LISTA DE INTRODUÇÃO Á ARQUITETURA DE COMPUTADORES

Codificar trechos em C e $assembly$ para as funcionalidades indicadas abaixo. Pede-se também comparar o código $assembly$ produzido pelo compilador com o seu código. Comentar as diferenças mais relevantes. Considerar, de modo geral, operações em $32\ bits$. O trecho abaixo implementa os exercícios 1 e 2 (de duas formas). A chamada $ler\ (v, \&n)$ preenche um vetor de int e o seu respectivo tamanho.

```
int v[100], n, t; char s[100];
int main(int argc, char *argv[]) {
```

```
// produto dos elementos
// de um vetor de inteiros
 printf ("entre com o vetor\n");
 ler (v, &n);

 asm (
 ".intel_syntax noprefix\n\
 mov eax, 1             \n\
 mov ecx, _n            \n\
 xor ebx, ebx           \n\
V0:                     \n\
 cmp ecx, 0             \n\
 je  V1                 \n\
 imul eax, [ebx*4+_v]   \n\
 inc ebx                \n\
 dec ecx                \n\
 jmp V0                 \n\
V1:                     \n\
 mov _t, eax            \n\
 .att_syntax prefix     \n"
  );

  printf ("%d\n", t);
```

```
// strlen


 printf ("entre com a string\n");
 gets (s);

 asm (
 ".intel_syntax noprefix  \n\
 xor ebx, ebx             \n\
S0:                       \n\
 cmp byte ptr [ebx+_s], 0 \n\
 je  S1                   \n\
 inc ebx                  \n\
 jmp S0                   \n\
S1:                       \n\
 mov _t, ebx              \n\
 .att_syntax prefix       \n"
  );


  printf ("%d\n", t);
```

```
// strlen

 asm (
 ".intel_syntax noprefix\n\
 lea ebx, _s            \n\
 mov edi, ebx           \n\
T0:                     \n\
 cmp byte ptr [ebx], 0  \n\
 je  T1                 \n\
 inc ebx                \n\
 jmp T0                 \n\
T1:                     \n\
 sub ebx, edi           \n\
 mov _t, ebx            \n\
 .att_syntax prefix     \n"
  );

 printf ("%d\n", t);

// outros blocos assembly

  return 0;
}
```

1. **Obter o produto dos elementos de um vetor de int;**
2. **Obter o comprimento da $string\ s$, sabendo-se que o fim da $string$ é indicado pelo caractere terminador NUL ($'\backslash 0'$), cujo valor é equivalente ao inteiro $zero$;**
3. Obter a soma dos elementos de um vetor de int;
4. Obter a soma dos elementos de um vetor de $long\ long\ int$;
5. Obter a soma dos quadrados dos elementos de um vetor de int;
6. Obter a quantidade de números pares contidos em um vetor de int;
7. Obter a quantidade de números negativos contidos em um vetor de int;
8. Obter o endereço de memória da primeira ocorrência de um determinado $x\ int$ no vetor de int. Retorna $NULL$ no caso de não encontrar.
9. Obter o endereço de memória do maior elemento em um vetor de int.
10. Obter a soma dos produtos dos elementos de dois vetores de int de mesmo comprimento ($\sum u[i] * v[i]$);
11. Contar a quantidade de bits ligados (1) em um $unsigned\ int$ $(Exemplo: x = 15 \rightarrow c = 4)$;
12. Contar a quantidade de dígitos decimais de um $unsigned\ int$ $(Exemplo: x = 4365 \rightarrow c = 4)$;
13. Somar os valores dos dígitos decimais de um $unsigned\ int$ $(Exemplo:\ x = 315 \rightarrow\ c = 9)$;
14. Somar o quadrado dos valores dos dígitos decimais de um $unsigned\ int$ $(Exemplo:\ x = 315 \rightarrow\ c = 3^2 + 1^2 + 5^2 = 35)$;
15. Copiar a string de src para dst: $char\ *\ strcpy\ (char\ *\ dest, const\ char\ *\ src)$;
16. Retornar o endereço da primeira ocorrência de um $char$ em uma string. Retorna $NULL$ em caso de não encontrar: $char\ *\ strchr\ (char\ *\ s, int\ c)$;
17. Testar se uma string forma um palíndromo;

$MOV\ REG/MEM, REG/MEM/IMM$	$AND \mid TEST\ REG/MEM, REG/MEM/IMM$
$MOVSX\ REG, REG/MEM$	$OR\ REG/MEM, REG/MEM/IMM$
$MOVZX\ REG, REG/MEM$	$XOR\ REG/MEM, REG/MEM/IMM$
$ADD\ REG/MEM, REG/MEM/IMM$	$NOT\ REG/MEM \mid NEG\ REG/MEM$
$ADC\ REG/MEM, REG/MEM/IMM$	$SHL - SAL\ REG/MEM, IMM/CL$
$INC\ REG/MEM$	$SHR - SAR\ REG/MEM, IMM/CL$
$SUB\ REG/MEM, REG/MEM/IMM$	$ROL - RCL\ REG/MEM, IMM/CL$
$SBB\ REG/MEM, REG/MEM/IMM$	$ROR - RCR\ REG/MEM, IMM/CL$
$DEC\ REG/MEM$	$SHLD\ REG/MEM, REG, IMM/CL$
$LEA\ REG, MEM$	$SHRD\ REG/MEM, REG, IMM/CL$
CBW	$BSF\ REG, REG/MEM \mid BSR\ REG, REG/MEM$
CWD	$BT \mid BTS \mid BTR \mid BTC\ REG/MEM, REG/IMM$
$CWDE$	$SetCC\ REG_8/MEM_8$
CDQ	$LOOP\ LABEL$
$MUL\ REG/MEM$	$LOOPE/LOOPZ\ LABEL$
$IMUL\ REG/MEM$	$LOOPNE/LOOPNZ\ LABEL$
$IMUL\ REG, REG/MEM/IMM$	$REP\ STRING$
$IMUL\ REG, REG/MEM, IMM$	$REPE/REPZ\ STRING$
$DIV\ REG/MEM$	$REPNE/REPNZ\ STRING$
$IDIV\ REG/MEM$	$PUSH\ REG/MEM/IMM$
$CMP\ REG/MEM, REG/MEM/IMM$	$POP\ REG/MEM/IMM$
$JMP\ LABEL$	$IN\ \{AL, AX, EAX\}, IMM/DX$
$J\{C, O, S, P, Z\} \mid JN\{C, O, S, P, Z\} \mid JP\{E, O\} LABEL$	$OUT\ IMM/DX, \{AL, AX, EAX\}$
$J\{E, A, B, G, L\} \mid JN\{E, A, B, G, L\}\ LABEL$	$INT\ IMM$
$JN\{A, B, G, L\}E\ LABEL$	$RET\ IMM$
$JECXZ\ LABEL$	$STD \mid\ STI \mid\ CLD \mid CLI$
Deve ser dada atenção à combinação MEM-MEM e ao tamanho dos operandos.	
$LODS\{B, W, D\}$ $\{AL, AX, EAX\} = [ESI]$ $ESI = ESI \pm \{1, 2, 4\}$ $STOS\{B, W, D\}$ $[EDI] = \{AL, AX, EAX\}$ $EDI = EDI \pm \{1, 2, 4\}$ $MOVS\{B, W, D\}$ $LODS\{B, W, D\}$ $STOS\{B, W, D\}$ *Não alteram FLAGS*	$SCAS\{B, W, D\}$ $CMP\ \{AL, AX, EAX\}, [EDI]$ $\{AL, AX, EAX\} - [EDI]$ $EDI = EDI \pm \{1, 2, 4\}$ $CMPS\{B, W, D\}$ $CMP\ [ESI], [EDI]$ $[ESI] - [EDI]$ $\{EDI, ESI\} = \{EDI, ESI\} \pm \{1, 2, 4\}$ *Alteram FLAGS*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	ID	VIP	VIF	AC	VM	RF	0	NT	IOPL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF