



ARM uvod

Primjer razvoja sustava

- 1965. godine Intelov suosnivač Gordon Moore predvidio je razvoj sustava na čipu rekavši da će se **broj tranzistora na čipu udvostručiti svake približno dvije godine (18 mjeseci)**. Ovo predviđanje pokazalo se vrlo dobrim i poznato je pod nazivom **Mooreov zakon**.
- Mooreov zakon na djelu možemo lako raspoznati samo ako pogledamo kakvo računalo smo po određenoj cijeni mogli kupiti npr. prije godinu dana u odnosu na danas...
- U našem primjeru možemo pogledati taj fenomen i s malo drugačije strane ...
- Jedna od najzahtjevnijih aplikacija na računalu svakako je dekompresija i prikazivanje video sadržaja
- Kvaliteta reprodukcije (npr kašnjenje, ...) razlikuje se od uređaja do uređaja..

Primjer razvoja sustava

- Danas filmove određene rezolucije i kvalitete možemo gledati i na mnogim pametnim telefonima
- Međutim brzina procesora nije jedini važan faktor u sustavu, već se mora gledati i propusna moć memorijskog sustava, tip aplikacije i slično...



Primjer razvoja sustava

- Arhitektura većine današnjih mobilnih telefona zasnovana je na procesoru ARM!!!
- Vidi npr:
 - <http://armdevices.net/>
 - <http://www.arm.com/markets/>



ARM najvažnija tržišta

- Embedded

- Automotive Infotainment
- Embedded Computing
- General-Purpose MCUs
- Smart Cards
- Smart Meter

- Mobile

- Computing
- Smartphones
- Feature Phones
- Connectivity and Modem
- Mobile Payments

- Home

- Blu-ray and DVD
- Computing
- Digital Set-top Box
- Digital Still Cameras
- Digital TV
- Gaming

- Enterprise

- HDD/SSD
- Flash Cards and UFD
- Home Networking

Zahtjevi industrije i tržišta

- Procesorska snaga
- Cijena
- Niska potrošnja
- Sigurnost/Pouzdanost
- Prilagodljivost
- Modularnost
- Široka primijenjenost
- ...

Nažalost mnogi od ovih zahtjeva su dijametralno suprotni i nemoguće ih je sve zadovoljiti pa tražimo optimalno rješenje za pojedinu primjenu

Primjer...

- Životni ciklus nekih proizvoda u današnje vrijeme je izuzetno kratak... (posebice potrošačka elektronika)
- Novi proizvod se mora
 - Projektirati
 - Proizvesti prototip
 - Ispitati
 - Pripremiti proizvodnju
 - Pripremiti tržište
 - Proizvesti
 - Distribuirati
- Ponekad za samo 3-6 mjeseci !!!

Stavljanje novog proizvoda na tržište

- U tih 3-6 mjeseci nemoguće je **uvijek iznova** projektirati uređaj
- Takav postupak bio bi
 - Skup
 - Spor
- U današnje vrijeme teži se sve većem “ponovnom” korištenju intelektualnog vlasništva (IP, Intellectual Property) što se obično naziva “reusability”
- Ideja je da dijelove sustava (sklopovske i programske) napravimo tako da su modularni kako bi ih mogli koristiti u što više proizvoda i u što više budućih inačica

Stavljanje novog proizvoda na tržište

- Dokazano je da je jedan od najvažnijih faktora uspjeha što raniji izlazak na tržište sa kvalitetnim proizvodom
 - To se može postići isključivo minimizacijom potrebe za redizajniranjem (tj. pokušava se iskoristiti što više postojećeg)
 - Nove funkcionalnosti koje su potrebne možda se mogu KUPITI na tržištu (ako se procjeni da je to isplativije nego razvijati ih unutar kuće)
- Upravo na tom načelu se zasniva uspjeh procesora ARM na tržištu

Stvarni svijet - predavanja: Što ih povezuje

- 4/8/16/32/64-bitni CPU ?
 - Neki procesori su ZANIMLJIVI (npr. nova Intel Haswell mikroarhitektura)
 - Neki procesori su JEDNOSTAVNI (npr. Microchip PIC MCU)
 - Neki procesori se KORISTE
- Naš izbor je ARM (jer se KORISTI)
 - Ali "ARM" u stvari NIJE PROCESOR u onom obliku kako se obično zamišlja procesor već predstavlja ideju pristupa tržištu
 - Približno 70% tržišta 32-bitnih procesora !!!!!!!
- Objasnimo što zapravo ARM radi na tržištu

Kratka povijest ARM-a

- Prva verzija procesora ARM razvijena je u *Acorn Computers Limited*, Cambridge, Engleska, između 1983. i 1985. godine.
- U to vrijeme dominaciju na tržištu imali su 8-bitni mikroprocesori sa relativno kompleksnim skupom instrukcija (CISC - Complex Instruction Set Computers).
- Samo nešto ranije, početkom 80-tih, u okviru tri gotovo usporedna istraživačka projekta (IBM 801, Berkeley RISC i Stanford MIPS) razvijena je potpuno nova arhitektura procesora koja se zasniva na jednostavnim instrukcijama koje se mogu izvoditi velikom brzinom (RISC -Reduced Instruction Set Computers).
- RISC procesor razvijen na sveučilištu Berkeley u to vrijeme pokazivao je izuzetne performanse u usporedbi sa komercijalnim procesorima uz znatno jednostavniju (i jeftiniju) sklopovsku izvedbu.

Prvi ARM procesor

- U to vrijeme razviti procesor zahtjevalo je milionske investicije, stotine projektanata i dugo vrijeme razvoja što si je moglo priuštiti samo nekoliko najvećih firmi – a Acorn nije spadao u tu grupu
- Projektanti u Acornu vidjeli su u RISC tipu arhitekture mogućnost da stvore svoj procesor bez potrebe dugogodišnjeg razvoja te velikog broja projektanata i financijskih sredstava
- Tako je nastao prvi procesor ARM, čije ime je bila skraćenica od Acorn RISC Machine.
- Prvi prototip procesora napravila je firma VLSI Technology Inc. i isporučila ga je Acornu u travnju 1985.

Advanced RISC Machines

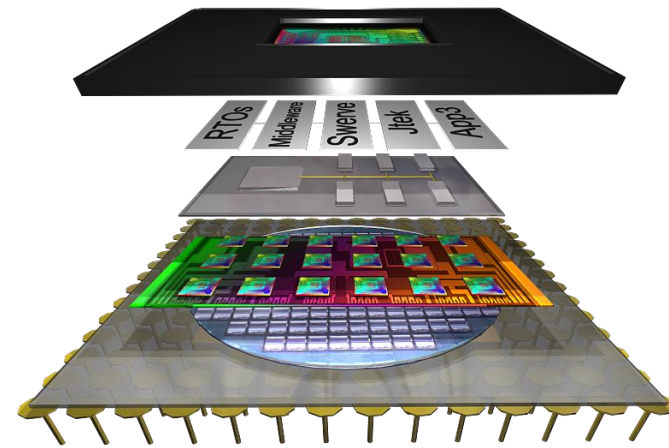
- Kako je ova arhitektura postala izuzetno uspješna na tržištu (zbog relativno niske cijene i dobrih performansi) tako je i 1990. od dijela firme Acorn u kojoj je stvoren prvi procesor ARM formirana nova firma pod nazivom Advanced RISC Machines koja je preuzela projekt širenja tržišta i daljnjeg razvoja ARM arhitekture.
- Od tada i skraćenica imena procesora (ARM) mijenja svoje značenje i postaje Advanced RISC Machine.

ARM danas

- Od tada do danas procesor ARM doživljava brojna poboljšanja i proširenja te su na tržištu prisutne razne generacije i varijante ARM arhitekture.
- Još jedna od specifičnosti i izuzetnih karakteristika firme ARM je u tome što je ona bila prva firma koja svoj cijeli poslovni model zasniva na licenciranju arhitekture svojih procesora bez da ima vlastite poluvodičke tvornice i bez da sama proizvodi procesore.
- Procesore zasnovane na ARM arhitekturi proizvode Intel, Motorola, ST, Philips, ... iako svi oni imaju i svoje vlastite procesore

Što radi ARM

- Projektiranje ARHITEKTURA procesora i sustava
- Licenciranje takvih sustava vodećim svjetskim elektroničkim firmama
- Razvoj i partnerstvo u razvoju ALATA i USLUGA namijenjenih razvoju ARM ARHITEKTURE
- Osnovna podloga su serije efikasnih procesorskih jezgri koje rezultiraju u optimalnim procesorima u pogledu cijene, performansi i ostalih bitnih karakteristika



ARM...

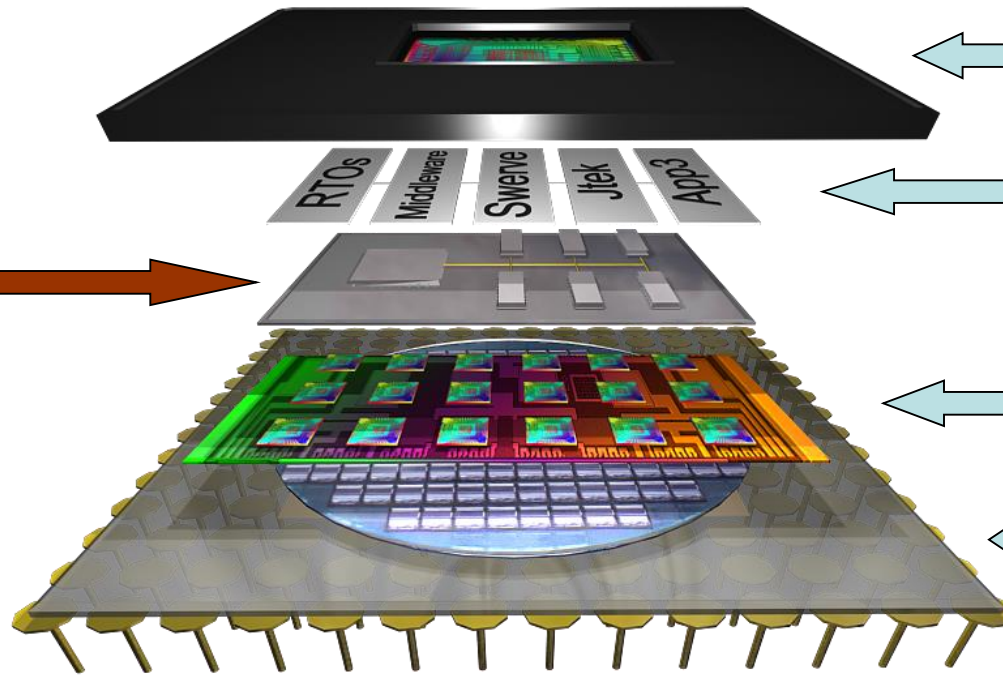
Partneri:

**Integracija
HW i SW**

SW podrška

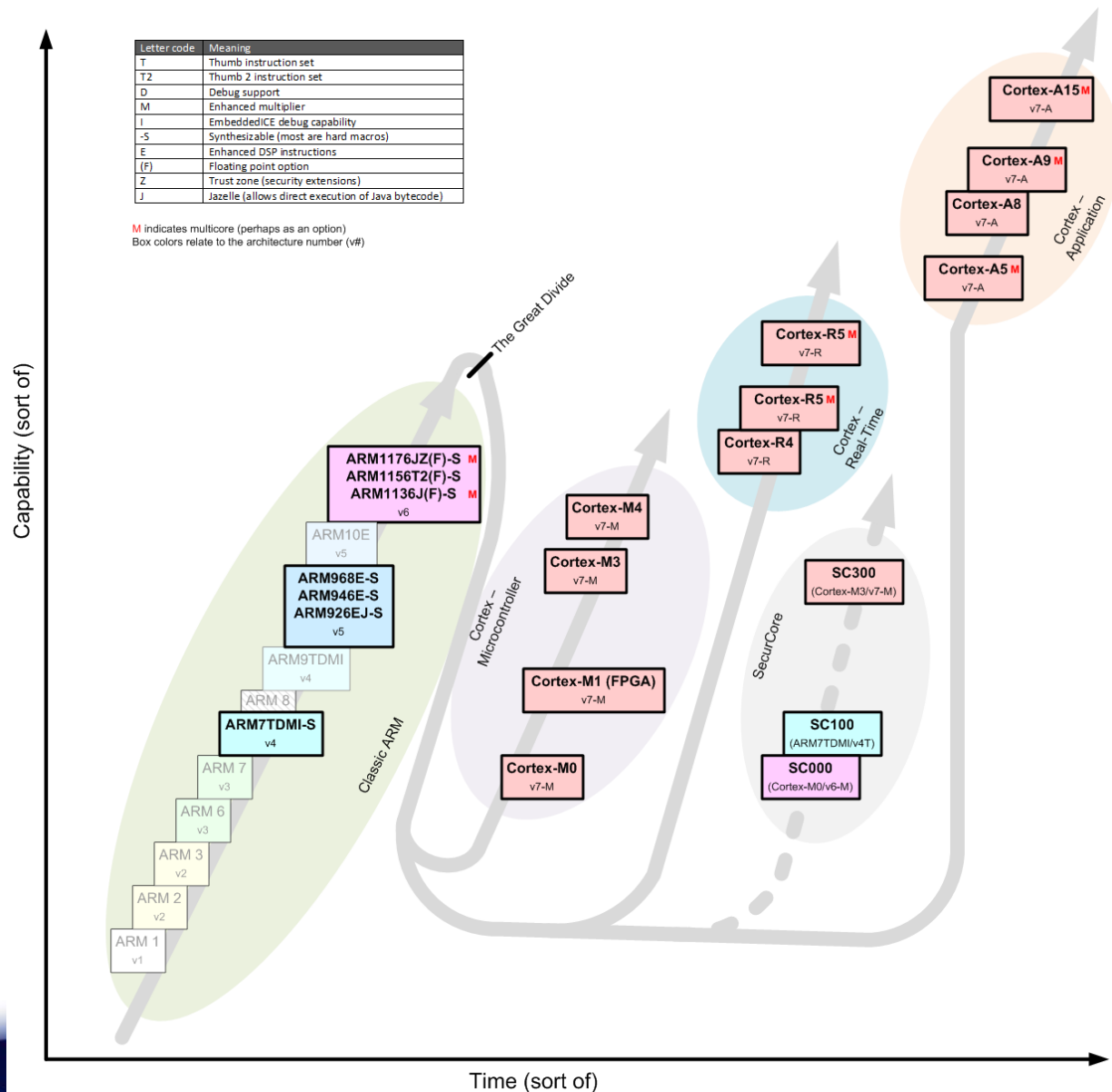
Proizvodnja

**Integracija
u HW sustav**



ARM

ARM processor families

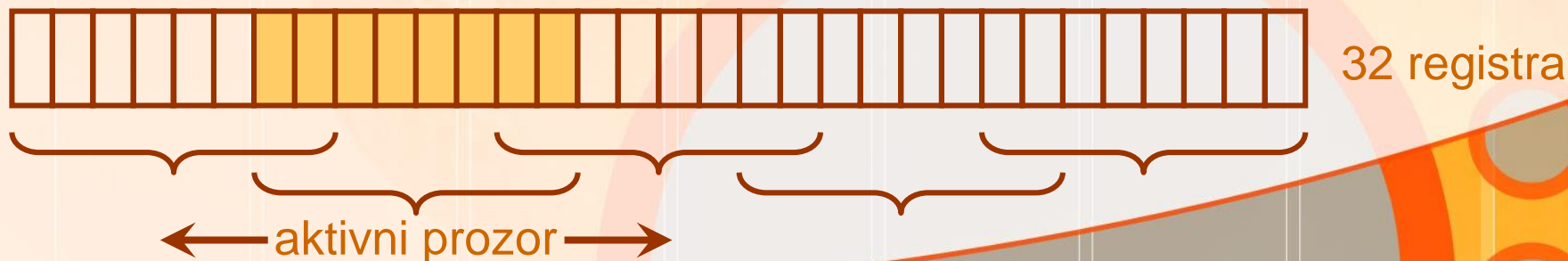


Programski model procesora ARM

Osnovne karakteristike arhitekture procesora ARM

- Vrsta arhitekture
 - Procesor ARM po većini svojih karakteristika spada u grupu RISC procesora, ali ima i neke karakteristike CISC-a.
- Skup registara
 - relativno veliki, uniformni skup registara
 - nema općenitih registarskih prozora*

i * primjer za 5 prozora sa po 8 registara i s preklapanjem od 2 registra:



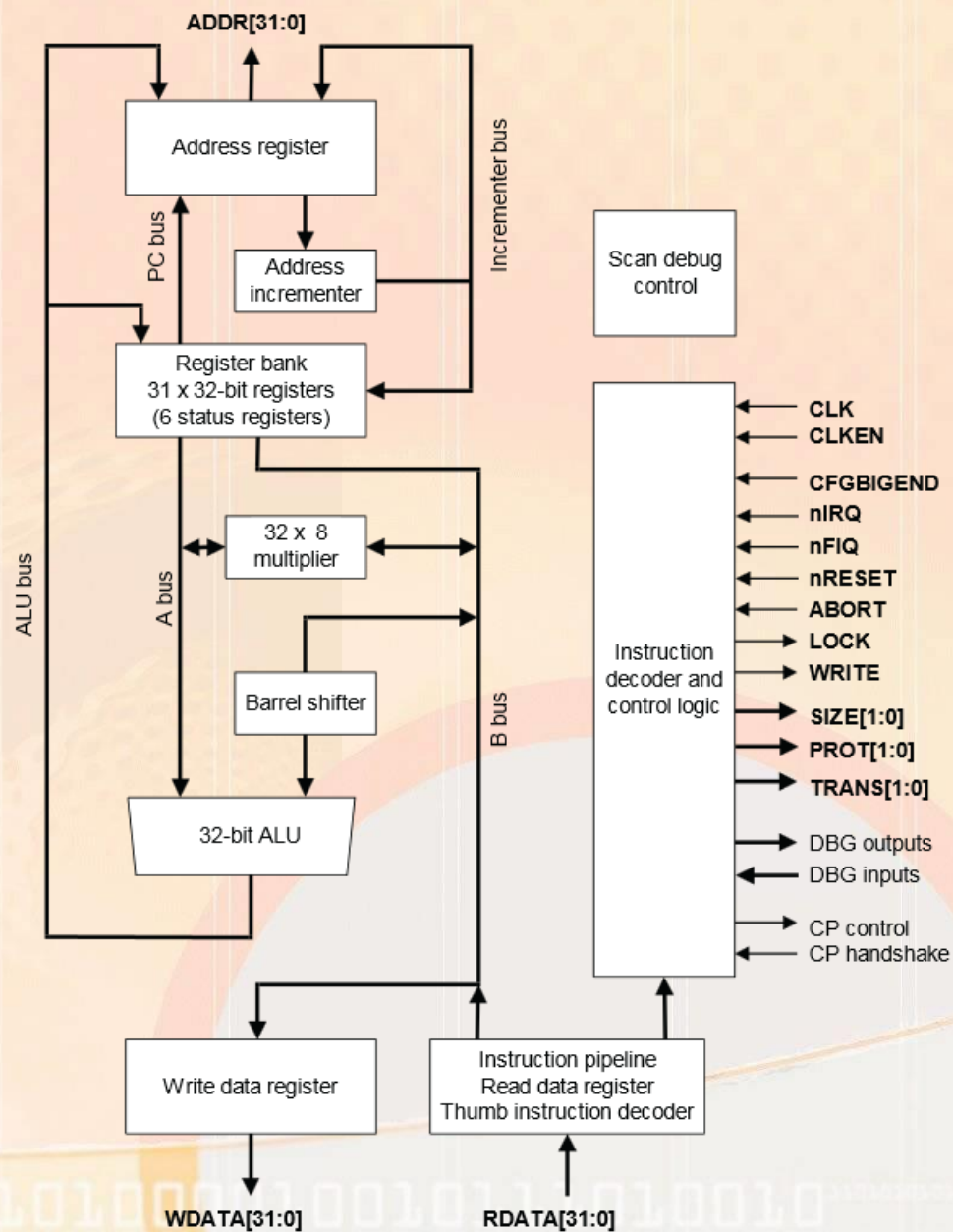
Skup naredaba

- "Load-store" arhitektura
 - Prednosti/nedostaci kao što smo ranije objašnjavali
- Sve naredbe su iste duljine od 32 bita
 - Prednosti/nedostaci kao što smo ranije objašnjavali
- Naredbe za obradu podataka sa 3 adrese (3 operanda)
- Uvjetno izvođenje gotovo svake naredbe
 - Drugi procesori uglavnom nemaju ovu mogućnost
 - Služi za ubrzanje malih odsječaka programskog koda
- Izvođenje općenitog pomaka i aritmetičko-logičke naredbe u jednom vremenskom periodu
 - Barrel shifter na ulazu u ALU
- "Thumb" skup naredaba
 - Za sustave kojima je važna minimizacija memorije

Opće karakteristike

- Mogućnost proširenja skupa naredaba i registara korištenjem koprocesorskih naredaba
 - Neki kodovi nisu iskorišteni te ih možemo sami definirati
- Protočna struktura
 - Protočna struktura od tri (ARM7), pet (ARM9), šest (ARM10) ili osam (ARM11) razina
 - Visoka efikasnost
- Memorijska arhitektura
 - Von Neumannova ili Harvardska arhitektura (različito za pojedine jezgre)
- Modularna arhitektura
 - Namijenjena za SoC (System on Chip)
- Mala potrošnja

ARM 7



Programski model procesora ARM

- Programski model - skup značajki određene arhitekture koji je na raspolaganju programeru:
 - tipovi podataka
 - procesorski načini rada
 - registri
 - registri opće namjene
 - registri programskih stanja
 - iznimke
 - memorija

Tipovi podataka

- Procesor ARM je 32-bitni procesor, ali može obrađivati i podatke manje širine. Programeru su na raspolaganju sljedeći tipovi podataka:
 - bajt (byte, 8 bitova), označava se slovom **B**
 - poluriječ (halfword, 16 bitova), označava se slovom **H**
 - riječ (word, 32 bita), označava se slovom **W**
- Iako ARM može koristiti kraće tipove podataka, sve osnovne operacije obavljaju se nad 32-bitnim riječima.

Procesorski načini rada

- ARM arhitektura podržava sedam procesorskih **načina rada** (engl. modes).
- Postoji jedan korisnički način rada (User) i šest privilegiranih načina rada
- U određenom procesorskom načinu omogućen je ili onemogućen pristup određenim resursima sustava
- Onemogućavanje pristupa nekim resursima u korisničkom načinu je od velike važnosti za izvedbu operacijskih sustava
- Procesorski način se definira postavljanjem najnižih 5 bitova u posebnom registru CPSR što će kasnije biti objašnjeno

Procesorski načini rada

Način	Oznaka	Opis načina
User	usr	Normalno izvođenje programa
System	sys	Podrška za izvođenje privilegiranih zadataka operacijskog sustava
Supervisor	svc	Zaštićen način za operacijski sustav
Abort	abt	Podrška za izvedbu virtualne memorije i zaštitu memorije
Undefined	und	Podrška za programsku emulaciju koprocesora
Interrupt	irq	Podrška za obradu općenitog prekida
Fast Interrupt	fiq	Podrška za brzi prekid



Privilegirani načini rada

"User" način

- Normalan način u kojem se izvode korisnički programi.
- U ovom načinu program ne može koristiti zaštićene resurse sustava te ne može promijeniti način rada.
- Pokušaj pisanja u CPSR[23:0] u User načinu rada procesor ignorira, tako da programi koji se izvode u User načinu ne mogu promijeniti način u neki od privilegiranih

Ostali procesorski načini

- Ostali načini smatraju se privilegiranim i služe pri izvođenju specifičnih operacija. Programi koji se izvode u privilegiranim načinima imaju potpuni pristup svim resursima sustava.
- Načini fiq, irq, svc, abt i und aktiviraju se kad se u sustavu generira iznimka
- Način sys namijenjen je izvođenju privilegiranih sistemskih funkcija, ali bez potrebe korištenja alternativnih registara. Obrada nekih važnijih iznimaka bit će objašnjena kasnije.
- Programi pisani u ATLASU imat će pristup svim resursima sustava.

Registri

- Procesor ARM ima ukupno 37 registara širine **32 bita**. Od ukupnog broja registara, postoji:
 - 31 registar opće namjene (uključujući i programsko brojilo).
 - Važno je uočiti da **u jednom trenutku možemo koristiti samo 16 registara** opće namjene (R0-R15) dok ostali nisu u to vrijeme dostupni !!!
 - 6 registara programskog stanja. Registri programskog stanja su također 32-bitni, no samo neki bitovi se koriste te ostali bitovi ne moraju nužno biti izvedeni u sklopovlju.

Registri

User	System	Superv.	Abort	Undefined	Interrupt	Fast Int.
R0						
R1						
R2						
R3						
R4						
R5						
R6						
R7						
R8						R8_fiq
R9						R9_fiq
R10						R10_fiq
R11						R11_fiq
R12						R12_fiq
R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15=PC						
CPSR						
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Registri

- Registri opće namjene (R0-R15)
- Registri opće namjene, R0-R15 mogu se podijeliti u tri grupe:
 - Jednoznačno definirani registri R0-R7
 - Višeznačno definirani registri R8-R14
 - Programsko brojilo R15
- Registri programskog stanja (CPSR, SPSR)
- **NAPOMENA: u ATLAS-u su svi registri jednoznačni i procesor ima samo jedan način rada - System!!!**

vidi sliku na sljedećem slajdu >>>

Registri

User	System	Superv.	Abort	Undef.	Interrupt	Fast Int.	
R0							Jednoznačno definirani registri R0-R7
R1							
R2							
R3							
R4							
R5							
R6							
R7							
R8						R8_fiq	Višeznačno definirani registri R8-R14
R9						R9_fiq	
R10						R10_fiq	
R11						R11_fiq	
R12						R12_fiq	
R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq	
R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq	
R15=PC							→ Programsko brojilo
							Registri prog. stanja
CPSR							
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq	

Jednoznačno definirani registri R0-R7

- Procesor ARM sadrži osam fizičkih registara nazvanih R0-R7.
- Registri R0-R7 su jednoznačno definirani, te će program pristupati istim fizičkim registrima bez obzira na to u kojem se načinu rada procesor nalazi.
- Ovi registri su, u punom značenju pojma, registri opće namjene jer za njih nije predviđeno nikakvo posebno značenje te se slobodno mogu koristiti u svim mogućim situacijama.
- Prema tome, za sve načine rada, registri R0- R7 su identični.

Višeznačno definirani registri R8-R14

- Za razliku od R0-R7, prilikom pristupa nekom od registara R8-R14 adresirat će se određeni fizički registar ovisno o tome u kojem načinu rada se procesor nalazi.
- Na primjer, ako se procesor nalazi u privilegiranom načinu Supervisor:
 - pri adresiranju registra R13 procesor će pristupiti posebnom fizičkom registru R13_svc koji je dostupan samo u ovom procesorskom stanju. Isto vrijedi i za registar R14.
 - prilikom pristupanja registrima koji nisu višeznačno definirani (R0-R12 i R15 za način Supervisor), uvijek se pristupa zajedničkim fizičkim registrima.

Višeznačno definirani registri R8-R14

- Razlog za korištenje dodatnih fizičkih registara R13 i R14 je u tome što se registri R13 i R14 koriste za posebne namjene.
- Registar R13 se (po dogovoru) koristi kao pokazivač na vrh stoga (SP, Stack Pointer) te je ovime omogućeno da se u svakom načinu rada može definirati neovisan stog.
- Registar R14 se koristi kao registar za pohranjivanje povratne adrese za vraćanje iz potpograma ili iznimke.
- Oba ova registra se mogu koristiti i kao registri opće namjene ukoliko ih sustav ne koristi za ove posebne namjene.

Višeznačno definirani registri R8-R14

- U načinu brzog prekida (fiq) adresira se skup od 7 posebnih fizičkih registara (R8_fiq-R14_fiq).
- To omogućuje da se obrada brzog prekida izvede čim brže - bez potrebe za spremanjem konteksta.

Programsko brojilo R15

- Registar R15 je **programsko brojilo** te se u većini slučajeva korištenje ovog registra za opće namjene ne preporuča, a često i nije dozvoljeno.
- Čitanjem podatka zapisanog u R15 dobije se vrijednost adrese trenutne naredbe + 8 bajtova (zbog protočne strukture).
- Valja uočiti da su pri dohvatu naredbe najniža dva bita uvijek postavljena u logičku nulu zbog toga jer su naredbe ARM-a uvijek poravnate na širinu riječi.
- Upis vrijednosti u R15 izvodi neku vrstu forsiranog skoka, no ovaj način korištenja se ne preporučuje.

