INF01113 - Organização de Computadores B
2ª Lista de Exercícios

Aula – Processadores Superescalares, VLIW e paralelismo

1) [4.1 Henessy – Computer Architecture] Para o seguinte fragmento de código, liste todas as dependências existentes (dependências de saída, anti-dependências e dependências verdadeiras). Demonstre que o laço não é paralelizável.

```
for(i = 2; i < 100; i++){
        a[i] = b[i] + a[i];
        c[i-1] = a[i] + d[i];
        a[i-1] = 2 * b[i];
        b[i+1] = 2 * b[i];
}
```

2)[4.2 Henessy – Computer Architecture] Este é um laço não muito comum. Liste todas as dependências e então reescreva o mesmo de maneira a torná-lo paralelizável.

```
for(i = 1; i < 100; i++){
        a[i] = b[i] + c[i];
        b[i] = a[i] + d[i];
        b[i+1] = a[i] + e[i];
}
```

3)[4.3 Henessy – Computer Architecture] Para o seguinte fragmento de código, liste as dependências de controle. Para cada dependência de controle, diga se a instrução pode ser escalonada (executada) antes da instrução de desvio (if) baseado nas referências de dados. Assuma que todas as referências são mostradas, que todos os valores são definidos antes do seu uso, e que somente b e c são utilizadas depois deste segmento. Ignore quaisquer possíveis exceções.

```
if (a > c){
        d = d + 5;
        a = b + d + e;
}
else{
        e = e + 2;
        f = f + 2;
        c = c + f;
}
b = a + f;
```

4)[6.30 Henessy – Computer Architecture] – Analisando o exemplo a seguir [exemplo da seção 6.8 Henessy – Computer Architecture, item Um Exemplo Simples de Escalonamento de Código Superescalar]:

Exemplo

Como o loop a seguir deve ser escalonado para ser executado em uma implementação pipeline do MIPS?

```
Loop:   lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $si, $s1, -4
        bne     $s1, $zero, Loop
```

Reordene as instruções para evitar tanto quanto possível os conflitos do pipeline.

Solução

As três primeiras instruções apresentam dependência de dados, e exemplo das duas ultimas. A tabela abaixo mostra o melhor escalonamento para estas instruções. Observe que exatamente um par de instruções executa no modo superescalar. São necessários quatro ciclos de clock para cada interação do loop; com 4 ciclos para executar 5 instruções, obtemos uma modesta CPI de 0,8 comparada ao melhor caso, de 0,5.

	Instruções de UAL ou de desvio condicional	Instrução de transferência de dados	Ciclos de clock
Loop:		Lw $t0, 0($s1)	1
	Addi $s1, $s1, -4		2
	Addu $t0, $t0, $s2		3
	Bne $s1, $zero, Loop	Sw $t0, 4($s1)	4

Considere os benefícios de se usar as técnicas do desdobramento do loop no código que é executado no pipeline padrão do MIPS desenvolvido na Seção 6.8 Henessy – Computer Architecture. Mais especificamente, considere o código a seguir, que foi desdobrado, mas ainda não escalonado. Neste caso o código foi desdobrado apenas uma vez, considerando que o índice do loop é múltiplo de dois (isto é, $1 é múltiplo de oito):

```
Loop:   lw      $t0, 0($s1)
        addu    $t0, $t0 , $s2
        sw      $t0, 0($s1)
        lw      $t1, -4($s1)
        addu    $t1, $t1 , $s2
        sw      $t1, -4($s1)
        addi    $s1, $s1 , -8
        sw      $s1, $zero, Loop
```

Em primeiro lugar, escalone este código para executar rapidamente no pipeline padrão do MIPS (suponha que ele suporta as instruções addi e addu). Levando em conta as paradas que forem necessárias, compare a diferença de performance entre o código original desdobrado descrito abaixo, [que aparece na seção 6.4 Henessy – Computer Architecture, item adiantamento] e o seu código, desdobrado e escalonado.

5)[Stallings 13.2] Considere a seguinte seqüência de instruções:

```
1 ADD   t3, t1, t2
2 lw    t6, 0x0(t3)
3 and   t7, t5, 3
4 add   t1, t6, t7
5 srl   t7, t0, 8
6 or    t2, t4, t7
7 sub   t5, t3, t4
8 add   t0, t1, 10
9 lw    t6, 0x0(t5)
10      sub     t2, t1, t6
11      and     t3, t7, 15
```

Assuma o uso de um pipeline de 4 estágios: fetch, decode/issue, execute, writeback. Assuma que todos os estágios do pipeline levam 1 ciclo de clock exceto para o estágio de execução: as instruções aritméticas inteiras simples e as instruções lógicas consomem 1 ciclo no estágio de execução, enquanto que as instruções de carga da memória (LW) consomem 5 ciclos.

Se nós temos um pipeline simples, mas que permite execução fora de ordem, nós podemos construir a seguinte tabela para a execução das primeiras 7 instruções:

Instruction	Fetch	Decode	Execute	Writeback
1	0	1	2	3
2	1	2	4	9
3	2	3	5	6
4	3	4	10	11
5	4	5	6	7
6	5	6	8	10
7	6	7	9	12

As entradas sobre os quatro estágios do pipeline indicam o ciclo de clock em que cada instrução começa naquele estágio. Neste programa, a segunda instrução ADD (4) depende da instrução LW (2) para um de seus operandos (t6). Pelo fato da instrução LW consumir 5 ciclos de clock e a lógica de despacho encontrar a instrução ADD que dela depende 2 ciclos depois, a lógica de despacho deve retardar a execução da instrução ADD por 3 ciclos. Com a capacidade de execução fora de ordem, o processador pode parar a execução da instrução 3 no ciclo de clock 4, e então despachar as três instruções independentes seguintes, que entram em execução nos ciclos 6, 8 e 9. A instrução LW (1) termina a execução no ciclo 9, e então a instrução ADD (3) dependente pode ser lançada para execução no ciclo 10. De acordo com o exposto:

Complete a tabela acima para as instruções 8 a 11.
Refaça a tabela, assumindo que não existe a capacidade de execução fora de ordem. Quantos ciclos são economizados através do uso da execução fora de ordem?
Refaça a tabela, assumindo uma implementação superescalar que pode manipular 2 instruções por vez em cada ciclo.

6)Se o fragmento de código abaixo fosse executado em um processador superescalar com um número infinito de unidades de execução (**com latência de um ciclo para todas as operações**) quantos ciclos seriam necessários para executá-lo? Em outras palavras, que limitações sobre o tempo de execução são impostas pelas dependências existentes no fragmento de código?

```
LD r7, (r8)
SUB r10, r11, r12
MUL r13, r7, r11
ST (r9), r13
ADD r13, r2, r1
LD r5, (r6)
SUB r3, r4, r5
```

7)Em quantos ciclos o seguinte fragmento de código seria executado em um processador superescalar (**com execução em ordem**) com duas unidades de execução, onde todas as instruções possuem latência de um ciclo e ambas as unidades de execução podem executar qualquer instrução?

```
LD r1, (r2)
SUB r4, r5, r6
ADD r3, r1, r7
MUL r8, r3, r3
ST (r11), r4
ST (r12), r8
ADD r15, r14, r13
SUB r10, r15, r10
ST (r9), r10
```

8)Quantos ciclos seriam necessários para executar a seqüência de instruções a seguir em um processador superescalar (**com execução em ordem**) com 4 unidades de execução, onde qualquer unidade pode executar qualquer operação, sendo que operações do tipo *load* possuem uma latência de 2 ciclos e todas as demais possuem latência de apenas 1 ciclo?

```
ADD r1, r2, r3
SUB r5, r4, r5
LD r4, (r7)
MUL r4, r4, r4
ST (r7), r4
```

9)Em quantos ciclos o fragmento de código do problema 8 seria executado em um processador superescalar (**agora com execução FORA DE ORDEM**) com duas unidades de execução, onde todas instruções possuem latência de um ciclo e ambas as unidades de execução podem executar qualquer instrução? Assuma que a janela de instruções é grande o suficiente para conter toda a seqüência de instruções e que o processador utiliza um método "guloso"* para a execução das instruções.

