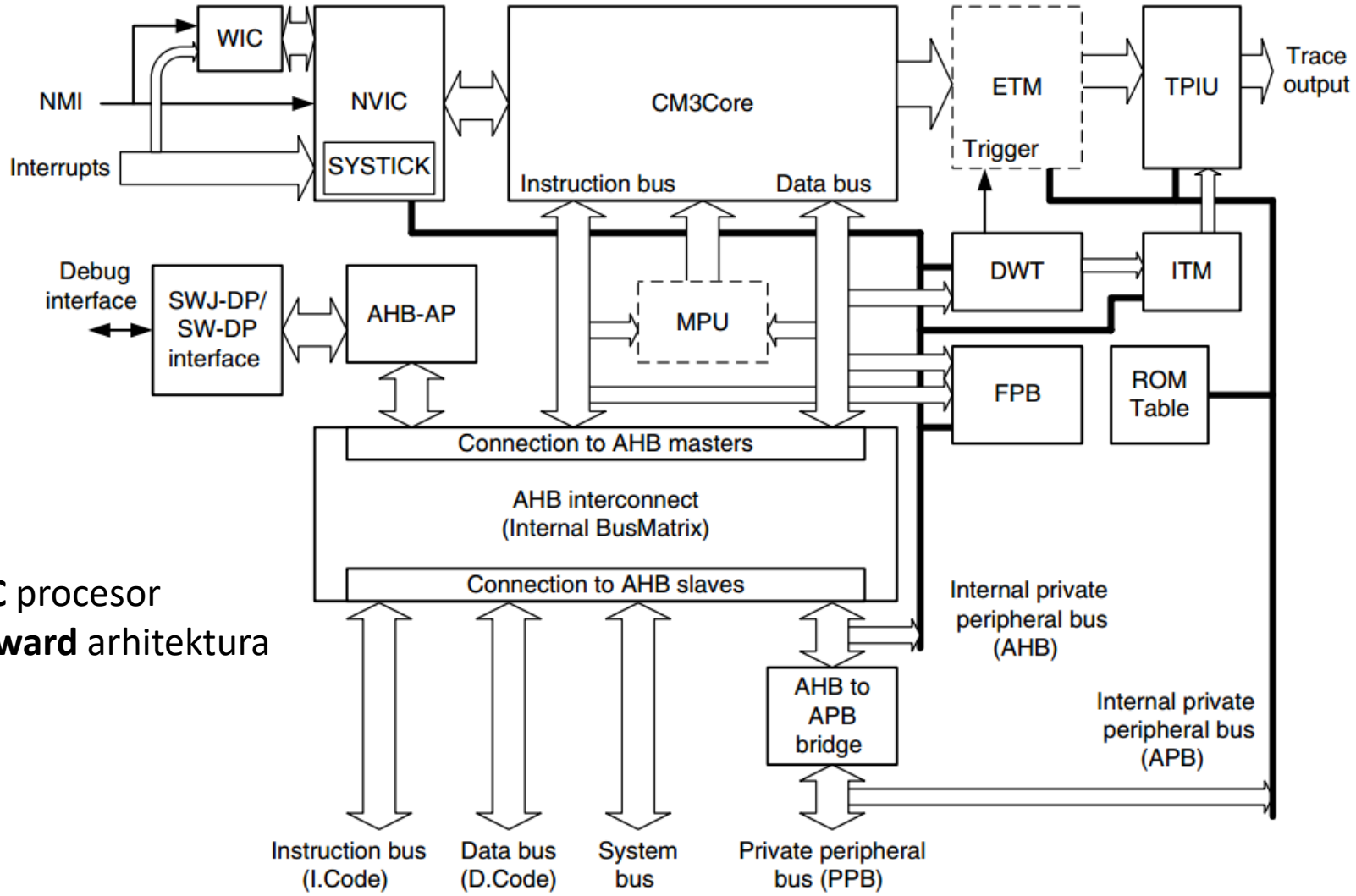
PRIMENA MIKROKONTROLERA-MS1PMK

2. deo

2017

Nenad Jovičić

Cortex-M3 – procesor baziran na ARMv7-M arhitekturi

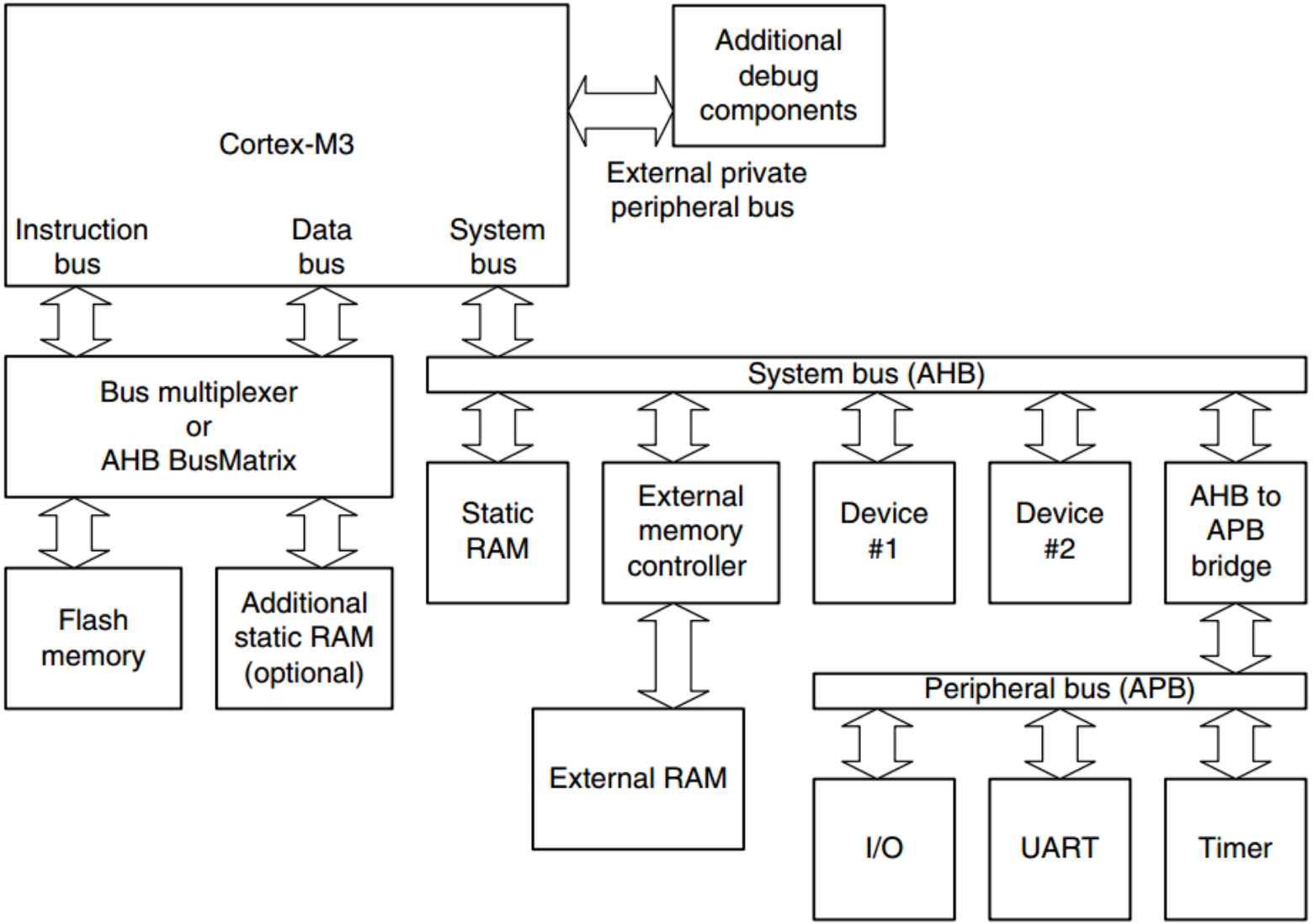


RISC procesor
Harward arhitektura

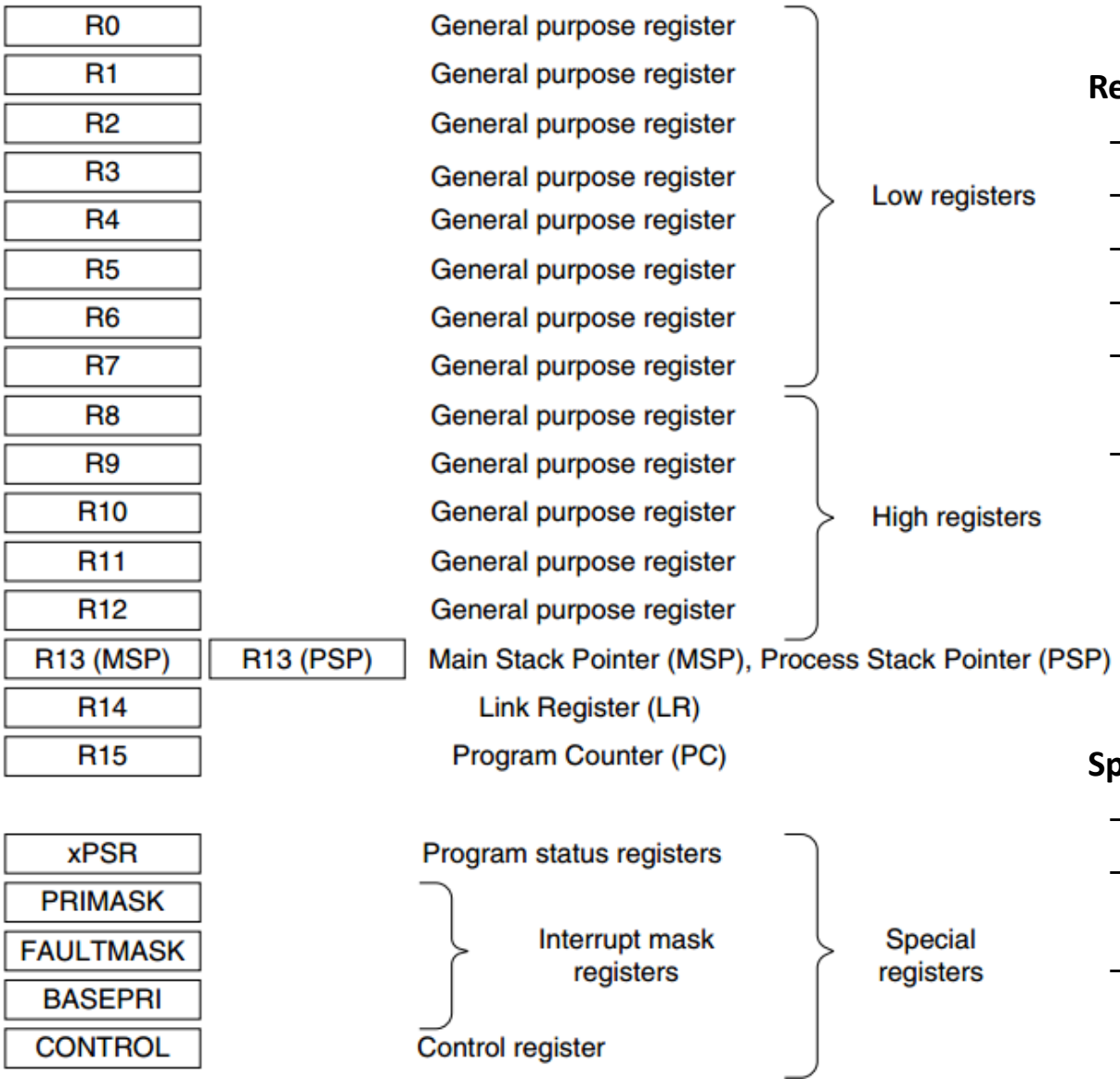
Cortex-M3 core periferije

- **NVIC** (Nested Vector Interrupt Controller) – integrisani prekidni kontroler koji omogućava procesiranje prekida sa malim kašnjenjem
- **WIC** (Wakeup Interrupt Controller) – opcioni kontroler zahteva za buđenje iz režima smanjene potrošnje.
- **SYSTICK** – sistemski tajmer tj. 24-bitni brojač na dole namenjen za podršku generisanju sistemskog prekida za operativne sisteme.
- **MPU** (Memory Protection Unit) – opciona jedinica za kontrolu pristupa pojedinim regionima u memoriji
- **ETM** (Embedded Trace Macrocell) – modul koji obezbeđuje logovanje izvršavanja instrukcija
- **DWT** (Data Watchpoint and Trace Unit) – modul koji implementira data watchpoint
- **FPB** (Flash Patch and Breakpoint Unit) – modul koji omogućava preusmeravanje izvršavanja programa iz CODE memorije u neki drugi deo memorije.

Tipična arhitektura Cortex-M3 mikrokontrolera



Registri procesora



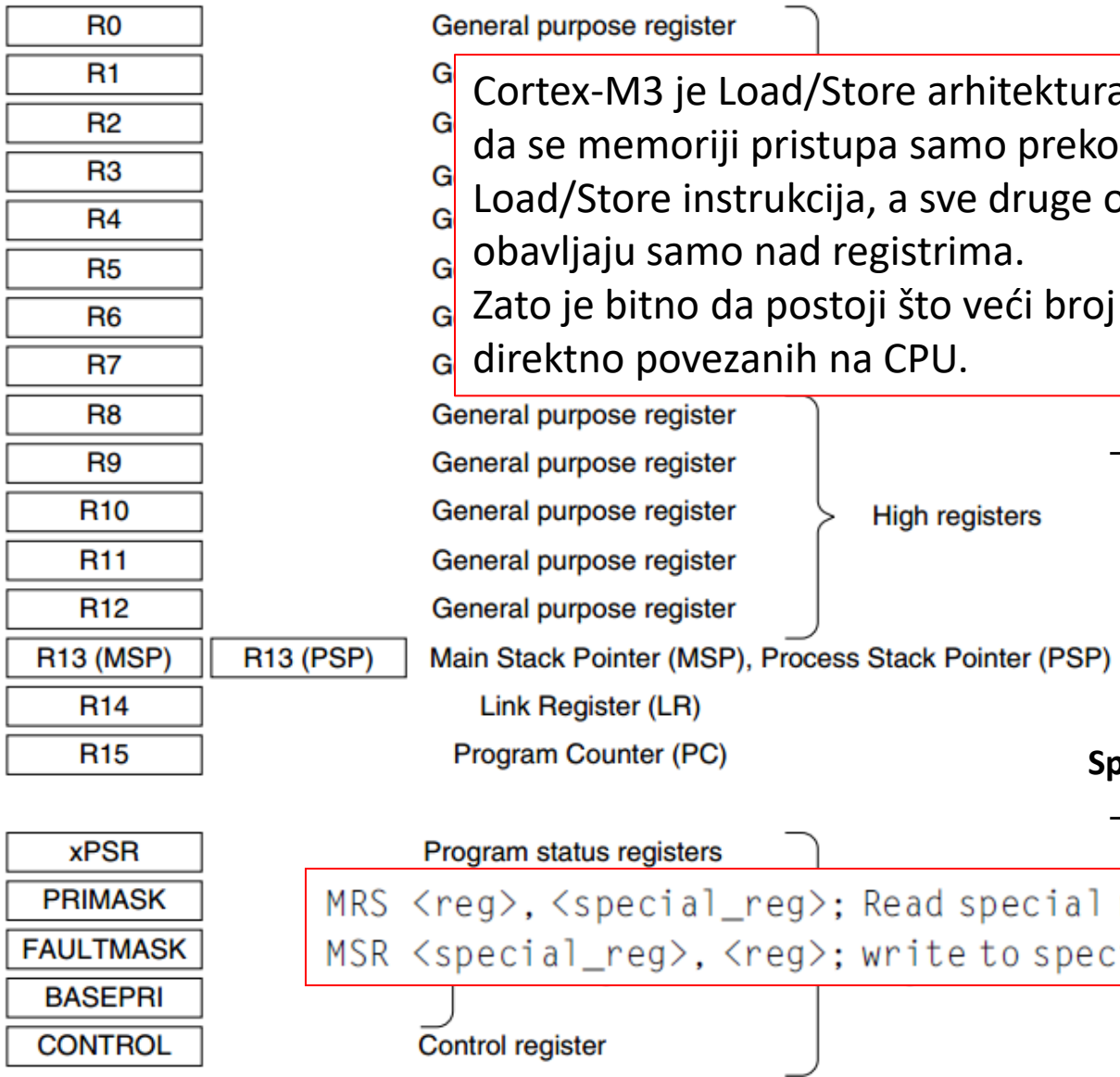
Registri opšte namene:

- 13 registara opšte namene, r0-r12
- Pokazivač steka SP (r13)
- Link registra LR (r14)
- Programski brojač PC (r15)
- *Registri r0-r7 (niži registri) su dostupni preko svih instrukcija*
- *Registri r8-r12 (viši registri) su dostupni preko svih 32-bitnih instrukcija.*

Specijalni registri:

- Programski statusni registar xPSR
- Registri za kontrolu prekida PRIMASK, FAULTMASK, BASEPRI
- Kontrlni registra CONTROL

Registri procesora



Cortex-M3 je Load/Store arhitektura, što znači da se memoriji pristupa samo preko Load/Store instrukcija, a sve druge operacije se obavljaju samo nad registrima.
Zato je bitno da postoji što veći broj registara direktno povezanih na CPU.

mene:

opšte namene, r0-r12

eka SP (r13)

LR (r14)

brojač PC (r15)

7 (niži registri) su dostupni preko svih instrukcija

- *Registri r8-r12 (viši registri) su dostupni preko svih 32-bitnih instrukcija.*

Specijalni registri:

- Programski statusni registar xPSR
- Registri za kontrolu prekida PRIMASK,

```
MRS <reg>, <special_reg>; Read special register into register
MSR <special_reg>, <reg>; write to special register
```

SP – Stack pointer

- Registar koji se fizički realizuje kao dva 32-bitna registra.
- Main Stack Pointer (MSP) – ovaj registar je podrazumevani registar posle reseta i mogu da ga koriste i korisnički program i prekidne rutine. U slučaju korišćenja operativnog sistema, ovaj registar koriste Kernel i prekidi, kao i neki delovi korisničkih Thread-ova sa posebnim privilegijama.
- Process Stack Pointer (PSP) – ovaj registar je namenjen za korišćenje od strane neprivilegovanih delova softvera. Na primer, to su korisnički Thread-ovi u Operativnom sistemu. Može da se koristi samo u Thread modu.
- U zavisnosti od odgovarajućeg konfiguracionog bita u CONTROL registru pristupa se jednom ili drugom od ova dva registra.
- Jednostavne aplikacije bez OS-a uglavnom korsite samo MSP.
- Kod Cortex-M3 familije stek je realizovan kao Full Descending Stack, što znači da stek raste prema nižim adresama i pokazuje na poslednju zauzetu adresu. Sve PUSH i POP operacije su 32-bitne.
- Najniža dva bita SP-a su uvek jednaka nuli jer se radi samo sa 32-bitni veličinama.
- Nakon reseta inicijalna vrednost MPS-a se hradverski inicijalizuje na vrednost sa početka momorije, a vrednost PSP-a je nedefinisana.

LR – Link Register

- LR se koristi za čuvanje adrese povratka iz potprograma - na primer kada je skok izvršen BL (Branch with Link) instrukcijom:

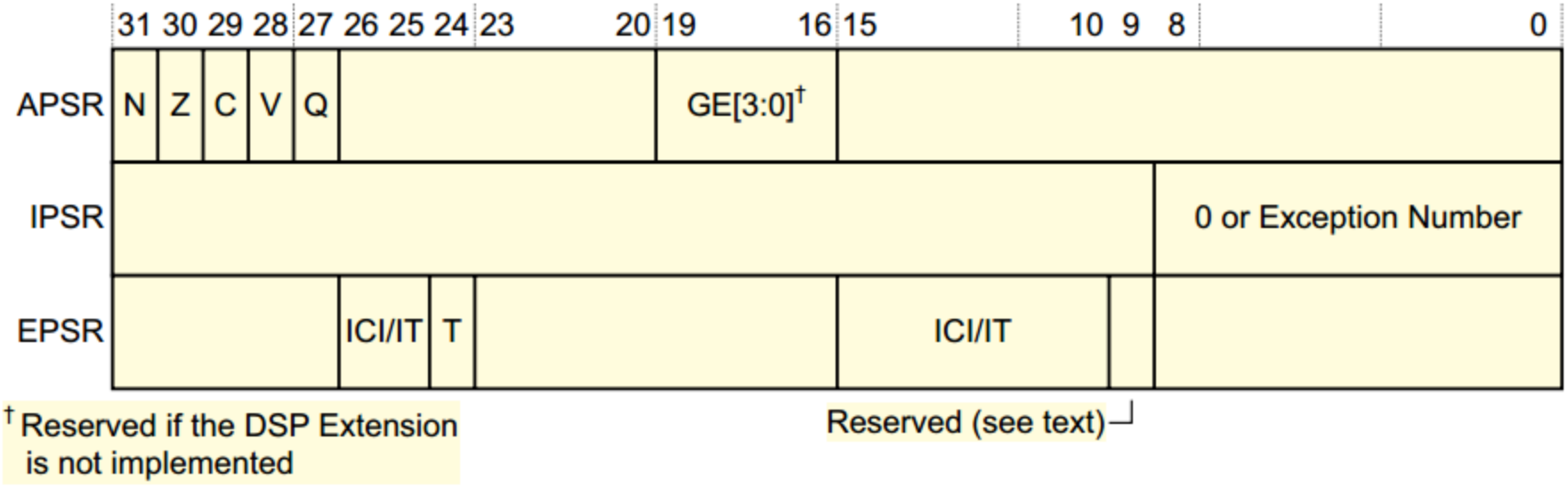
```
main ; Main program
     ...
     BL function1 ; Call function1 using Branch with Link instruction.
                  ; PC = function1 and
                  ; LR = the next instruction in main
     ...
function1
     ...          ; Program code for function 1
     BX LR        ; Return
```

- Osim ovoga koristi se i pri povratku iz prekidne rutine.

Statusni registri

- Application Program Status Register, APSR
  - Sadrži flegove koje koristi korisnički neprivilegovani aplikativni softver
- Interrupt Program Status Register, IPSR
  - Sadrži broj izuzetka/prekida koji se trenutno izvršava
- Execution Program Status Register, EPSR
  - Sadrži specijalne flegove koji kontrolišu rad procesora. Na primer biti ICI/IT se koriste za kontrolu prekidanja uslovnih blokova instrukcija i instrukcija koje rade višestruke Load/store operacije (LDM/STM)

Mnemonic	Registers accessed
IAPSR	IPSR and APSR
EAPSR	EPSR and APSR
XPSR	All three xPSR registers
IEPSR	IPSR and EPSR



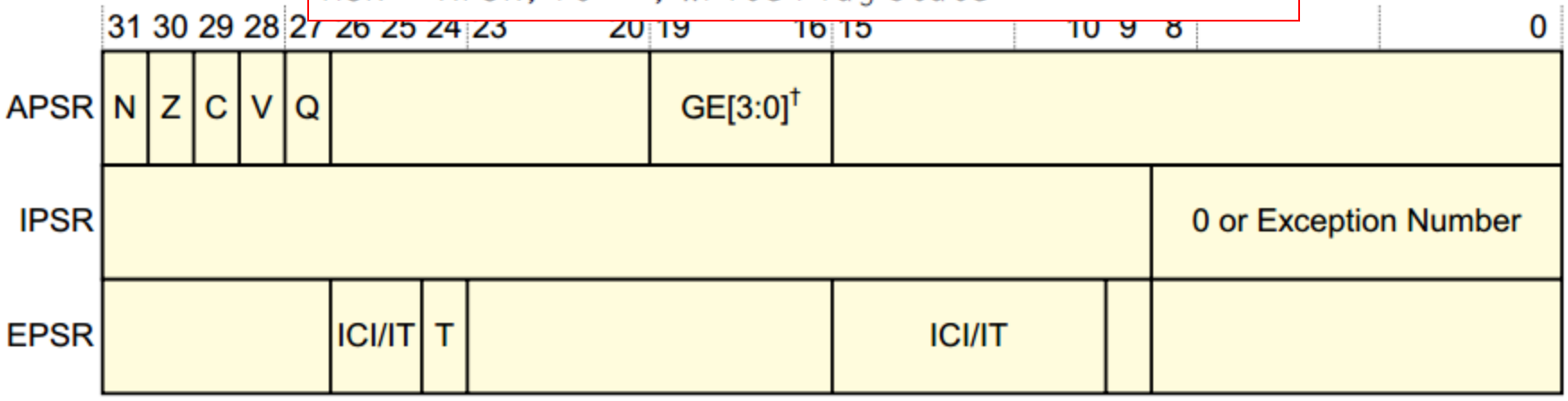
Statusni registri

- Application Program Status Register, APSR
  - Sadrži flegove koje koristi korisnički neprivilegovani aplikativni softver
- Interrupt Program Status Register, IPSR
  - Sadrži
- Executior
  - Sadrži specijalne flegove koji kontrolišu rad procesora. Na primer biti ICI/IT se koriste za kontrolu prekidanja uslovnih blokova i
    operacije

Mnemonic	Registers accessed
	PSR and APSR
	PSR and APSR
XPSR	All three xPSR registers
IEPSR	IPSR and EPSR

```
MRS   r0, PSR   ; Read the combined program status word
MSR   PSR, r0   ; Write combined program state word
```

```
MRS   r0, APSR   ; Read Flag state into R0
MRS   r0, IPSR   ; Read Exception/Interrupt state
MSR   APSR, r0   ; Write Flag state
```



† Reserved if the DSP Extension is not implemented

Aritmetika sa saturacijom

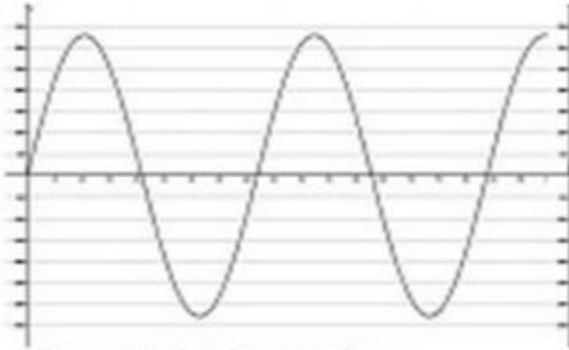


Figure 3-1. No overflow

Figure 3-2. Overflow – standard CPU arithmetic

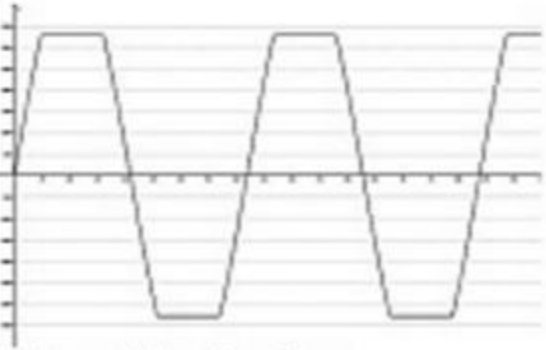
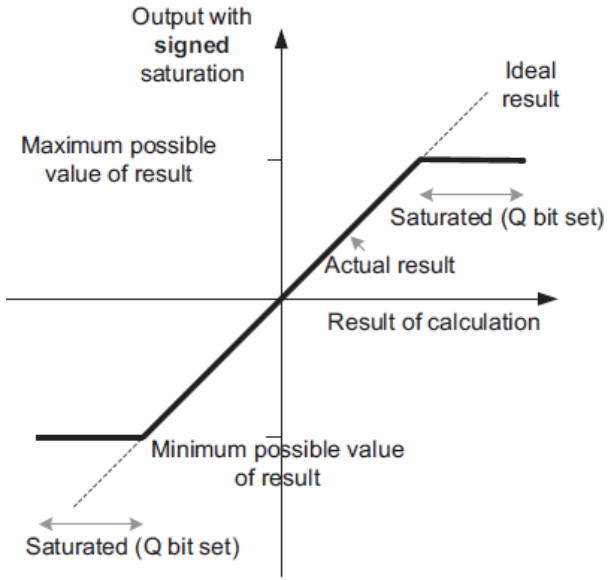


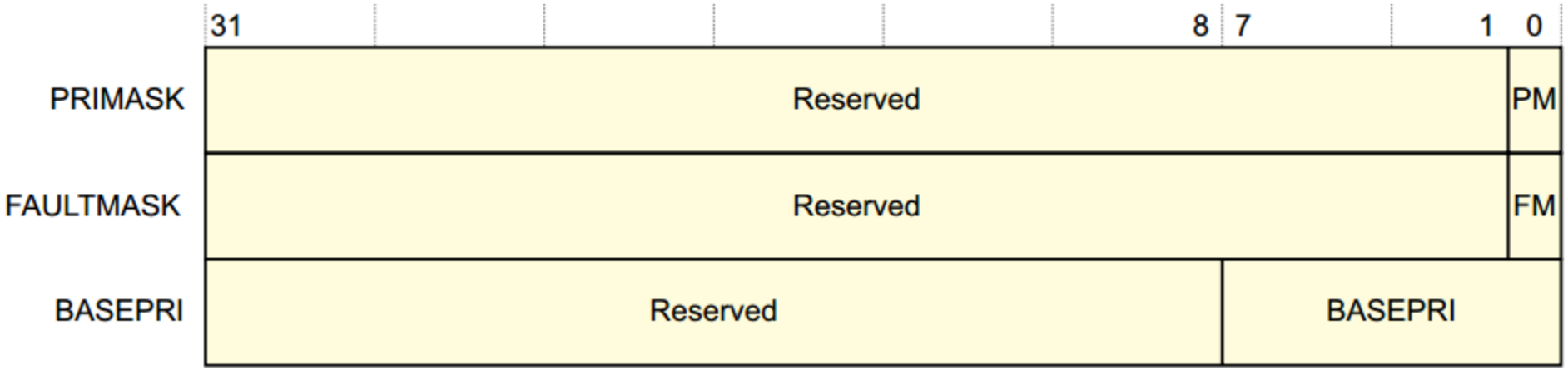
Figure 3-3. Overflow – saturating arithmetic



- Q flag je setovan nakon operacije u kojoj se pojavi saturacija.
- Takve instrukcije uglavnom počinju sa Q, kao na primer QADD, QSUB...