Registri programskog stanja (CPSR, SPSR)

- U registru trenutnog programskog stanja CPSR (Current Program Status Register) spremljeni su bitovi koji definiraju različita stanja procesora i programa.
- Registar CPSR je dostupan u svim procesorskim načinima.
- Registar pohranjenog programskog stanja SPSR (Saved Program Status Register) koristi se kad procesor obrađuje iznimke. U njega se pohranjuje vrijednost registra CPSR

Bitovi registara CPSR/SPSR

31	30	29	28	27	7	6	5	4	3	2	1	0
N	Z	C	V	Q		I	F	T	M				

- Bitovi M4-M0: način rada procesora (npr, M[4:0]=10000 znači način "usr").
- Bit T označava da ARM izvodi Thumb naredbu.
- Postavljanje bita I ili F u 1 onemogućit će prekid (I) ili brzi prekid (F).
- Najviših pet bitova u CPSR predstavljaju zastavice:
 - N (Negative) - negativna vrijednost,
 - Z (Zero) - nula,
 - C (Carry) - prijenos,
 - V (oVerflow) - preljev.
 - Q je zastavica koja označava da je došlo do preljeva i-ili zasićenja kod proširenih DSP naredaba (ARM procesori u E izvedbi).
- Ostali bitovi ne koriste se i sačuvani su za buduća proširenja.

Iznimke

- Tijekom rada procesor često treba obraditi neke događaje, poput obrade prekida ili obrade nepostojeće naredbe. U takvim situacijama sustav će generirati **iznimke**.
- Iznimke se općenito obrađuju ovako:
 - procesor pohranjuje povratnu adresu u registar R14
 - procesor pohranjuje trenutno programsko stanje (CPSR) u registar SPSR
 - procesor se prebacuje u potrebno privilegirano stanje
 - procesor započinje s izvođenjem potprograma za obradu iznimke
- Više o iznimkama u posebnom poglavlju...

Zapis podataka u memoriji

- ARM arhitektura koristi jedinstveni memorijski adresni prostor od 2^{32} 8-bitnih podataka
- Podaci u memoriji organizirani su kao:
 - 8-bitni bajtovi (byte)
 - 16-bitne poluriječi (half-word)
 - 32-bitne riječi (word).

Zapis podataka u memoriji

- Pri normalnom pristupu, pretpostavlja se da su podaci poravnati na sljedeći način:
 - riječi (32 bita) su poravnate tako da počinju na adresi djeljivoj s 4 (npr. 0, 4, 8, ...)
 - poluriječi (16 bitova) su poravnate tako da počinju na parnoj adresi (npr. 0, 2, 4, 6, 8, ...)
 - bajtovi (8 bitova) su bilo gdje u memorijskom prostoru

Zapis podataka u memoriji

- Ako se podatak sastoji od nekoliko bajtova, tada je potrebno definirati kojim redoslijedom se ti bajtovi zapisuju u memoriju (endianness).
- U inicijalnom stanju, procesor ARM podržava NV (Niži pa Viši) zapis bajtova tako da se na nižu memorijsku adresu zapisuje bajt manje važnosti (little-endian)*

* Kod procesora ARM moguće je odabrati i da zapisuje bajtove u redoslijedu VN (Viši pa Niži), tj. u redoslijedu big-endian

Pregled skupa naredaba procesora ARM

Skup naredaba procesora ARM - ukratko

- Sve naredbe iz skupa naredaba procesora ARM mogu se podijeliti u šest osnovnih grupa:
 - Naredbe load i store
 - Naredbe za obradu podataka
 - Naredbe grananja
 - Naredbe za prijenos registara stanja
 - Koprosesorske naredbe
 - Naredbe za generiranje iznimke

Naredbe load i store

- Programer ima na raspolaganju naredbe za prijenos podataka iz memorije u registar (load register, LDR) i iz registra u memoriju (store register, STR) u nekoliko različitih verzija.
- Najčešće korištene naredbe obavljaju prijenos samo jednog podatka koji može biti širok 8, 16 ili 32 bita
 - Može se odabrati predznačno proširivanje bitova do pune 32-bitne riječi (pri prijenosu bajta ili poluriječi) ili proširivanje ničicama
 - Postoji velik broj načina adresiranja

Naredbe load/store multiple, swap

- Postoje i naredbe za prijenos bloka podataka gdje se podaci spremaju ili uzimaju iz **više registara** (load/store multiple registers)
 - postoje razne modifikacije koje, na primjer, određuju kako su podaci smješteni u memoriji (povećanje/smanjenje pokazivača na podatak prije/poslije izvođenja naredbe).
 - efikasan način zapisivanja ili čitanja podataka sa stoga.
- U ovu grupu naredbi spadaju i naredbe za zamjenu sadržaja memorije i registara (swap) koje se najčešće koriste pri izvedbi semafora kod operacijskih sustava.

Naredbe za obradu podataka

- Naredbe za obradu podataka obavljaju razne operacije nad podacima koji se nalaze u registrima.
- Naredbe za obradu podataka dijele se na:
 - aritmetičko-logičke naredbe
 - naredbe za množenje i
 - naredba za brojenje vodećih nula.

Aritmetičko-logičke naredbe

- Imaju zajedničku karakteristiku da mogu obraditi do dva operanda i spremiti rezultat u zadani registar te, ako je zadano, osvježiti zastavice stanja.
- Od dva operanda koji se mogu koristiti u operaciji jedan uvijek mora biti registar, a drugi može biti ili neposredna vrijednost ili vrijednost registra nad kojom se još može obaviti određen pomak.
- U okviru aritmetičko-logičkih naredbi postoje i četiri naredbe za usporedbu koje su definirane vrlo slično aritmetičko-logičkim naredbama s razlikom da se rezultat nigdje ne sprema i da se uvijek osvježavaju zastavice stanja.

Naredbe za množenje

- Množenje se uvijek obavlja nad dva 32-bitna podatka.
- Različite naredbe za množenje omogućuju da se spremaju samo 32 bita rezultata u izabrani registar ili da se sprema svih 64 bita rezultata u dva registra.
- U oba slučaja moguće je još omogućiti zbrajanje sa prethodnom vrijednosti rezultatnih registara, čime se ostvaruje tzv. operacija MAC (Multiply-And-Accumulate) koja se vrlo često koristi u algoritmima za obradu signala.

Naredba clz

- Naredba CLZ (Count Leading Zeroes) služi za brojenje vodećih nula u podatku.
- Ovo je specifična naredba koja je veoma korisna kod izvedbe matematičkih algoritama (normiranje brojeva) i algoritama za kompresiju (npr. metode duljine niza).

Naredbe grananja

- Programsko brojilo kod procesora ARM je izvedeno kao običan registar te je grananja moguće izvesti bilo kojom naredbom load ili nekom aritmetičko/logičkom naredbom.
- Pored toga ARM definira i standardne naredbe za grananje poput:
 - Branch (B)
 - Branch and Link (BL).
- Posebnost procesora ARM je što neke izvedbe posjeduju takozvani 'Thumb' skup naredaba te postoje specijalne naredbe grananja BX kojima programer prebacuje procesor iz stanja dekodiranja standardnog skupa naredaba u 'Thumb' skup naredaba i obratno.

Naredbe za pristup registrima stanja

- Posebna grupa naredaba omogućuje prijenos podataka iz registara stanja (CPSR, SPSR) procesora ARM u neki registar opće namjene i obratno.
- Pisanjem u CPSR, na primjer, programer može postaviti stanja zastavica, stanja bitova za omogućavanje prekida kao i procesorska stanja.

Koprocesorske naredbe

- Služe za komunikaciju procesora ARM i koprocesora ako se oni nalaze u sustavu (prijenos podataka, operacije nad podacima, ...)

Naredbe za generiranje iznimke

- Omogućuju programeru i sistemu da se pokrene proces obrade iznimke
- Programski ekvivalent sklopovskim iznimkama koje se postavljaju na priključke procesora

Strojni kodovi ARM-a

- Procesor ARM ima strojne kodove fiksne širine 32 bita (osim za naredbe tzv. "Thumb" skupa)
- Ova fiksna duljina koda naredbe ima svoje prednosti i nedostatke.
 - Prednost je jednostavnost izvođenja i efikasnost protočne strukture za naredbe
 - Nedostatak je ograničenost broja bitova za opis pojedinih dijelova naredbe.
- Formati strojnih kodova dosta ovise o pojedinoj naredbi (vidi primjere na sljedećem slajdu)

Primjeri formata naredaba

Naredba	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Load/Store Multiple	Uvjet				1	0	0	P	U	S	W	L	Rn								Popis registara											
B, BL	Uvjet				1	0	1	L	24-bitni odmak																							
Aritmetičko logička, neposredni pomak	Uvjet				0	0	0	Op kod				S	Rn				Rd				Iznos pomaka				po-mak	0	Rm					

↑
Polje uvjeta

Polje uvjeta (condition field)

- Većina ARM-ovih naredaba **može se izvoditi uvjetno**. Te naredbe u strojnom kodu imaju **polje uvjeta** (condition field) u kojem se zadaje uvjet koji mora biti zadovoljen da bi se naredba izvela.
- Svi uvjeti temelje se na zastavicama stanja u CPSR.
- Ako uvjet nije zadovoljen, umjesto naredbe izvest će se NOP*.
- Uvjetno izvođenje omogućuje programeru izvedbu programa bez korištenja naredbi grananja, čime se može ubrzati izvođenje nekog kratkog dijela programa

* NOP je naredba koja ne izvodi ništa (kratica od No Operation)

Opcode [31:28]	Mnemonički naziv	Puni naziv (engleski)	Stanje zastavica
0000	EQ	Equal	Z
0001	NE	Not equal	!Z
0010	CS/HS	Carry set/unsigned higher or same	C
0011	CC/LO	Carry clear/unsigned lower	!C
0100	MI	Minus/negative	N
0101	PL	Plus/positive or zero	!N
0110	VS	Overflow	V
0111	VC	No overflow	!V
1000	HI	Unsigned higher	C and !Z
1001	LS	Unsigned lower or same	!C or Z
1010	GE	Signed greater than or equal	N == V
1011	LT	Signed less than	N != V
1100	GT	Signed greater than	!Z and N == V
1101	LE	Signed less than or equal	Z or N != V
1110	AL	Always (unconditional)	-
1111	(NV)	See Condition code 0b1111	-

Skup naredaba procesora ARM

Detaljan pregled svih naredbi

Naredbe load i store

- Među najvažnijim naredbama za svaki procesor, pa tako i ARM, su naredbe za prijenos podataka između procesora i memorije (naredbe load/store).
- Naredbe Load čitaju podatak iz memorije i spremaju ga u registar dok naredbe Store spremaju podatak iz registra u memoriju.
- Ovo su jedine naredbe ARM-a kojima se može pristupiti podatku iz memorije !!!

Naredbe load i store

- Procesor ARM posjeduje dva osnovna tipa naredaba load/store:
 - **Prvi tip** naredaba može učitati ili upisati 32-bitnu riječ ili bajt bez predznaka.
 - Takve naredbe imaju općeniti format:

LDR|STR{<cond>}{B} Rd, <addressing_mode>

- gdje pojedine oznake znače:
 - {<cond>} polje uvjeta
 - {B} adresiranje bajta
 - Rd odredišni registar
 - <addressing_mode> opis načina adresiranja

Load / store

- **Drugi tip** može učitati ili upisati 16-bitnu poluriječ bez predznaka, a također se može učitati poluriječ ili bajt te ih predznačno proširiti do širine riječi.
- Ovakve naredbe imaju općeniti format:

LDR|STR{<cond>}H|SH|SB Rd, <addressing_mode>

- gdje pojedine oznake znače:
 - {<cond>} polje uvjeta
 - H|SH|SB adresiranje poluriječi (H), predznačene poluriječi (SH) ili predznačenog bajta (SB)
 - Rd odredišni registar
 - <addressing_mode> opis načina adresiranja

Load / store

Ime naredbe	Engleski naziv
LDR	Load Word
LDRB	Load Unsigned Byte (zero extend)
LDRH	Load Unsigned Halfword (zero extend)
LDRSB	Load Signed Byte (sign extend)
LDRSH	Load Signed Halfword (sign extend)
STR	Store Word
STRB	Store Byte
STRH	Store Halfword

Load/store: Bazni registar

- Za sve naredbe load/store se adresa memorije izračunava korištenjem dva dijela:
 - vrijednost u nekom baznom registru
 - odmak (offset) u odnosu na vrijednost u baznom registru
- Bazni registar može biti bilo koji registar opće namjene*

* Ako se kao bazni registar izabere PC, tada se može postići relativno adresiranje u odnosu na trenutačnu poziciju u programu, te na taj način i izvedba programa koji su potpuno neovisni o položaju u memoriji. Kod direktnog kodiranja (bez pomoći assemblera i korištenja labela) programer mora paziti na vrijednost PC-a u trenutku izvođenja naredbe

Load/store: Odmak (offset)

- Za pojedinu naredbu programer može izabrati jedan od tri formata odmak:
- U najjednostavnijem formatu odmak se definira kao **broj, odnosno neposredna vrijednost** (immediate) koji se izravno upisuje u kôd naredbe. Neposredna vrijednost zapisana je jednim bitom predznaka i iznosom odmak od 12 ili 8 bitova (ovisno o naredbi). 12-bitnim odmakom se može adresirati memorijska lokacija odmaknuta za +/- 4095 mjesta, a 8-bitnim odmaknuta za +/- 255 mjesta u odnosu na vrijednost izabranog baznog registra
- Odmak se može definirati i pomoću **vrijednosti iz registra opće namjene***
- Treća mogućnost je definiranje odmak pomoću **vrijednosti registra opće namjene* koja je još pomaknuta ulijevo ili udesno**

* osim registra PC

Load/store...

- Osim prethodno opisana tri načina za definiciju odmaka, procesor ARM omogućuje još tri različite inačice adresiranja memorije.
- Ove inačice zadaju treba li i kako mijenjati bazni registar tijekom izvođenja naredbe load/store. Ove tri inačice su:
- **Osnovni odmak**
 - Adresa se izračuna zbrajanjem ili oduzimanjem odmaka od baznog registra, a zatim se pristupi memoriji. Vrijednost baznog registra ostaje nepromijenjena:
$$\text{Reg} \Leftrightarrow \text{mem}[\text{BazniReg} + \text{Odmak}]$$

Load /store

- **Predindeksiranje**

- Odmak se zbroji ili oduzme od baznog registra, a izračunata vrijednost se spremi natrag u bazni registar. Zatim se pristupi memoriji.

$\text{BazniReg} = \text{BazniReg} + \text{Odmak}$

$\text{Reg} \Leftrightarrow \text{mem}[\text{BazniReg}]$

- **Postindeksiranje**

- Memoriji se pristupi samo na temelju vrijednosti baznog registra. Tek nakon obavljenog prijenosa, odmak se zbroji ili oduzme od baznog registra, a izračunata vrijednost se spremi natrag u bazni registar.

$\text{Reg} \Leftrightarrow \text{mem}[\text{BazniReg}]$

$\text{BazniReg} = \text{BazniReg} + \text{Odmak}$

Load/store - Rekapitulacija

- Osnovni odmak:

$\text{Reg} \Leftarrow \text{mem}[\text{BazniReg} + \text{Odmak}]$

- Predindeksiranje:

$\text{BazniReg} = \text{BazniReg} + \text{Odmak}$

$\text{Reg} \Leftarrow \text{mem}[\text{BazniReg}]$

- Postindeksiranje:

$\text{Reg} \Leftarrow \text{mem}[\text{BazniReg}]$

$\text{BazniReg} = \text{BazniReg} + \text{Odmak}$

Pregled adresiranja u naredbama load/store

	Osnovni odmak	Predindeksiranje	Postindeksiranje
Neposredni	[R0, #4]	[R0, #4]!	[R0], #4
Registarski	[R7, -R3]	[R7, R3]!	[R7], R3
Registarski s pomakom	[R3, R5, LSL #2]	[R3,R5,LSL #2]!	[R3],R5,LSL #2

Primjer (8-bitno zbrajanje)

Treba napisati program za zbrajanje dvaju 8-bitnih brojeva bez predznaka koji su zapisani u memoriji odmah iza programa. Rezultat treba spremiti u memoriju iza operanada.

;Program za 8 bitno zbrajanje dva broja bez predznaka, ver.1

```
LDRB R0, A           ; prvi operand se učitava u R0 (za
                      ; signed se koristi LDRSB)
LDRB R1, B           ; drugi operand se učitava u R1 (za
                      ; signed se koristi LDRSB)
ADDS R4, R0, R1      ; izvodi se zbrajanje i postavljaju se
                      ; zastavice (S)
STRB R4, REZ         ; rezultat se sprema u memoriju
SWI 123456
```

A	DB %D 100	; prvi operand (oktet)
B	DB 20	; drugi operand (oktet)
REZ	DS 1	; rezultat

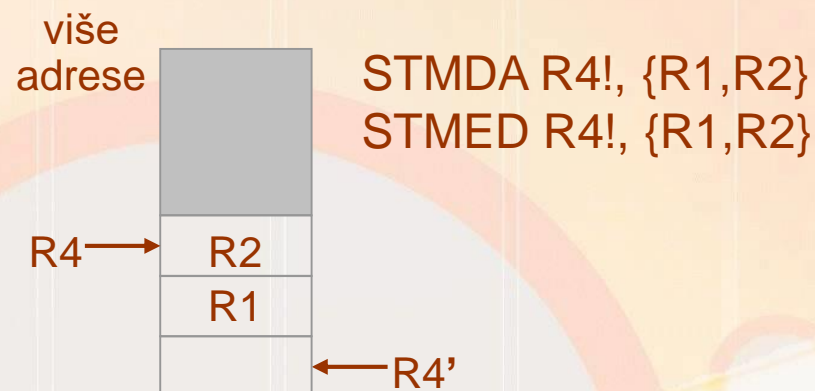
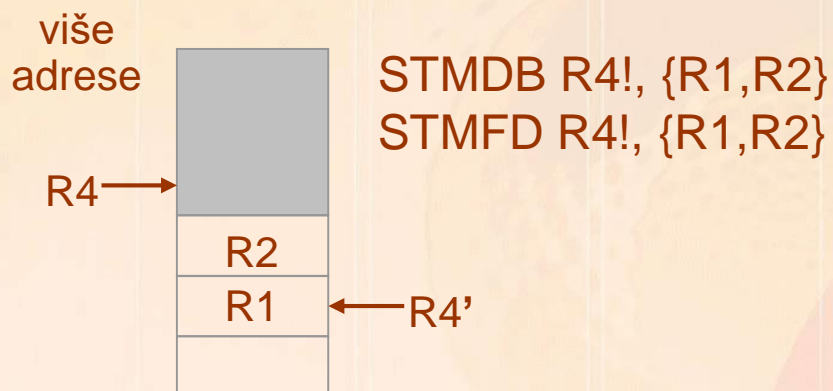
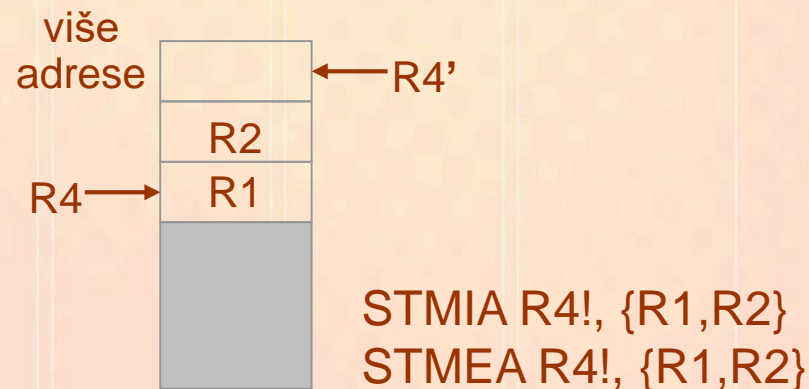
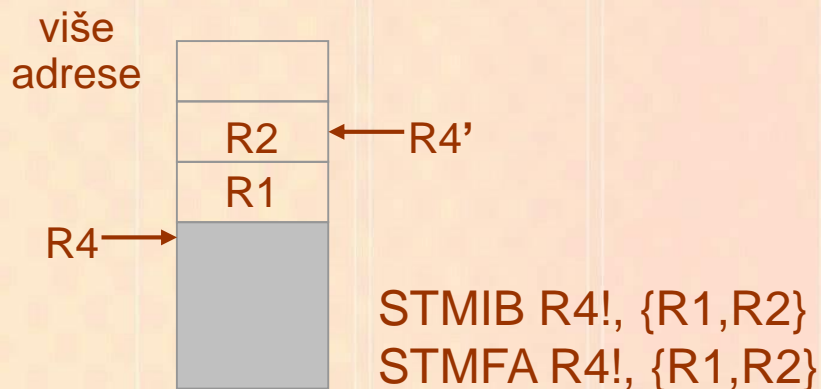
Naredbe Load Multiple i Store Multiple

- Pored osnovnih naredaba load i store, koje obavljaju prijenos podataka samo iz jednog registra, ARM ima dvije naredbe (Load Multiple i Store Multiple) koje programeru omogućuju da jednom naredbom obavi prijenos podataka između memorije i bilo kojeg podskupa registara ili čak svih registara.
- Osnovno ime naredbe je LDM i STM nakon čega se stavlja neki od nastavaka kojima se opisuje kako se sprema niz podataka (tj. niz registara) u memoriju

LDM/STM: načini adresiranja

- Pri pisanju (čitanju) više podataka u memoriju mora se definirati gdje će biti zapisan prvi te svaki sljedeći podatak.
- Da bi definirali gdje će se zapisati prvi podatak, moramo izabrati neki registar opće namjene koji pokazuje na početak memorijskog područja u koje će se upisivati podaci.
- Nakon toga moramo izabrati jednu od četiri kombinacije načina adresiranja niza: IB, IA, DB ili DA. Ove kratice znače:
 - IB - Increment Before (uvećaj prije)
 - IA - Increment After (uvećaj poslije)
 - DB - Decrement Before (smanji prije)
 - DA - Decrement After (smanji poslije)

LDM/STM: načini adresiranja



R4 = stanje prije naredbe

R4' = stanje poslije naredbe

Naredbe s normalnim nastavcima	Ekvivalentne naredbe za rad sa stogom
LDMIA	LDMFD
LDMIB	LDMED
LDMDA	LDMFA
LDMDB	LDMEA
STMIA	STMEA
STMIB	STMFA
STMDA	STMED
STMDB	STMFD

ldm/stm - adresiranje

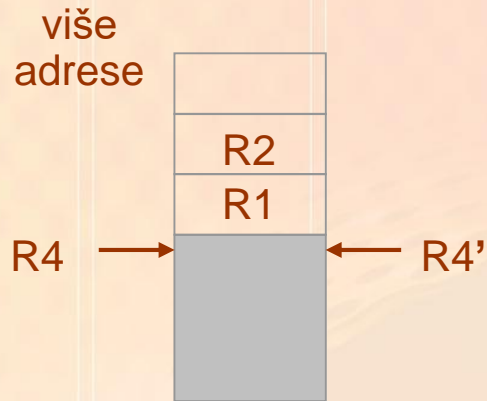
- Još jedna mogućnost koja se pruža programeru je automatsko pomicanje pokazivača, odnosno osvježavanje vrijednosti koja se nalazi u registru koji služi za adresiranje.
- Tako se na primjer naredbe

LDMIB R2, {R4,R8,R9}

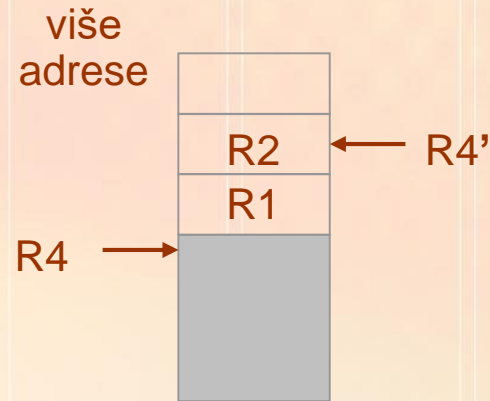
LDMIB R2!, {R4,R8,R9} ; iza R2 piše se uskličnik !

razlikuju po tome što će nakon izvođenja prve naredbe vrijednost registra R2 ostati **nepromijenjena**, dok će nakon druge naredbe registar R2 biti **promijenjen** (uvećan za 12₁₀, jer se R2 uvećava za 4 nakon čitanja podatka za svaki registar iz niza)

Ldm/stm korištenje



STMIB R4, {R1,R2}



STMIB R4!, {R1,R2}

R4 = stanje prije naredbe

R4' = stanje poslije naredbe

Ldm/stm PRAVILA!!!

- Bez obzira na izbor nekog od četiri adresiranja i bez obzira na redoslijed kojim su registri napisani u naredbi, **uvijek je na najnižoj adresi zapisan registar sa najnižim brojem**
- Ako se podaci žele zapisati u memoriju naredbom STM, a zatim pročitati u istom redoslijedu naredbom LDM te ako se koriste nastavci za OBIČNO ADRESIRANJE (ne ekvivalenti za stog!!!), tada način adresiranja u naredbi LDM mora biti **INVERZAN** načinu adresiranja u naredbi STM
- Primjer:
 - Zapisivanje STMIA
 - Čitanje LDMDB

U naredbama se koristi inverzno adresiranje
 $IA \leftrightarrow DB$

Ldm/stm PRAVILA!!!

- Ako se podaci žele zapisati u memoriju naredbom STM, a zatim pročitati u istom redoslijedu naredbom LDM te ako se koriste nastavci za RAD SA STOGOM, tada način adresiranja u naredbi LDM mora biti **ISTI** načinu adresiranja u naredbi STM

- Primjer:

- Zapisivanje
- Čitanje

STMFD

LDMFD

U obje naredbe
koristi se jednako
adresiranje FD

Primjeri korištenja LDM i STM ...



