P1, P2, P3

Luke Skywalker

17 de julho de 2015

1. Definir o alcance de um branch e jump:

   (a) a partir do endereço 0, o jump é 0xFFF FFFC

   (b) a partir do endereço 0, o branch é:

```
= pc + 4 + 0111 1111 1111 1111 palavras
= 0 + 4 + 0111 1111 1111 1111 00 bytes
= 4 + 0001 1111 1111 1111 1100 bytes
= 4 + 0x1FFFC = 0x20000
```

2. Ele dava uma memória virtual de 32 bits com endereços físicos de 28 bits, dizia que o tamanho da página era de 1 MB $= 2^{20}$. Pedia para calcular quantos bits ocupava o endereço físico disso na tabela de páginas; $= 28 - 20 = 8$.

3. Ele dava um cache com 1024 KB, com blocos de 64 bytes. E era uma cache do tipo 16-way. Uma memória com endereços de 40 bits. Calcular o tamanho do índice.

   Resposta: Ela tem (64 *bytes*)/(4 *bytes*) = 16 palavras por blocos. Ela é uma cache do tipo 16-way, portanto possui 16 blocos por conjunto. Ela possui (1024 KB) / (64 bytes) = (2^{20} bytes) / (2^6 bytes) = 2^{14} blocos Então (2^{14} blocos) / (16 blocos por conjunto) = 2^{10} conjuntos Assim precisamos de 10 bits para indexar os 1024 conjuntos.

4. Pedia para calcular o tamanho da TAG anterior 'd)'. $= 40 - 10(conjuntos) - 4(wordOFFset) - 2(byteOFFset) = 24$ bits de TAG.

5. Transformar para binário a instrução subi $t0, a0, 1. Lembrando que ela é pseudo e o certo seria um ADDI com um número complementado; isto é, transformar em:

```
addi, $t0, $a0, ffff
RT  = RS + EXT
$t0 = $a0 + -1
OPCODE  | RS     | RT     | IMMEDIATE
OPCODE  | $a0    | $t0    | -1
0010 00 | 00 100 | 0 1000 | 1111 1111 1111 1111
```

6. Relacionada a escalonamento estático, ela dá um código, e você tem que escalonar ele por que o processador não tem controle de hazards, e não pode desfazer as instruções feitas utilizando o branch prediction estático programado sempre para desvio não tomado, e precisa que se adicione nop (no operation) quando há necessidade, e o hardware, possui todos os atalhos, (full forwarding):

```
addi $s0, $zero, 0
loop: lw $t0, 0($s0)
add $v0, $v0, $t0
addi $s0, $s0, -4
bne $s0, $zero, loop
jr $ra
```

Que fica igual a:

```
addi $s0, $zero, 0
loop: lw $t0, 0($s0)
addi $s0, $s0, -4
add $v0, $v0, $t0
bne $s0, $zero, loop
nop
jr $ra
```

7. Ele pedia para fazer loop unrolling com 2 vezes, para emissão múltipla de 2 instruções por ciclo com qualquer combinação possível, e a pegadinha dele é que no anterior o branch prediction era estático, contava com desvio não tomado, e não podia desfazer as instruções que ele executava, agora ele pode desfazer instruções indevidamente previstas por um previsor dinâmico que agora, está programado para sempre tomar os desvios). O hardware também possui todos dos atalhos físicos, (full forwarding). Ah, outras pegadinhas: é só para desenrolar o laço, e descartar as instruções de fora: addi $s0, $zero, 0 e jr $ra.

loop: lw $t0, 0($s0)	nop
lw $t1, 0($s0)	addi $s0, $s0, -8
add $v0, $v0, $t0	nop
add $v0, $v0, $t1	bne $s0, $zero, loop
espaço vazio na tabela	espaço vazio na tabela

   (a) Qual o tamanho do LAÇO no item a) em bytes quando executado em uma máquila com emissão múltipla estática (nops também contam tamanho de código): Resposta: $4 * 8 = 32$ bytes.

   (b) Qual o tamanho do LAÇO no item a) em bytes quando executado em uma máquila com emissão múltipla dinâmica: Resposta: 4 * 8 = 32 bytes.

