103962

PC1:
Conjunto de Instruções e Arquitetura Base
(Laboratório de sistemas computacionais: Arquitetura e organização de computadores)

São José dos Campos - Brasil

Abril de 2018

103962

PC1:

Conjunto de Instruções e Arquitetura Base

(Laboratório de sistemas computacionais: Arquitetura e organização de computadores)

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 2018

Resumo

O relatório em questão apresenta detalhes a respeito da implementação e desenvolvimento de um sistema computacional com arquitetura baseada na MIPS monociclo. Durante este relatório será apresentado detalhadamente informações à respeito do conjunto e formato de instruções, modos de endereçamento e arquitetura-base, abordando os principais conceitos e componentes.

Palavras-chaves: MIPS, unidade de controle, arquitetura.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Com o avanço tecnológico, a utilização de sistemas computacionais em nosso dia-a-dia se tornou cada vez mais presente e relevante. E entender o funcionamento desta tecnologia é fundamental. Um computador é uma máquina capaz de realizar variados tipos de tratamentos de informações ou processamento de dados, possuindo inúmeros atributos como armazenamento e processamento de dados, operações lógicas e aritméticas, tratamento de imagens gráficas, etc...

Além disso, é composto por diversos componentes que possuem funções distintas e complexas, que quando são combinadas funcionam em perfeita harmonia. Um exemplo disso é a CPU (unidade central de processamento) que em termos didáticos poderia ser considerada o cérebro deste sistema.

Para garantir a realização correta de todas as suas funções, um computador necessita de um conjunto de instruções (código de máquina compreendido pela CPU, que atua como interface entre hardware e software), também conhecido como ISA (*Instruction Set Architecture*). Este conjunto é tratado pela Unidade de Processamento que garante todos os requisitos necessários para a execução de um programa, sejam elas operações lógicas e aritméticas, acesso à memória, entrada e saída de dados, etc. Outra unidade fundamental para o funcionamento do computador é a Unidade de Controle, que será mostrada mais adiante.

Este é apenas um resumo que mostra quão importante e complexo pode ser um computador, e nas próximas páginas iremos conhecer em detalhes o funcionamento e desenvolvimento de alguns destes componentes.

2 Objetivos

2.1 Geral

O objetivo geral deste projeto é o desenvolvimento de um sistema computacional que opere de forma similar a um processador baseado na arquitetura MIPS, de forma que suas instruções sejam testadas e seu funcionamento seja comprovado através de simulação em elementos de *hardware*.

2.2 Específico

O objetivo específico é a especificação do conjunto e formato de instruções, modos de endereçamento e na arquitetura base por qual funcionará o Caminho de Dados (*Datapath*).

3 Fundamentação Teórica

3.1 Sistema Computacional

Um sistema computacional consiste num conjunto de dispositivos eletrônicos (*hardware*) capazes de processar informações de acordo com um programa (software).

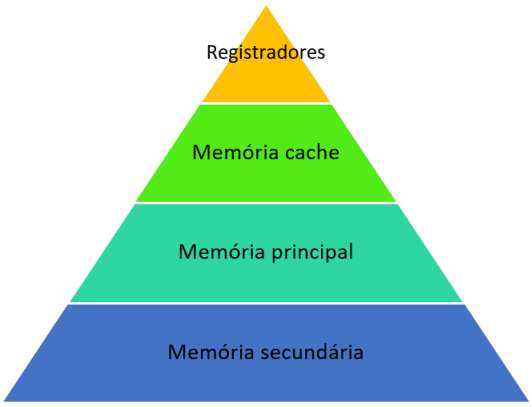
Um computador é composto de dispositivos de entrada e saída (como teclado, mouse, monitor, etc...), memória (que armazena todos os dados) e processador. Onde os dados que chegam através dos dispositivos de entrada são salvos na memória, para que após seu processamento sejam mostrados através do dispositivo de saída.

3.2 Hierarquia de memória

Dada a complexidade de um sistema computacional e suas operações realizadas é simples notar a necessidade da existência de diferentes níveis de armazenamento de dados, este conceito de criação é conhecido como hierarquia de memória.

Basicamente, a memória de um computador pode ser divida em quatro níveis principais, são eles: registradores, memória CACHE, memória principal e memória secundária. A figura abaixo representa a ordenação desses níveis hierárquicos, de forma que quanto mais no topo estiver, maior será o custo e velocidade de acesso e menor será sua capacidade de armazenamento .

Figura 1 – Hierarquia de Memória



Fonte: MaxiEduca (1)

De baixo para cima, a **memória secundária** é composta por dispositivos não voláteis como memórias externas de armazenamento em massa - por exemplo, disco rígido,

DVD, disquete, fita magnética, pen drive, entre outros. Sua principal característica é o armazenamento de dados que precisam se buscados antes de acessados, pois possuem baixa velocidade de acesso.

Subindo um nível, a **memória principal** possui uma alta velocidade de acesso, através da utilização da memória RAM (*Random Access Memory*). Que além de permitir leitura e escrita, mantém armazenado os programas operacionais básicos. A **memória CACHE** serve como ponte entre a troca de dados dos registradores e memória RAM. É uma memória de acesso rápido e otimiza os blocos de memória que são pertinentes ao processo em execução.

