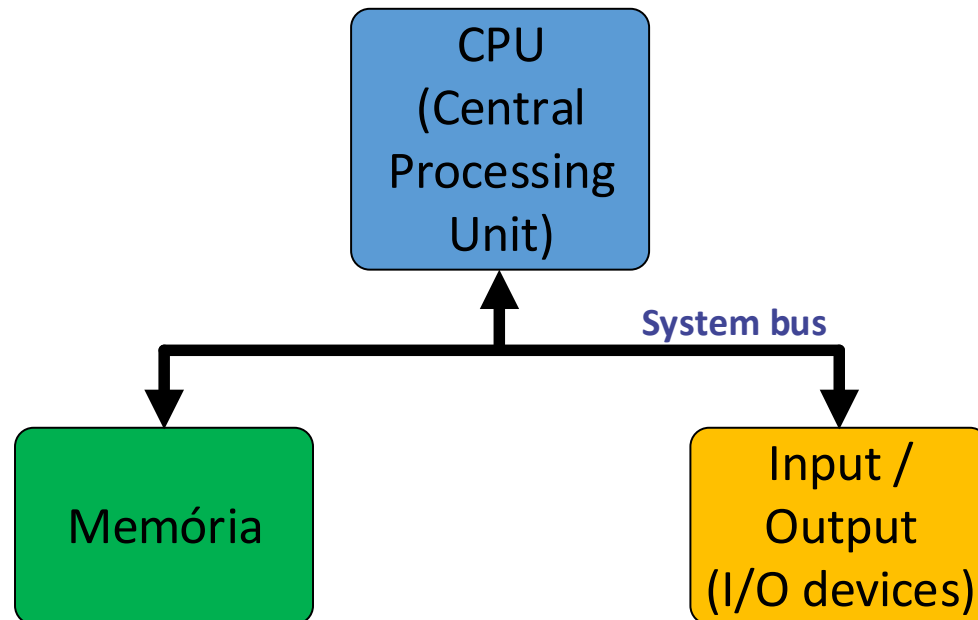


Aula 1

- Conceitos fundamentais em Arquitetura de Computadores

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira

Computador: the *big picture*



- **CPU** (ou microprocessador) – executa sequencialmente instruções
- **Memória** – armazena o programa (conjunto de instruções) e dados
- **I/O devices** – comunicação com o exterior
- **System Bus** – interliga os subsistemas

Visão simplificada do CPU

- O CPU é um sistema digital complexo. No entanto, numa visão simplificada, podemos descrevê-lo como contendo três blocos fundamentais:
 - **ALU** (Unidade Aritmética e Lógica)
 - **Registos**
 - **Unidade de controlo**
- **ALU** – realiza as operações aritméticas e lógicas mais comuns (por exemplo, soma, multiplicação, divisão, AND, OR, NOR, XOR)
- **Registos** – elementos de armazenamento (memória) localizados dentro do CPU
 - Usados para diversos fins
 - Um registo armazena uma única unidade de informação (ex. se o registo for de 8 bits pode armazenar 1 byte)
- **Unidade de controlo** - responsável pela coordenação dos vários blocos do CPU, durante a execução de uma instrução

Visão simplificada do CPU – Registos

- Na perspetiva do utilizador, os registos mais importantes são:
 - **Program Counter (PC)**
 - **Registos de utilização geral**, para armazenamento de dados (geralmente em número muito reduzido, por exemplo 32)
- **Program Counter**
 - Usado para guardar o endereço da memória onde se situa a próxima instrução a ser executada
 - No CPU, após a leitura do código de uma instrução, o valor do PC é atualizado para apontar para a instrução seguinte
- Os **registos de utilização geral** são, habitualmente, referenciados por nomes (e.g., \$4, \$a0, RBX)

Programa

- Um **programa** é um conjunto de **instruções** que vão ser executadas sequencialmente pelo CPU
- Uma instrução realiza uma operação básica, por exemplo, somar dois valores, transferir um valor da memória para o CPU, realizar uma operação lógica, ...
- A execução de uma instrução envolve, no CPU, 3 passos fundamentais:
 - **Leitura da instrução** (as instruções estão armazenadas na memória externa)
 - **Decodificação da instrução** (qual a operação a realizar, onde residem os operandos e onde deve ser colocado o resultado)
 - **Realização da operação** associada à instrução
- Após a execução de uma instrução, o CPU avança para a instrução seguinte

Assembly

- Linguagem básica de programação de microprocessadores, legível por humanos
- Conjunto de instruções que realizam operações simples
 - Somar o conteúdo de 2 registos
 - Subtrair o conteúdo de dois registos
 - Inicializar um registo com um valor
 - Transferir um valor de um registo interno para a memória
- Exemplos:

<code>add \$1, \$5, \$7</code>	<code># \$1 = \$5 + \$7</code>
<code>sub \$3, \$4, \$2</code>	<code># \$3 = \$4 - \$2</code>
<code>ori \$6, \$0, 0x1234</code>	<code># \$6 = \$0 0x1234</code>
	<code># \$6 = 0x1234</code>

Código máquina

- Sequência de bits que codifica cada uma das instruções *assembly*

- Exemplos:

Instrução *assembly*

add \$1, \$5, \$7

sub \$3, \$4, \$2

ori \$6, \$0, 0x1234

Código máquina

0x00A70820

0x00821822

0x34061234

- É gerado
 - Por um **compilador**, quando o programa é escrito numa linguagem de alto nível (por exemplo C)
 - Por um **assembler** quando o programa é escrito em ***assembly***

O MIPS

- É um **microprocessador de 32 bits**, isto é:
 - cada **registro interno** armazena uma *word* de **32 bits**
 - a **ALU** opera sobre quantidades de **32 bits**
- Tem **32 registros** internos de uso geral, com a designação nativa **\$0, \$1, \$2, ..., \$31**
- Estes registros são normalmente referenciados nos programas por um nome lógico (facilita a aplicação de uma convenção de utilização, a ver mais tarde)
 - **\$a0, \$a1, \$a2, \$a3**
 - **\$t0, \$t1, \$t2, ..., \$t9**
 - **\$s0, \$s1, \$s2, ..., \$s7**
 - **\$v0, \$v1**
 - **\$ra**
- O registro **\$0** é um caso particular, uma vez que não permite armazenamento e, quando lido, **retorna sempre o valor 0**

Exemplos de algumas instruções do MIPS

- Operações **aritméticas**

add **Rdst**,**Rsrc1**,**Rsrc2** # **Rdst** = **Rsrc1** + **Rsrc2**

▪ Ex: add \$t0,\$a0,\$t1

sub **Rdst**,**Rsrc1**,**Rsrc2** # **Rdst** = **Rsrc1** - **Rsrc2**

▪ Ex: sub \$a1,\$s0,\$t2

addi **Rdst**,**Rsrc1**,**Imm** # **Rdst** = **Rsrc1** + **Imm**

▪ Ex: addi \$t5,\$a3,0x13F4

- Operações **lógicas bitwise**

and **Rdst**,**Rsrc1**,**Rsrc2** # **Rdst** = **Rsrc1** & **Rsrc2**

or **Rdst**,**Rsrc1**,**Rsrc2** # **Rdst** = **Rsrc1** | **Rsrc2**

ori **Rdst**,**Rsrc1**,**Imm** # **Rdst** = **Rsrc1** | **Imm**

▪ Ex: ori \$v0,\$0,0x12 # \$v0 = 0x12 (zero é o
elemento neutro do OR)

Instruções MIPS – operações de deslocamento (*shift*)

- Deslocamento à esquerda (*shift left logical*):

```
sll Rdst,Rsrc,Imm # Rdst = Rsrc << Imm, em  
                  # que Imm é o n.º de bits  
                  # deslocamento à esquerda
```

- Deslocamento à direita lógico (*shift right logical*):

```
srl Rdst,Rsrc,Imm # Rdst = Rsrc >> Imm, em  
                  # que Imm é o n.º de bits  
                  # deslocamento à direita
```

- Deslocamento à direita aritmético (*shift right arithmetic*):

```
sra Rdst,Rsrc,Imm # Rdst = Rsrc >> Imm, em  
                  # que Imm é o n.º de bits  
                  # deslocamento à direita
```

