Hazards de Controle

Yuri Kaszubowski Lopes

UDESC

Anotações

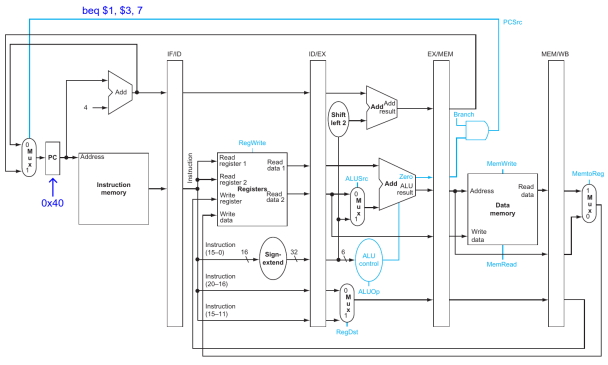
Hazards de Controle

- Por que as instruções abaixo geram um hazard de controle?
- Não sabemos qual a próxima instrução que deve ir para o pipeline
  - `0x44` ou `0x60`

```
1 0x40 beq $1, $3, 7
2 0x44 and $12, $2, $5
3 0x48 or $13, $6, $2
4 0x4C add $14, $2, $2
5 ...
6 0x60 lw $4, 50($7)
7 ...
```

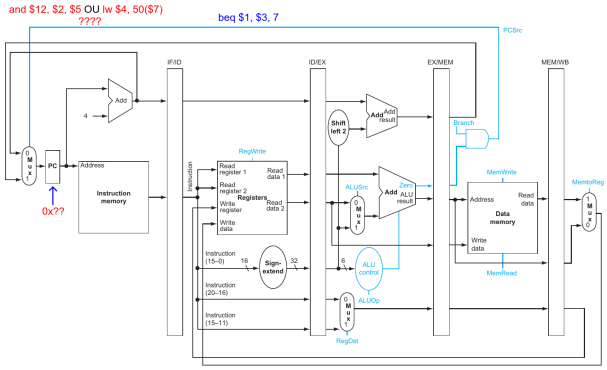

Anotações

Exemplo



Anotações

Exemplo



- O `beq` ainda não terminou de ser executado, e não sabemos se devemos executar o salto ou não!

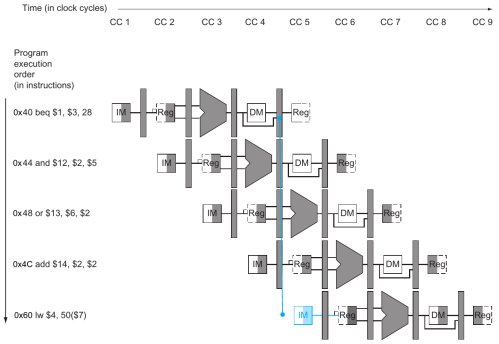
Anotações

Hazards

- Podemos fazer com que o pipeline entre em stall
  - Inserir nops
  - Ineficiente
- Outras soluções?
  - Partir do princípio que o desvio nunca é tomado
  - Sempre carregamos e executamos a próxima instrução
  - Se chegarmos a conclusão que estávamos errados, desfazemos tudo e continuamos a partir do endereço correto
  - Podemos assumir que acertamos em 50% das vezes

Anotações

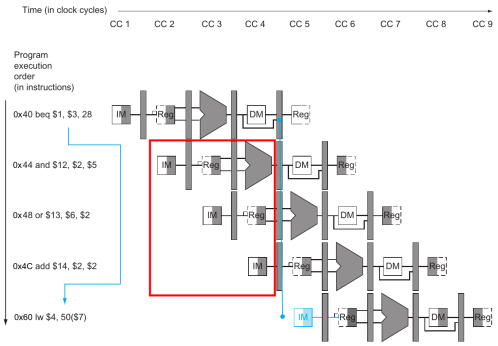
Assumir que o desvio nunca é tomado



- Continuamos a execução normalmente

Anotações

Assumir que o desvio nunca é tomado

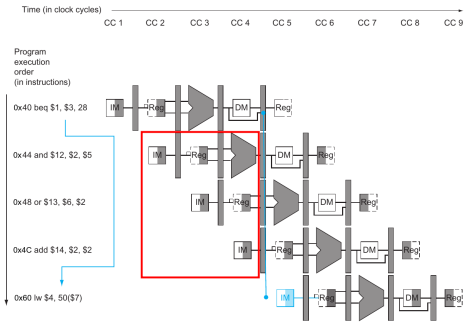


- Se concluirmos que o “chute” sobre o endereço da próxima instrução estava errado, será necessário descartar esses resultados intermediários

