

# Introdução à Arquitetura de Computadores

Aula16

## Assembly 1: Instruções do $\mu$ P MIPS

### Linguagem Máquina

- Introdução

### Codificação de Instruções

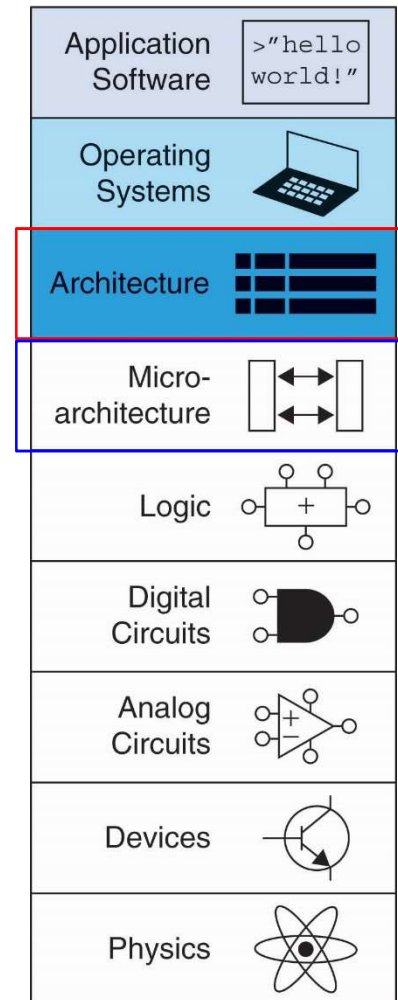
- Tipo-R, tipo-I e tipo-J
- Exemplos: Aritméticas (**add**) e de *Load/Store* (**lw**)

### Programa em Memória

- Inicialização e Execução

### Descodificação de Instruções

- Exemplos: tipo-R (**sub**) e tipo-I (**addi**)



# 1 - Linguagem Máquina - Introdução (1)

## Instrução

- Uma instrução corresponde a uma **única operação** que o processador pode executar, de entre as múltiplas definidas pelo respectivo *instruction set*.

## Linguagem Máquina (LM)

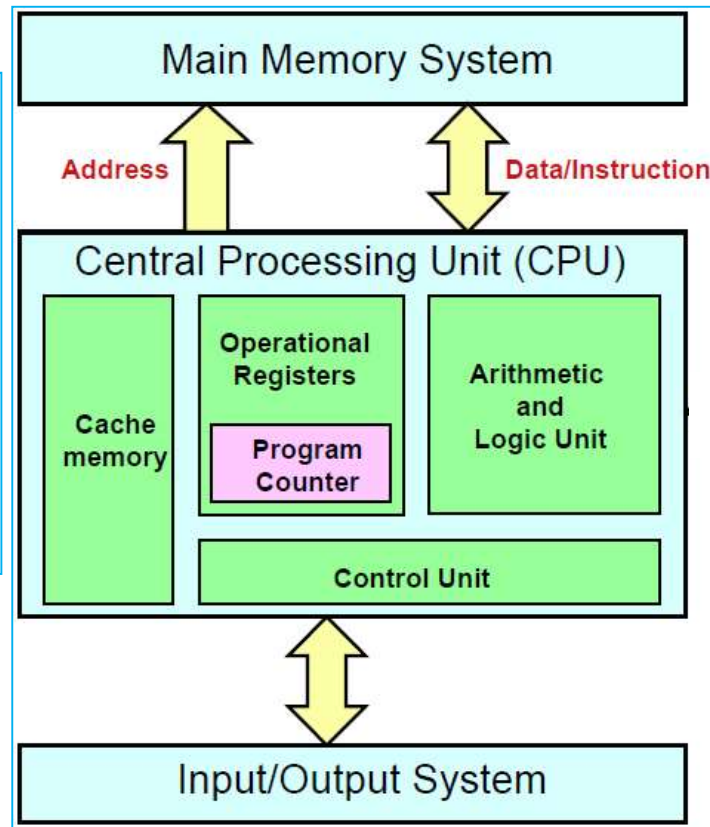
- Em LM, as **instruções** são representadas **em binário**, por *palavras* cujo **comprimento**, nas arquiteturas RISC, é normalmente **fixo** e igual a 32-bits.

# 1 - Linguagem Máquina - Introdução (2)

## Tipo de operações numa Instrução

Um computador possui Instruções para **quatro** tipo de operações:

- **Transferência de Dados** entre a Memória e os Registos do CPU (ex: *lw* e *sw*).
- **Execução de operações Aritméticas ou Lógicas** sobre os Dados (na ALU) (ex: *add* e *xor*)



- **Controlo do Fluxo de Execução** do programa (ex: *bne* e *j*)
- **Entrada/Saída de Dados** (opcional e ausente no MIPS)

As **Instruções são comandos** para: transferência de dados ou execução de operações aritméticas/lógicas ou ainda para controlo do fluxo de execução do programa.

# 1 - Linguagem Máquina - Introdução (3)

---

## Instruções na forma de números binários

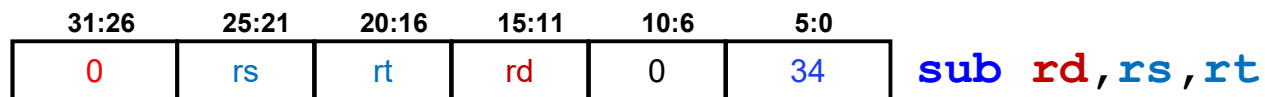
- Comprimento de palavra de 32-bits ( $\mu$ P de 32-bits):
  - Nos CPUs RISC, assume-se, que tanto os Dados como os Registos do CPU e ainda as Instruções possuem o **mesmo** comprimento.
- Como são codificadas as instruções?
  - Uma vez que a instrução possui um comprimento de 32-bits, esta é dividida em grupos-de-bits (*bitfields*);
  - Cada um destes grupos **diz algo** sobre a instrução:  
Exs de *bitfields*: código de operação, operandos, endereços.

Objetivo: Compreender o funcionamento do *Datapath* e da Unidade de Controlo do CPU.

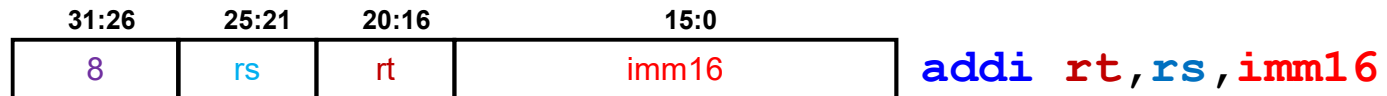
# 1 - Linguagem Máquina - As instruções do $\mu P$ MIPS

## Codificação binária das instruções

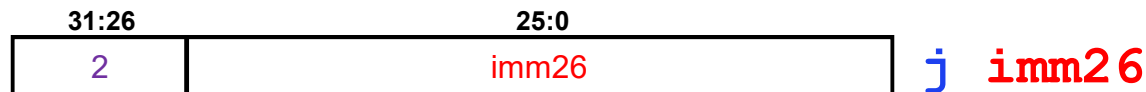
- Todas as instruções têm 32-bits!
- Existem três formatos de instrução:
  - **tipo-R:** dois operandos contidos em registros



- **tipo-I:** um dos operandos é uma constante



- **tipo-J:** o único operando é um endereço



# 1 - Instruções do tipo-R(*egister*) (1) - BitFields

---



3 Registos:

- **rs, rt:** 2 registos fonte (**rs**=primeiro e **rt**=segundo) ou operandos
- **rd:** 1 registo destino ou resultado

Outros campos:

- **op:** *opcode* ou *código de operação* (0 para instruções do tipo-R)
- **funct:** *função*, a qual juntamente com o *opcode*, especifica a operação a ser executada;
- **shamt:** o *shift amount* - é uma constante usada *só nas instruções de shift* (i.e., deslocamento de bits à esquerda ou à direita), nas restantes instruções possui o valor 0.

Tipo-R: Instruções Aritméticas e Lógicas com 3-registos

# 1 - tipo-R (2) - Assembly vs Máquina (1): Conversão

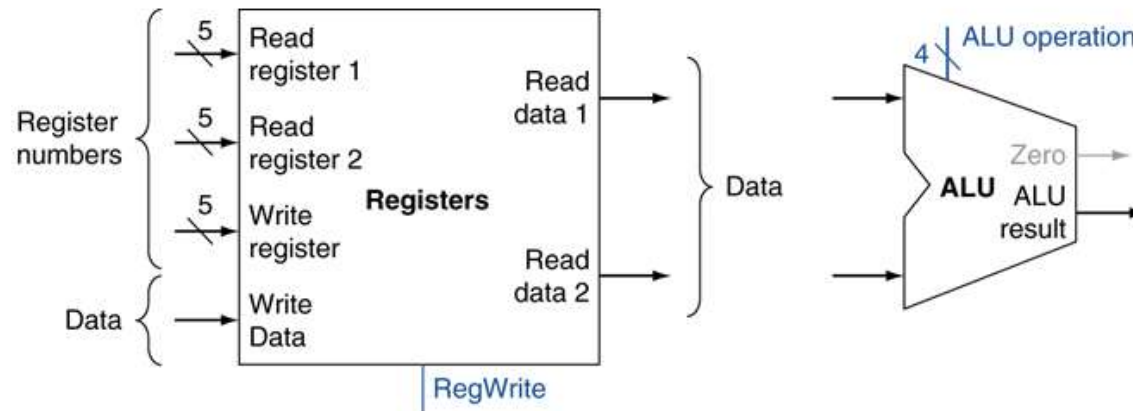
## Código Assembly

```
add $s0, $s1, $s2  
sub $t0, $t3, $t5
```

## Código Máquina

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
0	11	13	8	0	34
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Começamos com o tipo de instruções mais simples (i.e., aquelas cuja **execução** envolve só a interação entre o Banco de Registos e a Unidade Aritmética e Lógica (ou ALU).



## 1 - tipo-R (2) - Asm vs Máq (2): opcode & funct

### Código Assembly

```
add $s0, $s1, $s2  
sub $t0, $t3, $t5
```

### Código Máquina

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
0	11	13	8	0	34
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

As instruções do tipo-R têm a seguinte particularidade:

O código de **op**eração é **zero**, sendo a **fun**ção a ser executada determinada pelo campo **funct**.

<b>op</b> = 0	<b>rs</b>	<b>rt</b>	<b>rd</b>	<b>shamt</b>	<b>funct</b>
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Em geral, a conversão das instruções *Assembly* para Código Máquina é feita através da **consulta de tabelas**. Neste caso, precisamos duma tabela para os registos **rs**, **rt** e **rd** e outra tabela para o código de função **funct**.



# 1 - tipo-R (2) - Asm vs Máq (3): tabelas de registos + funct

## Código Assembly

```
add $s0, $s1, $s2
sub $t0, $t3, $t5
```

## Código Máquina

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
0	11	13	8	0	34
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

1. O número dos registos é dado pela Tabela 6.1 (para todas as instruções).

Name	Number	Use
\$t0-\$t7	8-15	temporary variables
\$s0-\$s7	16-23	saved variables
\$t8-\$t9	24-25	temporary variables

Tabela 6.1 - (pg. 300)

```
$s0 = 16, $s1 = 17, $s2 = 18
$t0 = 8,  $t3 = 11, $t5 = 13
```

2. O código de função é dado pela Tabela B.2. Para as instruções add = 32 e sub = 34.

Funct	Name
100000 (32)	add rd, rs, rt
100001 (33)	addu rd, rs, rt
100010 (34)	sub rd, rs, rt

Tabela B.2 - (pg. 622)

3. Estas instruções do tipo-R, add e sub, possuem um shamt igual a zero.

# 1 - tipo-R (2) - Tabela de Registos

Table 6.1 MIPS register set

Name	Number	Use
\$0	0	the constant value 0
\$at	1	assembler temporary
\$v0-\$v1	2-3	function return value
\$a0-\$a3	4-7	function arguments
\$t0-\$t7	8-15	temporary variables
\$s0-\$s7	16-23	saved variables
\$t8-\$t9	24-25	temporary variables
\$k0-\$k1	26-27	operating system (OS) temporaries
\$gp	28	global pointer
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	function return address

Tabela 6.1 - Nome, número e respectivo uso, de cada registo.

