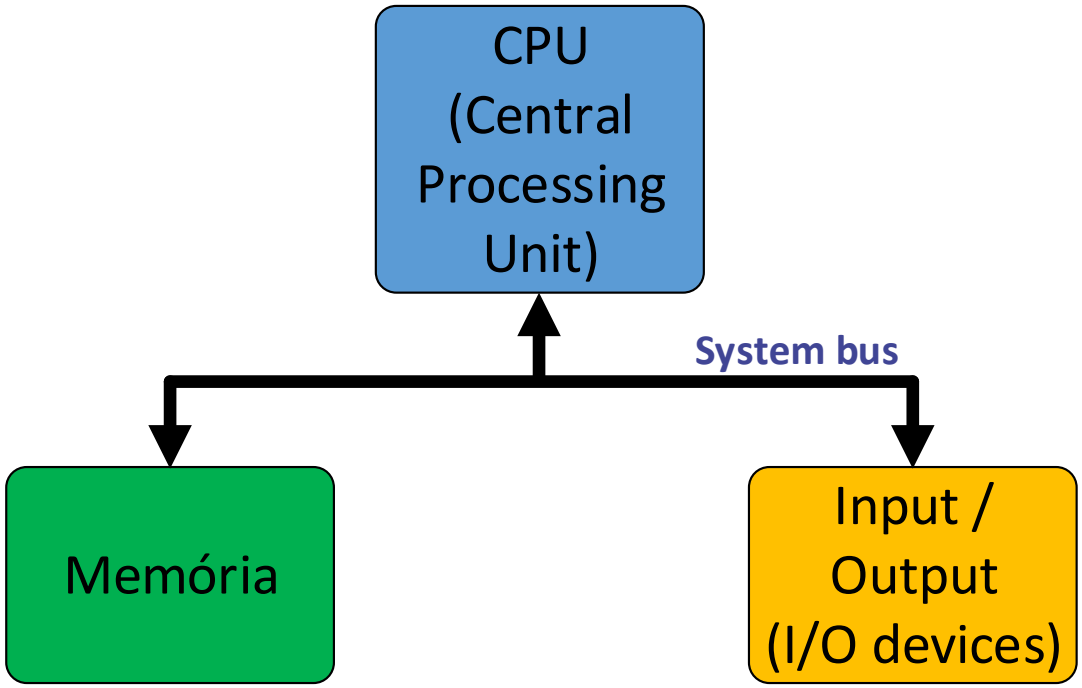
Aula prática 1

- Conceitos básicos de Arquitetura de Computadores.
- Programação em linguagem *assembly*: estrutura de um programa e instruções básicas do MIPS.
- Apresentação do MARS

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. 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, adição, 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

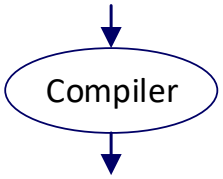
Visão simplificada do CPU – Registos

- 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., no MIPS: $0, $1,…,$31)

Níveis de Representação

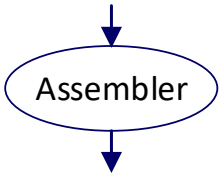
High-level language
program (in C)

```
unsigned char toUpper(unsigned char c)
{
    if(c >= 'a' && c <= 'z')
        return (c - 0x20);
    else
        return c;
}
```



Assembly language
program (for MIPS)

```
toUpper:    addiu $5,$4,-97
            sltiu $1,$5,26
            or    $2,$4,$0
            beq   $1,$0,else
            addiu $2,$4,-32
else:       jr    $31
```



Binary machine language
program (for MIPS)

```
00100100100001001111111110011111   (0x2485ff9f)
00101100101000010000000000011010   (0x2ca1001a)
00000000100000000001000000100101   (0x00801025)
00010000001000000000000000000001   (0x10200001)
00100100100000101111111111100000   (0x2482ffe0)
00000011111000000000000000001000   (0x03e00008)
```

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:

```
add $1,$5,$7          # $1 = $5 + $7
sub $3,$4,$2          # $3 = $4 - $2
ori $6,$0,0x1234      # $6 = $0 | 0x1234
                      # $6 = 0x1234
```

Código máquina

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

Instrução *assembly*	Código máquina
`add $1,$5,$7`	`0x00A70820`
`sub $3,$4,$2`	`0x00821822`
`ori $6,$0,0x1234`	`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 linguagem ***assembly***

O MIPS

- É um **microprocessador de 32 bits**, isto é:
  - cada **registo interno** armazena uma *word* de **32 bits**
  - a **ALU** opera sobre quantidades de **32 bits**
- Tem **32 registos** internos de uso geral, com a designação nativa em *assembly* **$0**, **$1**, **$2**, …, **$31**
- Estes registos são normalmente referenciados nos programas por um nome lógico (facilita a aplicação de uma convenção de utilização, a ver mais tarde)
  - **$a0**, **$a1**, **$a2**, **$a3**
  - **$t0**, **$t1**, **$t2**, …, **$t9**
  - **$s0**, **$s1**, **$s2**, …, **$s7**
  - **$v0**, **$v1**
  - **$ra**
- O registo **$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)

(Rdst – registo destino; Rsrc – Registo fonte)

Anatomia de um programa *Assembly*

```
            .data
            …                                   } Dados
            …
            .text
            .globl main
# label     # Instrução                        # comentário
main:       ori  $t0,$0,3                      # $t0 = 3
            ori  $t2,$0,8                      # $t2 = 8
            add  $t1,$t0,$t0                   # $t1 = $t0 + $t0   } Instruções
            add  $t1,$t1,$t2                   # $t1 = $t1 + $t2
            jr   $ra                           # fim do programa
```

`.text, .data` -> ordens para o Assembler (diretivas)

`nome:` -> label (nome dado a um endereço, e.g., main, str1,…)

`ori` -> mnemónica de uma instrução

`$t0,$0,3` -> operandos de uma instrução

Principais regras sintáticas:

1. **Linguagem orientada à linha**. Uma linha pode conter:
   - Nada
   - Um *label* e/ou uma diretiva e/ou um comentário
   - Um *label* e/ou uma instrução e/ou um comentário
2. **Labels**
   - São identificadores únicos terminados com o caracter ‘:’
   - Devem obrigatoriamente começar na primeira coluna (apenas os comentários podem também começar nesta coluna)
   - O primeiro caracter tem, obrigatoriamente, de ser uma letra ou o símbolo ‘_’
   - Os restantes caracteres podem ser letras, algarismos ou o símbolo’_’
   - Não podem conter separadores (espaços, vírgulas e/ou tabs) entre o nome e o terminador ':’

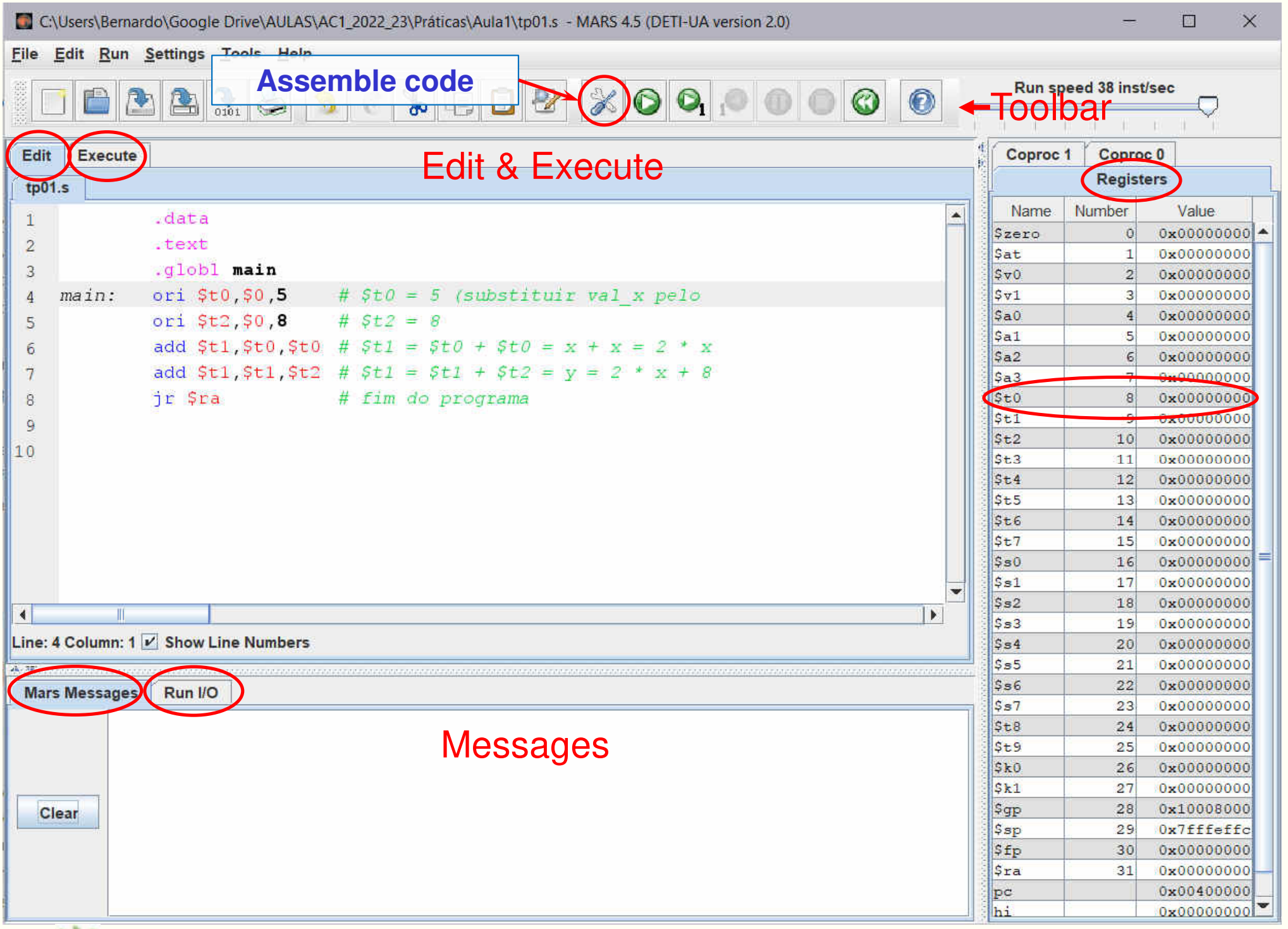
Principais regras sintáticas:

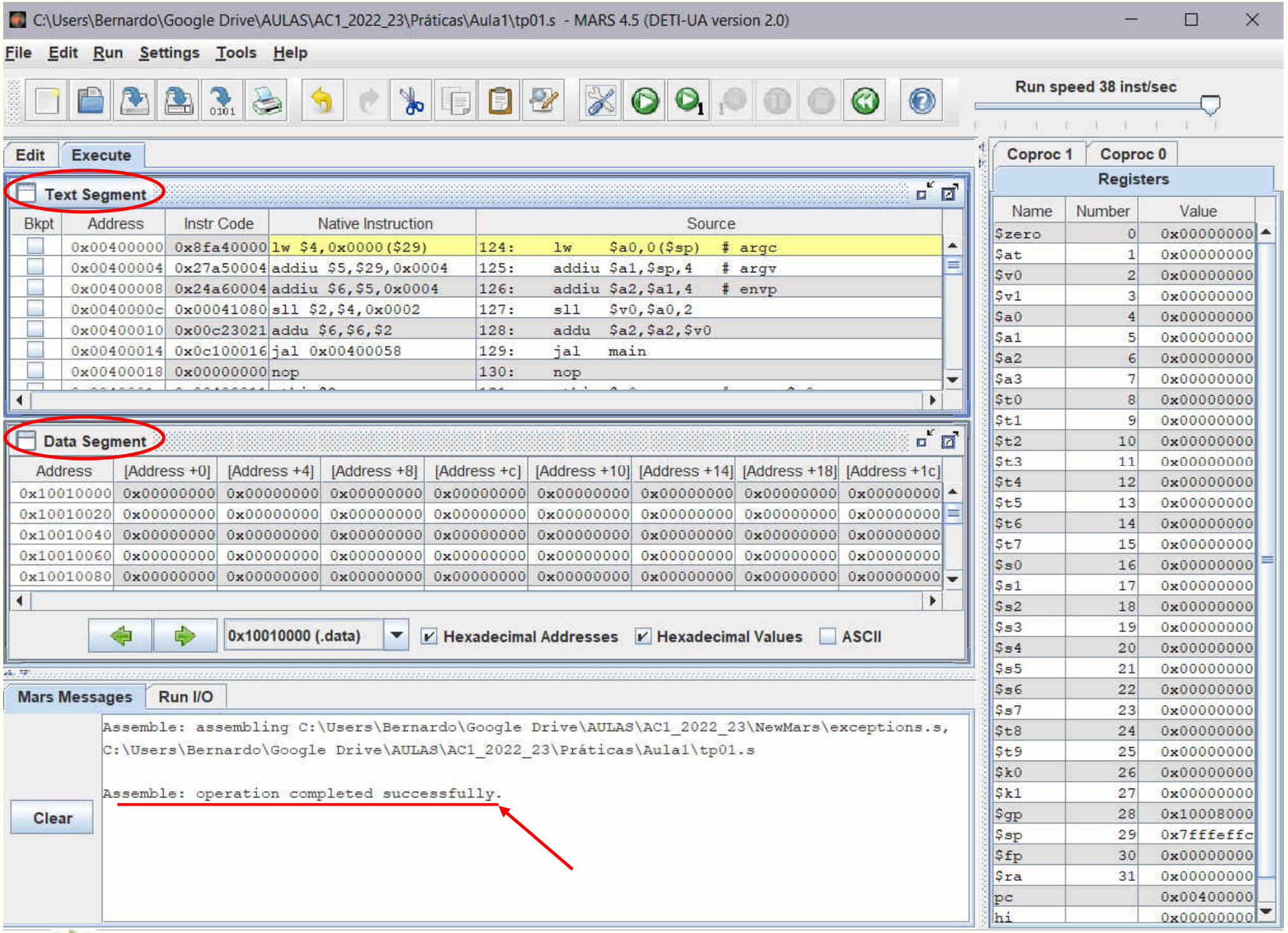
3. **Diretivas** (ordens para o Assembler)
   - Começam obrigatoriamente pelo caracter '.'
   - Só podem começar a partir da segunda coluna
   - Entre a diretiva e a expressão seguinte (quando se aplique), devem existir espaço(s) e/ou tabs, mas não vírgulas
   - Nas diretivas que aceitam múltiplas expressões, estas têm de estar separadas por uma única vírgula (podem conter também espaços e tabs)

