UNIVERSIDADE DE BRASÍLIA
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

116394 ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Trabalho: Simulador RISC-V

OBJETIVO

Este trabalho consiste na implementação de um simulador da arquitetura RV32I em linguagem de alto nível (C++). As funções básicas de busca e decodificação de instruções são fornecidas. Deve-se implementar a função de execução (*execute()*) das instruções para o subconjunto de instruções indicado. O programa binário a ser executado deve ser gerado a partir do montador RARS, juntamente com os respectivos dados. O simulador deve ler arquivos binários contendo o segmento de código e o segmento de dados para sua memória e executá-lo.

DESCRIÇÃO

Geração dos arquivos

As instruções e dados de um programa RV32I para este trabalho devem vir necessariamente de arquivos montados pelo RARS.

Montagem do programa

Antes de montar o programa deve-se configurar o RARS através da opção:

Settings->Memory Configuration, opção Compact, Text at Address 0

Ao montar o programa (F3), o RARS exibe na aba "Execute" os segmentos *Text* e *Data*. O segmento de código (*Text*) de um programa começa no endereço `0x00000000` de memória. O segmento de dados começa na posição `0x00002000`.

O armazenamento destas informações em arquivo é obtido com a opção:

File -> Dump Memory...

As opções de salvamento devem ser:

Código:

.text (0x00000000 - *fim-código*)

Dump Format: binary

Dados:

.data (0x00002000 - *fim-dados*)

Dump Format: binary

Gere os arquivos com nomes text.bin e data.bin.

Leitura do código e dos dados

O código e os dados contidos nos arquivos devem ser lidos para a memória do simulador. A função *load_mem()* realiza essa tarefa.

A memória é modelada como um arranjo de inteiros:

```
#define MEM_SIZE 4096
int32_t mem[MEM_SIZE];
```

Ou seja, a memória é um arranjo de 8KWords, ou 32KBytes.

Acesso à Memória

Reutilizar as funções desenvolvidas no trabalho anterior adaptadas ao contexto do RISC-V:

```
int32_t lb(uint32_t address, int32_t kte);
int32_t lw(uint32_t address, int32_t kte);
int32_t lbu(uint32_t address, int32_t kte);
void sb(uint32_t address, int32_t kte, int8_t dado);
void sw(uint32_t address, int32_t kte, int32_t dado);
```

Os endereços são todos de *byte*. A operação de leitura de *byte* retorna um inteiro com o *byte* lido na posição menos significativa. A escrita de um *byte* deve colocá-lo na posição correta dentro da palavra de memória.

Registradores

Os registradores *pc, sp, gp* e *ri*, e também os campos da instrução (*opcode*, *rs*, *rt*, *rd*, *shamt*, *funct*) são definidos como variáveis globais. *pc* e *ri* são do tipo *unsigned int (uint32_t)*, visto que não armazenam dados, apenas instruções, assim como *sp e gp*, que armazenam endereços de memória - nunca são negativos.

Valores iniciais dos registradores: (modelo compacto)

- `pc = 0x00000000`
- `ri = 0x00000000`
- `sp = 0x00003ffc`
- `gp = 0x00001800`

obs: sp é necessário para funções recursivas, iniciado no final da memória de 8KB.

Função fetch(): busca a instrução a ser executada da memória e atualiza o *pc*.

Função decode()

Extrai todos os campos da instrução_

- *opcode*: código da operação
- *rs1*: índice do primeiro registrador fonte
- *rs2*: índice do segundo registrador fonte
- *rd*: índice do registrador destino, que recebe o resultado da operação
- *shamt*: quantidade de deslocamento em instruções *shift* e *rotate*

- *funct3:* código auxiliar de 3 bits para determinar a instrução a ser executada
- *funct7:* código auxiliar de 7 bits para determinar a instrução a ser executada
- *imm12_i*: constante de 12 bits, valor imediato em instruções tipo I
- *imm12_s*: constante de 12 bits, valor imediato em instruções tipo S
- *imm13*: constante de 13 bits, valor imediato em instruções tipo SB, bit 0 é sempre 0
- *imm20_u*: constante de 20 bits mais significativos, 31 a 12
- *imm21*: constante de 21 bits para saltos relativos, bit 0 é sempre 0

Todos os valores imediatos tem o sinal estendido.

Função execute()

A função `void execute()` executa a instrução que foi lida pela função `fetch()` e decodificada por `decode()`.

Função step()

A função `step()` executa uma instrução do RV32I:

`step() => fecth(), decode(), execute()`

Função run()

A função `run()` executa o programa até encontrar uma chamada de sistema para encerramento, ou até o *pc* ultrapassar o limite do segmento de código (2k *words*).

Função dump_mem(int start, int end, char format)

Imprime o conteúdo da memória a partir do endereço *start* até o endereço *end*. O formato pode ser em hexa (‘h’) ou decimal (‘d’).

Função dump_reg(char format)

Imprime o conteúdo dos registradores do RV32I, incluindo o banco de registradores e os registradores *pc*, *hi* e *lo*. O formato pode ser em hexa (‘h’) ou decimal (‘d’).

Instruções a serem implementadas:

add	addi	and	andi	auipc
beq	bne	bge	bgeu	blt
bltu	jal	jalr	lb	or
lbu	lw	lui	nop	sltu
ori	sb	slli	slt	srai
srli	sub	sw	xor	ecall

Obs: *nop* é uma instrução nula. A instrução tem todos os bits em zero. Ela não faz nada, é usada em casos especiais, como conflitos de dependência de dados.

Syscall: implementar as chamadas para (ver *help* do RARS)

- imprimir inteiro
- imprimir string
- encerrar programa

Verificação do Simulador

Para verificar se o simulador está funcionando corretamente deve-se utilizar o RARS para geração de códigos de teste, que incluam código executável e dados. Os testes devem verificar todas as instruções implementadas no simulador.

Atentar para uso de pseudo-instruções. No RARS, elas são traduzidas para instruções nativas do RISC-V. Se utilizar pseudo-instruções, verificar se, depois da montagem, o RARS gera instruções aceitas pelo simulador.

Serão fornecidos códigos de teste para esta tarefa.

Entrega

Entregar:
- Relatório da implementação:
    - Apresentação do problema
    - Descrição das instruções implementadas
    - Testes e resultados
- O código fonte do simulador, com a indicação da plataforma utilizada:
    - Qual compilador empregado
    - Sistema operacional
    - IDE (Eclipse, XCode, etc)

Entregar no Moodle em um arquivo compactado, com o número de matrícula do aluno para identificar o arquivo.