Revisão – dependências de dados

- Dependências de dados resolvidas com adiantamento (quase sempre)
- Deve garantir que instruções anteriores escreverão resultado,
  destino é mesmo que fonte,
  e instrução anterior não tem prioridade
- Acrescentar circuito de adiantamento onde pode-se adiantar
  → força bloqueio se precisa esperar por resultado
  estágio EXEC, MEM para store, DECOD para desvio
- LOADs necessitam parada porque sobrepõem EXEC com MEM
  desvios podem necessitar de parada também

Desvios e Saltos

desvios `beq r1,r2,desl` — desl em 16 bits, compl-2
PC ←(r1=r2 ? PC+4+(extSinal(desl)≪ **2**) : PC+4)

saltos `j ender` — ender em 26 bits
PC ←PC(31..28) & (ender≪ **2**)

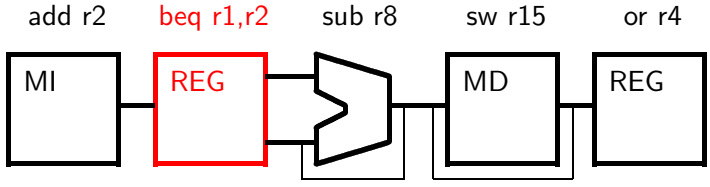
`jal ender` — ender em 26 bits
r31 ←PC+4 ; PC ←PC(31..28) & (ender≪ **2**)

`jr r3` — ender em 32 bits
PC ←r3

Dependências de Controle

```
1   or r4, r5, r6
2   sw r15, 0(r16)
3   sub r8, r9, r10
4   beq r1, r2, -4
5   add r12, r13, r13
6   and r14, r15, r16
```

Se (r1 = r2)
volta atrás 4 instruções (or)
senão
executa add na linha 5

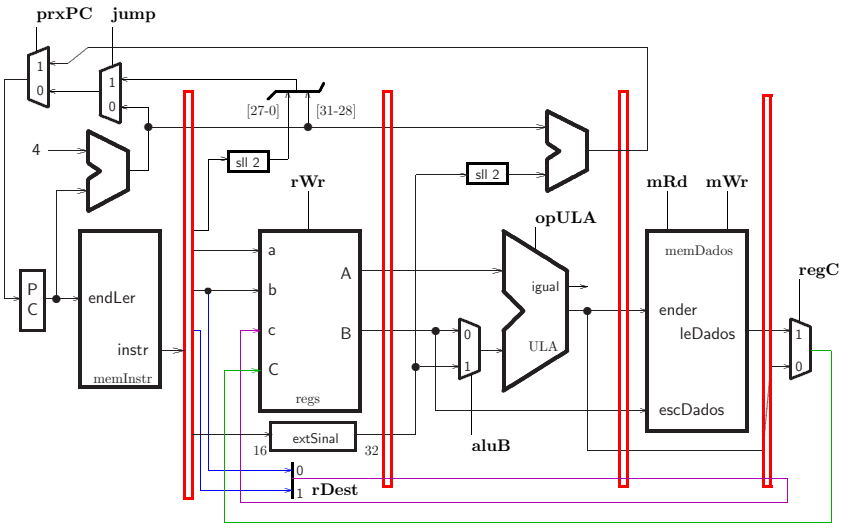


beq só escolhe novo PC no ciclo DEC!
O que acontece com a instrução em BUSCA?