8. Um sistema possui uma cache primária .de dados (L1-D), uma cache primária de instruções (L1 -I), uma cache unificada secundária (L2) e uma memória principal (MP). A penalidade de falta da cache secundária foi calculada na Questão 1a (expressa em ciclos do FSB). A penalidade de falta da cache primária é 5 (expressa em ciclos de CPU). A frequência

de relógio da CPU é 5 vezes a frequência de relógio do FSB. A taxa de fracassos combinada das caches primárias (mr = número total de acessos a L2 / número total de acessos à memória) é de 2%. A taxa de faltas global (gmr = número total de acessos à MP / número total de acessos à memória) é de 0,5%. Sabe-se que um programa executa 1 milhão de instruções, das quais 25% são loads ou stores.

(a) Quantos ciclos de CPU são gastos no acesso a L2? Mostre seus cálculos. Resposta: 125000 ciclos. Cálculos: número total de acessos = $(0,25 \times I + I) = 1250000$ para o acesso a L2 temos uma penalidade de Ciclos = $I \times mr \times \ penalidade \ = 1250000 \times 0,02 \times 5 = 125000$ ciclos

(b) Quantos ciclos de CPU são gastos no acesso a MP? Mostre seus cálculos. ciclos de CPU para a memória principal = 36*5 = 180 ciclos para 1 acesso. Ciclos totais = $1250000 \times 0,005 \times 180 = 1125000$

(c) Quantos ciclos de CPU são gastos na execução total do programa? Mostre seus cálculos. Número total de acessos = $(0,25 \times I + I) = 1250000$

Questão: Suponha os seguintes cenários: Cenário 1: uma instrução j L reside no endereço 0; Cenário 2 um instrução jr $s1 reside no endereço 0. Para o MIPS, quais os endereços alvos máximos atingíveis pelos desvios em cada um dos cenários. Resposta: Cenário 1: = 0x 0FFF FFFC; Cenário 2: 0xFFFF FFFC.

Questão: Um dispositivo de saída é mapeado em memória. Sabe-se que ele se comunica para com a CPU através da técnica de consulta periódica ("polling"). O esboço de código abaixo mostra parte da implementação da consulta do dispositivo. Conhece-se o conteúdo de quatro dos registradores: $s0 = 0x 0000 0004; $s1 = 0x 0000 0008; $s2 = 0x 0000 ABC1; $s4 = 0x 0000 FFAC. Para o qual endereço de memória está mapeado o registrador de status do dispositivo de saída? Não é preciso justificar.

Endereço = 0x 0000 0004, Pois no programa abaixo ele está setando justamente o bit "interrupt enable", está fazendo-o no registrador que tem o endereço 0x4.

```
P: ...
   lw $s2, 0($s0)
   andi s3, s2, 0x00000001
   beq $zero, $s3, L
   ...
   sw $s4, 0($s1)
   ...
   ...
   ...
   j P
```

Questão: Sejam D7 a D0 dispositívos de E/S capazes de ativar ('1') e desativar ('0') as requisições de iterrupção IRQ7 e IRQ0, respectivamente, à entrada de um controlador de interrupções. O controlador possui uma codificação que garante a prioridade de D_i sobre D_{i+1} e contém 2 registradores mapeados em memória:

- ireq-req (mapeando em 0x1000 8000): cada um de seus bits armazena (do mais significativo para o menos significativo) o estado de cada uma das entradas IRQ7 a IRQ0 (respectivamente).
- mask-reg (mapeando em 0x1000 8001): armazena uma máscara de interrupção. Escreva um programa com no máximo 4 intruções nativas do MIPS que torne IRQ7 a requisição de alta prioridade. Restrição: todas as constantes devem ser escritas em hexadecimal (0x).

Resposta:

```
lui $t0, 0x1000
ori $t0, $t0, 0x8001
addi $t1, $zero, 0x80
sb $t1, 0($t0)
```

Questão: Afirmação: "Uma vez configurado na CPU para transferir um grande bloco de dados, um controlador DMA nunca emite um sinal de requisição de interrupção para a CPU; daí sua vantagem em relação do mecanismo de E/S acionado por interrupção". A afirmação é Verdadeira ou Falsa? Justificação: Resposta: F (V ou F). Justificativa: O controlador de DMA interrompe a CPU ao final do tratamento para indicar que o bloco foi transferido.

