Universidade Federal do Espírito Santo

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Arquitetura de Computadores I – Turmas 01 e 02 (EARTE) – 2021/2
Prof. Rodolfo da Silva Villaça – rodolfo.villaca@ufes.br

Laboratório I – Iniciando com o MARS e a Linguagem Assembly MIPS[1]

Integrantes do Grupo:
- Dionatas Brito
- Maria Julia Damasceno
- Otávio Sales

Objetivos:
- Carregar e executar programas em assembly MIPS;
- Examinar posições de memória;
- Examinar registradores;
- Executar programas passo-a-passo.

Descrição:

O MARS[2] é um simulador escrito em Java que roda programas para as arquiteturas baseadas em MIPS. O simulador pode carregar e executar programas em assembly MIPS. O processo de translação do código MIPS é transparente para o usuário. Isto significa que você não tem que tratar com o assembler, linker e loader diretamente. Após você escrever seu programa em assembly, o simulador vai se encarregar de executá-lo e apresentar o resultado da execução em uma das saídas disponíveis (em geral, a saída padrão definida em uma das janelas do MARS).

Atividades:

1. Primeiros passos com o MARS

Usando a janela de edição do MARS, edite o programa p1.asm a seguir. No editor, o caractere (#) marca o início de um comentário; um nome seguido (:) é um label (identificador) e os nomes que se iniciam com (.) são diretivas para o montador (assembler). p1.asm:

```
.data
msg1: .asciiz "Digite um valor inteiro: "

.text
# print message on shell
li $s0, 0x00400000 # save return adress in $s0
li $v0, 4 # system call for print_str
la $a0, msg1 # address of string to print
syscall

# now get an integer from the user
li $v0, 5 # system call for read_int
syscall # the integer is placed in $v0

# do some computation here with the number
addu $t0, $v0, $0 # move the number to $t0
sll $t0, $t0, <digit> # change <digit> with the last digit of your UFES id
(matricula)  # print the result in shell
li $v0, 1 # system call for print_int
```

1 Material adaptado das apostilas do curso do Prof. Virgil Bistriceanu
Disponível em: www.cs.iit.edu/~virgil/cs470/Labs/
2 Disponível em: http://courses.missouristate.edu/kenvollmar/mars/download.htm

Departamento de Informática (DI) / Centro Tecnológico (CT)
Av. Fernando Ferrari, 514, Campus de Goiabeiras, CEP: 29.075-910, Vitória/ES

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

```
addu $a0, $t0, $0 # move number to be printed in $a0
syscall

# restore now the return address in $ra and return from main
addu $ra, $0, $s0 # return address back in $ra
jr $ra # return to the main label
```

Monte o programa usando o menu Run → Assemble, ou F3 e execute o programa. Não se esqueça de trocar o dígito da linha indicada para o seu último dígito da sua matrícula. Observe o resultado para diferentes números digitados como entrada. Preencha a tabela a seguir com diversos valores entrada e de saída (inteiros, positivos e negativos) para a execução do programa.

[Iremos usar o último número de matrícula, igual a 7]

Entrada	Saída
5	640
3	384
10	1280
-10	-1280
-20	-2560

Responda:

• Qual é a equação que define a transformação dos valores de entrada para saída?

Resposta:

$$2^{Nufes} * valorDeEntrada$$

*Nessa linha **"sll $t0, $t0, 7" ,** como o número escolhido (último número da matrícula de um integrante do grupo) foi 7, o cálculo será feito da seguinte forma:*

*Se o valor de entrada for 5 (101 em binário), como essa instrução é um "**Shift left logical (sll)** ", irá deslocar 7 unidades à esquerda, se tornando 1010000000, que é igual a 640 em decimal.*

• Qual o valor armazenado em $s0 e por que ele foi passado a $ra?