Operações de deslocamento – exemplos


- `sll $t1,$t0,1` # shift left logical

```
$t0 = 0010 0100 1010 1101 0000 1111 0101 0011
      ↓
$t1 = 0100 1001 0101 1010 0001 1110 1010 0110
```

- `srl $t1,$t0,2` # shift right logical

\$t0 = 1010 0100 1010 1101 0000 1111 0101 0011


\$t1 = 0010 1001 0010 1011 0100 0011 1101 0100



- `sra $t1,$t0,3` # shift right arithmetic

\$t0 = 1010 0100 1010 1101 0000 1111 0101 0011

\$t1 = 1111 0100 1001 0101 1010 0001 1110 1010



Anatomia de um programa *Assembly*

	<code>.data</code>		
	<code>...</code>		
	<code>...</code>		
	<code>.text</code>		
	<code>.globl main</code>		
<code># label</code>	<code># Instrução</code>	<code># comentário</code>	
<code>main:</code>	<code>ori \$t0,\$0,3</code>	<code># \$t0 = 3</code>	
	<code>ori \$t2,\$0,8</code>	<code># \$t2 = 8</code>	
	<code>add \$t1,\$t0,\$t0</code>	<code># \$t1 = \$t0 + \$t0</code>	
	<code>add \$t1,\$t1,\$t2</code>	<code># \$t1 = \$t1 + \$t2</code>	
	<code>jr \$ra</code>	<code># fim do programa</code>	
<hr/>			
<code>.text, .data</code>	<code>-> ordens para o Assembler (diretivas)</code>		
<code>nome:</code>	<code>-> label (nome dado a um endereço, e.g., main, str1,...)</code>		
<code>ori</code>	<code>-> mnemónica de uma instrução</code>		
<code>\$t0, \$0, 3</code>	<code>-> operandos de uma instrução</code>		

MARS – um ambiente de simulação para o MIPS

- MARS - MIPS Assembler and Runtime Simulator
- Ambiente integrado de Desenvolvimento (IDE), com:
 - Editor
 - Assembler
 - Simulador
- O simulador permite:
 - Execução do programa *assembly* de uma só vez, ou instrução a instrução (*single step execution*)
 - Acesso aos registos internos do CPU para visualizar/alterar o seu valor
 - Acesso à memória para visualizar/alterar o seu conteúdo

C:\AC1\aula1_2.s - MARS 4.5

File Edit Run Settings Tools Help

Edit Execute

aula1_2.s

```

1      .data
2      .text
3      .globl main
4  main: ori    $t0,$0,5    # $t0 = valor inicial
5        ori    $t2,$0,8    # $t2 = 8
6        add    $t1,$t0,$t0 # $t1 = $t0 + $t0 = x + x
7        add    $t1,$t1,$t2 # $t1 = $t1 + $t2 = y = 2x
8                                # ($t1 tem o valor calculado)
9        jr     $ra         # fim do programa
10

```

Line: 4 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

Clear

Coproc 1 Coproc 0

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x7ffff000
\$a2	6	0x7ffff004
\$a3	7	0x00000000
\$t0	8	0x00000005
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffefffc
\$fp	30	0x00000000
\$ra	31	0x00400018
pc		0x00400028
hi		0x00000000
lo		0x00000000

C:\AC1\aula1_2.s - MARS 4.5

File Edit Run Settings Tools Help

0101

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x8fa40000	lw \$4,0x00000000(\$29)	171: lw \$a0 0(\$sp) ...
<input type="checkbox"/>	0x00400004	0x27a50004	addiu \$5,\$29,0x000...	172: addiu \$a1 \$sp 4 ...
<input type="checkbox"/>	0x00400008	0x24a60004	addiu \$6,\$5,0x0000...	173: addiu \$a2 \$a1 4 ...
<input type="checkbox"/>	0x0040000c	0x00041080	sll \$2,\$4,0x00000002	174: sll \$v0 \$a0 2
<input type="checkbox"/>	0x00400010	0x00c23021	addu \$6,\$6,\$2	175: addu \$a2 \$a2 \$v0
<input type="checkbox"/>	0x00400014	0x00c10009	jal 0x00400024	176: jal main
<input type="checkbox"/>	0x00400018	0x00000000	nop	177: nop
<input type="checkbox"/>	0x0040001c	0x2402000a	addiu \$2,\$0,0x0000...	179: li \$v0 10

