Esta ficha foi elaborada para acompanhar o estudo da disciplina de Arquitetura de Computadores Avançada (a4s1) ou para complementar a preparação para os momentos de avaliação finais da mesma. Num segundo ficheiro poderás encontrar um conjunto de propostas de solução aos exercícios que estão nesta ficha. É conveniente relembrar que algum conteúdo destes documentos pode conter erros, aos quais se pede que sejam notificados pelas vias indicadas na página web, e que serão prontamente corrigidos, com indicações de novas versões.

1. Explica no que consiste o esquema de delayed branch (branch em atraso).

Este esquema foi bastante usado nas arquiteturas do tipo RISC e aqui, o ciclo de execução é definido pela instrução de branch propriamente dita, seguida do seu sucessor sequencial e do alvo (caso seja tomado o branch como taken). Ao sucessor sequencial damos o nome de delay slot e esta instrução é sempre executada (quer o branch seja taken ou not taken), não podendo ser uma instrução de branch.

2. O que é uma exceção? Dá exemplos de 2 exceções que ocorrem no curso (within) e entre (between) instruções. Identifica 2 exceções para cada tipo.

As exceções são acontecimento que causam a paragem da leitura sequencial de um programa de uma forma não esperada, inserindo no seu decurso uma sequência de código de uma rotina de tratamento da exceção, podendo mais tarde voltar ao estado pré-exceção ou simplesmente dar o programa como terminado.
As exceções podem ser agrupadas em vários tipos, entre os quais em exeções within e exceções between. As exceções within são tais que ocorrem no decurso de uma instrução. Por exemplo, se uma instrução quiser ler um endereço de memória poderá haver um page fault, exceção essa que é provocada no decurso de uma instrução. Outro exemplo será o caso de uma divisão por zero, que despoleta uma exceção aritmética.
Por outro lado, temos as exceções between (entre instruções). Este tipo de acontecimentos é provocado por perturbações fora do contexto da sequência de instrução, como uma remoção do módulo de I/O onde estavam a ser lidos ou efetuados trabalhos, ou simplesmente interrupções feitas ao kernel de um sistema operativo, provocadas por threads que necessitam de interagir com um nível fora do contexto do espaço de utilizador.

3. "As exceções são atendidas por uma rotina de tratamento que resolve os problemas que condicionaram a execução normal do programa e restaura o mesmo para o momento antes do seu lançamento". Concordas com a afirmação, para qualquer caso de exceção?

Não, porque nem todas as exceções permitem que seja restaurado o estado do programa anterior ao seu lançamento. Na verdade, nem todos os pipelines têm a capacidade de lidar com as exceções, guardando os estados e restaurando-os sem que haja afetação de qualquer outro programa já em execução. Quando estamos perante um pipeline que o consiga fazer, então dizemos que este é restaurável (restartable).
Em pipelines restauráveis, quando é lançada uma exceção é determinada a causa do seu lançamento, sendo esse o ponto em que o restauro dependerá.

4. As exceções podem ter um variado leque de causas para o seu lançamento, contudo, elas podem ser agrupadas em vários conjuntos diferentes que definem o seu tipo/a sua categoria. Dos cinco tipos seguintes, explica sucintamente no que se baseiam e dá um dois casos de aplicação (para uma categorida definida pelo par A, B dá um exemplo para A e um exemplo para B):

a) Exceções síncronas e assíncronas;

As exceções síncronas são tais que ocorrem no mesmo local sempre que o programa é executado na mesma região de memória e com os mesmos dados. Exemplo desta exceção é uma instrução ilegal (não implementada ou que não existe).
Por outro lado, as exceções assíncronas são tais que ocorrem em qualquer instante no programa e só podem ser tratadas depois do completamento das instruções que as causaram. Exemplo desta exceção é um pedido de interrupção por um módulo de I/O.

b) Exceções maskable e non-maskable;

As exceções maskable são tais que permitem ao hardware escolher quando é que podem ser atendidas (tratadas). Exemplo é uma invocação de uma interrupção por parte de uma thread de execução ao kernel.
As exceções non-maskable são tais não permitem a escolha ao hardware para o atendimento das mesmas. Sendo o caso mais comum, é exemplo um acesso à memória não alinhado.

c) Exceções within e between (se respondeste à questão 2 apenas precisas de explicar o conceito);

As exceções within são tais que ocorrem no decurso de uma instrução (desde que esta é lançada até que seja concluída). Exemplo é uma falha de página (page fault) no acesso à memória virtual.
As exceções between são tais que ocorrem entre instruções. Exemplo é a definição de um breakpoint num programa.

d) Exceções resume e terminate;

As exceções que permitem que o programa seja restaurado no ponto anterior ao seu lançamento são denominadas de resume. Exemplo é uma exceção causada por um overflow de um inteiro.
As exceções que não permitem o restauro são denominadas de terminate, e um exemplo destas é uma falha de energia.

e) Exceções pedidas pelo utilizador e coagidas.

