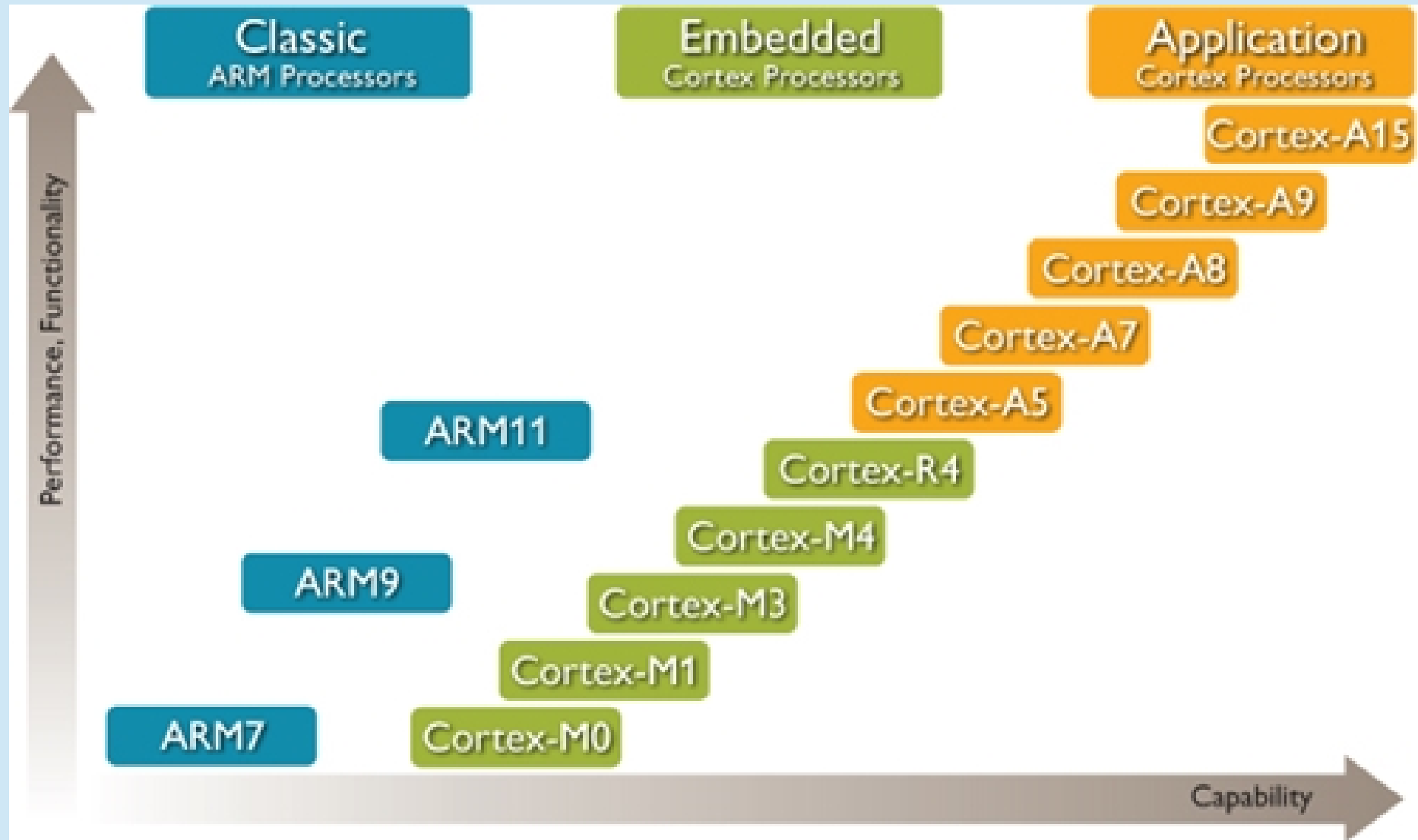


# Architektura ARM

# Přehled architektur ARM



# Velikosti dat a instrukční soubor

**ARM je 32-bitová architektura**

**Von-Neumannova architektura (společná paměťová oblast pro program i data)**

**Velikosti datových objektů:**

Byte ... 8 bitů

Halfword ... 16 bitů (2 byty)

Word ... 32 bitů (4 byty)

**Implementována jedna ze dvou nebo obě instrukční sady**

32-bitová ARM instrukční sada

16-bit Thumb (redukovaná) instrukční sada

**Jazelle jádra navíc umí přímo spustit Java bytecode (Android std. Aplikace,...)**

## ARM podporuje sedm operačních režimů

- **User** : neprivilegovaný režim, běh standardních uživatelských úloh
- **FIQ** : Fast Interrupt – nastaven při vstupu do obsluhy rychlého přerušení (nejvyšší priorita, externí vstup)
- **IRQ** : Interrupt – nastaven při vstupu do obsluhy standardního přerušení – externí události, události interních periférií...
- **Supervisor** : výchozí stav RESET a po provedení instrukce Software Interrupt
- **Abort** : obsluha chyby přístupu do paměti (mimo mapovanou paměť, adresa nesoudělná s délkou slova...)
- **Undef** : obsluha chyby při výskytu nedefinované instrukce
- **System** : privilegovaný režim, používá stejné registry jako režim User

# Soubor registrů ARM

## Viditelné registry

Abort Mode

|          |
|----------|
| r0       |
| r1       |
| r2       |
| r3       |
| r4       |
| r5       |
| r6       |
| r7       |
| r8       |
| r9       |
| r10      |
| r11      |
| r12      |
| r13 (sp) |
| r14 (lr) |
| r15 (pc) |
| cpsr     |
| spsr     |

