

# Organização Básica de computadores e linguagem de montagem

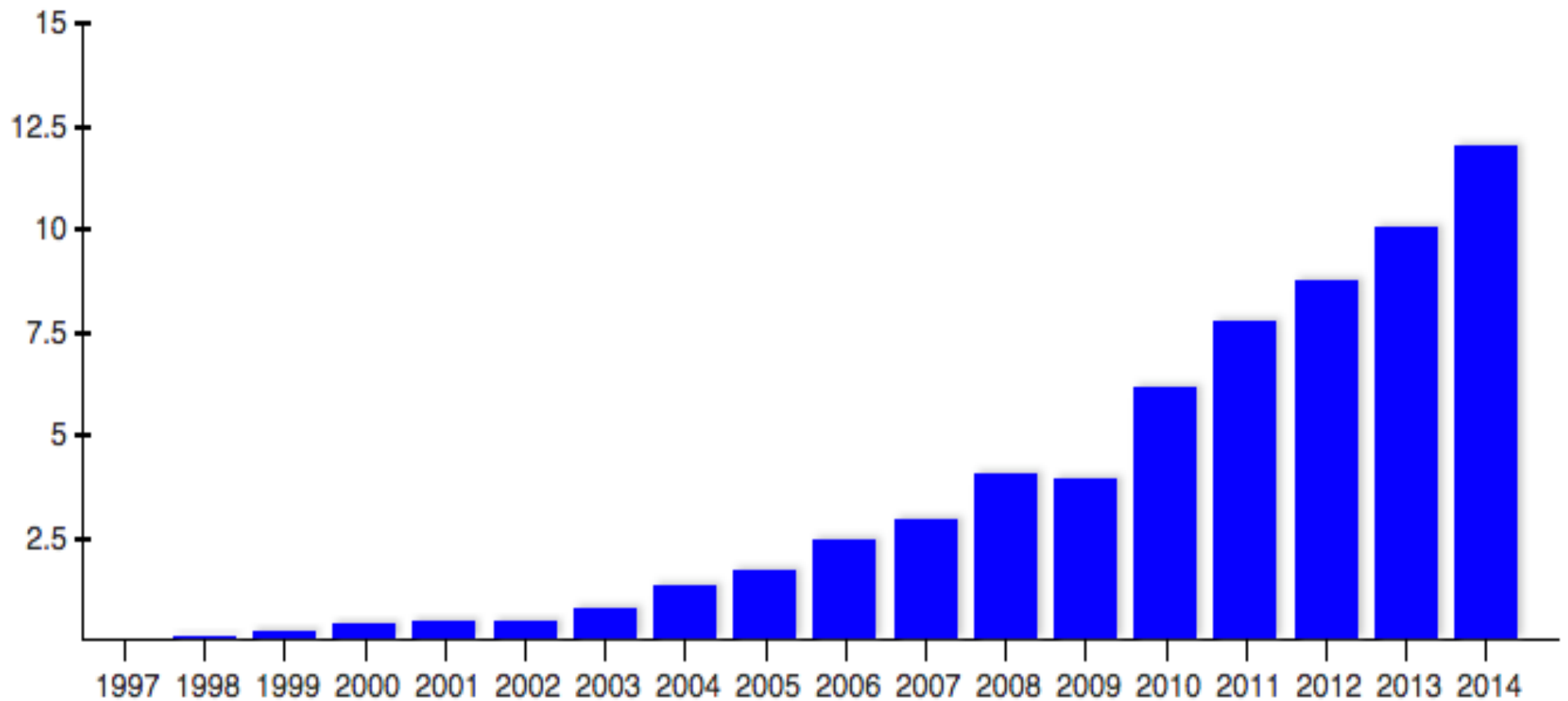
Prof. Edson Borin

2º Semestre de 2015

**In 2012, 8.7 billion ARM- based chips are reported as sold, and ARM processors surpassed 95% market share of mobile phones and tablets.**

**- ARM Ltd.:ARM Annual Report & Accounts 2012**

# Venda de chips contendo processadores ARM





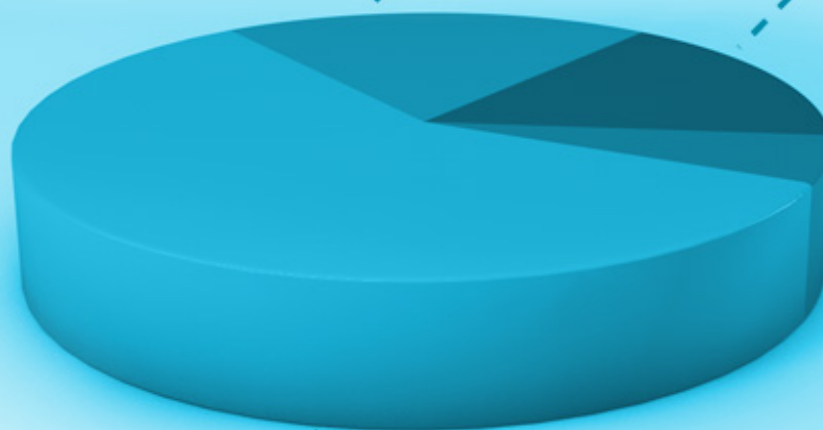
## 20% | Embedded

Applications including automotive, touch-screen controllers, industrial equipment, connectivity and smartcards



## 16% | Enterprise

Applications such as hard disk drives, and wireless/wireline networking infrastructure equipment



## 58% | Mobile

Devices including smartphones, mobile phones, tablets, e-readers and wearables



## 6% | Home

Consumer devices such as smart TVs, game consoles and home networking gateways

# Introdução à arquitetura ARM

# ARM

- Originalmente:
  - *Acorn RISC Machines*
  - Concebido originalmente para PCs
- Atualmente
  - *ARM: Advanced RISC Machines*
  - Largamente utilizado em sistemas embarcados e dispositivos móveis
  - Arquitetura simples com implementação pequena => baixo consumo de energia.

