Universidade Federal do Rio de Janeiro
Informática DCC/IM

Arquitetura de Computadores II

Arquiteturas Superescalares

Gabriel P. Silva

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

- **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 Multithreading e SMT**
  - Executam instruções de mais de um fluxo (threads) simultaneamente.
- **Arquiteturas Multicore**
  - Combinação de vários processadores de um mesmo tipo (acima descrito) em uma única pastilha.

Arquiteturas Superescalares

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

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

- O escalonamento das instruções pode ser feito por "software" e/ou por "hardware".

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

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;**

Arquiteturas Superescalares

- **Recuperação do estado da máquina em caso de exceções ou de previsões erradas de desvios ➔ *Reorder Buffer*;**

- **Remoção de dependências de dados ➔ *Scoreboarding*, Tomasulo e/ou Renomeação Dinâmica;**

- **Lógica de despacho concorrente para as instruções armazenadas na Janela de Instruções;**

Arquiteturas Superescalares

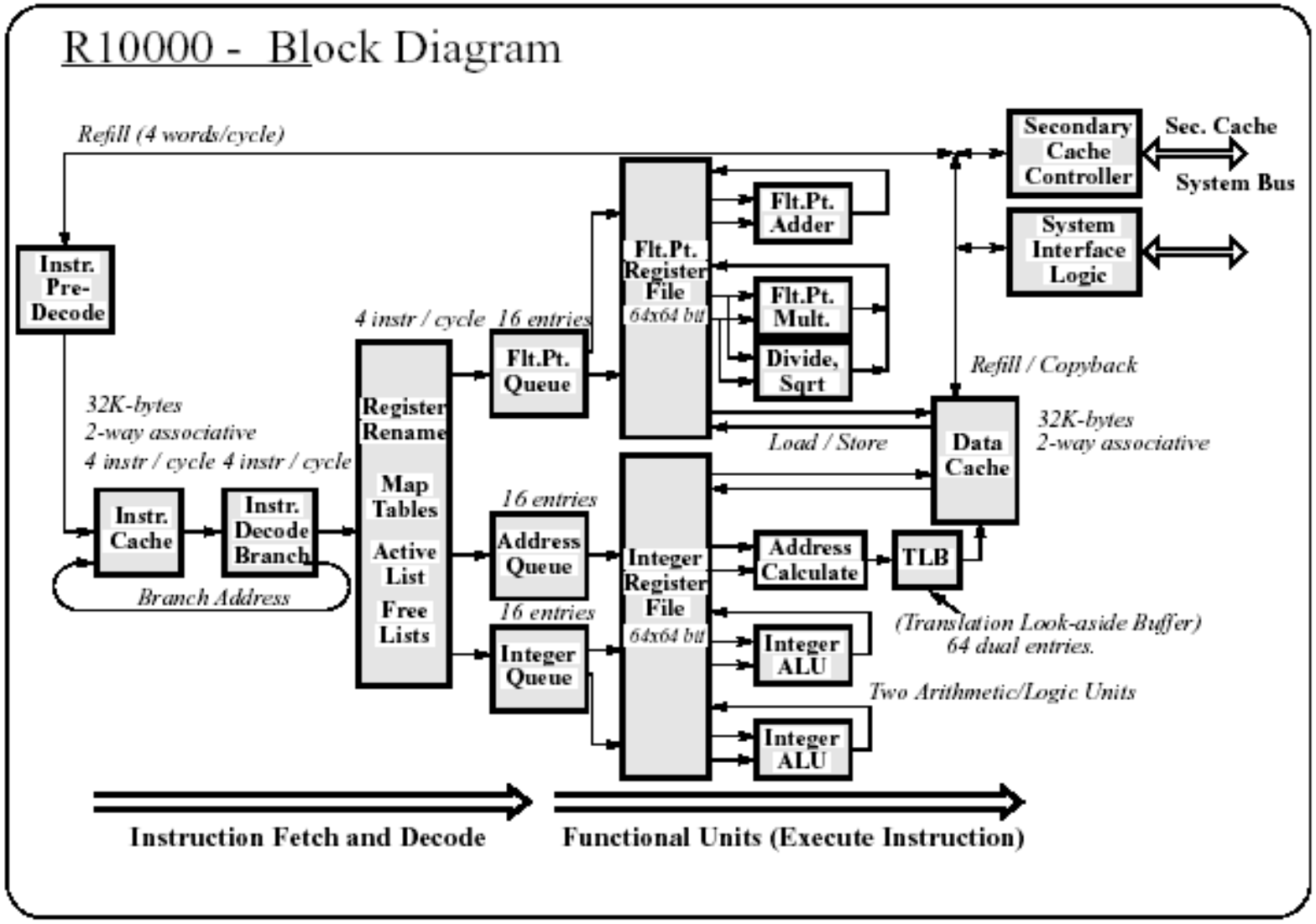
- **Múltiplos barramentos para comunicação de operandos e resultados e Banco de Registradores com múltiplas portas de leitura e de escrita, incluindo a existência de lógica de arbitração se o número de recursos é menor que o máximo possivelmente necessário;**

