Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Organização de Computadores I (DCC006) - 2018-1

Trabalho Prático 1 - Simulador MIPS

1. Descrição do Trabalho

Este trabalho consiste na implementação de um simulador para uma máquina MIPS modificada. O simulador deve executar todas as instruções apresentadas no fim deste documento, incluindo algumas instruções específicas que não existem no conjunto de instruções MIPS.

2. Especificações do Simulador

O simulador irá receber como entrada um arquivo **binário**, que é um programa executável constituído das instruções especificadas. O programa deve ser executado de forma sequencial, levando em consideração as instruções de desvio (branches e jumps), até o término do programa. O fim do programa é marcado pela instrução *Halt* ou quando não houver mais instruções para serem executadas.

O simulador precisa de uma área de memória para armazenar o código lido do arquivo de entrada e os dados utilizados pelo programa. Assuma as seguintes características:

- O código dos programas começa a ser armazenado no endereço **0x0** de memória e pode chegar até o endereço **0xFFF**;
- A área de memória que pode ser utilizada pelos programas (pelas instruções de load e store) começa no endereço **0x1000** e pode chegar ao endereço **0xFFFF**;
- O endereçamento de memória é feito no nível de **palavras de 4 bytes**. Isso significa que a primeira instrução do programa é armazenada no endereço 0x00, a segunda no endereço 0x01, a décima no endereço 0x0A, etc. As instruções de load (Lw) e store (Sw) também seguem a mesma regra de endereçamento.

Tanto os valores imediatos numéricos utilizados pelas instruções quanto os registradores do simulador armazenam **inteiros de 4 bytes com sinal, utilizando complemento de 2**.

3. Formato das Instruções

O formato das instruções segue o padrão do MIPS. **Todas as instruções possuem tamanho fixo de 32 bits** e são divididas em três tipos: Tipo R (instruções de registradores); Tipo I (instruções com valores imediatos); e Tipo J (instruções de jump).

Formato das instruções tipo R

Opcode	Rd	Rs	Rt	Shift	Funct
6 bit	5 bits	5 bits	5 bits	5 bits	6 bits

Exemplo: add $t1, $s0, $s2

Opcode	Rd	Rs	Rt	Shift	Funct
000000	01001	10000	10010	00000	100000

Formato das instruções tipo I

Opcode	Rs	Rt	Imediato
6 bits	5 bits	5 bits	16 bits

Exemplo: addi $v0, $a2, 200

Opcode	Rs	Rt	Imediato
001000	00010	00110	0000 0000 1100 1000

Formato das instruções tipo J

Opcode	Endereço
6 bits	26 bits

Exemplo: j label (onde label indica a instrução na posição 0xFAFA da memória)

Opcode	Endereço
000010	00 0000 0000 1111 1010 1111 1010

4. Registradores

O simulador terá 32 registradores de 32 bits cada. Cada registrador tem um inteiro identificador associado (indo de 0 a 31), utilizado para identificá-los nas instruções. Todos os registradores são considerados registradores de propósito geral, **exceto**:

- O registrador **$0** possui sempre o valor **0** e não pode ser sobrescrito;
- O registrador **$31** guarda o valor de retorno de procedimentos. Esse é o registrador que deve ser utilizado pela instrução **Jal** para armazenar o ponto de retorno do procedimento.

5. Conjunto de Instruções

As tabelas a seguir contém todas as instruções que o simulador deve reconhecer, o tipo da instrução, seu formato em binário e uma descrição de mais alto nível. No campo formato, as letras **'s', 't'** e **'d'** indicam a posição onde os respectivos registradores ($s, $t e $d) são codificados na instrução, a letra **'i'** indica a posição dos valores imediatos e a letra **'x'** indica bits que podem ser ignorados para a decodificação da instrução (don't care).

Instrução	Tipo	Formato	Descrição
Add	R	0000 00ss ssst tttt dddd d000 0010 0000	$d = $s + $t
Addi	I	0010 00ss ssst tttt iiii iiii iiii iiii	$t = $s + I
And	R	0000 00ss ssst tttt dddd d000 0010 0100	$d = $s AND $t
Andi	I	0011 00ss ssst tttt iiii iiii iiii iiii	$t = $s AND I
Beq	I	0001 00ss ssst tttt iiii iiii iiii iiii	If $s == $t Jump I
Bgtz	I	0001 11ss sss0 0000 iiii iiii iiii iiii	If $s > 0 Jump I
Bltz	I	0000 01ss sss0 0000 iiii iiii iiii iiii	If $s < 0 Jump I
Bne	I	0001 01ss ssst tttt iiii iiii iiii iiii	If $s != $t Jump I
Div	R	0000 00ss ssst tttt dddd d000 0010 1010	$d = $s / $t
J	J	0000 10ii iiii iiii iiii iiii iiii iiii	Jump I
Jal	J	0000 11ii iiii iiii iiii iiii iiii iiii	Jump I, Save PC
Jr	R	0000 00ss sss0 0000 0000 0000 0000 1000	Jump $s
Lw	I	1000 11ss ssst tttt iiii iiii iiii iiii	$t = MEM[$s + I]
Mult	R	0000 00ss ssst tttt dddd d000 0001 1000	$d = $s * $t
Or	R	0000 00ss ssst tttt dddd d000 0010 0101	$d = $s OR $t
Ori	I	0011 01ss ssst tttt iiii iiii iiii iiii	$t = $s OR I

