Gabriel Kenji de Almeida

Implementação de um processador monociclo para o FPGA com o Quartus Prime: Conjunto de instruções e arquitetura base.

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Laboratório de Sistemas Computacionais: Arquitetura e Organização de Computadores.

Docente: Prof. Dr. Tiago de Oliveira

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Abril de 2019

Resumo

O presente relatório tem por finalidade descrever como ocorreu o desenvolvimento das primeiras etapas na elaboração de um processador a ser simulado pelo FPGA (Field Programmable Gate Array), um chip reprogramável a nível de portas lógicas, por meio do uso do Verilog, uma linguagem de descrição de hardware. Todos os conceitos e métodos utilizados serão devidamente explicados posteriormente. As etapas iniciais do projeto, que serão descritas nesse relatório, correspondem, de modo geral, as decisões de projeto, que englobam todas as características e particularidades do processador a ser desenvolvido. No caso, estas seriam definidas pelo conjunto de instruções utilizado, pelo formato das instruções, pelo caminho de dados, pelo conjunto de registradores utilizados, e pelos modos de endereçamento. Vale ressaltar que, possivelmente por se tratar das primeiras etapas na implementação do processador, não houveram problemas conceituais e todas as questões foram devidamente trabalhadas.

Palavras-chaves: MIPS. Verilog. Processador. Quartus Prime.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

O rápido avanço da tecnologia na última década certamente elevou a importância da eletrônica digital em um patamar nunca visto antes. É fato que a última geração da humanidade possui, em todo nível social e econômico, podendo até ser de modo indireto, uma certa dependência ou até comodismo proporcionados pela eletrônica digital.

Os sistemas digitais, no mundo atual, se tornaram onipresentes em todos os setores inerentes à sociedade. Como exemplo de alguns desses setores, tem-se a economia, a agricultura, a industria, o entretenimento, a cultura, entre outros.

Segundo Tocci, 2011(1), um sistema digital é uma combinação de dispositivos projetados para manipular informação lógica no formato digital, isto é, informações que podem assumir valores discretos. Esses dispositivos podem ser eletrônicos, mecânicos, magnéticos ou até pneumáticos. Entre os sistemas digitais mais conhecidos, temos os computadores, as calculadoras, os equipamentos de áudio e vídeo e o sistema de telecomunicações.

Um dos componentes presentes na maioria dos sistemas digitais mais complexos é o processador, que poderia ser entendido como sendo o "cérebro"do sistema, local onde todas as operações e tomadas de decisões são realizadas.

Sendo assim, o profissional atuante em áreas que se relacionam com a eletrônica digital deve dominar com maestria uma variedade diversificada de conceitos teóricos e práticos sobre sistemas digitais, incluindo conceitos relacionados a processadores.

Um dos objetivos deste relatório, além do projeto de um processador funcional, foi expor esses conceitos de forma mais realista e prática, e sua realização tem um papel fundamental na formação profissional na área.

2 Objetivos

2.1 Geral

O projeto da disciplina Laboratório de Arquitetura e Organização de Computadores tem como objetivo principal a elaboração e o desenvolvimento de um processador funcional a partir de uma arquitetura customizada baseada no *MIPS*, que possua a capacidade de executar os algoritmos mais básicos da literatura de lógica de programação. A implementação ocorrerá por meio do uso da linguagem de descrição de hardware Verilog, no ambiente do software Quartus II. O processador será simulado no dispositivo FPGA (2) DE2-115 Cyclone IV.

2.2 Específico

A primeira etapa do projeto, como foi definido no Ponto de Checagem 1, teve como finalidade a escolha do conjunto de instruções e da arquitetura base do projeto, assim como todos os aspectos técnicos e conceituais a serem trabalhados no desenvolvimento do processador. Sendo assim, pontuando cada aspecto a ser trabalhado, tem-se:

- Escolha de um conjunto de instruções apropriado para executar algoritmos básicos.
- Definição de um conjunto de formatos de instrução que comporte todas as intruções escolhidas.
- Escolha dos modos de endereçamento com os quais o processador irá operar.
- Escolha de um conjunto de registradores a serem utilizados pelo processador durante as operações.
- Elaboração de um diagrama que agrupe todos os circuitos lógicos utilizados em blocos para ilustrar cada módulo utilizado no processador e como eles interagem entre si.
- Após a elaboração do diagrama, apresentar os sinais de controle escolhidos para a elaboração da unidade de controle do processador.

