Entrega 2 - 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

Para facilitar e acelerar o desenvolvimento nos separamos em 3 grupos, sendo o grupo A composto por Alexandre e Gabriel, grupo B por Igor e Vinícius, grupo C por Júlia e Lucas.

Grupo A

Responsabilidades:

- Implementar as instruções que faltaram na primeira entrega (Store/Load e branches)
- Incrementar o conjunto de instruções para suportar RV32IMA
- Aumentar o pipeline para 5 estágios, com suporte às novas instruções e tratamento de hazards
- Consolidar o projeto em uma CPU única para execução no simulador e na FPGA

Nós desenvolvemos essas tarefas em dupla sempre (estilo pair programming compartilhando tela), utilizando o computador do Gabriel, pois foi onde configuramos todas as ferramentas para execução e teste do código. A única tarefa que foi feita apenas pelo Gabriel foi a configuração da FPGA em sua máquina para conseguirmos executar mais testes.

Grupo B

Responsabilidades:

- Implementar uma interface de memória ROM para a leitura das instruções e do programa.
- Implementar uma interface de memória RAM.
- Implementar e integrar Cache L1 para ambos tipos de memória do processador

O desenvolvimento foi feito em dupla, tanto remota quanto presencialmente, utilizando o notebook do Igor o qual já possuía as configurações necessárias.

Grupo C

O objetivo desse grupo foi implementar a integração do processador em FPGA com um periférico. Algumas das responsabilidades eram:

- Desacoplar estado os periféricos do módulo de registradores
- Adicionar como entrada do módulo de periféricos referências à memória.
- Mapear os LEDs embarcados em posições de memória.
- Mapear os sinais dos botões embarcados em posições de memória.
- Decidir a natureza de um periférico entregável (LCD, teclado, sensor) e preparar caminhos de dados que sejam compatíveis.
- Dados os protocolos necessários, implementar um Driver para a placa tang nano

Fizemos todas as tarefas presencialmente e em dupla, utilizando o computador pessoal do Lucas e os materiais que ambos tinham em casa para a construção do circuito.

Desenvolvimento do projeto

Pipeline de 5 estágios

Na primeira entrega, tínhamos 2 versões diferentes do módulo de CPU, uma para execução em simulador (mock), que não possuía pipeline, e uma para execução na FPGA, que foi implementada com um pipeline de 3 estágios.

Para essa entrega, abandonamos a CPU mock e expandimos a CPU com pipeline para 5 estágios, adicionando o módulo de memória de dados (que permitiu a implementação de loads e stores), o módulo de branch e módulo de atômico. Também atualizamos o testbench da CPU para permitir a execução da nova CPU no simulador.

Com essas mudanças, surgiram *data hazards*, e implementamos um módulo de *forwarding* para manter a consistência dos registradores em operações de leitura após escrita. Além disso, com a implementação de instruções de branch e jump, tivemos que adicionar um mecanismo de *stall* nas duas primeiras pipelines, de forma que permitisse a inserção de bolhas no caso de um salto ser realizado.

Expansão do conjunto de Instruções

Iniciamos essa etapa completando as instruções que faltaram na primeira entrega: para Store e Load o único ponto faltante foi conectar ao componente de memória, pois o decode já estava pronto e para os branches, fizemos o decode e criamos a branch unit.

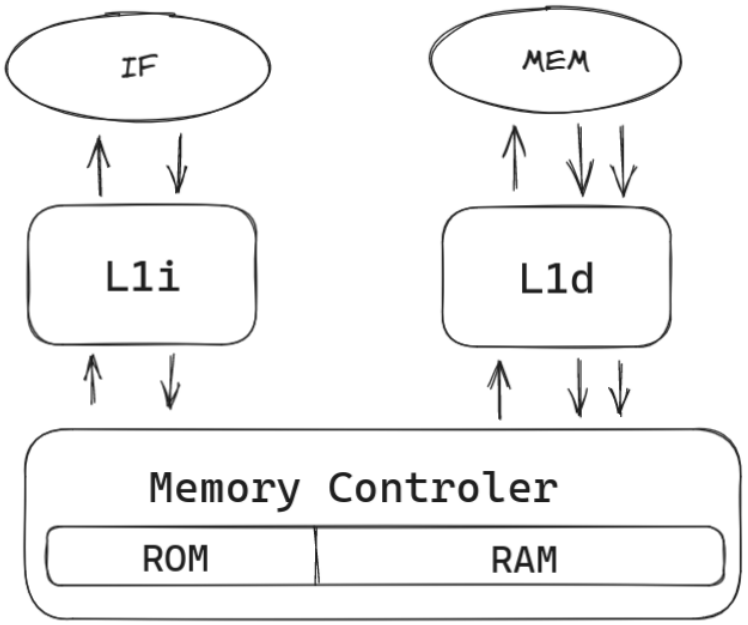
As instruções de multiplicação e divisão foram simples adições.

Para as instruções atômicas, criamos um Atomic Unit para conseguirmos lidar com o fluxo de leitura de registradores e escrita na memória. Ela possui 2 entradas para receber o dados da memória e de rs2 e uma entrada para definir a operação, retornando um result que é escrito na memória. A operação de leitura e escrita na memória é atualmente realizada no mesmo ciclo, mas no futuro talvez seja necessária a adição de *stalls*.

Memória e L1

Para o desenvolvimento da L1, também era necessário terminar a implementação da memória, que não havia sido concluída para a entrega anterior. Optamos, então, por fazer a implementação de ambos ao mesmo tempo, visto que isso nos permitiria ter uma maior liberdade no design dos módulos. Além disso, na primeira entrega, tínhamos 2 protótipos de memória: flash e array. Para essa entrega optamos pela memória em array por simplicidade de implementação e flexibilidade de mudanças.

Escolhemos um design em pilha onde a CPU tem acesso à um Memory Controller, que por sua vez tem acesso à L1, que finalmente tem acesso à Memória. Dessa maneira, todo acesso de memória passa pela L1. Porém após uma conversa com o professor percebemos algumas falhas e design e mudamos para o seguinte modelo:



Esse modelo consiste na comunicação direta dos estágios IF e MEM com suas respectivas L1s (L1 instruction, L1 data) e só então, caso necessário, as L1 se comunicam com o Memory Controller para buscar dados da memória principal. Note que a L1i tem fios a menos, isso porque o IF apenas lê da memória e nunca escreve.

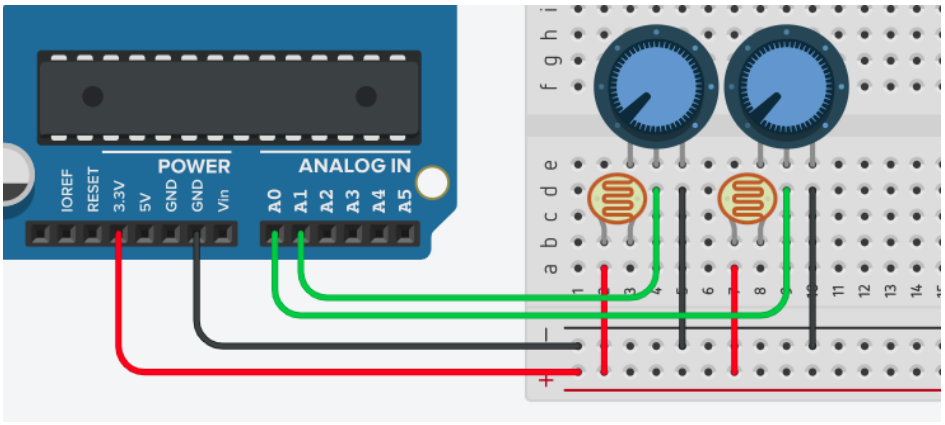
O Memory Controller por sua vez, faz a separação do que deve ser lido ou escrito, garantindo que apenas a ROM ou a RAM estará sendo acessada, já que elas são partes do mesmo componente físico. Com isso, também criamos outputs para o *stall* da L1i e L1d, sendo que optamos por manter a precedência na L1d.

Periféricos

Primeiramente, tratamos os elementos embarcados como dispositivos periféricos. Assim, mapeamos os LEDs e botões da placa em posições de memória que seriam então acessíveis por programas. Para fins de teste, fizemos com que um botão da placa alterasse uma posição da memória que estava associada a um LED, sendo o botão um dispositivo de entrada e o LED um de saída.

Com o mapeamento de memória feito, partimos para a integração de um periférico externo. Decidimos usar sensores de luz (LDR), que são dispositivos cuja resistência diminui quando o nível de luz aumenta.

Para conectar o sensor à placa, escolhemos um pino de entrada e ligamos ao módulo de periféricos, que faz o mapeamento de memória, associando o output dele a uma posição fixa. Para que funcionasse, montamos o circuito abaixo.



Pode-se observar que utilizamos a fonte de 3.3V da placa e um potenciômetro como resistor. O potenciômetro foi utilizado para regular a resistência de acordo com a luminosidade do ambiente, mantemos ele pois não tínhamos os resistores ideais para atingir o valor desejado depois da regulagem.

Essa regulagem é necessária pois estamos trabalhando com valores analógicos de tensão que são então interpretados pela placa como um valor digital binário. Sendo assim, ajustamos o valor do potenciômetro para que a tensão em luminosidade baixa seja interpretada como 0 pela placa, e em luminosidade alta seja interpretada como 1.

O driver para esses periféricos foi implementado no módulo *peripherals*. Esse módulo possui como entrada os fios que levam aos periféricos e as posições de memória que devem ser lidas e escritas nos periféricos, e como saída possui os valores de output a serem escritos na memória. Esses valores de write back são passados para o módulo da memória e preenchidos no final do clock. Um exemplo prático é que para ligar um LED, lemos os valores de entrada e escrevemos nele, mas para ler o valor do sensor, escrevemos na variável de saída que leva a memória.

Utilizamos dois sensores de luz para que conseguíssemos medir a velocidade de um objeto utilizando o tempo que o mesmo passa por eles e a distância entre eles. Porém o programa que faz esse cálculo e acessa o output dos sensores da memória ainda não foi implementado, esse é um dos próximos objetivos dessa frente.

Lições aprendidas

Os *data hazards* introduzidos nos demandaram bastante dedicação para entender como solucioná-los e conseguimos aplicar soluções, como os mecanismos de *stall* e *forwarding* que foram interessantes de estudar.

Além disso, essa entrega contou com um aumento significativo no número de componentes no processador, se tornando um desafio maior fazer a síntese dele na placa. Enfrentamos diversos problemas de limite de células na FPGA e de circuitos com clock

inválido, sendo que para a próxima entrega possivelmente teremos que realizar otimizações para incluir novas funcionalidades.

Melhoramos nossa experiência no uso da extensão DigitalJS como forma de *debugging*, quebrando o programa em partes mais simples, testando elas separadamente para reduzir a complexidade do teste, e ter certeza que as unidades funcionam.