# ARM

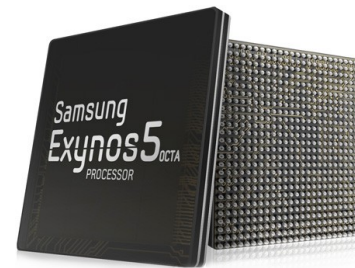
- É uma família de processadores RISC
- Várias **arquiteturas**: ARMv1, ARMv2, ... ARMv6, ARMv7-A, ARMv7-R, ARMv7-M.
- Com múltiplas **extensões**:
  - T: *Thumb*
  - E: *Enhanced DSP Instructions*
  - M: *Long Multiply* ( $32 \times 32 \Rightarrow 64$ )
  - J: *Jazelle*
- Ex: ARMv5TEJ = ARMv5 + Thumb + Enhanced Arithmetic Support (DSPs) + Jazelle

# ARM

<b>Arquitetura</b>	<b>Implementações</b>
<b>ARMv7-A</b>	<b>Cortex-A9, Cortex-A8</b>
<b>ARMv7-R</b>	<b>Cortex-R4</b>
<b>ARMv7-M</b>	<b>Cortex-M3, Cortex-M1</b>
<b>ARMv6</b>	<b>ARM1136JF-S, ARM1176JZF-S, ARM11 MPCore</b>
<b>ARMv5TE</b>	<b>ARM926EJ-S, ARM946E-S, ARM966E-S</b>
<b>ARMv4T</b>	<b>ARM7TDMI, ARM922T</b>



# ARM



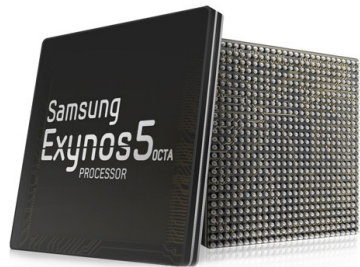
vs



vs



# ARM

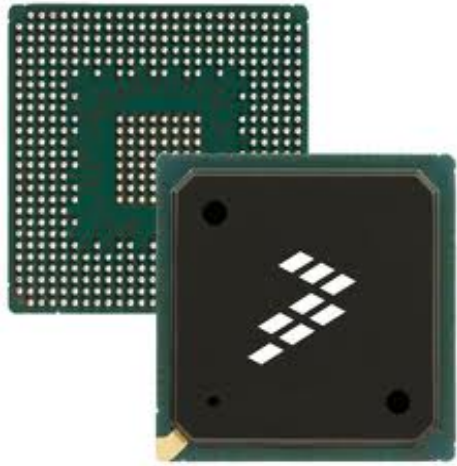


- Exynos 5 Octa
  - 4 cores ARM Cortex-A15 (~2 GHz)
  - 4 cores ARM Cortex-A7 (~1.4 GHz)
  - BIG.little
  - 2º trimestre de 2014
- Produzido pela Samsung

Galaxy S5



# ARM

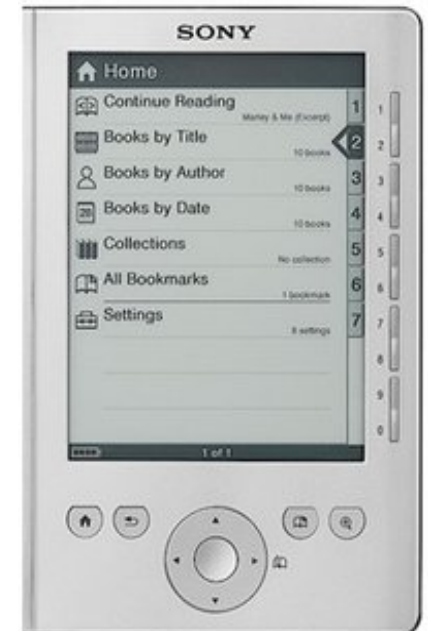


- Processadores Freescale da Família i.MX
  - SoC - *System-on-a-Chip: CPU, Video processing, graphics processing, etc...*



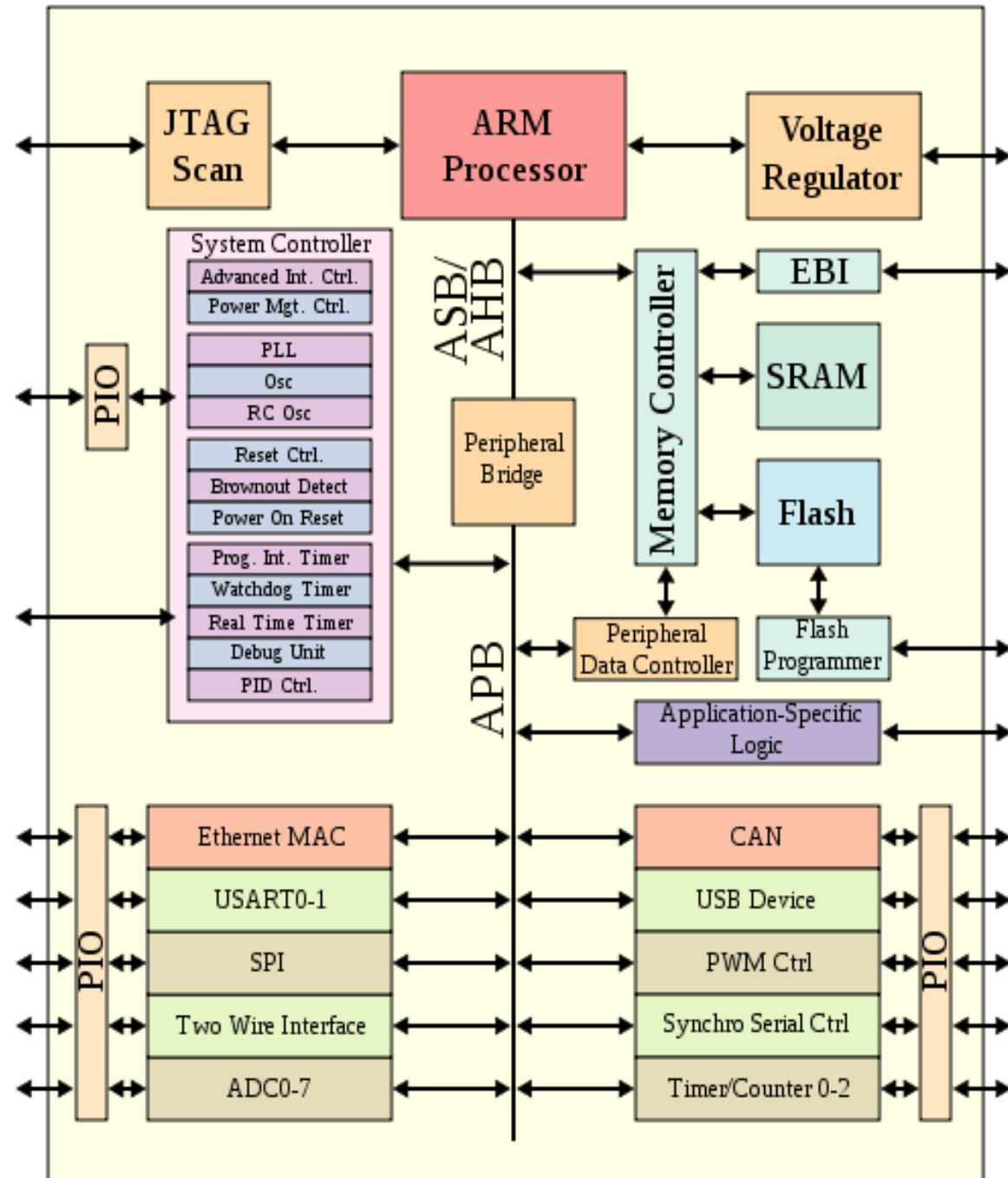
- Ford Sync
- Logitech Harmony *remote controls*
- Amazon Kindle
- Sony Reader

2010: 75% do mercado  
de eReaders



# ARM - SoC

- SoC (*System-on-a-Chip*): múltiplos componentes em um único *chip*.
- CPU, processador de Video, processador gráfico, etc...



