Laboratório 2 - OAC
Desenvolvimento da Unidade Operativa Pipeline MIPS

Rosana Santos Ribeiro, 16/0047269 - rosanasrib@gmail.com
Mateus Freitas Cavalcanti, 16/0137519 - mat.fcavalcanti@gmail.com
Mariana Mendanha, 16/0136784 - mariana_mendanha@hotmail.com

[1]Dep. Ciência da Computação - Universidade de Brasília (UnB)
CiC - Organização e Arquitetura de Computadores - Turma B

1. Objetivos

Permitir que o aluno(a) se familiarize com a construção, execução de instruções e controle da unidade operativa pipeline MIPS. Esta atividade tem como intuito montar os principais blocos da unidate operativa (datapath), que compõem a arquitetura MIPS Pipeline. Estes blocos deverão ser capazes de processar conjuntos de instruções de 32 bits pré-compiladas, respeitando todas as condições de implementações necessárias para a execução destes tipos de instruções, fazendo uso de um conjunto de memória seguindo a proposta apresentada em sala.

O projeto desta disciplina é uma atividade planejada de forma a complementar e reforçar o conteúdo programático da disciplina. Propiciando assim melhorias na capacidade de observação, análise e compreensão das metodologias de organização e arquitetura de computadores.

Desta forma cabe ao grupo, partindo da premissa que possui os requisitos para o curso, juntamente com o conteúdo adquirido nas aulas teóricas, desenvolver todas as etapas da implementação solicitada.

2. Introdução

O pipeline foi criado para melhorar o desempenho dos hardwares que até então só processavam uma instrução por vez. Uma instrução simples podia ser executada em apenas um ciclo de clock, enquanto instruções mais complexas demoravam vários ciclos de clock para serem concluídas. Seria mais ou menos como montar um carro de maneira artesanal, peça por peça. Uma instrução simples podia ser executada em apenas um ciclo de clock, enquanto instruções mais complexas demoravam vários ciclos de clock para serem concluídas. Seria mais ou menos como montar um carro de maneira artesanal, peça por peça.

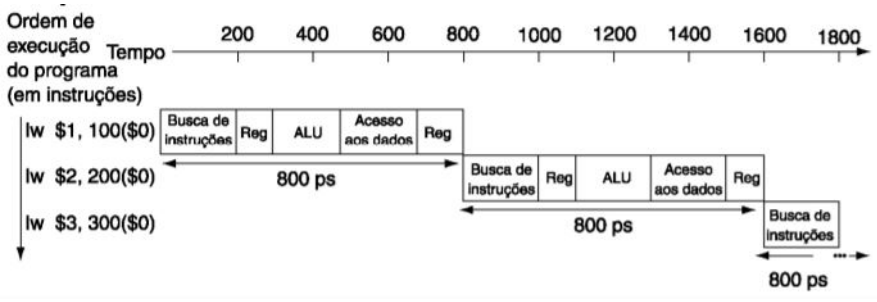

Figure 1. Processamento uniciclo

Surgiu-se então a necessidade de melhorar o desempenho das máquinas, para isso, o pipeline foi empregado, até então só era utilizado em processadores RISC e consiste em dividir o processador em vários estágios distintos. No caso deste laboratório foi implementado 5 níveis, ou seja, 5 estágios.

Em resumo, as instruções são divididas em 5 estágios uma vez que cada uma destas etapas é executada por uma porção especializada da CPU, podendo colocar mais de uma instrução em execução simultânea, e isto ocorre em cascata, como podemos ver a seguir:

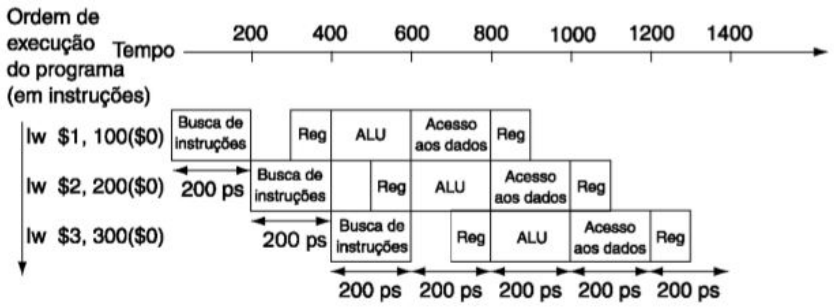

Figure 2. Processamento pipeline

Quando é carregada uma nova instrução, ela primeiramente passa pelo primeiro estágio, que trabalha nela durante apenas um ciclo de clock, passando-a adiante para o segundo estágio. A instrução continua então sendo processada sucessivamente pelo segundo, terceiro, quarto e quinto estágios do processador. A vantagem desta técnica, é que o primeiro estágio não precisa ficar esperando a instrução passar por todos os demais para carregar a próxima, e sim carregar uma nova instrução assim que se livra da primeira, ou seja, depois do primeiro pulso de clock. Se por acaso a instrução não tenha sido completada mesmo após passar pelos 5, voltará para o primeiro e será novamente processada, até que tenha sido concluída. Voltando ao exemplo de produção de carros, seria como se trocássemos a produção artesanal por uma linha de produção, onde cada departamento cuida de uma parte da montagem, permitindo montar vários carros simultaneamente.

Isto traz um uso mais racional da capacidade computacional com ganho substancial de velocidade. Entre os problemas enfrentados estão a dependência de instruções anteriores e desvios que dificultam o processo, bem como a diferença de complexidade de instruções que fazem com que as mesmas possam levar um tempo variável para execução.

Utilizando tudo que aprendemos anteriormente e passado em sala de aula, podemos pular para a implementação do nosso programa, que se trata de um processador pipeline, que em resumo, recebe instruções vindas de um arquivo .mif, MIPS de 32 bits são decifradas e executadas uma a uma em ordem, com a ajuda dos blocos necessários para cada tarefa. Isso tudo implementado no ambiente de desenvolvimento QUARTUS.

3. Materiais e métodos

De acordo com o roteiro fornecido, o primeiro requisito a ser desenvolvido é a criação do controle da unidade operativa, que foi desenvolvida a partir da unidade apresentada em sala, foram realizadas modificações para atender os requisitos do roteiro, que contavam com a implementação

de 31 instruções diferentes dos tipos R, I e J, além de módulos de tratamento de exceções, hazards de dados e detecção de overflow. Tudo isso com execução automática, o inputs vindo de arquivos .mif, todas as instruções e dados são armazenadas na memória de instruções e dados, respectivamente, e então inicia-se o processamento das mesmas.

Inicialmente, foi montado um modelo uniciclo do processador e assim, adicionadas todas as 31 instruções previamente definidas. Tendo o funcionamento correto deste modelo, pode-se adicionar os blocos (barramentos de dados, módulo de tratamento de exceções, hazards de dados e detecção de overflow) para transformar o processador no modelo pipeline.

A seguir será demostrado o funcionamento de cada bloco:

3.1. Bloco PC (contador de programa)

Este módulo corresponde a um registrador de 32 bits que contem o endereço de instrução, este é incrementado de 4 para que possamos compor o endereço da próxima instrução, ou pode receber outros valores no caso dos desvios incondicionais (jump, jr e jal).

3.2. Memória de instruções

Este módulo possui a entrada do endereço vindo de PC e tem saída que corresponde ao código referente de instrução 32 bits daquele endereço. Este bloco pode ser construído a partir da ferramenta *MegaWizard Plug-In Manager*, onde é usado um arquivo .mif para referenciar os endereços passados por meio de PC.

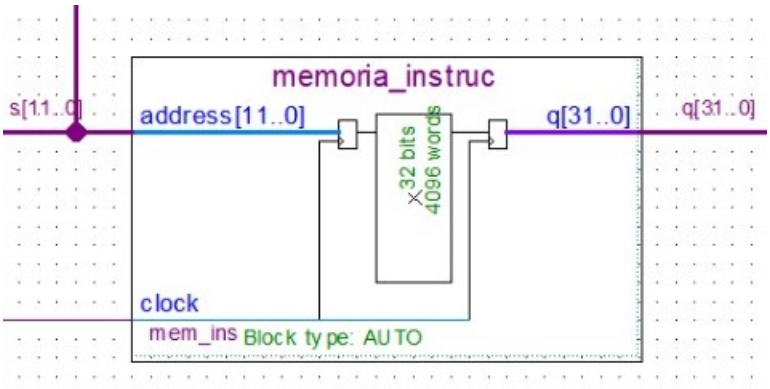

Figure 3. Memória de instruções

3.3. Memória de dados

Neste módulo ocorre o armazenamento e deslocamento de dados da memória, possui uma entrada para endereço, uma para dados de escrita que é usada quando necessário, duas entradas de controle vindas do controle principal, estas são ”LeMem” e ”EscreveMem” que decidem se haverá escrita ou leitura da memória e uma saída de dados para caso de leitura da memória. Assim como na memória de instruções, este bloco pode ser implementado por meio de um arquivo por meio da ferramenta *MegaWizard Plug-In Manager* e de um arquivo .mif que faz referência aos dados armazenados na memória e seus respectivos endereços que podem, ou não, ser acessados durante a execução das instruções.

