Universidade Federal do Rio de Janeiro
Pós-Graduação em Informática

Microarquiteturas Avançadas

Arquiteturas Superescalares

Gabriel P. Silva

Arquiteturas "Pipelined" com Desempenho Superior ao de uma Instrução por Ciclo

- ARQUITETURAS SUPERPIPELINED
  - Divisão de cada estágio do "pipeline" em sub-estágios com o uso de freqüências mais altas.
- ARQUITETURAS SUPERESCALARES
  - Execução de múltiplas instruções, escalonadas por "hardware" e/ou "software", concorrentemente.
- ARQUITETURAS VLIW (Very Long Instruction Word)
  - Execução de múltiplas operações, escalonadas por "software", concorrentemente.
- ARQUITETURAS MISTAS
  - Combinações dos modelos acima

Arquiteturas Superescalares

▪ Organizadas internamente como múltiplos "pipelines" e com banco de registradores com múltiplas portas de leitura e de escrita, com múltiplas instruções iniciadas a cada ciclo. O grau de concorrência de instruções situa-se na prática entre 2 e 4

▪ As instruções são despachadas para execução somente quando não violam regras de dependência de dados ou de controle e quando não existem conflitos estruturais.

• O escalonamento das instruções pode ser feito por "software" ou por "hardware".

• O código objeto para estas arquiteturas é compatível com o de arquiteturas escalares convencionais.

• Processamento das instruções pode seguir três possíveis modelos:
  - despacho "em-ordem" + término "em-ordem"
  - despacho "em-ordem" + término "fora-de-ordem"
  - despacho "fora-de-ordem" + término "fora-de-ordem"

Arquiteturas Superescalares

▪ Capacidade para realizar busca e decodificação de múltiplas instruções por ciclo;

▪ Existência de uma *Janela de Instruções* que isola os estágios de busca e decodificação dos estágios de execução propriamente dita da instrução ➔ modelo de despacho fora-de-ordem:

  - Janela Centralizada
  - Estações de Reserva (Algoritmo de Tomasulo)

▪ Esquemas eficientes de predição dinâmica de desvios;

▪ Recuperação do estado da máquina em caso de exceções ou de previsões erradas de desvios ➔ Reoder Buffer;

▪ Remoção de dependências de dados ➔ Scoreboarding ou Renomeação Dinâmica (Reorder Buffer);

▪ Lógica para despacho concorrente de instruções armazenadas na Janela de Instruções;

Arquiteturas Superescalares

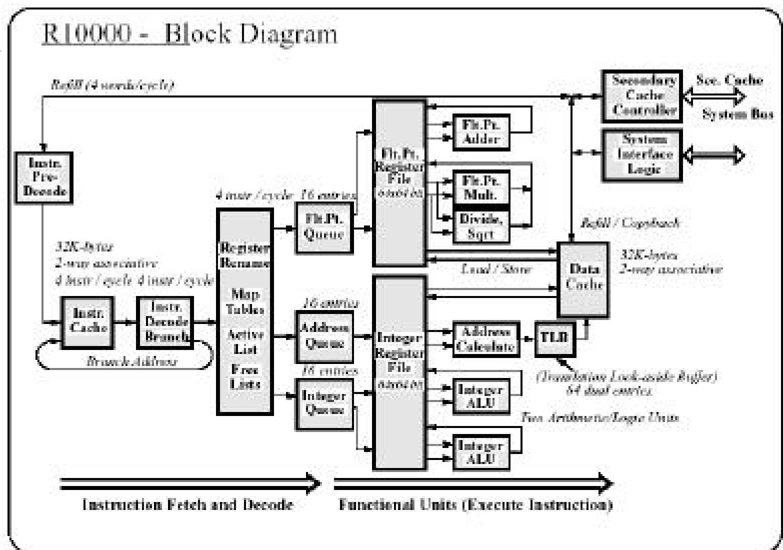
▪ **Múltiplos barramentos para comunicação de operandos e resultados e Banco de Registradores com múltiplas portas de leitura e de escrita, incluindo a existência de lógica de arbitração se o número de recursos é menor que o máximo possivelmente necessário;**

