UFBA - MATA48/MATC90 - Arquitetura de Computadores

Projeto 3 - Simulador MIPS em Nível de Instrução

1. Introdução

O propósito deste laboratório é fazer com que o aluno crie um programa em C que funcione como um simulador de nível de instrução para um subconjunto limitado do conjunto de instruções MIPS. Este simulador de nível de instrução irá modelar o comportamento de cada instrução e permitirá ao usuário executar programas MIPS e ver suas saídas.

O simulador irá processar um arquivo de entrada que contém um programa MIPS. Cada linha do arquivo de entrada corresponde a uma única instrução MIPS, escrita através de uma sequência de caracteres hexadecimal. Por exemplo, 2402000a é a representação hexadecimal de addiu $v0, $zero, 10. Será fornecido vários arquivos de entrada. Mas você também deve criar seus próprios arquivos de entrada com o objetivo de testar seu simulador de forma mais abrangente.

O simulador irá executar a entrada do programa uma instrução por vez. Depois de cada instrução, o simulador irá modificar o estado da arquitetura MIPS: valores armazenados nos registradores e memória. O simulador é dividido em duas seções principais: (1) shell e a (2) rotina de simulação.
Seu trabalho é implementar a rotina de simulação.

No diretório src/ existem dois arquivos (shell.c e shell.h) que já implementam o shell (Voce não precisa alterar esses arquivos!). Há um terceiro arquivo (sim.c) onde você irá implementar a rotina do simulador - este é o único arquivo que você tem permissão para alterar.

2. O Shell

O objetivo do shell é fornecer ao usuário comandos para controlar a execução do simulador. O shell aceita um ou mais arquivos de programa como argumentos através da linha de comando e os carrega para a imagem da memória. A fim de extrair informações do simulador, será criado um arquivo chamado dumpsim para conter a informação solicitada do simulador. O shell suporta os seguintes comandos:

1 - **go:** simula o programa até que ele indica que o simulador deve parar. (Como definimos abaixo, uma instrução SYSCALL é executada quando o valor em $v0 (registrador 2) é igual a 0x0A.)
2 – **run <n>:** simula a execução da máquina para n instruções.
3 - **mdump <low> <high>:** despeja o conteúdo de memória, da localização mais baixa para a localização mais alta, na tela e para o arquivo de despejo chamado (dumpsim).
4 - **rdump:** despeja o valor atual do contador de instruções, os conteúdos de R0 – R31, e do PC para a tela e para o arquivo de despejo (dumpsim).
5 - **input reg num reg val:** configura o registrador de propósito geral **num_reg** para valor de **reg_value .**
6 - **high value:** define o registrador HI com o valor.
7 - **low value:** define o registrador LO com o valor.
8 - **?:** imprimi uma lista de todos os comandos do shell.
9 - **quit:** para o shell.

3. A Rotina de Simulação

A rotina de simulação realiza a simulação a nível de instrução para o arquivo de entrada em formato MIPS. Durante a execução de uma instrução, o simulador deve levar o atual estado arquitetural e modificá-lo de acordo com a descrição ISA do Manual do usuário do MIPS R4000 (somente no modo de 32 bits), documento já fornecido no homework 1. O estado arquitetural inclui o PC, os registradores de propósito geral e a imagem da memória. O estado está contido nas seguintes variáveis globais:

```
#define MIPS_REGS 32
typedef struct CPU_State {
   uint32_t PC; //Contador do programa
   uint32_t REGS[MIPS_REGS]; // Registradores
   uint32_t HI, LO; //Multiplicador dos registradores HI e LO
} CPU_State;

CPU_State STATE_CURRENT, STATE_NEXT;
int RUN_BIT;
```

Além disso, o simulador modela a memória do sistema simulado. Você precisa usar as duas funções a seguir, que é fornecida para acessar a memória simulada:

```
uint32_t mem_read_32(uint32_t address);
void mem_write_32(uint32_t address, uint32_t value);
```

Note que em MIPS, a memória é byte-endereçável. Além disso, você deve implementar uma arquitetura little-endian. Isto significa que as palavras da máquina (32 bits) são armazenadas com o byte menos significativo no endereço mais baixo, e o byte mais significativo no endereço mais alto. Para implementar as instruções carregar(load) e armazenar(store) de valores de 16 bits e 8 bits, você precisará usar essas primitivas de acesso à memória de 32 bits (dica: certifique-se de modificar apenas a parte apropriada de uma palavra de 32 bits!).

Em particular, você deve chamar mem_read_32 e mem_write_32 com somente endereços alinhados de 32 bits (isto é, os dois bits inferiores do endereço devem ser zero).

