u évora
ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Arquitectura de Sistemas e Computadores I

Exercícios

Licenciatura em Engenharia Informática

23 de Maio de 2013

Aula Prática #1 (Revisões)

As alíneas seguintes destinam-se a rever a representação de números nas bases binária e hexadecimal. Para cada questão, considere registos de 8 e 32 bits (i.e. todos os números têm 8 ou 32 bits). Apresente todos os resultados nestas duas bases (binária e hexadecimal).

1. Converta para binário e hexadecimal os seguintes números:
   (a) 123
   (b) -12
2. Qual o maior e o menor número que pode ser representado em complemento para 2 em registos de 8 bits e 32 bits?
3. Qual o maior e o menor número que pode ser representado sem sinal em registos de 8 e 32 bits?
4. Efectue as seguintes somas em binário (considere registos de 8 bits):
   (a) $3 + 7$
   (b) $3 + (-7)$
5. Assumindo registos de 8 bits, explique qual o resultado das seguintes operações:
   (a) $Y = X$ AND 00000001
   (b) $Z = X$ ShiftRight 4 bits
   (c) $Y =$ NOT X; $Y = Y + 1$
6. Use as operações lógicas AND, OR, XOR, NOT, Shift Left e Shift Right, para a partir de um registo X de 8 bits:
   (a) Colocar o bit mais significativo a 1.
   (b) Colocar o bit mais significativo a 0.
   (c) Negar o valor do bit 5.
   (d) Copiar os bits 0 a 2 para Y, colocando os restantes bits a zero (proponha duas soluções: uma só com shifts e outro sem shifts).
   (e) Copiar os bits 3 a 5 de um registo para outro, colocando os restantes bits a zero.
   (f) Trocar a posição dos 4 bits mais significativos de um registo de 8 bits com os 4 menos significativos do mesmo registo.
7. Pretende-se escrever o número 256 numa *word* (32 bits) em memória. Assumindo que a ordenação dos bytes é *Little Endian* e que o endereço é `0x10008000`, represente cada um dos pares endereço/byte.
8. Pretende-se converter uma *word* em memória de *Little Endian* para *Big Endian* (ou vice-versa). O procedimento consiste em ler a *word* para o processador, fazer a conversão em registos do processador, e finalmente guardar o resultado de volta em memória. Escreva um conjuntos de instruções (como as instruções lógicas usadas nas alíneas anteriores) de modo a efectuar esta conversão.
9. Usando uma sequência de XORs, efectue a troca dos valores dos dois registos X e Y sem usar nenhum registo adicional.

Aula Prática #2 (Assembly)

Nesta aula prática irá fazer programas muito simples para avaliar expressões aritméticas. Será apresentado o simulador MARS: código fonte, instruções assembly geradas, código máquina, endereços de memória e registos do processador.

1. Usando as instruções `add`, `addi`, `sll` e `sub`, escreva programas em assembly para as seguintes expressões:
   (a) $t0 = 3 + 4$. (use instruções para colocar valores nos registos)
   (b) Supondo ordenação de bytes little endian, indique (desenhando o mapa de memória) qual o conteúdo de cada endereço de memória:

Address	Memory
⋮	⋮
0x00400002	??
0x00400001	??
0x00400000	??

   (c) $t0 = 3 + 4 + 5$. (use instruções para colocar valores nos registos)
   (d) Supondo ordenação de bytes big endian, indique (desenhando o mapa de memória) qual o conteúdo de cada endereço de memória. Qual o byte no endereço 0x00400011? e no endereço 0x0040000b?
   (e) $t1 = 2 \times t0 + 1$.
   (f) $t1 = 9 \times t0 + 5$.
   (g) $t1 = t0 - 1$. (haverá necessidade de uma instrução `subi`?)
   (h) $t2 = 2 \times t0 + 3 \times (t0 - t1)$.

Aula Prática #2A (Assembly)

1. Escreva um troço de código (em papel) para calcular o simétrico de um número guardado no registo `$t0`. O resultado deve ficar em `$t1`.

