih. Navadno imamo na vsakem jedru na nivoju L1 predpomnilnika za ukaze in podatke ter na nivoju L2 večji, enoten predpomnilnik. Če imamo gručo procesorjev ARM, podpira zunanji L3 nivo predpomnilnika, ki je deljen med vse procesorje [7].

Prvi ukaz za prenos podatkov iz glavnega pomnilnika v predpomnilnik ni nič hitrejši, kot če bi iz glavnega pomnilnika naložili podatke neposredno v registre. Hitrejši so dostopi do

podatkov, ki so že v predpomnilniku, kar pomeni, da je v predpomnilnik smiselno nalagati podatke, ki jih bomo večkrat uporabili.

Za upravljanje s predpomnilniki imamo v ARM-procesorjih del strojne opreme, ki se imenuje krmilnik predpomnilnika. Njegova naloga je, da avtomatično prenaša podatke iz glavnega pomnilnika v predpomnilnik. Krmilnik predpomnilnika zbudimo z zahtevami za branje ali pisanje iz jedra. Ko pride do zahteve, krmilnik najprej preveri, ali je zahteva na tem naslovu že v predpomnilniku. Če je podatek v predpomnilniku, se izvedejo ukazi s podatki iz predpomnilnika. V nasprotnem primeru krmilnik preveri, če je zahtevani podatek v L2-predpomnilniku. Če dobi podatek v L2-predpomnilniku, ga nato prenese v L1-predpomnilnik ter naprej do jedra. Ta proces se izvaja transparentno in ni neposredno viden programskemu razvijalcu.

Ena od slabosti predpomnilnika je, da čas izvajanja programa lahko postane ne deterministične. Do tega pride zato, ker je predpomnilnik zelo majhen del pomnilnika, ki se hitro napolni. Če se predpomnilnik napolni, moramo obstoječe ukaze oziroma podatke najprej izbrisati, da naredimo prostor za nove. Tako se lahko čas izvajanja programa spreminja.



Slika 8: Organizacija pomnilnika na ARM-sistemu na čipu [6].

5.5 Pomnilniška hierarhija

Običajna hierarhija pomnilnika v vseh računalniških sistemih je, da so hitri in manjši pomnilniki bližje jedru ter počasnejši in večji pomnilniki oddaljeni od jedra. V vgrajenih sistemih se za to uporabljata dva pojma, in sicer pomnilnik na čipu (angl. on-chip memory) in pomnilnik zunaj čipa (angl. off-chip memory). Pomnilnik, ki je na istem čipu kot jedro, je hitrejši od pomnilnika, ki je zunaj čipa. Predpomnilnik je lahko vključen na različne nivoje hierarhije. V ARM-procesorjih je L1-predpomnilnik, ki je povezan neposredno z jedrom in med drugim skrbi za pridobivanje ukazov. Najpogostejša velikost L1-predpomnilnika je 16 KB ali 32 KB. L2-predpomnilnik vsebuje tako ukaze kot tudi podatke in je večji od L1-predpomnilnika. Fizično je lahko implementiran kot del jedra ali pa kot zunanji del pomnilnika. Na sliki 8 vidimo, da sta podatkovni ter ukazni predpomnilnik blizu jedru. Med jedrom ter glavnim pomnilnikom je večji L2-predpomnilnik, ki je preko vodila povezan z glavnim pomnilnikom.

5.6 Urejenost pomnilnika

V starejših implementacijah arhitekture ARM ukaze izvajamo po urejenem vrstnem redu (angl. in-order execution). To pomeni, da se ukazi izvedejo eden za drugim ter da se izvedejo točno tolikokrat in v točno takem vrstnem redu, kot je program predvidel. V novejših implementacijah arhitekture ARM je vrstni red izvajanja ukazov določen na način, kako imamo urejen pomnilnik. Neurejeno izvajanje ukazov (angl. out-of-order execution) pomeni, da lahko začnemo z izvajanjem novega ukaza tudi, če se prejšnji ukaz še ni zaključil. Edini pogoj je, da podatki, do katerih dostopamo, niso medsebojno odvisni. Operacija, ki porabi največ časa med izvajanjem ukaza, je prav dostop do pomnilnika. Ta čas lahko zmanjšamo že z vpeljavo predpomnilnika ali medpomnilnika, kamor si pripravimo podatke. Drugi način za pospešitev časa izvajanja pa je prej omenjeno neurejeno izvajanje ukazov. Tako se vrstni red, v katerem se ukazi naloži ali shrani izvedejo, lahko razlikuje od vrstnega reda dostopov do pomnilnika, ki ga vidijo zunanje naprave.