▪ **Múltiplas unidades funcionais: ALU, Ponto Flutuante, Desvio, Load/Store;**

▪ **Suporte para tratamento de dependência de dados entre instruções de *load* e *store*;**

▪ **A figura a seguir mostra um diagrama de blocos de um estrutura típica de uma arquitetura superescalar.**

MIPS R10000



Busca e Decodificação

▪ A arquitetura superescalar só é efetiva se a taxa média com que as instruções são buscadas e decodificadas for superior à taxa média com que instruções são executadas.

▪ Esquema eficiente de predição de desvios é fundamental.

▪ Necessidade de se dispor de um barramento largo de acesso ao cache de instruções e de um decodificador de instruções capaz de decodificar múltiplas instruções simultaneamente:

  ▪ Decodificador para 4 instruções produz um desempenho 20-25% superior que um decodificador para apenas 2 instruções.

  ▪ Um decodificador de múltiplas instruções demanda a realização da busca de vários operandos em paralelo → o Banco de Registradores e/ou o conjunto de registradores rascunho, utilizado para recuperação do estado em ordem da arquitetura (Buffer de Reordenação, Buffer de História, etc.), com múltiplas portas de leitura.

Degradação do Desempenho da Operação de Busca de Instruções

▪ Falha no acesso ao cache de instruções:
  ▪ Bem mais do que um ciclo é gasto para a busca do próximo conjunto de instruções.

▪ Predição errada de desvios:
  ▪ As instruções buscadas antecipadamente mostram-se inúteis e novas instruções terão que ser buscadas.

  ▪ Alternativa possível: busca simultânea de instruções de dois ou mais caminhos possíveis do código a cada desvio condicional → aumento no custo de hardware (cache de instruções com múltiplas portas de acesso)

▪ Desalinhamento do endereço das instruções alvo de um desvio em relação ao início de um bloco de cache:
  ▪ Menos instruções úteis são efetivamente buscadas → *"slots"* de decodificação podem ficar subtilizados se a unidade de busca não for capaz de realinhar as instruções antes de passá-las à unidade de decodificação.

Trace Cache

- Esquema proposto por Rotenberg, Bennet e Smith (University of Wisconsin) para aumentar a eficiência das operações de busca de instruções.

- Utiliza uma memória cache adicional para armazenar os *"traces"* das instruções mais recentemente executadas, que é consultada pela unidade de busca de instruções.

- Em situações de "loop", o *Trace Cache* tende a ter uma alta taxa de acerto.

- Trabalhos bastante recentes tem proposto o uso de Trace Caches para implementar máquinas VLIW com código compatível com máquinas não VLIW:
  - Estas arquiteturas, denominadas arquiteturas VLIW com *trace scheduling* dinâmico, implementam 2 máquinas: uma máquina RISC convencional baseada em um pipeline simples e uma máquina VLIW.
  - A máquina VLIW só entra em operação quando o código de um *"trace"*, armazenado no *Trace Cache*, é executado pela segunda vez.
  - A compactação de código é gerada por hardware a partir do código armazenado no *Trace Cache*.

Janela de Instruções

- A Janela de Instruções armazena o resultado da decodificação das instruções e isola o estágio de busca e decodificação de instruções dos estágios de execução propriamente dita das instruções.

- Pode ser implementada de forma Centralizada *(Janela Centralizada ou "Central Window")* ou distribuída pelas unidades funcionais (Estações de Reserva *ou "Reservation Stations")*.

- A janela centralizada armazena todas as instruções já decodificadas e ainda não despachadas para execução, enquanto que cada Estação de Reserva só armazena as instruções destinadas à Unidade Funcional associada a ela.

