Fundamentos de Arquitetura de Computadores

Tiago Alves

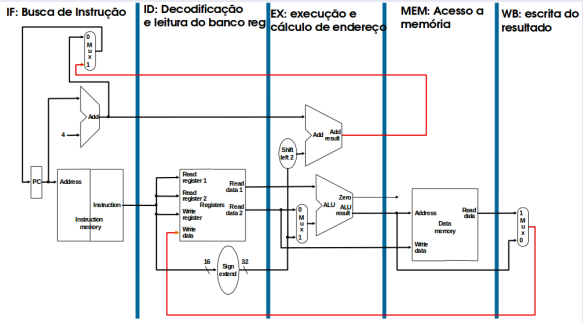
Faculdade UnB Gama
Universidade de Brasília

Controle do Pipeline

Temos 5 estágios. O que deve ser controlado em cada estágio?

- Busca de Instruções ou *Instruction Fetch* (e *PC Increment*).
- Decodificação da Instrução ou *Instruction Decode*/*Register Fetch*,
- Execução ou Cálculo de endereço ou *Execution*,
- Acesso à memória de dados ou *Memory Stage* e
- Escrever o Resultado em um registrador ou *Write Back*.

Caminho de Dados Uniciclo



Fluxo de dados *Esquerda para Direita* (com 2 exceções)

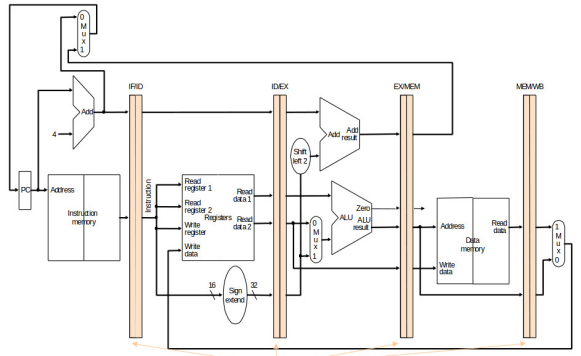
Duas Exceções

- O estágio de escrita (WB) coloca o resultado de volta no banco de registradores;
- A seleção do próximo PC:

  PC + 4; ou

  Endereço de Desvio do Estágio MEM;
- Os dados na direção contrária não afetam a instrução atual, somente a próxima instrução.

Caminho de Dados com Pipeline



Registradores dos estágios do pipeline

As barras que delimitam os estágios consistem em **registradores dos estágios** do pipeline.

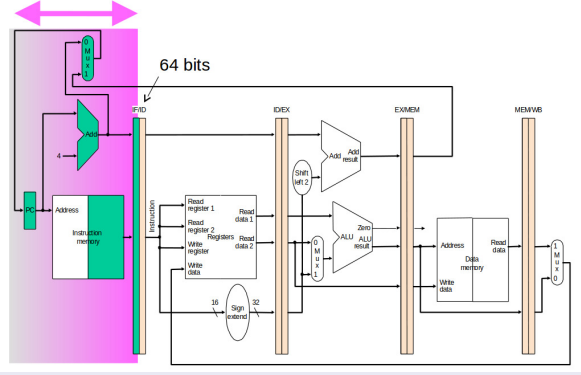
Tamanho dos Registradores de Pipeline

Precisam ser grandes o suficiente para suportar o armazenamento dos dados em cada estágio:

- IF/ID: 64 bits.
- ID/EX: 128 bits.
- EX/MEM: 97 bits.
- MEM/WB: 64 bits.