O esqueleto do simulador fornecido inclui uma função vazia chamada process_instruction () no arquivo sim.c. Esta função é chamada pelo shell para simular uma instrução de máquina. Você tem que escrever o código para a função process_instruction () que simule a execução das instruções. Você também pode escrever funções adicionais para tornar o programa modular. (Sugere-se gastar um certo tempo para tornar seu código fácil de ler e entender, para seu próprio benefício.)

4. O que Você Deve Fazer

Seu trabalho é implementar (dar corpo) a função process_instruction () localizada no arquivo sim.c. A função process_instruction () deve ser capaz de simular a execução em nível de instrução das seguintes instruções MIPS:

J	JAL	BEQ	BNE	BLEZ	BGTZ
ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI
XORI	LUI	LB	LH	LW	LBU
LHU	SB	SH	SW	BLTZ	BGEZ
BLTZAL	BGEZAL	SLL	SRL	SRA	SLLV
SRLV	SRAV	JR	JALR	ADD	ADDU
SUB	SUBU	AND	OR	XOR	NOR
SLT	SLTU	MULT	MFHI	MFLO	MTHI
MTLO	MULTU	DIV	DIVU	SYSCALL	

Note que para a instrução SYSCALL, você só precisa implementar o seguinte comportamento: se o registrador $v0 (registrador 2) tiver o valor 0x0A (10 em decimal) quando SYSCALL é executado, então o comando **go** deve parar seu loop da simulação e retornar para o prompt shell do simulador. Se $v0 tiver qualquer outro valor, a instrução não deve ter efeito. Em ambos os casos, nenhum registrador é modificado, exceto o PC que é incrementado para a próxima instrução como de costume. A função process_instruction () que você deve implementar irá fazer com que o loop da simulação principal termine com a variável global RUN_BIT em 0.

A precisão do seu simulador é sua principal prioridade. Especificamente, verifique se o estado arquitetural é atualizado corretamente após a execução de cada instrução. Vamos testar seu simulador com muitos programas de entrada para garantir que cada instrução é simulada corretamente.

Para testar se o seu simulador está funcionando corretamente, você deve executar os programas de entrada que será fornecido e escrever um ou mais programas usando todas as instruções MIPS necessárias listadas na tabela acima, e executá-las uma instrução por vez (run 1). Você pode usar o comando rdump para verificar se o estado da máquina é atualizado corretamente após a execução de cada instrução.

Enquanto a tabela parece ter muitas instruções, há realmente apenas alguns comportamentos de instrução única com um número de variações menores. Você deve abordar as instruções em grupos: R-type ALU, I-type ALU, LW, SW, Jump, Branch, e assim por diante. O Manual do utilizador MIPS R4000 contém a definição oficial para cada instrução nesta tabela (exceto SYSCALL, para a qual foi forneceda uma definição restrita acima). Implemente apenas o comportamento de instruções com 32 bits (o R4000 também tem um modo de 64 bits, que você deve ignorar para este laboratório.)

No entanto, ao contrário do manual, será implementado a arquitetura sem "branch delay slots". Para efeitos deste laboratório, isto significa que as instruções de desvios podem atualizar NEXT_STATE e PC diretamente para o destino do desvio quando este for identificado. Além disso, as instruções "jump-and-link" (JAL, JALR, BLTZAL, BGEZAL) armazenam PC + 4 em R31, ao invés de PC + 8 conforme especificado no manual nas descrições destas instruções. Neste laboratório, você não precisa lidar com exceções de estouro (overflow) que podem ser geradas por determinadas instruções aritméticas (por exemplo, ADDI).

Finalmente, observe que seu simulador não tem que lidar com instruções que não incluímos na tabela acima, ou quaisquer outras instruções inválidas. Seu simulador será testado apenas com código válido que usa as instruções listadas acima.

5. Arquivos do Laboratório

No pacote de arquivos, você encontrará dois subdiretórios src/ e inputs/. Em src/, está o esqueleto do simulador como descrito acima. Você pode compilar o simulador com o Makefile fornecido. Em inputs/, existe alguns arquivos de entrada para você testar o seu simulador. Você deve criar mais arquivos de entrada para ter certeza de que seu simulador está correto. Também em inputs/, você pode encontrar um script responsável por montar o código MIPS no formato hexadecimal, o qual é exigido pelo seu simulador. O arquivo README descreve como montar um programa MIPS com este script e carregá-lo no simulador.

6. Entrega

Você deve enviar eletronicamente no prazo estabelecido seu código do simulador modificado (somente precisa enviar o arquivo fonte `sim.c`). Seu código deve ser legível e bem documentado.

O processo de submissão do trabalho é via Google Forms no link abaixo: