Arquitetura de Computadores

Paralelismo em Nível de Instrução
Escalonamento Dinâmico e
Despacho Fora de Ordem

Escalonamento Dinâmico

- O escalonamento dinâmico é uma técnica pela qual o hardware reordena a execução de instruções para reduzir as paradas (stalls), mantendo o fluxo de dados e o comportamento das exceções.
- Vantagens:
  - permite que o código compilado com um pipeline em mente seja executado com eficiência em um pipeline diferente;
  - permite lidar com alguns casos em que as dependências são desconhecidas em tempo de compilação;
  - permite que o processador tolere atrasos imprevisíveis, como falhas de cache, executando outro código enquanto aguarda a resolução da falha.
- As vantagens do agendamento dinâmico são obtidas ao custo de um aumento significativo na complexidade do hardware.

Escalonamento Dinâmico

- Considere este código:

```
div f0, f2, f4
add f0, f0, f8
sub f12, f8, f14
```

- A instrução sub não pode ser executada porque a dependência de add em relação a div causa a paralisação do pipeline;
- no entanto, sub não depende de dados em nada no pipeline.
- Esse risco cria uma limitação de desempenho que pode ser eliminada ao não exigir que as instruções sejam executadas na ordem do programa.

Escalonamento Dinâmico

- Execução Fora de Ordem
  - Queremos que uma instrução inicie sua execução assim que seus dados operandos estiverem disponíveis.
  - Esse pipeline realiza a execução fora de ordem, o que implica a conclusão fora de ordem.
  - Isso introduz a possibilidade de riscos WAR e WAW, que não existem no pipeline clássico de cinco estágios.

Execução Fora de Ordem

- Considere este código:

  div f0, f2, f4

  mul f6, f0, f8

  add f0, f10, f14

  - Há uma antidependência entre mul e add (para o registrador f0), e se o pipeline executar add antes de muld (que está aguardando div), ele violará a antidependência, gerando um risco de WAR.
  - Há dependências de saída, a gravação de f0 por add antes da conclusão de div, os riscos WAW devem ser tratados.
  - Ambos os riscos são evitados pelo uso da renomeação de registradores.

Execução Fora de Ordem

- A conclusão fora de ordem também cria grandes complicações no tratamento de exceções.
- O escalonamento dinâmico com conclusão fora de ordem deve preservar o comportamento das exceções, no sentido de que exatamente as exceções que surgiriam se o programa fosse executado em ordem estrita de fato ocorram.
- Processadores escalonados dinamicamente preservam o comportamento das exceções, atrasando a notificação de uma exceção associada até que o processador saiba que a instrução deve ser a próxima a ser concluída.

Execução Fora de Ordem

- Para permitir a execução fora de ordem, essencialmente dividimos o estágio de identificação em dois estágios:

  1. Despacho — Decodificar instruções, verificar riscos estruturais.

  2. Ler operandos — Aguardar até que não haja riscos nos dados e, em seguida, ler os operandos.

Execução Fora de Ordem

- Em um pipeline escalonado dinamicamente, todas as instruções passam pelo estágio de despacho em ordem;
- no entanto, elas podem ser paralisadas ou podem ser adiantadas no segundo estágio (leitura de operandos) e, assim, entrar em execução fora de ordem.

Especulação de Hardware

- É uma técnica para reduzir o efeito das dependências de controle que:
  - prevê o resultado de um desvio,
  - executa instruções no endereço de destino previsto e;
  - executa ações corretivas quando a previsão estiver errada.

Algoritmo de Tomasulo

- Inventado por Robert Tomasulo para a unidade de ponto flutuante IBM 360/91;
- Permite a execução fora de ordem:
  - Rastreia quando os operandos das instruções estão disponíveis para minimizar os riscos de RAW; e
  - Faz a renomeação de registradores no hardware para minimizar os riscos de WAW e WAR.