Questão: Uma cache tem capacidade nominal de 512 Kilobytes e associatividade do tipo 8-way (8 BLOCOS POR CONJUNTO), bloco de 128 bytes e utiliza as políticas LRU e "write-back". Além do bit de validade, essa cache utiliza um bit para implementar o critério LRU e um outro bit ("dirty bit") para implementar o mecanismo de "write-back". Sabe-se que os endereços são representados em 32 bits.

Questão: Quantos bits são utilizados para indexar a cache? Mostrar seus cálculos. Resposta: 9 bits. Cálculos: #blocos $= 512\ KB/(128\ \frac{B}{bloco}) = 4K = 2^{12}$ # conjuntos $= 2^{12}/2^3 = 2^9$

Questão: Quantos bits sao utilizados para TAG? Mostrar seus cáculos. Resposta: 16 bits. Cálculos: $TAG\ + INDEX\ + BLOCKOFFSET\ = 32 \rightarrow TAG\ = 32 - 9 - 7 = 16(bloco\ tem\ 2^7\ bytes \rightarrow 7$ bits para block offset)

Questão: Qual o número total de bits armazenados na cache (expresso em Kbits; $K = 2^{10}$)? Justifique, mostrando seus principais cálculos.

Resposta: 4172 Kbits. Cálculos: # blocos $= 2^{12}$ Comentário: Note que o esquema da cache é o seguinte:

```
conj 0:  [VB][LRU][DB][tag 16 bits][data 128B] ... [VB][LRU][DB][tag 16
bits][data 128B]
.
.
.
```

```
conj 2^9-1:  [VB][LRU][DB][tag 16 bits][data 128B] ... [VB][LRU][DB][tag 16
bits][data 128B]
```

Cálculos: Cada bloco contém:

- CONTROL: 3 bits;
- TAG: 16bits;
- DATA: 128 x 8 = 1024 bits;

Total de bits = $2^{12} \times (3 + 16 + 1024) = 2^{18} x 2^2 x 1043\ bits\ = 2^{10} \times 4172\ bits\ =$ 4172 Kbits [SIMILAR AO EXERCÍCIO 5.x.y, MAS PARA CACHE 16-WAY!]

Questão: Um processador possui um subsistema de memória que consiste de uma uma cache de dados e uma memória principal. São executados I instruções de um programa das quais 15% são do tipo "load" e 5% do tipo "store". Suponha que a CPU tem um pipeline ideal (sem "hazards") que inicie uma intrução a cada ciclo. Assumindo que, para ambas as caches, a taxa de acertos é de 90% e a penalidade por causa de falta na cache é de 10 ciclos, calcule:

a) O número de ciclos de parada ("stall") gerados pelas faltas no acesso a instruções. Resposta: I ciclos. Cálculos: I x 10% x 10 [ciclos] = I ciclos. Comentário: Load e Stores são Dados, então não entra nem 15%, nem 5% aqui. Porém, entram 90% ou 10% (complemento). Se você quiser calcular o importânte que é stall gerados por instruções, você usa 10%, pois 90% de acertos não participa da penalidade por stalls.

b) O número de ciclos de parada ("stall") gerados pelas faltas no acesso a dados. Resposta: 0,2I ciclos. Cálculos: (20% x I) x 10 x 10% = 0,2I ciclos. Comentário: Aqui ele somou todas as porcentagens com relação a dados (LS = 15%+5%). Ou seja, De todas as instruções temos 20% que são do tipo LS. Agora que temos essa porcentagem em mãos, quanto será a penalidade total por elas? Note, que ele diz "penalidade por falta na cache é de 10 ciclos", isto é, falta de dados e falta de instruções. Poderia ser 10 ciclos de penalidade para instruções e "12" ciclos de penalidade para dados.

c) O número médio de ciclos por intrução total, incluindo o efeito de todas as falhas, CPI médio: Resposta: 2,2 [ciclos/instr.] Cálculos: CPI médio = CPI ideal x CPI stall = (I + (I + 0,2 I)) / I = I + 1,2I = 2,2. Comentário: Soma das médias. (1 x I [ciclos]) / I instr. seria o CPI ideal. Então basta somar com o CPI stall.