# 1 - tipo-R (2) - Tabela de Código de Função (*funct*)

Table B.2 R-type instructions, sorted by funct

Funct	Name	Description
000000 (0)	sll rd, rt, shamt	shift left logical
000010 (2)	srl rd, rt, shamt	shift right logical
000011 (3)	sra rd, rt, shamt	shift right arithmetic
000100 (4)	sllv rd, rt, rs	shift left logical variable
000110 (6)	srlv rd, rt, rs	shift right logical variable
000111 (7)	srav rd, rt, rs	shift right arithmetic variable
001000 (8)	jr rs	jump register
001001 (9)	jair rs	jump and link register
001100 (12)	syscall	system call
001101 (13)	break	break
010000 (16)	mfhi rd	move from hi
010001 (17)	mthi rs	move to hi
010010 (18)	mflo rd	move from lo
010011 (19)	mtlo rs	move to lo
011000 (24)	mult rs, rt	multiply
011001 (25)	multu rs, rt	multiply unsigned
011010 (26)	div rs, rt	divide
011011 (27)	divu rs, rt	divide unsigned

Table B.2 R-type instructions, sorted by funct field-

Funct	Name	Description
100000 (32)	add rd, rs, rt	add
100001 (33)	addu rd, rs, rt	add unsigned
100010 (34)	sub rd, rs, rt	subtract
100011 (35)	subu rd, rs, rt	subtract unsigned
100100 (36)	and rd, rs, rt	and
100101 (37)	or rd, rs, rt	or
100110 (38)	xor rd, rs, rt	xor
100111 (39)	nor rd, rs, rt	nor
101010 (42)	slt rd, rs, rt	set less than
101011 (43)	sltu rd, rs, rt	set less than unsigned

Tabela B.2

Instruções do tipo-R ordenadas pelo campo *funct*.

Type-R Function Code: ADD, SUB

# 1 - tipo-R (2) - Asm vs Máq (4) - Em binário e hexadecimal

## Código Assembly

**add** \$s0, \$s1, \$s2  
**sub** \$t0, \$t3, \$t5

## Código Máquina

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
0	11	13	8	0	34
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Em **binário** (32-bits):

Em **hexadecimal** (8-hexas):

0 op	2 rs	3 rt	2 rd	8 shamt	2 funct	
000000	10001	10010	10000	00000	100000	(0x02328020)
000000	01011	01101	01000	00000	100010	(0x016D4022)
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
0	1	6	D	4	0	2

Ordem dos *bitfields*: Em Assembly: **add** rd, rs, rt  
Em Máquina: **<0>** rs, rt, rd, **<funct>**

## 1 - tipo-R (3) - Exercício Codificação (1) - add (1) - método

---

Qual é o código máquina da seguinte instrução *Assembly*?

**add**    \$t0, \$s4, \$s5

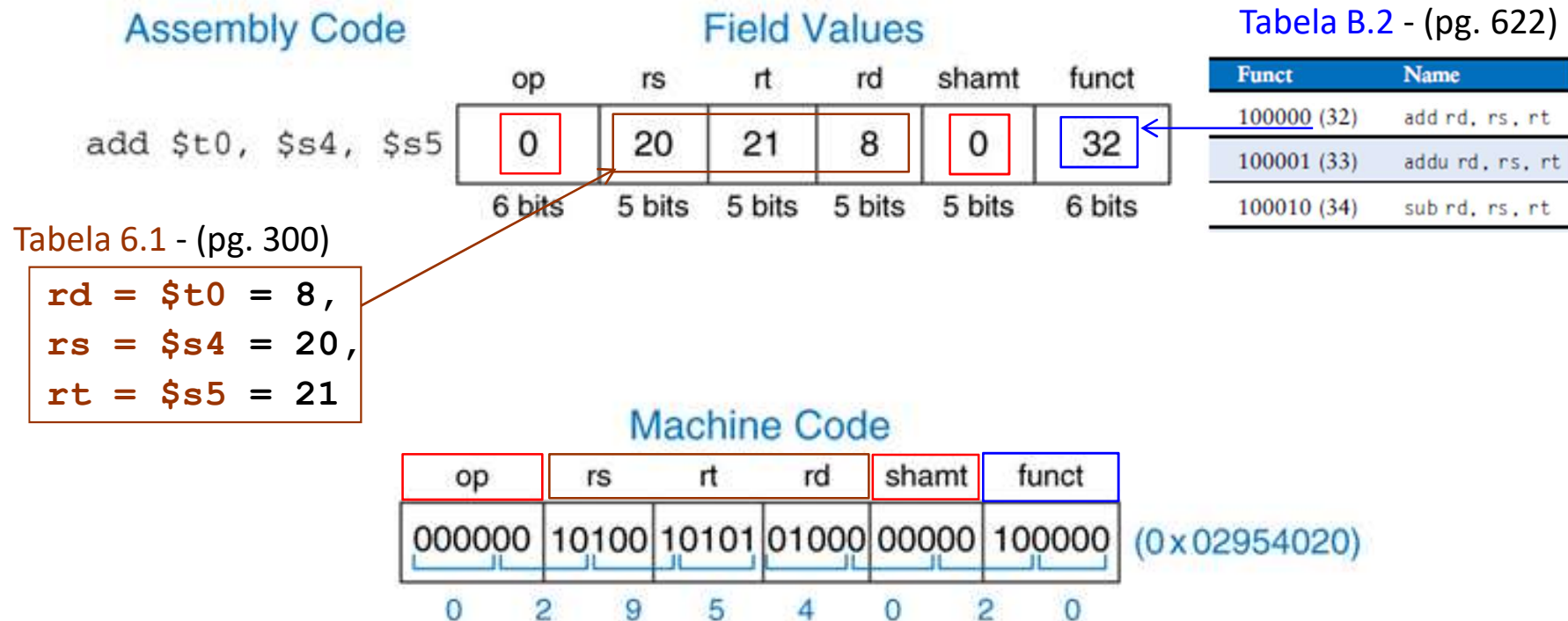
Método:

1. Sabemos que **add** é uma instrução do **tipo-R** (3-registos);
2. Da **Tabela 6.1** (pg. 300), tiramos:  
 $\$t0 = \mathbf{rd} = 8$ ,  $\$s4 = \mathbf{rs} = 20$  e  $\$s5 = \mathbf{rt} = 21$   
onde **rd**=registo destino; **rs** = registo **op1** e **rt** = registo **op2**;
3. Da **Tabela B.2** (pg. 622), o código de **função** de **add** é **32**;
4. Por ser do **tipo-R** o código de **operação** é **0** e, não sendo de *shift*, o **shamt** também é **0**.

## 1 - tipo-R (3) - Exercício Codificação (2) - add (2) - binário

Qual é o código máquina da seguinte instrução *Assembly*?

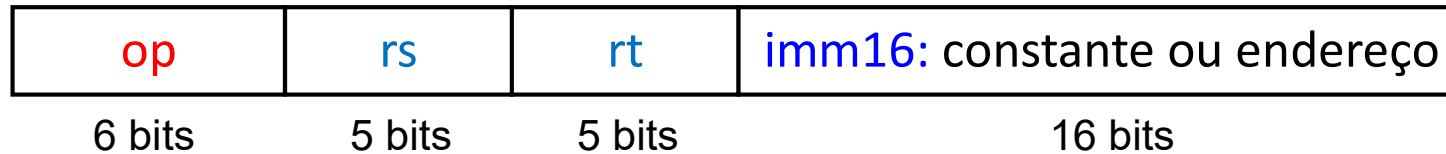
**add**    \$t0, \$s4, \$s5



Convem escrever em binário primeiro e só depois em hexadecimal.