5.7 Koprocesorji

Arhitektura ARM podpira uporabo koprocesorjev za razširitev funkcionalnosti ARM-procesorja. V prvih verzijah arhitekture ARM smo imeli ločen, zunanji koprocesorski vmesnik, ki je omogočal povezavo od enega do šestnajst koprocesorjev. Novejša jedra, iz profila Cortex-A, pa podpirajo samo notranje koprocesorje. Kot primer lahko vzamemo koprocesor CP15 [6], ki je uporabljen za nadzorovanje sistema. V tem koprocesorju lahko uporabljamo do šestnajst 32-bitnih registrov. Dostop do koprocesorja CP15 je privilegiran,

kar pomeni, da v uporabniškem načinu ne moremo dostopati do vseh registrov [6]. Za upravljanje s koprocesorji so na voljo posebni ukazi, ki so opisani v poglavju 4.7.

5.8 Tehnologija big.LITTLE

Naprave, kot so mobilni telefoni, imajo zahteve po visoki zmogljivosti ter nizki porabi energije. To pa sta si nasprotujoča pojma, navadno na enem procesorju ne moremo doseči zelo visokih zmogljivosti ter energijske varčnosti. Zelo zmogljiva jedra porabljajo za delovanje manj zahtevnih aplikacij preveč energije.

ARM-ova tehnologija big.LITTLE združuje manj zmogljiva jedra *LITTLE* z zelo zmogljivi jedri *big* v en večjedrni sistem. Tako dobimo energijsko varčen ter obenem zmogljiv sistem. S tehnologijo big.LITTLE dobimo heterogen sistem, ki deluje na istem operacijskem sistemu. Programska oprema nato določa, katero jedro bo aplikacija uporabljala glede na zahtevnost. Zelo zahtevne aplikacije lahko izkoristijo moč obeh jeder naenkrat.

Obe jedri uporabljata isti nabor ukazov, tako lahko aplikacije poganjamo na obeh jedrih brez sprememb. Na nivoju L3 si obe jedri tudi delita predpomnilnik, zato lahko med njima prenašamo podatke.

5.9 Procesor za obdelavo digitalnih signalov

Procesiranje digitalnih signalov zahteva visoko propustnost pomnilnika ter hitro izvedbo operacij množenje s seštevanjem (angl. multiply accumulate). Za procesiranje teh signalov je navadno uporabljen procesor za obdelavo digitalnih signalov (angl. digital signal processor), ki je bil v prvih verzijah arhitekture ARM ločen od glavnega procesorja. Od pete generacije arhitekture ARM dalje imamo podporo za obdelavo digitalnih signalov v jedru glavnega procesorja. To prinaša nižjo porabo energije ter tudi nižjo ceno razvoja samega sistema.

Osnovna DSP-operacija je množenje dveh 16-bitnih podatkov in seštevanje v 32-bitni register. V tretji generaciji arhitekture ARM smo to operacijo lahko izvedli v približno 12 urinih ciklih, medtem ko se ista operacija na procesorjih od pete generacije dalje lahko izvede v enem urinem ciklu.

Zaradi potrebe po visoki prepustnosti pomnilnika ter zaradi hitrosti izvedbe algoritmov so ti navadno napisani v zbirnem jeziku. Za doseganje najvišjih zmogljivosti je potrebno

premišljeno razporejanje ukazov ter dober nadzor nad zasedenostjo registrov, kar najlažje dosežemo prav z zbirnim jezikom.

Z naslednjimi posodobitvami arhitekture ARM dobimo podporo za naprednejše ukaze za obdelovanje multimedijskih podatkov, to sta tehnologiji SIMD in NEON, ki sta opisani v naslednjem poglavju.