> * *Método guloso:* as instruções vão sendo escalonadas de acordo com as possibilidades (unidades de execução disponíveis e dependências), sendo que, quando há mais de uma candidata para o escalonamento, a opção é pela primeira instrução que aparece no código. Não existe preocupação em encontrar um escalonamento ótimo de instruções. Entende-se por "ótimo" o escalonamento que possibilitaria a execução correta de todas as instruções no menor número de ciclos.

10)Quantos ciclos seriam necessários para executar a seqüência de instruções do exercício 8 em um processador superescalar (**agora com execução FORA DE ORDEM**) com 4 unidades de execução, onde qualquer unidade pode executar qualquer operação, sendo que operações do tipo *load* possuem uma latência de 2 ciclos e todas as demais possuem latência de apenas 1 ciclo? Assuma escalonamento guloso e que a janela de instruções é grande o bastante para conter todo fragmento de código.

11)Quantos registradores são necessários para permitir que o mecanismo de renomeação de registradores elimine todas as dependências do tipo WAR (*write-after-read*) e WAW (*write-after-write*) do seguinte conjunto de instruções?

```
LD r1, (r2)
ADD r3, r4, r1
SUB r4, r5, r6
MUL r7, r4, r8
ASH r8, r9, r10
SUB r11, r8, r12
DIV r12, r13, r14
ST (r15), r12
```

12)Mostre uma maneira de como a renomeação de registradores transformaria o fragmento de código do exercício anterior. Assuma que o processador possui registradores suficientes para efetuar a requerida renomeação.

13)Em quantos ciclos a seqüência original do problema 11 e a reescrita no exercício 12 seriam executadas em um processador superescalar (**com execução fora de ordem**) com 4 unidades de execução, cada uma podendo executar qualquer instrução? Assuma que a latência de todas instruções é de 1 ciclo, utilize escalonamento "guloso" e que a janela de instruções é capaz de conter todo código.

14)Mostre como um compilador poderia escalonar o código do problema 8 para a execução em um processador VLIW com o mesmo numero de unidades de execução e latências de instrução do exercício original. Ao contrário dos problemas de execução fora de ordem, assuma que o compilador examina todas as possibilidades de ordenamento de instruções para encontrar o melhor escalonamento possível (Isso advém do fato do compilador poder utilizar muito mais recursos e tempo para encontrar o melhor escalonamento do que o usualmente disponível no hardware de processadores superescalares). Certifique-se de incluir as instruções NOP para operações não utilizadas.

15)Mostre como um compilador poderia escalonar o seguinte programa para execução em um processador VLIW com 4 unidades de execução, sendo que cada uma pode executar qualquer tipo instrução. Instruções de *load* possuem uma latência de 3 ciclos e todas as outras instruções têm latência de apenas 1 ciclo. Tenha em mente que em um VLIW, o valor antigo de um registrador de destino de uma instrução mantém-se disponível para leitura até que a referida instrução seja completada.

```
SUB r4, r7, r8
MUL r10, r11, r12
DIV r14, r13, r15
ADD r9, r3, r2
LD r7, (r20)
LD r8, (r21)
LD r11, (r22)
LD r12, (r23)
LD r13, (r24)
LD r15, (r25)
LD r3, (r30)
LD r2, (r31)
ST (r26), r4
ST (r27), r10
ST (r28), r14
ST (r29), r9
```

16)Por que desenrolar um laço fornece um incremento de desempenho?

