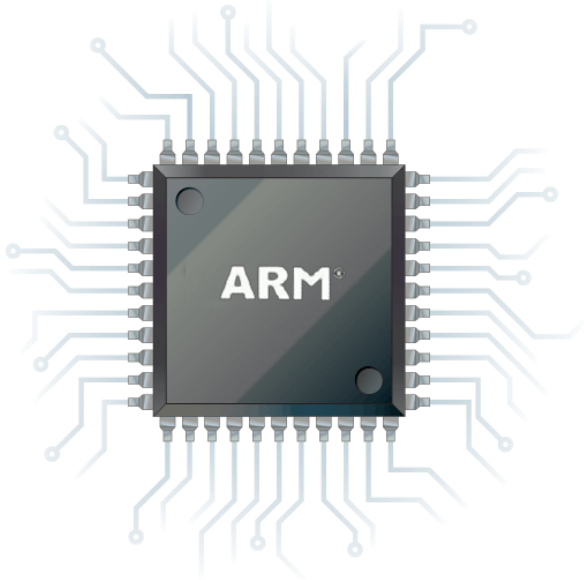
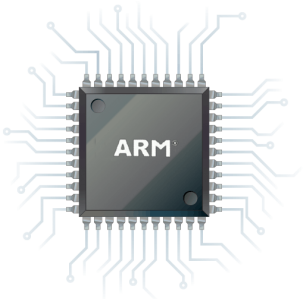


ARQUITETURA ARM

Linguagem Assembly



Assembly

Diretamente relacionada à arquitetura do computador (hardware)

Cada processador possui sua própria linguagem Assembly: ARM, 68000

Assembly

Linguagem de baixo nível ou "linguagem de montagem"

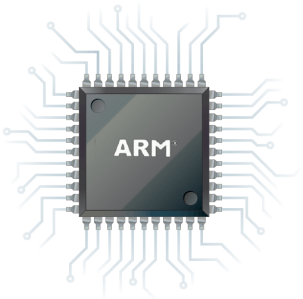
Substitui o código de máquina por uma linguagem mais simbólica

Facilita a programação!

```
1     VTMP    DS.W    1
2     a       DS.W    1
3     b       DS.W    1
4     valor   DS.W    1
5             MOVE.W  #10,D1
6             MOVE.W  D1,a
7             MOVE.W  #20,D1
8             MOVE.W  D1,b
9             MOVE.W  b,D1
10            ADD.W   #20,D1
11            MOVE.W  D1,VTMP
12            MOVE.W  a,D1
13            MULS.W  VTMP,D1
14            MOVE.W  D1,VTMP
15            MOVE.W  VTMP,D1
16            MOVE.W  D1,valor
```

ADD R1,R2,R3

00110101010101010101



Assembly

- ***Breve história da Linguagem Assembly***

Programas eram escritos em linguagem de máquina

Década de 50

Assembly surge para facilitar a programação

Surgimento das linguagens de alto nível

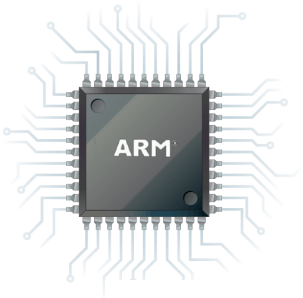
Década de 70/80

Assembly passou a ser pouco usada

"Era dos microprocessadores" e sistemas embarcados

Década de 90 até hoje

Assembly "volta" na busca de desempenho e velocidade

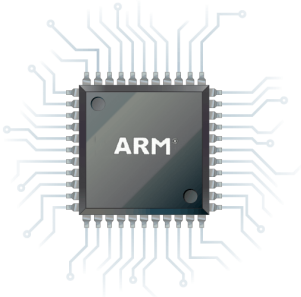


Assembly

Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2017	2012	2007	2002	1997	1992	1987
Java	1	1	1	1	12	-	-
C	2	2	2	2	1	1	1
C++	3	3	3	3	2	2	4
C#	4	4	7	17	-	-	-
Python	5	7	6	11	27	-	-
Visual Basic .NET	6	19	-	-	-	-	-
PHP	7	6	4	5	-	-	-
JavaScript	8	9	8	8	19	-	-
Perl	9	8	5	4	4	11	-
Assembly language	10	-	-	-	-	-	-
COBOL	25	28	17	9	3	10	8
Lisp	32	12	14	12	9	5	2
Prolog	33	32	26	15	20	12	3
Pascal	106	15	19	97	8	3	5