Finalizando, os **registradores** são os meios mais rápidos e computacionalmente caros de armazenamento de dados. Usualmente utilizados de forma temporária, registradores são as unidades de memória diretamente utilizadas pelas instruções atualmente sendo executadas pelo sistema.

3.3 Arquitetura Computacional

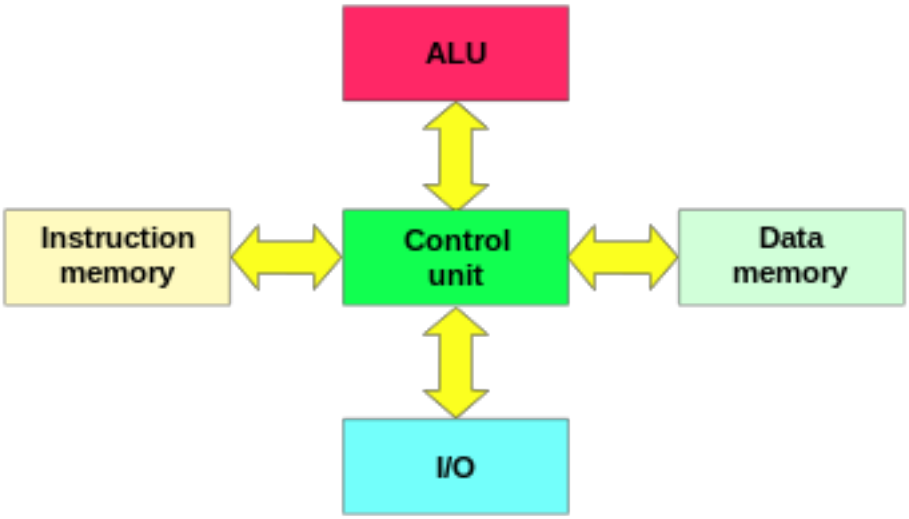
Para a construção de um processador é necessário definir quais instruções este deverá executar, definindo desta forma sua arquitetura. A arquitetura organiza a estrutura computacional de forma a otimizar seu funcionamento, a partir da combinação de diversas instruções e visando a execução de alguma função específica.

3.3.1 Arquitetura Havard

A Arquitetura de Harvard é uma arquitetura de computador que se distingue das outras por possuir separadamente circuito para sinais e armazenamento para, dados e instruções, que são independentes em termos de barramento e ligação ao processador (2).

Esta arquitetura se baseia no conceito da Arquitetura de Von Neumann e é composta por unidade de controle, memória de instrução, memória de dados, unidade lógica e aritmética e módulos de entrada e saída. A diferença entre a arquitetura de Von Neumann e a Harvard é que a última separa o armazenamento e o comportamento das instruções do CPU e os dados, enquanto a anterior utiliza o mesmo espaço de memória para ambos (2).

Figura 2 – Arquitetura Harvard



Fonte: Diego Macedo (2)

3.3.2 Arquitetura RISC

Ao contrário da CISC, a arquitetura RISC (*Reduced Instruction Set Computers*) possui um número reduzido e simplificado de instruções, podendo operar a velocidades maiores de clock.

Desta forma, este tipo de arquitetura possui uma ênfase maior em *software* e requer um trabalho maior do programador, uma vez que exige mais linhas de código. Suas principais características são:

1.Número reduzido de ciclos de clock por instrução;

2.Utiliza um formato fixo de instrução;

3.Conjunto reduzido de instruções.

Processadores baseados nesta arquitetura são mais simples e muito mais baratos, possuindo um menor número de circuitos internos, como por exemplo os processadores Alpha.

Apesar das diversas peculiaridades e diferenças destas arquiteturas, atualmente muitos modelos de processadores abrigam características de ambas. E as grandes fabricantes utilizam desta combinação visando melhorar o desempenho durante a execução das instruções.

3.4 MIPS

Vimos que a arquitetura de um processador é a forma como ele se comporta funcionalmente. Com o passar dos anos e do desenvolvimento de novas tecnologias, diversas arquiteturas foram criadas, dentre elas a arquitetura MIPS.

Desenvolvida nos anos 80 por pesquisadores da Universidade de Stanford, esta arquitetura segue o padrão de arquiteturas RISC. Utilizando de um número reduzido de registradores e um conjunto de instruções menor.

Diversos conjuntos de MIPS foram implementados utilizando diferentes números de registradores, contudo, os números principais são os de 32 bits (número que será utilizado neste projeto) e o de 64 bits.

Outra característica interessante é que a execução desta arquitetura é feita em cinco estágios, são estes: busca, decodificação, execução, acesso à memória e escrita de dados.

3.4.1 Tipos de Instruções

A arquitetura MIPS realiza diversos tipos de operações, de forma que seus tipos de instruções são divididos em três partes: R, I e J.