5.10 SIMD in NEON

Nekateri multimedijski programi ter grafični pospeševalniki delujejo na veliko podatkih, krajših od 32 bitov. Aplikacije za obdelavo zvoka pogosto upravljajo s 16-bitnimi podatki, aplikacije za grafično procesiranje pa z 8-bitnimi podatki. Če izvajamo take aplikacije na 32-bitnem mikroprocesorju, del računskih komponent ostane neuporabljen, vendar vseeno porablja energijo. Za bolj racionalno porabo razpoložljivih sredstev SIMD-tehnologija uporablja en ukaz na več podatkih, ki so enakega tipa ter enake velikosti. Na primer namesto enega 32-bitnega ukaza izvedemo vzporedno štiri operacije nad 8 bitnimi podatki. NEON-tehnologija je ARM-ovo tržno ime za napredne SIMD-ukaze. Fizično je to ena izmed komponent procesorja, ki izvaja SIMD-ukaze. Podpira 128-bitne vektorske ukaze, ki uporabljajo 32 registrov dolžine 64 bitov ali pa 16 registrov dolžine 128 bitov. V registre shranjujemo vektorje elementov, ki so iste dolžine. Tako lahko 64-bitni NEON-vektor vsebuje osem 8-bitnih, štiri 16-bitne, dva 32-bitna ter en 64-bitni element. Operacije nad temi elementi se nato izvajajo vzporedno. Ukazi NEON podpirajo samo ukaze za obdelovanje podatkov ter ukaze tipa naloži/shrani.

5.11 TrustZone

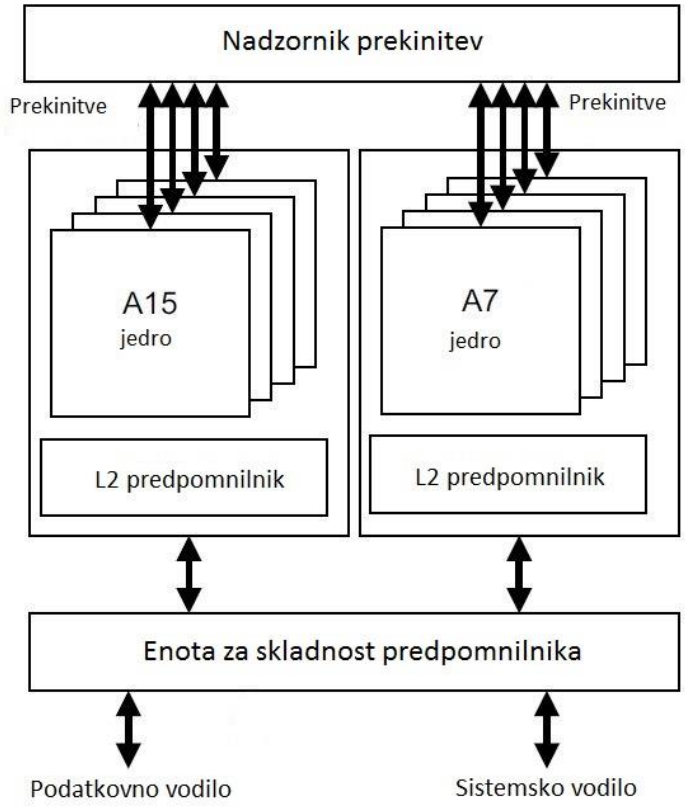
TrustZone je strojna varnostna razširitev v ARM-procesorjih. Varnost sistema dosežemo tako, da razdelimo strojno in programsko opremo celotnega sistema na dva dela, in sicer na normalen ter na varen del. V sistemih s TrustZone tehnologijo nam vodilo AMBA3 AXI [8] zagotavlja, da iz normalnega dela sistema ne moremo dostopati do podatkov, ki so v varnem delu sistema. V posebnih pogojih pa lahko iz varnega dela sistema dostopamo do podatkov, shranjenih v normalnem delu sistema. Navadno se uporabniški operacijski sistem izvaja v normalnem delu sistema, medtem ko se v varnem delu sistema izvaja lasten varen operacijski sistem. To omogoča, da imamo v varnem delu sistema shranjene občutljive podatke, kot so na primer gesla. Za prehod med obema deloma sistema služi poseben ukaz SMC (angl. Secure Monitor Call).

Programski del tehnologije TrustZone je tesno povezan s strojnim delom. Programski del nas tako obvaruje pred programskimi napadi na našo napravo.

5.12 Večjedrni procesorji

En ARM-procesor lahko vključuje do štiri procesorska jedra. Večjedrni sistemi tako omogočajo boljše zmogljivosti, saj imamo na voljo več računske moči, porazdeljene med več jeder. Zmanjša se tudi čas izvajanja programa, saj lahko naloge izvajamo vzporedno. Več jeder omogoča hitrejše izvajanje procesov, zato so lahko nekatere enote procesorja izključene za dalj časa.

Prednost večjedrnih procesorjev je v nižji porabi energije. Če ne potrebujemo procesorske moči vseh jeder, lahko posamezna jedra tudi izključimo. Večjedrni procesorji za doseganje istega pretoka podatkov delujejo z nižjimi frekvencami kot enojedrni procesorji, kar posledično prinese nižjo porabo energije [6].



