	USP/ICMC/SSC – SSC0112 Organização de Computadores Digitais I 2º Trabalho – Simulador de uma CPU MIPS Multiciclo de 32 bits Disponibilizado em 26/04/2018 Entrega do trabalho via Moodle Stoa.

Este trabalho visa implementar em C (gcc/Linux) a CPU MIPS Multiciclo de 32 bits vista em nossas aulas. A CPU deverá ser baseada no conteúdo do livro de *Organização e Projeto de Computadores: a interface hardware/software*, de Patterson & Hennessy, este usado na nossa disciplina.

O processador a ser simulado (ver diagrama de bloco nesta especificação) possui unidades funcionais e uma unidade de controle que, na prática, executam em paralelo, ou seja, podem ser utilizadas ao mesmo tempo. A aplicação deve preservar essa característica de paralelismo tanto quanto possível. Usando uma programação C 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, se necessário, as possíveis execuções.

A CPU deve executar corretamente um código fonte MIPS fornecido em um arquivo texto chamado ***code.bin***, este passado como argumento de entrada ao se carregar a aplicação que simula a CPU. Após terminar a execução deste código binário, a CPU finaliza e a aplicação se encerra. As instruções em ***code.bin***, estarão representadas como um inteiro de 32 bits. Inicialize todos os registradores e posições de memória com o valor zero no início da execução desta aplicação. Ao terminar a execução deve-se imprimir na tela o conteúdo de todos os registradores temporários, do banco de registradores e de controle, e também as 32 primeiras posições da memória RAM. Todas as saídas devem estar como inteiros em decimal.

O código fonte que você gerar deverá estar em apenas um arquivo, chamado de ***cpu_multi_code.c***.

À exceção da Unidade de Controle, todas as unidades funcionais presentes no diagrama de bloco desta especificação (multiplexadores, ULA, Banco de Registradores, ...) podem ser implementadas em C, livremente, i.e., não precisam ser implementadas considerando o nível de lógica digital, bastando simular o comportamento delas em C. A Unidade de Controle, por sua vez, deve ser implementada 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***. 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- BNE (BNE)	16- IREsc (IRWrite)	17- MemParaReg0 (MemtoReg0)
18- MemParaReg1 (MemtoReg1)		