3 Fundamentação Teórica

Nesta seção, buscou-se discorrer sobre os elementos teóricos e conceituais necessários no desenvolvimento dos objetivos específicos, discutidos no capítulo anterior.

3.1 A Arquitetura MIPS

A principal arquitetura utilizada como modelo ou referência no desenvolvimento do projeto foi a arquitetura MIPS que significa Microprocessor without interlocked pipeline stages, ou Microprocessador sem estágios intertravados de pipeline. Se trata de um microprocessador RISC, conceito que será esclarecido posteriormente, cuja CPU utiliza apenas registradores para realizar operações lógicas e aritméticas.

3.2 Arquitetura base e conjunto de instruções.

Antes de definir o conjunto de instruções, a decisão de projeto inicial nessa implementação deve ser o tipo de armazenamento interno do processador.Segundo Patterson, 2013(3), as principais opções dessa parte da arquitetura seriam o armazenamento por pilha, por acumulador, ou por um conjunto de registradores. Vale ressaltar que nas duas primeiras opções, os operandos são nomeados de forma implicíta. Na primeira, o operando está implicitamente no topo da pilha, e na segunda, o operando é o acumulador. Como a escolha de um conjunto de registradores faz parte dos objetivos específicos, nota-se que a arquitetura deste projeto será do tipo registrador-registrador, correspondente a terceira opção. Nesta arquitetura, todos os operandos são explícitos, sendo registradores ou posições na memória.

Também conhecido como código de máquina(4), o conjunto de instruções comporta todas as operações, sejam elas complexas ou simples, que o processador pode vir a executar. Os conjuntos de instruções são comumente classificados como RISC (Reduced Instruction Set Computer) ou CISC (Complex Instruction Set Computer). As instruções escolhidas para compor o conjunto de instruções utilizado e suas particularidades serão esclarecidas no próximo capítulo.

Segundo Patterson(3), um conjunto de instruções do tipo CISC geralmente utiliza uma arquitetura do tipo Registrador-Memória, possui um número elevado de instruções distintas, complexas e mais específicas, e acessa os dados via memória.

Um conjunto de instruções do tipo RISC, que foi o utilizado nesse projeto, apresenta uma arquitetura do tipo registrador-registrador, com um número baixo de instruções

simples. A proposta desse tipo de conjunto é realizar tarefas mais complexas utilizando operações simples. O acesso aos dados ocorre por meio de registradores.

Vale ressaltar que uma das decisões de projeto sobre a arquitetura base tomadas nas primeiras etapas foi seguir o modelo de Arquitetura Havard, que consiste basicamente no uso de duas memórias, uma contém as instruções a serem executadas e outra contém os dados.

3.3 Formatos de instruções

No caso do processador a ser implementado, uma instrução conterá 32 bits. Um formato de instrução, de forma resumida, seria o modo com que essa sequência de bits será interpretada pelo processador. Geralmente, instruções que realizam operações semelhantes possuem o mesmo formato de instrução. No caso do projeto desenvolvido, foi decido que haverá o uso de 3 formatos de instruções diferentes, do mesmo modo que ocorre no modelo MIPS. Esses formatos e como eles funcionam serão discutidos no próximo capítulo.

3.4 Modos de Endereçamento

O modo de endereçamento denota o modo com que o processador acessa a memória, a partir da instrução executada no momento. Uma escolha de modos de endereçamento apropriada permite o maior uso de memória, como será perceptível no próximo capítulo, quando esse assunto for discutido.

3.5 Conjunto de Registradores

Um dos módulos fundamentais do processador desenvolvido é o banco de registradores, que consiste em um conjunto de registradores responsável por armazenar os dados com os quais o processador trabalha. Durante o funcionamento do processador, esse módulo conversa diretamente com a memória de dados e de instruções, para atualizar o conteúdo dos registradores afim de satisfazer as necessidades da instrução executada. Nesse projeto, os registradores utilizado neste módulo apresentam funções e utilidades particulares, que serão devidamente esclarecidas no próximo capítulo.

3.6 Sinais de Controle

O módulo mais fundamental no projeto do processador é a unidade de controle, responsável por, a partir da instrução executada no momento, controlar o acionamento e o modo de operação dos demais módulos. Para que isso ocorra, cada módulo é ligado

a unidade de controle por um barramento, que carregam os sinais de controle. Sendo assim, um sinal de controle é o modo com o qual a unidade de controle manuseia os outros módulos. Os sinais de controle escolhidos nesse projeto apresentam funcionalidade semelhante aos utilizados pelo modelo MIPS, com algumas ressalvas e adições, que serão devidamente esclarecidas no capítulo de desenvolvimento.

