Proj 1: Um Processador MIPS Single-Cycle Completo

Centro de Ciências Exatas e Tecnológicas
Universidade Federal do Recôncavo da Bahia
GCET 231 - Circuitos Digitais II

2019.2
rev 1.0

Sumário

1. Introdução

A partir do projeto realizado neste roteiro de laboratório você irá aprender a:

- Integrar ALU, registradores, etc., para formar um caminho de dados completo.
- Projetar a unidade de controle de um processador MIPS.
- Integrar a unidade de controle ao caminho de dados.
- Integrar unidades de memória a um processador.
- Codificar um conjunto de instruções.
- Mais práticas com rotinas de teste para verificação de um processador.

---

2. Procedimento

2.1 Parte 0 - Projetando um display de 7 segmentos de 4 dígitos

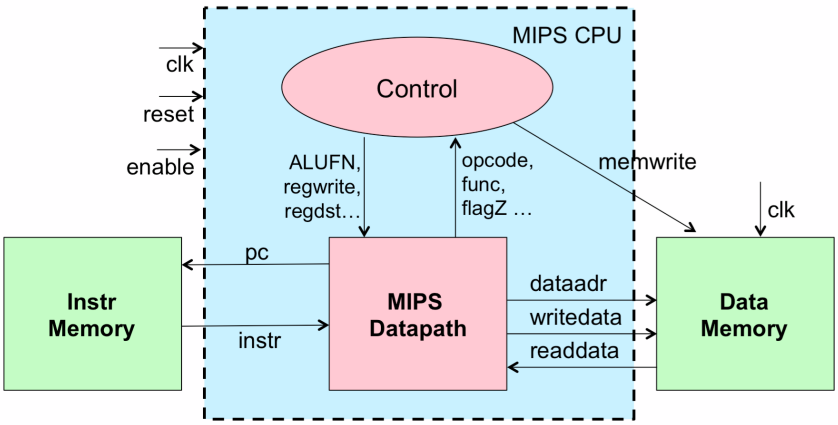
Estude os slides de aula cuidadosamente, e revise os Capítulos 7.1–7.3 do livro texto. Revise também o folheto verde do livro Patterson e Hennessy contendo o conjunto de instruções do processador MIPS; uma versão PDF deste folheto foi disponibilizado no site do curso. Apesar de haverem diferenças entre os livros de autoria de Patterson & Hennessy e o livro texto Harris e Harris, nós utilizaremos o primeiro. Nós faremos isso por que o simulador MARS, o qual você utilizará para construir seu código assembly segue o primeiro.

Estude especialmente os Slides 33–38 para perceber as decisões de projeto específicas para versão do MIPS utilizada em nosso laboratório:

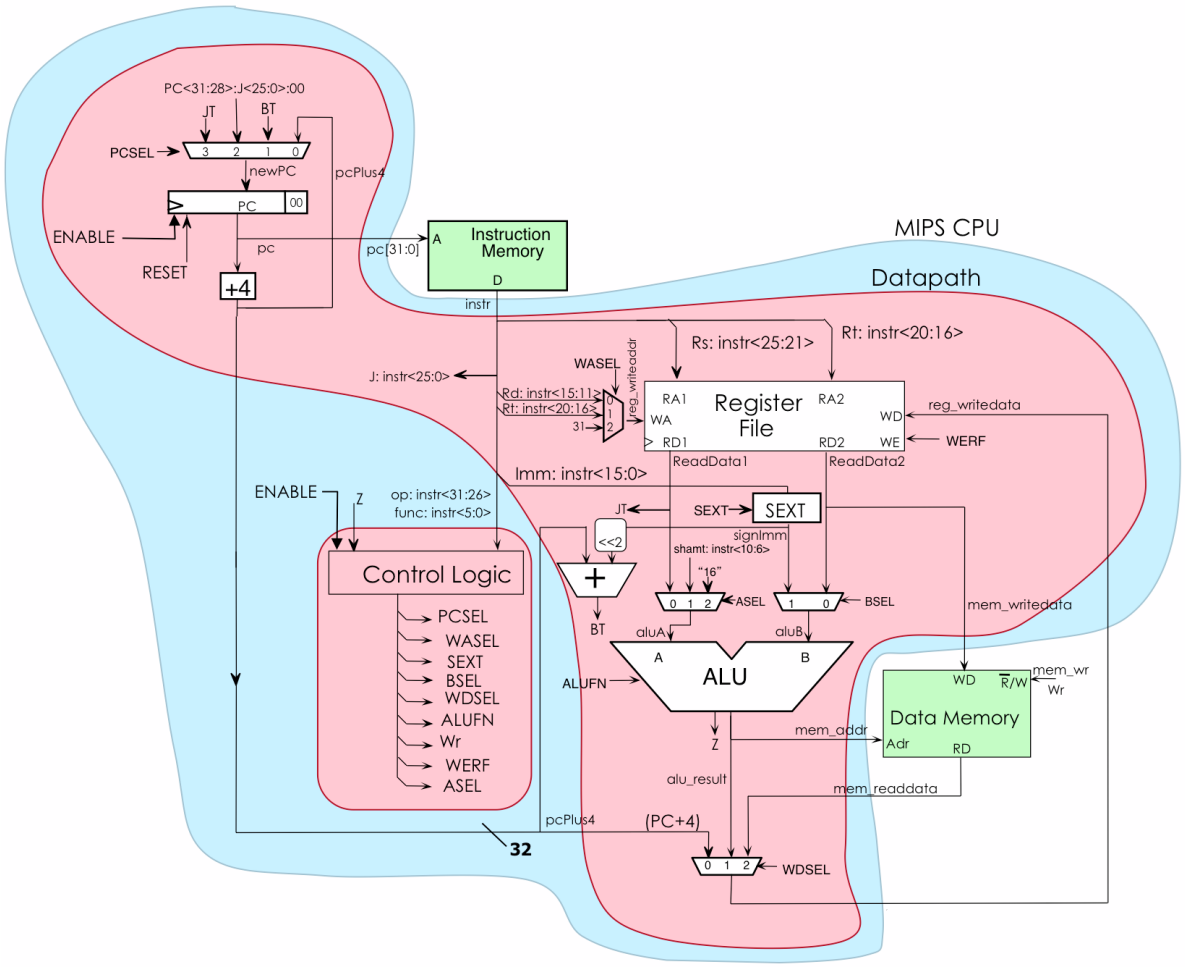
- *Exceções:* Nossa versão do MIPS não possui suporte a exceções.
- *Reset:* Nosso processador possui suporte a sinal de reset. Mais especificamente, se a entrada *reset* for acionada, o *program counter* é reiniciado para o endereço `0x0040_0000`. Este endereço foi escolhido tendo em vista compatibilizá-lo com o assembler MARS. Desta forma, o registrador PC, presente no *datapath*, deve ser inicializado neste endereço, e reiniciado sempre que o *reset* for acionado.
- *Enable:* Para auxiliar no processo de depuração, nós vamos incorporar um sinal de entrada *enable*. Quando ativo, o processador executa normalmente. Entretanto, quando *enable = 0*, o processador "congela". Esse procedimento é realizado desativando a escrita nos seguintes componentes: PC, *register file* e memória de dados. Tal modificação permitirá executar nossos programas passo-a-passo, auxiliando no processo de depuração.

2.1.1 A Unidade de Controle

No sentido de preparar o projeto de um processador MIPS completo, nós iniciaremos com o desenvolvimento da unidade de controle. A seguir são apresentados dois diagramas no nosso processador MIPS single-cycle, primeiro com uma visão de alto nível, e em seguida uma versão mais detalhada.



Abaixo apresentamos uma visão mais detalhada.



Vamos primeiro desenvolver a unidade de controle. Ela deve dar suporte a TODAS as instruções apresentadas a seguir:

- `lw` e `sw`
- `addi, addiu, slti, sltiu, ori, lui, andi, xori`
  - **Nota 1:** Diferente do que você possa ter aprendido, `addiu`, na verdade, não realiza uma soma sem sinal (*unsigned*). Na verdade, ela estende o sinal do imediato. A única diferença entre `addi` e `addiu` é que `addiu` não causa uma exceção na presença de um *overflow*, enquanto `addi` sim. Como não estamos implementando exceções, `addiu` e `addi` são idênticas para nossos propósitos.
  - **Nota 2:** Além disso, contrário ao que você possa ter aprendido, `sltiu` na verdade estende o sinal do imediato, *mas realiza uma comparação sem sinal*. Ou seja, a ALUFN é "LUT".
  - Note ainda que `ori` deve estender zero do imediato, por se tratar de uma operação lógica! Finalmente, a extensão do sinal para `lui` é um *don't-care*, uma vez que o imediato de 16-bits é posicionado na metade alta do registrador.
  - **Nota 3:** Os valores de **pcsel** dependerão da *flag* **Z**. Esta *flag* deve indicar se o resultado da operação aritmética é igual ou diferente de zero.
- R-type: `add, addu, sub, and, or, xor, nor, slt, sltu, sll, sllv, srl, sra`
  - **Note:** para nossos propósitos, a instrução `addu` é *idêntica* à instrução `add`. Elas diferem apenas na fomo como lidam com o *overflow*, que será ignorado no nosso projeto. A razão para darmos suporte à instrução `addu` é que o assembler MARS geralmente introduz automaticamente instruções `addu` no nosso código, especialmente para a faixa de endereços de memória que nós estamos usando.
- `beq, bne, j, jal` e `jr`

Primeiramente, estude os slides de aula sobre o processador MIPS single-cycle (incluindo os Slides 33–38). Em seguida preencha a Tabela 1 abaixo com os valores de todos os sinais de controle para as 28 instruções MIPS básicas listadas aqui. Se o valor de um sinal de controle não importa para uma dada instrução, você poderá usar o sinal *don't care* (1'bX, 2'bX, etc., dependendo da quantidade de bits).

A partir dos valores da Tabela 1, complete a descrição Verilog da unidade de controle no arquivo `controler.sv`, disponibilizado junto com os arquivos de laboratório.

Use o *testbench* fornecido para simular e validar o seu projeto. O *testbench* é auto-verificável, de modo que, se algum erro for identificado, este será sinalizado automaticamente no *waveform*. Para isso, certifique-se de utilizar os mesmos nomes, presentes no teste, para as entradas e saídas do *top-level*.

Type	Instr	werf	wdsel	wasel	asel	bsel	sext	wr	alufn	pcsel
I-Type	LW	1	10	01	00	1	1	0	0XX01	
	SW									
	ADDI									
	ADDIU									
	SLTI									
	SLTIU									
	ORI									
	LUI									
	ANDI									
	XORI									
	BEQ									Z=1 | Z=0
	BNE									Z=1 | Z=0
J-Type	J									
	JAL									
R-Type	ADD									
	ADDU									
	SUB									
	AND									
	OR									
	XOR									
	NOR									
	SLT									
	SLTU									
	SLL									
	SLLV									
	SRL									
	SRA									
	JR									

Tabela 1: Tabela de decodificação de instruções do MIPS.

2.2 Parte 1: Projete um processador MIPS single-cycle completo

Junte todas as partes para criar um processador MIPS single-cycle tal qual discutido em sala de aula. Os modelos em Verilog para alguns dos módulos de alto nível foram disponibilizados junto com os arquivos de laboratório. De forma mais específica, você deverá realizar as tarefas elencadas a seguir.

- **Projete um processador MIPS single-cycle completo, juntamente com as memórias de instruções e dados**, como apresentado nos slides de aula #37–38. Comece com o arquivo `top.sv`, fornecido junto com os arquivos de laboratório, o qual contém o módulo Verilog *top-level*. Perceba como ele corresponde ao diagrama de blocos apresentado na Página 3, e tome nota de todos os parâmetros presentes nele.

  Por enquanto, nós vamos escolher um tamanho relativamente pequeno para cada memória, por exemplo, 128 posições de memória para a memória de instruções e 64 para a memória de dados. *Observe:* Os endereços produzidos pelo processador para acessar tais memórias ainda continuarão sendo de 32 bits, mesmo que sejam utilizados menos bits de endereço. Use os módulos ROM e RAM do roteiro anterior (`rom.sv/ram.sv`), e perceba o seguinte:

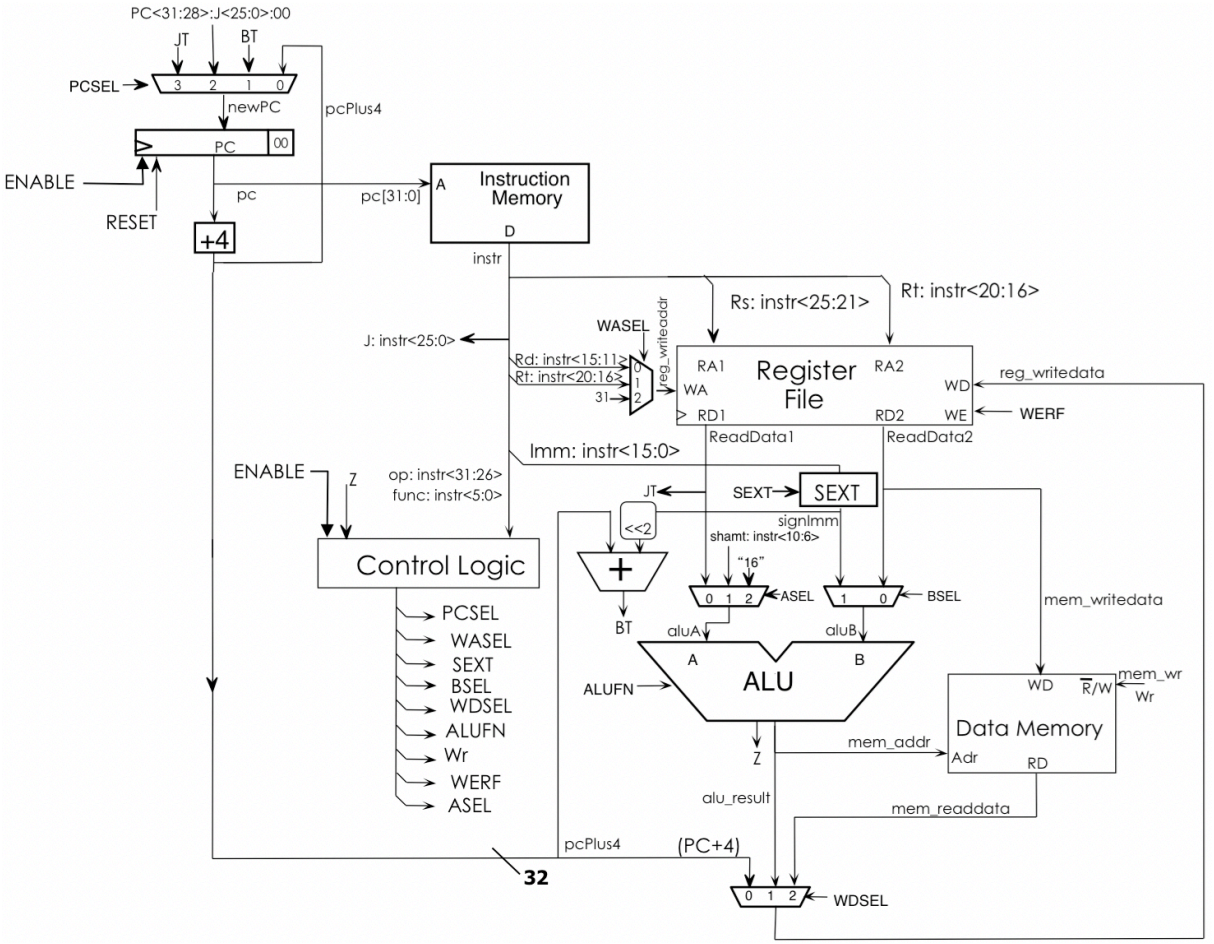
  - A memória de instruções será uma ROM, enquanto a memória de dados será uma RAM. Nós iremos instanciar os módulos ROM e RAM (sem modificar duas descrições Verilog), e fornecer os parâmetros apropriados, os quais, por sua vez, são definidos no módulo *top-level* de teste. (Ver `top.sv`).
  - Nós iremos enviar todos os 32 bits de PC para fora do processador e conectá-lo à memória de instruções, mas cortar os últimos dois bits na interface, de modo que apenas uma palavra de endereço de 30 bits é realmente enviada para dentro da memória de instruções. Isso irá converter um endereço de byte para um endereço de palavra. (Ver `top.sv`).
  - Ambas as memórias devem retornar uma palavra de 32 bits (`Dbits = 32`). Seus valores iniciais são lidos a partir do `initfile`, o qual corresponde ao nome do arquivo de inicialização da memória.

- **Inicialize as memórias de instruções e dados**, utilizando o método apresentado no Lab 7. O arquivo que contém os valores iniciais para a memória de instruções conterá uma instrução codificada de 32 bits por linha (em hexadecimal). O arquivo que contém os valores iniciais para a memória de dados também irá conter valores de dados de 32 bits a cada linha. Você pode criar estes arquivos dentro do próprio Quartus, ou utilizando seu editor de textos preferido.

