Microprocessadores e Microcontroladores (27146)

Conjunto de instruções (ARM)

Prof. Ricardo Menotti (menotti@ufscar.br)

Atualizado em: 26 de abril de 2021

Departamento de Computação
Centro de Ciências Exatas e de Tecnologia
Universidade Federal de São Carlos

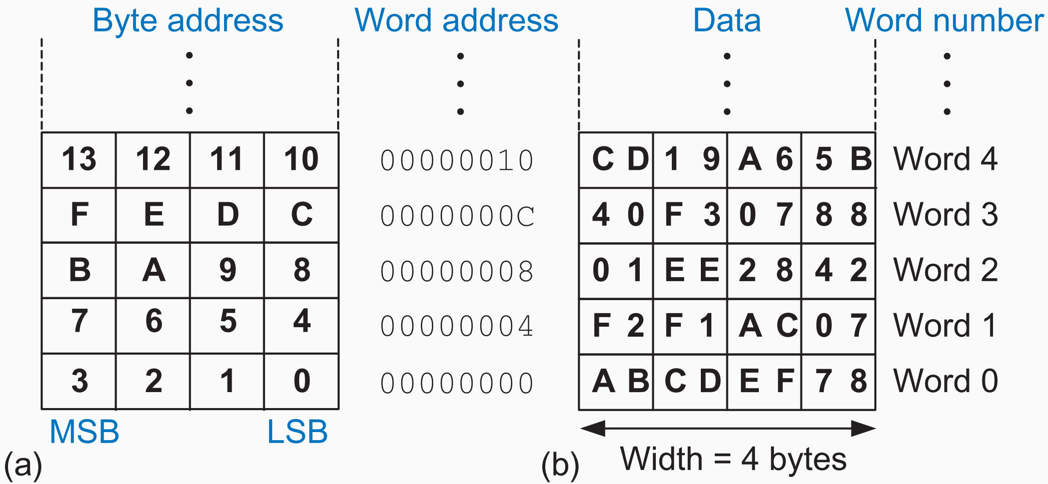
Sobre a arquitetura ARM

- Desenvolvida na década de 80 pela Advanced RISC Machines Ltd.
- Mais de 10 bilhões de processadores vendidos todos os anos (2016);
- ARM não vende processadores diretamente, mas licencia para outros fabricantes;

Particularidade

- A maioria dos conjuntos de instruções só permite que saltos sejam executados condicionalmente;
- Os processadores ARM possuem um mecanismo de execução condicional de instruções:
  - Todas as instruções contém um campo de condição, indicando as circunstâncias de execução;
  - Reusando o hardware de comparação, aumenta-se efetivamente o número de instruções;
  - Isso elimina a necessidade de muitos saltos;
- Os processadores ARM possuem uma unidade de deslocamento (*Barrel Shifter*) para o segundo operando da ULA:
  - Se o fator for um imediato, a operação não adiciona nenhum ciclo à instrução;
  - Isso pode ser usado, por exemplo, para escalar endereços;

Endereçamento



Registadores [Harris and Harris(2016)]

Table 6.1 ARM register set

Name	Use
R0	Argument / return value / temporary variable
R1–R3	Argument / temporary variables
R4–R11	Saved variables
R12	Temporary variable
R13 (SP)	Stack Pointer
R14 (LR)	Link Register
R15 (PC)	Program Counter

Sintaxe [Harris and Harris(2016)]

Code Example 6.8 READING MEMORY

High-Level Code

```
a = mem[2];
```

ARM Assembly Code

```
; R7 = a
  MOV R5, #0          ; base address = 0
  LDR R7, [R5, #8]    ; R7 <= data at memory address (R5+8)
```

Code Example 6.9 WRITING MEMORY

High-Level Code

```
mem[5] = 42;
```

ARM Assembly Code

```
MOV R1, #0            ; base address = 0
MOV R9, #42
STR R9, [R1, #0x14]   ; value stored at memory address (R1+20) = 42
```

Operações Lógicas [Harris and Harris(2016)]