Assembly

Muitas aplicações industriais são feitas em Assembly

Acesso direto e controle total do hardware

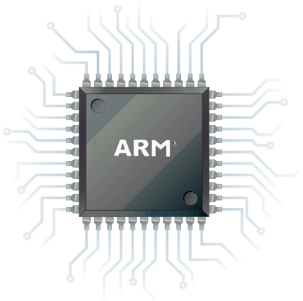
Mais eficiente
Não gera códigos supérfluos

Porque aprender a Linguagem Assembly?

Programas exigem menos memória e são menores

Facilita a programação em alto nível

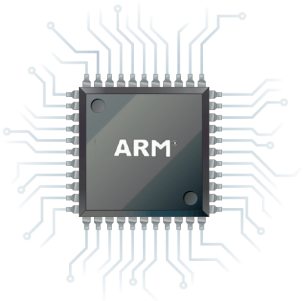
Possibilidade de desenvolver rotinas mais eficazes e incorporá-las a programas de alto nível



Assembly

O que é preciso saber para programar em Assembly?

- Conhecer o hardware: organização da memória, registradores, periféricos, arquitetura, etc.
- Conhecer profundamente os números binários: ponto flutuante, overflow, número negativo etc.
- Conhecer o conjunto de instruções do processador



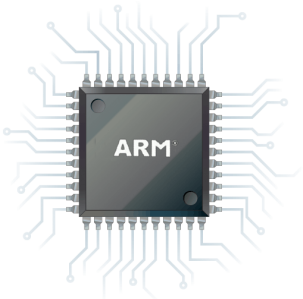
Assembly ARM

Conhecendo um pouco do Hardware do ARM ...

Principais Características (ARM7TDMI)

