ICMC USP SÃO CARLOS	Universidade de São Paulo - USP Instituto de Ciências Matemáticas e de Computação – ICMC Departamento de Ciências de Computação – SCC SSC610 Organização de Computadores Digitais I 2º Trabalho - Simulador de uma CPU MIPS Multiciclo de 32 bits Disponibilizado em 14/10/2013.

O objetivo deste Segundo Trabalho Prático é implementar em C (*gcc/Linux*) a CPU MIPS multiciclo de 32 bits vista em nossas aulas. Esta CPU MIPS multiciclo de 32 bits deverá ser baseada no conteúdo do livro de Organização de Patterson & Hennessy (1998 ou 2005), usado na nossa disciplina.

Abaixo há o arquivo ***cpu_mips_multiciclo_2013.c*** contendo o código fonte inicial necessário para a implementação. Este código ***deve ser*** obrigatoriamente ***seguido*** no trabalho. ***Não o altere***, pois a correção levará em conta a execução correta do algoritmo e a qualidade do código fonte feito. Para verificar a execução correta do algoritmo é necessário que o código no arquivo ***cpu_mips_multiciclo_2013.c*** permaneça como está. Qualquer erro/dúvida/dificuldade verificados no código fornecido, entre em contato diretamente com o professor para que a questão seja resolvida adequadamente.

Observe que o processador a ser simulado possui várias unidades funcionais que, na prática, executam em paralelo, ou seja, poderiam ser utilizadas ao mesmo tempo. O seu trabalho deve preservar essa característica de concorrência tanto quanto o possível. Portanto, usando uma programação C seqüêncial (convencional) a cada novo ciclo todas as unidades funcionais devem ter a oportunidade de executar (mesmo que sequencialmente). Caberá à Unidade de Controle (UC) determinar os sinais de controle necessários para permitir ou impedir as possíveis execuções.

Considerando esses pressupostos, as suas opções de implementação devem, obrigatoriamente, ser comunicadas ao professor para que haja um acompanhamento do trabalho realizado e também uma orientação sobre quais opções estão corretas ou não.

Implemente neste trabalho, no arquivo ***cpu_multi_code.c***, as instruções: ***add***, ***sub***, ***slt***, ***and***, ***or***, ***lw***, ***sw***, ***beq*** e ***j***.

A UC deverá ser implementada como uma ***ROM com Seqüenciador e Tabelas de Despacho***, conforme descrito em nossas aulas.

