

WRITE-UP

Nama : Azfa Radhiyya Hakim
NIM : 13523115
KELAS : K01



Soal Bonus yang Dikerjakan	Soal Wajib yang Dihapus dari Writeup
shikanoko_nokonoko	kitsch
le_souvenir_avec_le_erepuscule	mesmerizer

Soal 1 – chicken_or_beef

1. Solusi

```
int chicken_or_beef(int chicken, int beef){
    int chicken_bits = (chicken >> 4) & 0xF;
    int beef_bits = (beef << 1) & 0xF;
    return chicken_bits | beef_bits;
}
```

2. Penjelasan Singkat

- Pada soal ini, kami diperitahkan untuk mengambil 4 bit kedua dari parameter chicken dan 4 bit pertama dari nilai parameter beef dikalikan dengan 2, lalu di meng-OR kan kedua nilai tersebut.
- Untuk memperoleh 4 bit kedua dari chicken, kita perlu menggeser bit chicken sebesar 4 ke kanan, lalu di & dengan 1111 (0xF)
- Untuk memperoleh 4 bit pertama dari beef*2, kita perlu mengalikan beef dengan dua terlebih dahulu (menggeser ke kiri sebesar 1), lalu di & dengan 1111 (0xF)

3. Referensi yang Digunakan

- Tidak ada.

Soal 2 – masquerade

1. Solusi

```
int masquerade(){
    return (1 << 31) ^ 0x1;
}
```

2. Penjelasan Singkat

- Bilangan terkecil dalam representasi integer two's complement adalah 1000..00 sebesar 32 bit, yang dapat dinyatakan dalam bentuk $1 \ll 31$.
- Untuk memperoleh bilangan terkecil kedua, kita hanya perlu mengganti bit terakhir dari $(1 \ll 31)$, yaitu dengan XOR digit terakhirnya dengan 1.

Bilangan terkecil dalam representasi integer two's complement adalah 1000..0001, yaitu 1000.0001 (-2147483647).

3. Referensi yang Digunakan

- Tidak ada

Soal 3 – airani_iofifteen

1. Solusi

```
int airani_iofifteen(int ioifi){  
    return ((ioifi >> 3) & 1) & ((ioifi >> 2) & 7) & ((ioifi >>  
1) & 15) & !((ioifi >> 4)) & (ioifi & 15);  
}
```

2. Penjelasan Singkat

- Soal memerintahkan kami untuk mengecek apakah suatu input *iofi* merupakan 15.
- Disini, saya mengecek setiap elemen yang ada di masukan tersebut.
- Agar memudahkan, digit ke-n yang saya tuliskan berarti digit ke-n yang dihitung dari bit paling kanan.
- Pertama, saya mengecek apakah digit keempat bernilai 1, dan menghasilkan true jika benar.
- Kedua, saya mengecek apakah digit ketiga dan keempat bernilai 1, dan menghasilkan true jika benar.
- Ketiga, saya mengecek apakah digit keempat, ketiga, dan kedua bernilai 1, dan menghasilkan true jika benar.
- keempat, saya mengecek apakah digit keempat, ketiga, kedua, dan pertama bernilai 1, dan menghasilkan true jika benar.
- Terakhir, saya mengecek apakah setelah digit keempat masih terdapat angka lagi, dan menghasilkan 0 jika benar. (!0 = 1)

3. Referensi yang Digunakan

- Tidak ada

Soal 4 – yobanashi_deceive

1. Solusi

```
int yobanashi_deceive (int f){  
    return f >> 3;  
}
```

2. Penjelasan Singkat

- Soal memerintahkan kami untuk menghitung nilai dari $\sqrt{\sqrt{\sqrt{n}}}$ dan dibulatkan ke float terdekat
- Berdasarkan pola yang diberikan dari testcase, saya menebak bahwa jawaban dari soal ini adalah dengan menggeser bit dari n sebanyak 3 kali ke kanan.
- Setelah melihat referensi, eksponen hasil dari pengakaran akan sama dengan hasil eksponen dari pergeseran angka tersebut dengan 3.