- **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


R10000 - Block Diagram
Refill (4 words/cycle)
Secondary Cache Controller
Sec. Cache
System Bus
System Interface Logic
Instr. Pre-Decode
4 instr / cycle
16 entries
Flt.Pt. Register File
64x64 bit
Flt.Pt. Adder
Flt.Pt. Mult.
Divide, Sqrt
Flt.Pt. Queue
Refill / Copyback
32K-bytes
2-way associative
4 instr / cycle 4 instr / cycle
Register Rename
Map Tables
Active List
Free Lists
Load / Store
Data Cache
32K-bytes
2-way associative
Instr. Cache
Instr. Decode Branch
Branch Address
16 entries
Address Queue
Integer Register File
64x64 bit
Address Calculate
TLB
(Translation Look-aside Buffer)
64 dual entries.
16 entries
Integer Queue
Integer ALU
Two Arithmetic/Logic Units
Integer ALU
Instruction Fetch and Decode
Functional Units (Execute Instruction)

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 de acesso ao cache de instruções com largura adequada e de um decodificador de instruções capaz de decodificar múltiplas instruções simultaneamente.

- Um decodificador de múltiplas instruções demanda a realização da busca de vários operandos em paralelo ➔ o banco de registradores com múltiplas portas de leitura.

Degradação do Desempenho da Operação de Busca de Instruções

- **Falha no acesso à 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).**

Degradação do Desempenho da Operação de Busca de Instruções

- **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.**

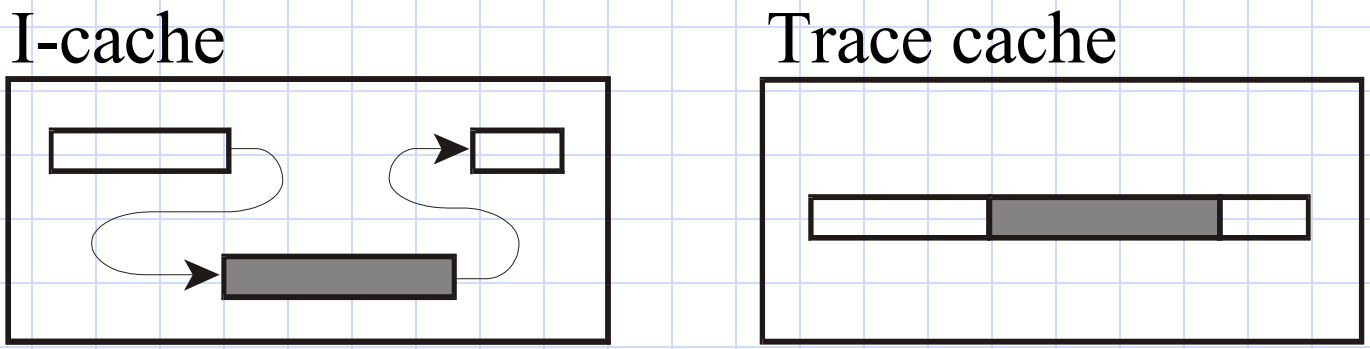
- **Em situações de "loop", o *Trace Cache* tende a ter uma alta taxa de acerto.**