Este trabalho deverá ser feito em grupo. O trabalho deverá ser enviado via ***Moodle*** do ***STOA/USP***. No corpo do código fonte há regras para a submissão e para o nome do arquivo a ser submetido. Siga-as com atenção.

```
/* Arquivo cpu_mips_multiciclo_2013.c
   Autor: Paulo Sergio Lopes de Souza

Observacoes:
(1)  Trabalho 2 - SSC610 - Organizacao de Computadores Digitais I
(2)  Disponibilizado em 14/10/2013
(3)  Este codigo fonte esta sendo disponibilizado aos alunos pelo professor
     e deve ser utilizado por todos os grupos. Nao o altere. Ele sera usado
                para a correcao do trabalho.
(4)  Para realizar o seu trabalho, edite um arquivo texto chamado
     cpu_multi_code.c  Insira nele todas as funcionalidades necessarias ao
                seu trabalho.
(5) O arquivo mascara.h possui alguns defines necessarios ao programa. Nao o
    altere. Utilize os defines se necessitar.
(6) Escreva, obrigatoriamente, como comentario nas primeiras linhas do arquivo
    cpu_multi_code.c, os nomes dos alunos integrantes do grupo que efetivamente
                contribuiram para o desenvolvimento deste trabalho.
(7) Antes de submeter o seu trabalho, renomeie o arquivo cpu_multi_code.c para
                T02-GXX.c (onde XX indica o nr do grupo).
(8) Submeta o arquivo T02-GXX.c (onde XX indica o nr do grupo) via Moodle do
    STOA/USP.
*/

#include <stdio.h>
#include <stdlib.h>

/**********************
prototipacao inicial
 **********************/

int main (int, char **);

/* ULA deve seguir a especificação dada em sala de aula.
   As funcoes necessarias a ULA devem estar implementadas em ula_code.c, cujo "#include" encontra-se logo abaixo
int ula (int a, int b, char ula_op, int *result_ula, char *zero, char *overflow)
args de entrada:                        int a, int b, char ula_op
args de saida:                          int *result_ula, char *zero, char *overflow */
int ula( int, int , char, int *, char *, char *);

/* UC principal
void UnidadeControle(int IR, short int *sc);
args de entrada:                         int IR
args de saida:                          short int *sc     */
void UnidadeControle(int, short int *);

/* Busca da Instrucao
void Busca_Instrucao(short int sc, int PC, int ALUOUT, int IR, int *PCnew, int *IRnew, int *MDRnew);
args de entrada:                        short int sc, int PC, int ALUOUT, int IR
args de saida:                          int *PCnew, int *IRnew, int *MDRnew     */
void Busca_Instrucao(short int, int, int, int, int*, int*, int*);

/* Decodifica Instrucao, Busca Registradores e Calcula Endereco para beq
void Decodifica_BuscaRegistrador(short int sc, int IR, int PC, int A, int B, int *Anew, int *Bnew, int *ALUOUTnew);
```

```
args de entrada:                        short int sc, int IR, int PC, int A, int B,
args de saida:                          int *Anew, int *Bnew, int *ALUOUTnew     */
void Decodifica_BuscaRegistrador(short int, int, int, int, int, int *, int *, int *);

/*Executa TipoR, Calcula endereco para lw/sw e efetiva desvio condicional e incondicional
void Execucao_CalcEnd_Desvio(short int sc, int A, int B, int IR, int PC, int ALUOUT, int *ALUOUTnew, int *PCnew);
args de entrada:                        short int sc, int A, int B, int IR, int PC, int ALUOUT
args de saida:                           int *ALUOUTnew, int *PCnew */
void Execucao_CalcEnd_Desvio(short int, int, int, int, int, int, int *, int *);

/* Escreve no Bco de Regs resultado TiporR, Le memoria em lw e escreve na memoria em sw
void EscreveTipoR_AcessaMemoria(short int sc, int B, int IR, int ALUOUT, int PC, int *MDRnew, int *IRnew);
args de entrada:                short int sc, int B, int IR, int ALUOUT, int PC
args de saida:                  int *MDRnew, int *IRnew */
void EscreveTipoR_AcessaMemoria(short int, int, int, int, int, int *, int *);

/* Escreve no Bco de Regs o resultado da leitura da memoria feita por lw
void EscreveRefMem(short int sc, int IR, int MDR, int ALUOUT);
args de entrada:                        short int sc, int IR, int MDR, int ALUOUT
args de saida:                          nao ha */
void EscreveRefMem(short int, int, int, int);


// contem as mascaras e algumas definicoes necessarias ao programa
// mascara.h pode ser alterado, conforme sua necessidade
#include "mascara.h"

/*****************************
definicao de variaveis globais
 *****************************/
int memoria[MAX];                                       // Memoria RAM
int reg[NUMREG];                                        // Banco de Registradores

char loop = 1;                                                  // variavel auxiliar que determina a parada da execucao deste programa


// contem todas as funcoes desenvolvidas para o processador multiciclo.
//Inclua aqui as suas funcoes/procedimentos
#include "cpu_multi_code.c"


int main (int argc, char *argv[])
{
        int PCnew = 0, IRnew, MDRnew, Anew, Bnew, ALUOUTnew;
                                                        // Registradores auxiliares usados para a escrita dentro de um ciclo.