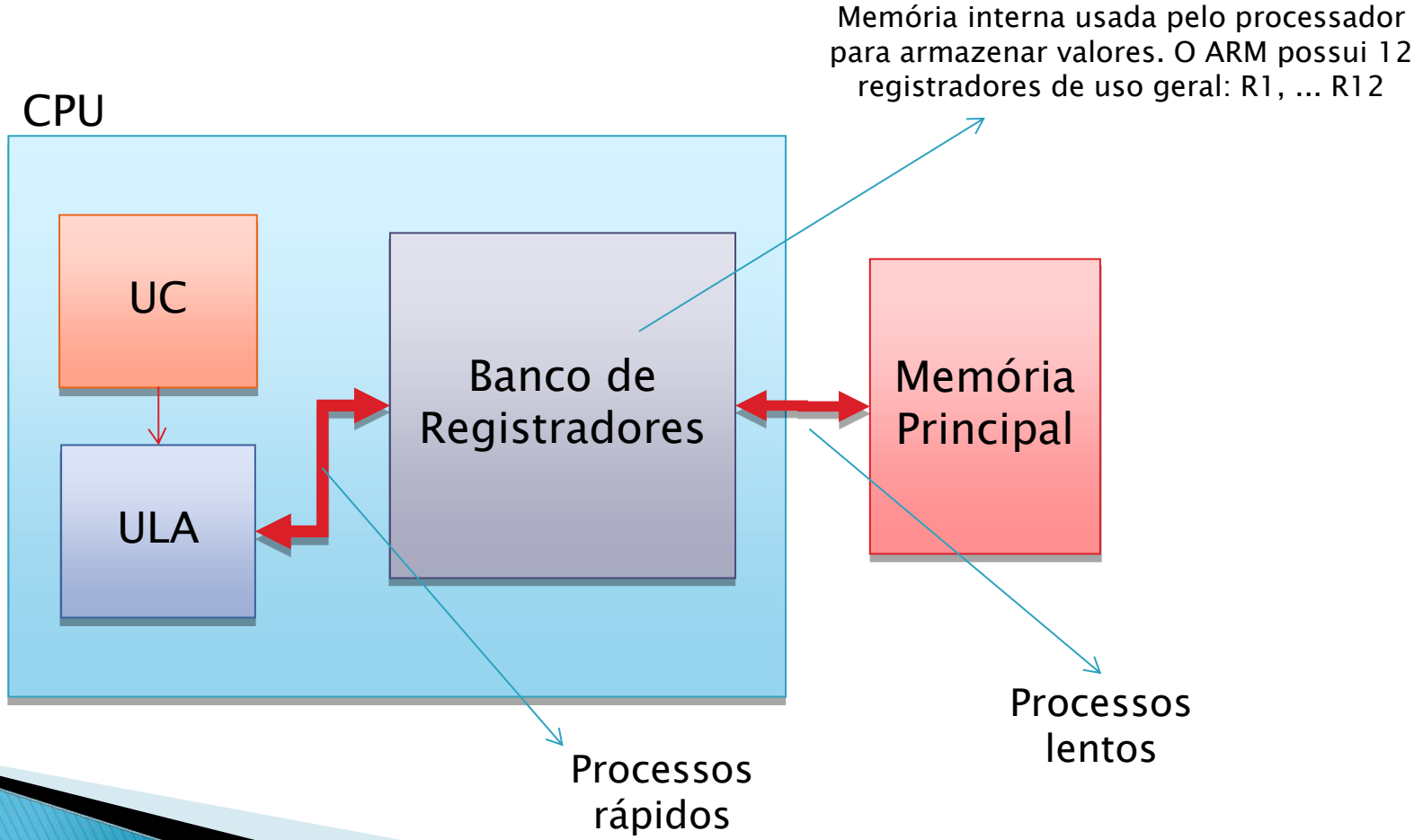
- Arquitetura de 32 bits
- Instruções simples executadas em um ciclo de relógio
- Grande número de registradores
- Dois conjuntos de instruções: ARM (32 bits) Thumb (16 bits)
- Arquitetura LOAD-STORE

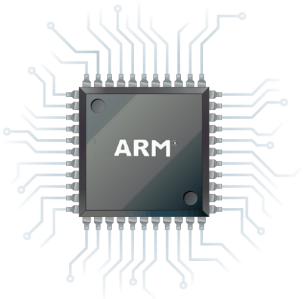
  Dados são colocados na CPU para serem modificados e depois reescritos na memória



Assembly ARM

- *Como se organizam os componentes em uma arquitetura ARM?*





Assembly ARM

```
1       START     ADD.L     D0,D1
2                 JMP       NEXT
3       LOOP      ADD.L     #1,D1
4       NEXT      CLR.L     D5
5                 JMP       LOOP
```

Mas, como escrevemos programas em Assembly?

Um Programa é composto por uma sequência de instruções armazenadas na memória em linguagem de máquina

Determinam as operações que o processador fará (ex: soma, movimento, subtração, comparação etc)

E como são compostas essas instruções?

- Um operador → Mnemônico
- Zero, um ou mais operandos

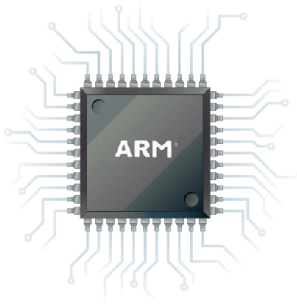
1°: local que será armazenado o resultado da operação