- **O módulo do processador, contendo os módulos controlador e datapath,** foi fornecido no arquivo `mips.sv`. Entenda como ele se adéqua ao diagrama de blocos apresentado a seguir. *Observe* que não há diferenças entre o projeto do MIPS do livro de Harris e Harris e o que nós estamos desenvolvendo neste roteiro. Entretanto, nossa versão possui uma versão muito mais sofisticada de ALU. Portanto, não siga de forma irrestrita as informações presentes no livro; ao invés disso, siga os slides de aula e anotações dos laboratórios. O caminho de dados deve ser de 32 bits (registradores, ALU, memória de dados e memória de instruções, usam palavras de 32 bits).

- **Complete todas as pequenas peças**, de modo que o projeto se pareça com aquele apresentado na aula, também reproduzido a seguir. Pequenos conjuntos de código podem ser "aninhados" (no lugar de escritos em módulos separados), por exemplo, multiplexadores, extensor de sinal, somadores, deslacamento-por-2, etc.

As figuras na Página 3 apresentam uma decomposição hierárquica do projeto *top-level*. Atente-se que o seu projeto deve segue tal hierarquia.



Outras dicas que irão de ajudar no processo de depuração:

- **Lembre-se de usar a diretiva 'default_nettype none** para facilitar a identificação de declarações incompletas ou erros nos nomes devido a falhas tipográficas, etc.
- **É altamente recomendado que você utilize portas nomeadas,** especialmente para módulos que contenham múltiplas entradas/saídas. Do contrário será fácil cometer erros devido ao não alinhamento das portas, causando (mais) dores de cabeça na depuração. Você pode seguir o estilo apresentado no arquivo `mips.sv` para a conexão do controlador e do *datapath*.
- **Use o testbench para testar seu processador em simulação.** Um teste de auto-verificação foi fornecido junto com os arquivos de laboratório. Ele foi inicialmente escrito em assembly MIPS, e então compilado usando o assembler MARS e, finalmente, convertido para um código de máquina hexadecimal, o qual deve ser utilizado para inicializar sua memória de instruções. Certifique-se de inicializar o *program counter* (PC) dentro do projeto do MIPS no endereço 0x0040_0000, para que ele inicie a execução do começo da memória de instruções. Da mesma forma, para inicializar sua memória de dados, coloque todos os valores iniciais no arquivo correspondente.
  - O testador é chamado de **full** (`tester_full.sv`): O programa assembly (`full.asm`) executa cada uma das 28 instruções que nós implementamos, incluindo chamadas/retornos a procedimentos, e recursão utilizando uma pilha. Você não precisa executar

este programa no MARS, mas você pode. Ele possui um total de 59 instruções, portanto, a memória de instruções deve comportar, pelo menos, tal conjunto de dados. O testador define como 64 o parâmetro `Nloc` da memória de instrução. Além disso, no testador, certifique-se de especificar os nomes corretos para os arquivos que possuem os valores de inicialização. ("full_imem.mem" e "full_dmem.mem", respectivamente).

- **Por enquanto, você não irá implementar seu projeto na placa.** Você fará isso no próximo laboratório.
- **Comece pensando sobre o que você gostaria de construir no seu projeto final!** Cada projeto deve usar, prioritariamente, um monitor VGA como saída. As placas DE2-115 também possuem saídas de áudio. Teclados e mouse de entrada serão disponibilizados para os projetos. Outros dispositivos de entrada, além daqueles presentes na placa, poderão ser disponibilizados. Comece a pensar!

No mais, boa sorte!

3. Acompanhamento

Envie os artefatos solicitados a seguir de acordo com as datas estipuladas no SIGAA:

- **Parte 1:** o arquivo `controller.sv`, um documento da tabela de decodificação de instruções, e uma captura de tela do *waveform* da simulação, utilizando o teste de auto-verificação.
- **Parte 2:** TODOS os arquivos Verilog. Além disso, uma captura de tela dos *waveforms* da simulação para o testador "full" fornecido.
- **Parte 3:** um relatório técnico contendo uma revisão técnica do trabalho desenvolvido, explicitando os resultados obtidos, deficiências encontradas e um destaque para a contribuição individual de cada membro da equipe.