MIPS

Sara Fernandes

GUIMARÃES, 2020

CONTEXTO

OBJETIVO

Familiarização com o SPIM, aprendizagem da estrutura de um programa em linguagem Assembly, utilização de ferramentas de depuração do simulador, gestão de dados em memória. Operações aritméticas e lógicas e avaliação de condições. (slides de suporte às fichas práticas disponibilizadas na plataforma de elearling da UM)

OVERVIEW

1	Introdução	Introdução à Linguagem Assembly; Estrutura dos programas em Assembly.
2	SPIM	Introdução ao QTSpim.
3	Gestão de Dados em memória	Declaração de dados em memória; Declaração de bytes em memória, Declaração de cadeias em caracteres; Reserva de espaço em memória.
4	Operações aritméticas e lógicas e avaliação de condições	Operações aritméticas com constantes e com dados em memória. Multiplicação, divisão e operações lógicas. Avaliação de condições simples e compostas por operadores lógicos. Definição de uma notação uniforme para a escrita de fluxogramas. Criação de fluxogramas a partir de exemplos de programas.

INTRODUÇÃO LINGUAGEM ASSEMBLY

- Representação simbólica da codificação binária de um computador: linguagem máquina.
- Composta por microinstruções que indicam que operação digital deve o computador fazer.
- Cada instrução é composta por um conjunto ordenado de “zeros” e “uns”, estruturado em campos.
- Cada campo contém informação que se complementa para indicar ao processador a ação a realizar.
- Cada instrução em linguagem Assembly corresponde a uma instrução de linguagem máquina, mas, em vez de ser especificada em termos de zeros e uns, é especificada utilizando mnemónicas e nomes simbólicos. Por exemplo: a instrução que soma dois números guardados nos registos $S3 e $S2 e coloca o resultado em $S1, poderá ser codificada como “add $1, $S2, $S3 (ver Ficha 1, programa1.s)

INTRODUÇÃO

ESTRUTURA DOS PROGRAMAS EM ASSEMBLY

- Ao longo do tempo descobriremos a sintaxe da linguagem Assembly. Contudo, torna-se necessário introduzir desde já a estrutura principal de um programa escrito em Assembly. Alguns conceitos básicos:

- **COMENTÁRIOS**: são especialmente importantes quando se trabalha com linguagens de baixo nível, pois ajudam ao desenvolvimento dos programas e são utilizados exaustivamente. Os comentários começam com o caracter #

- **IDENTIFICADORES:** Definem-se como sendo sequências de caracteres alfanuméricos, underscores (_) ou pontos (.) que não começam por um número. Os códigos de operações são palavras reservadas da linguagem e não podem ser usadas como identificadores (ex: *addu*)

- **ETIQUETAS**: Identificadores que se situam no principio de uma linha e que são sempre seguidos de dois pontos. Servem para dar um nome ao elemento definido num endereço de memória. Pode-se controlar o fluxo de execução de um programa criando saltos para as etiquetas.

INTRODUÇÃO

ESTRUTURA DOS PROGRAMAS EM ASSEMBLY

- **PSEUDO-INSTRUÇÕES:** Instruções que o Assembly interpreta e traduz numa ou mais microinstruções (em linguagem máquina).

- **DIRETIVAS:** Instruções que o Assembly interpreta a fim de informar ao processador a forma de traduzir o programa. Por exemplo, a diretiva .text informa que se trata de uma zona de código; a diretiva .data indica que se segue uma zona de dados. São identificadores reservados, e iniciam-se sempre por um ponto.

Dado o seguinte programa (programa1.s, Ficha 1) :

```
        .data
A: .word 14
B: .word 5
C: .word 0
        .text
main:                   # Start of code section
        lw $s3,A
        lw $s2,B
        add $s1,$s3,$s2
        sw $s1,C
```

- Indique as etiquetas, diretivas e comentários que surgem no mesmo.

CONTEXTO

OBJETIVO

Familiarização com o SPIM, aprendizagem da estrutura de um programa em linguagem Assembly, utilização de ferramentas de depuração do simulador, gestão de dados em memória. Operações aritméticas e lógicas e avaliação de condições.