3.7 Linguagem de descrição de hardware e o Verilog

Uma HDL (Hardware Descriptive Language), ou, em português, linguagem de descrição de hardware, é usada exclusivamente para projetos de sistemas digitais. Sua principal particularidade em relação com linguagens de programação tradicionais é o paralelismo. Em geral, uma HDL possui diversas semelhanças com linguagens de programação, mas, como os circuitos digitais operam de forma paralela, eles não podem ser descritos por uma linguagem de programação tradicional, que opera de modo sequencial.

Entre as características mais relevantes da HDL, pode-se citar:

- A possibilidade de simplificação de circuitos digitais complexos através de poucas linhas de código;
- Reutilização de bibliotecas e projetos já implementados
- O código é independente do hardware. Sendo assim, existe a possibilidade de um mesmo código ser compatível com diferentes tipos de dispositivos;
- Possue um conjunto de regras de sintaxe bem semelhante ao utilizado por linguagens de programação tradicionais.

O Verilog é uma HDL usada para modelar sistemas eletrônicos a nível de circuito, semelhante com a proposta de toda HDL. Essa ferramenta, por sua vez, suporta projeção, verificação e implementação de projetos analógicos, digitais e híbridos em vários níveis de abstração. Com placas de desenvolvimento baseadas nos Circuitos Integrados específicos como, no caso deste trabalho, o FPGA, é possível descarregar o código gerado nessa linguagem para matrizes de portas lógicas combinacionais e sequenciais.

Esta linguagem difere das outras, em especial, pela maneira como é executada. Ela segue padrões diferentes das demais linguagens. Uma implementação em Verilog separa hierarquicamente os módulos de conexões e registradores. Sendo assim, é possível dividir o programa em processos sequenciais e paralelos, com circuitos combinacionais ou sequenciais. Processos sequenciais são executados dentro de blocos "begin/end". Os demais processos são executados de forma paralela. Os circuitos combinacionais são implementáveis através de atribuições do tipo bloqueante, e os circuitos sequenciais através de atribuições do tipo

não-bloqueante. Os aspectos de sintaxe dessa linguagem são irrelevantes no entendimento do processo adotado nesse projeto. O único ponto que vale ressaltar é a possilidade de uso de todas as operações lógicas e aritméticas.(5)

4 Desenvolvimento

É importante agrupar nesta seção, todas as características da arquitetura base deste projeto que foram citadas e explicadas no capítulo anterior. Sendo assim, a arquitetura possuirá um tamanho de instruções de 32 bits e uma memória mapeada em palavras de 32 bits. O processador será monociclo, ou seja, realizará uma instrução por ciclo de clock. O módulo de entrada e saída não foi descrito de modo profundo, pois será implementado e trabalhado nas próximas etapas do projeto.

4.1 Conjunto de instruções escolhido e suas particularidades

Lembrando que os conjunto de instruções adotado segue as características de um conjunto RISC, que ja foi conceituado anteriormente. Para expor o conjunto adotado, haverá a sitação de cada instrução escolhida junto com uma breve descrição de sua utilizade. Vale ressaltar que o processador desenvolvido utilizará o complemento de 2 para representar números com sinal e do modo little endian, donde os números são apresentados com o digito mais significativo a esquerda e o menos significativo a direita.

Instruções de Operações Aritméticas

- **add**: Soma básica entre 2 valores armazenados em registradores.
- **addi**: Soma básica entre um valor armazenado em um registrador e um imediato. Note que um imediato é um número represetado diretamente no código da instrução.
- **addu**: Soma básica sem sinal. Nesse tipo de operação, o complemento de 2 não é utilizado e os números são sempre positivos.
- **addiu**: Soma básica entre um valor armazenado em um registrador e um imediato, sem sinal.
- **sub**: Subtração básica de dois valores armazenados em registradores.
- **subi**: Substração básica de um valor armazenado em um registrador com um imediato.
- **subu**: Subtração básica de dois valores armazenados em registradores, sem sinal.
- **mult**: Multiplicação de dois valores armazenados em registradores. Os 32 bits mais significativos do resultados são armazenados no registrador HI e os 32 menos significativos nos registrador LO.

- **multu**: Multiplicação que ocorre do mesmo modo da operação anterior, porém sem sinal.
- **div**: Divisão inteira entre dois valores armazenados em registradores. O resultado da divisão é armazenado no registrador LO e o resto no HI.
- **divu**: Divisão que ocorre do mesmo modo da citada no item anterior, porém sem sinal.

