# Questões de Operações

Lucas

25 de agosto de 2015

## 1 Operações, alguns comentários

- O princípio desse capítulo é: "relationship between high-level programming languages and this more primitive one."(página 76 - Organização de Computadores 4ª). Relacionamento entre programação de alto nível e sua linguagem mais primitiva (ou à nível mais primitivo).
- **stored-program concept**: The idea that instructions and data of many types can be stored in memory as numbers, leading to the stored-program computer. (Conceito de programa-armazenado, é a ideia que instruções e dados de muitos tipos podem ser armazenados em memória como valores).
- Na página 78 existem um quadro com as instruções da arquitetura mips, desse quadro podemos retirar alguns conceitos irados, de que $2^{30}$ palavras de memória são a sequência `Memory[0]`, `Memory[4]`, ..., `Memory[4294967292]`. `4294967292 bytes` são **4 GiB**.
- No final da tabela tem os jumps, j 2500, e go to 10000. Esse 2500 é em hexadecimal $(2500)_{base16}$ << 2 == $(10000)_{base10}$ == $(10011100010000)_{base2}$.
- Com o **nor** $s1 = ($s2 — $s3), você pode trabalhar ele zerando $s3 ou simplesmente colocar $zero no lugar, para negar $s2 e "armazenar no registrador $s1".
- O load linked: ll $s1,20($s2) e o Store Conditional: sc $s1,20($s2) são bastante importantes para se entender. Pois ambos trabalham 1st and 2nd half of atomic swap. O load linked carrega em $s1 = Memória[$s2 + Deslocamento] e o score conditional faz o seguinte Memória[$s2 + Deslocamento]=$s1;$s1=0 or 1. Isso quer dizer que o Load Linked pegou o que tinha em $s1 e armazenou na pilha algum valor e acionou o 1st half of atomic swap, engatilhou uma ação atômica, isto é, se ninguém armazenar nada no mesmo endereço de [$s2 + Deslocamento] até que o próximo store conditional complete a ação 2nd half of atomic swap, então o store conditional irá armazenar seja lá o que ele tiver em $s1 na posição Memória[$2 + Deslocamento] nisso ele sinaliza seu próprio registrador $s1 com o valor '1' (dizendo: "eu completei atomic exchange"); **caso contrário** se alguém mexeu no mesmo endereço de memória antes do store conditional (p.e: outro store conditional, store word, store half, etc..) então ele não vai mais completar o seu 2nd half of atomic swap o resultado disso é que ele sinaliza o próprio registrador $s1 como sendo 0 (dizendo: "eu não completei minha atomic exchange");

- Na página 84 diz que se deve cuidar com esquema o de alinhamento 4 por 4 que aparece no MIPS, e em outras arquiteturas parecidas com o MIPS, ou seja, um endereço apontando para uma instrução deve ser multiplicado por 4, para se ter certeza que você esta pegando um endereço válido nos termos naturais de alinhamento de instruções 4 por 4 bytes. Na página 84 do livro-texto de Organização de Computadores diz: "Computers divide into those that use the address of the leftmost or "big end" byte as the word address versus those that use the rightmost or "little end" byte. MIPS is in the big-endian camp.". Você pode testar essa afirmação dele no MARS, Simulador de MIPS. No apêndice

```
lui         $t0, 0x1DE              # $t0 = 0x01DE0000
ori         $t0, $t0, 0xCADE        # $t0 = 0x01DECADE
lui         $t1, 0x1001             # $t1 = 0x10010000
sb          $t0, 200 ($t1)          # $t1 + 200 bytes = 0x01DECADE
lb          $t2, 200 ($t1)          # $t2 = 0x01DECADE
```

  Se o byte 'DE' for para memória na posição do byte menos significativo, o MARS é little endian, quando reordenando na memória. Se você carregasse a palavra inteira 01DECADE com 'sw' a palavra apareceria nessa mesma ordem em memória.

- Quando você utilizar load byte signed (lb) para armazenar alguma coisa em registrador, saiba que você esta sujeito a estender o sinal (só isso). Por exemplo:

```
M($t0+20)=0x000000de
lb $s1, 0($t0+20)
$s1 <= 0xffffffde
```

- Portando te você precisar puxar um **dado**, use lbu, para ter certeza que seu byte não vai estragar o resto dos dados. Se você sabe que byte que você esta puxando é "positivo"(pe: 0x2), não vai causar grande estrago.

**2.1.1**

**2.2.1**

**2.3.1**

**2.2.4**

**2.3.4**

**2.4.1**

**2.4.4**

**2.5.4**

**2.5.5**

**2.11.5**

**2.12.1**

**2.12.2**