# Arquitetura do ARM

# Arquitetura do ARM

- Diversos conjuntos de instruções:
  - Padrão: Instruções de 32 *bits*
  - Thumb: instruções compactas, de 16 *bits*
  - Jazelle DBX: instruções para executar *bytecode* JAVA.
  - NEON: instruções SIMD
  - VFP: Instruções para operações vetoriais em ponto flutuante.

# Arquitetura do ARM

- Modo padrão: Instruções possuem 32 *bits*.
- Todas instruções devem estar alinhadas em 4 *bytes* (1 palavra).

**.align 4**

- PC (*Program Counter*) aponta apenas para endereços múltiplos de 4.

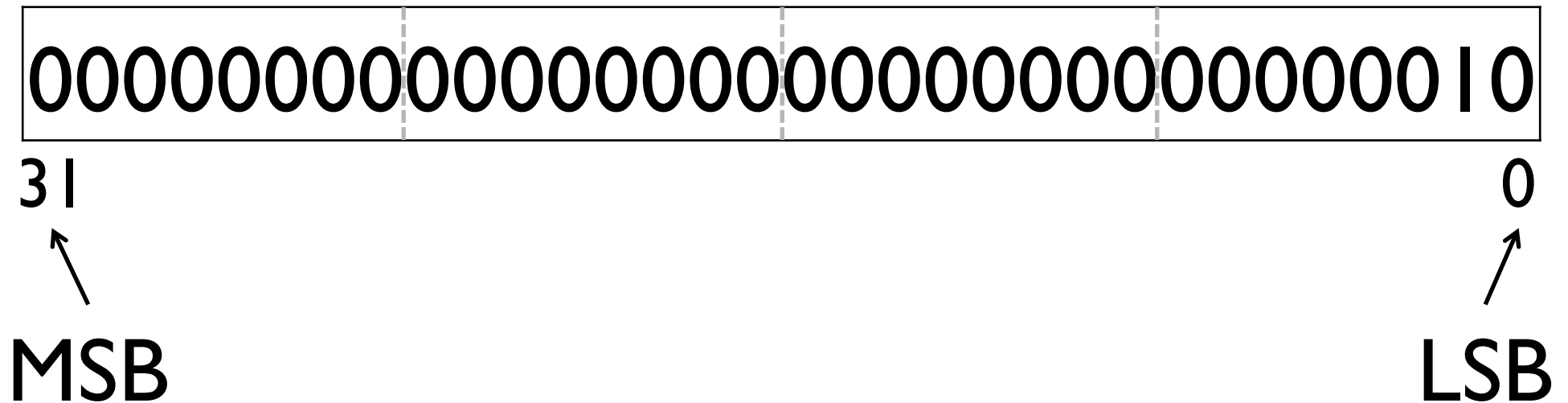
# Arquitetura do ARM

- Memória endereçada a *bytes*
- 3 tipos de dados
  - *byte*: 1 *byte*
  - *halfword*: 2 *bytes*
  - *word*: 4 *bytes*
- Dados armazenados na memória devem estar alinhados de acordo com o tamanho do dado



# Arquitetura do ARM

- Registradores (32 *bits*):



# Arquitetura do ARM

- Registradores (32 *bits*):

r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	r13	r14	r15
----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----

- Apelidos (*Aliases*)

a1	a2	a3	a4	v1	v2	v3	v4	v5	v6	sb	sl	fp	sp	lr	pc
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

- R13/SP: *stack pointer*
- R14/LR: *link register*
- R15/PC: *program counter*

# Arquitetura do ARM

- **Arquitetura Load/Store:** Os valores têm que ser carregados nos registradores antes de realizar operações.
- Não há instruções que operam diretamente em valores na memória!

```
LDR R0, [R1] ; R0 <= Mem[R1]
```

```
ADD R5, R0, R0 ; R5 <= R0+R0
```

```
STR R5, [R3] ; Mem[R3] <= R5
```

# Exemplo

LDR R0, [R1]

ADD R5, R0, R0

STR R5, [R3]

R1=0

R3=4

	Valor
0	06
1	00
2	00
3	00
4	00
5	00
6	00
7	00
...	...

Memória

# Instruções do ARM

- Referências
  - ARM Assembly Language – Fundamentals and Techniques. William Hohl. CRC Press
  - ARM Architecture Reference Manual. ARM DDI 0100E.

# Instruções do ARM

- Operações Lógicas
  - AND R1, R2, R3
  - ORR R1, R2, R3
  - EOR R1, R2, R3
  - BIC R1, R2, R3

<MNE> Rd, Rn, <Operand2>

Exemplo de <Operand2>: constante de 8 *bits* ou registrador

# Instruções do ARM

- Operações Lógicas
  - AND R1, R2, R3
  - ORR R1, R2, R3
  - EOR R1, R2, R3
  - BIC R1, R2, R3

<MNE> Rd, Rn, <Op2>

Exemplo de <Op2>: constante de 8 *bits* ou registrador

# Instruções do ARM

- Operações Aritméticas
  - `ADD R1, R2, R3`
  - `SUB R1, R2, R3`
  - `RSB R1, R2, R3`



# Instruções do ARM

- Operações Aritméticas com Carry
  - `ADC R1, R2, R3 // R1=R2+R3+Carry`
  - `SBC R1, R2, R3`
  - `RSC R1, R2, R3`
- O que é *carry*?
  - *Carry in* = “vem um”
  - *Carry out* = “vai um”
  - A *Carry* é uma *flag* (um único *bit*) que representa tanto o *carry in* como o *carry out*.

# Instruções do ARM

- Exemplo: Operações de 64 *bits*. Pares R1:R0 e R3:R2. Resultado em R5:R4

**ADDS** R4, R0, R2

**ADC** R5, R1, R3

- Análogo para SBC e RSC:

**SUBS** R4, R0, R2

**SBC** R5, R1, R3

<MNE>{S} Rd, Rn, <Op2>

# Instruções do ARM

- Formato das instruções lógicas e aritméticas

`<MNE> Rd, Rn, <Op2>`

Rd e Rn são registradores

`<Op2>` pode ser:

- Rm
- #imediato
- Rm, {LSL|LSR|ASR|ROR} {#imediato|Rs}
- Rm, RRX

# Instruções do ARM

- Op2 = Rm
- Exemplos:

AND R1, R2, **R3**

ADD R2, R2, **R0**

SUB R5, R3, **R5**

# Instruções do ARM

- Op2 = #imediato.
- Apenas números que podem ser formado a partir da rotação de constantes de 8 bits. A rotação deve ser um número par. Exemplos:

AND R1, R2, #16

SUB R2, R2, #0xFF00

ORR R5, R3, #0b00101100

- Imediatos inválidos: # 0x101, 0x102 e #0xFF04

# Instruções do ARM

- Op2 = Rm, {LSL|LSR|ASR|ROR} {#imediato|Rs}
- Registrador e parâmetro de deslocamento de *bits*.

Exemplos:

AND R1, R2, R3, LSL #16

ADD R2, R2, R2, ROR #2

ORR R5, R3, R0, LSR R2

EOR R5, R3, R0, ASR R1

BIC R5, R3, R0, LSR R2

# Instruções do ARM

- Op2 = Rm, RRX

- Exemplos:

AND R1, R2, **R3, RRX**

AND R2, R2, **R2, RRX**

# Organização Básica de computadores e linguagem de montagem

Prof. Edson Borin

2º Semestre de 2015



# Instruções do ARM

- Operações de movimentação de dados
  - MOV R1, R2
  - MVN R3, R4

<operation> Rd, <Operand2>

- Podem ser utilizadas para realizar deslocamento de *bits*:
  - MOV R1, R2, LSL #2
  - MVN R1, R2, LSR R3

# Instruções do ARM

- *Overflow* em operações aritméticas

- `ADD R1, R2, R3`

`R2 = 0x10000000`

`R3 = 0xF0000000`

`R1 =`

Ocorre *overflow* nesta operação?

# Instruções do ARM

- *Overflow* em operações aritméticas

- `ADD R1, R2, R3`

`R2 = 0x10000000`

`R3 = 0xF0000000`

`R1 = 0x00000000`

- Ocorreu *overflow* na representação sem sinal, mas não na representação com sinal.

# Instruções do ARM

- Outro exemplo
  - `ADD R1, R2, R3`

`R2 = 0x16000000`

`R3 = 0x70000000`

`R1 =`

Ocorre *overflow* nesta operação?

# Instruções do ARM

- Outro exemplo
  - `ADD R1, R2, R3`

`R2 = 0x16000000`

`R3 = 0x70000000`

`R1 = 0x86000000`

Ocorreu *overflow* na representação com sinal,  
mas não ocorreu na representação sem sinal.

# Instruções do ARM

- Como sabemos se o resultado de uma operação aritmética resultou em *overflow*?

# Instruções do ARM

- Como sabemos se o resultado de uma operação aritmética resultou em *overflow*?
- O processador armazena esta informação em duas *flags*: C e V
- A *flag* C (*Carry*) indica se a operação gerou um *overflow* na representação sem sinal.
- A *flag* V indica se a operação gerou *overflow* na representação com sinal.

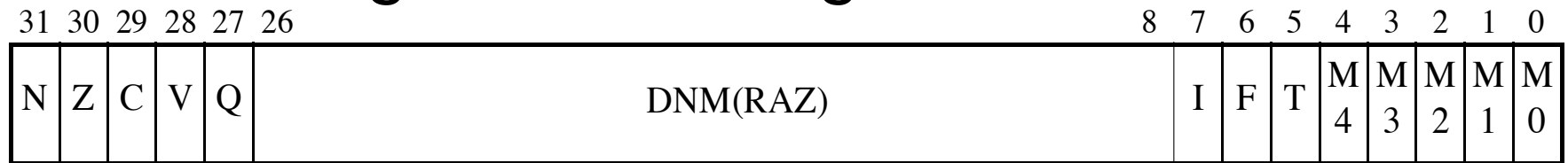
# Instruções do ARM

- Como sabemos se o resultado de uma operação aritmética resultou em *overflow*?
- O processador armazena esta informação em duas *flags*: C e V
  - A *flag* C (*Carry*) indica se a operação gerou um *overflow* na representação sem sinal.
  - A *flag* V indica se a operação gerou *overflow* na representação com sinal.
- Como fica o valor de C e V após as operações anteriores?



# Instruções do ARM

- ○ registrador CPSR
  - *Current Program Status Register*



- Armazena informações sobre a execução do programa. *As flags*, por exemplo.

*Flag N:* 1 se o resultado da operação é negativo

*Flag Z:* 1 se o resultado da operação é zero

*Flag C: I se houve carry-in ou carry-out*

*Flag V: 1 se houve overflow na representação com sinal*

# Instruções do ARM

- Operações aritméticas e lógicas só modificam as *flags* do registrador CPSR se a instrução solicitar.  
Exemplos:
- **ADDS** R1, R2, R3 modifica as flags, mas
- **ADD** R1, R2, R3 não modifica as flags

Formato:

<MNE> {S} Rd, Rn, <Op2>

# Instruções do ARM

- Comparação de números.
- Como fazemos para identificar se um número é maior ou igual ao outro?

# Instruções do ARM

- Comparação de números.
- Como fazemos para identificar se um número é maior ou igual ao outro?
- Por exemplo: Como fazemos para saber se o valor em R2 é maior ou igual ao valor em R1?

# Instruções do ARM

- Comparação de números.
- Como fazemos para identificar se um número é maior ou igual ao outro?
- Por exemplo: Como fazemos para saber se o valor em R2 é maior ou igual ao valor em R1?
- Podemos fazer uma subtração:
- `SUBS R0, R2, R1 // tmp = R2 - R1`
  - O que acontece se R2 for maior ou igual a R1?

# Instruções do ARM

- SUBS R0, R2, R1 //  $\text{tmp} = R2 - R1$
- Se não houve *overflow* ( $V==0$ ):
  - $N==0$  se  $R2 \geq R1$  (Resultado positivo)
  - $N==1$  se  $R2 < R1$  (Resultado negativo)
- E se houver *overflow* ( $V==1$ )?