Instruções de Transferência de Informação:

- **lw**: Carrega uma palavra de 32 bits da memória para um registrador do banco de registradores.
- **sw**: Salva uma palavra de 32 bits armazenada em um registrador na memória.
- **li**: Carrega uma palavra representada como um imediato em um registrador do banco de registradores.

Instruções de Deslocamento de bits:

- **sra**: Desloca de modo aritmético um número para a direita.
- **sla**: Desloca de modo aritmético um número para a esquerda.

Vale ressaltar que o deslocamento aritmético, diferente do deslocamento lógico, sempre multiplica por 2 (deslocando para a direita) ou divide por 2 (deslocando para a esquerda) um número binário, mesmo quando o número é representado em complemento de 2. No caso do deslocamento lógico, isso só funciona para números sem sinal.

Instruções de Operações Lógicas e de Comparação:

- **and**: Realiza uma operação de and lógica entre valores armazenados em 2 registradores.
- **andi**: Realiza uma operação de and lógica entre um valor imediato e um valor armazenado em um registrador.
- **or**: Realiza uma operação de or lógica entre valores armazenados em 2 registradores.
- **ori**: Realiza uma operação de or lógica entre um valor imediato e um valor armazenado em um registrador.
- **xor**: Realiza uma operação xor entre valores armazenados em 2 registradores.

- **xori**: Realiza uma operação xor com um valor armazenado em um registrador e um imediato.
- **nor**: Realiza uma operação nor lógica entre valores armazenados em 2 registradores.
- **slt**: Compara o valor de 2 registradores através de uma subtração. Se o valor do primeiro registrador for menor do que o do segundo, um regitrador especificado pelo código da instrução recebe o valor 1.
- **slti**: Compara o valor de um registrador com um imediato. Se o valor do registrador for menor do que o imediato, um registrador especificado pelo código da instrução recebe o valor 1.
- **sltu**: Realiza a mesma operação da instrução slt, porém com números sem sinal.
- **sltiu**: Realiza a mesma operação da instrução slti, porém com números sem sinal.
- **not**: Realiza uma operação de negação com o valor de um registrador.

Instruções de Salto e desvio:

- **j**: Operação que realiza um salto do endereço da instrução atual para um outro endereço da memória de instruções, determinado pelo código da instrução. Desse modo, há a possibilidade de "pular"um uma determinada quantidade de instruções, que acabam deixando de serem executadas. O modo como isso ocorre será explicado quando o diagrama em blocos do processador for apresentado.
- **jr**: Operação que realiza um salto do endereço da instrução atual para o endereço armazenado no registrador $ra.
- **beq**: Compara o valor de 2 registradores. Se forem iguais, um desvio é realizado para o endereço especificado no código da instrução. Um desvio pode ser entendido como um salto que ocorre condicionalmente.
- **bne**: Compara o valor de 2 registradores e realiza um desvio para um endereço especificado pelo código da instrução se esses dois valores forem diferentes.
- **jal**: Realiza a mesma operação da instrução j, porém salva o valor do endereço atual no registrador $ra. Essa instrução é utilizada em conjunto com a instrução jr para possibilitar a criação de funções na memória de instruções.

Vale ressaltar que, com o desenvolvimento do projeto, existe a possibilidade da inclusão de novas instruções no conjunto trabalhado, se for necessário, do mesmo modo que há a possibilidade da exclusão de instruções já existentes. Se isso acontecer, todo o processo e motivação será apropriadamente esclarecido nos próximos relatórios.

4.2 Formatos de instruções escolhidos

Os formatos de instruções escolhidos neste projeto são os mesmos dos utilizados pelo MIPS. As tabelas 1, 2 e 3 apresentam como esses três formatos escolhidos estão organizados. Vale ressaltar que o modo com que as tabelas apresentam os campos e a respectiva quantidade de bits reservada a esses campos também segue a ordem little endian. Desse modo, os 6 bits mais significativos da instrução representam o Opcode, que aparece mais à direita na tabela, e assim por diante. É importante ressaltar também a função de cada um dos campos utilizados para rotular cada parcela do código de instrução.