Riscos de Controle

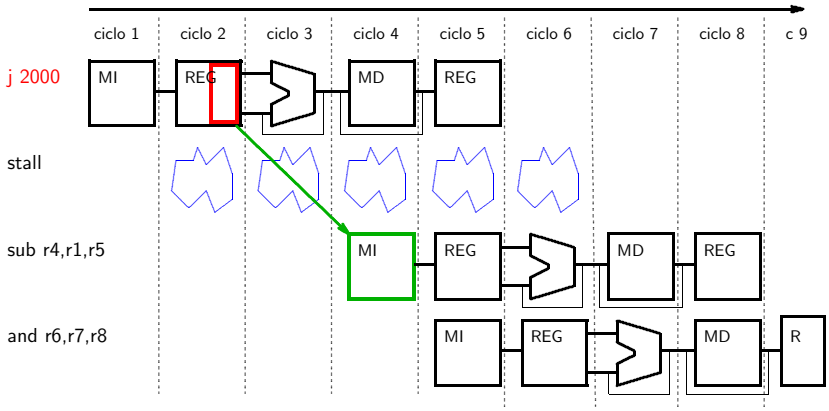
- Ocorrem quando seqüência de endereços não é linear
  → 'linear' é o 'normal' no processador segmentado
- riscos causados por instruções de controle de fluxo
  desvios condicionais `beq, bne`
  saltos `j, jal, jr`
- Soluções possíveis:
  (a) bloqueia (*stall*)
  (b) mover decisão para segmento próximo de BUSCA
  (c) atrasar a decisão (necessita ajuda do compilador)
  (d) prever a direção do desvio
- riscos de controle são menos freqüentes que riscos de dados;
  mas não há nada tão efetivo para riscos de controle quanto adiantamento para riscos de dados

Circuito com Desvios e Saltos

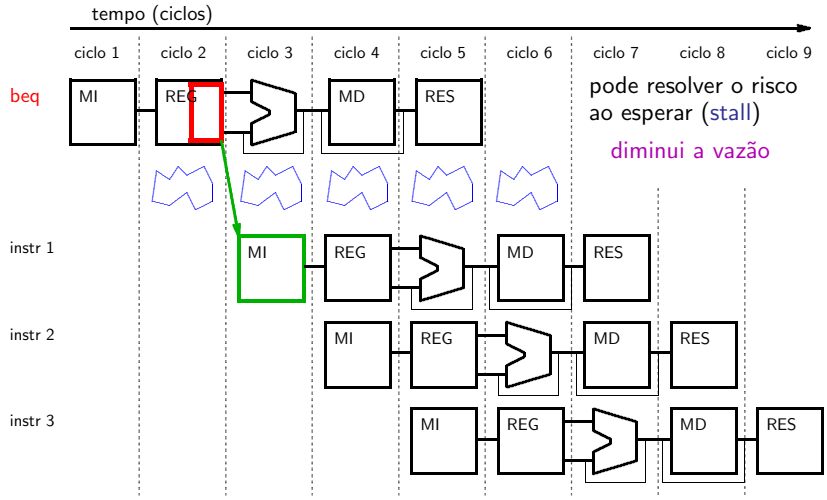


Saltos causam uma bolha

Saltos são decodificados em DECOD → necessitam um ciclo de stall
felizmente, saltos são infreqüentes: ≈2% nos programas SPECint



Desvios causam uma bolha



Decisão de desvio tomada no $2^{\underline{o}}$ estágio

- Adicionar circuito para computar endereço de destino e circuito para avaliar condição de desvio ao estágio DECOD
  - ▷ número de bolhas é uma (como nos saltos)
  - ▷ endereço de destino deve ser computado em paralelo com acesso ao bloco de registradores
    - ⇝ só usa endereço em desvios e saltos
  - ▷ a comparação só pode ocorrer após o acesso aos registradores para então alterar o PC – piora temporização do estágio
  - ▷ necessita adiantamento para o estágio DECOD
- em *pipelines* longos, decisão é mais tardia e causa mais bolhas
  → é necessário algo melhor...

Questões com adiantamento – desvios em DECOD

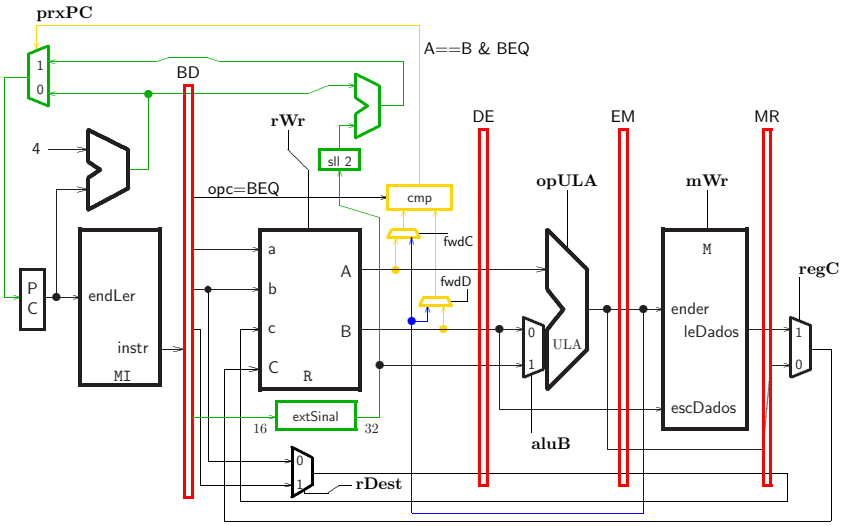
- Adiantamento de operandos no registrador EM
  adianta resultado da penúltima instr para entradas do comparador

```
if ( Dcntrl.desvio
 and ( EM.rd != 0 )
 and ( EM.rd == BD.rs ) ) { fwdC = 1 }
```

```
if ( Dcntrl.desvio
 and ( EM.rd != 0 )
 and ( EM.rd == BD.rt ) ) { fwdD = 1 }
```

- adiantamento de MR ocorre através do bloco de registradores
- se instrução imediatamente anterior ao desvio produz um dos operandos da comparação, então um bloqueio é necessário porque operação de ULA em EXEC ocorre ao mesmo tempo em que a comparação em DECOD

Desvios em DECOD



Tratamento de Riscos de Controle

- Tratamento de riscos de controle é MUITO importante
  pelo impacto no desempenho
  - ▷ de 1/3 a 1/6 de todas instruções são desvios
  - ▷ penalidade de um, dois ou três ciclos
  - ▷ processadores com mais estágios tem desempenho pior
- Atrasa instruções até decidir se desvia → perde desempenho
- Previsão de desvios → prever direção do desvio
  - ∗ reduz/elimina penalidade se previsão correta
  - ∗ pode aumentar penalidade quando previsão é errada
  - ∗ Técnicas:
    previsão estática – segue sempre mesmo caminho
    previsão dinâmica – depende do comportamento do programa

Previsão Estática de Desvios

Resolve risco de controle ao supor qual a direção tomada
e seguir executando sem esperar pelo resultado da comparação

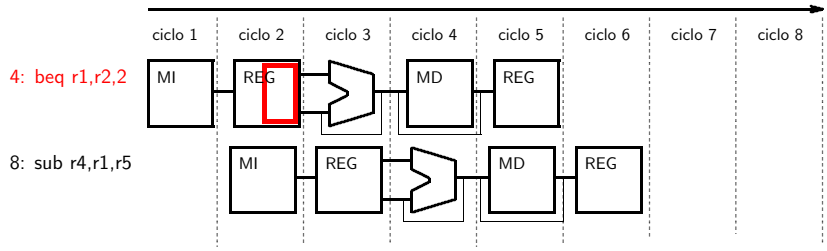
1. **Prevê não-tomado:** sempre prevê que desvios não serão tomados,
   continua buscando na seqüência das instruções (PC+4).
   Somente quando o desvio é tomado ocorrem bolhas.

Se desvio tomado, anula instruções após o desvio
– se decisão em MEM → três bolhas: BUSCA, DECOD, EXEC
– se decisão em DECOD → uma bolha: BUSCA

Implementação deve garantir que instr anuladas não mudam o estado
→ fácil no MIPS porque mudanças de estado ocorrem nos 2 últimos
estágios: MEM (`mWr`) ou RES (`rWr`)

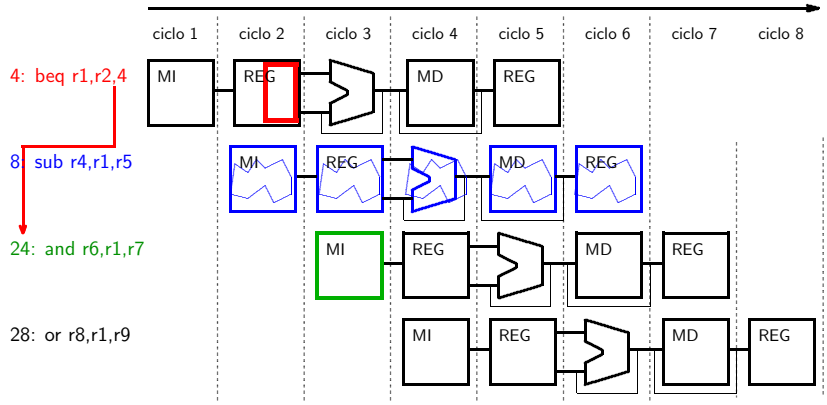
Busca re-inicia no destino do desvio (penalidade $\leq$ 2 bolhas)

Acerto na previsão → desvio não-tomado



Não há perda de tempo

Erro de previsão → tomado (⇝ anula instrução)



Para anular instrução na BUSCA, adicionar sinal `BUSCA.flush`
que força **0**s no campo do opcode no registrador B/D,
transformando instrução em NOP.

Previsão Estática de Desvios (cont)

Resolve risco de controle ao supor qual a direção tomada
e seguir executando sem esperar pelo resultado da comparação

2. **Prevê tomado:** prevê que todos os desvios serão tomados,
prevê tomado sempre causa uma bolha se circuito de previsão em DECOD

Na medida em que a penalidade aumenta (maior número de estágios),
previsão estática piora desempenho

Com circuito mais complexo é possível tentar prever comportamento
do desvio dinamicamente durante execução do programa

3. **Previsão Dinâmica de Desvios:** prevê desvios em
tempo de execução, usa informação coletada na execução adiante

Atrasar a Decisão

- Circuito de decisão e cálculo do endereço de destino em DECOD
- Um desvio atrasado sempre executa a próxima instrução (PC+4)
  → desvio tem efeito somente após aquela instrução
  * Montador do MIPS move uma "instrução segura" para a posição imediatamente após o desvio,
    assim escondendo a bolha causada pelo desvio
    instrução segura porque não é afetada pelo desvio
- Em processadores com mais de 5 estágios e emissão múltipla,
  o atraso no desvio fica maior e necessita mais de uma instrução
  ⋆ desvios atrasados perderam a vantagem/popularidade para outras abordagens dinâmicas mais flexíveis (embora mais caras)
  ⋆ crescimento no número de transistores disponíveis tornou abordagens dinâmicas relativamente baratas
- Desvios atrasados são um característica da arquitetura! B^(
  → CdI MIPS1 define desvios atrasados

Desvios Atrasados

- Sempre executa a próxima instrução OK com pipelines simples

```
i:    beq r1,r0, dst
i+1:  sub r2,r8,r9        sempre é executada
...
dst:  add r3,r4,r5
```

- É necessário que *branch delay slot* seja preenchido ≥ 1 NOP
  * preenche com instrução de antes do desvio (i-1)
  * preenche com instrução de destino
    ⋆ SE é seguro executar instrução de destino
    ⋆ ajuda somente no caso do desvio tomado
  * preenche com instrução após o desvio (i+2)
    ⋆ SE é seguro executar instrução após o desvio
    ⋆ ajuda somente no caso do desvio não-tomado

Desvios Atrasados

de antes	do destino	após desvio

add r1,r2,r3
beq r2,r0,dest
delay slot
...
...
	sub r4,r5,r6
	...
	...
	add r1,r2,r3
	beq r2,r0,antes
	delay slot
		add r1,r2,r3
		beq r2,r0,dest
		delay slot
		sub r4,r5,r6
		...

...
beq r2,r0,dest
add r1,r2,r3
...
...
	...
	...
	...
	add r1,r2,r3
	beq r2,r0,antes
	sub r4,r5,r6
		add r1,r2,r3
		beq r2,r0,dest
		sub r4,r5,r6
		...
		...
		...

Escalonamento de instruções

Compilador pode re-ordenar instruções para eliminar (algumas) bolhas

- muda ordem eliminando conflitos com recursos — risco estrutural
- aumenta distância entre produção e uso de valor — risco de dados
- decisão no início ou no final do laço — risco de controle

	original	re-escalonado
1:	lw r1, 0(r10)	lw r1, 0(r10)
2:	add r4, r1, r4	lw r5, 10(r12)
3:	sw r4, 0(r10)	add r4, r1, r4
4:	lw r5, 10(r12)	sub r8, r5, r9
5:	sub r8, r5, r9	sw r4, 0(r10)
6:	add r6, r8, r8	add r6, r8, r8 *
7:	sw r6, 20(r12)	sw r6, 20(r12) *