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

Mars Messages Run I/O

Assemble: assembling C:\AC1\exceptions.s, C:\AC1\aula1_2.s

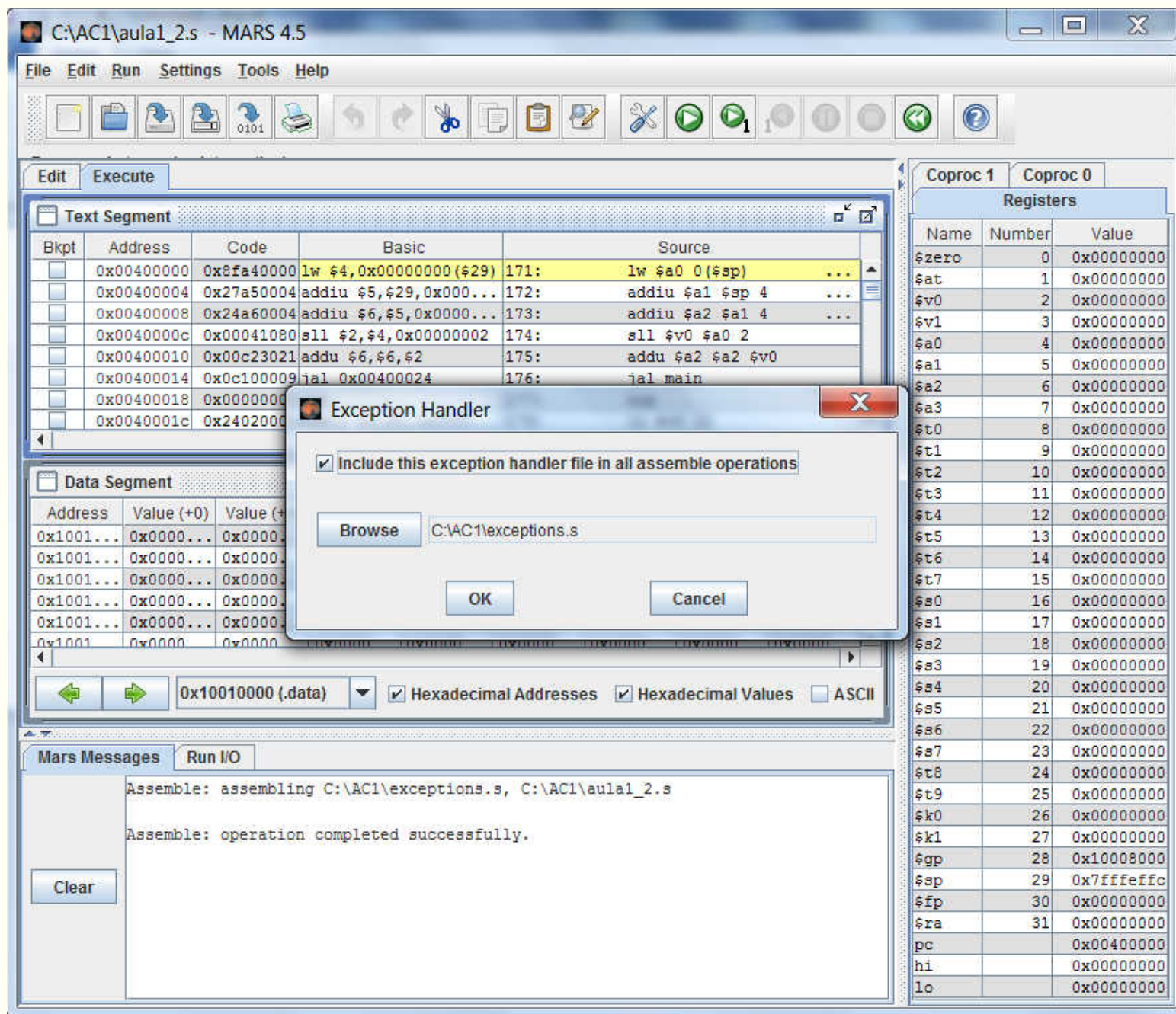
Assemble: operation completed successfully.

