



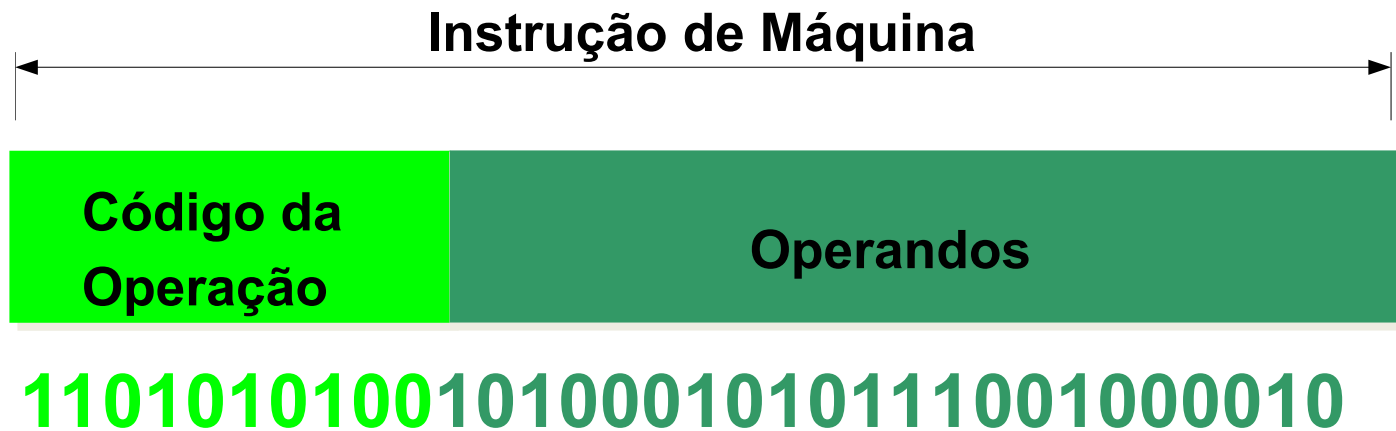
Universidade Federal da Fronteira Sul
Curso de Ciência da Computação
Campus Chapecó



Arquitetura do Conjunto de Instruções e Formato de Representação das Instruções

Prof. Luciano L. Caimi
lcaimi@uffs.edu.br

Arquitetura do Conjunto de Instruções



- A **instrução de máquina** é um conjunto de bits contendo dois campos:
 - Código da Operação (opcode): o que a instrução faz
Exemplos: add; lw; jump; beq
 - Operandos (operand): onde estão os dados a serem manipulados

Classes de instruções



Instruções lógicas e aritméticas

- ADD, SUB, AND, XOR, etc...

Instruções de desvio condicional

- BNE, BEQ, BLT, etc...

Instruções de desvio incondicional

- JUMP, CALL, etc...

Instruções de movimentação de dados

- MOV, LOAD, STORE, etc...

Instruções de controle

- NOP, EN_INT,

► A Instrução Assembly

As instruções assembly são definidas a partir da sua **sintaxe** e **semântica**

A **sintaxe** refere-se ao conjunto de regras que regem a formação do “*texto*”.

A **semântica** refere-se ao significado ou a interpretação deste “*texto*”.

sintaxe

Semântica

push op; pilha_{topo} \leftarrow (op)

onde: () indica acesso a memória

► Arquivo ASM

```
.data
txt_1  ascii  "Organização de Computadores"
var_A  word   -40
cnt     equ    5      # comentário
```

```
.text
main:
    addi    a4,s0,cnt
    add     a5,a4,a5
    j       L2
L3:
    lw      a5,-20(s0)
    slli    a5,a5,2
    addi    a4,s0,-16
    lw      a5,-20(s0)
    li      a5,99
    ble     a4,a5, L3
L2:
    lw      a4,-20(s0)
    li      a5,99
```

```
.data
txt_1  ascii  "Organização de Computadores"
var_A  word   -40
cnt     equ    5      # comentário

.text
main:
    addi    a4,s0,cnt
    add     a5,a4,a5
    j       L2
L3:
    lw      a5,-20(s0)
    slli    a5,a5,2
    addi    a4,s0,-16
    lw      a5,-20(s0)
    li      a5,99
    ble     a4,a5, L3
L2:
    lw      a4,-20(s0)
    li      a5,99
```

Arquitetura do Conjunto de Instruções



```
1 int main() {  
2     int num1[100];  
3     int num2[100];  
4     int result[100];  
5     for(int x = 0; x < 100; x++){  
6         result[x] = num1[x] + num2[x];  
7     }  
8 }
```

```
1 main:  
2     addi    sp,sp,-1232  
3     sw      s0,1228(sp)  
4     addi    s0,sp,1232  
5     sw      zero,-20(s0)  
6     j       .L2  
7 .L3:  
8     lw      a5,-20(s0)  
9     slli    a5,a5,2  
10    addi    a4,s0,-16  
11    add     a5,a4,a5  
12    lw      a4,-404(a5)  
13    lw      a5,-20(s0)  
14    slli    a5,a5,2  
15    addi    a3,s0,-16  
16    add     a5,a3,a5  
17    lw      a5,-804(a5)  
18    add     a4,a4,a5  
19    lw      a5,-20(s0)  
20    slli    a5,a5,2  
21    addi    a3,s0,-16  
22    add     a5,a3,a5  
23    sw      a4,-1204(a5)  
24    lw      a5,-20(s0)  
25    addi    a5,a5,1  
26    sw      a5,-20(s0)  
27 .L2:  
28    lw      a4,-20(s0)  
29    li      a5,99  
30    ble     a4,a5,.L3  
31    li      a5,0  
32    mv      a0,a5  
33    lw      s0,1228(sp)  
34    addi    sp,sp,1232  
35    jr      ra
```

```
00000000 <main>:  
0:      b3010113      addi x2 x2 -1232  
4:      4c812623      sw x8 1228 x2  
8:      4d010413      addi x8 x2 1232  
c:      fe042623      sw x0 -20 x8  
10:     0500006f      jal x0 80 <L2>  
  
00000014 <L3>:  
14:     fec42783      lw x15 -20 x8  
18:     00279793      slli x15 x15 2  
1c:     ff040713      addi x14 x8 -16  
20:     00f707b3      add x15 x14 x15  
24:     e6c7a703      lw x14 -404 x15  
28:     fec42783      lw x15 -20 x8  
2c:     00279793      slli x15 x15 2  
30:     ff040693      addi x13 x8 -16  
34:     00f687b3      add x15 x13 x15  
38:     cdc7a783      lw x15 -804 x15  
3c:     00f70733      add x14 x14 x15  
40:     fec42783      lw x15 -20 x8  
44:     00279793      slli x15 x15 2  
48:     ff040693      addi x13 x8 -16  
4c:     00f687b3      add x15 x13 x15  
50:     b4e7a623      sw x14 -1204 x15  
54:     fec42783      lw x15 -20 x8  
58:     00178793      addi x15 x15 1  
5c:     fef42623      sw x15 -20 x8  
  
00000060 <L2>:  
60:     fec42703      lw x14 -20 x8  
64:     06300793      addi x15 x0 99  
68:     fae7d6e3      bge x15 x14 -84 <L3>  
6c:     00000793      addi x15 x0 0  
70:     00078513      addi x10 x15 0  
74:     4cc12403      lw x8 1228 x2  
78:     4d010113      addi x2 x2 1232  
7c:     00008067      jalr x0 x1 0
```

```
10110011000000010000000100010011  
01001100100000010010011000100011  
01001101000000010000010000010011  
1111110000001000010011000100011  
000001010000000000000000110111
```

```
1111110110001000010011110000011  
00000000001001111001011110010011  
1111111000001000000011100010011  
000000001110111000001110110011  
11100110110001111010011100000011  
1111110110001000010011110000011  
00000000001001111001011110010011  
1111111000001000000011010010011  
000000001110110100001110110011  
11001101110001111010011110000011  
000000001110111000001100110011  
1111110110001000010011110000011  
0000000000100111100101111000011  
00000000001001111001011110010011  
10110100111001111010011000100011  
1111110110001000010011110000011  
00000000000101111000011110010011  
1111110111101000010011000100011
```

```
1111110110001000010011100000011  
00000110001100000000011110010011  
111101011100111110101011100011  
00000000000000000000011110010011  
00000000000001111000010100010011  
0100110011000001001001000000011  
01001101000000010000000100010011  
000000000000000010000000110111
```