## Skryté registry

User

FIQ

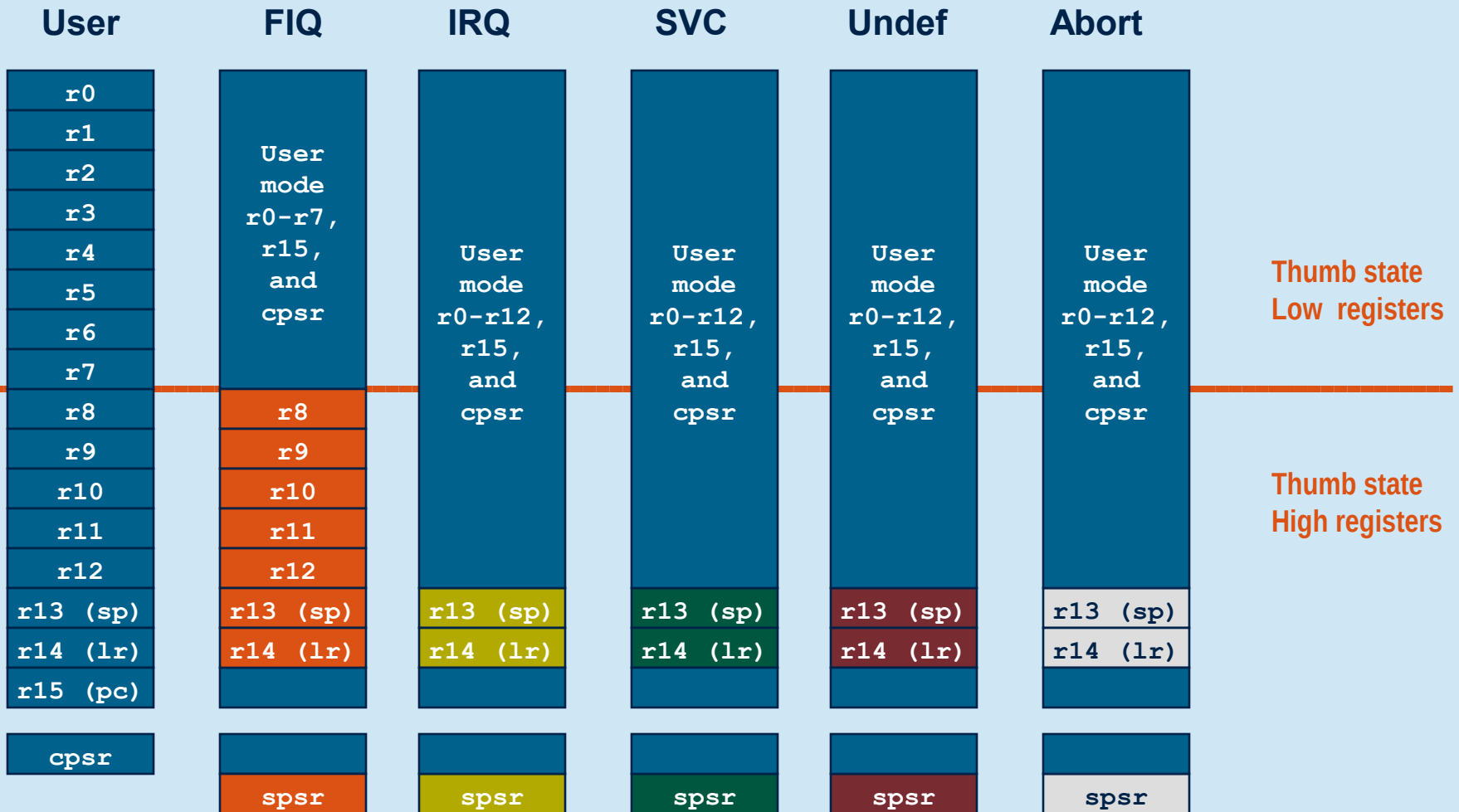
IRQ

SVC

Undef

|          |          |          |          |          |
|----------|----------|----------|----------|----------|
|          | r8       |          |          |          |
|          | r9       |          |          |          |
|          | r10      |          |          |          |
|          | r11      |          |          |          |
|          | r12      |          |          |          |
| r13 (sp) | r13 (sp) | r13 (sp) | r13 (sp) | r13 (sp) |
| r14 (lr) | r14 (lr) | r14 (lr) | r14 (lr) | r14 (lr) |
|          |          |          |          |          |
|          | spsr     | spsr     | spsr     | spsr     |

# Přehled organizace registrů



Pozn.: Režim System používá registrovou sadu jako režim User

# Vlastnosti registrů

## ARM obsahuje 37 registrů s šířkou slova 32 bitů

- 1 program counter
- 1 current program status register
- 5 saved program status registers
- 30 registrů pro obecné použití

## Dostupnost registrů závisí na operačním módu

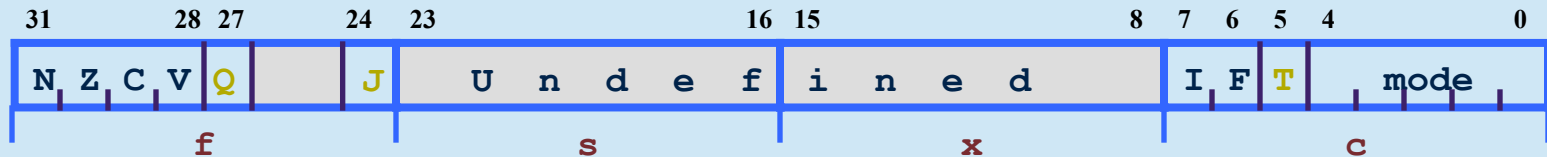
### V každém režimu jsou dostupné registry

- společná sada **r0-r12** (kromě FIQ, tam jsou vlastní **r8-r12**)
- individuální **r13** (stack pointer, **sp**) a **r14** (link register, **lr**, návratová adresa)
- program counter, **r15** (**pc**)
- current program status register, **cpsr**

### Privilegované režimy (kromě System) mají přístupný

- individuální **spsr** (saved program status register)

# Program Status Registers



## Condition code flags

- N = **N**egative result from ALU
- Z = **Z**ero result from ALU
- C = ALU operation **C**arried out
- V = ALU operation o**V**erflowed

## Sticky Overflow flag - Q flag

- Architecture 5TE/J only
- Indicates if saturation has occurred

## J bit

- Architecture 5TEJ only
- J = 1: Processor in Jazelle state

## Interrupt Disable bits.

- I = 1: Disables the IRQ.
- F = 1: Disables the FIQ.

## T Bit

- Architecture xT only
- T = 0: Processor in ARM state
- T = 1: Processor in Thumb state

## Mode bits

- Specify the processor mode



# Program Counter (r15)

## Při provádění instrukcí ARM:

- Šířka všech instrukcí je 32 bitů
- Všechny instrukce musí začínat na adrese dělitelné čtyřmi
- V důsledku toho je hodnota **pc** uložena v bitech [31:2] a bity [1:0] mají nedefinovaný obsah

## Při provádění instrukcí Thumb:

- Šířka všech instrukcí je 16 bitů, (některé jsou rozděleny na více slov 16 bitů)
- Všechny instrukce musí začínat na adrese dělitelné dvěma
- V důsledku toho je hodnota **pc** uložena v bitech [31:1] a bity [0:0] mají nedefinovaný obsah

# Zpracování výjimek

**Při výskytu výjimky je automaticky provedeno:**

- Zkopírování CPSR do SPSR\_<mode>
- Nastavení bitů v CPSR
  - Nastavení ARM stavu
  - Změna režimu
  - Zákaz přerušení
- Uložení návratové adresy do LR\_<mode>
- Nastavení PC na adresu vektoru

**Při návratu z obsluhy musí být provedeno:**

- Obnova CPSR ze SPSR\_<mode>
- Obnova PC z LR\_<mode>

**Toto smí být provedeno pouze v ARM stavu**

|      |                       |
|------|-----------------------|
|      |                       |
| 0x1C | FIQ                   |
| 0x18 | IRQ                   |
| 0x14 | (Reserved)            |
| 0x10 | Data Abort            |
| 0x0C | Prefetch Abort        |
| 0x08 | Software Interrupt    |
| 0x04 | Undefined Instruction |
| 0x00 | Reset                 |

**Tabulka vektorů**

# Instrukční sada

Provedení všech ARM instrukcí může být podmíněno postfixem, který vyjadřuje kombinaci stavových bitů (CPSR)

- Snižuje počet instrukcí a zejména skoků (rychlost)

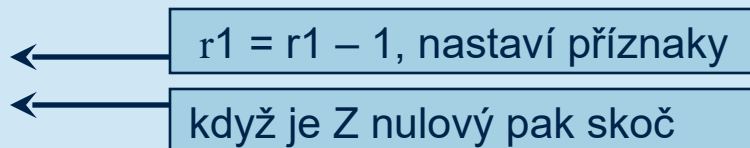
```
CMP    r3,#0
BEQ    skip
ADD    r0,r1,r2
skip:
```

```
CMP    r3,#0
ADDNE  r0,r1,r2
```



Defaultně nejsou ovlivňovány stavové bity, toto musí být explicitně vynuceno postfixem “S”. (kromě instrukce CMP)

```
loop
...
SUBS  r1,r1,#1
BNE  loop
```



