Entrega 3 - Relatório

Grupo - Popullacho (Placa 4)

Alexandre Ladeira	212328
Gabriel Pallotta Cardenete	216392
Igor Fernando Mandello	236769
Júlia Alves De Arruda	238077
Lucas Hideki Carvalho Dinnouti	220792
Vinícius Waki Teles	257390

Organização

Nas últimas etapas havíamos nos divididos em 3 grupos para facilitar e acelerar o desenvolvimento, sendo o grupo A composto por Alexandre e Gabriel, grupo B por Igor e Vinícius, grupo C por Júlia e Lucas. Nessa etapa, os grupos A e B trabalharam mais próximos para juntar todos os componentes implementados anteriormente, especialmente os módulos de memória implementados pelo grupo B, e garantir que a execução estava correta na placa.

Grupo A e B

Responsabilidades:

- Otimizações para garantir que o circuito não ultrapassa os limites da FPGA, como implementação do módulo de divisão;
- Integração do controlador de memória e caches no circuito principal da CPU;
- Ajustes e consertos da lógica dos módulos de memória com a execução de programas de teste;
- Migração da sintetização para o Gowin, evitando instabilidades com a execução na placa;
- Início da implementação de instruções compactas.

Grupo C

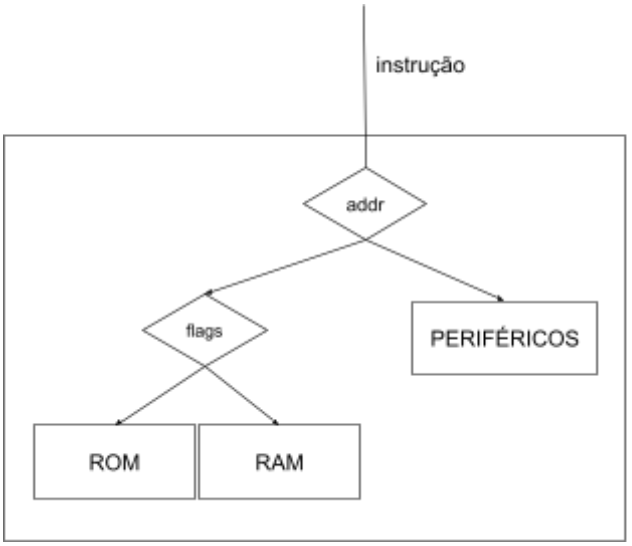
Responsabilidades:

- Integrar o módulo de periféricos ao controlador de memória;
- Passar a identificar instruções de periféricos pelo endereço de memória;
- Escolher um novo periférico para ser implementado;
- Integrar o novo periférico com a placa FPGA;
- Desenvolver um programa que utiliza todos os periféricos integrados e executar na placa;
- Demonstrar um código que utilize as instruções e os periféricos em FPGA.

Desenvolvimento do projeto

Periféricos

Anteriormente o módulo de periféricos estava completamente hardwired, significando que não era possível um programa acessá-lo através de instruções de load e write, sendo acessível somente pela CPU. Após a migração, o módulo passou a integrar o controlador de memória, que consegue verificar se a instrução está destinada a um periférico pelo endereço.



Além disso, os periféricos escolhidos para integrar o circuito foram dois LEDs, um vermelho e um verde. Esses foram usados junto aos sensores de luz para construírem um radar. Acendendo o LED vermelho quando a velocidade passa de um limite estabelecido, e o verde quando a velocidade medida pelos sensores está dentro do limite.

Módulo de divisão

Implementamos um módulo separado de divisão na ALU, que diminui consideravelmente o tamanho do circuito sintetizado, e permite que a CPU não exceda os limites físicos da FPGA.

Esse módulo de divisão realiza a operação em 32 ciclos, em cada ciclo é calculado um bit do resultado. Uma vantagem é que ele gera tanto o resultado da divisão como o resto, permitindo a substituição de ambas as operações na ALU. Além disso, também implementamos o suporte para divisão com números negativos, utilizando uma simples lógica de ajustar o sinal do resultado final dependendo dos sinais dos números de entrada.

Devido ao fato da operação levar mais de 1 ciclo, foi necessária a adição de uma flag a mais de stall, que permite a ALU segurar as instruções executando nas etapas de fetch e decode, e gerar uma bolha na etapa de memória. Assim que a operação é finalizada, a pipeline volta à sua execução normal.

Migração do sintetizador

Durante a integração dos módulos de memória com a pipeline da CPU implementada, tivemos diversos problemas de instabilidade com a execução na placa. Notamos que diversos módulos não estavam funcionando corretamente, e os programas que executávamos tinham saídas diferentes a cada execução, não correspondendo com a execução pelo simulador.

Decidimos então trocar o sintetizador Yosys (com a extensão Lushay Code no VSCode) para o sintetizador do Gowin, disponibilizado na Gowin IDE versão educacional. Também tivemos que utilizar o openFPGALoader para realizar o flash do programa na placa, pois tivemos problemas ao utilizar o Gowin Programmer. Além disso, devido a algumas peculiaridades da implementação do Gowin, foram necessárias algumas mudanças no nosso código, mas no geral a migração foi simples.

Com essa mudança, os problemas de instabilidade pararam, e percebemos que os resultados observados na placa correspondiam exatamente à execução com o simulador. Além disso, os tempos de síntese e roteamento diminuíram bastante, permitindo iterações mais rápidas de teste durante o desenvolvimento.

Instruções compactas

Iniciamos a implementação de instruções compactas, adicionando uma flag no decoder que identifica se a instrução atual é compacta, a partir dos bytes iniciais. Se a instrução é compacta, a decodificação é realizada utilizando as flags já existentes.

Além disso, para instruções compactas é necessário incrementar o PC em apenas 2 bytes, gerando um desalinhamento de memória na próxima leitura. Implementamos o suporte para leituras com alinhamento de 16 bytes no módulo da ROM.

Outro problema gerado pelo desalinhamento de memória é a necessidade de realizar a leitura de dois endereços na memória no caso de uma instrução normal que inicia em um endereço desalinhado. Para isso, foi implementado um novo módulo “fetch”, que é responsável por realizar duas leituras na ROM para esse cenário, gerando uma bolha de 1 ciclo na pipeline.

Para essa entrega ainda não terminamos a decodificação para todas as instruções compactas, que serão implementadas para a última entrega.

Lições aprendidas

Acabamos perdendo bastante tempo com os problemas de instabilidade na placa, e demoramos para considerar que a ferramenta de síntese poderia ser o problema.

Além disso, a experiência com a implementação do módulo de divisão na ALU mostrou a importância de considerar cuidadosamente o design dos circuitos durante o desenvolvimento para otimizar o uso de recursos da FPGA.