Trace Cache

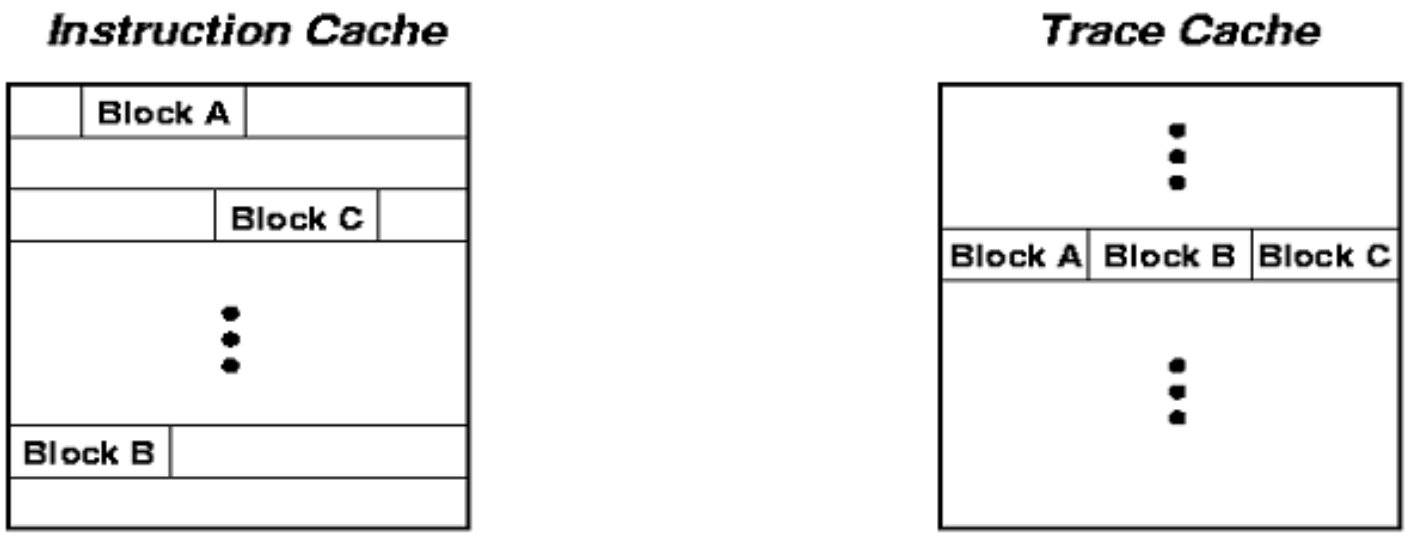
- **Trabalhos bastante recentes tem proposto o uso de Trace Caches para implementar máquinas VLIW com código compatível com máquinas não VLIW:**
  - **Estas arquiteturas, denominadas arquiteturas VLIW com *trace scheduling* dinâmico, implementam 2 máquinas: uma máquina RISC convencional baseada em um 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*.**

Trace Cache

- Utiliza uma memória cache adicional para armazenar os *"traces"* das instruções mais recentemente executadas, que é consultada pela unidade de busca de instruções.
- Quando muitos laços são executados, o "trace cache" tende a ter uma alta taxa de acerto.