Algoritmo de Tomasulo

- Seja o código:

```
div f0, f2, f4
add f6, f0, f8
sw f6, 0(x1)
sub f8, f10, f14
mul f6, f10, f8
```

Existem duas antidependências:
- entre add e sub (f8); e
- entre sw e mul (f6)

Existe uma dependência de saída :
- entre add e mul (f6)

Existem três dependências de dados reais:
- entre div e add(f0);
- entre add e sw(f6); e
- entre sub e mul(f8)

As dependências de nome são resolvidas renomeando (ou substituindo) :
- o registrador f6 de add e sw; e
- o registrador f8 de sub e mul.

Algoritmo de Tomasulo

- Seja o código:

```
div f0, f2, f4
add S, f0, f8
sw S, 0(x1)
sub T, f10, f14
mul f6, f10, T
```

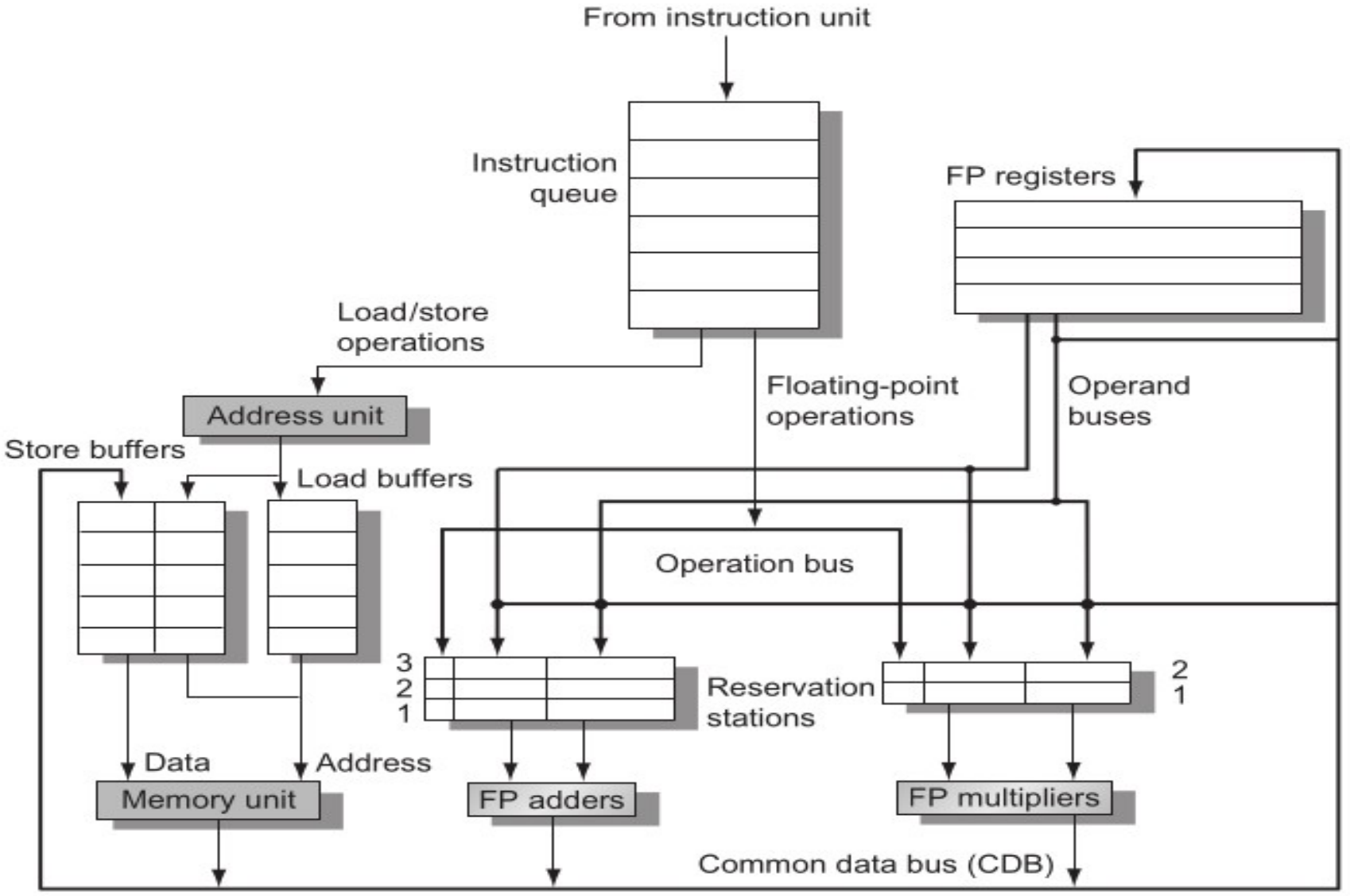
Em um compilador é fácil imaginar tal ação mas em hardware como fazê-lo ????