Slika 9: big.LITTLE sistem, ki vsebuje dva 4-jedrna procesorja [6].

Večjedrni sistemi se lahko istočasno odzovejo na prekinitve, kar privede do bolj odzivnega sistema, saj se posamezno jedro odziva na manj prekinitev. Večjedrnemu procesorju rečemo tudi gruča (angl. cluster) [6]. Sistemom, ki se poslužujejo big.LITTLE tehnologije, pravimo tudi sistemi z več gručami. Primer takega sistema je viden na sliki 9. Na tej sliki vidimo big.LITTLE sistem z dvema večjedrnima procesorjema. Procesor, ki uporablja štiri Cortex-A15 jedra, ima vlogo *big*. Poleg njega je procesor, ki uporablja štiri jedra Cortex-

A7 in ima vlogo *LITTLE*. Iz slike je vidno, da sta oba procesorja povezana z istim nadzornikom prekinitev, kar omogoča, da se oba procesorja odzoveta na iste prekinitve. Vsak procesor ima svoj L2-predpomnilnik, ki je deljen med vsa štiri jedra. Enota za skladnost predpomnilnika zagotavlja, da se podatki v nekem bloku v predpomnilniku ne razlikujejo od podatkov v istem bloku v glavnem pomnilniku.

6 ARHITEKTURA ARMV8

Leta 2013 je ARM predstavil 64-bitno arhitekturo ARMv8 [7]. Osma verzija arhitekture ARM podpira izvajanje tako 32-bitnih ukazov kot tudi 64-bitnih. Novost so tudi registri dolžine 64-bitov. Kljub vsem spremembam in novostim je arhitektura ARMv8 združljiva z arhitekturo ARMv7 in prejšnjimi verzijami. Izvajanje ukazov v 64-bitnem načinu imenujemo AArch64, v 32-bitnem načinu pa AArch32, ki pa je kljub novemu imenu skoraj identično kot pri arhitekturi ARMv7.

S prehodom na 64-bitno arhitekturo lahko uporabljamo pomnilnik, večji od 4 GB. V novem naboru ukazov A64 imamo 31 64-bitnih registrov, ki so vedno dostopni, ne glede na način izvajanja. Z uporabo večjih registrov dosežemo izboljšanje zmogljivosti, saj lahko obdelujemo 64-bitne podatke v enem samem ukazu, medtem ko bi na 32-bitnem procesorju potrebovali več ukazov. Posledično imamo zaradi večjih registrov tudi manj dostopov do sklada. Pri arhitekturi ARMv8 imamo samo štiri nivoje izjem, medtem ko jih imamo pri arhitekturi ARMv7 sedem. V naslednjih podpoglavjih si bomo vse spremembe podrobneje pogledali.

6.1 Izjeme ter nadzor prekinitev

Izjeme pri arhitekturi ARMv8 delimo na štiri nivoje z različno prioriteto. Označujemo jih z EL*n*, kjer je *n* število, ki označuje prioritetni nivo. Višje kot je število, večjo prioriteto ima. Različni nivoji izjem omogočajo logično ločevanje izvajanja programa v vseh procesorskih načinih delovanja. Na arhitekturi ARMv8 so naslednji nivoji izjem:

- EL0, uporabniške aplikacije,
- EL1, jedro operacijskega sistema,
- EL2, hipernadzornik (angl. Hypervisor) in
- EL3, mikrokoda in ostale nizkonivojske aplikacije.

Navadno se izjema v izvajanju nekega programa pojavi v samo enem izmed nivojev. Izjema je hipernadzornik jedra, ki ob izjemi zasede nivoja EL1 in EL2.

Arhitektura ARMv8 ima podporo za virtualizacijo, kar omogoča poganjanje več gostujočih operacijskih sistemov. Tako se vsak gostujoči operacijski sistem izvaja v navideznem stroju.

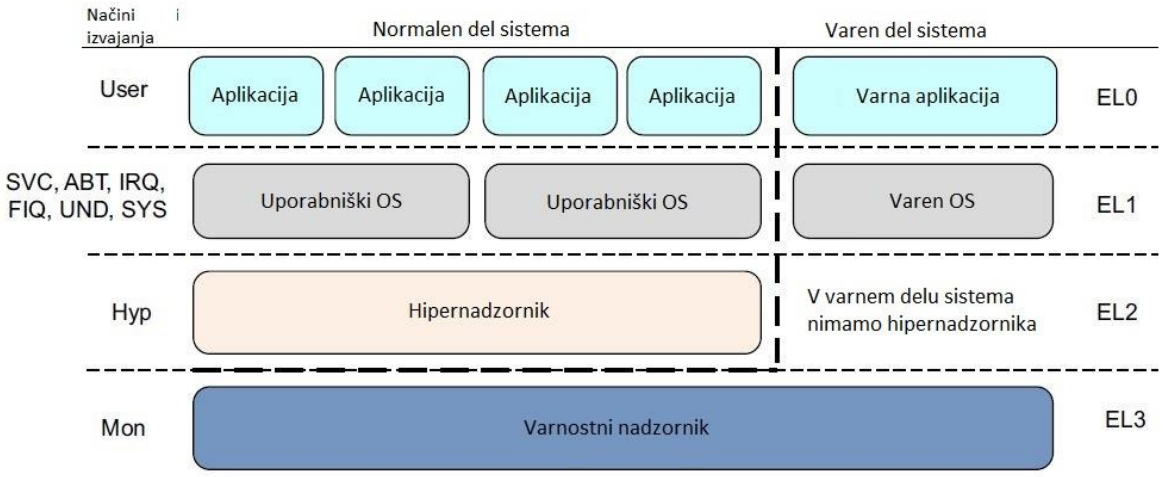
Nadzor prekinitev v AArch64 načinu izvajanja deluje podobno kot v prejšnjih verzijah arhitekture, omeniti pa je treba sinhrone in nesinhrone prekinitve. Sinhrona prekinitev se sproži med izvajanjem ukazov, kjer s povratnim naslovom, ki je shranjen v povezovalnem

registru, ugotovimo, kateri ukaz je sprožil prekinitev. Nesinhrone prekinitve se sprožijo nesinhrono z ukazom, ki je prekinitev povzročil. Tako procesor ne ve točno, kateri ukaz je sprožil prekinitev.

6.2 Načini izvajanja

Pri arhitekturi ARMv8 imamo dva načina izvajanja, in sicer AArch64, ki za svoje izvajanje uporablja 64-bitne registre, ter AArch32, ki za svoje izvajanje uporablja 32-bitne registre. Pri AArch32-načinu izvajanja obdržimo privilegirane načine iz prejšnjih arhitektur, pri novem načinu izvajanja AArch64 pa so privilegirani načini določeni s prioritetami izjem, ki so opisane v prejšnjem podpoglavju. Kadar smo v AArch64-načinu, uporabljano A64-nabor ukazov, kadar smo pa v AArch32-načinu, uporabljamo nabora ukazov ARM ali Thumb, ki sta predstavljena v prejšnjih poglavjih.

Če uporabljamo 64-bitni operacijski sistem, lahko poganjamo 32-bitne aplikacije na EL0-nivoju. Da to dosežemo, moramo spremeniti način izvajanja na AArch32, ko pa se 32-bitna aplikacija izvede, preklopimo nazaj na AArch64-način izvajanja. Na istem nivoju izjem lahko poganjamo samo aplikacije, ki se izvajajo v istem načinu izvajanja. Preklop na AArch32 zahteva prehod iz višjega nivoja izjem na nižji, preklop na AArch64 pa iz nižjega nivoja izjem na višjega. Za primer vzemimo izvajanje 64-bitnega operacijski sistem, ki želi poganjati 32-bitno aplikacijo. Trenutno smo v AArch64-načinu delovanja ter na EL1-nivoju izjem. Na sliki 10 vidimo, da izvajanje operacijskega sistema spada na EL1-nivo izjem. Za poganjanje 32-bitne aplikacije moramo zamenjati nivo izjem na EL0, saj je ta nivo namenjen izvajanju aplikacij, ter zamenjati na AArch32-način izvajanja. Prehod nazaj na AArch64-način izvajanja ter na višji nivo izjeme se zgodi s klicem ukaza ERET. Če uporabljamo 32-bitni operacijski sistem, ne moremo poganjati 64-bitnih aplikacij.

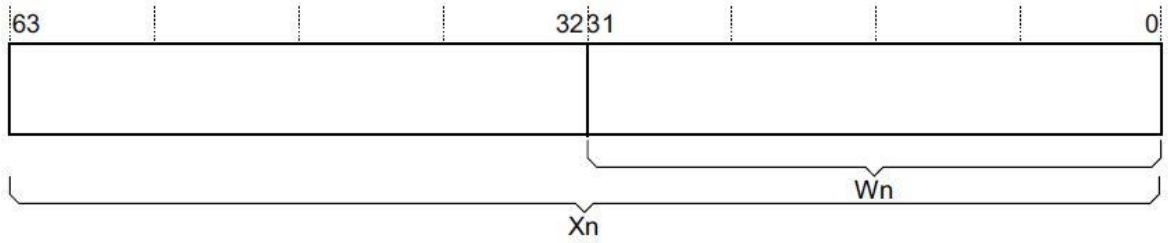


Slika 10: Načini izvajanja ter nivoji izjem na arhitekturi ARMv8 [7].

6.3 Registri

Način izvajanja AArch64 za delovanje uporablja 31 splošno namenskih registrov dolžine 64-bitov. Ti registri so vidni v vseh nivojih izjem, kar pomeni, da ne uporabljamo skritih registrov kot pri prejšnjih verzijah arhitekture. Registri v AArch64 so označeni z X0 do X30. Vsak od teh registrov ima svojo 32-bitno obliko, ki zajemajo spodnjo polovico 64-bitnga registra in so označeni z W0 do W30. Sestava registrov v arhitekturi ARMv8 je grafično predstavljena na sliki 11. Pri pisanju v W-registre se 32 bitov iz zgornje polovice 64-bitnega registra nastavi na 0. Pri branju W-registrov preberemo spodnjo polovico 64-bitnega registra.

Poleg splošno namenskih registrov imamo tudi pet posebnih registrov, to so: ničelni register, programski števec, kazalec na sklad, statusni register ter povezovalni register. Statusni ter povezovalni register imata enako vlogo kot pri prejšnjih verzijah arhitekture.



Slika 11: Sestava registra v arhitekturi ARMv8 [7].

6.3.1 Ničelni register

Pri branju ničelnega registra vedno preberemo nič, pisanja v ničelni register pa so prezrta. Uporabljamo ga lahko pri večini ukazov. Vrednost nič je pri pisanju ukazov zelo pogosta, zato je zelo priročno, če jo lahko imamo vedno na voljo v enem izmed registrov.

6.3.2 Kazalec na sklad

Na arhitekturi ARMv8 ima vsak nivo izjeme svoj kazalec na sklad. Ti se imenujejo SP_EL0, SP_EL1, SP_EL2 in SP_EL3 Pri izvajanju na nivoju EL0 lahko dostopamo samo do kazalca na sklad SP_EL0. V ostalih nivojih izjem lahko dostopamo do vseh štirih predstavljenih kazalcev na sklad. Večina ukazov ne more spreminjati kazalca na sklad. Ta register spreminja vsaka operacija, ki uporablja sklad. Lahko pa ga spreminjamo tudi z nekaterimi aritmetičnimi ukazi, kot je na primer ukaz ADD, ki lahko bere in piše iz kazalca na sklad [7]. Če smo v AArch32-načinu izvajanja, se ta register imenuje WSP.

6.3.3 Programski števec

Pri prejšnjih verzijah arhitekture ARM je bil programski števec splošno dostopen register. Pri arhitekturi ARMv8 pa je neposreden dostop do programskega števca onemogočen. Za dostop do programskega števca dobimo nove ukaze, kot je na primer relativno naslavljanje programskega števca. Pri tem načinu naslavljanja dejanski naslov ukaza določa trenutna vsebina programskega števca ter odmik ukaza.

6.6 Nabori ukazov

Največja sprememba v arhitekturi ARMv8 je nov nabor ukazov, imenovan A64. To omogoča dostope do 64-bitnih registrov ter obdelavo 64-bitnih podatkov, dolžina ukazov pa je še vedno 32 bitov. Nabor ukazov A64 lahko uporabljamo samo v AArch64-načinu izvajanja. Arhitektura ARMv8 prav tako omogoča nabora ukazov ARM ter Thumb, ki pa se v tej verziji arhitekture imenujeta A32 ter T32. Oba nabora ukazov lahko uporabljamo samo v AArch32-načinu izvajanja. Oba 32-bitna nabora ukazov v tej verziji arhitekture pridobita nekaj novih ukazov, ki pa niso združljivi s prejšnjimi verzijami. Med naboroma ukazov A64 ter A32 ne moremo preklapljati, saj je nov nabor ukazov A64 ločen in različen od ARM-nabora ukazov iz prejšnjih verzij arhitekture ARM. Med naboroma A32 ter T32 pa lahko v AArch32-načinu izvajanja preklapljamo tako, kot v prejšnjih verzijah arhitekture.

Večina ukazov v A32-naboru ukazov se lahko pogojno izvaja. Kaj je pogojno izvajanje ukazov, je opisano v poglavju 4.6. Pri naboru ukazov A64 pa je ukazov, ki se lahko pogojno izvajajo, bistveno manj.

7 PRIHODNOST ARHITEKTURE ARM

Uporaba ARM procesorjev se z razvojem pametnih prenosnih naprav vsako leto veča. Pri izdelavi procesorjev, za pametne telefone ter tablice, ki temeljijo na arhitekturi ARM prevladujejo podjetja Qualcomm, Apple ter Samsung. Trenutno najzmogljivejši procesor na arhitekturi ARM je procesor Apple A9X, ki se lahko v nekaterih primerih kosa z Intelovo družino izdelkov Core M. Procesorji iz družine Core M temeljijo na arhitekturi x86 in so namenjeni manjšim prenosnikom ter ostalim prenosnim napravam. Ti procesorji vsebujejo tranzistorje velikosti 14nm, kar prinaša manjšo velikost procesorja in zmogljivejše ter energijsko varčne procesorje. Z zmanjševanjem velikosti tranzistorjev, lahko na enako fizično velikost procesorja vgradimo več tranzistorjev, kar pomeni višjo zmogljivost. Z zmanjševanjem dolžine med tranzistorji pa dobimo energijsko varčnejše procesorje, saj se za prenos signalov med tranzistorji porabi manj energije. Tako lahko vidimo, da se zmogljivost procesorjev, ki temeljijo na arhitekturi ARM približuje manj zmogljivim procesorjem, ki temeljijo na arhitekturi x86. Vsekakor pa v bližnji prihodnosti ne moremo pričakovati osebnih računalnikov s procesorji, ki temeljijo na arhitekturi ARM, saj še vedno po zmogljivostih niso kos najzmogljivejšim procesorjem, ki se trenutno uporabljajo v osebnih računalnikih. Druga ovira pa je programska oprema, saj trenutno operacijski sistem Windows ne podpira ARM procesorjev.

Podjetje ARM vseeno nima ambicij samo na trgu prenosnih naprav, temveč tudi v strežniških sistemih ter ostali mrežni infrastrukturi. Na strežniških sistemih trenutno prevladujejo procesorji na arhitekturi x86, ki sicer omogoča velike zmogljivosti, njena slabost pa je predvsem velika poraba energije. Trenutno ima ARM na strežniških sistemih samo 0,3% delež. Do leta 2020 pa se napoveduje, da bo delež strežniških sistemov z ARM procesorji zrastel do 9,7% [10]. Uporaba v ARM procesorjev v strežniških sistemih je zaradi nižje porabe energije smiselna, saj ti sistemi navadno delujejo vse dni v letu. Eno izmed večjimi podjetji, ki si želijo prehoda na ARM procesorje v strežniških sistemih je Google.

Internet stvari (angl. internet of things) je trenutno ena izmed najbolj razvijajočih vej v tehnologiji. Ideja interneta stvari je, da imamo vse naprave v nekem območju povezane med seboj. Tako lahko imamo v enoten sistem povezane hladilnike, pečice, pametne televizije, pametne žarnice, termostate in ostale naprave v stanovanju. Za te naprave so idealni ARM procesorji, saj za delovanje potrebujejo majhne, poceni ter energijsko varčne procesorje. Internet stvari pa se lahko v prihodnosti razširi na večja območja, kar pomeni, da lahko v prihodnosti pričakujemo pametna, povezana mesta.

ARM procesorji igrajo tudi veliko vlogo v vse hitrejšem razvoju avtonomnih avtomobilov. Že v današnjih avtomobilih se procesorji ARM uporabljajo na primer v sistemu proti blokiranju koles ABS. Avtonomna vozila za zaznavanje okolice uporabljajo veliko število različnih senzorjev ter veliko število procesorjev za procesiranje vseh podatkov. V avtonomnih avtomobilih so uporabljeni predvsem procesorji iz družine Cortex-R, saj je ključna dostava in obdelava podatkov v realnem času. Pri delovanju celotnega sistema avtonomnega vozila ne smemo imeti velikih zakasnitev.

Glede na trenuten razvoj tehnologije lahko z gotovostjo rečemo, da so priložnosti za uporabo procesorjev ARM neskončne.

8 ZAKLJUČEK

ARM-procesorji že od začetka predstavljajo standard za uporabo v vgrajenih sistemih. V današnjih dneh že redko srečamo vgrajene sisteme, ki ne vsebujejo ARM-procesorja. Cilj te naloge je bil arhitekturo ARM čim bolj približati ljudem, čeprav so opisovane zelo tehnične stvari. Predvsem s poglavjem o naboru ukazov smo želeli približati programiranje v zbirnem jeziku, saj se premalo zavedamo, da je zbirni jezik še vedno osnova za višje nivojsko programiranje. Moderni operacijski sistemi omogočajo pisanje uporabniških aplikacij brez natančnega zavedanja, kako sistem deluje na najnižjem nivoju. Slednje je za hitro pisanje aplikacij zelo dobro, vendar sem mnenja, da dobra aplikacija ne more nastati brez zavedanja, kaj se dogaja v ozadju.

Glavne prednosti ARM procesorjev so energijska učinkovitost, majhna velikost ter vsestranskost. Ker se različni vgrajeni sistemi pojavljajo praktično vsepovsod je prihodnost za ARM procesorje vsekakor svetla.

9 LITERATURA IN VIRI

[1] ARMed solutions to the DSP war, https://www.cs.umd.edu/class/fall2001/cmsc411/proj01/arm/dsp.html.

[2] A. Krishnaswamy in R. Gupta, Profile Guided Selection of ARM and Thumb Instructions, 2002.

[3] A. Sloss, D. Symes in C. Wright, *ARM System Developer's Guide: Designing and Optimizing System Software*. Morgan Kaufmann Publishers Inc., San Francisco, 2004.

[4] ARM Keil, ARM Registers, http://www.keil.com/support/man/docs/armasm/armasm_dom1359731128950.htm.

[5] ARM, ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition, http://liris.cnrs.fr/~mmrissa/lib/exe/fetch.php?media=armv7-a-r-manual.pdf, 2007.

[6] ARM, ARM Cortex-A Series Programmers Guide, https://silver.arm.com/download/Software/BX100-DA-98001-r0p0-01rel3/DEN0013D_cortex_a_series_PG.pdf, 2011

[7] ARM, ARM Cortex-A Series, Programmers Guide for ARMv8-A, https://static.docs.arm.com/den0024/a/DEN0024.pdf, 2015.

[8] ARM, ARM Security Technology Building a Secure System using TrustZone Technology, https://www.sbs.ox.ac.uk/cybersecurity-capacity/system/files/ARM%20Security%20Technology.pdf.

[9] ARM, The ARM University Program, ARM Architecture Fundamentals, https://www.youtube.com/watch?v=7LqPJGnBPMM, 2013.

[10] ARM, ARM Holdings plc Q1 2016 Roadshow Slides, http://phx.corporate-ir.net/External.File?item=UGFyZW50SUQ9MzMyNDU3fENoaWxkSUQ9LTF8VHlwZT0z&t=1&cb=635961396194872632, 2016.

[11] J. Andrews, *Co-verification of hardware and software for ARM SoC design*. Elsevier, Burlington, 2005.

[12] J. Lemieux, Introduction to ARM Thumb, http://www.embedded.com/electronics-blogs/beginner-s-corner/4024632/Introduction-to-ARM-thumb, 2003.

[13] J. V. Vijay in B. Bansode, ARM Processor Architecture, Evolution and Applications, *International Journal of Science, Engineering and Technology Research*, 2015.

[14] L. Ryzhyk, The ARM Arhitecture, 2006.

[15] M. Esponda in R. Rojas, The RISC Concept - A survey of Implementations, 1991.

[16] M. Levy, The History of The ARM Architecture: From Inception to IPO, http://reds.heig-vd.ch/share/cours/reco/documents/thehistoryofthearmarchitecture.pdf.

[17] M. Munih, ARM, http://robo.fe.uni-lj.si/~marko/ur/poglavje_4_17-10-13.pdf, 2006.

[18] P. Knaggs in S. Welsh, ARM: Assembly language programming, http://www.cypress.com/file/55766/download, 2004.

[19] R. Koti in D. Meshram, An Overview of Advance Microcontroller Bus Architecture Relate on APB Bridge, 2013.

[20] S. Chattopadhyay, *Embedded System Design*, PHI Learning Pvt. Ltd., 2010.

[21] S. Furber, *ARM System-on-chip Architecture*, druga izdaja. Addison-Wesley Longman Publishing Co., Boston, 2000.