2. Sem usar o simulador, analise o seguinte troço de código e indique o valor final dos registos `$t0`, `$t1` e `$t2`.

```
lui $t0, 0x1234
ori $t0, $t0, 0x5678
ori $t1, $zero, 0x1234
sll $t1, $t1, 16
ori $t1, $t1, 0x5677
sub $t2, $t1, $t0
```

3. Sem usar o simulador, analise o seguinte troço de código e indique os valores finais de `$t0` e `$t1`.

```
lui $t0, 0x0001
ori $t0, $t0, 0xffff
lui $t1, 0x0002
ori $t1, $t1, 0x0ff7

or $t2, $zero, $t0
or $t0, $t1, $zero
or $t1, $t2, $t2
```

4. Supondo que `$t0=0x00107fff` e `$t1=0x80000000`, determine o resultado das seguintes operações:
   (a) `sra $t2, $t0, 4`
   (b) `srl $t2, $t0, 2`

(c) `sra $t2, $t1, 4`

(d) `srl $t2, $t1, 2`

5. Suponha que o programa da alínea 2 está carregado em memória no endereço `0x00400000` e que os registos do processador têm os valores `$t0=0xffffffff`, `$t1=0x80000000` e `PC = 0x00400010`. Qual o conteúdo dos registos todos (`$t0`, `$t1`, `$t2` e `PC`) após a execução de duas instruções.

Aula Prática #3 (Branches)

Nesta aula serão desenvolvidos pequenos troços de código assembly para implementar execuções condicionais IF/THEN/ELSE e ciclos FOR/WHILE. São dadas as instruções `slt` e `slti`. Estas instruções serão usadas conjuntamente com branches `beq` e `bne`. É necessário ter em consideração os *delayed branches*.

1. Assumindo que os valores das variáveis x e y estão nos registos $t0$ e $t1$, respectivamente, implemente em assembly MIPS cada um dos troços de código seguintes:

(a)
```
//-------------
if (x == y)
    x = 0;
else
    x = y;
```

(b)
```
//-------------
if (x < 0)
    y = -x;
else
    y = x;
```

(c)
```
//-------------
y = 0;
for (x=1; x<=10; x++)
    y += x;
```

2. Considere o pedaço de código seguinte:

```
A:  slt $t2, $t1, $zero
    bne $t2, $zero, B
    nop
    sub $t1, $zero, $t1
    beq $zero, $zero, A
    nop
B:
```

(a) Identifique o que faz.

(b) O que acontece se `$t1` for inicialmente zero? (corrija)

3. Considere o código assembly seguinte:

```
    xor  $t1, $t1, $t1
    ori  $t0, $zero, 1
R:  slti $t2, $t0, 0xb
    beq  $t2, $zero, SAIR
    nop
    add $t1, $t1, $t0
    addi $t0, $t0, 1
    beq $zero, $zero, R
    nop
SAIR:
```

(a) O que faz este código?

(b) Supondo que a frequência de relógio é $f = 500$ MHz, e assumindo 1 ciclo de relógio por instrução, quanto tempo leva a execução? (não se esqueça que as instruções do ciclo são executadas várias vezes)

(c) Optimize a velocidade de execução e calcule o *speedup*.

Aula Prática #4 (Jumps, Loads e Stores, Arrays)

Nesta aula iremos desenvolver rotinas para percorrer e modificar arrays.

1. Considere o seguinte "template" para uma rotina a desenvolver:

```
        .data
        .word 3,1,-2,0,3,-10,-1,3

        .text
main:
        lui $t0, 0x????
        ori $t0, $t0, 0x???? # coloca em t0 o endereco do primeiro elemento do array

        #
        # insert your code here
        #
```

   (a) Escreva uma rotina que converta os números do array de negativos para positivos.

   (b) Modifique a rotina anterior para, além de converter para positivos, também contar quantos negativos tem o array.

2. Escreva uma rotina para calcular o comprimento de uma string (sem contar com o carácter nulo). Admita que o seu endereço está no registo `$t0`.