Registri za kontrolu prekida

- Pristup ovim registrima je dozvoljen samo u privilegovanom režimu rada softvera.
- Setovanje PM bita u PRIMASK registru maskira sve prekide osim NMI i HardFault prekida.
- Setovanje FM bita u FAULTMASK registru maskira sve prekide osim NMI prekida.
- BASEPRI registra definiše nivo prioriteta prekida kojima je dozvoljeno generisanje.



Pristup registrima kontrole prekida

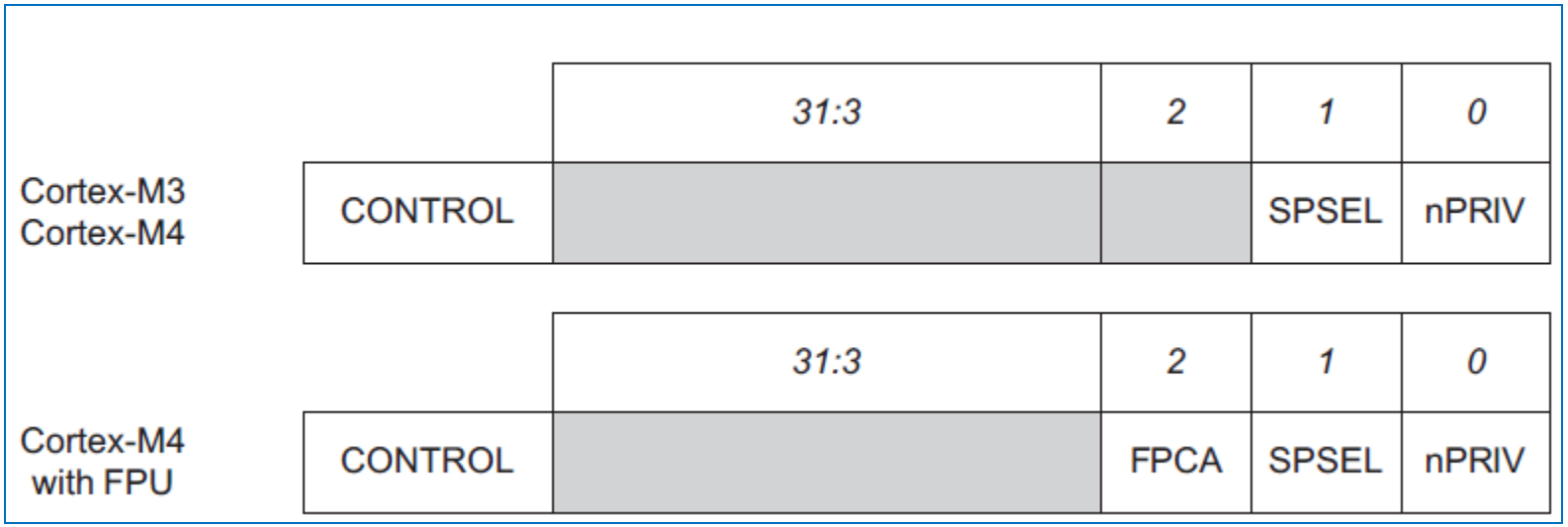
- Iz C-a:

```
x = __get_BASEPRI(); // Read BASEPRI register
x = __get_PRIMARK(); // Read PRIMASK register
x = __get_FAULTMASK(); // Read FAULTMASK register
__set_BASEPRI(x); // Set new value for BASEPRI
__set_PRIMASK(x); // Set new value for PRIMASK
__set_FAULTMASK(x); // Set new value for FAULTMASK
__disable_irq(); // Clear PRIMASK, enable IRQ
__enable_irq(); // Set PRIMASK, disable IRQ
```

- Iz Asembler-a:

```
MRS     r0, BASEPRI   ; Read BASEPRI register into R0
MRS     r0, PRIMASK   ; Read PRIMASK register into R0
MRS     r0, FAULTMASK ; Read FAULTMASK register into R0
MSR     BASEPRI, r0   ; Write R0 into BASEPRI register
MSR     PRIMASK, r0   ; Write R0 into PRIMASK register
MSR     FAULTMASK, r0 ; Write R0 into FAULTMASK register
```

CONTROL registar



		31:3	2	1	0
Cortex-M3 Cortex-M4	CONTROL			SPSEL	nPRIV

		31:3	2	1	0
Cortex-M4 with FPU	CONTROL		FPCA	SPSEL	nPRIV

CONTROL registar

- Control registar poseduje samo dva bita (3 ako postoji FPU).
- Njime se kontroliše korišćenje steka i nivo privilegije softvera.
- Control registar može da se menja samo iz privilegovanog softvera.
- Bit 2 (FPCA) govori da li se FPU koristi ili ne.

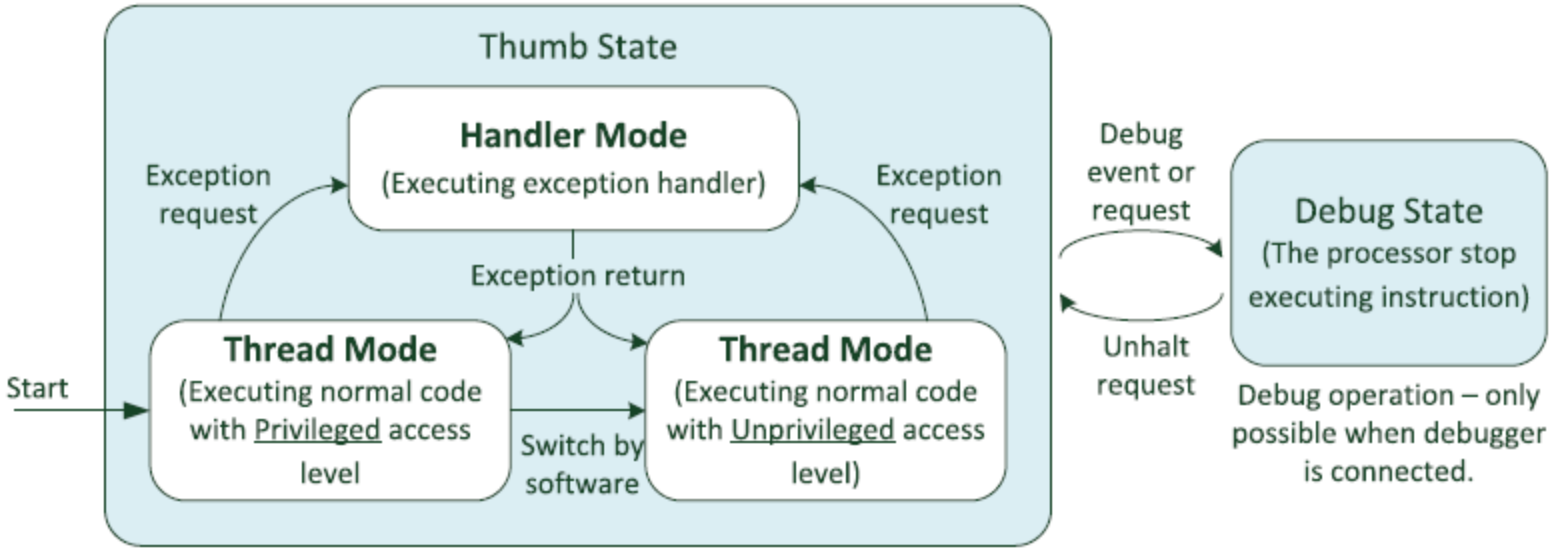
Bit	Function
CONTROL[1]	Stack status: 1 = Alternate stack is used 0 = Default stack (MSP) is used If it is in the thread or base level, the alternate stack is the PSP. There is no alternate stack for handler mode, so this bit must be 0 when the processor is in handler mode.
CONTROL[0]	0 = Privileged in thread mode 1 = User state in thread mode If in handler mode (not thread mode), the processor operates in privileged mode.

```
x = __get_CONTROL(); // Read the current value of CONTROL
__set_CONTROL(x); // Set the CONTROL value to x
```

```
MRS     r0, CONTROL ; Read CONTROL register into R0
MSR     CONTROL, r0 ; Write R0 into CONTROL register
```

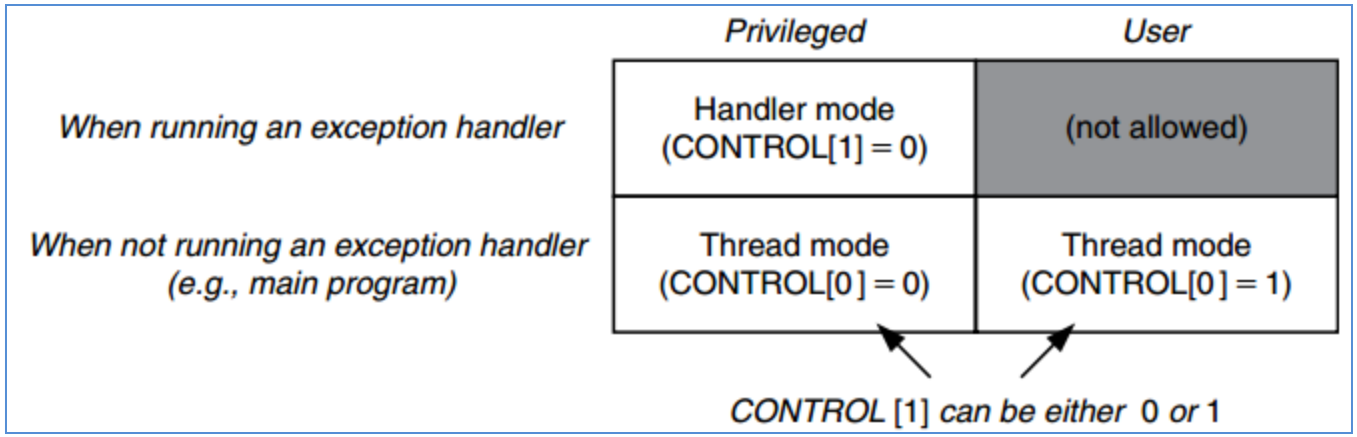
Stanja i režimi rada procesora

- Dva stanja – thumb (radno stanje) i debug (zaustavljen rad)
- Dva režima rada procesora – thread i handler režim.
- Dva nivoa privilegija – privilegovani nivo (OS, prekidi) i neprivilegovani (korisnički softver).



Režimi privilegija softvera

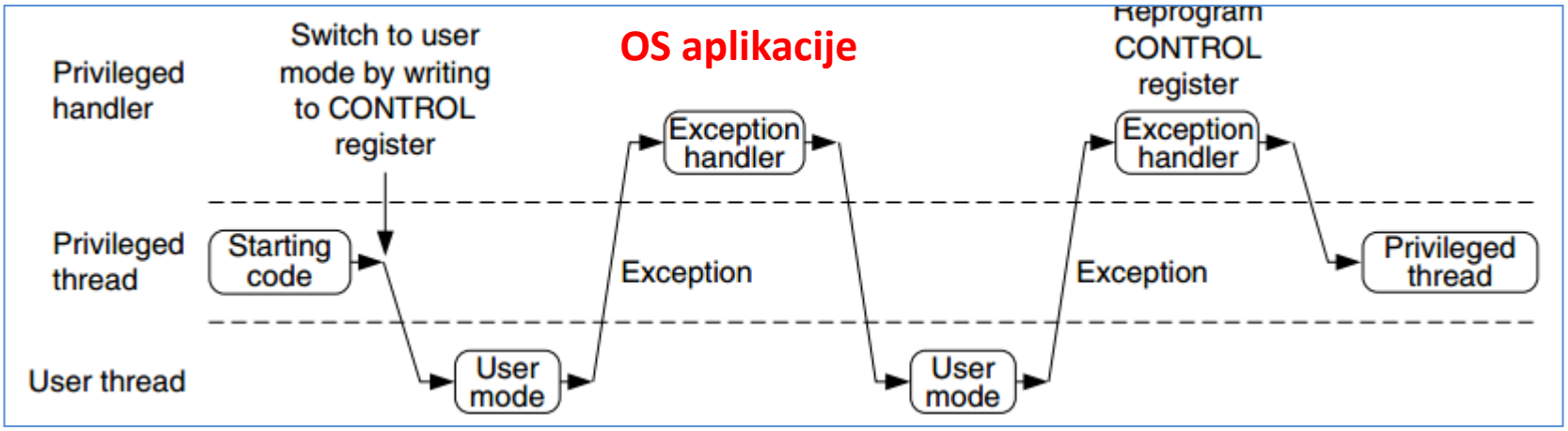
- Privilegovani režim:
  - U njega se automatski prelazi ulaskom u prekid.
  - Potpuno su dostupni svi sistemski resursi.
  - U ovom režimu je moguće menjati nivo privilegije.
- Neprivilegovani ili tzv. korisnički režim:
  - Uobičajeni režim običnog korisničkog programa.
  - Nisu dostupni svi sistemski resursi.
  - Promena nivoa privilegija moguća samo nakon prekida.

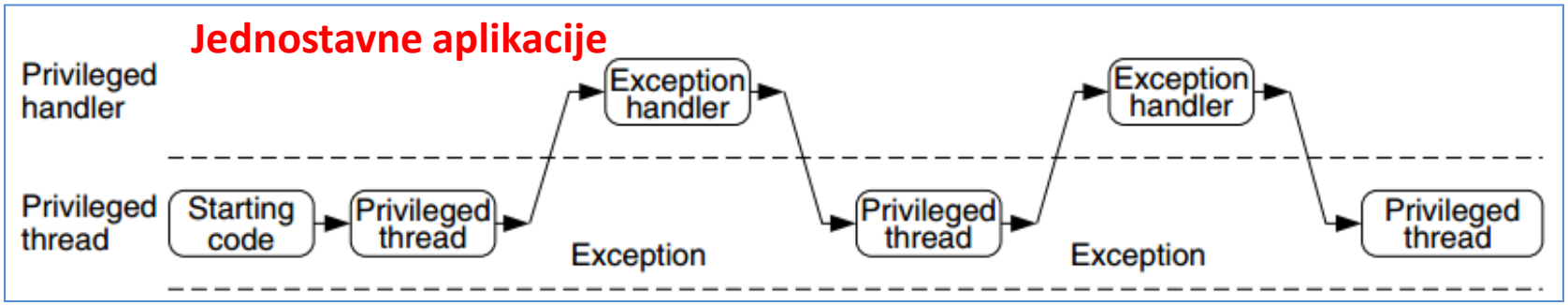


- Za razliku od režima rada procesora, nivo privilegija se menja programski.

