# 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 = $(0,25 \times I + I) = 1250000$