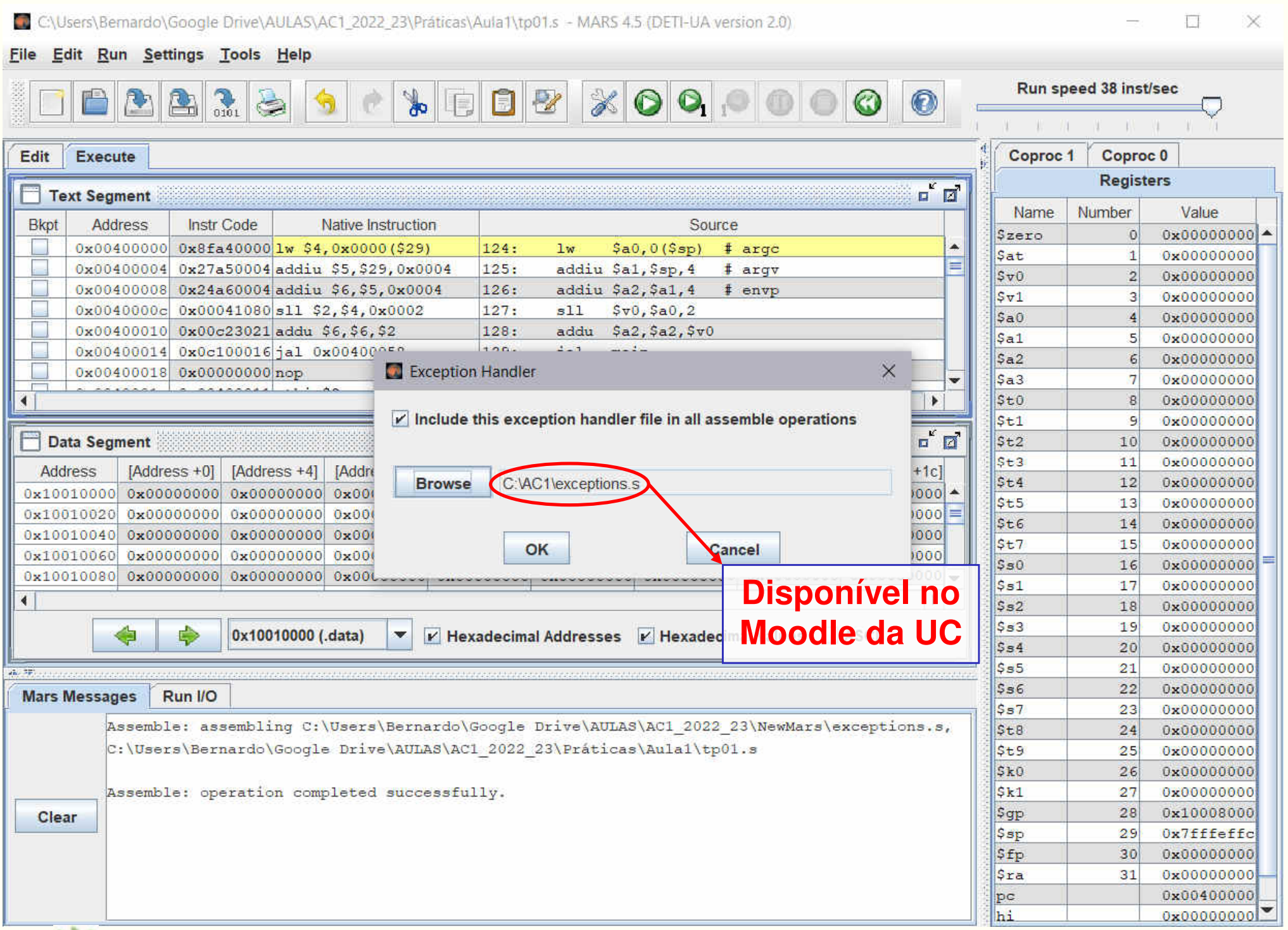
4. **Instruções**
   - Podem conter apenas uma expressão (mnemónica) e entre zero a três operandos
   - Só podem começar a partir da segunda coluna
   - Entre a instrução e o primeiro operando(quando se aplique), deve(m) existir espaço(s) e/ou tabs, mas não vírgulas
   - Nas instruções com mais do que um operando, estes têm de estar separadas por uma única vírgula (podem conter também espaços e tabs)

