

Problema: converta a seguinte função para assembly do MIPS, respeitando as convenções de utilização de registos.

```
void madd(unsigned int n,unsigned int *a,unsigned int *b,unsigned int *c)
{ // c += a * b
  void normalize(unsigned int n,unsigned int *a);
  unsigned int i,j,t = 0;

  if(n == 0)
    print_string("warning: zero digits!"); // syscall
  for(i = 0;i < n;i++)
    if(a[i] != 0)
    {
      for(j = 0;j < n;j++)
        c[i + j] += a[i] * b[j];
      t++;
      if((t & 15) == 0)
        normalize(2 * n,c);
    }
  if((t & 15) != 0)
    normalize(2 * n,c);
}
```

```
n: $a0 -> $s0
a: $a1 -> $s1
b: $a2 -> $s2
c: $a3 -> $s3
i:      $s4
j:      $t0
t:      $s5
```

Solução possível:

```
.data
str: .asciiz "warning: zero digits!"

.text
.globl madd
madd: subiu $sp,$sp,28
      sw $ra,0($sp)
      sw $s0,4($sp)
      sw $s1,8($sp)
      sw $s2,12($sp)
      sw $s3,16($sp)
      sw $s4,20($sp)
      sw $s5,24($sp)
      move $s0,$a0
      move $s1,$a1
      move $s2,$a2
      move $s3,$a3
      li $s5,0
if1: bne $s0,$zero,end1
     la $a0,str
     li $v0,4
     syscall
end1:
for2: li $s4,0
test2: bgeu $s4,$s0,end2
if3: sll $t1,$s4,2 # 4*i
     addu $t1,$s1,$t1 # &a[i]
     lw $t1,0($t1) # a[i]
     beq $t1,$zero,end3
for4: li $t0,0
test4: bgeu $t0,$s0,end4
      sll $t2,$t0,2 # 4*j
      addu $t2,$s2,$t2 # &b[j]
      lw $t2,0($t2) # b[j]
      mul $t3,$t1,$t2 # a[i]*b[j]
      addu $t4,$s4,$t0 # i+j
      sll $t4,$t4,2 # 4*(i+j)
      addu $t4,$s3,$t4 # &c[i+j]
      lw $t5,0($t4) # c[i+j]
      addu $t5,$t5,$t3 # +
      sw $t5,0($t4) # +=

next4: addiu $t0,$t0,1 # j++
      j test4
end4: addiu $s5,$s5,1 # t++
if5: andi $t1,$s5,15
     bne $t1,$zero,end5
     sll $a0,$s0,1 # 2*n
     move $a1,$s3 # c
     jal normalize
end5:
end3:
next2: addiu $s4,$s4,1
      j test2
end2:
if6: andi $t1,$s5,15
     bne $t1,$zero,end6
     sll $a0,$s0,1 # 2*n
     move $a1,$s3 # c
     jal normalize
end6: lw $ra,0($sp)
      lw $s0,4($sp)
      lw $s1,8($sp)
      lw $s2,12($sp)
      lw $s3,16($sp)
      lw $s4,20($sp)
      lw $s5,24($sp)
      addiu $sp,$sp,28
      jr $ra
```

Problema: converta a seguinte função para assembly do MIPS, respeitando as convenções de utilização de registros.

```
int main(void)
{
    int n,m,t;

    n = read_int();      // syscall
    m = n;
    for(t = n; t > 1; t >>= 1)
    {
        print_int10(t);  // syscall
        print_string("\n"); // syscall
        if((t & 1) != 0)
        {
            t = 3 * t + 1;
            if(t > m)
                m = t;
        }
    }
    print_string("max:"); // syscall
    print_int10(m)        // syscall;
    print_string("\n")    // syscall;
}                         n: $t0
                          m: $t1
                          t: $t2
```

Solução possível:

```
.data
str1: .asciiz "\n"
str2: .asciiz "max:"

.text
.globl main
main:  li      $v0,5
      syscall      # read_int
      move      $t0,$v0
      move      $t1,$t0
for1:  move      $t2,$t0
test1: ble      $t2,1,endi
      li      $a0,t2
      li      $v0,1
      syscall      # print_int10
      la      $a0,str1
      li      $v0,4
      syscall      # print_string
if2:   andi      $t3,$t2,1
      beq      $t3,0,endi
      mul      $t2,$t2,3
      addi     $t2,$t2,1
if3:   ble      $t2,$t1,endi
      move     $t1,$t2
endi3:
endi2:
next1: sra      $t2,$t2,1
      j      test1
endi1: la      $a0,str2
      li      $v0,4
      syscall      # print_string
      li      $a0,t1
      li      $v0,1
      syscall      # print_int10
      la      $a0,str1
      li      $v0,4
      syscall      # print_string
      jr      $ra
```

Problema: converta a seguinte função para assembly do MIPS, respeitando as convenções de utilização de registros.

```
int xpto(int n,int *d)
{
    int i,s;

    s = 0;
    for(i = 0;i < n;i++)
    {
        if(d[0] == 0 || (i + 1 < n && d[1] == 0))
            s += 100;
        else
        {
            s--;
            (*d)++;
        }
        d++;
    }
    return s;
}
```

n: \$a0
d: \$a1
i: \$t0
s: \$t1

[Note que *d++ faz um acesso à memória e depois incrementa o ponteiro, enquanto que (*d)++ incrementa o que está na memória.]

Solução possível:

```
.text
.globl xpto
xpto:  li      $t1,0
for1:  li      $t0,0
test1: bge     $t0,$a0,end1
if2:   lw      $t2,0($a1)      # d[0]
      beq     $t2,$zero,then2
      addi    $t3,$t0,1
      bge     $t3,$a0,else2
      lw      $t3,4($a1)
      bne     $t3,$zero,else2
then2: addi    $t1,$t1,100      # s += 100
      j      end2
else2: subi    $t1,$t1,1        # s--
      addi    $t2,$t2,1        # d[0]+1
      sw      $t2,0($a1)      # (*d)++
end2:  addiu   $a1,$a1,4
next1: addi    $t0,$a0,1
      j      test1
end1:  move    $v0,$t1
      jr      $ra
```