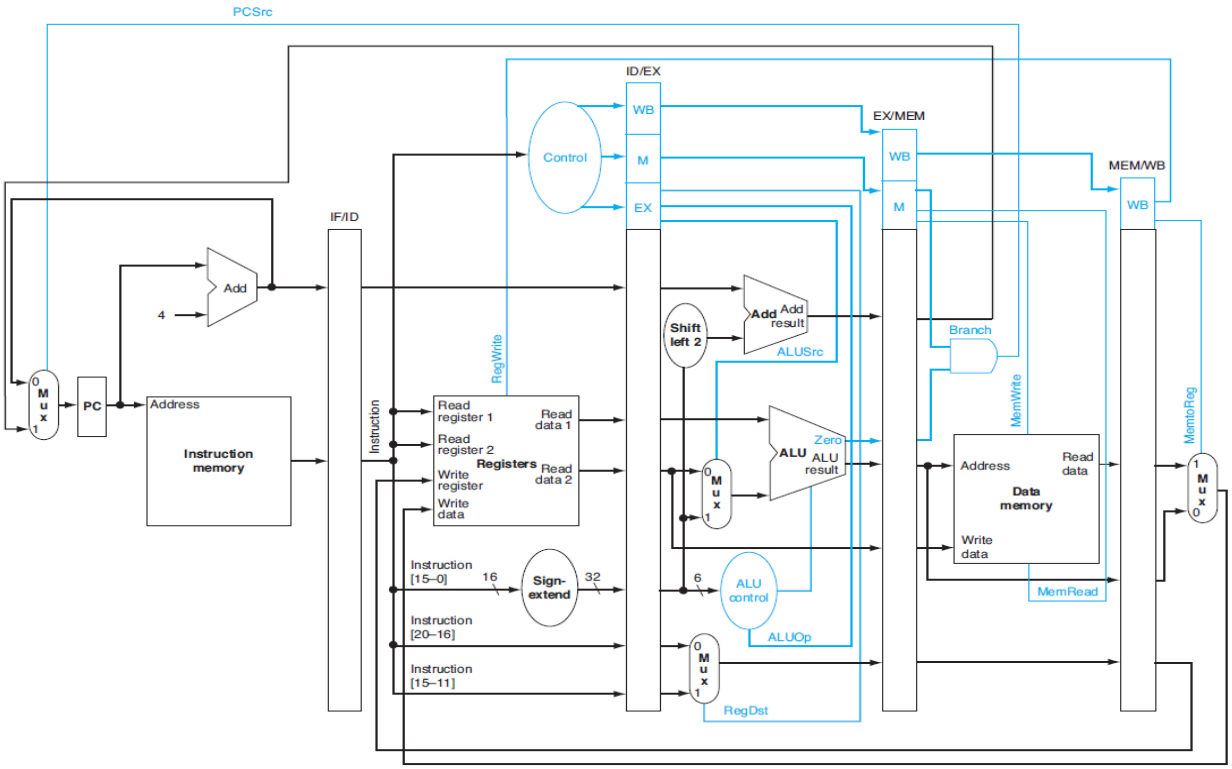
Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Disciplina Organização de Computadores II (DCC007) - 2014/1

Entrega Final.

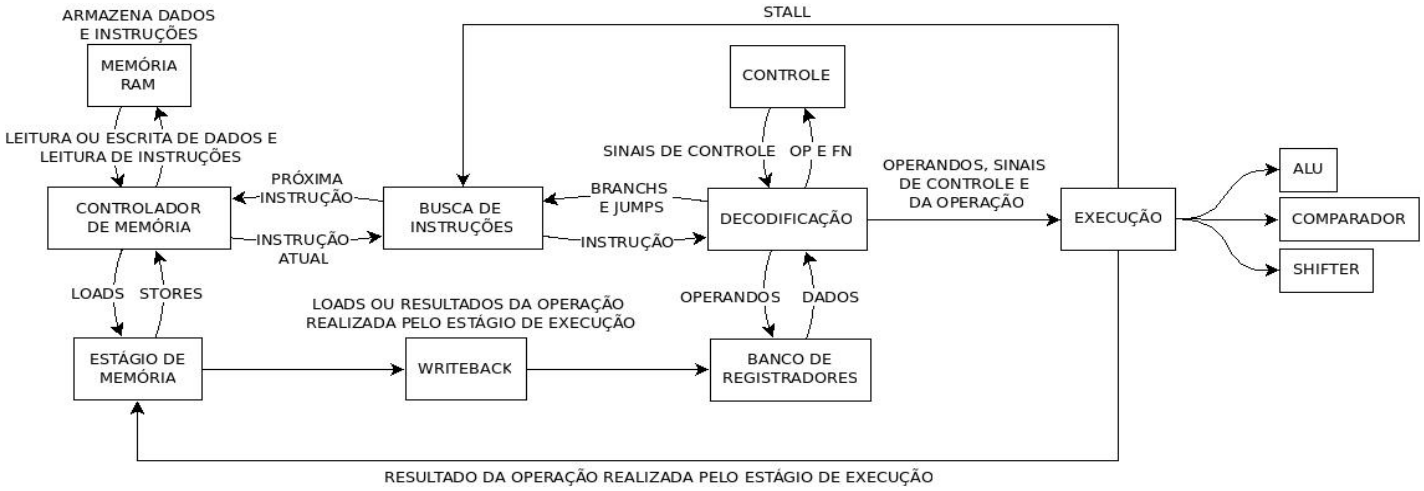
Grupo 1 Alunos:
Manassés Ferreira Neto
Elias da Silva Barroso Soares
Nivaldo Roberto Teixeira Junior
Vitor Guilherme Ribeiro Lopes
Douglas Sales Silva.

1. Introdução:

Ao longo do semestre fomos implementando pequenos módulos em verilog para que no final pudéssemos junta-los e construir um MIPS simplificado. Tal MIPS é parecido com o da figura abaixo:



Entretanto, no nosso a memória de dados é a mesma que a de instruções, além de ter um conjunto de instruções reduzido. A ideia seria juntar tudo e rodar na FPGA Altera.
O fluxo de execução do MIPS implementado por nós encontra-se abaixo:



2. Descrição por módulos:

2.1 - ALU

A unidade lógica aritmética (ALU – Arithmetic Logic Unit) é o módulo responsável por operações lógicas e aritméticas. O módulo implementado possui dois arrays de 32 bits, A e B, que servirem como entrada dos operandos. Além disso, terá como entrada um array de 3 bits que serve para receber os códigos da operação a ser realizada.
Inicialmente, o módulo compara os códigos de operações, executando o que cada um determina. Dentro das operações de adição e subtração, é feita a verificação de overflow de acordo com a tabela anterior. Por fim, é feita a comparação entre os operandos. Se caso o código de operação não for reconhecido, foi tomada a decisão de projeto de não executar quaisquer operação sobre os operandos, apenas comparando seus valores e atualizando a saída.

2.2 - Shifter

O shifter é a unidade resposável pelo deslocamento de bits, que pode ser lógico ou aritmético e para esquerda ou para a direita. Sua entrada é um valor de 32 bits que pode ser deslocados por até 31 posições.
Este módulo foi implementado através de um seletor onde reconhece qual tipo de deslocamento será calculado e o executa no operando de entrada. Este operando é deslocado de acordo com o indicado. Quando não houver a codificação correta do desclocamento, o resultado reflete apenas o que foi dado no operando de entrada.

2.3 - Comparador

O comparador é um módulo que compara dois valores tem como saída o resultado dessa comparação que pode ser utilizada em instruções de desvio. Este módulo possui dois arrays com 32 bits chamados de A e B que serão usados para abrigar os valores a serem comparados, além de um array com 3 bits que será usado como controle que indicará qual operação de comparação a ser realizada de acordo com a tabela abaixo e uma saída que retornará 1 caso a comparação realizada seja verdadeira e 0 caso contrário.

O módulo foi implementado como um seletor, onde o código de comparação é verificado de acordo com os códigos anteriormente citados e o valor da comparação entre os operandos A e B é retornado na saída. Por padrão, é retornado o valor 0 caso o código de comparação não seja reconhecido.

2.4 - Banco de Registradores:

O Banco de Registradores é o módulo responsável por armazenar dados de tal forma que o acesso a eles seja veloz e pouco custoso. As instruções da máquina tem como operandos registradores, então, quando necessários dados da memória, estes precisam ser carregados para o banco. Além disso, os resultados das instruções são também armazenados nos registradores. No banco implementado, em um mesmo ciclo de clock é possível retornar o valor de dois registradores e alterar o valor de um.

Para a implementação do reset foi utilizado um generate block junto a um for, evitando assim a necessidade de escrever uma linha de código para cada um dos 32 registradores.

2.5 - Controle:

O Controle é o módulo responsável por informar aos vários componentes do hardware, nas várias etapas das instruções, o que deve ser feito ali e com quem deve ser feito. Ele é quem controla tudo o que está acontecendo e tudo o que vai acontecer. Para isso, ele emite, para cada unidade, sinais de controle a partir do estágio de decodificação.

Para a implementação do Controle, são utilizados cases, que, para cada combinação de entrada, emitem determinadas saídas.

2.6 - Busca de Instruções

O módulo da Busca de Instruções tem duas funções:

• Enviar a instrução do controlador de memória ao estágio de decodificação

• Contagem do programa através do Program Counter (PC)

Como o PC pode ser alterado de forma sequencial ou com dados vindos de instruções de desvios (branches), é essa unidade a responsável por realizar esse gerenciamento. Além disso, essa unidade é informada quando há a necessidade de realizar um stall no

pipeline, devido a hazard estrutural (a memória de instruções e a memória de dados são a mesma).
Esse módulo recebe como entrada a instrução lida da RAM, sinais que indicam se haverá e como serão os acessos à memória, e sinais que parametrizam e definem o próximo PC. As saídas são sinais referentes ao próximo PC e sinais que definem a leitura ou não na RAM.

2.7 - Controle
O Controlador de Memória é responsável por fazer a interface com a memória RAM e controlador. É ele quem diz onde quando ocorre leitura e escrita na memória, além de controlar as posições onde elas ocorrem. Como a memória de instruções e a memória de dados são a mesma, todo esse acesso é controlado por esse módulo.
A implementação do MIPS é de 32 bits; entretanto, a FPGA a ser utilizada no projeto tem palavras de 16 bits. Por isso, o sistema foi implementado de tal forma que duas palavras consecutivas na memória formam um dado - ou uma instrução - de 32 bits, fazendo com que os endereços sejam sempre múltiplos de dois. Como o endereçamento é feito por byte, os endereços para acesso só são válidos se múltiplos de 4. Devido a esses motivos, o acesso é realizado em dois ciclos de clock - em cada um é acessado uma metade da palavra.

2.8 - Decodificação
A unidade de decodificação é responsável por receber uma instrução do estágio de busca de instruções, decodificá-la passando os opcodes e functions para a unidade de controle e acessar os registradores a serem utilizados como operandos. Além disso, esta unidade é responsável por receber os sinais vindos da unidade de controle de acordo com a operação a ser realizada e propagá-los à unidade de execução. Instruções de branch e jump também são resolvidas neste estágio. Este módulo depende dos módulos de controle e comparação já implementados em entregas anteriores.

2.9 - Execução
O estágio de execução é responsável por receber operandos do estágio de decodificação assim como a indicação de qual operação será realizada e realiza-la. O resultado é passado para o estágio de memória. Este módulo depende da ALU e do Shifter já implementados em entregas anteriores.

2.10 - Memória
O módulo de memória é responsável ler e escrever dados da/para a RAM. Este estágio tem prioridade máxima para acessar a RAM. Além disso, este estágio é responsável por

receber os dados do controlador de memória e também é responsável por passar os valores que estão relacionados ao estágio de writeback para o banco de registradores.