# Condition Codes

## Seznam možných podmínek:

- AL je defaultní a nemusí být uvedena

| Suffix       | Description             | Flags tested         |
|--------------|-------------------------|----------------------|
| <b>EQ</b>    | Equal                   | <b>Z=1</b>           |
| <b>NE</b>    | Not equal               | <b>Z=0</b>           |
| <b>CS/HS</b> | Unsigned higher or same | <b>C=1</b>           |
| <b>CC/LO</b> | Unsigned lower          | <b>C=0</b>           |
| <b>MI</b>    | Minus                   | <b>N=1</b>           |
| <b>PL</b>    | Positive or Zero        | <b>N=0</b>           |
| <b>VS</b>    | Overflow                | <b>V=1</b>           |
| <b>VC</b>    | No overflow             | <b>V=0</b>           |
| <b>HI</b>    | Unsigned higher         | <b>C=1 &amp; Z=0</b> |
| <b>LS</b>    | Unsigned lower or same  | <b>C=0 or Z=1</b>    |
| <b>GE</b>    | Greater or equal        | <b>N=V</b>           |
| <b>LT</b>    | Less than               | <b>N!=V</b>          |
| <b>GT</b>    | Greater than            | <b>Z=0 &amp; N=V</b> |
| <b>LE</b>    | Less than or equal      | <b>Z=1 or N!=V</b>   |
| <b>AL</b>    | Always                  |                      |

# Příklady podmíněných instrukcí

## Použití sekvence stejně podmíněných instrukcí

```
if (a==0) func(1);  
    CMP      r0,#0  
    MOVEQ    r0,#1  
    BLEQ     func
```

## Použití sekvence různě podmíněných instrukcí

```
if (a==0) x=0;  
if (a>0)  x=1;  
    CMP      r0,#0  
    MOVEQ    r1,#0  
    MOVGT    r1,#1
```

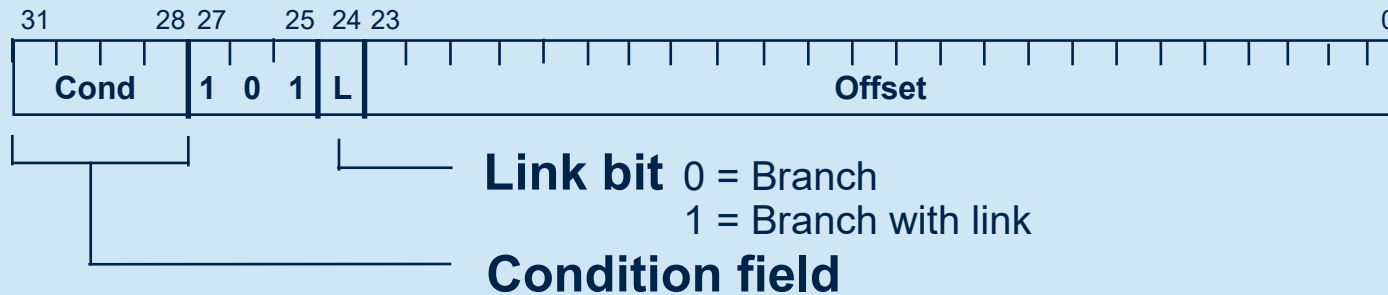
## Použití podmíněné instrukce CMP

```
if (a==4 || a==10) x=0;  
    CMP      r0,#4  
    CMPNE    r0,#10  
    MOVEQ    r1,#0
```

# Instrukce skoku

Branch : **B{<cond>} label**

Branch with Link : **BL{<cond>} subroutine\_label**



Procesorové jádro posune pole offset o 2 bity vlevo, provede znaménkové rozšíření na 32 bitů a přičte k PC

- Rozsah  $\pm 32$  Mbyte
- Jak provést delší skok?

# Instrukce pro operace s daty

## Přehled :

|                |     |     |     |     |     |     |
|----------------|-----|-----|-----|-----|-----|-----|
| ■ Aritmetické: | ADD | ADC | SUB | SBC | RSB | RSC |
| ■ Logické:     | AND | ORR | EOR | BIC |     |     |
| ■ Porovnání :  | CMP | CMN | TST | TEQ |     |     |
| ■ Přesun:      | MOV | MVN |     |     |     |     |

Instrukce pracují pouze s registry, NE s pamětí

## Syntaxe:

**<Operation>{<cond>}{S} Rd, Rn, Operand2**

- Instrukce porovnání pouze nastaví příznakové bity – nobsahují Rd
- Instrukce pro přesun neobsahují Rn

Druhý operand je pro ALU předzpracován jednotkou barrel shifter.

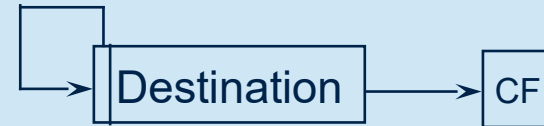
# Barrel Shifter

## LSL : Logical Left Shift



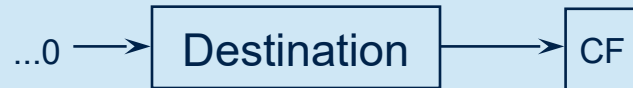
Násobení mocninou 2

## ASR: Arithmetic Right Shift



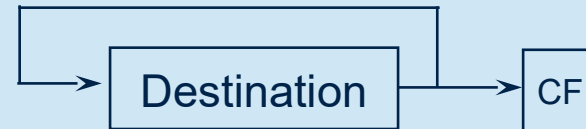
Division by a power of 2,  
preserving the sign bit

