ES575 - Laboratório de circuitos lógicos

Projeto: Processador Simples

Prof. André R. Fioravanti

A Figura 1 mostra um sistema digital que contém um certo número de registradores de 16 bits, um multiplexador, uma unidade somadora/subtratora e uma unidade de controle (máquina de estados finitos). Dado é inserido neste sistema através da entrada de 16 bits DIN. Este dado pode ser carregado através do multiplexer de 16 bits nos diversos registradores, como $R0, \ldots, R7$ e A. O multiplexer também permite que os dados sejam transferidos de um registrador para outro. A conexão de saída de um multiplexer é conhecido como barramento, pois este termo é normalmente utilizado para as conexões que permitem que os dados sejam transferidos de um lugar do sistema para outro.

Soma e subtração é executada utilizando um multiplexer para inicialmente colocar um número de 16 bits no barramento e carregar este número no registrador A. Uma vez feito isso, um segundo número é colocado no barramento, a unidade de soma/subtração efetua a operação necessária, e o resultado é carregado no registrador G. O dado em G pode então ser transferido para um dos outros registradores se necessário.

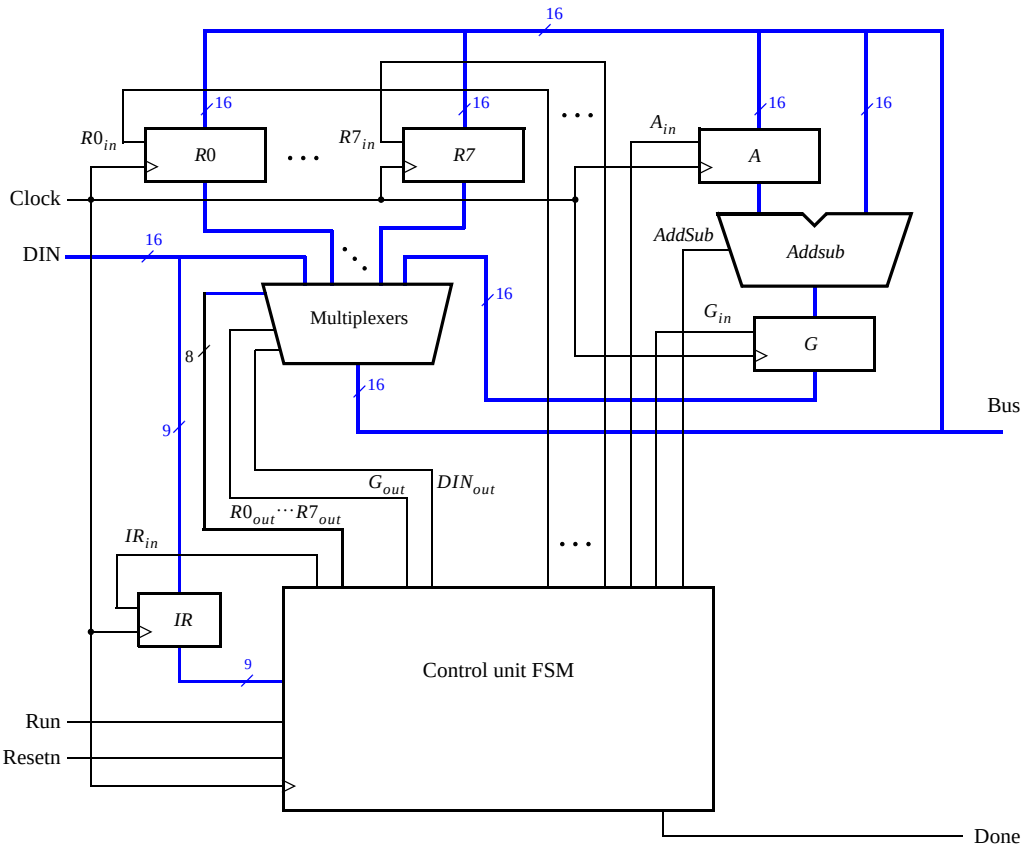

Figura 1: Um sistema digital.

O sistema pode realizar diferentes operações em cada ciclo de relógio, como controlado pela *unidade de controle*. Esta unidade determina quando cada dado particular é colocado no barramento e controla quais dos registradores devem carregar este dado. Por exemplo, se a unidade de controle ativa os sinais $R0_{out}$ e A_{in}, então

o multiplexador irá colocar o conteúdo do registrador $R0$ no barramento e este dado será carregado no próxima borda do relógio para o registrador *A*.

Um sistema como este é normalmente denominado um *processador*. Ele executa operações específicas na forma de instruções. A Tabela 1 lista as instruções que o processador tem que executar neste exercício. A coluna da esquerda mostra o nome da instrução e seus operandos. O significado da sintaxe RX ← [RY] impõe que o conteúdo do registrador RY é movido para o registrador RX. A instrução **mv** (move) permite que dado seja copiado de um registrador para outro. Para a instrução **mvi** (move immediate), a expressão RX ← D indica que a constante de 16 bits D é carregada no registrador RX.

Operação	Função Executada	Opcode
mv Rx,Ry	$Rx \leftarrow [Ry]$	000
mvi Rx,#D	$Rx \leftarrow D$	001
add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$	010
sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$	011

Tabela 1. Instruções executadas no processador.

Cada instrução pode ser codificada e armazenada no registrador *IR* usando o formato de 9 bits IIIXXYYY, onde III representa a instrução, XXX fornece o registrador RX e YYY o registrador RY. Apesar de apenas 2 bits serem necessários para codificar nossas 4 instruções, estamos usando 3 bits pois outras instruções serão adicionadas no processador em futuras partes do exercício. Assim sendo, IR precisa ser conectado em 9 bits dos 16 existentes da entrada *DIN*, como indicado na Figura 1. Para a instrução **mvi**, o campo YYY não tem significado, e o dado imediato #D precisa ser fornecido pela entrada de 16 bits *DIN* após a palavra da instrução **mvi** ser carregada em *IR*.

Algumas instruções, como soma ou subtração, levam mais do que um ciclo de relógio para serem executadas, pois diversas transferências precisam ser realizadas através do barramento. A máquina de estados finitos na unidade de controle "caminha" pelas instruções, ativando os sinais de controle necessários nos sinais de clock sucessivos até que a instruções esteja completa. O processador começa executando a instrução na entrada *DIN* quando o sinal *Run* estiver ativo, e o processador ativa a saída *Done* quando a instrução terminar. A Tabela 2 indica os sinais de controle que podem ser ativados em cada pulso de relógio para implementar as instruções na Tabela 1. Note que o único sinal de controle ativo no tempo 0 é IR_{in}, então este tempo não será apresentado na tabela.

	T_1	T_2	T_3
(**mv**): I_0	RY_{out}, RX_{in}, *Done*		
(**mvi**): I_1	DIN_{out}, RX_{in}, *Done*		
(**add**): I_2	RX_{out}, A_{in}	RY_{out}, G_{in}	G_{out}, RX_{in}, *Done*
(**sub**): I_3	RX_{out}, A_{in}	RY_{out}, G_{in}, *AddSub*	G_{out}, RX_{in}, *Done*

Tabela 2: Sinais de Controle ativos em cada instrução/pulso de relógio.

Parte I

Projete e implemente o processador apresentado na Figura 1.

Parte II

Nesta parte iremos projetar o circuito representado na Figura 4, na qual um módulo de memória e um contador estão conectados ao processador da Parte I. O contador é utilizado para ler o conteúdo de endereços de memória sucessivos, e estes dados são fornecidos ao processador como um fluxo de instruções. Para simplificar o projeto e os testes do circuito, usaremos sinais separados de clock, *PClock* e *MClock*, para o processador e a memória.

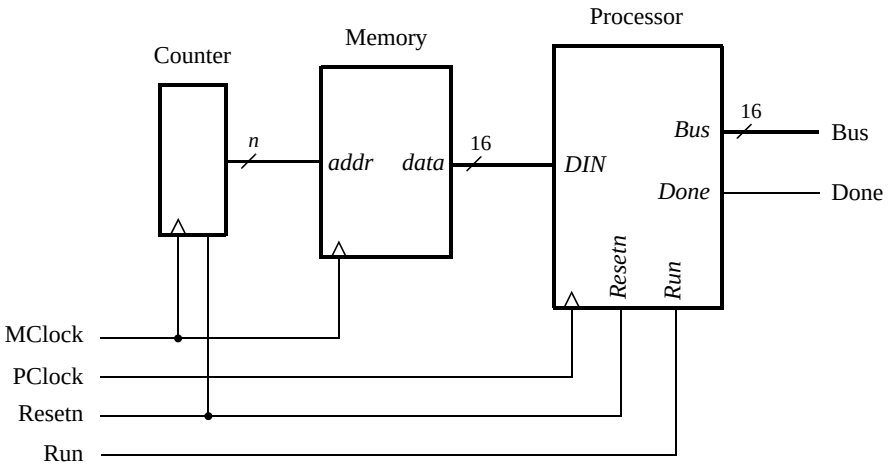

Figure 4. Conectando o processador em uma memória e um contador.

1. Crie um novo projeto no Quartus II para testar o circuito.

2. Gere um arquivo VHDL para definir o processador, a memória e o contador. Use o MegaWizard Plug-in Manager para criar o módulo da memória a partir das LPMs. A LPM correta se encontra sob *Memory Compiler–ROM: 1-PORT*. Siga as instruções para criar uma memória de leitura com 32 palavras de 16 bits. Como esta memória possui apenas uma porta de leitura, ela é denominada *ROM síncrona*. Note que a memória inclui um registrador para o carregamento de endereços síncronos. Considere este fato no seu projeto.

   Para inserir as instruções para o processador na memória, é necessário fornecer *valores iniciais* que devem ser armazenados na memória no momento da programação da FPGA. Isso pode ser realizado ao informar para o wizard que o conteúdo de memória deve ser obtido em um aruqivo MIF *memory initialization file*, como ilustrado na Figura 6. Use a ajuda on-line do Quartus II para aprender sobre o formato do arquivo MIF e criar um arquivo suficientemente grande para testar o processador.

3. Crie um novo projeto Quartus II que será usado para implementar o circuito na placa Altera DE2i-150. Use a chave SW_{17} como a entrada *Run*. Use o botão KEY_0 para *Resetn*, KEY_1 para *MClock* e KEY_2 para *PClock*. Conecte o barramento do processador em $LEDR_{15-0}$ e conecte o sinal Done em $LEDR_{17}$.

4. Teste e apresente a funcionalidade do seu processador.

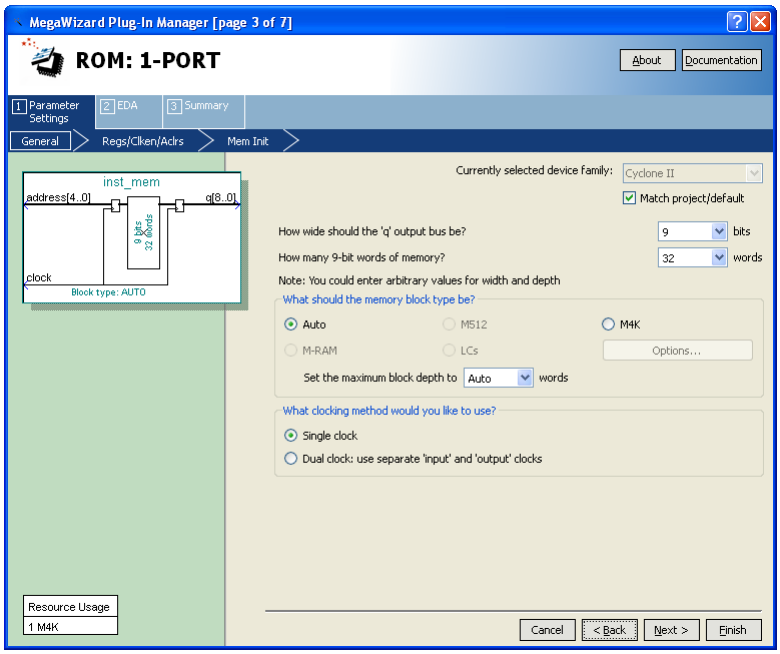

Figura 5. Configuração da 1-PORT.

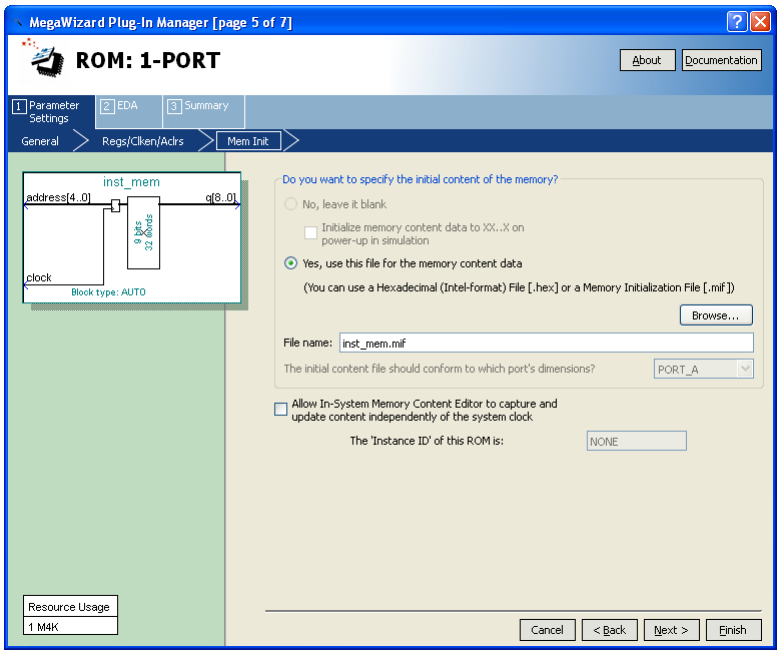

Figura 6. Especificando um arquivo (MIF).

Parte III

Nesta parte vamos estender a capacidade do processador para que o contador externo não seja mais necessário e também que o processador seja capaz de realizar operações de leitura e escrita em memória ou outros dispositivos. Você deverá adicionar três novas instruções para o processador, como apresentado na Tabela 3. A instrução

ld (load) carrega no registrador RX dados da memória externa com endereço especificado no registrador RY. A instrução **st** (store) carrega o dado contido no registrador RX no endereço de memória especificado em RY. Finalmente, a instrução **mvnz** (move if not zero) permite que uma operação **mv** seja executada apenas sob uma certa circunstância, a de que o conteúdo atual do registrador G não seja 0.

Operação	Função Executada	Opcode
ld Rx,$[Ry]$	$Rx \leftarrow [[Ry]]$	100
st Rx,$[Ry]$	$[Ry] \leftarrow [Rx]$	101
mvnz Rx, Ry	if G != 0, $Rx \leftarrow [Ry]$	110

Tabela 3. Novas instruções executadas no processador.

O esquemático desta versão do processador é dado na Figura 7. Nesta figura, os registradores $R0$ até $R6$ são os mesmos da Figura 1, mas o registrador $R7$ foi alterado para um contador. O contador é usado para fornecer o endereço na memória no qual a instrução do processador será lida: na parte anterior, um contador externo tinha este papel. Iremos nos referir portanto ao registrador $R7$ como contador de programa (*PC*). Quando o processador é resetado, *PC* contém o valor 0. No início de cada instrução (no passo 0), o conteúdo de *PC* é usado como endereço para ler uma instrução da memória. A instrução é salva em *IR* e *PC* é automaticamente incrementado para apontar para a próxima instrução (em caso da instrução **mvi** o *PC* também fornece o endereço do dado imediato e depois é incrementado de novo).

A unidade de controle do processador incrementa *PC* através do sinal *incr_PC*, que é simplesmente um enable para o contador. Também é possível carregar um endereço para *PC* ($R7$) ao fazer o processador executar uma instrução **mv** ou **mvi** com destino especificado para $R7$. Nesse caso, a unidade de controle utiliza o sinal $R7_{in}$ para executar um carregamento paralelo do contador. Assim sendo, o processador pode executar instruções em qualquer posição da memória, em vez de poder apenas executar operações que estejam em endereços sucessivos. De forma análoga, o conteúdo de *PC* pode ser copiado para qualquer outro registrador através da instrução **mv**. Um exemplo de código que utiliza o registrador *PC* para implementar um laço é mostrado abaixo. A instrução **mv** R5,R7 coloca em *R5* o endereço de memória da instrução sub **sub** R4,R2. A instrução **mvnz** R7,R5 define que a instrução sub seja executada repetidamente, até *R4* se tornar 0. Este tipo de laço pode ser usado em um programa para criar-se um atraso.

```
mvi   R2,#1
mvi   R4,#10000000   % binary delay value
mv    R5,R7          % save address of next instruction
sub   R4,R2          % decrement delay count
mvnz  R7,R5          % continue subtracting until delay count gets to 0
```

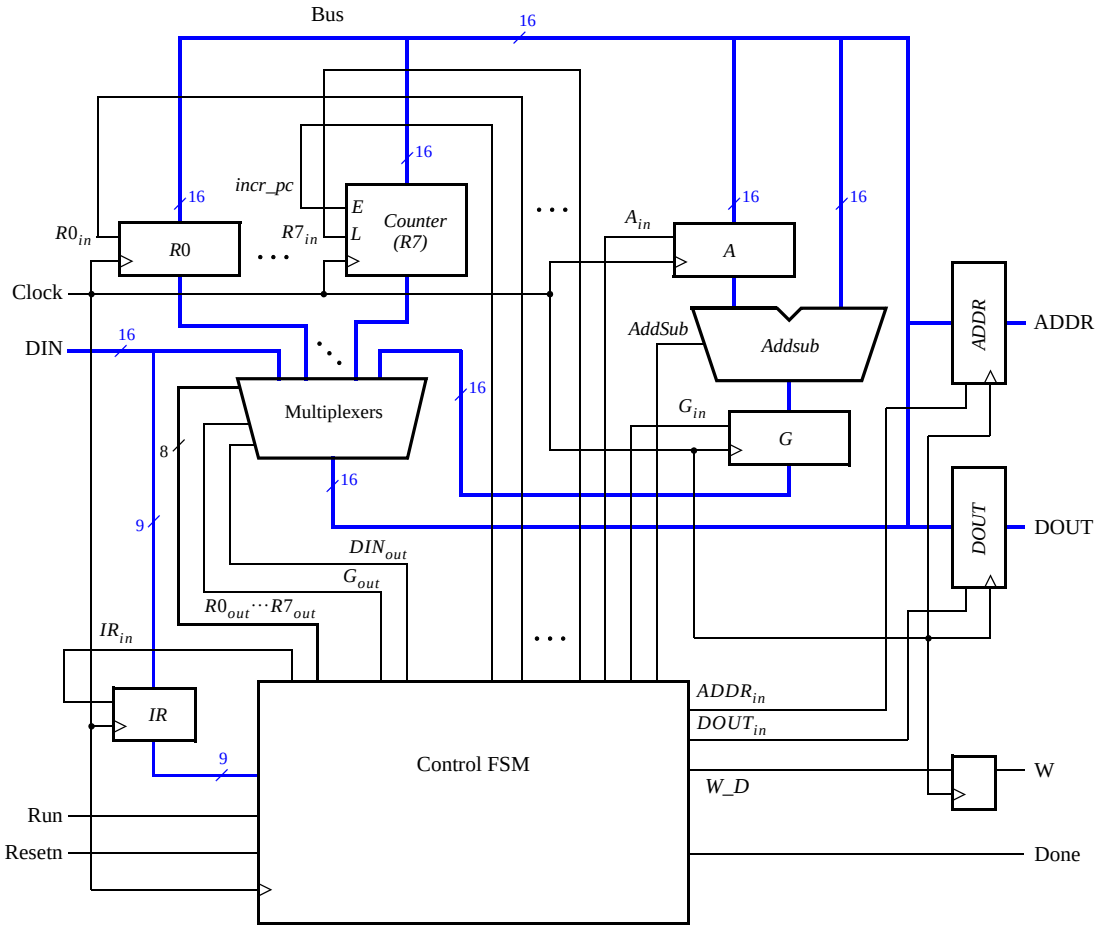

Figura 7. Processador com novas capacidades.

A Figura 7 mostra 2 registradores no processador que são usados para transferência de dados. O registrador *ADDR* é usado para enviar endereços para um periférico externo, como um módulo de memória, e o registrador *DOUT* é usado pelo processador para fornecer dados que possam ser armazenados fora do processador. Um dos usos do registrador *ADDR* é ler (ou *fetch*) instruções da memória; quando o processador deseja fazer o fetch de uma instrução, o conteúdo de *PC* ($R7$) é transferido pelo barramento e carregado em *ADDR*. Este endereço é fornecido à memória. Além de fazer a leitura de instruções, o processador pode ler dado de qualquer endereço usando o registrador *ADDR*. Tanto dados quanto instruções são colocadas para o processador através da porta *DIN*. O processador pode escrever dado para armazenamento em um endereço externo ao se colocar este endereço em *ADDR*, o dado a ser armazenado em *DOUT* e ativando a saída do flip-flop W (write) para 1.

A Figura 8 ilustra como este processador é conectado à memória ou outros periféricos. A unidade de memória na figura permite tanto operações de leitura quanto de escrita e portanto possui tanto entrada de dado quanto de endereço. A memória também possui uma entrada de relógio, pois o endereço, dado e sinal de enable precisam ser carregados na memória em uma borda de subida do relógio. Este tipo de memória é normalmente chamada de *RAM síncrona*.

A Figura 8 inclui um registrador de 16 bit que pode ser usado para armazenar dados do processador. Este registrador pode ser usado para armazenar dados do processador; este registrador pode ser conectado a um conjunto de LEDs. Para permitir que o processador possa selecionar seja a unidade de memória seja o registrador quando estiver executando uma operação de escrita, o circuito inclui algumas portas lógicas para permitir a decodificação do endereço. Se as maiores linhas de endereço forem $A_{15}A_{14}A_{13}A_{12} = 0000$, então o módulo de memória será escrito no endereço das linhas menores de endereço. A Figura 8 mostra n linhas de memória conectadas à memó-

ria. Para este exercício, uma memória de 128 palavras é provavelmente suficiente, de forma que $n = 7$, e a porta de endereços da memória contém então $A_6 \ldots A_0$. Para endereços que $A_{15}A_{14}A_{13}A_{12} = 0001$, o dado escrito pelo processador é carregado no registrador cujas saídas são os *LEDs*. Portanto:

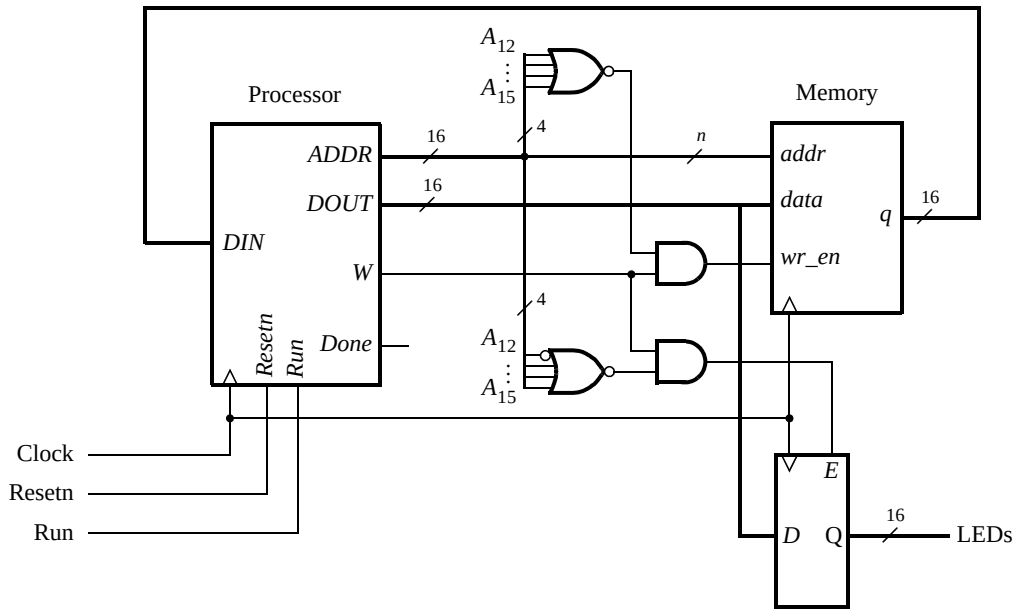

Figure 8. Connecting the enhanced processor to a memory and output register.

1. Crie um novo projeto no Quartus II.

2. Escreva o código VHDL para este processador e teste o circuito usando simulação funcional: aplique instruções na porta *DIN* e observe os sinais internos do processador conforme as operações forem sendo executadas. Atente-se cuidadosamente à temporização dos sinais entre seu processador e a memória externa; leve em conta que a memória possui registradores na porta de entrada.

3. Crie outro projeto no Quartus II que crie seu processador, um módulo de memória e um registrador, como mostrado na Figura 8. Use o MegaWizard Plug-in Manager para criar um módulo RAM: 1 – PORT. Crie uma memória que tenha uma porta de 16 bits de leitura/escrita e possua 128 palavras. Use um arquivo .mif para armazenar instruções na memória que serão executadas pelo processador.

4. Utilize simulação funcional para testar o circuito. Garanta que a leitura e escrita na RAM estejam funcionando corretamente.

5. Utilize SW_{17} como entrada *Run*, KEY_0 como *Resetn*, e use o clock de 50MHz interno à placa como sinal *Clock*. Como o circuito precisa executar corretamente em 50MHz, garanta que uma restrição de tempo é inserida no Quartus II nesta frequência. Leia o relatório produzido pelo Quartus II Timming Analyzer para garantir que o circuito opere nesta frequência, caso contrário, use as ferramentas do Quartus II para modificar seu código VHDL para fazer um projeto mais eficiente que respeite esta restrição. Note também que a porta de entrada *Run* é assíncrona, então garanta que esta entrada seja sincronizada utilizando flip-flops. Finalmente, conecte o registrador de LEDs para $LEDR_{15-0}$ para que você possa observar a saída produzida pelo processador.

6. Compile e teste seu projeto executando código da RAM e observando os LEDs.

Parte IV

Nesta parte, iremos conectar outros módulos de I/O no circuito da Parte III e escrever código que será executado pelo processador.

Adicione um módulo chamado *seg7_scroll* no circuito. Este módulo deverá conter um registrador para cada display de 7 segmentos. Cada registrador deverá ativar diretamente as linhas de um display, de forma que o processador possa escrever caracteres neles. Crie a decodificação de endereço necessária para permitir que o processador escreva nestes displays.

1. Crie um projeto Quartus II para o circuito e faça o código VHDL que inclua o circuito da Figura 8 e o módulo *seg7_scroll*.
2. Teste o circuito com simulação funcional.
3. Inclua as restrições temporais adequadas e escreva um arquivo .mif adequado que permita testar a habilidade do processador em escrever nos displays. Para isso, crie um programa que faça uma palavra "rolar" entre os displays.
4. Implemente e teste a funcionalidade do projeto executando código da RAM e observando os diplays.

Parte V

Adicione ao circuito da Parte IV outro módulo, chamado *port_n*, que permita ao processador ler o estado de algumas chaves da placa. O valor das chaves devem ser guardados em um registrador que possa ser possível ler através da instrução **ld**. Deve-se fazer a decodificação de endereços e criar multiplexadores para permitir que o processador leia tanto da RAM quanto da unidade de *port_n*, de acordo com o endereço usado.

1. Desenhe um diagrama mostrando como a unidade *port_n* é incorporada ao circuito;
2. Crie um projeto Quartus II, escreva o código VHDL e o arquivo .mif que mostre o uso do módulo *port_n*. Teste com um programa que faça uma palavra "rolar" entre os displays com velocidade dependente dos estados das chaves.
3. Teste o circuito tanto por simulação quanto implementando na placa.