- A Janela Centralizada pode ser, em geral, dimensionada com uma capacidade menor do que a capacidade total das diversas Estações de Reserva.

Janela de Instruções

- Na implementação com Janela Centralizada mais de uma instrução pode ser despachada por ciclo para as diferentes Unidades Funcionais enquanto que cada Estação de Reserva pode despachar no máximo uma instrução por ciclo para sua Unidade Funcional.

- A Janela Centralizada deve ser capaz de suportar instruções de diferentes formatos (diferentes tamanhos) enquanto que as Estações de Reserva podem ser dedicadas apenas ao formato das instruções executadas pela sua Unidade Funcional.

- A lógica de controle das implementações baseadas em Janela Centralizada é usualmente mais complexa que a das Estações de Reserva, já que é necessário administrar diferentes tipos de instruções e Unidades Funcionais, disparar simultaneamente mais de uma instrução para as diferentes Unidades Funcionais, implementar o mecanismo de *tag* de seqüencialização , etc.

Janela Centralizada

- A Janela Centralizada é uma memória com 16 a 32 entradas, inferior ao número total de entradas numa implementação alternativa com Estações de Reserva.

- O decodificador de múltiplas instruções armazena na Janela Centralizada todas as instruções e operandos.

- O tamanho da Janela Centralizada, em número de entradas, é dimensionado para evitar que o decodificador tenha que suspender sua operação por não haver mais espaço para armazenar instruções e operandos.

- As seguintes funções devem ser desempenhadas pela lógica de controle da Janela Centralizada;
  - Identificar as instruções que estão prontas para serem despachadas para execução (instruções livres de dependência);
  - Selecionar dentre as instruções prontas as mais antigas para serem despachadas para cada Unidade Funcional;
  - Despachar as instruções selecionadas se a Unidade Funcional correspondente estiver livre;
  - Liberar prontamente as entradas correspondentes às instruções despachadas para uso do decodificador;

Estações de Reserva

- As Estações de Reserva são implementadas como memórias com poucas entradas (2 a 8) associada a cada Unidade Funcional da arquitetura.

- O decodificador de múltiplas instruções armazena em cada Estação de Reserva as instruções e operandos relativos àquela Unidade Funcional.

- O tamanho de cada Estação de Reserva em número de entradas é dimensionado para evitar que o decodificador tenha que suspender sua operação por não haver mais espaço para armazenar instruções relativas a uma dada Unidade Funcional.

- A lógica de controle das Estações de Reserva pode ser bastante simplificado se, localmente, for adotada a política de despacho em ordem, considerando apenas a instrução mais antiga como candidata a ser despachada ➔ a Estação de Reserva funciona como sendo uma FIFO ➔ A queda de desempenho provocada por esta simplificação é, em geral, pequena, na faixa de 1% a 2%.

Estações de Reserva

- As seguintes funções devem ser desempenhadas pela lógica de controle das Estações de Reserva:

  - Identificar as instruções que estão prontas para serem despachadas para execução (instruções livres de dependência);

  - Selecionar dentre as instruções prontas a mais antiga para ser despachada;

  - Despachar a instrução selecionada se a Unidade Funcional estiver livre;

  - Liberar prontamente a entrada correspondente à instrução despachada para uso do decodificador;

  - Monitorar os tags que transitam nos barramentos de resultados para, quando houver correspondência, armazenar estes resultados no espaço reservado a eles nas estações de reserva ➔ mecanismo de "bypass" ➔ grande número de comparadores.