Resposta:

$t7	15	0
$s0	16	4194304
$s1	17	0
$s2	18	0

O valor contido em $s0 é igual a 4194304, o motivo vem da linha **"addu $ ra, $ 0, $ s0 ",** onde a instrução restaura o endereço de retorno em "**$ra**" , então o conteúdo que está contido em "**$s0**" é armazenado no registrador "**$ra**".

$s0	16	4194304
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	4194304
pc		4194360

2. Usando o MARS para entender a arquitetura

Usando o MARS, é possível observar a ocupação da memória e dos registradores, assim como o endereçamento de instruções e dados de um programa MIPS. Também é possível fazer a execução de um programa passo a passo, permitindo a observação detalhada da variação dos valores dos registradores e do fluxo de execução dos programas.

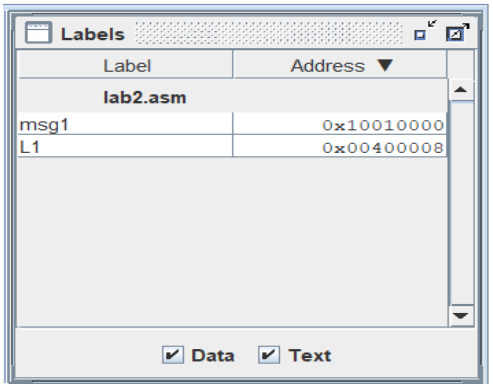
Modifique o programa p1.asm, incluindo um label L1: na 2ª linha do segmento de texto, que

passará a ser: L1: li $v0, 4 # system call for print_str

Observe a tabela de símbolos do programa. Dois labels devem aparecer (L1 e msg1). Anote os valores desses labels (eles estão em hexadecimal, prefixados sempre com 0x) e explique como esses valores foram definidos.

Label	Valor	Justificativa
msg1	**0x10010000**	*msg1 foi separado como segmento de dados do programa*
L1	**0x00400008**	*L1 foi separado como segmento de texto/instruções*

Resposta:



Neste ponto, você já deverá entender como um programa é armazenado, como os valores dos labels são definidos e como os valores dos registradores são modificados. Importantes questões podem ser respondidas agora:

• Qual o tamanho das instruções do programa? Quantos bytes no total ele ocupa?

Resposta:

Uma instrução tem "32" bits, como o programa tem "10" instruções no total (considerando as modificações feitas até aqui), o tamanho das instruções do programa é de 320 bits.
Uma instrução possui "4" bytes, como o programa tem "10" instruções no total (considerando as modificações feitas até aqui), o tamanho das instruções do programa é de "40" bytes.

• A instrução li $s0, 0x00400000 tem qual propósito? E se esse valor fosse mudado, o que aconteceria?

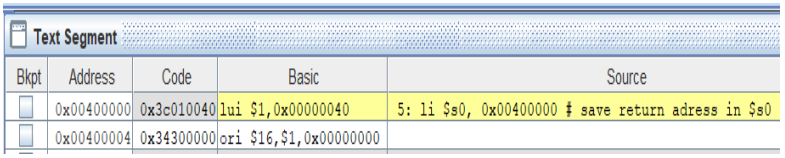
Resposta:

Tem o propósito de carregar 0x00400000 de forma imediata no registrador $s0.
Se esse valor fosse mudado, iria mudar o endereço de memória.

• Essa instrução foi substituída por duas instruções. Quais? Por que?

Resposta:

*Foi substituída por **"lui"** e **"ori".***



Porque a constante excede o valor de 16 bits, então o lui irá colocar os 16 bits mais significativos e o ori, os 16 bits menos significativos, passando assim a constante para os registradores de 32 bits.

Departamento de Informática (DI) / Centro Tecnológico (CT)
Av. Fernando Ferrari, 514, Campus de Goiabeiras, CEP: 29.075-910, Vitória/ES

Universidade Federal do Espírito Santo

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

• A instrução la $a0, msg1 também foi substituída por 2 instruções. Quais? Por que?

Resposta:

*"**lui**" e "**ori**", pelo mesmo motivo da questão anterior, como a constante excede o valor de 16 bits, usando lui e ori, irá conseguir passar a constante para registradores de 32 bits.*

É possível fazer a execução passo-a-passo do programa usando o botão correspondente na interface ou a tecla F7. Execute o programa passo-a-passo e observe a mudança nos valores do registrador PC, assim como a instrução executada no momento, além das mudanças nos valores dos registradores utilizados no código.

Em 3 pontos do programa a instrução syscall foi utilizada. Note que essa instrução faz a chamada de um serviço (I/O) e que alguns argumentos devem ser passados antes da chamada. Anote na tabela a seguir os registradores que são alterados em cada uma das instruções syscall do programa, assim como os valores dos argumentos usados como entrada e saída nas chamadas.

Endereço	Tipo de Serviço	Entrada (registrador)	Saída (registrador)
0x00400014	*system call for print_str*	*Não possui*	*Endereço para a string (msgl)*

0x0040001C	*system call for read_int*	*Número inteiro*	*Não possui*
0x00400030	*system call for print_int*	*Não possui*	`Resultado da` **Shift left logical** ($2^{Nufes} * valorDeEntrada$)

Resposta:

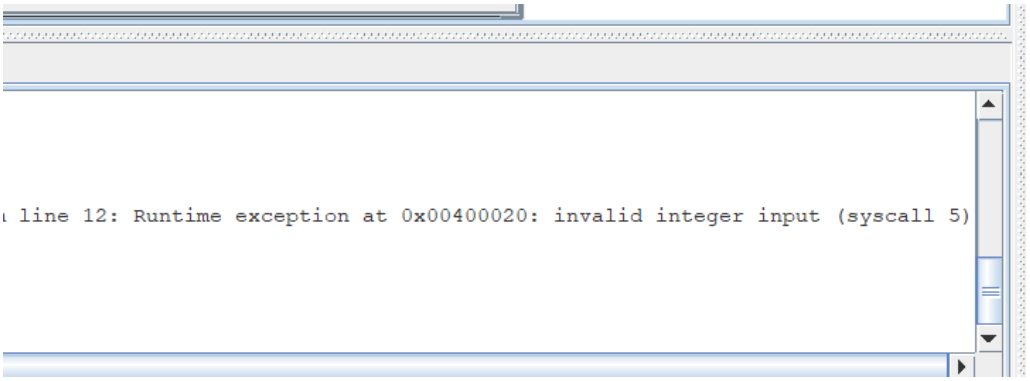
Por opção de projeto, a CPU MIPS foi projetada com:
- Uma unidade de inteiros (ULA dentro da CPU);
- Um coprocessador 0, com suporte ao tratamento de interrupções, exceções e memória virtual; • Um coprocessador 1, correspondendo a uma unidade de operações em ponto flutuante.

Estes tópicos ainda não foram vistos no curso, porém os coprocessadores já podem ser testados:

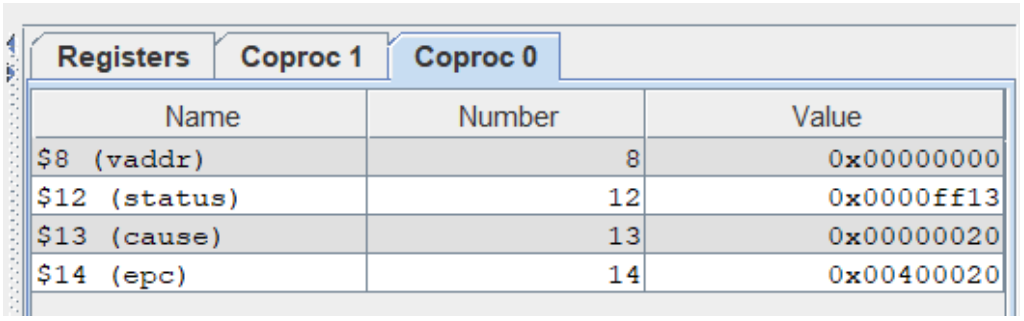
• Escolha a aba Coproc 0 na janela direita do simulador. Digite um valor de entrada “muito grande” e observe o erro de execução e sua indicação em um dos registradores. Qual registrador indica o erro? Qual o erro foi indicado? Como essa indicação poderia ser usada pelo SO?