# Instruções do ARM

- SUBS R0, R2, R1 //  $\text{tmp} = R2 - R1$
- Se não houve *overflow* ( $V==0$ ):
  - $N==0$  se  $R2 \geq R1$  (Resultado positivo)
  - $N==1$  se  $R2 < R1$  (Resultado negativo)
- E se houver *overflow* ( $V==1$ )?
  - $N==0$  se  $R2 < R1$
  - $N==1$  se  $R2 \geq R1$

# Instruções do ARM

- SUBS R0, R2, R1 // tmp = R2 – R1
  - R2 >= R1 se N == V
  - R2 < R1 se N != V
- E para números sem sinal? Como fazemos?



# Instruções do ARM

- SUBS R0, R2, R1 //  $\text{tmp} = R2 - R1$ 
  - $R2 \geq R1$  se  $N == V$
  - $R2 < R1$  se  $N != V$
- E para números sem sinal? Como fazemos?
  - Utilize a *flag* C.

# Instruções do ARM

- SUBS R0, R2, R1 // tmp = R2 – R1
- Como fazemos para detectar se R2 == R1?
  - Z == 0

# Instruções do ARM

- Instruções de Comparação
  - `CMP R1, R2 // R2-R1`
  - `CMN R1, R2 // R2-(-R1)`
  - `TST R1, R2 // R2 AND R1`
  - `TEQ R1, R2 // R2 EOR R1`

Formato:

`<MNE> Rd, <Op2>`

Nota: Não precisa do {S}

# Instruções do ARM

- Já sabemos comparar números. Como utilizamos esta informação?

# Instruções do ARM

- Já sabemos comparar números. Como utilizamos esta informação?
- Podemos desviar o fluxo de controle

```
CMP R1, R2
```

```
BEQ rotulo1 // Salta se R1==R2
```

```
BNE rotulo2 // Salta se R1!=R2
```

```
BLE rotulo3 // Salta se R1<=R2
```

```
...
```

# Instruções do ARM

- Formato: B{cond} <endereco>
- cond:

Sufixo	Condição	Flags
EQ	Igual	$Z = 1$
NE	Diferente	$Z = 0$
CS/HS	Carry setada/maior ou igual (sem sinal)	$C = 1$
CC/LO	Carry limpa/menor (sem sinal)	$C = 0$
MI	negativo	$N = 1$
PL	positivo ou zero	$N = 0$
VS	<i>overflow</i>	$V = 1$
VC	<i>no overflow</i>	$V = 0$
HI	maior (sem sinal)	$(C = 1) \text{ e } (Z = 0)$
LS	menor ou igual (sem sinal)	$(C = 0) \text{ ou } (Z = 1)$
GE	maior ou igual (com sinal)	$N = V$
LT	menor (com sinal)	$N \neq V$
GT	maior (com sinal)	$(Z = 0) \text{ e } (N = V)$
LE	menor ou igual (com sinal)	$(Z = 1) \text{ ou } (N \neq V)$
AL	sempre	—

# Instruções do ARM

## Instruções de salto

- Salto direto

B{cond} <endereço>

- Salto direto com *link* –  $LR := PC + 4$

BL{cond} <endereço>

- Salto indireto

BX{cond} Rn

- Salto indireto com *link* –  $LR := PC + 4$

BXL{cond} Rn

# Instruções do ARM

- Os sufixos condicionais podem ser utilizados em quase todas as instruções do ARM. Exemplo:

`CMP R1, R2`

`ADDGE R3, R3, R1`

`ADDLT R3, R3, R2`



# Instruções do ARM

Formato das instruções:

- Lógicas e Aritméticas: ADD, AND...

`<MNE>{cond}{S} Rd, Rn, <Op2>`

- Transferência de dados: MOV, MVN

`<MNE>{cond}{S} Rd, <Op2>`

- Comparação: CMP, CMN, TST, TEQ

`<MNE>{cond} Rn, <Op2>`

- Salto: B, BL, BX, BXL

`<MNE>{cond} <endereco>`

# Controle do Fluxo de Execução

# Controle do Fluxo de Execução

- Sentença condicional “Se-Então” (If-Then)

Exemplo: C/C++

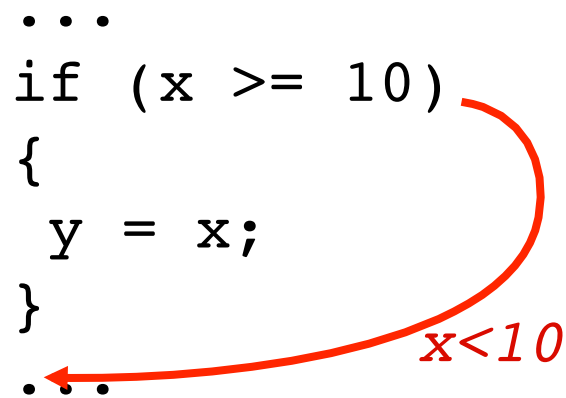
```
...  
if (x >= 10)  
{  
    y = x;  
}  
...
```

# Controle do Fluxo de Execução

- Sentença condicional “Se-Então” (If-Then)

Exemplo: C/C++

```
...  
if (x >= 10)  
{  
    y = x;  
}  
...
```



A red curved arrow originates from the condition `(x >= 10)` and points to the opening curly brace of the if block. Another red curved arrow originates from the closing curly brace of the if block and points back to the line following the if statement. A red label `x < 10` is positioned near the second arrow, indicating the path taken when the condition is false.

Exemplo: Ling. de Montagem


```
@ x está em r1  
@ y está em r2
```

# Controle do Fluxo de Execução

- Sentença condicional “Se-Então” (If-Then)

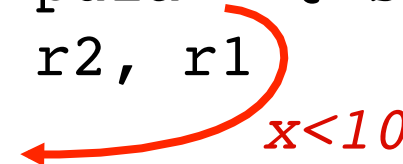
Exemplo: C/C++

```
...  
if (x >= 10)  
{  
    y = x;  
}  
...
```



Exemplo: Ling. de Montagem

```
@ x está em r1  
@ y está em r2  
...  
CMP r1, #10  
BLT pula @ salta se r1<10  
MOV r2, r1  
pula:  
...
```




- BLT salta para o alvo se o resultado da última comparação foi “menor que” supondo números com sinal. Para números sem sinal use “BLO” (Veja o manual do ARM)

# Controle do Fluxo de Execução

- Sentença condicional “Se-Então” (If-Then)

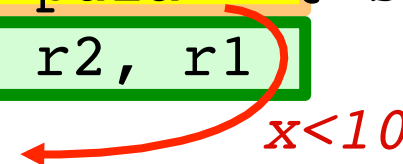
Exemplo: C/C++

```
...  
if (x >= 10)  
{  
    y = x;  
}  
...
```