OVERVIEW

1	Introdução	Introdução à Linguagem Assembly; Estrutura dos programas em Assembly;
2	SPIM	Introdução ao QTSpim
3	Gestão de Dados em memória	Declaração de dados em memória; Declaração de bytes em memória, Declaração de cadeias em caracteres; Reserva de espaço em memória
4	Operações aritméticas e lógicas e avaliação de condições	Operações aritméticas com constantes e com dados em memória. Multiplicação, divisão e operações lógicas. Operadores de rotação. Avaliação de condições simples e compostas por operadores lógicos. Definição de uma notação uniforme para a escrita de fluxogramas. Criação de fluxogramas a partir de exemplos de programas.

SPIM

O simulador SPIM

- O SPIM é um simulador que corre programas para as arquiteturas MIPS R2000 e R3000. O simulador pode carregar e executar programas em linguagem Assembly destas arquiteturas. O processo através do qual um ficheiro fonte em linguaem Assembly é traduzido num ficheiro executável tem duas fases:

    1. *Assembling*, implementada pelo *assembler*.
    2. *Linking*, implementada pelo *linker*.

- A atual versão do mips, e que utilizamos nas aulas, é o **QTSPIM** .

- A janela do SPIM encontra-se dividida em 4 painéis:
    - **Painel dos Registos**: Atualizado sempre que o programa pára de correr. Mostra o conteúdo de todos os registos do MIPS.
    - **Painel de Mensagens**. Mostra as mensagens de erro, sucesso, etc.
    - **Segmento de Dados**: Mostra os endereços e conteúdos das palavras em memória.

O simulador SPIM

- **Segmento de Texto**: Mostra as instruções do nosso programa e também as instruções do núcleo (kernel) do MIPS. Cada instrução é apresentada numa linha:

[0x00400000] 0x8fa4000 lw $4,0($29) ; 89:lw $a0, 0($sp)

✓ O primeiro número, entre parêntesis retos, é o endereço de memória(em hexadecimal) onde a instrução reside. O segundo número é a codificação numérica da instrução. O terceiro valor é a descrição mnemónica da instrução, e tudo o que segue o ponto e vírgula constituiu a linha de ficheiro Assembly que produziu a instrução. O número 89 é o número da linha nesse ficheiro.

UTILIZAÇÃO DAS FERRAMENTAS DE DEPURAÇÃO

Usando o editor de texto respetivo (Notepad ++ (Windows), Sublime (MAC)), abra o programa1.s da Ficha Prática 1 e analise o mesmo.

Usando o Step do simulador, e depois de responder às questões da Ficha, preencha a seguinte tabela:

Etiqueta	Endereço (OS)	Instrução nativa	Instrução Fonte

CONTEXTO

OBJETIVO

Familiarização com o SPIM, aprendizagem da estrutura de um programa em linguagem Assembly, utilização de ferramentas de depuração do simulador, gestão de dados em memória. Operações aritméticas e lógicas e avaliação de condições.

OVERVIEW

1	Introdução	Introdução à Linguagem Assembly; Estrutura dos programas em Assembly;
2	SPIM	Introdução ao QTSpim
3	Gestão de Dados em memória	Declaração de dados em memória; Declaração de bytes em memória, Declaração de cadeias em caracteres; Reserva de espaço em memória
4	Operações aritméticas e lógicas e avaliação de condições	Operações aritméticas com constantes e com dados em memória. Multiplicação, divisão e operações lógicas. Operadores de rotação. Avaliação de condições simples e compostas por operadores lógicos. Definição de uma notação uniforme para a escrita de fluxogramas. Criação de fluxogramas a partir de exemplos de programas.

GESTÃO DE DADOS EM MEMÓRIA

INTRODUÇÃO

- O primeiro passo para o desenvolvimento de programas em Assembly consiste em saber gerir os dados em memória.
- Em Assembly, é da responsabilidade do programador toda a gestão de memória, o que pode ser feito através das diretivas que irão ser estudadas.
- Antes de prosseguir, convém relembrar que a unidade base de endereçamento é o **byte**.
- Uma palavra ou ***word*** tem 4 bytes (32 bits), a mesma dimensão que o barramento de dados do MIPS. Desta forma, qualquer acesso à memória supõe a leitura de 4 bytes (1 palavra): o byte com o endereço especificado e os 3 bytes que se seguem.
- Os endereços devem estar alinhados em posições múltiplas de quatro. Também é possível transferir meia-palavra (***half-word***).

DECLARAÇÃO DE PALAVRAS EM MEMÓRIA

- Começaremos por utilizar as diretivas .data e .word. Crie o ficheiro aula2a.s:

```
        .data           #segmento de dados
palavra1:  .word 13     #decimal
palavra2: .word 0x15    # hexadecimal
```

A diretiva .data [end] indica o início da zona designada por segmentos de dados. Se não especificarmos o endereço [end] então o SPIM utilizará, por omissão, o endereço 0x10010000. (faça o teste!)