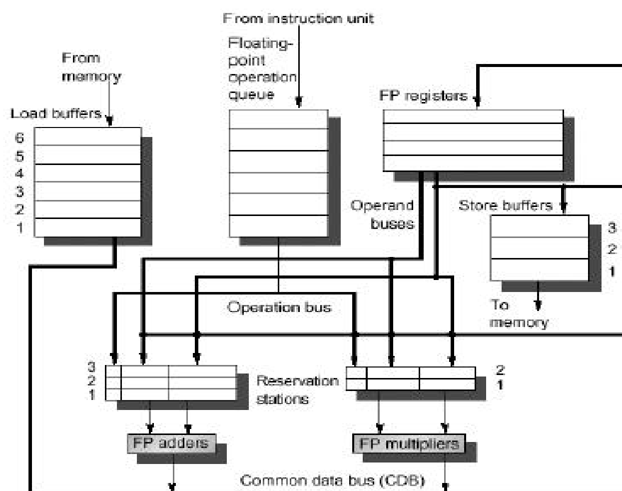
Algoritmo de Tomasulo

- O algoritmo de Tomasulo, desenvolvido para o sistema IBM 360/91 em 1967, ainda é o mais eficiente para o despacho seqüencial de instruções fora-de-ordem.

- Também conhecido como algoritmo associativo, o algoritmo de Tomasulo realiza a difusão dos resultados produzidos pelas unidades funcionais através de um barramento global (CDB) que interconecta os componentes do sistema: unidades funcionais, estações de reserva e banco de registradores.

- Diversos processadores (PowerPC, Pentium, SPARC64) da atualidade utilizam variações deste algoritmo para o despacho de múltiplas instruções por ciclo.

Algoritmo de Tomasulo

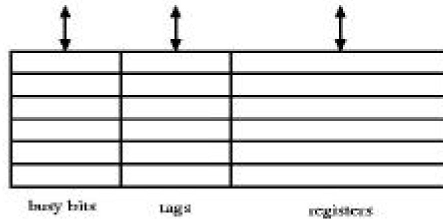


O Banco de Registradores

▪ Cada entrada no banco de registradores do IBM 360/91 é formada por três campos. O primeiro campo, denominado "busy bit", indica se o valor armazenado está atualizado.

▪ Caso o valor do registrador esteja desatualizado, o segundo campo, denominado "tag", aponta para a estação de reserva contendo a instrução mais recente que vai produzir este valor.

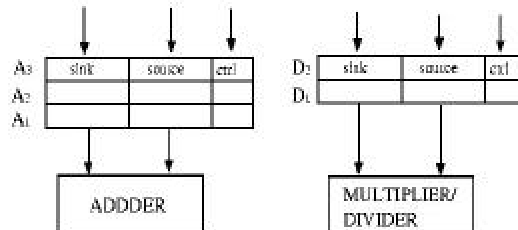
▪ Finalmente, o terceiro campo é o registrador propriamente dito.



Algoritmo de Tomasulo

▪ A unidade de ponto flutuante do processador é constituída por um somador e por uma unidade para multiplicação /divisão.

▪ O somador e o multiplicador possuem três e duas estações de reserva, respectivamente.



Estações de Reserva

▪ São "buffers" colocados entre o estágio de despacho e o de execução que permitem o despacho de instruções mesmo que os operandos não estejam disponíveis.

▪ Ao invés de transferir o valor do operando, o algoritmo associativo transfere para a estação de reserva o "tag" correspondente à estação que armazena a instrução que produzirá o resultado esperado.

▪ Uma vez despachada, a instrução permanece na estação de reserva até que seja concluída. Nesse momento, ela é descartada, liberando a estação.

▪ A instrução será iniciada quando seus operandos e a unidade funcional apropriada estiverem disponíveis.

▪ Graças ao esquema de despacho para a estações de reserva, instruções no IBM 360/91 podem ser iniciadas, executadas e terminadas fora da ordem original em que aparecem no código objeto.

O Banco de Registradores

▪ Antes de transferir a instrução corrente para a estação de reserva, o algoritmo de Tomasulo examina os "busy bits" dos registradores fonte.

▪ Se estiveram atualizados, o conteúdo dos registradores é transferido para a estação de reserva. Caso contrário, o campo "tag" do registrador fonte é copiado no campo "tag" da estação de reserva onde a instrução atual está sendo alocada.