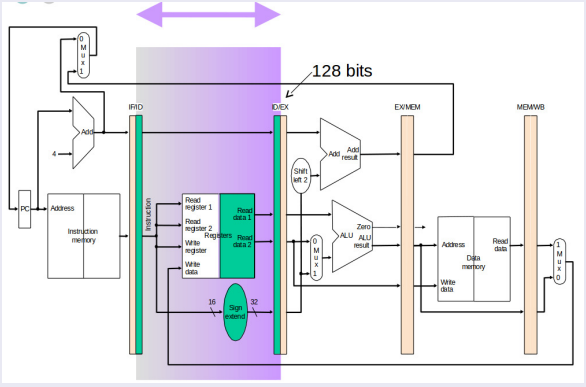
Evolução da instrução Load (5 estágios)

Busca de instrução:



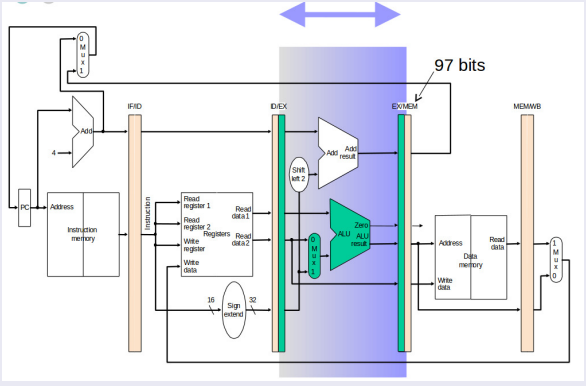
Evolução da instrução Load (5 estágios)

Decodificação:



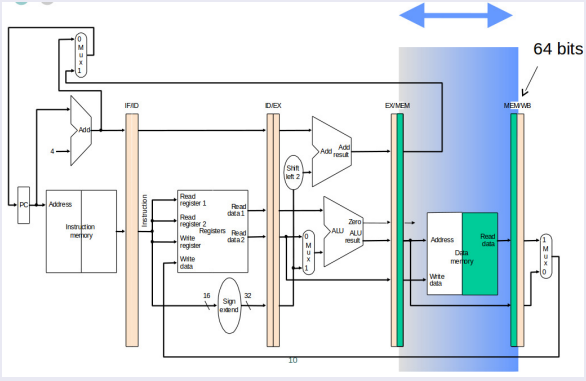
Evolução da instrução Load (5 estágios)

Execução/Cálculo do Endereço:



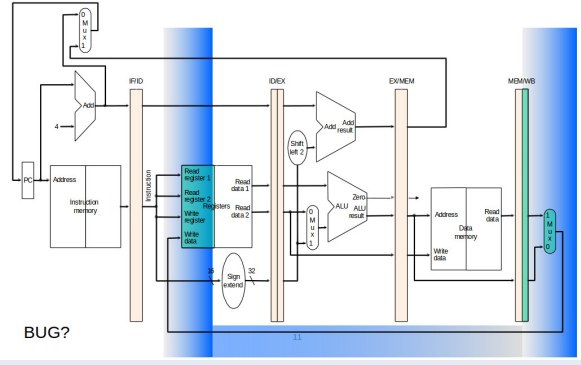
Evolução da instrução Load (5 estágios)

Acesso à Memória:



Evolução da instrução Load (5 estágios)

Escrita do Resultado:

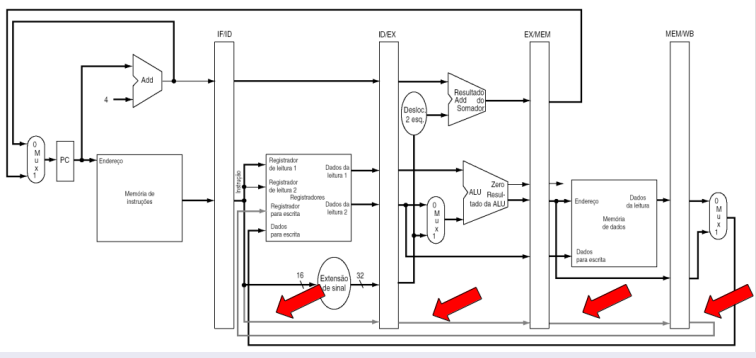


Bug (Inconsistência Estrutural) e Solução

- Ao realizar uma instrução do tipo Load (`lw`) temos mais um estágio após o uso do registrador MEM/WB para inserir o dado lido da memória no registrador de destino.
- Deve-se preservar a instrução lida após o estágio IF.
- Solução: Passar o número do registrador de escrita primeiro para ID/EX e depois para o EX/MEM, e finalmente, para MEM/WB.