- Os campos **Opcode** e **funct** denotam os bits utilizados pelo processador com a função de identificar a instrução a ser executada.
- Os campos **R1** e **R2** armazenam os endereços para os registradores a serem acessados no banco de registradores que possuem os operandos da instrução.
- O campo **Rd** denota o endereço do registrador de destino, o registrador do banco de registradores que receberá o resultado da operação executada pelo processador.
- O campo **shamt** É utilizado exclusivamente para as instruções de deslocamento e armazena o número de deslocamentos a ser realizado pela instrução.
- O campo **Imediato**, exclusivo para instruções do tipo I, é utilizado para armazenar um número ou endereço diretamente no código da instrução, que será utilizado de modo direto ou imediato na execução da instrução.
- O campo **Endereço**, exclusivo para instruções do tipo J, é utilizado para armazenar diretamente no código da instrução, o endereço de destino do salto a ser realizado.

Tabela 1 – Instruções do tipo R

Opcode	R1	R2	Rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Tabela 2 – Instruções do tipo I

Opcode	R1	Rd	Imediato
6 bits	5 bits	5 bits	16 bits

Tabela 3 – Instruções do tipo J

Opcode	Endereço
6 bits	26 bits

4.3 Modos de endereçamento escolhidos

Como já dito no caítulo anterios, os modos de endereçamentos são os modos com os quais o processador utiliza o código de instrução para acessar os dados necessários nas operações.

Os modos de endereçamento escolhidos, baseados no modelo MIPS, foram 5:

- **Imediato**, onde o operando é uma constante. Será utilizado para instruções do tipo I de operações lógicas e aritméticas. Uma instrução que utilizará esse método de endereçamento é o **addi**.
- **Por Registrador**, onde as constantes da instrução carregam um endereço para um registrador. Será utilizado para todas as instruções do tipo R, como por exemplo, **add**.
- **Por deslocamento**, onde o operando é um endereço de memória estabelecido a partir da soma de um registrador e uma constante da instrução. Será utilizado para instruções de transferência de informação do tipo I, como por exemplo, **lw**.
- **Relativo a PC**, usado para saltos do tipo I e do tipo J, é semelhante ao endereçamento por deslocamento, mas o registrador utilizado é o PC. O PC é um registrador que possui uma função especial, que será descrito na seção de registradores utilizados. Um exemplo de instrução que usa esse modo de endereçamento é o **beq**.
- **Pseudo Direto**, usado para instruções do tipo J, o endereço do salto é deslocado 2 bits à esquerda e concatenado com os 4 bits mais significativos do PC. Esse é o modo com o qual o endereço de um salto é calculado. Instruções que utilizam esse modo são **j, jr e jal**.

Após a descrição da escolha de todos esses elementos para o desenvolvimento do processador, será possível visualizar, no diagrama em bloco a ser apresentado em alguma das próximas seções, como que cada um desses elementos compõem o processador.

4.4 Sinais de controle utilizados

Como já explicado na seção anterior, os sinais de controle possuem o propósito de controlar o funcionamento de cada módulo do processador, de acordo com a instrução executada. No caso do projeto desenvolvido, os sinais de controle escolhidos diferem do MIPS em alguns aspectos, com a ideia de possibilitar a implementação de algumas instruções adicionais. Vale ressaltar novamente que eles serão melhor compreendidos após a exibição do diagrama em bloco do processador. Os sinais de controle escolhidos foram:

- **RegDst**: Se trata de um sinal de 2 bits que opera de 3 modos diferentes: se os 2 bits estão em 0, a entrada "Registrador para Escrita", do banco de registradores, que poderá ser visualizada no diagrama em bloco da próxima seção, recebe o campo R2 do código de instrução. Se o primeiro bit está em 0 e o segundo em 1, A mesma entrada recebe o campo R1 do código de instrução. Se a primeira entrada está em 1 e a segunda em 0, a entrada citada anteriormente recebe o endereço do registrador $ra, exclusivo para uso da instrução **jal**.

- **EscreveReg**: Quando inativo, não faz nada. Quando ativo, o registrador apontado pelo endereço contido no registrador "Registrador para escrita"é escrito com o valor de "Dados para escrita".

- **OrigALU**: Quanto inativo, o segundo operando da ALU vem da segunda saída do banco de registradores. Quando ativo, o segundo operando da ALU consiste nos 16 bits menos significativos da instrução, que correspondem ao campo "Imediato", com sinal extendido.

- **OrigPC**: Quando inativo, o PC é substituído pela saída do somador, que calcula o valor da próxima instrução. Quando ativo, o PC é substituído pela saída do somador que calcula o desvio.

- **LeMem**: Quanto inativo, não faz nada. Quando ativo, o conteúdo da memória de dados designado pela entrada "Endereço"é colocado na saída de "Dados da leitura".