3.4. Banco de registradores

Este módulo possui 3 entradas de 5 bits correspondentes a dois registradores de leitura e um de escrita, uma entrada de 32 bits correspondente aos dados para escrita em certo registrador, uma entrada de controle que diz ser necessário ou não escrever no módulo de registradores e duas saídas de dado de leitura. Segue esquemático e datapath no bloco para escrita e para leitura no bloco:

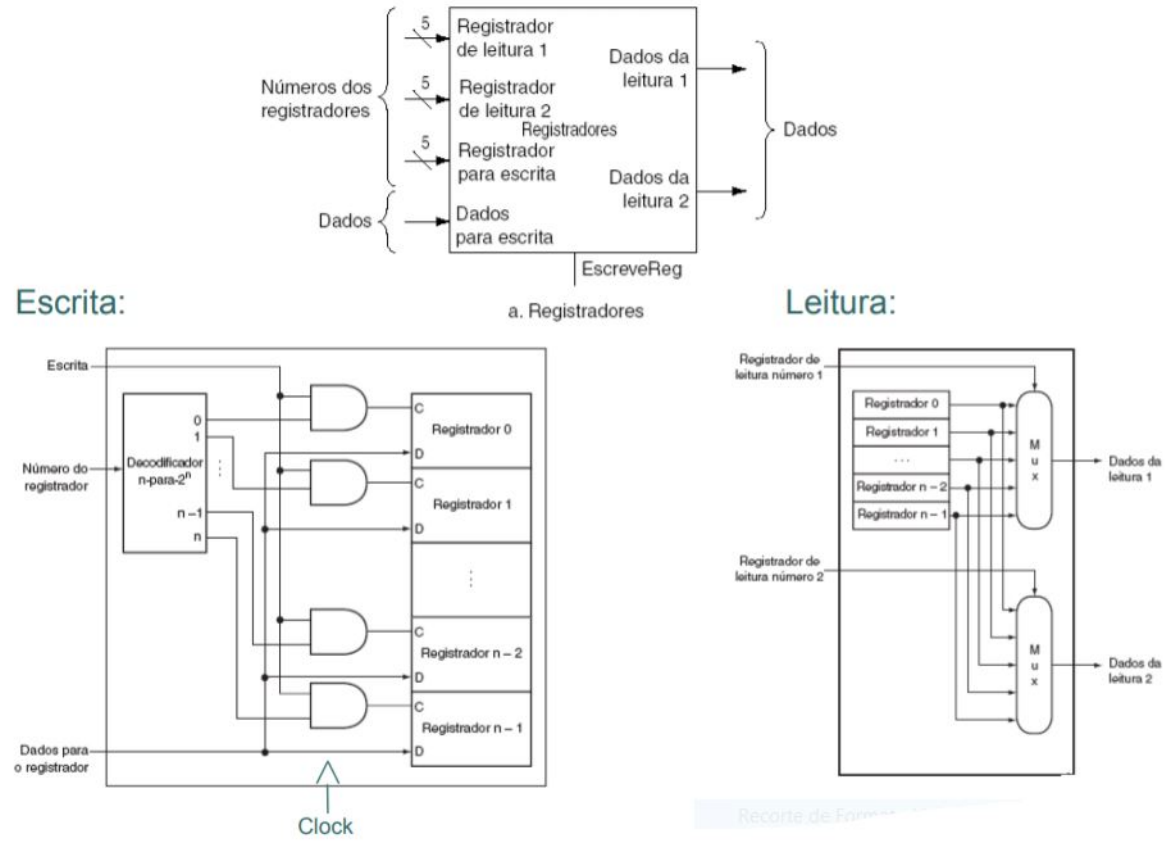

Figure 4. Banco de registradores

(adicionar informações sobre high-low)

3.5. ULA

Como seu nome já diz (unidade lógica e aritmética), a ULA é responsavel pelas operações lógicas e aritméticas realizadas no processador. Para realizar tal tarefa, essa unidade é controlada por diversos sinais definidos no processador:

- Entradas:
  - Operandos **a** e **b**, ambos de 32 bits que efetuam diversas operações dentro da ULA;
  - Campo **Shamt** da instrução, que fornece as informações necessárias para as operações de shift;

  - O código que seleciona qual operação calculada deve ser retornada pelo multiplexador que seleciona as saídas da ULA (Tabela 1).
- Saídas:
  - Resultado desejado e selecionado por meio do código de entrada (4 bits);
  - O bit "Zero" que determina as condições de salto condicional das intruções beq e bne;
  - O bit de Overflow que determina se ocorreu overflow na operação retornada;
  - O resultado da multiplicação (1 conjunto de 64 bits) das entradas **a** e **b** que deve ser armazenado nos registradores HI e LO (instrução mult);
  - O resultado da divisão (2 conjuntos de 32 bits - quociente e resto) das entradas **a** e **b** que devem ser armazenados nos registradores HI e LO (instrução div).

Vale observar que, no modelo implementado, as operações de multiplicação (mult) e divisão (div) saem da ULA sem passar pelo multiplexador de seleção de operação. Pode-se observar que, nestes casos, são retornados valores de 64 bits que não poderiam ser selecionados pelo multiplexador de 32 bits da saída. Logo, executando também essas instruções no terceiro estágio do pipeline, tem-se que essas saídas estão disponíveis sem depender da seleção do código do controle da ULA.

3.6. Controle principal

Para um funcionamento correto do processador pipeline projetado, é preciso uma unidade de controle geral que organiza o fluxo de dados no circuito. Para isso, foi projetado a unidade de controle do processador que fornece os seguintes bits de controle para o processador de acordo com uma determinada instrução lida:

- RegDST - 2 bits que selecionam qual o registrador de destino da instrução (em que registrador será salvo o dado de retorno);
- OrigULA - 1 bit que seleciona qual dado (segundo dado de leitura do banco de registradores ou campo imediato extendido) será utilizado para a operação na ULA;
- MemparaReg - 2 bits que selecionam qual dado (leitura da memória, PC+8 ou resultado da ULA) será retornado para o banco de registradores para ser salvo;
- EscreveReg - 1 bit que determina se o dado que retorna para o banco de registradores será salvo;
- LeMem - 1 bit que determina se a memória será utilizada para leitura;
- EscreveMem - 1 bit que determina se o dado na entrada da memória de dados será escrito;
- Branch - 2 bits que determinam as condições de desvios condicionais (beq, bne e bgez) do processador;
- OpULA - 2 bits que determinam que tipo de operação será executada na ULA (acesso à memória, desvio ou aritmética);
- Controle_ULA - 1 bit que determina qual controle da ULA (controle ula ou controle ula auxiliar) vai fornecer o código de operação para execução da operação na ULA;
- Jump - 1 bit que determina os casos Jump e Jal de desvios incondicionais;
- Ext_Zero - 1 bit que determina se o imediato será extendido com sinal ou com zeros;
- Tipo_r - 1 bit que determina se a instrução sendo executada é do tipo R ou não.

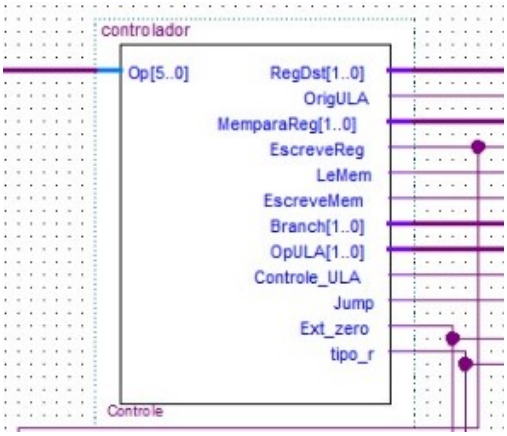

Figure 5. Banco de controle

Para o modelo pipeline, os bits de controle provenientes dessa unidade são transmitidas aos diversos estágios por meio dos barramentos para que não ocorra a perda destes sinais.