17)Mostre como um compilador desenrolaria o seguinte laço infinito 4 vezes. Inclua o código inicial (o código que calcula todos os ponteiros necessários para as operações de cada iteração do laço desenrolado). Assuma que o processador possue tantos registradores quanto o necessário.

```
loop:
       LD r1, (r2)
       LD r3, (r4)
       LD r5, (r6)
       ADD r1, r1, r3
       ADD r1, r1, r5
       DIV r1, r1, r7
       ST (r0), r1
       ADD r2, #4, r2
       ADD r4, #4, r4
       ADD r6, #4, r6
       ADD r0, #4, r0
       BR loop
```

18)Mostre como um compilador poderia escalonar as versões originais e modificadas ("desenroladas") do laço do exercício anterior para a execução em um processador VLIW com 4 unidades de execução que podem executar qualquer tipo de instrução. Assuma latência de 3 ciclos

para instruções LD e 2 ciclos para DIVs e ADDs. Assuma que o atraso do *branch* é grande o suficiente para que todas as operações de uma iteração se completem antes do início da próxima iteração. Como nos outros exercícios de VLIW, assuma que o compilador examine todas as possibilidades de ordenamento para encontrar aquela que permite a execução no menor número de instruções possível. Para o laço modificado, escalone apenas o laço e não o código inicial.

19)Quais as diferenças entre máquinas superescalares e máquinas VLIW

Aula – Memórias Cache

20)[7.2 Henessy – Computer Architecture] – Descreva as características gerais de um programa que exiba alta localidade temporal e baixa localidade espacial, no que diz respeito aos acessos a dados. Escreva um programa que exemplifique a situação. Esse exemplo pode ser em pseudocódigo.

21)[7.3 Henessy – Computer Architecture] - Descreva as características gerais de um programa que exiba baixa localidade temporal e alta localidade espacial, no que diz respeito aos acessos a dados. Escreva um programa que exemplifique a situação. Esse exemplo pode ser em pseudocódigo.

22)[7.7 Henessy – Computer Architecture] – A seguir apresentaremos uma série de referências, a endereços de palavras: 1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17. Considerando uma cache mapeada diretamente, com 16 blocos de uma palavra cada, que estava inicialmente vazia, identifique cada referência na lista como uma falta ou como um acerto no acesso a cache e mostre o conteúdo final da cache, após o processamento de todas essas referências.

23)[7.8 Henessy – Computer Architecture] – Usando a série de referências do exercício 22, mostre as faltas, os acertos e o conteúdo final da cache quando os acessos forem a uma cache mapeada diretamente, com quatro palavras por bloco, e um *tamanho total* de 16 palavras.

24)[7.11 Henessy – Computer Architecture] – Considere uma hierarquia de memória usando uma das três organizações para a memória principal mostradas na Figura 1 [7.13]. Suponha que o tamanho do bloco da cache é de 16 palavras, que o comprimento da organização b da figura é de quatro palavras, e que a organização c tem quatro bancos. Se a latência de memória principal para um novo acesso for de 10 ciclos de clock e o tempo de transferência for de 1 ciclo, qual será a penalidade por falta para cada uma dessas organizações?

