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

116394 ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Trabalho 2: Simulador completo do MARS em C

Aluno: Ian Nery Bandeira
Matrícula: 17/0144739

Descrição do problema

Diferente do trabalho 1, o trabalho 2 necessita da presença de dois arquivos binários *text.bin* e *data.bin* obtidos a partir do comando dump memory do MARS, para obter os comandos do MIPS em binário e assim repassar para a execução do simulador. O simulador conta com 4 comandos, sendo eles *step,* que executa um comando de *ri; run,* que roda o programa até o syscall 10 ou até o registrador PC atingir 8192 (Maximo valor possível à este receber do text.bin); *dump_mem* que consiste em mostrar na tela todo o conteúdo lido pelo text.bin e pelo data.bin; e *dump_R,* que consiste em mostrar na tela o conteúdo de todos os registradores do MIPS.

Descrição das funções

Além das funções acima descritas, possuímos as seguintes funções no programa:

- *Printfirst:* Printa o menu de funções principal do programa.
- *Readbin:* Lê os arquivos binários text.bin e data.bin e assimila seus valores ao vetor de memória interna do programa *mem.*
- *Fetch:* Possui a função de passar para o registrador *ri* a posição de memória necessária para execução do comando em MIPS.
- *Decode:* Decodifica o valor guardado em *ri* para as variáveis *opcode, rs, rt, rd, shamt, funct, k16 e k26,* para execução do comando.
- *Execute:* Executa os comandos com base nos valores recebidos das variáveis *opcode, rs, rt, rd, shamt, funct, k16 e k26* inicializadas na função Decode.

- *Lbfunc:* Recicla a função lb do trabalho anterior, que consiste em ler um byte da memória e retorna para que seja atribuido ao respectivo registrador.
- *Lhfunc:* Recicla a função lh do trabalho anterior, que consiste em ler um half word da memória e retorna para que seja atribuido ao respectivo registrador.
- *Lwfunc:* Recicla a função lw do trabalho anterior, que consiste em ler um word da memória e retorna para que seja atribuido ao respectivo registrador.
- *Sw:* Recicla a função sw do trabalho anterior, com algumas modificações, cuja função é passar para a posição especificada da memória o valor de um word de um registrador.
- *Sh:* Recicla a função sh do trabalho anterior, com algumas modificações, cuja função é passar para a posição especificada da memória o valor de um half word de um registrador.
- *Sb:* Recicla a função sb do trabalho anterior, com algumas modificações, cuja função é passar para a posição especificada da memória o valor de um byte de um registrador.

Os comandos mais complexos utilizados no switch(opcode) estão explicados em comentários no arquivo .c do trabalho.

Testes e resultados

Com base no arquivo repassado pelo professor jacobi no dia da entrega do trabalho, chamado testador.asm, recebemos o seguinte resultado ao fazer o dump do text.bin e data.bin, e rodar no algoritmo desenvolvido:

teste1: # Le palavra da memoria com Lw e compara com constante	Teste1 OK
teste2: # Le byte da memoria com lb e compara com constante	Teste2 OK
teste3: # Le byte da memoria com lbu e compara com constante	Teste3 OK
teste4: # Le meia palavra da memoria com lh e compara com constante	Teste4 OK
teste5: # Le meia palavra da memoria com lhu e compara com constante	Teste5 OK
teste6: # testando LUI e SLL	Teste6 OK
teste7: # Altera valor na memoria, testa SW	Teste7 OK
teste8: # Altera 1o byte na palavra, testa SB	Teste8 OK
teste9: # Altera 16 bits superiores com LH	Teste9 OK
teste10: # Testa blez salto falha	Teste10 OK
teste11: # Testa blez salto ocorre	Teste11 FAIL
teste12: # Testa bgtz salto falha	Teste12 FAIL
teste13: # Testa bgtz salto ocorre	Teste13 OK
teste14: # Testa bgtz salto ocorre da primeira vez	Teste14 OK
teste15: # Testa SLTiu	Teste15 OK
teste16: # Testa ANDi	Teste16 OK
teste17: # Teste ORi	Teste17 OK
teste18: # Testa XORi	Teste18 OK
teste19: # Testa JAL	Teste19 OK
teste20: # Testa ADD	Teste20 OK
teste21: # Testa ADDu	Teste21 OK
teste22: # Testa SUB	Teste22 OK
teste23: # Testa MULT	Teste23 OK
teste24: # Testa DIV	Teste24 OK
teste25: # Testa AND	Teste25 OK
teste26: # Testa OR	Teste26 OK
teste27: # Testa XOR	Teste27 OK

teste28: # Testa NOR	**Teste28 OK**
teste29: # Testa SLT	**Teste29 OK**
teste30: # Testa SRL	**Teste30 OK**
teste31: # Testa SRA	**Teste31 OK**

Pode-se perceber que os testes para o comando blez possuiu 50% de eficácia, enquanto o para o comando bgtz possuiu 66% de eficácia, falhando apenas em um teste cada. Por falta de tempo hábil para teste e debug do algoritmo, não foi possível a correção de tais problemas, porém, visto que blez e bgtz podem ser facilmente substituídos por comandos equivalentes, os quais estão em perfeita funcionalidade no projeto, não comprometeu o funcionamento de tal.