É no **tipo R** que todas as instruções de operação lógica e aritmética com dados dos registradores se encontram, como por exemplo soma e subtração. Ainda há subdivisões neste tipo de instrução chamados: opcode, RS, RT, RD, *shamt* e *funct*.

Tabela 1 – Instruções de formato R

Campo	*Opcode*	RS	RT	RD	*shamt*	*funct*
Tamanho (bits)	6	5	5	5	5	6
Bits	31 - 26	25 - 21	20 - 16	15 - 11	10 - 6	5-0
Função	Funciona como um identificador de instruções	Representam os endereços dos bancos de registradores que os dados serão retirados	Representam os endereços dos bancos de registradores que os dados serão retirados	Endereço do registrador que receberá o resultado da operação	Quantidade de bits que serão deslocados (se for necessário)	Diferencia instruções na ULA, funciona como uma extensão do opcode

Fonte: Autor

O **tipo I**, possui instruções que realizam operações lógicas e aritméticas com dados de *offset* a partir do campo imediato, ou seja, executam acesso à memória e executam saltos condicionais. A tabela abaixo ilustra quais são os campos deste tipo de instrução.

Tabela 2 – Instruções de formato I

Campo	*Opcode*	RS	RT	*offset*
Tamanho (bits)	6	5	5	11
Bits	31 - 26	25 - 21	20 - 16	15 - 0
Função	Funciona como um identificador de instruções	Representam os endereços dos bancos de registradores que os dados serão retirados	Endereço do registrador que receberá o resultado da operação	O campo offset ou imediato, possui um valor codificado que será usado para possíveis saltos condicionais ou operações aritméticas

Fonte: Autor

Finalizando, o terceiro tipo de instrução é o **tipo J**, responsável pelas instruções de saltos. Por esta característica, possui uma divisão mais simples.

Tabela 3 – Instruções de formato J

Campo	*Opcode*	*adress*
Tamanho (bits)	6	26
Bits	31 - 26	25 - 0
Função	Funciona como um identificador de instruções	Endereço no qual o salto será destinado

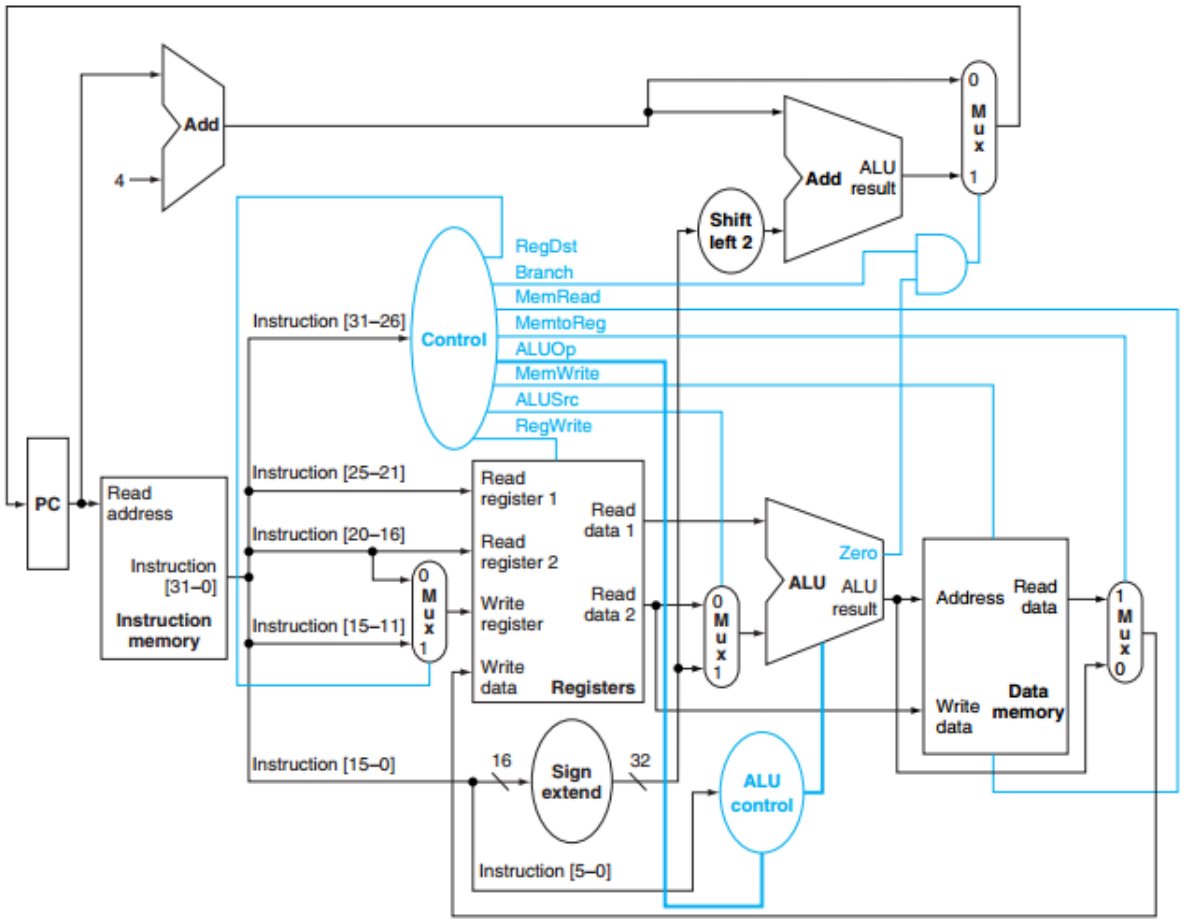
Fonte: Autor

3.4.2 Datapath

Como o nome sugere, o *Datapath* ou caminho de dados, é o caminho tomado pelos bits processados em uma instrução, representando o total de componentes presentes na arquitetura.