As exceções pedidas pelo utilizador são tais que são previsíveis e de algum modo podem não ser referenciadas como exceções. Dá-se o exemplo das interrupções ao kernel, pelo sistema operativo.
As exceções coagidas são tais que podem ser lançadas por um evento de um hardware imprevisto pelo programa. Caso exemplo é uma avaria de um hardware montado na máquina.

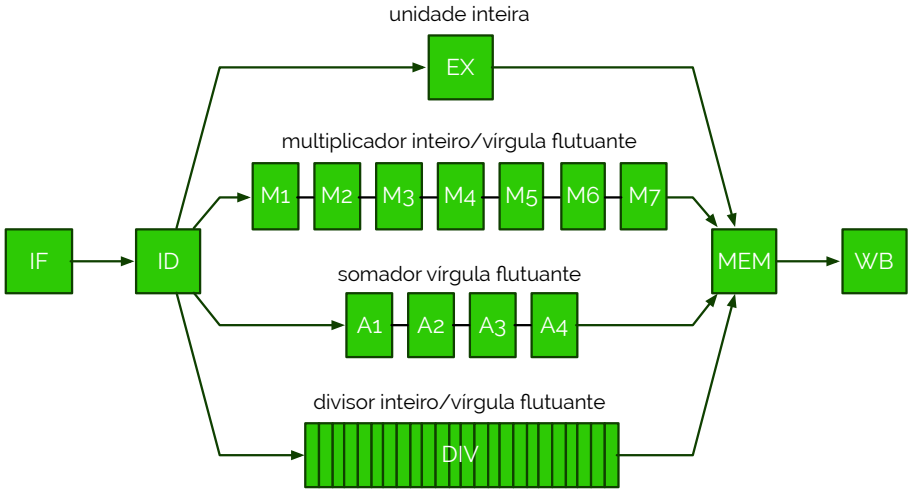
f) Uma exceção pode fazer parte de duas ou mais categorias, ou é exclusiva a uma só, das referidas? Se puder fazer parte de duas ou mais categorias dá exemplo de duas exceções e categoriza-as com o máximo de tipos possível.

Uma exceção faz sempre parte de um de dois estados de todas as categorias. Damos o exemplo das exceções de acesso à memória não alinhado e de breakpoints. Um acesso à memória é síncrono, coagido, non-maskable, within e terminate. A definição de um breakpoint é síncrono, pedido pelo utilizador, maskable, between e resume.

5. O que é que significam "exeções precisas"? Porque é que muitos dos processadores atuais introduziram dois modos de operação: um com exceções precisas e outro sem?

Uma exceção precisa é tal que o pipeline pode ser parado de forma a que as instruções precedentes à falha são completadas e este mesmo, juntamente com as instruções que lhe sucedem, pode reiniciar do zero. Idealmente, a instrução em falha não deve alterar o estado da execução e, assim, de forma a manipular corretamente algumas exceções, é requirido que a instrução em falha não produza qualquer efeito. Para ultrapassar este tipo de problemas os processadores mais recentes possuem um modo com exceções precisas e outro sem, isto porque tornar-se-ia muito difícil e muito lento restaurar estados e ordens de lançamento de exeções se todas estas fossem precisas, dado que muitas delas inclusive são terminate.

6. Consideremos a figura abaixo onde mostramos um pipeline clássico de cinco andares com uma pequena variante de operações multi-cycle na fase de execução.



a) Explica de que forma é que a execução das instruções neste pipeline podem ser descritas como em--ordem com, eventualmente, out-of-order completion (conclusão fora-de-ordem).

O pipeline que vemos na figura é muito semelhante ao pipeline clássico de cinco andares que estudámos, onde as instruções eram executadas e completadas em-ordem. A pequena (grande) diferença é que agora temos uma fase de execução (EX) que é variável em número de ciclos de execução. Por outras palavras, como estamos a discriminar níveis de execução diferentes para unidades funcionais diferentes, e como cada unidade funcional possui uma latência de execução diferente, uma instrução de divisão embora chegue primeiro a esta fase que uma instrução de subtração (por exemplo), terminará a execução muito depois (24 ciclos depois, neste caso) da instrução de subtração inteira.

b) Que tipos de hazards estruturais podem ocorrer neste pipeline?

Existem dois casos de hazards estruturais que podem ocorrer neste pipeline. Primeiro, poderão ocorrer hazards no acesso a memória de dados. Segundo, como a unidade de divisão inteira/vírgula flutuante não tem uma implementação pipeline, também nesta poderão ocorrer hazards estruturais.

c) Que tipos de hazards de dados podem ocorrer neste pipeline?