## LSR : Logical Shift Right



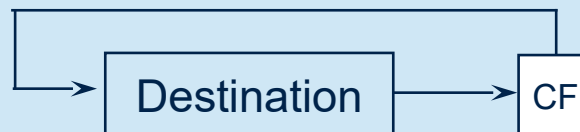
Dělení mocninou 2

## ROR: Rotate Right



Bitová rotace s přenosem z LSB do MSB

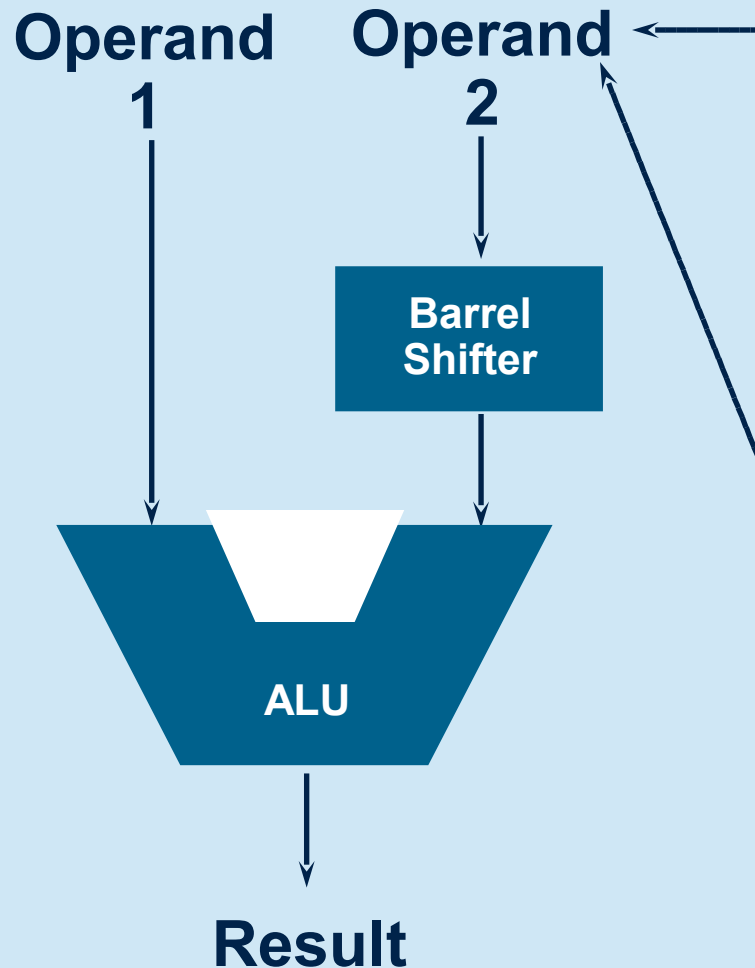
## RRX: Rotate Right Extended



Bitová rotace přes CF



# Použití Barrel Shifter: (druhý operand)



## Registr, volitelne s operací posunu

- Hodnota posunu může být:
  - 5 bitová neznaménková hodnota
  - Hodnota spodního bytu jiného registru
- Používáno pro násobení konstantou

## Přímá hodnota

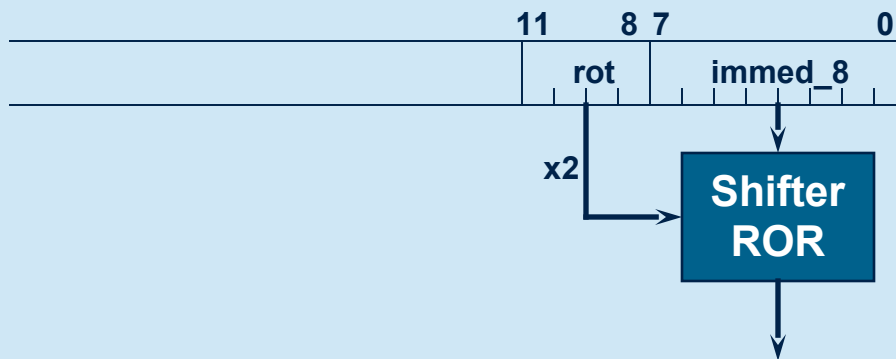
- osmibitové číslo, rozsah 0-255.
  - Možnost bitové rotace vpravo o sudý počet bitů
- Umožňuje zadání většího rozsahu 32-bitových konstant (značně omezený)

# Přímé konstanty (1)

ARM instrukce nemůže obsahovat 32 bitovou konstantu

- Všechny ARM instrukce jsou dlouhé 32 bitů

Instrukce po datové operaci mají k dispozici 12 bitů pro druhý operand

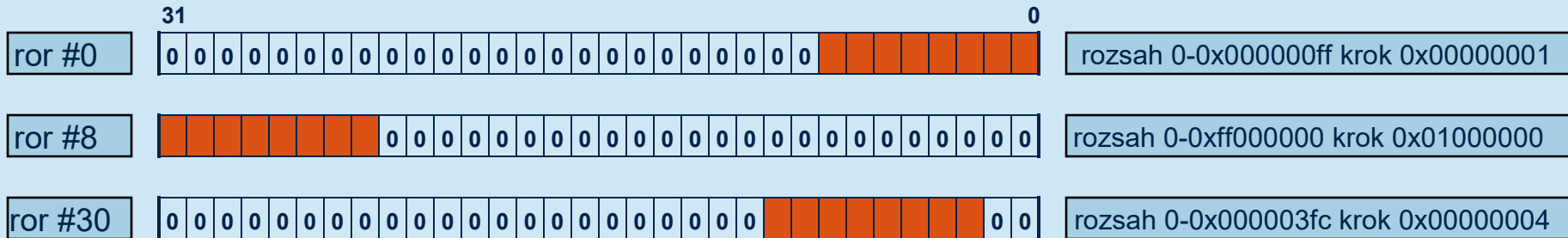


4 bit rotate value (0-15) is multiplied by two to give range 0-30 in steps of 2

Rule to remember is “8-bits shifted by an even number of bit positions”.

# Přímé konstanty (2)

## Examples:



## Assembler převede zápis konstant na použití rotací konstanty:

- `MOV r0, #4096` ; uses 0x40 ror 26
- `ADD r1, r2, #0xFF0000` ; uses 0xFF ror 16

## Bitový doplněk může být zapsán s použitím instrukce MVN:

- `MOV r0, #0xFFFFFFFF` ; přeloženo jako `MVN r0, #0`

Hodnoty, které nemohou být takto převedeny způsobí chybové hlášení překladače

# 32-bitové konstanty

Pro obecné použití konstant s vyššími hodnotami je v assembleru zavedena pseudo-instrukce:

- `LDR rd, =const`


Která zajistí:

- Použití `MOV` or `MVN` pokud je to možné.

nebo

- Generování `LDR` instrukce s adresou relativní k PC, načte hodnotu přímo ze 32b slova z paměti

Příklady:

- |                                   |    |                                 |
|-----------------------------------|----|---------------------------------|
| ■ <code>LDR r0,=0xFF</code>       | => | <code>MOV r0,#0xFF</code>       |
| ■ <code>LDR r0,=0x55555555</code> | => | <code>LDR r0,[PC,#Imm12]</code> |
|                                   |    | ...                             |
|                                   |    | ...                             |
|                                   |    | <code>DCD 0x55555555</code>     |
- 

## Syntaxe:

- **MUL**{<cond>}{S} Rd, Rm, Rs
- **MLA**{<cond>}{S} Rd, Rm, Rs, Rn
- **[U|S]MULL**{<cond>}{S} RdLo, RdHi, Rm, Rs
- **[U|S]MLAL**{<cond>}{S} RdLo, RdHi, Rm, Rs

$Rd = Rm * Rs$

$Rd = (Rm * Rs) + Rn$

$RdHi, RdLo := Rm * Rs$

$RdHi, RdLo := (Rm * Rs) + RdHi, RdLo$

## Čas instrukce

- Základní instrukce MUL
  - 2-5 cyklů v ARM7TDMI
  - 1-3 cykly v StrongARM/XScale
  - 2 cykly v ARM9E/ARM102xE
- +1 cykl pro ARM9TDMI (oproti ARM7TDMI)
- +1 cykl pro součet
- +1 cykl pro verzi “long”

# Přesun dat Registr <> Paměť

|              |             |                      |
|--------------|-------------|----------------------|
| <b>LDR</b>   | <b>STR</b>  | Word                 |
| <b>LDRB</b>  | <b>STRB</b> | Byte                 |
| <b>LDRH</b>  | <b>STRH</b> | Halfword             |
| <b>LDRSB</b> |             | Signed byte load     |
| <b>LDRSH</b> |             | Signed halfword load |

**Paměť musí podporovat všechny šířky slov**

**Syntax:**

- **LDR**{<cond>}{<size>} Rd, <address>
- **STR**{<cond>}{<size>} Rd, <address>

e.g. **LDREQB**

**Adresa pro instrukci LDR/STR je zapsána jako bazový registr plus offset**

**Pro přístup WORD a neznaménkový BYTE**

- Přímá hodnota 12 bitů (tzn. 0 - 4095 bytů).  
`LDR r0, [r1, #8]`
- Registr bitově posunutý o přímo zadanou hodnotu  
`LDR r0, [r1, r2]`  
`LDR r0, [r1, r2, LSL#2]`

