ICMC USP SÃO CARLOS	Universidade de São Paulo - USP Instituto de Ciências Matemáticas e de Computação – ICMC Departamento de Sistemas de Computação – SSC SSC0112 Organização de Computadores Digitais I 2º Trabalho - Simulador de uma CPU MIPS Multiciclo de 32 bits Disponibilizado em 22/05/2014.

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_2014.c*** contendo o código fonte inicial necessário para a implementação. Este código ***deve ser*** obrigatoriamente ***seguido*** no trabalho. ***Não o altere***, pois ele deverá ser o mesmo para todos os grupos. 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 sequencial (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. A correção deste trabalho levará em conta a execução correta do algoritmo e a qualidade do código fonte feito, conforme explicado em sala de aula.

As instruções a serem implementadas são todas as originalmente detalhadas em sala de aula (***add***, ***sub***, ***slt***, ***and***, ***or***, ***lw***, ***sw***, ***beq*** e ***j***) e também mais as seguintes: ***lui, ori, andi, slti, bltz, bne*** e ***move***.

A UC deverá ser implementada como um controle microprogramado, conforme descrito em nossas aulas. Desse modo, deve haver um microcódigo para a representação inicial, um contador de microprograma + tabelas de despacho para o controle de sequenciamento, tabelas verdade para a representação no nível de lógica digital e memórias ROMs para a técnica de implementação. Esta memória ROM deve ser representada no seu código como um vetor de inteiros.

Os sinais de controle emitidos pela UC serão representados no seu código como bits de uma variável do tipo inteiro (32 bits). A ordem dos bits de controle, nessa variável de 32 bits, é a seguinte:

00- RegDst (RegDst)	01- EscReg (RegWrite)	02- IsZero
03- UALFonteA (ALUSrcA)	04- UALFonteB0 (ALUSrcB0)	05- UALFonteB1 (ALUSrcB1)
06- UALOp0 (ALUOp0)	07- UALOp1(ALUOp1)	08- UALOp2(ALUOp2)
09- FontePC0 (PCSource0)	10- FontePC1 (PCSource1)	11- PCEscCond (PCWriteCond)
12- PCEsc (PCWrite)	13- IouD (IorD)	14- LerMem (MemRead)
15- EscMem (MemWrite)	16- MemParaReg0 (MemtoReg0)	17- MemParaReg1 (MemtoReg1)
18- IREsc (IRWrite)	19- ControleSeq0	20- ControleSeq1

A implementação do trabalho deve obrigatoriamente utilizar esses sinais de controle, nesta ordem. As alterações no caminho de dados da arquitetura da CPU MIPS Multiciclo, para que as novas instruções possam ser implementadas, já estão no diagrama de blocos da figura em anexo (destacadas em vermelho). Este caminho de dados também não pode ser alterado. Novamente, qualquer erro/dúvida/dificuldade verificados no caminho de dados fornecido, entre em contato diretamente com o professor para que a questão seja resolvida adequadamente.

As novas instruções ***lui***, ***ori***, ***andi***, ***slti***, ***bltz*** e ***bne*** estão especificadas no livro de Organização de Patterson & Hennessy (1998 ou 2005). A nova instrução ***move*** (antes uma pseudoinstrução) terá agora o formato ***Tipo-I*** e o código de operação ***0x3F***. Esta instrução ***move rt, rs*** tem a seguinte semântica: ***rt = rs***. Fora ***opcode***, ***rs*** e ***rt***, os demais campos da instrução têm o valor ***zero***.

A quantidade de ciclos gasta na execução das novas instruções é: ***lui/bltz/bne/move*** 3 ciclos e ***ori/andi/slti*** 4 ciclos.

Você deve entregar neste trabalho: o código fonte em C, o microcódigo da UC, uma tabela com o mapeamento dos campos e dados em sinais de controle (conforme feito em sala de aula), a nova UC da ULA, as tabelas de despacho, uma tabela correspondente à memória ROM implementada na UC, identificando cada bit existente nesta tabela.