Algoritmo de Tomasulo

- No algoritmo de Tomasulo, a renomeação de registradores é realizada por **estações de reserva**,
  - que armazenam os operandos das instruções até serem emitidas;
  - estão associadas às unidades funcionais.
- Uma estação de reserva busca e armazena um operando assim que ele estiver disponível, eliminando a necessidade de obter o operando de um registrador.
- Assim, as instruções pendentes designam a estação de reserva que fornecerá sua entrada.
- E, quando gravações sucessivas em um registrador se sobrepõem a execução apenas a última é utilizada para atualizar o registrador

Algoritmo de Tomasulo

- A figura mostra a estrutura básica de um processador baseado em Tomasulo:
  - unidade de ponto flutuante;
    - Somadores de ponto-flutuante;
    - Multiplicadores de ponto-flutuant
  - unidade de carga/armazenamento
    - Unidade de endereço;
    - Buffers de carga e armazenamento
    - Unidade de Memória
  - Estações de Reservas
  - Memória de Instruções


From instruction unit
Instruction queue
FP registers
Load/store operations
Floating-point operations
Operand buses
Address unit
Store buffers
Load buffers
Operation bus
3
2
1
Reservation stations
2
1
Data
Address
Memory unit
FP adders
FP multipliers
Common data bus (CDB)

Algoritmo de Tomasulo

- Etapas pelas quais as instruções devem fluir
  - Despacho : Obtém a próxima instrução do início da fila de instruções, e:
    - Se houver uma estação de reserva vazia, coloca a instrução na estação com os valores dos operandos ou com a indicação da unidade funcional que o produzirão.
    - Se não houver uma estação de reserva vazia, há um risco estrutural e a emissão da instrução é interrompida até que uma estação ou buffer seja liberado.

    Esta etapa renomeia os registradores, eliminando os riscos WAR e WAW.

Algoritmo de Tomasulo

- Execução:
  - Se um ou mais operandos ainda não estiverem disponíveis, monitora o barramento de dados, quando o operando se torna disponível, é colocado nas estações de reserva que o aguarda.
  - Assim que todos os operandos estiverem disponíveis, a operação pode ser executada na unidade funcional correspondente.

  Ao atrasar a execução da instrução até que os operandos estejam disponíveis, evitam-se riscos de RAW.

Algoritmo de Tomasulo

- Carregamentos e armazenamentos requerem um processo de execução em duas etapas.
  - A primeira etapa calcula o endereço efetivo e o endereço efetivo é então colocado no buffer de carregamento ou armazenamento.
  - Carregamentos no buffer são executados assim que a unidade de memória está disponível.
  - Armazenamentos no buffer de armazenamento aguardam que o valor seja colocado no buffer antes de serem enviados para a unidade de memória.

  Carregamentos e armazenamentos são mantidos na ordem do programa por meio do cálculo do endereço efetivo, o que ajudará a prevenir riscos por meio da memória.