Privilegovani/neprivilegovani rad

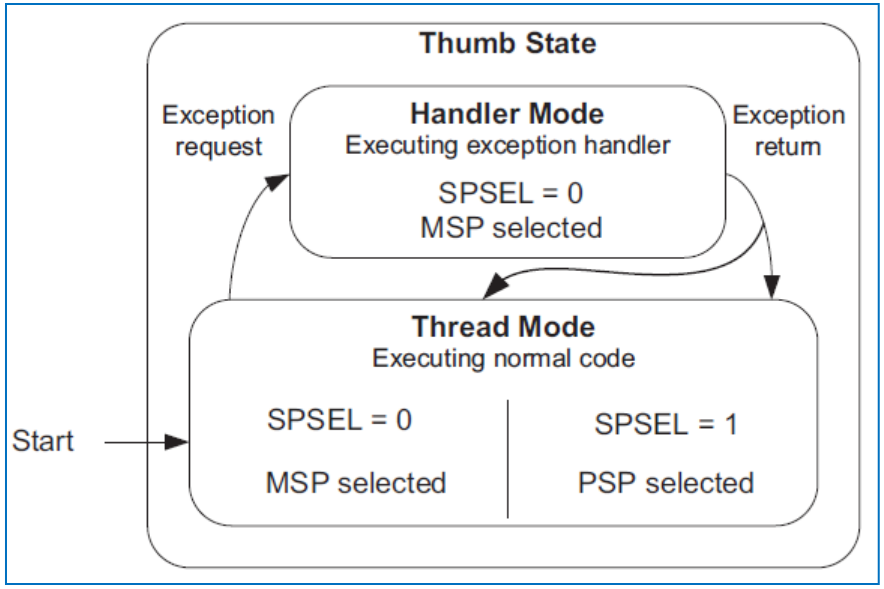
- Promena nivoa privilegije je moguća samo u privilegovanom modu rada.
- U prekidu je moguće uticati na nivo privilegije prilikom povratka u thread režim procesora.





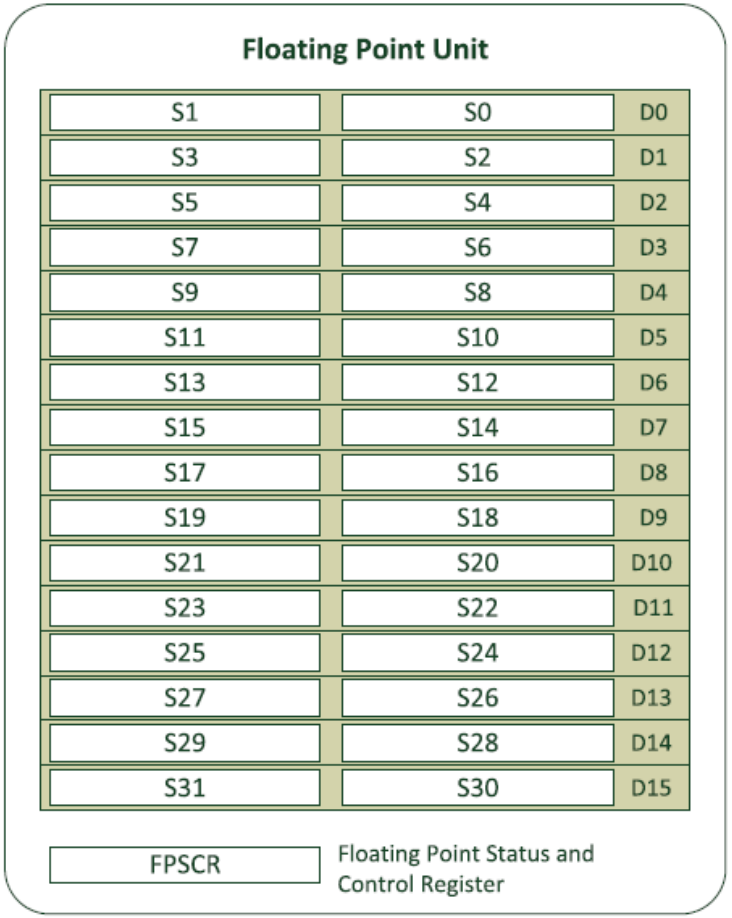
Jednostavne OS aplikacije

- Softver je praktično uvek privilegovan, ali se deo softvera koji je Kernel OS-a izvršava na MSP steku, a deo softvera koji je korisnički na PSP steku.



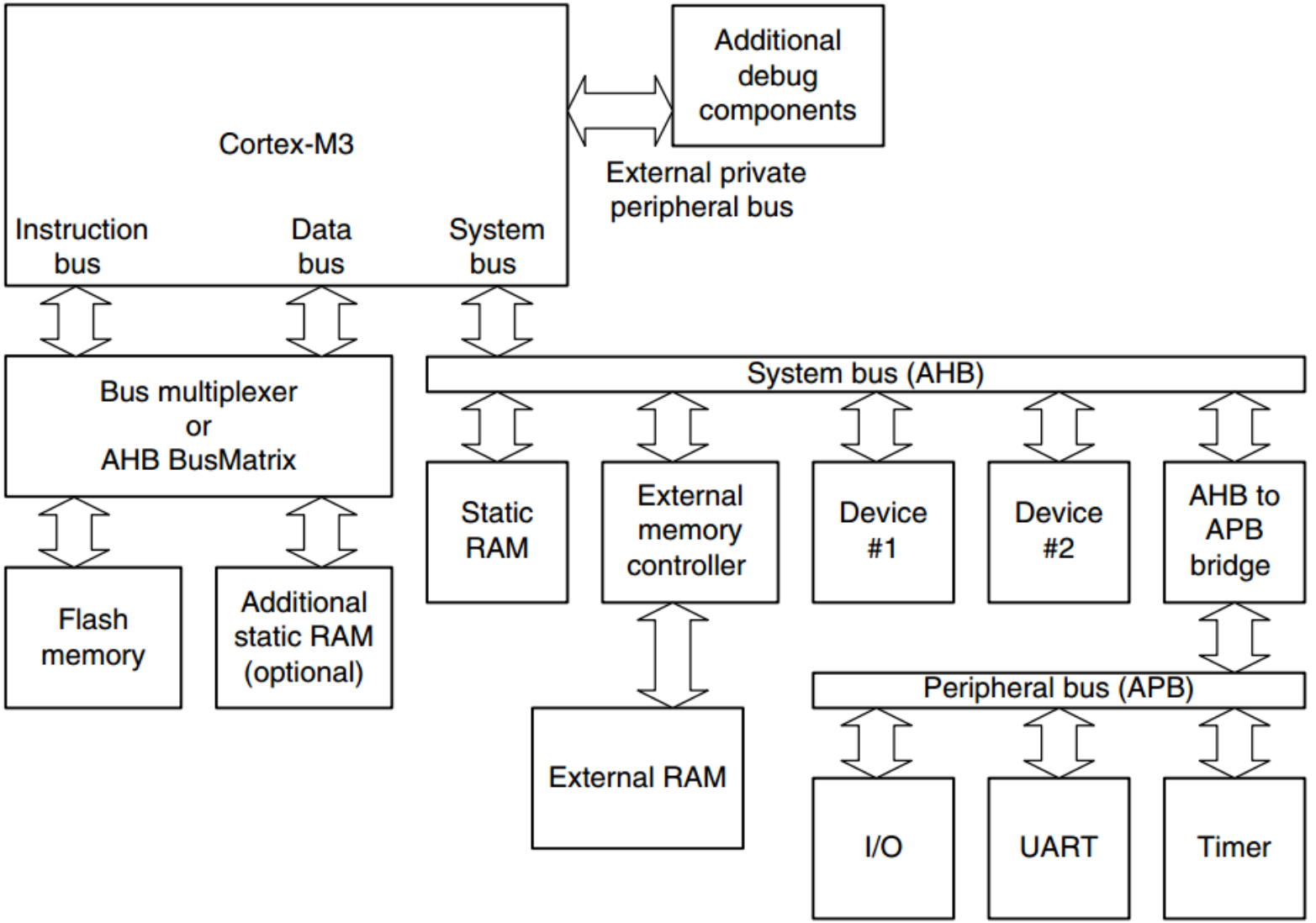
Floating Point Unit

- Registri Sx su u osnovi 32-bitni (single precision).
- Grupisanjem se može efektivno dobiti dvostruka preciznost (Dx).

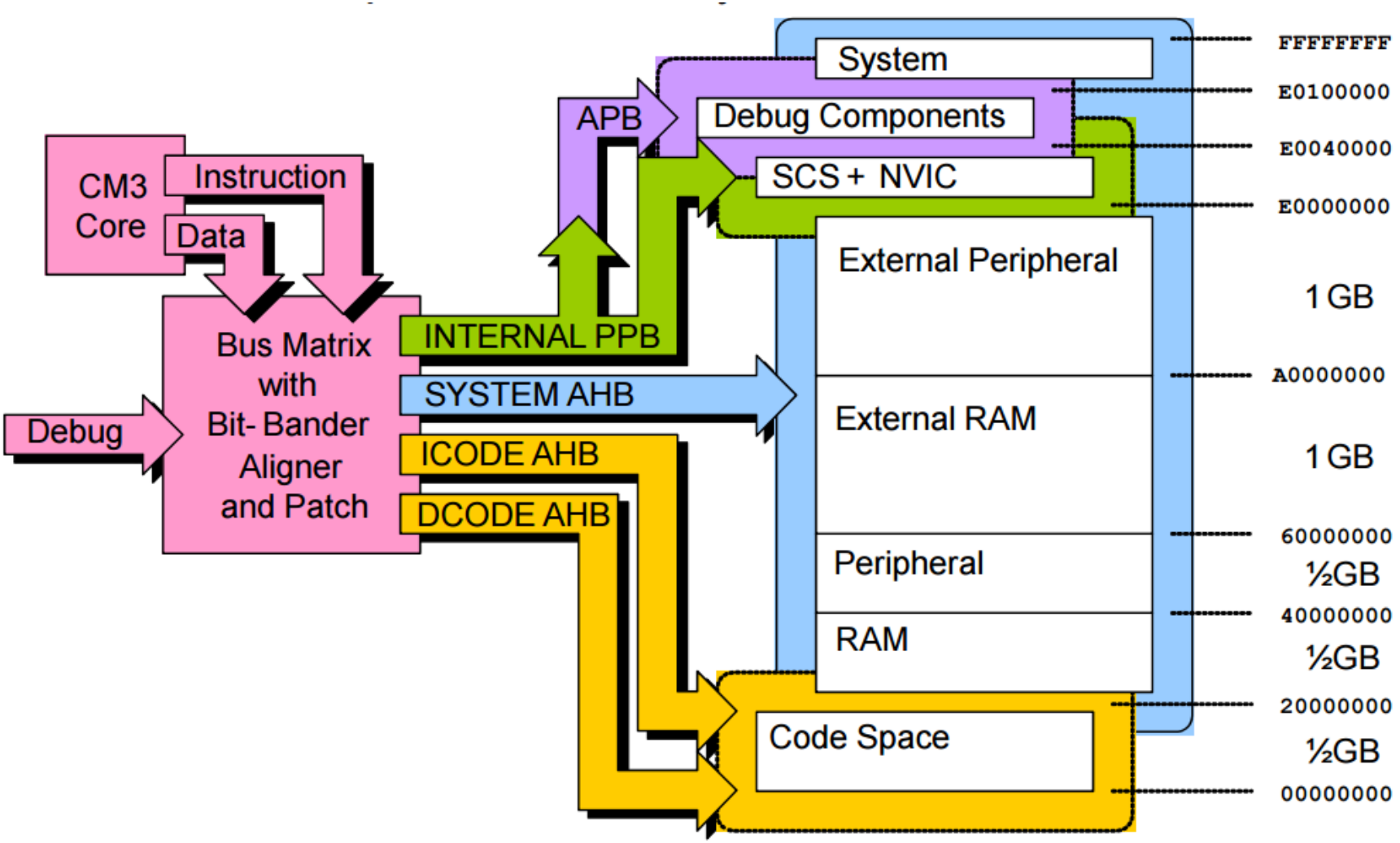


	31	30	29	28	27	26	25	24	23:22	21:8	7	6:5	4	3	2	1	0
FPSCR	N	Z	C	V		AHP	DN	FZ	RMode	Reserved	IDC	Reserved	IXC	UFC	OFC	DZC	IOC