Em termos de hazards de dados podemos ter hazards do tipo WAW E RAW. Os hazards do tipo WAW ocorrem, por exemplo, quando uma instrução de multiplicação guarda o seu resultado ao mesmo tempo que uma instrução de adição, lançada 4 ciclos depois. Os hazards do tipo RAW ocorrem, por exemplo, quando uma instrução tenta ler um registo antes que este seja escrito por uma instrução.

d) Considera o seguinte código a ser executado neste pipeline:

```
div.d   f0, f2, f4
add.d   f10, f10, f8
sub.d   f12, f12, f14
```

É possível a operação de subtração causar uma exceção aritmética de vírgula flutuante em tal ponto que a instrução de soma já terminou a execução, mas a operação de divisão ainda não? Em qualquer dos casos, numa situação semelhante, supõe que a operação de divisão lançava uma exceção antes da operação de soma terminar. Classifica essa exceção como precisa ou imprecisa, justificando.

Sim, é possível. No segundo caso enunciado a exceção deverá ser imprecisa, dado que, de facto, como a operação de soma destroi um dos seus operandos, não conseguiríamos restaurar o estado antes da operação de divisão, mesmo com a ajuda de um software.

7. "Qualquer dependência entre instruções inibe estas de serem executadas em paralelo". Concordas com a afirmação?

Não. Argumentando a resposta a partir de um contraexemplo, consideremos as dependências de nome (quer antidependências, como dependências de saída) onde, não havendo qualquer fluxo de informação a tomar lugar entre instruções, são meramente virtuais, dado que são passíveis de serem resolvidas através de mecanismos de renomeação de registos (não são dependências verdadeiras).

8. Distingue antidependências de dependências de saída.

Ambas antidependências e dependências de saída são dependências de nome, no entanto, os seus conceitos distinguem-nas. Diz-se que as instruções i e j são antidependentes quando a instrução i escreve num registo ou posição de memória que é lida pela instrução j (a ordem original das instruções deve ser preservada para garantir que o valor correto é lido). Por outro lado, diz-se que as instruções i e j possuem uma dependência de saída quando ambas as instruções escrevem um valor para um mesmo registo ou posição de memória.

9. Dá um caso de aplicação (código ou figura) onde o hazard de dados do tipo RAR está presente.

Não existe nenhum hazard de dados do tipo RAR. Seguindo a mesma convenção que WAR, WAW e RAW, RAR (Read After Read) não provoca qualquer situação de perigo porque a operação de leitura é idempotente.

10. Consideremos as instruções A e B, com A a preceder B no programa. "O hazard de dados WAR ocorre quando A tenta escrever um valor num operando antes que B o leia, pelo que a instrução B irá obter um valor errado". Concordas com a afirmação?

Não. A afirmação estaria correta se as instruções consideradas fossem A e B, mas com B a preceder A no programa, dado que este hazard ocorre quando uma segunda instrução tenta escrever um valor num operando da primeira, antes que esta lhe aceda.

11. Porque é que o hazard do tipo WAR não poderá ocorrer em grande parte dos pipelines?

Este hazard de dados não poderá ocorrer em grande parte dos pipelines porque, usualmente, todas as leituras são feitas muito cedo e todas as escritas são feitas muito tarde por cada instrução.

12. A execução fora-de-ordem é uma funcionalidade do agendamento dinâmico. De forma a implementá-lo, a fase de instruction decode do pipeline clássico de cinco andares deve ser decomposto em duas sub-fases. Que fases são essas e qual é o papel delas?

Com o agendamento dinâmico a fase de instruction decode (ID) do pipeline clássico de cinco andares é decomposto nas fases de lançamento (issue) de instruções e leitura de operandos. Numa fase de lançamento uma instrução, de cada vez, é levada até às entradas das unidades funcionais, identificando o tipo de operação a cumprir. No entanto, porque ainda não possui qualquer informação sobre o valor dos operandos, há que efetuar uma leitura

dos mesmos antes da fase de execução. Assim, é feita a leitura dos operandos, onde se obtém o valor para cada operando, preparando a instrução para poder ser levada para execução com sucesso.

13. Duas técnicas básicas para implementação do agendamento dinâmico são o scoreboarding e o algoritmo de Tomasulo. Qual é a diferença substancial entre ambas?