Anotações

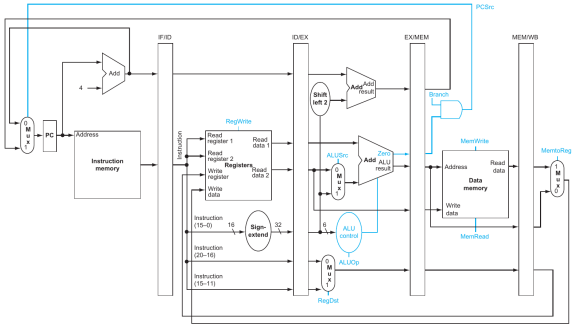
Hazards de Controle

- Caso a “previsão” esteja incorreta, temos 3 instruções que precisam ser descartadas
  - Uma no estágio `EX`, uma no `ID`, e uma no `IF`
  - Como podemos fazer esse descarte?



Anotações

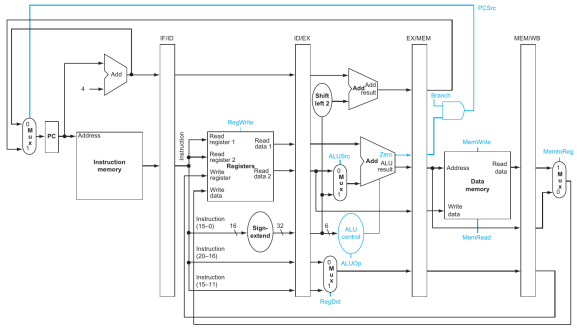
Hazards de Controle



- Injetar nop em `IF`
- Zerar sinais de controle para instruções que estão em `ID` e `EX`

Anotações

nops



- No caso de “previsão incorreta”, estamos desperdiçando três instruções
- Três nop são processados
- Três ciclos de clock inutilizados

Anotações

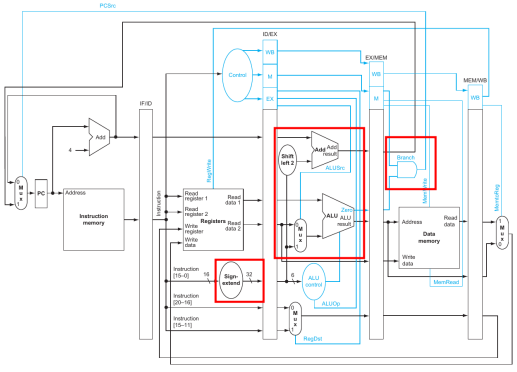
Reduzindo atrasos

- No nosso processador MIPS visto até então:
  - Por enquanto, assumimos que o resultado do branch só está pronto nos registradores `EX/MEM`
    - A instrução deve estar no estágio `MEM`
- Se conseguirmos **calcular os resultados antes**, podemos reduzir o custo de uma previsão incorreta
  - Chegamos antes a conclusão de que o desvio está incorreto ou não
  - Se a previsão está incorreta, menos trabalho executado é jogado fora

Anotações

Reduzindo atrasos

- Destaque de algumas das principais estruturas envolvidas em um branch
- Como realizar tudo no estágio `ID`?

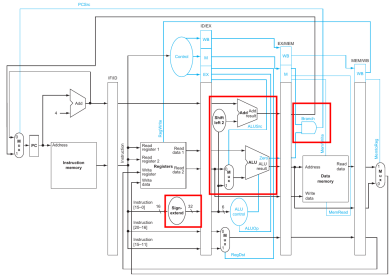


Anotações

Reduzindo atrasos: trazer branch para o estágio `ID`

Calcular o endereço do salto

- Deslocar os bits do imediato para a esquerda 2x e somar com PC + 4
- Todas essas informações já estão disponíveis no estágio `ID`
- Mudança simples, basta mover os componentes responsáveis
- Não há atraso extra
  - Esses valores podem ser calculados em paralelo, enquanto o banco de registradores busca pelas informações

