Universidade de São Paulo - USP
Instituto de Ciências Matemáticas e de Computação – ICMC
Departamento de Sistemas de Computação – SSC
SSC-112 Organização de Computadores Digitais I
2º Trabalho – Simulador de uma UC Hardwired para a CPU MIPS Multiciclo
Disponibilizado em11/11/2009. Entrega do trabalho 07/12/2006, segunda, às 07:00h, por email.

O objetivo deste Segundo Trabalho Prático é implementar em C (gcc/Linux) a UC Hardwired para a CPU MIPS Multiciclo vista em nossas aulas. A CPU MIPS multiciclo a ser implementada deverá ser baseada no conteúdo do livro de Organização de Patterson & Hennessy (1998 ou 2005), este usado na nossa disciplina.

Abaixo há o arquivo ***cpu_mips.c*** contendo um 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 deste código. Qualquer erro/dúvida/dificuldade verificados no código fornecido, entre em contato diretamente com o Professor para que a questão seja resolvida adequadamente.

Implemente a função ***int UnidadeControle(int opcode)*** contendo a UC Hardwired pedida. Os códigos que você gerar deverão estar apenas em um arquivo, chamado de ***hardwired_code.c***

O seu trabalho deve implementar a UC representando cada sinal de controle como uma Equação Lógica, considerando a soma-de-produtos, ou seja, implemente em C uma simulação de um Arranjo Lógico Programável (PLA). Devem ser feitas as operações *bit-a-bit* adequadas das entradas e do estado atual da UC para determinar as saídas (sinais de controle). Para o controle de sequenciamento use ***Função Próximo Estado Explícita***. Lembre-se: deve haver a implementação em C do que seria uma "equação lógica" para cada bit, tanto para sinais de controle quanto para o controle de sequenciamento.

As instruções a serem implementadas são as 9 já detalhadas em sala de aula (***add***, ***sub***, ***slt***, ***and***, ***or***, ***lw***, ***sw***, ***beq*** e ***j***), e também as seguintes: ***jal, jr, jalr, addi, andi*** e ***bne***. Assim, devem ser implementadas 15 instruções ao todo.

Os sinais de controle emitidos pela UC serão representados como bits de uma variável do tipo inteiro (32 bits).
A ordem, nessa variável de 32 bits, dos sinais de controle já propostos por Patterson & Hennessy para as 9 instruções implementadas no livro, além dos sinais criados para que as novas instruções possam ser implementadas, é a seguinte:

00- RegDst0 (RegDst0)	01- RegDst1 (RegDst1)	02- EscReg (RegWrite)
03- UALFonteA (ALUSrcA)	04- UALFonteB0(ALUSrcB0)	05- UALFonteB1(ALUSrcB1)
06- UALOp0 (ALUOp0)	07- UALOp1(ALUOp1)	08- FontePC0 (PCSource0)
09- FontePC1 (PCSource1)	10- PCEscCond (PCWriteCond)	11- PCEsc (PCWrite)
12- IouD (IorD)	13- LerMem (MemRead)	14- EscMem (MemWrite)
15- MemParaReg0 (MemtoReg0)	16- MemParaReg1 (MemtoReg1)	17- IREsc (IRWrite)
18- BNE (BNE)		