Outros: dados imediatos ou locais dos dados-fonte

Ex: ADD R1 R2 #4

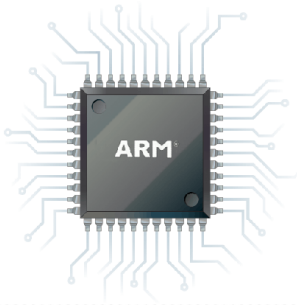
$R1 \leftarrow (R2) + 4$

Processador ARM



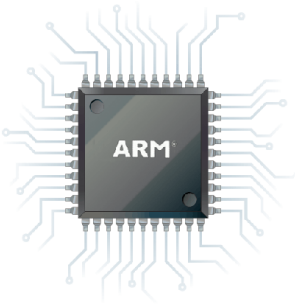
- **Características (ARM7TDMI):**
  - Representação em 32 bits.
  - 16 registradores visíveis a cada momento:
    - r0 – r12: uso geral
    - r13: *stack pointer*
    - r14: *link*
    - r15: PC
  - Dois conjuntos de instruções: ARM (32 bits) e THUMB (16 bits).
  - Arquitetura *load-store*: as instruções somente lidam com valores que estejam nos registradores (ou imediatos) e sempre armazenam resultados em algum registrador. O acesso à memória é feito apenas através das instruções *load* (para carregar o valor da memória em um registrador) e *store* (para armazenar o valor de um registrador na memória).

Processador ARM



- **Características (ARM7TDMI):**
  - Permite a execução condicional de várias instruções.
  - Máquina de 3 endereços – dois registradores operandos e um registrador de resultado são especificados.
    - 1º operando: especifica o local onde será armazenado o resultado da operação.
    - Outros: valores imediatos ou locais onde se encontram os dados.
    - **Exemplo:** ADD R1, R2, #4 R1 ← (R2) + 4
  - Uniformidade e tamanhos fixos dos campos das instruções para facilitar a decodificação.
  - I/O mapeado em memória: a comunicação com dispositivos periféricos é feita no próprio espaço de endereçamento de memória (via *load* e *store*).
  - Endereços de memória se referem a *bytes* individuais.

Processador ARM



- **Operandos:**
  - Registradores:
    - ADD r1, r2, r3
    - MOV r1, r2
  - Imediatos: identificados pelo símbolo #
    - ADD r1, r2, #10
    - MOV r1, #0

- **Tipos de instruções:**
  - Processamento de dados: operações lógico-aritméticas e de movimentação entre registradores.
    - **Exemplos:** ADD, SUB, MOV
  - Fluxo de controle: instruções de desvio (*branch*)
    - **Exemplo:** B rótulo
  - Carregamento / Armazenamento: *load-store*

Processador ARM

- **Linguagem Assembly:**

LABEL: MNEMÔNICO Op1, Op2, Op3 @ comentários

- LABEL: rótulo que serve para identificar o endereço de memória onde será colocada a instrução.
- MNEMÔNICO: identifica a operação desejada (MOV, ADD, B etc.).
- Op1: registrador alvo.
- Op2 e Op3: registradores, imediatos ou endereços (desvio).

Processador ARM

- **Exemplo de um programa:**

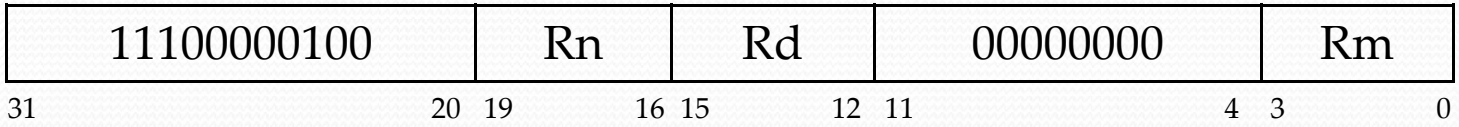
... E como ele está alocado na memória.

```
 1 00000000                     AREA            Demo, CODE, READONLY
 2 00000000                     IMPORT          main
 3 00000000                     EXPORT          start
 4 00000000
 5 00000000           start
 6 00000000 E1A00001            MOV             r0, r1
 7 00000004 E0800002            ADD             r0, r0, r2
 8 00000008 E0800003            ADD             r0, r0, r3
 9 0000000C E0800004            ADD             r0, r0, r4
10 00000010
11 00000010 EAFFFFFE
                       stop     B               stop
```

