

Aluno: _____
Matrícula: _____ Data: _____

Sprint 6 – Memórias – CPU MIPS

Descrição geral do problema: Utilizando a ferramenta *megawizard plugin-manager*, do Quartus II, implemente uma nova memória ROM de instruções, assim como uma memória RAM de dados. Isso possibilitará o uso das instruções LW e SW.

Requisitos mínimos:

Abra o projeto da Sprint5 e edite-o para incluir as funcionalidades dessa sprint. **Obs: “File > Open Project” e NÃO “File > Open”.**

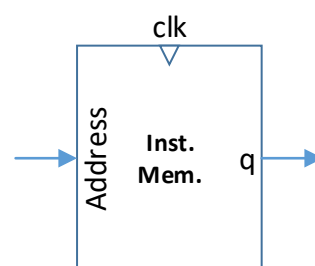
1. Criação de uma nova memória ROM de instruções, para substituir a memória implementada como um decodificador, na sprint anterior.

Criação do arquivo de inicialização da memória ROM

- *File/New/Memory Initialization File*
- *Number of words: 256*
- *Word size: 32*
- Salvar o arquivo com o seguinte nome: *RomInstMem_init.mif*
- *View/Cells Per Row: 1*
- *View/Memory Radix: Binary*
- Copie e cole os binários das instruções a serem executadas no programa de teste da Tabela 1

Criação da memória ROM de instruções

- *tools/megawizard plugin-manager*
- Create a new custom megafunction variation
- Selecione a opção de memória ROM de uma porta (ROM: 1-PORT)
- Selecione sua pasta pessoal e nomeie o arquivo como *RomInstMem*
- Largura da palavra (**32 bits**) e a quantidade de palavras da memória (**256 palavras**)
- Desmarque a opção ‘q’ output port
- Inicialize a memória com o arquivo *RomInstMem_init.mif* criado anteriormente
- Finish



Perceba que foi criado um arquivo *RomInstMem.qip* na árvore de arquivos do seu projeto. Clique na seta “>” e visualize o arquivo *RomInstMem.v*

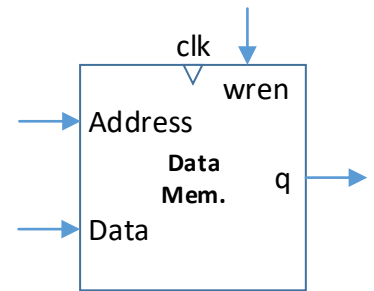
Endereço	Assembly	Código de máquina
00	ADDi \$1, \$0, AB	00100000000000010000000010101011
01	SW \$1, 9(\$0)	10101100000000010000000000001001
02	LW \$2, 9(\$0)	10001100000000010000000000001001

Tabela 1 –programa teste

2. Criação de uma nova memória RAM de dados. A partir de agora, será possível armazenar e carregar o conteúdo de registradores, através das instruções SW e LW.

Criação da memória RAM de dados

- *tools/megawizard plugin-manager*
- Create a new custom megafunction variation
- Selecione a opção de memória RAM de uma porta (RAM: 1-PORT)
- Selecione sua pasta pessoal e nomeie o arquivo como *RamDataMem*
- Largura da palavra (**8 bits**) e a quantidade de palavras da memória (**256 palavras**)
- Desmarque a opção 'q' output port
- Finish



Perceba que foi criado um arquivo *RamDataMem.qip* na árvore de arquivos do seu projeto. Clique na seta ">" e visualize o arquivo *RamDataMem.v*

3. A fim de completar a segunda versão da CPU v0.2, todos os módulos desenvolvidos até agora devem ser **instanciados** e **conectados** conforme o circuito da Figura 1.
 - Substitua a memória de instruções implementada na sprint anterior pela memória ROM criada agora (módulo *RomInstMem.v*)
 - Instancie o **MuxDDest**
 - Instancie a memória RAM de dados (módulo *RamDataMem.v*)
 - Os fios devem ser criados com uma nomenclatura lógica para facilitar o entendimento geral do circuito e facilitar o debug.
 - Essa CPU v0.2 será capaz de rodar as instruções ADD, SUB, AND, OR, SLT, ADDi, LW e SW.

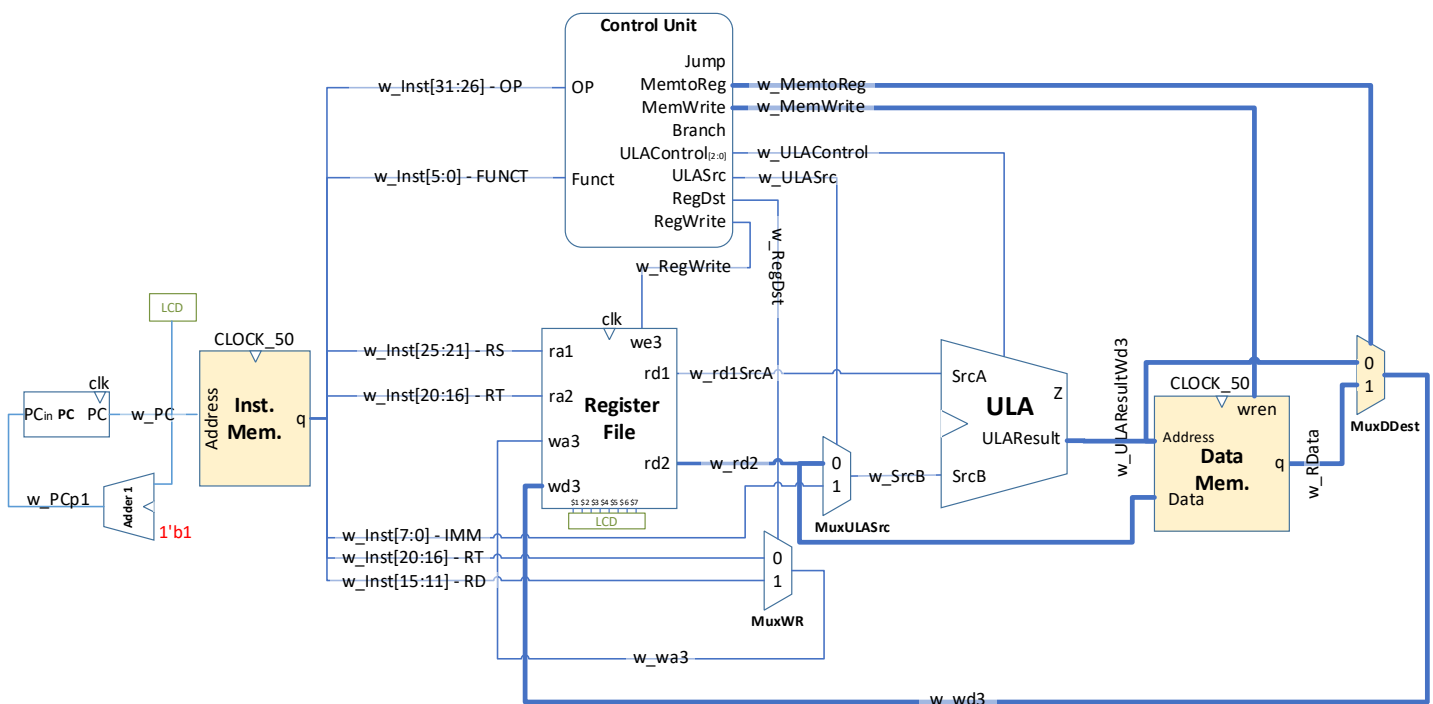


Figura 1 – CPU V0.2 e memórias

Nome	Tamanho
w_MemtoReg	1 bit
w_MemWrite	1 bit
w_wd3	8 bit
w_RData	8 bit

Tabela 2 – novos fios, utilizados na montagem

ALERTA: A CPU de ciclo único necessita de memórias puramente combinacionais. Como a ferramenta do Quartus II somente cria memórias com no mínimo um registro na entrada, o comportamento combinacional será emulado conectando um clock muito mais rápido nessas memórias.

O sinal de 50 MHz, que é disponibilizado na porta de entrada CLOCK_50 do *Mod_teste*, deverá ser ligado às portas de entrada “clock” de ambas as memórias. Já o clock principal da CPU (PC e Register File), deverá ser de 1Hz, gerado a partir de um divisor de frequência. Os pontos de clock estão resumidos na Tabela 3.