Resposta:

line 12: Runtime exception at 0x00400020: invalid integer input (syscall 5)



Registrador $14 (epc), poderia ser usada para limitar o inteiro a ser inserido, pois após um certo valor de inteiro o erro acontece;

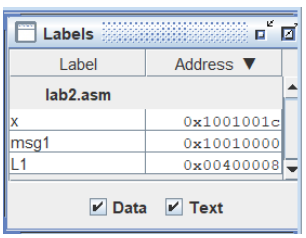


Name	Number	Value
$8 (vaddr)	8	0x00000000
$12 (status)	12	0x0000ff13
$13 (cause)	13	0x00000020
$14 (epc)	14	0x00400020

• Inclua uma declaração da variável x no segmento de dados com o comando x: .float 1.5674. Observe a tabela de labels do programa. O valor da variável x está correto? Que representação é essa?

Resposta:

Não está correto, o que realmente mostra é o endereço de memória da variável x e não o valor contido nela.



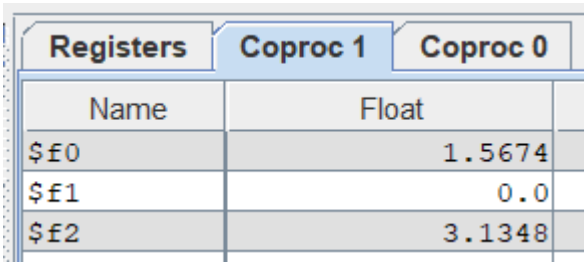
• Inclua as instruções a seguir no segmento de código após o último syscall:

l.s $f0, x
add.s $f2, $f0, $f0

• Escolha a aba Coproc 1 na janela direita do simulador. O que representam os valores que foram gerados nos registradores do Coproc 1?

Resposta:

*Representa o valor de x correto (em "**l.s $f0, x**") e o segmento de código "**add.s $f2, $f0**", representa o dobro de x.*



Código Fonte:

C:\Users\diona\OneDrive\Área de Trabalho\ufes\Arquitetura de Computadores (ArqComp)\Laboratórios\Lab2\lab2.as

File Edit Run Settings Tools Help

Edit Execute

mips1.asm lab2.asm

```
1  .data
2  msg1: .asciiz "Digite um valor inteiro: "
3  x: .float 1.5674
4  .text
5  # print message on shell
6  li $s0, 0x00400000 # save return adress in $s0
7  L1: li $v0, 4 # system call for print_str
8  li $v0, 4 # system call for print_str
9  la $a0, msg1 # address of string to print
10 syscall
11 # now get an integer from the user
12 li $v0, 5 # system call for read_int
13 syscall # the integer is placed in $v0
14 # do some computation here with the number
15 addu $t0, $v0, $0 # move the number to $t0
16 sll $t0, $t0,7 # change <digit> with the last digit of your UFES id (matricula)
17 # print the result in shell
18 li $v0, 1 # system call for print_int
19 addu $a0, $t0, $0 # move number to be printed in $a0
20 syscall
21 l.s $f0, x
22 add.s $f2, $f0, $f0
23 # restore now the return address in $ra and return from main
24 addu $ra, $0, $s0 # return address back in $ra
25 jr $ra # return to the main label
```

4. Execução

- Grupos de até 3 (três) alunos;
- Submissão até 02/12 (9h);

Departamento de Informática (DI) / Centro Tecnológico (CT)
Av. Fernando Ferrari, 514, Campus de Goiabeiras, CEP: 29.075-910, Vitória/ES