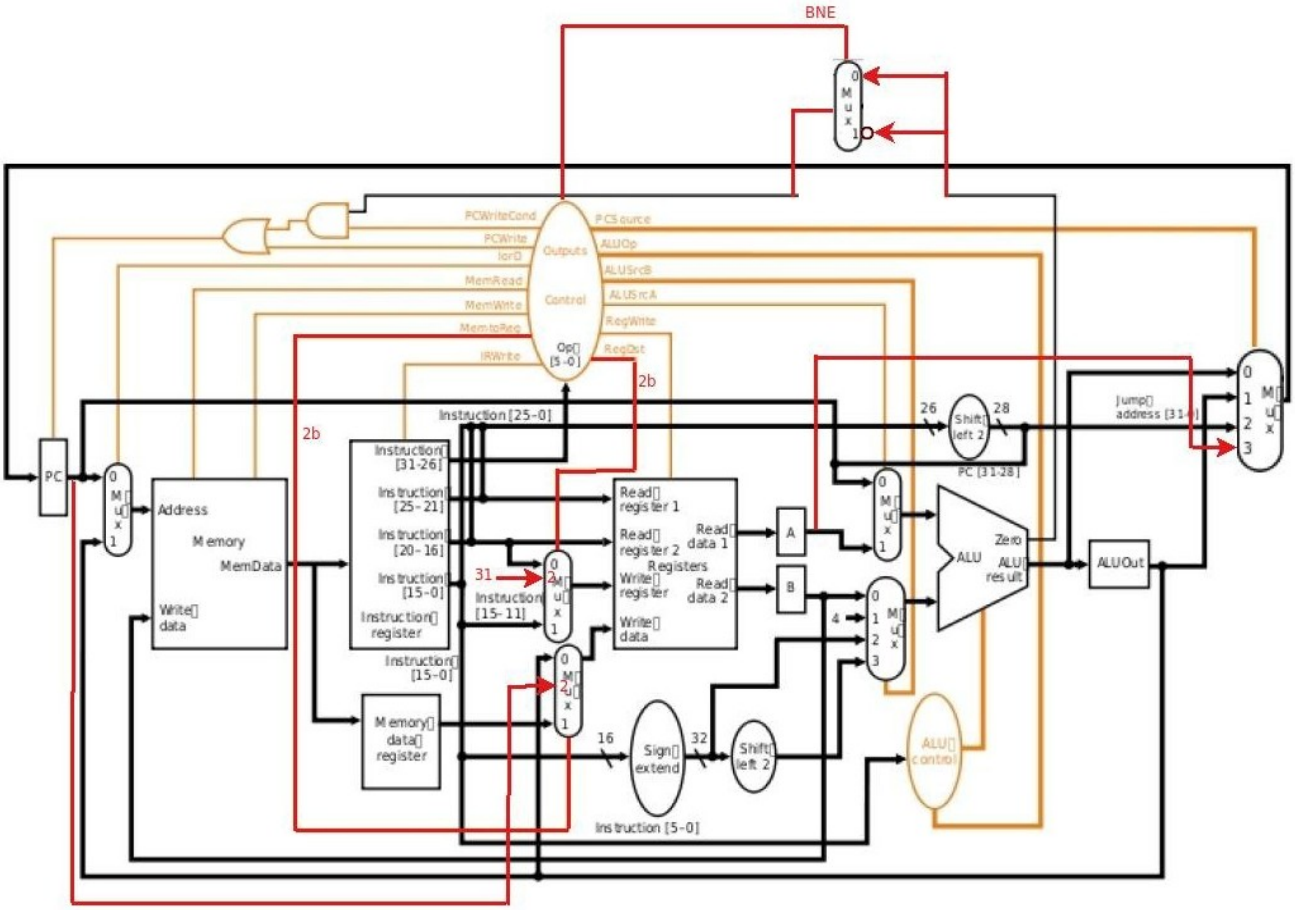
A implementação do trabalho deve obrigatoriamente utilizar esses sinais de controle. As alterações no caminho de dados da arquitetura da CPU MIPS Multiciclo, para que as instruções ***jal, jr, jalr, addi, andi*** e ***bne*** possam ser implementadas, já estão no diagrama de blocos da figura em anexo (destacadas em vermelho). O valor 3 (11) para o sinal PCSource seleciona o valor de A no mux, o valor 2 (10) para o sinal RegDst seleciona a constante 31, e o valor 2 (10) para MemtoReg seleciona PC+4. Este caminho de dados também não pode ser alterado.

Importante: Os códigos de operação utilizados no trabalho para as instruções **jr** e **jalr** devem ser 20 (010100) e 21 (010101), respectivamente. Essa mudança visa simplificar a implementação, já que essas instruções são do tipo-R (seria necessário a utilização do campo de função na UC Principal). O código de operação das demais instruções permanece inalterado.

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.

Para a implementação da UC, desenvolva primeiro a MEF correspondente à implementação a ser feita. Esta MEF deverá ser entregue junto com o código fonte.

Este trabalho deverá ser feito em grupo, este já determinado no início do semestre letivo. O trabalho deverá ser enviado até as 07:00h do dia 07/12/2009 (segunda-feira) para o e-mail ***orgcomp1@gmail.com***. Digite no *subject*, obrigatoriamente: Trab02 - Grupo XXX (onde XXX indica a Turma e o número do grupo). Forneça, obrigatoriamente, no corpo do e-mail o número do grupo, os integrantes e, em anexo, apenas os arquivos ***hardwired_code.c*** e ***mef.pdf)***.