- **EscreveMem**: Quando inativo, não faz nada. Quando ativo, o conteúdo da memória de dados designado pela entrada "Endereço"é substituído pelo valor na entrada "Dados para escrita".

- **MemparaReg**: Quando inativo, o valor enviado para a entrada "Dados para a escrita"do banco de registradores vem da ALU. Quando ativo, o valor enviado para a entrada de memória vem da memória de dados.

- **OpALU**: Assim como o primeiro sinal apresentado, também possui 2 bits e 3 modos de operação. Com os dois sinais em 0, para instruções de transferência de informação, a ALU realiza uma adição. Se o primeiro bit assumir o valor 0 e o segundo assumir 1, a ALU realiza uma subtração, para instruções de comparação. Se o primeiro bit assumir o valor 1 e o segundo assumir 0, a operação da ALU será determinada pela Unidade de controle da ALU, outro módulo presente no processador que possui a finalidade de determinar a operação a ser realizada pela ALU a partir do campo funct do código de instrução.

- **WriteRa**:Quando inativo, o campo "Dados para escrita"do banco de registradores recebe o resultado da ALU. Quando ativo, esse mesmo campo recebe o endereço atual do PC. Esse sinal é especifico para instruções **jal**.

- **BifNot**: Ativo quando uma instrução **bne** é executada, sinalizando que, caso a condição do desvio seja satisfeita, o PC irá receber o endereço calculado.
- **Branch**: Análogo ao sinal anterior, porém para a instrução **beq**.
- **Jump**: Quando inativo, o registrador PC recebe o endereço da próxima instrução ou o endereço do desvio condicional. Quando ativo, o registrados PC recebe o endereço de desvio incondicional de uma instrução do tipo. Esse sinal sempre é ativo quando uma instrução de salto incondicional é realizada **J**.
- **JumpR**: Esse sinal é sempre ativo quando uma instrução **jr** é executada. Quano inativo, caso uma instrução de salto incondicional seja realizada, ela acontece utilizando os 26 bits menos significativos da instrução. Quando ativo, caso esse tipo de instrução seja realizado, utiliza-se os 26 bits menos significativos do registrador **$ra**.

4.5 Conjunto de registradores escolhidos

Para este projeto, optou-se por utilizar 32 registradores, com funções e particularidades pré definidas. Entre os registradores utilizados, tem-se:

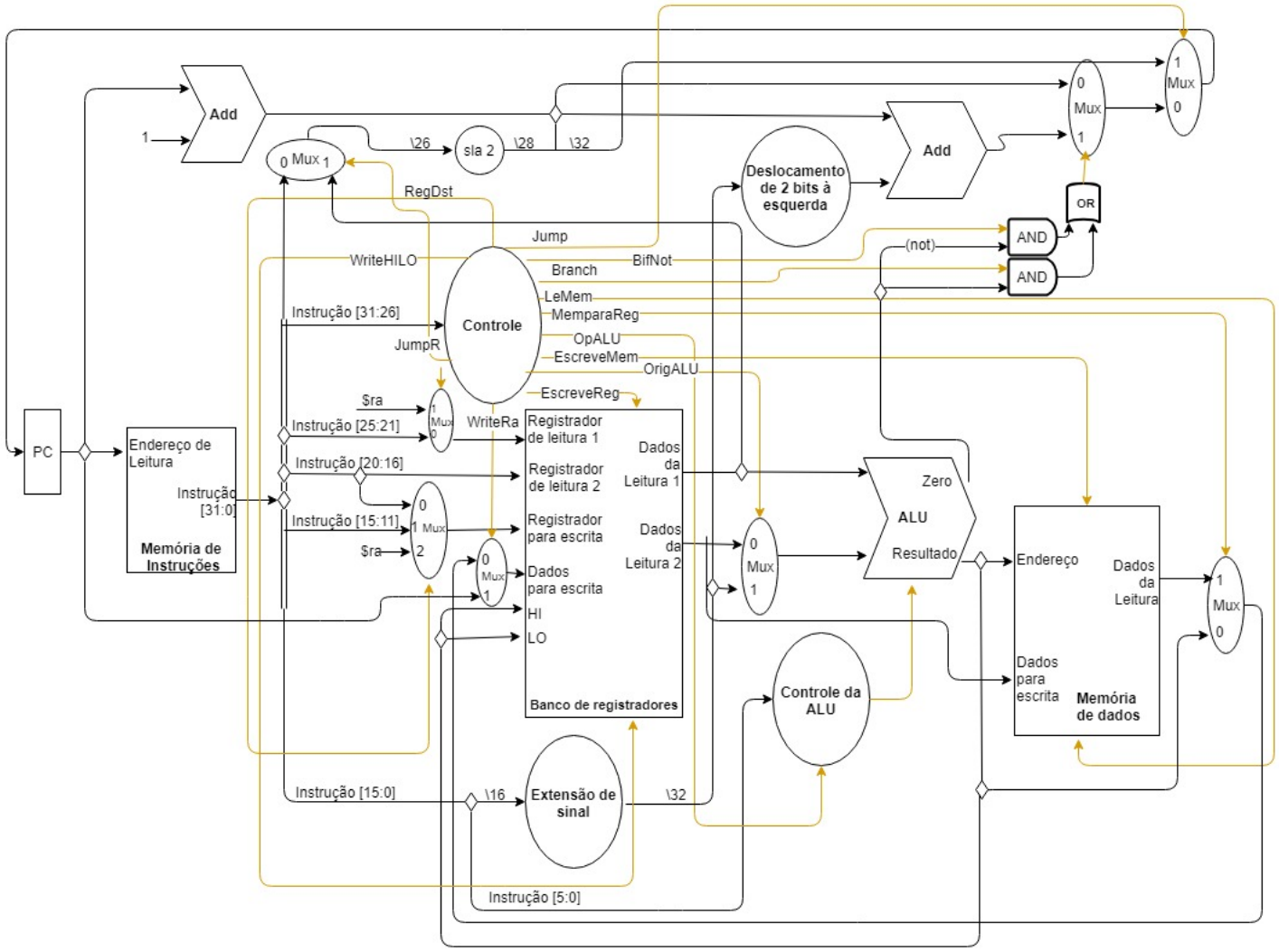
- Os registradores High (HI) e Low(LO), utilizados para armazenar a parte mais significativa e a menos significativa de uma multiplicação, ou o resto e o quociente de uma divisão, respectivamente.
- 10 registradores temporários.
- 10 registradores de uso geral.
- 5 registradores que armazenam argumentos para funções.
- 2 registradores que armazenam resultados das funções
- Um registrador de endereço ($ra), utilizado para armazenar o endereço de retorno da instrução **jal**.
- Um registrador de pilha, que armazena o endereço do topo de uma pilha na memória.
- Um registrador para armazerar o endereço da próxima instrução a ser executada. Este registrador não está no banco de registradores. No diagrama em bloco, ele é representado pelo módulo com o rótulo "PC"(Program Counter).

É importante dizer que posteriormente existe a possibilidade de incluir registradores "invisíveis"ao usuário com funções específicas, para atender necessidades do sistema operacional.

4.6 Diagrama em bloco do processador

A seguir, tem-se o caminho de dados ou diagrama em bloco do processador a ser projetado a partir de todas as escolhas e decisões apresentadas nas seções anteriores:

Figura 1 – Última versão do caminho de dados desenvolvido, desenhado através do uso da ferramenta do site draw.io



O diagrama foi desenvolvido no menor espaço possível, para possbilitar a visualização neste relatório, mas mesmo assim, a ilustração acabou por ocupar muito espaço.

O nome dos blocos passa uma ideia clara de suas funcionalidades, fato que facilita o entendimento do diagrama. O único caso em que essa intuição pode vir a não ocorrer é no bloco denotado por "sla 2"que realiza a mesma operação do bloco denotado por "Deslocamento de 2 bits à esquerda". Esse rótulo estranho foi atribuído com a finalidade logística de reduzir o espaço ocupado.

Como o caminho de dados ficou bem denso, haverá uma breve explicação para facilitar o entendimento. Uma particularidade desta ilustração é o uso de losangos pequenos

em cima dos fios para indicar uma bifurcação, quando é necessário enviar a mesma informação para dois ou mais módulos distintos. As setas coloridas representam os sinais de controle.

Vale ressaltar que, como a memória vai ser mapeada em palavras de 32 bits, diferente do que ocorre no MIPS, a cada instrução o PC será incrementado de modo unitário.

As instruções do formato **J** ocorrem da seguinte forma: Primeiramente, a Memória de Instruções recebe do registrador PC o Endereço de leitura da instrução a ser executada e a envia para um barramento indicado pelos traços duplos, que contém a instrução toda. Logo depois disso, os 26 bits menos significativos são enviados ao único multiplexador vertical da ilustração, que possui a função de distinguir se a operação de salto se trata de uma instrução "j"ou "jr". Logo após isso, o sinal é deslocado de 2 bits para à esquerda, pelo módulo "sla 2"e depois é concatenado com os 4 bits mais significativos do PC, para formar o endereço do desvio, e, assim, o valor obtido é enviado para o PC.