Quando estes componentes interligados recebem a ação de uma sinal de controle (*flag*), direcionam a operação através das diferentes partes do circuito com o intuito de obter um resultado específico. A figura 2, mostra o caminho de dados da arquitetura MIPS.

Figura 3 – MIPS Monociclo



Fonte: Acervo Livro (3)

4 Desenvolvimento

Para este projeto, o modelo MIPS foi escolhido pois é baseado na arquitetura RISC, portante é mais simplificado e fácil de implementar. Além disso, é um modelo didaticamente viável e possui uma vasta literatura para fins de estudo.

4.1 Conjunto de instruções

A arquitetura MIPS diversos tipos de instruções com características que variam de acordo com o tipo de implementação. Em um MIPS de 32 bits, pode-se trabalhar com 32 registradores, acessados por um intervalo de 5 bits.

Para este projeto, utilizaremos um conjunto semelhante e reduzido ao MIPS. Consistindo por 26 instruções, que visam abranger todas as operações básicas do processador, como operações lógicas e aritméticas, saltos, acesso à memória, etc...

4.1.1 Conjunto de Instruções

Para este projeto, será utilizado quatro modos de endereçamentos utilizados na arquitetura MIPS.

Endereçamento imediato: considerado o modo mais simples de endereçamento, o endereçamento imediato possui o operando contido no campo de endereço, portanto, não necessita acesso à memória para buscar o operando.

Endereçamento por registrador: este modo referencia o respectivo registrador no campo de endereço e nele já contém o operando, sem fazer acesso à memória. É utilizado por instruções como sub, add, or, etc.

Endereçamento por deslocamento: no endereçamento por deslocamento ou de base, o registrador contém um endereço base e um campo de endereço é um deslocamento.

Endereçamento relativo ao PC: neste modo, a próxima instrução a ser executada é relativa ao endereço da instrução atual. Desta forma, seu endereço é calculado através da soma entre uma constante da instrução e o PC. Geralmente é utilizada por instruções de desvio condicional, como a beq.

Endereçamento pseudodireto: utilizado por instruções de salto incondicional, no endereçamento pseudodireto, o endereço é formado pela concatenação dos 26 bits da instrução com os bits mais altos do PC.