Source registers

R1	0100 0110	1010 0001	1111 0001	1011 0111
R2	1111 1111	1111 1111	0000 0000	0000 0000

Assembly code

```
AND  R3, R1, R2
ORR  R4, R1, R2
EOR  R5, R1, R2
BIC  R6, R1, R2
MVN  R7, R2
```

Result

R3	0100 0110	1010 0001	0000 0000	0000 0000
R4	1111 1111	1111 1111	1111 0001	1011 0111
R5	1011 1001	0101 1110	1111 0001	1011 0111
R6	0000 0000	0000 0000	1111 0001	1011 0111
R7	0000 0000	0000 0000	1111 1111	1111 1111

Deslocamentos [Harris and Harris(2016)]

Source register

R5	1111 1111	0001 1100	0001 0000	1110 0111

Assembly Code		Result			
`LSL R0, R5, #7`	R0	1000 1110	0000 1000	0111 0011	1000 0000
`LSR R1, R5, #17`	R1	0000 0000	0000 0000	0111 1111	1000 1110
`ASR R2, R5, #3`	R2	1111 1111	1110 0011	1000 0010	0001 1100
`ROR R3, R5, #21`	R3	1110 0000	1000 0111	0011 1111	1111 1000

Deslocamentos [Harris and Harris(2016)]

Source register

R5	1111 1111	0001 1100	0001 0000	1110 0111

Assembly Code		Result			
`LSL R0, R5, #7`	R0	1000 1110	0000 1000	0111 0011	1000 0000
`LSR R1, R5, #17`	R1	0000 0000	0000 0000	0111 1111	1000 1110
`ASR R2, R5, #3`	R2	1111 1111	1110 0011	1000 0010	0001 1100
`ROR R3, R5, #21`	R3	1110 0000	1000 0111	0011 1111	1111 1000

Source registers

R8	0000 1000	0001 1100	0001 0110	1110 0111
R6	0000 0000	0000 0000	0000 0000	0001 0100

Assembly code		Result			
`LSL R4, R8, R6`	R4	0110 1110	0111 0000	0000 0000	0000 0000
`ROR R5, R8, R6`	R5	1100 0001	0110 1110	0111 0000	1000 0001

CPU Flags [Harris and Harris(2016)]

Table 6.2 Condition flags

Flag	Name	Description
N	Negative	Instruction result is negative, i.e., bit 31 of the result is 1
Z	Zero	Instruction result is zero
C	Carry	Instruction causes a carry out
V	oVerflow	Instruction causes an overflow

Execução condicional [Harris and Harris(2016)]

cond	Mnemonic	Name	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\overline{Z}$
0010	CS/HS	Carry set / unsigned higher or same	C
0011	CC/LO	Carry clear / unsigned lower	$\overline{C}$
0100	MI	Minus / negative	N
0101	PL	Plus / positive or zero	$\overline{N}$
0110	VS	Overflow / overflow set	V
0111	VC	No overflow / overflow clear	$\overline{V}$
1000	HI	Unsigned higher	$\overline{Z}C$
1001	LS	Unsigned lower or same	Z OR $\overline{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\overline{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	Z OR $(N \oplus V)$
1110	AL (or none)	Always / unconditional	Ignored

	Unsigned	**Signed**
A = 1001_2	A = 9	A = –7
B = 0010_2	B = 2	B = 2

(a)

A – B: 1001
+ 1110
10111

$NZCV = 0011_2$
HS: TRUE
GE: FALSE

	Unsigned	**Signed**
A = 0101_2	A = 5	A = 5
B = 1101_2	B = 13	B = –3

(b)

A – B: 0101
+ 0011
1000

$NZCV = 1001_2$
HS: FALSE
GE: TRUE

Figure 6.7 Signed vs. unsigned comparison: HS vs. GE

Execução condicional [Harris and Harris(2016)]

$R2 = 0x80000000$, $R3 = 0x00000001$

$R2 - R3 = 0x80000000 + 0xFFFFFFFF = 0x7FFFFFFF \Rightarrow C{=}1,\ V{=}1,\ N{=}0,\ Z{=}0$

