Instituto Federal Fluminense

Lucas Diniz Santos Henrique, Ronald Soares Lopes, Sara da Cunha Monteiro de Souza
Engenharia de Computação, 7º Período
Disciplina de Sistemas Embarcados

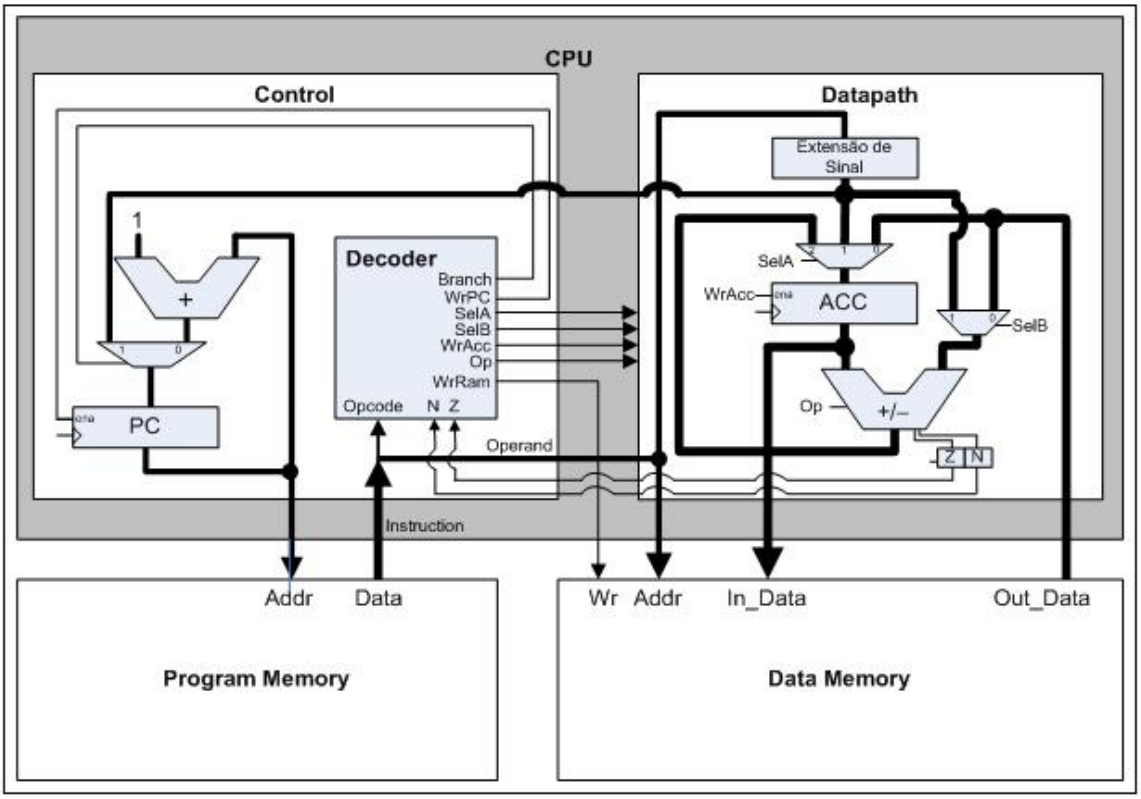
Processador BIP II

13 de setembro de 2018

VISÃO GERAL

Os processadores BIP constituem uma família de processadores, desenvolvida por pesquisadores do Grupo de Sistemas Embarcados e Distribuídos da Universidade do Vale do Itajaí (UNIVALI). Sua criação objetiva a contribuição para o ensino de conceitos sobre programação e funcionamento de sistemas computacionais.

Neste documento, será abordada a segunda versão desses processadores, o BIP II, que utiliza uma arquitetura baseada na arquitetura RISC do microcontrolador PIC (Programmable Intelligent Computer) e na arquitetura Harvard, já que as duas memórias utilizadas são diferentes e independentes em termos de barramento e ligação ao processador.



ESPECIFICAÇÕES

Cada passagem de tempo descrita equivale a 100ns.

Todos os valores descritos abaixo são valores binários.

MÓDULOS DO PROCESSADOR

Datapath

Um caminho de dados (datapath) é uma coleção de unidades funcionais, como unidades lógicas aritméticas, que executam operações de processamento de dados, registros e barramentos. Ela inclui os circuitos necessários para executar a instrução.

Teste

As entradas do Datapath foram inicializadas com 0, com exceção do **operand_i**, que recebeu o valor 1. Na faixa dos 200ns, o valor de **selA_i** é alterado para 1, assim como o de **WRACC_i**, que faz a seleção do valor vindo do Extensor de Sinal e o guarda no ACC. A próxima subida de clock faz com que o ACC mande o seu valor para o ambos **dm_in_data** e ULA. Na faixa dos 400ns, o valor de **op_i** (operação) é mudado para 1 (subtração). O **selA_i** recebe o valor 10, fazendo a saída da ULA (de valor 1) ser jogada no ACC, saída essa que será subtraída pelo novo valor de **dm_out_data**, que é 10. O resultado será o número negativo -1, representado em complemento a 2, que faz com que a **flagN_o** seja levantada.

Control

O setor de controle (Control) contém uma coleção de unidades funcionais responsáveis pela identificação e interpretação de cada instrução, bem como o controle de qual instrução está sendo executada e qual será a próxima instrução a ser carregada. Esse setor inclui os módulos de contador de programa (PC), Decoder e incrementer.

Teste

Todas as entradas do módulo de controle foram inicializadas com zero, em seguida na primeira subida do clock o valor de DATA_im_i mudou para 0001100000000010 e Addr_ext_i para 00000000010 indicando que se deseja realizar uma operação LDI 2 e deve-se verificar se o valor de WrAcc foi para 1 e se SelA também foi para 1 indicando que o valor 2 em teoria foi carregado no registrador. Na subida seguinte se realiza uma operação SUBI onde o valor de DATA_im_ é alterado para 0011100000000001 e Addr_ext_i para 00000000001, o valor de PC deve ir para 1 indicando q foi incrementado e os valores de SelA para 0, SelB para 1 e WrAcc permanecendo em 1 o que indica q o valor em ACC foi subtraído pelo valor q estavo em Addr_ext_i. No pulso seguinte se deseja realizar um branch, o valor de DATA_im_i foi definido como 0100011111111111, z_i como 1 e Addr_ext_i como 00000111111, como forma de se verificar o valor de PC deve, no pulso de clock seguinte, mudar para o valor vindo de Addr_ext_i.

Incrementer

Trata-se de um incrementador que recebe a saída do registrador PC e adiciona à ela o valor 1.

Teste

A saída do PC, que é o valor de entrada do módulo Incrementer, recebe valores a cada 100ns e verifica-se se o valor de saída foi o recebido na entrada somado à 1.

PC

É um registrador chamado de contador de programa, que é utilizado para armazenamento de dados durante uma operação.

Teste

Sua entrada é a saída do Mux 2x1, que é inicializada com um determinado valor, para a verificação do funcionamento do registrador, que registra o valor recebido durante a borda de subida do clock e quando o **enable_i** está em 1.

Decoder

O decoder é o coração do processador. Entre outras palavras, o decoder junto ao clock controla toda a execução. De acordo com a literatura consultada, decodificadores apresentam basicamente três estados: **Busca**, **Decodificação** e **Execução**. As três fases completas executam uma única instrução.

Ainda de acordo com a literatura, comumente esses estados são descritos junto a um conjunto de variáveis responsáveis pelo gerenciamento de memória, barramento de endereços e de dados, entre outras funcionalidades. A maior parte dessas variáveis não foram usadas nesse trabalho porque o objetivo era construir um processador fiel ao modelo BIP2 de acordo com a documentação[1] do BIP2 e de acordo com o modelo arquitetural[1] de seus

componentes. Para exemplificar: O BIP2 não possui RI (Registrador de Instrução), responsável por armazenar a instrução atual que está sendo executada, portanto a literatura referência comumente cita RI, entre outras flags e sinais de controle, que não estão presentes na atual implementação. É importante portanto salientar que esse processador é um processador fictício e que, por isso, possui uma lógica de controle tipicamente diferente da encontrada na literatura.

Com as considerações acima, a implementação do decoder para este trabalho contou com os 4 estados descritos a seguir. Cada estado consome um ciclo de clock completo da CPU:

- **Busca:**

  O estado de busca compreende todas as ações necessárias para o carregamento da próxima instrução, vinda da ROM, para o módulo de controle dentro do processador. Dentro do módulo de controle, a instrução completa é dividida em **OpCode** (os primeiros 5 bits) e **Operando** (os outros 11 bits) e encaminhada ao decodificador e ao módulo de dados respectivamente. Em outras palavras, esse estado é responsável por colocar o OpCode a ser decodificado no decodificador. No final desse estado o sinal de controle de escrita para o PC é desabilitado, isso faz com que nos próximos ciclos de clock, o PC não seja incrementado até que um novo estado de busca seja alcançado, isso permite que o OpCode e o Operando não sejam alterados durante toda a execução da instrução atual.

  Como o modelo arquitetural proposto representa uma arquitetura de Harvard, isto é, memória para dados e para instrução separadas. A ROM contém apenas as instruções. Além disso, como as únicas interfaces também definidas pelo modelo foram o endereço da instrução (entrada) e os dados de saída para leitura do conteúdo naquele endereço de memória (saída) e não teve a presença de outros sinais de controle de memória, a lógica da memória era

atualizar a saída de dados sempre que um endereço de entrada fosse modificado, por essa razão, a preocupação em não sobrescrever o valor do PC. Caso, ele fosse sobrescrito, automaticamente o valor da nova instrução iria ser carregado para o decodificador.

- **Decodificação**

O estado de decodificação compreende todas as ações necessárias para a leitura e entendimento do OpCode (decodificação) e preparação para a execução.

Durante o ciclo de decodificação o decodificador prepara todos os sinais de controle significativos para a execução de acordo com o OpCode analisado. Esses sinais de controle são todos saída do decodificador e abrangem:

Wrpc_o: O Sinal que habilita a atualização da instrução através da escrita no PC. Esse sinal é habilitado no final do estado de execução e resetado no final do estado de busca. Deve ser observado que ele é preservado resetado durante todo o estado de decodificação e no primeiro estágio do estado de execução para não haver atualizações inesperadas.

Wrram_o: Habilita a escrita na memória RAM do conteúdo presente na interface de entrada de dados, que é a saída de ACC, na posição de memória presente na interface de endereço, que é o operando.

Wracc_o: Habilita a atualização do valor de saída de ACC. Esse sinal é habilitado no final do estado de decodificação (segundo estágio), nesse momento espera-se que todos os outros bits de seleção como SelA_o, SelB_o, já tenham sido configurados no primeiro estágio da decodificação e os valores já estejam preparados para que no próximo estado (o de execução), ao habilitar este sinal no segundo estágio da decodificação, a escrita seja feita com o valores corretos. O mesmo acontece com o **Wrram_o.** Ambos os sinais são posteriormente

resetados no fim do estado de execução para não haver escritas equivocadas tanto de ACC quanto da RAM.

SelA_o: Seleciona a entrada que será encaminhada ao **ACC**, que por sua vez quando for habilitado por **Wracc_o** encaminhará esse valor para a 1ª entrada da ULA. Esses bits de seleção podem escolher entre a saída da ULA, o extensor de sinal (operando) ou um valor proveniente da RAM na posição de memória do operando.

SelB_o: Seleciona qual valor irá para a 2ª entrada da ULA, podendo ser um valor proveniente da RAM na posição de memória do operando ou o extensor de sinal (operando) .

Op_o: Define a operação da ULA. O valor 0 indica que a ULA deve somar, e 1 que a ULA deve subtrair os valores de suas entradas (1ª entrada - 2ª entrada).

reset_pc_o: Uma funcionalidade incorporada ao BIP2 sob responsabilidade da equipe, no final do trabalho, foi o reset. Este pino não está presente no modelo arquitetural, e foi incorporado para possibilitar que ao sinal de um reset, vindo exteriormente, bem como o clock, o PC possa retornar ao seu valor original de 0 e reiniciar a execução do código.

Branch_o: Sinal de seleção do mux de entrada do PC. Responsável por selecionar se a entrada do PC será um operando (vindo do módulo extensor de sinal) ou se será o valor do PC anterior incrementado a 1 (vindo do módulo incrementador). Esse sinal é apenas usado por instruções de branches, isto é, instruções que fazem desvio de código para um endereço da ROM, que é o próprio operando. Quando esse sinal é levantado, ele deve ser resetado posteriormente, para que o fluxo de instrução volte a executar sequencialmente. O reset é feito na instrução de busca. Entretanto, o leitor deve se questionar se isso não geraria um erro, já que o próximo estado é o

estado de execução. Nesse ponto valem duas observações: a 1ª é que o PC apenas é atualizado se o sinal de controle de escrita **Wrpc_o** também estiver habilitado em uma borda de subida de clock, e a 2ª é que todas as instruções de branch não possuem o estado de execução, ao invés disso, o próximo estado é o estado de busca. Pois não há nada para ser executado. As instruções de branch de desvio condicional consultam as flags **N** e **Z** (vindas do módulo de dados) resultantes da operação anterior da **ULA** e decidem se o Branch será habilitado ou não. Instrução de desvio não condicional não consulta flags e apenas habilita o Branch independente dos valores da flag. Instruções de transferência como LD, STO ou aritméticas como ADD, SUBI, não consultam flags, nem atribuem valores ao Branch. O leitor também deve se perguntar se este sinal não fica com um valor indeterminado durante a decodificação de outras instruções que não sejam de desvio, entretanto, isso não acontece, pois o valor é simplesmente mantido. Os blocos de initial contido nos códigos também servem para atribuir um valor inicial as variáveis, esse tipo de recurso, permite tanto inicializar uma condição para todos as variáveis de saída quanto estabelecer um valor para que nada fique indeterminado, mesmo que não seja configurado em nenhum momento. Outra observação pertinente englobando instruções de desvio é que dentro do decodificado dois registradores de 1 bit cada, foram criados para armazenar os valores anteriores das flags, caso isso não fosse feito, durante a decodificação de uma instrução de desvio, as flags consultadas seriam resultantes das operações atuais do módulo de dados e da ULA, que não fariam sentido, principalmente porque, a segunda entrada da ULA é constantemente e automaticamente atualizada ou pelos valor vindo do extensor de sinal ou algum valor da memória RAM de acordo com os bits de seleção do MUX da segunda entrada da ULA.

A memória RAM, bem como a ROM, também obedece fielmente o modelo arquitetural e de interface proposto, por isso, a saída é atualizada sempre que um endereço na entrada é modificado. Portanto, a mudança de uma instrução, muda o operando, consequentemente muda o valor do endereço de entrada na RAM e consequentemente o valor de saída que é encaminhado ao MUX da 2ª entrada da ULA.

- **Execução:**

  O estado de execução compreende toda as ações necessárias para executar a instrução. O primeiro estágio desse estado significa na verdade a execução de uma subida de clock para que as ações configuradas no estado de decodificação tomem efeito. As escritas da RAM e do ACC ocorrem nesse momento. O segundo estágio da decodificação compreende resetar os sinais de escrita para RAM e ACC e habilita o sinal de escrita para o PC para que no próximo estado (o de busca), o PC seja atualizado.

- **RESET:**

  Estado no qual a o PC não é atualizado, esse estado é apenas alcançado com a instrução HLT que “trava” a CPU. E permanece assim até que um sinal de RESET externo seja habilitado. Este por sua vez, atualiza o PC para 0 (A primeira instrução).

Todos os estados são executados em um ciclo de clock e foram aproveitadas tanto a borda de subida quanto a borda de descida do clock para atribuir ações às transições. Esse tipo de recurso foi usado para que não haja inconsistências, se uma variável é sensível a borda positiva ela deve ser configurada antes dessa subida, por vezes, variáveis assim foram configuradas nas bordas de descida.

Foram criados dois registradores auxiliares internos no decodificador para armazenar o estado atual e o próximo estado. Esse recurso permitiu que em todas as bordas de subida de um ciclo o próximo estado fosse determinado através do estado atual, e nas bordas de descida de um ciclo o estado atual fosse finalmente atualizado com o próximo estado.

A representação a seguir é uma máquina de estado que ilustra as possibilidades de transições entre os estados.

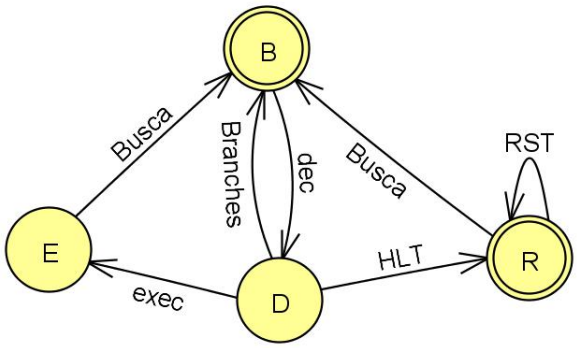


Figura 1 - Máquina de estados possíveis do decodificador.

[1]OBS: A documentação com detalhes a respeito da palavra de instrução e as ações associadas a cada OpCode podem ser encontradas na aba “Ajuda” do programa Bipide. Juntamente com o desenho que representa o modelo arquitetural.

Teste

O arquivo de teste do decoder não abrangeu todas as possibilidades. O teste mais poderoso foi o teste final: ComputadorTeste. Este testou código de máquina gerador por programas nos quais foram usadas todas as instruções.

Entretanto, essa imagem exemplifica o comportamento das variáveis comentadas acima no tempo. O diagrama abrange o teste dos opcodes de LDI e STO sequencialmente. As variáveis são apresentadas a esquerda e o valor delas no tempo no diagrama.

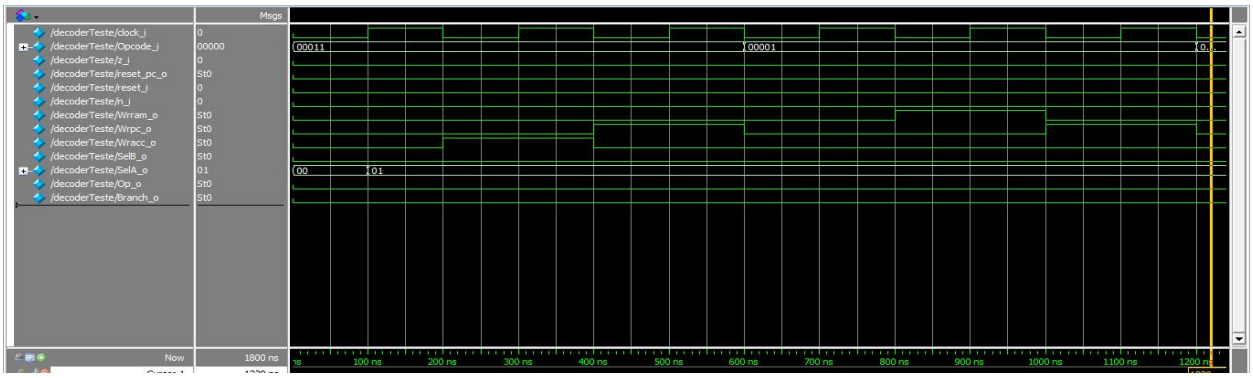


Figura 2 - Teste simples do decodificador.

Extensão de Sinal

Literalmente um extensor de sinal do operando que vem do módulo de controle.

Teste

Este teste apenas trata de verificar se o que é recebido na entrada do módulo é o mesmo que é enviado para a sua saída.

Mux 3x2

Trata-se de um multiplexador com três entradas, selecionadas a partir de um seletor (SelA) de dois bits.

Teste

Cada uma das entradas é inicializada com um valor diferente, a fim de se verificar se a saída do Mux está de acordo com o critério de seleção. Inicialmente, como o **SelA_i** está em 0, é enviado para a saída o valor de **out_data_i**. Após 100ns, como o **SelA_i** está em 1, é enviado para a saída o valor do Extensor de Sinal. Na faixa de 200ns, como o **SelA_i** está em 10, é enviado para a saída o valor da saída da ULA.

Mux 2x1

Trata-se de um multiplexador com duas entradas, selecionadas a partir de um seletor (SelB, no módulo Datapath, ou Branch, no módulo Control) de um bit. Ele é usado tanto no Control quanto no Datapath.

Teste

Cada uma das entradas é inicializada com um valor diferente, a fim de se verificar se a saída do Mux está de acordo com o critério de seleção. Inicialmente, como o seletor está em 0, é enviado para a saída o valor de out_data (no caso do módulo Control, o valor de Incrementer). Após 100ns, como o seletor está em 1, é enviado para a saída o valor do Extensor de Sinal.

ACC

É um registrador chamado de acumulador, que é utilizado para armazenamento de dados durante uma operação.

Teste

Sua entrada é a saída do Mux 3x2, que é inicializada com um determinado valor, para a verificação do funcionamento do registrador, que acumula o valor recebido durante a borda de subida do clock e quando o **WrACC** está em 1.

ULA

A unidade lógica aritmética é aquela responsável por executar operações de adição e subtração, baseadas numa operação, que pode ser a adição ou a subtração, e dois operandos. O resultado da operação faz com que certos sinais sejam enviados para o módulo de Flags.

Teste

No primeiro momento, os operandos **operand1** e **operand2** estão com os valores 11 e 10, respectivamente. O valor de **op_i** está em 0, indicando a execução de uma adição. A saída nesse momento é 101, e as flags **z_o** e **n_o** permanecem com o valor 0. Na faixa dos 200ns, o **op_i** é alterado para 1, indicando a execução de uma subtração, onde o valor 11 será subtraído de 10, resultando no número negativo **-1**, representado em complemento a 2, fazendo com que a saída **n_o** transmita 1. Na faixa de 300ns, ocorre uma subtração entre os operandos, ambos com o valor 101, resultando no valor 0, fazendo com que a saída **z_o** transmita 1.

Flags

Este módulo apenas transmite os sinais de flag negativa ou nula, baseado na saída do módulo ULA.

Teste

Este teste trata apenas de verificar se o valor das flags, recebidos na entrada, são iguais aqueles enviados para a saída.

COMPUTADOR

O computador é o arquivo no qual é a entidade de hierarquia superior, isso significa que ele é o responsável em fazer as ligações internas entre os módulos de hierarquia inferior. Para ilustrar as ligações, (e serve também como uma forma de conferir se as ligações foram feitas corretamente), pode-se usar o RTL viewer do Quartus em Tools.

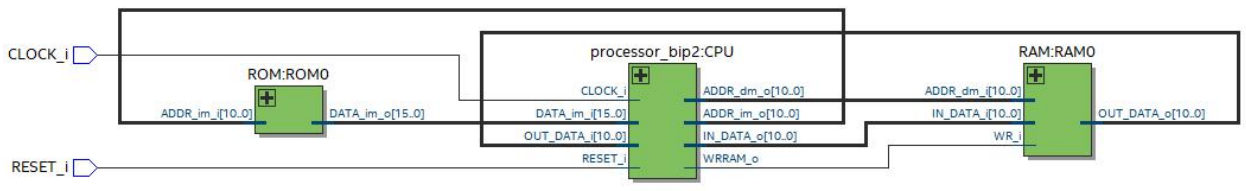


Figura 3 - Módulos e ligações de Computador

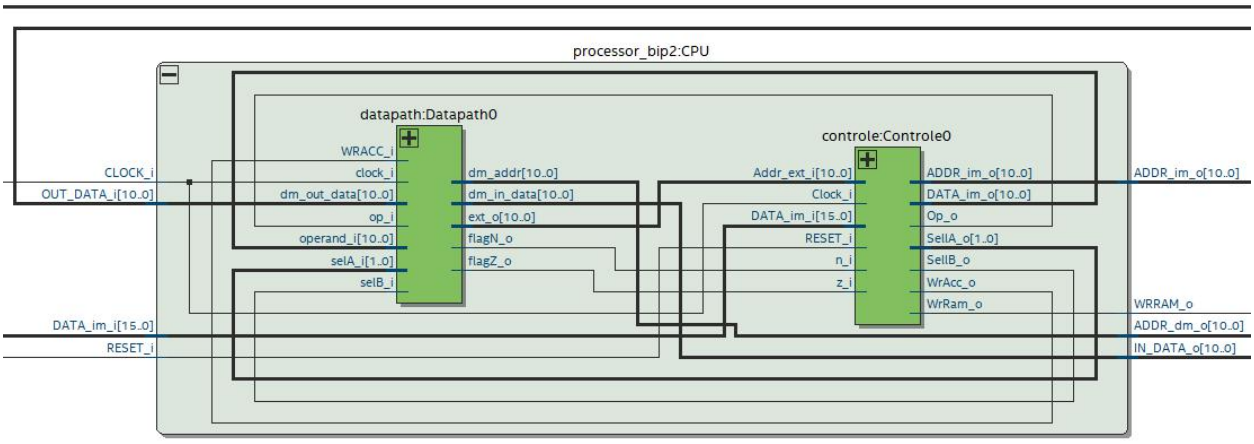


Figura 4 - Módulos e ligações do processador.

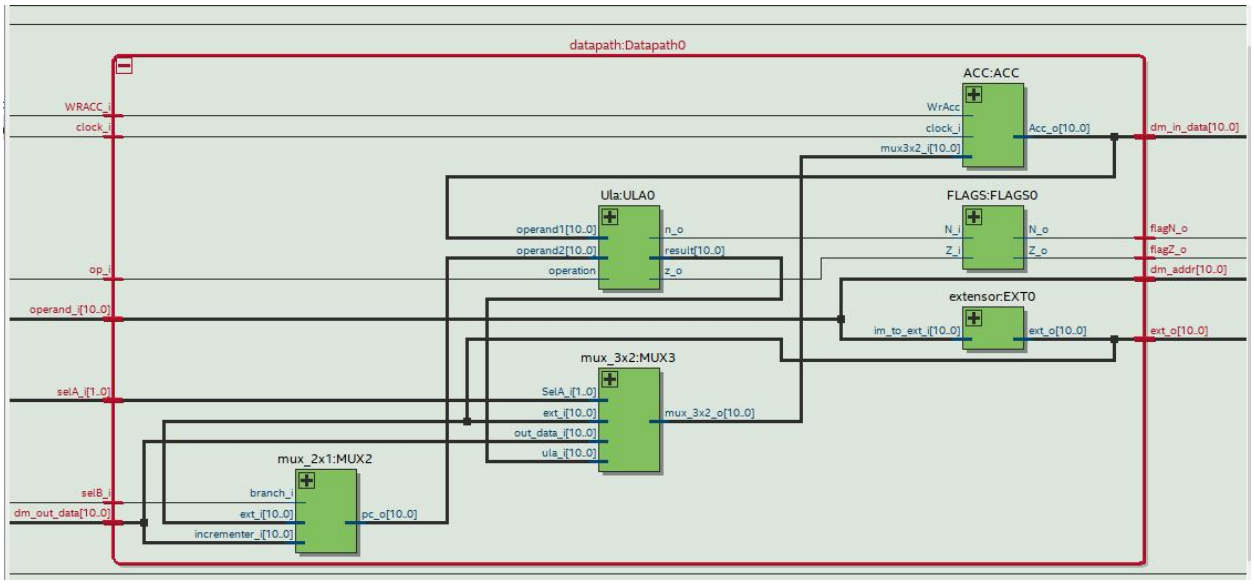


Figura 5 - Módulos e ligações do datapath

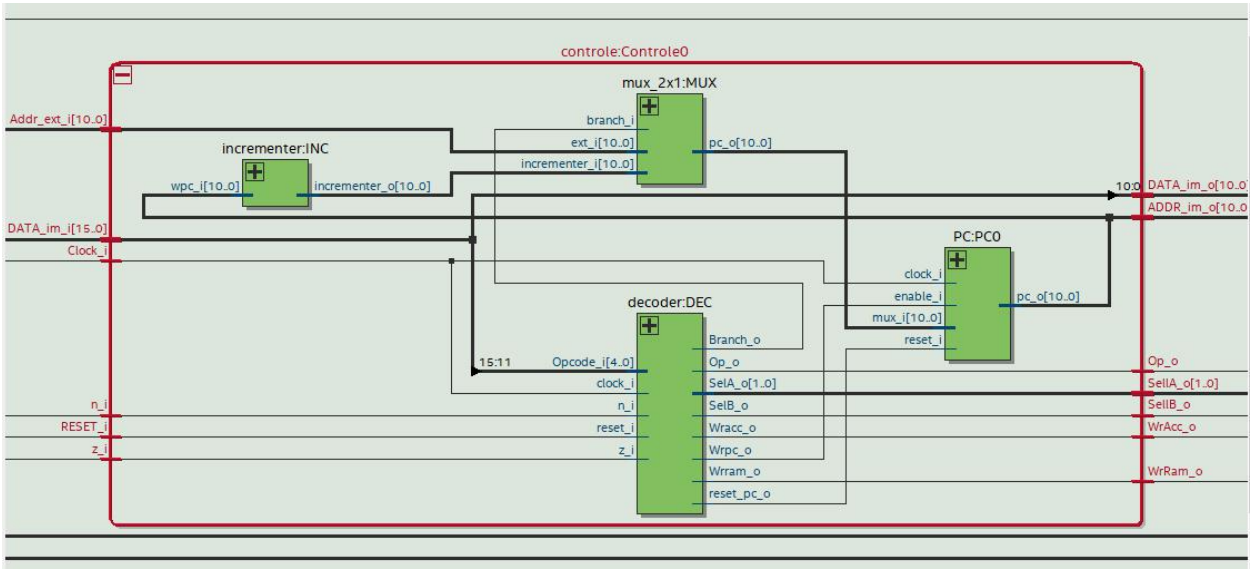


Figura 6 - Módulos e ligações do controle

Teste Final - De integração

Para testar o resultado final foi carregado diferentes códigos na ROM englobando todas as instruções e diversos algoritmos. No final, era verificado se os valores da RAM coincidiam com o esperado. No Model Sim, ao carregar o arquivo de teste do computador e optar por simulá-lo, para verificar também os estados na memória, basta ir em view e habilitar o MemoryList, que carrega todas as listas de memória para serem visualizadas no diagrama de tempo também. O Model Sim também oferece uma aba no canto inferior esquerdo chamada Transcript, é uma CLI e foi muito útil para fazer o debug do código. Pois com a função $display do Quartus foi possível plotar os estados e verificar se a sequência estava correta. O código final que está disponível em programMemory.list é o código a seguir:

0001000000000000 //LD 0

0010100000000001 //ADDI 1

0101000000000101 //BGT 5

0000100000000011 //STO 3

0000000000000000 //HLT

0000100000000010 //STO 2

0011100000000011 //SUBI 3

0000100000000000 //STO 0

0000000000000000 //HLT

O código de Teste consiste na seguinte estratégia:

Um clock simulado e depois de algumas execuções do HLT, o reset é acionado para avaliar se o código será executado novamente e dessa vez não haverá desvio condicional. O arquivo de dataMemory.list que é a memória RAM inicializa com zero em todas as posições. E no final da simulação é possível analizar os valores através do ModelSim.

Observações importantes: deve-se atualizar o valor na ROM da constante MEMSIZE se o arquivos da ROM for mudado, por exemplo, para esse código de 9 instruções, o valor de MEMSIZE é 9.

Depuração do código: (Lembre-se que a memória RAM inicializa com zero).

```
//1ª EXECUÇÃO
LD 0 // ACC = RAM[0], RAM[0] = 0
ADDI 1 // ACC = 0 + 1 = 1
BGT 5 //SE ACC > 0, DESVIE PARA ROM[5], ROM[5] -> STO 2
STO 3//NÃO É EXECUTADA
HLT//NÃO É EXECUTADA
STO 2//RAM[2] = ACC, ACC = 1, RAM[2] = 1
SUBI 3//ACC = ACC - 3, ACC = 1 - 3 = -2
STO 0// RAM[0] = -2
HLT //TRAVA CPU ATÉ QUE OCORRA UM RESET PC = PC

//2ª EXECUÇÃO - PÓS RESET
LD 0 // ACC = RAM[0], RAM[0] = -2
ADDI 1 // ACC = -2 + 1 = -1
BGT 5 //SE ACC > 0, DESVIE PARA ROM[5], ROM[5] -> STO 2
STO 3//RAM[3] = -1
HLT////TRAVA CPU, NÃO HÁ MAIS RESET PC = PC
STO 2//NÃO É EXECUTADA
SUBI 3//NÃO É EXECUTADA
STO 0//NÃO É EXECUTADA
HLT //NÃO É EXECUTADA
```

Figura 7 - Depuração do código para teste

Ao final, deve-se esperar então que a RAM esteja da seguinte maneira:

RAM[0] = -2

RAM[2] = 1

RAM[3] = - 1

Observação importante: a intenção do sinal do reset não é provocar desvios para o código como loop ou repetições, é reinicializar o código do zero para permitir sua execução como se nunca tivesse sido executada, portanto, o reset também deveria reinicializar a RAM com zeros novamente para que fosse possível obter o mesmo resultado do mesmo código. Entretanto, isso não foi feito para que fosse possível testar o comportamento do RESET com o desvio BGT.

Após a execução do teste ComputadorTeste a RAM se apresenta da seguinte maneira:

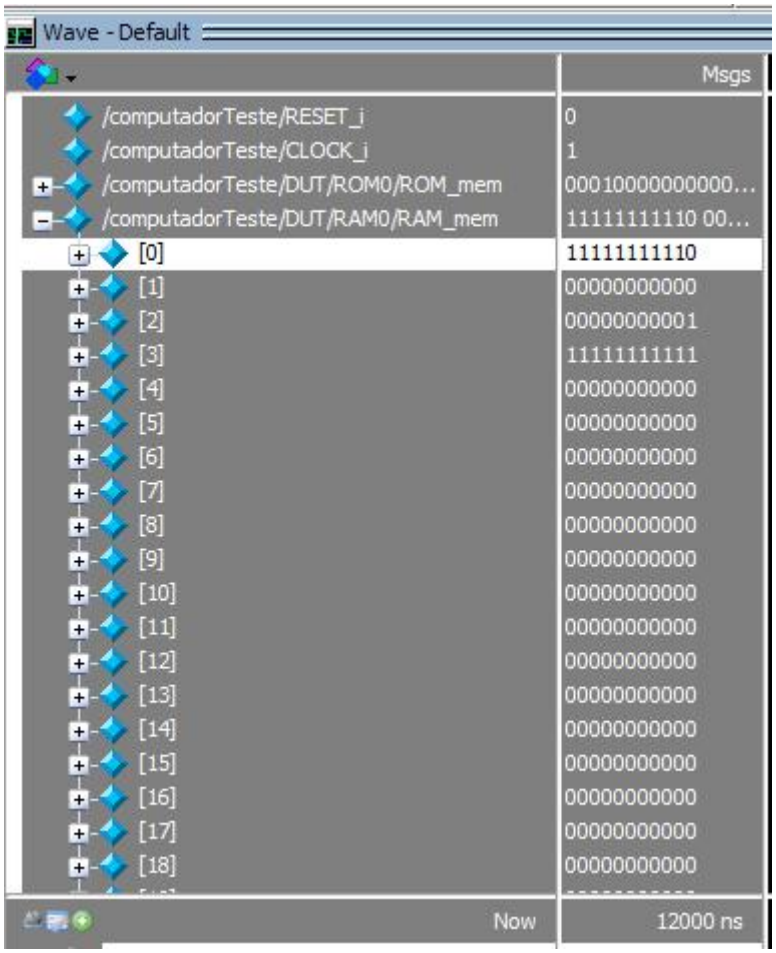


Figura 8 - Resultado dos valores da RAM após a execução do código de teste

Os valores coincidem com o resultado da depuração feita anteriormente como queríamos demonstrar.

OBSs:

1- O resultado está em complemento a dois.

2- O reset está em 0, pois durante a segunda execução o teste leva o resultado para zero para que não haja um loop infinito. Depois da segunda execução o processador encontra novamente a instrução HLT e permanece travado.

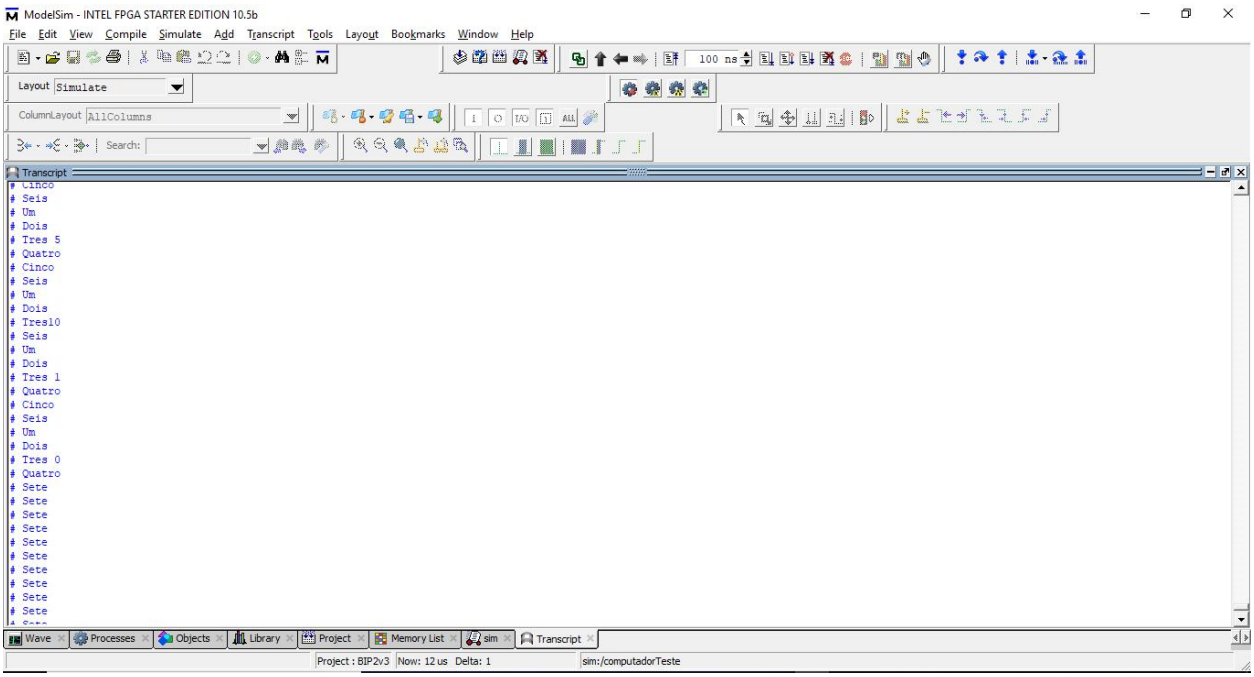


Figura 9 - Transcript (Final da 2ª execução)

Na aba do transcript foi visto o resultado da task $display para monitorar os estados e transições (estágios do estado). O número ao lado da palavra três é o valor do Opcode em decimal. Três é o primeiro estágio do estado de decodificação (Borda de Subida), e Quatro por exemplo, é o segundo estágio do estado de decodificação (Borda de Descida). Aqui é interessante observar que a última instrução de OpCode 0 (HLT), pula do estágio 4 para o sete, que é em seguida repetido. Pois o processador sai do estágio de decodificação e como não há o que executar migra ao estado de RESET no qual permanecerá.

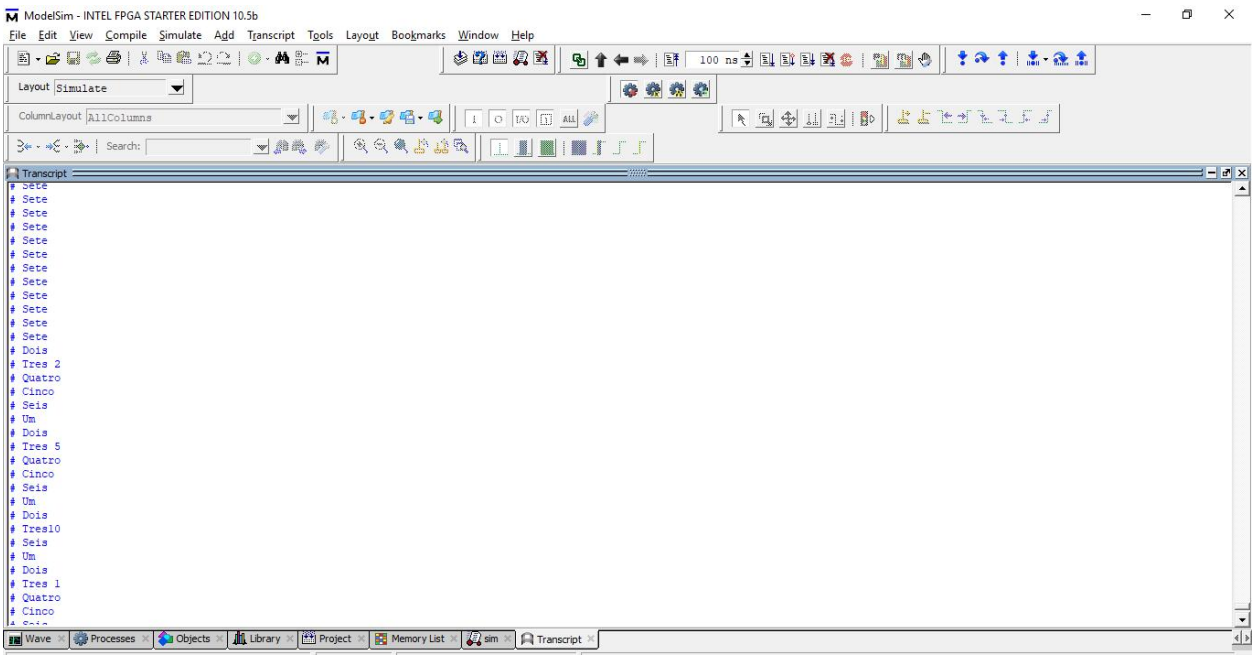


Figura 10 - Transcript (Transição para a 2ª execução)

A figura 10 já exibe a transição do estado de RESET para o estado de busca. A 2ª execução é precedida de múltiplos estados de RESET porque propositalmente o sinal de RESET no teste foi levantado após alguns ciclos de clock depois que a 1ª execução já tinha sido concluída. Vale também observar que inicia-se a partir do 2º estágio do estado de busca (Dois), pois não há necessidade de incrementar o PC para a primeira execução.