Ambas as técnicas implementam o agendamento dinâmico. O algoritmo de Tomasulo, no entanto, trata as dependências de nomes, renomeando, dinamicamente, os registos, contrariamente ao scoreboarding. O algoritmo de scoreboarding é implementado em quatro fases: lançamento, leitura de operandos, execução e escrita de resultados. Por outro lado, o algoritmo de Tomasulo implementa apenas três das fases anteriores: lançamento, execução e escrita de resultados. O que acontece é que a primeira fase do algoritmo de Tomasulo funde as primeiras duas fases do scoreboarding, fazendo "desaparecer" a fase de leitura de operandos, para onde o scoreboarding apenas transitava se os operandos estivessem prontos a serem lidos. Com a renomeação de registos é agora possível que ao invés de apontar para o registo onde o operando reside, se possa apontar para a unidade funcional que está ainda a produzir o operando, caso isso aconteça (condição de espera para transição para a fase de leitura de operandos no scoreboarding).

14. Considera a seguinte sequência de instruções, a ser executado num pipeline onde o agendamento dinâmico é feito por scoreboarding, tendo duas unidades funcionais de soma e duas unidades funcionais de multiplicação (ambas em vírgula flutuante):

```
add.d   f0, f2, f4
mult.d  f2, f6, f8
mult.d  f10, f0, f2
add.d   f0, f12, f14
```

estado de instrução				
instrução	**lançamento**	**leitura de operandos**	**execução**	**escrita de resultado**
`add.d f0,f2,f4`				
`mul.d f2,f6,f8`				
`mul.d f10,f0,f2`				
`add.d f0,f12,f14`				

estado de unidades funcionais									
nome	**busy**	**op**	**F_i**	**F_j**	**F_k**	**Q_j**	**Q_k**	**R_j**	**R_k**
mult1	não								
mult2	não								
soma1	não								
soma2	não								

estado de registos								
F0	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**...**	**F30**

a) Completa a tabela acima considerando que apenas foi lançada a instrução add.d f0, f2, f4.

estado de instrução				
instrução	**lançamento**	**leitura de operandos**	**execução**	**escrita de resultado**
`add.d f0,f2,f4`	sim			
`mul.d f2,f6,f8`				
`mul.d f10,f0,f2`				
`add.d f0,f12,f14`				

estado de unidades funcionais									
nome	busy	op	F_i	F_j	F_k	Q_j	Q_k	R_j	R_k
mult1	não								
mult2	não								
soma1	sim	soma	f0	f2	f4			sim	sim
soma2	não								

estado de registos								
F0	F2	F4	F6	F8	F10	F12	...	F30
soma1								

b) Completa a tabela acima considerando que a instrução add.d f0, f2, f4 terminou a escrita de resultados.

estado de instrução				
instrução	lançamento	leitura de operandos	execução	escrita de resultado
`add.d f0,f2,f4`	sim	sim	sim	sim
`mul.d f2,f6,f8`	sim	sim	sim	
`mul.d f10,f0,f2`	sim			
`add.d f0,f12,f14`				

estado de unidades funcionais									
nome	busy	op	F_i	F_j	F_k	Q_j	Q_k	R_j	R_k
mult1	sim	mult	f2	f6	f8			não	não
mult2	sim	mult	f10	f0	f2	soma1	mult1	sim	não
soma1	não								
soma2	não								

estado de registos								
F0	F2	F4	F6	F8	F10	F12	...	F30
	mult1				mult2			

c) Completa a tabela no ciclo seguinte ao da alínea b).

estado de instrução				
instrução	lançamento	leitura de operandos	execução	escrita de resultado
`add.d f0,f2,f4`	sim	sim	sim	sim
`mul.d f2,f6,f8`	sim	sim	sim	
`mul.d f10,f0,f2`	sim			
`add.d f0,f12,f14`	sim			

estado de unidades funcionais									
nome	busy	op	F_i	F_j	F_k	Q_j	Q_k	R_j	R_k
mult1	sim	mult	f2	f6	f8			não	não
mult2	sim	mult	f10	f0	f2	soma1	mult1	sim	não
soma1	sim	soma	f0	f12	f14			sim	sim
soma2	não								

estado de registos								
F0	F2	F4	F6	F8	F10	F12	...	F30
soma1	mult1				mult2			

d) Considerando o primeiro ciclo de execução em t = 0, em que ciclo de execução é que a alínea c) está?

A alínea c) está no ciclo t = 5.