Cortex-M povezivanje sa periferijama

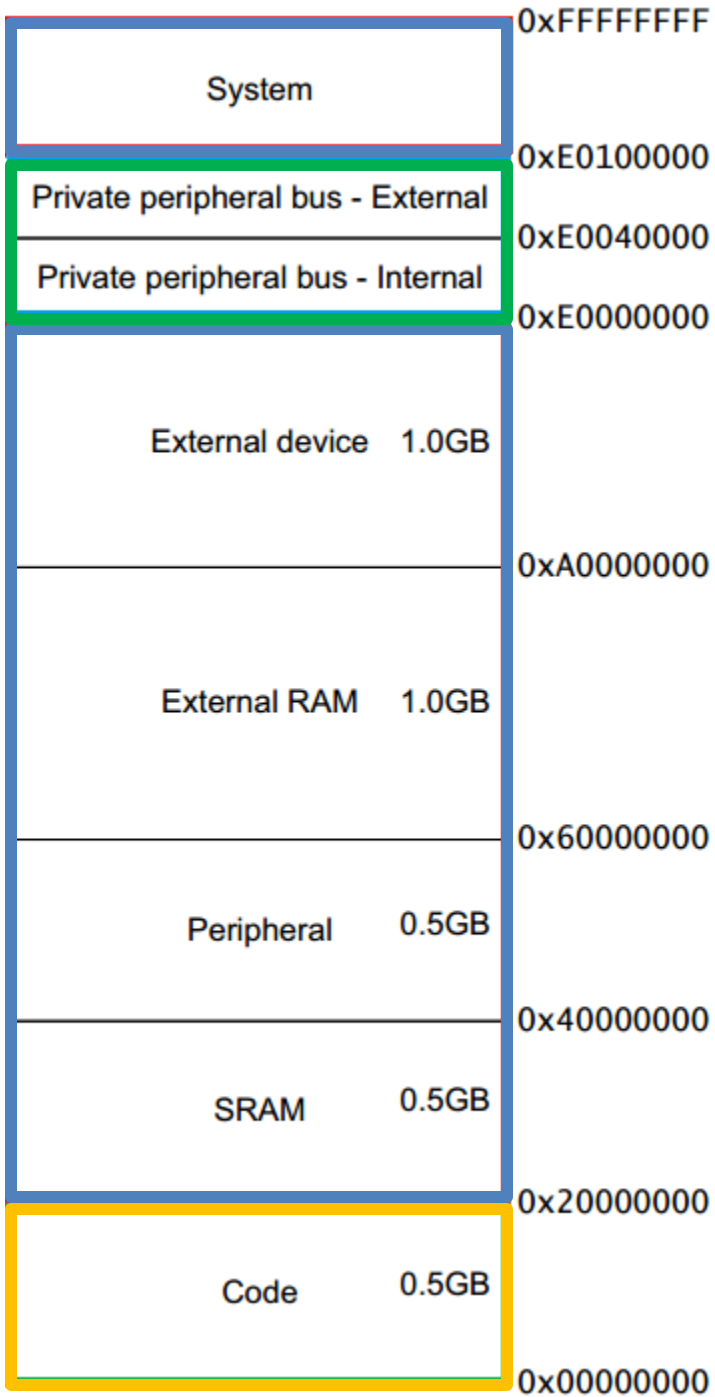


Magistrale

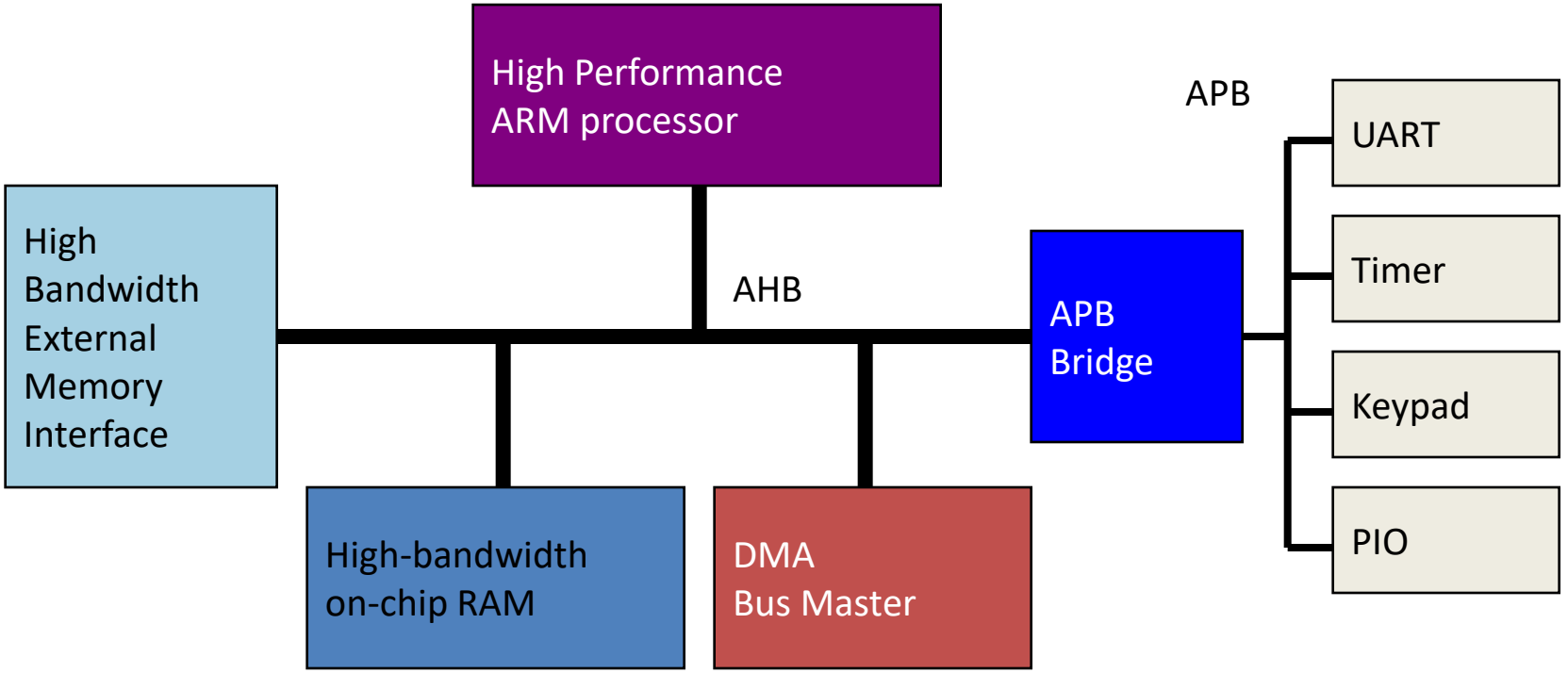

CM3 Core
Instruction
Data
Debug
Bus Matrix with Bit- Bander Aligner and Patch
APB
INTERNAL PPB
SYSTEM AHB
ICODE AHB
DCODE AHB
System
Debug Components
SCS + NVIC
External Peripheral
External RAM
Peripheral
RAM
Code Space
FFFFFFFF
E0100000
E0040000
E0000000
1 GB
A0000000
1 GB
60000000
½GB
40000000
½GB
20000000
½GB
00000000

Magistrale

- **ICode** memorijska magistrala bazirana na AHB-Lite standardu – koristi se za prenos instrukcija iz Code adresnog prostora (0x00000000 do 0x1FFFFFFF).
- **Dcode** memorijska magistrala bazirana na AHB-Lite standardu – koristi se za prenos podataka iz Code adresnog prostora (0x00000000 do 0x1FFFFFFF).
- **System** memorijska magistrala bazirana na AHB-Lite standardu – koristi se za prenos i instrukcija i podataka iz System adresnog prostora (0x20000000 do 0xDFFFFFFF i 0xE0100000 do 0xFFFFFFFF). Uglavnom za RAM I periferije.
- **PPB** magistrala – pristup internim i eksternim privatnim periferijama, komponentama Debug sistema,



AMBA – advanced microcontroller bus architecture



AHB – advanced high-performance bus

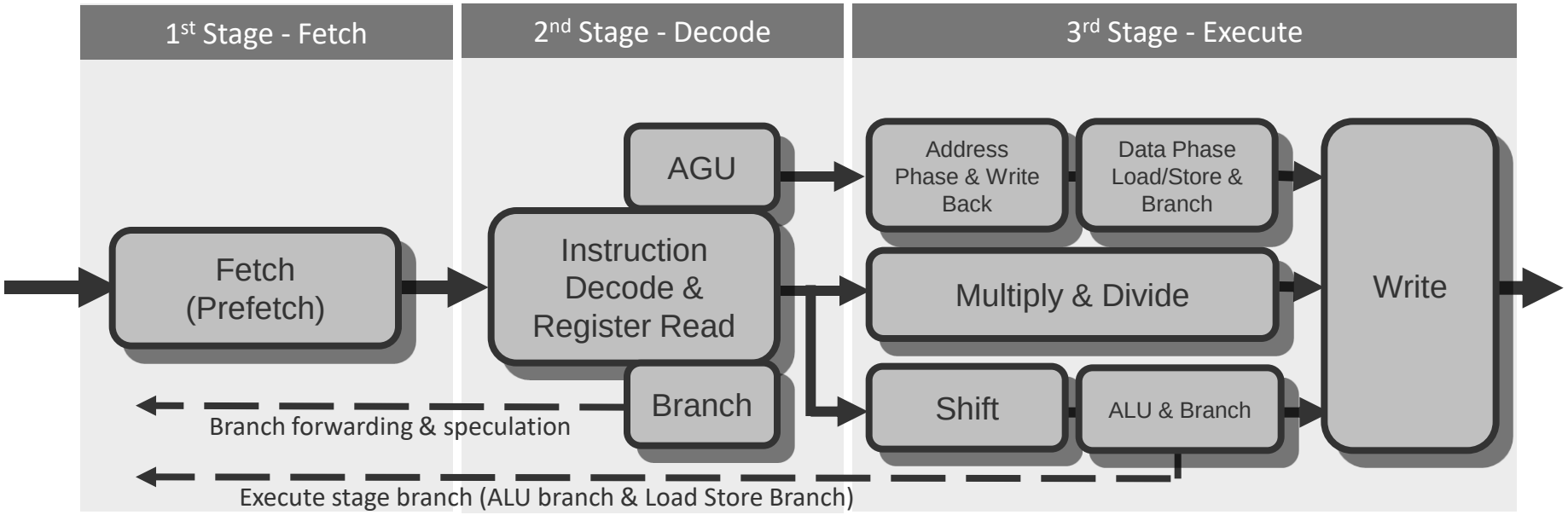
Visoke performanse
Protočni prenos
Burst prenosi
Višestruki masteri

APB – advanced peripheral bus

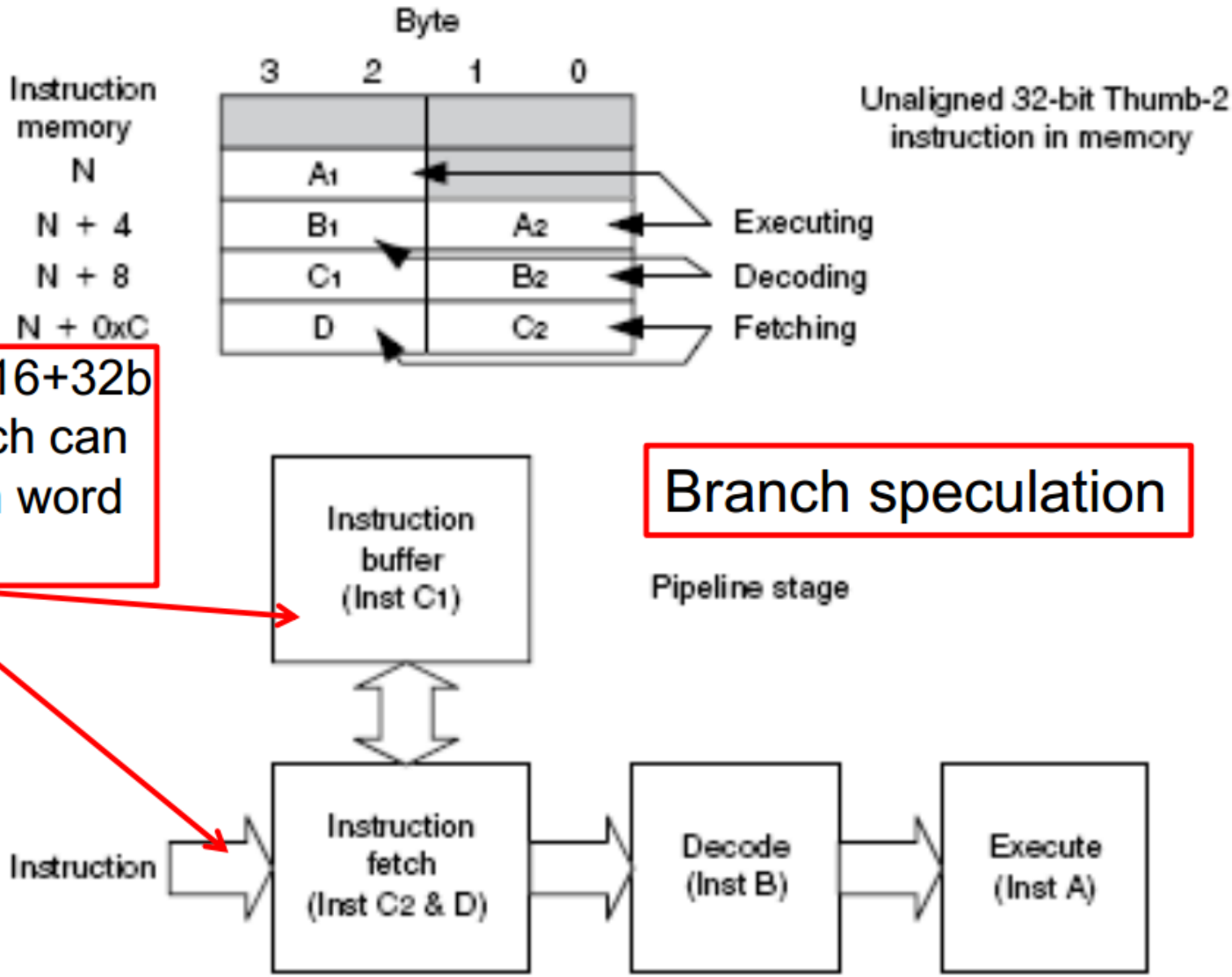
Niska potrošnja
Mala brzina
Jednostavan interfejs
Veza sa Slave periferijama

Cortex-M3 Pipeline

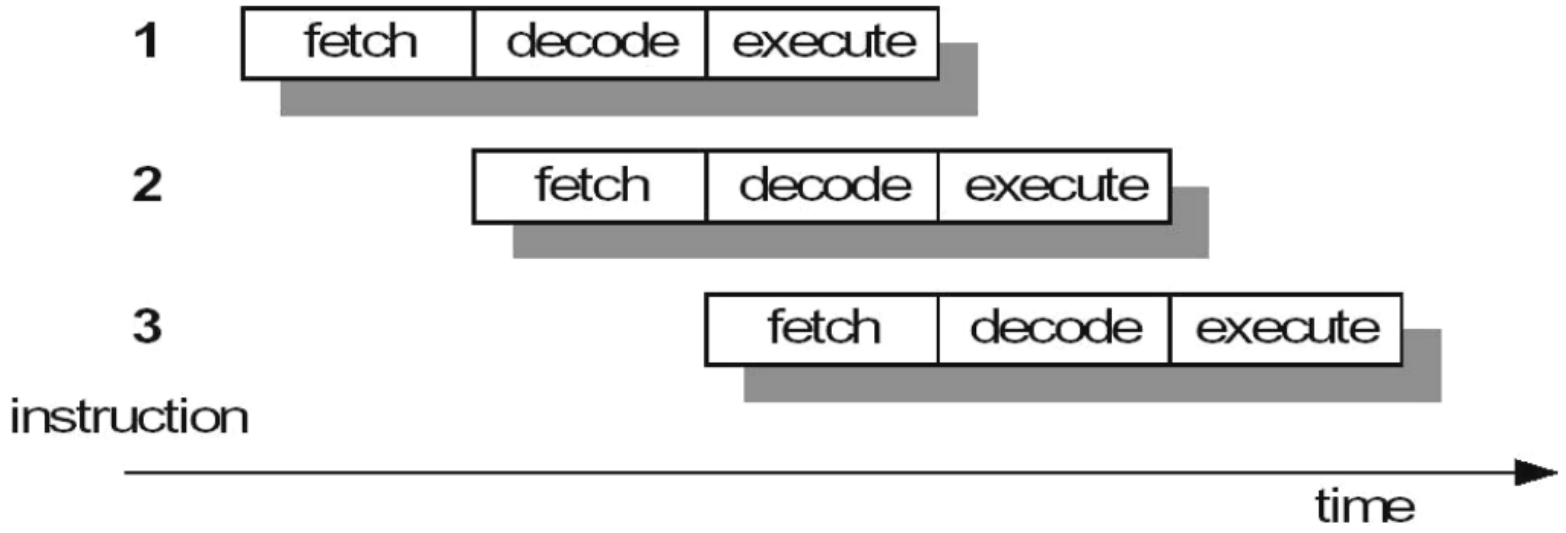
- Trostepeni fetch-decode-execute pipeline
  - U prefetch buffer može da se učita 72 bita, tj. tri 32-bitne ili šest 16-bitnih instrukcija ili njihova kombinacija.
  - Na ovaj način je omogućena jednociklusna realizacija direktnih skokova. Indirektni skokovi su oni koji se izračunavaju u toku run-time-a.
  - Zbo ovoga Cortex-M odskače od tradicionalnih RISC procesora.