▪Em seguida, o "busy bit" do registrador destino da instrução corrente é modificado, indicando para as instruções subseqüentes que o conteúdo daquele registrador não está atualizado.

▪ Finalmente, o algoritmo associativo transfere para o campo "tag" do registrador a identificação da estação de reserva que recebeu a instrução corrente.

A Difusão de Resultados no CDB

- Toda vez que uma unidade funcional conclui uma operação, o resultado é difundido, juntamente com o seu "tag", através de um barramento global, para o banco de registradores e estações de reserva. O barramento global é denominado CDB ("Common Data Bus") .

- A difusão é associativa: o resultado da operação é transmitido associativamente para os demais componentes, junto com o "tag" que identifica a estação de reserva que armazena a instrução que produziu o dado.

- Os componentes do sistema monitoram continuamente o barramento de resultados, verificando se o "tag" ora transmitido coincide com o campo de "tag" aguardado.

- Ocorrendo a coincidência, o resultado é copiado no campo correspondente da estação de reserva ou do banco de registradores. Caso contrário é ignorado. Essa transferência associativa deu origem ao termo "algoritmo de despacho associativo''.

Resolução das Dependências de Dados

- Quando da ocorrência de dependências falsas o algoritmo de Tomasulo despachando a instrução para uma estação, fazendo cópia dos operandos fonte, caso disponíveis.

- Se os recursos (unidade funcional e operandos fonte) estiverem disponíveis, a instrução é iniciada imediatamente, antes mesmo do término de suas sucessoras. As instruções subseqüentes são despachadas, mesmo que uma determinada instrução não possa ser despachada.

- No caso de dependências verdadeiras, a instrução sucessora é despachada para uma estação de reserva, junto com o "tag" indicando qual das instruções precedentes que já foram despachadas, irá gerar o resultado que será o operando fonte.

- Ao simular automaticamente um número infinito de registradores virtuais para a renomeação, o algoritmo de Tomasulo torna desnecessárias as atualizações dos registradores destino de instruções apresentando dependências falsas.

Gerenciamento das Dependências de Dados Bit de Scoreboard

▪ Portanto, para que este método se torne operacional é preciso que a lógica de controle da Janela de Instruções seja capaz de monitorar os identificadores de registradores que transitam nos barramentos de resultados para armazenar esses resultados no espaço destinado a eles nas entradas correspondentes da Janela de Instruções, implementando um mecanismo de *bypass*.

▪ É necessário utilizar-se um grande número de comparadores, capazes de realizar em paralelo a comparação do identificador presente nos barramentos de resultados com os identificadores armazenados em cada entrada da Janela de Instruções.

▪ O esquema baseado no uso do bit de *scoreboard* é razoavelmente simples, mas limita a exploração de paralelismo entre as instruções na ocorrência de dependências de saída, pois bloqueia a ação do decodificador quando este tipo de dependência é detectado entre instruções. O esquema a ser descrito a seguir, baseado na renomeação dinâmica dos registradores, evita este problema.

Recuperação do Estado do Processador em Caso de Exceção ou Predição Errada de Desvio

▪ Dada uma seqüência de instruções e considerando-se o modelo de despacho e conclusão de instruções fora de ordem, podem ser definidos os seguintes estados do processador:

▪ Estado em ordem:
  ▪ Constituído por todas as atribuições de valores já realizadas pela seqüência em ordem mais longa de instruções já concluídas. É o estado que precisa ser recuperado quando uma exceção ocorre.

▪ Estado futuro ou *"look-ahead"*:
  ▪ Constituído por todas as atribuições já feitas ou pendentes a partir da primeira instrução não completada na seqüência em ordem.

▪ Estado arquitetural:
  ▪ Constituído pelas últimas atribuições feitas a cada registrador na seqüência em ordem de instruções. É o estado que deve ser percebido por uma instrução executada após a seqüência.