**Hodnota může být přičtena nebo odečtena s bazovým registrem:**

```
LDR r0, [r1, #-8]
LDR r0, [r1, -r2]
LDR r0, [r1, -r2, LSL#2]
```

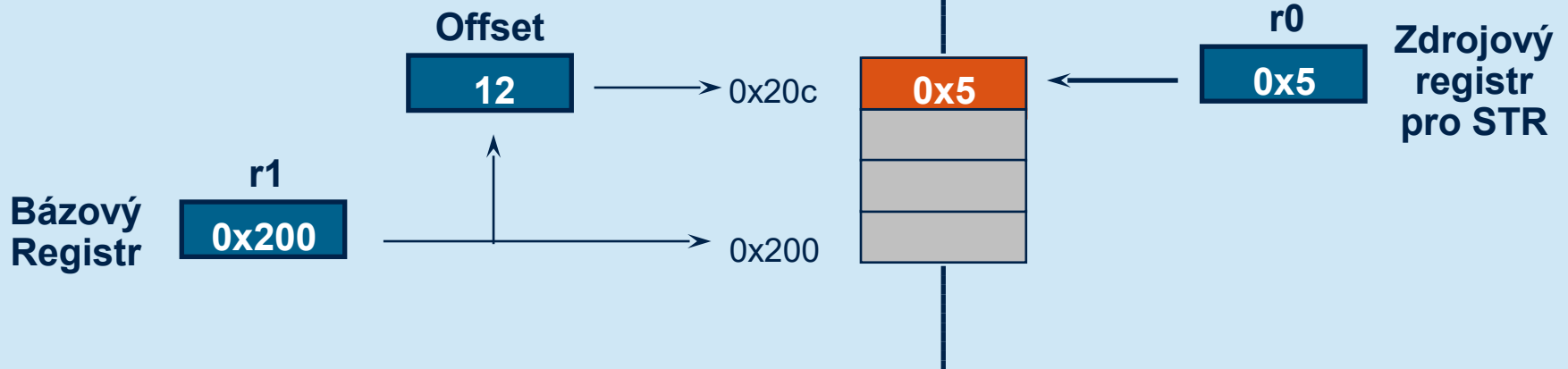
**Pro halfword a znaménkový halfword / byte, může být offset:**

- Přímá neznaménková osmibitová hodnota (tzn. 0-255 bytů).
- Registr (bez možnosti bitového posunu).

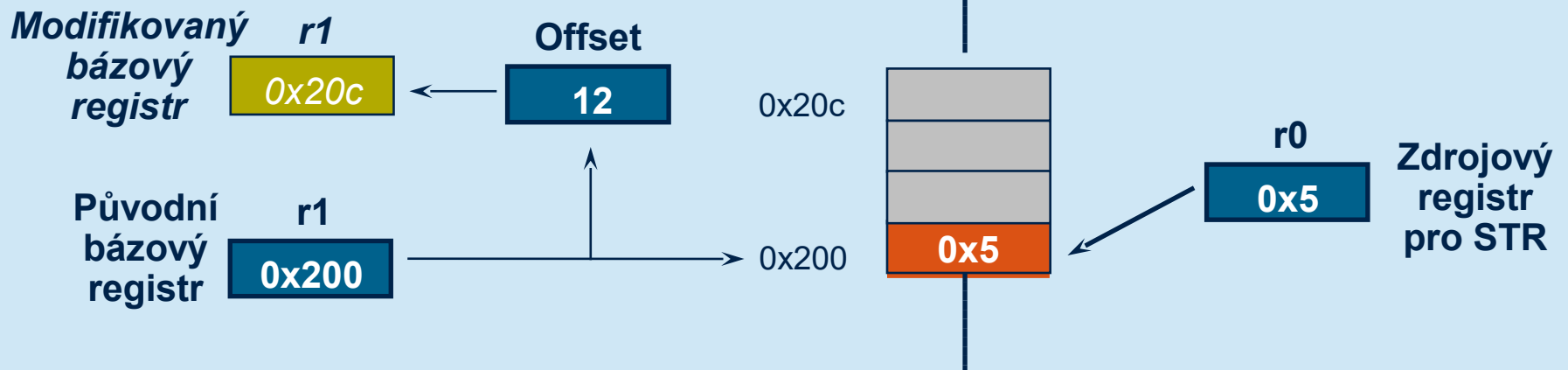
**Možnost pre nebo post modifikace registru**

# Pre or Post Indexed Addressing?

**Pre-indexed: STR r0, [r1, #12]**



■ **Post-indexed: STR r0, [r1], #12**





# LDM / STM operace

## Syntaxe:

**<LDM | STM>**{<cond>}<addressing\_mode> Rb{!}, <register list>

## 4 adresní módy:

**LDMIA / STMIA**

increment po

**LDMIB / STMIB**

increment před

**LDMDA / STMDA**

decrement po

**LDMDB / STMDB**

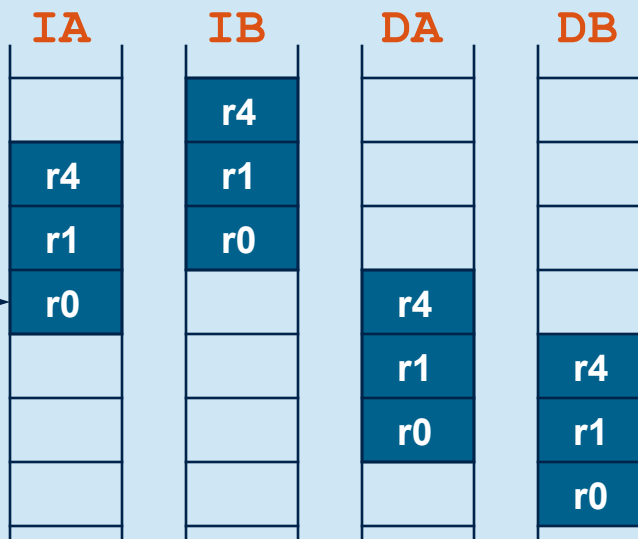
decrement před

**LDMxx** r10, {r0,r1,r4}

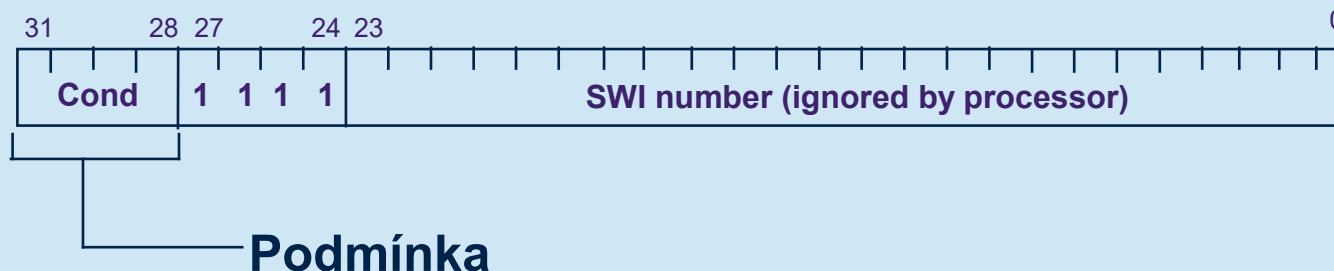
**STMxx** r10, {r0,r1,r4}

Bázový Registr (Rb)

r10



# Software Interrupt (SWI)



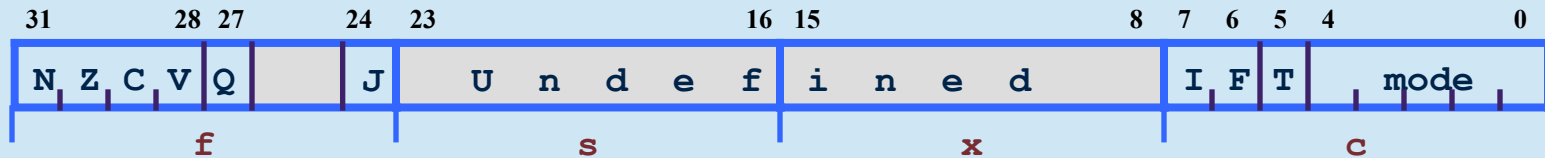
Softwarové vyvolání výjimky (přerušení), jediná SW možnost přechodu do privilegovaného módu

