**EA773 - Laboratório de Circuitos Lógicos**

**Relatório do Projeto Final**

**Autor:** Mateus Henrique Silva Araújo

**RA: 184940**

**Data:** 01/11/2022

1. **Introdução:**

Esta atividade tem como intuito juntar as habilidades e os circuitos desenvolvidos até então para possibilitar a implementação de um computador simplificado. O objetivo final é implementar um circuito com comportamento análogo ao de um computador, o qual deverá ser composto por uma unidade central de processamento (CPU-*Central Processing Unit*), uma unidade de memória ROM (responsável pelo armazenamento dos programas) e uma unidade de memória RAM (responsável pelo armazenamento dos dados), operando com instruções de 1 e 2 bytes (com jogo de instruções especificado) e dados de 4 bits.

Para facilitar a leitura e compreensão tanto deste relatório quanto do projeto em si, iniciaremos este texto apresentando uma visão geral dos componentes mais importantes de cada parte do computador simplificado. Isto será feito visando não só introduzir o comportamento dos circuitos implementados, mas também explicitar os sinais de controle que percorrem cada um deles. Em seguida, as especificações, minimizações e simulações desses sistemas serão apresentadas.

1. **Visão Geral:**

O computador simplificado é composto por três grandes módulos: “*memprog”*, “*ram”* e *“cpu”*. O primeiro destes implementa a unidade de memória de programa do computador, contendo quatro módulos de memória ROM 64x8 selecionáveis individualmente, cada qual apresentando um programa específico. O segundo módulo implementa uma memória RAM 16x4, a qual será utilizada para o armazenamento de dados manipulados por programas em execução. Já o terceiro destes, o módulo *“cpu”*, implementa a unidade de processamento do computador, sendo responsável tanto por gerar os sinais de controle para comunicação entre as três partes do sistema, quanto por realizar operações lógicas e aritméticas dos programas em execução. O diagrama esquemático do computador simplificado está disposto na figura 1.

|  |
| --- |
|  |
| *Figura 1: Diagrama esquemático do circuito final do computador simplificado.* |

O módulo *“memprog”* foi desenvolvido de forma semelhante ao módulo *“rom\_4block”* da atividade de laboratório anterior, isto é, as quatro unidades de memória ROM 64x8 que o integram são implementadas a partir de unidades de memória ROM menores. Estas, por sua vez, são construídas a partir de um decodificador binário para *one-hot* e de um arranjo de portas *GND* e *VCC* ligadas a um único barramento de saída, protegido com lógica *tri-state*.

A principal diferença deste módulo em questão ao já feito anteriormente é que, ao invés das unidades 64x8 usarem recursivamente de unidades de memória menores (32x8, 16x8 e 16x4), no módulo *“memprog”* elas são compostas diretamente por memórias 16x8, configuradas uma a uma com as combinações de bits que implementam os programas indicados no roteiro da atividade. Nas figuras 2, 3 e 4, estão dispostos, respectivamente, os circuitos internos do módulo *“memprog”*, de uma das unidades de memória 64x8 deste, bem como de uma das memórias base 16x8 desta.

|  |  |
| --- | --- |
|  | *Figura 2: Diagrama esquemático do módulo “memprog”. Este módulo implementa a memória de programa do computador, sendo constituído por 4 unidades de memória ROM não volátil 64x8. Cada uma destas unidades contém um programa que pode ser executado pelo computador simplificado.* |
|  | *Figura 3: Diagrama esquemático do módulo “rom64x8\_3” apresentado logo acima. Este módulo implementa uma das unidades de memória ROM 64x8 do módulo “memprog” utilizando de 4 unidades de memória ROM 16x8 (uma para cada 16 endereços).* |
|  | |
| *Figura 4: Diagrama esquemático do módulo “rom16x8\_30” apresentado logo acima. São módulos deste tipo que agem como base para guardar o código dos programas a serem executadas para o teste do computador simplificado.* | |

O módulo *“ram”* é exatamente igual ao implementado na atividade de laboratório anterior, já tendo sido, portanto, especificado, minimizado e testado. Dessa forma, não será necessário repetir tal processo, fato que facilita consideravelmente o desenvolvimento do circuito final.

O diagrama esquemático do módulo *“cpu”* está disposto na figura 5. Este sistema é composto por quatro subcircuitos, cuja interação possibilita o desempenho das funções da unidade de processamento central. São eles:

* Relógio (figura 6): é responsável por gerar o *clock* fornecido a todos os componentes sequenciais do computador;
* Contador de Programa (figura 7): utiliza do módulo *“contador\_mod\_256”* implementado em atividades anteriores para fornecer os endereços a serem acessados na memória ROM do computador. Além disso, ele também apresenta um multiplexador (controlado pelo decodificador de instruções) que seleciona entre dois barramentos para possibilitar a implementação de instruções de desvio;
* Decodificador de Instruções (figura 8): é responsável por gerar todos os sinais de controle que comandam não só os circuitos internos do módulo *“cpu”*, mas também as entradas de seleção e endereço da memória RAM. Ele é constituído por três decodificadores distintos, cada um específico para um tipo de instrução;
* ULA (figura 9): é encarregado de realizar as operações lógicas e aritméticas do computador simplificado. Ele está fundamentado no módulo *“bus\_ula”*, que implementa a ULA Estendida desenvolvida em laboratórios passados. Além disso, ele também consta de um multiplexador que seleciona entre barramentos para determinar qual será o operador de entrada da ULA.

|  |
| --- |
| Diagrama, Esquemático  Descrição gerada automaticamente |
| *Figura 5: Diagrama esquemático do módulo “cpu”. Este módulo é composto por 4 subcircuitos com funções distintas: Relógio, Contador de Programa, ULA e Decodificador de Instruções. Estes módulos estão apresentados nas figuras 6 a 9 com maior definição.* |

|  |  |
| --- | --- |
| Uma imagem contendo Interface gráfica do usuário  Descrição gerada automaticamente | *Figura 6: Subcircuitos Relógio e Contador de programas apresentados em maior definição. O relógio é responsável por fornecer o “clock” para todos os componentes sequenciais do computador. O Contador de programa é encarregado de gerar os endereços para acesso da memória de programa. Este subcircuito emprega o módulo “contador\_mod\_256”, que foi desenvolvido na atividade de laboratório 3.* |

|  |
| --- |
| Diagrama  Descrição gerada automaticamente com confiança média |
| *Figura 7: Subcircuito Decodificador de Instruções, responsável por gerar os sinais de controle tanto para os demais subcircuitos do módulo “cpu”, quanto para as entradas do módulo “ram”.* |

|  |
| --- |
| Diagrama, Esquemático  Descrição gerada automaticamente |
| *Figura 7: Subcircuito ULA, responsável por realizar as operações lógicas e aritméticas do computador simplificado. Ele emprega o módulo “bus\_ula”, que foi desenvolvido na atividade de laboratório 4.* |

1. **Jogo de instruções:**

Antes de nos aprofundarmos no funcionamento de cada componente do computador simplificado, é conveniente especificarmos o jogo de instruções sobre o qual ele operará. Nesse projeto, teremos instruções de 1 e 2 bytes divididas em três tipos: aritméticas, de transferência e de desvio. As instruções do primeiro tipo realizam operações lógicas e aritméticas na ULA da unidade de processamento central. As do segundo tipo promovem a movimentação de dados entre os registradores da ULA, as memórias e as chaves externas. As do terceiro tipo são capazes de alterar condicional e incondicionalmente o fluxo de execução sequencial do programa.