Recuperação do Estado do Processador em Caso de Exceção ou Predição Errada de Desvio

Seqüência em ordem de Instruções	Estado em ordem	Estado futuro	Estado arquitetural
(1) R3 <= ...	(1) R3 <= ...		
(2) R7 <= ...			
(3) R8 <= ...	(3) R8 <= ...		
(4) R7 <= ...	(4) R7 <= ...		(4) R7 <= ...
(5) R4 <= ...		(5) R4 <= ...	(5) R4 <= ...
(6) R3 <= ...		(6) R3 <= ...	
(7) R8 <= ...		(7) R8 <= ...	(7) R8 <= ...
(8) R3 <= ...		(8) R3 <= ...	(8) R3 <= ...

Buffer de Reordenação

- O problema de recuperação do estado em ordem de um processador surge tanto quando ocorrem exceções como também quando se detecta uma predição errada de desvio condicional.

- O Buffer de Reordenação é implementado como uma memória fifo associativa, tipicamente com cerca de 32 entradas.

- Quando uma instrução é decodificada, é alocada a ela uma entrada no final da fila do Buffer de Reordenação.

- Quando uma instrução é completada, o valor do resultado gerado por ela é escrito na posição a ela alocada no Buffer de Reordenação (busca associativa) e não no Banco de Registradores.

- Quando uma instrução atinge a primeira posição da fila armazenada no Buffer de Reordenação, todas as instruções anteriores a ela na seqüência em ordem necessariamente já completaram.

Buffer de Reordenação

- Se esta instrução já tiver sido completada sem geração de exceções, o valor do resultado por ela gerado é escrito no Banco de Registradores e a instrução é eliminada da fila.

- Caso contrário, a instrução permanece na fila até ser completada e o decodificador de instruções permanece funcionando enquanto houver entradas livres no Buffer de Reordenação.

Buffer de Reordenação

- O estado em ordem dos registradores do processador é dado pelo Banco de Registradores, já que este só é alterado como resultado de instruções que completaram em ordem.

- O estado futuro é dado pelo Buffer de Reordenação, que fica armazenando temporariamente todas as alterações em registradores pendentes ou já realizadas pelas instruções que completaram após a primeira instrução ainda não completada segundo a seqüência em ordem.

- O estado arquitetural resulta da combinação do estado em ordem e da atualização mais recente (mais próxima do final da fila) de cada registrador no Buffer de Reordenação.

- O Buffer de Reordenação pode ser utilizado para recuperar o estado em ordem tanto quando ocorrem exceções como quando ocorrem predições erradas de desvios.

Buffer de Reordenação

- O Buffer de Reordenação deve ter uma entrada alocada a toda instrução decodificada. Para as instruções de desvio, o valor do contador de programas da instrução alvo do desvio deve ser armazenado nessa entrada. Para as instruções que modificam registradores, o valor escrito e a identificação do registrador devem ser armazenados.

- Cada entrada do Buffer de Reordenação deve conter bits de status informando se a instrução foi completada, se ocorreu alguma exceção ou se a instrução correspondente é de desvio e que tipo de predição de desvio foi utilizado (desvio tomado ou não tomado).

- Há sempre um valor do contador de programas associado pelo Buffer de Reordenação à instrução situada no início da fila. Esse valor é incrementado sempre que uma instrução é descartada no início da fila e é alterado sempre que é encontrada uma instrução de desvio efetivamente tomado no início da fila.

Buffer de Reordenação

- Para recuperar o estado em ordem quando ocorre uma predição errada de desvio, basta aguardar que a instrução de desvio correspondente atinja à primeira posição da fila. Nesse ponto, o Buffer de Reordenação é totalmente descartado e o valor do contador de programas é restaurado com o valor do contador de programas referente à instrução alvo do desvio armazenado nessa entrada se uma predição de desvio não tomado havia sido feita.

- Em caso contrário, o contador de programas é restaurado pelo simples incremento do valor do contador de programas associado à instrução pelo Buffer de Reordenação.