Exemplo: Ling. de Montagem

```
@ x está em r1  
@ y está em r2  
...  
CMP r1, #10  
BLT pula @ salta se r1<10  
MOV r2, r1  
pula:  
...
```



- BLT salta para o alvo se o resultado da última comparação foi “menor que” supondo números com sinal. Para números sem sinal use “BLO” (Veja o manual do ARM)

# Controle do Fluxo de Execução

- Sentença condicional “Se-Então-Senão” (If-Then-Else)

Exemplo: C/C++

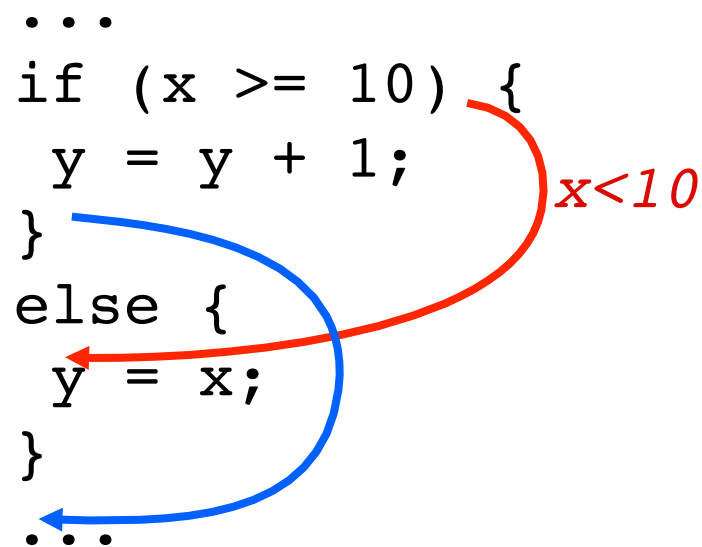
```
...  
if (x >= 10) {  
    y = y + 1;  
}  
else {  
    y = x;  
}  
...
```

# Controle do Fluxo de Execução

- Sentença condicional “Se-Então-Senão” (If-Then-Else)

Exemplo: C/C++

```
...  
if (x >= 10) {  
    y = y + 1;  
}  
else {  
    y = x;  
}  
...
```



Exemplo: Ling. de Montagem

@ x está em r1

@ y está em r2

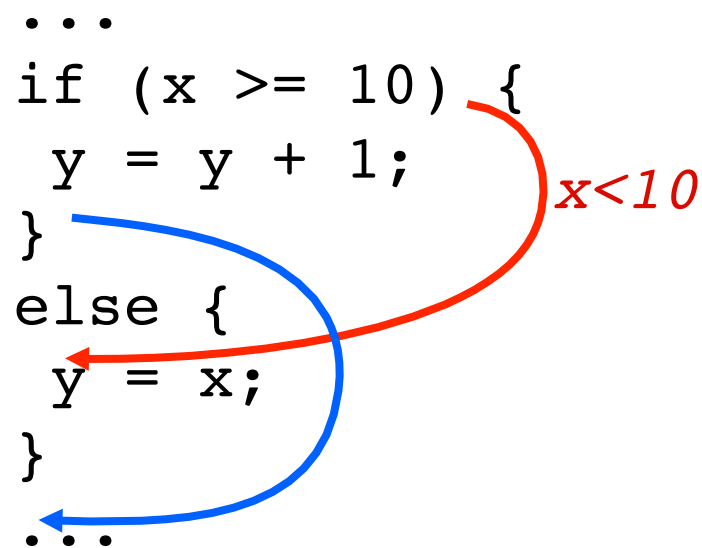


# Controle do Fluxo de Execução

- Sentença condicional “Se-Então-Senão” (If-Then-Else)

Exemplo: C/C++

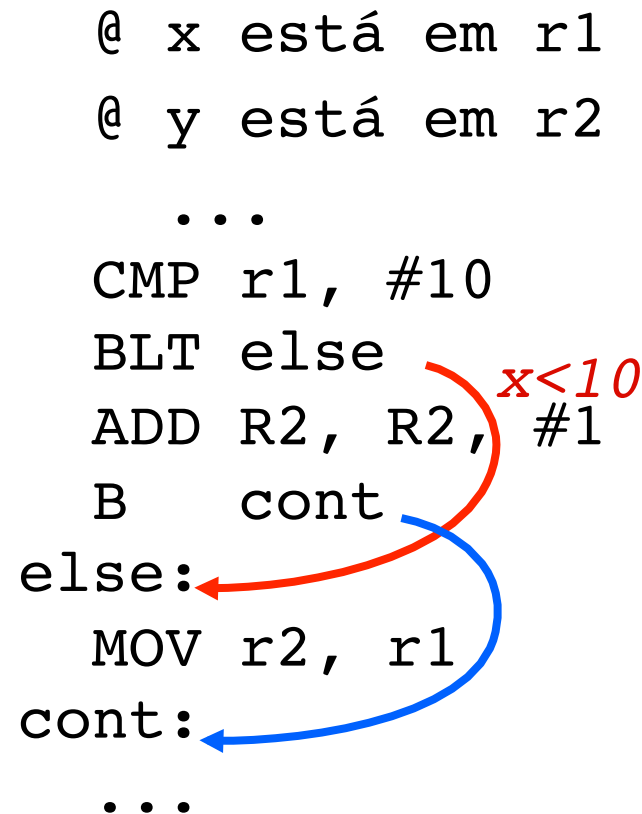
```
...  
if (x >= 10) {  
    y = y + 1;  
}  
else {  
    y = x;  
}  
...
```



The flowchart illustrates the execution of the C/C++ code. A red arrow labeled  $x < 10$  originates from the closing brace of the 'if' block and points to the 'else' block. A blue arrow originates from the closing brace of the 'else' block and points to the final '...' line, representing the flow after the conditional execution.

Exemplo: Ling. de Montagem

```
@ x está em r1  
@ y está em r2  
...  
CMP r1, #10  
BLT else  
ADD R2, R2, #1  
B cont  
else:  
    MOV r2, r1  
cont:  
...
```



The flowchart illustrates the execution of the assembly code. A red arrow labeled  $x < 10$  originates from the 'BLT else' instruction and points to the 'else:' label. A blue arrow originates from the 'B cont' instruction and points to the 'cont:' label. The flow continues from 'cont:' to the final '...' line.

# Controle do Fluxo de Execução

- Sentença condicional “Se-Então-Senão” (If-Then-Else)

Exemplo: C/C++

```
...  
if (x >= 10) {  
    y = y + 1;  
}  
else {  
    y = x;  
}  
...
```

The diagram illustrates the flow of an if-else statement in C/C++. The condition `x >= 10` is highlighted in a yellow box. A red arrow labeled `x < 10` points from the condition box to the `else` block, indicating the path taken when the condition is false. A blue arrow points from the `if` block to the `else` block, and another blue arrow points from the `else` block to the end of the code block, representing the flow of execution.

