Organização e Arquitetura de Processadores

Assembly do MIPS

Exercícios em Laboratório

Principais ferramentas: ***MARS***
Apêndice A do livro do Patterson e Hennessy

Prof. Dr. César Marcon

Exercícios

1. Fazer o programa ***SomaInt*** que lê dois inteiros da entrada padrão e apresenta a soma destes inteiros na saída padrão
2. Fazer o programa ***SomaVetores*** que soma elemento a elemento de dois vetores de mesmo tamanho armazenados em memória e coloca o resultado de cada soma em cada elemento de um terceiro vetor
3. Dado que os caracteres ASCII (American Standard Code for Information Interchange) minúsculos estão no intervalo 97 ('a') a 122 ('z') e os maiúsculos estão no intervalo de 65 ('A') a 90 ('Z'), faça o programa ***ToLower*** que lê um vetor ASCII armazenado em memória e altera o mesmo para ter todos os caracteres em minúsculo
4. Implementar o programa ***ProgFib*** que armazena em um vetor os primeiros 15 números da série de Fibonacci
5. Implementar o programa ***OrdenaCrescente*** em assembly do MIPS que ordena de forma crescente um vetor de entrada

Programa SomaInt (1)

1. Programa que lê 2 inteiros da entrada padrão e apresenta a soma destes na saída padrão

```
void main() {
   int a = getchar();
   int b = getchar();
   int c = a + b;
   printf("%d", c);
}
```

OBS. Na implementação assembly, as variáveis são apenas registradores

Programa SomaInt (2)

Programa que lê 2 inteiros da entrada padrão e apresenta a soma destes na saída padrão

```
void main() {
   int a = getchar();
   int b = getchar();
   int c = a + b;
   printf("%d", c);
}
```

OBS. Na implementação assembly, as variáveis são apenas registradores

```
.text
   .globl main
main:                          # void main() {
   li   $v0, 5                 #
   syscall                     #
   move $t0, $v0               #  int a = getchar();
   li   $v0, 5                 #
   syscall                     #
   move $t1, $v0               #  int b = getchar();
   add     $t0, $t0, $t1       #  int c = a + b;
   move $a0, $t0               #
   li   $v0, 1                 #
   syscall                     #  printf("%d", c);
   li   $v0, 10                #
   syscall                     # }
```

Programa SomaInt – com macros(3)

1. Programa que lê 2 inteiros da entrada padrão e apresenta a soma destes na saída padrão

```
void main() {
   int a = getchar();
   int b = getchar();
   int c = a + b;
   printf("%d", c);
}
```

```
.text
.globl main

main:                       # void main() {
   readInt($t0)             #  int a = getchar();
   readInt($t1)             #  int b = getchar();
   add  $t0, $t0, $t1       #  int c = a + b;
   printInt($t0)            #  printf("%d", c);
   exit()                   # }
```

```
.macro exit()
   li $v0, 10
   syscall
.end_macro

.macro readInt(%registrador)
   li $v0, 5
   syscall
   move %registrador, $v0
.end_macro

.macro printInt(%registrador)
   li $v0, 1
   move $a0, %registrador
   syscall
.end_macro
```

Programa SomaVetores (1)

2. Programa que soma elemento a elemento de 2 vetores de mesmo tamanho armazenados em memória e coloca o resultado de cada soma em cada elemento de um terceiro vetor

```
int vetA[] = {0,  3,  6,  9, -2, -5, -4, 5, 1, -6};
int vetB[] = {6, -9, -6, -3, -2,  4, -3, 2, 2, -1};
int vetC[10];
int *pA, *pB, *pC, *pCFim;

void main() {
  int *pA = &vetA[0];
  int *pB = &vetB[0];
  int *pC = &vetC[0];
  int *pCFim = &vetC[10];
  while(pC < pCFim) {
     *pC = *pA + *pB;
     pA++;
     pB++;
     pC++;
  }
}
```