Lze specifikovat 24-bitovou hodnotu, která může být použita obslužnou rutinou

**Syntax:**

■ `SWI{<cond>} <SWI number>`

# Instrukce pro přesun PSR



MRS a MSR přesouvá obsah CPSR / SPSR do nebo z obecného registru

Syntaxe:

- **MRS**{<cond>} Rd,<psr> ; Rd = <psr>
- **MSR**{<cond>} <psr[\_fields]>,Rm ; <psr[\_fields]> = Rm

kde

- <psr> = CPSR or SPSR
- [\_fields] = libovolná kombinace 'fsxc'

Možnost použití přímé konstanty

- **MSR**{<cond>} <psr\_fields>,#Immediate

V režimu User mohou být všechny bity CPSR čteny ale zápis je možný pouze u podmínkových bitů (f)

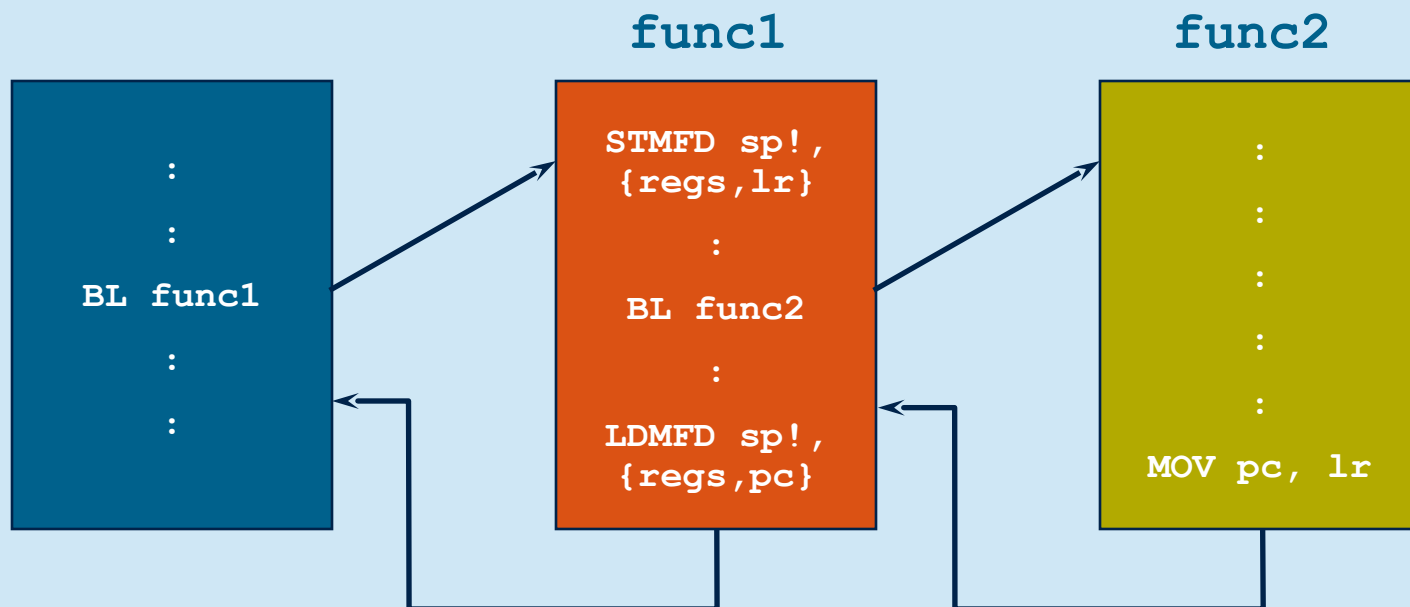
# ARM skoky a podprogramy

## B <label>

- Relativně k PC, rozsah  $\pm 32$  MB

## BL <subroutine>

- Uloží návratovou adresu do LR
- Návrat se provede obnovením PC z LR
- LR musí být většinou ukládán na zásobník programem

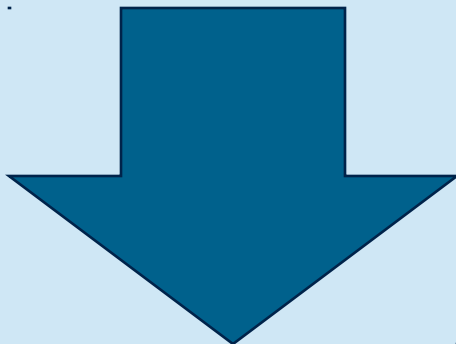


## Thumb je 16-ti bitová instrukční sada

- Optimalizována pro výstup překladače C (~65% velikosti ARM kódu)
- Zlepšuje využitelnost menší kapacity paměti
- Instrukce jsou interně převáděny na instrukce ARM a takto vykonávány
- Přepínání mezi ARM a Thumb módem se provede instrukcí **BX**