Exemplo: Ling. de Montagem

```
@ x está em r1  
@ y está em r2  
...  
CMP r1, #10  
BLT else  
ADD R2, R2, #1  
B cont  
else:  
MOV r2, r1  
cont:  
...
```

The diagram illustrates the flow of an if-else statement in assembly language. The assembly code is as follows:  
`CMP r1, #10`  
`BLT else`  
`ADD R2, R2, #1`  
`B cont`  
`else:`  
`MOV r2, r1`  
`cont:`  
The `CMP r1, #10` and `BLT else` instructions are highlighted in a yellow box. A red arrow labeled `x < 10` points from the `BLT else` instruction to the `else:` label. A blue arrow points from the `B cont` instruction to the `cont:` label, and another blue arrow points from the `else:` label to the `cont:` label, representing the flow of execution.

# Organização Básica de computadores e linguagem de montagem

Prof. Edson Borin

2º Semestre de 2015

# Controle do Fluxo de Execução

- Laço “enquanto” (while)

Exemplo: C/C++

```
...  
i=0;  
while (i < 20)  
{  
    y = y+3;  
    i = i+1;  
}  
...
```

# Controle do Fluxo de Execução

- Laço “enquanto” (while)

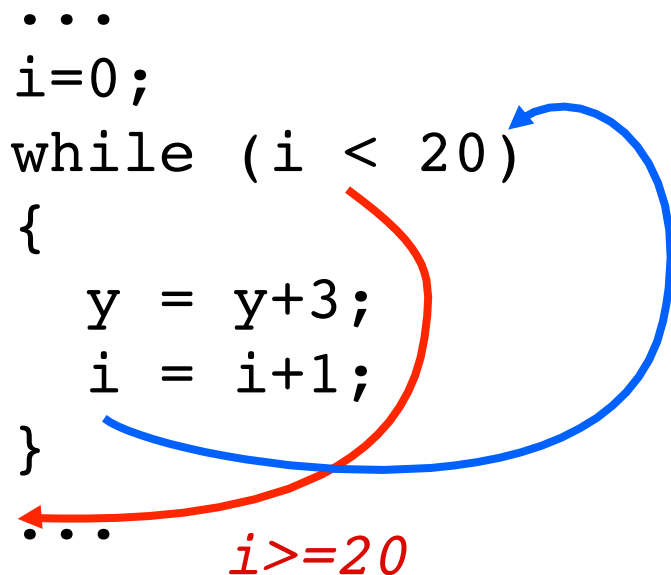
Exemplo: C/C++

Exemplo: Ling. de Montagem

@ i está em r1

@ y está em r2

```
...  
i=0;  
while (i < 20)  
{  
    y = y+3;  
    i = i+1;  
}  
...
```



The diagram illustrates the execution flow of a while loop. A blue arrow starts from the closing brace of the loop body and points back to the condition `(i < 20)`, representing the loop's continuation. A red arrow starts from the condition `(i < 20)` and points to the text `i >= 20` below the loop, representing the exit condition. The text `i >= 20` is written in red.

# Controle do Fluxo de Execução

- Laço “enquanto” (while)

Exemplo: C/C++

```
...  
i=0;  
while (i < 20)  
{  
    y = y+3;  
    i = i+1;  
}  
...
```

The diagram illustrates the execution flow of a while loop in C/C++. A blue arrow starts at the beginning of the loop body, goes around the right side, and points back to the start of the loop body, indicating the loop's continuation. A red arrow starts at the end of the loop body, goes around the right side, and points to the condition 'i < 20', indicating the loop's exit path. Below the loop, the text 'i >= 20' is written in red, representing the condition that causes the loop to terminate.

Exemplo: Ling. de Montagem

```
@ i está em r1  
@ y está em r2  
...  
MOV R1, #0      @ i=0  
enquanto:  
    CMP R1, #20  @ se i >= 20  
    BGE cont     @ sai do laco  
    ADD R2, R2, #3 @ y=y+3  
    ADD R1, R1, #1 @ i=i+1  
    B  enquanto  
cont:  
...
```

The diagram illustrates the execution flow of a while loop in assembly language. A blue arrow starts at the beginning of the loop body, goes around the right side, and points back to the start of the loop body, indicating the loop's continuation. A red arrow starts at the end of the loop body, goes around the right side, and points to the 'BGE cont' instruction, indicating the loop's exit path. Below the loop, the text 'r1 >= 20' is written in red, representing the condition that causes the loop to terminate.

# Controle do Fluxo de Execução

- Laço “enquanto” (while)

Exemplo: C/C++

```
...  
i=0;  
while (i < 20)  
{  
    y = y+3;  
    i = i+1;  
}  
...
```

*i >= 20*

Exemplo: Ling. de Montagem

@ i está em r1  
@ y está em r2

```
...  
MOV R1, #0          @ i=0  
enquanto:  
    CMP R1, #20      @ se i >= 20  
    BGE cont         @ sai do laço  
    ADD R2, R2, #3    @ y=y+3  
    ADD R1, R1, #1    @ i=i+1  
    B enquanto  
cont:  
...
```

*r1 >= 20*

# Controle do Fluxo de Execução

- Laço “para” (for)

Exemplo: C/C++

```
...  
for (i=0; i<10; i++)  
{  
    y = y+2;  
}  
...
```



# Controle do Fluxo de Execução

- Laço “para” (for)

Exemplo: C/C++

```
...  
for (i=0; i<10; i++)  
{  
    y = y+2;  
}  
...
```

Exemplo: Ling. de Montagem

```
@ i está em r1  
@ y está em r2  
...  
MOV R1, #0      @ i=0  
for:  
    CMP R1, #10  @ se i >= 10  
    BGE cont     @ sai do laço  
    ADD R2, R2, #2 @ y=y+2  
    ADD R1, R1, #1 @ i=i+1  
    B     for  
cont:  
...
```

# Controle do Fluxo de Execução

- Laço “para” (for)

Exemplo: C/C++

```
...  
for (i=0; i<10; i++)  
{  
    y = y+2;  
}  
...
```

Exemplo: Ling. de Montagem

@ i está em r1

@ y está em r2

```
...  
MOV R1, #0 @ i=0  
for:  
    CMP R1, #10 @ se i >= 10  
    BGE cont @ sai do laço  
    ADD R2, R2, #2 @ y=y+2  
    ADD R1, R1, #1 @ i=i+1  
B for  
cont:  
...
```

# Controle do Fluxo de Execução

- Laço “faça-enquanto” (do-while)

Exemplo: C/C++

```
...  
i=0;  
do  
{  
    y = y+2;  
    i = i+1;  
} while (i < 10);  
...
```

Exemplo: Ling. de Montagem

```
@ i está em r1  
@ y está em r2  
...
```

=> Exercício

# Controle do Fluxo de Execução

- Sentenças com múltiplas condições:

Exemplo: C/C++

Exemplo: Ling. de Montagem

@ x está em r1

@ y está em r2

...

```
if ( (x>=10) && (y<20) )
```

```
{
```

```
    x = y;
```

```
}
```

...