## 2 - Instruções do tipo-I(*immediate*) (1) - BitFields

---



2 registos:

- **rs:** registo fonte (**addi**) ou endereço-base (eg, **lw**, **sw**)
- **rt:** registo destino para algumas (eg, **addi**, **lw**)
- **imm16:** valor imediato de 16-bits (em 2C, exceto nas lógicas)  
**constante:**  $-2^{15}$  a  $+2^{15} - 1$  (mas nas lógicas: 0 a  $2^{16} - 1$ )  
**endereço:** *offset* adicionado ao endereço-base em **rs**
- **op:** *opcode* (ou *código de operação*) é diferente de zero, e está presente em todas as instruções do tipo-I; a operação a ser executada é inteiramente determinada pelo *opcode* (só!).

tipo-I: Usadas em instruções Aritméticas/Lógicas Imediatas e de *Load/Store*



## 2 - tipo-I (2) - Exemplos: Aritmética\_imm, lw e sw

### Código Assembly

```
addi    $s0, $s1, 5
addi    $t0, $s3, -12
lw      $t2, 32($s0)
sw      $s1, 4($t1)
```

### Valor dos Campos

op	rs	rt	imm
8	17	16	5
8	19	8	-12
35	0	10	32
43	9	17	4
6 bits	5 bits	5 bits	16 bits

### Código Máquina

	op	rs	rt	imm	
addi	rt, rs, imm	001000	10001	10000	0000 0000 0000 0101 (0x22300005)
addi	rt, rs, imm	001000	10011	01000	1111 1111 1111 0100 (0x2268FFF4)
lw	rt, imm(rs)	100011	00000	01010	0000 0000 0010 0000 (0x8C0A0020)
sw	rt, imm(rs)	101011	01001	10001	0000 0000 0000 0100 (0xAD310004)
		6 bits	5 bits	5 bits	16 bits

offset      endereço-base



## 2 - tipo-I (3) - Exercício Codificação (1) - lw (1) - método

---

Qual o código máquina da seguinte instrução *Assembly*?

**lw**     \$s3, -24(\$s4)

Método:

- 1. Sabemos que **lw** é uma instrução do tipo-I (2-registos);
- 2. Da Tabela B.1 (pg. 620), o código de operação para **lw** é 35;
- 3. Da Tabela 6.1 (pg. 300), tiramos \$s3=rt=19 e \$s4=rs=20, onde **rt**=registo destino; **rs** = registo (com o) endereço-base;
- 4. O valor imediato -24, representa o *offset* (16-bits em 2C) a adicionar ao endereço-base (**rs**) para gerar o endereço efetivo.

Esta instrução lê uma palavra de 32-bits (*word*) do endereço de memória dado por:  
"\$s4 - 24" e coloca-a no registo \$s3.

# 1 - tipo-I (4) - Tabela de Código de Operação (*opcode*)

Opcode	Name	Description	Opcode	Name	Description
000000 (0)	R-type	all R-type instructions	011100 (28)	mul rd, rs, rt (func = 2)	multiply (32-bit result)
000001 (1)	bltz rs, label / bgez rs, label (rt = 0/1)	branch less than zero/branch greater than or equal to zero	100000 (32)	lb rt, imm(rs)	load byte
000010 (2)	j label	jump	100001 (33)	lh rt, imm(rs)	load halfword
000011 (3)	jal label	jump and link	100011 (35)	lw rt, imm(rs)	load word
000100 (4)	beq rs, rt, label	branch if equal	100100 (36)	lbu rt, imm(rs)	load byte unsigned
000101 (5)	bne rs, rt, label	branch if not equal	100101 (37)	lhu rt, imm(rs)	load halfword unsigned
000110 (6)	blez rs, label	branch if less than or equal to zero	101000 (40)	sb rt, imm(rs)	store byte
000111 (7)	bgtz rs, label	branch if greater than zero	101001 (41)	sh rt, imm(rs)	store halfword
001000 (8)	addi rt, rs, imm	add immediate	101011 (43)	sw rt, imm(rs)	store word
001001 (9)	addiu rt, rs, imm	add immediate unsigned	110001 (49)	lwc1 ft, imm(rs)	load word to FP coprocessor 1
001010 (10)	slti rt, rs, imm	set less than immediate	111001 (56)	swc1 ft, imm(rs)	store word to FP coprocessor 1
001011 (11)	sltiu rt, rs, imm	set less than immediate unsigned			
001100 (12)	andi rt, rs, imm	and immediate			
001101 (13)	ori rt, rs, imm	or immediate			
001110 (14)	xori rt, rs, imm	xor immediate			
001111 (15)	lui rt, imm	load upper immediate			
010000 (16)	mfc0 rt, rd / (rs = 0/4) mtc0 rt, rd	move from/to coprocessor 0			
010001 (17)	F-type	fop = 16/17: F-type instructions			
010001 (17)	bc1f label / (rt = 0/1) bc1t label	fop = 8: branch if fpcond is FALSE/TRUE			

Table B.1

Instructions sorted by *opcode* field.

**Tipo-I: e.g., ADDI, LW, SW**

**Tipo-J: J, JAL**

## 1 - tipo-I (3) - Exercício Codificação (2) - lw (2) - binário

Qual é o código máquina da seguinte instrução Assembly?

**lw**      **\$s3, -24(\$s4)**



Tabela B.1 - (pg. 620)