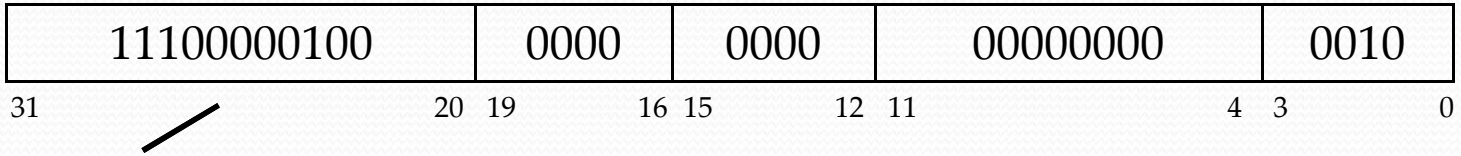
Processador ARM

- **Como uma instrução é representada?**
  - Ao utilizar uma instrução, é importante saber como ela é escrita na memória.
  - Lembrando que a arquitetura ARM é de 32 bits, analisemos o seguinte exemplo:

ADD Rd, Rn, Rm



Caso particular: ADD r0, r0,r2



Contém o código da operação, a indicação da presença de um imediato, a condição para sua execução etc.

- **Dica:** conferir o *datasheet* completo do ARM (*link* na página).

Processador ARM

- **Como uma instrução é representada?**
  - Especificação de um valor imediato na própria instrução.

ADD Rd, Rn, #imm

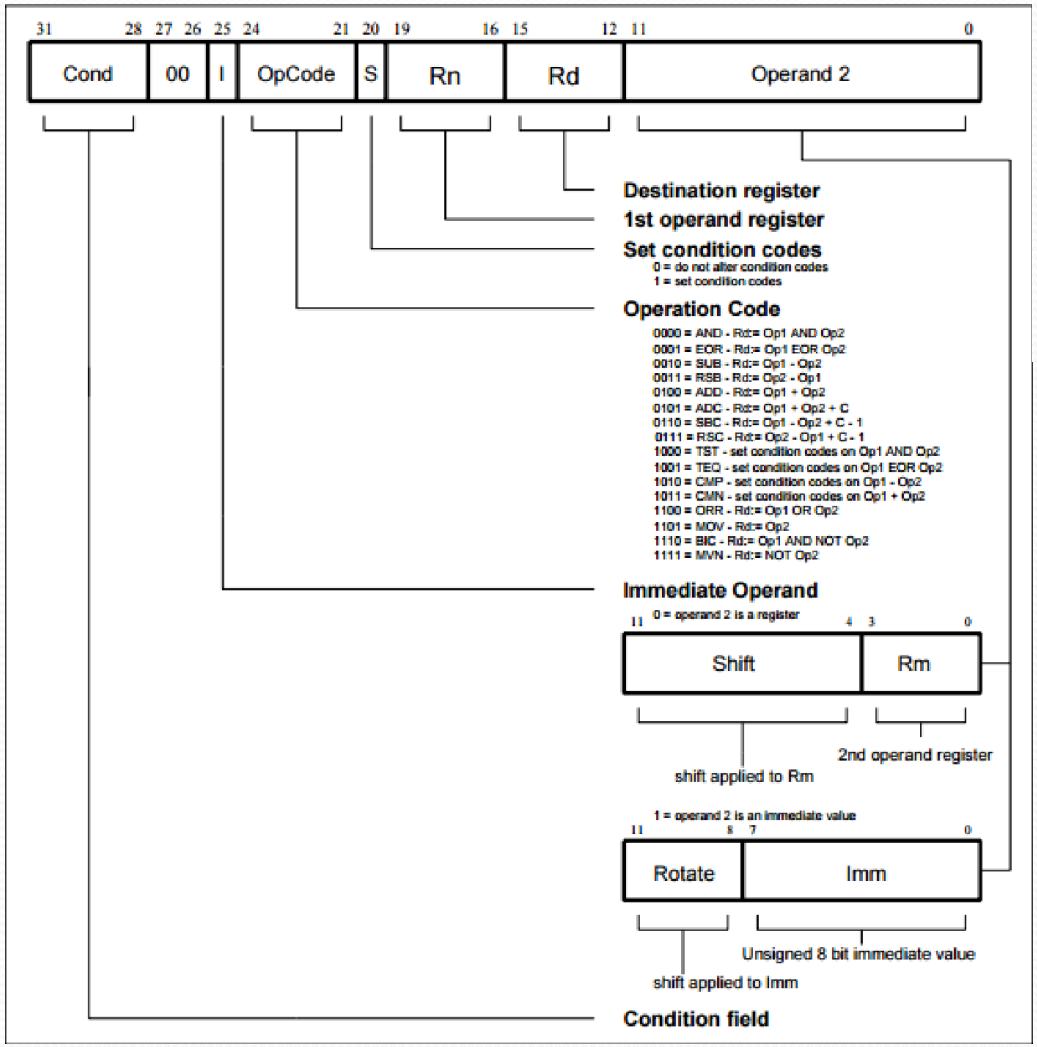
11100010100	Rn	Rd	deslocamento	imm
31 20	19 16	15 12	11 8	7 0

