GBC046 – Arquitetura e Organização de Computadores II
Prof. Dr. rer. nat. Daniel Duarte Abdala

# Lista de Revisão para a Primeira Avaliação

1. Explique a diferença entre as instruções **`add`** e **`addi`**. Qual formato de instrução cada uma delas utiliza?
2. Quantos bits a instrução **`j`** reserva para o endereço do jump? Qual o formato da instrução?
3. Para que servem as instruções **`jal`** e **`jr`**? Em que contexto elas são indispensáveis?
4. Descreva a utilização dada para os seguintes registradores:

$s0 - $s7: _____

$t0 - $t7: _____

$pc: _____

$sp: _____

$ra: _____

$gp: _____

$fp: _____

$zero: _____

$a0-a3: _____

$v0-v1: _____

$hi: _____

$lo: _____

5. Converta as parcelas de código C abaixo para assembly do MIPS3000. Especifique quais variáveis serão atribuidas a quais registradores.

```
long a, b, c, d, e, f, g, h;

a) a = 0;

b) b = 42;

c) a = (b+c) / (c-d);

d) a = (a+b+c+d)/(e-f-g-h);

e) a = a/b + c/d + e*f - g*h;

f) a = 42 + b - (54 * a);
```

```
unsignedlong a, b, c, d, e, f, g, h;

g) a=((b+c)*(d-e)/(f*g*h))/a+b+c+d+e;

h) a = 0xFFFFFFFA;

i) a = (a + b + d + e) / (b*b*b);

j) h = g*i + h*h - c/c;
```

```
k) a = (~b & c) | (b & ~c);

l) a = b >> 2;

m) c = ~(d << 4);

n) a = a^b^c^d;
```

6. Descreva o que significa os acronismos que definem as seguintes instruções:

sll :_____

srl :_____

slt :_____

slti :_____

beq :_____

bne :_____

beqz :_____

bgtz :_____

blez :_____

bltz :_____

j :_____

jal :_____

jr :_____

7. Converta as parcelas de código C abaixo para assembly do MIPS3000.

```
if (c < 42) {...}
```

```
if (c < d) {...}
```

```
if ((c >=0) && (c <= 42) ) {...}
```

```
if (a > 0) {...} else {...}
```

```
if (a < 0) {...}
else if (a < - 42) {...} else {...}
```

```
long c = 0;
while (c < 42)
{
c++;
}
```

```
for (long c = 0; c < 42; c++) {...}
```

```
for (long c = 42; c >=0; --c) {...}
```

```
switch(expression){
case 42 :
       ...
break;
case 31415:
       ...
break;
default :
       ...
}
```

8. Explique a diferença entre uma instrução nativa e uma pseudo-instrução. Dê exemplos.
9. Para que servem as instruções LI e LA. Forneça exemplos de suas utilizações.
10. Converta os programas abaixo para sub-rotinas em assembly. Considere que os registradores utilizados que não forem temporários devem ser salvos. O mesmo vale para o $ra, $a0-$a3 e $v0-$v1. Explique o que as funções fazem.

```
longm(long a, long b){
longi, r = 0;
for(i=a; i>0; i--){
r += b;
}
return r;
}
```

```
int e(intb, intexp){
int i, r = b;
for(i=exp; i>1; i--){
r = m(r,b);
}
return r;
}
```

11. Sabemos que a instrução **j** utiliza o tipo de instrução **Type-J**, que reserva 6 bits para o opcode e os 26 bits restantes para o endereço de memória para o qual deve ser executado o salto. Qual a faixa de endereços endereçáveis utilizando a instrução **j**? Caso seja necessário saltar para uma posição de memória que esteja fora do limite imposto pelos 26 bits de **j,** o que poderíamos fazer? Considere que desejamos saltar para uma posição na faixa 0x04000000 – 0xFFFFFFFF.
12. Considere o programa abaixo. Ele multiplica uma série de 4 números (a1xb1, a2xb2, a3xb3 e a4xb4) por fim, soma os resultados das multiplicações e então imprime seu resultado. No exercício anterior fizemos um algoritmo de multiplicação através de somas sucessivas.

```
void main(){
long i, r1,r2,r3,r4;
longa1=1,a2=3,a3=5,a4=7;
longb1=2,b2=4,b3=8,b4=16;

r1=r2=r3=r4 = 0;
  //----secao----//
for(i=a1; i>0; i--){
r1 += b1;}
for(i=a2; i>0; i--){
r2 += b2;}
for(i=a3; i>0; i--){
r3 += b3;}
for(i=a4; i>0; i--){
r4 += b4;}
//------------//
int r = r1+r2+r3+r4;
printf("%d\n", r);
}
```

a) Converta o programa acima para assembly plano sem a utilização de subrotinas e conte quantas operações são necessárias para a sua execução.
b) Substitua a parte do programa salientada pelos comentários "seção" por quatro chamadas à subrotina "m" do exercício 5 e então conte quantas operações serão necessárias para a execução do programa.
c) Que conclusões você tira da diferença de números de instruções a serem executadas nos dois casos? Quando esta diferença é justificada?