Trace Cache


Instruction Cache
Block A
Block C
Block B
Trace Cache
Block A
Block B
Block C

Trace Cache

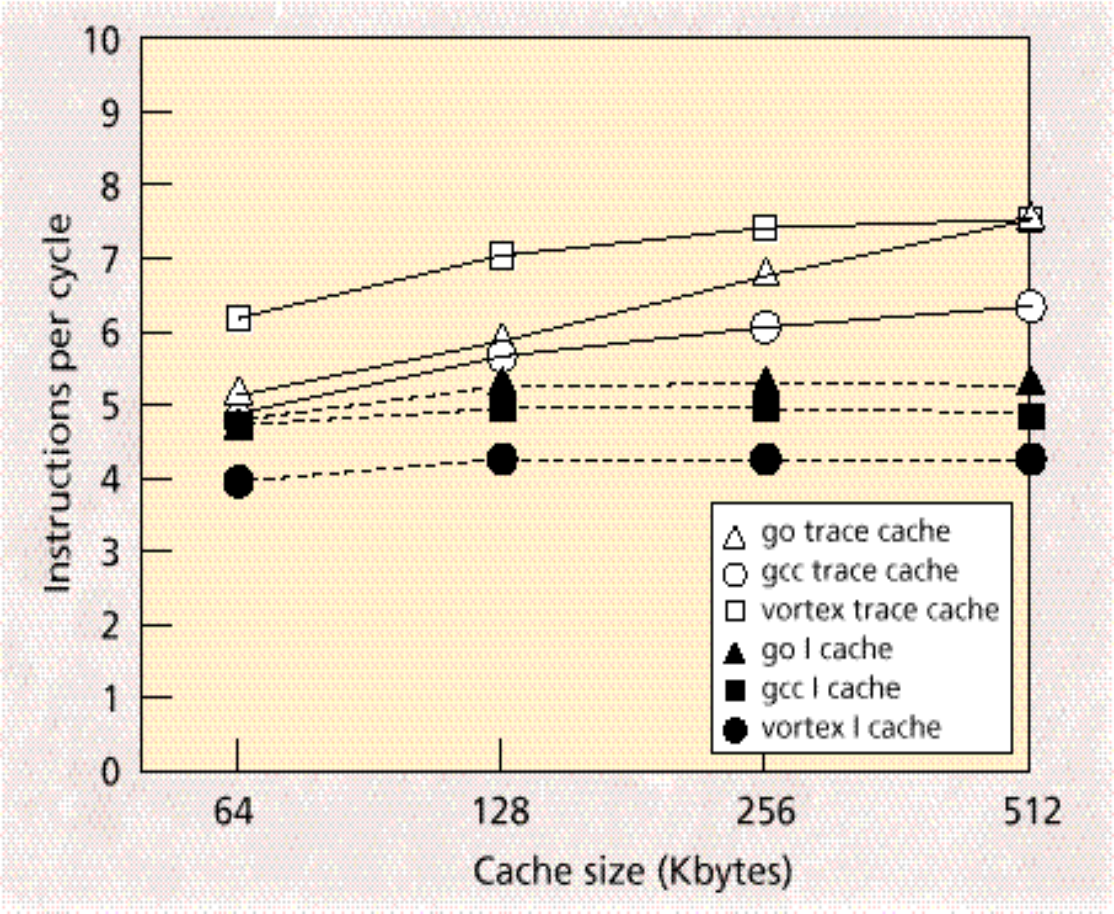
- Trace cache é um novo paradigma para as caches de instrução.
- A “Trace cache” é uma cache de instruções especial que captura seqüência dinâmicas de instruções, em contraste com a cache de instruções que contém uma seqüência estática de instruções.
- Como a cache de instruções, a “trace cache” é acessada usando o endereço inicial do próximo bloco de instruções.
- Diferentemente da cache de instruções, ela armazena instruções que são logicamente contíguas em posições que são físicamente contíguas.
- Uma linha da “trace cache” armazena um segmento do *trace* de instruções dinâmico através de múltiplos desvios executados.