Caso particular: ADD r0, r0, #15

11100010100	0000	0000	0000	00001111
31 20	19 16	15 12	11 8	7 0

Processador ARM

- **Como uma instrução é representada?**



Processador ARM

- **Como uma instrução é representada?**

  - Em um caso mais geral de instrução com um imediato, o valor do campo deslocamento multiplicado por 2 define o número de rotações à direita.

  - O campo imediato é visto como um número de 8 bits sem sinal.

  - **Exemplo:** MOV r0, #65536

1110	00	1	1101	0	0000	0000	1000	00000001
Cond		I	Opcode	S			Deslocamento	Imm

Processador ARM

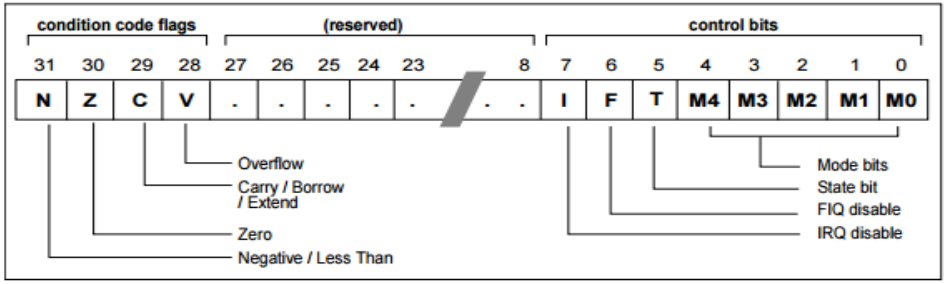
Instruções de Processamento de Dados

	aritmética		lógica		movimento
manipulação	ADD	SUB	AND	EOR	MOV
comparação	CMN	CMP	TST	TEQ	

Exemplo: CMP R1, R2

- Se R1 for igual a R2, uma flag é setada, a qual poderá ser testada por outras instruções ...

Registrador de estado



Processador ARM

Instruções de Armazenamento e Carregamento

- São as instruções de acesso a memória.

  - LDR: carrega no registrador Rd o conteúdo de uma posição de memória

    Exemplo:

    LDR R0, [R1] — R0 vai receber o conteúdo da memória que está na posição indicada por R1

  - STR: coloca o conteúdo do registrador Rd em uma determinada posição de memória

    Exemplo:

    STR R0, [R1] — A posição de memória indicada por R1 irá receber o conteúdo de R0

Processador ARM

Instruções de Armazenamento e Carregamento

Registradores

R0	500
R1	12
R2	14

Memória

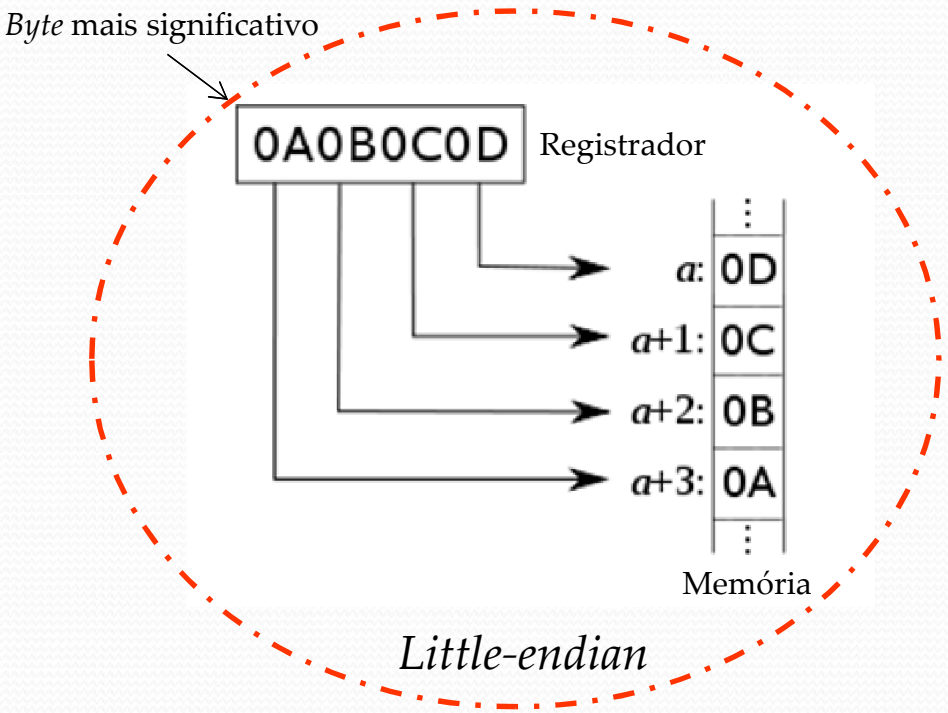
Endereço	Conteúdo
11	530
12	500
13	150
14	620

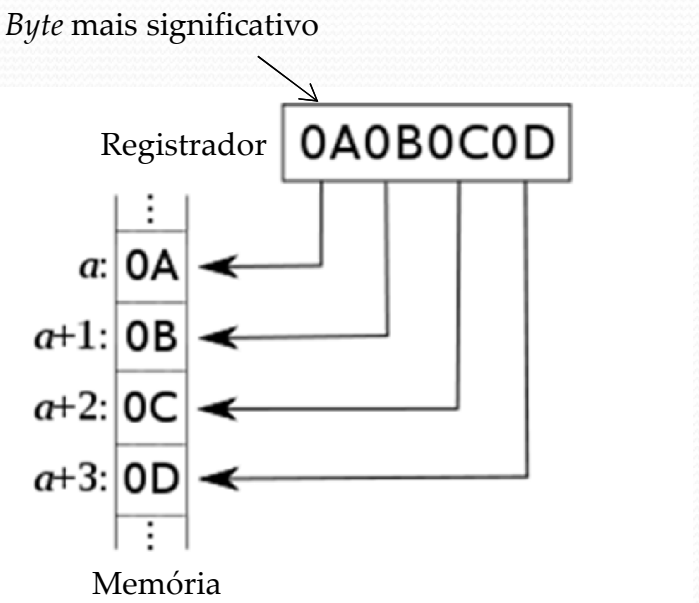
Executando: LDR R0, [R1]

STR R0, [R2]

Processador ARM

Disposição dos *bytes* na memória





Adotado no ARMSim

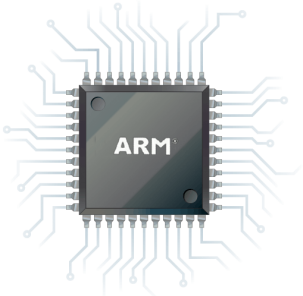
Processador ARM

Instruções de Desvio

- **Desvio incondicional:** B rótulo

- **Desvio condicional:** B*cond* rótulo
  - O sufixo *cond* determina o tipo de condição a ser testada – envolve os bits do registrador de estado.

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always



Assembly ARM

Simulador

- Exemplos

Referências

- ARMSim: http://armsim.cs.uvic.ca/
- ARM Instruction set: http://infocenter.arm.com/help/topic/com.arm.doc.qrc0001l/QRC0001_UAL.pdf