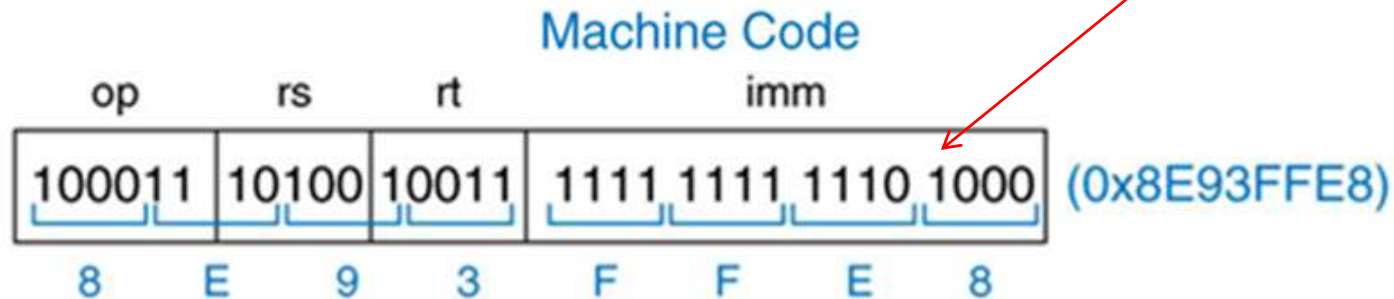
Tabela 6.1 - (pg. 300)

rs = \$s4 = 20  
rt = \$s3 = 19

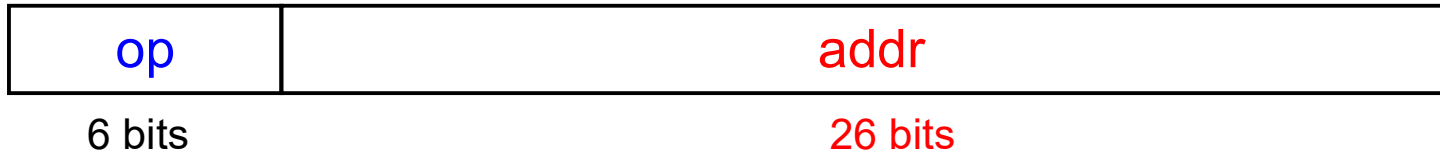
Valor **imediato** de 16-bits, em 2C :

$$24_{10} = 0000\ 0000\ 0001\ 1000_2$$

$$\boxed{-24_{10}} = 1111\ 1111\ 1110\ 1000_2 = \boxed{\text{FFE8}_{16}}$$



### 3 - Instruções do tipo-J(ump)



- 1 único operando:
  - **addr:** endereço com 26-bits
- Outros campos:
  - **op:** o código de *operação* da instrução **j**ump (op=2)

```
# MIPS assembly - j(ump)
addi    $s0, $0, 4      # $s0 = 4
addi    $s1, $0, 1      # $s1 = 1
→ j      target         # jump to target
sra     $s1, $s1, 2      # not executed
addi    $s1, $s1, 1      # not executed
target:
add     $s1, $s1, $s0    # $s1 = 1 + 4 = 5
```

próxima aula!

tipo-J - Usadas em instruções do tipo j (jump) e jal (jump and link).

## 4 - Formato das Instruções do $\mu$ P MIPS - Resumo

---

### Tipo-R

0	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- **3 registros:** operandos (**rs**, **rt**) e resultado (**rd**) ; **opcode=0**

---

- **2 registros:** operando (**rs**) e resultado/fonte (**rt**) ; operando (**imm**)

---

- um único operando (**addr**)

## 5 - Programa em Memória (1)

### Programa em memória?

- É um conjunto de **instruções e dados** para um CPU
- É a diferença entre duas aplicações
- **Vantagens do programa**
  - Não é necessário refazer ligações elétricas;
  - Basta armazenar um novo programa na memória, para alterar a funcionalidade da máquina.
- **Como é feita a Execução do programa?**
  - O CPU lê as instruções da memória (sequencial/);
  - Em seguida, descodifica cada instrução e executa a operação que lhe está associada.

O programa cria uma máquina cujo modo de funcionamento é (re)programável!

## 5 - Programa em Memória (2) - Inicialização

### Código Assembly

```
lw    $t2, 32($0)
add   $s0, $s1, $s2
addi  $t0, $s3, -12
sub   $t0, $t3, $t5
```

### Código Máquina

```
0x8C0A0020
0x02328020
0x2268FFF4
0x016D4022
```

### Assembly -> Código Máquina:

Código *Assembly* e a respectiva tradução em código máquina.

### Programa em Memória

Endereço	Instruções
⋮	⋮
0040000C	0 1 6 D 4 0 2 2
00400008	2 2 6 8 F F F 4
00400004	0 2 3 2 8 0 2 0
00400000	8 C 0 A 0 0 2 0
⋮	⋮

Main Memory

PC

### Programa em Memória:

O código máquina carregado na Memória (de Instruções).

### Program Counter (PC):

Registo que aponta para a próxima instrução a ser executada (pelo CPU).

## 5 - Programa em Memória (3) - Início de Execução

### Código Assembly

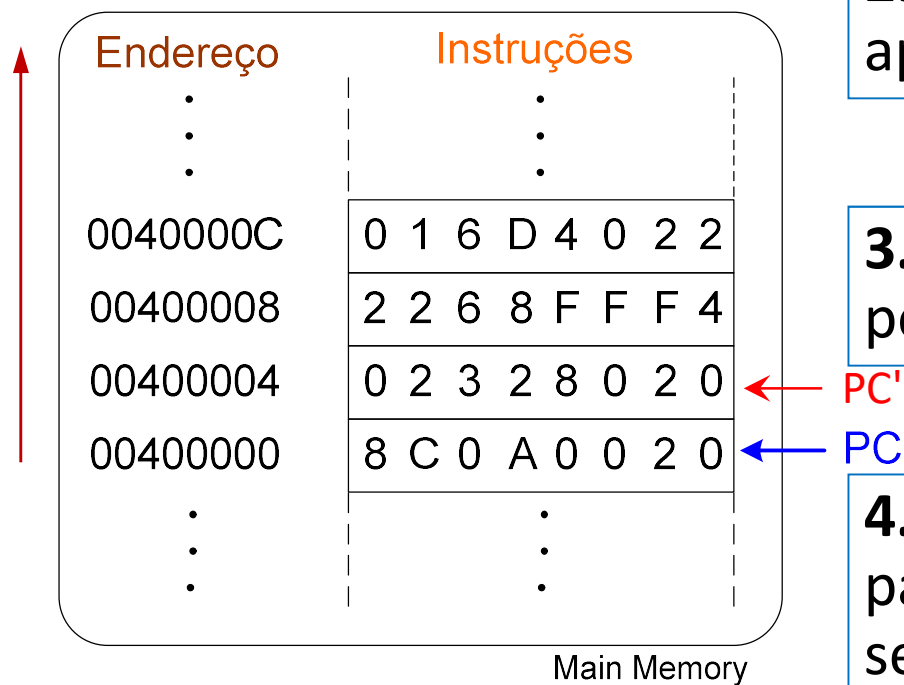
```
lw    $t2, 32($0)
add   $s0, $s1, $s2
addi  $t0, $s3, -12
sub   $t0, $t3, $t5
```

### Código Máquina

```
0x8C0A0020
0x02328020
0x2268FFF4
0x016D4022
```

1. O código *Assembly* é traduzido em código Máquina e carregado em memória.

### Programa em Memória



Endereço	Instruções
⋮	⋮
0040000C	0 1 6 D 4 0 2 2
00400008	2 2 6 8 F F F 4
00400004	0 2 3 2 8 0 2 0
00400000	8 C 0 A 0 0 2 0
⋮	⋮