Algoritmo de Tomasulo

- Escrita de resultado:
  - Quando o resultado estiver disponível, escreva-o no CDB (barramento comum de dados) e,
  - nos registradores e
  - em quaisquer estações de reserva (incluindo buffers de armazenamento) que aguardam esse resultado.
  - Os armazenamentos são colocados no buffer de armazenamento até que o valor a ser armazenado e o endereço do armazenamento estejam disponíveis; então, o resultado é gravado assim que a unidade de memória estiver livre.

Algoritmo de Tomasulo

- As estações de reservas têm sete campos:
  - **Op**: operação a ser executada sobre os operandos S1 e S2.
  - **Qj**, **Qk**: estações de reserva que produzirão os operandos; zero indica que o operando de origem já está disponível em Vj ou Vk, ou é desnecessário.
  - **Vj**, **Vk**: valor dos operandos. Observe que apenas um dos campos V ou Q é válido para cada operando. Para cargas, o campo Vk é usado para armazenar o campo de deslocamento.
  - **A**: armazenar informações para o cálculo do endereço de memória. Inicialmente, o campo imediato , depois o endereço efetivo é armazenado aqui.
  - **Busy**: Indica que esta estação de reserva e sua unidade funcional correspondente estão ocupadas.

Algoritmo de Tomasulo

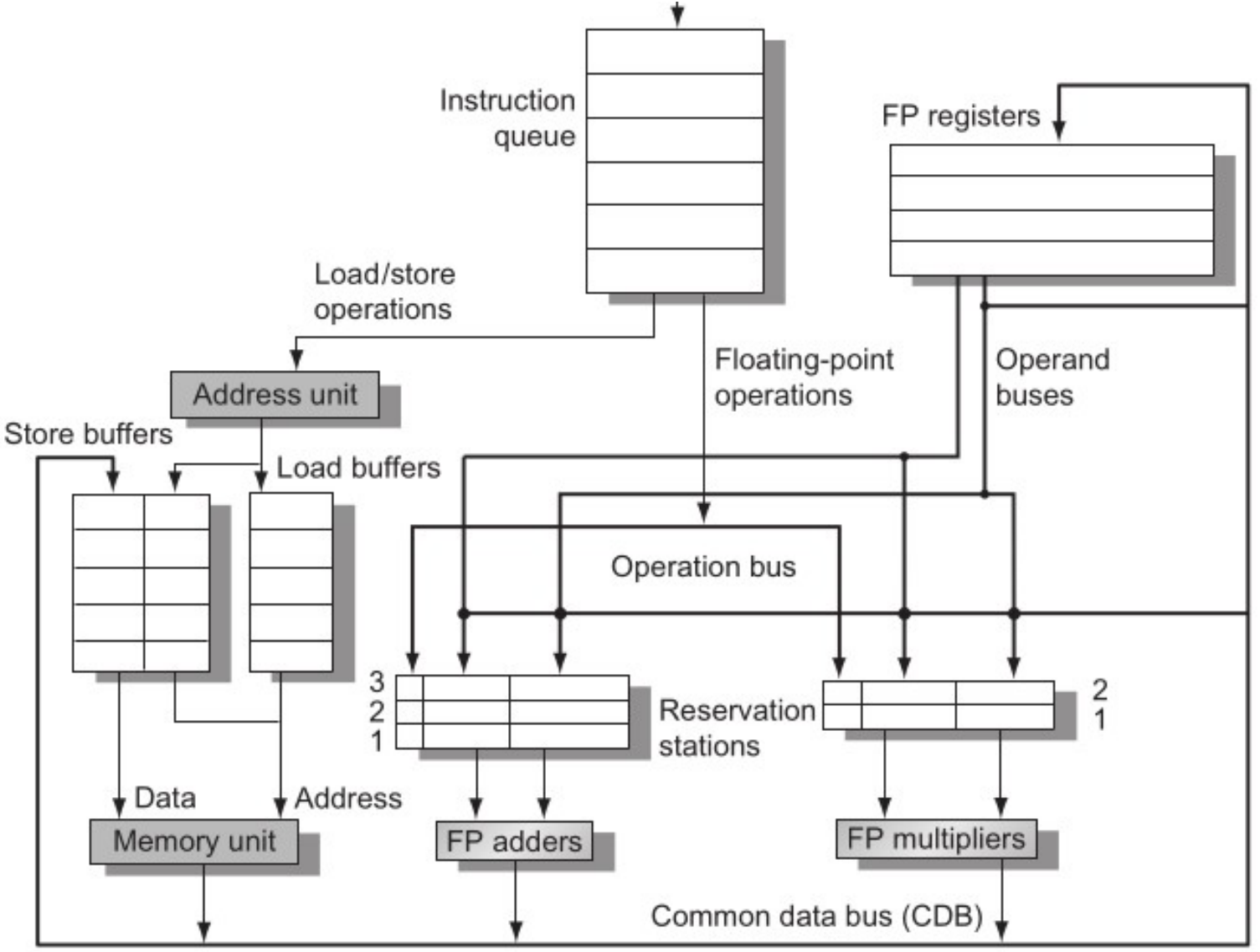
- O arquivo de registradores possui um campo:
  - Qi: número da estação de reserva que contém a operação cujo resultado deve ser armazenado neste registrador. Se o valor de Qi estiver em branco (ou 0), nenhuma instrução ativa no momento está computando um resultado destinado a este registrador, o que significa que o valor é simplesmente o conteúdo do registrador.

