Arq1-ListaExercícios-2

Arquitetura e Organização de Computadores 1 LISTA DE EXERCÍCIOS 2

junho/2025

Prof. Marcio Merino Fernandes

DC-UFSCAR – BCC/EnC

RISC-V: Microarquitetura **Monociclo**

1-Considere o **datapath monociclo** da figura abaixo e detalhamento da unidade de controle correspondente. Suponha que um dos seguintes sinais de controle na versão monociclo do processador RISC-V tem uma falha, estando **travado sempre com o sinal = 0.** Quais instruções funcionariam mal? Por quê ?

RegWrite

O RegWrite é o sinal de controle responsável por escrever valores no arquivo de registradores, como ele está travado em 0 não será possível escrever nada no registrador, logo instruções que necessitam da escrita no processador seriam afetadas, como: add, sub, or, and, slt, lw, jal. Instruções como sw funcionam pois escrevem valores na memória de dados e não no arquivo de registradores.

ALUOp

ALUOp é responsável por definir a operação a ser executada na ALU. AluOp está travada em 000 que corresponde a operação de soma, logo todas as outras operações que necessitam da ULA não funcionariam corretamente como sub, or, and, slt

MemWrite

MemWrite é o sinal de controle que habilita a escrita na memória de dados, com ele sendo 0, não é possível realizar nenhuma escrita. A instrução afetada é sw

ImmSrc

Com o immSrc sempre 0 o processador interpreta como se fosse uma instrução do tipo I, logo as outras instruções que necessitam do immextend e que não sejam do tipo I são afetadas. Como sw e beq. As duas instruções que terminam com 1 não funcionam

ResultSrc

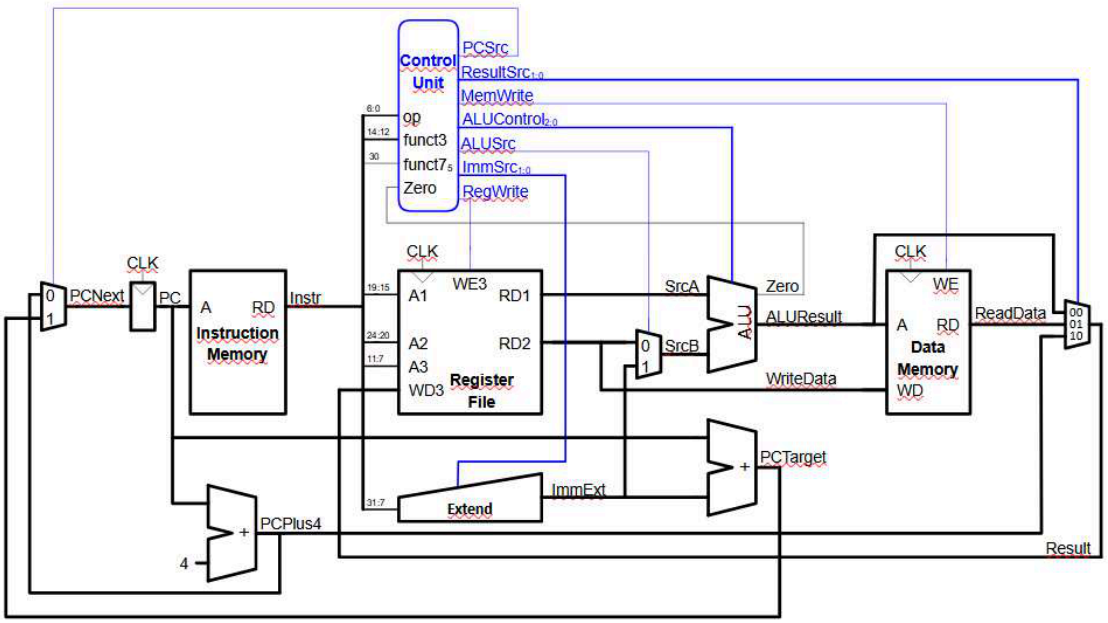
ResultSrc é o sinal de controle que seleciona qual fonte para gravar no registrador de destino, com ele sendo 0 ele irá sempre selecionar ALUResult e instruções que passam pela memória de dados ou pelo que gravam PC + 4 não funcionarão corretamente.

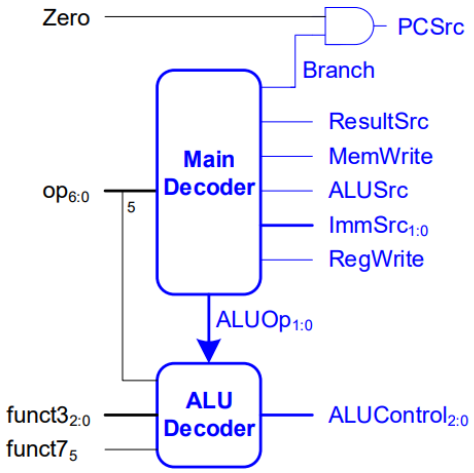
PCSrc

PCSrc é o sinal de controle que seleciona qual será a próxima instrução, ela pode ser PC + 4 (PCScr = 0) ou um salto PCTarget (PCSrc = 1). Com o sinal travado em 0, não será possível realizar instruções de salto/branch como beq

ALUSrc

ALUSrc é o sinal de controle que seleciona a segunda entrada na ULA que pode ser rd2 ou immExt. Instruções do tipo I usam o rd2 pois realizam operações entre dois registradores, já operações do tipo I ou load/store usam immExt como valor ou deslocamento imediato. Com o sinal travado em 0, instruções do tipo I e load/sw não funcionarão corretamente.





2- Modifique o processador RISC-V de ciclo único abaixo para implementar uma das seguintes instruções. Anote as modificações necessárias no diagrama da figura, incluindo novos sinais de controle, caso necessário. Faça também as alterações de controle necessárias na tabelas abaixo, de modo a incluir as mudanças necessárias no decodificador principal e no decodificador da ALU.

XOR

O xor já é uma operação do tipo R que necessita de dois registradores logo ele segue o caminho normal sem nenhuma grande alteração no datapath.

No entanto um passo importante é adição da nova instrução na ULA. Para isso, é necessário inicialmente identificar funct7 e funct3 correspondentes a xor no manual do risc-v, após isso dentro da unidade de controle, mais especificamente no decodificador da ula devemos adicionar os novos valores de funct7 e funct3 e atribuir um valor para ALUControl.

Após isso, dentro da ULA devemos implementar a operação do xor relacionada ao sinal definido anteriormente da alucontrol

SLL

SLL é uma instrução do tipo R que também faz uso de 3 registradores rs1, rs2, rd, logo seu datapath continuará o mesmo. No entanto para funcionamento correto é necessário seguir os passos abaixo.

Adicionar no decodificador da ULA uma nova linha associando funct7 = 0000000 e funct3 = 001 de sll a uma valor de ALUControl

Após isso, dentro da ULA é necessário relacionar o código da ALUControl com a operação. Por exemplo: em verilog usaríamos >> para o deslocamento.

bne

BNE é uma instrução do tipo B que pode seguir o mesmo caminho que beq que já está implementado. Logo vamos reutilizar o caminho de beq

Dentro da unidade de controle é necessário adicionar a identificação da instrução no decodificador principal, como o opcode de ambas é igual a diferença será feita por funct3 = 001. Assim como em beq, o controle precisa ser configurado de modo ativar a comparação na ULA, habilitar o uso do deslocamento imediato e selecionar o valor correto para o PC.

No decodificador da ULA a operação continua sendo de subtração, logo ALUControl = subtração

No controle de desvio, o resultado dependerá do resultado da ULA, se for zero (ou seja diferentes) faz o desvio.

A coluna branch deverá ter dois bits para saber qual a instrução está sendo executada OU criar um campo novo (flag para identificar se é bne ou beq)

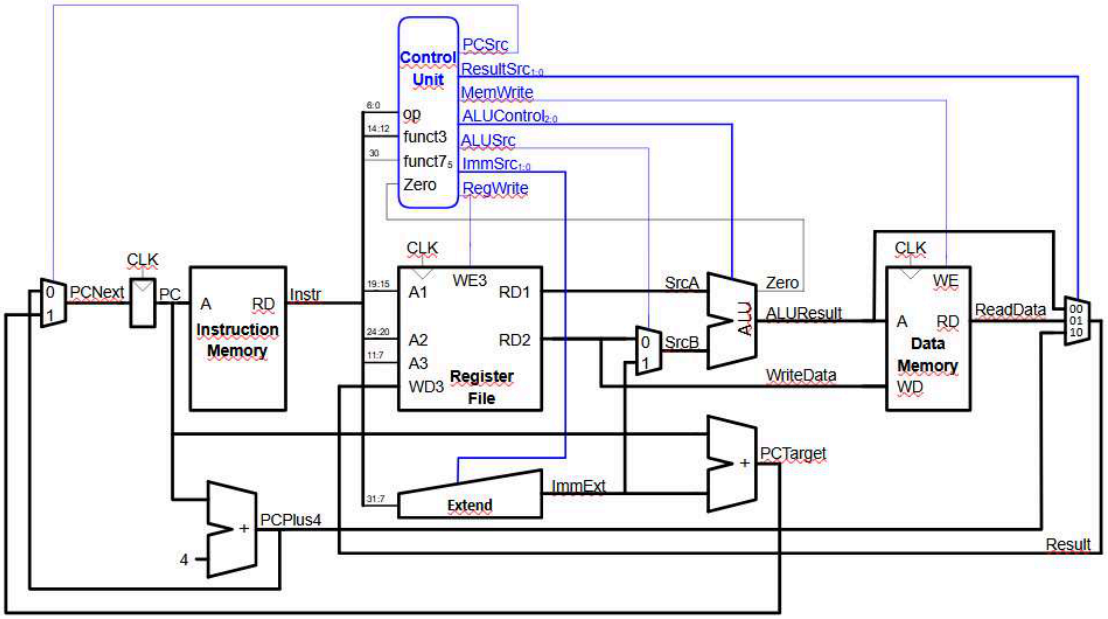


Table 7.6 Main Decoder truth table enhanced to support `jal`

Instruction	Opcode	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
`lw`	0000011	1	00	1	0	01	0	00	0
`sw`	0100011	0	01	1	1	xx	0	00	0
R-type	0110011	1	xx	0	0	00	0	10	0
`beq`	1100011	0	10	0	0	xx	1	01	0
I-type ALU	0010011	1	00	1	0	00	0	10	0
`jal`	1101111	1	11	x	0	10	0	xx	1

Table 7.3 ALU Decoder truth table

ALUOp	funct3	{op₅, funct7₅}	ALUControl	Instruction
00	x	x	000 (add)	`lw, sw`
01	x	x	001 (subtract)	`beq`
10	000	00, 01, 10	000 (add)	`add`
	000	11	001 (subtract)	`sub`
	010	x	101 (set less than)	`slt`
	110	x	011 (or)	`or`
	111	x	010 (and)	`and`

RISC-V: Microarquitetura **Multiciclo**

3- Considere o **datapath multiciclo** da Figura abaixo e detalhamento da unidade de controle correspondente. Suponha que um dos seguintes sinais de controle na versão multiciclo do processador RISC-V tem uma falha**, estando travado sempre com o sinal = 0.** Quais instruções funcionariam mal? Por que?

ResultSrc

Com o ResultSrc travado em 0 apenas operações do tipo R que tem a saída em ALUResult funcionariam, porém instruções como lw que deve gravar o valor da memória não funcionará corretamente.

ALUSrcB

ALUSrcB controla qual valor será o segundo operando da ULA, com ele travado em 0, instruções que precisam somar imediato ou calcular endereço de memória (ex: lw, sw, beq) não funcionarão corretamente, pois apenas valores vindos de rd2 serão utilizados, ou seja, apenas instruções do tipo R funcionarão corretamente.

ALUSrcA

ALUSrcA controla qual valor será o primeiro operando da ULA, travando em 0 instruções do tipo R que necessitam de dois registradores não funcionarão.

ImmSrc

Assim como no processador monociclo, o immsrc seleciona a forma de montar o imediato. Com ele travado em 0 apenas instruções do tipo I funcionarão corretamente, já instruções do tipo S e não funcionarão, elas usarão imediatos incorretos.

RegWrite

RegWrite controla a escrita no registrador, com ele travado em zero nenhuma instrução conseguirá gravar nos registradores. Com isso todas as instruções que necessitam gravar no registrador são afetadas, como lw e instruções do tipo R

PCUpdate

PCUpdate atualiza o PC para a próxima instrução, com ele travado em 0 nenhuma instrução conseguirá atualizar o pc, logo o processador vai travar, repetindo a mesma instrução.

Branch

Controla o desvio condicional de operações do tipo B. Se travar em 0 as instruções nunca desviam, logo apenas instruções do tipo B serão afetadas.

AdrSrc

Seleciona a origem do endereço para acessar a memória (PC ou ULA). Se travar em 0 a memória sempre será acessada com o PC (não com endereços calculados pela ULA). Instruções como lw e sw vão acessar o endereço errado (usando o PC em vez do endereço calculado)

MemWrite

Controla a escrita na memória. Se ele travar em 0 não será possível escrever na memória, a instrução afetada será sw que não vai conseguir gravar o valor na memória.

IRWrite

IRWrite permite gravar a próxima instrução no registrador IR. Com ele travado em zero não será possível fazer essa escrita, logo o processador não consegue carregar novas instruções, afetando TODAS as instruções. O processador trava no fetch

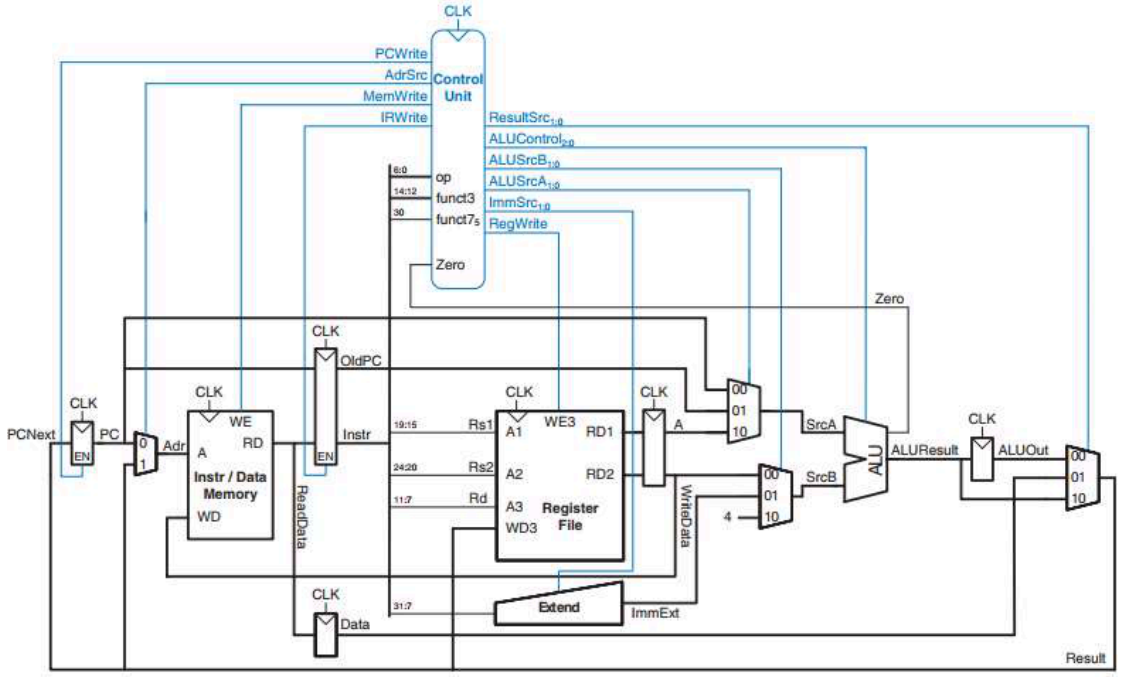
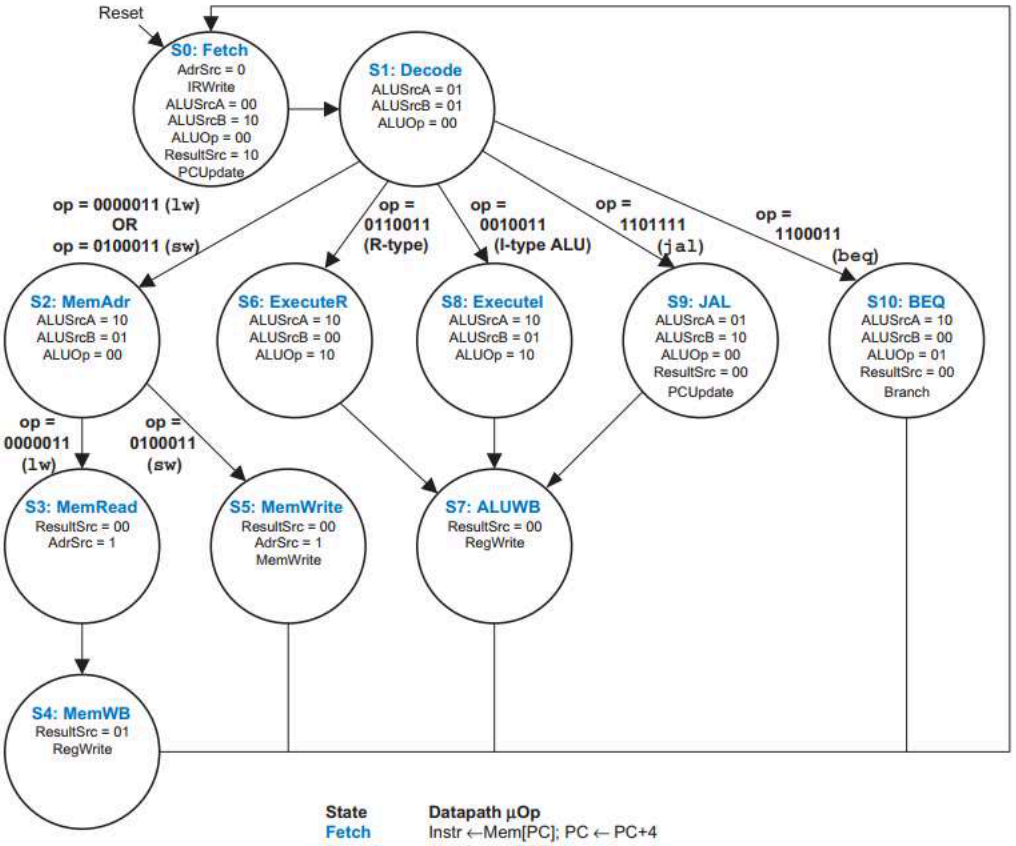


Figure 7.27 Complete multicycle processor



State	Datapath µOp
Fetch	Instr ← Mem[PC]; PC ← PC+4
Decode	ALUOut ← PCTarget
MemAdr	ALUOut ← rs1 + imm
MemRead	Data ← Mem[ALUOut]
MemWB	rd ← Data
MemWrite	Mem[ALUOut] ← rd
ExecuteR	ALUOut ← rs1oprs2
ExecuteI	ALUOut ← rs1opimm
ALUWB	rd ← ALUOut
BEQ	ALUResult = rs1-rs2; if Zero, PC = ALUOut
JAL	PC = ALUOut; ALUOut = PC+4

Figure 7.45 Complete multicycle control FSM

4- Considere as figuras do exercício anterior. Modifique o processador RISC-V monociclo para implementar uma das seguintes instruções. Anote as modificações necessárias no diagrama da nas figuras, incluindo novos sinais de controle, caso necessário. Faça também as alterações de controle necessárias na FSM principal, de modo a incluir as mudanças necessárias no decodificador principal e no decodificador da ALU.

xor

mesma coisa do monociclo +

No FSM:

- Não é necessário criar estados novos
- Necessário garantir que o caminho da instrução Tipo R cubra o xor
  - Estado fetch → decode → execução ALU → wirte back no registrador

sll

- Mesmo processo do monociclo, mas a FSM já tem o caminho

- Basta adicionar no ALU Decoder e usar o caminho já existente na FSM

bne

Para implementar o bne podemos usar o mesmo datapath de bew, no entanto, devemos diferenciar as duas funções por meio do funct3 (000 = beq, e 001 = bne) e devemos implementar a lógica para flag zero (1 quando rs1 e rs2 forem iguais e 0 quando rs1 diferente de rs2), com isso criamos um novo sinal de controle que diferencia nossas instruções (branch Invert). Com zero e branch invert realizamos um and entre os dois que caso retorne 1 faz o salto, caso contrário não.

mexer na control unit na flag com a inversão de zero assim não precisa criar um estado novo na FSM

5- Quantos ciclos são necessários para executar o seguinte programa em o processador multiciclo RISC-V? Qual é o valor de CPI (Ciclos por Instrução) deste programa? Qual é a influência deste parâmetro no desempenho do processador?

```
addi s0, zero, 5 # result = 5

L1:
     bge zero, s0, done # se result <= 0, exit loop
     addi s0, s0, -1 # result = result - 1
j L1

done:
```

Pela figura da FSM devemos calcular o número de ciclo de cada instrução

- addi: 4 ciclos
- bge: 3 ciclos
- j: 3 ciclos

O loop começa em 5 e a cada iteração o valor é subtraído em 1 até que o resultado seja menor igual a 0. No total o loop roda 6 vezes

addi s0, zero, 5 → 1x

bge → 6x

addi → 5x

j → 5x

Soma instruções: 1 + 6 + 5 + 5 = 17

addi: 4 ciclos * 6 = 24

bge: 3 ciclos * 6 = 18

j: 3 ciclos * 5 = 15

Soma ciclos: 57

CPI: 57/17 = 3,35

Impacto do CPI no processador é que quanto maior ele, mais lento será o processador.

RISC-V: Microarquitetura c/ **Pipeline**

6- Explique as vantagens dos microprocessadores com pipeline.

O processador pipeline é substancialmente mais rápidos que os outros, pois depois do enchimento do pipeline é possível realizar uma instrução por ciclo de clock, esse clock também é menor do que o clock do monociclo (ele leva em conta o tempo de execução da etapa mais lenta).

Arquitetura	Execução de instruções	Ciclo de clock	Paralelismo	Eficiência
Monociclo	1 instrução por ciclo	Longo (instrução mais lenta)	Nenhum	Baixa
Multiciclo	1 etapa por ciclo	Curto (etapa mais lenta)	Nenhum	Média
Pipeline	Várias instruções em diferentes etapas	Curto (etapa mais lenta)	Alto	Alta

---

7- Se estágios de pipeline adicionais permitem que um processador vá mais rápido, por que os processadores não têm 100 estágios de pipeline?

Não há motivo para implementar mais estágios se minhas instruções continuarão demorando o mesmo tempo dentro dos blocos, logo para a execução de próximas instruções ainda seria necessário esperar a instrução mais lenta terminar para passar de bloco → aumenta o risco de hazards

sobrecarga de registradores

MUITOOOOS HAZARDS

8- Descreva o que é hazard em um microprocessador com pipeline, e explique como pode ser resolvido. Quais são os prós e contras de cada forma ?

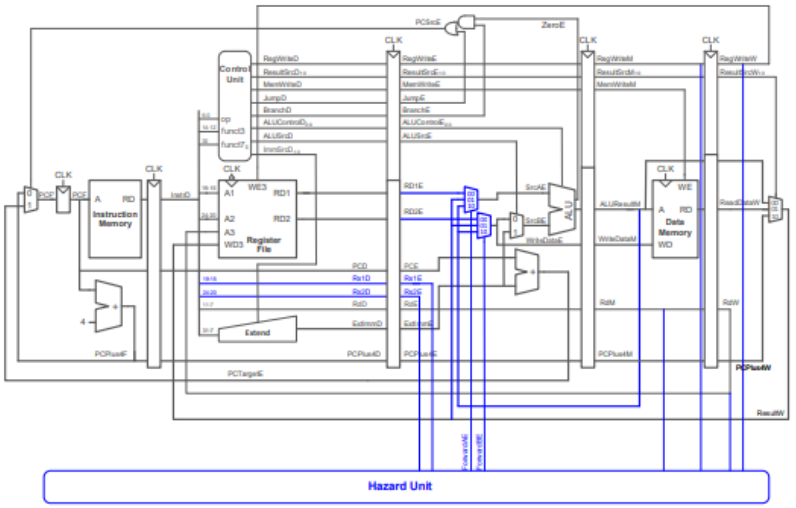
Os hazards são problemas que podemos encontrar nos pipelines por conta do paralelismo. Existem dois tipos:

- O **hazard de dados**: quando um dado ainda não está pronto para ser utilizado na próxima instrução
- Ele pode ser resolvido com:
  - Inserção de nops
    - Insere nops até que o resultado esteja pronto para ser utilizado
    - É uma solução ineficiente pois gera perda de desempenho
  - Reorganizar o código
    - Reorganiza o código movendo instruções úteis e independentes adiante no código

    - Forwarding
        - Quando os dados estão disponíveis nos barramentos internos do pipeline antes de serem gravados no arquivo de registradores
        - Encaminha os dados diretamente do armazenamento interno para ser usado no execute
        - Usa o hazard unit (lógica de controle do forwarding)
    - Stall
        - Para um estágio desativando seu registrador de pipeline para que as entradas do estágio não mudem
        - Quando um estágio está parado, todos os estágios anteriores também devem ser parados para que nenhum instrução subsequente seja perdida
        - O registrador de pipeline diretamente após o estágio parado deve ser **esvaziado (flushed)** para evitar que as informações errôneas se propaguem nos próximos estados
        - **As paradas degradam o desempenho**, então eles devem ser usado somente quando necessários
- **Harzard de controle**
    - quando o cáculo na ULA ainda não foi executado para saber se a próxima instrução é PC + 4 ou um salto na branch.
    - **Previsão de desvios**
        - Prever se os desvios serão tomados e começar a executar as instruções com base na previsão (**branch prediction)**
        - Uma vez que a decisão sobre o desvio estiver disponível, o processador pode descartar as instruções se a previsão estava errada
        - quando o desvio for tomado, as instruções que seguem a ramificação devem ser interrompidas e descartadas
        - Os ciclos de instrução desperdiçados são chamados de **branch misspredictionpenalty**
    - **Lógica de descarte (flushing)**
        - Se o desvio for tomado no estágio de execução, será ncessário descartar as instruções nos estágios de busca e decodificação → limpar os registradores de pipeline decode e execute (flushD = PCSrcE e FlushE = lwStall OR PCSrcE)

9- Considere o processador RISC-V com pipeline da figura abaixo, que está executando o seguinte trecho de código. trecho. Quais registradores estão sendo escritos e quais estão sendo lidos no quinto ciclo? Lembre-se de que o processador RISC-V com pipeline possui uma hazard unit. Você pode assumir um sistema de memória que retorna o resultado lido em um e um ciclo.

```
xor s1, s2, s3 # s1 = s2 ^ s3
addi s0, s3, -4 # s0 = s3 - 4
lw s3, 16(s7) # s3 = memory[s7+16]
sw s4, 20(s1) # memory[s1+20] = s4
or t2, s0, s1 # t2 = s0 | s1
```



	IM	Decode	Execute	Memory	Writeback
1	xor				
2	addi	xor			
3	lw	addi	xor		
4	sw	lw	addi	xor	
5	or	sw	lw	addi	xor

Os registradores lidos são s4 e s1

escritos são s1

NÃO TEM HAZARD

10- Repita o exercício anterior para o seguinte trecho de código:

```
addi s1, zero, 52 # s1 = 52
addi s0, s1, -4 # s0 = s1 - 4 = 48
lw s3, 16(s0) # s3 = memory[64]
sw s3, 20(s0) # memory[68] = s3
xor s2, s0, s3 # s2 = s0 ^ s3
or s2, s2, s3 # s2 = s2 | s3
```

Ciclo	Fetch	Dec	Ex	Mem	Writeback
1	addi				
2	addi	addi			
3	lw	addi	addi		
4	sw	lw	addi	addi	

