Prosedur

```
int soal(int (*f)(int, int), int* arr, int c)
{
  int i, x;

  if( c <= 0 )

    return c;

  x = arr;

  for(i = 1; i <= c; i++)

    x = (*f)(arr, arr[i]);

  return x;
}
```

```
(gdb) disas soal
Dump of assembler code for function soal:
0x080483a4 <soal+0>:push %ebp
0x080483a5 <soal+1>:mov %esp,%ebp
0x080483a7 <soal+3>:push %edi
0x080483a8 <soal+4>:push %esi
0x080483a9 <soal+5>:push %ebx
0x080483aa <soal+6>:sub $0xc,%esp
0x080483ad <soal+9>:mov 0xc(%ebp),%edi    edi = arr
0x080483b0 <soal+12>:mov 0x10(%ebp),%esi  esi = c
0x080483b3 <soal+15>:test %esi,%esi            if c <= 0
0x080483b5 <soal+17>:jle 0x80483db <soal+55>
0x080483b7 <soal+19>:mov (%edi),%edx     edx = arr
0x080483b9 <soal+21>:cmp $0x1,%esi
0x080483bc <soal+24>:jle 0x80483d9 <soal+53>
0x080483be <soal+26>:mov $0x1,%ebx
0x080483c3 <soal+31>:mov (%edi,%ebx,4),%eax    eax = arr[i]
0x080483c6 <soal+34>:mov %eax,0x4(%esp)
0x080483ca <soal+38>:mov %edx,(%esp)
0x080483cd <soal+41>:call *0x8(%ebp)    panggil fungsi f
0x080483d0 <soal+44>:mov %eax,%edx  harusnya disini nilai edx diganti tapi dikodenya gaada
0x080483d2 <soal+46>:add $0x1,%ebx
0x080483d5 <soal+49>:cmp %ebx,%esi
0x080483d7 <soal+51>:jg 0x80483c3 <soal+31>
0x080483d9 <soal+53>:mov %edx,%esi
0x080483db <soal+55>:mov %esi,%eax
0x080483dd <soal+57>:add $0xc,%esp
0x080483e0 <soal+60>:pop %ebx
0x080483e1 <soal+61>:pop %esi
0x080483e2 <soal+62>:pop %edi
0x080483e3 <soal+63>:pop %ebp
0x080483e4 <soal+64>:ret
```

```
End of assembler dump.
```

a. Pada alamat `0x080483a9` terdapat instruksi `push %ebx`. Jelaskan 2 hal yang terjadi sebagai hasil dari eksekusi instruksi tersebut, dan jelaskan mengapa hal ini diperlukan.

b. Misalkan setelah eksekusi instruksi pada alamat `0x080483a9 (push %ebx)`, nilai `%esp` adalah `0xffff0000`, pada alamat berapakah parameter f berada?

Cache Simulation

a) Pada cache, apa yang dimaksud dengan:
   a. Bit valid
   b. Bit set
   c. Bit tag
   d. Bit block

b) Pada cache, berapakah ukuran berikut:
   a. Jumlah set, jika set bit adalah 2.
   b. Berapa ukuran byte sebuah block, jika block bit adalah 4.

c) Jelaskan kapan terjadi:
   a. Cache Hit
   b. Cache Miss

d) Perhatikan *two-way set associative cache* berikut dengan aturan pergantian dengan *Least Recently Used (LRU)* dimana baris ke 2 pada set adalah data terbaru. Dengan komposisi bit tag (t) 2 bit, bit set (s) 1 bit, dan bit block (b) 4 bit seperti tabel berikut:

t	s	b
xx	x	xxxx

Cache sebelumnya telah terisi data sebagai berikut (v dan tag *field* adalah biner, sedangkan *field* block adalah *decimal*).

	v	tag	Block
set 0	1	00	M[0-15]
	1	11	M[96-111]

	V	Tag	block
set 1	1	01	M[48-63]
	1	00	M[16-31]

Selanjutnya program memanggil alamat-alamat berikut.

a. Isilah kolom *hit/miss*!
b. Isilah *cache* (set 0 & 1) dengan kondisi terakhir setelah alamat no 6 dijalankan!
c. Hitunglah *miss rate*-nya!