**ADDS r2,r2,#1**

32-bit ARM Instruction



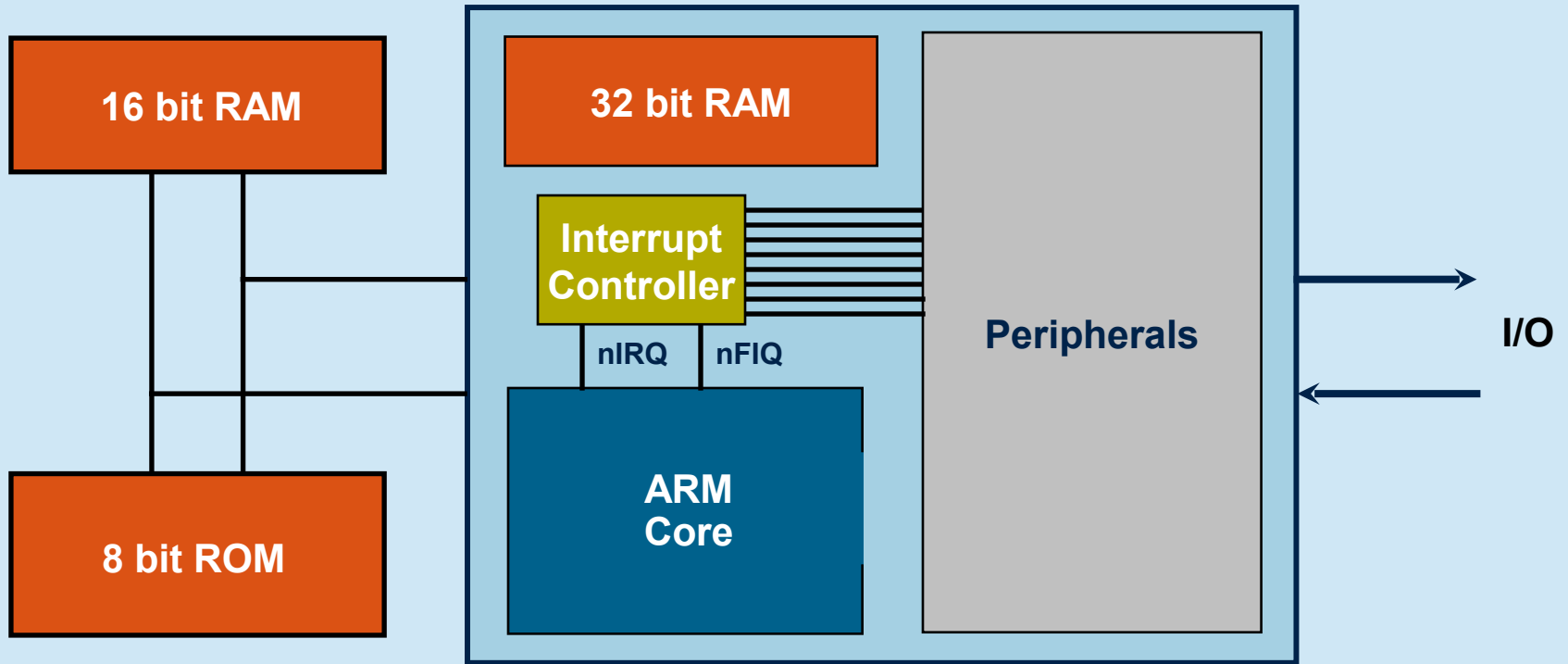
**ADD r2,#1**

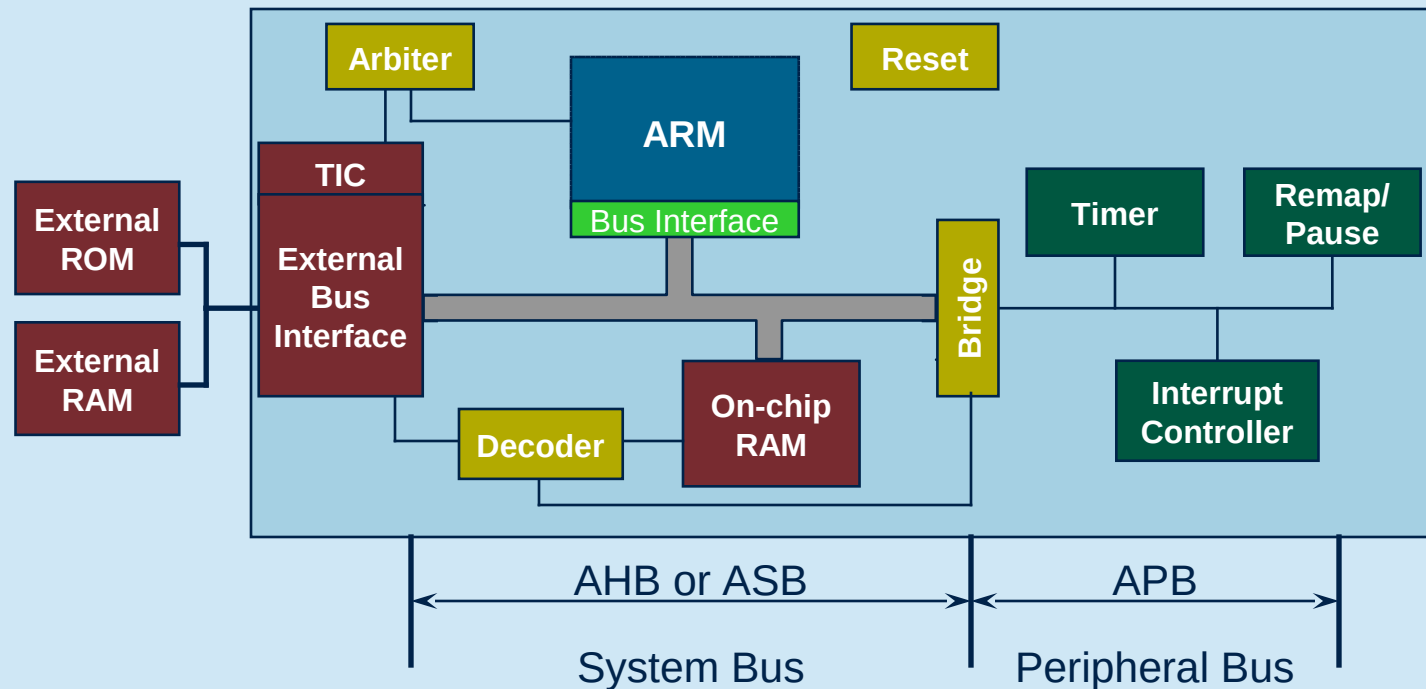
16-bit Thumb Instruction

## Většina instrukcí generovaných překladačem:

- Nepoužívá podmíněné instrukce
- Stejný cílový registr s operandem
- Používá pouze spodní množinu registrů
- Konstanty mají omezenou velikost
- Inline barrel shifter není použit