BNE
PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op[5-0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
2b
2b
Instruction [25-0]
26
Shift left 2
28
Jump address [31-0]
PC [31-28]
PC
Address
Memory
MemData
Write data
Instruction [31-26]
Instruction [25-21]
Instruction [20-16]
Instruction [15-0]
Instruction register
Instruction [15-0]
Memory data register
31
Instruction [15-11]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
ALU
Zero
ALU result
ALUOut
16
Sign extend
32
Shift left 2
ALU control
Instruction [5-0]

```
/*****************************************************************************************************/
/* Arquivo trab2.c
   Autor: Paulo Sergio Lopes de Souza

Observacoes:
(1)  Trabalho 2 - SSC0112 - Organizacao de Computadores Digitais I
(2)  Disponibilizado em 11/10/2009 - Data da entrega do trabalho: 07/12/2009
(3)  O arquivo trab2.c deve ser utilizado por todos os grupos.  Nao o altere.
(4)  Para realizar o seu trabalho, edite um arquivo texto chamado hardwired_code.c  Insira nele todas as funcionalidades necessarias ao seu trabalho.
(7) A entrega do trabalho devera ser feita para o email orgcomp1@gmail.com
    Digite, obrigatoriamente, no subject: Trab02 - Grupo XXX (onde XXX indica a Turma e o nr do grupo).
    Forneça, obrigatoriamente, no corpo do email o número do grupo, os integrantes e, em anexo, apenas os arquivos hardwirde_code.zip e MEF.pdf
*/

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

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

int main (void);
int UnidadeControle(char opcode);


/* contem todas as funcoes desenvolvidas por voce neste Trabalho para a UC hardwired*/
#include "hardwired_code.c"

int main (void)
{

        int ciclo, opcode;
        int sc = 0;             // Sinais de Controle. Cada bit determina um dos sinais de controle que saem da UC

        while(1) // loop para solicitar o opcode
        {
                printf("Entre com o opcode da instrucao (valor -1 encerra o programa): ");
                scanf("%d", &opcode);
                printf("\n");
                if(opcode == -1) {
                        printf("Encerrando o trab2! \n");
                        exit(0);
                }
                if (opcode != 35 && opcode != 0  && opcode != 43 && opcode != 8  && opcode != 12 &&
                   opcode != 4  && opcode != 2  && opcode != 3  && opcode != 20 && opcode != 21 &&
                   opcode != 5)
                {
                        printf("Opcode invalido. Tente outro! \n");
                        continue;
                }
                ciclo = 1; //variavel que indica o ciclo da instrucao

                while(1) // aqui comeca o que seria a execucao da instrucao
                {       // aqui comeca o que seria um novo ciclo

                        // sc representa os sinais de controle vindos da UC
                        sc = UnidadeControle(opcode);

                        printf("instrucao: %x, ciclo da instrucao: %d, sinais de controle: %x\n", opcode, ciclo, sc);

                        // criterio de parada da instrução. Como estamos simulando apenas o comportamento
                        // da UC, sem o caminho de dados completo, este IF finalizará a instrução, permitindo
                        // fornecer outra para execucao.
                        // Determina se estamos no ultimo ciclo de cada instrucao para finalizar a execucao
                        // if  (lw E ciclo 5) OU
                        //    ((Tipo-R OU sw OU addi OU andi) E ciclo 4) OU
                        //    ((beq OU j OU jal OU jr OU jalr OU bne) E ciclo 3) OU
                        //    (esta tentando usar mais que 5 ciclos para a instrucao)
                        //
                        //                          então finaliza o loop da instrucao
                        //
                        if ( ( (opcode == 35) && ciclo == 5) ||
                            ( (opcode == 0  || opcode == 43 || opcode == 8 || opcode == 12) && ciclo == 4)  ||
                            ( (opcode == 4  || opcode == 2  || opcode == 3 || opcode == 20 || opcode == 21 || opcode == 5) && ciclo == 3) )
                        {
                                break;  // ultimo ciclo da instrução. Termina este loop interno para pedir outro opcode ao usuario.
                        }
                        ciclo++; // incrementa variavel que indica o ciclo da instrucao
                        // aqui termina um ciclo
                } // fim da execução da instrução
        } // fim do loop para solicitar o opcode
        exit(0);
} // fim da main
```