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. Antes de submeter o seu trabalho, compacte os arquivos no padrão zip, gerando o arquivo **TB2-TurmaX-GrupoYY.zip** (***X*** indica o número da turma (1, 2 ou 3) e ***YY*** indica o número do grupo na turma).

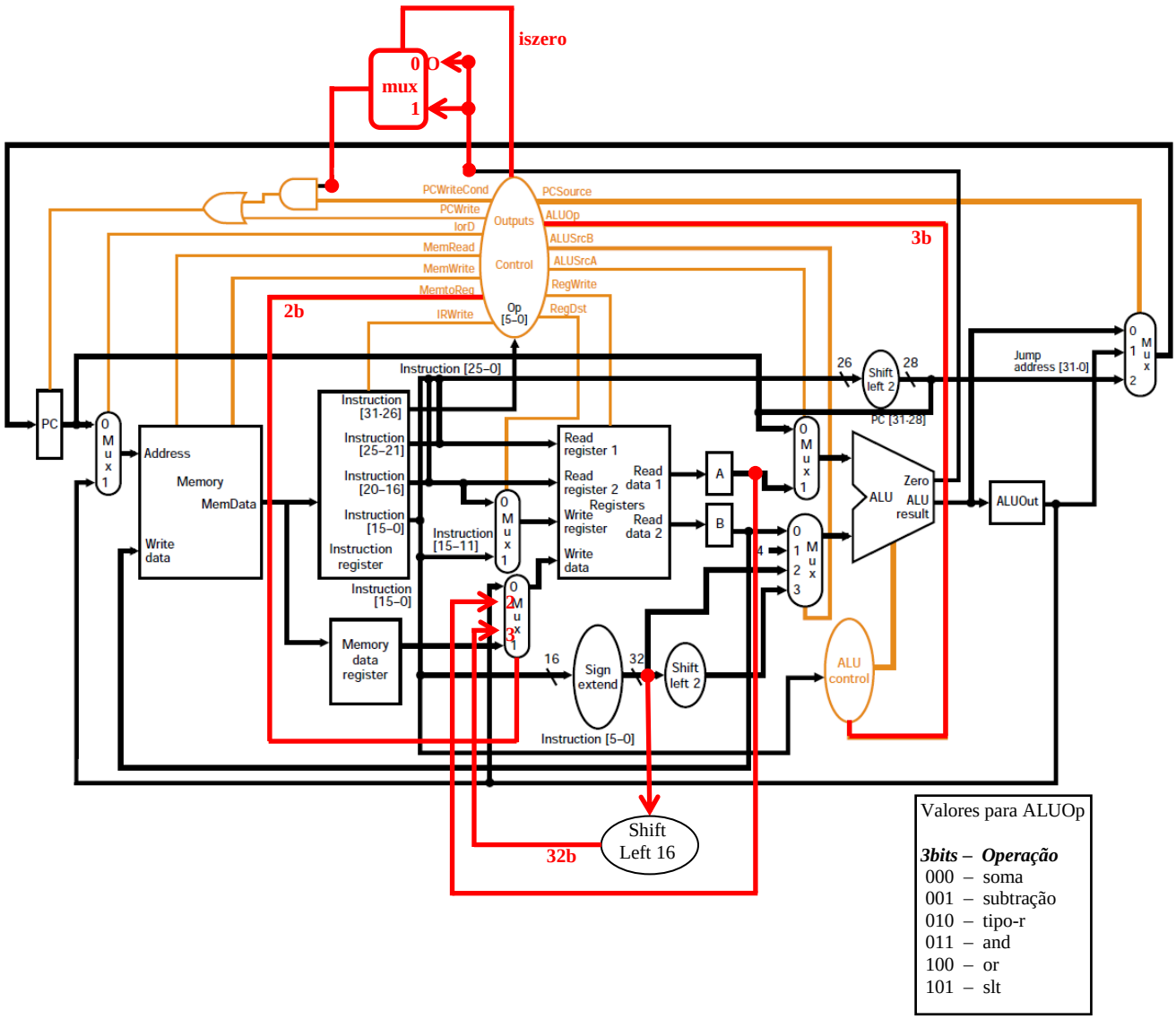
A linha de comando a ser utilizada para a compilação do código será: ***gcc cpu_mips_multiciclo_2014.c -Werror***

```
/* Arquivo cpu_mips_multiciclo_2014.c
   Autor: Paulo Sergio Lopes de Souza

Observacoes:
(1)  Trabalho 2 – SSC0112 - Organizacao de Computadores Digitais I
(2)  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.
(3)  Para realizar o seu trabalho, edite um arquivo texto chamado
     cpu_multi_code.c  Insira nele todas as funcionalidades necessarias ao
                seu trabalho.
(4) 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.
(5) Antes de submeter o seu trabalho, compacte os arquivos no  padrão zip,
      gerando o arquivo  TB2-TurmaX-GrupoYY.zip (X indica o nr da turma (1, 2 ou 3)
      e YY indica o nr do grupo na turma).
(6) Submeta via Moodle do STOA/USP o arquivo TB2-TurmaX-GrupoYY.zip
*/

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

#define MAX 512
#define NUMREG 32

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

int main (int, char **);

/* ULA segue a especificação dada em sala de aula.
void 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 */
void ula( int, int , char, int *, char *, char *);

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

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

/* Decodifica Instrucao, Busca Registradores e Calcula Endereco para beq
void Decodifica_BuscaRegistrador(int sc, int IR, int PC, int A, int B, int *Anew, int *Bnew, int *ALUOUTnew);
args de entrada:                        int sc, int IR, int PC, int A, int B,
args de saida:                          int *Anew, int *Bnew, int *ALUOUTnew     */
void Decodifica_BuscaRegistrador(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(int sc, int A, int B, int IR, int PC, int ALUOUT, int *ALUOUTnew, int *PCnew);
args de entrada:                        int sc, int A, int B, int IR, int PC, int ALUOUT
args de saida:                          int *ALUOUTnew, int *PCnew */
void Execucao_CalcEnd_Desvio(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(int sc, int B, int IR, int ALUOUT, int PC, int *MDRnew, int *IRnew);