3.7. Controles secundários

Tirando os o controle principal do pipeline, é preciso implementar uma unidade de controle secundária que determina que operação será retornada pela ULA por meio de um conjunto de 4bits que controlam a seleção de um mux de 16 entradas na saída desta unidade. Porém, como existem instruções que não possuem o campo Function mas executam operações aritméticas na ULA, é preciso criar 2 unidades de controle da ULA (controle ULA e controle ULA auxiliar) para efetuar a codificação de todas as instruções implementadas. Para efetuar a seleção de qual controle da ULA fornece o código para tal instrução, pode-se utilizar um bit proveniente do controle principal (controle_ULA) que determina qual unidade fica encarregada da instrução atual. A tabela 1 apresenta todas os possíveis códigos de controle da ULA para as instruções implementadas.

Table 1. Tabela de controle da ula

Instruções	Referência	Controle
add	100000	0010
addu	100001	0010
and	100100	0000
xor	100110	0011
sub	100010	0110
slt	101010	0111
subu	100011	0110
or	100101	0001
nor	100111	1100
addi	001000	0010
andi	001100	0000
xori	001110	0011
ori	001101	0001
sll	000000	0100
srl	000010	0101
srav	000111	1000
sw	101011	0010
lw	100011	0010
beq	000100	0110
bne	000101	0110
lui	001111	1001
clo	100111	1100

3.7.1. Controle ULA

Esta unidade de controle da ULA é utilizada quando se trata de instruções do tipo R, acesso à memória ou de desvio condicional. Sendo assim, os bits de controle da ULA são:

- Op - 4 bits que determinam a operação que será selecionada na saída da ULA;
- Jr - 1 bit de determina o desvio incondicional da instrução Jr;
- C_Overflow - 1 bit que efetua o controle dos casos de overflow, desconsiderando essa exceção para execução das instruções addu e subu;
- HILO_enable - 1 bit que efetua o controle da entrada ENABLE do bando de registradores HILO para que ocorra a gravação de ”lixo” nesses registradores;
- eh_div - 1 bit que seleciona qual a entrada dos registradores HI e LO (divisão ou multiplicação);
- move_HILO - 2 bits que efetuam a seleção do dado de *write back* a ser escrito no banco de registradores, no caso das instruções mfhi e mflo.

3.7.2. Controle ULA auxiliar

Uma versão do controle ULA, porém só codifica as intruções que não possuem campo function e executam operações aritméticas na ULA. Os únicos bits de controle retornado por essa unidade controlam a saída da ULA.

3.8. Barramentos

Como visto anteriormente, o pipeline projetado apresenta 5 estágios de funcionamento, onde existem controles específicos para cada parte do processamento. Porém, como o processo é paralelo, ou seja, várias instruções são executadas simultaneamente, é preciso transmitir esses dados de controle para as próximas etapas sem perder informação. Para isso, pode-se usar os barramentos, que são unidades de armazenamento em série que efetuam a transmissão de dados de controle durante o processamento das instruções. Para o pipeline implementado, foram utilizados 4 barramentos : if/id, id/ex, ex/mem e mem/wb.