U donjim primjerima pretpostavljene vrijednosti prije izvođenja naredaba LDM/STM su:

r0=0 r1=1 r2=2 r3=3 r13=9000

a) STMIB r13, {r0,r1,r2,r3} ; vrijednosti se sprema, r13 ostane nepromijenjen

Nakon izvođenja gornje naredbe, stanje u memoriji je (heksadekadski, little-endian):

0x00009000: 10 00 FF E7 00 00 00 00 01 00 00 00 02 00 00 00

0x00009010: 03 00 00 00 00 E8 00 E8 10 00 FF E7 00 E8 00 E8

r13=9000

b) STMIB r13, {r3,r1,r0,r2} ; redoslijed pisanja registara ne utječe na redoslijed spremanja

0x00009000: 10 00 FF E7 00 00 00 00 01 00 00 00 02 00 00 00

0x00009010: 03 00 00 00 00 E8 00 E8 10 00 FF E7 00 E8 00 E8

r13=9000

c) STMIB r13!, {r0,r3} ; osvježava se r13

0x00009000: 10 00 FF E7 00 00 00 00 03 00 00 00 E8 00 E8 00

r13=9008

... Primjeri korištenja LDM i STM ...



d) **STMDB r13!, {r0,r1,r2,r3}** ; primjer koristenja DB

0x00008FF0: 00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00

0x00009000: 10 00 FF E7 00 E8 00 E8 10 00 FF E7 00 E8 00 E8

r13 = 8FF0

e) **STMDA r13!, {r0,r1,r2,r3}** ; primjer koristenja DA i kasnijeg obnavljanja registara

;0x00008FF0 00 E8 00 E8 00 00 00 00 01 00 00 00 02 00 00 00

;0x00009000 03 00 00 00 00 E8 00 E8 10 00 FF E7 00 E8 00 E8

;r13 = 8FF0

Da bi u registre r0, r1, r2 i r3 učitali iste podatke koje smo iz njih spremili u memoriju pomoću naredbe STMDA, moramo upotrijebiti naredbu LDM s inverznim nastavkom IB:

LDMIB r13!, {r0,r1,r2,r3}

Primjeri korištenja LDM i STM



f) STMFA r13!, {r0,r1,r2,r3}

; primjer spremanja registara pomoću STM i kasnijeg
; obnavljanja registara naredbom LDM uz korištenje
; nastavaka za STOG

0x00009000: 00 E8 00 E8 00 00 00 00 01 00 00 00 02 00 00 00

0x00009010: 03 00 00 00 00 E8 00 E8 10 00 FF E7 00 E8 00 E8

r13 = 9010

Da bi u registre r0, r1, r2 i r3 učitali iste podatke koje smo iz njih spremili u memoriju pomoću naredbe STMFA, moramo upotrijebiti naredbu LDM s istim nastavkom FA:

LDMFA r13!, {r0,r1,r2,r3}

; za adresiranje stoga nastavci moraju biti isti

Naredbe za obradu podataka

- ARM-ove naredbe za obradu podataka obrađuju podatke iz registara
- Postoje tri grupe naredaba za obradu podataka:
 - aritmetičko-logičke naredbe i naredbe za usporedbu
 - naredbe za množenje
 - naredba za brojenje vodećih nula

Aritmetičko-logičke naredbe i naredbe za usporedbu

Ime	Engleski naziv	Opis naredbe
AND	Logical AND	$Rd := Rn \text{ AND drugi_operand}$ (logičko I)
EOR	Logical Exclusive OR	$Rd := Rn \text{ EOR drugi_operand}$ (logičko ekskl. ILI)
ORR	Logical (inclusive) OR	$Rd := Rn \text{ OR drugi_operand}$
BIC	Bit Clear	$Rd := Rn \text{ AND NOT}(\text{drugi_operand})$
ADD	Add	$Rd := Rn + \text{drugi_operand}$
ADC	Add with Carry	$Rd := Rn + \text{drugi_operand} + C \text{ zastavica}$
SUB	Subtract	$Rd := Rn - \text{drugi_operand}$
SBC	Subtract with Carry	$Rd := Rn - \text{drugi_operand} - \text{NOT}(C \text{ zastavica})$
RSB	Reverse Subtract	$Rd := \text{drugi_operand} - Rn$
RSC	Reverse Subtract with Carry	$Rd := \text{drugi_operand} - Rn - \text{NOT}(C \text{ zastavica})$
CMP	Compare	Osvježi zastavice nakon $Rn - \text{drugi_operand}$
CMN	Compare Negated	Osvježi zastavice nakon $Rn + \text{drugi_operand}$
TST	Test	Osvježi zastavice nakon $Rn \text{ AND drugi_operand}$
TEQ	Test Equivalence	Osvježi zastavice nakon $Rn \text{ EOR drugi_operand}$
MOV	Move	$Rd := \text{drugi_operand}$ (prvog operanda nema)
MVN	Move Not	$Rd := \text{NOT drugi_operand}$ (prvog operanda nema)

Aritmetičko-logičke naredbe

- Aritmetičko-logičke naredbe imaju dva operanda (osim naredaba MOV i MVN koje imaju samo jedan operand) i spremaju rezultat u zadani registar* (osim naredaba za usporedbu koje zanemaruju rezultat)
- Od dva operanda koji se mogu koristiti u operaciji jedan uvijek mora biti registar, a drugi može biti
 - neposredna vrijednost
 - registar
 - vrijednost registra nad kojom se obavlja određen pomak

* Za razliku od FRISC-a kod kojeg se odredišni registar piše na kraju - iza operandada, kod ARM-a se odredišni registar zadaje na početku - prije operandada

Aritmetičko-logičke naredbe

- Ako se aritmetičko-logičkoj naredbi doda nastavak 'S' (Save condition codes) tada će se nakon izvršene naredbe osvježiti zastavice stanja
- Bez nastavka 'S', naredba **ne mijenja** zastavice*
- Naredbe za usporedbu uvijek osvježavaju zastavice stanja (i nigdje ne spremaju rezultat)

* Različito od FRISC-a, koji uvijek osvježava zastavice

Primjer programa za aritm. naredbe

64-bitno oduzimanje

Treba napisati program za oduzimanje dvaju 64-bitnih brojeva. Operandi neka su zapisani u memoriji s početkom na adresi 8100_{16} . Rezultat treba spremiti u memoriju iza operanada. Za dohvat operanada i spremanje rezultata koristiti naredbu Load/Store Multiple.

Rješenje:

```
MOV R4, #81<8           ; postavlja registar R4 na 8100(16)
                        ; ovaj način zadavanja konstante biti će detaljno opisan kasnije
LDMIA R4!,{R5,R6,R7,R8} ; učitavanje oba operanda
                        ; korištenjem Load Multiple naredbe
SUBS R5, R5, R7           ; oduzimanje niza 32 bita
                        ; s postavljanjem zastavica
SBCS R6, R6, R8           ; oduzimanje visa 32 bita i
                        ; posudbe iz prethodne operacije
STMIA R4!, {R5,R6}       ; spremanje rezultata
SWI 123456
```


Naredbe za množenje

- Naredbe za množenje nisu smještene u grupu osnovnih aritmetičko-logičkih naredbi, već se definiraju zasebno
- Postoji šest naredbi za množenje: MUL, MLA, SMULL, UMULL, SMLAL i UMLAL
 - **U ATLAS-u su implementirane samo naredbe MUL i MLA!!!**
- Obično množenje (MUL) se obavlja nad dva 32-bitna podatka iz registara opće namjene, a 32-bitni rezultat (nižih 32 bita umnoška) se ponovo sprema u registar
- Naredbe za dugo množenje (SMULL i UMULL) množe dva 32-bitna podatka iz registara i spremaju svih 64 bita rezultata u dva odabrana registra za rezultat. Dugo množenje može množiti brojeve s predznakom (SMULL) ili bez predznaka (UMULL)

Naredbe za množenje

- Obično 32-bitno množenje koristimo kad znamo da su operandi "dovoljno mali" da rezultat stane u 32-bita, a dugo 64-bitno množenje koristimo u općenitim slučajevima
- Kao dodatna opcija gore navedenim naredbama nudi se mogućnost da se umnožak pribraja sadržaju nekog registra (ili paru registara za dugo množenje). Takve naredbe općenito su poznate kao Multiply Accumulate, a kod procesora ARM nazivaju se: MLA, SMLAL i UMLAL
- Ako se naredbama za množenje doda nastavak S, tada će se nakon izvođenja osvježiti zastavice N i Z.

Naredbe za množenje

Ime	Engleski naziv	Opis naredbe
MUL	Multiply	$Rd = (Rm * Rs)[31:0]$
MLA	Multiply Accumulate	$Rd = (Rm * Rs + Rn)[31:0]$
SMULL	Signed Multiply Long	$RdHi = (Rm * Rs)[63:32]$ /*Sa predznakom*/ $RdLo = (Rm * Rs)[31:0]$
UMULL	Unsigned Multiply Long	$RdHi = (Rm * Rs)[63:32]$ /*Bez predznaka*/ $RdLo = (Rm * Rs)[31:0]$
SMLAL	Signed Multiply Accumulate Long	$RdLo = (Rm * Rs)[31:0] + RdLo$ /*Sa predznakom*/ $RdHi = (Rm * Rs)[63:32] + RdHi +$ prijenos iz $((Rm * Rs)[31:0] + RdLo)$
UMLAL	Unsigned Multiply Accumulate Long	$RdLo = (Rm * Rs)[31:0] + RdLo$ /*Bez predznaka*/ $RdHi = (Rm * Rs)[63:32] + RdHi +$ prijenos iz $((Rm * Rs)[31:0] + RdLo)$

Primjer programa za množenje (bez korištenja naredaba za množenje)

Program za množenje dvaju 16-bitnih brojeva treba napisati metodom zbrajanja. Neka su multiplikator i multiplikand zapisani u memoriji s početkom na adresi 8100₁₆. Pretpostavite da je multiplikator broj bez predznaka dok multiplikand može biti s predznakom. Rezultat treba spremiti u memoriju iza operanada.

; Uvjet zadatka: multiplikator je broj bez predznaka

```
MOV R4, #81<8           ; postavlja R4 adresu podataka
LDRH R5, [R4], #2        ; multiplikator se učitava u R5
LDRSH R6, [R4], #2       ; multiplikand se učitava u R6 (predznak sacuvan)
MOV R7, #0               ; cisti se registar za spremanje rezultata (R7)
```

```
PETLJA  CMP R5, #0        ; usporedba multiplikatora i nule
        BEQ KRAJ          ; ako je nula, mnozenje je gotovo
        ADD R7, R7, R6     ; pribroji multiplikand rezultatu.
        SUB R5, R5, #1     ; umanji multiplikator za jedan
        B PETLJA
KRAJ    STR R7, [R4]       ; spremi rezultat
        SWI 123456
```

Naredba za brojenje vodećih nula

- Naredba za brojenje vodećih nula je specifična naredba koja je vrlo korisna kod izvedbe matematičkih algoritama (normiranje brojeva) i algoritama za kompresiju (npr. metode duljine niza).
 - **U ATLAS-u nije implementirana naredba CLZ!!!**
- Ova naredba uzima jedan registar kao operand i vraća rezultat u drugom registru.
- Rezultat predstavlja broj nula koje prethode najvišoj jedinici u ulaznom operandu (promatranom kao niz binarnih znamenaka).

CLZ

- Primjer korištenja naredbe za brojenje vodećih nula:

Neka je u registru R1=**00000**111100011110111111111100011

Nakon izvođenja naredbe:

CLZ R2, R1

u registru R2 bit će rezultat **5**

Naredbe grananja

- Naredba B (Branch): bezuvjetno ili uvjetno grananje na memorijsku adresu koja se nalazi 32 MB ispred ili iza trenutačne naredbe* (relativan skok u odnosu na sadržaj PC)
- Drugi način grananja je da se u registar PC izravno stavi neka vrijednost, čime se skok ne ograničava na udaljenost od 32 MB, već se može skočiti na bilo koju adresu u adresnom području (tj. može se skočiti bilo gdje unutar 4 GB) **

* Slično FRISC-ovoj naredbi **JR**

** Sličan učinak kod FRISC-a možemo dobiti naredbom **JP (Rx)**

Naredbe grananja - uvjeti

Mnemonički naziv	Puni naziv (engleski)	Ispitivano stanje zastavica
EQ	Equal	Z
NE	Not equal	!Z
CS/HS	Carry set/unsigned higher or same	C
CC/LO	Carry clear/unsigned lower	!C
MI	Minus/negative	N
PL	Plus/positive or zero	!N
VS	Overflow	V
VC	No overflow	!V
HI	Unsigned higher	C and !Z
LS	Unsigned lower or same	!C or Z
GE	Signed greater than or equal	N == V
LT	Signed less than	N != V
GT	Signed greater than	!Z and N == V
LE	Signed less than or equal	Z or N != V
AL	Always (unconditional)	-
(NV)	See Condition code 0b1111	-

Grananje u potprogram

- Naredba BL (Branch and Link) poziva potprogram:
 - Sprema adresu sljedeće naredbe (povratnu adresu) **u registar R14** (koji se tada naziva Link Registrar, LR)
 - nakon toga izvodi grananje na početak potprograma.
- Povratak iz potprograma izvodi se jednostavnim kopiranjem sadržaja registra LR u PC (npr. **naredbom MOV PC,LR**)
- Koji problemi se mogu pojaviti ?

- Ovakvim pozivom i povratkom iz potprograma nije moguće ugniježđeno pozivanje potprograma!
- Za ugniježdene pozive, na početku svakog potprograma mora se spremiti vrijednost iz R14 na stog, a prije povratka iz potprograma treba vratiti tu vrijednost sa stoga u R14.

Primjeri naredaba za grananje

B labela ; bezuvjetni skok

BCC labela ; uvjetni skok (carry clear)

BEQ labela ; uvjetni skok (equal)

MOV PC, #0 ; R15 = 0, tj. skoči na adresu 0

MOV PC, R3 ; R15 = R3, skok na bilo koju 32-bitnu adresu
; koja je zadana registrom R3

BL func ; poziv potprograma

MOV PC, LR ; R15=R14, tj. povratak iz potprograma

Primjer uvjetnog izvođenja niza naredaba

1. način - korištenjem naredbe uvjetnog grananja:

```
CMP R0, #0
BNE DALJE
MOV R1, #1    ; uvjetni dio koda
MOV R2, #2    ; uvjetni dio koda
MOV R3, #3    ; uvjetni dio koda
DALJE
...
```

Ovisno o protočnoj strukturi i ovisno koliko često je ispitivani uvjet istinit, može se odrediti koji način pisanja je efikasniji u pojedinoj situaciji.

2. način - korištenjem uvjetnog izvođenja naredaba:

```
CMP R0, #0
MOVEQ R1, #1    ; uvjetni dio koda
MOVEQ R2, #2    ; uvjetni dio koda
MOVEQ R3, #3    ; uvjetni dio koda
```

Naredbe za pristup registrima stanja

- Omogućuju prijenos podataka iz registara stanja (CPSR, SPSR) procesora ARM u neki registar opće namjene i obratno.
- Pisanjem u CPSR, na primjer, programer može postaviti stanja zastavica, stanja bitova za omogućavanje prekida kao i procesorski način.
- Naredba **MRS** (Move to Register from Status register) kopira sadržaj registra stanja CPSR ili SPSR iz načina rada u kojem se procesor trenutno nalazi u jedan od registara opće namjene koji se može dalje ispitivati ili mijenjati.
- Naredbom **MSR** Move to Status register from Register može se upisati neposredna vrijednost ili sadržaj registra opće namjene u registre stanja CPSR ili SPSR iz načina rada u kojem se procesor trenutno nalazi.

Naredbe za pristup registrima stanja

Oznaka	Bitovi	Naziv polja u registru
f	[31:24]	Polje zastavica (flags)
s	[23:16]	Polje stanja (status)
e	[15:8]	Polje proširenja (extension)
c	[7:0]	Polje upravljanja (control)

- Svaki registar stanja podijeljen je u 4 polja kojima se može neovisno pristupati. Prilikom upisa u registar stanja naredbom MSR, programer mora definirati u koje polje želi upisati podatak
- Primjer naredaba:
 - MRS R0, CPSR
 - MSR CPSR_cesf, R0
 - MSR CPSR_f, R0
 - MSR CPSR_fsc, R0

Primjer korištenja

- primjer omogućavanja prekida (brisanje bita I u registru CPSR)

```
MRS R0, CPSR
```

```
BIC R0, R0, #0x80
```

```
MSR CPSR_C, R0
```

ARM Adresiranja i potprogrami

Načini adresiranja

- Procesor ARM omogućuje različita adresiranja. Oblici adresiranja svrstani su u pet načina (engl. Addressing Modes) i u tablicama u Prilogu su označeni brojevima 1 do 5.
- Načini adresiranja koriste se u pojedinim naredbama prema sljedećoj tablici:
 - načini adresiranja 1: naredbe za obradu podataka
 - načini adresiranja 2: naredbe Load i Store word ili unsigned byte
 - načini adresiranja 3: naredbe Load i Store halfword ili signed byte
 - načini adresiranja 4: naredbe Load i Store Multiple
 - načini adresiranja 5: naredbe Load i Store Coprocessor

Načini adresiranja 1

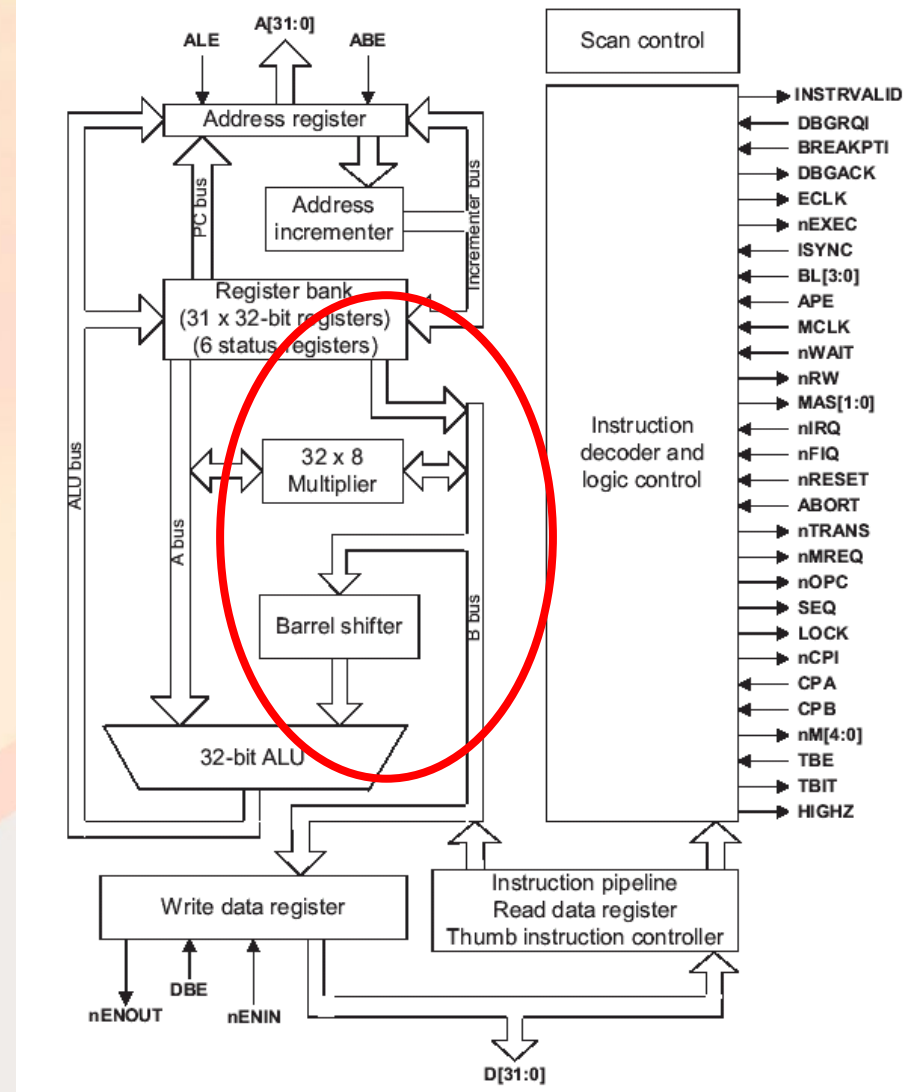
- Načini adresiranja 1 su adresiranja u “širem smislu” jer se u njima ne dohvaća podatak iz memorije (ne koristi se adresa)
- Koriste se za **definiranje jednog od operandada u naredbama za obradu podataka**. Osnovni oblik naredaba koje koriste taj način adresiranja izgleda ovako:

`<opcode>{<cond>}{S} <Rd>, <Rn>, <shifter_operand>`

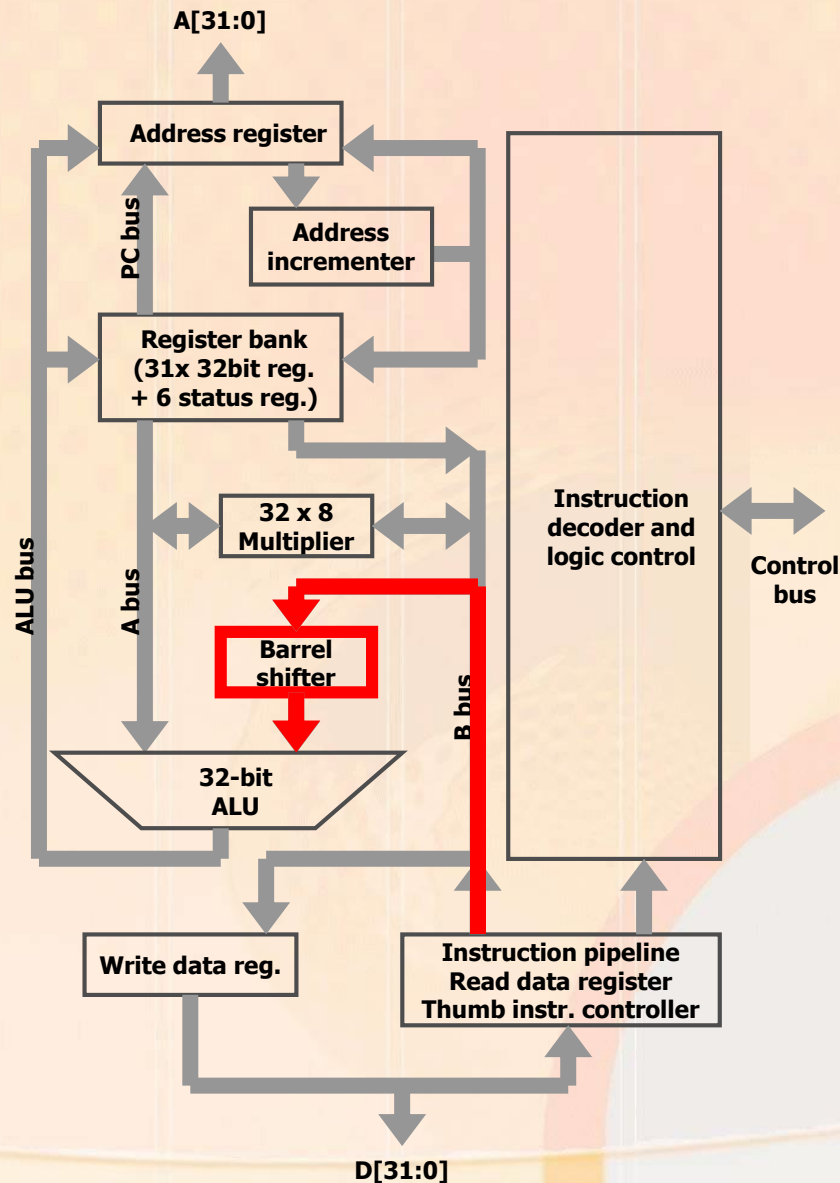
- Drugi operand `<shifter_operand>` može se “adresirati” na 11 različitih načina koji tvore načine adresiranja 1

Načini adresiranja 1

- Već smo prije naučili da drugi operand može biti
 - Neposredna vrijednost
 - Vrijednost iz registra
 - Vrijednost iz registra s pomakom
- Sve ove mogućnosti zasnivaju se na arhitekturi puta podataka prema ALU

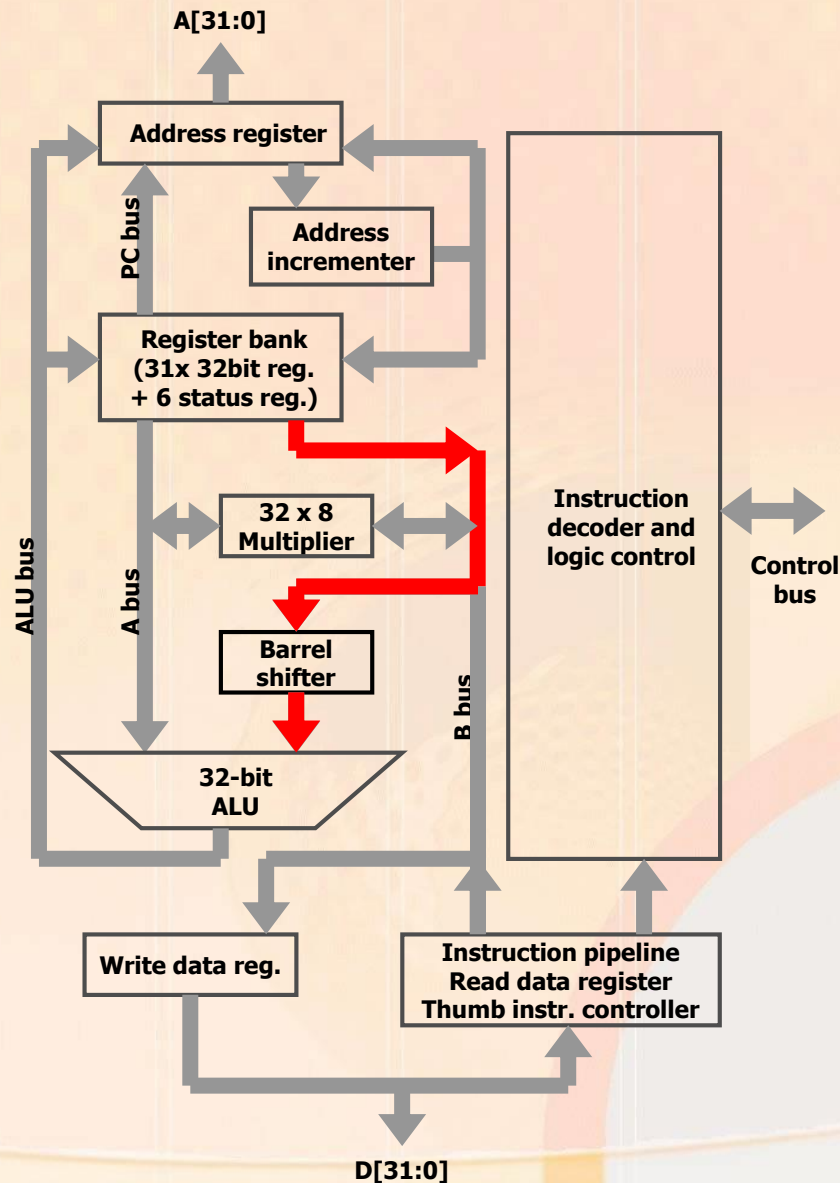


Podloga za način1 u arhitekturi procesora



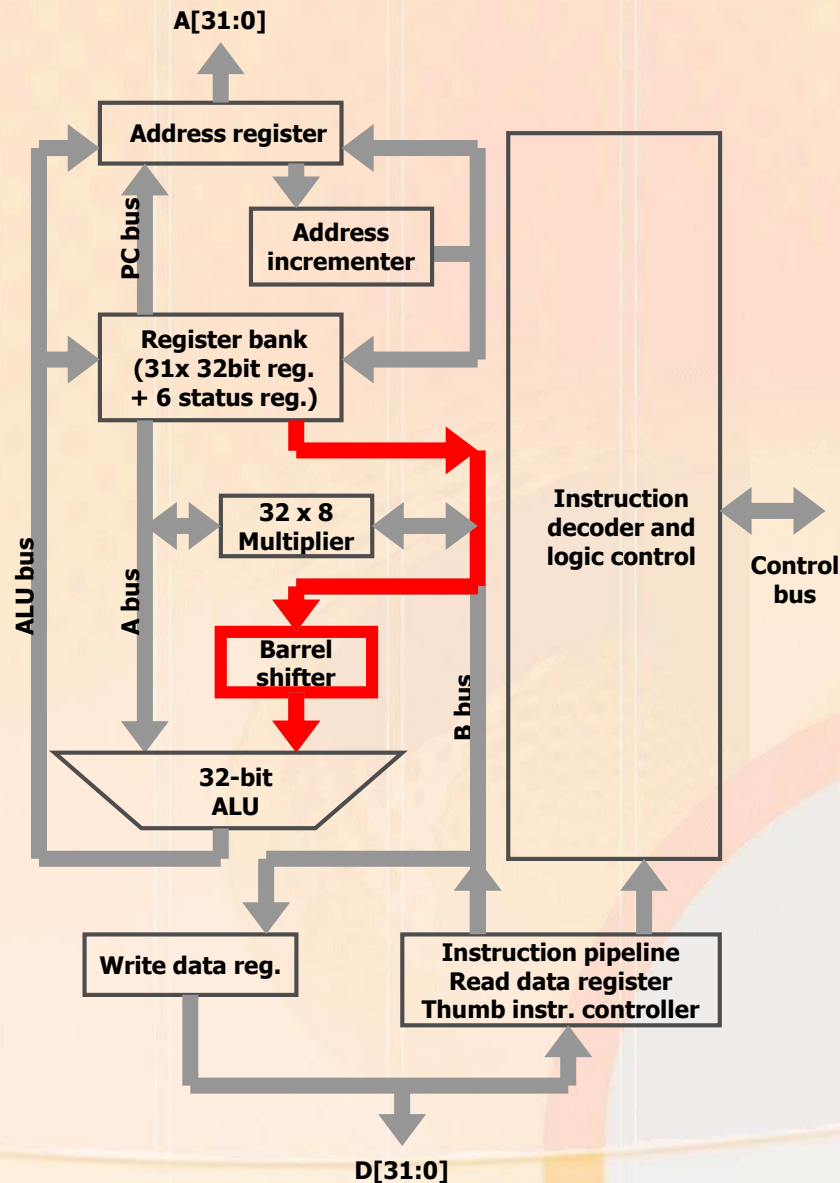
Neposredna vrijednost

Podloga za način1 u arhitekturi procesora



Vrijednost iz registra

Podloga za način1 u arhitekturi procesora



Vrijednost iz registra s pomakom

Načini adresiranja 1

Adresiranje	Sintaksa adresiranja
Neposredno	#<immediate>
Registarsko	<Rm>
Registarsko s neposrednim logičkim pomakom ulijevo (ASL je sinonim za LSL)	<Rm>,LSL #<rshift_imm> <Rm>,ASL #<shift_imm>
Registarsko s registarskim logičkim pomakom ulijevo (ASL je sinonim za LSL)	<Rm>,LSL <Rs> <Rm>,ASL <Rs>
Registarsko s neposrednim logičkim pomakom udesno	<Rm>,LSR #<shift_imm>
Registarsko s registarskim logičkim pomakom udesno	<Rm>,LSR <Rs>
Registarsko s neposrednim aritmetičkim pomakom udesno	<Rm>,ASR #<shift_imm>
Registarsko s registarskim aritmetičkim pomakom udesno	<Rm>,ASR <Rs>
Registarsko s neposrednim rotiranjem udesno	<Rm>,ROR #<shift_imm>
Registarsko s registarskim rotiranjem udesno	<Rm>,ROR <Rs>
Registarsko s proširenim rotiranjem udesno	<Rm>,RRX

Neposredno #<immediate>

- Neposredna 32-bitna konstanta
- Za zapis konstante u strojnom kôdu naredbe na raspolaganju je samo 12 bitova !!!
- poseban način računanja konstanti:
 - od 12 bitova koji su na raspolaganju, 8 ih se koristi za neposrednu vrijednost (immed_8), a preostala 4 bita (rotate_imm) definiraju broj rotacija udesno 8-bitne neposredne vrijednosti:

Naredba		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Aritmetičko logička, neposredna vrijednost	Uvjet	0			0	1	Op kod				S	Rn				Rd				rotate_imm				immed_8									

Definiranje neposredne vrijednosti

- Neposredna 32-bitna vrijednost računa se sljedećom formulom:
 - $\text{<immediate>} = \text{<immed_8> rotirano udesno za } (2 * \text{<rotate_imm>})$
- Ovakvim postupkom opseg svih brojeva koji se mogu opisati neposredno, proširen je na sva 32 bita, odnosno od 0 do $2^{32}-1$
 - Naravno, na ovaj način **ne mogu se zapisati sve moguće 32-bitne konstante**, već samo one koje se mogu dobiti rotiranjem 8-bitnog broja (0-255) za **paran** broj mjesta udesno (0, 2, 4, ..., 30).
- U **ARM-ovim** alatima je dovoljno napisati željeni broj, a asemblerski prevoditelj će **sam izračunati** može li se taj broj prikazati na ovaj način. Nažalost u ATLAS-u to nije moguće.

Primjeri dozvoljenih i nedozvoljenih vrijednosti

- Dozvoljene vrijednosti
 - FF, 104, FF0, FF00, FF000, FF000000, F000000F,...
- Nedozvoljene vrijednosti
 - 101, 102, FF1, FF04, FF003, FFFFFFFF, F000001F,...
- Napomena:
 - mnoge vrijednosti mogu se dobiti upotrebom naredbe MVN (Move NOT) koja radi komplement operanda
 - Npr: običnom naredbom ne možemo koristiti neposrednu vrijednost FFFFFFF00. No, tu neposrednu vrijednost možemo učitati u registar koristeći naredbu MVN, SUB i slično:

NE!!! MOV R0, #0FFFFFFF

Da!!! MVN R0, #0

Primjeri dozvoljenih i nedozvoljenih vrijednosti

NE!!! MOV R4, #0FFFFFF0

Da!!! MVN R4, #0F00000F

- Neke aritmetičke operacije možemo zamijeniti KOMPLEMENTARNIM OPERACIJAMA

NE!!! ADDS R4, R1, #0FFFFFFF

Da!!! SUBS R4, R1, #1

- Primjer dozvoljene neposredne vrijednosti kodiran u binarnom obliku:
[e29142ff] adds r4,r1,#0f00000f

Naredba	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Aritmetičko logička, neposredna vrijednost	Uvjet				0	0	1	Op kod				S	Rn				Rd				rotate_imm				immed_8							
ADDS R4,R1,#0F00000F	1110				0	0	1	0100				1	0001				0100				0010				11111111							

ATLAS - NAPOMENA

- U primjerima koje radite za laboratorijske vježbe (**u ATLAS-u**) neposredne vrijednosti **morate kodirati sami** (assembler ne podržava automatsku pretvorbu)

- Način pisanja:

baza < rotacija (u ATLAS-u se koristi **LIJEVA** rotacija)

- Primjer:

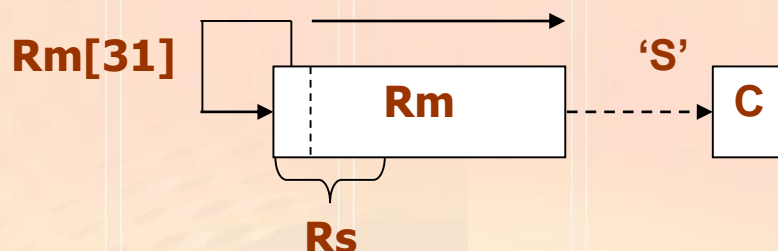
broj 100_{16} piše se kao $1 < 8$ $\underline{1} \ 0000 \ 0000_2 = 1_2 < 8$

broj 180_{16} piše se kao $18 < 4$ $\underline{1 \ 1000} \ 0000_2 = 11000_2 < 4$

ili kao $6 < 6$ $\underline{1 \ 1000} \ 0000_2 = 110_2 < 6$

Registarsko i registarsko s pomakom

- Ako se nastavak 'S' doda naredbi koja inače ne mijenja zastavicu C, tada se C puni bitom označenim strelicom
 - Primjer: ... <Rm>, ASR <Rs>



- Registar Rm se ne mijenja, nego njegova pomaknuta vrijednost služi samo kao operand
- Iznos pomaka zadaje se registrom ili neposredno (brojem)
- Ostale pomake i rotiranja proučite iz tablice u Prilogu

Načini adresiranja 1 u Prilogu "Zbrike ARM"

- Pogledajte tablicu za npr. naredbu MOV:

MOV{cond}{S} Rd, <Oprnd2>

- U tablici "Polje Operand2 <Oprnd2>" izaberete neki od načina adresiranja
- Napomena!!! **Ne postoji** naredba npr. LSL R0, #2 (dešava se na ispitima). Treba koristiti naredbu MOV (u ovom primjeru: MOV R0, R0, LSL #2).

Načini adresiranja 2

- Karakteristični su za naredbe load i store kojima se premješta riječ ili nepredznačeni bajt
- Postoji devet mogućih kombinacija adresiranja s obzirom na vrstu odmaka te vrstu indeksiranja
- Odmak se zadaje neposrednim podatkom, registrom ili pomaknutim registrom*. Ako je izabran pomak registra, tada se vrsta pomaka (LSL, LSR, ASR, ROR, RRX) zadaje kao u načinu adresiranja 1, no uz ograničenje da se iznos pomaka zadaje samo neposredno

* Odmak se ne mora napisati i tada se podrazumijeva da je 0 (zadan neposredno)

Načini adresiranja 2

Adresiranje	Sintaksa adresiranja
Neposredni odmak	[<Rn>, #+/-<offset_12>]
Registarski odmak	[<Rn>, +/-<Rm>]
Registarski odmak s pomakom	[<Rn>, +/-<Rm>, LSL #<shift_imm>] [<Rn>, +/-<Rm>, LSR #<shift_imm>] [<Rn>, +/-<Rm>, ASR #<shift_imm>] [<Rn>, +/-<Rm>, ROR #<shift_imm>] [<Rn>, +/-<Rm>, RRX]
Neposredni predindeksirani	[<Rn>, #+/-<offset_12>]!
Registarski predindeksirani	[<Rn>, +/-<Rm>]!
Registarski predindeksirani odmak s pomakom	[<Rn>, +/-<Rm>, LSL #<shift_imm>]! [<Rn>, +/-<Rm>, LSR #<shift_imm>]! [<Rn>, +/-<Rm>, ASR #<shift_imm>]! [<Rn>, +/-<Rm>, ROR #<shift_imm>]! [<Rn>, +/-<Rm>, RRX]!
Neposredni postindeksirani	[<Rn>], #+/-<offset_12>
Registarski postindeksirani	[<Rn>], +/-<Rm>
Registarski postindeksirani odmak s pomakom	[<Rn>], +/-<Rm>, LSL #<shift_imm> [<Rn>], +/-<Rm>, LSR #<shift_imm> [<Rn>], +/-<Rm>, ASR #<shift_imm> [<Rn>], +/-<Rm>, ROR #<shift_imm> [<Rn>], +/-<Rm>, RRX

Načini adresiranja 2 u Prilogu "Zbirke ARM"

- Pogledajte tablicu za npr. naredbu LDR:

LDR{cond} Rd, <a_mode2>

- U tablici "Način adresiranja <a_mode2>" izaberete neki od načina adresiranja i upotrijebite ga u naredbi. Na primjer:

LDRHI R0, [R3, -R5, ROR #22]!

Načini adresiranja 3

- Šest formata adresa koje se koriste kod ostalih naredaba load i store (za poluriječ i predznačeni bajt)
- Ovih šest formata adresa slični su formatima iz načina adresiranja 2. Razlike su:
 - odmak se može zadati neposrednom vrijednošću ili registrom (ne može se zadati registar s pomakom)
 - neposredna vrijednost odmaka se zapisuje samo sa osam bitova

Načini adresiranja 3

Adresiranje	Sintaksa adresiranja
Neposredni odmak	[<Rn>, #+/-<offset_8>]
Registarski odmak	[<Rn>, +/-<Rm>]
Neposredni preindeksirani odmak	[<Rn>, #+/-<offset_8>]!
Registarski preindeksirani odmak	[<Rn>, +/-<Rm>]!
Neposredni postindeksirani odmak	[<Rn>], #+/-<offset_8>
Registarski postindeksirani odmak	[<Rn>], +/-<Rm>

Načini adresiranja 3 u Prilogu "Zbirke ARM"

- Pogledajte tablicu za npr. naredbu LDR:

LDR{cond}SB Rd, <a_mode3>

- u tablici "Način adresiranja <a_mode3>" izaberete neki od načina adresiranja i upotrijebite ga u naredbi. Na primjer:

LDRGTSB R0, [R3], #20

Načini adresiranja 4

- Naredbama load i store multiple moguće je pročitati ili upisati sadržaj jednog, sadržaj više ili čak sadržaje svih registara u memoriju, odnosno u niz uzastopnih memorijskih lokacija.
- Registar s najmanjim brojem je uvijek zapisan na najmanju memorijsku adresu, a registar s najvećim brojem na najveću.
- Na isti način, kod čitanja: u registar s najmanjim brojem učitava se podatak s najmanje adrese, a u registar s najvećim brojem učitava se podatak s najveće adrese.
- IA, IB, DA, DB (FD, FA, ED, EA)

Načini adresiranja 4 u Prilogu "Zbirke ARM"

- Pogledajte tablicu za npr. naredbu LDM:
LDM{cond} <a_mode4L> Rd{!}, <reglist-pc>
- U tablici "Način adresiranja <a_mode4L> izaberete neki od načina adresiranja i upotrijebite ga u naredbi. Npr.
LDMGTIA R0!, {R4,R5,R6}
- Analogno vrijedi za naredbe STM i tablicu "Način adresiranja <a_mode4S>.
- Napomena: <reglist-pc> u opisu naredbe znači da se u popis registara ne smije staviti PC

Načini adresiranja 5

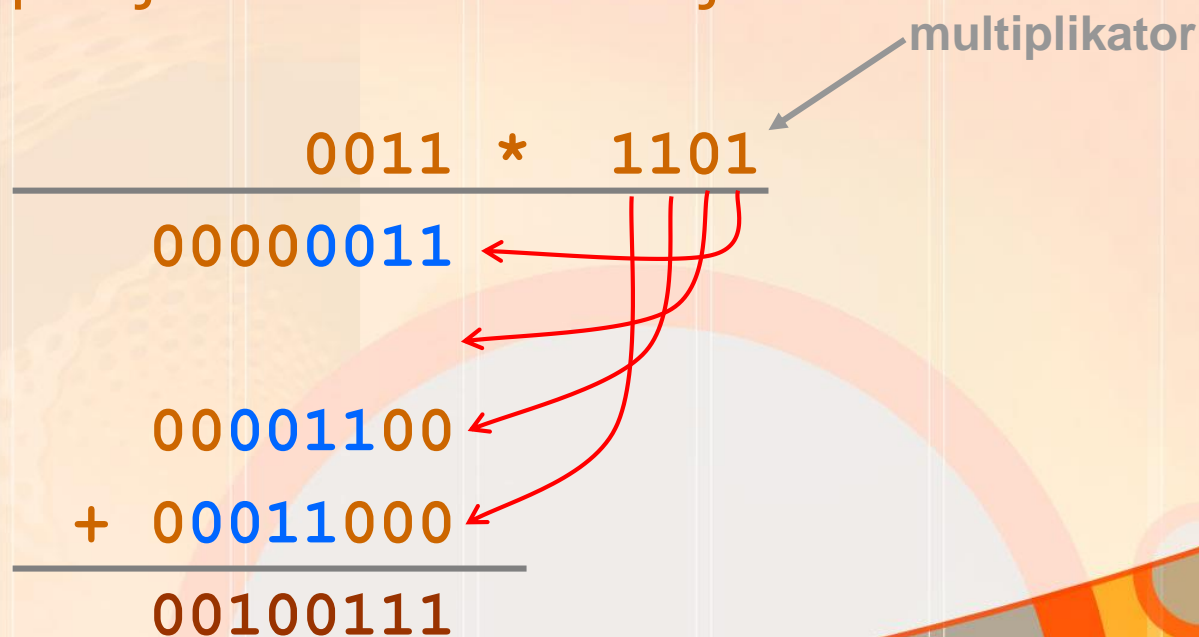
- Koriste se kod naredaba za prijenos podataka prema koprocesoru
- Nećemo ih koristiti

Primjer: množenje dva 16-bitna broja

Treba pomnožiti dva 16-bitna broja metodom pomaka. Operandi su spremljeni od adrese 8100, a 32-bitni rezultat treba staviti iza operandada. Multiplikand može biti u zapisu NBC ili 2'k, a multiplikator može biti samo u zapisu NBC.

Načelo množenja na primjeru dva 4-bitna broja:

multiplikator

$$\begin{array}{r} 0011 * 1101 \\ \hline 00000011 \\ 00001100 \\ + 00011000 \\ \hline 00100111 \end{array}$$


Primjer programa za množenje

; Program za množenje 16-bitnih brojeva metodom pomaka

```
MOV R4, #81<8      ;postavlja u registar R4 adresu podataka 8100
LDRH R5, [R4], #2    ;multiplikator se učitava u R5
LDRSH R6, [R4], #2   ;multiplikand se učitava u R6 (predznak sačuvan)
MOV R7, #0           ;čisti se registar za spremanje rezultata (R7)
```

PETLJA

```
MOVS R5, R5, LSR #1 ;pomak multiplikatora za jedan bit udesno,
                    ;najniži bit otići će u C

ADDCS R7, R7, R6     ;ako je zastavica C=1, R6 se dodaje privremenom rezultatu
MOV R6, R6, LSL #1   ;pomak R6 za jedan bit ulijevo
                    ;(priprema za mogući sljedeći korak)

CMP R5, #0           ;provjerava je li multiplikator različit od nule
BNE PETLJA           ;ako jeste, onda se petlja ponavlja

STR R7, [R4]         ;spremanje rezultata
```

■ ■ ■

Primjer dijeljenja metodom pomaka



- Dijeljenje A/B (djelitelja B , djeljeniku A) odvija se u dva koraka:
- **Prvi korak:** poravnanje brojeva – traženje najvećeg višekratnika djelitelja sadržanog u djeljeniku
 - Djelitelj B pomičemo u lijevo sve dok vrijedi $A \geq 2B$ i $\text{MSB}(B)=0$ te pamtimo **broj_pomaka**
 - Ponavlja se dok $A \geq 2B$ zato da pronađemo **najveći** višekratnik
 - Prije ispitivanja $A \geq 2B$ provjeravamo da li je $\text{MSB}(B)=1$
 - Ako $\text{MSB}(B)=1$ tada smo sigurni da je to najveći višekratnik jer bi množenjem s 2 dobili broj koji je veći od opsega brojeva koji se mogu prikazati u 32 bita što bi sigurno bilo veće od djeljenika
 - U slučaju $\text{MSB}(B)=1$ moramo preskočiti ispitivanje $A \geq 2B$ jer bi usporedba mogla dovesti do krivog rezultata (B pomaknuto u lijevo za jedan bit prelazi 32 bita registra)
- **Drugi korak:** Nakon što od djeljenika oduzmemo najveći višekratnik djelitelja, postupak se ponavlja **broj_pomaka** puta (kao dijeljenje na papiru - vidi sljedeći slajd)

Postupak...



1101101001 : 1010 = 1

max.6 pomaka djelitelja u lijevo, oduzmemo od djeljenika i stavimo 1 u rez

- 1010000000

0011101001 : 1010 = 10

(5) Djelitelj za 1 u desno, rez 1 u lijevo

1010000000

-Ne može se oduzeti, dodamo 0 u rez

0011101001 : 1010 = 101

(4) Djelitelj za 1 u desno, rez 1 u lijevo

- 10100000

-Može se oduzeti, dodamo 1 u rez

0001001001 : 1010 = 1010

(3) Djelitelj za 1 u desno, rez 1 u lijevo

1010000

-Ne može se oduzeti, dodamo 0 u rez

0001001001 : 1010 = 10101

(2) Djelitelj za 1 u desno, rez 1 u lijevo

- 101000

-Može se oduzeti, dodamo 1 u rez

0000100001 : 1010 = 101011

(1) Djelitelj za 1 u desno, rez 1 u lijevo

- 10100

-Može se oduzeti, dodamo 1 u rez

0000001101 : 1010 = 1010111

(0) Djelitelj za 1 u desno, rez 1 u lijevo

- 1010

-Može se oduzeti, dodamo 1 u rez

0000000011 (ostatak)

Broj_pomaka = 0 -> KRAJ POSTUPKA

Primjer programa za dijeljenje



; Program za 32-bitno dijeljenje metodom pomaka

MOV R4, #81<8	; postavlja u registar R4 adresu podataka 8100
LDR R5, [R4], #4	; djeljenik (A) se učitava u R5
LDR R6, [R4], #4	; djelitelj (B) se učitava u R6
MOV R7, #0	; čisti se registar za spremanje rezultata (R7)
MOV R8, #0	; brojač inicijalnih pomaka
CMP R6, R5	; ako je B>A, nema potrebe za dijeljenjem
BHI KRAJ	

PORAVNAJ ;inicijalni korak: pronalaženje najvećeg višekratnika djelitelja

ANDS R3, R6, #80<24	; provjerava je li MSB djelitelja jednak 1
BNE DIV	; ako jeste, poravnavanje je nepotrebno
CMP R5, R6, LSL #1	; provjerava je li $A \geq 2*B$
MOVHS R6, R6, LSL #1	; ako je, pomiče B ulijevo
ADDHS R8, R8, #1	; povećava brojač pomaka
BHI PORAVNAJ	; samo ako je $A > 2*B$, poravnavanje se nastavlja

Primjer programa za dijeljenje (2. dio)



DIV

CMP R5, R6	; uspoređuje trenutni ostatak i B
SUBHS R5, R5, R6	; ako je ostatak \geq B, onda ga umanji za B
ADDHS R7, R7, #1	; i poveća rezultat za 1
CMP R8, #0	; ako je brojač pomaka > 0 , nastavi, a inače KRAJ
MOVHI R6, R6, LSR #1	; pomakne B udesno,
MOVHI R7, R7, LSL #1	; a rezultat ulijevo
SUBHI R8, R8, #1	; umanji brojač pomaka
BHI DIV	; i ponovi petlju

KRAJ

STR R5, [R4], #4	; spremanje ostatka
STR R7, [R4]	; spremanje rezultata
■ ■ ■	

Još jedan primjer....

```
*****  
; Program koji niz podataka (terminiran nulom) koji se nalazi u memoriji  
; na adresi 8200 ispisuje u obrnutom redoslijedu na istu adresu. (obrni_niz_stog)  
*****
```

```
MOV    R13, #84<8    ; inicijalizacija stoga  
MOV    R1, #82<8     ; učitavanje početne adrese  
MOV    R0, #0         ; brojač duljine niza
```

```
PETLJA LDR    R2, [R1], #4    ; petlja kojom čitamo niz  
        CMP    R2, #0        ; ako je nula, niz je gotov  
        BEQ    ISPIS  
        STMFD  SP!, {R2}  
        ADD    R0, R0, #1     ; povećaj brojač  
        B      PETLJA        ; idi dalje
```

>>>>

... nastavak

<<<<

ISPIS	MOV	R1, #82<8	; učitavanje početne adrese
POM	CMP	R0, #0	
	BEQ	KRAJ	
	LDMFD	SP!, {R2}	
	STR	R2, [R1], #4	; prema početku originalnog niza
	SUB	R0, R0, #1	; smanjujemo brojač
	B	POM	
KRAJ	SWI	123456	; Kraj

Ispitivanje specijalnog broja...

```
*****  
;  
;      Ispitivanje je li broj specijalan  
;      specNum.s  
;  
*****  
;  
; Napisati program koji ispituje je li troznamenkasti dekadski broj 'xyz' zapisan  
; kao niz ASCII znamenaka (tzv. string) "xyz\0" specijalan broj za kojeg vrijedi:  
;  $xyz = xy * xy - z * z$ , gdje su x y i z znamenke stotice, desetice i jedinice.  
; Primjer takvog broja su brojevi:  $100 = 10*10 - 0*0$  i  $147 = 14*14 - 7*7$   
; Broj zapisan kao niz ASCII znamenki sa završnim null-znakom nalazi se na  
; adresi 1000. Ukoliko broj zadovoljava gornji uvjet, tada je potrebno u registar  
; r1 staviti sve jedinice, a ako uvjet nije ispunjen, tada u r1 treba staviti sve  
; nule.  
*****  
;
```

Specbroj...

; ovaj odsječak vadi znamenke iz ASCII zapisa: u r1 će se nalaziti znamenka
; stotica, u r2 će se nalaziti znamenka desetica, a u r3 će biti znamenka jedinica

```
main    MOV  r0, #10<8
        LDRB r1, [r0], #1
        LDRB r2, [r0], #1
        LDRB r3, [r0]
        SUB  r1, r1, #100
        SUB  r2, r2, #10
        SUB  r3, r3, #1
```

; znamenka stotica (48 je ASCII znak od 0)
; desetice
; jedinice

; množenje s konstantom 100 ostvareno je kombinacijom naredaba ADD i MOV
; s pomakom, što je efikasnije od korištenja naredbe MUL (multiply)

```
ADD  r5, r1, r1, LSL #3
ADD  r5, r5, r1, LSL #4
MOV  r5, r5, LSL #2
```

; $r5 = r1 + 8 * r1 = 9 * r1$
; $r5 = r5 + 16 * r1 = 9 * r1 + 16 * r1 = 25 * r1$
; konačno $r5 = 4 * r5 = 4 * 25 * r1 = 100 * r1$

>>>>

Specbroj...

<<<<

; množenje s konstantom 10 pomoću kombinacije naredaba ADD i MOV s pomakom

```
MOV r6, r2
```

```
ADD r6, r6, r6, LSL #2
```

```
MOV r6, r6, LSL #1 ; r6 = 10 * r2
```

```
ADD r7, r5, r6
```

```
ADD r7, r7, r3 ; konačan broj u binarnom zapisu, potreban za ispitivanje  
; uvjeta zadatka  $xyz = xy * xy - z * z$ 
```

; generiranje broja xy a nakon toga i broja $xy * xy$ te $z * z$

```
ADD r1, r1, r1, LSL #2
```

```
MOV r1, r1, LSL #1
```

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

```
MUL r4, r1, r1
```

```
MUL r5, r3, r3
```

; u r1 se nalazi broj xy

; u r4 se nalazi broj $xy * xy$

; u r5 se nalazi broj $z * z$

>>>>

Specbroj...

<<<<

SUB r4, r4, r5 ; r1 = xy*xy - z*z

CMP r4, r7

MVNEQ r1, #0 ; sve jedinice u R1

MOVNE r1, #0 ; sve nule u R1

KRAJ SWI 123456 ; Kraj programa

ORG 1000

; neki ASCII podaci ...

Specbroj...

Komentari:

Množenje kombinacijom ALU-operacija može se ostvariti kad množimo s konstantom. Obično se može ostvariti na više načina. Na primjer, množenje sa 100:

$$100 = ((1+8)+16)*4 \quad (\text{iz primjera: 3 naredbe})$$

$$100 = 128-32+4 \quad (3 \text{ naredbe})$$

$$100 = (1+32)*3+1 \quad (3 \text{ naredbe})$$

$$100 = 64+32+4 \quad (3 \text{ naredbe})$$

$$100 = (1+16)*3*2-2 \quad (4 \text{ naredbe})$$

Potprogrami

Potprogrami

- Poziv potprograma: BL
- Povratak iz potprograma: MOV PC, LR
- Povratna adresa sprema se u registar R14 (LR)

Primjer atoh_v1.s

Zadatak:

Napisati program koji 32-bitni heksadekadni broj zapisan u ASCII kodu (niz od 8 ASCII znakova) pretvara u broj. Znakovi se nalaze u memoriji od adrese 8100_{16} , a rezultat treba spremiti u memoriju iza znakova.

Za pretvorbu ASCII heksadekadske znamenke u njenu brojevnju vrijednost treba koristiti potprogram. Potprogram prima ASCII znamenku preko R0 i vraća rezultat preko R0. Pretpostavka je da je heksadekadska ASCII znamenka ispravna, tj. da može sadržati dekadski znamenke od "0" do "9", mala i velika slova od "a" do "f".

Podsjetnik za ASCII kodove:

'0'	48	'A'	65	'a'	97
'1'	49	'B'	66	'b'	98
...
'9'	57	'F'	70	'f'	102

Primjer atoh_v1.s

```
GLAVNI MOV R4, #81<8  
      MOV R7, #8  
      MOV R6, #0
```

; postavlja R4 na početak podataka
; brojač hex znamenki = 8
; resetiranje rezultata

PETLJA

```
LDRB R0, [R4], #1  
BL FUNC_ATOH  
ADD R6, R0, R6, LSL #4
```

; učitavanje ASCII-znaka
; poziv potprograma za pretvorbu
; spremanje rezultata iz R0 u registar R6 uz
; pomak trenutačnog rezultata za 4 bita

```
SUBS R7, R7, #1  
BHI PETLJA
```

; brojač prolaza petlje

```
STR R6, [R4]
```

; spremanje rezultata

```
KRAJ SWI 123456
```

>>>>

Primjer atoh_v1.s

```
<<<<
FUNC_ATOH                ; potprogram za pretvorbu ASCII-znaka u HEX
                          ; ulazni ASCII-znak je u R0, a rezultat je također u R0

SUB  R0, R0, #D48         ; ASCII:0 = DEC: 48
CMP  R0, #D10             ; ako je broj 10 ili veći, znači da se radi o slovu
MOVLO PC,LR              ; povratak iz potprograma ako je R0<10

SUB  R0, R0, #D7          ; ASCII:A = DEC: 65
                          ; treba oduzeti 65-55=10
                          ; ranije smo već oduzeli 48, pa trebamo još 7 (48+7=55)
CMP  R0, #D16             ; ako je broj 16 ili veći, znači da se radi o malom slovu
MOVLO PC,LR              ; povratak iz potprograma ako je R0<16

SUB  R0, R0, #D32         ; ASCII:a = DEC: 97
                          ; treba oduzeti 97-87=10
                          ; ranije smo već oduzeli 55, pa trebamo još 32 (55+32=87)
MOV  PC, LR              ; povratak iz potprograma
```

Napomena: potprogram ne mijenja registre pa nema spremanja konteksta

Prijenos parametara

- preko registara*
 - problem sa brojem registara koji su na raspolaganju
 - isto kao kod FRISC-a
- preko stoga
 - često najpraktičnije rješenje
- parametri iza naredbe poziva
- preko fiksnih memorijskih lokacija
 - isto kao kod FRISC-a

* pokazano u prethodnom primjeru

Poziv potprograma s parametrima na stogu

Potprogram računa $f(X,A,B) = (X \ll A) + B$, parametri i rezultat se prenose stogom. Parametre uklanja potprogram. **Potprogram mijenja registre.**

GLAVNI	MOV R13, #81<8	; definicija vrha stoga	(1)
	MOV R0, #1	; parametar X=1	
	MOV R1, #4	; parametar A=4	
	MOV R2, #3	; parametar B=3	
	STMFD R13!, {R0, R1, R2}	; parametri se spremaju na stog	(2)
	BL FUNC_PARAM	; poziv potprograma	
	LDMFD R13!, {R0}	; rezultat se sa stoga učitava u R0	(5)
DALJE	...	; nastavak glavnog programa	

FUNC_PARAM		; potprogram	
	LDMFD R13!, {R4, R5, R6}	; učitavanje parametara X,A,B	(3)
	ADD R4, R6, R4, LSL R5	; računanje funkcije	
	STMFD R13!, {R4}	; spremanje rezultata	(4)
	MOV PC, LR	; povratak iz potprograma	

Poziv potprograma s parametrima na stogu

Izgled stoga:

nakon (1):

80F0	
80F4	
80F8	
80FC	
8100	←R13

nakon (2):

80F0	
80F4	1 ←R13
80F8	4
80FC	3
8100	

nakon (3):

80F0	
80F4	
80F8	
80FC	
8100	←R13

nakon (4):

80F0	
80F4	
80F8	
80FC	rez. ←R13
8100	

nakon (5):

80F0	
80F4	
80F8	
80FC	
8100	←R13

Poziv potprograma s parametrima na stogu

Potprogram računa $f(X,A,B) = (X \ll A) + B$ kao i prije, **ali uz čuvanje registara (DOBRO RJEŠENJE)**. Parametri se prenose stogom i uklanja ih pozivatelj. Rezultat se vraća u R0.

GLAVNI	MOV R13, #81<8	; definicija vrha stoga	(1)
	MOV R0, #1	; parametar X=1	
	MOV R1, #4	; parametar A=4	
	MOV R2, #3	; parametar B=3	
	STMFD R13!, {R0, R1, R2}	; parametri se spremaju na stog	(2)
	BL FUNC_PARAM	; poziv potprograma	
	ADD R13, R13, #0D 12	; ukloni parametre sa stoga	(6)
DALJE	STR R0, REZ	; spremi rezultat i nastavi	
	...		
FUNC_PARAM		; potprogram	
	STMFD R13!, {R4, R5, R6}	; spremi registre	(3)
	ADD R0, R13, #0D 12	; sa R0 "preskoči" spremljene registre	(4)
	LDMFD R0, {R4, R5, R6}	; učitavanje ulaznih parametara X,A,B	
	ADD R0, R6, R4, LSL R5	; računanje funkcije	
	LDMFD R13!, {R4, R5, R6}	; obnovi registre	(5)
	MOV PC, LR	; povratak iz potprograma	

Poziv potprograma s parametrima na stogu

Izgled stoga:

nakon (1):

80F0	
80F4	
80F8	
80FC	
8100	←R13

nakon (2):

80F0	
80F4	1 ←R13
80F8	4
80FC	3
8100	

nakon (3):

80E8	R4	←R13
80EC	R5	
80F0	R6	
80F4	1	
80F8	4	
80FC	3	
8100		

nakon (4):

80E8	R4	←R13
80EC	R5	
80F0	R6	
80F4	1	←R0
80F8	4	
80FC	3	
8100		

nakon (5):

80F0	
80F4	1 ←R13
80F8	4
80FC	3
8100	

nakon (6):

80F0	
80F4	
80F8	
80FC	
8100	←R13

Poziv potprograma s parametrima iza poziva

Potprogram računa $f(X)=(X \ll A) + B$, parametri A i B su iza naredbe, a parametar X je u registru R0. Rezultat se vraća preko R0. **Potprogram mijenja registre!!!**

```
GLAVNI MOV R0, #1           ; parametar X=1
        BL FUNC_PARAM       ; poziv potprograma
        DW 4                 ; parametar A=4
        DW 3                 ; parametar B=3
DALJE   ...                  ; nastavak programa, rezultat je u R0
```

```
FUNC_PARAM ; potprogram, adresa prvog parametra je u R14
LDR R1, [R14], #4 ; prvi parametar (A) se učitava u R1, a R14 se
                  ; poveća tako da pokazuje na drugi parametar
LDR R2, [R14], #4 ; drugi parametar (B) se učitava u R2, a R14 se
                  ; poveća tako da je u njemu povratna adresa
ADD R0, R2, R0, LSL R1 ; računanje funkcije
MOV PC, LR             ; povratak iz potprograma
```

Poziv potprograma s parametrima iza poziva

Potprogram računa $f(X)=(X \ll A) + B$, parametri A i B su u lokacijama iza naredbe, a parametar X je u registru R0. Rezultat se vraća preko R0.

Potprogram čuva registre (DOBRO RIJEŠENJE).

```
GLAVNI MOV R13, #81<8      ; definicija vrha stoga
        MOV R0, #1          ; parametar X=1
        BL FUNC_PARAM      ; poziv potprograma
        DW 4                ; parametar A=4
        DW 3                ; parametar B=3
DALJE   ...                ; nastavak programa, rezultat je u R0
```

```
FUNC_PARAM      ; potprogram, adresa prvog parametra je u R14
    STMFD R13!, {R1, R2} ; spremi registre
    LDR R1, [R14], #4     ; prvi parametar (A) se učitava u R1
    LDR R2, [R14], #4     ; drugi parametar (B) se učitava u R2
                    ; sada je u R14 povratna adresa
    ADD R0, R2, R0, LSL R1 ; računanje funkcije
    LDMFD R13!, {R1, R2}  ; obnovi registre
    MOV PC, LR            ; povratak iz potprograma
```

Ugniježđeni pozivi

- Problem: R14 služi za spremanje povratne adrese
 - već prvi ugniježđeni poziv uništava početni R14
- Jedino dobro rješenje je spremanje R14 na stog na početku potprograma te čitanje povratne adrese sa stoga prije povratka iz potprograma
- Primjeri:
 - Poziv potprograma iz potprograma
 - Rekurzivna funkcija

Primjer poziva potprograma iz potprograma

Treba napisati funkciju $c(x)=(x/2) + 3$, ali tako da se koriste pomoćne funkcije $a(x)=x/2$ i $b(x)=x+3$. Potprogrami trebaju čuvati registre.

```
glavni      MOV  R13, #80<8      ; inicijalizacija SP
            MOV  R0, #7          ; parametar x=7
            BL   FUNC_C          ; poziv potprograma
            ...                  ; nastavak glavnog programa, rezultat je u R1

FUNC_A      MOV  R0, R0, ASR #1   ; R0= R0/2
            MOV  PC, LR          ; povratak iz potprograma

FUNC_B      ADD  R0, R0, #3       ; R0 = R0+3
            MOV  PC, LR          ; povratak iz potprograma

FUNC_C      STMFD r13!, {R0,R14} ; R1 = R0/2 + 3
            BL   FUNC_A          ; x=x/2
            BL   FUNC_B          ; x=x/2 +3
            MOV  R1, R0          ; rezultat stavi u R1
            LDMFD r13!, {R0,R14}
            MOV  PC, LR          ; povratak iz potprograma
```


Primjer računanja rekurzivne funkcije (faktorijela)

Napisati potprogram za računanje faktorijele korištenjem rekurzivnog poziva. Parametar funkcije neka se prenosi preko registra R0, a rezultat funkcije neka se vraća u registru R1.

Glavni program treba izračunati faktorijela(6). Potprogram treba čuvati registre.

```
int faktorijela (int x) {  
    if( x == 1)  
        return (1);  
    return ( faktorijela(x-1) * x );  
}
```

Primjer računanja rekurzivne funkcije (faktorijela)

GLAVNI	MOV R13, #81<8	; vrh stoga
	MOV R0, #6	; ulazni parametar=6
	BL FUNC_FACT	; rezultat je u R1
KRAJ	SWI 123456	; Kraj
FUNC_FACT	STMFD R13!, {R0, LR}	; spremanje registara na stog
	CMP R0, #1	
	BNE X_NIJE_1	; ako x != 1 onda rekurzija
X_JE_1	MOV R1, #1	; inicijalna vrijednost faktorije
	LDMFD R13!, {R0, LR}	; obnovi registre sa stoga
	MOV PC, LR	; povratak iz potprograma za x==1
X_NIJE_1	SUB R0, R0, #1	; R0 = x-1
	BL FUNC_FACT	; R1 = f(x-1)
	ADD R0, R0, #1	; R0 = x
	MUL R1, R0, R1	; izračunaj R1 = x*f(x-1)
	LDMFD R13!, {R0, LR}	; obnovi registre sa stoga
	MOV PC, LR	; povratak iz potprograma

Primjer Manhattan

- Poziv potprograma iz potprograma
- Prijenos parametara preko stoga
- Čuvaju se registri

- Zadatak:

Napisati potprogram MANH koji računa Manhattan-udaljenost dvije točke: $(x1, y1)$ i $(x2, y2)$. Manhattan-udaljenost definirana je formulom:

$$|x1 - x2| + |y1 - y2|$$

Potprogram MANH preko stoga prima koordinate točaka $x1, y1, x2, y2$ kao parametre, a rezultat vraća u registru R7. Apsolutna vrijednost razlike dva broja računa se posebnim potprogramom ADIFF koji parametre prima preko stoga i rezultat vraća u registru R8.

...glavni program

ORG	0	
GLAVNI	MOV R13, #1<16	; inicijalizacija pokazivača stoga
	MOV R0, #9	; Prva točka je (9, 3)
	MOV R1, #3	
	STMFD R13!, {R0, R1}	; koordinate prve točke se stavljaju ; kao parametri na stog
	MOV R0, #5	; Druga točka je (5, 8)
	MOV R1, #8	
	STMFD R13!, {R0, R1}	; koordinate druge točke se stavljaju ; kao parametri na stog
	BL MANH	
	ADD R13, R13, #0D16	; uklanjaju se parametri sa stoga ; rezultat je u R7
	SWI 123456	; kraj

... potprogram MANH

MANH

STMFD R13!, {R0,R1,R2,R3,R4,R8,R14} ; registri koji se koriste moraju se sačuvati
ADD R0, R13, #0 ; R0 preskače nove podatke na stogu
LDMFD R0, {R1, R2, R3, R4} ; učitavaju se pohranjeni parametri
; R1=x2,R2=y2,R3=x1,R4=y1

STMFD R13!, {R1, R3}
BL ADIFF ; poziva se ADIFF(R1,R3)
ADD R13, R13, #8 ; uklanjaju se parametri sa stoga
MOV R1, R8 ; rezultat iz r8 sprema se u r1

STMFD R13!, {R2, R4}
BL ADIFF ; poziva se ADIFF(R2,R4)
ADD R13, R13, #8 ; uklanjaju se parametri sa stoga
MOV R2, R8 ; rezultat iz r8 sprema se u r2

ADD R7, R1, R2 ; zbrajaju se dvije apsolutne razlike

LDMFD R13!, {R0,R1,R2,R3,R4,R8, R14} ; obnavljaju se registri
MOV PC, LR

... potprogram ADIFF

ADIFF

STMFD R13!, {R0, R1, R2} ; registri koji se koriste moraju se sačuvati

ADD R0, R13, #0 ; R0 preskače nove podatke na stogu

LDMFD R0, {R1, R2} ; učitavaju se pohranjeni parametri

SUBS R8, R1, R2 ; izračunavanje razlike i postavljanje zastavica

RSBLT R8, R8, #0 ; ako je R8 negativan, onda $R8 = 0 - R8 = -R8$

LDMFD R13!, {R0, R1, R2} ; obnavljaju se pohranjeni registri

MOV PC, LR

Računalni sustav s procesorom ARM

Iznimke

Obrada iznimaka

- U ARM-u se **iznimkama** nazivaju razne vrste događaja koje ne ulaze u normalno slijedno izvođenje naredaba. Na primjer: pojava prekida, dohvat neispravnog strojnog koda, resetiranje procesora, itd.
- Procesor ARM načelno obrađuje iznimke na sljedeći način:
 - ARM pohranjuje R15 (programsko brojilo) u registar LR, a zatim pohranjuje CPSR (registar trenutnog programskog stanja) u registar SPSR*
 - ARM prelazi u privilegirani način rada koji ovisi o vrsti iznimke
 - ARM skače u potprogram za obradu iznimke za dotični način rada
- Adrese potprograma za obradu iznimke su fiksno definirane za svaku pojedinu iznimku
- Ako se u isto vrijeme pojave dvije ili više iznimaka, ARM definira prioritete izvođenja

* osim za RESET

Iznimke: adrese potprograma i prioriteti

Tip iznimke	ARM se prebacuje u način rada	Adresa potprograma*	Prioritet
Reset	Supervisor	0x00000000	1
Undefined instruction	Undefined	0x00000004	6
Software interrupt (SWI)	Supervisor	0x00000008	6
Prefetch Abort (instruction fetch memory abort)	Abort	0x0000000C	5
Data Abort (data access memory abort)	Abort	0x00000010	2
IRQ (interrupt)	IRQ	0x00000018	4
FIQ (fast interrupt)	FIQ	0x0000001C	3

* Uočite da je za svaki potprogram na raspolaganju samo 4 bajta, pa se tu u pravilu stavlja naredba skoka na odsječak za obradu iznimke (osim potprograma na adresi 0x1C)

Reset

- Slijed operacija koje se obave pri pojavi pojedine iznimke vrlo je sličan za sve iznimke
- Kada se na ulazu u procesor aktivira signal Reset, procesor odmah prekida izvođenje naredbe. Nakon što se Reset deaktivira, obavi se sljedeći niz operacija:
 - R14_svc = UNPREDICTABLE value
 - SPSR_svc = UNPREDICTABLE value
 - CPSR[4:0] = %B10011 /* Enter Supervisor mode */
 - CPSR[5] = 0 /* Execute in ARM state */
 - CPSR[6] = 1 /* Disable fast interrupts */
 - CPSR[7] = 1 /* Disable normal interrupts */
 - PC = %H00000000

Reset

- Povratak iz iznimke Reset nije predviđen. U slučaju inicijalizacije procesora (npr. uključivanje napajanja) očekuje se da će vanjska logika aktivirati signal Reset kako bi procesor normalno započeo s radom.
- Prema tome, svaki program mora pretpostaviti obradu iznimke Reset i nakon toga prelazak na izvođenje izabrane aplikacije.

IRQ (prekid)

- Ako je u programu omogućeno prihvaćanje prekida (bit I u CPSR je obrisan), procesor će na kraju izvođenja svake naredbe provjeriti je li ulaz IRQ aktivan. Ako je neki vanjski sklop aktivirao signal prekida (IRQ), procesor će nakon završetka trenutne naredbe obaviti sljedeći niz operacija:

- R14_irq = address of next instruction to be executed + 4
- SPSR_irq = CPSR
- CPSR[4:0] = %B10010 /* Enter IRQ mode */
- CPSR[5] = 0 /* Execute in ARM state */
- /* CPSR[6] is unchanged */
- CPSR[7] = 1 /* Disable normal interrupts */
- PC = %H00000018

IRQ

- Za povratak iz IRQ-potprograma treba izvesti naredbu:

SUBS PC, R14, #4

- Ova naredba* će obnoviti PC (iz R14_irq) i CPSR (iz SPSR_irq) te nastaviti izvođenje programa na mjestu gdje je prekinut

* SUB napisan s ekstenzijom S i odredišnim registrom PC, znači da treba obnoviti CPSR

FIQ (brzi prekid)

- Brzi prekid namjenjen je primjenama gdje je bitno brzo reagirati i obaviti niz operacija
- FIQ zato ima dovoljan broj 'privatnih' registara tako da ne treba obavljati operacije pohranjivanja i vraćanja konteksta na stog
- Pored toga, adresa prekidnog potprograma FIQ je namjerno zadnja na listi tako da se potprogram može napisati odmah od te adrese bez potrebe za skokom na drugo mjesto u memoriji (izbjegnuto je kašnjenje zbog takvog skoka)

FIQ

- Ako je u programu omogućeno prihvaćanje brzog prekida (bit F u CPSR je obrisan) procesor će na kraju izvođenja svake naredbe provjeriti da li je ulaz FIQ aktivan. Ako je na ulazu u procesor aktiviran signal brzog prekida (FIQ), procesor će nakon završetka trenutne naredbe obaviti sljedeći niz operacija:
 - `R14_fiq` = address of next instruction to be executed + 4
 - `SPSR_fiq` = CPSR
 - `CPSR[4:0]` = `%B10001` /* Enter FIQ mode */
 - `CPSR[5]` = 0 /* Execute in ARM state */
 - `CPSR[6]` = 1 /* Disable fast interrupts */
 - `CPSR[7]` = 1 /* Disable normal interrupts */
 - `PC` = `%H0000001C`

FIQ

- Za povratak iz FIQ-potprograma treba izvesti naredbu (isto kao za IRQ):

SUBS PC, R14,#4

- Ova naredba će obnoviti PC (iz R14_fiq) i CPSR (iz SPSR_fiq) te nastaviti izvođenje programa na mjestu gdje je prekinut

Primjer obrade iznimke

Glavni program treba povećavati registar R1. Kada dođe do prekida IRQ, treba postaviti oznaku da treba završiti glavni program.

```
ORG 0  
B GLAVNI
```

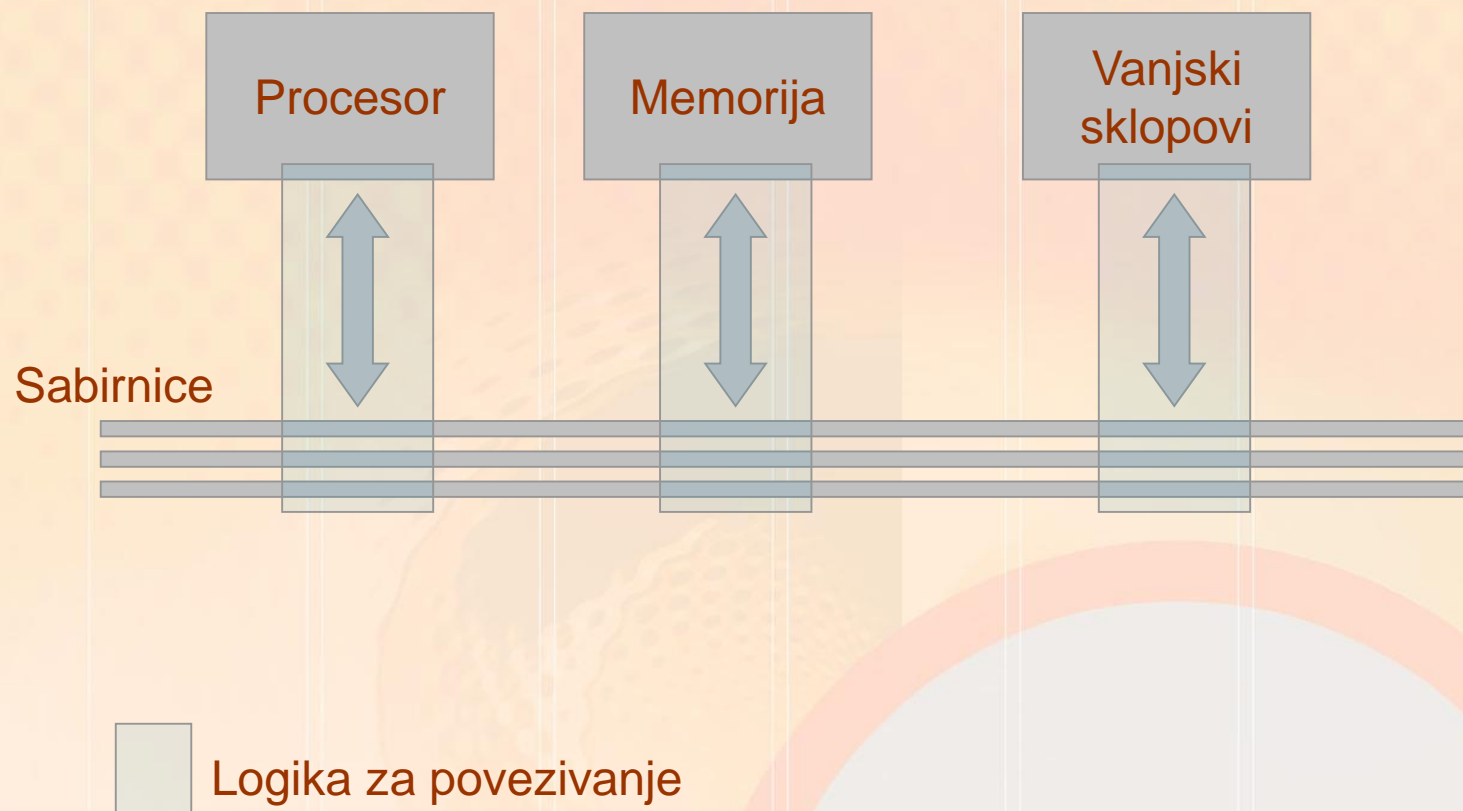
```
ORG 18  
B OBR_IRQ
```

```
GLAVNI  MRS  r0, CPSR      ; pročitaj CPSR  
        BIC  r0, r0, #80   ; pobriši bit I (za bit F koristi se maska 40)  
        MSR  CPSR_c, r0   ; upiši promjenu u CPSR (dozvoli IRQ)  
  
        MOV  r0, #1       ; oznaka da treba ponavljati petlju  
  
PETLJA  ADD  r1, r1, #1     ; povećavaj r1 dok ne dođe prekid  
        CMP  r0, #0        ; provjeri oznaku kraja  
        BNE  PETLJA  
  
        SWI  123456
```

```
OBR_IRQ ; Potprogram za obradu iznimke  
        MOV  R0, #0        ; oznaka da treba završiti petlju  
        SUBS PC, R14, #4    ; povratak iz iznimke
```

Računalni sustav s procesorom ARM

Osnovni dijelovi računalnog sustava



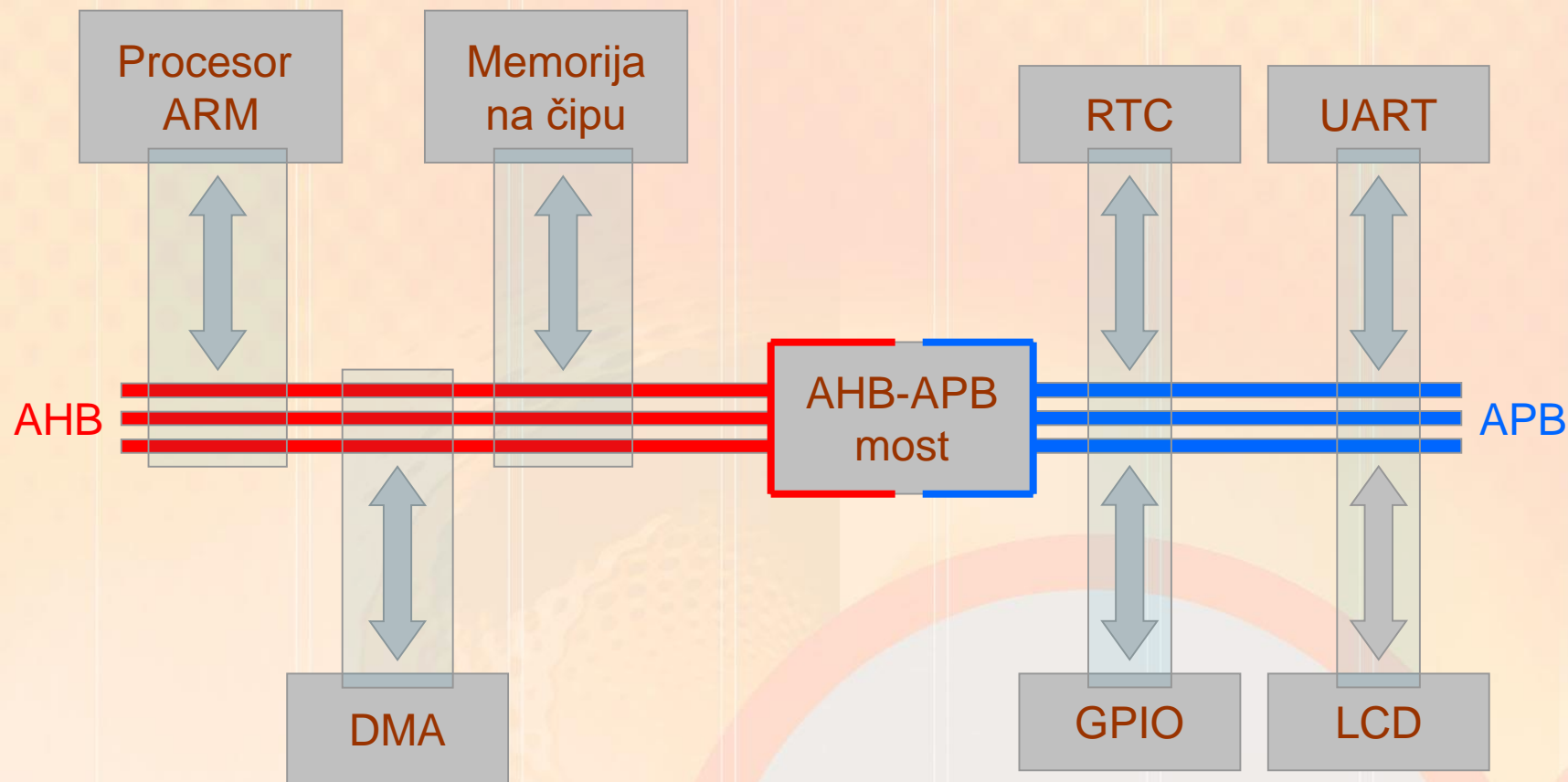
AMBA

- AMBA, kratica za Advanced Microcontroller Bus Architecture
- Trenutno aktualne specifikacije su AMBA 4
(ACE,ACE-Lite,AXI4, AXI4-Lite,AXI4-Stream,AHB,APB,ATB)
- Mi ćemo u okviru ovog predmeta proučiti samo načelno dvije sabirnice koje su definirane još u AMBA2 specifikacijama:
 - AHB (Advanced High-performance Bus)
 - APB (Advanced Peripheral Bus).

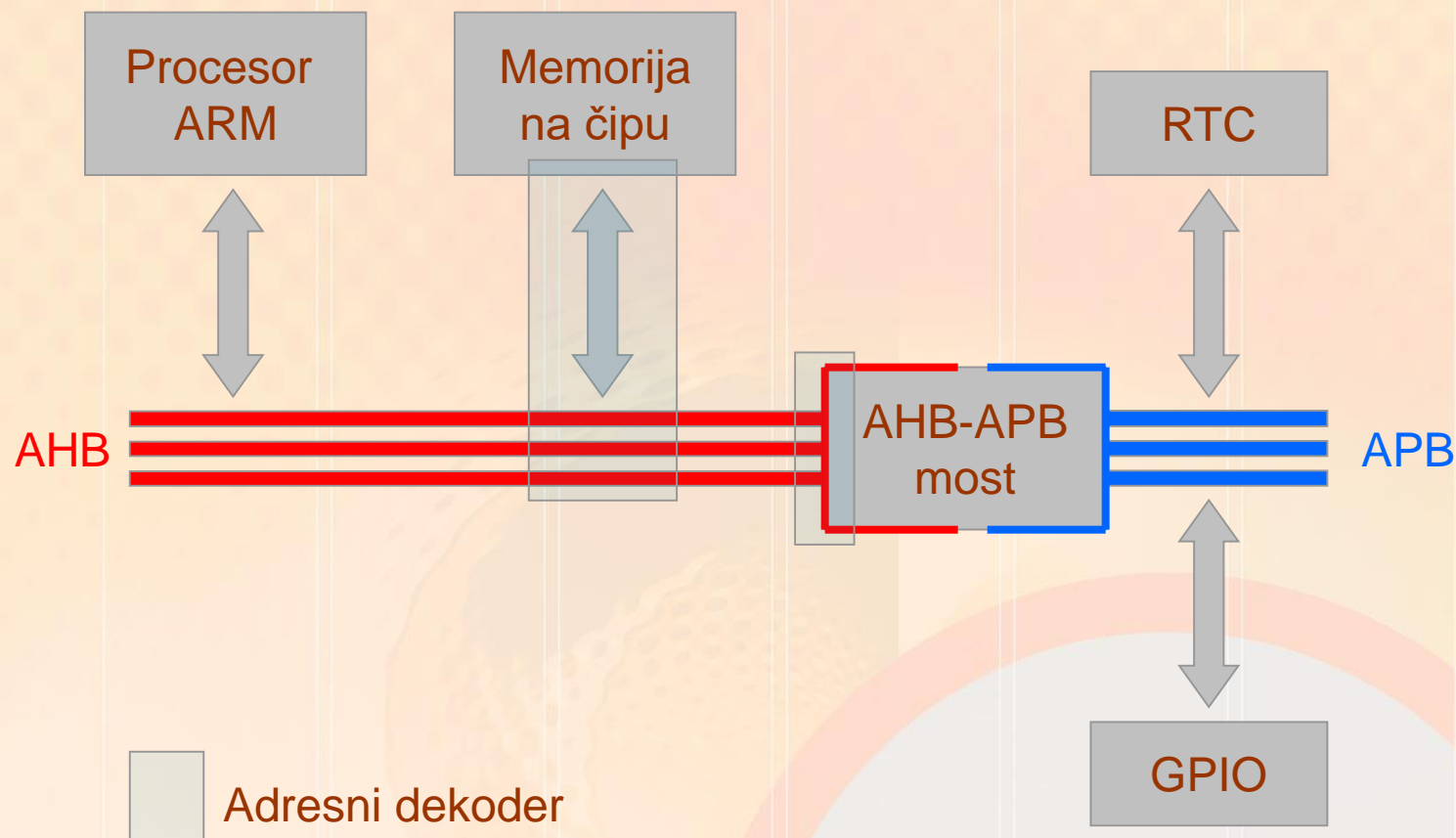
AMBA

- **AMBA AHB** je sabirnica namijenjena za sustave visokih performansi i visokih frekvencija signala vremenskog upravljanja i koristi se isključivo kao brza središnja (memorijska) sabirnica. AHB omogućuje efikasno povezivanje procesora, memorije koja se nalazi na čipu kao i vanjske memorije.
- **AMBA APB** je sabirnica za povezivanje vanjskih uređaja u sustav. Karakteristike ove sabirnice su mala potrošnja i jednostavnija izvedba u usporedbi sa središnjim sabirnicama, s ciljem što jednostavnijeg povezivanja vanjskih jedinica u cjelovit sustav. Na APB sabirnicu se povezuju vanjski uređaji koji ne zahtijevaju visoke performanse i nemaju kompleksna sučelja. Primjeri takvih uređaja su serijski kontroler (UART), LCD-kontroler, vremenski sklop (RTC) i paralelni sklop (GPIO).