A tabela a seguir expõe o jogo de instruções a ser implementado. Nela, as duas primeiras colunas correspondem a parte mais (“4 bits +s”) e menos (“4 bits -s”) significativa do código de operação, a terceira coluna indica se a operação apresenta operando (ou seja, se a próxima instrução fornecida pela memória de programa deve ser tratada como entrada para alguma função), a quarta coluna indica o apelido da instrução, a quinta apresenta a operação que o código deve gerar e, por fim, a sexta mostra uma descrição desta operação. Além disso, vale destacar que as instruções com operando, isto é, aquelas que apresentam dois bytes, são executadas em dois ciclos de *clock*, pois foi definido que as memórias ROMs empregues deveriam apresentam palavras se somente um byte.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Código de operação (1 byte) | | Operando (1 byte) | Mnemônico da instrução | Operação | Descrição |
| 4 bits +s | 4 bits -s |
| Aritméticas | | | | | |
| 0001 | 0000 | -- | ADD |  | Soma os registradores A e B, colocando o resultado no acumulador e alterando as flags |
| 0001 | 0010 | -- | ADDC |  | Soma os registradores A e B e o carry, colocando o resultado no acumulador e alterando as flags |
| 0001 | 0100 | -- | SUB |  | Subtrai os registradores A e B, colocando o resultado no acumulador e alterando as flags |
| 0001 | 0110 | -- | INC |  | Incrementa o registrador A em 1, colocando o resultado no acumulador e alterando as flags |
| 0001 | 0111 | -- | DEC |  | Decrementa o registrador A em 1, colocando o resultado no acumulador e alterando as flags |
| 0001 | 1000 | -- | NEG |  | Realiza o complemento de 2 do registrador A, colocando o resultado no acumulador e alterando as flags |
| 0001 | 1001 | -- | CMPL |  | Realiza a negação bit a bit do registrador A, colocando o resultado no acumulador e alterando as flags |
| 0001 | 1010 | -- | CLRA |  | Zera o registrador A sem afetar as flags |
| 0001 | 1011 | -- | CLRB |  | Zera o registrador B sem afetar as flags |
| 0001 | 1100 | -- | CLRAC |  | Zera o registrador AC sem afetar as flags |
| 0001 | 1101 | -- | CLRFL |  | Zera o registrador FL (alterando as flags) |
| Transferência | | | | | |
| 0100 | 0000 | -- | MOV A, AC |  | Move o conteúdo do acumulador para o registrador A sem afetar as flags |
| 0100 | 0001 | -- | MOV B, AC |  | Move o conteúdo do acumulador para o registrador B sem afetar as flags |
| 0100 | 0010 | OP | LDA OP |  | Carrega o registrador A com o valor armazenado no endereço OP da memória de programa sem afetar as flags |
| 0100 | 0011 | OP | LDB OP |  | Carrega o registrador B com o valor armazenado no endereço OP da memória de programa sem afetar as flags |
| 0100 | 0100 | OP | ST OP |  | Armazena o conteúdo do registrador AC no endereço OP da memória de dados sem afetar as flags |
| 0100 | 0101 | OP | LDAI OP |  | Carrega o registrador A com o valor de OP sem afetar as flags |
| 0100 | 0110 | OP | LDBI OP |  | Carrega o registrador B com o valor de OP sem afetar as flags |
| 0100 | 0111 | -- | INA |  | Carrega o registrador A com o valor das chaves sem afetar as flags |
| 0100 | 1000 | -- | INB |  | Carrega o registrador B com o valor das chaves sem afetar as flags |
| 0100 | 1001 | OP | IN OP |  | Carrega na posição de memória de programa indicada por OP o valor das chaves sem afetar as flags |
| Desvio | | | | | |
| 1000 | 0000 | OP | B OP |  | Continua a execução a partir do endereço fornecido por OP |
| 1000 | 0001 | OP | BEQ OP | se | Continua a execução a partir do endereço fornecido por OP se . Caso contrário mantém a execução sequencial |
| 1000 | 0010 | OP | BNEQ OP | se | Continua a execução a partir do endereço fornecido por OP se . Caso contrário mantém a execução sequencial |
| 1000 | 0011 | OP | BCS OP | se | Continua a execução a partir do endereço fornecido por OP se . Caso contrário mantém a execução sequencial |
| 1000 | 0100 | OP | BCC OP | se | Continua a execução a partir do endereço fornecido por OP se . Caso contrário mantém a execução sequencial |
| 1000 | 0101 | OP | BMI OP | se | Continua a execução a partir do endereço fornecido por OP se . Caso contrário mantém a execução sequencial |
| 1000 | 0110 | OP | BPL OP | se | Continua a execução a partir do endereço fornecido por OP se . Caso contrário mantém a execução sequencial |
| 1000 | 0111 | OP | BVS OP | se | Continua a execução a partir do endereço fornecido por OP se . Caso contrário mantém a execução sequencial |
| 1000 | 1000 | OP | BVC OP | se | Continua a execução a partir do endereço fornecido por OP se . Caso contrário mantém a execução sequencial |
| 1000 | 1001 | OP | BHI OP | se e | Continua a execução a partir do endereço fornecido por OP se e . Caso contrário mantém a execução sequencial |
| 1000 | 1010 | OP | BLS OP | se ou | Continua a execução a partir do endereço fornecido por OP se ou . Caso contrário mantém a execução sequencial |
| 1000 | 1011 | OP | BGE OP | se | Continua a execução a partir do endereço fornecido por OP se . Caso contrário mantém a execução sequencial |
| 1000 | 1100 | OP | BLT OP | se | Continua a execução a partir do endereço fornecido por OP se . Caso contrário mantém a execução sequencial |
| 1000 | 1101 | OP | BGT OP | se e | Continua a execução a partir do endereço fornecido por OP se e . Caso contrário mantém a execução sequencial |
| 1000 | 1110 | OP | BLE OP | se ou | Continua a execução a partir do endereço fornecido por OP se ou . Caso contrário mantém a execução sequencial |
| 1000 | 1111 | -- | STP |  | Mantém o contador de programa no valor atual |

1. **Programas do computador simplificado:**

Os programas que serão executados pelo computador foram estabelecidos no roteiro do projeto, sendo cada um armazenados em uma das unidades de memória ROM 64x8 do módulo *“memprog”*. Nas quatro tabelas abaixo estão dispostos tanto as instruções quanto suas respectivas representações binárias dos códigos que formam dos programas A (Teste de Aritmética), B (Teste de transferência), C (Teste de Desvio) e D (Multiplicação). A relação de endereço e conteúdo da tabela foi utilizada para a implementação das unidades de memória ROM 16x8 que integram a memória de programa.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Programa A – Teste de Aritmética | | | | |
| Rótulo | Instrução | Operando | Endereço | Conteúdo |
| -- | LDAI | 5 | 00000000 | 01000101 |
| 00000001 | 00000101 |
| -- | LDBI | 3 | 00000010 | 01000110 |
| 00000011 | 00000011 |
| -- | SUB | -- | 00000100 | 00010100 |
| -- | INC | -- | 00000101 | 00010110 |
| -- | DEC | -- | 00000110 | 00010111 |
| -- | NEG | -- | 00000111 | 00011000 |
| -- | CMPL | -- | 00001000 | 00011001 |
| -- | CLRA | -- | 00001001 | 00011010 |
| -- | CLRB | -- | 00001010 | 00011011 |
| -- | CLRAC | -- | 00001011 | 00011100 |
| -- | CLRFL | -- | 00001100 | 00011101 |
| -- | STP | -- | 00001101 | 10001111 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Programa B – Teste de Transferência | | | | |
| Rótulo | Instrução | Operando | Endereço | Conteúdo |
| -- | INA | -- | 00000000 | 01000111 |
| -- | INB | -- | 00000001 | 01001000 |
| -- | ADD | -- | 00000010 | 00010000 |
| -- | MOVA | -- | 00000011 | 01000000 |
| -- | MOVB | -- | 00000100 | 01000001 |
| -- | ADD | -- | 00000101 | 00010000 |
| -- | ST | 0 | 00000110 | 01000100 |
| -- | 00000111 | 00000000 |
| -- | ST | 1 | 00001000 | 01000100 |
| -- | 00001001 | 00000001 |
| -- | LDA | 0 | 00001010 | 01000010 |
| -- | 00001011 | 00000000 |
| -- | LDB | 1 | 00001100 | 01000011 |
| -- | 00001101 | 00000001 |
| -- | IN | 3 | 00001110 | 01001001 |
| -- | 00001111 | 00000011 |
| -- | IN | 4 | 00010000 | 01001001 |
| -- | 00010001 | 00000100 |
| -- | LDA | 3 | 00010010 | 01000010 |
| -- | 00010011 | 00000011 |
| -- | LDB | 4 | 00010100 | 01000011 |
| -- | 00010101 | 00000100 |
| -- | LADI | 7 | 00010110 | 01000101 |
| -- | 00010111 | 00000111 |
| -- | LDBI | 1 | 00011000 | 01000110 |
| -- | 00011001 | 00000001 |
| -- | ADD | -- | 00011010 | 00010000 |
| -- | CLRA | -- | 00011011 | 00011010 |
| -- | CLRB | -- | 00011100 | 00011011 |
| -- | CLRAC | -- | 00011101 | 00011100 |
| -- | CLRFL | -- | 00011110 | 00011101 |
| -- | STP | -- | 00011111 | 10001111 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Programa C – Teste de Desvio | | | | |
| Rótulo | Instrução | Operando | Endereço | Conteúdo |
| -- | B | P1 | 00000000 | 10000000 |
|  | 00000001 | 00000011 |
|  | STP |  | 00000010 | 10001111 |
| P1: | CLRFL |  | 00000011 | 00011101 |
|  | BNEQ | P2 | 00000100 | 10000010 |
|  | 00000101 | 00000111 |
|  | STP |  | 00000110 | 10001111 |
| P2: | BCC | P3 | 00000111 | 10000100 |
|  | 00001000 | 00001010 |
|  | STP |  | 00001001 | 10001111 |
| P3: | BPL | P4 | 00001010 | 10000110 |
|  | 00001011 | 00001101 |
|  | STP |  | 00001100 | 10001111 |
| P4: | BVC | P5 | 00001101 | 10001000 |
|  | 00001110 | 00010000 |
|  | STP |  | 00001111 | 10001111 |
| P5: | INA |  | 00010000 | 01000111 |
|  | INB |  | 00010001 | 01001000 |
|  | SUB |  | 00010010 | 00010100 |
|  | BCS | P6 | 00010011 | 10000011 |
|  | 00010100 | 00010110 |
|  | STP |  | 00010101 | 10001111 |
| P6: | BHI | P7 | 00010110 | 10001001 |
|  | 00010111 | 00011001 |
|  | STP |  | 00011000 | 10001111 |
| P7: | BGE | P8 | 00011001 | 10001011 |
|  | 00011010 | 00011100 |
|  | STP |  | 00011011 | 10001111 |
| P8: | BGT | P9 | 00011100 | 10001101 |
|  | 00011101 | 00011111 |
|  | STP |  | 00011110 | 10001111 |
| P9: | INA |  | 00011111 | 01000111 |
|  | INB |  | 00100000 | 01001000 |
|  | SUB |  | 00100001 | 00010100 |
|  | BMI | P10 | 00100010 | 10000101 |
|  | 00100011 | 00100101 |
|  | STP |  | 00100100 | 10001111 |
| P10: | BLS | P11 | 00100101 | 10001010 |
|  | 00100110 | 00101000 |
|  | STP |  | 00100111 | 10001111 |
| P11: | BLT | P12 | 00101000 | 10001100 |
|  | 00101001 | 00101011 |
|  | STP |  | 00101010 | 10001111 |
| P12: | BLE | P13 | 00101011 | 10001110 |
|  | 00101100 | 00101110 |
|  | STP |  | 00101101 | 10001111 |
| P13: | INA |  | 00101110 | 01000111 |
|  | INB |  | 00101111 | 01001000 |
|  | SUB |  | 00110000 | 00010100 |
|  | BVS | P14 | 00110001 | 10000111 |
|  | 00110010 | 00110100 |
|  | STP |  | 00110011 | 10001111 |
| P14: | INA |  | 00110100 | 01000111 |
|  | INB |  | 00110101 | 01001000 |
|  | SUB |  | 00110110 | 00010100 |
|  | BEQ | P15 | 00110111 | 10000001 |
|  | 00111000 | 00111010 |
|  | STP |  | 00111001 | 10001111 |
| P15: | STP |  | 00111010 | 10001111 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Programa D – Multiplicação | | | | |
| Rótulo | Instrução | Operando | Endereço | Conteúdo |
|  | CLRAC |  | 00000000 | 00011100 |
|  | CLRFL |  | 00000001 | 00011101 |
|  | ST | 2 | 00000010 | 01000100 |
|  | 00000011 | 00000010 |
|  | ST | 3 | 00000100 | 01000100 |
|  | 00000101 | 00000011 |
|  | INA |  | 00000110 | 01000111 |
|  | ADD |  | 00000111 | 00010000 |
|  | BEQ | FIM | 00001000 | 10000001 |
|  | 00001001 | 00101010 |
|  | ST | 0 | 00001010 | 01000100 |
|  | 00001011 | 00000000 |
|  | CLRA |  | 00001100 | 00011010 |
|  | INB |  | 00001101 | 01001000 |
|  | ADD |  | 00001110 | 00010000 |
|  | ST | 1 | 00001111 | 01000100 |
|  | 00010000 | 00000001 |
|  | ST | 2 | 00010001 | 01000100 |
|  | 00010010 | 00000010 |
| VOLTA: | LDA | 0 | 00010011 | 01000010 |
|  | 00010100 | 00000000 |
|  | DEC |  | 00010101 | 00010111 |
|  | BEQ | FIM | 00010110 | 10000001 |
|  | 00010111 | 00101010 |
|  | ST | 0 | 00011000 | 01000100 |
|  | 00011001 | 00000000 |
|  | LDA | 2 | 00011010 | 01000010 |
|  | 00011011 | 00000010 |
|  | LDB | 1 | 00011100 | 01000011 |
|  | 00011101 | 00000001 |
|  | ADD |  | 00011110 | 00010000 |
|  | ST | 2 | 00011111 | 01000100 |
|  | 00100000 | 00000010 |
|  | BCC | VOLTA | 00100001 | 10000100 |
|  | 00100010 | 00010011 |
|  | LDA | 3 | 00100011 | 01000010 |
|  | 00100100 | 00000011 |
|  | INC |  | 00100101 | 00010110 |
|  | ST | 3 | 00100110 | 01000100 |
|  | 00100111 | 00000011 |
|  | B | VOLTA | 00101000 | 10000000 |
|  | 00101001 | 00010011 |
| FIM: | LDA | 3 | 00101010 | 01000010 |
|  | 00101011 | 00000011 |
|  | LDB | 2 | 00101100 | 01000011 |
|  | 00101101 | 00000010 |
|  | CLRAC |  | 00101110 | 00011100 |
|  | CLRFL |  | 00101111 | 00011101 |
|  | STP |  | 00110000 | 10001111 |