No	Alamat	Binary	Hit/Miss
1	0	0000000	hit
2	33	0100001	miss
3	100	1100100	miss
4	117	1110101	miss
5	40	0101000	hit
6	20	0010100	hit

	v	tag	Block
set 0	1	11	M[96-111]
	1	01	M[32-47]

	v	tag	Block
set 1	1	11	M[112-127]
	1	00	M[16-31]

Array
Diberikan kode berikut:

```
int mat1[M][N];
int mat2[N][M];
int tambah(int i, int j)
{
     return mat1[i][j] + mat2[i][j];
}
```

Dengan hasil translasi kode assembly sebagai berikut:

```
tambah:
     pushl %ebp
     movl %esp,%ebp
     movl 8(%ebp),%eax
     movl 12(%ebp),%ecx
     sall $2,%ecx
     leal (%eax,%eax,4),%edx
     sall $1, %edx
     leal (%eax,%eax,2),%eax
     movl mat1(%ecx,%eax,4),%eax
     addl mat2(%ecx,%edx,4),%eax
     movl %ebp,%esp
     popl %ebp
     ret
```

eax = i
ecx = j
ecx = 4*j
edx = 10*i
eax = 3*i

eax = mat1[j + 3*i*4] + mat2[j + 10*i*4]

maka M = 10 dan N = 3

Tentukan nilai M dan N!
M =
N =

Struktur
Berikut adalah suatu deklarasi struktur :

```
struct s1 {
     int a;
     char b;
     short c;
     float* d;
     double e;
     int f[3];
     int g[3][4]
};
```

s1+0 - s1+4 = int a
s1+4 - s1 + 5 = char b
s1+5 - s1 + 8 = padding ukuran 3 byte
s1+8 - s1+10 = short c
s1+10 - s1+12 = padding ukuran 2 byte
s1+12 - s1+16 = float* d
s1+16 - s1+24 = double e
s1+24 - s1+36 = int f[3]
s1+36 - s1+84 = int g[3][4]
s1+84 - s1+88 = padding ukuran 4 byte

Kode struktur tersebut di-compile menggunakan mesin IA32-Windows.

a) Nyatakan hasil implementasi struktur tersebut sebagai penempatan pada suatu rangkaian alamat memory, beserta penambahan paddingnya!
(Gunakan notasi *base+offset* misal {s1+0 – s1+4: a, s1+4 – s1+?: b, s1+? – s1+?: padding, dst})

b) Berapa total bit memory yang terpakai untuk implementasi struktur tersebut?

c) Lakukan penataan ulang terhadap struktur tersebut agar dapat menghemat

penggunaan memory! (Gunakan notasi yang sama dengan poin a)

tinggal susun dari yg terbesar

Diketahui fungsi "hitung", "panjang", dan "tunjuk" yang melakukan penghitungan jumlah digit dari suatu bilangan integer:

```
int panjang(char *s) {
      return strlen(s);
}


void tunjuk(char *s, long *p) {
      long val = *p;
      sprintf(s, "%ld", val);
}


int hitung(long x) {
      long y;
      char buf[12];
      y = x;
      tunjuk(buf, &v);
      return panjang(buf);
}
```

Di bawah ini adalah potongan kode assembly dari fungsi "hitung", yang di-compile dengan 2 cara:

(a) Tanpa stack protector/canary:

```
int hitung(long x)
x disimpan di %rdi
1  hitung:
2     subq  $40, %rsp
3     movq  %rdi, 24(%rsp)
4     leaq  24 (%rsp), %rsi
5     movq  %rsp, %rdi
6     call  tunjuk
```

(b) Dengan stack protector/canary:

```
int hitung(long x)
x disimpan di %rdi
1  hitung:
2     subq  $56, %rsp
3     movq  %fs:40, %rax
4     movq  %rax, 40(%rsp)
5     xorl  %eax, %eax
6     movq  %rdi, 8(%rsp)
7     leaq  8(%rsp), %rsi
8     leaq  16(%rsp), %rdi
9     call  tunjuk
```

Pertanyaan 1:
Dari informasi di atas, tentukan posisi-posisi berikut pada stack frame:

untuk cara (a),

- **y** : …… byte offset relatif terhadap %rsp
- **buf** : …… byte offset relatif terhadap %rsp

untuk cara (b),

- **y** : …… byte offset relatif terhadap %rsp
- **buf** : …… byte offset relatif terhadap %rsp
- **canary** : …… byte offset relatif terhadap %rsp

(catatan: posisi byte offset dapat bernilai positif atau negatif)

Pertanyaan 2:
Jelaskan secara sederhana bagaimana perubahan di bagian (b) dapat meningkatkan keamanan terhadap buffer overflow attack!

Pertanyaan 3:
Jelaskan 2 metode perlindungan di level sistem (system-level protection) terhadap buffer overflow attack!