- Para recuperar o estado em ordem quando ocorre uma exceção basta aguardar que a instrução causadora da exceção atinja à primeira posição da fila. Nesse ponto, o conteúdo do Buffer de Reordenação é descartado e o valor do contador de programas é aquele associado pelo Buffer de Reordenação à instrução situada no início da fila, causadora da exceção.

Buffer de Reordenação

- O esquema descrito é suficiente para que o estado em ordem seja recuperado desde que uma instrução de *store* só seja executada quando todas as instruções anteriores a ela na seqüência em ordem já tenham completado

- Uma vez que o Buffer de Reordenação é incapaz de recuperar o estado em ordem na memória, após uma operação de escrita realizada por uma instrução de *store*, eventualmente executada fora da ordem do programa.

- Tipicamente o Buffer de Reordenação deve ter capacidade para armazenar tantas instruções quanto for o número máximo de instruções que possam estar em execução (especulativa ou não) no processador.

- Este número é função do número de posições na janela de instruções, número e latência das unidades funcionais, quantidade de desvios que podem ser executados especulativamente, entre outros fatores.

Buffer de Histórico

- Também é uma estrutura fifo que armazena para cada entrada o valor do contador de programas correspondente à instrução decodificada, o registrador destino alterado por ela, o valor do registrador antes da alteração e bits de status indicando a ocorrência de exceção ou se a instrução é uma instrução de desvio com predição errada.

- É uma alternativa ao uso do Buffer de Reodenação para recuperação do estado em-ordem quando da ocorrência de exceções.

- Com esse esquema, os operandos fonte das instruções são sempre lidos do Banco de Registradores principal.

- O novo valor do registrador destino produzido pela instrução é escrito no banco de registradores principal, mas o valor anterior do registrador continua armazenado no Buffer de História.

- Sempre que a instrução na primeira posição na fila do Buffer de História completa ela é descartada da fila.

Buffer de Histórico

- Quando a instrução que ocupa a primeira posição da fila está marcada como gerando uma ocorrência de uma exceção, o estado em ordem é recuperado, percorrendo-se o Buffer de História do final para o início, copiando-se os valores dos registradores armazenados de volta no banco de registradores principal.

- Procedimento semelhante pode ser usado para recuperar o estado em ordem em caso de execução especulativa com erro na predição de um desvio.

Gerenciamento das Dependências de Dados Bit de Scoreboard

- Este método se baseia na associação de um único bit (bit de *scoreboard*) com cada registrador para indicar se existe alguma instrução ainda não completada que modifica aquele registrador.

- O bit de *scoreboard* associado a cada registrador é ativado quando uma instrução que escreve nesse registrador é decodificada. O bit é desativado novamente quando a operação de escrita no registrador se consuma.

- Como só há um bit de *scoreboard* por registrador, apenas uma instrução que modifique aquele registrador pode estar pendente. O decodificador suspende sua operação quando uma instrução que altera um registrador com o bit de scoreboard já ativado é detectada → elimina as dependências de saída.

Gerenciamento das Dependências de Dados
Bit de Scoreboard

- Quando o decodificador detecta uma instrução, que usa como operando fonte um registrador com o bit de *scoreboard* ativado, esta instrução é armazenada na Janela de Instruções juntamente com os valores conhecidos dos registradores relacionados com os demais operandos e com a identificação do registrador que possui o bit de *scoreboard* ativado.

- O armazenamento na Janela de Instruções, no momento da decodificação, dos valores conhecidos dos demais operandos evita anti-dependências já que essas cópias não são alteradas por instruções subseqüentes.

- Quando uma instrução é completada, a identificação do seu registrador destino é comparada com as identificações armazenadas nas diversas entradas da Janela de Instruções. Para todas as instruções em que houve correspondência, o valor escrito no registrador é copiado para a entrada correspondente da Janela de Instruções, fazendo com que as dependências diretas sejam cumpridas corretamente.

