



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 17

Execução Superscalar

1. INTRODUÇÃO

Como já discutimos, o processador SweRV EH1 da Western Digital é um núcleo **superescalar de 2 vias de 32 bits** com pipeline de 9 andares. Nos Laboratórios 11-13, analisámos o fluxo das instruções básicas através do processador SweRV e os detalhes de cada um dos andares do pipeline, e nos Labs 14-16 analisámos a forma como os dados, o controlo e as dependências estruturais são tratadas neste processador. Agora, com esse conhecimento básico, você está pronto para analisar a execução superescalar!

NOTA: Antes de iniciar este laboratório, recomendamos a leitura da Secção 7.7.4 do livro de S. Harris e D. Harris, "*Digital Design and Computer Architecture: RISC-V Edition*", Morgan Kaufmann [DDCARV]. Parte do conteúdo deste laboratório é inspirado nesse livro.

A execução superescalar é uma técnica de microarquitetura que melhora o desempenho de um processador. Um processador superescalar contém várias cópias do hardware da via de dados para executar várias instruções em simultâneo. A latência de execução de uma única instrução permanece inalterada, mas o processador pode executar e concluir mais instruções por ciclo, melhorando assim o seu rendimento. Figura 1 mostra um exemplo de diagrama de blocos de um processador superescalar de duas vias extraído de [DDCARV].

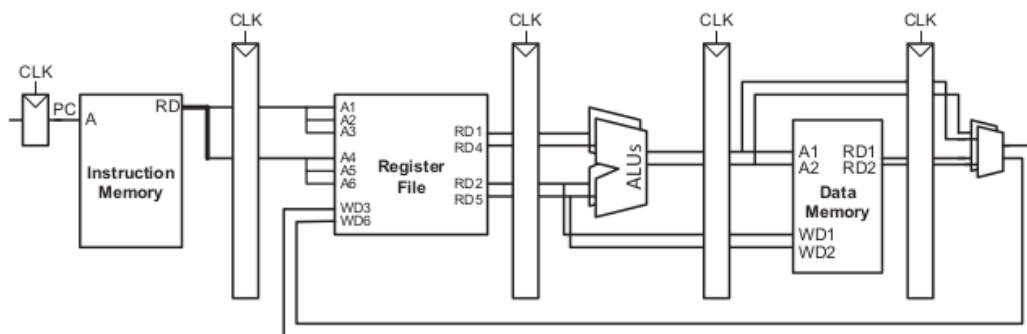


Figura 1 . Figura 7.68 de [DDCARV]: Diagrama de blocos do processador superescalar de duas vias

O SweRV EH1 é um processador superescalar de 2 vias semelhante ao mostrado na Figura 1 que pode ir buscar, executar e concluir até duas instruções por ciclo. O caminho de dados prepara duas instruções de cada vez que vêm da memória de instruções. O Register File multi-portal pode ler até quatro operandos de origem e escrever dois valores de volta em cada ciclo (mais um valor proveniente de uma leitura não-bloqueante, como analisámos no Lab 15). O processador SweRV EH1 contém dois Pipes de Inteiros, um Pipe Multiplicador, um Pipe Load-Store e um Divisor não-pipelined. Todos estes pipes são completamente independentes, pelo que um par de instruções aritméticas-lógicas (AL) independentes ou qualquer par de duas instruções diferentes independentes pode ser executado em simultâneo. No entanto, tal como discutido no Laboratório 14, um par de multiplicações, divisões ou instruções load/store não pode ser executado no mesmo ciclo, porque só existe um pipe de cada no processador, pelo que duas instruções sequenciais iguais não AL conduzirão a um conflito estrutural.

O SweRV EH1 não inclui suporte para programação dinâmica de instruções com execução fora-de-ordem, exceto no caso de leituras não-bloqueantes. No entanto, é possível reordenar estaticamente o código de modo a explorar melhor os recursos, incluindo as duas vias do pipeline. Idealmente, num processador superescalar de 2 vias como o SweRV EH1, a taxa de transferência (IPC) duplicaria quando comparada com um de uma só via.

Infelizmente, os programas reais não costumam atingir esse valor óptimo: em programas reais, o desempenho normalmente melhora em 1,3x-1,5x ao passar de processadores de 1 para 2 vias; no entanto, adicionar a segunda via requer muito mais hardware.

Na Secção 2, analisamos dois programas simples, comparando o comportamento quando se utilizam configurações de emissão única e de emissão dupla do SweRV EH1. Depois, na Secção 3, propomos vários exercícios relacionados com a execução superescalar.

2. SINGLE-ISSUE VS. DUAL-ISSUE

Nesta secção, trabalhamos com dois programas simples: o primeiro (Secção 2.A) contém um ciclo com quatro instruções AL e o segundo (Secção 2.B) contém um ciclo com duas multiplicações intercaladas com duas instruções AL.

A. Quatro instruções AL independentes

Nesta secção, comparamos o desempenho do programa de Figura 2 executado no núcleo SweRV EH1 *single-issue* ou *dual-issue*. Lembre-se de que, no Apêndice B do Laboratório 11, descrevemos como os diferentes recursos do núcleo (execução em pipeline, previsão de saltos, superescalar, etc.) podem ser configurados.

O programa contém um ciclo que efetua 1.000.000 (0xF4240) iterações; o corpo do ciclo contém quatro instruções independentes de A-L (*add*, *sub*, *or* e *xor*), rodeadas por instruções *nop* que nos permitem ver cada iteração isolada das outras. A pasta `[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions` fornece o projeto PlatformIO que pode ser analisado, simulado e modificado conforme desejado.

```
.globl Test_Assembly

.text

Test_Assembly:

li t2, 0x400          # Desativar a Execução de Dual-Issue
csrrs t1, 0x7F9, t2

li t0, 0x0
li t1, 0x1
li t2, 0x1
li t3, 0x3
li t4, 0x4
li t5, 0x5
li t6, 0x6

lui t2, 0xF4
add t2, t2, 0x240

REPEAT:
    add t0, t0, 1
    INSERT_NOPS_10
    INSERT_NOPS_4
    add t3, t3, t1
    sub t4, t4, t1
    or  t5, t5, t1
    xor t6, t6, t1
    INSERT_NOPS_10
    INSERT_NOPS_3
```

```
bne t0, t2, REPEAT # Repeat the loop
.end
```

Figura 2 . Programa com quatro instruções A-L

Na secção 2.A.i, analisamos a simulação no Verilator e a execução na placa Nexys A7 do programa da Figura 2 num processador SweRV EH1 de uma só emissão. Para o efeito, desativamos a capacidade de *dual-issue* utilizando as duas instruções seguintes no início do programa:

```
li t2, 0x400
csrrs t1, 0x7F9, t2
```

Na secção 2.A.ii, analisamos a simulação no Verilator e a execução na placa Nexys A7 do programa da Figura 2 num processador SweRV EH1 **dual-issue**. Para o fazer, basta comentar as duas instruções anteriores.

i. Execução num processador SweRV EH1 single-issue

Na sua configuração de emissão única, o SweRV EH1 ignora simplesmente a segunda via, executando apenas uma instrução por ciclo. Figura 3 ilustra a simulação do programa da Figura 2 nesta configuração do SweRV EH1. Escolhemos uma iteração aleatória do ciclo *REPEAT* (exceto a primeira), dado que todas as iterações são iguais.

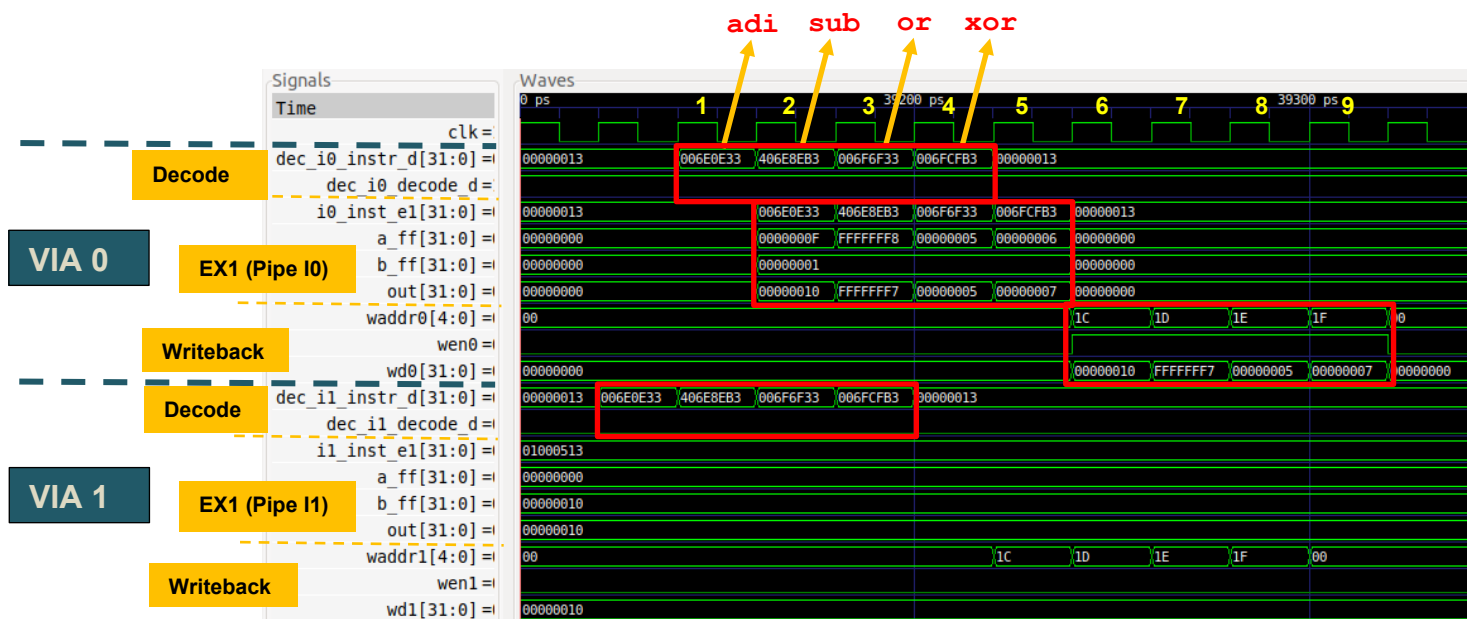


Figura 3 . Simulação do programa da Figura 2 num SweRV EH1 de uma só emissão

As instruções são recebidas em ambas as vias em tempo de descodificação (ver sinais `dec_i0_instr_d[31:0]` e `dec_i1_instr_d[31:0]`), mas só são enviadas para execução na Via 0, porque a Via 1 está desativada. Os sinais `dec_i0_decode_d` e `dec_i1_decode_d` determinam se a instrução é propagada do andar Decode para a Etapa EX1, e os sinais `i0_inst_e1[31:0]` e `i1_inst_e1[31:0]` contêm a instrução na Via 0 e na Via 1, respetivamente, no andar E1.

- **Via 0:**
 - o O sinal `dec_i0_decode_d` é sempre 1 no nosso exemplo;

especificamente, é 1 para as quatro instruções A-L em análise.

- A instrução no andar de decodificação (`dec_i0_instr_d[31:0]`) é **propagada** para o Pipe I0 (`i0_inst_e1[31:0]`)

- **Via 1:**

- O sinal `dec_i1_decode_d` é sempre 0 no nosso exemplo; especificamente, é 0 para as quatro instruções AL em análise.
- A instrução no andar de decodificação (`dec_i1_instr_d[31:0]`) **NÃO é propagada** (`i1_inst_e1[31:0]`) para o andar Execute.

Por conseguinte, apenas a ALU do Pipe I0 é utilizada (ver sinais `aff`, `bff` e `out` em ambos os sentidos) e apenas o porto de escrita 0 do Register File é utilizado (ver sinais `waddr`, `wen` e `wd` em ambos os sentidos).

Figura 4 ilustra o fluxo das quatro instruções A-L através do Pipe I0, desde o Decode até ao Writeback (EX5). Mostra os nove ciclos (1 a 9) especificados na Figura 3. Os buracos vazios correspondem às instruções `nop` que rodeiam as quatro instruções A-L, que não são mostradas na figura por uma questão de simplicidade.

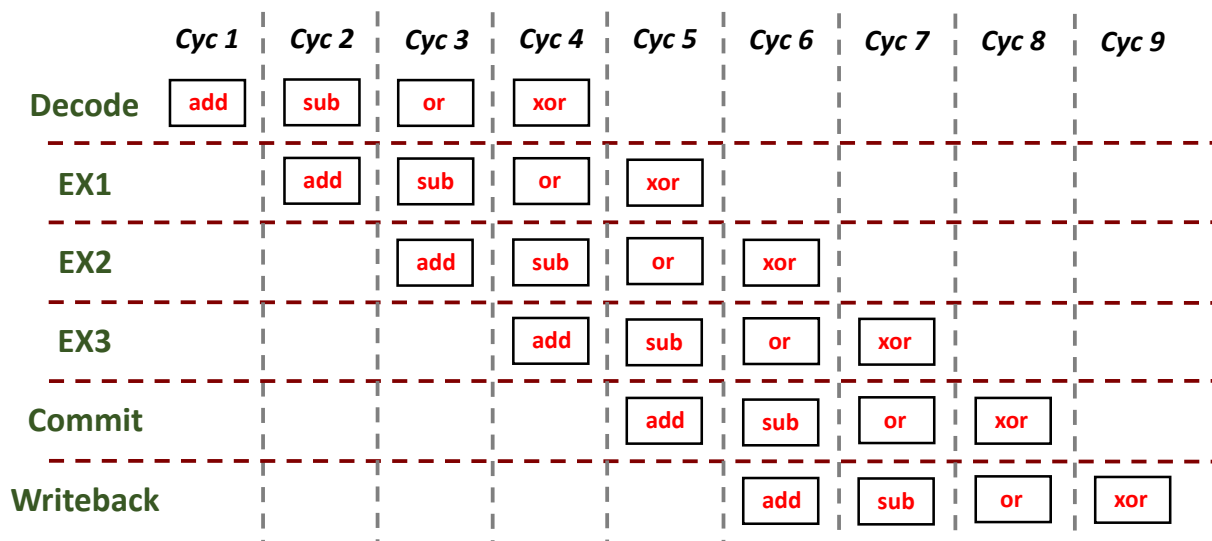


Figura 4 . Fluxo das 4 instruções A-L através do Pipe I0

TAREFA: Na simulação da Figura 3 inclua os sinais de *trace* e destaque as instruções à medida que elas passam pelo pipeline desde o andar de Decode até o andar de Writeback, de forma semelhante à Figura 4. Pode utilizar o ficheiro `.tcl` fornecido em: `[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions/test_task1.tcl`.

TAREFA: Remover todas as instruções `nop` dentro do corpo do ciclo da Figura 2. Repetir a simulação da Figura 3. Qual é o IPC esperado para este programa? Executar o programa na placa e verificar se o IPC obtido é o esperado.

Figura 6 ilustra o fluxo das quatro instruções AL através do Pipe I0 e do Pipe I1, desde a fase de decodificação até à fase EX5. Mostra os sete ciclos (1 a 7) especificados na Figura 5. Os buracos vazios correspondem às instruções nop que rodeiam as quatro instruções A-L, que não são mostradas na figura por uma questão de simplicidade.

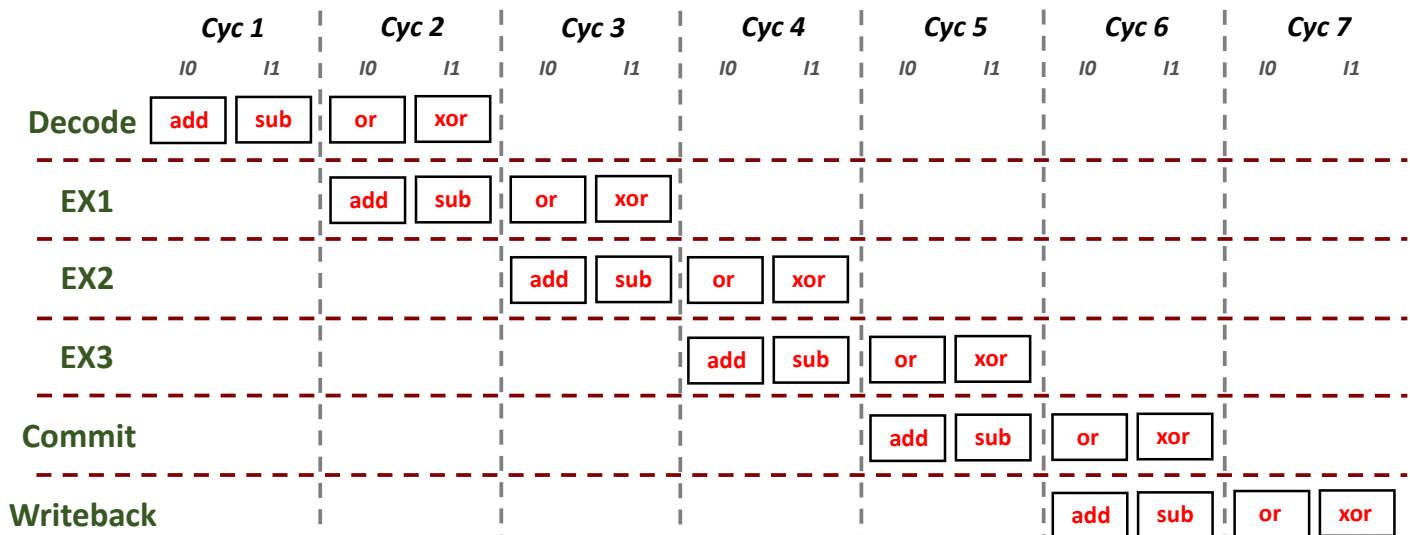


Figura 6 . Fluxo das 4 instruções através de ambos os Pipes (I0 e I1)

TAREFA: Na simulação da Figura 5 adicione sinais de *trace* e destaque as instruções à medida que elas passam pelo pipeline, desde os estágios de Decodificação até Writeback, semelhante ao que é mostrado na Figura 6. Você pode usar o arquivo *.tcl* fornecido em: [\[RVfpgaPath\]/RVfpga/Labs/Lab17/Four_AL_Instructions/test_task2.tcl](#).

TAREFA: Remover todas as instruções *nop* dentro do corpo do ciclo da Figura 2. Repetir a simulação da Figura 5. Qual é o IPC esperado para este programa? Executar o programa na placa e verificar se o IPC obtido é o esperado.

B. Duas instruções *mul* intercaladas com duas instruções A-L

Nesta secção, analisamos o programa da Figura 7 num núcleo SweRV EH1 *dual-issue*. O programa executa um ciclo que efetua 1.000.000 de iterações; o corpo do ciclo contém duas instruções *mul* intercaladas com duas instruções A-L (*mul*, *add*, *mul* e *sub*), rodeadas por instruções *nop* que nos permitem ver cada iteração isolada das outras. As quatro instruções são independentes. Analisamos a simulação no Verilator, destacando os principais sinais e fornecendo uma breve explicação do comportamento do programa em cada caso. A pasta [\[RVfpgaPath\]/RVfpga/Labs/Lab17/TwoAL_TwoMUL_Instructions](#) fornece o projeto PlatformIO que pode ser analisado, simulado e modificado como desejar.

```
.globl Test_Assembly

.text

Test_Assembly:
```

```
# li t2, 0x400          # Desativar a Execução em Dual-Issue
# csrrs t1, 0x7F9, t2

li t3, 0x3
li t4, 0x4
li t5, 0x5
li t6, 0x6
li t0, 0x0
lui t1, 0xF4
add t1, t1, 0x240

REPEAT:
    add t0, t0, 1
    INSERT_NOPS_10
    INSERT_NOPS_4
    mul t3, t3, t1
    add t4, t4, t1
    mul t5, t5, t1
    sub t6, t6, t1
    INSERT_NOPS_10
    INSERT_NOPS_3
    bne t0, t1, REPEAT # Repete o ciclo
.end
```

Figura 7 . Programa com 2 instruções mul e 2 instruções A-L

Neste programa, o processador enviará para execução uma instrução `mul` e uma instrução A-L por ciclo, pelo que é utilizado o Pipe Multiplicação, bem como o Pipe I0 ou I1. Figura 8 ilustra a simulação do programa da Figura 7 no processador superescalar de 2 vias SweRV EH1. Escolhemos uma iteração aleatória do ciclo *REPEAT* (exceto a primeira), dado que todas as iterações são iguais.

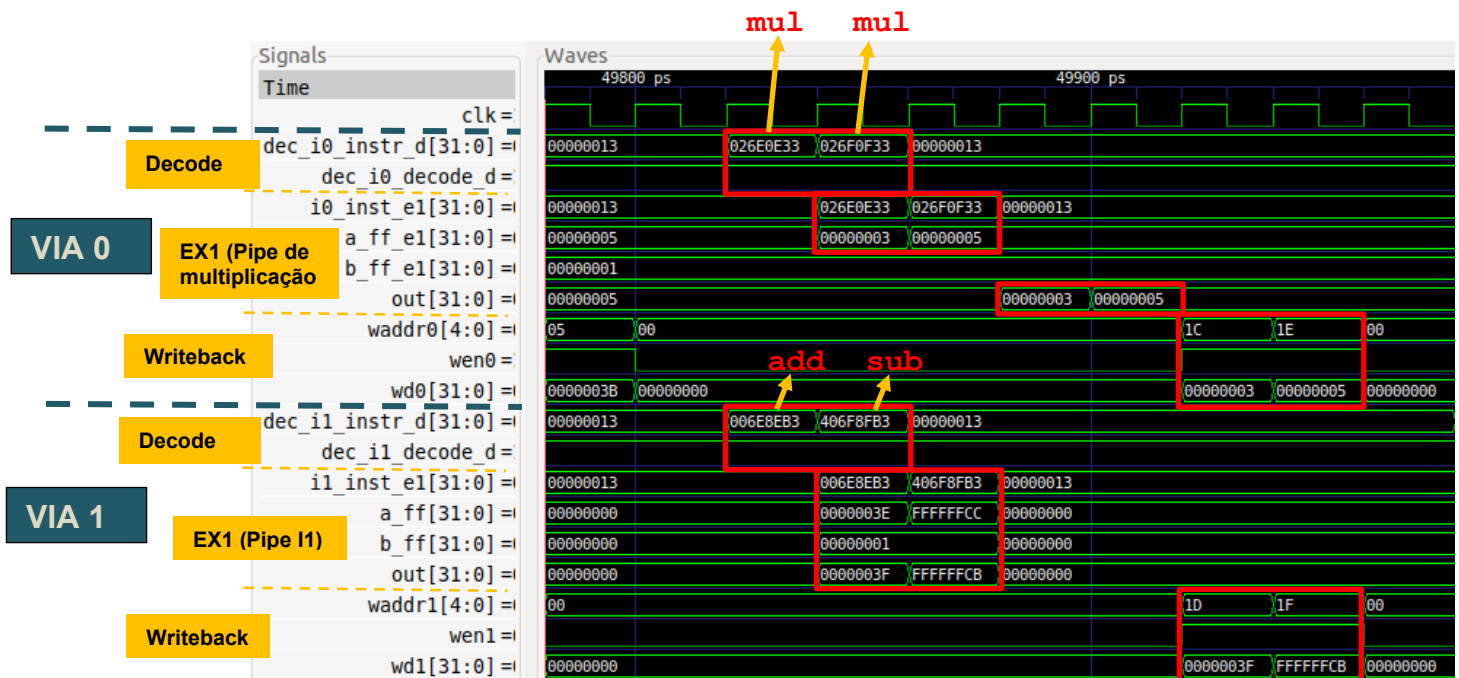


Figura 8 . Simulação do programa da Figura 7

As instruções são recebidas de ambas as vias no momento da decodificação (ver sinais `dec_i0_instr_d[31:0]` e `dec_i1_instr_d[31:0]`) e são enviadas para as fases de execução de ambas as vias.

- **Via 0:**
 - o O sinal `dec_i0_decode_d` é sempre 1 - para duas das quatro instruções analisadas no nosso exemplo (as outras duas instruções são decodificadas na Via 1).
 - o A instrução no andar de Decode (`dec_i0_instr_d[31:0]`) é propagada para o Pipe de multiplicação (`i0_inst_e1[31:0]`)
- **Via 1:**
 - o O sinal `dec_i1_decode_d` é sempre 1 - para duas das quatro instruções analisadas no nosso exemplo (as outras duas instruções são decodificadas na Via 1).
 - o A instrução no DECO (`dec_i1_instr_d[31:0]`) é propagada para o Pipe I1 (`i1_inst_e1[31:0]`)

Assim, a ALU do Pipe I1 e o Multiplicador são utilizados (ver sinais `a_ff_e1`, `b_ff_e1`, `e_out` e sinais `a_ff`, `b_ff`, `e_out`), e ambas as portas de escrita do Register File são utilizadas (ver sinais `waddr`, `wen`, e `wd` em ambas as vias).

TAREFA: Na simulação da Figura 8 adicione sinais de rastreamento e destaque as instruções à medida que elas passam pelo pipeline, desde os andares de Decodificação até Writeback. Pode utilizar o ficheiro `.tcl` fornecido em:
`[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL_TwoMUL_Instructions/test_taskMuls.tcl`.

TAREFA: Remover todas as instruções `nop` dentro do corpo do ciclo da Figura 7. Repetir a simulação da Figura 8. Qual é o IPC esperado para este programa? Executar o programa na placa e verificar se o IPC obtido é o esperado.

Repita as mesmas experiências para a configuração de *single-issue* e compare os resultados.

3. EXERCÍCIOS

- 1) Criar programas semelhantes aos das Figura 2 e Figura 7 usando combinações de instruções que mostrem novas situações relacionadas com a execução *dual-issue*.
- 2) Analise as diferenças entre o processador SweRV EH1 (*dual-issue*) e o exemplo de processador superescalar proposto na Secção 7.7.4 do livro de S. Harris e D. Harris, "Digital Design and Computer Architecture: RISC-V Edition" [DDCARV] (apresentado na Figura 1 por conveniência).
- 3) Analise o programa da Figura 7.70 na Secção 7.7.4 do DDCARV, que é fornecido num projeto PlatformIO na pasta `[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_SuperscalarExample`. Execute o programa no SweRV EH1, tanto em simulação como na placa (para esta última, remova as instruções `nop`). Explique os resultados. Se necessário, reordene o programa tentando obter o IPC ótimo.

Em seguida, desative a execução *dual-issue* conforme explicado neste laboratório - e em SweRVref.docx (Secção 2). Compare a simulação e os resultados obtidos na placa quando o recurso *dual-issue* está ativado.

- 4) Modifique o programa do Exercício 3 substituindo a instrução `add s9, s8, t1` pela instrução `add t2, s8, t1`. Explique os resultados. Se necessário, reordene o programa para tentar obter o IPC ótimo.

Em seguida, desative a execução de *dual-issue* conforme explicado neste laboratório e em SweRVref.docx (Secção 2). Compare a simulação e os resultados obtidos na placa com os obtidos quando o recurso *dual-issue* está ativado.

- 5) (O exercício seguinte é baseado no exercício 4.31 do livro "Computer Organization and Design - RISC-V Edition", de Patterson & Hennessy ([HePa]).

Neste exercício, comparamos o desempenho de processadores de *single-issue* e *dual-issue*, levando em consideração as transformações de programa que podem ser feitas para otimizar a execução *dual-issue*. Os problemas deste exercício referem-se ao seguinte ciclo (escrito em C):

```
for(i=0; i!=j; i+=2) b[i]=a[i]-a[i+1];
```

Um compilador com pouca ou nenhuma otimização pode produzir o seguinte código Assembly RISC-V:

```
li x12, 0
li x13, 8000
li x14, 0
TOP:
    slli x5, x12, 2
    add x6, x10, x5
    lw x7, 0(x6)
    lw x29, 4(x6)
    sub x30, x7, x29
    add x31, x11, x5
    sw x30, 0(x31)
    addi x12, x12, 2
    ENT: bne x12, x13, TOP
```

Este código utiliza os seguintes registos:

i	j	a	b	Temporary values
x12	x13	x10	x11	x5–x7, x29–x31

Este código é fornecido em

`[RVfpgaPath]/RVfpga/Labs/Lab17/PaHe_SuperscalarExample` com algumas pequenas modificações em relação ao código fornecido pelo exercício do livro que não afetam o comportamento do programa:

- O registo `x13` é inicializado a 8000, para que o ciclo execute 4000 iterações.
- A instrução `jal` é removida.
- As instruções `ld` e `sd` são substituídas pelas instruções `lw` e `sw`. Isto implica a alteração dos acessos de 4 para 8 bytes de largura.