                                                        // Guardam temporariamente o resultado durante um ciclo. Os resultados aqui armazenados estarao
                                                        // disponiveis para leitura no final do ciclo atual (para que o mesmo esteja disponivel apenas no
                                                        // incio do ciclo seguinte).
                                                        // Em um ciclo sempre e lido o conteudo de um registrador atualizado no ciclo anterior.

        int PC = 0, IR=-1, MDR, A, B, ALUOUT;
                                                        // Registradores especiais usados para a leitura em um ciclo.
                                                        // Guardam os resultados que poderao ser utilizados ja neste ciclo, pois foram atualizados no final
                                                        // do ciclo anterior.
                                                        // Ex.: em um ciclo, o resultado da ULA e inserido inicialmente em ALUOUTnew.  Apenas no final
                                                        // do ciclo esse conteudo podera ser atribuido para ALUOUT, para que o mesmo possa ser
                                                        // usado no ciclo seguinte.
                                                        // Em um ciclo sempre e lido o conteudo de um registrador atualizado no ciclo anterior.

        short int sc = 0;                                       // Sinais de Controle (16 sinais)
                                                                                //              cada bit determina uma dos sinais de
controle que saem da UC
                                                                                // a posicao de cada sinal dentro do short int esta
especificada em mascara.h
        char ula_op = 0;                                        // Sinais de Controle de entrada para a ULA
                                                                                //              sao usados apenas os 4 bits menos
significativos dos 8 disponiveis.

        int nr_ciclos = 0; // contador do numero de ciclos executados

/*
As variaveis zero e overflow nao precisam definidas na main.
Serao argumentos de retorno da ula e devem ser definidas localmente nas
rotinas adequadas.
char  zero, overflow;                           // Sinais de Controle de saida da UL: bit zero e bit para indicar overflow
*/


        memoria[0] = 0x8c480000;  // 1000 1100 0100 1000 0000 0000 0000 0000  lw $t0, 0($v0)
                5
        memoria[1] = 0x010c182a;  // 0000 0001 0000 1100 0001 1000 00101010   slt $v1, $t0, $t4
                4
        memoria[2] = 0x106d0004;  // 0001 0000 0110 1101 0000 0000 0000 0100  beq $v1, $t5, fim(4 palavras abaixo de PC+4)        3
        memoria[3] = 0x01084020;  // 0000 0001 0000 1000 0100 0000 0010 0000  add $t0, $t0, $t0
                4
        memoria[4] = 0xac480000;  // 1010 1100 0100 1000 0000 0000 0000 0000  sw $t0, 0($v0)
                4
        memoria[5] = 0x004b1020;  // 0000 0000 0100 1011 0001 0000 0010 0000  add $v0, $t3, $v0
                4
        memoria[6] = 0x08000000;  // 0000 1000 0000 0000 0000 0000 0000 0000  j inicio (palavra 0)
        3
        memoria[7] = 0;         // fim (criterio de parada do programa)    (27*6)+(5+4+3)+1
        memoria[8] = 0;
        memoria[9] = 0;
```

```
		// Dados
		memoria[20] = 10;
		memoria[21] = 12;
		memoria[22] = 14;
		memoria[23] = 16;
		memoria[24] = 18;
		memoria[25] = 20;
		memoria[26] = -1;

		reg[2]  = 80;  // $v0

		reg[11] = 4;  // $t3
		reg[12] = 0;  // $t4
		reg[13] = 1;  // $t5

		while(loop){
			// aqui comeca um novo ciclo

			// abaixo estao as unidades funcionais que executarao em todos os ciclos
			// os sinais de controle em sc impedirao/permitirao que a execucao seja, de fato, efetivada

			UnidadeControle(IR, &sc);
			Busca_Instrucao(sc, PC, ALUOUT, IR, &PCnew, &IRnew, &MDRnew);
			Decodifica_BuscaRegistrador(sc, IR, PC, A, B, &Anew, &Bnew, &ALUOUTnew);
			Execucao_CalcEnd_Desvio(sc, A, B, IR, PC, ALUOUT, &ALUOUTnew, &PCnew);
			EscreveTipoR_AcessaMemoria(sc, B, IR, ALUOUT, PC, &MDRnew, &IRnew);
			EscreveRefMem(sc, IR, MDR, ALUOUT);

			// contador que determina quantos ciclos foram executados
			nr_ciclos++;

			// atualizando os registradores temporarios necessarios ao proximo ciclo.
			PC                      = PCnew;
			IR                      = IRnew;
			MDR                     = MDRnew;
			A                       = Anew;
			B                       = Bnew;
			ALUOUT  = ALUOUTnew;

			// aqui termina um ciclo
		} // fim do while(loop)

		// impressao da memoria para verificar se a implementacao esta correta
		{
			int ii;
			for (ii = 20; ii < 27; ii++) {
				printf("memoria[%d]=%ld \n", ii, memoria[ii]);
			}
			printf("Nr de ciclos executados =%d \n", nr_ciclos);
		}
		exit(0);
} // fim de main
```