Para explicar como ocorre uma instrução do tipo **I**, vamos utilizar a instrução "beq". Assim como nas intruções do formato anterior e nas demais, em um primeiro momento, a memória de instruções passa para o barramento denotado pelas barras duplas a instrução a ser executada. No caso de uma instrução "beq", o banco de registradores recebe o endereço dos 2 registradores a serem comparados envia seus respectivos valores para a ALU. Enquanto isso, O campo "Imediato"da instrução é enviado para a unidade de "Extensão de sinal"na parte inferior do diagrama, que possui a finalidade de extender o sinal de 16 bits para 32, afim de possibilitar a soma com o endereço atual do PC, e enviar o resultado para o PC novamente, caso o desvio seja tomado. Em um terceiro momento, a ALU realiza a subtração dos operandos que recebeu e, caso a subtração resulte em 0, significando que os valores são iguais, o sinal de saída "Zero"da ALU é enfiado para região superior direita da ilustração, que possui uma lógica que faz com que o desvio seje tomado caso o sinal de saída apropriado seja enviado.

Uma instrução do tipo **R** ocorre da seguinte forma: Após a leitura e a passagem da instrução para o barramento de barras duplas, a unidade de controle faz com que os campos "Registrador de leitura 1"receba o campo "R1"da instrução, "Registrador de leitura 2"receba o campo "R2"da instrução e o campo "Registrador para escrita"receba o campo "Rd"da instrução. Posteriormente, os valores de "R1"e "R2"são enviados para a ALU, que realiza a operação e envia o resultado para o campo "Dados para escrita"que acabam por serem escritos no registrador apontado pelo campo "Rd", presente em "Registrador para escrita"

5 Considerações Finais

O ponto crucial dessa primeira etapa no desenvolvimento do projeto foi a tomada de decisão sobre os aspectos e características da arquitetura base. Antes da ideia de se basear na arquitetura MIPS, houve um breve estudo sobre a arquitetura ARM, especificamente da implementação que recebeu o nome de Amber, baseada no ARMv2, umas das versões mais antigas e simples desta arquitetura.

Em um segundo momento, percebeu-se que a ideia de implementar um processador baseado na arquitetura ARM ia se tornar muito complicada e trabalhosa, e assim, optou-se pela arquitetura MIPS, que já foi estudada em disciplinas anteriores.

Depois dessa primeira decisão foram definidos os aspectos da arquitetura desenvolvida, utilizando as características do MIPS, e , depois disso, todos os aspectos técnicos foram definidos:

- O conjunto de instruções foi definido.
- Os modos de endereçamento foram escolhidos.
- Os formatos de instruções foram definidos.
- Os registradores e suas funções contidos no banco de registradores foram definidos.
- Através do uso de todas essas características, um primeiro caminho de dados foi desenvolvido.

Pode-se afirmar que o produto desta primeira etapa do projeto foi o caminho de dados, que será utilizado como "planta"do processador, durante a implementação. Pelo fato da primeira etapa neste projeto se tratar de tomadas de decisão e revisão de conceitos teóricos, não houveram maiores problemas ou dificuldades.

Referências

1 TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas Digitais: Princípios e Aplicações*. 11th edition. ed. São Paulo: Pearson Education, 2011. Citado na página 9.

2 TERASIC TECHNOLOGIES. *DE2-115 User Manual*. Hsinchu, Taiwan, 2013. Citado na página 11.

3 HENNESSY, J. L.; PATTERSON, D. A. *Arquitetura de Computadores: Uma Abordagem Quantitativa*. 3th edition. ed. Rio de Janeiro, RJ: Campus, 2003. Citado na página 13.

4 PATTERSON, D. A.; HENNESSY, J. L. *Organização e Projeto de Computadores: a interface hardware/software*. 5th edition. ed. Rio de Janeiro, RJ: Campus, 2005. Citado na página 13.

5 CRAIBAS, J. J. S. *Dicas de implementação de circuitos digitais em verilog através do software Quartus Prime*. São Paulo. Apostila disponibilizada pelo docente Prof. Dr. Tiago de Oliveira, na disciplina Laboratório de Arquitetura e Organização de Computadores. Citado na página 16.