Značaj prefetch buffer-a



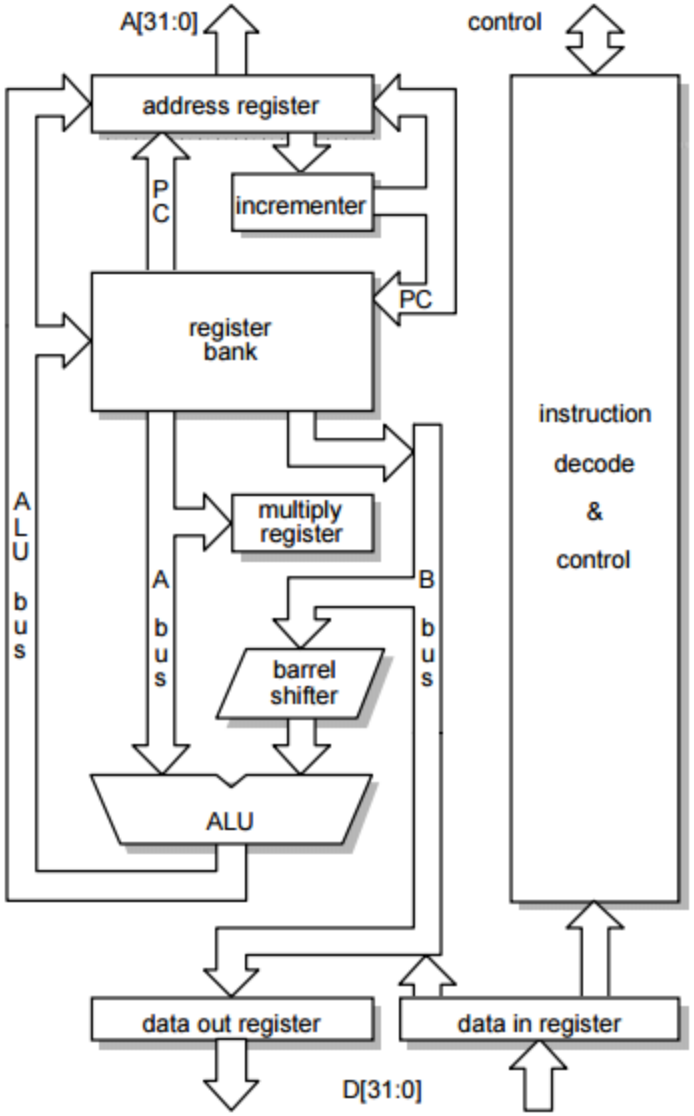
Pipeline



- **Fetch** – učitavanje instrukcije iz memorije i smeštanje u prefetch buffer
- **Decode** – dekodiranje instrukcije i pripremanje kontrolnih signala koij regulišu protok podataka
- **Execute** – Očitavanje registara, pomeranje operanda, generisanje rezultata operacije (ALU) i upisivanje rezultata odredišni registar.

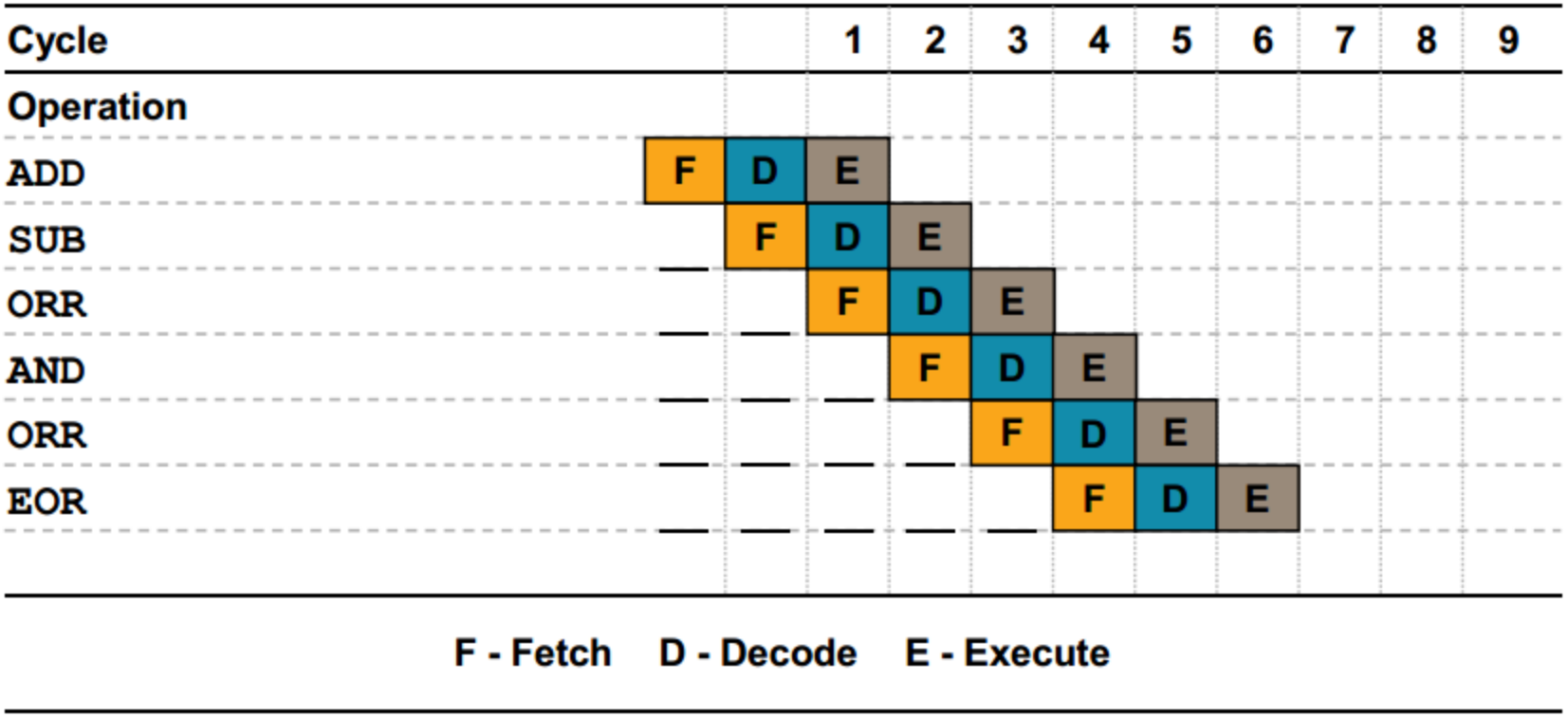
Interna struktura CPU-a

- Registarska banka:
  - 2 porta za čitanje, 1 port za upis
  - 1 dodatni port za čitanje i 1 dodatni port za upis u PC
- Barrel pomerač
  - Pomera za bilo koji broj bita
- ALU
- Adresni registar i inkrementer
- Data registar koji čuva podatke koji se razmenjuju sa memorijom
- Dekoder instrukcija i sa njim povezan kontrolni blok celog sistema

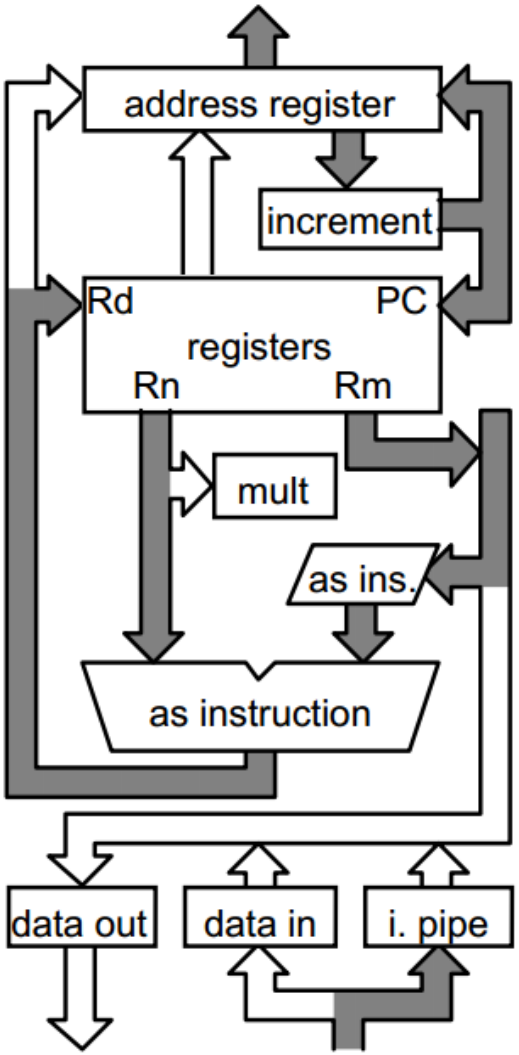


Optimalni pipeline

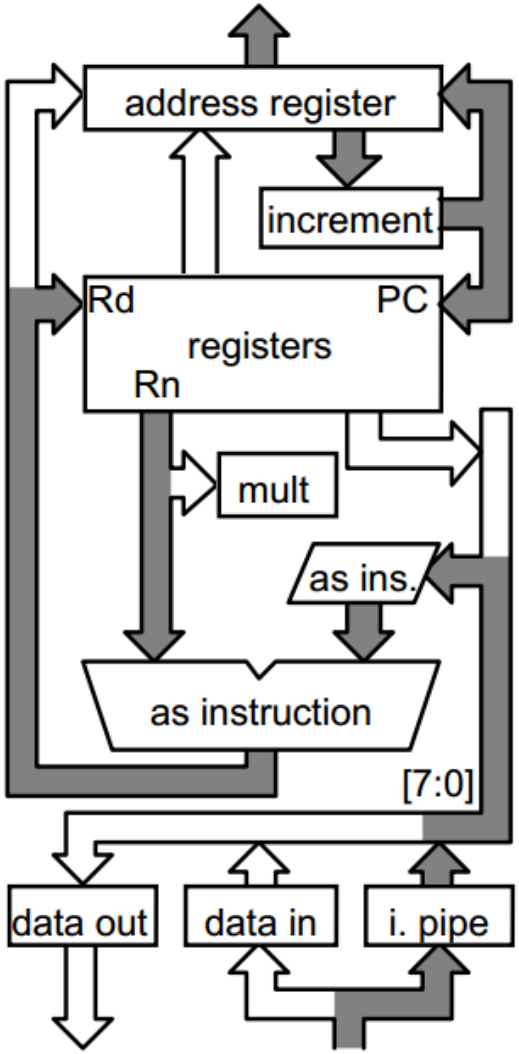
- Sve instrukcije su nad registrima
- Efektivno dobijamo instrukciju po ciklusu



Optimalni pipeline – obrada podataka



(a) register - register operations



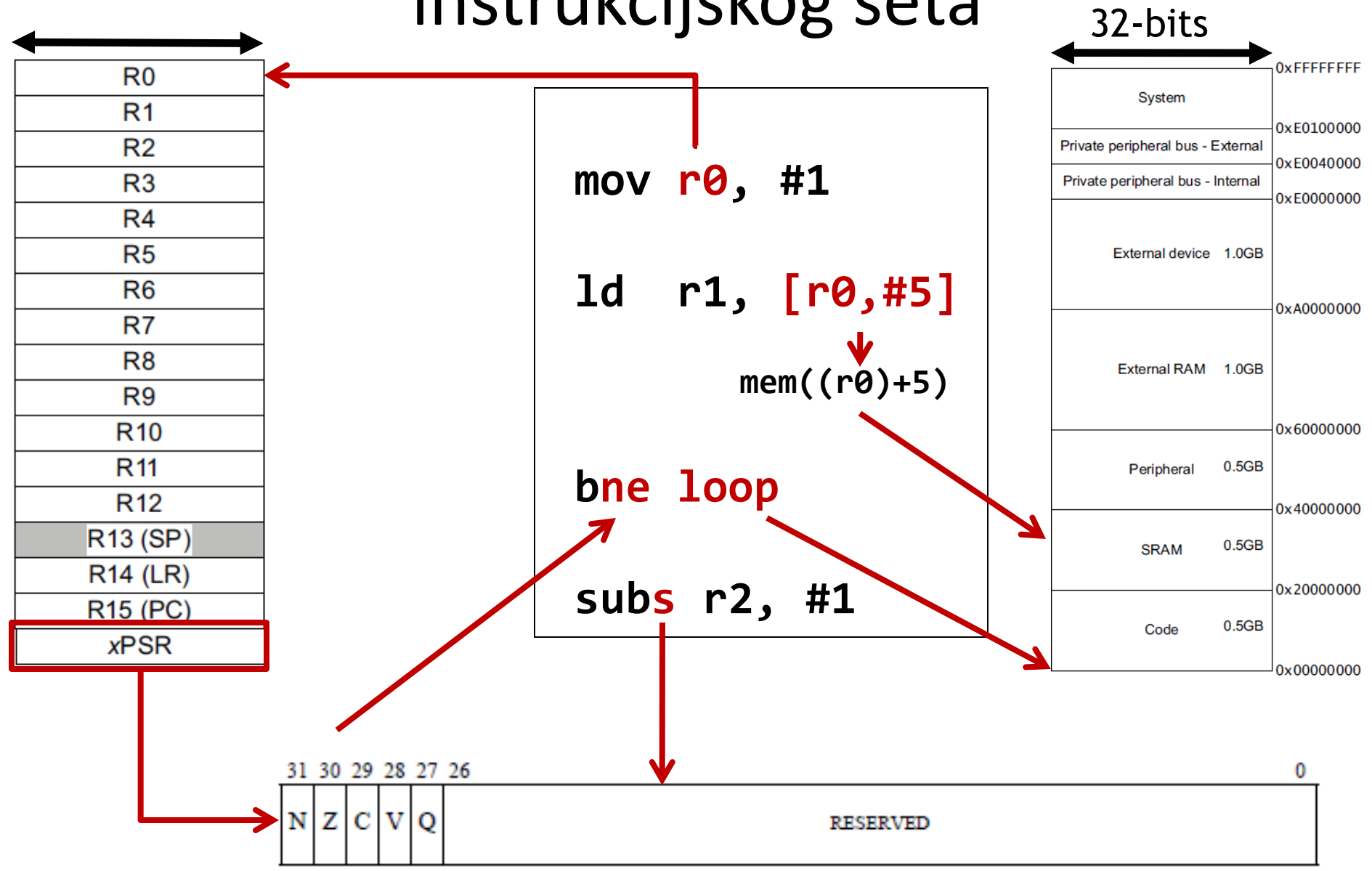
(b) register - immediate operations

Arhitektura instrukcijskog seta
ISA Instruction set architecture

"Instruction set architecture (ISA) is the structure of a computer that a machine language programmer (or a compiler) must understand to write a correct (timing independent) program for that machine"

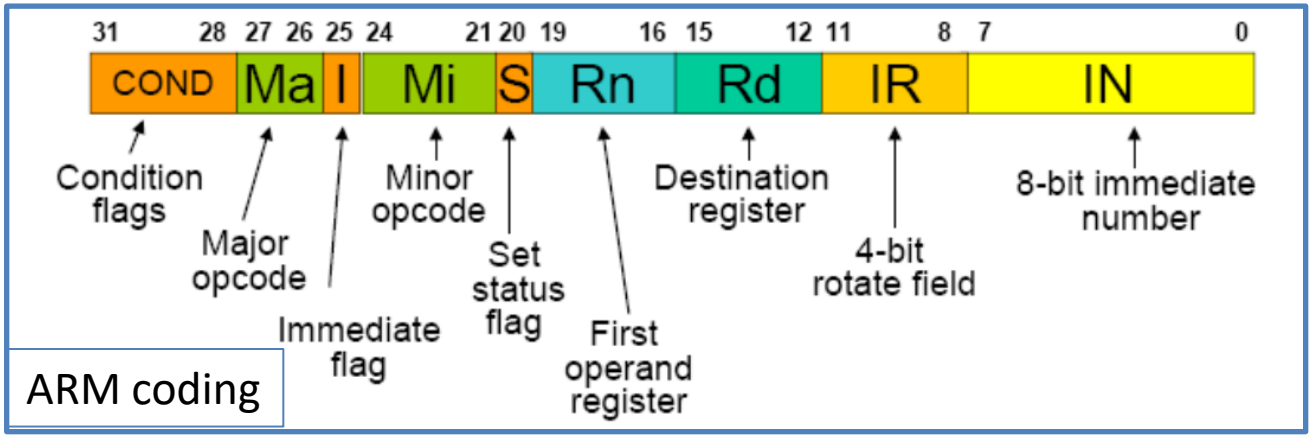
IBM introducing 360 in 1964

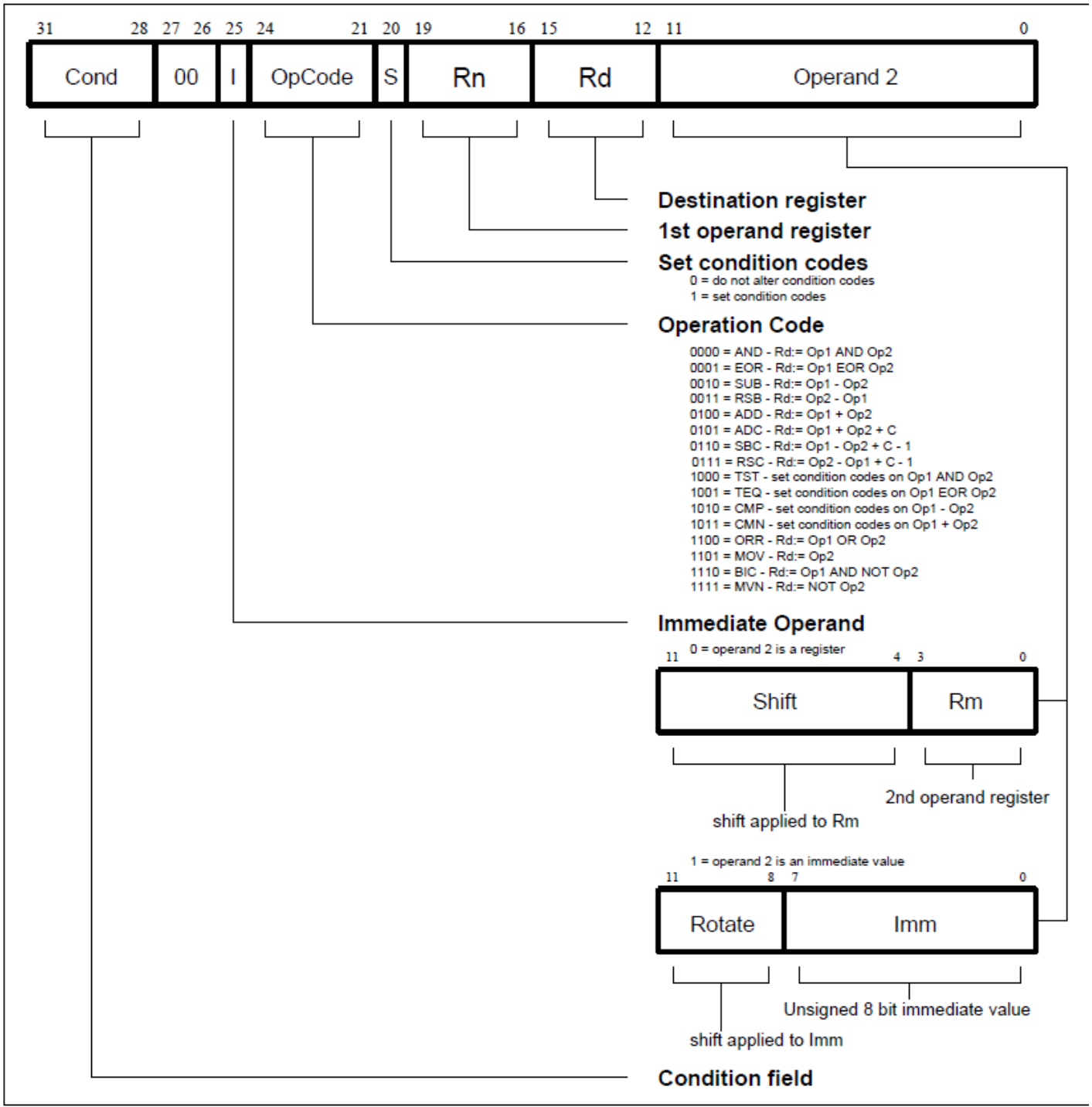
Glavni elementi arhitekture instrukcijskog seta



ARM/Thumb instrukcijski set

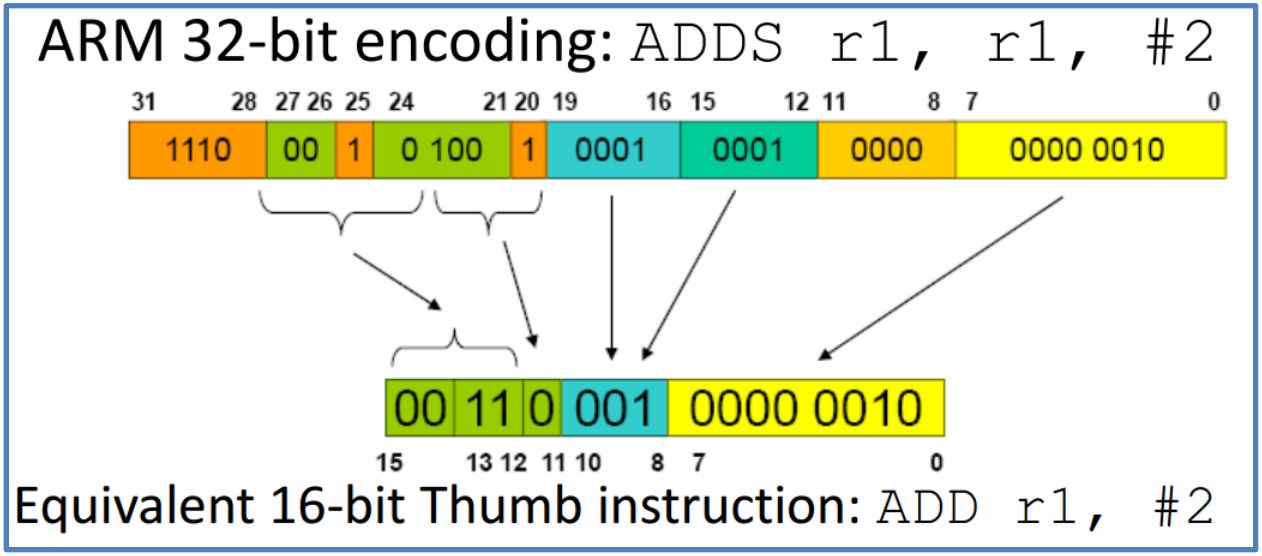
- Tradicionalni ARM set:
  - fiksna dužina od 32 bita
  - Dva ili tri operanda
  - Radi se sa podacima u registrima
  - Memorijske transferi se obavljaju samo kroz Load/Store instrukcije
  - Uslovno izvršavanje instrukcija
- Originalni Thumb instrukcijski set:
  - Implemenaticija određenih instrukcija iz ARM seta, ali sa 16 bita
  - Većina Thumb instrukcija ima dva operanda, tj. Jedan od izvorišnih je i odredišni operand
  - Nisu dostupni registri r8 do r15
  - Nema uslovnog izvršavanja




31 28 27 26 25 24 21 20 19 16 15 12 11 0
Cond
00
I
OpCode
S
Rn
Rd
Operand 2
Destination register
1st operand register
Set condition codes
0 = do not alter condition codes
1 = set condition codes
Operation Code
0000 = AND - Rd:= Op1 AND Op2
0001 = EOR - Rd:= Op1 EOR Op2
0010 = SUB - Rd:= Op1 - Op2
0011 = RSB - Rd:= Op2 - Op1
0100 = ADD - Rd:= Op1 + Op2
0101 = ADC - Rd:= Op1 + Op2 + C
0110 = SBC - Rd:= Op1 - Op2 + C - 1
0111 = RSC - Rd:= Op2 - Op1 + C - 1
1000 = TST - set condition codes on Op1 AND Op2
1001 = TEQ - set condition codes on Op1 EOR Op2
1010 = CMP - set condition codes on Op1 - Op2
1011 = CMN - set condition codes on Op1 + Op2
1100 = ORR - Rd:= Op1 OR Op2
1101 = MOV - Rd:= Op2
1110 = BIC - Rd:= Op1 AND NOT Op2
1111 = MVN - Rd:= NOT Op2
Immediate Operand
0 = operand 2 is a register
11 4 3 0
Shift
Rm
shift applied to Rm
2nd operand register
1 = operand 2 is an immediate value
11 8 7 0
Rotate
Imm
shift applied to Imm
Unsigned 8 bit immediate value
Condition field

ARM/Thumb instrukcijski set

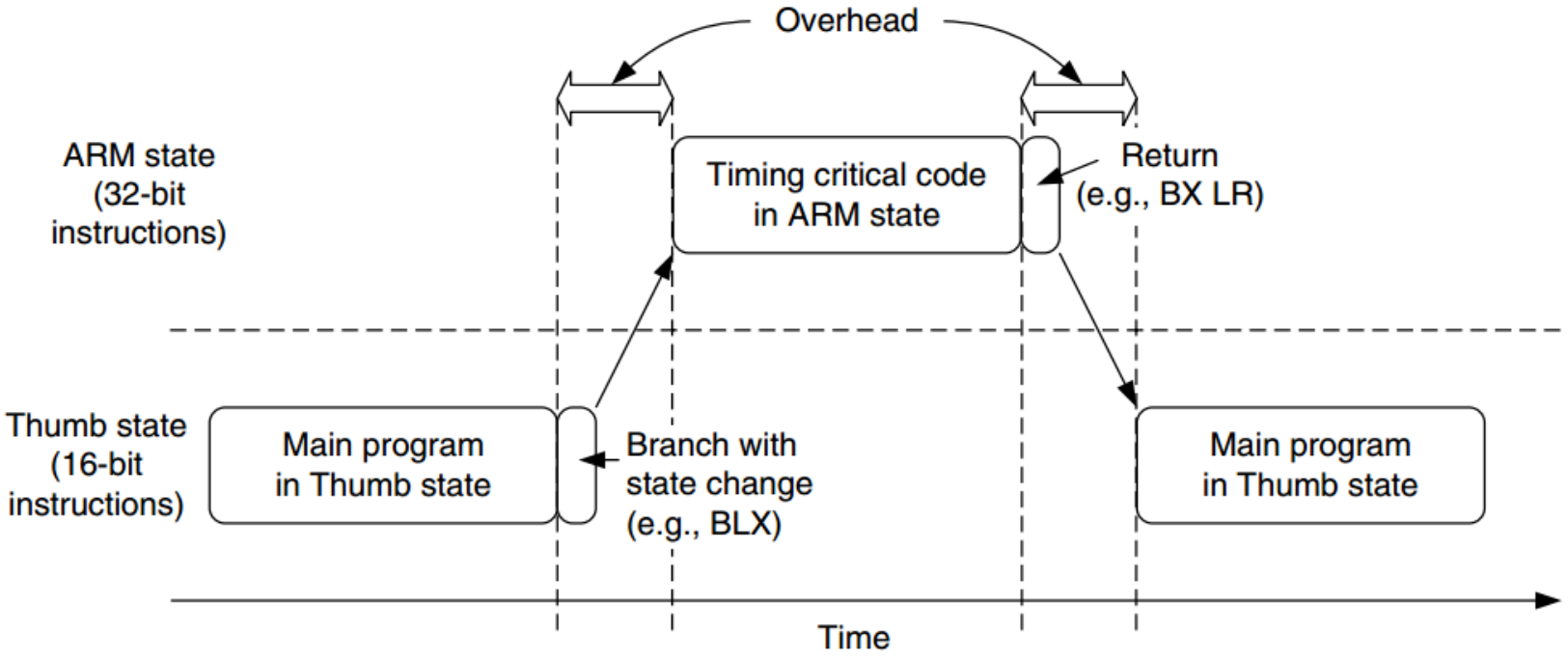
- Primer implementacije instrukcije sabiranja na ARM i Thumb formatu:



-nema uslovnog izvršavanja
-nema rotacije neposrednog operanda
-statusni flagovi se uvek postavljaju (nema S)

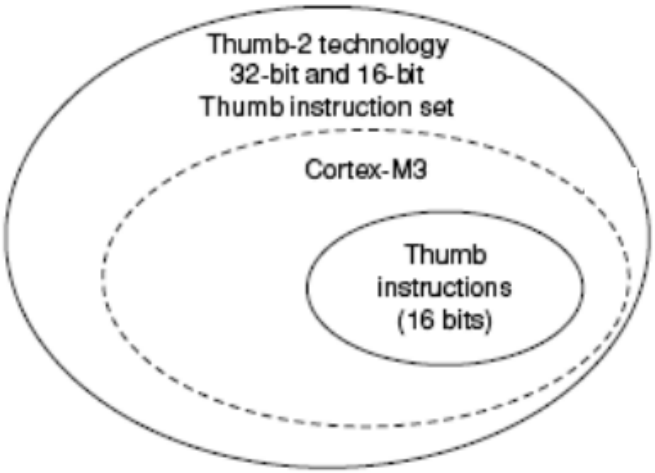
Prelaz sa ARM na Thumb i obrnuto

- Kod ranijih ARM arhitektura prebacivanje sa jednog na drugi instrukcijski set je posedovalo značajan overhead.

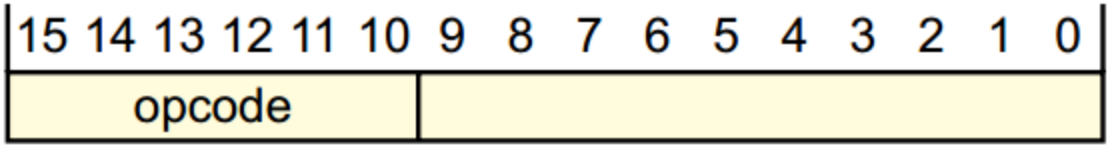


Thumb-2

- Thumb-2 obuhvata i 16-bitne i 32-bitne instrukcije.
- Napravljen je sa osnovnom idejom da se izbegne potreba za izmenom režima rada procesora.
- Thumb-2 je zadržao skoro sve insturkcije starijeg Thumb seta i implementirao većinu instrukcija iz ARM seta.
- Veći deo instrukcija može da se prevede i u 16-bitni i u 32-bitni format u zavisnosti od parametara konkretne instrukcije.
- Odluku o implementaciji donosi kompajler osim u slučaju kada se implicitno traži jedna ili druga implementacija.