Trace Cache

- Quando um grupo de instruções é processado, ele é capturado pela "unidade de preenchimento" ou "fill unit".
- A "**fill unit**" maximiza o tamanho do segmento e finaliza-o quando o segmento não pode ser mais expandido.
  - O número de instruções em um trace é limitado pelo tamanho da linha da trace cache.
  - Os segmentos finalizados são escritos na "trace cache".
- As instruções podem ser enviadas diretamente para as unidades funcionais sem necessidade de serem novamente decodificadas,.
- Isto é particularmente útil no caso do Pentium IV, que faz a conversão de instruções CISC para instruções RISC, durante o processo de execução das instruções.

Trace Cache – Desempenho

Três aplicações do SPECint95 foram simuladas em uma máquina com largura de busca de 16 instruções por ciclo e com predição perfeita de desvio.



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, etc.**

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.**

Tabela de Renomeação

- Para realizar a renomeação normalmente é utilizada uma tabela de renomeação, que é endereçada com os números dos registradores arquiteturais (vistos pelo programador) que estão sendo lidos pelas instruções.

- A tabela de renomeação faz a conversão fornecendo na saída os registradores físicos para onde foram mapeados os registradores arquiteturais.

- Uma condição necessária para isso é que o número de registradores físicos seja maior que o número de registradores arquiteturais.

- Cada vez que uma instrução que escreve em um registrador é despachada, esse registrador é mapeado para um novo registrador físico a partir de uma lista de livres.

Tabela de Renomeação

- A tabela de renomeação deve ser capaz de renomear em um ciclo tantos registradores quantos forem os operandos lidos pelas instruções que a arquitetura consegue despachar em um ciclo.

- Também tem que atualizar o mesmo número de registradores que são escritos.

- Deste modo, é uma estrutura que possui múltiplas portas de leitura e de escrita.

- Por exemplo: se a arquitetura pode despachar até 4 instruções por ciclo, então a tabela de renomeação deve ter 8 portas de leituras e 4 de escrita.

- A tabela de renomeação guarda o mapeamento mais recente para cada registrador arquitetural.

Tabela de Renomeação

- Deste modo, existirão algumas outras instruções, já despachadas, que estão mapeando o mesmo registrador arquitetural para outros registradores físicos.

- Um registrador físico já alocado pode ser colocado na lista de livres quando não houver nenhuma instrução já despachada precisando do seu resultado E o registrador arquitetural já tenha sido mapeado para outro registrador físico por uma nova instrução.

- Mais adiante, veremos quando essa condição é satisfeita.

- Se não houver registradores físicos disponíveis (lista de livres vazia) o despacho das instruções é suspenso.

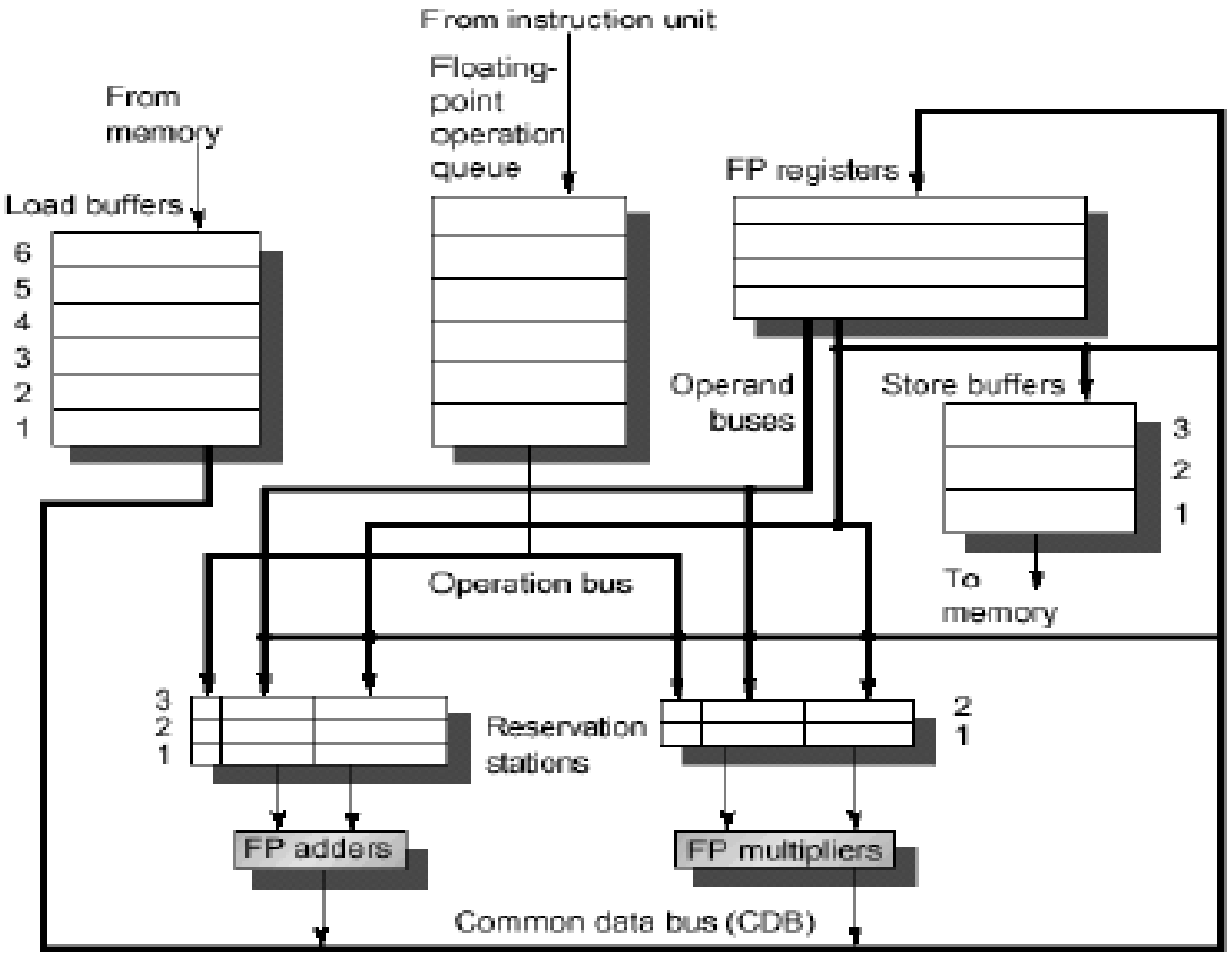
Algoritmo de Tomasulo

- 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

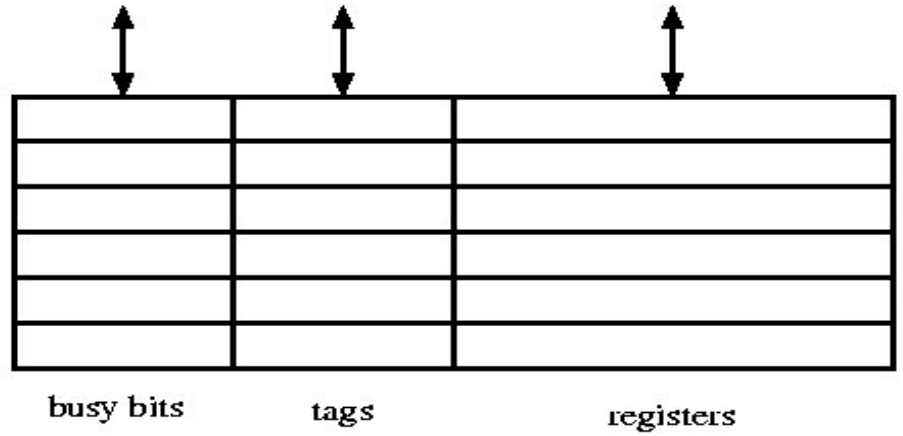


O Banco de Registradores

- 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.