Main Memory

2. O registo **PC** é inicializado para apontar para a 1ª instrução.

3. O CPU lê a instrução apontada pelo PC, descodifica-a e executa-a.

4. O valor do **PC** é incrementado para apontar para a instrução seguinte, i.e.,  $PC' = PC + 4$ .



## 6 - Descodificação\* de Instruções (1) - método

---

- Começamos com o **opcode** (6-bits mais signif.)
  - Se for igual a zero (0)
    - É uma instrução do **tipo-R**
    - Os 6 bits da **função** determinam a operação
  - Caso contrário
    - É uma instrução do **tipo-I** ou do **tipo-J**
    - O **opcode**, por si só, determina a operação

### Exemplo:

Converter o código máquina seguinte em instruções *Assembly*:

**0x02F34022**

**0x2237FFF1**

\*Interpretação ou *DisAssembly*. Processo inverso à codificação de ASM em Cód. Máquina.

## 6 - Descodificação (2) - tipo-R (1) - *opcode*

As instruções *Assembly*:

**0x02F34022** ←

**0x2237FFF1**

1. Consideremos a *primeira* instrução;  
Verificamos que os **6bits** mais significativos (MS) são iguais a zero; Logo é uma instrução do **tipo-R**.

1. O *opcode* (6-bits MS)

Machine Code

	op	rs	rt	rd	shamt	funct		
(0x02F34022)	000000	10111	10011	01000	00000	100010		
	0	2	F	3	4	0	2	2

Field Values

op	rs	rt	rd	shamt	funct
0	23	19	8	0	34

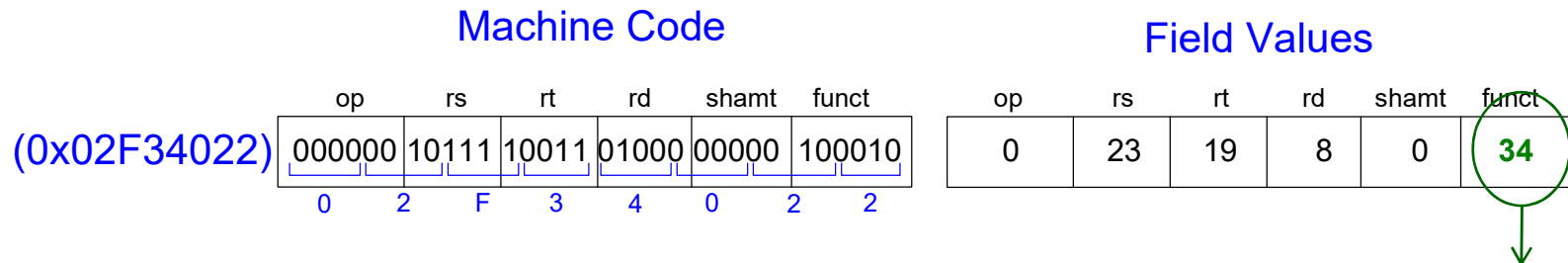
## 6 - Descodificação (2) - tipo-R (2) - funct

As instruções *Assembly*:

**0x02F34022** ←  
**0x2237FFF1**

1. Consideremos a primeira instrução;  
Verificamos que os 6bits mais significativos (MS) são iguais a zero; Logo é uma instrução do tipo-R.

1. O *opcode* (6-bits MS)



2. O *funct* (6-bits mS)

**Tabela B.2 - tipo-R (pg. 621/2)**

Sendo do tipo-R, analisamos em seguida o campo *funct* (6bits mS) para determinar qual a operação a ser executada.

A tabela B.2 dá-nos que a operação 34 corresponde a *sub*.



Funct	Name
100000 (32)	add rd, rs, rt
100001 (33)	addu rd, rs, rt
100010 (34)	sub rd, rs, rt

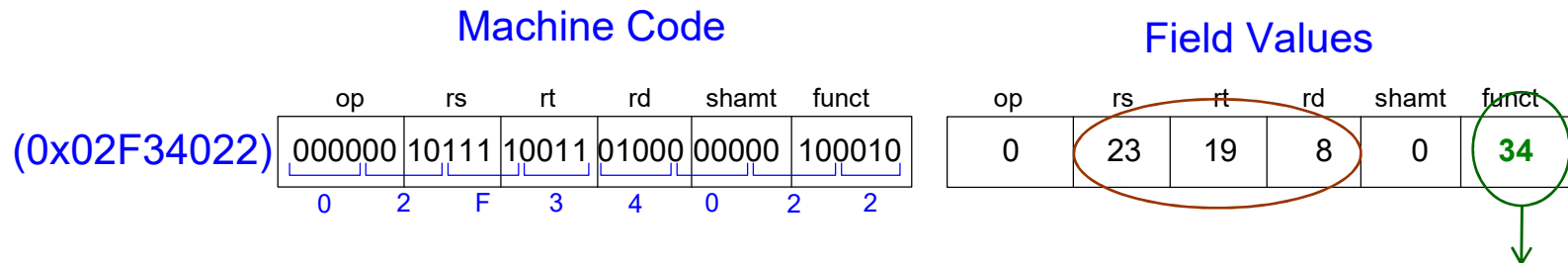
## 6 - Descodificação (2) - tipo-R (3) - registos

As instruções *Assembly*:

**0x02F34022** ←  
**0x2237FFF1**

1. Consideremos a **primeira** instrução;  
Verificamos que os **6bits** mais significativos (MS) são iguais a zero; Logo é uma instrução do **tipo-R**.

1. O **opcode** (6-bits **MS**)



2. O **funct** (6-bits **mS**)

**Tabela B.2 - tipo-R (pg. 621/2)**

Sendo do tipo-R, analisamos em seguida o campo **funct** (6bits **mS**) para determinar qual a operação a ser executada.

A **tabela B.2** dá-nos que a operação **34** corresponde a **sub**. →

Funct	Name
100000 (32)	add rd, rs, rt
100001 (33)	addu rd, rs, rt
100010 (34)	sub rd, rs, rt

3. O valores dos registos **rs**, **rt** e **rd** é-nos dado pela **Tabela 6.1**:

**rs** = 23 -> **\$s7**; **rt** = 19 -> **\$s3**; **rd** = 8 -> **\$t0**

→ **sub \$t0, \$s7, \$s3**

## 6 - Descodificação (3) - tipo-I (1) - *opcode*

As instruções *Assembly*:

**0x02F34022**

**0x2237FFF1** ←

2. Consideremos a *segunda* instrução.

Os **6bits** mais *significativos* são diferentes de zero. Logo **não** é uma instrução do *tipo-R*.

1. O *opcode* (6-bits **MS**)

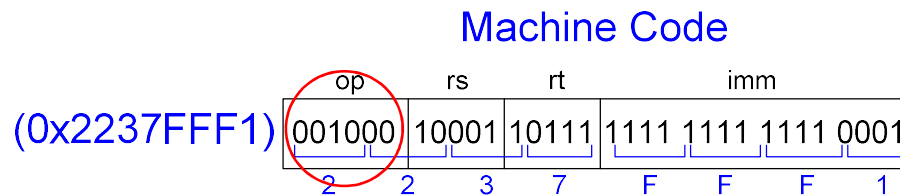


Tabela B.1 - (pg. 620)

Field Values

op	rs	rt	imm
8	17	23	-15

Consultando a **Tabela B.1**, verificamos que a instrução com o *opcode* igual a 8 é **addi**.



Opcode	Name	Description
001000 (8)	addi rt, rs, imm	add immediate

## 6 - Descodificação (3) - tipo-I (2) - registros

As instruções *Assembly*:

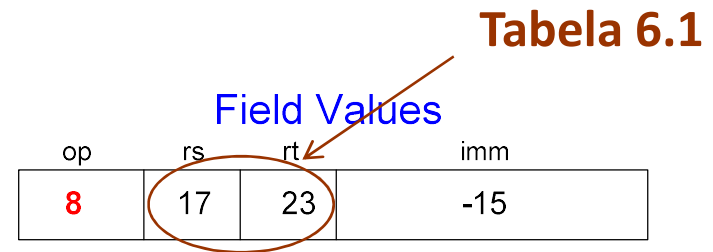
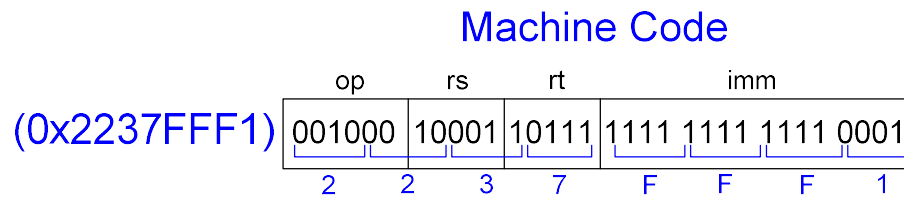
**0x02F34022**

**0x2237FFF1** ←

2. Consideremos a **segunda** instrução.

Os **6bits mais significativos** são diferentes de zero. Logo **não** é uma instrução do **tipo-R**.

1. O **opcode** (6-bits **MS**)



Consultando a **Tabela B.1**, verificamos que a instrução com o **opcode** igual a 8 é **addi**.



Opcode	Name	Description
001000 (8)	addi rt, rs, imm	add immediate

2. O valores dos registros **rs**, **rt** é : **rs** = 17 -> **\$s1**; **rt** = 23 -> **\$s7**

## 6 - Descodificação (3) - tipo-I (3) - imm16

As instruções *Assembly*:

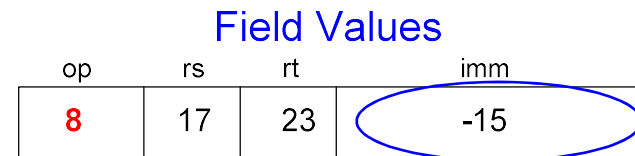
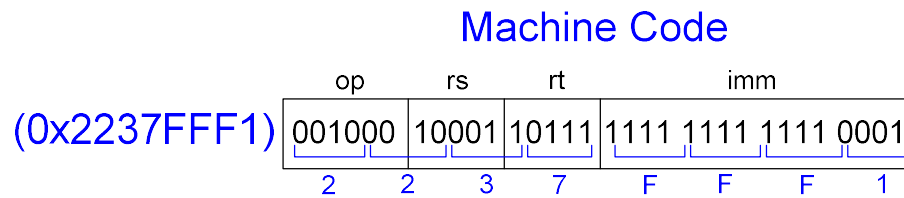
**0x02F34022**

**0x2237FFF1** ←

2. Consideremos a **segunda** instrução.

Os **6bits mais significativos** são diferentes de zero.  
Logo **não** é uma instrução do **tipo-R**.

1. O **opcode** (6-bits **MS**)



Consultando a **Tabela B.1**, verificamos que a instrução com o **opcode** igual a 8 é **addi**.

Opcode	Name	Description
001000 (8)	addi rt, rs, imm	add immediate

2. O valores dos registos **rs**, **rt** é : **rs** = 17 -> **\$s1**; **rt** = 23 -> **\$s7**

3. O valor **imediato** (16 bits menos signif.) é uma **constante** em **2C**:

**0xFFF1**<sub>16</sub> a que corresponde -0x000F<sub>16</sub> ou -15<sub>10</sub>.

**2C** = Two's Complement .

→ **addi \$s7,\$s1,-15**

## 6 - Descodificação (4) - tipo-R e tipo-I : Resumo

As instruções *Assembly*:

**0x02F34022**

**0x2237FFF1**

### tipo-R

**0x02F34022**

31:26	25:21	20:16	15:11	10:6	5:0	
0	rs	rt	rd	0	34	sub rd,rs,rt

### tipo-I

**0x2237FFF1**

31:26	25:21	20:16	15:0	
8	rs	rt	imm	addi rt,rs,imm



## ***Livro de Referência***

Digital Design and Computer Architecture, Second Edition

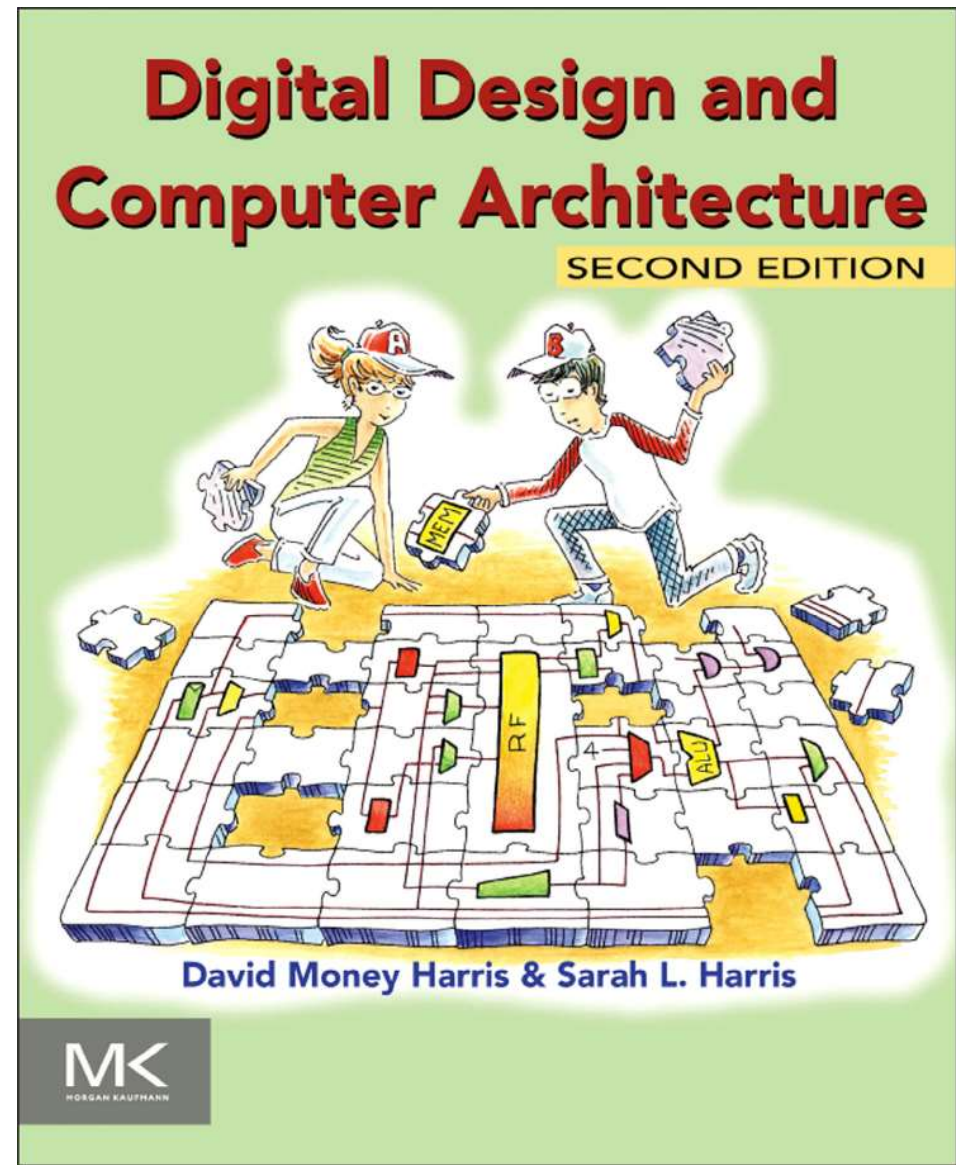
David Money Harris & Sarah L. Harris

ISBN: 978-0-12-394424-5

© 2013 Elsevier

Chapter 6 - Architecture  
(Assembly)

Chapter 7 - Microarchitecture  
(Datapath)



# Z - Programa das Teóricas

---

## Assembly do $\mu$ P MIPS

Aula	Descrição
ASM1	Codificação de instruções no MIPS: Instruções do tipo R, do tipo I e do tipo J. Programa em Memória. DesCodificação de instruções MIPS.
ASM2	Instruções do MIPS: Aritméticas, Lógicas e de <i>Shift</i> . *Multiplicação/Divisão de Inteiros. <i>Jumps e Branches</i> : “Saltos” condicionais e incondicionais. Estruturas de controlo de fluxo de execução: <i>if</i> , <i>if-else</i> ; ciclos <i>while</i> e <i>for</i>
ASM3	<i>Arrays</i> : Acesso a elementos de um <i>array</i> residente em memória. <i>Funções (Subrotinas)</i> : Procedimentos de invocação e de retorno. Convenções para: a passagem de parâmetros, a devolução de valores e a salvaguarda de registos.
ASM4	<i>Stack</i> : Conceito e regras básicas de utilização. Recursividade*. <i>Modos de endereçamento do MIPS</i> : Resumo.

# Z - Programa das Teóricas

---

## Assembly do $\mu$ P MIPS

Aula	Descrição
ASM5	<i>Mais Assembly:</i> Ponteiro: definição e propriedades Uso de arrays com ponteiros Índices versus ponteiros; Exemplos. Instruções signed/unsigned; Resumo.
ASM6	<i>Assembling &amp; Loading:</i> Fases de tradução dum Programa; Mapa de Memória Segmentos de Texto e de Dados O Assembler: Diretivas; Pseudo Instruções. O Código Executável (Linking e Loading)

## Z - Programa das Teóricas

---

### Datapath do $\mu$ P MIPS: Single-cycle

Aula	Descrição
22	<b>VI</b> - Organização interna do processador: Unidades operativas e de controlo. <i>Datapath</i> : Pressupostos para a construção de um <i>datapath</i> genérico para uma arquitectura tipo MIPS. Análise dos blocos constituintes necessários à execução de cada tipo de instruções básicas: tipo R; <i>load</i> e <i>store</i> ; salto condicional.
23	<i>Unidade de Controlo</i> Descodificador da ALU; Exemplo de ALU Principal Descodificador Principal Exercício: Execução da instrução <i>or</i> Instruções adicionais: <i>addi</i> e <i>j(ump)</i>
24	Resolução de <i>problemas</i> sobre uArquiteturas Single-cycle.

## Z - Programa das Teóricas

---

### *Datapath do $\mu$ P MIPS: Multicycle*

Aula	Descrição
25	<i>Multicycle</i> : Limitações das arquiteturas <i>single-cycle</i> ; Versão de referência duma arquitetura <i>multicycle</i> ; Exemplos do processamento das instruções numa arquitetura <i>multicycle</i> .
26	<i>Unidade de Controlo para datapath multicycle</i> : diagrama de estados. Sinais de controlo e valores do <i>datapath multicycle</i> . Exemplos da execução sequencial de algumas instruções <i>no datapath multicycle</i> .
27	Resolução de <i>problemas</i> sobre uArquiteturas Multicycle.

## Z - Programa das Teóricas

---

### Comunicação do $\mu P$ com o exterior (I/O)

Aula	Descrição
28	<b>VII - Comunicação com o exterior:</b> Entrada e saída de dados (I/O) Acesso a dispositivos de I/O: I/O Memory-Mapped Hardware e Software (Asm). Dispositivos de I/O Embedded uControlador PIC32 (MIPS): I/O Digital: Switches e LEDs I/O Série : SPI e UART.
29	<b>I/O sob interrupção:</b> Timers e Interrupções I/O analógico: ADC e DAC; Outros periféricos: LCD e Bluetooth.