

Exame 2022

$$\textcircled{1} \quad 6288_{10}, 5430_{10}$$

$$\begin{array}{r}
 6288 \xrightarrow{16} 393 \\
 -48 \\
 \hline
 140 \\
 -144 \\
 \hline
 48 \\
 -48 \\
 \hline
 \end{array}
 \quad
 \begin{array}{r}
 393 \xrightarrow{16} 24 \\
 -32 \\
 \hline
 73 \\
 -64 \\
 \hline
 9
 \end{array}
 \quad
 \begin{array}{r}
 24 \xrightarrow{16} 1 \\
 -16 \\
 \hline
 8
 \end{array}$$

$6288_{10} = 0000100001100000_2$

$$16 \times 3 = 48 \quad 16 \times 5 = 80$$

$$16 \times 9 = 144$$

$$16 \times 3 = 48$$

$$16 \times 4 = 64$$

$$5430_{10} = \boxed{0000100001100000_2}$$

$$\begin{array}{r}
 5430 \xrightarrow{16} 339 \\
 -48 \\
 \hline
 63 \\
 -48 \\
 \hline
 150 \\
 -144 \\
 \hline
 6
 \end{array}
 \quad
 \begin{array}{r}
 339 \xrightarrow{16} 21 \\
 -32 \\
 \hline
 19 \\
 -16 \\
 \hline
 3
 \end{array}
 \quad
 \begin{array}{r}
 21 \xrightarrow{16} 1 \\
 -16 \\
 \hline
 1
 \end{array}
 \quad
 \begin{array}{r}
 1 \xrightarrow{16} 0 \\
 -0 \\
 \hline
 0
 \end{array}$$

R: $6288_{10}, 0000100001100000_2$
 os últimos 2 bits
 estão a zero //

② Michael Flynn classificou os processadores em 2 categorias: instrução e dados. Ele divide em SISD, SIMD, MIMD e MISD. SISD é single instruction e single data, ou seja, o processador só faz uma instrução para um bloco de dados. SIMD é single instruction multiple data, executa a mesma inst. para vários dados, normalmente usado em processadores vetoriais e GPUs. MIMD Multiple Instruction Multiple Data executa várias instruções em vários dados normalmente usado em processadores multicore. MISD multiple instruction single data executa várias instruções num só set de dados, algo que não existe comercialmente. O processador single core é SISD porque só realiza uma instrução num set de dados de cada vez.

③ Pipeling consiste na divisão de uma supertask em sub-tasks menores, independentes e que demoram o mesmo tempo; com isto podemos lancer novas instruções a cada ciclo de relógio. No entanto uma instrução também é finalizada a todo ciclo do relógio. Desta modo, conseguimos executar n instruções em $K(n+k) + (n-1)$ em vez de nK (non-pipeline). Apesar disto o pipelining não muda o tempo de execução de uma instrução porque o

$$\sum T_{sub} > T_{super\ task}$$

④ Um hazard estrutural acontece quando duas unidades de hardware são requeridas ao mesmo tempo. Uma unidade single integer nunca é usada duas vezes no mesmo tempo porque só é requerida no estado Exec e só demora 1 ciclo de relógio. Temos múltiplas unidades FP, que podem demorar mais de 1 ciclo de relógio, assumimos que temos unidades suficientes para executar todos

as instruções de FP requeridas, caso contrário, necessitaremos de criar stalls entre instruções. O recurso que pode ser requerido duas vezes ao mesmo tempo é a memória, no estado MEM e estado IF para dar fetch de dados e instruções. A solução para este problema seria ter 2 memórias, uma para instruções e outra para dados (arquitetura Harvard).

→ Um hazard impede que a próxima instrução seja executada a tempo

Referir no inicio

5) Num pipeline de 5 estados clássico, as únicas hazards que não acontecem são WAW e WAR. Isto acontece porque o pipeline clássico não tem instruções fora de ordem. O WAW e WAR sãodeps. de maneira que podem ser resolvidos mudando ordenações dos registos e que só acontecem se uma instrução é reordenada.

Resposta só está correta se não houver FPs, Se houver, acontecem WAW

- 6
- the *write enable* signal is asserted high by the *memory controller* to mean a read operation is taking place
 - a N-bit address is placed in the address bus and is latched into the *row address buffer* upon the falling edge of the *row access strobe* asserted by the *memory controller*
 - the data values stored in the selected row of memory cells are then sensed and maintained in the *sense amplifiers array* to be later on written back to the memory cells
 - a M-bit address is placed next in the address bus and is latched into the *column address buffer* upon the falling edge of the *column access strobe* asserted by the *memory controller*
 - the data values kept in the selected column of the *sense amplifiers array* is latched into the *data out buffer* to drive the data bus

7

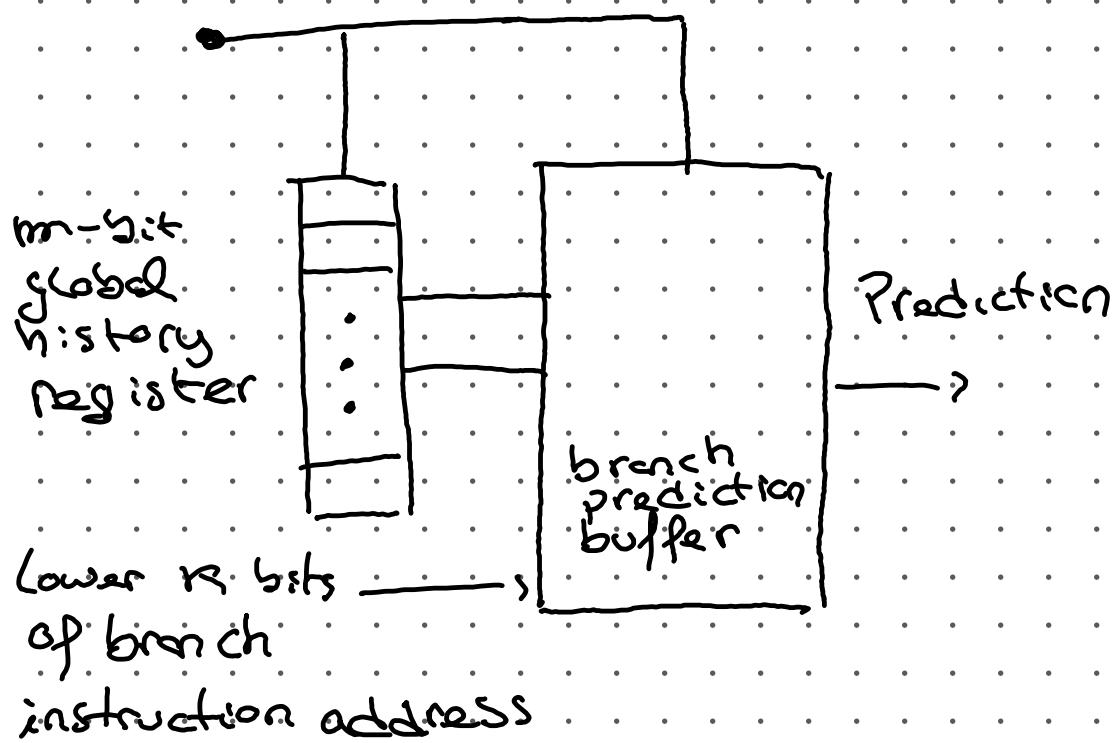
- The traditional approach to improve the behavior of a cache is to minimize the miss rate. Misses can be modelled into three basic categories
 - *compulsory misses* – misses that occur even if the cache had an infinite size; the very first access to any byte or word will always translate into a miss because the memory block where the byte or word resides must be brought first into the cache; they are also known as *cold-start misses* or *first-reference misses*
 - *capacity misses* – misses that occur in a fully associative cache; they are directly related to the cache size, if the cache is not large enough, memory blocks will be discarded during program execution and later will be retrieved because they are needed again
 - *conflict misses* – misses that occur specifically due to the internal organization of the cache; putting aside the case of a fully associative cache and considering the direct mapped cache as an instance of an 1-way set associative cache, memory blocks may be discarded and later retrieved simply because too many blocks are mapped in some of the sets; they are also known as *collision misses*.

+ associatividade { + capacity
- conflicts

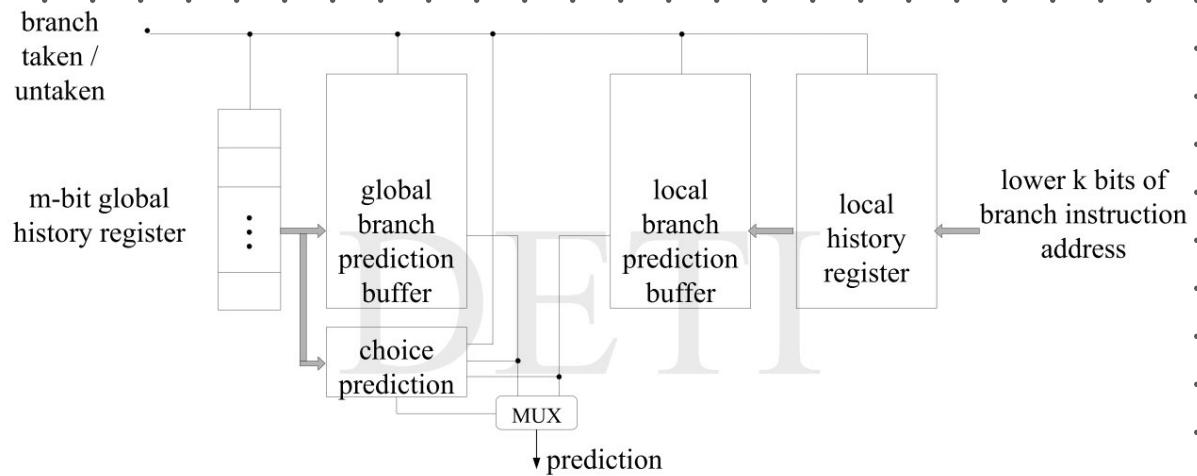
8) Capacity misses, quanto maior o bloco a ser transferido maior a prob de não haver espaço e haver um miss.

⑨

branch taken/not taken



Tournament Branch Predictor



10) a)

Especulação consiste no lançamento de instruções mesmo que tenham dependências de controlo, tendo em conta que os efeitos de uma especulação incorreta podem ser revertidos.

b)

Introduzindo um ROB podemos fazer isto, dado que este até chegar ao estado de "com:t", em que considerarmos se o branch foi realmente tomado ou não, (removendo as instruções nesse ROB).

11)

LV	V1, R1	$V1[k] = \text{mem}[R1 + k]$ (vector elements are adjacent)
SV	R1, V1	$\text{mem}[R1 + k] = V1[k]$ (vector elements are adjacent)
LVWS	$V1, (R1, R2)$	$V1[k] = \text{mem}[R1 + k \times R2]$ (vector elements are separated by the fixed value in R2)
SVWS	$(R1, R2), V1$	$\text{mem}[R1 + k \times R2] = V1[k]$ (vector elements are separated by the fixed value in R2)
LVI	$V1, (R1+V2)$	$V1[k] = \text{mem}[R1 + V2[k]]$ (vector elements are random separated by values in V2)
SVI	$(R1+V2), V1$	$\text{mem}[R1 + V2[k]] = V1[k]$ (vector elements are random separated by values in V2)

12) Uma GPU é considerada uma arquitetura SIMD porque cada uma das suas unidades de processamento contém 1 CU e várias ES, normalmente 32 igual ao warp size, este conjunto forma um SIMD, uma unidade que executa uma instrução para vários sets de dados. Como uma GPU contém várias unidades SIMD, por isso, executar várias instruções diferentes sobre dados diferentes, consideremos que GPU tem uma arquitetura SIMD.