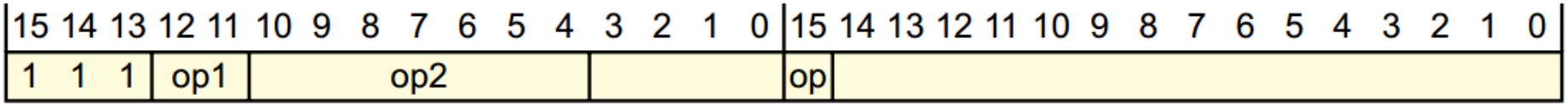


16-bitne Thumb-2 instrukcije



opcode	Instruction or instruction class
00xxxx	*Shift (immediate), add, subtract, move, and compare* on page A5-157
010000	*Data processing* on page A5-158
010001	*Special data instructions and branch and exchange* on page A5-159
01001x	Load from Literal Pool, see *LDR (literal)* on page A7-289
0101xx 011xxx 100xxx	*Load/store single data item* on page A5-160
10100x	Generate PC-relative address, see *ADR* on page A7-229
10101x	Generate SP-relative address, see *ADD (SP plus immediate)* on page A7-225
1011xx	*Miscellaneous 16-bit instructions* on page A5-161
11000x	Store multiple registers, see *STM, STMIA, STMEA* on page A7-469
11001x	Load multiple registers, see *LDM, LDMIA, LDMFD* on page A7-283
1101xx	*Conditional branch, and supervisor call* on page A5-163
11100x	Unconditional Branch, see *B* on page A7-239

32-bitne Thumb-2 instrukcije



- Ako učitana reč (16 bita) sadrži najviših 5 bitova sledećeg sadržaja:
  - 0b11101
  - 0b11110
  - 0b11111

  instrukcija je 32-bitna, tj. Dekodira se i sledeća reč.

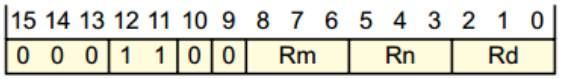
op1	op2	op	Instruction class
01	00xx0xx	x	*Load Multiple and Store Multiple* on page A5-171
01	00xx1xx	x	*Load/store dual or exclusive, table branch* on page A5-172
01	01xxxxx	x	*Data processing (shifted register)* on page A5-179
01	1xxxxxx	x	*Coprocessor instructions* on page A5-188
10	x0xxxxx	0	*Data processing (modified immediate)* on page A5-165
10	x1xxxxx	0	*Data processing (plain binary immediate)* on page A5-168
10	xxxxxxx	1	*Branches and miscellaneous control* on page A5-169
11	000xxx0	x	*Store single data item* on page A5-178
11	00xx001	x	*Load byte, memory hints* on page A5-176
11	00xx011	x	*Load halfword, memory hints* on page A5-174
11	00xx101	x	*Load word* on page A5-173
11	00xx111	x	UNDEFINED
11	010xxxx	x	*Data processing (register)* on page A5-181
11	0110xxx	x	*Multiply, multiply accumulate, and absolute difference* on page A5-186
11	0111xxx	x	*Long multiply, long multiply accumulate, and divide* on page A5-187
11	1xxxxxx	x	*Coprocessor instructions* on page A5-188

16 ili 32 bita?

- Primer instrukcije ADD (sabiranje) kada se radi nad registrima.
- U zavisnosti od upotrebljenih registara ili opcionih polja u instrukciji moguće su različite vrste kodiranja instrukcije.
- To se u krajnjoj liniji reflektuje na gustinu koda.

Encoding T1 All versions of the Thumb instruction set.

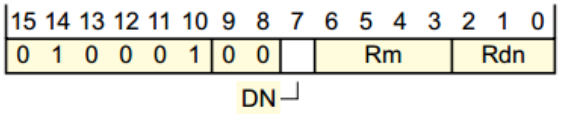
```
ADDS <Rd>,<Rn>,<Rm>                Outside IT block.
ADD<c> <Rd>,<Rn>,<Rm>              Inside IT block.
```




```
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  setflags = !InITBlock();
(shift_t, shift_n) = (SRType_LSL, 0);
```

Encoding T2 All versions of the Thumb instruction set.

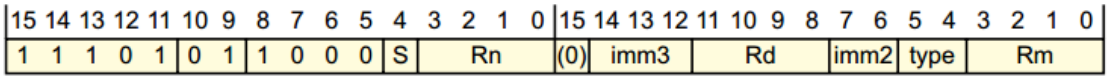
```
ADD<c> <Rdn>,<Rm>
```




```
if (DN:Rdn) == '1101' || Rm == '1101' then SEE ADD (SP plus register);
d = UInt(DN:Rdn);  n = UInt(DN:Rdn);  m = UInt(Rm);  setflags = FALSE;
(shift_t, shift_n) = (SRType_LSL, 0);
if d == 15 && InITBlock() && !LastInITBlock() then UNPREDICTABLE;
if d == 15 && m == 15 then UNPREDICTABLE;
```

Encoding T3 ARMv7-M

```
ADD{S}<c>.W <Rd>,<Rn>,<Rm>{,<shift>}
```



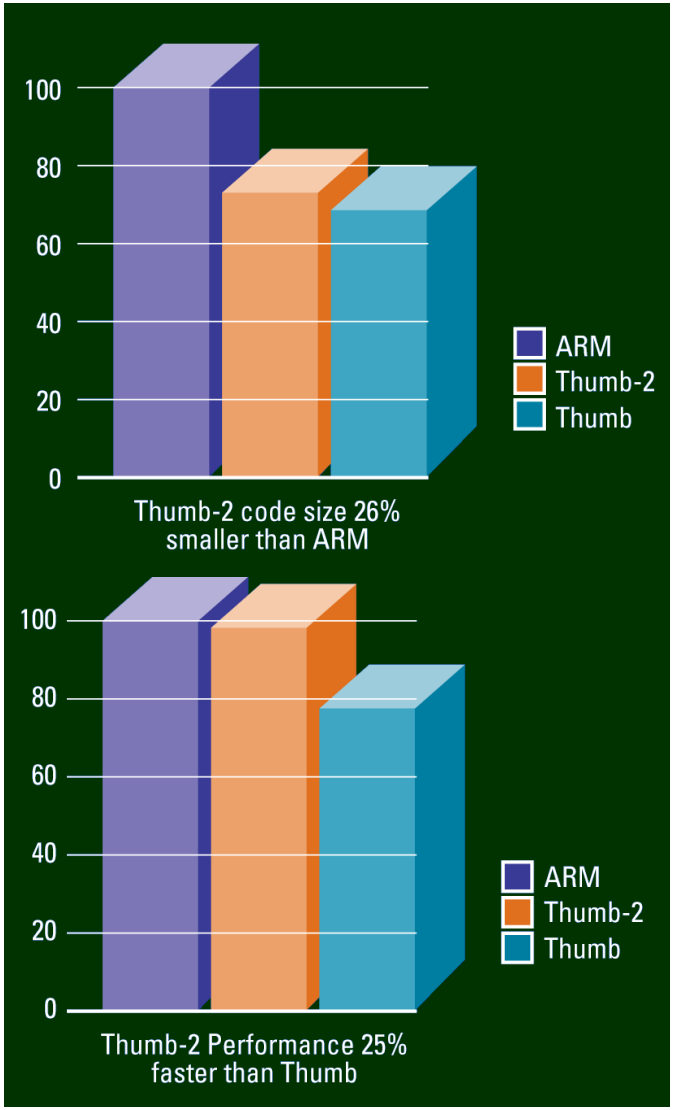

```
if Rd == '1111' && S == '1' then SEE CMN (register);
if Rn == '1101' then SEE ADD (SP plus register);
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  setflags = (S == '1');
(shift_t, shift_n) = DecodeImmShift(type, imm3:imm2);
if d == 13 || (d == 15 && S == '0') || n == 15 || m IN {13,15} then UNPREDICTABLE;
```

Poređenje performansi instrukcijskih setova

- Instrukcijski set promenljive dužine
  - ARM instrukcije su bile isključivo 32-bitne
  - Thumb instrukcije su bile isključivo 16-bitne
  - Thumb-2 instrukcije mogu da budu i 16-bitne i 32-bitne,

- Thumb-2 daje za oko 26% veću gustinu koda od ARM

- Thumb-2 daje za oko 25% veće performanse u odnosu na Thumb

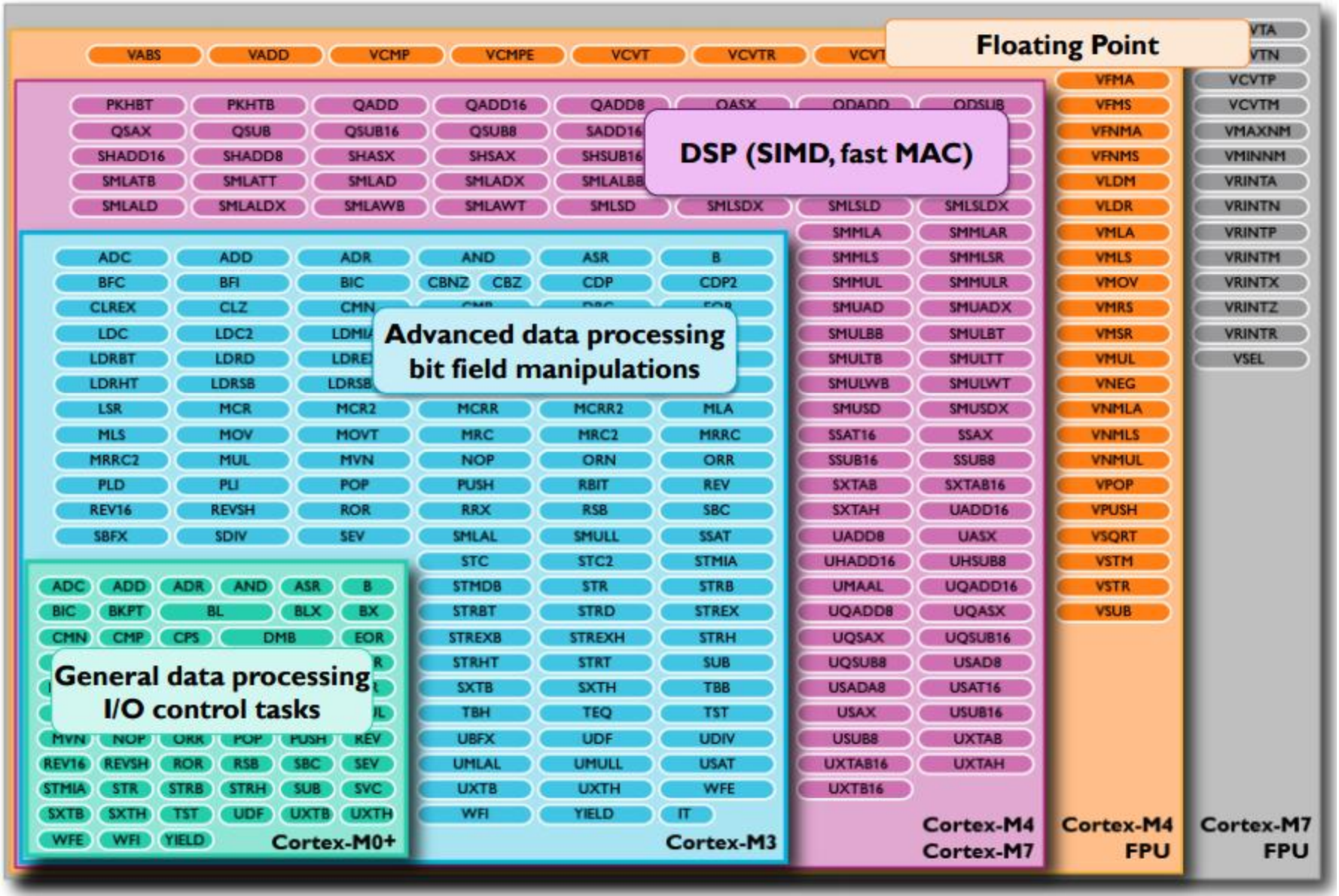


Instrukcijski set kod Cortex-M processora

Instruction Groups	Cortex-M0, M1	Cortex-M3	Cortex-M4	Cortex-M4 with FPU
16-bit ARMv6-M instructions	●	●	●	●
32-bit Branch with Link instruction	●	●	●	●
32-bit system instructions	●	●	●	●
16-bit ARMv7-M instructions		●	●	●
32-bit ARMv7-M instructions		●	●	●
DSP extensions			●	●
Floating point instructions				●

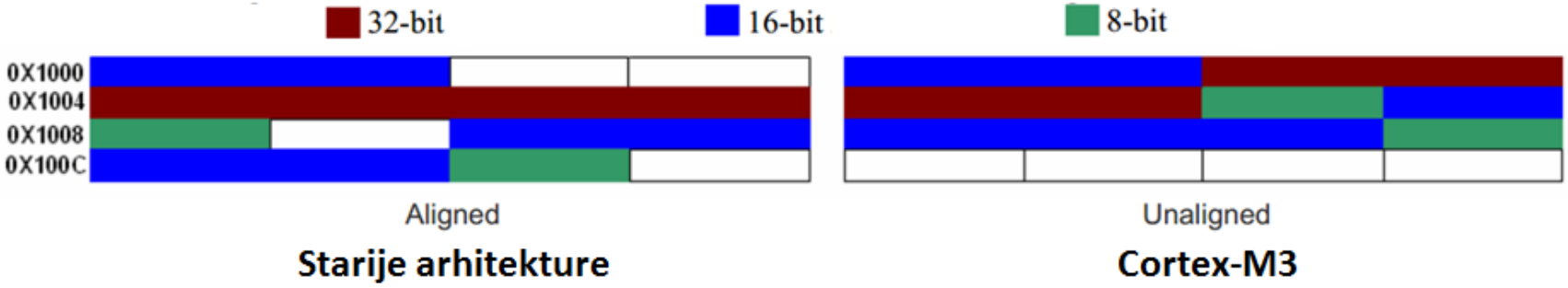
Sa stanovišta korisnika Cortex-M4 se razlikuje od Cortex-M3 procesora u proširenom instrukcijskom setu

Cortex-M instrukcijski set - poređenje



Tipovi podataka

- ARM podržava tri tipa podataka:
  - Byte – 8-bitni podatak
  - Halfword – 16-bitni podatak
  - Word – 32-bitni podatak
- Pristup memoriji kod Cortex-M3 omogućava takozvani neporavnat (unaligned) pristup.



- Ali to ne važi kod nekih instrukcija. Na primer PUSH i POP rade samo aligned acces. Takođe i specijalne instrukcije za prenos blokova podataka.