Módulo	Frequência do clock
PC	1 Hz
Register File	1 Hz
Instruction Memory	50 MHz
Data Memory	50 MHz

Tabela 3 – Frequência dos clocks

Relembrando da tabela do decodificador de instruções:

Instr	ENTRADAS		SAÍDAS							
	OP	Funct	RegWrite	RegDst	ULASrc	ULAControl	Branch	MemWrite	MemtoReg	Jump
ADD	000000	100000	1	1	0	010	0	0	0	0
SUB	000000	100010	1	1	0	110	0	0	0	0
AND	000000	100100	1	1	0	000	0	0	0	0
OR	000000	100101	1	1	0	001	0	0	0	0
SLT	000000	101010	1	1	0	111	0	0	0	0
LW	100011	xxxxxx	1	0	1	010	0	0	1	0
SW	101011	xxxxxx	0	x	1	010	0	1	x	0
BEQ	000100	xxxxxx	0	x	0	110	1	0	x	0
ADDi	001000	xxxxxx	1	0	1	010	0	0	0	0
J	000010	xxxxxx	0	x	x	xxx	x	0	x	1

Tabela 4 – Tabela do decodificador da Unidade de Controle

Relembrando o conjunto de instruções suportadas pela CPU

Instrução	Descrição	Algoritmo
ADD \$X, \$Y, \$Z	Adicionar	$\$X = \$Y + \$Z$
SUB \$X, \$Y, \$Z	Subtrair	$\$X = \$Y - \$Z$
AND \$X, \$Y, \$Z	AND Bit a bit	$\$X = \$Y \& \$Z$
OR \$X, \$Y, \$Z	OR Bit a bit	$\$X = \$Y \$Z$
SLT \$X, \$Y, \$Z	Menor que	$\$X = 1$ se $\$Y < \Z e 0 c.c.
LW \$X, i(\$Y)	Carregar da memória	$\$X \leftarrow \text{Cont. do end. } (\$Y + i)$
SW \$X, i(\$Y)	Armazenar na memória	$\text{End. } (\$Y + i) \leftarrow \X
BEQ \$X, \$Y, i	Desviar se igual	Se $\$X == \Y , $\text{PC} = \text{PC} + 1 + i$
ADDi \$X, \$Y, i	Adicionar Imediato	$\$X = \$Y + i$
J i	Desvio incondicional	$\text{PC} = i$

Tabela 5 –Conjunto de instruções MIPS suportadas pela CPU do LASD

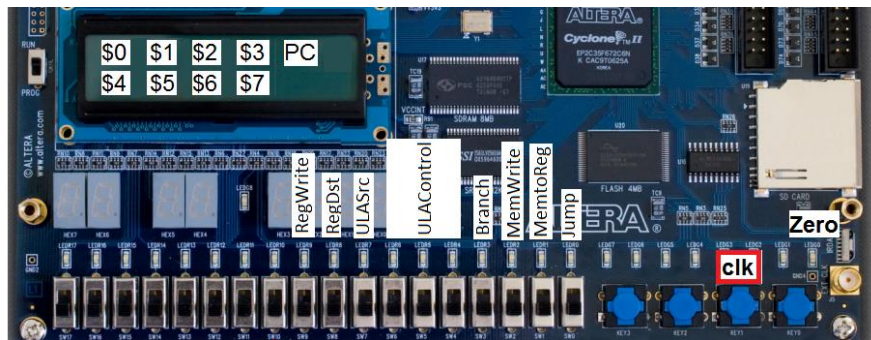


Figura 5 – Placa

4. Rode o programa da Tabela 1 e diga qual o conteúdo dos registradores e da memória de dados, ao finalizá-lo:

Registradores:

Registrador	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7
Dado								

Memória de dados:

Endereço	0x00	0x01	0x02	0x03	...	0x08	0x09	0x0A	0x0B
Dado									

Desafio (Valendo +0,1 na média geral)

- Escreva um programa para carregar as constantes C, A, F e E, respectivamente nos endereços 0, 1, 2 e 3 da memória de dados. Para checar se realmente foram escritos, dê um echo no registrador \$1.