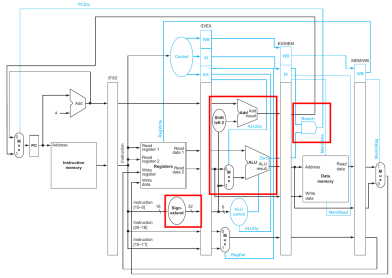


Anotações

Reduzindo atrasos: trazer branch para o estágio `ID`

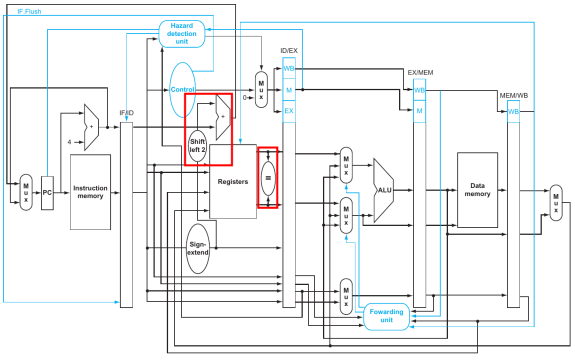
Comparação

- É mais complicada
- No momento quem faz isso é a ALU através de uma subtração
- Podemos criar um circuito rápido especialista, que faz a comparação
  - `REG1` **`xnor`** `REG2`: retorna 1 se forem iguais
  - Precisamos colocar esse circuito após a carga dos dados dos registradores



Anotações

Reduzindo atrasos: trazer branch para o estágio `ID`



- Em destaque: cálculo do endereço e `xnor` entre registradores

Anotações

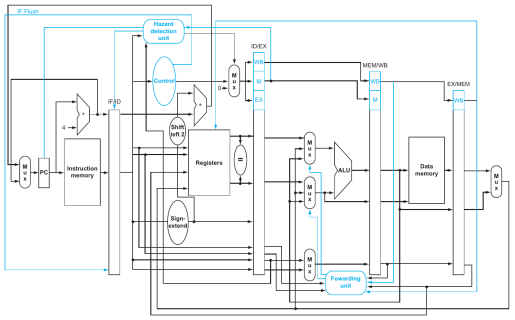
Exercício

1. Quais são os custos envolvidos nessa alteração?
   - Custo em dinheiro, tempo, complexidade
   - Parte da resposta: O estágio `ID` vai necessitar de tempo extra devido a comparação (só pode ser feita após a carga dos registradores)
2. Trazer a comparação para um estágio anterior, adicionando-se mais hardware como fizemos, é uma boa ideia em qualquer processador?

Anotações

Hazards de Dados

- Diminuímos o custo de uma previsão incorreta
  - Agora no máximo uma bolha (nop) é necessária devido ao hazard de controle
  - **Criamos novos hazards de dados!**
    - ⋆ Por quê? Onde? Como?



Anotações

Novos hazards de dados com branch em `ID`

- Criamos novos hazards de dados, agora no estágio `ID`
- Os operandos do **`beq`** podem estar sendo calculados em algum dos estágios do pipeline
- Precisamos de lógica extra de forwarding para o estágio `ID`
- Precisamos detectar hazards de dados sem solução, para inserir nops
  - e.g., um **`lw`** antes de um **`beq`**, sendo que o **`beq`** precisa do dado do **`lw`**
  - Mais complexidade na unidade de detecção de hazards
- Vamos nos contentar em saber que esses novos problemas existem, mas não vamos colocar o hardware para resolver

Anotações

Previsão dinâmica de desvios

- O custo de uma previsão incorreta pode ser excessivamente alto em uma CPU de pipeline profundo
- Podemos melhorar o sistema através de um sistema que tenta aprender se os desvios estão sendo tomados ou não
- e.g., buffer de previsão de desvios

Anotações

Buffer de Previsão de Desvios

- Um **buffer de previsão de desvios** é uma pequena memória, que contém uma tabela com o **endereço da instrução**[1], e **um bit indicando se o desvio foi tomado ou não a última vez que executamos a instrução** nesse endereço
- Especialmente útil em casos de loops
- Depois de calcular se o endereço realmente foi tomado ou não, podemos atualizar o valor no buffer
  - Para melhorar nossa escolha na próxima vez que passarmos por esta instrução
    - Pense em loops

[1] Na verdade parte do endereço

Anotações

Buffer de Previsão de Desvios

- O buffer é pequeno, e obviamente não podemos armazenar o endereço de todas instruções
- Solução: utilizar os bits mais baixos do endereço de memória (a partir do 3º bit menos significativo) para endereçar o buffer
  - Muitas instruções vão compartilhar o mesmo local no buffer
    - O único problema é que se verificarmos o buffer para uma instrução, mas o bit se refere a alguma outra, nossa probabilidade de errar é muito maior
    - Porém nem todas as instruções compartilhando o mesmo endereço do buffer são desvios
    - Ideia similar a de uma memória cache