Questão: Um Tratador de exceções para o MIPS invoca procedimentos de até 3 parâmetros, que obedecem à convernção de chamada, utiliza pseudo-instruções implementadas pelo montador e usa os registradores $k0 e $k1. Exceto pelos registradores reservados para o "kernel" ($k0 e $k1), pelos registradores envolvidos na invocação e retorno de procedimentos e no registrador de suporte a pseudo-instruções ($at), o tratador não modifica quaisquer outros registradores. Sabe-se que o tratador preserva o conteúdo de todos os registradores que possam por ele ser modificados, exceto aqueles reservados para o "kernel", armazenando-os numa área estática de dados que se inicia no endereço 0x80010000.

Questão: Escreva uma sequência de código que armazena, a partir do endereço 0x80010000, todos os valores que precisam ser preservados pelo tratador. Restrição: Use a intrução add $k0, $zero, $at para preservar o valor de $at no registrador $k0 até o final do tratador.

Questão: É possível modificar o código do item anterior para preservar o conteúdo de $at na área estática de dados do "kernel", ao invés de preservá-lo no registrador $k0, liberando esse registrador para uso no próprio tratador. Mostre como modificar o código da questão anterior para liberar o registrador $k0. Dica: Em ambos os itens desta questão, você pode usar instruções nativas ou pseudo-instruções.

(a)

```
add $k0, $zero, $at
sw $a0, 0x8001 0000
sw $a1, 0x8001 0004
sw $a1, 0x8001 0004
sw $a2, 0x8001 0008
sw $ra, 0x8001 000C
```

(b)

```
add $k0, $zero, $at
sw $a0, 0x8001 0000
sw $a1, 0x8001 0004
sw $a2, 0x8001 0008
sw $ra, 0x8001 000C
sw $k0, 0x8001 0010
```

Questão: O processador Opteron usa endereços virtuais de 48 bits e endereços físicos de 40 bits e admite um tamanho de página de 4MB.

* Quantos bits são necessários para armazenar o número de um página física (real) na tabela de páginas?

Resposta: 18 bits. Cálculos: Página = 2^{22} bytes, page offset representado em 22 bits #páginas físicas: endereço físico - page offset = $40 - 22 = 18$ bits

Questão: Qual o tamanho da tabela de páginas expresso em megabytes (2^{28} bytes = 1 MB), considerando que cada linha da tabela ocupe exatamente 4 bytes? Resposta: 256 MB. Cálculos: #linhas / #páginas = $2^{48}/2^{22}$ Tamanho = $2^{26}x2^2 = 2^{28}$ bytes = 256 MB

1. Fazer AND do campo de pending interrupt e a máscara interrupção (mfc0) ? 2. Setar? máscara de interrupções pendentes. 3. Selecione a interrupção de maior prioridade. Select the higher priority of these interrupts. 4. Salvar a máscara

de interrupção. 5. Desabilitar interrupções menos prioritarias. 6. Salvar o contexto. 7. Chamar a rotina de tratamento apropriada. 8. Restaurar o campo de máscara. 9. Restaurar o contexto. 10. Retornar do tratador.

Para que o tratador de interrupções seja capaz de ser interrompido por interrupções de maior prioridade, o bit IE deve ser ativado (em nível 1) entre quais os passos acima enumerados? Não é preciso justificar. Resposta: Entre os passos 6 e 7.

Elaboration: The two least significant bits of the pending interrupt and interrupt mask fields are for software interrupts, which are lower priority. These are typically used by higher-priority interrupts to leave work for lower-priority interrupts to do once the immediate reason for the interrupt is handled. Once the higher-priority interrupt is finished, the lower-priority tasks will be noticed and handled.

Here are the steps that must occur in handling an interrupt: 1. Logically AND the pending interrupt field and the interrupt mask field to see which enabled interrupts could be the culprit. Copies are made of these two registers using the mfc0 instruction. 2. Select the higher priority of these interrupts. The software convention is that the leftmost is the highest priority. 3. Save the interrupt mask field of the Status register. 4. Change the interrupt mask field to disable all interrupts of equal or lower priority. 5. Save the processor state needed to handle the interrupt. 6. To allow higher-priority interrupts, set the interrupt enable bit of the Cause register to 1. 7. Call the appropriate interrupt routine. 8. Before restoring state, set the interrupt enable bit of the Cause register to 0. This allows you to restore the interrupt mask field.

Questão: Desenrolar o corpo do laço abaixo para expor duas iterações ... resultante para minimizar o número de ciclos necessários à correta ... Suponha que cada pipeline tenha 5 estágios e utilize todos os atalhos físico ... desvio condicional seja resolvido no estagio ID e que, no estágio WB ... primeiro semi ciclo enquanto, no estágio IDm a leitura dos registradores ... que a unidade de controle não possua um detector de hazards, sendo ... todos os nops necessários para preservar a semântica do código ... instruções pode ser emitida em um mesmo ciclo de relógio, mas pela ... emitida a cada ciclo, um nop deve ocupar a parte de um slot em que ... útil.

```
        add $s1, $zero, $zero
loop:   lw $t0, 100($s0)
        add $s1, $t0, $s1
        addi $s0, $s0, 4
        beq $s0, $s2, loop


        add $s1, $zero, $zero
loop:   lw $t0, 100($s0)
lw $t1, 104($s0)
        add $s1, $t0, $s1
add $s1, $t1, $s1 # podemos optimizar o código nessa questão?
        addi $s0, $s0, 8
        beq $s0, $s2, loop


        add $s1, $zero, $zero
```

```
loop:   lw $t0, 100($s0)
lw $t1, 104($s0)
        add $s1, $t0, $s1
        addi $s0, $s0, 8
        beq $s0, $s2, loop
```

Cenário único: Pipeline emite 2 instruções por ciclo; o desvio condição:

```
lw $t0, 100($s0)
lw $t1, 104($s0)
addi $s0, $s0, 8
o dado em $t0 demora a ficar pronto?
nop
add $s1, $t0, $s1
Alu Alu Forwarding $s1 -> $s1
nop
add $s1, $t1, $s1
nop
beq $s0, $s2, loop
nop
```

(Critério de correção: Cada erro desconta 0,5 ponto até que se anule a questão)

Questão: Cada um dos códigos abaixo .. diferente, cujos atalhos sao desconhecidos, mas sabe-se que todos os pipelines ... basica de 5 estágios (IF, ID, EX, MEM, WB) ilustrada no Anexo V. Sabe-se ... escrita de um registrador ocorre no primeiro semi ciclo do estágio WB e dois ... segundo semi ciclo do estágio ID. Sabe-se também que o número de nops inseridos ... necessário para que nenhuma pausa seja gerada pelo hardware durante a ... datapath, marque com um 'E' o(s) atalho(s) que garantidamente existe(m), e com '?' se nada puder ser garantido sobre... Deduza a existência ou inexistência só a partir do código e das informações ... diagrama de ocupação como auxílio à resolução, mas ele não será corrigido; só o ....

Datapath 1: Resposta ( ) ALU → ALU; ( ) ALU → MEM; ( ) MEM → ALU?

Questão: Suponha que uma instrução add causou "overflow" e chamou um tratador de exceções, do qual o código abaixo faz parte. Lembre que o registrador $14 é o EPC. Suponha que o $k1 contenha um valor (que corresponde à saturação em um dos limites da faixa de representação) a ser armazenado no registrador destino da instrução "add" que causou "overflow". Complete o código abaixo, de forma que o valor saturado armazenado em $k1 seja escrita no registrador-destino da instrução que causou a exceção. (Esta é uma oportunidade de demonstrar que você realmente participou do desenvolvimento do código durante o Lab 07).

(Critério de avaliação: Esta questão não será pontuada parcialmente)

```
            mfc0 $k0, $14
            lw   $k0, 0($k0)
            andi $t0, $k0, 0xF800
            lw   $k0, corretiva
            li   $t1, 0x FFFF 07FF
            and  $k0, $k0, $t1
            or   $k0, $k0, $t0
            sw   $k0, corretiva
corretiva: add  $t0, $zero, $k1
```

Questão: Um Tratador de exceções para o MIPS invoca procedimentos de até 3 parâmetros, que obedecem à convenção de chamada, utiliza pseudo-instruções implementadas pelo Montador e usar os registradores $k0 e k1. Exceto os registradores reservados para o "kernel" ($k0 e k1), pelos registradores envolvidos na invocação e retorno de procedimentos e no registrador de suporte a pseudo-instruções ($at), o tratador não modifica quaisquer outros registradores. Sabe-se que o tratador preserva o conteúdo de todos os registradores que possam por ele ser modificados, exceto aqueles reservados para o "kernel", armazenando-os numa área estática de dados que se inicia no endereço 0x80010000.

Questão: Escreva uma sequência de código que armazena, a partir do endereço 0x80010000, todos os valores que precisam ser preservados pelo tratador. Restrição: Use a instrução add $k0, $zero, $at para preservar o valor de $at no registrador $k0 até o final do tratador. É possível modificar o código do item anterior para preservar o conteúdo de $at na área estática de dados do "kernel", ao invéz de preservá-lo no registrador $k0, liberando esses registrador para uso no próprio tratador. Mostre como modificar o código da questão anterior para liberar o registrador $k0. Dica: em ambos os itens desta questao, você pode usar instruções nativas ou pseudo-instruções.

Questão: Dado o código abaixo, escrito em linguagem C, onde A[I][J] representa o elemento à linha i e à coluna j da matriz A. Suponha que as variáveis I, J e x sejam todas alocadas em registrador e não interferem na cache (D-cache). Cada elemento de A é um número inteiro representado em 32 bits e a matriz A é de 4×4. Suponha que a D-cache tenha 16 bytes por bloco, que o elemento A[0][0] foi armazenado no endereço 0 da MP e que a D-cache esteja vazia (todos os blocos inválidos) antes de se iniciar a execução desse código. Nestas condições, qual a sequência resultante de acertos (A) e falhas (F) a D-cache depois da execução desse código? Lembrete em C, os elementos de uma mesma linha da matriz são armazenados em ... (contíguos?) somente depois de armazenada uma linha completa, a próxima linha é armazenada em endereços contíguos e crescentes.

```
for (int i = 0; i < 4; i = i + 1)
  for (int j = 0; j < 4; j = j + 1)
    x = x + a[j][i]
```

Questão: Um sistema usa endereços virtuais de 32 bits e paginas de 4KB com os seguintes atributos V (valid bit), D (dirty bit) e R (reference bit). Quando o conteúdo do TLB é o abaixo mostrado, em ... física reside a variável cujo endereço virtual e 1010 0000 0000 0000 0000 1111 1111 1111. Dica: 1010 0000 0000 0000 0000 1111 1111 = 0x A000 0FFF.

v	D	R	TAG	Número
0	0	0	0xA0000	0x0001
0	0	0	0xC0000	0x7777
1	0	0	0xA0000	0xFFF0

Questão: Um subsistema de memoria consiste de uma cache primaria (L1), uma cache secundária (L2) e uma memória principal (MP). Seja TA o tempo de acesso. Sabe-se que TA(L1) = 1ns, TA(L2) = 10ns e TA(MP) = ? e que a taxa de faltas em L1 é de 2% e que a taxa global de faltas medida na cache equivalente ... Qual o %tempo medio (TM) para acessar um item em memória?

Questão: Ao ler a seção 5.3 do livro texto, você aprendeu a noção de tempo de acesso à memória (average memory access time) e como calculá-lo para uma hierarquia de 2 níveis; cache L1 e memória (Exercicio 5.7.4), você aprendeu a generalizar o AMAT para uma hierarquia de 3 níveis (L1) agora que você é capaz de generalizar esse cáculo para uma hierarquia de 4 níveis (L1, L2, Memória) AMAT para o sistema descrito na tabela abaixo. Não é preciso justificar.

L1	L2	L3	MP
1ns	5ns	20ns	100ns
10%	20%	30%	

AMAT = ns

Questão: Suponha que P1 e P2 realizem acessos de leitura e escrita em um arranjo de inteiros X ocupa exatamente um bloco em cache. Cada processador realiza acessos de leitura e escrita em diferentes ... bloco de sua cache privada que contém X. Suponha que o estado inicial desse bloco seja X[0] = X[1] = 1 em cada cache privada. Sejam os acessos ao arranjo X especificados no código abaixo:

P1	P2
X[0]++; X[1]=8;	X[0]=6; X[1]++;

(Refazer a questão, considerando uma falha no mecanismo de Write Throught onde ele não consegue invalidar blocos através desse mecanismo.)