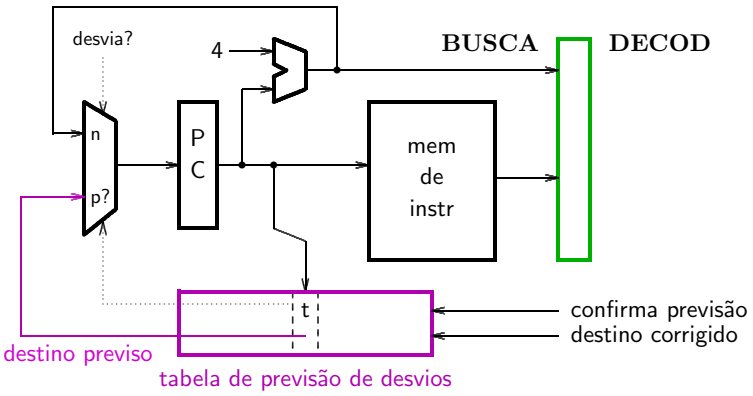
* dependência em r6 permanece

Previsão Dinâmica de Desvios

Para prever desvios é necessário:

1. coletar e registrar histórico de cada instrução de desvio
2. registro deve ter "boa qualidade"
3. usar a previsão para antecipar a busca da instrução de destino

Previsão Dinâmica de Desvios – *hardware*



PC indexa tabela e se previsão é para tomar desvio (**t**),
então carrega destino previsto no PC

Previsão Dinâmica de Desvios cont

Uma tabela com histórico de desvios (*branch history table, BHT*) no estágio de busca contém um bit que indica se o desvio foi tomado na sua última execução

BHT é indexada pelos m bits menos-significativos do PC

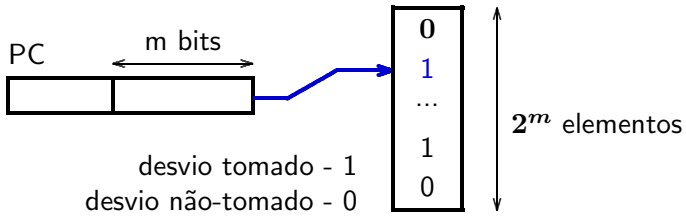
Bit pode prever incorretamente:
pode ser de outro desvio com os mesmos m bits-ms do PC,
ou pode ser a previsão errada para esta 'passada'
→ não pode afetar corretude, só desempenho

Se previsão é errada, anula instruções incorretas,
re-inicia busca no endereço correto, e
inverte bit de previsão

Previsão Dinâmica de Desvios (cont)

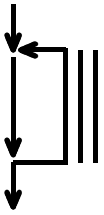
Tabela com $\mathbf{2}^{m}$ bits no estágio de busca

- acessa tabela com **m** bits do PC
  - tabela contém **1** se desvio foi tomado na última passada
  - tabela contém **0** se desvio não foi tomado na última passada
- prevê que nesta execução tomará mesmo caminho que na anterior
- atualiza o bit quando errar a previsão



Previsão Dinâmica de Desvios (cont)

Exemplo: executa três voltas do laço e então prossegue:



previsão	?	t	t	T	N	t	t	T	N	t	t	T	N
resultado	t	t	t	**n**	t	t	t	**n**	t	t	t	**n**	t
certo?				E	E			E	E			E	E

Duas previsões erradas,
a cada vez que seqüência muda,
ou a cada passagem pelo laço
neste exemplo

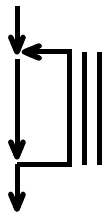
```
x = 10;
for (i=0; i<3; i++) {
    x = ...
}
z = x - 2;
```

Previsão Dinâmica de Desvios (cont)

Na previsão com um bit, ocorrem duas previsões erradas
a cada mudança com relação à história recente

p.ex: executa três voltas do laço e então prossegue:

previsão	?	t	t	T	N	t	t	T	N	t	t	T	N
resultado	t	t	t	**n**	t	t	t	**n**	t	t	t	**n**	t
certo?				E	E			E	E			E	E

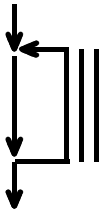


Com este método são **duas** previsões erradas
a cada vez que seqüência muda ou a cada passagem pelo loop

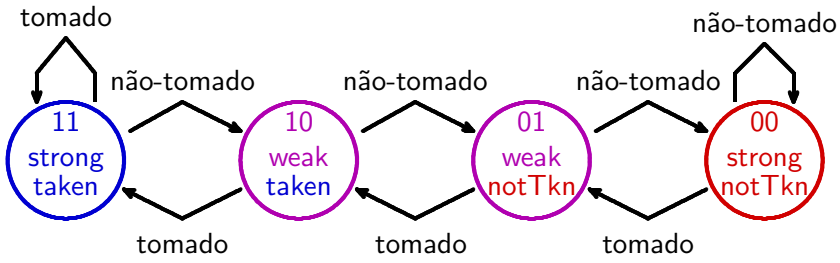
Previsores melhores: contadores de 2 bits

Contador de dois bits implementa histerese e melhora previsão:

previsão	t	t	t	t	t	t	t	t	t	t	t	t	t
estado	wt	st	st	st	wt	st	st	st	wt	st	st	st	wt
resultado	t	t	t	**n**	t	t	t	**n**	t	t	t	**n**	t
certo?				E				E				E	



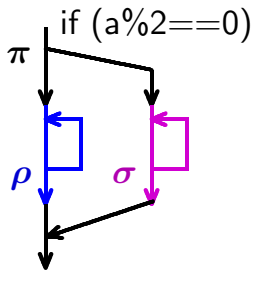
Previsões erradas caem para a metade neste exemplo.



Previsores melhores: correlação de previsões

Desvios em instruções distintas podem ser relacionados:

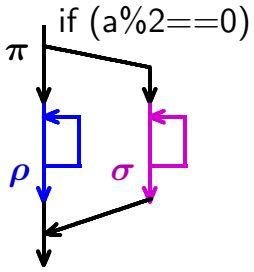
```
     if (a%2 == 0) {
f1:     for (i=0; i<a; i++) {
           ...
           // beq r1,r2,f1
        }
     } else {
f2:     for (j=N; j>a; j--) {
           ...
           // beq r3,r4,f2
        }
     }
```



desvios ρ e σ
dependem de π

Previsores melhores: correlação de previsões (cont)

```
     if (a%2 == 0) {
f1:     for (i=0; i<a; i++)
           // beq r1,r2,f1
     } else {
f2:     for (j=N; j>a; j--)
           // beq r3,r4,f2
     }
```



Portanto a idéia é:

desvios ρ e σ dependem de π

- manter história de todos os desvios recentes
  $\Longrightarrow$ história aproxima caminho seguido pelo programa
- Branch History Register (BHR) é um registrador de deslocamento que mantém resultado dos últimos **N** desvios
- PC e BHR são combinados para acessar tabela de previsão
- taxas de acerto ficam entre 80% e 90%

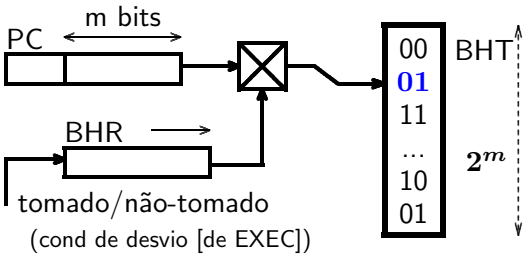
Prev. Dinâmica de Desvios – correlacionamento (i)

Correlacionamento de desvios:

Branch History Register (BHR) é um reg de deslocamento que recebe resultado da avaliação da condição de desvio

m bits do PC combinados com r bits do BHR indexam BHT

BHR registra história dos r desvios mais recentes, porque estes interferem na escolha do previsor do desvio corrente



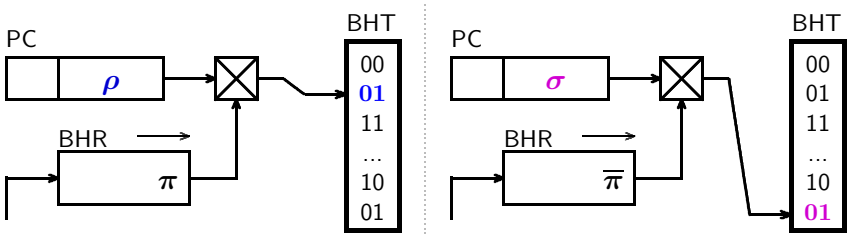
previsor deve ser "treinado" durante execução