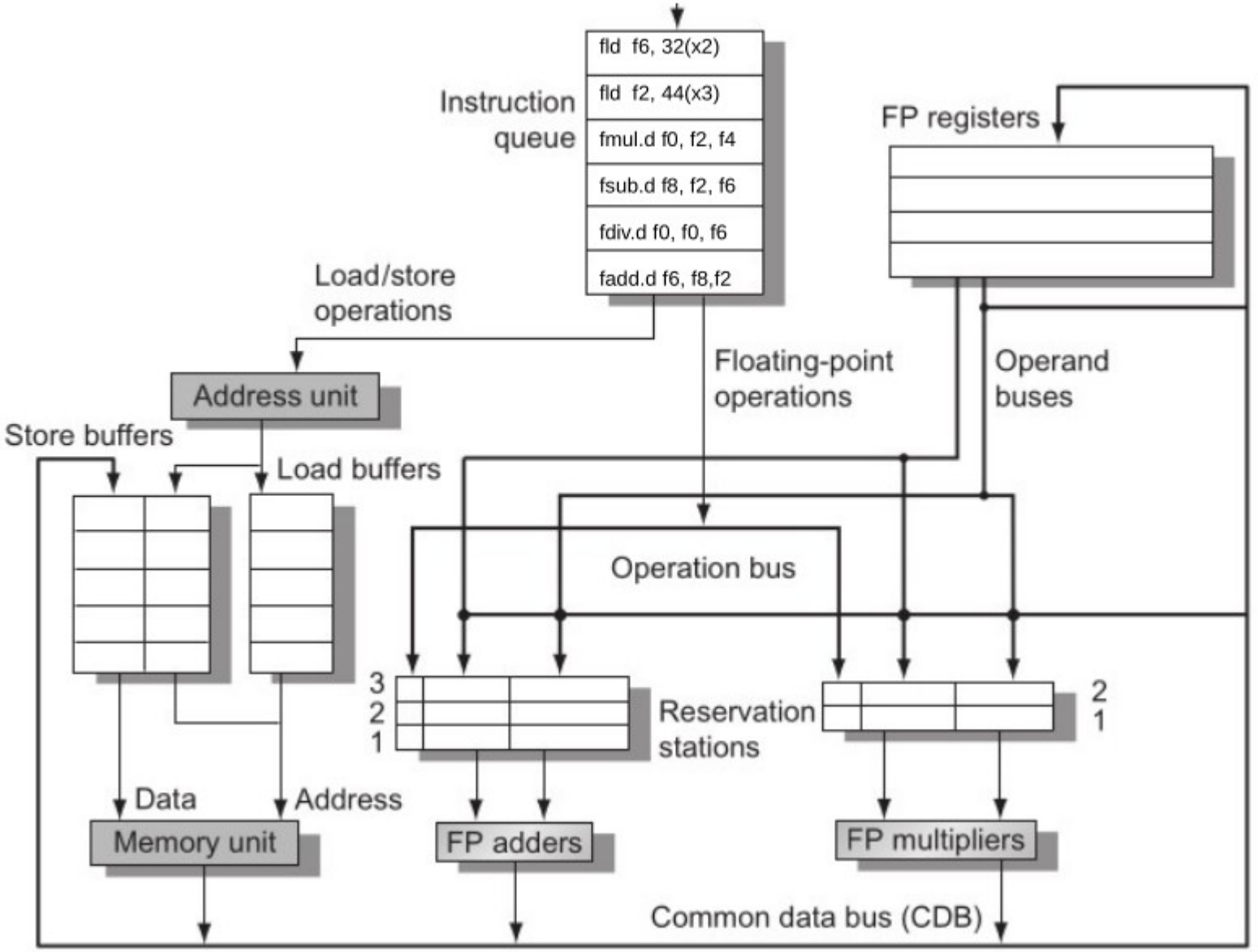
Algoritmo de Tomasulo

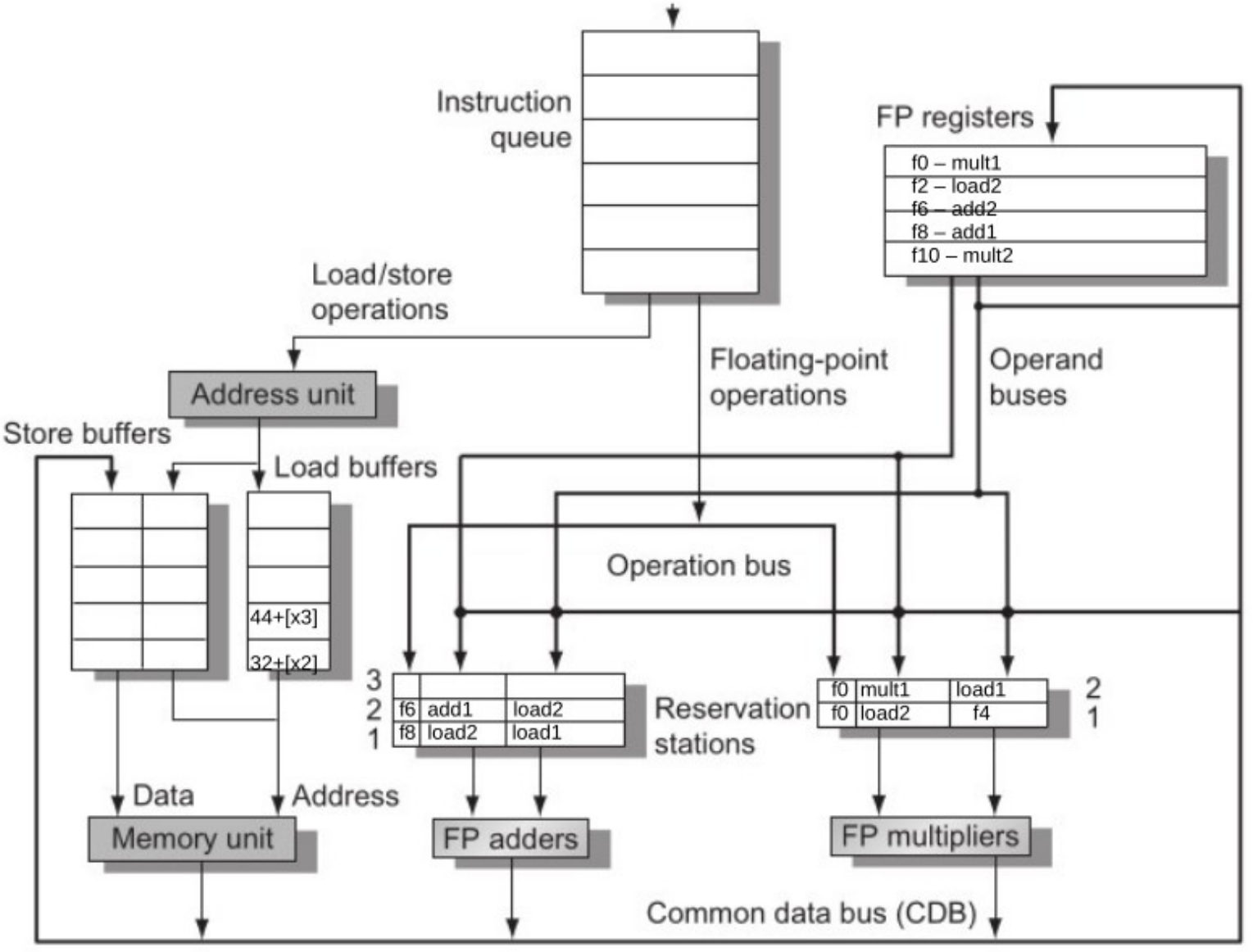
- Exemplo: Seja a sequência de código

```
fld  f6,32(x2)
fld  f2, 44(x3)
fmul.d f0, f2, f4
fsub.d f8, f2, f6
fdiv.d  f0, f0, f6
fadd.d f6, f8, f2
```