3. Escreva uma rotina que converta uma string arbitrária para maiúsculas. (Atenção que apenas as letras minúsculas devem ser convertidas, os números e símbolos devem ser preservados)

Aula Prática #5 (Pseudoinstruções, Funções, Stack)

1. **Sem usar o simulador Mars**, converta as pseudoinstruções seguintes em instruções reais MIPS:

   (a) `bge $t0, $t1, L`

   (b) `ble $t0, 5, L`

   (c) `li $a0, 0x7ffffffc`

   (d) `lw $t0, 0x10010014`

2. Para as questões seguintes, escreva uma função `main` que prepare os argumentos e chame cada uma das funções.

   (a) Implemente a função $\text{exp2}(y) \triangleq 2^y$. Assuma $y \geq 0$.

   (b) Implemente a função $\text{fact}(y) \triangleq 1 \times 2 \times 3 \times \cdots \times y = \prod_{n=1}^{y} n$.

   (c) Implemente a função $\log2(y)$ que obtém a parte inteira do logaritmo na base 2 de y.

   (d) Implemente a função $f(y) \triangleq \log2(\text{exp2}(y)) + \text{fact}(y)$.

Aula Prática #6 (Funções recursivas)

1. Implemente a função `factorial` usando recursão.

Aula Prática #7 (Funções Arrays e Strings)

1. As funções seguintes são funções existentes na biblioteca `libc` (biblioteca de funções disponíveis em programas escritos em C). Pode consultar informação com o comando '`man nome_da_funcao`' em Linux. Escreva cada uma destas funções em assembly MIPS.
   (a) `strlen`
   (b) `strcpy`
   (c) `strncpy`
   (d) `strcat`
   (e) `strcmp`
   (f) `strncmp`
   (g) `strtok`
2. Escreva funções para operarem sobre arrays de inteiros:
   (a) `min(a,n)` - dado um array `a` de tamanho `n`, calcula o valor mínimo presente no array.
   (b) `sum(a,n)` - dado um array `a` de tamanho `n`, calcula a soma dos elementos.
   (c) `inverte(a,n)` - dado um array `a` de tamanho `n`, inverte a ordem dos elementos.

Aula Prática #8 (Código máquina)

1. Converta a função `factorial` (da aula #6) para código máquina. Confira o resultado com o produzido pelo simulador MARS.
2. Usando as instruções de branch – `beq` ou `bne` – será possível saltar para qualquer outra instrução de um programa? Qual a gama de instruções acessíveis por um branch?
3. Considerando agora as instruções de salto – `j` ou `jal` – responda novamente à pergunta anterior.
4. O registo `$gp` contém o endereço `0x10008000` e mantém-se constante ao longo da execução de um programa. Este registo é usado para aceder a uma certa região de memória com as instruções de load e store. Qual a gama de endereços acessíveis com instruções do tipo `lw $t0, offset($gp)`, onde `$gp` é fixo e `offset` é livre? Que região de memória é essa? (text/global/dynamic/stack)
5. Sabe-se que uma função está carregada em memória no endereço `0x004010a0` com ordenação de bytes little endian. Fazendo um *dump* da memória a partir do endereço da função, obteve-se a sequência de bytes abaixo. O que faz esta função?

```
06 00 a0 10 25 10 00 00 00 00 88 8c 04 00 84 20 ff ff a5 20 fc ff a0 14 20 10 48 00
08 00 e0 03 00 00 00 00
```

Aula Prática #10 (Vírgula flutuante)

1. Represente os números seguintes em vírgula flutuante IEEE754 em precisão simples e dupla:
   (a) 1.0
   (b) 0.785
   (c) -Inf
   (d) NaN
   (e) -0.0
   (f) 13.75×10^1
2. Converta os números seguintes para decimal:
   (a) `0x7ff8000000000000`
   (b) `0x4031e00000000000`

(c) `0x8000000000000000`

(d) `0x4046080000000000`

3. Escreva um programa em assembly que faça os dois cálculos seguintes em vírgula flutuante, precisão dupla:

   - $f6 = (0.1 + 0.2) + 0.3$
   - $f8 = 0.1 + (0.2 + 0.3)$

   Compare os resultados obtidos.

   (Sugestão: Use as instruções `lwc1`, `mtc1` e a directiva `.double`)