# Struktura procesoru (MCU)





## AMBA

- Advanced Microcontroller Bus Architecture

## ADK

- Complete AMBA Design Kit

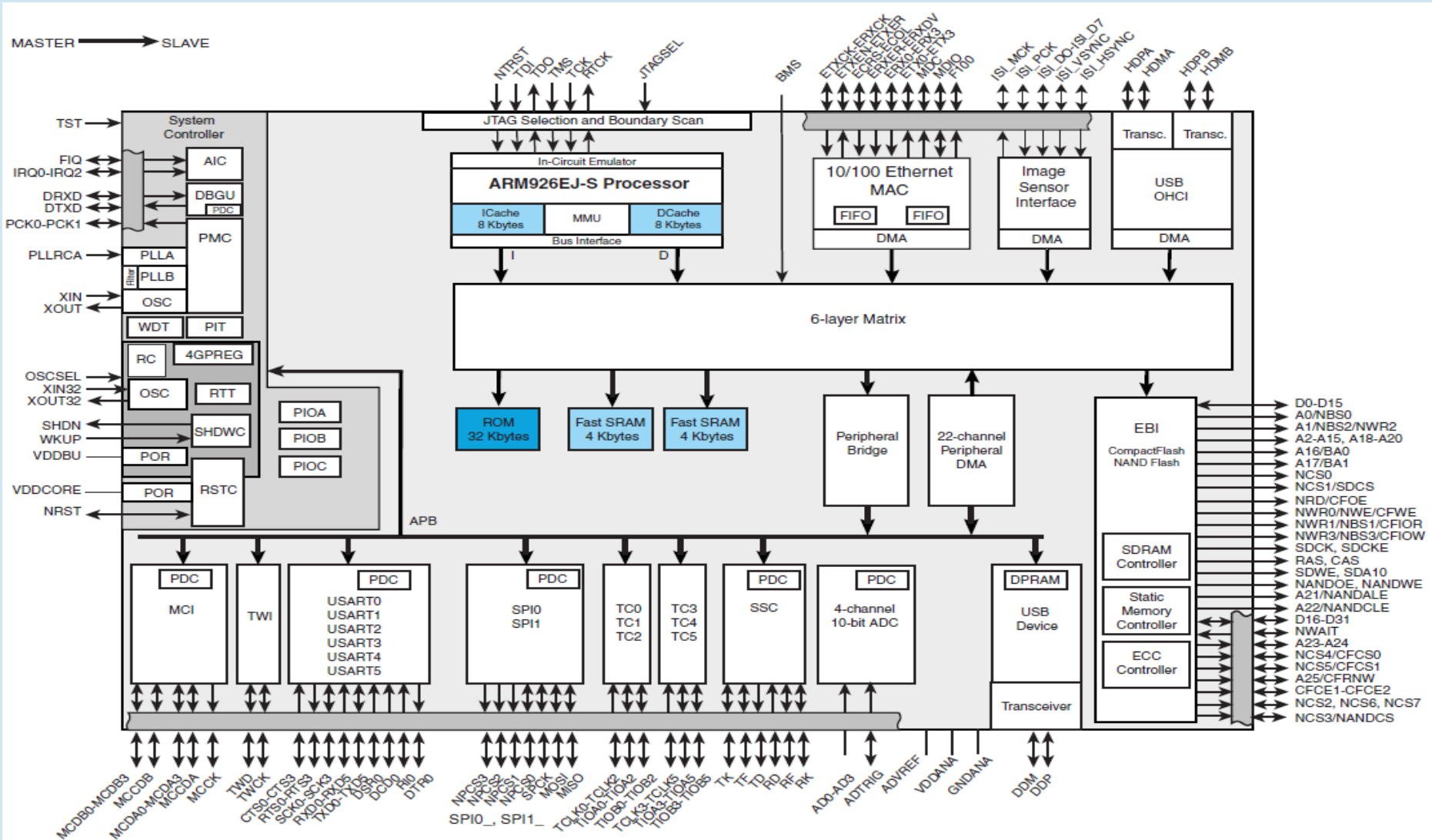
## ACT

- AMBA Compliance Testbench

## PrimeCell

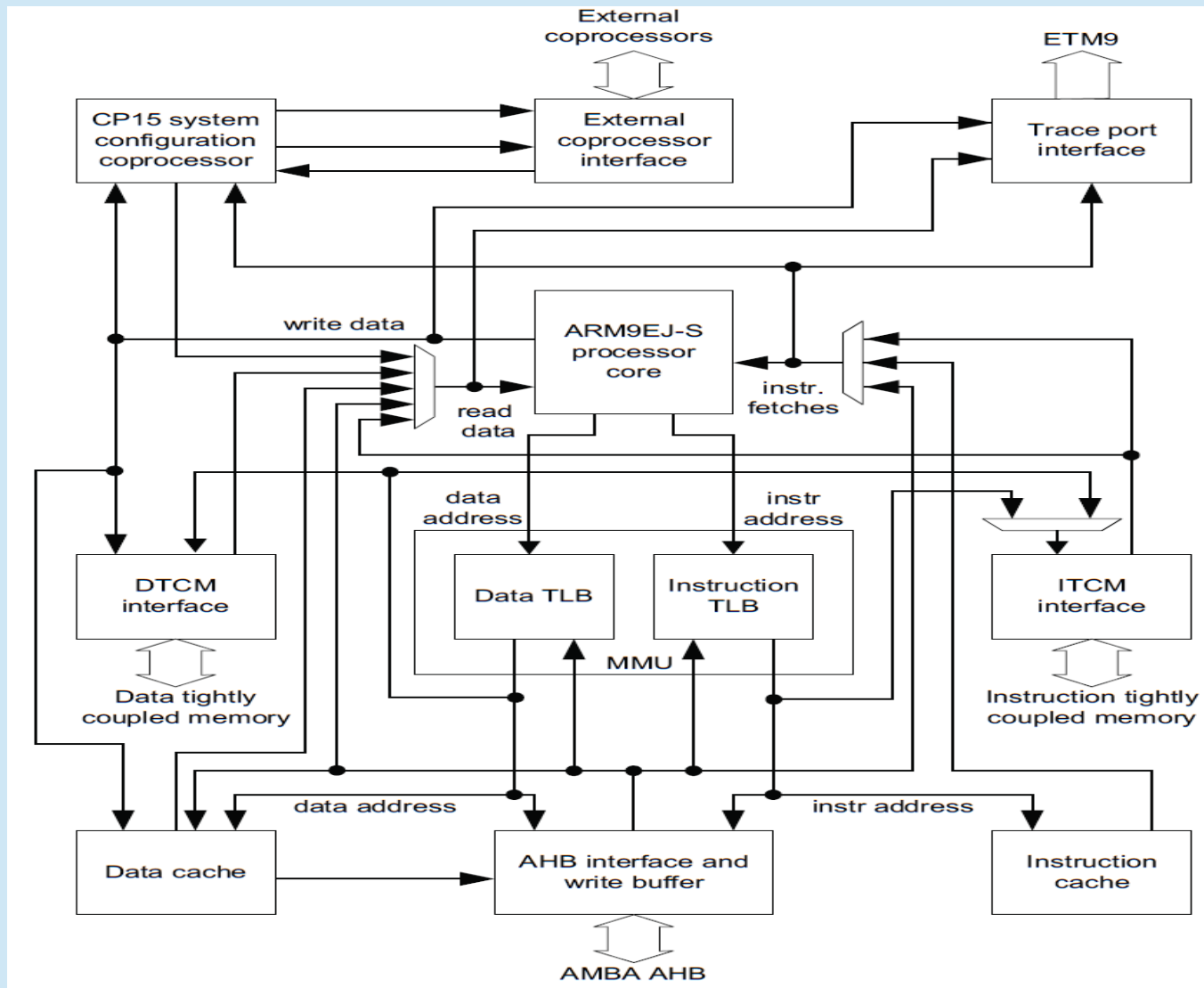
- ARM's AMBA compliant peripherals

# Blokové schéma AT91SAM9260





# Blokové schéma jádra ARM926



# Vlastnosti AT91SAM9260

**Incorporates the ARM926EJ-S™ ARM® Thumb® Processor**

- **DSP Instruction Extensions, ARM Jazelle® Technology for Java® Acceleration**
- **8-KByte Data Cache, 8-KByte Instruction Cache, Write Buffer**
- **200 MIPS at 180 MHz**
- **Memory Management Unit**
- **EmbeddedICE™, Debug Communication Channel Support**

**Additional Embedded Memories**

- **One 32 KByte Internal ROM, Single-cycle Access At Maximum Matrix Speed**
- **Two 4 KByte Internal SRAM, Single-cycle Access At Maximum Matrix Speed**

**External Bus Interface (EBI)**

- **Supports SDRAM, Static Memory, ECC-enabled NAND Flash and CompactFlash®**

**USB 2.0 Full Speed (12 Mbits per second) Device Port**

**USB 2.0 Full Speed (12 Mbits per second) Host Single Port (1 or 2)**

# Vlastnosti AT91SAM9260

## Ethernet MAC 10/100 Base T

- Media Independent Interface or Reduced Media Independent Interface
- 28-byte FIFOs and Dedicated DMA Channels for Receive and Transmit

## Image Sensor Interface

## Bus Matrix

- Six 32-bit-layer Matrix
- Boot Mode Select Option, Remap Command

## Fully-featured System Controller, including

- Reset Controller, Shutdown Controller
- Four 32-bit Battery Backup Registers for a Total of 16 Bytes
- Clock Generator and Power Management Controller
- Advanced Interrupt Controller and Debug Unit
- Periodic Interval Timer, Watchdog Timer and Real-time Timer