13. Como converteríamos as seguintes estruturas de C para assembly?

```
i=0;
while(i<N){
//faça algo útil
i = i+1;}
```

```
for(int i =0; i< N; i++){
//faça algo útil
}
```

14. Escreva um programa que leia 10 inteiros longos não sinalizados da entrada padrão e some apenas os números impares. O teste para números impares deve ser implementado em uma função separada do programa principal. Por fim, imprima a soma de todos os números impares.
15. Desenvolva um programa que leia 42 inteiros longos não sinalizados (32 bits unsigned long) e armazene os valores lidos em um array de 42 inteiros inicializado inicialmente com zeros em todas as posições. A seguir o programa deve chamar três funções, uma que calcula a soma de todos os elementos do array, outra que retorne o menor valor e outra que retorne o maior valor. Imprima todos os três resultados das funções na saída padrão. Salve e restaure da pilha todas as variáveis pertinentes (Housekeeping).
16. Escreva uma função recursiva que calcule o fatorial de um número inteiro longo não sinalizado. A função deve salvar na pilha todos os registradores pertinentes utilizados na função.

17. Quantas palavras de 32 bits seriam salvas na pilha para na execução de FAT(10) implementado na questão anterior?
18. Escreva um programa recursivo que calcule o valor do elemento N da série de Fibonacci.

```
Fibo = 1 1 2 3 5 8 13 21 34 55 89, 144, •••
Fibo(N) = Fibo(N-1) + Fibo(N-2)
```

19. Forneça o código equivalente das pseudo-instruções a seguir utilizando apenas instruções reais.

```
li$s0,-100
beq  $s0, 42, LABEL
move $s0, $s1
sw   $s0, $s1($s2)
```

20. Comente o programa abaixo e explique o que ele faz.

```
.data
str1:.asciiz "digite a string\n"
size_str2:.word 30
str2:.space 30

.text
li    $v0,4
la    $a0,str1
syscall
li    $v0,8
la    $a0,str2
li    $a1,30
syscall
addi  $s7, $zero, 0x0a
la    $t0, str2
L:lb    $s0, 0($t0)
beq   $s0,$s7, END
addi  $s0, $s0, -32
sb    $s0, 0($t0)
addi$t0, $t0,1
jL
li$v0,4
la$a0,str2
syscall
li$v0, 10
syscall
```

21. Desenhe um diagrama que mostra como o dado `0x45ffabcd` localizado na posição de memória `0x10000048` armazenado no registrador `$s7` é carregado no registrador `$s0`.

```
lw $s0,0($s7)
```

22. Desenvolva um programa que leia um inteiro **N** e teste se este inteiro é maior que zero e menor que 1000. Aloque **N words** dinamicamente na memória. A seguir leia N inteiros e armazene-os na memória alocada dinâmicamente previamente. A seguir, o programa deve chamar uma função, que calcula ocubo de todos os elementos do array e substitua o valor previamente armazenado. Por fim a função deve imprimir os cubos dos números lidos. Salve e restaure da pilha todos os registradores pertinentes. Comente seu código.
23. O que seria necessário para que o processador no anexo fosse capaz de executar a instrução **BGEZ**? Especifique as alterações de multiplexação e indique que subsistemas deverão ser alterados.
24. Com relação a Unidade de Controle da implementação Monociclo, temos basicamente o problema de criar um circuito decodificador que converte 12 bits de entrada (opcode – 6 bits e funct - 6 bits) nos sinais de controle que habilitarão o caminho de dados correto para cada instrução. Utilize o circuito do processador em anexo e preencha a tabela abaixo.

| Instruction | op[5,4,3,2,1,0] | fc[5,4,3,2,1,0] | RegDst | Branch | MemRd | MemWr | M2R | AluSrc | RegWr | Jump |
|---|---|---|---|---|---|---|---|---|---|---|
| add | 000000 | 100000 | | | | | | | | |
| sub | 000000 | 100010 | | | | | | | | |
| and | 000000 | 100100 | | | | | | | | |
| or | 000000 | 100101 | | | | | | | | |
| nor | 000000 | 100111 | | | | | | | | |
| xor | 000000 | 100110 | | | | | | | | |
| slt | 000000 | 101010 | | | | | | | | |
| addi | 001000 | ---------- | | | | | | | | |
| andi | 001100 | ---------- | | | | | | | | |
| ori | 001101 | ---------- | | | | | | | | |
| xori | 001110 | ---------- | | | | | | | | |
| sll | 000000 | 000000 | | | | | | | | |
| srl | 000000 | 000010 | | | | | | | | |
| beq | 000100 | ---------- | | | | | | | | |
| j | 000010 | ---------- | | | | | | | | |
| lw | 100011 | ---------- | | | | | | | | |
| sw | 101011 | ---------- | | | | | | | | |

25. O que seria necessário para que o processador no anexo fosse capaz de executar a instrução **jr**(considere que a sintaxe de jr seja: `100001 ssssssxxxxxxxxx 00000 xxxxxx` e que o registrador que contêm o endereço para o salto seja jr $rs)? Especifique as alterações de multiplexação e indique que subsistemas deverão ser alterados. Desenhe as alterações necessárias no diagrama do processador em anexo.
26. Consider that the processor presented in the appendix is able to execute the instruction mul $rd, $rs, $rt. In order to retrieve the upper part (MSB 32bits) of the multiplication located in the register $HI the instruction mfHI is required. According to the table of registers presented in the appendix, $HI is an external register not accessible as a regular register. Propose the extensions necessary in order to implement the instruction mfHI $rs.
27. Desenvolva um programa capaz de iterar sobre um array de 21 posições, atribuindo o valor 42 a primeira posição, 42*2 a segunda, 43*3 a terceira posição do array e assim sucessivamente. Identifique o formato de cada instrução do programa. A seguir converta o programa resultante para código de máquina. Por fim, forneça o código de máquina em hexadecimal. Assuma que o programa começa na posição 0x0000 em hexa para fins de resolução de ETIQUETAS (LABELS).

**TABELA 1 - CHAMADAS DO SISTEMA**

| $V0 | DESCRIÇÃO | ARGUMENTOS | VALOR DE RETORNO |
|---|---|---|---|
| 1 | imprime inteiro | $a0-int para impressão | |
| 4 | imprime string | $a0-end do 1º byte da string | |
| 5 | lê inteiro | $v0-int lido | |
| 8 | lê string | $a0-end da string / $a1-num de bytes | |
| 9 | aloca mem. na heap | $a0-num de bytes / $v0 -end do 1º byte | |
| 10 | termina execução | retorna o controle ao SO | |
| 11 | imprime caractere | $a-char para impressão | |
| 12 | lê caractere | $v0-char lido | |

**TABELA 2 - REGISTRADORES**

| # | NOME | FUNÇÃO |
|---|---|---|
| 00 | $zero | valor fixo em zero (0) |
| 01 | $at | assemblertemporary (reservado para o assembler) |
| 02 | $v0 | valor de retorno de subrotinas |
| 03 | $v1 | valor de retorno de subrotinas |
| 04 | $a0 | quatro primeiros argumentos de funções |
| 05 | $a1 | |
| 06 | $a2 | |
| 07 | $a3 | |
| 08 | $t0 | temporários<br>funções podem utiliza-los sem salvar |
| 09 | $t1 | |
| 10 | $t2 | |
| 11 | $t3 | |
| 12 | $t4 | |
| 13 | $t5 | |
| 14 | $t6 | |
| 15 | $t7 | |
| 16 | $s0 | s - subrotina<br>devem ser preservados se usados em subrotinas |
| 17 | $s1 | |
| 18 | $s2 | |
| 19 | $s3 | |
| 20 | $s4 | |
| 21 | $s5 | |
| 22 | $s6 | |
| 23 | $s7 | |
| 24 | $t8 | temporários |
| 25 | $t9 | |
| 26 | $k0 | usados por interrupções / traps |
| 27 | $k1 | |
| 28 | $gp | global pointer |
| 29 | $sp | stack pointer |
| 30 | $fp | frame pointer |
| 31 | $ra | return address |
| | PC | program counter |
| | LO | low - 32 bits menos sig.(mult),quociente(div) |
| | HI | high - 32 bits mais sig.(mult), resto(div) |

# Diagrama do Processador Estudado