DECLARAÇÃO DE BYTES EM MEMÓRIA

- A diretiva .Byte valor inicializa uma posição de memória da dimensão de um byte, contendo o valor.

Crie um ficheiro com o seguinte código:

```
        .data
octal: .byte 0x10
```

Apague todos os registos e a memória e carregue o novo programa.

❖ Que endereço de memória foi inicializado com o conteúdo especificado?

❖ Que valor foi armazenado na palavra que contém o byte?

DECLARAÇÃO DE CADEIAS DE CARATERES

A diretiva . ascii (cadeia) permite carregar, em posições de memória consecutivas (cada uma com a dimensão de 1 byte), o código ASCII de cada um dos caracteres da cadeia. Crie o seguinte código:

```
        .data
cadeia: .ascii “abcde”
octal:  .byte 0xff
```

- Localize na cadeia de memória.

- Altere a diretiva .ascii para .asciiz e diga o que acontece. O que faz esta diretiva?

RESERVA DE ESPAÇO EM MEMÓRIA

A diretiva .space n serve para reservar espaço para uma variável em memória, incializando-a com o valor 0. Crie o seguinte código:

```
            .data
palavra1:   .word 0x20
espaço:     .space 8
palavra2:   .word 0x30
```

- Que intervalo de posições foram reservadas para a variável espaço? Quantos bytes foram reservados no total? E quantas palavras?

ALINHAMENTO DOS DADOS EM MEMÓRIA

A diretiva .align n alinha os dados seguintes a um endereço múltiplo de 2^n . Crie o seguinte código:

```
            .data
byte1:      .byte 0x10
espaço:     .space 4
byte2:      .byte 0x20
pal:        .word 10
```

- Que intervalo de posições de memória foram reservadas para a variável espaço? A partir de que endereços se inicializaram byte1 e byte 2 ? E a partir de que endereço se inicializou pal? Porquê? Faça agora a seguinte alteração:

```
.data
byte1:      .byte 0x10
            align 2
espaço:     .space 4
byte2:      .byte 0x20
pal:        .word 10
```

- Responda novamente à questão anterior. O que fez a diretiva .align?

CARREGAMENTO E ARMAZENAMENTO DOS DADOS

- O carregamento dos dados consiste no transporte dos dados da memória para os registos.
- O armazenamento consiste no transporte dos dados dos registos para a memória.
- No MIPS R2000, as instruções de carregamento começam com a letra “l” (do inglês *load*) e as de armazenamento com a letra “s” (do inglês *store*), seguidas da letra inicial correspondente ao tamanho do dado que se vai mover: “b” para *byte*, “h” para *half-word* e “w” para *word*.

CARREGAMENTO E ARMAZENAMENTO DOS DADOS

- CARREGAMENTO DE DADOS IMEDIATOS (CONSTANTES)

Crie o seguinte programa:

```
        .text
main:   lui $s0, 0x4532
```

Consulte o manual para descobrir o que faz a instrução lui. Carregue o programa e anote o conteúdo do registo $s0. Qual a dimensão total (em bits) dos registos?

Agora faça a seguinte alteração:

```
        .text
main:   lui $s0, 0z98765432
```

Apague os valores da memória e carregue o novo programa. Verifique novamente o estado do registo $s0.

CARREGAMENTO E ARMAZENAMENTO DOS DADOS

➢ CARREGAMENTO DE PALAVRAS (MEMÓRIA -> REGISTO)

Como transferir uma palavra (32 bits) da memória para um registo ($s0). A instrução lw carrega para um registo (primeiro argumento desta instrução) a palavra contida numa posição de memória cujo endereço é especificado no segundo argumento desta instrução. Este endereço é obtido somando o conteúdo do registo com o de um identificador, como se vê neste código que deverá carregar para o simulador:

```
        .data
pal:    .word 0x10203040
        .texto
main:   lw $s0, pal ($0)
```

- ❖ Reinicialize a memória e os registos do simulador e carregue este programa. Verifique o conteúdo do registo $s0. Localiza a instrução lw $s0, pal($0) na memória e repare como o simulador traduziu esta instrução.
- ❖ Faça a seguinte modificação no código: em vez de transferir a palavra contida no endereço de memória referenciado pela etiqueta pal, transfira a palavra contida no endereço de memória referenciado por pal +1. Explique o que acontece e porquê.