Prev. Dinâmica de Desvios – correlacionamento (ii)

```
     if (a%2 == 0) {          π
f1:     for (i=0; i<a; i++)
           // beq r1,r2,f1    π → ρ
     } else {                 π̄
f2:     for (j=N; j>a; j--)
           // beq r3,r4,f2    π̄ → σ
     }
```

História dos desvios divide BHT em regiões

previsor deve ser "treinado" durante execução



Previsão Dinâmica de Desvios (cont)

A Tabela de Histórico de Desvios (THD) prevê quando o desvio é tomado, mas não diz para onde é o desvio

Uma Tabela de Previsão de Destino (*branch target buffer, BTB*) no estágio de busca mantém o endereço de destino, evitando a bolha.

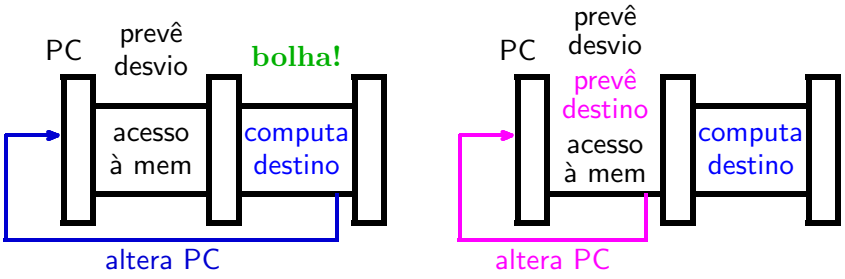
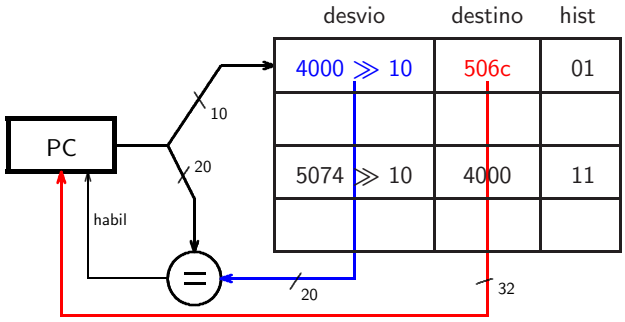


Tabela de Previsão de Destino

```
PC ──▶ 0x4000 beq r1, r2, 0x1068
       ...
       0x506c add r3, r4, r5
       0x5070 slti r8, r10, 100
       0x5074 bne r0, r8, -0x1078
```



indexa tabela com 2^m elementos com m bits do PC

Interrupções e Excessões

- Interrupções
  - ∗ geralmente com causa externa ao processador — assíncronas
  - ∗ não são relacionadas a uma instrução específica
  - ∗ Exemplos: interrupção de dispositivo; falta de energia
- Excessões
  - ∗ relacionadas à execução de uma instrução específica — síncronas
  - ∗ Exemplos: overflow, instrução inválida
- *Traps*
  - ∗ relacionadas à execução de uma instrução específica
  - ∗ rotina (hw+sw) de tratamento de excessões, chamadas de sistema

Interrupções e Excessões – arquitetura

- Registrador de Interrupção
  * vetor de bits indicando quais interrupções/excessões ocorreram
- Registrador de máscara
  * vetor de bits indica quais interr/excessões estão desabilitadas
  * escrever no registrador de máscara é instrução privilegiada
  * alguns bits podem ser atualizados em modo usuário (overflow)
  * algumas interrupções/excessões não são mascaráveis
- dois modos de execução: usuário e sistema
  * **modo usuário** tem poucos privilégios – execução de instrução privilegiada causa excessão (violação de previlégio)
  * **modo sistema** pode executar $\forall$ instrução – SO tem acesso a todos os recursos do sistema

Interrupções/Excessões Precisas

Uma interrupção ou excessão é considerada precisa se existe
uma única instrução (ou ponto de interrupção) tal que
todas as instruções anteriores àquela tenham alterado o estado,
e nenhuma instrução após (e incluindo) aquela
tenham modificado o estado.