Suponha um processador *dual-issue*, estaticamente programado, que tem as seguintes propriedades:

1. Uma instrução deve ser uma operação de memória; a outra deve ser uma instrução aritmética-lógica ou um salto.
2. O processador tem todos os caminhos de Forwarding possíveis entre os andares.
3. O processador tem uma previsão de saltos perfeita.
4. Duas instruções não podem ser enviadas em conjunto se uma depender da outra.
5. Se for necessário parar o pipeline (*stall*), ambas as instruções de um andar devem parar.

- a) Compare as propriedades deste processador de exemplo e as propriedades do processador SweRV EH1.
- b) Desenhe um diagrama de pipeline e uma simulação que mostre como uma iteração aleatória do ciclo (exceto a primeira) do código RISC-V dado acima é executada no processador SweRV EH1 *dual-issue*. Suponha que o ciclo termina após quatro mil iterações (é o caso no código acima).
- c) Qual é o aumento de velocidade ao passar de um processador SweRV EH1 *single-issue* para um processador SweRV EH1 *dual-issue*? Explique os resultados. Teste o programa na placa e active/desactive a execução *dual-issue*.
- d) Reorganize/reescreva o código RISC-V dado acima para obter um melhor desempenho no processador SweRV EH1 *dual-issue*. (No entanto, não desenrole o ciclo).
- e) Agora, desenrole o código RISC-V de forma que cada iteração do ciclo desenrolado manipule duas iterações do ciclo original. Em seguida, reorganize/reescreva seu código desenrolado para obter um melhor desempenho no processador SweRV EH1 *dual-issue*.

6) (O exercício seguinte baseia-se nos exercícios 7.30, 7.32 e 7.34 do Capítulo 7 da DDCARV).

Suponha que o processador SweRV EH1 está a executar o seguinte trecho de código. Recorde-se que o SweRV EH1 tem uma unidade de conflitos (Hazard Unit). Pode assumir um sistema de memória que devolve o resultado num ciclo (para esse efeito, utilizamos a DCCM e inserimos o excerto de código num ciclo e evitamos a primeira iteração para que não haja *miss* na l\$).

```
addi s1, t0, 11    # t0 contém o endereço base do DCCM
lw    s2, 25(s1)
lw    s5, 16(s2)
add   s3, s2, s5
or    s4, s3, t4
and   s2, s3, s4
```

- a) Simular o programa com o Verilator e o GTKWave. Analise os resultados e, para cada ciclo, especifique:
 - * Que instruções são decodificadas, enviadas para execução e concluídas?
 - * Que registos estão a ser escritos e quais estão a ser lidos?
 - * Que encaminhamentos e bloqueios ocorrem?
- b) Qual é o IPC do processador neste programa? Primeiro responda teoricamente e depois confirme a sua resposta executando o programa na placa.
- c) Efetue a mesma análise no processador *single-issue* e compare os resultados com

os resultados do processador *dual-issue*.

Um projeto PlatformIO é fornecido em:

[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_Exercises-30-32-34. O programa em análise é inserido num ciclo para que seja mais fácil de compreender na simulação (qualquer iteração, exceto a primeira, pode ser utilizada para análise) e pode ser medido utilizando contadores de desempenho.

7) (O exercício seguinte baseia-se nos exercícios 7.31, 7.33 e 7.35 do Capítulo 7 da *DDCARV*).

Repita o exercício 7 para o seguinte trecho de código.

```
addi s1, t0, 52
addi s0, s1, -4
lw s3, 16(s0)
sw s3, 20(s0)
xor s2, s0, s3
or s2, s2, s3
```

Um projeto PlatformIO é fornecido em:

[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_Exercises-31-33-35.