args de entrada:                int sc, int B, int IR, int ALUOUT, int PC
args de saida:                  int *MDRnew, int *IRnew */
void EscreveTipoR_AcessaMemoria(int, int, int, int, int, int *, int *);

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


/****************************
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

void ula (int a, int b, char ula_op, int *result_ula, char *zero, char *overflow)
{
    *overflow = 0;
    switch (ula_op)
    {
            case 0:    // and
                *result_ula = a & b;
                break;

            case 1:   // or
                *result_ula = a | b;
                break;

            case 2:  // add
                *result_ula = a + b;
                if ( (a >= 0 && b >= 0 && *result_ula < 0) || (a < 0 && b < 0 && *result_ula >= 0) )
                        *overflow  = 1;
                break;

            case 6:   // sub
                *result_ula = a – b;
                if ( (a >= 0 && b < 0 && *result_ula < 0) || (a < 0 && b >= 0 && *result_ula >= 0) )
                        *overflow  = 1;
                break;
```

```
                case 7:   // slt
                    if(a < b)
                            *result_ula = 1;
                    else
                            *result_ula = 0;
                    break;

                case 12:   // nor
                    *result_ula = ~(a | b);
                    break;

    }
    if (*result_ula == 0)
        *zero = 1;
    else
        *zero = 0;

   return;

} // fim da ULA

// contem todas as funcoes desenvolvidas por você 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.

                int sc = 0;                             // Sinais de Controle
                                                        //              cada bit determina um dos sinais de controle que saem da UC.
                                                        // A posicao de cada sinal dentro do int esta especificada no enunciado
                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]=%d \n", ii, memoria[ii]);
                }
                printf("Nr de ciclos executados =%d \n", nr_ciclos);
        }
        exit(0);
} // fim de main
// ***********************************************************************************************************
```



Valores para ALUOp

3bits – Operação
000 – soma
001 – subtração
010 – tipo-r
011 – and
100 – or
101 – slt