Bug (Inconsistência Estrutural) e Solução

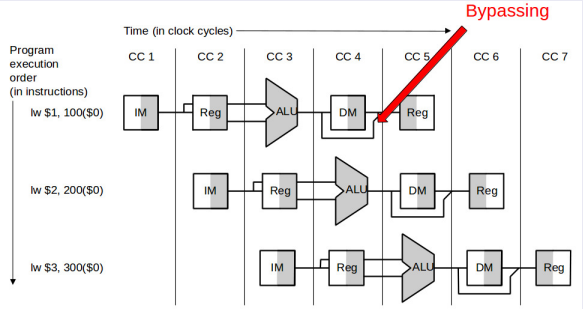
Atenção: faz-se necessária mudança no tamanho dos registradores de pipeline após essa alteração.



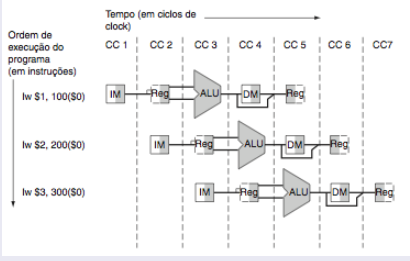
Fluxo de Execução do Pipeline

- Diagrama Visual representando 5 estágios do Pipeline MIPS;
- Incorpora os módulos de Bypassing e Forwarding na representação do Pipeline.

Fluxo de Execução do Pipeline



Fluxo de Execução do Pipeline



Facilita no entendimento do processo:

- Quantos ciclos leva para executar esse código?
- Qual é o valor da ALU sendo executado durante o ciclo 4?

Essa representação é fundamental para entender o caminho de dados e os possíveis hazards no pipeline.

Controle do Pipeline

Temos 5 estágios. O que precisa ser controlado em cada estágio?

- Busca da instrução:

  Nenhum sinal de controle

- Decodificação da instrução / leitura do banco de regs.:

  Nenhum sinal de controle

- Execução / cálculo de endereço:

  `RegDst, ALUOp, ALUSrc`

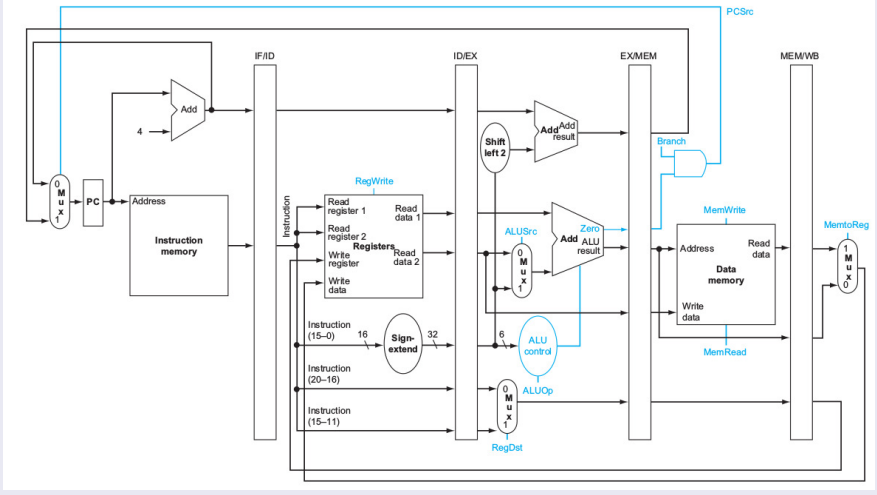
- Acesso a memória:

  `Branch, MemRead, MemWrite`

- Escrita do resultado:

  `MemtoReg, RegWrite`

Controle do Pipeline: Sinais



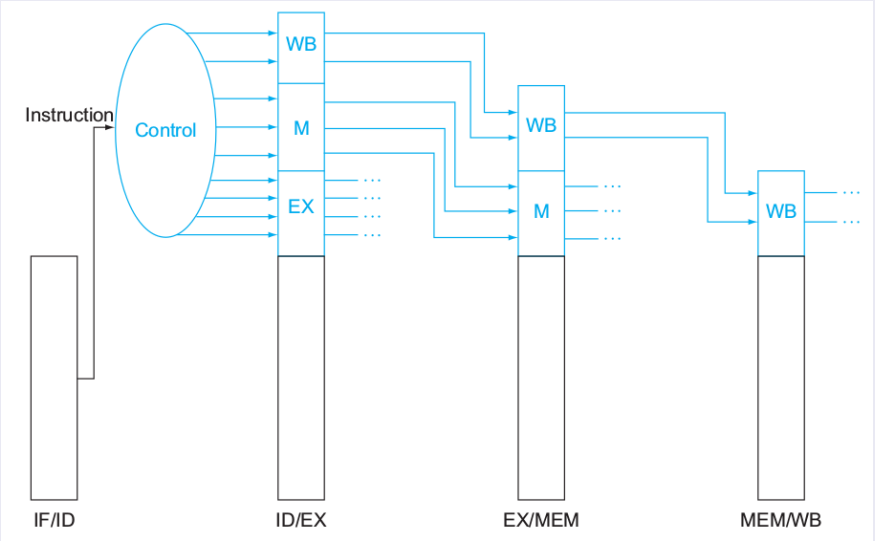
Controle do Pipeline: Sinais

Sinais de controle são transmitidos da mesma forma que os dados

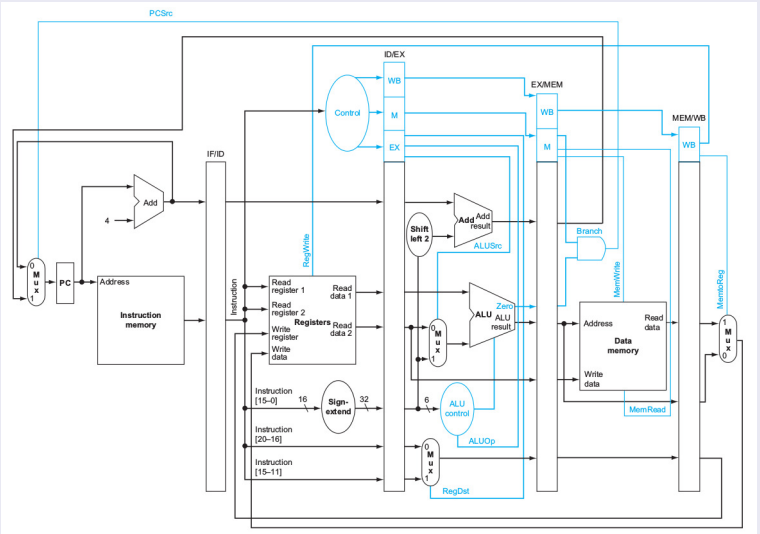
Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
Instruction	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	1	1	0	0	0	0	0	1	0
`lw`	0	0	0	1	0	1	0	1	1
`sw`	X	0	0	1	0	0	1	0	X
`beq`	X	0	1	0	1	0	0	0	X

Controle do Pipeline: Sinais



Caminho de Dados com Pipeline e Controle



Risco de Dados e Forwarding

- Os Hazards são obstáculos para a execução do Pipeline.
- Hazard de dados estão diretamente correlacionados ao código a ser executado: provocado principalmente por dependências entre a instrução atual e a anterior.