2.11 Entrega final
A entrega final pode ser resumida como a união de todos os módulos no MIPS.
Na pasta Entrevista3 (https://github.com/mfer/pinca-puca/tree/master/Entrevista3), concentramos todos os nossos códigos, bem como os arquivos de projeto do Quartus. Há nessa pasta, um Makefile que permite 1) a "compilação" do verilog Mips.v, 2) a execução do testbench tb_Mips.v e 3) abertura do vcd, através do seguinte comando:
$ make tb_Mips_wave

3. Implementação e Decisões de projeto:

Em cada módulo a equipe de teste e desenvolvimento trocava, sendo que Manassés e Douglas pertenciam ao mesmo grupo e o Vitor, Elias e Nivaldo pertenciam a outro.
Os módulos sofreram muitas alterações, conforme íamos incluindo uns aos outros ou descobrindo erros. O controlador de memória foi refeito de 3 a 5 vezes. O decode, o execute, o banco de registradores, a memória, foram alterados na última entrega, pois, refizemos todos os testes benchs e encontramos vários sinais conectados erroneamente. Forneceram-nos uma FPGA Altera De2-115, onde as duas primeiras entrevistas foram apresentadas nela, mas na última entrevista notamos que nossa placa estava com problema, ao testar um código que funcionava numa FPGA mais simples, tal código não funcionou, além de que os outros grupos que pegaram a FPGA Altera De2-115 também tiveram problemas. Trocamos de placa muito em cima da hora e na hora de colocar o projeto na placa, começaram a aparecer vários erros de compilação, que infelizmente, não foram resolvidos a tempo da entrega.

4. História do Trabalho Prático:

Aprender sintetizar hardware com software é muito desafiante. Nossas maiores dificuldade foram:
0) aprender verilog/gtkwave;
1) implementar testbenches decentes;
2) criar/aprender a usar a infraestrutura quartus-de2-115-qdz-qof-controlpannel;
3) fazer o memcontroller funcionar na de2-115;
4) integrar os módulos;

O grupo foi bem regular, procurou dividir bem as tarefas, quando a coisa ficou feia, reuniu-se. Tivemos uma baixa importante no final, Vitor acidentou-se (está bem) mas não pôde participar dessa última entrega/entrevista.
Consideramos que aprendemos demais! Estamos tendo uma formação melhor do que teríamos há dois anos atrás (Certo?), quando não empregava-se verilog nessa disciplina. Lamentamos não termos feito o MIPS funcionar na FPGA. Por outro lado, temos orgulho de ter feito a ALU funcionar!
Não foi por falta de dedicação ou por constância de trabalho que nos surpreendemos com esse resultado negativo na FPGA. Desde o ínicio buscamos concentrar o código no github https://github.com/mfer/pinca-puca para acompanhar as versões e facilitar edição conjunta do código. Além disso criamos um quadro na ferramenta trello para acompanhar a evolução das entregas e entrevistas, e melhorar a nossa comunicação. Com esse objetivo criamos também um grupo no whatsapp. Reunimos presencialmente na UFMG e na casa do Vitor.
Acreditamos que a especificação de um dos tps, o do controlador da memória, estava bem difícil de ser compreendida. O fórum, no entanto, foi sempre bem útil para tirar dúvidas. Mas entendemos que isso deve ser alterado para semestres seguintes.
Possivelmente, aprendemos tarde demais que o gtkwave poderia ser usado para amplificar a nossa capacidade de desenvolver/debugar. Ao invés de usar apenas o tradicional imprimir textualmente valores de sinais com **$display** e **$monitor**.

5. Conclusão:

Foi interessante implementar do “zero” uma arquitetura de computador, foi difícil, pois, várias tecnologias eram novas para a maioria do grupo, que tiveram que correr para aprender pelo menos o básico. Mas junto com a dificuldade veio o aprendizado e a satisfação de ver algo sair da teoria e funcionar na prática (pelo menos o que funcionou).