Programa SomaVetores (2)

```
.text
  .globl main
main:                    # void main() {
  la    $s0, vetA        #  int *pA = &vetA[0];
  la    $s1, vetB        #  int *pB = &vetB[0];
  la    $s2, vetC        #  int *pC = &vetC[0];
  la    $s3, vetC+40     #  int *pCFim=&vetC[10];
while:                   #
  bge   $s2, $s3, fim    #  while(pC < pCFim) {
  lw    $t0, 0($s0)      #   // $t0 = *pA;
  lw    $t1, 0($s1)      #   // $t1 = *pB;
  add   $t2, $t0, $t1    #   // $t2 = *pA + *pB;
  sw    $t2, 0($s2)      #   *pC = *pA + *pB;
  addi  $s0, $s0, 4      #     pA++;
  addi  $s1, $s1, 4      #     pB++;
  addi  $s2, $s2, 4      #     pC++;
  j     while            #  }
fim:                     #
  li    $v0, 10          #
  syscall                # }
```

```
int vetA[] = {0,  3,  6,  9, -2, -5, -4, 5, 1, -6};
int vetB[] = {6, -9, -6, -3, -2,  4, -3, 2, 2, -1};
int vetC[10];
int *pA, *pB, *pC, *pCFim;

void main() {
  int *pA = &vetA[0];
  int *pB = &vetB[0];
  int *pC = &vetC[0];
  int *pCFim = &vetC[10];
  while(pC < pCFim) {
          *pC = *pA + *pB;
          pA++;
          pB++;
          pC++;
  }
}
```

```
# Implementaremos ponteiros como regs

.data
vetA:  .word 0, 3, 6, 9,-2,-5,-4,5,1,-6
vetB:  .word 6,-9,-6,-3,-2, 4,-3,2,2,-1
vetC:  .space 40
```

Programa ToLower (1)

3. Intervalo ASCII ['a', 'z'] está no intervalo numérico [97, 122] e intervalo ASCII ['A', 'Z'] está no intervalo numérico [65, 90]. Faça programa que leia um vetor ASCII armazenado em memória e altere o mesmo para ter todos os caracteres em minúsculo

```
char vetAscii[] = "IstO eh 1 TEste";

void toLower(char *p) {
   if(*p >= 'A' && *p <= 'Z')
      *p = *p - 'A' + 'a';
}

void main() {
   char *p = &vetAscii[0];

   while(*p != 0) {
      toLower(p);
      p++;
   }
}
```

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Programa ToLower (2)

```
char vetAscii[] = "IstO eh 1 TEste";

void toLower(char *p) {
      if(*p >= 'A' && *p <= 'Z')
            *p = *p - 'A' + 'a';
}

void main() {
      char *p = &vetAscii[0];

      while(*p != 0) {
            toLower(p);
            p++;
      }
}
```

```
.data
vetAscii: .asciiz "IstO eh 1 TEste"
```

```
.text
   .globl main
main:                          # void main() {
   la $s0, vetAscii            #    char *p = &vetAscii[0];
while:                         #
   lb $t0, 0($s0)              #
   beq $t0, $zero, fim         #    while(*p != 0) {
   jal toLower                 #       toLower(p);
   addi $s0, $s0, 1            #       p++;
   j while                     # }
fim:                           #
   li $v0, 10                  #
   syscall                     # }

toLower:                       # void toLower(char *p) {
   bge $t0, 'A', verZ          #
   jr $ra                      #
verZ:                          #
   ble  $t0, 'Z', toLow        #
   jr $ra                      #  if(*p >= 'A' && *p <= 'Z')
toLow:                         #
   subi $t0, $t0, 'A'          #
   addi $t0, $t0, 'a'          #
   sb $t0, 0($s0)              #     *p = *p - 'A' + 'a';
   jr $ra                      # }
```

Programa ProgFib

4. Implementar o programa *ProgFib* que armazena em um vetor os primeiros 15 números da série de Fibonacci

```
int vet[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int *p = &vet[0], *pFim = &vet[14];
int a = 0, b = 1;

void main() {
  *p++ = a;
  *p++ = b;
  while(p <= pFim) {
    int aux = a + b;
    a = b;
    b = aux;
    *p++ = b;
}
```

Saída: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Programa OrdenaCrescente

5. Implementar o programa OrdenaCrescente em assembly do MIPS que ordena de forma crescente um vetor de entrada

```
int vet[10] = {3, 6, 6, 9, 12, 12, 7, 3, 5, 7};
int *pFim=&vet[9];

void main() {
  for(int *p = &vet[0]; p < pFim; p++) {
    for(int *k = p+1; k <= pFim; k++) {
      if(*p > *k) {
        int swap = *p;
        *p = *k;
        *k = swap;
      }
    }
  }
}
```

Resultado:

Input vet: 3 6 6 9 12 12 7 3 5 7

Output vet: 3 3 5 6 6 7 7 9 12 12