Pequena Apresentação Sobre Paralelismo em nível de instrução

- Primórdios do Paralelismo:
  - Pipeline
  - Múltiplas unidades funcionais (sem *pipeline*)
- Paralelismo em Nível de Instrução
  - processadores *superescalares*
  - processadores *VLIW*

Breve Histórico

- Anos 40 e 50
  - O microcódigo horizontal inicia as primeiras experiências em paralelismo
- Anos 60
  - CDC6600: Computador da Control Data Corporation, continha 10 unidades funcionais que podiam executar uma operação independentemente das outras unidades.
  - IBM 360/91: possuia menos unidades funcionais que o CDC mas usava técnicas de re-organização das instruções muito mais agressivas que o anterior.

O salto do silício

- Anos 80 e 90 O silício em fartura e as nanotecnologias permitem um grande salto de desempenho aos processadores.

Funcionamento Básico

- Paralelismo do VLIW
  - o VLIW além de oferecer *pipelining* oferece paralelismo em nível de instrução. As instruções do VLIW são formadas de várias operações, cada uma semelhante às instruções de um processador superescalar.
  - O compilador deve fornecer essas instruções ao processador, ou seja o compilador faz todo o trabalho de organização e escalonamento das operações.
- Papel do Compilador
  - As dependências entre as operações devem ser determinadas.
  - As operações que são que são independetes de outras operações que ainda não foram executadas devem ser determinadas.
  - Essas operações independentes devem ser agendadas para execução em algum momento em particular e em uma unidade funcional em particular. Devem também ser alocados registradores.

O processo de escalonamento de Instruções

- O escalonamento de instruções pode ser feito de muitas maneiras diferentes:
  - O principal mecanismo para processadores VLIW é o *trace schedule*.
  - Além dele outras abordagens tratam de blocos de código específi cos. Existem inúmeros algoritmos para *desenrolar laços*.

O Algoritmo do trace schedule

- Selecionar uma sequência de operações a serem agendadas juntas. Essa sequência é chamada de *trace*. Esses *traces* tem um limite de comprimento que é definido por vários fatores, entre os mais importantes estão os limites modulares (entrada/retorno), limites de loop e código já agendado.
- Remover esse trace do grafo de fluxo, e repassá-lo para o agendador de instruções.
- Quando o trace estiver pronto em um agendamento, passar esse agendamento para o grafo de fluxo, substituindo as operações que estavam originalmente no trace. Possivelmente será necessário fazer cópias das operações para recolocá-las, afim de não ultrapassar os limites definidos no escalonamento. Código de compensação também pode ser necessário.
- Re-iterar até que todas as operações tenham sido incluídas em algum trace e todos os traces tenham sido substituidos por um agendamento.
- Selecionar a melhor ordem linear para o código e emití-lo.

Código de compensação

- São gerados basicamente por dois fenômenos:
  - *split* Acontece quando se chega a uma operação que leva a dois ou mais possíveis caminhos, possui, logo mais que 1 sucessor.
  - *join* É o contrário, é uma operação que pode ser alcançada por mais de um caminho, possui, logo mais que um precedente.
- Os *splits* e *joins* geralmente trazem a necessidade de se fazer cópias de instruções para que o Grafo de Precedência de Dados fi que correto.

Agendador de Instruções

- Construir um grafo de precedência de dados [a] (GPD) a partir do bloco recebido.
- Atravessar o GPD e designar operações a unidades funcionais e valores a bancos de registradores.
- Criar o agendamento alocando registradores. Agrupar referências à memória para minimizar confi tos

[a] data precedence graph(DPG)