Assim como requisitado pelo roteiro da atividade, o programa A foi armazenado no bloco 0 da unidade de memória ROM, o programa B, no bloco 1; o programa C, no bloco 2 e o programa D, no bloco 4. A próxima seção apresenta a implementação destas memórias.

1. **Memória de Programa:**

Iniciando a análise particular dos componentes desenvolvidos, vamos, primeiramente, abordar o módulo *“memprog”* que, assim como mostrado em seu diagrama esquemático na figura 2, é composto por quatro unidades de memória ROM 64x8 selecionáveis. Cada uma dessas memórias é composta por 4 unidades de memória ROM 16x8 independentes e selecionáveis, as quais foram implementadas de maneira análoga às memórias ROM 16x4 da atividade de laboratório 5. Para facilitar o entendimento da hierarquia desse módulo, a figura 8 apresenta a relação de módulos usados dentro de cada circuito.

|  |  |
| --- | --- |
| Tabela  Descrição gerada automaticamente | *Figura 8: hierarquia do módulo “memprog”. Este módulo consiste em quatro memórias ROM 64x8 selecionáveis, cada qual contendo 4 memórias ROM 16x8 selecionáveis.* |

A seguir, estão a especificação, minimização e simulações feitas para cada um dos módulos internos de *“memprog”* construídos:

1. **Projeto dos módulos ROM 16x8:**
   1. **Escopo:**

Projeto de circuitos combinacionais que implementem dispositivos de memória não volátil de leitura exclusiva com 16 palavras de memória de 8 bits. Os dispositivos devem ser implementados de modo a conter o conteúdo adequado para a implementação de cada módulo usado no projeto.

* 1. **Especificação de alto nível:**

*Entradas:*

,com e ;

.

*Saída:*

, com e .

*Função de Saída:*

,

onde significa o conteúdo da memória no endereço indicado pela entrada .

* 1. **Especificação binária:**

*Entradas:*

,com e ;

.

*Saída:*

, com e .

*Função de saída:*

Cada módulo do tipo ROM 16x8 apresenta uma função de saída distinta, a qual depende do conteúdo contido nele. Este conteúdo varia de acordo com o programa que será implementado em cada unidade ROM 64x8. Dessa forma, nas tabelas abaixo estão apresentadas as relações de conteúdo em função dos endereços utilizada para implementar cada uma das ROM 16x8 deste projeto. Para compreender tais tabelas, é de suma importância verificar não só a composição de cada módulo ROM 64x8 apresentada na figura 7, mas também os programas que devem ser armazenados neles (mostrados na seção anterior). É válido ressaltar que, como não foi especificado o tratamento que deveria ser dado para posições de memória não usadas, elas foram preenchidas com zero.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Módulo “*rom16x8\_00*” | |  | Módulo “*rom16x8\_zerada*” | |  | Módulo “*rom16x8\_10*” | |  | Módulo “*rom16x4\_11*” | |
|  |  |  |  |  |  |  |  |  |  |  |
| 0000 | 01000101 |  | 0000 | 00000000 |  | 0000 | 01000111 |  | 0000 | 01001001 |
| 0001 | 00000101 |  | 0001 | 00000000 |  | 0001 | 01001000 |  | 0001 | 00000100 |
| 0010 | 01000110 |  | 0010 | 00000000 |  | 0010 | 00010000 |  | 0010 | 01000010 |
| 0011 | 00000011 |  | 0011 | 00000000 |  | 0011 | 01000000 |  | 0011 | 00000011 |
| 0100 | 00010100 |  | 0100 | 00000000 |  | 0100 | 01000001 |  | 0100 | 01000011 |
| 0101 | 00010110 |  | 0101 | 00000000 |  | 0101 | 00010000 |  | 0101 | 00000100 |
| 0110 | 00010111 |  | 0110 | 00000000 |  | 0110 | 01000100 |  | 0110 | 01000101 |
| 0111 | 00011000 |  | 0111 | 00000000 |  | 0111 | 00000000 |  | 0111 | 00000111 |
| 1000 | 00011001 |  | 1000 | 00000000 |  | 1000 | 01000100 |  | 1000 | 01000110 |
| 1001 | 00011010 |  | 1001 | 00000000 |  | 1001 | 00000001 |  | 1001 | 00000001 |
| 1010 | 00011011 |  | 1010 | 00000000 |  | 1010 | 01000010 |  | 1010 | 00010000 |
| 1011 | 00011100 |  | 1011 | 00000000 |  | 1011 | 00000000 |  | 1011 | 00011010 |
| 1100 | 00011101 |  | 1100 | 00000000 |  | 1100 | 01000011 |  | 1100 | 00011011 |
| 1101 | 10001111 |  | 1101 | 00000000 |  | 1101 | 00000001 |  | 1101 | 00011100 |
| 1110 | 00000000 |  | 1110 | 00000000 |  | 1110 | 01001001 |  | 1110 | 00011101 |
| 1111 | 00000000 |  | 1111 | 00000000 |  | 1111 | 00000011 |  | 1111 | 10001111 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Módulo “*rom16x8\_20*” | |  | Módulo “*rom16x8\_21*” | |  | Módulo “*rom16x8\_22*” | |  | Módulo “*rom16x4\_23*” | |
|  |  |  |  |  |  |  |  |  |  |  |
| 0000 | 10000000 |  | 0000 | 01000111 |  | 0000 | 01001000 |  | 0000 | 00010100 |
| 0001 | 00000011 |  | 0001 | 01001000 |  | 0001 | 00010100 |  | 0001 | 10000111 |
| 0010 | 10001111 |  | 0010 | 00010100 |  | 0010 | 10000101 |  | 0010 | 00110100 |
| 0011 | 00011101 |  | 0011 | 10000011 |  | 0011 | 00100101 |  | 0011 | 10001111 |
| 0100 | 10000010 |  | 0100 | 00010110 |  | 0100 | 10001111 |  | 0100 | 01000111 |
| 0101 | 00000111 |  | 0101 | 10001111 |  | 0101 | 10001010 |  | 0101 | 01001000 |
| 0110 | 10001111 |  | 0110 | 10001001 |  | 0110 | 00101000 |  | 0110 | 00010100 |
| 0111 | 10000100 |  | 0111 | 00011001 |  | 0111 | 10001111 |  | 0111 | 10000001 |
| 1000 | 00001010 |  | 1000 | 10001111 |  | 1000 | 10001100 |  | 1000 | 00111010 |
| 1001 | 10001111 |  | 1001 | 10001011 |  | 1001 | 00101011 |  | 1001 | 10001111 |
| 1010 | 10000110 |  | 1010 | 00011100 |  | 1010 | 10001111 |  | 1010 | 10001111 |
| 1011 | 00001101 |  | 1011 | 10001111 |  | 1011 | 10001110 |  | 1011 | 00000000 |
| 1100 | 10001111 |  | 1100 | 10001101 |  | 1100 | 00101110 |  | 1100 | 00000000 |
| 1101 | 10001000 |  | 1101 | 00011111 |  | 1101 | 10001111 |  | 1101 | 00000000 |
| 1110 | 00010000 |  | 1110 | 10001111 |  | 1110 | 01000111 |  | 1110 | 00000000 |
| 1111 | 10001111 |  | 1111 | 01000111 |  | 1111 | 01001000 |  | 1111 | 00000000 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Módulo “*rom16x8\_30*” | |  | Módulo “*rom16x8\_31*” | |  | Módulo “*rom16x8\_32*” | |  | Módulo “*rom16x4\_33*” | |
|  |  |  |  |  |  |  |  |  |  |  |
| 0000 | 00011100 |  | 0000 | 00000001 |  | 0000 | 00000010 |  | 0000 | 10001111 |
| 0001 | 00011101 |  | 0001 | 01000100 |  | 0001 | 10000100 |  | 0001 | 00000000 |
| 0010 | 01000100 |  | 0010 | 00000010 |  | 0010 | 00010011 |  | 0010 | 00000000 |
| 0011 | 00000010 |  | 0011 | 01000010 |  | 0011 | 01000010 |  | 0011 | 00000000 |
| 0100 | 01000100 |  | 0100 | 00000000 |  | 0100 | 00000011 |  | 0100 | 00000000 |
| 0101 | 00000011 |  | 0101 | 00010111 |  | 0101 | 00010110 |  | 0101 | 00000000 |
| 0110 | 01000111 |  | 0110 | 10000001 |  | 0110 | 01000100 |  | 0110 | 00000000 |
| 0111 | 00010000 |  | 0111 | 00101010 |  | 0111 | 00000011 |  | 0111 | 00000000 |
| 1000 | 10000001 |  | 1000 | 01000100 |  | 1000 | 10000000 |  | 1000 | 00000000 |
| 1001 | 00101010 |  | 1001 | 00000000 |  | 1001 | 00010011 |  | 1001 | 00000000 |
| 1010 | 01000100 |  | 1010 | 01000010 |  | 1010 | 01000010 |  | 1010 | 00000000 |
| 1011 | 00000000 |  | 1011 | 00000010 |  | 1011 | 00000011 |  | 1011 | 00000000 |
| 1100 | 00011001 |  | 1100 | 01000011 |  | 1100 | 01000011 |  | 1100 | 00000000 |
| 1101 | 01001000 |  | 1101 | 00000001 |  | 1101 | 00000010 |  | 1101 | 00000000 |
| 1110 | 00010000 |  | 1110 | 00010000 |  | 1110 | 00011100 |  | 1110 | 00000000 |
| 1111 | 01000100 |  | 1111 | 01000100 |  | 1111 | 00011101 |  | 1111 | 00000000 |

* 1. **Minimizações:**

Para a implementação das ROMs 16x8, foi utilizado a mesma configuração que as unidades de memória ROM 16x4 desenvolvidas na atividade anterior, com a diferença que foram adicionadas mais outros 4 bits de saída. Dessa forma, o módulo *“decodificadorBIN\_ONEH”* (descrito no laboratório anterior) e as portas *GND* e *VCC* foram empregues, tornando desnecessária qualquer minimização.

* 1. **Esquemático dos circuitos:**

Os diagramas esquemáticos de cada módulo ROM 16x8 desenvolvidos estão apresentados nas figuras 9 a 20.

|  |
| --- |
|  |
| *Figura 9: Diagrama esquemático do circuito do módulo “rom16x8\_00” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá o módulo “rom64x8\_0”.* |

|  |
| --- |
|  |
| *Figura 10: Diagrama esquemático do circuito do módulo “rom16x8\_zerada” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá algumas das ROM 64x8 do módulo “memprog”. Esse circuito foi desenvolvido para poder preencher endereços não codificados com instruções com bits zero.* |

|  |
| --- |
|  |
| *Figura 11: Diagrama esquemático do circuito do módulo “rom16x8\_10” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá o módulo“rom64x8\_1”.* |

|  |
| --- |
|  |
| *Figura 12: Diagrama esquemático do circuito do módulo “rom16x8\_11” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá o módulo“rom64x8\_1”.* |

|  |
| --- |
|  |
| *Figura 13: Diagrama esquemático do circuito do módulo “rom16x8\_20” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá o módulo“rom64x8\_2”.* |

|  |
| --- |
|  |
| *Figura 14: Diagrama esquemático do circuito do módulo “rom16x8\_21” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá o módulo“rom64x8\_2”.* |

|  |
| --- |
|  |
| *Figura 15: Diagrama esquemático do circuito do módulo “rom16x8\_22” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá o módulo“rom64x8\_2”.* |

|  |
| --- |
|  |
| *Figura 16: Diagrama esquemático do circuito do módulo “rom16x8\_23” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá o módulo“rom64x8\_2”.* |

|  |
| --- |
|  |
| *Figura 17: Diagrama esquemático do circuito do módulo “rom16x8\_30” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá o módulo“rom64x8\_3”.* |

|  |
| --- |
|  |
| *Figura 18: Diagrama esquemático do circuito do módulo “rom16x8\_31” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá o módulo“rom64x8\_3”.* |

|  |
| --- |
|  |
| *Figura 19: Diagrama esquemático do circuito do módulo “rom16x8\_32” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá o módulo“rom64x8\_3”.* |

|  |
| --- |
|  |
| *Figura 20: Diagrama esquemático do circuito do módulo “rom16x8\_33” o qual implementa uma unidade de memória com 16 palavras de 8 bits e constituirá o módulo“rom64x8\_3”.* |

* 1. **Simulações:**

Para testar as unidades de memórias desenvolvidas, cada uma delas passou por uma simulação acessando todos os seus endereços e comparando os valores de saída obtidos com os esperados (presentes nas tabelas de especificação). Um exemplo dessas simulações está apresentado na figura 21, na qual as palavras de memória guardadas no módulo *“rom16x8\_00”* foram testadas. Os demais módulos também passaram pela mesma simulação, porém os resultados delas não serão apresentados para evitar tornar a leitura deste relatório repetitiva.

|  |
| --- |
|  |
| *Figura 21: Simulação funcional do módulo “rom16x8\_00”. Na simulação, todos os endereços disponíveis da memória são acessados e, assim como se pode observar, as palavras de memória obtidas na saída são iguais as esperadas para a codificação do programa. Os outros 11 módulos desse tipo também passaram pela mesma simulação, sendo que todos mantiveram o comportamento esperado. As outras simulações não serão mostradas em figuras para evitar repetitividade.* |

1. **Projeto dos módulos ROM 16x8:**
   1. **Escopo:**

Projeto de circuitos combinacionais que implementem dispositivos de memória não volátil de leitura exclusiva com 64 palavras de memória de 8 bits. Os dispositivos devem ser implementados de modo a conter o conteúdo adequado para a implementação de cada módulo usado no projeto, bem como utilizarão dos módulos ROM 16x8 desenvolvidos anteriormente para facilitar este processo.

* 1. **Especificação de alto nível:**

*Entradas:*

,com e ;

.

*Saída:*

, com e .

*Função de Saída:*

,

onde significa o conteúdo da memória no endereço indicado pela entrada .

* 1. **Especificação binária:**

*Entradas:*

,com e ;

.

*Saída:*

, com e .

*Função de Saída:*

Cada módulo do tipo ROM 64x8 apresenta uma função a qual depende do conteúdo contido nele. Dessa forma, os módulos *“rom64x8\_0”, “rom64x8\_1”, “rom64x8\_2”* e *“rom64x8\_3”* serão implementados de modo a armazenar os programas A, B, C e D, respectivamente, usando de combinações dos módulos ROM 16x8 criados na seção anterior. Caso selecionados, as saídas de cada módulo deverão ser idênticas às palavras mostradas na coluna “Conteúdo” das tabelas desses programas, sendo zero no caso de endereços não especificados. Portanto, a combinação adequada de módulos menores a serem empregues é a dada pela tabela abaixo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Endereços | Módulos usados em *“rom64x8\_0”* | Módulos usados em *“rom64x8\_1”* | Módulos usados em *“rom64x8\_2”* | Módulos usados em *“rom64x8\_3”* |
| 0 a 15 | *“rom16x8\_00”* | *“rom16x8\_10”* | *“rom16x8\_20”* | *“rom16x8\_30”* |
| 16 a 31 | *“rom16x8\_zerada”* | *“rom16x8\_11”* | *“rom16x8\_21”* | *“rom16x8\_31”* |
| 32 a 47 | *“rom16x8\_zerada”* | *“rom16x8\_zerada”* | *“rom16x8\_22”* | *“rom16x8\_32”* |
| 48 a 63 | *“rom16x8\_zerada”* | *“rom16x8\_zerada”* | *“rom16x8\_23”* | *“rom16x8\_33”* |

* 1. **Minimizações:**

Como o circuito das unidades de memória ROM 64x8 serão construídos a partir de quatro módulos de memória 16x8 já desenvolvidos, será necessário criar uma lógica para alocar 16 endereço a cada uma dessas unidades. Tal lógica pode ser implementada a partir dos sinais de entrada , e acionando o sinal de seleção dos módulos internos ( para a memória com os primeiros 16 endereços, para a memória que terá os endereços 16 a 31, para memória que receberá os endereços 32 a 47 e para a memória que guardará os últimos 16 endereços), de acordo com a tabela verdade abaixo:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |
| 0 | 0 | 1 | 0 | ‘0 | 0 | 0 |  |  |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |  |  |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |  |  |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |  |  |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |  |  |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |  |  |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |  |  |

Assim, basta ligar os bits restantes do sinal , isto é, () nas entradas respectivas de ambos os módulos e ligar a saída destes ao sinal de saída .

* 1. **Esquemático dos circuitos:**

Os diagramas esquemáticos de cada módulo ROM 64x8 desenvolvidos estão apresentados nas figuras 22 a 25.

|  |
| --- |
| Diagrama, Esquemático  Descrição gerada automaticamente |
| *Figura 22: Diagrama esquemático do circuito do módulo “rom64x8\_0”. Este módulo foi implementado de forma a armazenar o programa A e será empregue para a construção do circuito final do módulo “memprog”.* |

|  |
| --- |
| Diagrama, Esquemático  Descrição gerada automaticamente |
| *Figura 23: Diagrama esquemático do circuito do módulo “rom64x8\_1”. Este módulo foi implementado de forma a armazenar o programa B e será empregue para a construção do circuito final do módulo “memprog”.* |

|  |
| --- |
| Diagrama, Esquemático  Descrição gerada automaticamente |
| *Figura 24: Diagrama esquemático do circuito do módulo “rom64x8\_2”. Este módulo foi implementado de forma a armazenar o programa C e será empregue para a construção do circuito final do módulo “memprog”.* |

|  |
| --- |
| Diagrama, Esquemático  Descrição gerada automaticamente |
| *Figura 25: Diagrama esquemático do circuito do módulo “rom64x8\_3”. Este módulo foi implementado de forma a armazenar o programa D e será empregue para a construção do circuito final do módulo “memprog”.* |

* 1. **Simulações:**

Para testar os módulos criados, como os módulos de memória menor já haviam sido aprovados nas simulações que verificavam todo seu conteúdo, cada uma delas passou por uma simulação acessando somente alguns de seus endereços e comparando os valores de saída obtidos com os esperados (presentes nas tabelas de especificação). Isto foi feito com o intuito de se testar a lógica de seleção implementada. Um exemplo dessas simulações está apresentado na figura 26, na qual as instruções armazenadas no módulo *“rom64x8\_3”* foram testadas. Os demais módulos também passaram pela mesma simulação, porém os resultados delas não serão apresentados para evitar tornar a leitura deste relatório repetitiva.

|  |
| --- |
| Interface gráfica do usuário, Aplicativo, Word  Descrição gerada automaticamente |
| *Figura 26: Simulação funcional do módulo “rom64x8\_3”. Na simulação, alguns dos endereços da memória são acessados e, assim como se pode observar, as palavras de memória obtidas na saída são iguais as esperadas para a codificação do programa. Os outros 3 módulos desse tipo também passaram pela mesma simulação, sendo que todos mantiveram o comportamento adequado. As outras simulações não serão mostradas em figuras para evitar repetitividade.* |

1. **Projeto do módulo “memprog”:**
   1. **Escopo:**

Projeto de um circuito combinacional que reúna os quatro módulos de memória ROM 64x8 desenvolvidos em um único sistema, tornando-os accessíveis a partir de uma entrada seletora que especifica qual bloco deve fornecer seu conteúdo à saída. É válido destacar que essa implementação é idêntica à empregue na construção do módulo *“rom\_4blocks”* da atividade de laboratório 5, mudando-se somente alguns nomes de entradas e saídas. Por isso, boa parte da especificação e minimização pode ser reaproveitada.

* 1. **Especificação de alto nível:**

*Entradas:*

,com e ;

, com e ;

.

*Saída:*

, com e .

*Função de Saída:*

,

onde significa o conteúdo do módulo k de memória no endereço indicado pela entrada .

* 1. **Especificação binária:**

*Entradas:*

,com e ;

, com e ;

.

*Saída:*

, com e .

*Função de Saída:*

Esse circuito será implementado a partir dos 4 módulos de memória do tipo ROM 64x8 desenvolvidos na seção anterior, portanto, sua função de saída será praticamente igual as funções de saída de cada um deles. A única diferença é que a saída dependera dos sinais de seleção da entrada (que atuará como multiplexador) e habilitação da entrada (que permitirá ou não a escolha de um desses blocos).

* 1. **Minimizações:**

A partir do raciocínio apresentado anteriormente, sendo os sinais de entrada de ativação dos módulos *“rom64x8\_0”, “rom64x8\_1”, “rom64x8\_2”* e *“rom64x8\_3”* respectivamente iguais a , , e , teremos a seguinte tabela verdade:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | ; |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |  |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |  | ; |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |  |  |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |  | ; |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |  |  |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |  | ; |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |  |  |

Dessa forma, para completar o circuito basta ligar o sinal entrada na entrada de endereço de cada módulo e conectar as saídas deles em um único barramento que constitui a saída .

* 1. **Esquemático do circuito:**

O diagrama esquemático do módulo *“memprog”* já foi apresentado anteriormente. Ele se encontra na seção “Visão geral”, figura 2.

* 1. **Simulações:**

A simulação funcional feita para testar esse módulo está disposta figura 27. Nela, percorreu-se os primeiros endereços de cada bloco de memória para se certificar que eles estavam sendo acessados apropriadamente. Os demais endereços não foram verificados, pois, como as unidades de memória 64x8 e 16x8 foram testadas uma a uma, admitiu-se que não seria necessário.

|  |
| --- |
|  |
| *Figura 27: Simulação funcional do módulo “memprog”. Nota-se que os blocos são acessados assim como o esperado e as saídas geradas são as mesmas que as esperadas.* |

1. **Unidade central de processamento (CPU):**

Dentre os módulos usados para construir os três subcircuitos que constituem a CPU do computador simplificado, dois deles (*“bus\_ula”* e *“contador\_mod\_256”*) já foram abordados em atividades anteriores. Portanto, resta especificar o comportamento dos 3 multiplexadores (espalhados entre os subcircuitos) e dos 3 decodificadores (pertencentes ao subcircuito Decodificador de instruções). Porém, antes de iniciar a descrição destes elementos, convém analisar alguns sinais de controle importantes, os quais coordenarão o funcionamento do circuito total. A tabela abaixo relaciona cada sinal do módulo *“cpu”* a seu emissor(es), receptor(es), bem como a sua função.

|  |  |  |  |
| --- | --- | --- | --- |
| Sinal | Emissor(es) | Receptor(es) | Função |
|  | Módulo *“memprog”* (externo / entrada da cpu) | Subcircuitos Contador de Programa, Decodificador de Instruções e ULA | Fornece a instrução a ser decodificada, ou um operando a ser usado em operações de transferência ou de desvio, ou selecionar uma operação da ULA. |
|  | Módulo *“ram”* (externo/entrada da CPU) | Subcircuito ULA | Fornece um operando a ser usado em operações de transferência |
|  | Chaves externas (entrada da CPU) | Subcircuitos Decodificador de Instruções e ULA | Fornece um operando a ser usado para operações de transferência |
|  | Chave externa (entrada da CPU) | Subcircuitos Contador de Programa, Decodificador de Instruções e ULA | Limpar todos os elementos sequenciais do processador |
|  | Chave externa (entrada da CPU) | Subcircuito Relógio | Gerar o clock com o qual a CPU trabalhará |
|  | Subcircuito Contador de Programa | Subcircuito Contador de Programa e módulo *“memprog”* (externo) | Atuar como contador de programa, acessando sequencialmente instruções na memória ROM , mas podendo ser alterado por instruções de desvio. |
|  | Subcircuito ULA | Display de 7 segmentos externo | Alimenta um display de 7 segmento com o valor presente no barramento da ULA |
|  | Subcircuito ULA | Display de 7 segmentos externo | Alimenta um display de 7 segmento com o valor presente no registrador acumulador da ULA |
|  | Subcircuito ULA | Display de 7 segmentos externo | Alimenta um display de 7 segmento com o valor presente no registrador B da ULA |
|  | Subcircuito ULA | Display de 7 segmentos externo | Alimenta um display de 7 segmento com o valor presente no registrador A da ULA |
| , , e | Subcircuito ULA | LEDs externos | Alimenta LEDs externos ao módulo com o valor presente no registrador flags da ULA |
|  | Subcircuito ULA | Subcircuitos Decodificador de Instruções | Fornece o valor do registrador acumulador da ULA para eventuais operações de transferência para a memória RAM |
|  | Subcircuito ULA | Subcircuitos Decodificador de Instruções | Fornece o valor do registrador de flags da ULA para cálculo de condições de desvio |
|  | Subcircuitos Decodificador de Instruções | Módulo *“ram”* (externo/saída da CPU) | Fornece, quando necessário, um valor para a escrita na memória RAM |
|  | Subcircuitos Decodificador de Instruções | Módulo *“ram”* (externo/saída da CPU) | Seleciona a opção de leitura ou escrita na memória RAM conforme necessário |
|  | Subcircuitos Decodificador de Instruções | Módulo *“ram”* (externo/saída da CPU) | Habilita a leitura/escrita da memória RAM quando necessário |
|  | Subcircuito Relógio | Subcircuitos Contador de Programa, Decodificador de Instruções e ULA, bem como módulo *“ram”* | Fornece o *clock* em que o computador operará |
|  | Subcircuitos Decodificador de Instruções | Subcircuito ULA | Habilita leitura ou escrita no barramento da ULA conforme necessário |
|  | Promove a limpeza síncrona do registrador A da ULA, conforme solicitada |
|  | Promove a limpeza síncrona do registrador B da ULA, conforme solicitada |
|  | Promove a limpeza síncrona do registrador AC da ULA, conforme solicitada |
|  | Promove a limpeza síncrona do registrador FL da ULA, conforme solicitada |
|  | Promove a carga síncrona do registrador AC da ULA, conforme solicitada |
|  | Promove a limpeza síncrona do registrador FL da ULA, conforme solicitada |
|  | Sinais de multiplexação que selecionam o que será fornecido à entrada de dados da ULA |
|  |
|  | Promove a carga síncrona do registrador A da ULA, conforme solicitada |
|  | Promove a carga síncrona do registrador B da ULA, conforme solicitada |
|  | Subcircuitos Decodificador de Instruções | Subcircuitos Decodificador de Instruções | Indica se a próxima instrução é um operando, ou seja, ela não deve ser codifica, mas sim utilizada em alguma operação de transferência |
|  | Subcircuitos Decodificador de Instruções | Subcircuitos Decodificador de Instruções | Sinal de multiplexação que seleciona o que será fornecido para escrita na RAM |
|  | Subcircuitos Decodificador de Instruções | Subcircuitos Contador de Programa | Sinal de multiplexação que seleciona o que será fornecido para carga no contador de programa |
|  | Subcircuitos Decodificador de Instruções | Subcircuitos Contador de Programa | Promove a carga síncrona do contador de programa |
|  | Subcircuitos Decodificador de Instruções | Subcircuitos Decodificador de Instruções | Indica se a próxima instrução é um operando, ou seja, ela não deve ser codifica, mas sim utilizada em alguma operação de desvio |

Agora, daremos início a especificação dos circuitos implementados para compor o módulo *“cpu”*. Iniciaremos pelos multiplexadores em virtude de sua simplicidade.

1. **Projeto do módulo *“mux8x4”*:**
   1. **Escopo:**

Projeto de um circuito combinacional que implemente um multiplexador capaz de transmitir para a saída os sinais de um entre dois barramentos de entrada. Essa transmissão deve estar condicionada ao sinal de habilitação .

* 1. **Especificação de alto nível:**

*Entradas:*

,com e ;

,com e ;

;

.