CARREGAMENTO E ARMAZENAMENTO DOS DADOS

➢ CARREGAMENTO DE BYTES (MEMÓRIA -> REGISTO)

Como transferir uma palavra (32 bits) da memória para um registo ($s0). A instrução lw carrega para um registo (primeiro argumento desta instrução) a palavra contida numa posição de memória cujo endereço é especificado no segundo argumento desta instrução. Este endereço é obtido somando o conteúdo do registo com o de um identificador, como se vê neste código que deverá carregar para o simulador:

```
        .data
oct:    .byte 0xf3
seg:    .byte 0x20
        .text
main:   lb $s0, oct($0)
```

- ❖ Que instrução lb carrega um byte a partir do endereço de memória para um registo. O endereço byte obtém-se somando o conteúdo do registo $0 (sempre igual a zero) e o identificador oct. Localiza a instrução na zona de memória das instruções e indique como é que o simulador transforma esta instrução.
- ❖ Substitua lb por lbu. O que acontece quando executamos o programa?

CARREGAMENTO E ARMAZENAMENTO DOS DADOS

➢ ARMAZENAMENTO DE PALAVRAS (REGISTO -> MEMÓRIA)

Crie o seguinte programa:

```
        .data
pal1:   .word 0x10203040
pal2:    .space 4
Pal3:   .word 0xffffffff
        .text
main:   lw $s0, pal1($0)
        sw $s0, pal2($0)
        sw $s0, pal3($0)
```

A instrução sw armazena a palavra contida num registo para uma posição de memória. O endereço dessa posição é, tal como anteriormente, obtido somando o conteúdo de um registo com o deslocamento especificado na instrução (identificador).

❖ Verifique o que faz o programa.

CARREGAMENTO E ARMAZENAMENTO DOS DADOS

- ARMAZENAMENTO DE PALAVRAS (REGISTO -> MEMÓRIA)

Altere o programa anterior para

```
        .data
pal1:   .word 0x10203040
pal2:    .space 4
Pal3:   .word 0xffffffff
        .text
main:   lw $s0, pal1
        sw $s0, pal2
        sw $s0, pal3+0
```

- Que diferenças existem entre as duas versões?

CARREGAMENTO E ARMAZENAMENTO DOS DADOS

- ARMAZENAMENTO DE BYTES (REGISTO -> MEMÓRIA)

Crie o seguinte programa:

```
        .data
pal:    .word 0x10203040
oct:    .space 2
        .text
main:   lw $s0, pal($0)
        sb $s0, oct($0)
```

A instrução sb armazena o byte de menor peso de um registo numa posição de memória. Apague os registos e a memória e carregue o código. Comprove o efeito deste programa.

CONTEXTO

OBJETIVO

Familiarização com o SPIM, aprendizagem da estrutura de um programa em linguagem Assembly, utilização de ferramentas de depuração do simulador, gestão de dados em memória. Operações aritméticas e lógicas e avaliação de condições.

OVERVIEW

1	Introdução	Introdução à Linguagem Assembly; Estrutura dos programas em Assembly;
2	SPIM	Introdução ao QTSpim
3	Gestão de Dados em memória	Declaração de dados em memória; Declaração de bytes em memória, Declaração de cadeias em caracteres; Reserva de espaço em memória
4	Operações aritméticas e lógicas e avaliação de condições	Operações aritméticas com constantes e com dados em memória. Multiplicação, divisão e operações lógicas. Operadores de rotação. Avaliação de condições simples e compostas por operadores lógicos. Definição de uma notação uniforme para a escrita de fluxogramas. Criação de fluxogramas a partir de exemplos de programas.

OPERAÇÕES ARITMÉTICAS

- As **operações aritméticas** são constituídas por operações de soma e subtração (add, addu, addi, addiu, sub e subu) e por operações de multiplicação e divisão (mul, multu, div, divu).
- As instruções que terminam em “u” causam uma exceção de overflow no caso do resultado da operação não ser de dimensão acomodável em 32 bits (dimensão dos registos).
- As instruções or (ou ori), and (ou andi) e xor (ou xori) permitem realizar as conhecidas operações lógicas com o mesmo nome. As instruções de rotação (aritméticas / lógicas), também conhecidas por shift são: sra, sll e srl.

OPERAÇÕES ARITMÉTICAS

Crie um ficheiro com o seguinte programa:

```
            .data
numero:     .word 2147483647
            .text
main:       lw $t0, numero($0)
            addiu $t1, $t0, 1
```

- Carregue o programa no QTSPIm e comprove o que acontece.