15. Considera novamente o código do exercício 14 (repetido em baixo) a ser executado num pipeline onde o agendamento dinâmico é feito, agora, pelo algoritmo de Tomasulo, tendo duas unidades funcionais de soma e duas unidades funcionais de multiplicação (ambas em vírgula flutuante):

```
add.d   f0, f2, f4
mult.d  f2, f6, f8
mult.d  f10, f0, f2
add.d   f0, f12, f14
```

estado de instrução (não faz parte do hardware)			
instrução	lançamento	execução	escrita de resultado
`add.d f0,f2,f4`			
`mul.d f2,f6,f8`			
`mul.d f10,f0,f2`			
`add.d f0,f12,f14`			

estações de reserva / buffers							
nome	busy	op	V_j	V_k	Q_j	Q_k	A
mult1	não						
mult2	não						
soma1	não						
soma2	não						

estado de registos								
Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
Q_i									

a) Completa a tabela acima considerando que apenas foi lançada a instrução add.d f0, f2, f4.

estado de instrução (não faz parte do hardware)			
instrução	lançamento	execução	escrita de resultado
`add.d f0,f2,f4`	sim		
`mul.d f2,f6,f8`			
`mul.d f10,f0,f2`			
`add.d f0,f12,f14`			

estações de reserva / buffers							
nome	busy	op	V_j	V_k	Q_j	Q_k	A
mult1	não						
mult2	não						
soma1	sim	soma	reg[f2]	reg[f4]			
soma2	não						

estado de registos									
Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
Q_i	soma1								

b) Completa a tabela acima considerando que a instrução add.d f0, f2, f4 terminou a escrita de resultados no bus de dados comum (CDB).

estado de instrução (não faz parte do hardware)			
instrução	lançamento	execução	escrita de resultado
`add.d f0,f2,f4`	sim	sim	sim
`mul.d f2,f6,f8`	sim	sim	
`mul.d f10,f0,f2`	sim		
`add.d f0,f12,f14`	sim		

estações de reserva / buffers							
nome	busy	op	V_j	V_k	Q_j	Q_k	A
mult1	sim	mult	reg[f6]	reg[f8]			
mult2	sim	mutl	reg[f0]			mult1	
soma1	não						
soma2	sim	soma	reg[f12]	reg[f14]			

estado de registos									
Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
Q_i	soma1	mult1				mult2			

c) Completa a tabela no ciclo seguinte ao da alínea b), considerando que a multiplicação tem uma latência maior que 1.

estado de instrução (não faz parte do hardware)			
instrução	lançamento	execução	escrita de resultado
`add.d f0,f2,f4`	sim	sim	sim
`mul.d f2,f6,f8`	sim	sim	
`mul.d f10,f0,f2`	sim		
`add.d f0,f12,f14`	sim	sim	

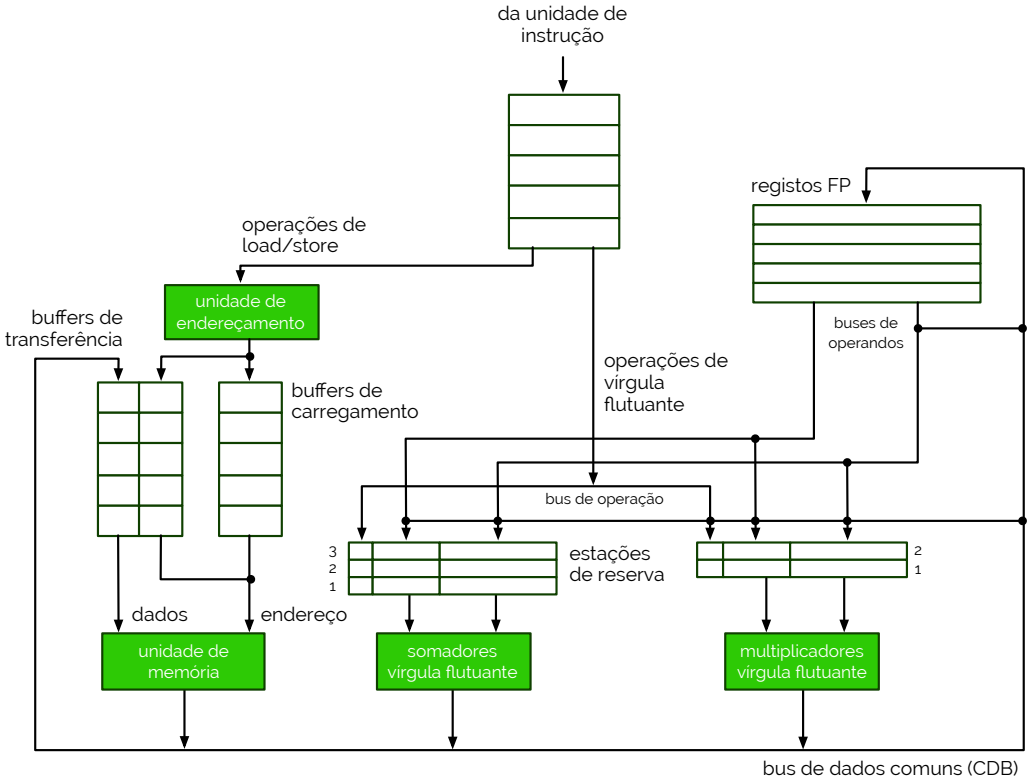
estações de reserva / buffers							
nome	busy	op	V_j	V_k	Q_j	Q_k	A
mult1	sim	mult	reg[f6]	reg[f8]			
mult2	sim	mutl	reg[f0]			mult1	
soma1	não						
soma2	sim	soma	reg[f12]	reg[f14]			

estado de registos									
Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
Q_i	soma1	mult1				mult2			

d) Considerando o primeiro ciclo de execução em t = 0, em que ciclo de execução é que a alínea c) está?