5	xor	sw	lw	addi	addi

Leitura de registradores: s3 e s0

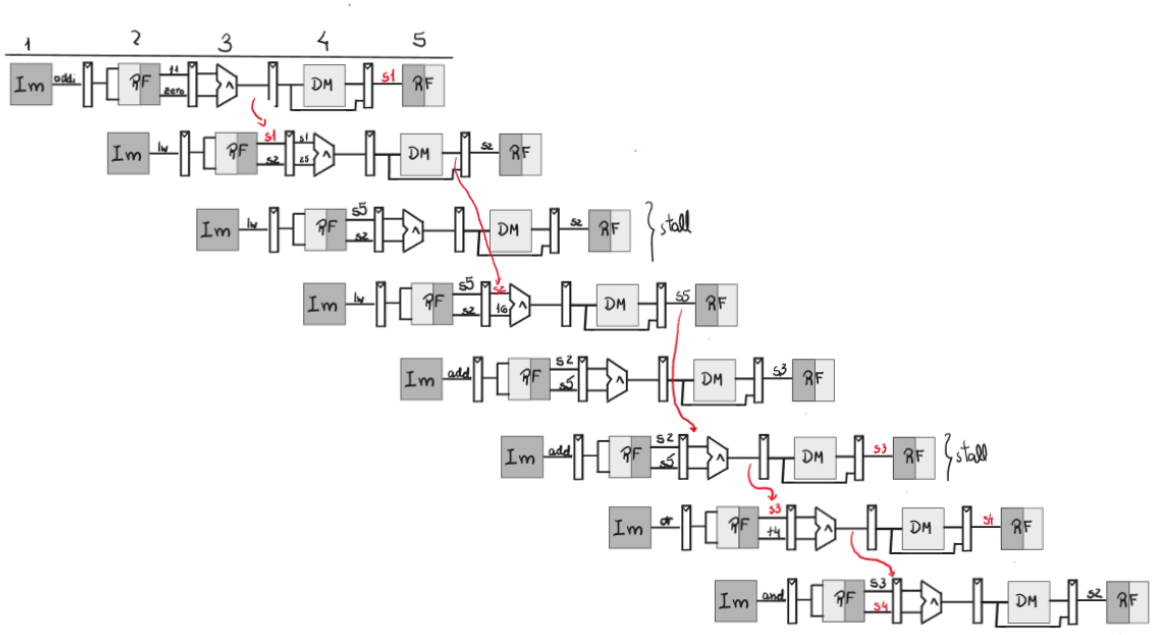
Escrita de registradores: s1

stall no lw e forwarding no addi

11- Usando um diagrama semelhante à figura abaixo (*desconsidere os nros de registradores indicados*), mostre o encaminhamento (forwarding) e paradas (stalls) necessários para executar as instruções abaixo no pipeline do processador RISC-V.

```
addi s1, zero, 11 # s1 = 11
lw s2, 25(s1) # s2 = memory[36]
lw s5, 16(s2) # s5 = memory[s2+16]
add s3, s2, s5 # s3 = s2 + s5
or s4, s3, t4 # s4 = s3 | t4
and s2, s3, s4 # s2 = s3 & s4
```

- stall no lw s2
- stall no lw s5
- forward no add, or, and, addi
- fazer desenho com as caixinhas nos estágios com forwarding e stalls
- stall é sempre que tenho duas instruções seguintes → faço o stall e depois faço o forwarding (no caso do load, no tipo R não)
- forward quando consigo passar um resultado antes de passar no registrador para usar
- store não causa hazard pois guarda na memória não nos registradores
- forwar e stall → quando envolve escrita e leitura de REGISTRADORES



12- Quantos ciclos são necessários para o processador RISC-V com pipeline emitir (buscar) todas as instruções para o programa abaixo ? Qual é o valor de CPI (Ciclos por Instrução) deste programa?

```
addi s1, zero, 52 # s1 = 52
addi s0, s1, -4 # s0 = s1 - 4 = 48
lw s3, 16(s0) # s3 = memory[64]
sw s3, 20(s0) # memory[68] = s3
xor s2, s0, s3 # s2 = s0 ^ s3
or s2, s2, s3 # s2 = s2 | s3
```

Qtd de ciclos necessários para emitir → 7 (6 INSTRUÇÕES + 1 STALL(em sw))

CPI = 7 + 4 = 11 / 6 = 1,83333...

CPI → precisa ver quanto demorou para executar tudo de emitir até acabar o programa → num de ciclos / num de instruções

---