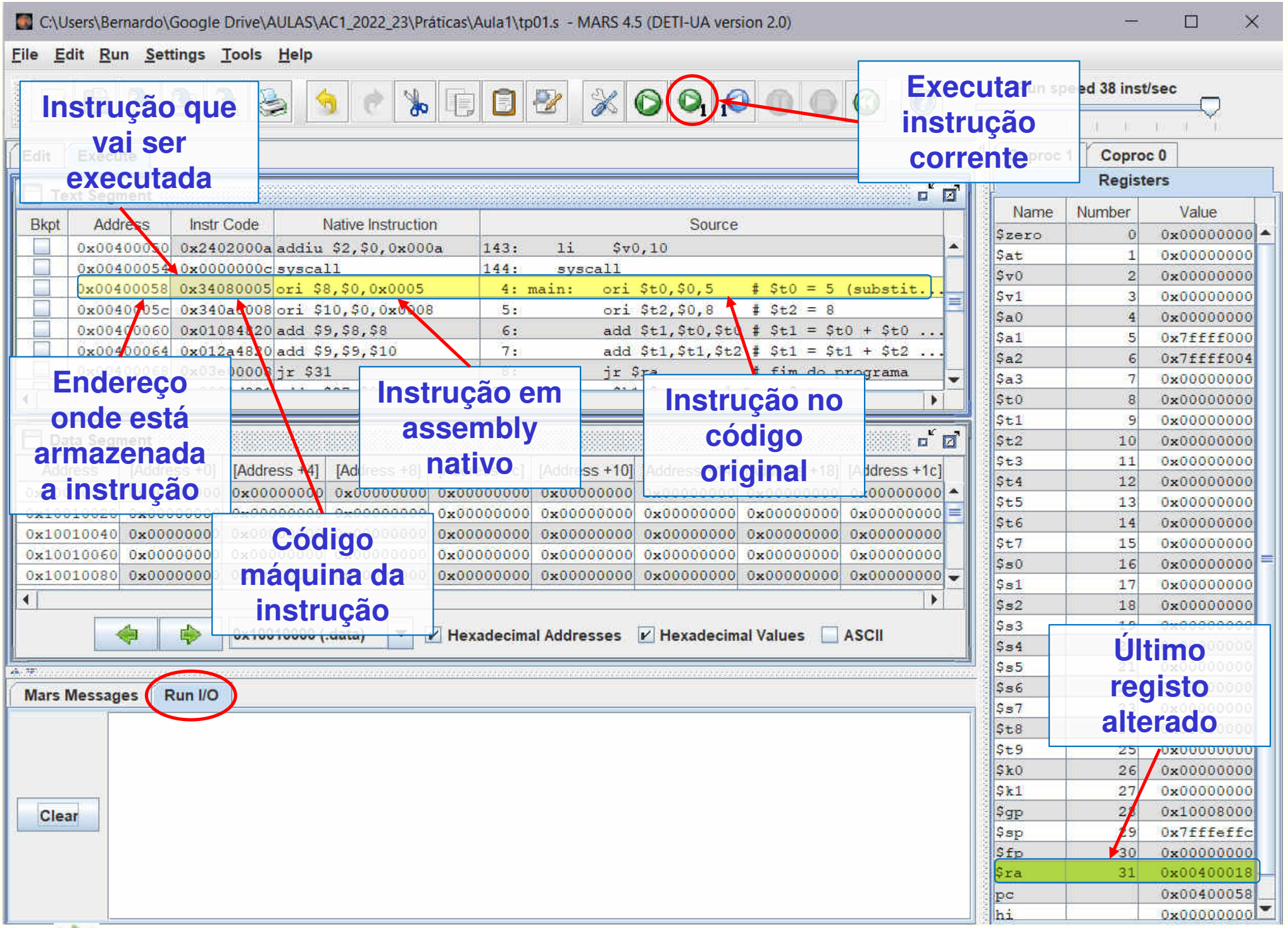
MARS – um ambiente de simulação para o MIPS