► Classificação quanto ao tipo de armazenamento interno:

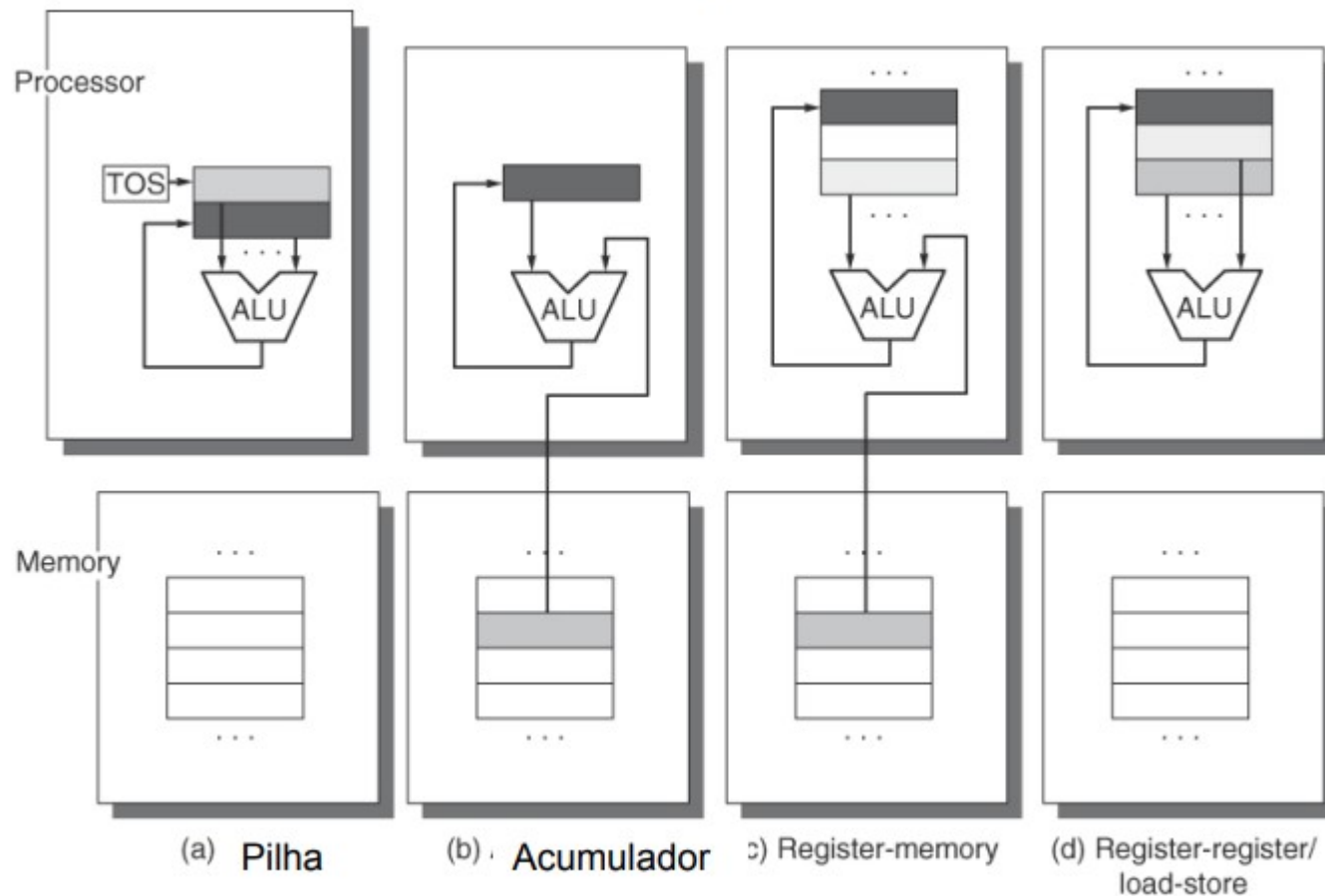
- Arquitetura de Pilha
- Arquitetura de Acumulador
- Arquitetura de Registradores de Propósito Geral
 - *arquitetura register-memory* (obsoleto)
 - *arquitetura load-store* (*register-register*)
- Arquitetura Memória-Memória (obsoleto)

<https://www.cpushack.com/CPU/cpuAppendA.html>

<https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risciscisc/>

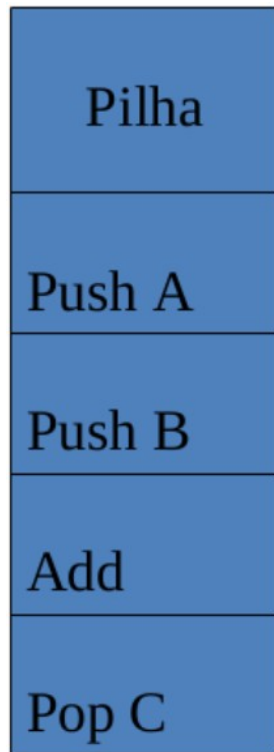
Arquitetura do Conjunto de Instruções

► Classificação quanto ao tipo de armazenamento interno:



Arquitetura do Conjunto de Instruções

Exemplo: $C = A + B$



Pilha

Pop op; $(op) \leftarrow topo$

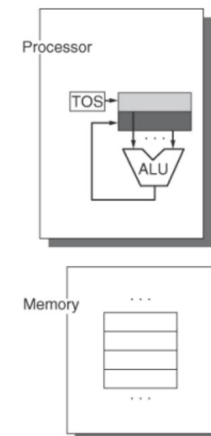
Push op; $topo \leftarrow (op)$

Add ; $topo \leftarrow topo + topo_{-1}$

Sub ; $topo \leftarrow topo - topo_{-1}$

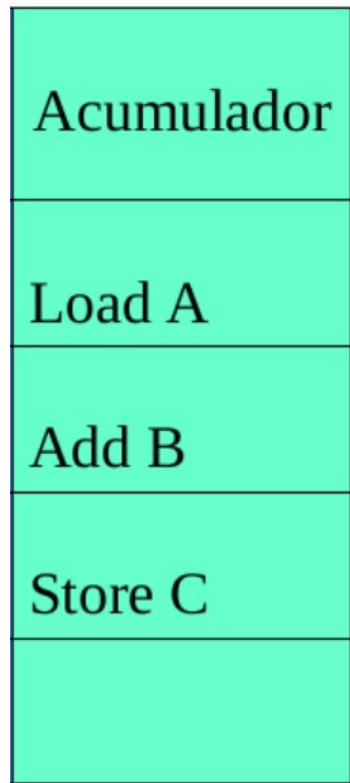
Mul ; $topo \leftarrow topo * topo_{-1}$

Div ; $topo \leftarrow topo / topo_{-1}$

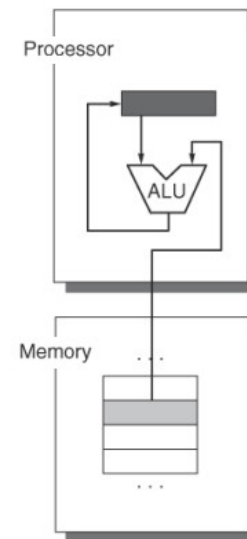


Arquitetura do Conjunto de Instruções

Exemplo: $C = A + B$



```
Store op; w ← (op)
Load op; (op) ← w
Add    op; w ← w + (op)
Sub    op; w ← w - (op)
Mul    op; w ← w * (op)
Div    op; w ← w / (op)
```



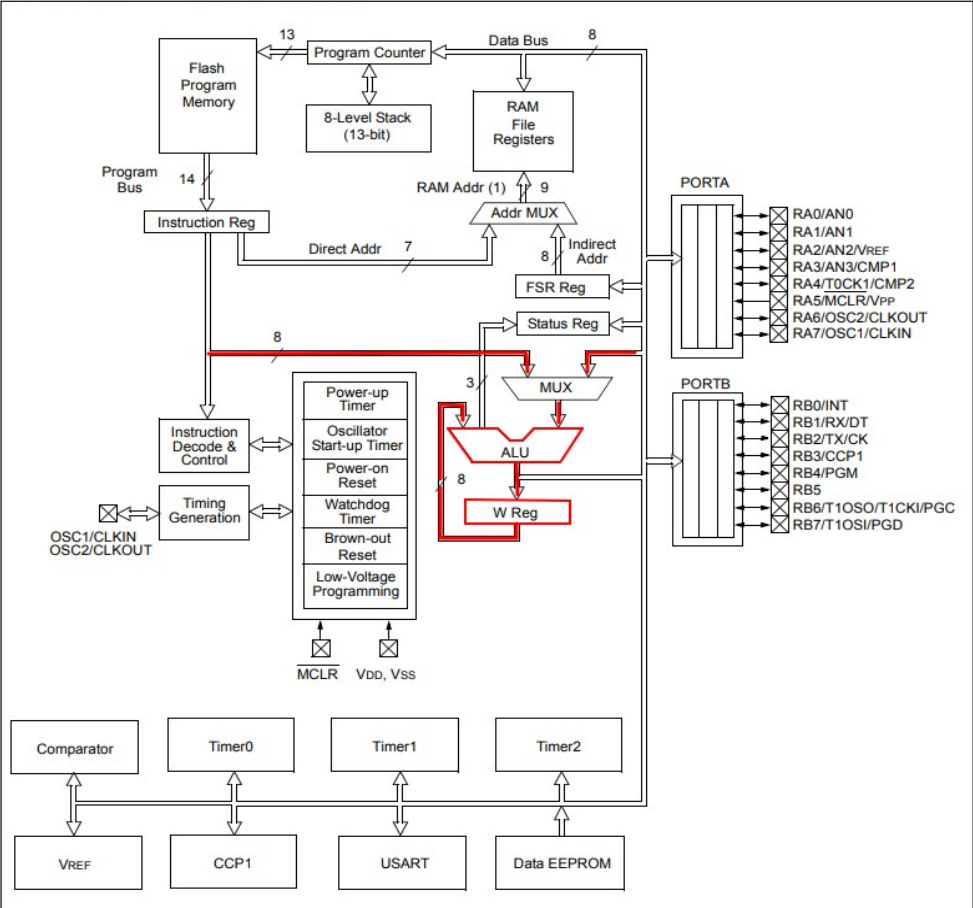
Arquitetura do Conjunto de Instruções



TABLE 15-2: PIC16F627A/628A/648A INSTRUCTION SET

Mnemonic, Operands	Description	Cycles	14-Bit Opcode				Status Affected	Notes	
			MSb		LSb				
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1, 2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1, 2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	—	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1, 2
DECf	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1, 2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1, 2, 3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1, 2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1, 2, 3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1, 2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1, 2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	—	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1, 2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1, 2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1, 2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1, 2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1, 2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1, 2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1, 2
BTFS	f, b	Bit Test f, Skip if Clear	1(2)	01	10bb	bfff	ffff		3
BTSS	f, b	Bit Test f, Skip if Set	1(2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWD	—	Clear Watchdog Timer	1	00	0000	0110	0100	<u>TO</u> , <u>PD</u>	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	—	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	—	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	—	Go into Standby mode	1	00	0000	0110	0011	<u>TO</u> , <u>PD</u>	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

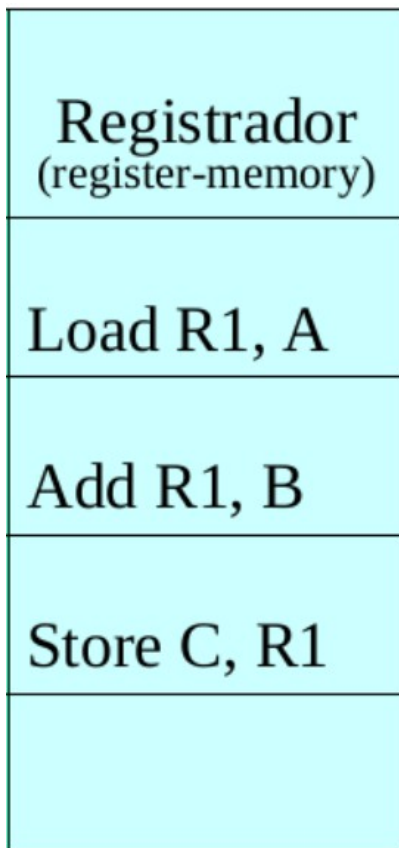
FIGURE 3-1: BLOCK DIAGRAM



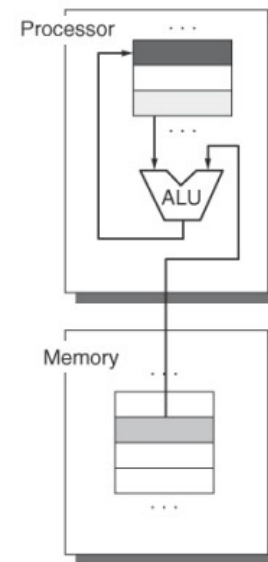
Arquitetura do Conjunto de Instruções



Exemplo: $C = A + B$



```
Load  R, op   ;  $R \leftarrow (op)$   
Store op, R   ;  $(op) \leftarrow R$   
Add   R, op   ;  $R \leftarrow R + (op)$   
Sub   R, op   ;  $R \leftarrow R - (op)$   
Mul   R, op   ;  $R \leftarrow R * (op)$   
Div   R, op   ;  $R \leftarrow R / (op)$ 
```



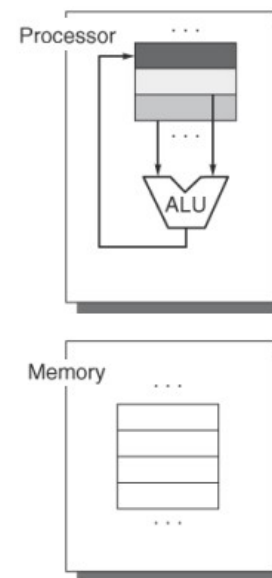
Arquitetura do Conjunto de Instruções



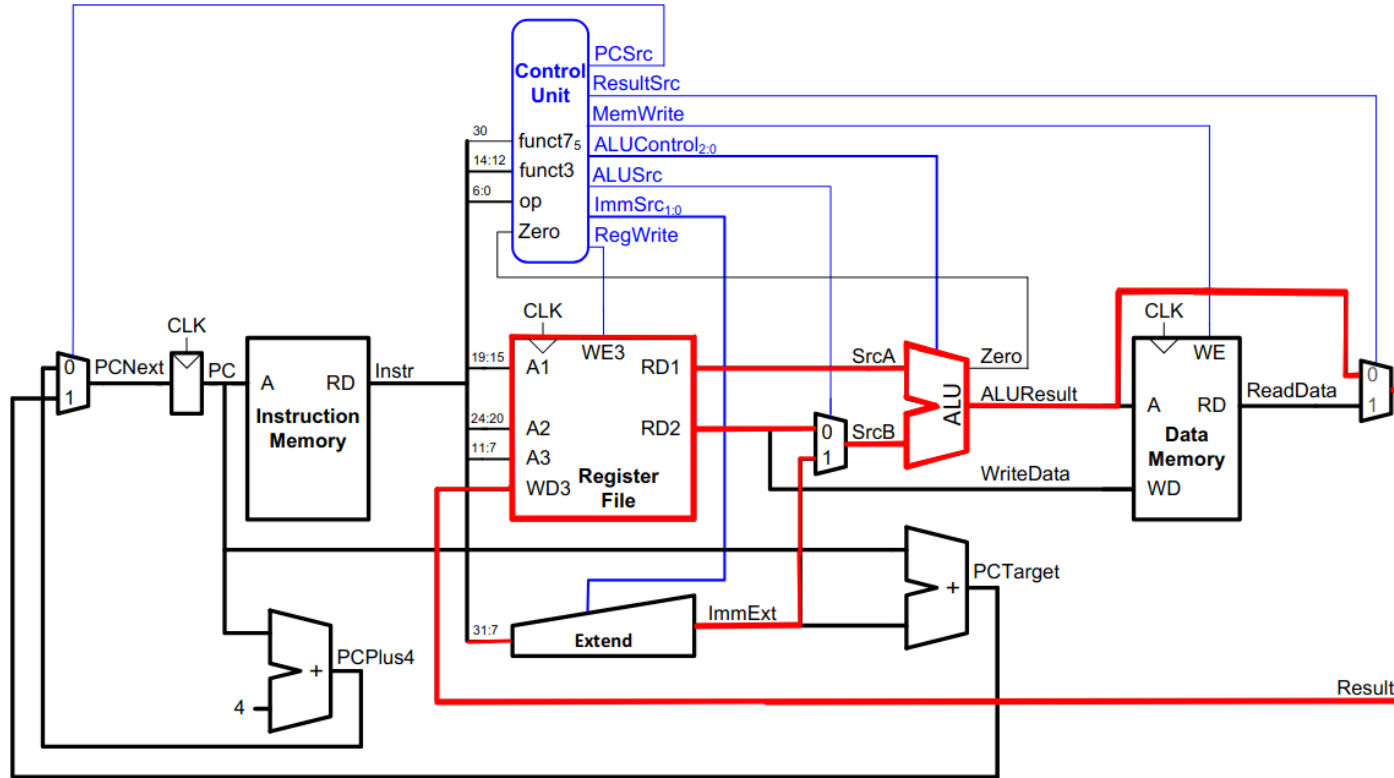
Exemplo: $C = A + B$

Registrador (load-store)
Load R1, A
Load R2, B
Add, R3, R1, R2
Store C, R3

```
Load  R, op      ;  $R \leftarrow (op)$   
Store op, R      ;  $(op) \leftarrow R$   
Add  rd,rs1,rs2  ;  $rd \leftarrow rs1 + rs2$   
Sub  rd,rs1,rs2  ;  $rd \leftarrow rs1 - rs2$   
Mul  rd,rs1,rs2  ;  $rd \leftarrow rs1 * rs2$   
Div  rd,rs1,rs2  ;  $rd \leftarrow rs1 / rs2$ 
```



Arquitetura do Conjunto de Instruções



Pilha

Pop op; $(op) \leftarrow topo$

Push op; $topo \leftarrow (op)$

Add ; $topo \leftarrow topo + topo_{-1}$

Sub ; $topo \leftarrow topo - topo_{-1}$

Mul ; $topo \leftarrow topo * topo_{-1}$

Div ; $topo \leftarrow topo / topo_{-1}$

Arquitetura do Conjunto de Instruções Pilha

Pop op; $(op) \leftarrow topo$

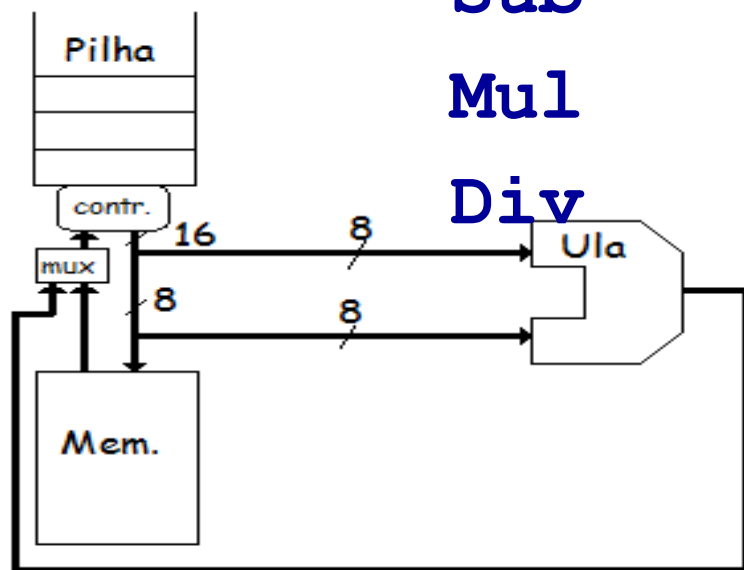
Push op; $topo \leftarrow (op)$

Add ; $topo \leftarrow topo + topo_{-1}$

Sub ; $topo \leftarrow topo - topo_{-1}$

Mul ; $topo \leftarrow topo * topo_{-1}$

Div ; $topo \leftarrow topo / topo_{-1}$

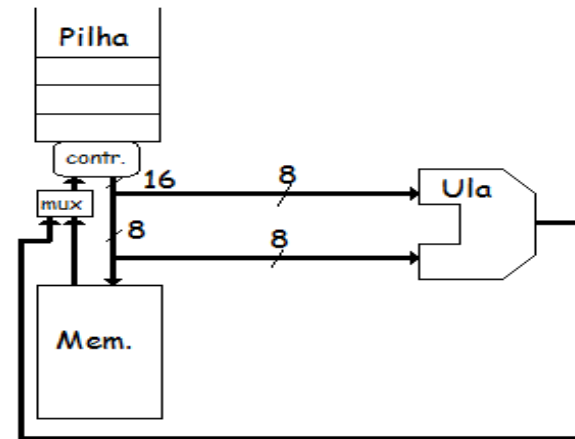


Arquitetura do Conjunto de Instruções



Pilha:

$$S = \frac{(A - C) * D}{(C - B)}$$



Pilha

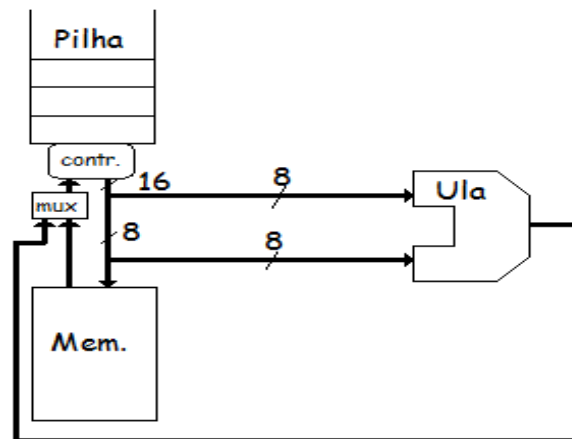
```
Pop  op; (op) ← topo
Push op; topo ← (op)
Add   ; topo ← topo + topo-1
Sub   ; topo ← topo - topo-1
Mul   ; topo ← topo * topo-1
Div   ; topo ← topo / topo-1
```

Arquitetura do Conjunto de Instruções



Pilha:

$$S = \frac{(A - C) * D}{(C - 5)}$$



Pilha

```
Pop  op; (op) ← topo
Push op; topo ← (op)
Add   ; topo ← topo + topo-1
Sub   ; topo ← topo - topo-1
Mul   ; topo ← topo * topo-1
Div   ; topo ← topo / topo-1
```

Acumulador (work register)

Movmw op; $w \leftarrow (op)$

Movwm op; $(op) \leftarrow w$

Add op; $w \leftarrow w + (op)$

Sub op; $w \leftarrow w - (op)$

Mul op; $w \leftarrow w * (op)$

Div op; $w \leftarrow w / (op)$

Acumulador (work register)

Movmw $op; w \leftarrow (op)$

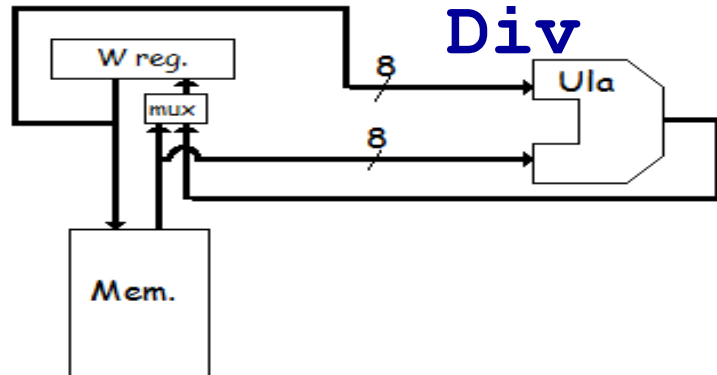
Movwm $op; (op) \leftarrow w$

Add $op; w \leftarrow w + (op)$

Sub $op; w \leftarrow w - (op)$

Mul $op; w \leftarrow w * (op)$

Div $op; w \leftarrow w / (op)$



Acumulador (work register)

$$S = \frac{(A - C) * D}{(C - B)}$$

Movmw op; $w \leftarrow (op)$

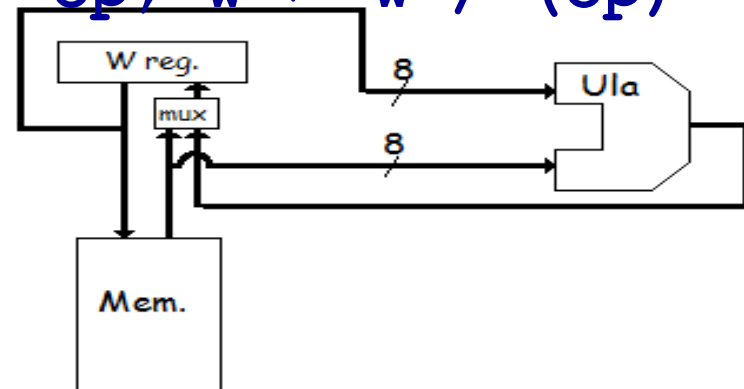
Movwm op; $(op) \leftarrow w$

Add op; $w \leftarrow w + (op)$

Sub op; $w \leftarrow w - (op)$

Mul op; $w \leftarrow w * (op)$

Div op; $w \leftarrow w / (op)$



Arquitetura do Conjunto de Instruções

Acumulador (work register)