Code Example 6.10 CONDITIONAL EXECUTION

ARM Assembly Code

```
CMP    R2, R3
ADDEQ  R4, R5, #78
ANDHS  R7, R8, R9
ORRMI  R10, R11, R12
EORLT  R12, R7, R10
```

ADDEQ ✗ ANDHS ✓ ORRMI ✗ EORLT ✓

Salto [in]condicional [Harris and Harris(2016)]

Code Example 6.11 UNCONDITIONAL BRANCHING

ARM Assembly Code

```
  ADD R1, R2, #17     ; R1 = R2 + 17
  B   TARGET          ; branch to TARGET
  ORR R1, R1, R3      ; not executed
  AND R3, R1, #0xFF   ; not executed

TARGET
  SUB R1, R1, #78     ; R1 = R1 − 78
```

Code Example 6.12 CONDITIONAL BRANCHING

ARM Assembly Code

```
  MOV R0, #4         ; R0 = 4
  ADD R1, R0, R0     ; R1 = R0 + R0 = 8
  CMP R0, R1         ; set flags based on R0−R1 = −4. NZCV = 1000
  BEQ THERE          ; branch not taken (Z != 1)
  ORR R1, R1, #1     ; R1 = R1 OR 1 = 9

THERE
  ADD R1, R1, #78    ; R1 = R1 + 78 = 87
```

Estrutura If/Else [Harris and Harris(2016)]

Code Example 6.14 IF/ELSE STATEMENT

High-Level Code	ARM Assembly Code

<pre>if (apples == oranges)
  f = i + 1;
 	
else
  f = f − i;</pre>
	<pre>; R0 = apples, R1 = oranges, R2 = f, R3 = i
	  CMP R0, R1 ; apples == oranges?
	  BNE L1 ; if not equal, skip if block
	  ADD R2, R3, #1 ; if block: f = i + 1
	  B L2 ; skip else block
	L1
	  SUB R2, R2, R3 ; else block: f = f − i
	L2</pre>

Estrutura If/Else [Harris and Harris(2016)]

Code Example 6.14 IF/ELSE STATEMENT

High-Level Code	ARM Assembly Code

<pre>if (apples == oranges)
  f = i + 1;
 	
else
  f = f − i;</pre>
	<pre>; R0 = apples, R1 = oranges, R2 = f, R3 = i
	  CMP R0, R1 ; apples == oranges?
	  BNE L1 ; if not equal, skip if block
	  ADD R2, R3, #1 ; if block: f = i + 1
	  B L2 ; skip else block
	L1
	  SUB R2, R2, R3 ; else block: f = f − i
	L2</pre>

```
CMP   R0, R1        ; apples == oranges?
ADDEQ R2, R3, #1    ; f = i + 1 on equality (i.e., Z = 1)
SUBNE R2, R2, R3    ; f = f − i on not equal (i.e., Z = 0)
```

Estrutura Case [Harris and Harris(2016)]

Code Example 6.15 SWITCH/CASE STATEMENT

High-Level Code	ARM Assembly Code
	; R0 = button, R1 = amt
switch (button) {	

case 1: amt = 20; break;	CMP R0, #1 ; is button 1 ?
	MOVEQ R1, #20 ; amt = 20 if button is 1
	BEQ DONE ; break

case 2: amt = 50; break;	CMP R0, #2 ; is button 2 ?
	MOVEQ R1, #50 ; amt = 50 if button is 2
	BEQ DONE ; break

case 3: amt = 100; break;	CMP R0, #3 ; is button 3?
	MOVEQ R1, #100 ; amt = 100 if button is 3
	BEQ DONE ; break

default: amt = 0;	MOV R1, #0 ; default amt = 0
}	DONE

```
// equivalent function using
// if/else statements
  if      (button == 1) amt = 20;
  else if (button == 2) amt = 50;
  else if (button == 3) amt = 100;
  else                  amt = 0;
```

Estrutura While [Harris and Harris(2016)]

Code Example 6.16 WHILE LOOP

High-Level Code

```
int pow = 1;
int x = 0;

while (pow != 128) {
  pow = pow * 2;
  x = x + 1;
}
```