- Altere a instrução addiu para addi. O que acontece?

OPERAÇÕES ARITMÉTICAS COM DADOS DE MEMÓRIA

Crie um ficheiro com o seguinte programa:

```
        .data
num1:   .word 0x80000000
num2:   .word 1
num3:   .word1
        .text
main:   lw $t0, num1($0)
        lw $t1, num2($0)
        subu $t0, $t0, $t1
        lw $t1, num3($0)
        subu $t0, $t0, $t1
        sw $t0, num1($0)
```

- O que faz este programa? O resultado que fica armazenado no endereço num3 é o resultado correto? Porquê?

- O que acontece quando alteramos as instruções subu por sub?

OPERAÇÕES LÓGICAS

As operações lógicas de deslocamento (shifts) permitem deslocar bits dentro de uma palavra, "shift left" para a esquerda e "shift right" para a direita.
Outra classe de operações lógicas são o AND, OR e XOR, que operam entre duas palavras, numa correspondência unívoca, e operam bit a bit.

- Por exemplo, para colocar o bit 2 de uma palavra de 32 bits a 1:

```
addi  $t0, $0, 0x000055aa      # t0 = 0000 0000 0000 0101 0101 1010 1010
```

Faz-se um OR com umas máscara 0x00000004:

```
move $t1, 0x00000004      #t1= 0000 0000 0000 0000 0000 0000 0000 0100
or $t2, $t0, $t1          #t2 = 0000 0000 0000 0000 0101 0101 1010 1110
```

OPERAÇÕES LÓGICAS

As operações lógicas de deslocamento (shifts) permitem deslocar bits dentro de uma palavra, "shift left" para a esquerda e "shift right" para a direita.
Outra classe de operações lógicas são o AND, OR e XOR, que operam entre duas palavras, numa correspondência unívoca, e operam bit a bit.

- Por outro lado, para colocar o bit 5 de uma palavra de 32 bits a 0:

```
addi  $t0, $0, 0x000055aa      # t0 = 0000 0000 0000 0000 0101 0101 1010 1010
```

Faz-se um AND com umas máscara 0xffffffdf:

```
move $t1, 0xffffffdf      #t1= 1111 1111 1111 1111 1111 1111 1101 1111
and $t2, $t0, $t1
```

AVALIAÇÃO DE CONDIÇÕES SIMPLES

A linguagem em Assembly não possui qualquer estrutura de controlo do fluxo do programa que permita decidir sobre dois ou mais caminhos de execução distintos. Para implementar uma estrutura deste tipo é necessário avaliar previamente uma condição, simples ou composta. O caminho de execução que o programa segue dependerá então desta avaliação.

- O seguinte programa compara as variáveis var1 e var2 e deixa o resultado na variável (booleana) res:

```
        .data
var1:   .word 30
var2:   .word 40
res:    .space 1
        .text
main:   lw $t0, var1 ($0)      # carrega var1 em t0
        lw $t1, var2($0)       # carrega var2 em t1
        slt $t2, $t0, $t1      # coloca t2 a 1 se t0<t1
        sb $t2, res($0)        # armazena t2 em res
```

- Verifique que valor se armazena na posição de memória res.

AVALIAÇÃO DE CONDIÇÕES COMPOSTAS POR OPERADORES LÓGICOS

- Crie o seguinte programa:

```
            .data
var1:       .word 40
var2:       .word -50
res:        .space 1
            .text
main:       lw $t8, var1 ($0)
            lw $t9, var2($0)
            and $t0, $t0, $0
            and $t1, $t1, $0
            beq $t8, $0, igual
            ori $t0, $0, 1
            igual: beq $t9, $0, fim
            ori $t1, $0, 1
            fim: and $t0, $t0, $t1
            sb $t0, res($0)
```

AVALIAÇÃO DE CONDIÇÕES COMPOSTAS POR OPERADORES LÓGICOS

Apague a memória e os registos e carregue e execute este programa.

- Qual o valor que fica armazenado na posição de memória res? Desenhe um fluxograma descrevendo este programa.
- Responda à questão anterior inicializando var1 e var2 com os valores 0 e 20, respetivamente
- Faça o mesmo mas agora inicializando var1 e var2 com os valores 20 e 0, respetivamente.
- Que comparação composta se realizou entre var1 e var2?

Autora: Sara Fernandes
sara.fernandes@dsi.uminho.pt