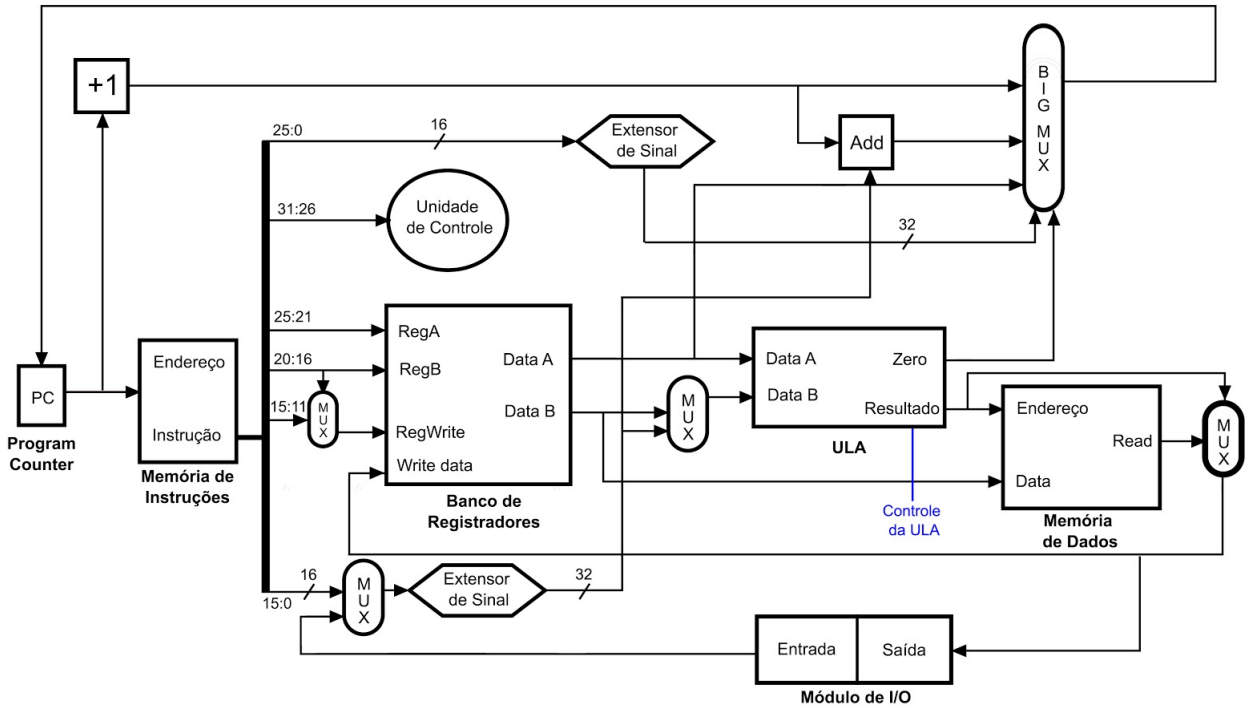
Tabela 4 – Conjunto de Instruções

Categoria	**Instrução**	**Tipo**	**tipo de instrução**
Aritmética	**adição**	**add**	**R**
Aritmética	**subtração**	**sub**	**R**
Aritmética	**adição com imediato**	**addi**	**R**
Aritmética	**subtração com imediato**	**subi**	**R**
Aritmética	**multiplicação**	**mult**	**R**
Lógicas	**NOT**	**not**	**R**
Lógicas	**AND**	**and**	**R**
Lógicas	**OR**	**or**	**R**
Lógicas	**XOR**	**xor**	**R**
Lógicas	**set less than**	**stl**	**R**
Lógicas	**deslocamento à esquerda**	**shfl**	**R**
Lógicas	**deslocamento à direita**	**shfr**	**R**
Memória	**load word**	**lw**	**I**
Memória	**load imediato**	**li**	**I**
Memória	**store worf**	**sw**	**I**
Saltos	**branch and equal**	**beq**	**I**
Saltos	**branch and not equal**	**bne**	**I**
Saltos	**branch and equal zero**	**beqz**	**I**
Saltos	**jump**	**jump**	**J**
Saltos	**jump to register**	**jumpr**	**J**
Outros	**NOP**	**nop**	**R**
Outros	**Input**	**in**	**Outros**
Outros	**Output**	**out**	**Outros**
Outros	**HLT**	**hlt**	**R**

Fonte: Autor

4.2 Arquitetura-base do Processador

Figura 4 – Arquitetura Base



Fonte: Autor

O caminho de dados deste projeto é semelhante ao da arquitetura MIPS monociclo, porém com um grau de complexidade menor. Nele, estão presentes alguns componentes importantes para o funcionamento do processador, que serão explicados detalhadamente.

4.2.1 Contador de Programa

O **Contador de Programa**, ou *Program Counter* (PC), é o primeiro elemento do caminho de dados. Nele é armazenado o endereço atual da instrução em execução através de um registrador, que sempre é atualizado a cada subida do *clock*.

Além disso, em determinadas *flags*, ou seja, sinais de controle, o endereço é incrementado ou substituído por determinados valores, como por exemplo em instruções condicionais ou de salto.

4.2.2 Memória de Instruções

A **Memória de Instruções** (*Instruction Memory*) é responsável pelo armazenamento de todas as instruções realizadas pelo processador, em posições contíguas de memória.

Estas instruções são enviadas para outros componentes referenciados pelo endereço contido no PC, que após isso é atualizado na próxima subida de *clock* e busca a instrução armazenada no endereço seguinte.

4.2.3 Banco de Registradores

O Banco de Registradores é o componente em que são armazenados dados temporários pertinentes à execução das instruções, sendo o tipo de memória situada no topo da hierarquia (4).

Neste componente, existem 32 registradores que são constantemente alterados com base nos endereços de entrada das instruções. E seu armazenamento é realizado com base em um sinal, que indica quando os dados provenientes do final do circuito devem ou não ser escritos nestes endereços.

4.2.4 Unidade Lógica e Aritmética

O componente responsável por realizar cálculos e deslocamentos dos dados de entrada é a **Unidade Lógica e Aritmética** (ULA) ou ALU (*Arithmetic Logical Unity*). Pode ser utilizada para diversos propósitos, verificar igualdade entre dados para determinando saltos condicionais, calcular um endereço de escrita ou ainda realizar operações lógicas e aritméticas.

Seu funcionamento não depende do *clock*, como outros componentes, mas de variações de seus sinais de entrada. Devido ao fato de realizar diversas operações, a ULA conta com um único sinal de controle, gerado com base na leitura do opcode da instrução, diferenciando suas possíveis formas de execução.

4.2.5 Memória de Dados

A **Memória de Dados** tem o objetivo de armazenar eventuais informações durante a execução de uma instrução.

Sua escrita ocorre a partir do direcionamento de uma *flag*, semelhante ao Banco de Registradores e suas únicas entradas são o endereço de acesso e o dado que será salvo neste respectivo endereço. Contudo, apenas as instruções load e store realizam acesso direto aos seus endereços e os dados contidos nele.

4.2.6 Unidade de Controle

Já a **Unidade de Controle** é o módulo responsável por realizar a troca de todas as *flags* com base na instrução em execução. Possui um papel de extrema importância no

funcionamento do processador, uma vez que diferencia o resultado das instruções a partir dos seus sinais de controle.

Podemos comparar este componente ao cérebro do processador, orquestrando seu funcionamento e ativando cada flag de cada instrução. Ou seja, a UC que é um circuito combinacional - é dito combinacional pois a saída depende única e exclusivamente das combinações das variáveis de entrada recebidas em um dado momento (5) - recebe e interpreta o opcode de cada instrução e a partir desta interpretação define a disposição adequada dos passos de execução do caminho de dados.

4.2.7 MUX

Os **multiplexadores** (MUX), funcionam como um seletor do que qual informação enviará para a saída. Visto que, o mesmo pode possuir dois ou mais dados de entradas.

4.2.7.1 Big MUX

Um elemento importante no desenvolvimento da arquitetura é o chamado **Big MUX**, que possui o objetivo de determinar qual será o endereço correto do Contador de Programa ao término de cada instrução. Seu funcionamento é baseado em sinais seletores que selecionam qual entrada será a saída de retorno do PC.

Este componente foi pensado para contemplar corretamente o funcionamento das instruções *branch on equal* e *branch on not equal*. Que são executadas a partir de um sinal 0 ou 1.

4.2.8 Extensor

Outro componente auxiliar do processador, é o extensor de *bit* (*Sign Extend*). Que possui a função de receber uma entrada de um determinado tamanho em bits, concatená-la com 0's ou 1's até atingir um tamanho específico e mandar esta nova informação para outro componente.

Por exemplo, um extensor pode transformar um código de 16 *bits* em um de 32 *bits*, sem alterar a informação original.

4.2.9 Entrada e Saída

Em relação a entrada e saída de dados do processador, será utilizado um módulo de entrada e saída com o objetivo de efetuar transferências de dados entre o processador e os periféricos de I/O. Existem três possíveis formas de realizar esta ação, por interrupção, por programação e por DMA.

Tomando conhecimento das principais formas de implementação, para este projeto o I/O por programação será utilizado. Visto que ele é o que melhor se adequa às necessidades do processador e ao grau de complexidade.

4.2.9.1 I/O por programação

Ocorre a partir do resultado das instruções de I/O que estão presentes no programa de um computador. Cada transferência de dados é iniciado por uma instrução no programa.Este tipo pode admitir dois tipos de transferência:

Transferência incondicional: a transferência é realizada independentemente do estado da interface ou periférico.

Transferência condicional: a operação de E/S só é realizada se o dispositivo estiver pronto para tal (6). Em geral, o programa deste tipo de transferência possui um laço de espera que efetua constantes testes do estado do dispositivo até que o mesmo indique que a operação pode ser realizada.

4.2.9.2 I/O por interrupção

Neste modo, a transferência é acionada através de uma interrupção, ou seja, a operação De E/S é requerido pelo dispositivo externo através de um pedido de interrupção. Normalmente, este pedido é solicitado quando o dispositivo ou interface está pronto para realizar a transferência.

4.2.9.3 I/O por DMA

O último tipo é o por DMA (*Direct Memory Access*), que pode ser definido como uma técnica de transferência de dados onde os periféricos se comunicam diretamente entre si, utilizando os barramentos de memória e removendo a intervenção da CPU (7). Basicamente, o controlador DMA assume os barramentos para gerenciar diretamente a transferência entre os dispositivos de E/S e a unidade de memória.

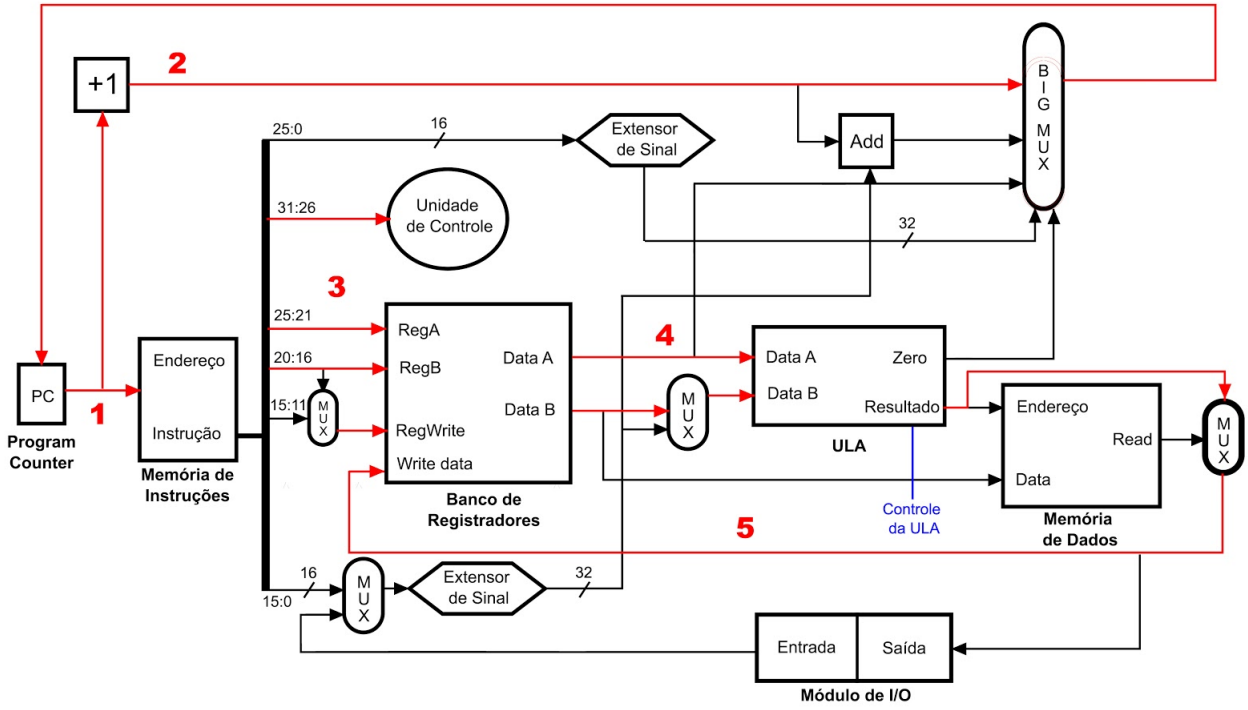
4.3 Exemplos

Para validar o desenvolvimento e funcionamento da arquitetura base proposta, foram efetuados testes no caminho de dados a partir da simulação de algumas instruções e seus diferentes tipos.

Esta etapa é importante pois servirá de guia para o funcionamento do processador ao longo de sua implementação.

4.3.1 Instruções do tipo R

Figura 5 – Caminho de dados para instruções do tipo R



Fonte: Autor

Inicialmente o primeiro teste é para instruções do tipo R, como soma e subtração. Vale ressaltar que todo o funcionamento dos passos citados são orquestrados a partir da unidade de controle que recebe o opcode e determina quais serão os passos adotados no *datapath*.

Passo 1: Inicialmente o endereço da instrução que está disponível no PC é encaminhado para a Memória de Instruções.

Passo 2: Em seguida, o endereço atual do PC é incrementado e ao ser selecionado como saída pelo Big MUX, este novo endereço atualizado é retornado ao módulo do *Program Counter*. Vale ressaltar que este novo código referencia a próxima instrução que será executada.

Passo 3: Ao mesmo tempo, o endereço da primeira etapa é interpretado pela Memória de Instruções que seleciona qual é a instrução referente a este código e envia estas informações ao Banco de Registradores e a Unidade de Controle.

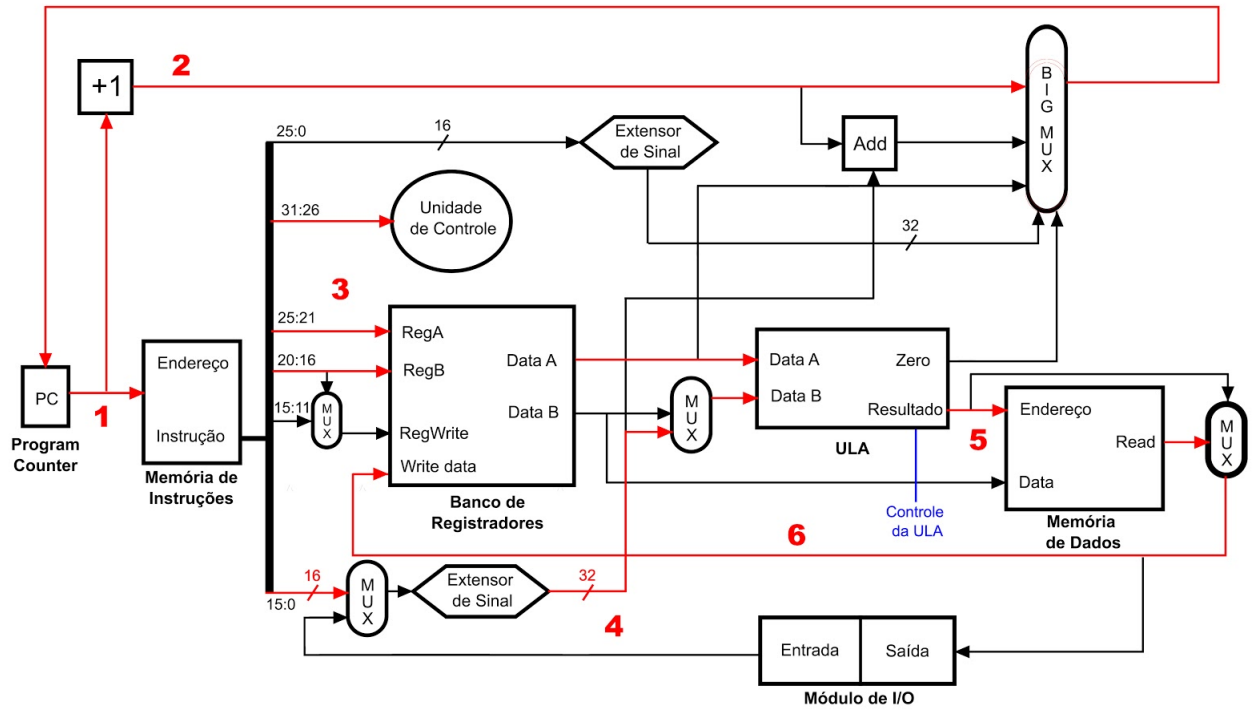
Passo 4: Instruções do tipo R, possuem três registradores. Dois registradores de origem (RegA e RegB), que recebem os dados que servirão como argumento para a ULA efetuar determinada operação. E um registrador (RegWrite) de destino, que recebe o resultado da operação realizada.

Passo 5: O dois valores dos registradores de origem são transferidos para a ULA através dos campos Data A e Data B.

A partir do opcode da instrução, a ULA irá determinar qual operação realizar com os dados e o respectivo resultado é encaminhado para Memória de Instruções (mais especificamente no RegWrite), após ser selecionado como saída do MUX.

4.3.2 Instrução *Load Word*

Figura 6 – Caminho de dados para instrução lw



Fonte: Autor

Outra instrução pertinente é a lw, que possui a função de de movimentação de dados da memória para registrador(8).

Os passos 1 e 2 são os mesmos citados na sessão 4.1

Passo 3: A instrução *lw* utiliza apenas dois registradores (RegA e RegB), o registrador de endereço base é transferido como dado de saída do Banco de Registradores (Data A).

Passo 4: Paralelamente a isso, um imediato de 16bits é estendido pelo módulo de Extensor de Sinal e através do MUX, este sinal estendido serve como entrada para ULA (mais especificamente na entrada Data B).

Passo 5: A partir do opcode da instrução, a ULA irá determinar somar as informações provindas das entradas, ou seja, do sinal estendido e do registrador de endereço base.

Passo 6: O resultado desta operação é transferido para a Memória de Dados que identifica qual é o dado referente ao endereço de entrada. A partir deste endereço, o dado referente volta para o banco de registradores e é salvo.

5 Considerações Finais

Analisando as etapas anteriores deste relatório, é possível perceber que desenvolver um sistema computacional pode ser algo complexo. E para esta arquitetura não será diferente.

Durante o desenvolvimento deste relatório, a principal dificuldade encontrada foi saber por onde começar, visto que devido a complexidade da arquitetura, foi necessário a orientação de alguém que já tinha passado pela experiência do desenvolvimento do projeto em questão. Contudo, com o auxílio dos monitores e de alunos de outros anos, foi possível sanar todas as dúvidas apresentadas.

Os próximos passos para o desenvolvimento deste projeto será iniciar o processo de implementação de cada componente presente na arquitetura base. Para isso será utilizado o *Varilog*, uma linguagem de descrição de hardware que permite a possibilidade de criação de um nível de abstração compatível com as propostas apresentadas.

Após o desenvolvimento e testes de cada componente, será feito a junção do caminho de dados, interligando cada bloco já desenvolvido e garantindo seu total funcionamento em conjunto. Com isso, o programa será compilado em uma placa FPGA (circuito integrado que possui componentes programados) e a funcionalidade será comprovada através da execução de alguns algoritmos.

Referências

1 MAXIEDUCA. *Descubra a importância da memória cache para o computador*. 2017. Disponível em: <http://blog.maxieduca.com.br/memoria-cache-computador/>. Acesso em: 13/04/2018. Citado na página 11.

2 MACEDO, D. *Arquitetura: Von Neumann Vs Harvard*. 2012. Disponível em: <http://www.diegomacedo.com.br/arquitetura-von-neumann-vs-harvard/>. Acesso em: 13/04/2018. Citado 2 vezes nas páginas 12 e 13.

3 PATTERSON, D. A.; HENNESY, J. L. *Computer Organization and Design*. 5th edition. ed. Waltham/MA, EUA: Morgan Kaufmann, 2007. Citado na página 16.

4 COMPUTER Hardware/Software Architecture. [S.l.]: Morgan Kaufmann, 1998. Citado na página 20.

5 UFMT. *Circuito Combinacional*. 2016. Disponível em: <http://araguaia2.ufmt.br/professor/disciplina_arquivo/90/201608301202.pdf>. Acesso em: 13/04/2018. Citado na página 21.

6 MARTINO, P. J. M. D. *Interface de Entrada e Saída*. 2004. Disponível em: <http://www.dca.fee.unicamp.br/courses/EA078/1s2004/arquivos/turma_ab/cap7.pdf>. Acesso em: 10/04/2018. Citado na página 22.

7 GEEKS, G. for. *I/O Interface (Interrupt and DMA Mode)*. 2016. Disponível em: <https://www.geeksforgeeks.org/io-interface-interrupt-dma-mode/>. Acesso em: 11/04/2018. Citado na página 22.

8 CENTODUCATTE, P. C. *Conjunto de Instruções MIPS*. 2002. Disponível em: <http://www.ic.unicamp.br/~pannain/mc542/aulas/ch3_arq.pdf>. Acesso em: 08/04/2018. Citado na página 24.