Sll	R	0000 00xx xxxt tttt dddd dhhh hh00 0000	$d = $t << H
Srl	R	0000 00xx xxxt tttt dddd dhhh hh00 0010	$d = $t >> H
Sub	R	0000 00ss ssst tttt dddd d000 0010 0010	$d = $s - $t
Sw	I	1010 11ss ssst tttt iiii iiii iiii iiii	MEM[$s + I] = $t
Xor	R	0000 00ss ssst tttt dddd d000 0010 0110	$d = $s XOR $t

Instruções Extras

Instrução	**Tipo**	**Formato**	**Descrição**
Halt	J	1111 11xx xxxx xxxx xxxx xxxx xxxx xxxx	Termina o programa
In	I	1111 10ss sssx xxxx xxxx xxxx xxxx xxxx	$s = Entrada
Out	I	1111 01ss sssx xxxx xxxx xxxx xxxx xxxx	Print $s

Obs.: (1) Ao executar a instrução **In**, o simulador deve aguardar a entrada de um valor **inteiro** pelo usuário e armazená-lo no registrador indicado. A leitura do valor deve ser feito pela entrada padrão.
(2) Ao executar a instrução **Out**, o simulador deve imprimir o valor decimal do registrador indicado, seguido de uma quebra de linha (\n). O valor deve ser impresso na saída padrão.
(3) As instruções com endereçamento de memória (branches, jumps, load e store) utilizarão o padrão de endereçamento especificado na parte dois desse documento.
(4) As instruções **Mult** e **Div** utilizam apenas um registrador para armazenar o resultado. O *overflow* (no caso do Mult) e o resto (no caso do Div) devem ser ignorados.
(5) O campo *Shift* nas instruções tipo R será utilizado apenas em instruções de shift (Sll e Srl), podendo ser ignorado nas demais instruções.
(6) Todas as instruções trabalham com valores inteiros, não há unidades de ponto flutuante nesse nosso MIPS simplificado.

6. Exemplo

Considere o programa a seguir, escrito com o conjunto de instruções apresentado anteriormente:

```
In $1             #Lê um inteiro e guarda o resultado no registrador 1
In $2             #Lê um inteiro e guarda o resultado no registrador 2
Add $3, $2, $1    #$3 = $2 + $1
Out $3            #Imprime na tela o conteúdo do registrador 3
Halt              #Termina o programa
```

Após compilado, o código binário desse programa se torna:

```
1111 1000 0010 0000 0000 0000 0000 0000
1111 1000 0100 0000 0000 0000 0000 0000
0000 0000 0100 0001 0001 1000 0010 0000
1111 0100 0110 0000 0000 0000 0000 0000
1111 1100 0000 0000 0000 0000 0000 0000
```

Esse código binário será a entrada do simulador, que deve ler os dois inteiros, somá-los e imprimir o resultado na tela. Lembre-se que o arquivo de entrada estará em formato binário, não abra o arquivo para leitura em modo texto.

7. Informações Importantes

- O caminho do arquivo binário será passado como o primeiro argumento do programa. Exemplo de execução:

./simulador arquivo_de_entrada

- O trabalho deve ser feito individualmente, podendo ser discutido entre os colegas desde que não haja cópia ou compartilhamento do código fonte.
- A data de entrega será especificada através de uma tarefa no Moodle.
- Os trabalhos poderão ser entregues até às 23:55 do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle. Haverá uma tolerância de 5 minutos de atraso, de forma que os alunos podem fazer a entrega até às 0:00. A partir desse horário, os trabalhos já estarão sujeitos a penalidades. A fórmula para desconto por atraso na entrega do trabalho prático é:

$$\text{Desconto} = (2^5 / 0.32)\,\%$$

onde d é o atraso em dias úteis. Note que após 5 dias úteis, o trabalho não pode ser mais entregue.

- O trabalho deve ser implementado obrigatoriamente na linguagem C.
- Deverá ser entregue o código fonte com os arquivos de dados necessários para a execução e um arquivo *Makefile* que permita a compilação do programa nas máquinas Linux do DCC.
- Além disso, deverá ser entregue uma pequena documentação contendo todas as decisões de projeto que foram tomadas durante a implementação, sobre aspectos não contemplados na especificação, assim como uma justificativa para essas decisões. Esse documento não precisa ser extenso. A documentação deve indicar o nome do aluno. O código fonte não deve ser incluído no arquivo PDF da documentação.

- Todas as dúvidas referentes ao trabalho serão esclarecidas por meio do fórum disponível no ambiente Moodle da disciplina.
- A entrega do trabalho deverá ser realizada pelo Moodle, na tarefa criada especificamente para tal. Deverá ser entregue um único arquivo compactado em formato ".zip" com nome "tp1_NomeDoAluno.zip", contendo todo o código fonte, o arquivo *Makefile* e a documentação em formato PDF.