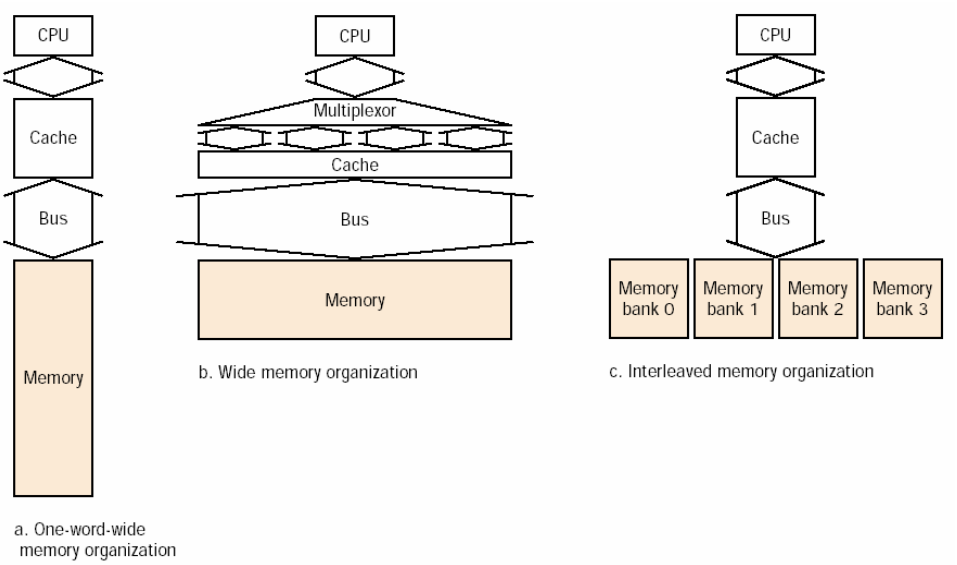

Figura 1 [7.13 Henessy – Computer Architecture]

25)[7.12 Henessy – Computer Architecture] – Considere um processador ligado a uma cache com um bloco de 16 palavras, com uma taxa efetiva de faltas por instrução de 0.5%. Suponha que a CPI, sem levar em conta as faltas no acesso à cache, é de 1,2. Utilizando as memórias descritas na Figura 1 [7.13 Henessy – Computer Architecture] e no exercício 24 [7.11 Henessy – Computer Architecture], diga quão mais rápido vai ficar o processador quando utilizarmos cada uma das organizações de memória da figura 1.

26)[7.13 Henessy – Computer Architecture] Considere uma cache C1, mapeada diretamente com 16 blocos de uma palavra cada. Considere também uma cache C2, mapeada diretamente, com blocos de 4 palavras cada. Suponha que a penalidade por falta em C1 é de 8 ciclos de clock e para C2 é de 11 ciclos de clock. Partindo do princípio de que as caches estão inicialmente vazias, encontre um conjunto de referências para o qual C2 tenha uma taxa de faltas menor do que a de C1, mas gaste mais ciclos nas faltas do que C1. Use endereçamento a palavra.

27)[7.18 Henessy – Computer Architecture] – Suponha que você possua 18 módulos de memória SRAM, cada um dos quais com 32K x 8, para construir uma cache de instruções para um processador que endereça 32 bits. Qual o maior tamanho (isto é, o maior tamanho de área para armazenamento de informações em bytes) da cache de instruções mapeada diretamente que você pode construir com blocos de uma palavra de (32 bits)? Mostre a divisão do endereço de 32 bits em cada um dos seus componentes relacionados ao acesso da cache (como exemplo, veja a Figura 2 [7.8 Henessy – Computer Architecture]) e descreva como os vários módulos de SRAM serão usados. (Dica: Pode ser que você não precise usar todos os módulos de memória SRAM colocados à sua disposição.)

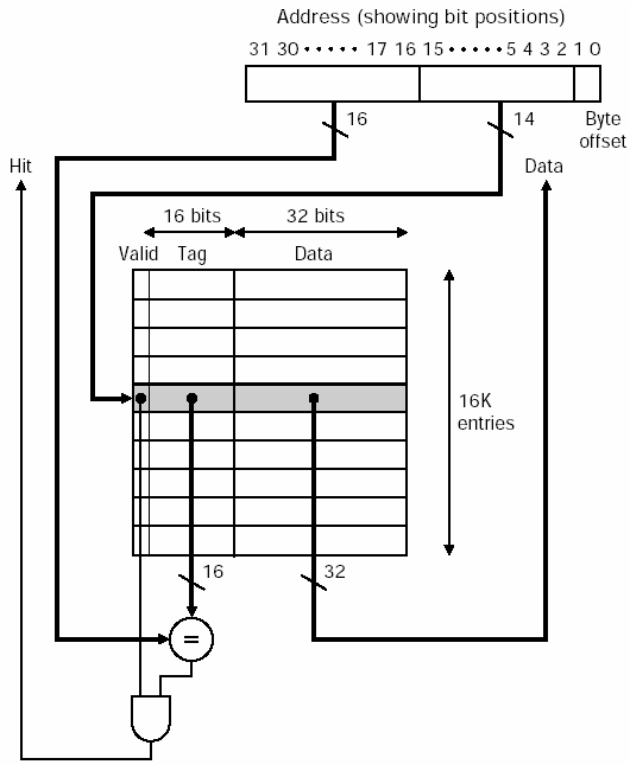

Figura 2 [7.8 Henessy – Computer Architecture]

28)[7.20 Henessy – Computer Architecture] – Usando a série de referências do Exercício 22 [7.7 Henessy – Computer Architecture], mostre os acertos, as faltas e o conteúdo final armazenado em uma cache associativa por conjunto com duas posições, com blocos de uma palavra e um *tamanho tota*l de 16 palavras. Considere o uso da substituição LRU.

29)[7.22 Henessy – Computer Architecture] - Usando a série de referências do Exercício 22 [7.7 Henessy – Computer Architecture], mostre os acertos, as faltas e o conteúdo final armazenado em uma cache totalmente associativa, com blocos de 4 palavras e *tamanho tota*l de 16 palavras. Considere o uso da substituição LRU.

30)[7.27 Henessy – Computer Architecture] – Considere três máquinas com diferentes configurações na cache:

Cache 1: mapeamento direto com blocos de uma palavra
Cache 2: mapeamento direto com blocos de 4 palavras
Cache 3: associativa por conjunto com duas posições e blocos de 4 palavras

Foram feitas as seguintes medidas para a taxa de faltas:

Cache 1: taxa de faltas para instruções de 4% e para dados de 8%
Cache 2: taxa de faltas para instruções de 2% e para dados de 5%
Cache 3: taxa de faltas para instruções de 2% e para dados de 4%

Considere que metade das instruções contém uma referência a dado. Suponha que a penalidade por falta no acesso a cache é de (6 + tamanho do Bloco em palavras). A CPI para este *workload* foi medida em uma máquina com a cache 1, e o valor obtido foi 2,0. Determine qual das máquinas gasta mais ciclos para atender a uma falta no acesso à cache.

31)[7.28 Henessy – Computer Architecture] – Os tempos de ciclo para as máquinas do

Exercício 30 [7.27 Henessy – Computer Architecture] são de 2 ns para a primeira e para a segunda máquina, e de 2,4 ns para a terceira. Determine qual a máquina mais rápida e qual a mais lenta.

32)[7.29 Henessy – Computer Architecture] – O programa a seguir, escrito em C, roda em uma máquina (sem otimização) com uma cache com blocos de quatro palavras (16 bytes) e que armazena 256 bytes de informação:

```
int i,j, c, stride, array[256];
...
for (i=0; i<10000; i++)
        for (j=0; j<256; j=j+stride)
                c = array[j] + 5;
```

Se considerarmos apenas a atividade da cache gerada pelas referências ao array, e se supusermos que valores inteiros ocupam uma palavra, qual será a taxa de faltas esperada quando a cache for mapeada diretamente com stride = 132? E se o stride for igual a 131? Algo mudaria se a cache fosse associativa por conjunto com 2 posições?

Aula – Memória Virtual

33) [7.32 Henessy – Computer Architecture] Considere um sistema de memória virtual com as seguintes propriedades:

- Endereço virtual de 40 bits referenciando byte
- Páginas de 16KB
- Endereço físico de 36 bits referenciando byte

Com base nisto, calcule o tamanho total da tabela de páginas para cada um dos processos desta máquina, assumindo que os bits de residência, de proteção, de modificação e de utilização gastam um total de quatro bits e que todas as páginas virtuais estão sendo usadas. (Suponha que os endereços do disco não estão armazenados na tabela de páginas)

34) [7.33 Henessy – Computer Architecture] Suponha que o sistema de memória virtual do exercício anterior é implementado com um TLB associativa por conjunto de duas posições, com um total de 256 entradas. Mostre como se realiza o mapeamento dos endereços virtuais em endereços físicos, por meio de uma figura semelhante à figura 3 [7.25 - Henessy – Computer Architecture]. Certifique-se de ter identificado o tamanho de cada um dos campos e sinais envolvidos.

35) [7.37 Henessy – Computer Architecture] Alguns programas complexos, a exemplo de simulações sobre o comportamento do tempo, são carregados em um computador onde devem rodar, ininterruptamente, por longos intervalos de tempo. Para processar tais aplicações, utilizam-se máquinas muito caras, adquiridas para este propósito. Discuta os motivos de a técnica da memória virtual ser ou não ser útil para máquinas projetadas para rodar especificamente esse tipo de aplicação. Você acha que uma cache teria utilidade neste caso?

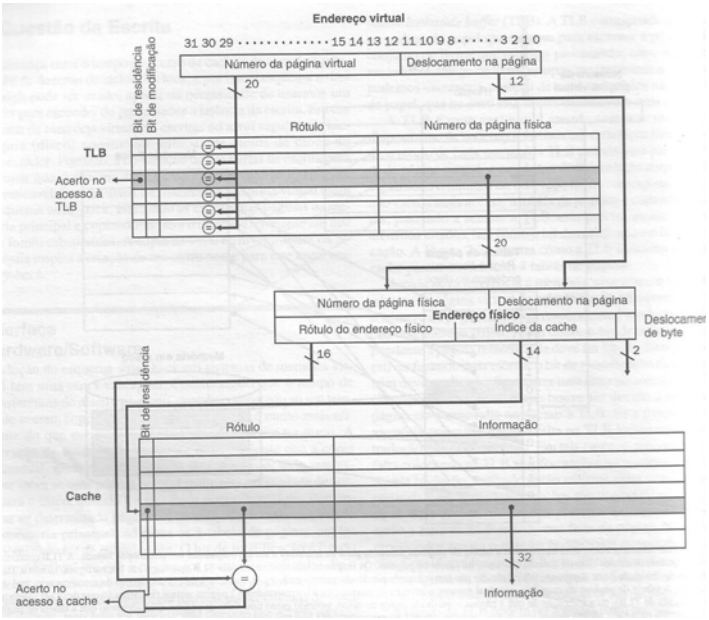

Figura 3 [7.25 Henessy – Computer Architecture]

Aulas várias

36)Qual o CPI de uma maquina com pipeline, se o hit na cache de instruções é de 95%, e o da cache de dados é de 90%? 30% das instruções fazem acesso à memória, e destas, 30% são stores. A penalidade de acesso à memória principal é de 50 ciclos, e a política de escrita é de write-through sem buffer.
Repita para o caso onde uma cache L2 foi adicionada. A penalidade para uma ida na L2 é de 5 ciclos. O hit-ratio na L2 é de 98% para instruções e 95% para dados.
Repita ambos os exercícios assumindo que existe um write-buffer, e que somente em 10% das escritas deve-se efetivamente pagar a penalidade.

37)Qual seria a organização de uma cache para processamento de vídeo? E para um programa como um compilador ou um jogo de IA? Explique separadamente para instruções e dados.

38)Quais as diferenças entre segmentação e paginação? Para que servem, fundamentalmente?

Referências:

[Stallings] STALLINGS, William. *Computer Organization and Architecture: Designing for Performance*. 4a. Ed. Upper Saddle River: Prenctice-Hall, 1996.

[Wilkinson] WILKINSON, Barry. *Computer Architecture – Design and Performance*. 2a. Ed. Hertfordshire: Prenctice-Hall, 1996.

[Hennessy] HENNESSY, J.; PATTERSON, D. *Computer Architecture: A Quantitative Approach*. 2a. ed. San Francisco: Morgan Kaufmann, 1996.