A alínea c) está no ciclo t = 4.

16. Considera a figura abaixo, onde se exibe o conceito de algoritmo de Tomasulo.



a) Quantos registos-resultado estão presentes na figura acima?

Na figura estão presentes 10 registos-resultado (5 buffers de load/store, 3 estações de reserva para a unidade funcional de soma em vírgula flutuante e 2 estações de reserva para a unidade multiplicadora em vírgula flututante).

b) O que são os registos-resultado?

Os registos-resultado são os registos que preservam a informação lançada na fase de lançamento (issue) do algoritmo de Tomasulo. Basicamente são os buffers de load/store e as estações de reserva.

c) "As instruções, até à sua execução, estão em-ordem, com o algoritmo de Tomasulo". Concordas com a afirmação?

Não. Afirmar tal frase é semelhante a dizer que as estações de reserva se comportam como filas (FIFOs), o que está errado. Como estamos perante uma aplicação de agendamento dinâmico, o que acontece é que as instruções são ordenadas no seu lançamento mas perdem a ordem quando estão nas estações de reserva. No entanto, há uma pequena exceção: porque as instruções que manipulam dados da memória deverão ter mais cuidado na possibilidade de efetuarem hazards de dados (devem previnir mais tais acontecimentos), estas serão ordenadas em-ordem no buffer de load/store.

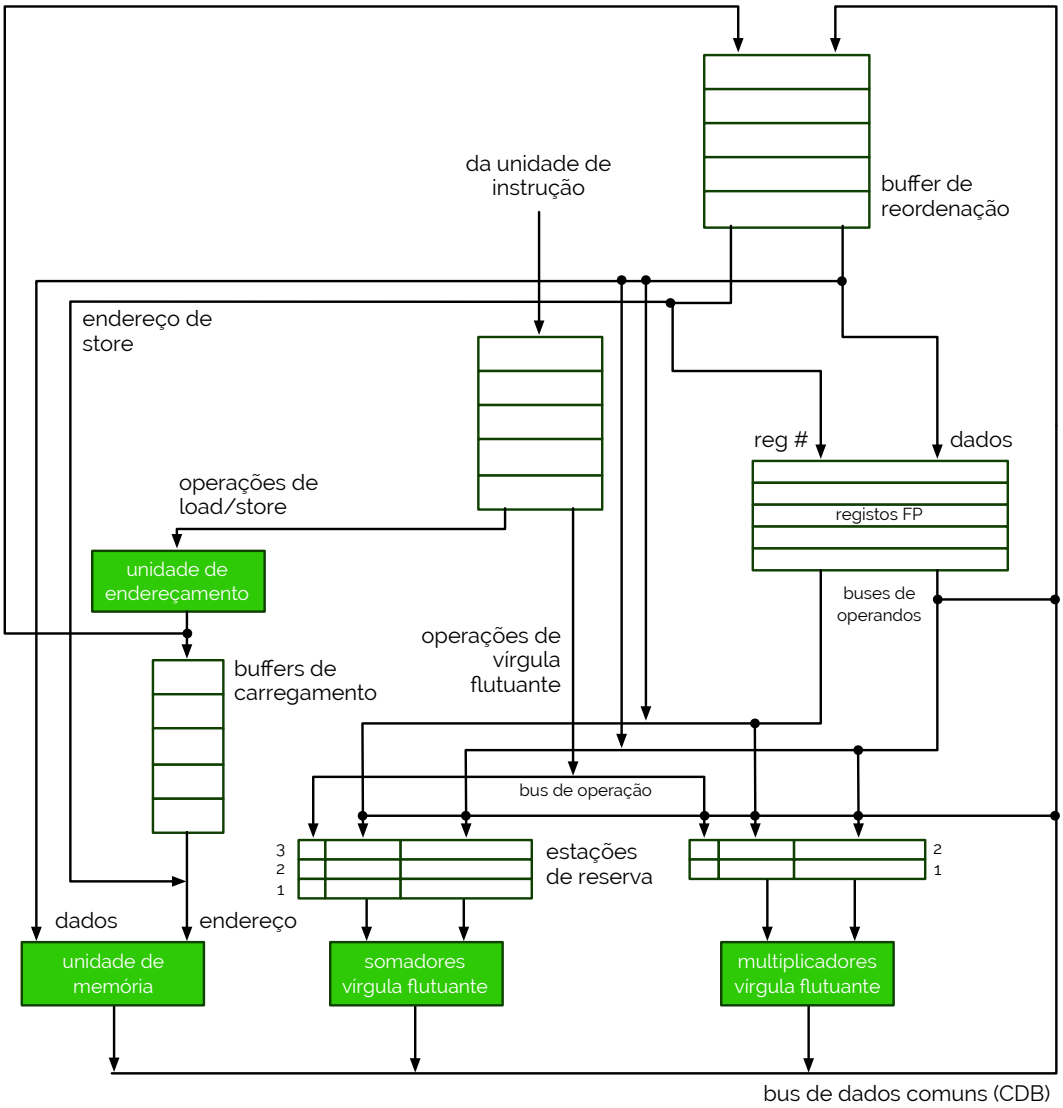
d) Como é que as instruções são identificadas nas estações de reserva? Qual é a importância da sua identificação?

Nas estações de reserva, dado que as instruções estão fora-de-ordem, estas ficam identificadas por uma etiqueta (tag field). A importância deste campo é tal que permite a identificação do registo-resultado que irá produzir um resultado a ser guardado num registo. Um valor de zero, nesta etiqueta, indica que nenhuma instrução ativa está a calcular o valor a ser guardado em tal registo.

e) Se o algoritmo de Tomasulo se baseia na correção de dependências de nome, como é que resolve os hazards do tipo RAW?

As renomeação de registos resolve as dependências de nome e os hazards de dados WAW e WAR. Já o hazard do tipo RAW é resolvido pelo atraso de execução até que todos os operandos estejam prontos.

17. Considera a figura abaixo.



a) Que aplicação de agendamento dinâmico é que está a ser representada na figura acima? Justifica a tua resposta.

A aplicação representada é a especulação por hardware. Esta identificação provém do facto desta se basear no algoritmo de Tomasulo e sobre ele implementar um novo buffer, denominado buffer de reordenação (ROB).

b) O que é que foi introduzido com esta implementação que o algoritmo de Tomasulo não assumia?

A especulação por hardware permite que se especule como é que o código deve ser executado. Mas se as execuções forem todas baseadas numa mera especulação, é fácil o processador trocar a ordem das instruções e depois não conseguir restaurar um determinado estado. Por esta mesma razão, todas as alterações são feitas num local que não as autentica como finais (assume alterações de código e escrita em registos, mas não escreve resultados diretamente no banco de registos). Criam-se assim duas fases globais de processamento: uma fase que consiste no lançamento e execução e outra fase que consiste no commit de tais alterações (caso esteja tudo de acordo com o correto). Este comportamento é semelhante ao processamento de transações nos mais vulgares Sistemas de Gestão de Bases de Dados (SGBDs), onde uma ação é conduzida e, se não houver erros até ao fim é feito o commit, caso contrário, é feito um rollback (restauro do estado inicial).

c) O que é o "buffer de reordenação" visível na figura? Qual foi o intuito da sua criação?

O buffer de reordenação (também denominado ROB) é o espaço de trabalho das instruções até que estas sejam sujeitas a commit. O intuito é, ao invés das instruções irem escrevendo no banco de registos os seus valores (resultados), escrevem-no no ROB até ao seu commit no banco de registos.

18. Considera o código seguinte, a ser executado num pipeline com especulação por hardware, considerando também a figura do exercício 17.

```
l.d     f6, 34(r2)
l.d     f2, 45(r3)
mul.d   f0, f2, f4
sub.d   f8, f6, f2
div.d   f10, f0, f6
add.d   f6, f8, f2
```

buffer de reordenação						
entrada	estado	busy	instrução	destino	value	ready
1		não				
2		não				
3		não				
4		não				
5		não				
6		não				

estações de reserva / buffers								
nome	busy	op	V_j	V_k	Q_j	Q_k	destino	A
load1	não							
load2	não							
add1	não							
add2	não							
add3	não							
mult1	não							
mult2	não							

estado de registos									
Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
busy									
ent. ROB									

a) Completa a tabela acima considerando que a primeira instrução foi lançada.

buffer de reordenação						
entrada	estado	busy	instrução	destino	value	ready
1	lançamento	sim	`l.d f6,34(r2)`	f6		não
2		não				
3		não				
4		não				
5		não				
6		não				

estações de reserva / buffers								
nome	busy	op	V_j	V_k	Q_j	Q_k	destino	A
load1	sim	load		reg[r2]			#1	reg[r2]+34
load2	não							
add1	não							
add2	não							
add3	não							
mult1	não							
mult2	não							

estado de registos									
Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
busy				sim					
ent. ROB				#1					

Considera a partir deste ponto, as seguintes latências: operações de soma e load/store - 2 ciclos; operações de multiplicação - 10 ciclos; operações de divisão - 40 ciclos.

b) Completa a tabela considerando que a operação de multiplicação foi lançada.

buffer de reordenação						
entrada	estado	busy	instrução	destino	value	ready
1	execução	sim	`l.d f6,34(r2)`	f6		não
2	execução	sim	`l.d f2,45(r3)`	f2		não
3	lançamento	sim	`mul.d f0,f2,f4`	f0		não
4		não				
5		não				
6		não				

estações de reserva / buffers								
nome	busy	op	V_j	V_k	Q_j	Q_k	destino	A
load1	sim	load		reg[r2]			#1	reg[r2]+34
load2	sim	load		reg[r3]			#2	reg[r3]+45
add1	não							
add2	não							
add3	não							
mult1	sim	mult		reg[f4]	#2		#3	
mult2	não							

estado de registos									
Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
busy	sim	sim		sim					
ent. ROB	#3	#2		#1					

c) Completa a tabela acima considerando o ciclo seguinte à alínea b).

buffer de reordenação						
entrada	estado	busy	instrução	destino	value	ready
1	escrita de res.	sim	`l.d f6,34(r2)`	f6	mem[reg[r2] + 34]	sim
2	execução	sim	`l.d f2,45(r3)`	f2		não
3	lançamento	sim	`mul.d f0,f2,f4`	f0		não
4	lançamento	sim	`sub.d f8,f6,f2`	f8		não
5		não				
6		não				

estações de reserva / buffers								
nome	busy	op	V_j	V_k	Q_j	Q_k	destino	A
load1	não							
load2	sim	load		reg[r3]			#2	reg[r3]+45
add1	sim	subtração	mem[reg[r2] + 34]			#2	#4	
add2	não							
add3	não							
mult1	sim	mult		reg[f4]	#2		#3	
mult2	não							

estado de registos									
Campo	F0	F2	F4	F6	F8	F10	F12	. . .	F30
busy	sim	sim		sim	sim				
ent. ROB	#3	#2		#1	#4				

d) Completa a tabela considerando que é feito o commit da primeira instrução.

buffer de reordenação						
entrada	estado	busy	instrução	destino	value	ready
1	commit	não	`l.d f6,34(r2)`	f6	mem[reg[r2] + 34]	sim
2	escrita res.	sim	`l.d f2,45(r3)`	f2	mem[reg[r3] + 45]	não
3	lançamento	sim	`mul.d f0,f2,f4`	f0		não
4	lançamento	sim	`sub.d f8,f6,f2`	f8		não
5	lançamento	sim	`div.d f10,f0,f6`	f10		não
6		não				

estações de reserva / buffers								
nome	busy	op	V_j	V_k	Q_j	Q_k	destino	A
load1	não							
load2	não							
add1	sim	subtração	mem[reg[r2] + 34]	mem[reg[r3] + 45]			#4	
add2	não							
add3	não							
mult1	sim	mult	mem[reg[r3] + 45]	reg[f4]			#3	
mult2	não			mem[reg[r2] + 34]	#3		#5	

estado de registos									
Campo	F0	F2	F4	F6	F8	F10	F12	. . .	F30
busy	sim	sim			sim	sim			
ent. ROB	#3	#2			#4	#5			

e) Completa a tabela considerando que a última instrução foi lançada.

buffer de reordenação						
entrada	estado	busy	instrução	destino	value	ready
1	commit	não	`l.d f6,34(r2)`	f6	mem[reg[r2] + 34]	sim
2	commit	sim	`l.d f2,45(r3)`	f2	mem[reg[r3] + 45]	sim
3	execução	sim	`mul.d f0,f2,f4`	f0		não
4	execução	sim	`sub.d f8,f6,f2`	f8		não
5	lançamento	sim	`div.d f10,f0,f6`	f10		não
6	lançamento	sim	`add.d f6,f8,f2`	f6		não

estações de reserva / buffers								
nome	busy	op	V_j	V_k	Q_j	Q_k	destino	A
load1	não							
load2	não							
add1	sim	subtração	mem[reg[r2] + 34]	mem[reg[r3] + 45]			#4	
add2	sim	soma		mem[reg[r3] + 45]	#4		#6	
add3	não							
mult1	sim	mult	mem[reg[r3] + 45]	reg[f4]			#3	
mult2	não			mem[reg[r2] + 34]	#3		#5	

estado de registos									
Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
busy	sim			sim	sim	sim			
ent. ROB	#3			#6	#4	#5			