Apresente um par de valores (X[0], X[1]), observável na cache privada do processador P2 imediatamente após a execução do código acima, que atestaria a implementação incorreta do protocolo de coerência. Considere que X[0]++ precede X[0] = 6; X[1] = 8 precede X[1]++.

Resposta: (,). Não é preciso justificar.

Nesse mesmo cenário, se o protocolo estiver correto, qual o valor do bit de validade do bloco do processador P1? Resposta: V = 0 (zero, invalidado). Não é preciso justificar.

Comentário: Uma vez que B escreveu X[1] = 8++, sendo esse 8 um valor anterior cujo Processador A salvou anteriormente.

E agora B está reescrevendo 8 + 1 = 9. Por esse motivo, por B estar salvando um novo valor em X[1] o valor na cache de A será invalidado (Valid Bit = 0).

Questão: A Tabela l mostra de forma simbólica, para alguns endereços de memória na faixa de 0x0 a 0x31C, o conteúdo da memória principal. As colunas em branco são campos auxiliares para facilitar a correspondência entre endereços hexadecimais e binários (seu preenchimento não será pontuado).

Referências à memória: 1ªref - 0xE4, 2ªref - 0x60, 3ªref - 0x1Fc, 4ªref - 0x3E8, 5ªref – 0xFC. Considere uma cache do tipo 2-way, inicialmente vazia, com 128 palavras, sendo que cada bloco contém 8 palavras. Preencha a Tabela 2 com o conteúdo final da cache imediatamente após aplicada a sequência de referências acima, usando os seguintes critérios e convenções: 1-Havendo 2 blocos livres num conjunto, o bloco trazido da memória deve ser armazenado no bloco livre de menor número (o preenchimento dos blocos na Tabela 2 deve ser da esquerda para a direita); 2-Havendo 1 bloco livre, nele deve ser armazenado o bloco trazido da memória; 3-Não havendo blocos livres, um dos blocos deve ser substituído de acordo com o critério LRU (dentre os dois blocos o último preenchido ainda é mais atual que o segundo, então o segundo será anulado e preenchido com novos dados, ou seja o segundo foi o menos usado recentemente MUR = LRU); 4-O conteúdo de cada bloco válido deve ser indicado explicitando todas as suas palavras.

Comentário: O endereço são os 7 bits mais a esquerda, isso quer dizer ele ele mapeia 128 palavras chave dentro da cache. Porém a estrutura → [tag—set—offset] nos diz que o bit 0 define o bloco 0, 1 em que será escolhido para ser gravado informação, os bits 1-3 definem em qual endereço dentro do bloco poderemos escrever informação 000 – 111 e os bits 4-6 definem quais dos 7 conjuntos escolher. Todo esses números juntos definem onde o dado será gravado. Agora. Quando ele acessa um endereço para (READ/WRITE operations) ele não trará para cache apenas o dado acessado (p.e.: 1 única palavra). Ele trás um bloco inteiro, ou trás 4 palavras contíguas, ou seja, trás grupos de bytes para serem alocados em cache, assim aumentando a localidade espacial. Todos os restos da divisão por 128, de todos os endereços acima, estão a seguir listados: [96, 100, 104, 108, 112, 116, 120, 124, 96, 100, 104, 108, 112, 116, 120, 124, 96, 100, 104, 108, 112, 116, 120, 124, 96, 100, 104, 108, 112, 116, 120, 124]. Só que você não precisa ficar quebrando a cabeça pensando em achar os restos. Basta selecionar os bits do endereço corretamente!

Questão: Qual o IPC de um GNU/ARM.

32G / 2G = 16 CPI = 1/2 = 1/IPC

Questão: IRQ_PRIORITY. Nessa questão você tem que dar prioridade para D i+1.

Questão: In computer science, load-link and store-conditional (LL/SC) are a pair of instructions used in multithreading to achieve synchronization. Load-link returns the current value of a memory location, while a subsequent store-conditional to the same memory location will store a new value only if no updates have occurred to that location since the load-link. Together, this implements a lock-free atomic read-modify-write operation.