Ao serem colocadas no pipeline com Tomasulo ficaria:


Instruction queue
FP registers
Load/store operations
Floating-point operations
Operand buses
Address unit
Store buffers
Load buffers
Operation bus
3
2
1
Reservation stations
2
1
Data
Address
Memory unit
FP adders
FP multipliers
Common data bus (CDB)


fld f6, 32(x2)
fld f2, 44(x3)
fmul.d f0, f2, f4
fsub.d f8, f2, f6
fdiv.d f0, f0, f6
fadd.d f6, f8,f2
Instruction queue
FP registers
Load/store operations
Floating-point operations
Operand buses
Address unit
Store buffers
Load buffers
Operation bus
3
2
1
Reservation stations
2
1
Data
Address
Memory unit
FP adders
FP multipliers
Common data bus (CDB)


Instruction queue
FP registers
f0 – mult1
f2 – load2
f6 – add2
f8 – add1
f10 – mult2
Load/store operations
Floating-point operations
Operand buses
Address unit
Store buffers
Load buffers
Operation bus
44+[x3]
32+[x2]
3
2
1
f6 add1 load2
f8 load2 load1
Reservation stations
f0 mult1 load1
f0 load2 f4
2
1
Data
Address
Memory unit
FP adders
FP multipliers
Common data bus (CDB)

	Reservation stations						
Name	**Busy**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**	**A**
Load1	No						
Load2	Yes	Load					`44 + Regs[x3]`
Add1	Yes	SUB		`Mem[32 + Regs[x2]]`	Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL		`Regs[f4]`	Load2		
Mult2	Yes	DIV		`Mem[32 + Regs[x2]]`	Mult1		

	Register status								
Field	**f0**	**f2**	**f4**	**f6**	**f8**	**f10**	**f12**	**…**	**f30**
Qi	Mult1	Load2		Add2	Add1	Mult2			

Algoritmo de Tomasulo

	Instruction status		
Instruction	Issue	Execute	Write result
`fld f6,32(x2)`	√	√	√
`fld f2,44(x3)`	√	√	
`fmul.d f0,f2,f4`	√		
`fsub.d f8,f2,f6`	√		
`fdiv.d f0,f0,f6`	√		
`fadd.d f6,f8,f2`	√		