Universidade de Brasília

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Modelagem de Sistemas em Silício

Relatório
Trabalho IV: Modelagem RISC16

Aluno:
Lucas Santos - 14/0151010

Professor:
Ricardo Jacobi

14 de maio de 2017

1 Descrição textual da implementação

1.1 Modelo com *threads*

A implementação utilizando *threads* possui 4 *threads* principais sendo elas:

- **Start**: Inicializa o funcionamento do RISC16 por meio de um evento que aciona o *fetch* (execute_ev);
- **Fetch**: Espera um evento de acionamento (execute_ev) e realiza a busca de uma instrução na memória, incrementando o contador de programa. Ao fim da busca aciona o evento que dispara o *decode* (fetch_ev);
- **Decode**: Espera um evento de acionamento (fetch_ev) e realiza a decodificação da instrução para a execução. Ao fim dispara o *execute* por meio de um evento (decode_ev);
- **Execute**: Espera um evento de acionamento (decode_ev) e realiza a execução da instrução. Ao fim dispara o *fetch* novamente por meio de um evento (execute_ev).

O código correspondente a esta implementação se encontra nos arquivos *risc16.h*, *risc16.cpp* e *main.cpp*.

1.2 Modelo com módulos interligados por filas bloqueantes

A implementação utilizando *módulos interligados por filas bloqueantes* possui 7 módulos (*struct*, no caso do *contexto*) principais sendo eles:

- **Mem**: É a memória do RISC16, implementa a interface *mem_if.h*, que tem funções de leitura, escrita e impressão da memória. A definição do tamanho da mesma é feita na instanciação.
- **Breg**: Representa o banco de registradores do RISC16, implementa a interface *breg_if.h*, que tem funções de leitura, escrita e impressão do banco de registradores, possui 17 registradores como especificado.
- **Contexto**: É uma *struct* que possui as principais informações para o funcionamento do RISC16, como valor do *pc*, *ri*, *regs*, *regs2*, *regd*, *op*, *const4*, *const8*. O contexto é passado pelos módulos por meio das filas bloqueantes.
- **Fetch**: Representa a fase de busca da instrução, possui: Uma porta que implementa a interface *mem_if.h* conectada a memória do RISC16 que funciona como um ponteiro que aponta para a mesma; Um ponteiro que aponta para o contexto da instrução atual; uma entrada e uma saída de filas bloqueantes para comunicação com os outros módulos.
- **Decode**: Representa a fase de decodificação da instrução, possui: Um ponteiro que aponta para o contexto da instrução; Uma entrada e uma saída de filas bloqueantes para comunicação com os outros módulos.
- **Execute**: Representa a fase de busca da instrução, possui: Uma porta que implementa a interface *mem_if.h* conectada a memória do RISC16 que funciona como um ponteiro que aponta para a mesma; Uma porta que implementa a interface *breg_if.h* conectada ao banco de registradores do RISC16 que funciona como um ponteiro que aponta para o mesmo; Um ponteiro que aponta para o contexto da instrução; Uma entrada e uma saída de filas bloqueantes para comunicação com os outros módulos.
- **Top**: Realiza as interconexãos entre os módulos descritos acima por meio de filas bloqueantes, ou por meio de interfaces.

O código correspondente a esta implementação se encontra nos arquivos *mem_if.h*, *mem.h*, *mem.cpp*, *breg_if.h*, *breg.h*, *breg.cpp*, *risc16.c*, *contexto.h*, *fetch.h*, *fetch.cpp*, *decode.h*, *decode.cpp*, *execute.h*, *execute.cpp*, *top.h*, *main.cpp*.

2 Descrição da verificação do funcionamento

A verificação do funcionamento é dada por meio dos seguintes testes, iguais para ambos os modelos implementados:

2.1 Testes Aritméticos

- addi \$1, 0 $\Rightarrow reg1+=0$
- addi \$1, 8 $\Rightarrow reg1+=8$
- addi \$2, -12 $\Rightarrow reg1-=12$
- add \$3, \$2, \$1 $\Rightarrow reg3 = reg2 + reg1$
- sub \$4, \$2, \$3 $\Rightarrow reg4 = reg2 - reg3$
- add \$5, \$0, \$1 $\Rightarrow reg5 = reg0 + reg1$
- shift \$5, 2 $\Rightarrow reg5 >> 2$
- add \$6, \$0, \$1 $\Rightarrow reg6 = reg0 + reg1$
- shift \$6, -4 $\Rightarrow reg6 << 4$

2.2 Testes Lógicos

- and \$8, \$7, \$4 $\Rightarrow reg8 = reg7 and reg4$
- not \$9 $\Rightarrow reg9 = not(reg9)$
- xor \$10, \$4, \$7 $\Rightarrow reg10 = reg4 xor reg7$
- slt \$11, \$5, \$1 $\Rightarrow reg11 = reg5 < reg1?1 : 0$

2.3 Testes de Transferência

- lui \$7, 0xFF $\Rightarrow reg7 = const8 << 8$
- sw \$5, \$0, \$6 $\Rightarrow$ salva o que está em \$5 no endereço que está em \$6 da memória
- lw \$12, \$0, \$6 $\Rightarrow$ carrega em \$12 o que está no endereço que está em \$6 da memoria

2.4 Testes de Saltos

- jal 20 $\Rightarrow PC = 20$ e salvamento do valor atual do PC no registrador número 16
- j 30 $\Rightarrow PC = 30$
- beq \$0, \$8, 5 $\Rightarrow reg8 == reg0?PC+=5 : PC+=1 => PC = 36$
- blt \$0, \$1, 5 $\Rightarrow reg0 < reg1?PC+=5 : PC+=1 => PC = 42$

Os testes foram todos bem sucedidos. A avaliação dos testes foi feita por meio das funções de impressão.