Programski model procesora Arm

AArch32

- **U okviru AR1R mi ćemo objašnjavati 32-bitnu Arm arhitekturu i skup naredaba (AArch32)**

- Novi procesori zasnovani na AArch64 podržavaju kompatibilnost s AArch32

(Za praćenje predavanja ili ponavljanja obavezno pored sebe otvorite službeni šalabahter za naredbe Arm koji možete pronaći na WEB stranicama predmeta)

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
- Protočna struktura
  - Visoka efikasnost (o tome ćemo govoriti kasnije tijekom semestra)
- Memorijska arhitektura
  - Von Neumannova ili Harvardska arhitektura (različito za pojedine jezgre)
- Modularna arhitektura
  - Namijenjena za SoC (System on Chip)
- Mala potrošnja

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 za CISC/RISC
- Naredbe za obradu podataka s 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
- "Thumb" skup naredaba
  - Za sustave kojima je važna minimizacija memorije

Registri

- Programer na raspolaganju ima 16 32-bitnih registara opće namjene R0-R15

- Registar R15 procesor koristi kao PC

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15=PC

Arhitektura v1 ☺

Aritmetičko-logičke naredbe

- Naredbe koje postoje u svim procesorima su aritmetičko-logičke naredbe
- Osnovne operacije koje su uvijek potrebne su:
  - zbrajanje - ADD
  - oduzimanje - SUB
  - logički I na bitovima - AND
  - logički ILI na bitovima - OR
  - logički EKSKLUZIVNI ILI na bitovima - XOR

Aritmetičko-logičke naredbe

- Za svaku od ovih operacija imat ćemo jednu naredbu, a za svaku naredbu moramo zadati operande

- Budući da procesor Arm ima arhitekturu load-store (tj. registarsko-registarsku arhitekturu), svi operandi i rezultat nalazit će se u općim registrima

- Opći oblik naredaba izgledat će ovako:

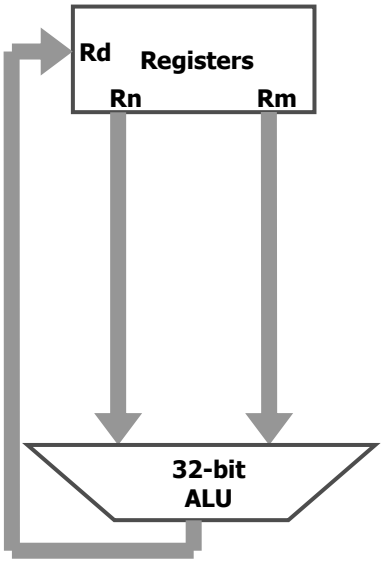
  naredba rezultat, operand_1, operand_2

- izvedi operaciju između operanda_1 i operanda_2 i stavi rezultat u rezultat

- Primjer naredbe:

  ADD R1, R2, R3 R1=R2+R3

- odredišni registar zadaje se na početku a nakon toga operandi (kao kod normalnog pisanja jednadžbi)



- Rn – prvi operand
- Rm – drugi operand
- Rd – rezultat

- Aritmetičko logička jedinica (ALU) je širine 32bita jer su i osnovni podaci u procesoru te širine (32-bitni procesor !!)

Aritmetičko logičke naredbe

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

Prvi program

- U registrima R5 i R7 nalaze se dva 32-bitna broja. Treba ih zbrojiti i rezultat spremiti u R11.

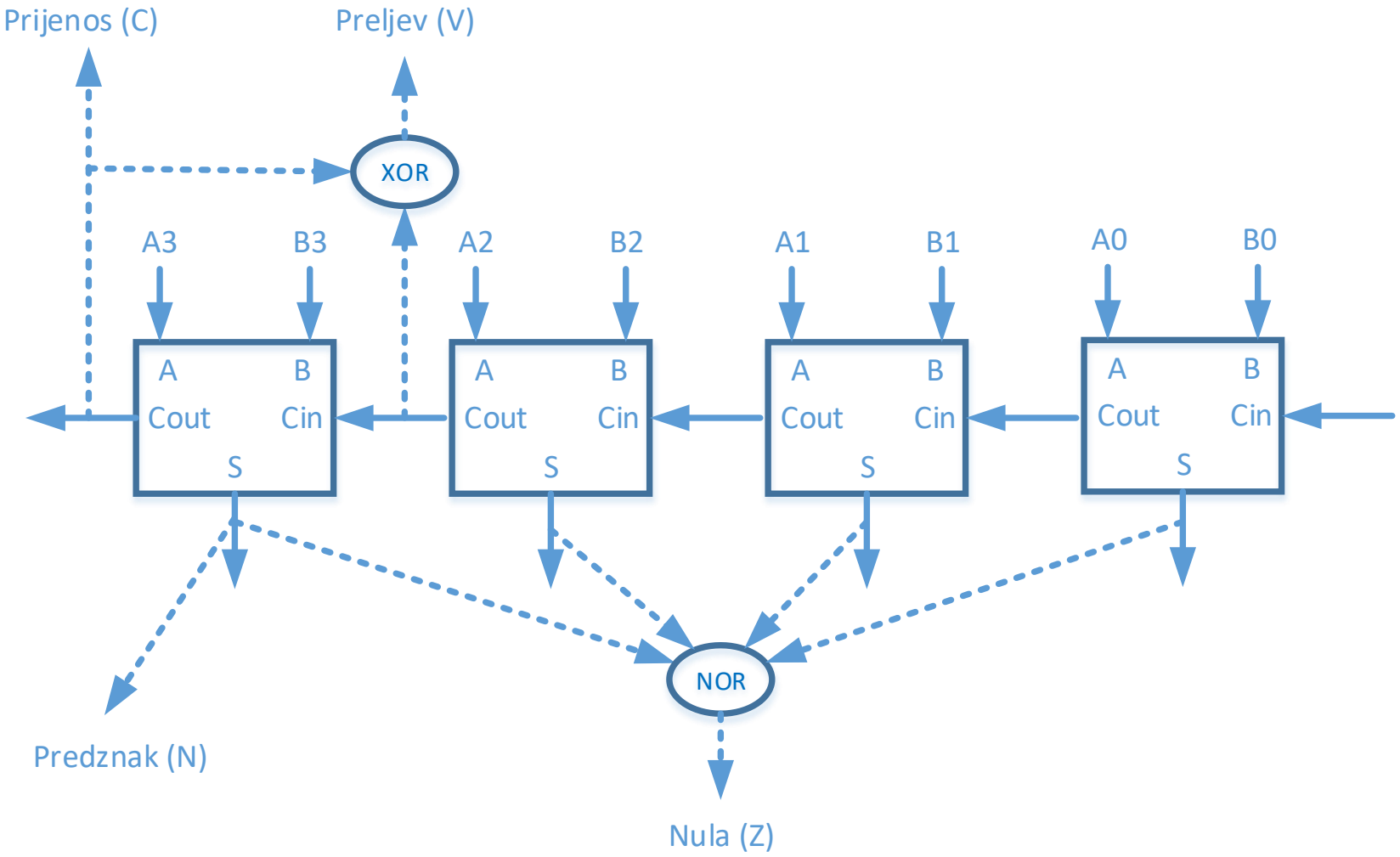
```
ADD R11,R5,R7
```

Drugi program

- Treba napisati program za zbrajanje dva 64-bitna broja. Prvi broj nalazi se u registrima R4(nižih 32 bita) i R5 (viših 32 bita). Drugi broj nalazi se u registrima R6 i R7. Sumu treba spremiti u registre R10 i R11.

- Rješenje:
  - Da bi mogli zbrojiti dva 64 bitna broja koristeći 32 bitno zbrajalo moramo koristiti bit prijenosa (C)
  - Zastavice !!!

Zastavice podsjetnik



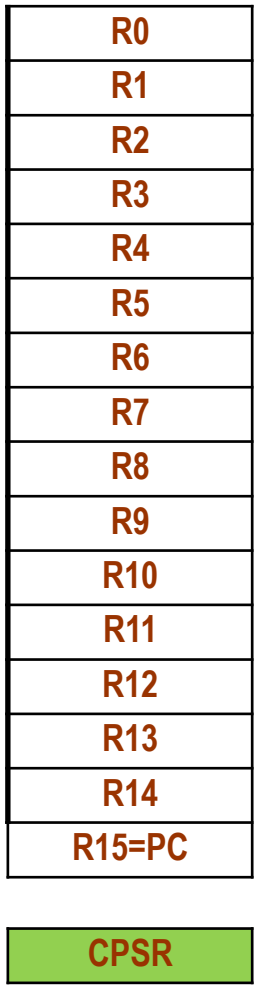
Gdje se nalaze zastavice?

- Zastavice se nikada ne nalaze u procesoru kao zasebni bistabili, nego su uvijek unutar registra koji čuva i druge zastavice ili neke status podatke (npr. prekidne zastavice, zastavice koje označuju stanje procesora i sl. - više o tome kasnije)
- Procesor Arm ima izvedene zastavice unutar posebnog registra koji se naziva CPSR (Current Program Status Register)
- CPSR
- Najviša četiri bita u CPSR predstavljaju zastavice:
  - N (Negative) - negativna vrijednost, predznak
  - Z (Zero) - nula,
  - C (Carry) - prijenos,
  - V (oVerflow) - preljev.

31	30	29	28		....	7	6	5	4	3	2	1	0
N	Z	C	V										

Gdje se nalazi CPSR?

- CPSR je poseban registar (status registar) koji se nalazi unutar skupa registara ali ne pripada registrima opće namjene



Utjecaj aritmetičko-logičkih naredaba na zastavice

- Artimetičke operacije postavljaju sve 4 zastavice:
  - ADD, SUB,…: N,Z,C,V

- Logičke operacije nad bitovima postavljaju samo neke:
  - AND,ORR,EOR,..: N,Z (Zastavica C se postavlja na temelju izlaza iz sklopa za pomak – o tome kasnije, Zastavica V se ne mijenja)

- Detaljan opis imate u službenom šalabahteru pa ne morate pamtiti

TABLICA NAREDABA PROCESORA ARM (za AR1R)

	Naziv	Asembler	Operacija	N	Z	C	V
Obrada podataka	**Aritmetičke naredbe**						
	Add	ADD{cond}{S} Rd, Rn, <oprnd2>	Rd := Rn + oprnd2	n	z	c	v
	Add with carry	ADC{cond}{S} Rd, Rn, <oprnd2>	Rd := Rn + oprnd2 + C				
	Subtract	SUB{cond}{S} Rd, Rn, <oprnd2>	Rd := Rn - oprnd2				
	Subtract with carry	SBC{cond}{S} Rd, Rn, <oprnd2>	Rd := Rn - oprnd2 - not(C)				
	Reverse subtract	RSB{cond}{S} Rd, Rn, <oprnd2>	Rd := oprnd2 - Rn				
	Reverse subtract with carry	RSC{cond}{S} Rd, Rn, <oprnd2>	Rd := oprnd2 - Rn - not(C)				
	Logičke naredbe						
	Bitwise AND	AND{cond}{S} Rd, Rn, <oprnd2>	Rd := Rn AND oprnd2	n	z	x	-
	Bitwise inclusive OR	ORR{cond}{S} Rd, Rn, <oprnd2>	Rd := Rn OR oprnd2				
	Bitwise Exclusive OR	EOR{cond}{S} Rd, Rn, <oprnd2>	Rd := Rn XOR oprnd2				
	Bit clear	BIC{cond}{S} Rd, Rn, <oprnd2>	Rd := Rn AND NOT(oprnd2)				
	Ispitivanje vrijednosti						
	Compare	CMP{cond} Rn, <oprnd2>	Rn - oprnd2 i osvježi CPSR	n	z	c	v
	Compare negative	CMN{cond} Rn, <oprnd2>	Rn + oprnd2 i osvježi CPSR				
	Test	TST{cond} Rn, <oprnd2>	Rn and oprnd2 i osvježi CPSR	n	z	x	-
	Test equivalence	TEQ{cond} Rn, <oprnd2>	Rn xor oprnd2 i osvježi CPSR				

Rješenje zadatka


```
ADDS R10, R4, R6
ADC  R11, R5, R7
```

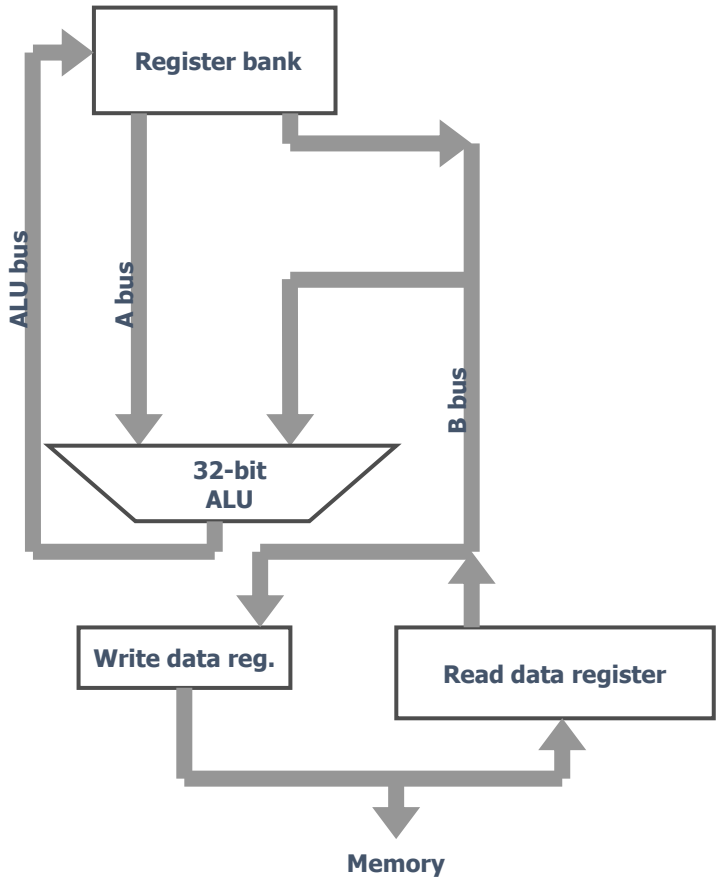
ADDS - nastavak ‘S’ u naredbi kaže procesoru da spremi zastavice koje su rezultat operacije. U primjeru naredbe zbrajanja Arm će spremiti sve 4 zastavice u CPSR

ADC - naredba zbraja dva operanda i njima zbroji vrijednost zastavice C

Naredbe load i store

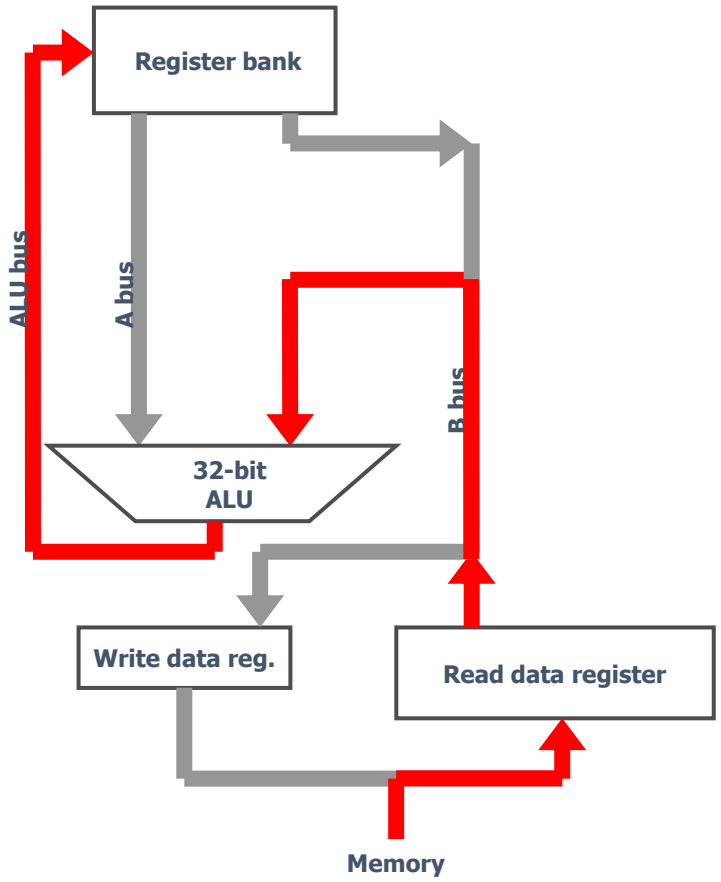
- Među najvažnijim naredbama za svaki procesor, pa tako i Arm, su naredbe za prijenos podataka između procesora i memorije (naredbe load/store).
- Programer ima na raspolaganju naredbe za prijenos podataka iz memorije u registar (load register, LDR) i iz registra u memoriju (store register, STR) u nekoliko različitih verzija.
- Ovo su jedine naredbe Arm-a kojima se može pristupiti podatku iz memorije !!!

Kako se dohvaćaju podatci iz memorije

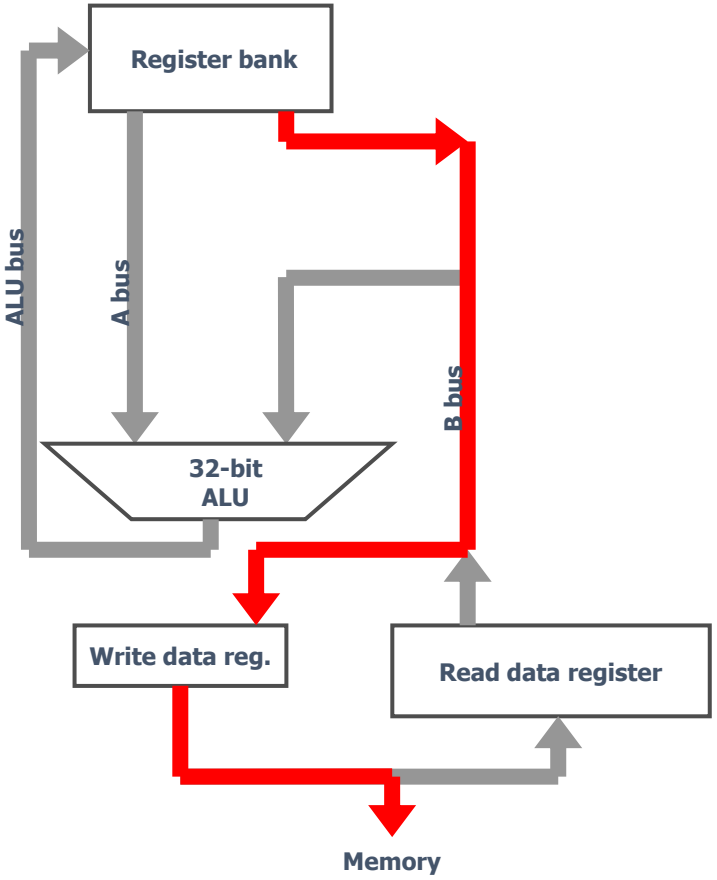


LDR

- Za sada nećemo gledati kako se adresira memorija



STR


Register bank
ALU bus
A bus
B bus
32-bit
ALU
Write data reg.
Read data register
Memory

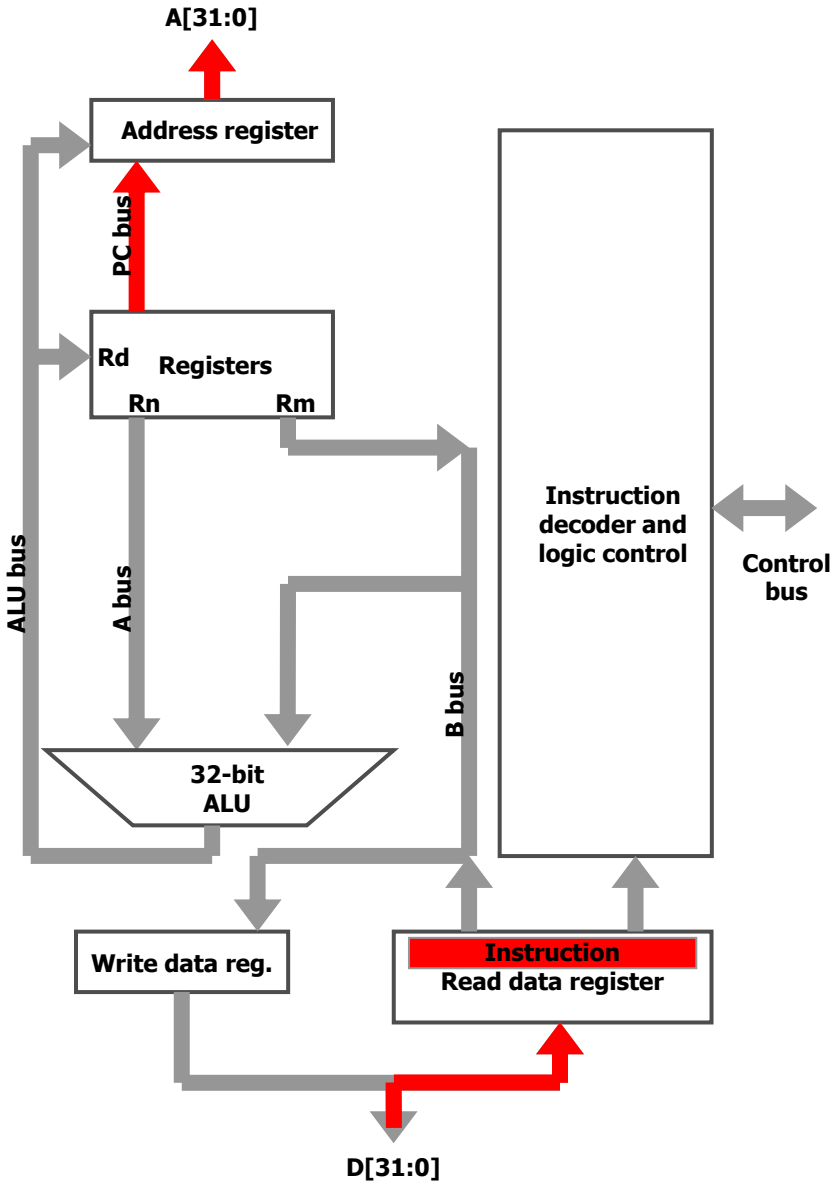
Primjer

- Napraviti logičku ekskluzivno_ili operaciju na podacima iz memorije koji su na adresama 100 i 200. Rezultat spremiti na adresu 300

```
LDR R0, 100
LDR R1, 200
EOR R0, R0, R1
STR R0, 300
```

Dohvat naredbe (osnovno)

- Objasnili smo neke najjednostavnije naredbe i put podataka za njih
- Prije objašnjenja novih naredaba proučimo kako processor Arm dohvaća naredbe
- Podsjetnik:
  - PC
  - Kod Arma PC=R15


A[31:0]
Address register
PC bus
Rd
Registers
Rn
Rm
ALU bus
A bus
B bus
32-bit
ALU
Instruction
decoder and
logic control
Control
bus
Write data reg.
Instruction
Read data register
D[31:0]

Drugi operand

- Jedna od najčešćih operacija koje koristimo je da nekoj varijabli trebamo dodati neku konstantu (npr x=x+7)
- Kako bi to izveli s trenutnim naredbama?
- Pretpostavimo da je varijabla x trenutno spremljena u registru R0

```
        LDR R1, 100
        ADD R0,R0,R1

100     DW    7       ; na adresi 100 zapisan je podatak 7
```

Ovo nije idealno rješenje:

- Potrebne su dvije naredbe (brzina i zauzeće memorije)
- Potrebna je dodatna memorijska lokacija (s brojem 7)
- Treba koristiti dodatni registar (npr. R1). Moguće je da on nije slobodan, pa će ga trebati spremiti i kasnije obnoviti (za što trebaju još dvije dodatne naredbe i još jedna dodatna memorijska lokacija)

- Kako to efikasnije izvesti?
- Strojni kod AL naredbe
  - Bitovi za opis operacije
  - Bitovi za izbor registara Rd, Rn, Rm
  - …
- Unutar 32 bita AL naredbe imamo neiskorištenih bitova !!!

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					Rn				Rd				NEISKORIŠTENI BITOVI											

- Ovo možemo iskoristiti da u njih upišemo neku konstantu (neposrednu vrijednost)

```
ADD R0,R0,#7
```

Puno efikasnije rješenje:

- Neposredna vrijednost nalazi se u kodu naredbe
- Ne treba LDR, ne treba dodatni registar

Jedno od pravila pri oblikovanju procesora:

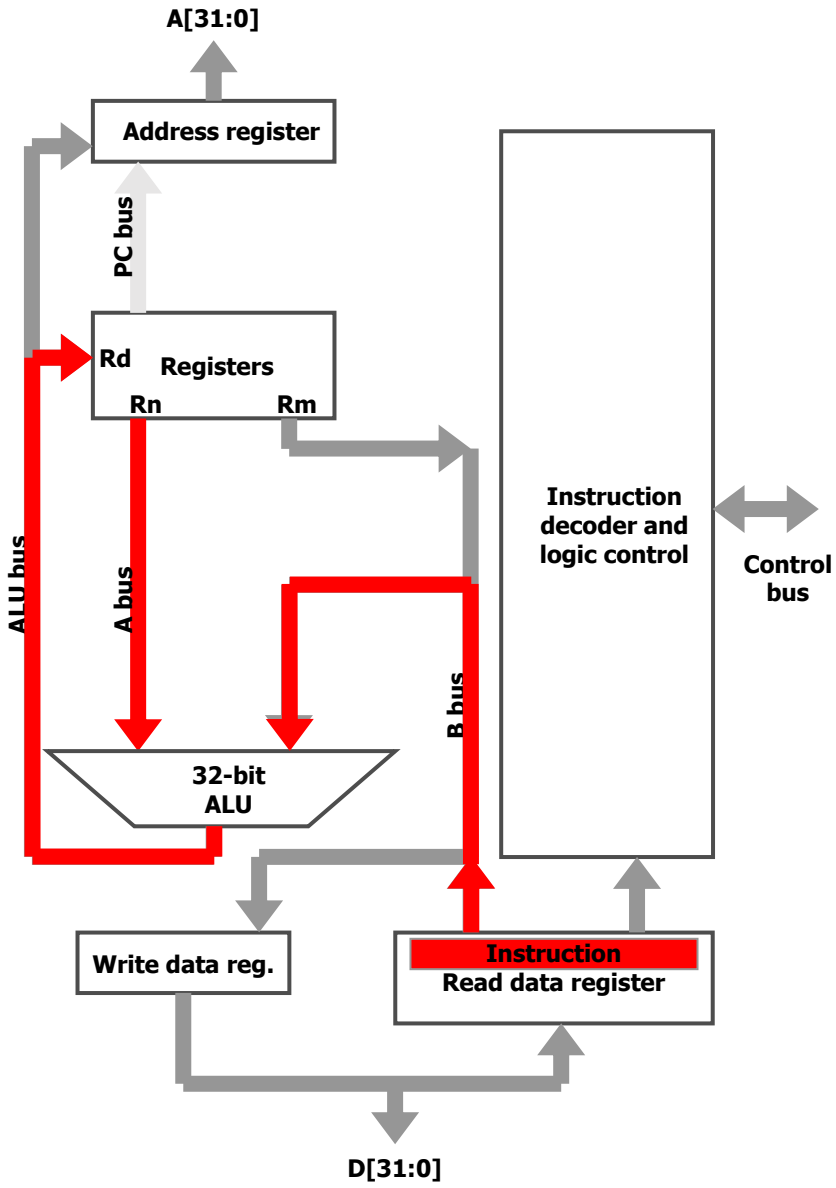
- Ubrzati rad onoga što se često koristi

Nedostatak: logika za izvođenje naredbe je složenija

- Operandi više nisu samo registri, nego mogu biti i brojevi

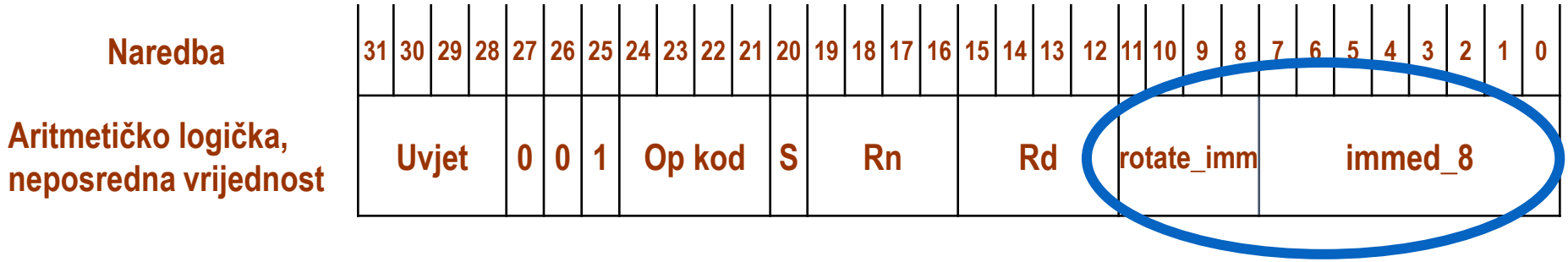
Napomena:

- S obzirom da je samo dio od 32 bita naredbe dostupan opseg brojeva koje možemo direktno zapisati je ograničen što ćemo objasniti kasnije


A[31:0]
Address register
PC bus
Rd
Registers
Rn
Rm
ALU bus
A bus
B bus
32-bit
ALU
Instruction
decoder and
logic control
Control
bus
Write data reg.
Instruction
Read data register
D[31:0]

Neposredna vrijednost #<immediate>

- Specifično za svaki pojedini procesor
- Za zapis konstante u strojnom kôdu AArch32 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:



Definiranje neposredne vrijednosti

- Neposredna 32-bitna vrijednost računa se sljedećom formulom:
  - <immediate> = <immed_8> rotirano udesno za (2 *<rotate_imm>)

- Ovakvim postupkom opseg svih brojeva koji se mogu opisati neposredno, proširen je na sva 32 bita, odnosno [0, 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 većini alata je dovoljno napisati željeni broj, a asemblerski prevoditelj će **sam izračunati** može li se taj broj prikazati na ovaj način.

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 FFFFFF00. No, tu neposrednu vrijednost možemo učitati u registar koristeći naredbu MVN, SUB i slično:

NE!!! MOV R0, #0FFFFFFFF

Da!!! MVN R0, #0

Primjeri dozvoljenih i nedozvoljenih vrijednosti

NE!!! MOV R4, #0FFFFFF0

Da!!! MVN R4, #0F000000F

- 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,#0f000000f

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,#0F000000F	1110				0	0	1	0100				1	0001				0100				0010				11111111							

Neposredna vrijednost

- Prilikom programiranja vi ne morate razmišljati o načinu kodiranja neposredne vrijednosti u naredbi
- Vi samo napišete konstantu a assembler će provjeriti da li se ta konstanta može zapisati u naredbi (I ako ne može javiti će vam grešku)
- Npr ako napišete naredbu

  ADD R0,R0, #257

"Error in line 4: Number 257(10) cannot be coded as 8-bit value rotated for even number of bits"

- U slučaju da se konstanta ne može zapisati u naredbi vi uvijek konstantu možete klasično zapisati u memoriju i učitati naredbom LDR (kao u inicijalnom primjeru)

Tipovi podataka

- Izuzetno često u programima koristimo podatke koji su manje širine od širine riječi procesora
- Npr: bajt (8b) ili poluriječ (16b)
- Zato su i kod Arm-a programeru 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.

Podaci manji od 32 bita

- Spremati takve podatke u memoriji u 32b da bi ih mogli pročitati u procesoru nije efikasno
- Procesori zato često imaju naredbe za čitanje i pisanje podataka manje širine od širine riječi ako je to moguće zbog organizacije pristupa memoriji
- Procesor Arm ima organizaciju memorije tako da se na jednoj memorijskoj adresi nalazi bajt a memorijski sustav omogućuje čitanje 1,2 ili 4 bajta istovremeno
- Procesor Arm koristi jedinstveni memorijski adresni prostor od 2^{32} 8-bitnih podataka

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

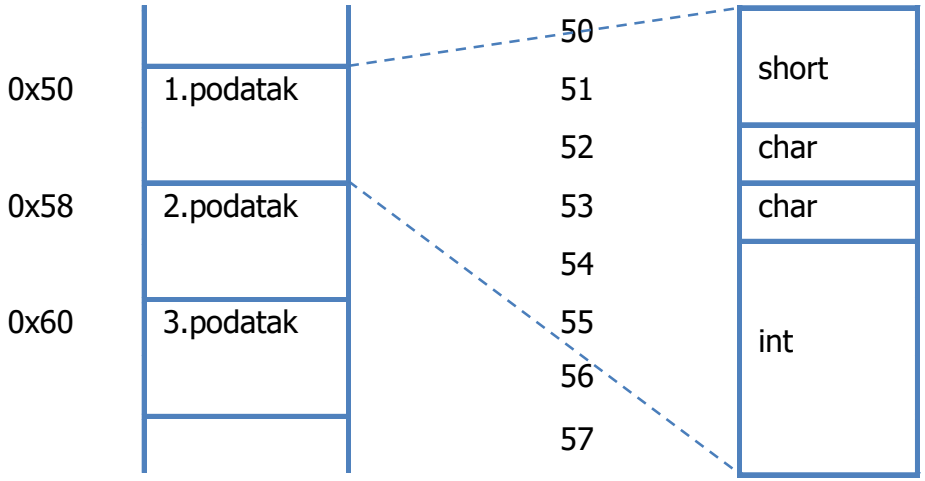
Pristup podacima različitih širina

- U pristupanju bajtovima, riječima i poluriječima postoje ograničenja:
  - Riječi imaju adrese djeljive s 4
  - Poluriječi imaju adrese djeljive s 2
  - Bajtovi nemaju ograničenja na adresu

- U nekim procesorima će u slučaju zadavanja pogrešne adrese doći do tzv. iznimke. Kod simulacije Arm procesora također će se javiti greška.
- Zato pri pisanju programa treba obratiti pažnju:
  - Kod pristupa podatku širine riječi adresa mora biti djeljiva s 4 (dva najniža bita adrese moraju biti nula)
  - Kod pristupa podatku širine poluriječi adresa mora biti djeljiva s 2 (najniži bit adrese mora biti nula)
  - Kod pristupa podatku širine bajta adresa može biti proizvoljna

Primjer

- U memoriji se od adrese 0x50 nalazi niz 64-bitnih složenih podataka kao na slici (npr. C struktura sa short-om, dva char-a i int-om)



- Treba kopirati short i int iz prvog podatka u nizu u treći podatak u nizu, a dva char-a treba kopirati iz trećeg podatka u prvi.

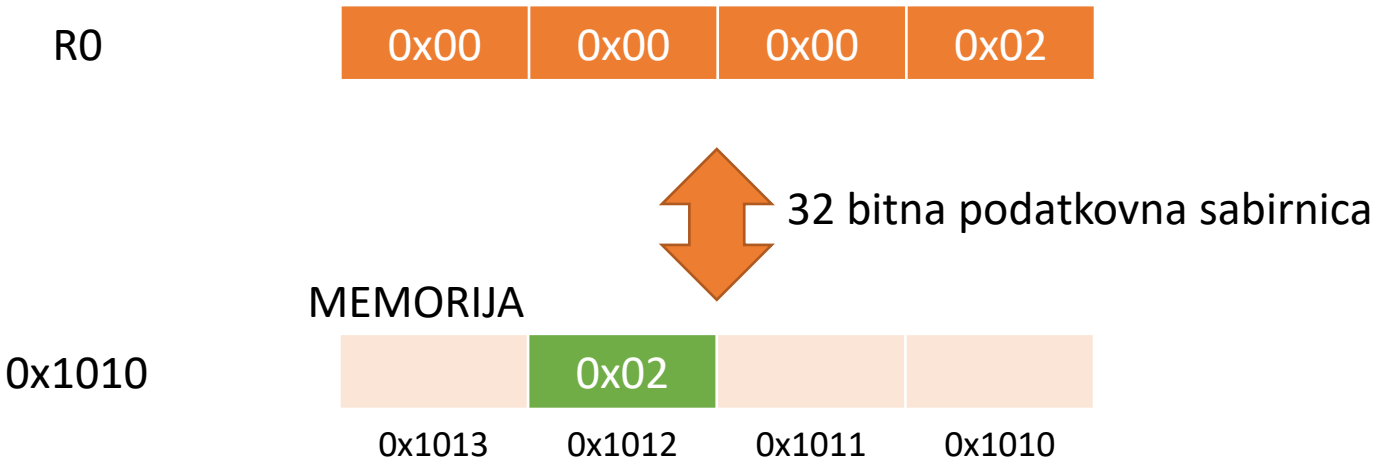
```
LDRH  R0, 0x50
STRH  R0, 0x60
LDR   R0, 0x54
STR   R0, 0x64
LDRB  R0, 0x62
STRB  R0, 0x52
LDRB  R0, 0x63
STRB  R0, 0x53
```

Razmislite:

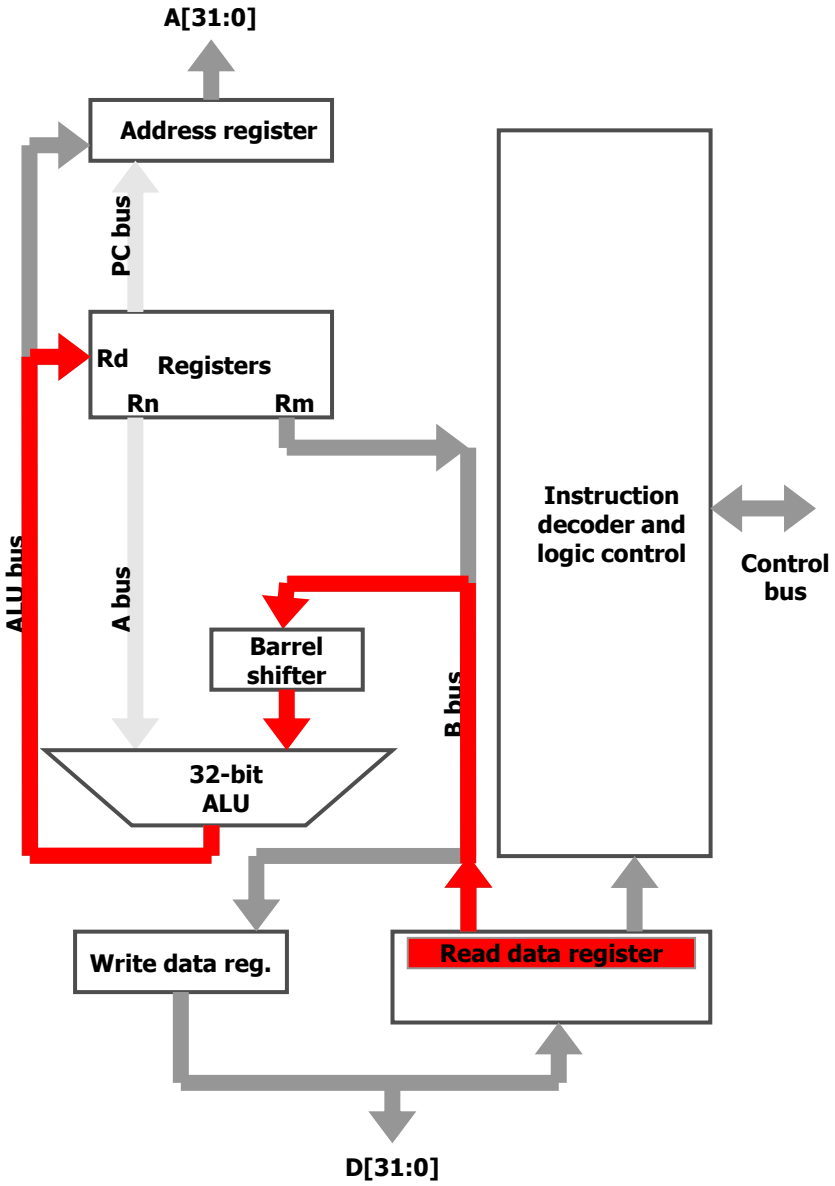
- Da li prethodno rješenje radi ako su podaci na adresama
  - 0x100 i 0x110
  - 0x1FF0 i 0x2000

Kako se bajt pravilno učita u registar?

- `LDRB R0, 0x1012`



LDRB,LDRH



- Ako adrese podataka B i H koji se čitaju nisu djeljive s 4, tada se podaci pomiču u Barrel shifteru i nakon toga zapisuju u željeni registar

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)

Naredbe za množenje

- Naredbe za množenje nisu smještene u grupu osnovnih aritmetičko-logičkih naredbi, već se definiraju zasebno
- Postoji šest naredbi za množenje ali ćemo mi u okviru AR1R koristiti samo dvije:
  - MUL, 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 množenje

- Obično 32-bitno množenje koristimo kad znamo da su operandi "dovoljno mali" da rezultat stane u 32-bita

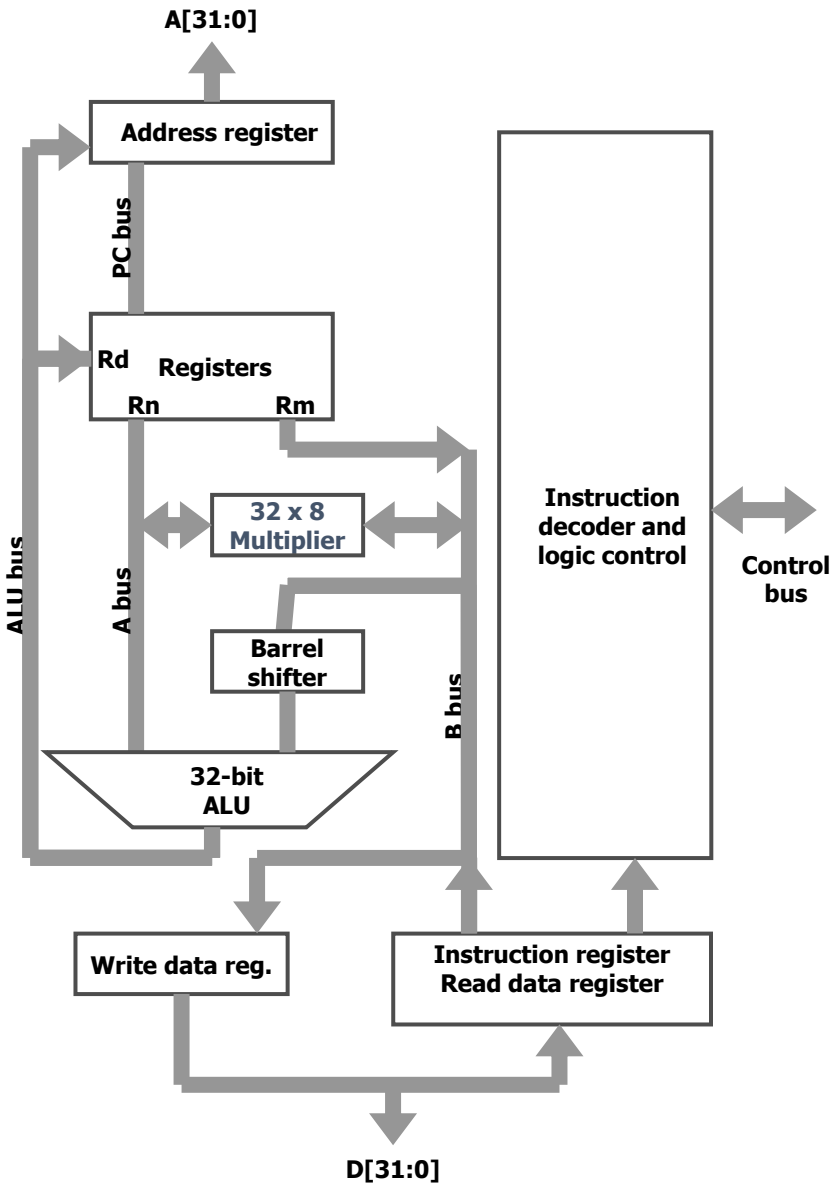
- Kao dodatna opcija gore navedenim naredbama nudi se mogućnost da se umnožak pribraja sadržaju nekog registra. Takve naredbe općenito su poznate kao Multiply Accumulate, a kod procesora Arm nazivaju se: MLA

- 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]

Naredbe za množenje



Aritmetičko-logičke naredbe

- Aritmetičko-logičke naredbe imaju dva operanda i spremaju rezultat u zadani registar (osim naredaba za usporedbu koje zanemaruju rezultat)

- Iznimke su naredbe MOV, MVN koje imaju samo jedan operand i naredba MLA koja ima 4 operanda

- Prvi operand uvijek je registar
- Drugi operand može biti:
  - Neposredna vrijednost
  - Vrijednost iz registra
  - Vrijednost iz registra s pomakom/rotacijom

Aritmetičko-logičke naredbe

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