$$S = \frac{(A - C) * D}{(C - 5)}$$

Movmw op; $w \leftarrow (op)$

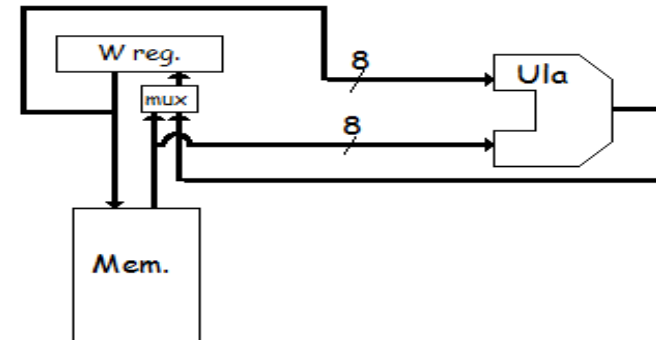
Movwm op; $(op) \leftarrow w$

Add op; $w \leftarrow w + (op)$

Sub op; $w \leftarrow w - (op)$

Mul op; $w \leftarrow w * (op)$

Div op; $w \leftarrow w / (op)$



Load-Store

Load op, R ; $R \leftarrow (op)$

Store op, R ; $(op) \leftarrow R$

Add rd,rs1,rs2 ; $rd \leftarrow rs1 + rs2$

Sub rd,rs1,rs2 ; $rd \leftarrow rs1 - rs2$

Mul rd,rs1,rs2 ; $rd \leftarrow rs1 * rs2$

Div rd,rs1,rs2 ; $rd \leftarrow rs1 / rs2$

Arquitetura do Conjunto de Instruções



Load-Store

Load op, R ; $R \leftarrow (op)$

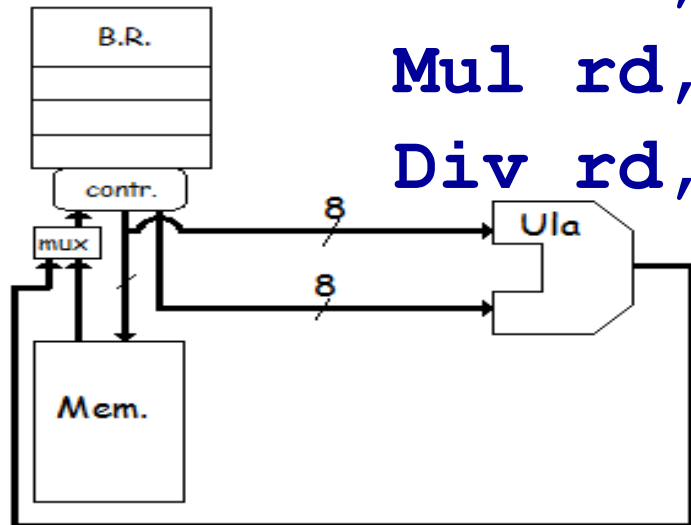
Store op, R ; $(op) \leftarrow R$

Add rd,rs1,rs2 ; $rd \leftarrow rs1 + rs2$

Sub rd,rs1,rs2 ; $rd \leftarrow rs1 - rs2$

Mul rd,rs1,rs2 ; $rd \leftarrow rs1 * rs2$

Div rd,rs1,rs2 ; $rd \leftarrow rs1 / rs2$



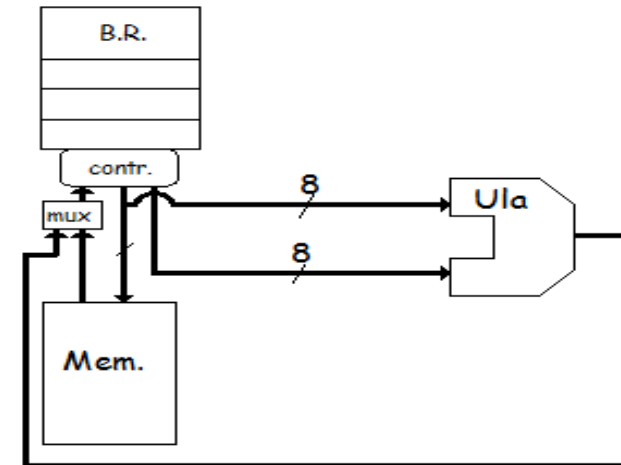
Arquitetura do Conjunto de Instruções



Load-Store

$$S = \frac{(A - C) * D}{(C - B)}$$

Load op, R ; R ← (op)
Store op, R ; (op) ← R
Add rd,rs1,rs2 ; rd ← rs1 + rs2
Sub rd,rs1,rs2 ; rd ← rs1 - rs2
Mul rd,rs1,rs2 ; rd ← rs1 * rs2
Div rd,rs1,rs2 ; rd ← rs1 / rs2



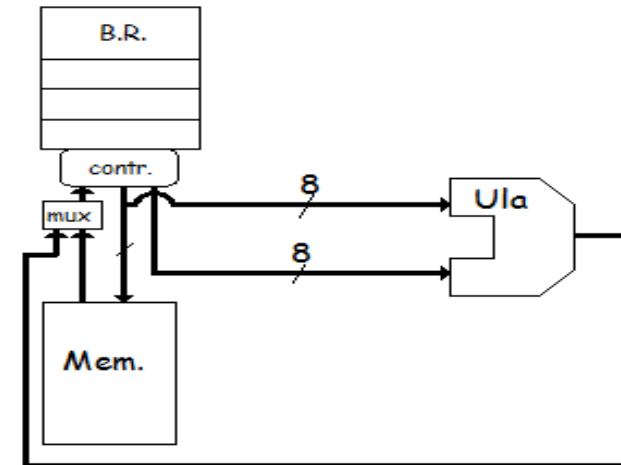
Arquitetura do Conjunto de Instruções



Load-Store

$$S = \frac{(A - C) * D}{(C - 5)}$$

Load op, R ; R ← (op)
Store op, R ; (op) ← R
Add rd,rs1,rs2 ; rd ← rs1 + rs2
Sub rd,rs1,rs2 ; rd ← rs1 - rs2
Mul rd,rs1,rs2 ; rd ← rs1 * rs2
Div rd,rs1,rs2 ; rd ← rs1 / rs2



Arquitetura do Conjunto de Instruções



Exercício:

1) Considerando a equação:

$$S = \left(A^2 - \frac{(C + B)}{A} \right) * \frac{(D - B)}{2^C}$$

- a) Defina um conjunto de instruções (ISA) com armazenamento interno tipo pilha para implementar a equação
- b) Apresente o programa assembly utilizando o ISA definido acima para a equação

Arquitetura do Conjunto de Instruções



Exercício:

1) Considerando a equação:

$$S = \left(A^2 - \frac{(C + B)}{A} \right) * \frac{(D - B)}{2^C}$$

- a) Defina um conjunto de instruções (ISA) com armazenamento interno tipo acumulador para implementar a equação
- b) Apresente o programa assembly utilizando o ISA definido acima para a equação

Arquitetura do Conjunto de Instruções



► Tipos de Ordenação de dados

Refere-se a ordem que os dados são armazenados na memória

Imagine o valor hexadecimal 0x12345678 a ser armazenado a partir do endereço 0x120

Endereço	Big Endian	Little Endian
120	12	78
121	34	56
122	56	34
123	78	12

Little Endian: Intel; Risc-V

Big Endian: Motorola 68XX

Bi Endian: PowerPC; ARM (geralmente Little)

Arquitetura do Conjunto de Instruções



► Tipos de Ordenação de dados

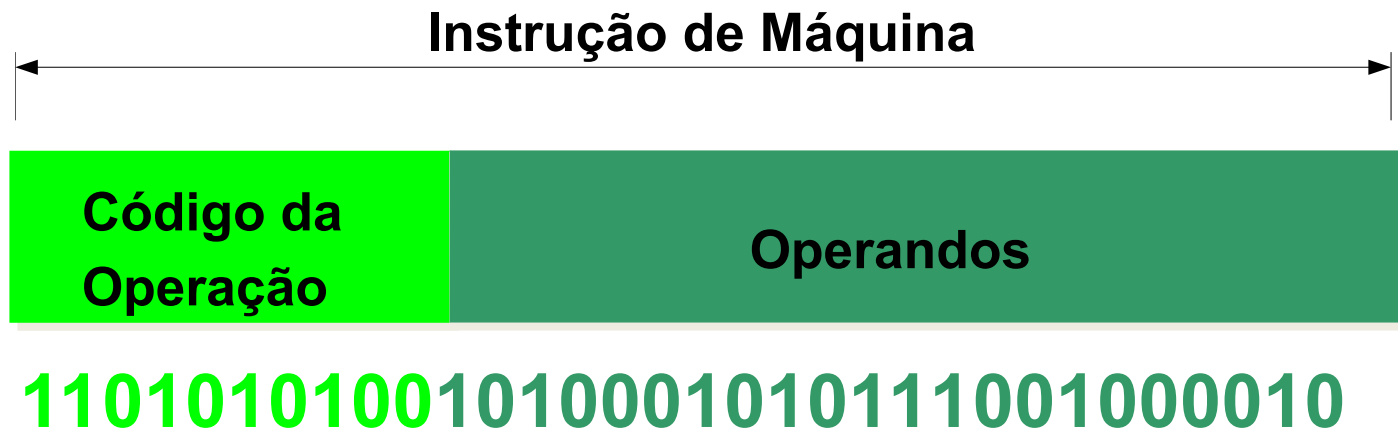
Refere-se a ordem que os dados são armazenados na memória

Endereço	Big Endian	Little Endian
120	12	78
121	34	56
122	56	34
123	78	12

Little Endian: O endereço de um dado (0x120) é o endereço do byte menos significativo (0x78)

Big Endian: O endereço de um dado (0x120) é o endereço do byte mais significativo (0x12)

Arquitetura do Conjunto de Instruções



- A **instrução de máquina** é um conjunto de bits contendo dois campos:
 - Código da Operação (opcode): o que a instrução faz
Exemplos: add; lw; jump; beq
 - Operandos (operand): onde estão os dados a serem manipulados

Arquitetura do Conjunto de Instruções



► O campo operando pode variar quanto a:

Quantidade de operandos:

- 3 operandos - SUB R1,R2,R3; $R1 \leftarrow R2 - R3$
- 2 operandos - MOV R1,R2; $R1 \leftarrow R2$
- 1 operando - ADD X; $W \leftarrow W + (X)$
- 0 operandos NOP

Modos de Endereçamento: como interpretar o campo operando no que diz respeito a onde se encontra o dado utilizado pela instrução

► **Modos de Endereçamento:** como interpretar o campo operando no que diz respeito a onde se encontra o dado utilizado pela instrução

- Imediato
- Direto
- Indireto
- Registrador Direto
- Registrador Indireto
- Base + Deslocamento

...

Arquitetura do Conjunto de Instruções



► **Modo Imediato:** o dado a ser manipulado está indicado no próprio campo operando da instrução;

Utilizadas na inicialização de variáveis e ponteiros; operações com constantes e desvios;

- **Vantagem:** poucos acessos a memória
- **Desvantagem:** limitação do campo operando restringe o valor máximo manipulado

Exemplo:

ADDI R1, R2, 3; $R1 \leftarrow R2 + 3$

Arquitetura do Conjunto de Instruções

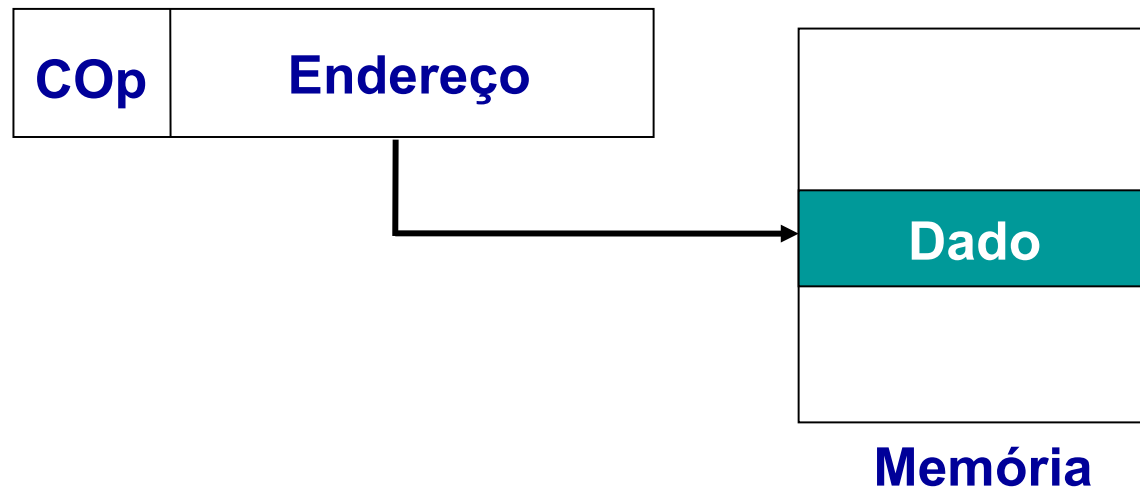


► Modo Direto

- O valor contido no campo operando indica o endereço de memória onde se localiza o dado a ser manipulado
- Um dos formatos naturais de implementar as variáveis do programa. Cada variável representa um endereço de memória