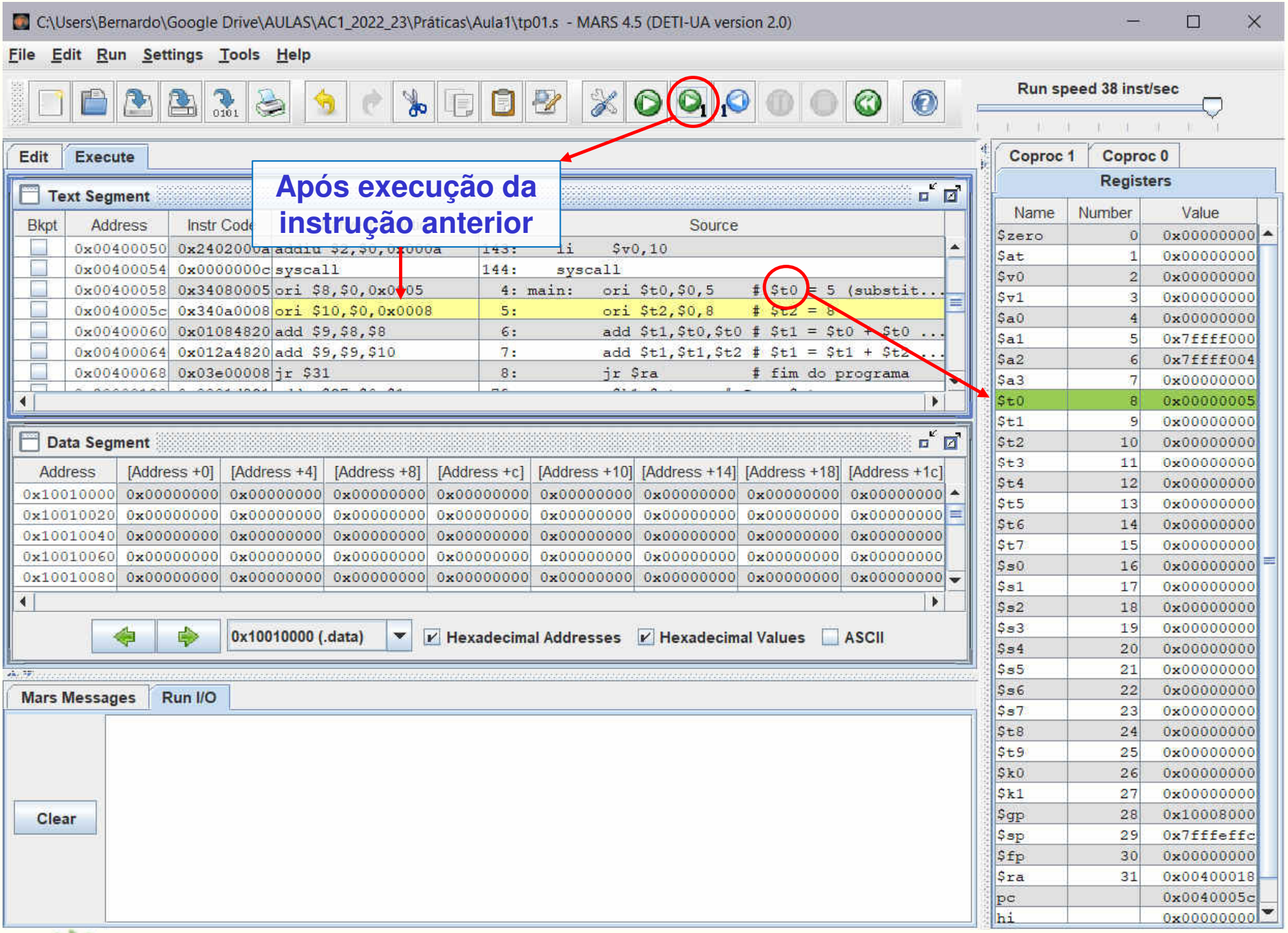
- MARS - MIPS Assembler and Runtime Simulator
- Ambiente integrado de Desenvolvimento (IDE), com:
  - Editor
  - Assembler
  - Simulador
- O simulador permite:
  - Execução do programa *assembly* de uma só vez, ou instrução a instrução (*single step execution*)
  - Acesso aos registos internos do CPU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


C:\Users\Bernardo\Google Drive\AULAS\AC1_2022_23\Práticas\Aula1\tp01.s - MARS 4.5 (DETI-UA version 2.0)
File Edit Run Settings Tools Help
Assemble code
Run speed 38 inst/sec
Toolbar
Edit
Execute
Edit & Execute
tp01.s
1 .data
2 .text
3 .globl main
4 main: ori $t0,$0,5 # $t0 = 5 (substituir val_x pelo
5 ori $t2,$0,8 # $t2 = 8
6 add $t1,$t0,$t0 # $t1 = $t0 + $t0 = x + x = 2 * x
7 add $t1,$t1,$t2 # $t1 = $t1 + $t2 = y = 2 * x + 8
8 jr $ra # fim do programa
9
10
Line: 4 Column: 1
Show Line Numbers
Mars Messages
Run I/O
Messages
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


C:\Users\Bernardo\Google Drive\AULAS\AC1_2022_23\Práticas\Aula1\tp01.s - MARS 4.5 (DETI-UA version 2.0)
File Edit Run Settings Tools Help
Run speed 38 inst/sec
Edit Execute
Text Segment
Bkpt Address Instr Code Native Instruction Source
0x00400000 0x8fa40000 lw $4,0x0000($29) 124: lw $a0,0($sp) # argc
0x00400004 0x27a50004 addiu $5,$29,0x0004 125: addiu $a1,$sp,4 # argv
0x00400008 0x24a60004 addiu $6,$5,0x0004 126: addiu $a2,$a1,4 # envp
0x0040000c 0x00041080 sll $2,$4,0x0002 127: sll $v0,$a0,2
0x00400010 0x00c23021 addu $6,$6,$2 128: addu $a2,$a2,$v0
0x00400014 0x0c100016 jal 0x00400058 129: jal main
0x00400018 0x00000000 nop 130: nop
Data Segment
Address [Address +0] [Address +4] [Address +8] [Address +c] [Address +10] [Address +14] [Address +18] [Address +1c]
0x10010000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010020 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010040 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010060 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010080 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010000 (.data)
Hexadecimal Addresses
Hexadecimal Values
ASCII
Mars Messages
Run I/O
Assemble: assembling C:\Users\Bernardo\Google Drive\AULAS\AC1_2022_23\NewMars\exceptions.s,
C:\Users\Bernardo\Google Drive\AULAS\AC1_2022_23\Práticas\Aula1\tp01.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


C:\Users\Bernardo\Google Drive\AULAS\AC1_2022_23\Práticas\Aula1\tp01.s - MARS 4.5 (DETI-UA version 2.0)
File Edit Run Settings Tools Help
Run speed 38 inst/sec
Edit Execute
Text Segment
Bkpt Address Instr Code Native Instruction Source
0x00400000 0x8fa40000 lw $4,0x0000($29) 124: lw $a0,0($sp) # argc
0x00400004 0x27a50004 addiu $5,$29,0x0004 125: addiu $a1,$sp,4 # argv
0x00400008 0x24a60004 addiu $6,$5,0x0004 126: addiu $a2,$a1,4 # envp
0x0040000c 0x00041080 sll $2,$4,0x0002 127: sll $v0,$a0,2
0x00400010 0x00c23021 addu $6,$6,$2 128: addu $a2,$a2,$v0
0x00400014 0x0c100016 jal 0x00400058
0x00400018 0x00000000 nop
Exception Handler
Include this exception handler file in all assemble operations
Browse
C:\AC1\exceptions.s
OK
Cancel
Disponível no Moodle da UC
Data Segment
Address [Address +0] [Address +4]
0x10010000 0x00000000 0x00000000
0x10010020 0x00000000 0x00000000
0x10010040 0x00000000 0x00000000
0x10010060 0x00000000 0x00000000
0x10010080 0x00000000 0x00000000
0x10010000 (.data)
Hexadecimal Addresses
Mars Messages Run I/O
Assemble: assembling C:\Users\Bernardo\Google Drive\AULAS\AC1_2022_23\NewMars\exceptions.s,
C:\Users\Bernardo\Google Drive\AULAS\AC1_2022_23\Práticas\Aula1\tp01.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