Anotações

Exemplo: Buffer com capacidade para 8 instruções

Nesse caso, utilizamos 3 bits (do 3º ao 5º bits menos significativos) para endereçar

Endereço (binário)	Instrução
0000 0000 0000 0000	Instrução 1
0000 0000 0000 0100	Instrução 2
0000 0000 0000 1000	Instrução 3
0000 0000 0000 1100	Instrução 4
0000 0000 0001 0000	Instrução 5
0000 0000 0001 0100	Instrução 6
0000 0000 0001 1000	Instrução 7
0000 0000 0001 1100	Instrução 8
0000 0000 0010 0000	Instrução 9
0000 0000 0010 0100	Instrução 10
0000 0000 0010 1000	Instrução 11
0000 0000 0010 1100	Instrução 12
…	…

Endereço	Desviar?
000	0
001	0
010	1
011	0
100	1
101	1
110	0
111	1

Temos instruções competindo pelo mesmo lugar no buffer.

Anotações

Buffer de Previsão de Desvios

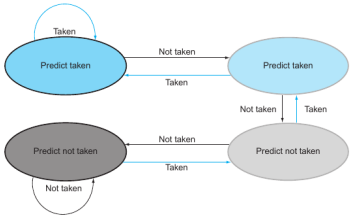
- O buffer pode ser facilmente instalado no estágio `IF` do pipeline
  - Primeiro estágio do pipeline
  - Não precisamos da instrução em si, apenas do seu endereço que está em PC

Anotações

Outros esquemas de previsão

- Buffers de previsão que utilizam mais de 1 bit
  - Pipelines mais profundos
  - Utilização de máquinas de estados simples
  - e.g., quatro estados (2 bits), necessita duas decisões erradas seguidas
  - A última decisão de um loop (quando sai) é sempre errada (ficar no loop)
    - Com um bit: a primeira decisão do while também é sempre errada (trocou para sair na última saída do while
    - Com dois bits: isto é resolvido. Como?

```
1 for (int i = 0; i < 100; i++){
2     j = 3;
3     while(j--) {
4         ...
5     }
6 }
```



Anotações

Outros esquemas de previsão

- Delayed Slots
  - Pipelines rasos e com decisões sobre desvios tomadas no início do pipeline
  - Um bom exemplo é o processador MIPS sendo analisado em aula
  - **Branch Delayed Slots**: No MIPS: um slot, abaixo de um branch com atraso, que é ocupado por uma instrução que não afeta o desvio
    - Sempre executa a instrução abaixo do *Delayed Branch* (no slot)
    - Tenta-se colocar uma instrução que é sempre executada após o branch (independentemente dele ter sido tomado ou não)
    - Instrução sempre útil
- Esquemas sofisticados conseguem uma taxa de acertos de cerca de 90%
- Reduzir o custo de um desvio quando nossa previsão está incorreta

Anotações

Exemplo em Pipelines profundos

- Considere o custo de uma previsão de branch incorreta em um Pentium 4 Prescott!
- Note que depois do Pentium 4, o número de estágios no pipeline reduziu
- Um pipeline profundo é ideal **se conseguimos mantê-lo cheio**
  - Mas é complexo, e stalls custam caro
  - Difícil manter o pipeline sempre cheio
  - Principalmente considerando que a memória principal da máquina, de onde vêm as instruções, é muito mais lenta que a CPU

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power	
Intel 486	1989	25 MHz	5	1	No	1	5	W
Intel Pentium	1993	66 MHz	5	2	No	1	10	W
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29	W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75	W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103	W
Intel Core	2006	2930 MHz	14	4	Yes	2	75	W
Intel Core i5 Nehalem	2010	3300 MHz	14	4	Yes	1	87	W
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	Yes	8	77	W

Anotações

Referências

- D. Patterson; J. Henessy. **Organização e Projeto de Computadores: Interface Hardware/Software**. 5a Edição. Elsevier Brasil, 2017.
- Andrew S. Tanenbaum. **Organização estruturada de computadores**. 5. ed. São Paulo: Pearson, 2007.
- Harris, D. and Harris, S. **Digital Design and Computer Architecture**. 2a ed. 2012.

Anotações