*Saída:*

, com e .

*Função de Saída:*

;

* 1. **Especificação binária:**

*Entradas:*

,com e ;

,com e ;

;

.

*Saída:*

, com e .

*Função de Saída:*

Como a multiplexação será feita entre dois barramentos de dados, ambos com 4 bits, podemos fazer uso do módulo *“bus\_tristate”* implementado durante a atividade de laboratório 4. Dessa forma, basta conectar cada barramento de entrada a uma instância desse módulo e habilitá-la aplicando uma lógica nas entradas de seleção. Essa lógica é descrita pela tabela verdade abaixo, na qual representa a entrada de habilitação do módulo *“bus\_tristate”* ao qual está ligado ao barramento de entrada .

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

Para completar o circuito, é preciso ainda ligar cada bit do barramento de saída selecionado a uma porta OR com a entrada de *enable*, pois a saída do multiplexador deve zerar quando for zero.

* 1. **Minimizações:**

A minimização da tabela verdade acima é trivial:

|  |  |
| --- | --- |
| ; | ; |

* 1. **Esquemático do circuito:**

Seguindo a lógica descrita, o circuito do módulo *“mux8x4”* foi implementado e está disposto na figura 28.

|  |
| --- |
|  |
| *Figura 28: Diagrama esquemático do circuito do módulo “mux8x4”* |

* 1. **Simulações:**

Uma simulação funcional do circuito foi efetuada para testar sua capacidade de multiplexação, porém foi necessário alterar os nomes dos barramentos de entradas por problemas de incompatibilidade do simulador. Dessa forma, o resultado da simulação está disposto na figura 29, sendo , iguais a ,, respectivamente.

|  |
| --- |
|  |
| *Figura 29: Simulação funcional do módulo “mux8x4”. Nela, os nomes dos barramentos de entrada tiveram que ser alterados por motivos de incompatibilidade do simulador. Portanto, os barramentos , correspondem aos barramentos , respectivamente.* |

1. **Projeto do módulo *“mux16x4”*:**
   1. **Escopo:**

Projeto de um circuito combinacional que implemente um multiplexador capaz de transmitir para a saída os sinais de um entre quatro barramentos de entrada. Essa transmissão deve estar condicionada ao sinal de habilitação . É válido destacar que também seria possível projetar esse circuito usando-se de dois módulos *“mux8x4”*. Entretanto, como a saída desses módulos teria que ser inserida novamente em *buffers tri-state*, optou-se por projetar um novo circuito.

* 1. **Especificação de alto nível:**

*Entradas:*

,com e ;

,com e ;

,com e ;

,com e ;

;

;

.

*Saída:*

, com e .

*Função de Saída:*

;

* 1. **Especificação binária:**

*Entradas:*

,com e ;

,com e ;

,com e ;

,com e ;

;

;

.

*Saída:*

, com e .

*Função de Saída:*

Como a multiplexação será feita entre quatro barramentos de dados, todos com 4 bits, podemos fazer uso do módulo *“bus\_tristate”* de modo análogo ao feito no módulo *“mux8x4”*. Nesse caso, a lógica a ser aplicada nas entradas de seleção é descrita pela tabela verdade abaixo, na qual representa a entrada de habilitação do módulo *“bus\_tristate”* ao qual está ligado ao barramento de entrada .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

Para completar o circuito, é preciso ainda ligar cada bit do barramento de saída selecionado a uma porta OR com a entrada de *enable*, pois a saída do multiplexador deve zerar quando for zero.

* 1. **Minimizações:**

Como cada saída apresentada na tabela verdade acima só é alta para uma combinação das entradas, não é necessário o uso de mapas de Karnaugh para a minimização. Assim, teremos que:

|  |  |
| --- | --- |
| ; | ; |
| ; | ; |
|  |  |

* 1. **Esquemático do circuito:**

Seguindo a lógica acima, o circuito do módulo *“mux16x4”* foi implementado e está disposto na figura 30.

|  |
| --- |
|  |
| *Figura 30: Diagrama esquemático do circuito do módulo “mux16x4”* |

* 1. **Simulações:**

Uma simulação funcional do circuito foi efetuada para testar sua capacidade de multiplexação, porém foi necessário (novamente) alterar os nomes dos barramentos de entradas por problemas de incompatibilidade do simulador. Dessa forma, o resultado da simulação está disposto na figura 31, sendo ,,, iguais a ,,, , respectivamente.

|  |
| --- |
|  |
| *Figura 31: Simulação funcional do módulo “mux16x4”. Nela, os nomes dos barramentos de entrada tiveram que ser alterados por motivos de incompatibilidade do simulador. Portanto, os barramentos , , , correspondem aos barramentos , , , respectivamente.* |

1. **Projeto do módulo *“mux16x8”*:**
   1. **Escopo:**

Projeto de um circuito combinacional que implemente um multiplexador capaz de transmitir para a saída os sinais de um entre dois barramentos de entrada . Essa transmissão deve estar condicionada ao sinal de habilitação . Neste módulo, duas instâncias do módulo *“mux8x4”* serão empregues, uma vez que , diferentemente do módulo *“mux16x4”*, o módulo *“mux16x8”* tem seu desenvolvimento facilitado pela utilização de tal módulo.

* 1. **Especificação de alto nível:**

*Entradas:*

,com e ;

,com e ;

;

.

*Saída:*

, com e .

*Função de Saída:*

;

* 1. **Especificação binária:**

*Entradas:*

,com e ;

,com e ;

;

.

*Saída:*

, com e .

*Função de Saída:*

Para implementar este multiplexador usando dois módulos *“mux8x4”*, basta ligar os 4 bits mais significativos dos dois barramentos de entrada nas entradas de dados de um dos módulos, bem como conectar os bits menos significativos restantes nas entradas de dados do outro módulo. Além disso, é necessário não só conectar as entradas e dos dois módulos às entradas do circuito de nome igual, mas também juntar as saídas dos multiplexadores internos a um único barramento de saída ().

* 1. **Minimizações:**

Seguindo o raciocínio apresentado acima, não será necessária nenhuma minimização.

* 1. **Esquemático do circuito:**

O circuito do módulo *“mux16x8”* foi construído seguindo o descrito anteriormente. Seu diagrama esquemático e está disposto na figura 32.

|  |
| --- |
|  |
| *Figura 30: Diagrama esquemático do circuito do módulo “mux16x8”* |
|  |

* 1. **Simulações:**

Uma simulação funcional do circuito foi feita para testar sua capacidade de multiplexação, porém foi necessário alterar (outra vez) os nomes dos barramentos de entradas por problemas de incompatibilidade do simulador. Dessa forma, o resultado da simulação está disposto na figura 31, sendo , iguais a ,, respectivamente.

|  |
| --- |
|  |
| *Figura 31: Simulação funcional do módulo “mux16x8”. Nela, os nomes dos barramentos de entrada tiveram que ser alterados por motivos de incompatibilidade do simulador. Portanto, os barramentos , correspondem aos barramentos , respectivamente.* |

1. **Projeto do decodificador de instruções aritméticas:**
   1. **Escopo:**

Projeto de um circuito combinacional que gere os sinais de controle para a efetuação das operações relacionadas às instruções aritméticas.

* 1. **Especificação de alto nível:**

*Entradas:*

;

.

*Saída:*

.

*Função de saída:*

|  |  |
| --- | --- |
| Entrada | Sinais de saída devem gerar a operação (caso habilitado) |
| 0 | ADD |
| 2 | ADDC |
| 4 | SUB |
| 6 | INC |
| 7 | DEC |
| 8 | NEG |
| 9 | CMPL |
| 10 | CLRA |
| 11 | CLRB |
| 12 | CLRAC |
| 13 | CLRFL |

* 1. **Especificação binária:**

*Entradas:*

, com e ;

.

*Saída:*

.

*Função de saída:* Tabela verdade abaixo

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | x | x | x | x | x | x | x |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | x | x | x | x | x | x | x |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | x | x | x | x | x | x | x |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | x | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | x | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | x | 0 | 0 | 1 | 0 | x | 0 |
| 1 | 1 | 0 | 1 | x | 0 | 0 | 0 | 1 | 0 | x |
| 1 | 1 | 1 | 0 | x | x | x | x | x | x | x |
| 1 | 1 | 1 | 1 | x | x | x | x | x | x | x |

* 1. **Minimizações:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| : |  |  |  |  |  | : |  |  |  |  |
|  | 1 | x | x | 1 |  |  | 0 | x | x | 0 |
|  | 1 | x | 1 | 1 |  |  | 0 | x | 0 | 0 |
|  | x | x | x | x |  |  | 0 | 0 | x | x |
|  | 1 | 1 | x | x |  |  | 0 | 0 | 0 | 1 |
|  |  |  |  |  |  |  |  |  |  |  |
| : |  |  |  |  |  | : |  |  |  |  |
|  | 0 | x | x | 0 |  |  | 0 | x | x | 0 |
|  | 0 | x | 0 | 0 |  |  | 0 | x | 0 | 0 |
|  | 0 | 0 | x | x |  |  | 1 | 0 | x | x |
|  | 0 | 0 | 1 | 0 |  |  | 0 | 0 | 0 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |
| : |  |  |  |  |  | : |  |  |  |  |
|  | 0 | x | x | 0 |  |  | 1 | x | x | 1 |
|  | 0 | x | 0 | 0 |  |  | 1 | x | 1 | 1 |
|  | 0 | 1 | x | x |  |  | x | 0 | x | x |
|  | 0 | 0 | 0 | 0 |  |  | 1 | 1 | 0 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | : |  |  |  |  |  |  |  |
|  |  |  |  | 1 | x | x | 1 |  |  |  |
|  |  |  |  | 1 | x | 1 | 1 |  |  |  |
|  |  |  |  | 0 | x | x | x |  |  |  |
|  |  |  |  | 1 | 1 | 0 | 0 |  |  |  |

A partir dos mapas de Karnaugh apresentados acima e adicionando a lógica de *enable* da entra , obtemos as seguintes expressões minimizadas:

|  |  |  |
| --- | --- | --- |
|  | ; |  |
|  | ; |  |
|  | ; |  |
|  | | |
|  | | |

É válido destacar que a saída deve ser protegida por lógica *“tri-state”*, pois esse sinal é gerado tanto pelo decodificador de instruções aritméticas, quanto pelo decodificador de instruções de transferência.