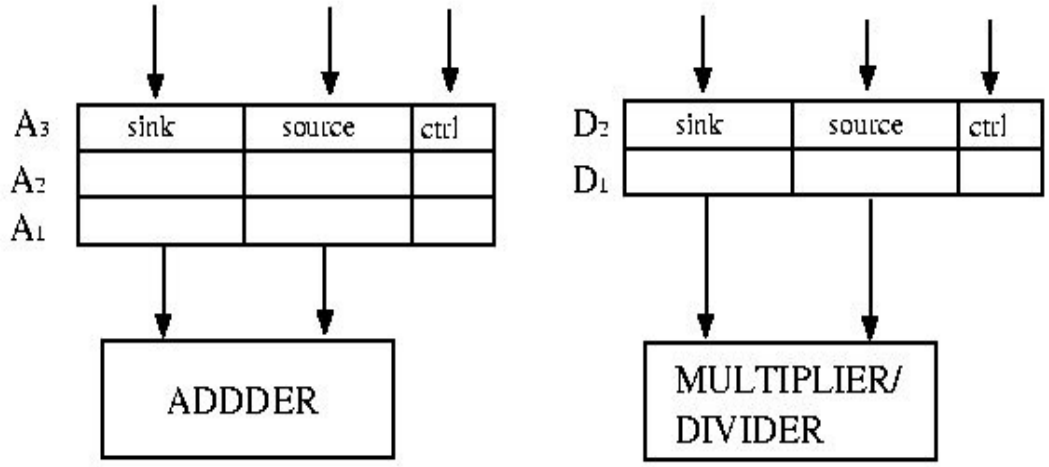
- Caso o valor do registrador esteja desatualizado, o segundo campo, denominado "tag", aponta para a estação de reserva contendo a instrução mais recente que vai produzir este valor.

- 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.



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.

A Difusão de Resultados no CDB

- 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

- Se os recursos (unidade funcional e operandos fonte) estiverem disponíveis, a instrução é iniciada imediatamente, antes mesmo do término de suas sucessoras.

- Caso contrário, é feira a cópia dos operandos disponíveis para as estações de reserva, o que resolve o problema das antidependências.

- As instruções subseqüentes podem ser enviadas para execução nas unidades funcionais, mesmo que uma instrução anterior não possa ser executada imediatamente.

Resolução das Dependências de Dados

- 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.

Buffer de Reordenação

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

Buffer de Reordenação

- 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.

- Se essa instrução não gerou exceções, o valor do seu resultado é escrito no Banco de Registradores e a instrução é eliminada da fila.

- O registrador físico alocado para o mesmo registrador arquitetural por uma instrução prévia pode ser então liberado para a lista de livres.

- O decodificador de instruções permanece funcionando enquanto houver entradas livres no Buffer de Reordenação.

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 resultado da operação 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).

Buffer de Reordenação

- Para recuperar o estado do processador quando ocorre uma predição errada de desvio ou exceção, basta aguardar que a instrução de desvio correspondente atinja a 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.

Buffer de Reordenação e Tabela de Renomeação

- Existe uma cópia da saída do "buffer" da tabela de renomeação, que guarda o mapeamento seguro dos registradores arquiteturais para os registradores físicos.

- Essa tabela é atualizado com o mapeamento do registrador que é escrito por cada instrução que chega ao final da fila, se houver.

- Ou seja, ela guarda o mapeamento não especulativo dos registradores .

- Em caso de exceção ou predição incorreta do desvio, esta cópia da tabela é restaurada para a verdadeira tabela de renomeação, restabelecendo o estado correto do processador.

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 do sistema 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.**

Unidade Funcional de Load/Store

- 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*.

Unidade Funcional de Load/Store

- 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*").