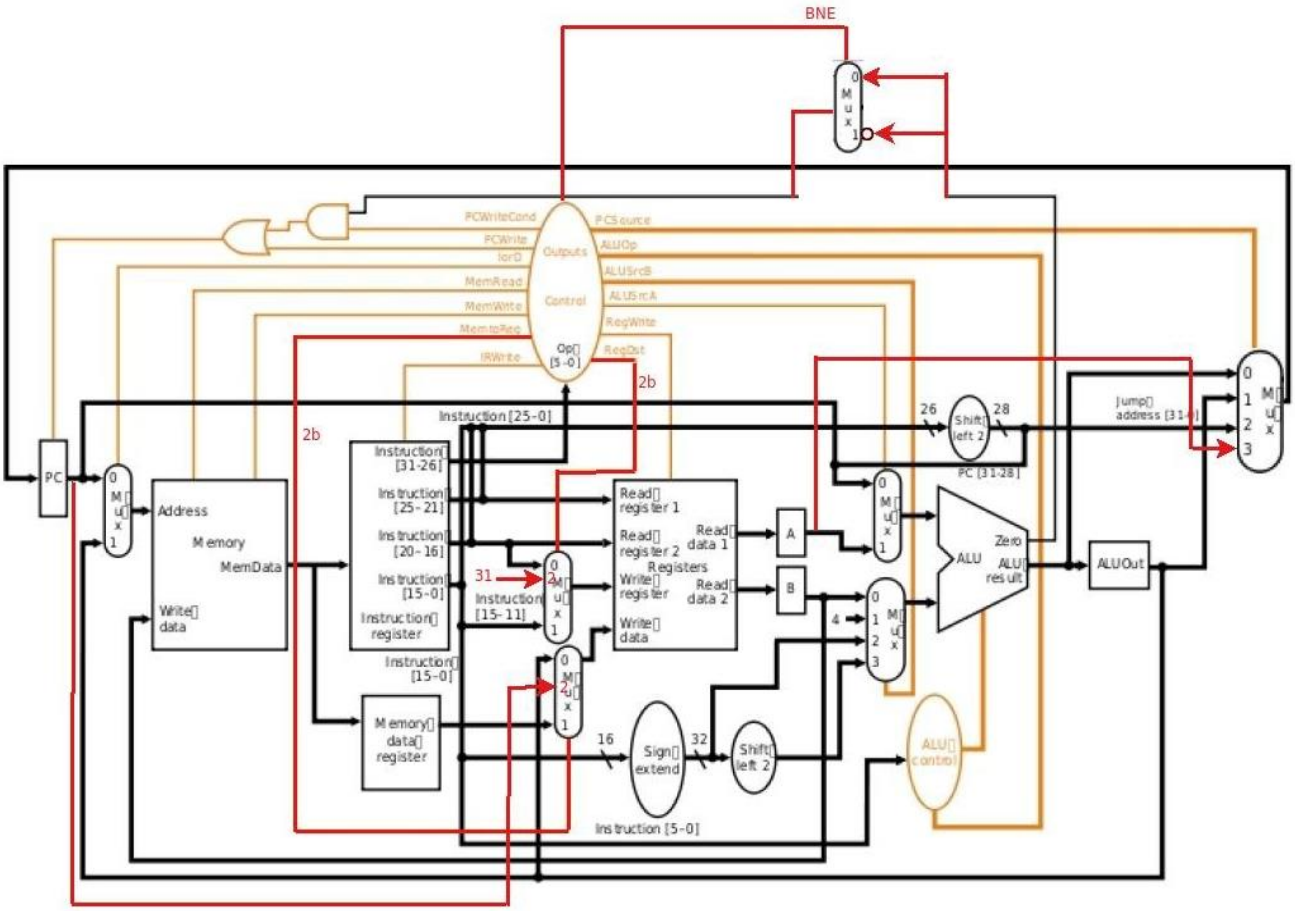
A implementação do trabalho deve obrigatoriamente utilizar estes sinais de controle, nesta ordem. As alterações no caminho de dados da arquitetura da CPU MIPS Multiciclo, para que as instruções ***jal, jr, jalr, addi, andi*** e ***bne*** possam ser implementadas, já estão no diagrama de blocos a seguir (destacadas em vermelho). O valor 3_{dec} (11_{bin}) para o sinal PCSource seleciona o valor de A no mux, o valor 2_{dec} (10_{bin}) para o sinal RegDst seleciona a constante 31_{dec}, e o valor 2_{dec} (10_{bin}) para MemtoReg seleciona PC+4. Este caminho de dados também não pode ser alterado. Qualquer erro/dúvida/dificuldade verificados no caminho de dados fornecido, entre em contato diretamente com o professor.

Importante: Os códigos de operação utilizados no trabalho para as instruções **jr** e **jalr** devem ser 20_{dec} (010100_{bin}) e 21_{dec} (010101_{bin}), respectivamente. Elas passam a ser ***tipo-I***. Essa mudança visa simplificar a implementação, já que essas instruções são originalmente do ***tipo-R*** e, em se mantendo da forma original, seria necessária a utilização do campo de função na UC Principal. Os códigos de operação das demais instruções permanecem inalterados.

Este trabalho deverá ser feito em grupo, este já determinado no início do semestre letivo. O trabalho deverá ser enviado pelo Moodle do Stoa até a data combinada em sala de aula. Forneça, obrigatoriamente, no corpo do arquivo enviado o número da turma (1 ou 2), o número do seu grupo e o nome de todos os integrantes do grupo que participaram do desenvolvimento.

Vc deve seguir rigorosamente toda a especificação deste texto a fim de validar a sua nota. Essa validação será feita por um fator de multiplicação à nota final do Trabalho 2: 0(zero) por não seguir a especificação ou 1(um) por seguir.

Bônus: a implementação deste trabalho pode ser feita em paralelo, sendo cada unidade funcional e a unidade de controle uma *thread* (ou *lightweight process*) à parte. Todas as threads são disparadas no início da execução do programa e são sincronizadas obrigatoriamente com semáforos ou variáveis de condição do padrão POSIX. Ao final da execução do programa, as threads são eliminadas. O valor do bônus não será divulgado antes da entrega do trabalho.


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
PC [31-28]
Jump address [31-0]
PC
Address
Memory
MemData
Write data
Instruction [31-26]
Instruction [25-21]
Instruction [20-16]
Instruction [15-0]
Instruction register
Instruction [15-0]
31
Instruction [15-11]
Memory data register
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]