# Controle do Fluxo de Execução

- Sentenças com múltiplas condições:

Exemplo: C/C++

```
...  
if ((x>=10) && (y<20))  
{  
    x = y;  
}  
...
```

Exemplo: Ling. de Montagem

```
@ x está em r1  
@ y está em r2  
...  
CMP R1, #10  
BLT pula    @ Pula se x < 10  
CMP R2, #20  
BGE pula    @ Pula se y >= 20  
MOV r1, r2  @ x = y  
pula:  
...
```

# Exemplo

Escreva um trecho de programa que determina qual o maior valor de uma cadeia de números de 32 *bits*, sem sinal, cujo endereço inicial é dado em R2. Inicialmente, R3 contém o número de valores presentes na cadeia; suponha que  $R3 > 0$ . Ao final do trecho, R0 deve conter o valor máximo e R1 deve conter o endereço do valor máximo.

# Transferência de Dados

# Instruções do ARM

- Transferência de Dados de/para um registrador

`<MNE>{cond}{tam} Rd, <endereço>`

`tam = {B|SB|H|SH}`

- Exemplos:

`LDR R0, [R1]`

`LDRB R1, [R2]`

`STR R5, [R3]`

`STRH R5, [R9]`

- OBS: Palavras de 32/16 *bits* devem estar alinhadas em 4/2 *bytes*



# Instruções do ARM

- Modo de endereçamento **Pré-indexado**

<MNE>{cond}{tam} Rd, [base, deslocamento]

- Base é sempre um registrador: Ex: **R2**
- Deslocamento pode ser:

LDR RI, [**R2**] @ sem deslocamento

LDR RI, [**R2**, **#4**] @ Reg. {+|-} Imediato

LDR RI, [**R2**, **R5**] @ Reg. {+|-} Reg.

LDR RI, [**R2**, **-R5**, **LSL #4**] @ Reg. {+|-} Reg. modif.

- [Rn, {+|-}Rm, {LSL|LSR|ASR|ROR} {#imm}]

# Exemplo

Escreva uma função que verifica se uma cadeia de caracteres terminada em zero possui uma determinada letra.

r0: endereço inicial da cadeia

r1: letra a ser procurada

retorna o endereço da primeira posição da cadeia onde a letra ocorre se a letra for encontrada. Senão, retorna zero.

# Instruções do ARM

- Modo de endereçamento **Pré-indexado com Writeback**

<MNE>{cond}{tam} Rd, [base, deslocamento] !

- Após a execução da instrução, o registrador base é atualizado da seguinte forma:

base = base + deslocamento

- Exemplo:

LDR R1, [R2, #4] !

LDR R5, [R3, R4, LSL #2] !

# Instruções do ARM

- O que este código faz?

```
MOV R0, #0
```

```
loop:
```

```
    CMP R2, R3
```

```
    BHS exit
```

```
    LDR R1, [R2, #4]!
```

```
    ADD R0, R0, R1
```

```
    B loop
```

```
exit:
```

# Instruções do ARM

- Modo de endereçamento **Pós-indexado**

<MNE> {cond} {tam} Rd, [**base**], **deslocamento**

- **Base** é sempre um registrador: Ex: **RI**

- **Deslocamento** pode ser:

LDR **RI**, [**R2**], **#4** @ Reg. {+|-} Imediato

LDR **RI**, [**R2**], **R5** @ Reg. {+|-} Reg.

LDR **RI**, [**R2**], **-R5, LSL #4** @ Reg. {+|-} Reg. modif.

- $[R_n], \{+|- \} R_m, \{LSL | LSR | ASR | ROR \} \{ \#imm \}$

# Instruções do ARM

- Pseudo instrução:

`<MNE>{cond}{tam} Rd, <rotulo>`

- Exemplo:

```
ldr r0, x
```

```
...
```

```
x: .word 10
```

# Instruções do ARM

- Pseudo instrução:

`<MNE>{cond}{tam} Rd, <rotulo>`

- Exemplo:

```
ldr r0, x
```

...

```
x: .word 10
```

- O montador gera:

```
ldr r0, [PC, #168]
```

onde 168 é o deslocamento de PC (instrução atual)  
para o endereço do rótulo x.

# Instruções do ARM

- Pseudo instrução:

`LDR{cond}{tam} Rd, =constante`

- Move uma constante numérica para o registrador Rd
- O montador tenta utilizar a instrução MOV, mas se não for possível (a constante for muito grande), gera uma instrução LDR.

`ldr r0, =100`      gera MOV R0, #100

`ldr r0, =1000`      gera MOV R0, #1000

`ldr r0, =1001`      gera LDR R0, [PC, #des1]

...

end: 000003e9



# Instruções do ARM

- LDR carrega dados da memória para os registradores
- STR armazena dados dos registradores na memória
- B, H, SB, SH especificam o tamanho do dado
  - Ex: LDRSB carrega número de um *byte* com sinal em um registrador.
- Endereçamento pré-indexado computa o endereço efetivo antes de realizar a transferência.
- Endereçamento pré-indexado com writeback atualiza o valor do registrador base após a execução.
- Endereçamento pós-indexado usa o endereço no registrador base, e atualiza o base após a execução.

# Instruções do ARM

- Transferência de múltiplos dados de/para memória  
    <MNE>{cond}<modo> Rn{!}, <reglist>  
    <reglist>: lista de registradores entre chaves  
    <modo>: controla como o Rn será modificado
  - IA: incrementado depois
  - DA: decrementado depois
  - IB: incrementado antes
  - DB: decrementado antes

# Instruções do ARM

- Transferência de múltiplos dados de/para memória  
`<MNE>{cond}<modo> Rn{!}, <reglist>`

Exemplos:

- Salvar R1, R2 e R8 na memória a partir do endereço apontado por R0

```
STMIA R0!, {R1,R2,R8}
```

... @ podemos sujar r1, r2 e r8

```
LDMDB R0!, {R1,R2,R8}
```