Isso significa que a execução pode ser re-iniciada a partir do ponto de interrupção e resultados corretos serão produzidos

Interrupções Precisas

- **Interrupções Precisas** facilitam MUITO o trabalho do SO na continuação do programa, ou na depuração
- Excessão/trap
  * todas instruções antes da causadora completam
  * nenhuma das instruções após a causadora completam
  * instrução causadora **ou** completou **ou** não iniciou
  * PC aponta instrução causadora
- Interrupção
  * Mesmo que excessão, mas $\nexists$ instrução causadora
  * Deve parecer que ocorreu **entre** instruções
  * PC aponta para instrução que seria executada

Tratamento de interrupções/traps

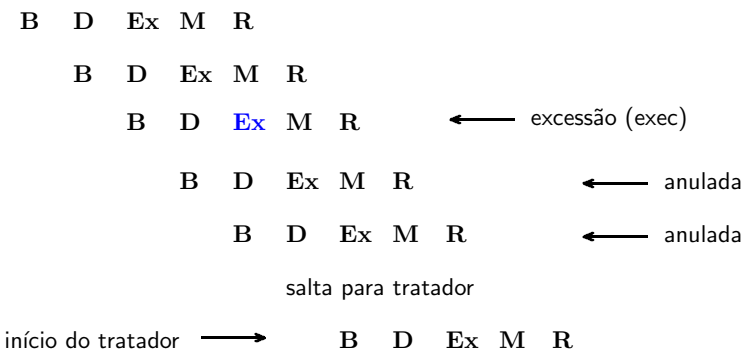
Quando ocorre interrupção/trap:

- efetua salto para rotina do SO
  - vetor de tratadores, em geral fixado pela arquitetura/CdI
    → CdI MIPS1 define 3-6, CdI x86 define 256 (?)
- computa endereço de retorno
- salva informação de estado essencial
  - PC
  - CCs (condition codes)
  - PSW (processor status word)
- troca modo de execução: usuário → sistema

Implementação de interrupções/traps

- Precisão é importante
- Interrupções e excessões simultâneas nos vários segmentos
  - interrupções de E/S – "antes" da busca
  - **busca** – falta de página, referência desalinhada, protection fault
  - **decod** – instrução ilegal ou privilegiada
  - **exec** – excessões de aritmética (overflow, divisão por zero)
  - **mem** – falta de página, referência desalinhada, protection fault
  - **res** – nenhuma

Excessão de Aritmética



- Instruções anteriores podem completar
- anula todas instruções subseqüentes (mais novas)

Resumo

- Desvios incorrem três bolhas a não ser que:
  - ⋆ circuito de decisão seja movido para DECOD e
  - ⋆ cálculo do endereço de destino seja tbém movido para DECOD*
- Desvios atrasados – sempre executa instr após desvio (em PC+4)
  - ⋆ se instrução é útil, então desvio completa em um ciclo*
  - ⋆ nem sempre é possível encontrar instrução útil
- Previsão estática de desvios – prevê não-tomado ou prevê tomado
- Previsão dinâmica de desvios – prevê com comportamento dinâmico
  - ⋆ tabela de previsão de desvios – lembra se desviou nas últimas vezes
  - ⋆ tabela de destinos – para onde desvia (qual instrução no destino)
- Interrupções e excessões complicam muito projeto de pipelines
  - ⋆ precisão é útil para SO, mas cara em termos de hardware

Exercícios

1) Complemente o desenho da aula passada incluindo todos os circuitos mostrados nos slides 5 e 10.

2) Complemente o diagrama do slide 5 com os circuitos para as para as instruções JAL e JR.

3) Como o circuito do slide 5, combinado com um dos métodos de previsão de desvios, resolve seqüências como as abaixo?

	A	B	C
end1:	add r1,r2,r3	add r1,r2,r3	add r1,r2,r3
end2:	beq r4,r0,end10	beq r4,r0,end5	beq r4,r0,end10
end3:	sub r5,r6,r7	beq r5,r0,end10	sub r6,r6,r7
end4:	j end20	j end1	beq r8,r0,end10
end5:	xor r8,r9,r9	sub r6,r7,r8	xor r9,r2,r3
end6:			beq r9,r0,end20

4) Projete uma Tabela de Previsão de Destino que armazene o endereço de destino. Qual a organização da tabela? Quais cuidados devem ser tomados com a atualização da tabela?