* 1. **Esquemático do circuito:**

O circuito foi implementado seguindo as minimizações apresentadas anteriormente. Seu diagrama esquemático está apresentado na figura 32.

|  |  |
| --- | --- |
|  | *Figura 32: Diagrama esquemático do módulo “decod\_inst\_art”* |

* 1. **Simulações:**

Uma simulação funcional considerando todas as entradas válidas do módulo *“decod\_inst\_art”* foi efetuada. Seu resultado está disposto na figura 33.

|  |
| --- |
|  |
| *Figura 33: Simulação funcional do módulo “decod\_inst\_art”. Nela, todas as possibilidades de entradas válidas para o módulo são testadas. Nota-se que o resultado obtido é coerente com o especificado pela tabela verdade.* |

1. **Projeto do decodificador de instruções de transferência:**
   1. **Escopo:**

Projeto de um circuito sequencial o qual gere os sinais de controle para a efetuação das operações relacionadas às instruções de transferência. Neste decodificador, o momento em que os sinais devem ser gerados precisa ser considerado, pois as operações que apresentam operando ocorrem em dois pulsos de *clock*. Isto faz com que instruções as quais indiquem operações deste tipo precisem armazenar seu resultado de decodificação, para, então, fornecê-lo ao sistema somente após a próxima borda ativa do *clock*. Para ser possível a implementação desse mecanismo, o módulo do decodificador de instruções de desvio (*“decod\_inst\_trf”*) foi divido em duas partes: o módulo *“decod\_inst\_trf\_interno”* (parte combinacional, responsável por gerar os sinais de saída adequados para cada operação) e o conjunto de módulos “*decod\_inst\_signal\_register”* (parte sequencial, ativada por dois sinais de controle e responsável por emitir o sinal imediatamente, ou armazena-lo em um registrador de um bit e emiti-lo no próximo pulso de *clock*).

* 1. **Especificação de alto nível:**

*Entradas:*

;

;

*Saída:*

.

*Função de saída:*

|  |  |  |
| --- | --- | --- |
| Entrada | Sinais de saída devem gerar a operação (caso habilitado) | Quando gerar os sinais |
| 0 | MVA, AC | Imediatamente |
| 1 | MVB, AC | Imediatamente |
| 2 | LDA, OP | Após o próximo pulso de *clock* |
| 3 | LDB, OP | Após o próximo pulso de *clock* |
| 4 | ST OP | Após o próximo pulso de *clock* |
| 5 | LDAI, OP | Após o próximo pulso de *clock* |
| 6 | LDBI, OP | Após o próximo pulso de *clock* |
| 7 | IN A | Imediatamente |
| 8 | IN B | Imediatamente |
| 9 | IN OP | Após o próximo pulso de *clock* |

* 1. **Especificação binária:**

*Entradas:*

, com e ;

.

*Saída:*

.

*Função de saída:*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tabela verdade para o módulo interno *“decod\_inst\_trf\_interno”*: | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | x | x | 1 | 0 | 0 | x | x |
| 0 | 0 | 0 | 1 | 0 | x | x | 0 | 1 | 0 | x | x |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | x | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | x | 0 |
| 0 | 1 | 0 | 0 | x | x | x | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | x | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | x | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | x | x |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | x | x |
| 1 | 0 | 0 | 1 | x | x | x | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | x | x | x | x | x | x | x | x |
| 1 | 0 | 1 | 1 | x | x | x | x | x | x | x | x |
| 1 | 1 | 0 | 0 | x | x | x | x | x | x | x | x |
| 1 | 1 | 0 | 1 | x | x | x | x | x | x | x | x |
| 1 | 1 | 1 | 0 | x | x | x | x | x | x | x | x |
| 1 | 1 | 1 | 1 | x | x | x | x | x | x | x | x |

|  |  |  |
| --- | --- | --- |
| Tabela verdade de excitação do *flip-flop* interno do módulo *decod\_inst\_signal\_register*: | | |
|  |  |  |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| Tabela verdade de habilitação dos *buffers tri-state* internos do módulo *decod\_inst\_signal\_register*: | | | |
|  |  | *enable* da saída imediata () | *enable* da saída ligada ao *flip-flop* () |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |

* 1. **Minimizações:**

Para as saídas do módulo interno *“decod\_inst\_trf\_interno”*, teremos os seguintes mapas de Karnaugh:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| : |  |  |  |  |  | : |  |  |  |  |
|  | 0 | 0 | 1 | 1 |  |  | x | x | 0 | 0 |
|  | x | 1 | 1 | 1 |  |  | x | 0 | 1 | 0 |
|  | x | x | x | x |  |  | x | x | x | x |
|  | 1 | x | x | x |  |  | 1 | x | x | x |
|  |  |  |  |  |  |  |  |  |  |  |
| : |  |  |  |  |  | : |  |  |  |  |
|  | x | x | 1 | 1 |  |  | 1 | 0 | 0 | 1 |
|  | x | 0 | 0 | 0 |  |  | 0 | 1 | 1 | 0 |
|  | x | x | x | x |  |  | x | x | x | x |
|  | 0 | x | x | x |  |  | 0 | 0 | x | x |
|  |  |  |  |  |  |  |  |  |  |  |
| : |  |  |  |  |  | : |  |  |  |  |
|  | 0 | 1 | 1 | 0 |  |  | 0 | 0 | 1 | 1 |
|  | 0 | 0 | 0 | 1 |  |  | 1 | 1 | 0 | 1 |
|  | x | x | x | x |  |  | x | x | x | x |
|  | 1 | 0 | x | x |  |  | 0 | 1 | x | x |
|  |  |  |  |  |  |  |  |  |  |  |
| : |  |  |  |  |  | : |  |  |  |  |
|  | x | x | x | x |  |  | x | x | 0 | 0 |
|  | 0 | x | x | x |  |  | 1 | 0 | x | 0 |
|  | x | x | x | x |  |  | x | x | x | x |
|  | x | 1 | x | x |  |  | x | 1 | x | x |

Estes mapas gerarão as seguintes expressões mínimas (adicionando-se a lógica de *enable* da entrada ):

|  |  |  |
| --- | --- | --- |
|  | ; |  |
|  | ; |  |
|  | ; |  |
|  | ; |  |

Já para o módulo *“decod\_inst\_signal\_register”*, as seguintes expressões minimizadas são obtidas diretamente da tabela verdade:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ; |  | ; |  |

Neste decodificador, os dois sinais de controle dos registradores devem ser e (sendo este o valor anterior de , armazenado dentro de um dos registradores), pois, como nessa implementação os *flip-flops* invariavelmente vão zerar em no máximo dois pulsos de *clock* , se for igual a um, o circuito não deve emitir valor algum, sendo necessário liberar a saída sequencial. Além disso, alto implica que a instrução anterior era de transferência e apresenta operando, fazendo com que o circuito tenha que liberar os valores guardados nos registradores.

Outro detalhe é que a saída recebe um circuito especial, pois, diferentemente das demais, quando o decodificador não está selecionado e não está trabalhando no processamento de um operando, essa saída deve ficar em alta impedância, para não atrapalhar possíveis operações aritméticas. Por fim, para implementar o mecanismo de limpeza síncrona, basta conectar a entrada do decodificador nas entradas dos registradores de sinal.

* 1. **Esquemático do circuito:**

Seguindo a lógica explicitada acima, os módulos descritos foram implementados e seus diagramas esquemáticos estão apresentados nas figuras 34 (módulo *“decod\_inst\_trf\_interno”*), 35 (módulo *“decod\_inst\_signal\_register”*), 36 (circuito final do módulo *“decod\_inst\_trf”*).

|  |  |
| --- | --- |
|  | *Figura 34: diagrama esquemático do módulo* *“decod\_inst\_trf\_interno”. Este circuito foi utilizado para implementar o circuito do decodificador de instruções de transferência.* |

|  |  |
| --- | --- |
|  | *Figura 35: diagrama esquemático do módulo* *“decod\_inst\_signal\_register”. Este circuito foi utilizado para implementar não só o circuito do decodificador de instruções de transferência, mas também o circuito do decodificador de instruções de desvio .* |

|  |  |
| --- | --- |
|  | *Figura 36: diagrama esquemático do circuito final do módulo* *“decod\_inst\_trf”.* |

* 1. **Simulações:**

O módulo *“decod\_inst\_trf\_interno”* passou por uma simulação funcional, cujo resultado está exposto na figura 37 para verificar se os sinais gerados por ele estavam de acordo com o estabelecido pela tabela verdade. Além disso, o circuito final do módulo *“decod\_inst\_trf”* também passou por uma simulação funcional (figura 38) para testar sua capacidade de emitir os sinais de saída no momento adequado.

|  |
| --- |
|  |
| *Figura 37: Simulação funcional do módulo “decod\_inst\_trf\_interno”. Nela é possível observar que os sinais gerados estão de acordo com o indicado pela tabela verdade.* |

|  |
| --- |
|  |
| *Figura 38: Simulação funcional do módulo “decod\_inst\_trf”. Nela, é possível notar que os sinais emitidos não só estão de acordo com o estabelecido pela tabela verdade, mas também atuam no tempo adequado para se possibilitar o uso de instruções com operando. Além disso, assim como desejado, também nota-se que o valor dentro dos flip-flops sempre volta a ser zero em no máximo dois pulsos de clock.* |

1. **Projeto do decodificador de instruções de desvio:**
   1. **Escopo:**

Projeto de um circuito sequencial o qual gere os sinais de controle para a efetuação das operações relacionadas às instruções de desvio. Neste decodificador o momento em que os sinais devem ser produzidos também precisa ser considerado, pois ele também deve utilizar de dois pulsos de *clock* para lidar com instruções que tenham operandos. Portanto, o módulo *“decod\_inst\_signal\_register”*, desenvolvido logo acima, será empregue para a emissão dos sinais no momento adequado. Além disso, com o intuito de organizar o circuito final do decodificador, também será desenvolvido um módulo (*“decod\_inst\_dsv\_calc\_LDCNT”*) que, juntamente com uma instância do módulo *“decoderBIN\_ONEH”,*  calculará as condições de desvio a partir das flags.

* 1. **Especificação de alto nível:**

*Entradas:*

;

, com e

.

*Saída:*

.

*Função de saída:*

|  |  |  |  |
| --- | --- | --- | --- |
| Entrada | Sinais de saída devem gerar a operação (caso habilitado) | Quando gerar os sinais | Condição para gerar os sinais |
| 0 | B OP | Após o próximo pulso de *clock* | -- |
| 1 | BEQ OP | Após o próximo pulso de *clock* | Se |
| 2 | BNEQ OP | Após o próximo pulso de *clock* | Se |
| 3 | BCS OP | Após o próximo pulso de *clock* | Se |
| 4 | BCC OP | Após o próximo pulso de *clock* | Se |
| 5 | BMI OP | Após o próximo pulso de *clock* | Se |
| 6 | BPL OP | Após o próximo pulso de *clock* | Se |
| 7 | BVS OP | Após o próximo pulso de *clock* | Se |
| 8 | BVC OP | Após o próximo pulso de *clock* | Se |
| 9 | BHI OP | Após o próximo pulso de *clock* | Se e |
| 10 | BLS OP | Após o próximo pulso de *clock* | Se ou |
| 11 | BGE OP | Após o próximo pulso de *clock* | Se |
| 12 | BLT OP | Após o próximo pulso de *clock* | Se |
| 13 | BGT OP | Após o próximo pulso de *clock* | Se e |
| 14 | BLE OP | Após o próximo pulso de *clock* | Se ou |
| 15 | STP | Imediatamente | -- |

* 1. **Especificação binária:**

*Entradas:*

, com e ;

, com e

.

*Saída:*

.

*Função de saída:*

A tabela verdade completa deste decodificador apresentaria 9 entradas e, portanto, 512 possibilidades a serem analisadas dos sinais de saída. Como avaliar todos esses casos não seria prático, um método alternativo foi utilizado para se calcular as condições de desvio, fazendo com que a tabela verdade do circuito se reduzisse à apresentada abaixo:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tabela verdade do decodificador desconsiderando-se as condições de desvio | | | | | | |
|  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |

Como as condições a serem calculadas para a efetuação ou não da carga por meio do sinal são simples e, em sua maioria, independentes, elas foram obtidas diretamente, sem a construção de uma tabela verdade. Suas expressões estão expostas na seção a seguir.

* 1. **Minimizações:**

Nesta tabela, é possível notar que para qualquer combinação da entrada a saída é alta. Isto ocorre, pois, desconsiderando-se as condições impostas pelas *flags*, qualquer instrução provocaria um salto e, portanto, acarretaria uma carga no contador. Tal fato indica que as condições avaliadas a partir de expressões lógicas usando dos sinais do barramento modulam o sinal . Dessa forma, para implementar o cálculo dessas condições e a geração do sinal adequado (alto se a condição for verdadeira e baixo se falsa), primeiramente uma instância do módulo *“decoderBIN\_ONEH”* foi empregue para se obter a representação do valor do operando em *one-hot*. Em seguida, essa representação é enviada à uma instância do módulo *“decod\_inst\_dsv\_calc\_LDCNT”*, no qual as condições de desvio são calculadas por meio de lógica utilizando os sinais do barramento . Como cada valor da entrada está associado à uma condição diferente, as condições calculadas foram ligadas no mesmo sinal de saída (que será o sinal do módulo externo) e protegidas com *buffers tri-state*. A habilitação destes *buffers* está condicionada à representação *one-hot* recebida pelo módulo, isto é, dependendo do valor da entrada , somente uma das condições (no caso, aquela que corresponde a estabelecida pela instrução) fornecerá seu valor à saída (podendo ele ser 1, se a condição for verdadeira, ou 0, se falsa). Assim, como cada condição é simples e pode ser convenientemente obtida por meio do seu próprio texto, a seguinte tabela foi usada para o cálculo das expressões lógicas que as implementam:

|  |  |  |
| --- | --- | --- |
| Valor da entrada | Condição em forma textual | Expressão lógica gerada diretamente a partir da conversão do texto |
| 0 | Incondicional (sempre ocorre) |  |
| 1 | Se |  |
| 2 | Se |  |
| 3 | Se |  |
| 4 | Se |  |
| 5 | Se |  |
| 6 | Se |  |
| 7 | Se |  |
| 8 | Se |  |
| 9 | Se e |  |
| 10 | Se ou |  |
| 11 | Se |  |
| 12 | Se |  |
| 13 | Se e |  |
| 14 | Se ou |  |
| 15 | Incondicional (sempre ocorre) |  |

Esse modelo de implementação foi adotado, pois não só reutiliza um circuito já criado anteriormente para compactar o projeto, mas também facilita o cálculo das expressões lógicas que representam as condições, bem como possibilita trabalhar com uma tabela verdade mais simples para o circuito final.

Além disso, da tabela verdade apresentada na especificação binária e adicionando a lógica de *enable* da entrada , obtemos o seguinte mapa de Karnaugh e expressão mínima para o sinal :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| : |  |  |  |  |  |  |
|  | 1 | 1 | 1 | 1 |  |
|  | 1 | 1 | 1 | 1 |  |
|  | 1 | 1 | 0 | 1 |  |
|  | 1 | 1 | 1 | 1 |  |

Como tanto a saída quanto a saída de podem ser emitidas imediatamente ou após um pulso de *clock* (dependendo da instrução decodificada ter ou não operando), mais duas instâncias do módulo *“decod\_inst\_signal\_register”* foram empregues para implementar essa funcionalidade. Cada um desses sinais de saída foi ligado a uma dessas instâncias, sendo elas coordenadas pelos sinais de controle e (valor anterior de , armazenado dentro do registrador), por motivos análogos aos do decodificador de instrução de transferência.

Por fim, obtemos a expressão minimizada da saída diretamente da tabela verdade, pois ela só é alta em uma combinação das entradas:

Ou, para a implementação utilizada, basta ligar a saída 15 do decodificador binário para one hot à uma porta AND com a entrada . Como este sinal só em alto em um caso no qual não há operando (), não é necessário armazená-lo em um registrador, pois naturalmente ele estará zerado quando o decodificador não estiver selecionado.

* 1. **Esquemático do circuito:**

Seguindo o raciocínio apresentado acima, os módulos descritos foram implementados e seus diagramas esquemáticos estão apresentados nas figuras 39 (módulo *“decod\_inst\_dsv\_calc\_LDCNT”*), 40 (módulo *““decod\_inst\_dsv”*).

|  |
| --- |
|  |
| *Figura 39: diagrama esquemático do módulo “decod\_inst\_dsv\_calc\_LDCNT”. Este circuito foi utilizado para implementar o circuito do decodificador de instruções de desvio.* |

|  |
| --- |
|  |
| *Figura 40: diagrama esquemático do circuito final do módulo “decod\_inst\_dsv”.* |

* 1. **Simulações:**

Uma sequência de simulações funcionais com o intuito de avaliar a capacidade do circuito de gerar as saídas adequadas mediante as condições e momento devidos foi feita com o módulo *“decod\_inst\_dsv”*. Nestas simulações, todos os valores possíveis da entrada foram testados para cada entrada . Alguns exemplos dos resultados destas operações estão expostos nas figuras 41 , 42 e 43. Os demais resultados não serão mostrados para evitar repetitividade.

|  |
| --- |
|  |
| *Figura 41: Simulação funcional do circuito do decodificador de instruções de desvio considerando todas as possibilidades de flags para a instrução B OP. Nota-se que o sinal que gera o desvio () ocorre em todos os casos, pois este desvio é incondicional.* |

|  |
| --- |
|  |
| *Figura 42: Simulação funcional do circuito do decodificador de instruções de desvio considerando todas as possibilidades de flags para a instrução BLS OP. Nota-se que o sinal que gera o desvio () só ocorre se ou , assim como esperado.* |

|  |
| --- |
|  |
| *Figura 43: Simulação funcional do circuito do decodificador de instruções de desvio considerando todas as possibilidades de flags para a instrução STP. Nota-se que, nesse caso, como não há operando, os sinais que geram os efeitos da instrução são liberados no mesmo momento em que a instrução é recebida. Além disso, essa simulação também mostra a capacidade do circuito ser desativado pela entra .* |

1. **Finalização do computador simplificado:**

Agora que todos os módulos que integram o computador simplificado foram devidamente especificados, minimizados e testados, resta só uni-los, utilizando os esquemas de ligações apresentadas no roteiro do projeto. O resultado dessa junção já foi apresentado anteriormente na figura 1.

O circuito final do computador simplificado passou por três simulações funcionais, as quais testaram a capacidade dele de executar os programas A, B e C, sendo todas as três bem-sucedidas. Entretanto, só é possível apresentar de forma completa e coerente neste relatório o resultado da simulação do programa A (figura 44), pois ela é menor e pode ser exposta em uma única tela do simulador do Quartus II. Para avaliar o resultado das demais, foi necessário o uso da ferramenta de lupa e a varredura de todo o período simulado, processo o qual não é possível ser refeito de forma adequada através de *screenshots*. Convido o leitor a executar os arquivos *“WaveformPB.vwf”* e *“WaveformPC.vwf”* contidos no projeto *“computador\_simplificado”* e repetir o processo de varredura, visando verificar que o afirmado anteriormente é verdade.

|  |
| --- |
|  |
| *Figura 44: Simulação funcional do circuito do computador simplificado executando o programa A “Teste de aritmética”. Essa simulação é menor e pode ser exibida em uma única tela do simulador do Quartus II. As demais, por demandarem mais pulsos de clock, tiveram que ser avaliadas com a funcionalidade de lupa, por isso não estão expostas nesse relatório.* |

Por fim, o computador simplificado foi carregado na placa de desenvolvimento e todos os seus programas foram testados (desta vez, incluindo-se o programa D). Tais testes foram todos bem-sucedidos, fato que indica sucesso da implementação adotada.