

Nº Mec.: _____ Nome: _____ MIEET / MIECT

NOTE BEM: Leia atentamente todas as questões, comente o código usando a linguagem C e respeite a convenção de passagem de parâmetros e salvaguarda de registos que estudou. Na tradução para o *Assembly* do MIPS respeite rigorosamente os aspetos estruturais e a sequência de instruções indicadas no código original fornecido.

O código em C apresentado pode não estar funcionalmente correcto, pelo que **não deve ser interpretado**.

Este teste é constituído por 4 folhas. Preencha o cabeçalho em todas elas.

1) Codifique em *Assembly* do MIPS a seguinte função `muc()` :

```
char to_up(char);
int muc(char *s)
{
    int i, count = 0;
    for (i=0; s[i] != '\0'; i++)
    {
        if ((s[i] >= 'a') && (s[i] <= 'z'))
        {
            s[i] = to_up(s[i]);
            count++;
        }
    }
    return count;
}
```

Variável	Registo(s)
s	\$a0->\$s0
i	\$s1
count	\$s2
&s[i]	\$s3

Função Intermédia

Label	Instrução em <i>assembly</i>	Comentário em C
muc:	subu \$sp,\$sp,20	# Função intermédia
	sw \$ra,0(\$sp)	#
	sw \$s0,4(\$sp)	#
	sw \$s1,8(\$sp)	# Prólogo
	sw \$s2,12(\$sp)	#
	sw \$s3,16(\$sp)	#
	move \$s0,\$a0	# \$s0 = s
	li \$s2,0	# count=0
	li \$s1,0	# i=0
for:	addu \$s3,\$s0,\$s1	# while(s[i] != '\0')
	lb \$a0,0(\$s3)	#
	beq \$a0,'\0',endfor	# {
if:	blt \$a0,'a',endif	# if(s[i] >= 'a' &&
	bgt \$a0,'z',endif	# s[i] <= 'z')
		# {
	jal to_up	#
	sb \$v0,0(\$s3)	# s[i] = to_up(s[i])
	addi \$s2,\$s2,1	# count++
		# }
endif:	addi \$s1,\$s1,1	# i++
	j for	# }
endfor:	move \$v0,\$s2	# return count
	lw \$ra,0(\$sp)	#
	lw \$s0,4(\$sp)	#
	lw \$s1,8(\$sp)	# Epílogo
	lw \$s2,12(\$sp)	#
	lw \$s3,16(\$sp)	#
	addu \$sp,\$sp,20	#
	jr \$ra	#

2) Codifique em *Assembly* do MIPS a seguinte função **main()** :

```
#define N    10
#define LEN  20
int fun(char *);
int max(int *, int);
int main(void)
{
    static char str[LEN];
    static int cnt[N];
    int i=0;
    do
    {
        read_string(str, LEN);
        cnt[i] = fun(str);
        print_string(str);
    } while (++i < N);
    print_int10(max(cnt, N));
    return 0;
}
```

Função Intermédia

```
.eqv    N,10
.eqv    LEN,20
.eqv    read_string,8
.eqv    print_string,4
.eqv    print_int10,1
```

Variável	Registo(s)
i	\$s0
&cnt[i]	\$t0

Label	Instrução em <i>assembly</i>
	.data
str:	.space LEN
cnt:	.space 40 # N * 4
	.text
	.globl main
main:	subu \$sp,\$sp,8 #
	sw \$ra,0(\$sp) # Prólogo
	sw \$s0,4(\$sp) #
	li \$s0,0 # i = 0
do:	la \$a0,str # do {
	li \$a1,LEN #
	li \$v0,read_string #
	syscall # read_string(str,LEN)
	la \$a0,str #
	jal fun #
	la \$t1,cnt #
	sll \$t0,\$s0,2 #
	addu \$t0,\$t0,\$t1 #
	sw \$v0,0(\$t1) # cnt[i] = fun(str)
	la \$a0,str #
	li \$v0,print_string #
	syscall # print_string(str)
	addi \$s0,\$s0,1 # i++
	blt \$s0,N,do # } while(i < N)
	la \$a0,cnt #
	li \$a1,N #
	jal max # max(cnt,N)
	move \$a0,\$v0 #
	li \$v0,print_int10 #
	syscall # print_int10(max(cnt,N))
	li \$v0,0 # return 0
	lw \$ra,0(\$sp) #
	lw \$s0,4(\$sp) # Epílogo
	addu \$sp,\$sp,8 #
	jr \$ra

4) Analise o programa *Assembly* seguinte e responda às questões:

```

        .data                # 0x10010000
D1:      .byte    1,2        #
S1:      .asciiiz "AC1-2018" # Códigos ASCII-> '0':0x30, 'A':0x41, '-':0x2D
        .align    2         #
A1:      .space   48         #
D2:      .text             # 0x00400000
        .globl   main      #
main:    la        $t0,S1    #
        la        $t1,A1    #
        ori       $t2,$0,8   #
        addi      $t2,$t2,-1 #
        sll       $t2,$t2,2   #
        addu      $t2,$t2,$t1 #
C1:      lb        $t3,0($t0) #
        sw        $t3,0($t2) #
        add       $t0,$t0,1   #
C2:      addi      $t2,$t2,-4  #
        bge       $t2,$t1,C1  #
        jr        $ra        #
    
```

- a) O espaço total de memória ocupado pela *string* referenciada pelo *label* "S1" é: 9 bytes
- b) O conteúdo de cada uma das posições de memória ocupadas pela *string* (do endereço mais baixo para o mais alto) é: 0x41, 0x43, 0x31, 0x2D, 0x32, 0x30, 0x31, 0x38, 0x00
- c) Considerando que o segmento de dados começa no endereço 0x10010000, o valor dos registos \$t0 e \$t1 após a execução das duas primeiras instruções do trecho de código é:
 \$t0: S1=0x10010002 \$t1: A1=0x1001000C
- d) Se "A1" for o endereço inicial de um *array* de inteiros, a dimensão máxima desse *array* é: 12,
 calculada como: 48 / 4 = 12 e o endereço de memória do elemento A1[2] é:
&A1[2]=0x1001000C + 2 * 4 = 0x10010014
- e) O número total de bytes de memória usado pelo segmento de dados (D2-D1) é: 60 bytes
2 + 9 + 1 + 48 = 60 bytes
- f) Considerando que a primeira instrução do trecho de código fornecido está armazenada a partir do endereço 0x00400000, os endereços a que correspondem os *labels* "C1" e "C2" são (tenha em atenção as instruções virtuais do código e a respetiva decomposição em instruções nativas):
 C1: 0x00400020 C2: 0x0040002C
- g) O valor do registo \$t2 calculado na instrução "addu" é: &A1[7]=0x1001000C + 7 * 4
&A1[7]=0x1001000C + 1C = 0x10010028
- h) Na execução do programa, o número de vezes que o ciclo é realizado é: 8,
 controlado pela evolução do conteúdo do registo: \$t2
- i) O valor das *words* de 32 bits armazenadas pelo programa nas posições A1[2] e A1[6] do *array* é:
 A1[2]: 0x00000030 A1[6]: 0x00000043
- j) O valor dos registos \$t0 e \$t2 no fim do programa é, \$t0: 0x1001000A, \$t2: 0x10010008