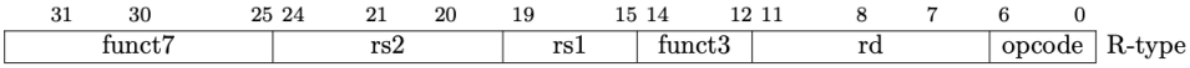
1. Tipos de instruções:

funct3 e funct7: auxílio para definição da operação
opcade: código da operação

Tipo-R:

- Operações aritméticas com registradores.



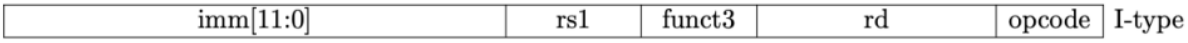
- Tem rs1, rs2 e rd.
- funct3 serve para falar qual tipo de operação é, exemplo, add, sub.
- funct7 serve para entrar na ULA e ela saber qual operação deve fazer

- Ordem: rd, rs1, rs2

- add s2, s1, s0
  - rd = s2
  - rs1 = s1
  - rs2 = s0

Tipo-I:

- Todas as operações que contém immediate (menos sw)



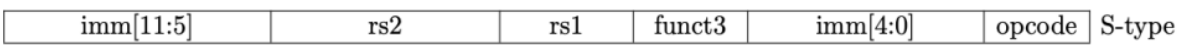
- Contém rs1, rd e imm

- Ordem: rd, rs1

- lw s2, 0(sp)
  - rs1 = sp
  - rd = s2
  - imm = 0

Tipo-S:

- Reservado somente para o store



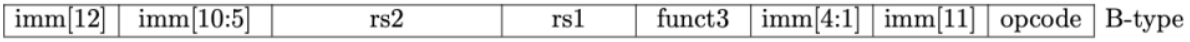
- Contém: rs1, rs2 e immediate

- Ordem: rs2, rs1

- sw s2, 0(sp)
  - rs1 = sp
  - rs2 = s2
  - imm = 0

Tipo-B:

- Comandos do tipo branch



- Contém rs1, rs2 e imm
- rs1 e rs2 serão comparados e imm será para onde irá acontecer o jump

- beq s0, s1, 2
  - rs1 = s1
  - rs2 = s0
  - imm = 2

Tipo-U:

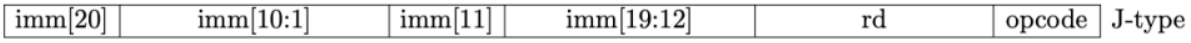
- Carrega um unsigned int em um registrador

- Contém immediate e um rd

- lui s0, 0x01234
  - rd = s0
  - imm = 0x01234

Tipo-J:

- jal



- Contém rd e imm

- jal s0, 2
  - rd = s0
  - imm = 2

Circuitos em: Diagramas

2. Instruções:

add (sub, mult, div):

- Tipo r
- add t0, t1,t2
  - rd = t0
  - rs1 = t1
  - rs2 = t2

addi:

- Tipo I
- addi t0, t1, 4
  - rd = t0
  - rs1 = t1
  - imm = 4

lw:

- Tipo I
- lw t0, 0(sp)
    - *t0 = *(0 + *sp)
    - rd = t0
    - rs1 = sp
    - imm = 0

sb:

- Tipo S
- sb t0, 0(t1)
    - *(0 + *t1) = *t0
    - rs1 = t1
    - rs2 = t0
    - imm = 0

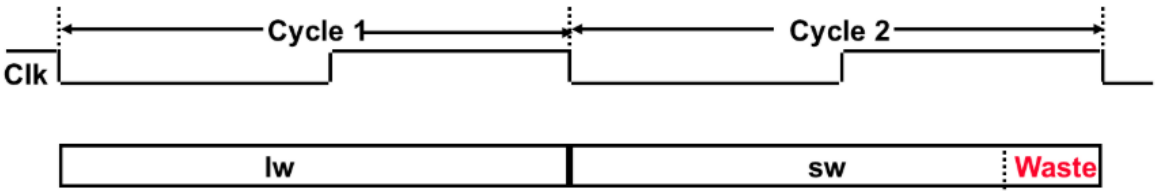
beq:

- Tipo b
- beq t0, t1, 8
    - rs1 = t1
    - rs2 = t0
    - imm = 8

3. Multiciclo

Em uma arquitetura de monociclo o clock será dado pela operação mais lenta, entretanto nem todas as operações vão precisar de um clock tão longo para ser realizada, portanto em várias instruções vai existir tempo perdido como visto na figura 1.

Figura 1:



Dessa forma a ideia é separar uma instrução em partes, sendo elas:

- IF
    - Busca da instrução
- ID/WB
    - Busca dos registradores (Reg) e decodificação da instrução (Dec)
    - Tem um /WB, pois nessa mesma instrução pode ser realizado a escrita em um registrador
- EXEC
    - Calcula o endereço de memória ou executa a operação na ULA

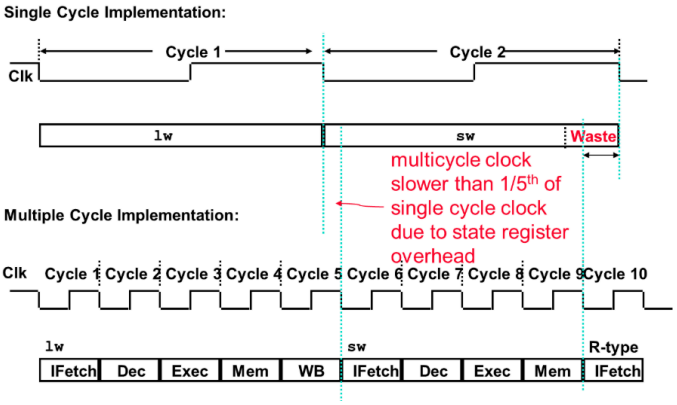
- MEM
    - Acesso a memória (leitura ou escrita)
- WB
    - Escreve o dado nos registradores

Dessa forma é possível perceber que nem todas as instruções vão precisar do WB, por exemplo, fazendo com que diminua o tempo para executar a instrução no geral.

3.1. Comparação Multiciclo X Monociclo

Como se pode ver na figura 2 realizar a instrução nos 5 ciclos é mais demorado do que em somente 1 ciclo, isso se dá, pois no multiciclo entre cada um dos ciclos é necessário armazenar informações em registradores para que a mesma não seja perdida no próximo ciclo.

Figura 2



4. Pipeline

- pipeline não diminui o tempo de execução de cada tarefa, mas aumenta o
- throughput de toda a carga de trabalho
- A taxa do pipeline é limitado pelo estágio mais lento
- Várias tarefas são realizadas simultaneamente utilizando diferentes recursos
- Um speedup potencial é o número de estágios do pipeline
    - Speedup é o quanto tempo vc demorou de maneira sequencial e o quanto tempo vc demorou de maneira paralela
    - É a razão entre os dois tempos.
- O tempo para “encher” o pipeline e para “esvaziá-lo” diminui o speedup
    - O tempo para encher o pipeline é o quanto tempo demorou para a primeira tarefa ser concluída
        - A partir dessa primeira conclusão as outras tarefas vão concluir mais rapidamente
    - Esse tempo para esvaziar é se, por exemplo, tiver uma dependência o pipeline vai ter que parar, consequentemente ele vai ser esvaziado e depois você vai ter que encher ele novamente. Por isso que esvaziar diminui o speedup, pq isso significa ter que reiniciar o pipeline e encher ele dnv.

- O pipeline trava se houver dependências
- Exemplo com a execução de 100 instruções:

    - Computador com um único ciclo: 45 ns/ciclo * 1 CPI * 100 instruções = 4500 ns
    - Computador com multiciclo: 10 ns/ciclo * 4,2 CPI * 100 instruções = 4200 ns
    - Computador com 5 estágios ideais de pipeline: 10 ns/ciclo * (1 CPI * 100 instruções + 4 ciclos para esvaziar) = 1040 ns

IF:

- Registradores:
    - PC
        - PC vai ser utilizado em EXEC para o cálculo de branch
    - IR
        - Para que as informações sobre as instruções possam ser buscadas em ID
- Sinais de controle:
    - Nenhum (nem foram gerados ainda)

ID:

- Registradores (ID/EX):
    - PC
        - PC vai ser utilizado em EXEC para o cálculo de branch
    - A
        - Cálculo da ULA em EXEC
    - B
        - Cálculo da ULA em EXEC e destino em WB para a função SW
    - IMM
        - Cálculo da ULA em EXEC e Branch
    - RD
        - Será necessário em WB
- Sinais de controle:
    - ImmSRC
        - Usado em ID para controlar o IMMControl
    - RegWrite
        - Teoricamente é usado em ID, mas de verdade ele é importante no WB (que vai voltar no ID) para dizer se deve ou não escrever no banco de registradores
- Sinais de controle (ID/EX):
    - ALUop
        - Usado para controlar o ALUControl
    - ALUsrc
        - Usado para controlar o MUX que escolhe entre B e IMM
    - MemRead
    - MemWrite
    - MemToReg
    - Branch
    - RegWrite

EXEC:

- Registradores (EXEC/MEM)
    - B
        - Vai ser preciso em MEM
    - RD
        - Vai ser preciso em WB
    - ALUOutput
        - Vai ser preciso em MEM
    - BranchTarget
        - Vai ser preciso em MEM
    - Zero
        - Flag zero
- Sinais de controle:
    - ALUop
        - Controlar qual operação a ULA vai fazer
    - ALUSrc
        - Controlar quem o mux vai escolher B ou IMM
    - Branch
        - Para o cálculo do BranchTarget
- Sinais de controle (EXEC/MEM):
    - MemToReg
    - MemRead
    - MemWrite
    - RegWrite

Mem:

- Registradores (MEM/WB)
    - ALUoutput
        - Fazer a escolha em WB de quem que vai ser escrito no banco de registradores (LMD) ou ALUOutput
    - LMD
        - Fazer a escolha em WB de quem que vai ser escrito no banco de registradores (LMD) ou ALUOutput
    - RD
        - Precisa dele em WB
- Sinais de controle:
    - MemRead
    - MemWrite
- Sinais de controle (MEM/WB):
    - RegWrite
    - MemToReg

5. Conflito estruturais

Acontece quando duas partes são acessadas no mesmo ciclo de clock por duas operações diferentes.

Memória

- A memória é acessada em ID (banco de registradores) e MEM (memória ram).
    - Se ambas as memórias fossem juntas ia dar merda, então se teve a necessidade de separar elas para não dar conflito estrutural.

ULA

- A ula é utilizada em IF e EXEC
    - Em IF ela é usada para calcular o novo PC, já em EXEC ela é utilizada para fazer coisa pra karai.
    - A solução foi criar mais ulas

Banco de registradores

- Em ID o banco de registradores é utilizado para ler os registradores, mas ao mesmo tempo ele é utilizado para escrever no registrador em WB.
- A solução para esse problema foi fazer com que na primeira metade do clock a leitura fosse realizada e na segunda metade do clock a escrita é realizada.

6. Dependências

A principal dependência é aquela onde é feita a escrita em um registrador e logo depois é feito a leitura nesse registrador.

Vamos imaginar a seguinte situação:
add t0, t1, t2
sub t3, t0, t4

O que acontece no Pipeline vai ser

CC1	CC2	CC3	CC4	CC5
IF	ID	EXEC	MEM	WB
	IF	ID		

Quando chega em CC3 a instrução sub precisa do valor de t0, porém ele ainda não está pronto, pois só vai ter o resultado dele em CC5 no WB. Dessa forma ID deve ser congelado até que chegue em CC5 resultado em:

CC1	CC2	CC3	CC4	CC5
IF	ID	EXEC	MEM	WB
	IF	ID	ID	ID

Dessa forma agora em CC5 ID lê o T0 certo.

Seguindo essa lógica é fácil perceber que é necessário 2 paradas no pipeline para que a dependência seja ajustada. Essa parada na realidade é uma instrução chamada NOP que é colocada entre as instruções.

7. Forward

Para resolver o problema da dependência e para que não tenha hazard (para obrigatória do pipeline) foi criado o forward.

Em ID os valores dos registradores só são buscados e não são utilizados, então tudo bem você está buscando lixo, o que importa é que em EXEC você saiba que aquilo que você buscou é um lixo. Logo, se você está em EXEC da instrução I+1 significa que a instrução I vai estar em MEM, portanto, se a instrução I escreveu em um dos registradores que a instrução I+1 está usando isso significa que EXEC/MEM.RD == (ID/EXEC.rs1 OR ID/EXEC.rs2), dessa forma se você fizer essa checagem você sabe que está pegando lixo.

Se você estiver pegando lixo isso significa que você deve ignorá-lo e pegar o valor correto. O interessante é que o valor correto não está no banco de registradores, mas ele é conhecido, pois ele está em EXEC/MEM.ALUOutput, portanto, isso resolve o problema da instrução I escrever e I+1 utilizar. Entretanto um outro problema emerge, e se a instrução I escrever a instrução I+2 utilizar? Nessa ocasião o valor correto não vai mais estar em EXEC/MEM.ALUOutput e sim em MEM/WB.ALUOutput, portanto deve ser feito essa checagem.

Um outro problema emerge, e este problema é o fato de que o que está escrito até agora funciona para todos os casos menos para a LW, pois a resposta da LW nunca vai estar em ALUOutput e sim em LMD, logo, para resolver em I+2 é fácil, pois invés de pegar MEM/WB.ALUOutput é só pegar o resultado do MUX entre MEM/WB.ALUOutput e MEM/WB.LMD. Já na caso de I+1 é, infelizmente, impossível de resolver vai ter que acontecer um hazard.

Essa escolha entre pegar A,B ou ALUOutput ou o resultado do mux de WB é feita por meio de dois MUXs, um para a primeira entrada da ULA e um segundo para a segunda entrada da ULA. Os sinais que controlam esses MUXs são gerados pela Unidade de Forward que contém a seguinte lógica:

IF (EXEC/MEM.rd == ID/EXEC.rs1) and (EXEC/MEM.rd != 0) and (