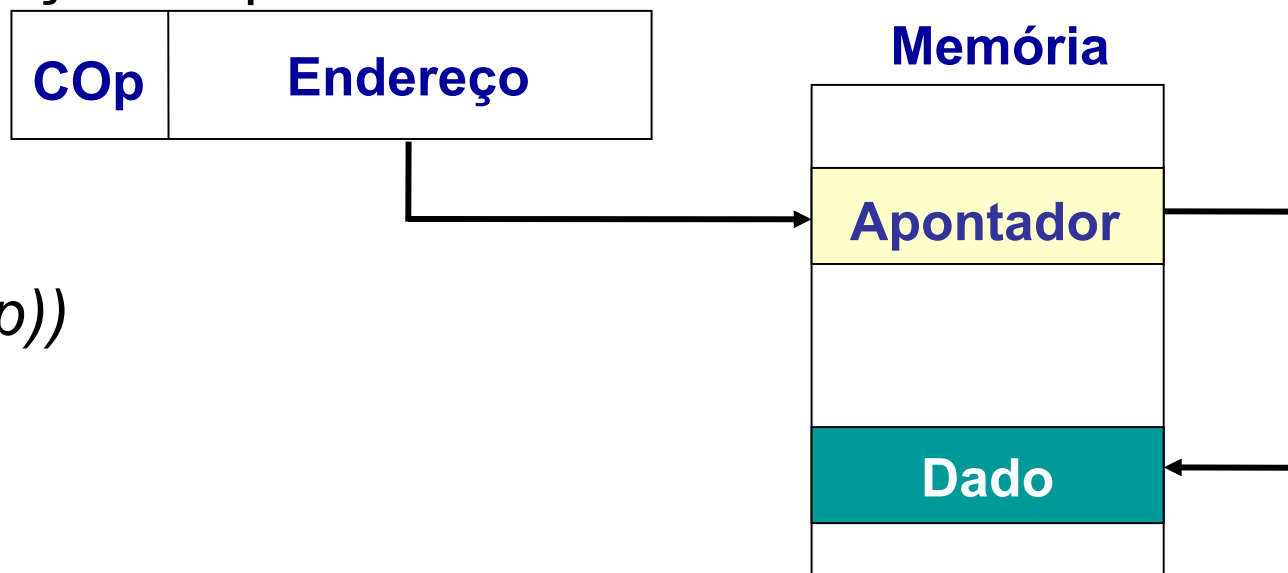
Exemplo:

LDA X; $W \leftarrow (X)$



► Modo Indireto

- O campo operando representa uma célula de memória, entretanto o valor contido neste endereço não é o dado e sim o endereço onde o dado se encontra
- Utilizado na implementação de ponteiros



Exemplo:

LDAIn Op; $W \leftarrow ((Op))$

LDAin ptr;

► Registrador Direto

- O campo operando representa o número do registrador onde o dado se encontra
- **Vantagens:**
 - acesso aos registradores é mais rápida que o acesso a memória
 - número menor de bits para endereçar registradores
- Principal modo de endereçamento nas arquiteturas RISC

Exemplo:

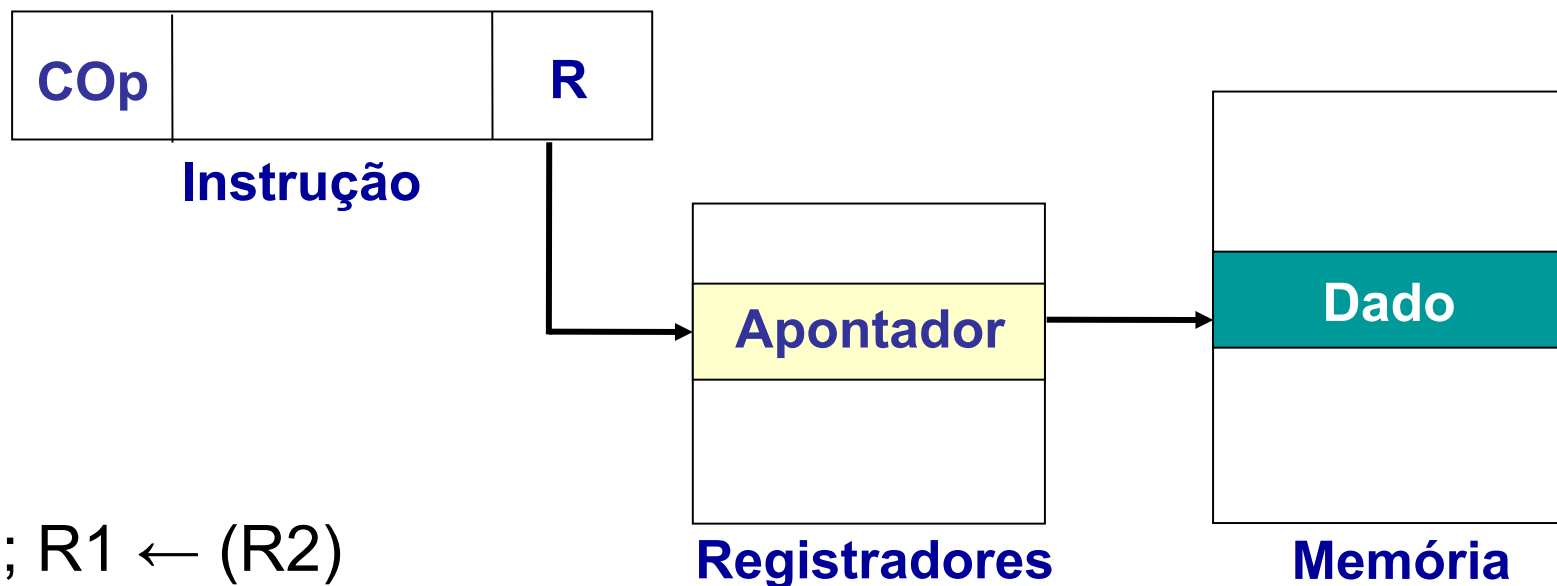
ADD R1, R2, R3; $R1 \leftarrow R2 + R3$

► Registrador Indireto

- O operando (registrador) aponta para o endereço de memória onde o dado se encontra;
- Pode ser implementado de várias maneiras; os modos a seguir são casos especiais deste modo:
 - Indireto
 - Base+deslocamento

► Registrador Indireto

- Indireto: operando é um registrador cujo conteúdo aponta para um endereço de memória



- Exemplo:

LDAr R1,R2; $R1 \leftarrow (R2)$

► Modo por Base + Deslocamento

- Neste modo o registrador base aponta para o início de um bloco e o deslocamento informa qual é o deslocamento dentro daquele bloco
- O endereço onde o dado se encontra é obtido:
 - a partir da soma entre o valor contido no registrador base e no deslocamento

Exemplo:

– $\text{LW R1, 4(R2) ; R1} \leftarrow (\text{R2} + 4)$

Arquitetura do Conjunto de Instruções