3. Referensi yang Digunakan

- ChatGPT
<https://chatgpt.com/share/671cc451-8acc-8013-81aa-1e9e1c61d73a>

Soal 5 – snow_mix

1. Solusi

```
int snow_mix(int n){  
    int m = 1 < 23;  
    int first = n ^ m;  
    int a1 = (n & m) << 1;  
    int second = first ^ a1;  
    int a2 = (first & a1) << 1;  
    int third = (second & a2);  
    int a3 = (second & a2) << 1;  
    int fourth = third ^ a3;  
    int a4 = (third & a3) << 1;  
    return fourth ^ a4;  
}
```

2. Penjelasan Singkat

- Soal memerintahkan kami untuk menghitung nilai dari penjumlahan n dengan 2^{23} .

- Pertama-tama, saya mendefinisikan m terlebih dahulu sebagai 2^{23} .
- Kemudian, saya melakukan bruteforce sebanyak 4 kali untuk menambahkan bit pada n ke nilai 2^{23} . Langkah yang saya tuliskan seolah-olah menambah n dengan m, namun ada kemungkinan bahwa masih ada carry yang belum tersimpan. Sehingga, jika ada 1 yang teletak di bit yang sama, akan di shift ke left untuk ditambah.

3. Referensi yang Digunakan

- Mendapat ide dan bantuan dari teman, Barru Adi Utomo

Soal 6 – sky_hundred

1. Solusi

```
int sky_hundred(int n){
    int rem = n & 3;
    int nol = !(rem^0);
    int satu = !(rem ^ 1);
    int dua = !(rem ^ 2);
    int tiga = !(rem ^ 3);
    return (~(!(n >> 31)) + 1) & ((satu & 1) | (n & (~nol + 1)))
    | ((n+1) & (~dua + 1));
}
```

2. Penjelasan Singkat

- Pertama-tama, saya meninjau bahwa nilai dari fungsi ini akan bersifat periodik, yaitu memiliki pola sebagai berikut.
 - Jika $(n \% 4 == 0)$ -> mengeluarkan n
 - Jika $(n \% 4 == 1)$ -> mengeluarkan 1
 - Jika $(n \% 4 == 2)$ -> mengeluarkan n+1
 - Jika $(n \% 4 == 3)$ -> mengeluarkan 0
- Maka dari itu, saya mendeklarasi variable rem untuk menghitung sisa pembagiannya dengan 4, dan variabelen nol, satu, dua, dan tiga yang bernilai 1 jika nilai rem sesuai dengan nilai-nilai tersebut.
- Jika salah satu variable diantara nol, satu, dua, dan tiga bernilai 1, maka sudah dipastikan yang lainnya bernilai 0.
- Untuk mereturn 1, kita hanya perlu menggunakan $(satu \& 1)$
- Untuk mereturn n atau n+1, kita perlu melakukan operasi bitwise & pada nilai yang akan dikeluarkan dengan $\sim 1 + 1$.
- $(\sim(! (n \gg 31)) + 1)$, kode tersebut akan handle kasus ketika

nilai n yang dimasukkan adalah negatif.

- Jika semua kondisi salah (kecuali kasus negatif), maka fungsi akan mengeluarkan 0, yang merupakan kasus ketika n bersisa jika dibagi 3.