C:\Users\Bernardo\Google Drive\AULAS\AC1_2022_23\Práticas\Aula1\tp01.s - MARS 4.5 (DETI-UA version 2.0)
File Edit Run Settings Tools Help
Instrução que vai ser executada
Executar instrução corrente
Endereço onde está armazenada a instrução
Instrução em assembly nativo
Instrução no código original
Código máquina da instrução
Último registo alterado
Coproc 0
Registers
Bkpt Address Instr Code Native Instruction Source
0x00400050 0x2402000a addiu $2,$0,0x000a 143: li $v0,10
0x00400054 0x0000000c syscall 144: syscall
0x00400058 0x34080005 ori $8,$0,0x0005 4: main: ori $t0,$0,5 # $t0 = 5 (substit...
0x0040005c 0x340a0008 ori $10,$0,0x0008 5: ori $t2,$0,8 # $t2 = 8
0x00400060 0x01084820 add $9,$8,$8 6: add $t1,$t0,$t0 # $t1 = $t0 + $t0 ...
0x00400064 0x012a4820 add $9,$9,$10 7: add $t1,$t1,$t2 # $t1 = $t1 + $t2 ...
jr $31 jr $ra # fim do programa
Hexadecimal Addresses
Hexadecimal Values
ASCII
Mars Messages
Run I/O
Clear
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
$ra 31 0x00400018
pc 0x00400058
hi 0x00000000


C:\Users\Bernardo\Google Drive\AULAS\AC1_2022_23\Práticas\Aula1\tp01.s - MARS 4.5 (DETI-UA version 2.0)
File Edit Run Settings Tools Help
Run speed 38 inst/sec
Edit Execute
Após execução da instrução anterior
Text Segment
Bkpt Address Instr Code Source
0x00400050 0x2402000a addiu $2,$0,0x000a 143: li $v0,10
0x00400054 0x0000000c syscall 144: syscall
0x00400058 0x34080005 ori $8,$0,0x0005 4: main: ori $t0,$0,5 # $t0 = 5 (substit...
0x0040005c 0x340a0008 ori $10,$0,0x0008 5: ori $t2,$0,8 # $t2 = 8
0x00400060 0x01084820 add $9,$8,$8 6: add $t1,$t0,$t0 # $t1 = $t0 + $t0 ...
0x00400064 0x012a4820 add $9,$9,$10 7: add $t1,$t1,$t2 # $t1 = $t1 + $t2 ...
0x00400068 0x03e00008 jr $31 8: jr $ra # fim do programa
Data Segment
Address [Address +0] [Address +4] [Address +8] [Address +c] [Address +10] [Address +14] [Address +18] [Address +1c]
0x10010000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010020 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010040 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010060 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010080 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
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 0x7fffeffc
$fp 30 0x00000000
$ra 31 0x00400018
pc 0x0040005c
hi 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
```

System Calls mais importantes

Tabela IV: *System Calls* do MARS			
Protótipo equivalent em C	**$v0**	**Parâmetros de entrada**	**Retorno**
void print_int10(**int** value)	1	$a0 = value (int)	
void print_float(**float** value)	2	$f12 = value (float)	
void print_double(**double** value)	3	$f12 = value (double)	
void print_string(**char** *str)	4	$a0 = str	
int read_int(**void**)	5		$v0
float read_float(**void**)	6		$f0
double read_double(**void**)	7		$f0
void read_string(**char** *buf, **int** length)	8	$a0 = buf, $a1 = length	
void *sbrk(**int** amount)	9	$a0 = amount	$v0
void exit(**void**)	10		
void print_char(**char** value)	11	$a0 = value (char)	
char read_char(**void**)	12		$v0
void print_int16(**unsigned int** value)	34	$a0 = value (unsigned int)	
void print_int2(**unsigned int** value)	35	$a0 = value (unsigned int)	
void print_intu10(**unsigned int** value)	36	$a0 = value (unsigned int)	