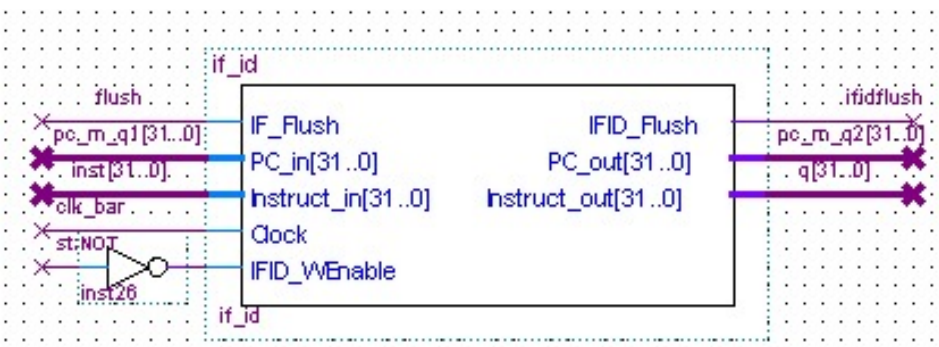

Figure 6. Barramento if/id

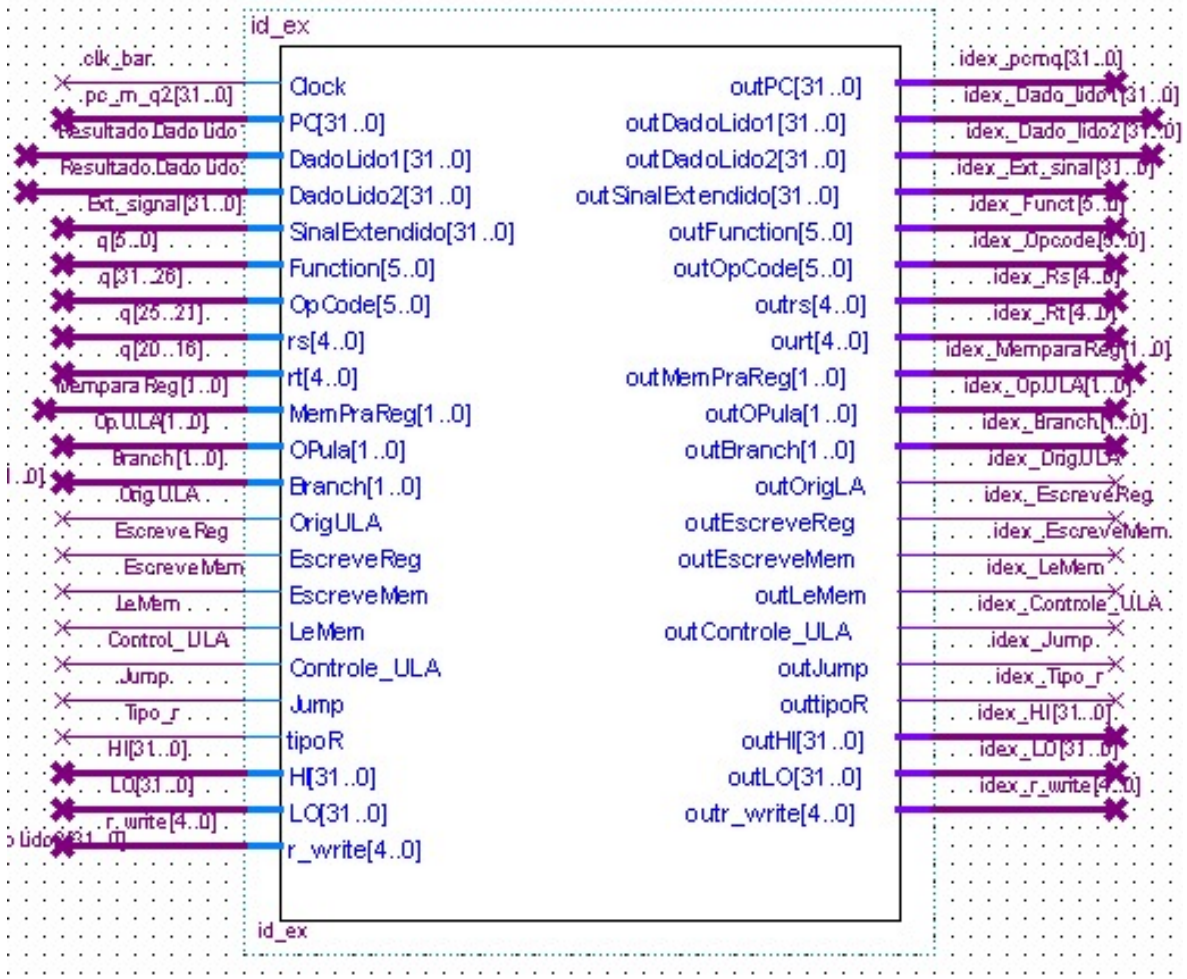

Figure 7. Barramento id/ex

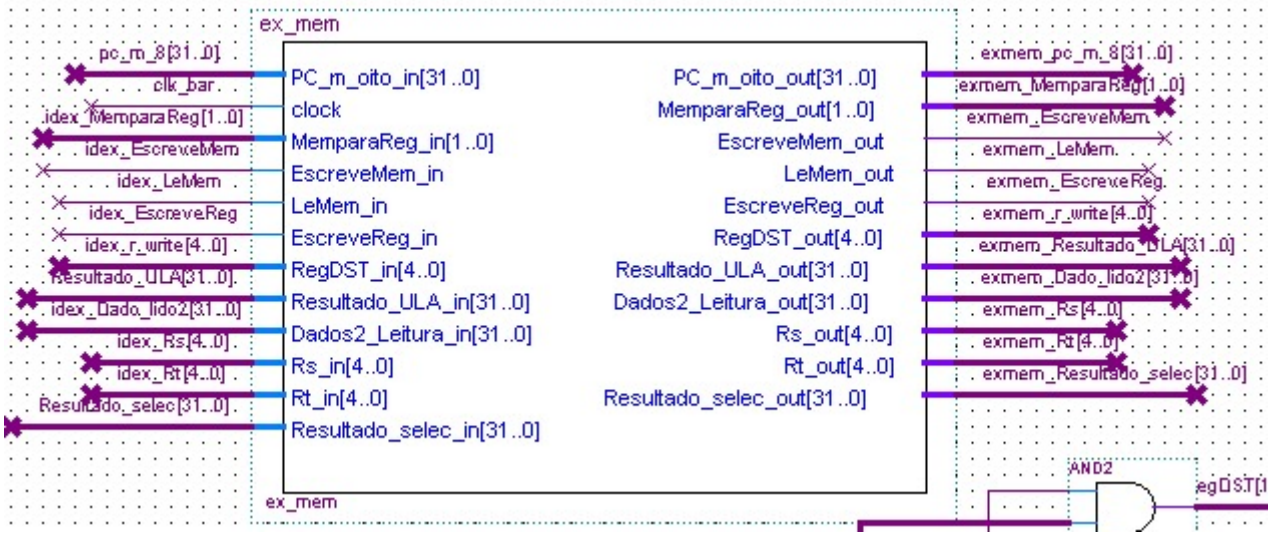

Figure 8. Barramento ex/mem

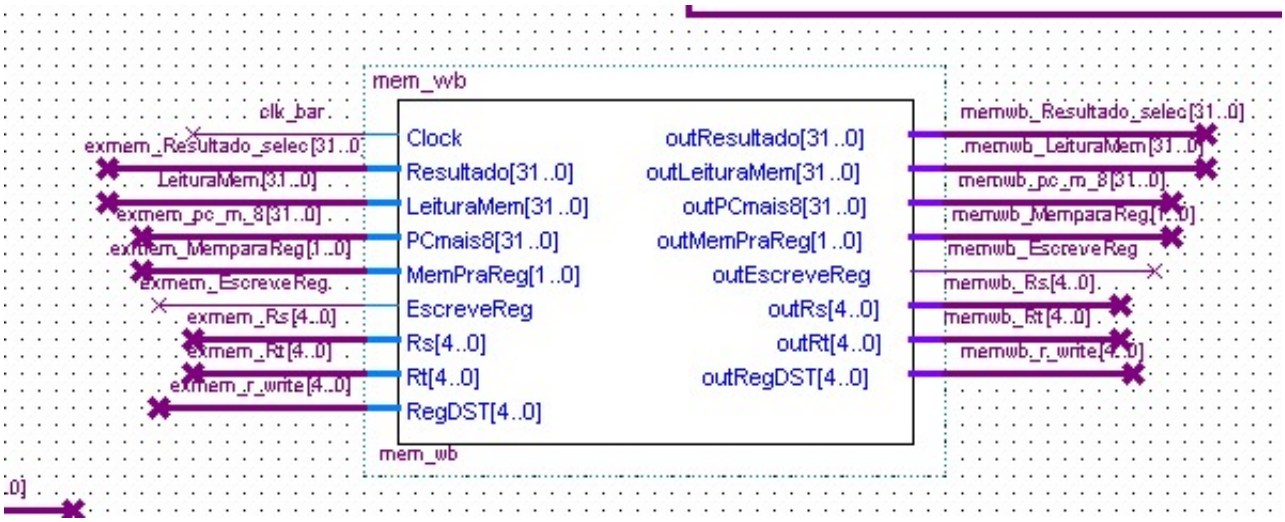

Figure 9. Barramento mem/wb

Como pode-se ver nas figuras acima, os barramentos que foram utilizados armazenam os dados necessários para cada etapa das instruções implementadas:

- IF/ID - Contém o endereço de PC e a instrução que vai ser processada;
- ID/EX - Contém o endereço de PC + 4, os bits de controle do circuito e os dados que são passados para a operação na ULA;
- EX/MEM - Contém os dados para um possível acesso da memória e de armazenamento de conteúdo no banco de registradores;
- MEM/WB - Contém os dados para a escrita no banco de registradores e os bits de controle para definir o caminho do dado de alimentação.

3.9. Hazards e exceções

Um dos principais obstáculos para a implementação efetiva do processador modelo pipeline é a ocorrência de Hazards estruturais, de dados e controle. Para esses casos, foram implementadas unidades de detecção e processamento desses problemas. Para isso, a execução stalls e fowards foram essenciais para o funcionamento do trabalho implementado. Para o caso das exceções, foram considerados 2 casos:

- Instrução inválida;
- Overflow na operação da ULA

Para o caso de instrução inválida, o endereço PC daquela instrução é gravada no registrador EPC e o CAUSE é definido para o valor 1, levando PC para o endereço 0x80000180. Se ocorreu um overflow na operação da ULA, o endereço PC + 4 do barramento id/ex é subtraído em 4 e tem seu valor gravado em EPC, definindo CAUSE para o valor 1, levando PC também para o endereço 0x80000180.

4. Resultados

Utilizando uma composição de inúmeras instruções de teste para analisar o funcionamento do processador desenvolvido, foi possível obter resultados diversos que demonstram o funcionamento de tal. Porém, tais resultados foram encontrados para a implementação do modelo uniciclo e por isso não foram documentados já que não são o objetivo principal deste trabalho. Ao adicionar os blocos necessários para a montagem no pipeline, foram encontradas saídas indesejadas e não foi possível corrigí-las.

Pode-se observar a tabela (Table 1), em que a primeira coluna informa as possíveis instruções utilizadas no arquivo .mif, a segunda são os seus tipos a terceira coluna representa os seus respectivos códigos hexadecimais de representação.

Table 2. Tabela

Entradas	Tipo	Ref
add	rd, rs,rt	0x00000020
addu	rd, rs,rt	0x00000021
and	rd, rs,rt	0x00000024
xor	rd, rs,rt	0x00000026
sub	rd, rs,rt	0x00000022
slt	rd, rs,rt	0x0000002A
subu	rd, rs,rt	0x00000023
or	rd, rs,rt	0x00000025
nor	rd, rs,rt	0x00000027
addi	rt, rs, imm	0x20000000
andi	rt, rs, imm	0x30000000
xori	rt, rs, imm	0x38000000
ori	rt, rs, imm	0x34000000
div	rs, rt	0x0000001A
mult	rs, rt	0x00000018
jr	rs	0x00000008
mfhi	rs	0x00000010
mflo	rs	0x00000012
sll	rd, rt, sa	0x00000000
srl	rd, rt, sa	0x00000002
srav	rd, rt, rs	0x00000007
sw	rt, offset	0xAC000000
lw	rt, offset	0x8C000000
beq	rs, rt, offset	0x10000000
bne	rs, rt, offset	0x14000000
bgez	rs, offset	0x04000000
j	target	0x08000000
jal	target	0x0C000000
lui	target	0x3C000000
li	target	0x00000000
clo	target	0x00000000

5. Discussão e conclusões

Comparando os resultados previstos pela teoria com os obtidos, chegamos à conclusão de que todos os casos pedidos no roteiro alcançaram resultados esperados para o modelo uniciclo do processador, isto pode ser confirmado anteriormente em ”resultados”, onde todas as instruções desenvolvidas pelo grupo podem ser visualizadas.

O grupo conseguiu implementar todas as funções, vale observar que instruções que não possuem campo function, mas que na ula a operação deve ser aritmética necessitaram da implementação de um novo controle de ula para que fossem processadas corretamente, tais instruções trouxeram maior dificuldade ao grupo, são elas: lui,addi,andi,ori,xori e clo.

Neste experimento, pôde-se visualizar também problemas referentes ao efeito overflow, que pode ser causado pela extensão de sinal, caso visto em sala de aula. Esse fenômeno foi corrigido usando um bit de controle adequado.

Apesar de todas as dificuldades, o grupo conseguiu obter resultados satisfatórios para o modelo uniciclo. Os conceitos passados em sala de aula como algumas funções e lógicas básicas foram reforçados, enquanto que novos conceitos foram introduzidos e causaram questionamento, acarretando em muita pesquisa e aprendizado, mesmo com as falhas da aplicação.

References

[1] ASCII TABLE. **Ascii Table and Description**. Disponível em : http://www.asciitable.com Acesso em: 03/05/2019.

[2] OPEN CORES. **Plasma - most MIPS I(TM) opcodes :: Opcodes**. Disponível em: https://opencores.org/projects/plasma/opcodes. Acesso em: 03/05/2019.

[3] VIDAL, FLÁVIO DE BARROS. **Laboratório 2**. Disponível em: https://aprender.ead.unb.br/pluginfile.php/195139/mod_resource/content/9/Lab02-1-2019.pdf. Acesso em 20/06/2019.