3. Referensi yang Digunakan

- Berdiskusi dengan teman

Soal 7 – ganganji

1. Solusi

```
int ganganji(int x){  
    int a = x + (x >> 3);  
    int cek = a >> 31;  
    int max = (1 << 31) - 1;  
    return (~cek & a) | (cek & max);  
}
```

2. Penjelasan Singkat

- Soal memerintahkan kami untuk menghitung nilai dari x dikalikan dengan 1,125.
- Pertama-tama, saya mengalikan a dengan 9/8, yaitu dengan menggesernya ke kanan sebanyak 3 bit (dibagi 9), lalu menambahkannya dengan x itu sendiri.
- Cek merupakan variabel untuk mendeteksi apakah bilangan tersebut positif atau negatif. Jika a positif, maka cek = 0 dan ~cek = -1. Namun jika a negatif, maka cek = -1 dan ~cek = 0.
- Dalam operasi (~cek & a), akan menghasilkan a jika cek = 0 dan menghasilkan 0 jika cek = -1.
- Dalam operasi (cek & max), akan menghasilkan nilai max jika cek = -1 dan 0 jika menghasilkan 0 jika cek = 0.
- Dengan mengoperasikan kedua nilai dengan operasi OR, maka diperoleh nilai a jika positif, dan max jika a menjadi overflow.

3. Referensi yang Digunakan

- Berdiskusi dengan teman-teman.

Soal 8 (bonus) – shikanoko_nokonoko

1. Solusi

```
int shikanoko_shikanoko (int x){
    int tigaenam = 255 + 105;
    int duatujuh = 255 + 15;
    int sisa = ((x % tigaenam) + tigaenam) % tigaenam;
    int cekjadi1 = (sisa == 0);
    int cekjadimin1 = (sisa < duatujuh) & (sisa > 90);
    return (cekjadi1 | (~cekjadimin1 + 1));
}
```

2. Penjelasan Singkat

- Pertama-tama, saya mendefinisikan variabel tigaenam dan duatujuh sebagai 360 dan 270 karena kami tidak diperbolehkan menulis konstanta yang lebih besar dari 255.
- Nilai cos dari suatu bilangan yang dibulatkan ke bawah hanya akan memunculkan 3 kemungkinan, yaitu -1, 0, dan 1.
- Maka dari itu, saya membuat variable sisa, untuk menyimpan nilai sisa pembagian x dengan 360. Kemudian, saya membuat variable cekjadi1 dan cekjadimin1 jika nilai x memenuhi conditional (dapat dilihat pada kode). Masing masing dari variable tersebut akan mengeluarkan nilai 1 atau 0.
- Terakhir, saya melakukan operasi OR pada cekjadi1 dan cekjadimin1, namun saya melakukan modifikasi pada operasi cekjadimin1 agar dapat mengeluarkan -1 sebagai hasilnya. Jika kedua kondisi tersebut tidak terpenuhi, maka fungsi akan mengeluarkan nilai 0.

3. Referensi yang Digunakan

- Tidak ada

Soal 9 – how_to_sekai_seifuku

1. Solusi

```
unsigned how_to_sekai_seifuku(unsigned f){
    int sign = f & 0x8000;
    int exp = (f >> 10) & 0x1F;
    int frac = f & 0x3FF;
}
```

```

if ((exp | frac) == 0){
    return sign << 16;
}
if (exp == 0x1F){
    if (frac == 0){
        return (sign << 16) | 0x7F800000;
    }
    return 0x7F800001;
}
if (exp == 0){
    while ((frac & 0x400) == 0){
        frac = frac << 1;
        exp = exp - 1;
    }
    frac = frac & 0x3FF;
    exp = exp + 1;
}
exp = exp + 112;

return (sign<<16) | (exp << 23) | (frac << 13);
}

```

2. Penjelasan Singkat

- Soal memerintahkan kami untuk merubah masukkan unsigned yang berupa half precision menjadi unsigned yang merupakan single precision.
- Berdasarkan referensi, sebuah half precision dapat diubah menjadi 2 bagian, yaitu sign (bit pertama), exponen (5 bit setelahnya), dan fraction/mantissa (10 bit setelahnya).
- Terdapat beberapa kasus khusus dalam mengubah format ini. Jika exp dan frac bernilai 0, maka dikeluarkan sign << 16. Selanjutnya, jika exp = 11111 (0x1F), jika frac bernilai 0, maka dikeluarkan infinity. Jika frac != 0, maka dikeluarkan -infinity.
- Lalu untuk kasus (exp == 0), saya menggeser bit dari frac (mantissa) hingga menjadi bentuk 1.xxx, lalu menyesuaikannya dengan eksponen. Lalu setelah loop selesai, saya melakukan operasi & 0x3FF kepada frac, untuk menyesuaikannya dengan 10 digit terakhir (0x3FF = 1111111111).
- Kemudian, saya mengubah exp dari single precision dengan cara menambahkan exp saat ini dengan (127-15).
- Terakhir, saya menggabungkan sign, exp, dan frac yang telah disesuaikan.