Clear

Coproc 1 Coproc 0

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffeffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000



C:\AC1\aula1_2.s - MARS 4.5

File Edit Run Settings Tools Help

0101

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x0040001c	0x2402000a	addiu \$2,\$0,0x0000...	179: li \$v0 10
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	180: syscall ...
<input type="checkbox"/>	0x00400024	0x34080005	ori \$8,\$0,0x00000005	4: main: ori \$t0,\$0,5 #...
<input type="checkbox"/>	0x00400028	0x340a0008	ori \$10,\$0,0x00000008	5: ori \$t2,\$0,8 #...
<input type="checkbox"/>	0x0040002c	0x01084820	add \$9,\$8,\$8	6: add \$t1,\$t0,\$t0 #...
<input type="checkbox"/>	0x00400030	0x012a4820	add \$9,\$9,\$10	7: add \$t1,\$t1,\$t2 #...
<input type="checkbox"/>	0x00400034	0x03e00008	jr \$31	9: jr \$ra #...
<input type="checkbox"/>	0x80000180	0x0001d821	addu \$27,\$0,\$1	81: move \$k1 \$at ...

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

Mars Messages Run I/O

Clear

Coproc 1 Coproc 0

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x7ffff000
\$a2	6	0x7ffff004
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffeffc
\$fp	30	0x00000000
\$ra	31	0x00400018
pc		0x00400024
hi		0x00000000
lo		0x00000000

C:\AC1\aula1_2.s - MARS 4.5

File Edit Run Settings Tools Help

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x0040001c	0x2402000a	addiu \$2,\$0,0x0000...	179: li \$v0 10
	0x00400020	0x0000000c	syscall	180: syscall
	0x00400024	0x34080005	ori \$8,\$0,0x00000005	4: main: ori \$t0,\$0,5 #...
	0x00400028	0x340a0008	ori \$10,\$0,0x00000008	5: ori \$t2,\$0,8 #...
	0x0040002c	0x01084820	add \$9,\$8,\$8	6: add \$t1,\$t0,\$t0 #...
	0x00400030	0x012a4820	add \$9,\$9,\$10	7: add \$t1,\$t1,\$t2 #...
	0x00400034	0x03e00008	jr \$31	9: jr \$ra #...
	0x80000180	0x0001d821	addu \$27,\$0,\$1	81: move \$k1 \$at

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...
0x1001...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...	0x0000...

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

Mars Messages Run I/O

Clear

Coproc 1 Coproc 0

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x7ffff000
\$a2	6	0x7ffff004
\$a3	7	0x00000000
\$t0	8	0x00000005
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00400018
pc		0x00400028
hi		0x00000000
lo		0x00000000

System Calls

- *System Calls* são funções do sistema operativo (SO) que implementam serviços básicos de I/O:
 - imprimir uma *string* no ecrã, ler um inteiro do teclado, ler uma *string* do teclado, imprimir um inteiro, etc.
- O MARS disponibiliza cerca de 50 *system calls*
 - O registo **\$v0** é usado para identificar a *system call*
 - Os registos **\$a0** a **\$a3** são usados para transferir valores (argumentos) para a *system call*
 - O *system call* pode usar **\$v0** para devolver um valor
- Exemplo

```
ori    $v0,$0,11      # $v0=11 (system call
                        #   print_char()
ori    $a0,$0,0x31     # $a0 = 0x31 = '1'
syscall                # chama a system call
```

System Calls

- Como funciona um *system call*, na perspetiva do utilizador:
 1. O Sistema Operativo verifica **\$v0** para saber qual a tarefa a realizar
 2. Se necessário o Sistema Operativo lê os valores de entrada dos registos **\$a0 a \$a3** (e.g. imprimir um carater no ecrã)
 3. O Sistema Operativo executa a tarefa
 4. O Sistema Operativo coloca o resultado no registo **\$v0** (se isso se aplicar, e.g. ler um inteiro do teclado)

```
ori    $v0,$0,11      # $v0=11 (system call
                        #   print_char()
ori    $a0,$0,0x31     # $a0 = 0x31 = '1'
syscall                # chama a system call
```