ARM Assembly Code

```
; R0 = pow, R1 = x
  MOV R0, #1        ; pow = 1
  MOV R1, #0        ; x = 0

WHILE
  CMP R0, #128      ; pow != 128 ?
  BEQ DONE          ; if pow == 128, exit loop
  LSL R0, R0, #1    ; pow = pow * 2
  ADD R1, R1, #1    ; x = x + 1
  B   WHILE         ; repeat loop
DONE
```

Estrutura For [Harris and Harris(2016)]

Code Example 6.17 FOR LOOP

High-Level Code

```
int i;
int sum = 0;


for (i = 0; i < 10; i = i + 1) {
  sum = sum + i;
}
```

ARM Assembly Code

```
; R0 = i, R1 = sum
  MOV R1, #0        ; sum = 0
  MOV R0, #0        ; i = 0          loop initialization

FOR
  CMP R0, #10       ; i < 10 ?       check condition
  BGE DONE          ; if (i >= 10) exit loop
  ADD R1, R1, R0    ; sum = sum + i  loop body
  ADD R0, R0, #1    ; i = i + 1      loop operation
  B   FOR           ; repeat loop
DONE
```

Acesso a arranjos [Harris and Harris(2016)]

Code Example 6.18 ACCESSING ARRAYS USING A FOR LOOP

High-Level Code	ARM Assembly Code

`int i;`
`int scores[200];`
`...`
	`; R0 = array base address, R1 = i`
	`; initialization code ...`
	`  MOV R0, #0x14000000   ; R0 = base address`
	`  MOV R1, #0            ; i = 0`

`for (i = 0; i < 200; i = i + 1)`
 	
`  scores[i] = scores[i] + 10;`
	`LOOP`
	`  CMP R1, #200          ; i < 200?`
	`  BGE L3                ; if i ≥ 200, exit loop`
	`  LSL R2, R1, #2        ; R2 = i * 4`
	`  LDR R3, [R0, R2]      ; R3 = scores[i]`
	`  ADD R3, R3, #10       ; R3 = scores[i] + 10`
	`  STR R3, [R0, R2]      ; scores[i] = scores[i] + 10`
	`  ADD R1, R1, #1        ; i = i + 1`
	`  B   LOOP              ; repeat loop`
	`L3`

Modos de indexação [Harris and Harris(2016)]

Table 6.4 ARM indexing modes

Mode	ARM Assembly	Address	Base Register
Offset	`LDR R0, [R1, R2]`	R1 + R2	Unchanged
Pre-index	`LDR R0, [R1, R2]!`	R1 + R2	R1 = R1 + R2
Post-index	`LDR R0, [R1], R2`	R1	R1 = R1 + R2

Acesso a arranjos com pós-incremento [Harris and Harris(2016)]

Code Example 6.19 FOR LOOP USING POST-INDEXING

High-Level Code

```
int i;
int scores[200];
...

for (i = 0; i < 200; i = i + 1)
    scores[i] = scores[i] + 10;
```

ARM Assembly Code

```
; R0 = array base address
; initialization code      ...
  MOV R0, #0x14000000      ; R0 = base address
  ADD R1, R0, #800         ; R1 = base address + (200*4)

LOOP
  CMP R0, R1               ; reached end of array?
  BGE L3                   ; if yes, exit loop
  LDR R2, [R0]             ; R2 = scores[i]
  ADD R2, R2, #10          ; R2 = scores[i] + 10
  STR R2, [R0], #4         ; scores[i] = scores[i] + 10
                           ; then R0 = R0 + 4
  B   LOOP                 ; repeat loop
L3
```

Para saber mais e praticar...

- https://www.arm.com/resources
- https://salmanarif.bitbucket.io/visual/
- https://cpulator.01xz.net/?sys=arm
- https://azm.azerialabs.com/
- https://www.edaplayground.com/x/vcGc

Bibliografia

David Harris and Sarah Harris.
Digital Design and Computer Architecture: ARM® Edition.
Morgan Kaufmann, 2016.