Gerenciamento das Dependências de Dados
Renomeação de Registradores com Reorder Buffer

- O Buffer de Reordenação, com busca associativa e organizado como uma memória fifo, pode fazer automaticamente o trabalho de renomeação dinâmica de registradores em arquiteturas superescalares.

- Quando uma instrução é decodificada, o identificador do seu registrador destino é associado a uma nova entrada do Buffer de Reordenação (renomeação). Um identificador único, associado por hardware a esse resultado, é armazenado nessa entrada do Buffer de Reordenação.

- Os operandos fonte de cada instrução são procurados, a pedido da Unidade de Busca e Decodificação de Instruções, no Buffer de Reordenação, de forma associativa, e no Banco de Registradores, por acesso direto.

- Não havendo entrada no Buffer de Reordenação que corresponda à identificação do registrador fonte, o valor deste é fornecido pelo Banco de Registradores.

Gerenciamento das Dependências de Dados
Renomeação de Registradores com Reorder Buffer

- Havendo uma entrada no Buffer de Reordenação com a identificação do registrador fonte procurado, o valor do registrador é obtido do Buffer de Reordenação. Se o valor estiver pendente, o identificador correspondente é devolvido para a instrução → respeita as dependências diretas e evita anti-dependências.

- Havendo mais de uma entrada no Buffer de Reordenação com a identificação do registrador procurado, a que estiver mais próxima do final da fila é usada por ser o valor mais recente a ser atribuído ao registrador → são eliminadas as dependências de saída entre as instruções.

- Quando um novo resultado é produzido por uma instrução, ele substitui o identificador correspondente no Buffer de Reordenação e nas instruções contendo este mesmo identificador na Janela de Instruções.

Gerenciamento das Dependências de Dados
Renomeação de Registradores com Reorder Buffer

- Também neste método a lógica de controle da Janela de Instruções deve ser capaz de monitorar os identificadores que transitam nos barramentos de resultados para, quando houver correspondência, armazenar esses resultados no espaço destinado a eles nas entradas correspondentes da Janela de Instruções, implementando um mecanismo de *bypass*.

- Mais uma vez, uma implementação eficiente deste mecanismo requer o uso de um grande número de comparadores.

Unidade Funcional de Load/Store

- Instruções de *Store* só podem ser executadas depois que todas as instruções anteriores a ela, na ordem definida pelo programa, já completaram ➔ preserva o estado em ordem nos diferentes níveis de hierarquia do sistema de memória.

- As instruções de *Load* podem ser despachadas para execução fora de ordem em relação às instruções de *Store*, desde que elas não possuam dependência de dados em relação às instruções de *Store* pendentes.

- A Unidade Funcional de *Load/Store* é freqüentemente implementada com três componentes básicos: Estação de Reserva, Unidade de Endereçamento e *Store Buffer*.

- As instruções na Estação de Reserva são enviadas em ordem para a Unidade de Endereçamento, que calcula o endereço efetivo de memória a ser utilizado pelas instruções de *Load/Store*.

Unidade Funcional de Load/Store

- O *Store Buffer* retém as instruções de *Store* despachadas com o endereço de memória já calculado e o valor do dado a ser escrito, até que todas as instruções anteriores a ela já tenham sido completadas, ou seja, até que a instrução de *Store* atinja a primeira posição da fila no Buffer de Reordenação.

- Para as instruções de *Load*, é verificado se o endereço de memória gerado pela Unidade de Endereçamento não está presente no *Store Buffer*.

- Caso não esteja, as instruções de *Load* realizam acesso à memória fora de ordem em relação às instruções de *Store* armazenadas no *Store Buffer* (*"Load Bypassing*).

- Se o endereço de memória de uma instrução de *Load* está presente no *Store* Buffer, o valor a ser armazenado em memória pela instrução de *Store* presente no *Store Buffer* é o valor usado pela instrução de Load, que, nesse caso, nem precisa realizar o acesso à memória (*"Load Bypassing + Forwarding"*).