LISTA 2 - Período 2021-1 Remoto
105 pontos

O objetivo desta lista é sedimentar o conhecimento das manipulações feitas pelas instruções de ponto flutuante sobre a pilha x87, bem como analisar o comportamento da pilha dos processos e outras gerações de código de montagem feitas pelo GCC.

Questão 1) (45 pontos) Na lista 1 foi dado o seguinte programa C:

```
int main (){
   float x = 4.3;
  double z = 3.2, y;
  y = x - z;
  printf ("y = 4.3 - 3.2 = %10.13f \n", y);}
```

Compilando com “gcc -m32 -O2 -fno-PIC -S”, descartadas algumas linhas de diretivas, temos:

```
.LC1:
        .string         "y = 4.3 - 3.2 = %10.13f \n"
main:
1       endbr32
2       leal    4(%esp), %ecx
3       andl    $-16, %esp
4       pushl -4(%ecx)
5       pushl %ebp
6       movl %esp, %ebp
7       pushl %ecx
8       subl    $8, %esp
9       pushl $1072798105
10      pushl $-858993460
11      pushl $.LC1
12      call    printf
13      movl -4(%ebp), %ecx
14      addl    $16, %esp
15      xorl    %eax, %eax
16      leave
17      leal    -4(%ecx), %esp
18      ret
```

Ao entrar em main, o topo da pilha contém o endereço de retorno para o SO e a base aponta para um determinado endereço. A pilha irá crescer para baixo, em direção a endereços mais baixos.

endereço	**Conteúdo da Pilha <4 bytes>**	***Comentários***
%ebp		*Base da pilha*
⋮	⋮	⋮
%esp	RIP SO	*Topo da pilha, end de retorno a SO*
		Fora da pilha

a) (10) Desenhe o conteúdo da pilha imediatamente após a execução da linha 12. Na coluna endereço, represente sempre a situação dos registradores ebp e esp no momento do desenho. Referencie os endereços das linhas da pilha sempre em relação à base de main após ela ser estabelecida. Na coluna comentários, preencha com “Após L?”, identificando a linha da instrução de montagem que gera o preenchimento daquela linha da pilha. Acrescente algum comentário que for pertinente. Marque com x16 os endereços que estiverem alinhados em múltiplos de 16.

b) (5) Qual a razão da existência da linha 7? Justifique, pois qualquer ação feita pelo GCC tem que ter uma justificativa plena.
c) (5) Qual a razão da execução da linha 4? Justifique, pois qualquer ação feita pelo GCC tem que ter uma justificativa plena.
d) (5) Justifique a existência da linha 15.
e) (5) No código de montagem mostrado, existe alguma linha que possa ser suprimida, sendo desnecessária? Justifique.
f) (5) Normalmente após um leave é executado o ret. Por que no código acima, entre o leave e o ret existe a linha 17?
g) (5) Qual o significado das constantes *1072798105 e -858993460? Justifique. Pode usar resultado da lista 1.*
h) (5) Considerando como m o custo de execução de uma instrução que acessa a memória física e n o custo de instrução que não acessa a memória, calcule a estimativa de custo do código de montagem.

Questão 2) (30 pontos) Na lista 1 foi dado o seguinte programa:

```
int main (){
     float x = 4.3;
    double z = 3.2, y;
    y = x - z;
    printf ("y = 4.3 - 3.2 = %10.13f \n", y);}
```

Compilando com “gcc -m32 -fno-PIC -S”, descartadas algumas linhas de diretivas, temos:

```
.LC2:
    .string  "y = 4.3 - 3.2 = %10.13f \n"
main:
1   endbr32
2   leal 4(%esp), %ecx
3   andl $-16, %esp
4   pushl    -4(%ecx)
5   pushl    %ebp
6   movl %esp, %ebp
7   pushl    %ecx
8   subl $36, %esp
9   flds .LC0
10  fstps    -28(%ebp)
11  fldl .LC1
12  fstpl    -24(%ebp)
13  flds -28(%ebp)
14  fsubl    -24(%ebp)
15  fstpl    -16(%ebp)
16  subl $4, %esp
17  pushl    -12(%ebp)
18  pushl    -16(%ebp)
19  pushl    $.LC2
20  call printf
21  addl $16, %esp
22  movl $0, %eax
23  movl -4(%ebp), %ecx
24  leave
25  leal -4(%ecx), %esp
26  ret
    .align 4
.LC0:
    .long    1082759578
    .align 8
.LC1:
    .long    2576980378
    .long    1074370969
```

Ao entrar em main, o topo da pilha contém o endereço de retorno para o SO e a base aponta para um determinado endereço.

endereço	**Conteúdo da Pilha <4 bytes>**	***Comentários***
%ebp		*Base da pilha*
⋮	⋮	⋮
%esp	RIP SO	*Topo da pilha, end de retorno a SO*
		Fora da pilha

a) (5) Justifique as diretivas .align 4 e .align 8. O que representam as constantes nos endereços .LCO e .LC1?
b) (5) Comente cada uma das linhas de 9 a 13, indicando o que elas causam. Indique o conteúdo de ST(0) após a execução de cada linha. Aponte as ineficiências de movimentação que pode detetar.
c) (5) Comente cada uma das linhas de 14 e 15, indicando o que elas causam. Indique o conteúdo de ST(0) após a execução de cada linha.
d) (10) Desenhe a pilha após serem executadas as linhas 1 a 20, ou seja, imediatamente após o call printf. Na coluna endereço, represente sempre a situação dos registradores ebp e esp no momento do desenho. Referencie os endereços das linhas da pilha sempre em relação à base de main após ela ser estabelecida. Na coluna comentários, preencha com “Após L?”, identificando a linha da instrução de montagem que gera o preenchimento daquela linha da pilha. Acrescente algum comentário que for pertinente. Marque com x16 os endereços que estiverem alinhados em múltiplos de 16. Preencha a pilha com os conteúdos de x, z e y em hexadecimal.
e) (5) Tente explicar a razão da Linha 16 e faça as observações pertinentes em relação a ineficiências no uso de memória e alinhamentos nas operações entre FPU x87 e a memória.

Questão 3) (15 pontos) Um valor inteiro foi apagado na rotina C abaixo.

```
int foo(unsigned int n) {unsigned int x; x = n/… ;return x;}
```

O código de montagem gerado pelo GCC com otimização -O1 é:

```
foo:
1    endbr32
2    movl $1374389535, %edx
3    movl %edx, %eax
4    mull 4(%esp)
5    movl %edx, %eax
6    shrl $3, %eax
7    ret
```

Comente cada linha, sob o ponto de vista de engenharia reversa, para descobrir o valor apagado. A justificativa tem que ser clara e todas as linhas têm que ser comentadas, sem exceção, de 1 a 7. Pode ser usada calculadora e eventuais conversões para binário e/ou hexadecimal, se necessárias, podem ser apresentadas sem o passo a passo.

Questão 4) (15 pontos) É dada uma rotina que recebe um valor inteiro com sinal e imprime o resultado da divisão, que pode ser negativo ou positivo. Determine o valor do divisor apagado na rotina C abaixo:

```
int foo (int n) {int x; x = n/… ; return x;}
```

O código de montagem gerado pelo GCC com otimização -O1 é:

```
foo:
1    endbr32
2    movl 4(%esp), %ecx
3    movl $-1307163959, %edx
4    movl %ecx, %eax
5    imull    %edx
6    leal (%edx,%ecx), %eax
7    sarl $4, %eax
8    sarl $31, %ecx
9    subl %ecx, %eax
10   ret
```

Comente cada linha, sob o ponto de vista de engenharia reversa, para descobrir o valor apagado. Todas as linhas devem ser justificadas. GCC insere instrução por uma razão que tem que ficar clara. Pode ser usada calculadora e eventuais conversões para binário e/ou hexadecimal, se necessárias, podem ser apresentadas sem o passo a passo.