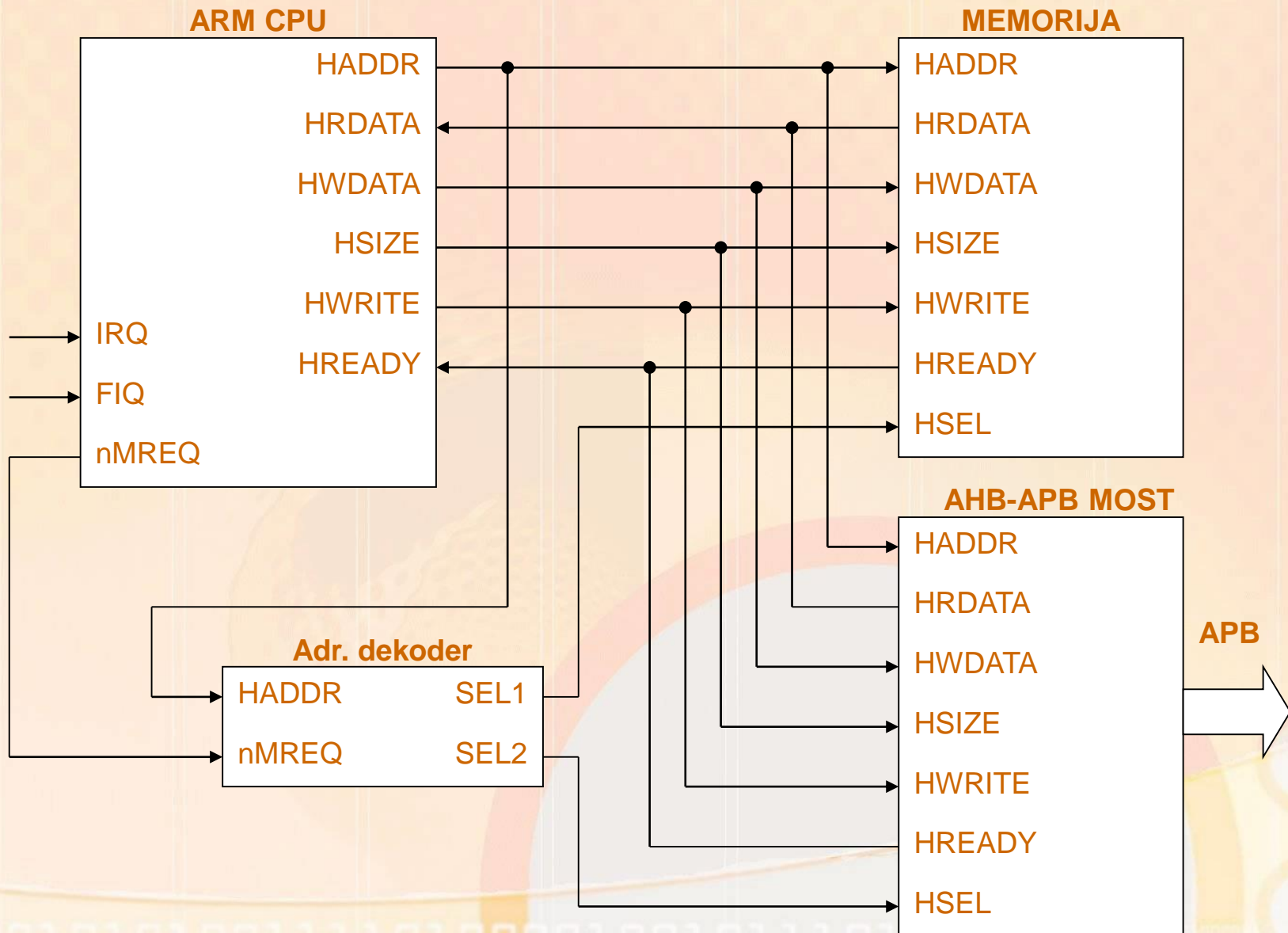
ARM sustav



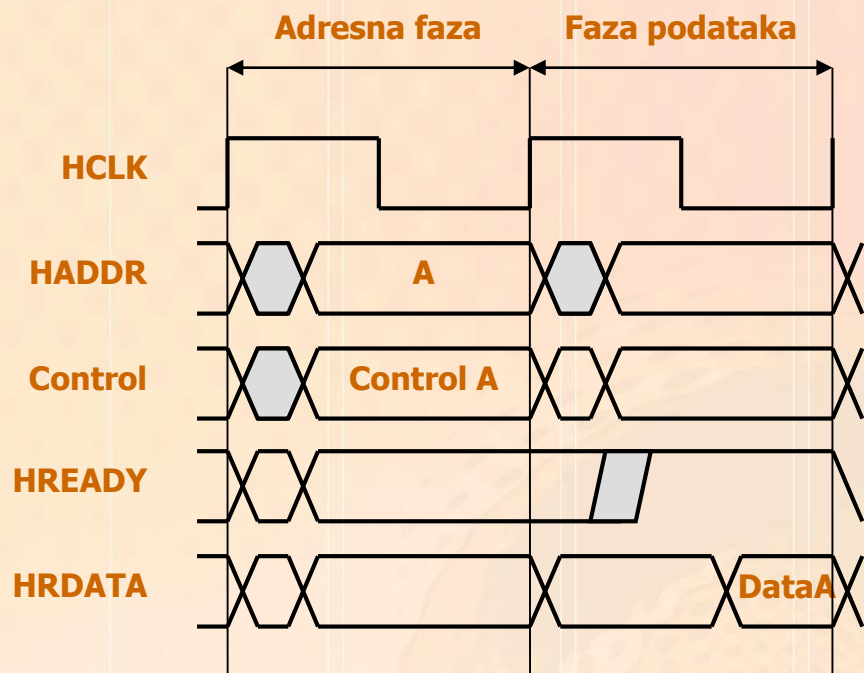
... i naš sustav u ATLAS-u



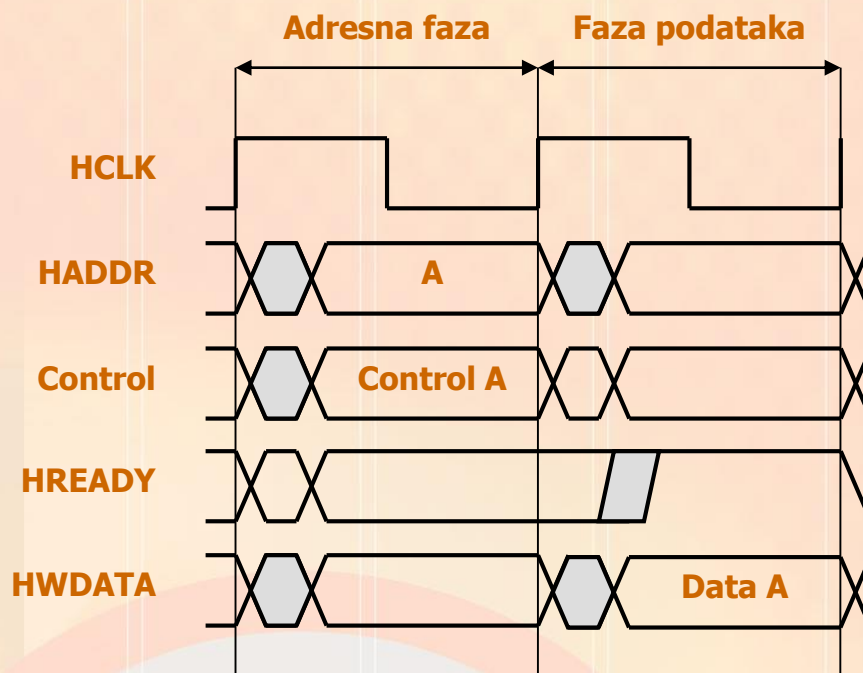
Središnji sustav sa sabirnicom AHB



Adresna i podatkovna faza na AHB

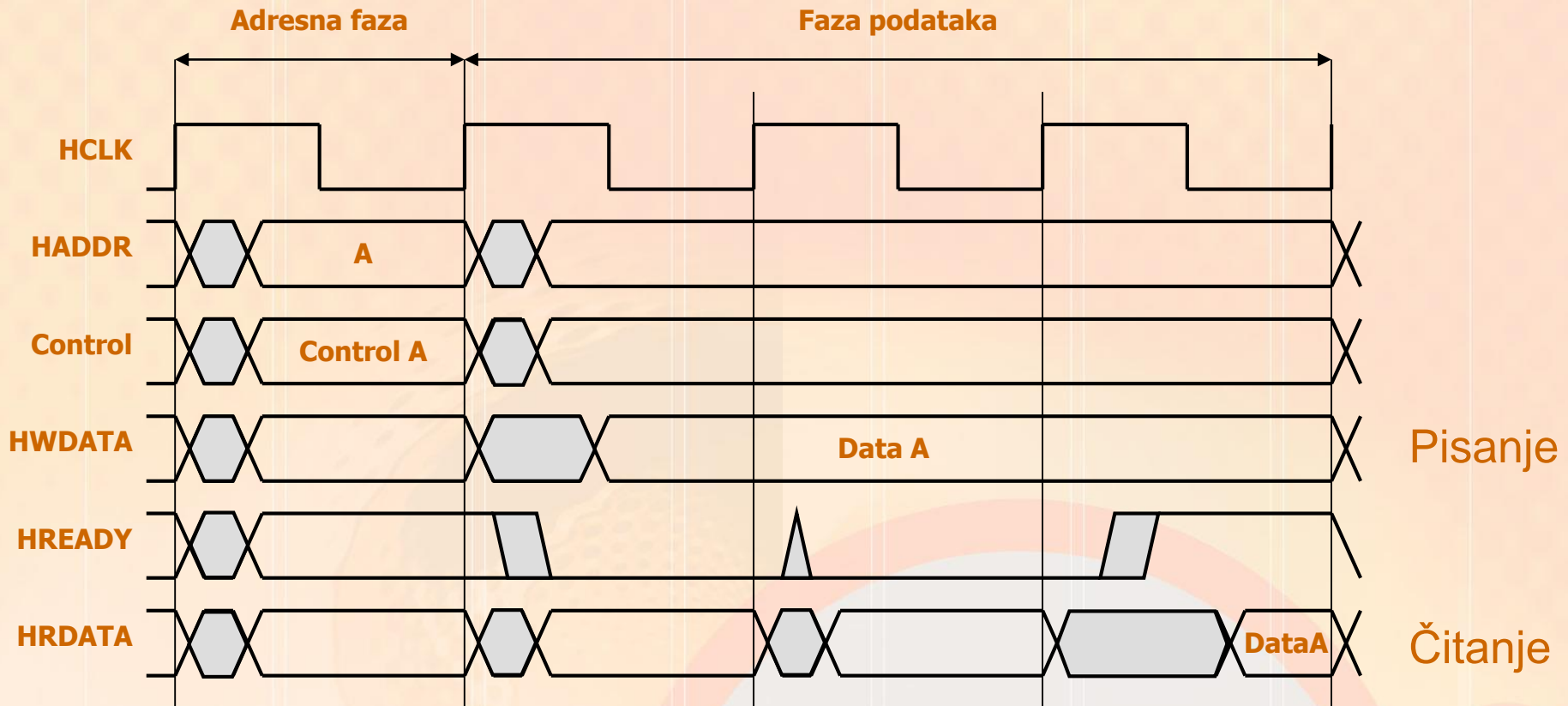


Čitanje podatka



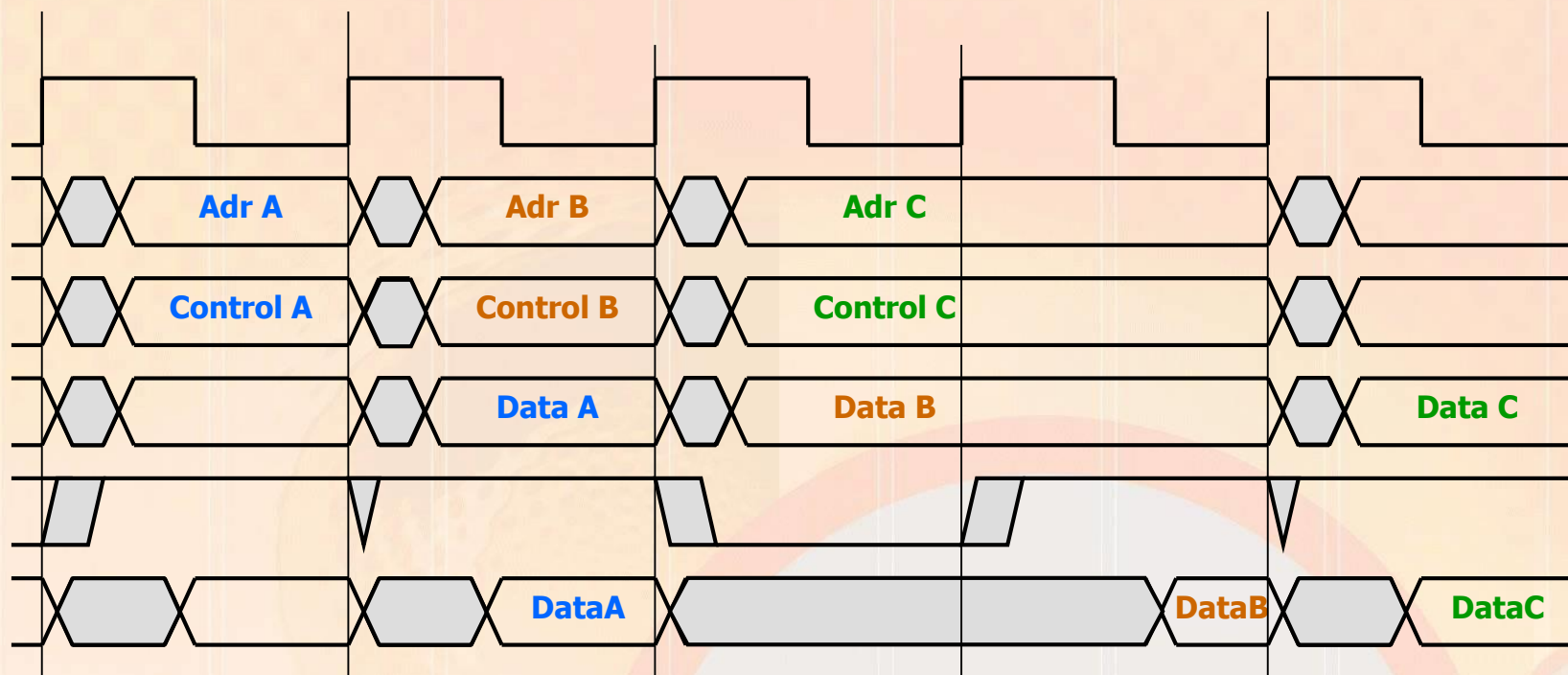
Pisanje podatka

Pristup memoriji sa stanjem čekanja na AHB



Preklapanje adresne i podatkovne faze na AHB

- Kod sabirnice AHB postoji **preklapanje** između adresne faze jednog pristupa i podatkovne faze prethodnog pristupa* čime se ubrzava komunikacija

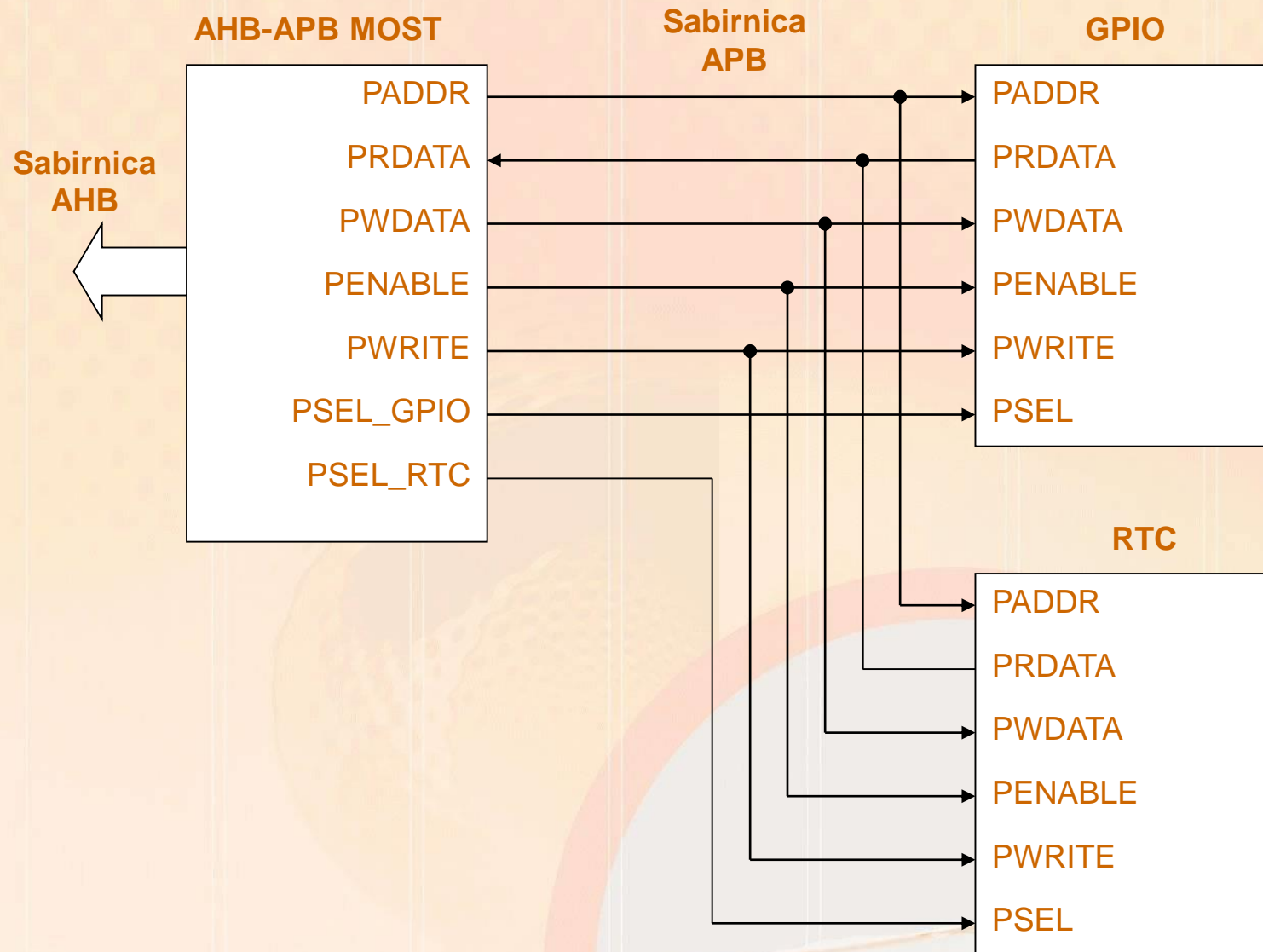


* slično ideji preklapanja faza u protočnoj strukturi

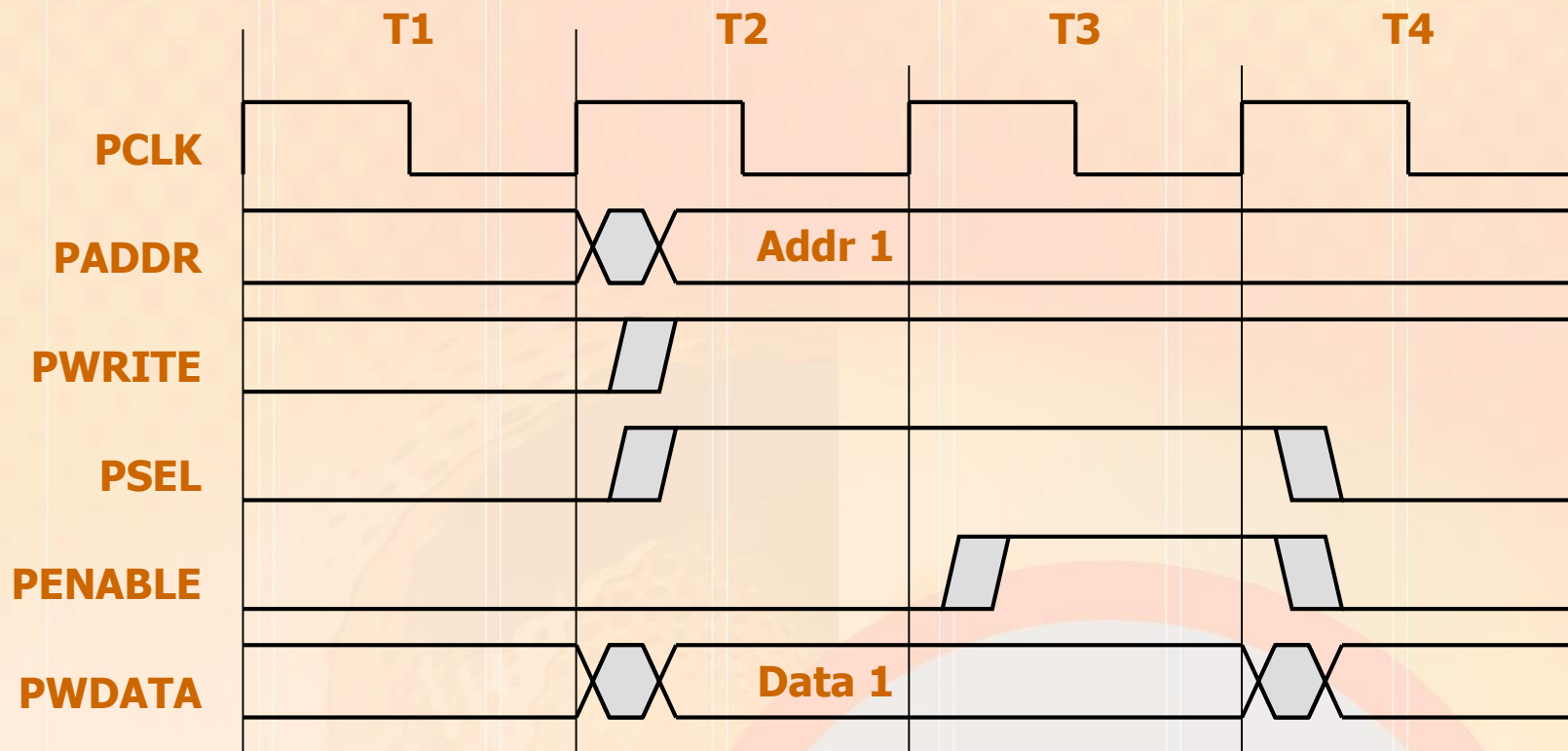
Sabirnički periodi na AHB

nMREQ	SEQ	Tip perioda	Opis
0	0	N-period	Neslijedni period (engl. Nonsequential)
0	1	S-period	Slijedni period (engl. Sequential)
1	0	I-period	Interni period (engl. Internal)
1	1	C-period	Period prijenosa sadržaja koprocesorskog registra (engl. Coprocessor register transfer)

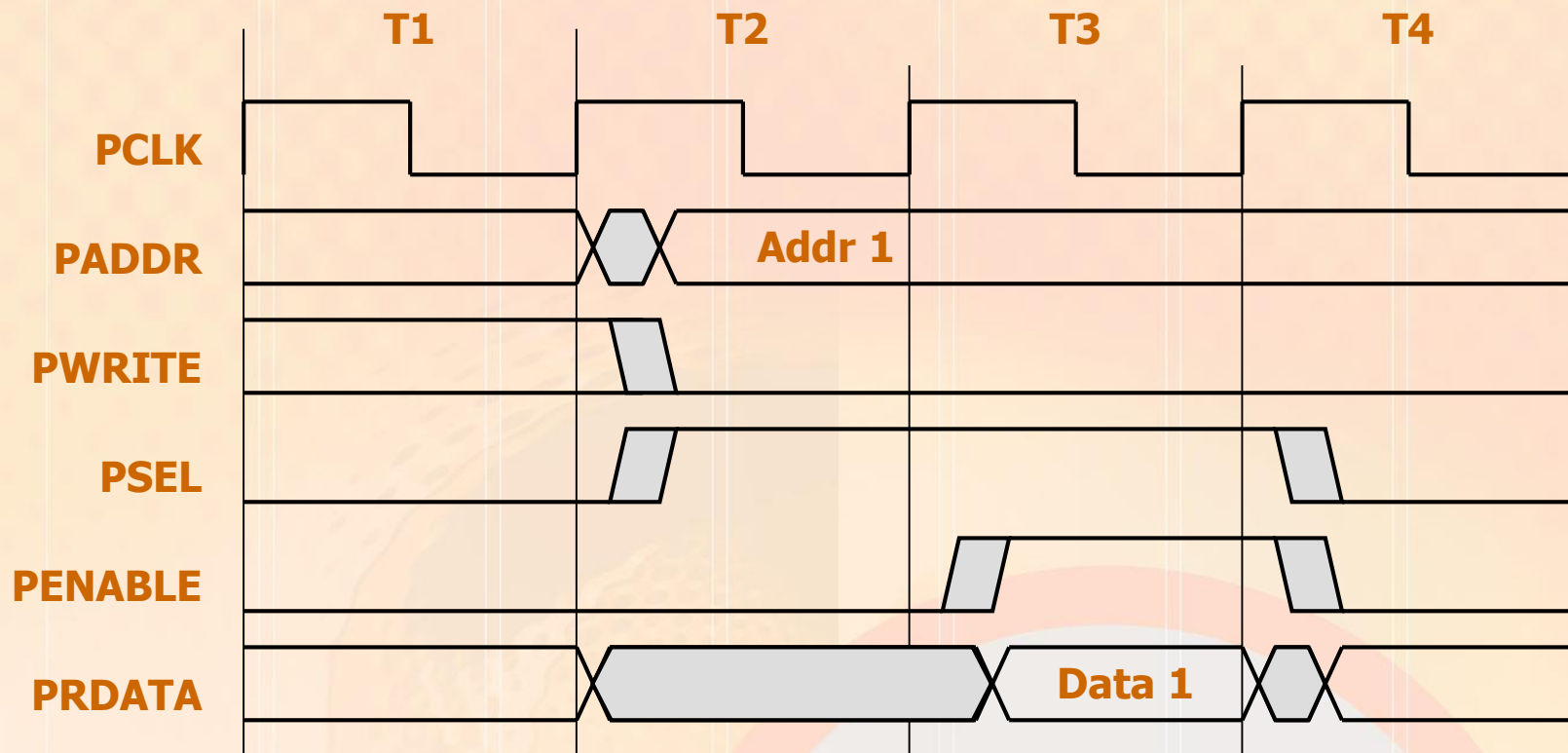
Sabirnica APB



Period pisanja na APB



Period čitanja na APB



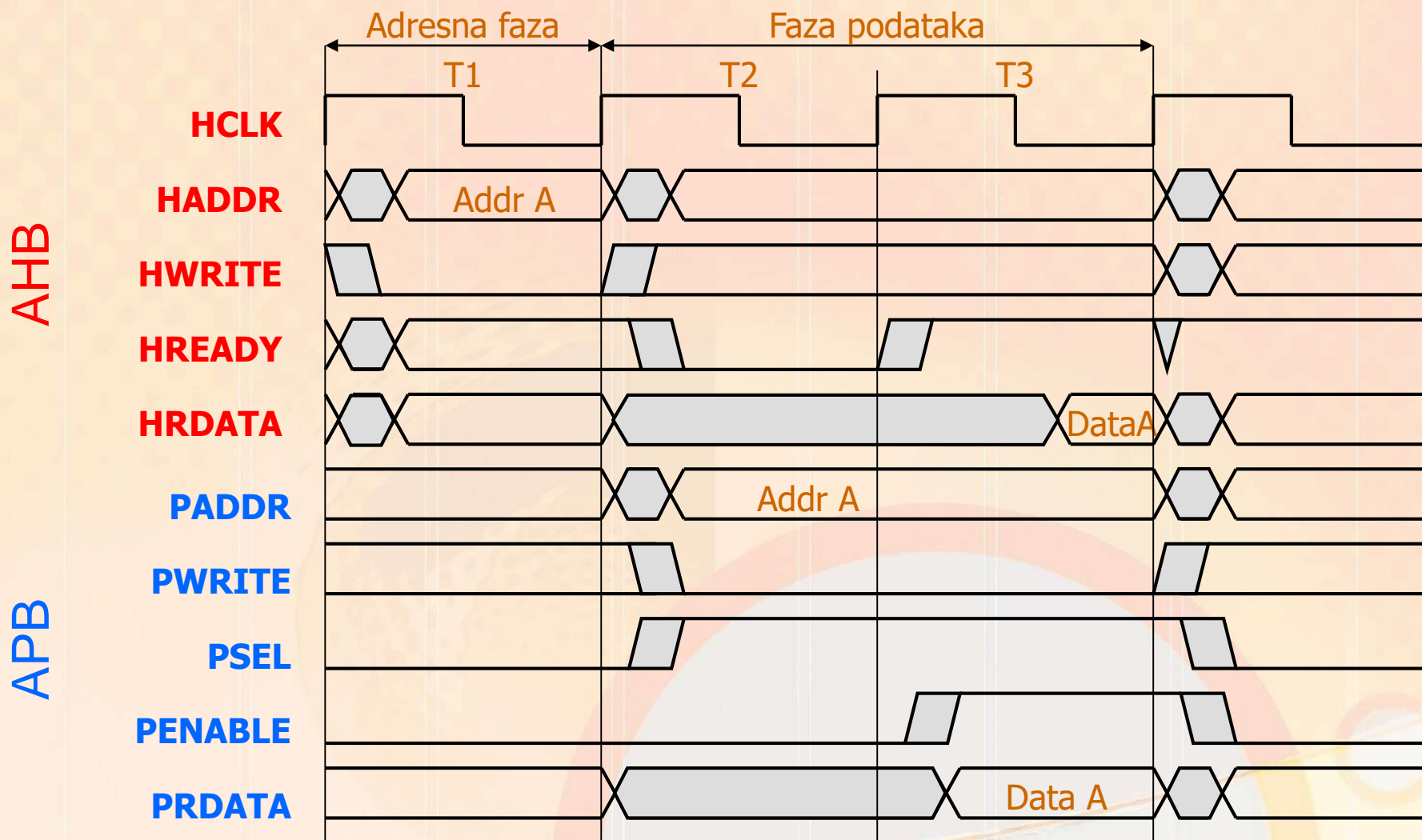
AHB APB most

- Sabirnice AHB i APB predviđene su za različite tipove sklopova:
 - AHB je središnja, brza sabirnica,
 - APB je jednostavnija sabirnica namijenjena vanjskim sklopovima koji su obično sporiji.
- Most AHB-APB je sklop koji omogućuje povezivanje ovih sabirnica i uređaja na njima u cjelovit sustav te prijenos podataka između uređaja sa sabirnicama AHB i APB.

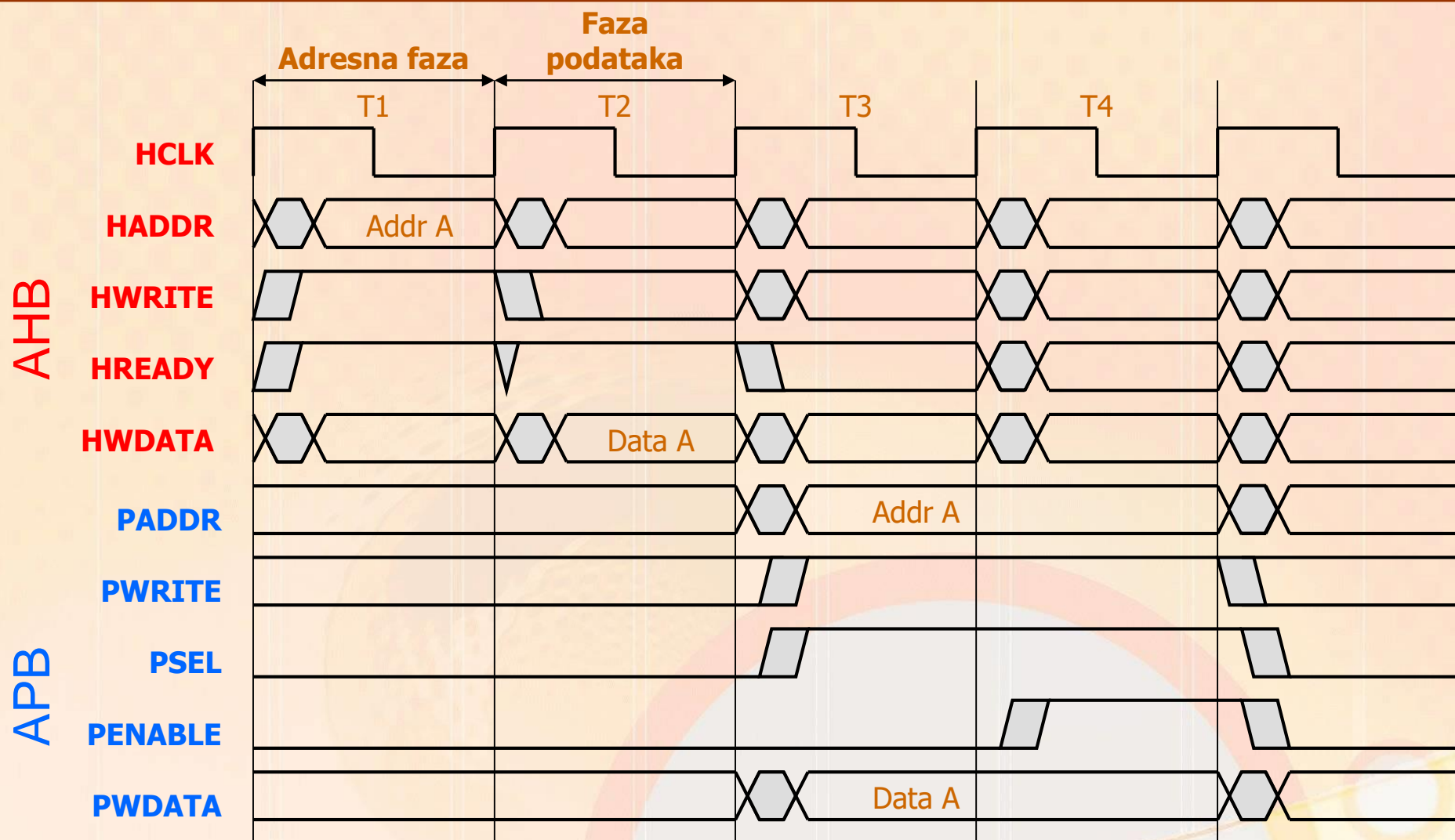
AHB APB most

- Funkcije mosta su sljedeće:
 - uzima adresu i održava je valjanom tijekom cijelog prijenosa
 - dekodira adresu i generira signal PSEL_x kojim se izabire jedna od vanjskih jedinica kojom se izvodi prijenos podataka
 - postavlja podatke na sabirnicu APB kod perioda pisanja
 - postavlja podatke sa sabirnice APB na sabirnicu AHB tijekom perioda čitanja
 - generira vremenski signal PENABLE kojim se omogućuje prijenos

Čitanje s vanjske jedinice



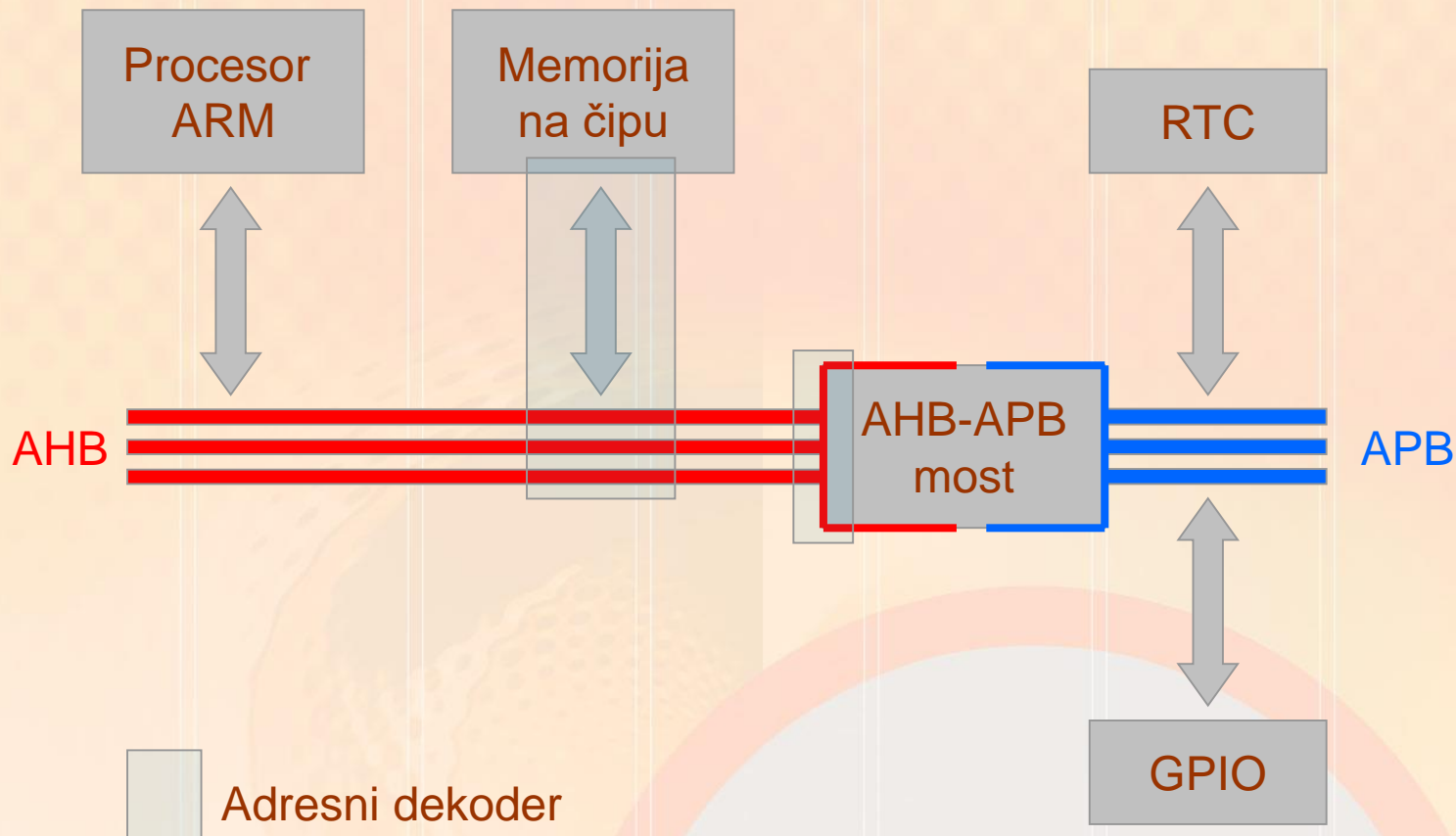
Pisanje na vanjsku jedinicu



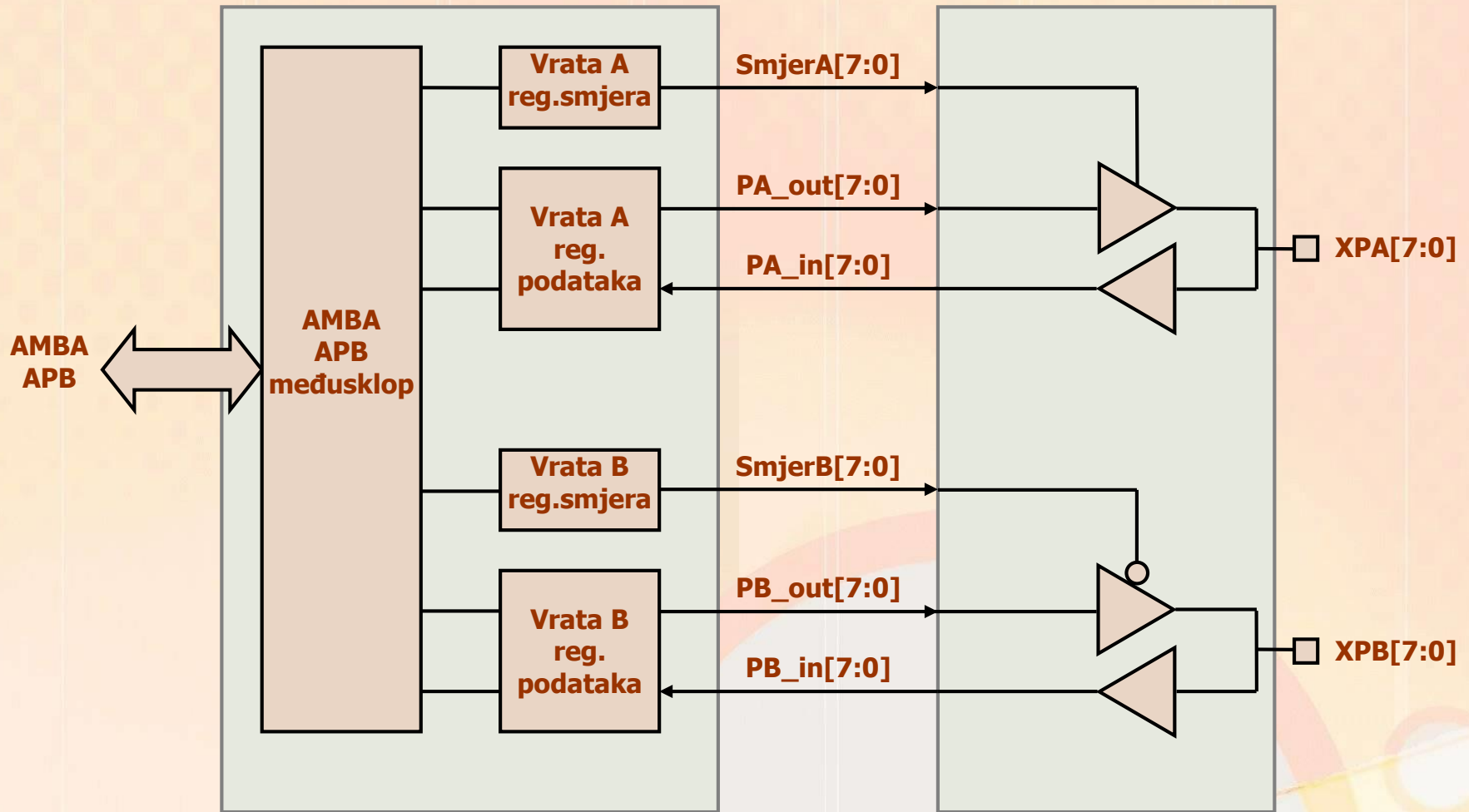
Vanjske jedinice za procesor ARM

GPIO i RTC

Naš ARM-sustav u ATLAS-u



Sklop GPIO (General Purpose Input Output)



Registri sklopa GPIO

Adresa	Naziv registra	Opis
GPIO_bazna_adr	GPIOPADR	8-bitni registar podataka, vrata A
GPIO_bazna_adr + 4	GPIOPBDR	8-bitni registar podataka, vrata B
GPIO_bazna_adr + 8	GPIOPADDR	8-bitni registar smjera podataka za vrata A
GPIO_bazna_adr + C	GPIOPBDDR	8-bitni registar smjera podataka za vrata B

Registri GPIO

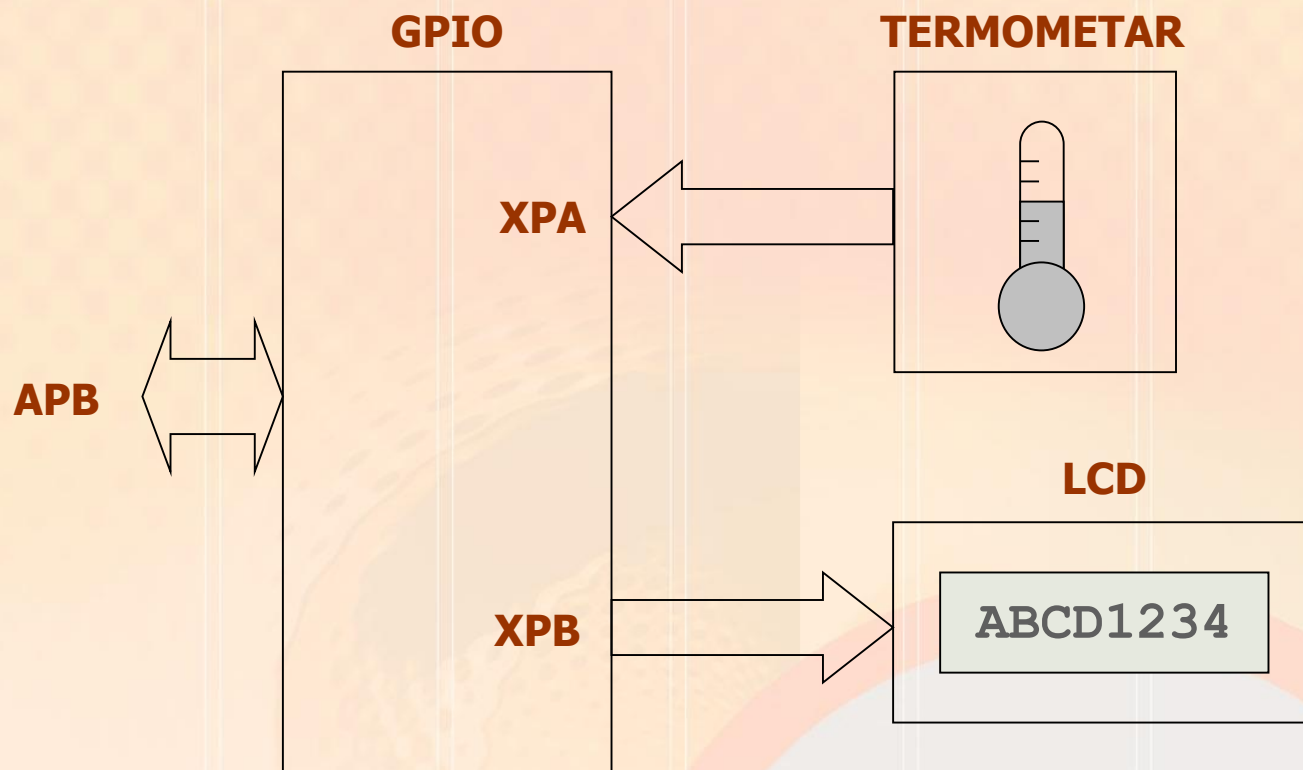
- GPIOPADR (GPIO Port A Data Register)
 - 8-bitni registar podataka za vrata A.
 - Podatci upisani u ovaj registar postavljeni su na izlazne priključke ako je pripadni bit u registru smjera podataka (GPIOPADDR) postavljen u logičku jedinicu.
 - Čitanje ovog registra vraća trenutno stanje priključaka koji su programirani kao ulazni. Za priključke programirane kao izlazne, vraća se vrijednost zadnjeg podatka upisanog u ovaj registar.
- GPIOPBDR (GPIO Port B Data Register)
 - 8-bitni registar podataka za vrata B.
 - Podatci upisani u ovaj registar postavljeni su na izlazne priključke ako je pripadni bit u registru smjera podataka (GPIOPBDDR) postavljen u logiku nulu.
 - Čitanje ovog registra vraća trenutno stanje priključaka koji su programirani kao ulazni. Za priključke programirane kao izlazne, vraća se vrijednost zadnjeg podatka upisanog u ovaj registar.

Registri GPIO

- GPIOPADDR (GPIO Port A Data Direction Register)
 - 8-bitni registar smjera podataka za vrata A. Vrijednost bita u ovom registru zadaje smjer odgovarajućeg priključka na vratima A:
 - 0 – ulazni smjer
 - 1 – izlazni smjer
- GPIOPBDDR (GPIO Port B Data Direction Register)
 - 8-bitni registar smjera podataka za vrata B. Vrijednost bita u ovom registru zadaje smjer odgovarajućeg priključka na vratima B:
 - 0 – izlazni smjer
 - 1 – ulazni smjer
- Početna vrijednost registara
 - Svi registri unutar GPIO-a nakon inicijalizacije (resetiranja) se postavljaju u logičku nulu. Ovime se inicijalno vrata A postavljaju kao ulazna, a vrata B kao izlazna. Sadržaji obaju registara podataka su nula.

← obrnuto od vrata A

Vanjski uređaji



Temperaturni sklop

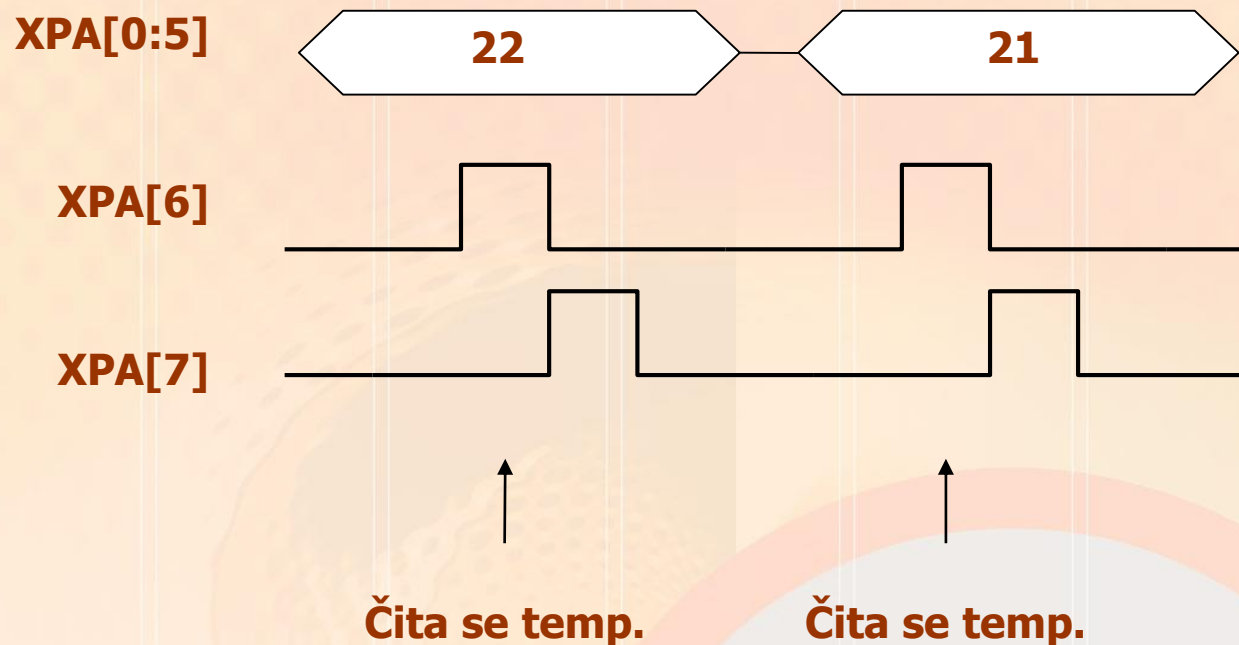
- Temperaturni sklop je jednostavna vanjska jedinica koja omogućuje mjerenje i očitavanje vanjske temperature. Komunikacija s temperaturnim sklopom objašnjena je na primjeru kad je on spojen na vrata A sklopa GPIO.
- Temperaturni sklop periodički mjeri temperaturu i izmjerenu vrijednost u formatu 6-bitnog cijelog broja postavlja na ulaz GPIO i to na priključke XPA[0] do XPA[5].
- Čitanjem vrata A može se pročitati trenutna vrijednost temperature (u nižih 6 bitova).

Temperaturni sklop

- Dva najviša bita (povezana na GPIO-priključke XPA[6] i XPA[7]) služe za sinkronizaciju*. Bitovi 6 i 7 su aktivni u visokoj razini.
- Bit 6 ulazi u GPIO i pomoću njega temperaturni sklop signalizira da je izmjerio temperaturu i postavio njezinu vrijednost na nižih šest bitova. Kad procesor ustanovi da je stanje na bitu 6 u jedinici, može očitati temperaturu s nižih šest bitova.
- Nakon što je pročitao podatke, procesor pomoću izlaznog bita 7 dojavljuje temperaturnom sklopu da je vrijednost temperature pročitana i da treba izbrisati bit 6, sve do trenutka dok temperaturni sklop ne postavi novu vrijednost temperature te ponovo ne aktivira bit 6.

* GPIO nema priključke za sinkronizaciju pa je treba obaviti programski

Vremenski dijagram očitavanja temperature



Primjer: Čitanje temperature

Temperaturni sklop spojen je na vrata B sklopa GPIO. Na najniži bit vrata A spojena je tipka koja daje pozitivni impuls kad je pritisnuta. GPIO je na adresi FFFF1000(16).

Napisati program koji očitava temperaturu svaki puta kad se pritisne tipka i sprema je u memoriju na adresu 1000(16). Nakon 100(16) očitavanja zaustaviti procesor.

ORG 0

GLAVNI

LDR R1, GPIO ; R1 = GPIO bazna adresa

; inicijalizacija GPIO

MOV R0, #0 ; smjer vrata A, bit 0 je ulazni (tipka)

STR R0, [R1, #08]

MOV R0, #1 ; smjer vrata B, bit 7 je izlazni, ostali su ulazni

STR R0, [R1, #0C]

; priprema varijabli za glavnu petlju

MOV R0, #100 ; brojač očitavanja

MOV R4, #1000 ; adresa za spremanje temperatura

Primjer: Čitanje temperature

PETLJA

LDR R2, [R1, #0]	; glavna petlja
ANDS R2, R2, #0x00000001	; čitaj stanje tipke (najniži bit na portu A)
BEQ PETLJA	; ako nije pritisnuta, čekaj

CEKAJ_TEMP

LDR R2, [R1, #4]	; čekaj spremnost temperaturnog sklopa
ANDS R2, R2, #0x01000000	
BEQ CAKAJ_TEMP	

CITAJ_TEMP

LDR R2, [R1, #4]	; čitaj temperaturu (nižih 6 bitova)
AND R2, R2, #0x00111111	
STRB R2, [R4], #1	; spremi temperaturu u memoriju

ORR R2, R2, #0x10000000	; postavi bit 7 u jedan
-------------------------	-------------------------

STR R2, [R1, #4]	
AND R2, R2, #0x01111111	; postavi bit 7 u nulu
STR R2, [R1, #4]	

SUBS R0, R0, #1	; smanji brojač za 100 očitavanja
-----------------	-----------------------------------

BHI PETLJA	
SWI 123456	

GPIO	DW	0xFFFF1000	; GPIO bazna adresa
------	----	------------	---------------------

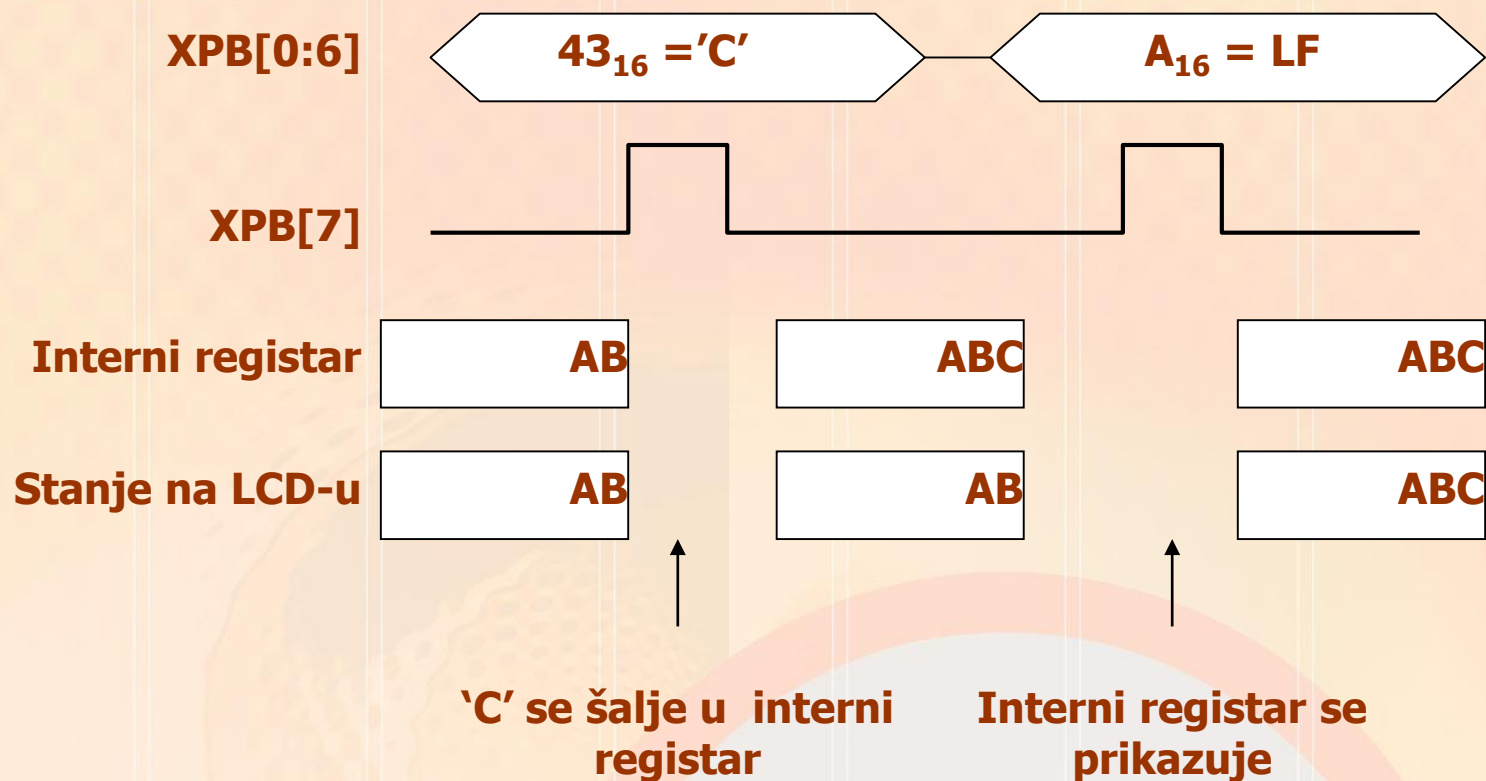
LCD

- LCD-prikaznik je izlazna vanjska jedinica koja omogućuje ispisivanje do osam ASCII-znakova na zaslonu LCD-a. LCD je jednostavna jedinica preko koje se rezultati operacija izvedenih u procesoru mogu prikazati korisniku (ispis znakova tijekom simulacije bit će prikazan u prozoru simulatora ATLAS).
- Komunikacija s LCD-om objašnjena je na primjeru kad je on spojen na vrata B sklopa GPIO.
- Sabirnica LCD-a je 8-bitna i organizirana tako da se nižih sedam bitova (bitovi 0 do 6) koriste za podatke
- Najviši bit (bit 7) je upravljački bit (WR) kojim se podatci upisuju u interni registar LCD-a.
- Registar LCD-a pamti do osam ASCII-znakova.

Način ispisa znakova na LCD

- Dok je bit WR neaktivan (XPB[7] u logičkoj nuli), na bitove XPB[0:6] treba postaviti ASCII-kôd znaka koji se želi ispisati.
- Kad se postavi vrijednost znaka, na bit WR treba poslati pozitivan impuls (stanje mu se iz nule postavi na jedinicu i ponovo vrati u nulu). Impuls uzrokuje upisivanje znaka (postavljenog na XPB[0:6]) na krajnje desno mjesto u interni registar LCD-a.
- Svi ostali do tada ispisani znakovi pomiču se za jedno mjesto ulijevo, a prethodni krajnje lijevi znak se gubi.
- Da bi se znakovi pohranjeni u internom registru prikazali na zaslonu LCD-a, programer treba poslati specijalan ASCII-znak LF (line feed) s kôdom 0A(16).
- Kada LCD primi znak LF, on ne biva spremljen u interni registar, već uzrokuje ispis svih znakova iz internog registra na zaslon LCD-a.
- Postoji još jedan specijalan znak, ASCII-znak CR (carriage return) s kôdom 0D(16), koji briše sadržaj internog registra (na svih osam mjesta upisuje se ASCII kôd 20(16) odnosno znak praznine ili razmaka).

Vremenski dijagram ispisa na LCD



Primjer: Ispis teksta 'FER' na LCD

LCD je spojen na vrata B sklopa GPIO. GPIO je na adresi FFFFFFF00(16). Treba ispisati tekst "FER". Slanje pojedinog znaka na vrata B treba riješiti potprogramom. Potprogram preko R0 prima ASCII znak, a preko R2 baznu adresu sklopa GPIO.

```
ORG 0
MOV R13,#1<16           ; stog
MOV R1, #1<8            ; R1=100(16), jer se tu nalazi adresa GPIO
LDR R2, [R1]             ; u R2 se učitava bazna adresa GPIO-a

START MOV R0, #0D        ; znak 0xD, briše se interni registar
BL LCDWR                 ; piše se na LCD
MOV R0, #46              ; ASCII kod znaka 'F'
BL LCDWR                 ; piše se na LCD
MOV R0, #45              ; ASCII kod znaka 'E'
BL LCDWR                 ; piše se na LCD
MOV R0, #52              ; ASCII kod znaka 'R'
BL LCDWR                 ; piše se na LCD
MOV R0, #0A              ; znak 0xA, ispis znakova na zaslon
BL LCDWR                 ; piše se na LCD

SWI 123456

ORG 100
DW 0FFFFFFF00           ; adresa sklopa GPIO
```

Primjer: Ispis teksta 'FER' na LCD

; potprogram za ispis jednog znaka na LCD spojen na port B sklopa GPIO

; R0 = ASCII kôd znaka koji treba ispisati

; R2 = bazna adresa sklopa GPIO

LCDWR STMFD R13!, {R0}

AND R0, R0, #7F

; postavi bit 7 u nulu (za svaki slučaj) i pošalji znak

STRB R0, [R2, #4]

ORR R0, R0, #80

; postavi bit 7 u jedan (podigni impuls)

STRB R0, [R2, #4]

AND R0, R0, #7F

; postavi bit 7 u nulu (spusti impuls)

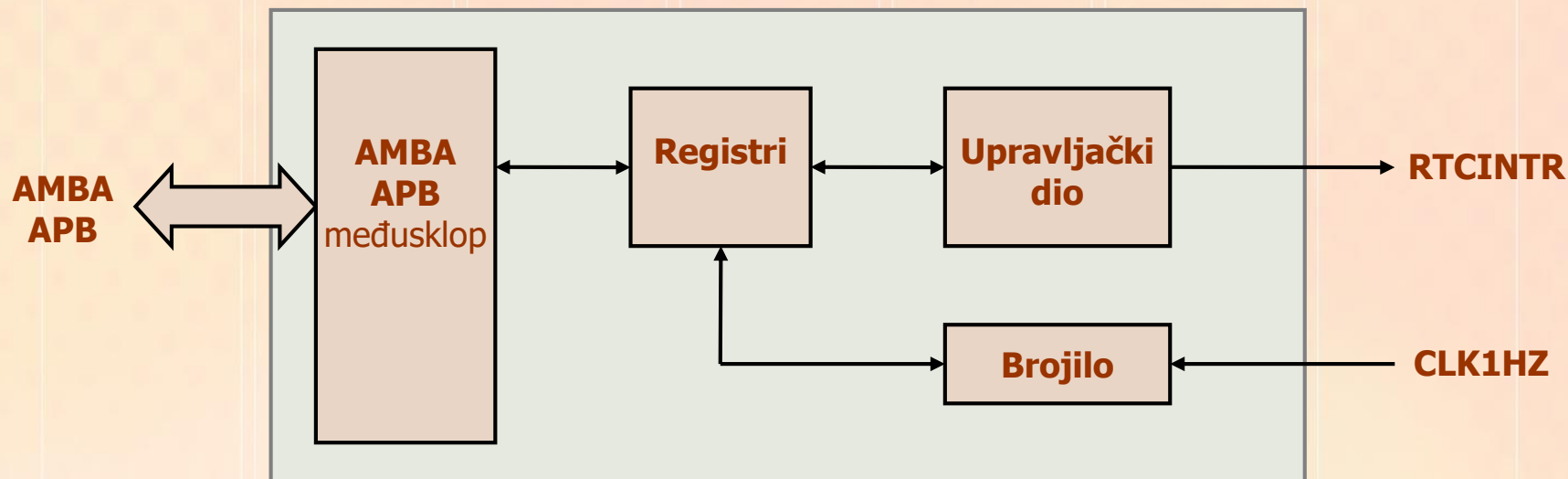
STRB R0, [R2, #4]

LDMFD R13!, {R0}

MOV PC, LR

; povratak

Sklop RTC (Real Time Clock)



Dijelovi arhitekture

- AMBA APB sučelje
 - 32-bitno brojilo
 - 32-bitni registar usporedbe (match register)
 - 32-bitni sklop za usporedbu.
-
- 32-bitno brojilo je glavni dio sklopa RTC. Ovo brojilo uvećava svoj sadržaj za jedan na svaki rastući brid signala CLK1HZ. Ako brojilo dođe do vrijednosti FFFFFFFF, tada se na sljedeći brid vrijednost brojila postavlja na 00000000 i nastavlja dalje normalno brojiti.

Programski model

Adresa	Naziv registra	Opis
RTC_bazna_adr	RTCDR	32-bitni registar podataka (može se samo čitati)
RTC_bazna_adr + 4	RTCMR	32-bitni registar usporedbe
RTC_bazna_adr + 8	RTCSTAT/RTCEOI	1-bitni registar stanja prekida (ako se čita) 0-bitni registar za brisanje prekida (ako se piše)
RTC_bazna_adr + C	RTCLR	32-bitni registar za punjenje brojila
RTC_bazna_adr + 10	RTCCR	1-bitni upravljački registar

RTC: Registri

- RTCDR (Real Time Clock Data Register)
 - RTCDR je 32-bitni registar podataka. Čitanjem ovog registra dobiva se trenutna vrijednost brojila. Pisanje u ovaj registar nije dozvoljeno.
- RTCMR (Real Time Clock Match Register)
 - RTCMR je 32-bitni registar usporedbe. Upisivanjem podatka u ovaj registar postavlja se nova vrijednost koja služi za usporedbu s brojilom. Čitanjem ovog registra dobiva se zadnja vrijednost upisana u registar usporedbe.

RTC: Registri

- RTCSTAT/RTCEOI (Real Time Clock Interrupt STATus Register/Real Time Clock Interrupt Clear Register)
 - RTCSTAT/RTCEOI je virtualni registar bez fizičkog sklopovlja za pohranjivanje podataka. Pisanjem bilo kojeg podatka na ovu adresu čisti se prekidni signal RTCINTR i pripadni registar*. Čitanjem s ove adrese dobiva se podatak koji na bitu 0 (najniži bit) ima trenutачnu vrijednost RTCINTR. Ako je bit 0 postavljen na jedinicu, to znači da je prekidni signal aktivan.
- RTCLR (Real Time Clock Load Register)
 - RTCLR je 32-bitni registar koji služi za upis vrijednosti u brojilo ili čitanje zadnje vrijednosti koja je upisana. Upisana vrijednost se upisuje u brojilo (koje se kasnije mijenja prilikom odbrojavanja), a vrijednost u ovom registru ostaje nepromijenjena.

* interni status-bistabil

RTC: Registri

- RTCCR (Real Time Clock Control Register)
 - RTCCR je 1-bitni upravljački registar kojim programer može omogućiti ili onemogućiti generiranje prekida. Ako se na bit 0 (najniži bit) ovog registra upiše logička nula, tada se RTC-u onemogućuje generiranje prekida. Ako se upiše jedinica, tada se omogućuje generiranje prekida. Čitanjem ovog registra na bitu 0 dobiva se zadnja upisana vrijednost prekidnog bita. Ostali bitovi u ovom registru ne postoje.

RTC: način rada

- Brojilo se povećava na svaki rastući brid na signalu CLK1HZ
- Kad brojilo dosegne vrijednost upisanu u registru usporedbe (match register), onda RTC postaje spreman i postavlja prekid (ako je omogućen u registru RTCCR)
- Brojilo nastavlja s brojanjem i ne vraća se automatski na nulu*
 - Ako se želi ponoviti ciklus brojenja, onda programski treba u brojilo upisati ničticu
- Nakon što postane spreman (ili nakon što postavi prekid) RTC-u treba obrisati stanje (tj. prekid ako ga je postavio)

* Za razliku od FRISC-CT-a u kojem ciklus brojenja automatski započinje iznova

Primjer

Treba napisati program za izmjenično ispisivanje znaka 0 i znaka 1 na zaslonu LCD-a i to tako da se znakovi mijenjaju svakih 5 sekundi. Mjerenje perioda valja riješiti korištenjem RTC-a. Program napisati za ATLAS.

LCD je spojen na vrata B sklopa GPIO (GPIO je na adresi FFFFFFF00).

Na ulaz CLK1HZ od RTC-a je spojen signal frekvencije 1 Hz. RTC je na adresi FFFFE00 i spojen je na IRQ

Primjer

ORG 0
B GLAVNI

ORG 18 ; adresa za obradu iznimke IRQ

PREKIDNI

; druge iznimke se ne koriste pa možemo odmah ovdje napisati cijeli prekidni potprogram
STMFD R13!, {R0, R3, R14}

LDR R3, RTC ; dohvati adresu sklopa RTC

; reinicijaliziraj RTC

STR R0, [R3, #8] ; čisti se bit RTCINTR (dojava prihvata prekida)

MOV R0, #0

STR R0, [R3, #0C] ; vrati brojač na nulu

; MOV R0, #0A

; STR R0, [R3, #4] ; alternativno stavi RTCMR na 10

BL PISI_ZNAK ; ispiši sljedeći znak

LDMFD R13!, {R0, R3, R14}

SUBS PC, R14, #4 ; povratak iz obrade iznimke

Primjer

; potprogram koji dohvaća znak, mijenja ga iz '0' u '1' (ili obratno) te ga ispisuje na LCD

PISI_ZNAK

STMFD R13!, {R0, R2, R14}

LDR R2, GPIO ; u R2 dohvati adresu sklopa GPIO

MOV R0, #0D ; znak 0xD, briše se interni registar

BL LCDWR ; šalje se na LCD

LDR R0, ZNAK ; dohvati znak iz memorije

EOR R0, R0, #1 ; mijenjaj znak '0'<--->'1'

STR R0, ZNAK ; spremi znak natrag u memoriju

BL LCDWR ; šalje se znak '0' ili '1' na LCD

MOV R0, #0A ; znak 0xA, ispis znaka na zaslonu

BL LCDWR ; šalje se na LCD

LDMFD R13!, {R0, R2, R14}

MOV PC, LR

; potprogram za ispis znaka iz R0 na LCD (spojen na vrata B sklopa GPIO na baznoj adresi R2)

LCDWR ... (potprogram kao u primjeru za LCD na slajdu 17)

Primjer

GLAVNI

```
MOV R13, #1<16      ; stog

; inicijalizacija RTC-a
LDR R3, RTC          ; u R3 učitaj adresu RTC-a
MOV R4, #5
STR R4, [R3, #4]      ; napuni RTCMR
MOV R4, #1
STR R4, [R3, #10]     ; omogući prekid (RTCCR)

BL PISI_ZNAK          ; inicijalno pisanje znaka (GPIO ne treba inicijalizirati)

MRS R0, CPSR          ; pročitaj CPSR u R0
BIC R0, R0, #80        ; na mjestu bita I se stavlja nula
MSR CPSR_c, R0        ; i to se ponovo zapisuje u CPSR
```

PETLJA

```
B PETLJA              ; beskonačna petlja
```

```
GPIO    DW  0FFFFFFF00 ; adresa sklopa GPIO
RTC      DW  0FFFFFFE00 ; adresa sklopa RTC
```

```
ZNAK     DW  030        ; trenutni znak za ispis (inicijalno ASCII kod znaka '0')
```

Razni primjeri programa za GPIO+RTC

Primjer 1

Računalni sustav sastoji se od procesora ARM, jedinice GPIO (FFFF0100) i uređaja za ispis računa koji je spojen na port A (8 podatkovnih priključaka) i na port B (izlazni signal SEND (bit 0) i ulazni signal ACK (bit 1)). Uređaj radi tako da u trenutku aktiviranja impulsa na signalu SEND počne ispis 8-bitnog podatka koji je prisutan na podatkovnim priključcima (port A). Dovršetak ispisa svakog 8-bitnog podatka uređaj za ispis dojavljuje procesoru generiranjem impulsa na signalu ACK.

Potrebno je napisati potprogram SEND koji inicijalizira sklop GPIO, te dani niz podataka ispisuje na papirnu traku. Adresa niza prenosi se preko registra R0. Duljina niza nije unaprijed poznata, ali se zna da je niz završen podatkom AA. Adresa GPIO-a zapisana je na fiksnoj memorijskoj lokaciji (na adresi 100)

Glavni program treba korištenjem potprograma SEND poslati na ispis podatke na adresi 200.

Rješenje primjera 1

ORG 0

; glavni program
MOV R13,#1<16

; stog

MOV R0, #2<8
BL SEND
SWI 123456

; parametar za SEND (adresa niza)

; bazna adresa GPIO-a
ORG 100
DW 0FFFF0100

; podaci za ispis
ORG 200
DB 0F0, 14, 12, 05, 0C4, 0AA

Rješenje primjera 1

; potprogram SEND

SEND STMFD R13!, {R0, R1, R2}

MOV R1, #1<8

LDR R2, [R1] ; stavi adresu GPIO-a u R2

; inicijalizacija GPIO-a

; port A: izlazni - 8-bitni podatak

MOV R1, #0xB 11111111

STRB R1, [R2, #8]

; port B: bit 0 izlazni, bit 1 ulazni

MOV R1, #0xB 00000010

STRB R1, [R2, #0C]

Rješenje primjera 1

; petlja za ispis podatka na traku

PISI LDRB R1, [R0], #1 ; dohvat podatka iz niza
CMP R1, #0AA ; provjeri kraj
BEQ KRAJ

STRB R1, [R2]; šalji podatak na port A

; slanje impulsa SEND (port B, bit 0)

MOV R1, #1
STRB R1, [R2, #4]
MOV R1, #0
STRB R1, [R2, #4]

; čekanje impulsa ACK (port B, bit 1)

ACK LDRB R1, [R2, #4]
CMP R1, #0
BEQ ACK
B PISI

; ponovi za sljedeći znak

KRAJ LDMFD R13!, {R0, R1, R2}
MOV PC, LR

Primjer 2

Računalni sustav sastoji se od procesora ARM, sklopa GPIO (FFFF0000) i kontrolne jedinice CTRL koja služi za mjerenje udjela triju sirovina u proizvodnom procesu.

Jedinica CTRL spojena je na GPIO na port A i to sa 4 priključka. Prva tri priključka su ulazni i spojeni su na bitove 0, 1 i 2 (XPA[0]-XPA[2]). CTRL na ovim priključcima daje logičku 1 za svaku od 3 sirovine kada je njezin udio u proizvodnom procesu zadovoljavajući, a inače daje logičku 0. Četvrti priključak je izlazni i spojen je na bit 3 (XPA[3]). Pomoću ovog priključka procesor ARM zaustavlja i pokreće proizvodni proces.

Ako udio bilo koje od sirovina nije zadovoljavajući, potrebno je ugasiti proces slanjem logičke 0 na XPA[3]. Kada udio svih sirovina postane zadovoljavajući, treba uključiti proces slanjem logičke 1 na XPA[3]. Upravljanje procesom ponavljati beskonačno.

Rješenje primjera 2

ORG 0

LDR R1, GPIO

; GPIO Port A inicijalizacija

MOV R0, #0xB 1000

STRB R0, [R1, #8]

PETLJA

; dohvat niža 3 bita podatka s porta A

LDR R0, [R1]

AND R3, R0, #0xB 0111

; ako je udio svih sirovina OK, onda će

; u R2 biti broj 0xB 111

CMP R3, #0xB 111

; stavi 1 na bit 4 od R0,

; tj. uključi proces

MOVEQ R0, #0xB 1000

STREQB R0, [R1]

BEQ PETLJA

; stavi 0 na bit 4 od R0,

; tj. isključi proces

MOV R0, #0xB 0000

STRB R0, [R1]

B PETLJA

; bazna adresa GPIO-a

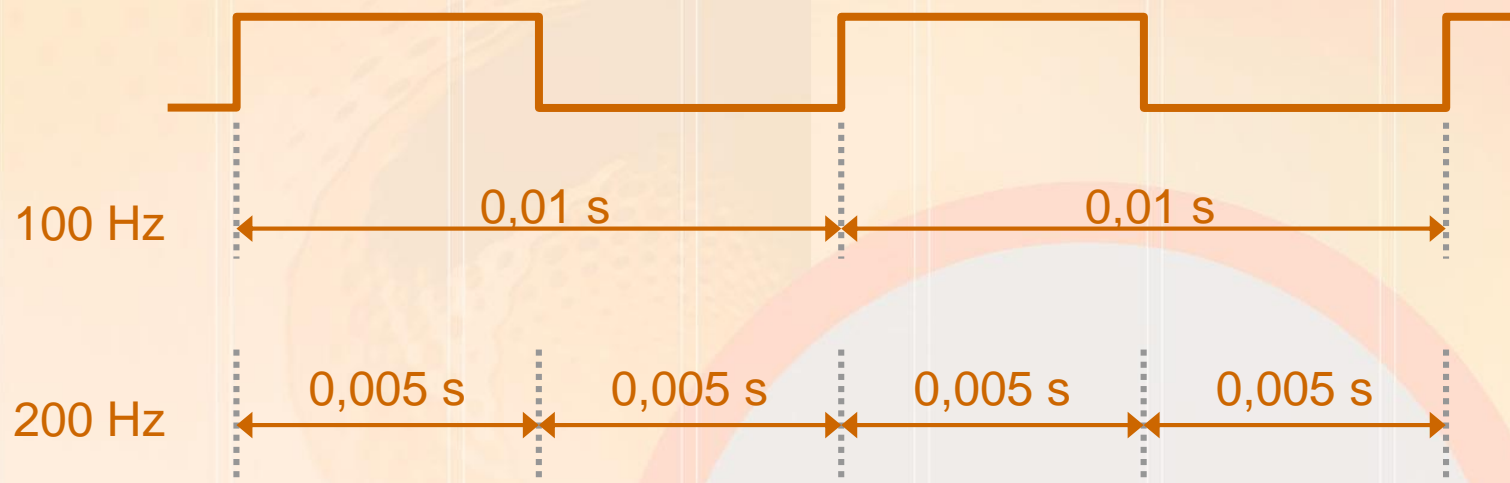
GPIO DW 0xFFFF0000

Komentar: uočite da za komunikaciju sa GPIO-om možemo koristiti naredbe load/store i za riječi i za bajtove

Primjer 3

U računalnom sustavu nalaze se ARM, sklop RTC (na adresi FFFF0000) i sklop GPIO (na adresi FFFF1000). RTC je spojen na FIQ.

Potrebno je na sklopu GPIO, na priključku 0 porta A (XPA[0]) generirati pravokutni signal frekvencije 100 Hz. Na ulaz sklopa RTC doveden je signal frekvencije 10 kHz.



Rješenje primjera 3

ORG 0
B GLAVNI

ORG 1C ; Prekidni potprogram za FIQ

; ne spremamo kontekst jer koristimo registre od načina rada FIQ

MOV R8, #1<8
LDR R9, [R8], #4 ; R9 = RTC bazna adresa
LDR R10, [R8] ; R10 = GPIO bazna adresa

LDRB R8, [R10] ; promijeni stanje
EOR R8, R8, #1 ; pravokutnog signala na
STRB R8, [R10] ; GPIO-u

STR R8, [R9, #8] ; prihvati prekida RTC-a

MOV R8, #0 ; reinicijalizacija RTC-a
STR R8, [R9, #0C]

SUBS PC, LR, #4

Rješenje primjera 3

GLAVNI

MOV R13, #81<8 ; inicijalizacija SP-a

MOV R0, #1<8

LDR R1, [R0], #4 ; R1 = RTC bazna adresa

LDR R2, [R0] ; R2 = GPIO bazna adresa

; inicijalizacija GPIO port A - XPA[0] je izlazni

MOV R0, #0B 00000001

STRB R0, [R2, #8]

; inicijalizacija RTC-a

MOV R0, #0D 50 ; konstanta brojenja

STR R0, [R1, #4] ; RTCMR = 50 = 10 kHz / 200Hz = 10000 / 200

; enable interrupt u RTC-u

MOV R0, #1

STR R0, [R1, #10]

MOV R0, #0

; briši brojilo u RTC-u

STR R0, [R1, #0C] ; RTCLR = 0

Rješenje primjera 3

; omogućavanje prekida FIQ

MRS R0, CPSR

BIC R0, R0, #40

MSR CPSR_c, R0

PETLJA B PETLJA

; bazne adrese RTC-a i GPIO-a

ORG 100

DW 0FFFF0000 ; RTC

DW 0FFFF1000 ; GPIO

Primjer 4

U računalnom sustavu nalaze se ARM, GPIO (FFFF1000) i RTC (FFFF0000).

GPIO preko vrata A prima 8-bitni NBC podatak koji predstavlja temperaturu nekog procesa (u Celsijevim stupnjevima). Pinovi 0, 1 i 2 od vrata B služe za signalizaciju - pomoću njih se pale i gase lampice: crvena (XPB[0]), žuta (XPB[1]) i zelena (XPB[2]). Lampice se pale slanjem logičke 1, a gase slanjem logičke 0 na odgovarajući priključak.

Ako je temperatura veća od 128°C, treba upaliti crvenu lampicu, a ugasiti žutu i zelenu. Isto treba učiniti sa žutom lampicom, ako je temperatura u rasponu od 55°C do 128°C, odnosno sa zelenom ako je temperatura manja od 55°C.

Svake sekunde potrebno je mjeriti temperaturu i obaviti navedenu signalizaciju. Na RTC je spojen signal frekvencije 100 Hz.

Rješenje primjera 4

ORG 0
B GLAVNI

ORG 18 ; adresa prekidnog potprograma
B PREKIDNI

GLAVNI

MOV R13, # 1<16 ; inicijalizacija SP-a
; GPIO je inicijalno dobro postavljen
; inicijalizacija RTC-a
LDR R2, RTC ; R2 = bazna adresa RTC-a
MOV R3, #0
STR R3, [R2, #0C] ; RTCLR - brojilo
MOV R3, #0xD 100
STR R3, [R2, #4] ; RTCMR - konstanta
MOV R3, #1
STR R3, [R2, #10] ; RTCCR - prekidanje
; omogućiti prekid IRQ
MRS R0, CPSR
BIC R0, R0, #80
MSR CPSR_c, R0

PETLJA B PETLJA

GPIO DW 0FFFF1000 ; bazna adresa GPIO-a
RTC DW 0FFFF0000 ; bazna adresa RTC-a

Rješenje primjera 4

PREKIDNI

STMFD R13!, {R2, R4, R7}

LDR R2, RTC ; R2 = RTC bazna adresa
STR R7, [R2, #8] ; brisanje int-zastavice u RTC-u

MOV R7, #0
STR R7, [R2, #0C] ; RTCLR - brisanje brojila

; dohvati temperaturu sa porta A

LDR R2, GPIO ; R2 = bazna adresa GPIO-a
LDR R4, [R2]

; ispitaj u kojem je opsegu temperatura

CMP R4, #0xD 128
MOVHI R7, #0xB 001 ; CRVENA = bit 0
BHI SIG

CMP R4, #0xD 55
MOVLO R7, #0xB 100 ; ZELENA = bit 2
BLO SIG

MOV R7, #0xB 010 ; ŽUTA = bit 1

SIG STR R7, [R2, #4] ; signalizacija lampicama

LDMFD R13!, {R2, R4, R7}
SUBS PC, LR, #4