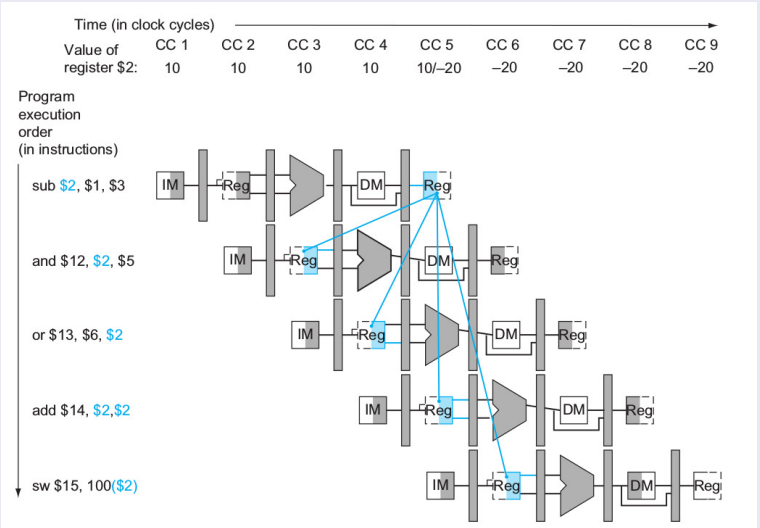
Risco de Dados e Forwarding

- Suponha a execução das instruções a seguir:

```
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
```

- Suponha $2 iniciado com valor 10 e depois do `sub` com valor -20.
- É fácil perceber que todos são dependentes do resultado da função `sub`!

Risco de Dados e Forwarding



Pares de Risco de Dados - Detecção

Os tipos são:

- 1a) `EX/MEM.RegistradorRd = ID/EX.RegistradorRs`
- 1b) `EX/MEM.RegistradorRd = ID/EX.RegistradorRt`
- 2a) `MEM/WB.RegistradorRd = ID/EX.RegistradorRs`
- 2b) `MEM/WB.RegistradorRd = ID/EX.RegistradorRt`

Para o exemplo anterior: `EX/MEM.RegistradorRd = ID/EX.RegistradorRs = $2`

Avaliando o Exemplo

Para a sequência de código:

```
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
```

Classifique os tipos de Hazards de Dados.

Avaliando o Exemplo

Respostas....

- As instruções `sub-and` é um hazard tipo 1a:

  `EX/MEM.RegistradorRd = ID/EX.RegistradorRs`

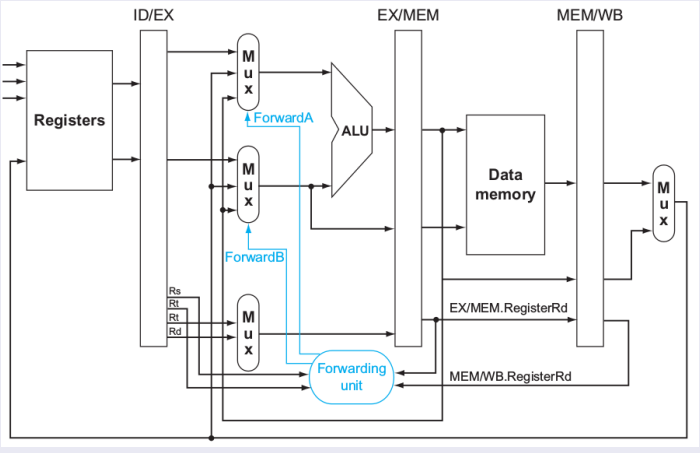
Para as demais instruções:

- `sub-or`: tipo 2b

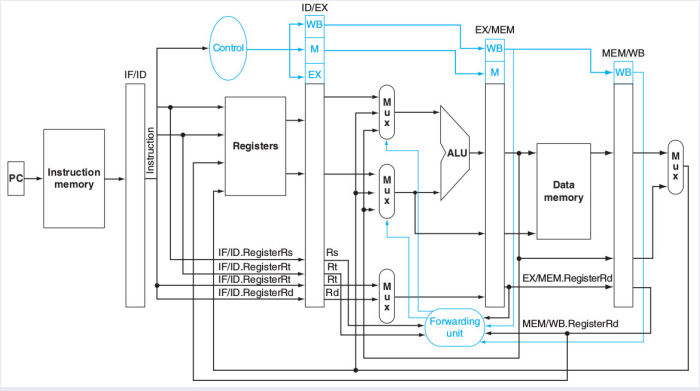
  `MEM/WB.RegistradorRd = ID/EX.RegistradorRt = $2`

- `sub-add` não são hazards pois o banco de registradores fornece dados apropriados para estágio ID do `add`;
- Não existe hazard de dados entre `sub` e `sw`, porquê `sw` lê $2 no cc depois que sub escreve $2.

Hazards de Dados: Forwarding



Hazards de Dados: Forwarding



Detecção de Hazards de Dados

Comparar índices dos registradores a serem lidos e escritos, armazenados nos registradores do pipeline:

- Deve-se comparar igualmente se a instrução escreve em um registrador (`RegWrite = 1`) ou lê da memória (`MemRead = 1`)

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Detecção de Hazards de Dados

Ex:

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10
```

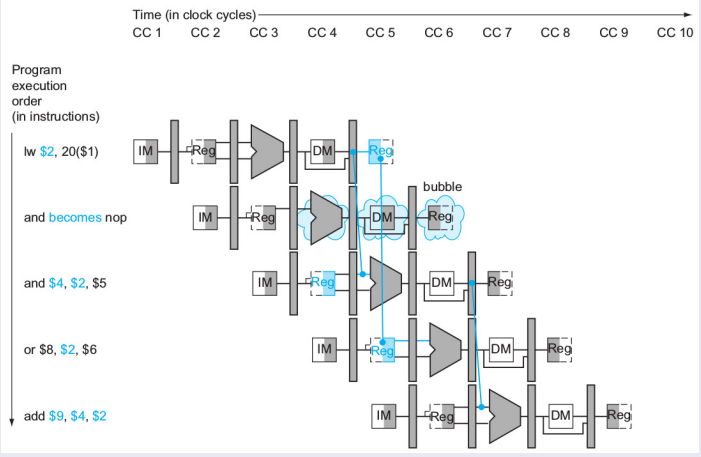
Mem:

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01
```

Hazards de Dados: Quando o forward não é possível

Atraso de 1 cc!

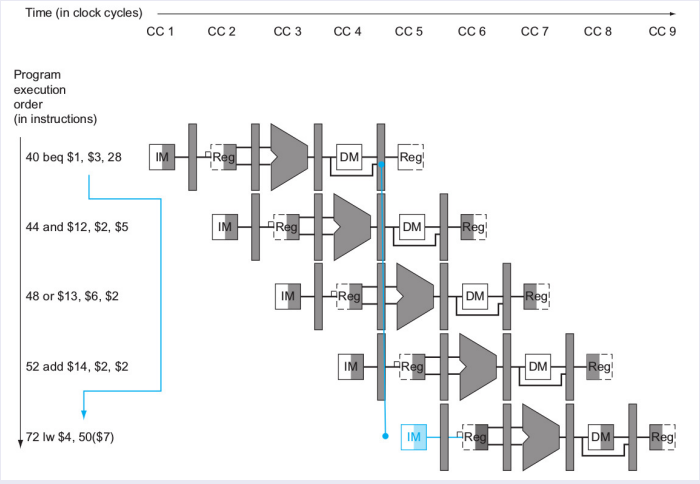


Detecção de Hazards de Dados

```
if (ID/EX.MemRead and
   ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
     (ID/EX.RegisterRt = IF/ID.RegisterRt)))
     stall the pipeline
```

Hazard de Controle

Prevendo desvio não tomado:



Hazard de Controle Otimizado