3. Referensi yang Digunakan

- Berdiskusi dengan teman-teman.
- Modul pembelajaran IF2130 - Organisasi dan Arsitektur Komputer: Floating Point
- ChatGPT
<https://chatgpt.com/share/671d0ca7-cb9c-8012-8c29-feceac7bb74c>

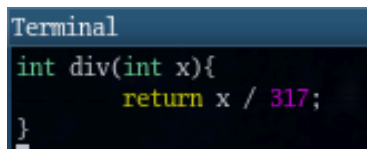
Soal 10 (bonus) – le_souvenir_avec_le_erepuscule

1. Solusi

```
int le_souvenir_avec_le_erepuscule(int x) {  
    int sign = x >> 31;  
    long long num = x;  
    int cons = 0x33af3e2f;  
    long long kali = num * cons;  
    int result =(kali >> 38);  
    return result + (~sign + 1);  
}
```

2. Penjelasan Singkat

- Soal ini perlu diselesaikan dengan mencari suatu konstanta yang ekuivalen dengan $1/317$.
- Disini, saya membuat files baru pada virtual machine, yang bernama test.c. File ini berisi fungsi yang me return nilai dari masukkan dibagi dengan 317. Setelah di save, saya akan meninjau fungsi ini melalui kode



```
Terminal  
int div(int x){  
    return x / 317;  
}
```

assembly yang dimiliki.

- Pertama-tama, saya mengcompile file test.c menjadi file executable dengan cara berikut.
- `>>> gcc -c -o test test.c`
- Setelah sukses tercompile, saya menulis command berikut.
- `>>> gdb test`
- Di dalam gdb tersebut, saya menulis command berikut

- >>> disas div
- Yang merupakan fungsi untuk melihat kode assembly dari fungsi C. Berikut adalah kode assembly yang saya dapat.

```
(gdb) disas div
Dump of assembler code for function div:
0x00000000 <+0>:    push    %ebp
0x00000001 <+1>:    mov     %esp,%ebp
0x00000003 <+3>:    call   0x4 <div+4>
0x00000008 <+8>:    add     $0x1,%eax
0x0000000d <+13>:   mov     0x8(%ebp),%ecx
0x00000010 <+16>:   mov     $0x33af3e2f,%edx
0x00000015 <+21>:   mov     %ecx,%eax
0x00000017 <+23>:   imul    %edx
0x00000019 <+25>:   sar     $0x6,%edx
0x0000001c <+28>:   mov     %ecx,%eax
0x0000001e <+30>:   sar     $0x1f,%eax
0x00000021 <+33>:   sub     %eax,%edx
0x00000023 <+35>:   mov     %edx,%eax
0x00000025 <+37>:   pop     %ebp
0x00000026 <+38>:   ret
End of assembler dump.
(gdb) █
```

- Lalu, saya menggunakan ClaudeAI untuk menganalisis kode tersebut, dan mendapatkan bahwa konstanta perkalian yang digunakan adalah 0x33af3e2f, yang kemudian di shift sebanyak 38.
- Kemudian saya handle kasus ketika nilai yang dimasukkan berupa negatif. ^^

3. Referensi yang Digunakan

- Berdiskusi dengan teman.
- Claude AI
<https://claude.ai/chat/fe8cd654-ea41-4b1b-9495-0477e7949c33>