WRITE-UP.

Nama: Muhammad Iqbal Haidar
NIM: 13523111
KELAS: K01

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

- Ekstrak 4 bit kedua dari chicken: Ini berarti kita mengambil bit ke-4 hingga ke-7. Untuk ini, kita bisa menggeser `chicken` ke kanan sebanyak 4 bit, lalu mengambil 4 bit paling kanan dengan operasi AND terhadap `0xF` (yang adalah 0b1111).
- Ekstrak 4 bit pertama dari beef*2: Pertama, kita kalikan `beef` dengan 2 (geser `beef` ke kiri 1 bit). Lalu, ambil 4 bit pertama dengan AND terhadap `0xF`.
- Gabungkan hasilnya dengan bitwise OR: Setelah mendapatkan 4 bit dari kedua operasi di atas, kita gabungkan hasilnya dengan operator OR.

3. Referensi yang Digunakan

https://chatgpt.com/share/671b96dd-5660-8004-b8bc-ab0b1253870e

Soal 2 – masquerade

1. Solusi

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

2. Penjelasan Singkat

- 1 << 31 menghasilkan 0x80000000, yaitu angka terkecil.
- Kemudian, dengan operasi XOR (^ 1), kita menambah 1 pada nilai tersebut, sehingga hasil akhirnya adalah 0x80000001, yang merupakan angka terkecil kedua.

3. Referensi yang Digunakan

https://chatgpt.com/share/671c7dac-c4f8-8004-b088-5bef3e2efc09

Soal 3 – airaniiofifteen

1. Solusi

```
int airaniiofifteen(int iofi){
    int b0 = iofi & 1;
    int b1 = (iofi >> 1) & 1;
    int b2 = (iofi >> 2) & 1;
    int b3 = (iofi >> 3) & 1;
    int b4plus = !(iofi >> 4);

    return (b0 & b1 & b2 & b3) & b4plus;
}
```

2. Penjelasan Singkat

- b0, b1, b2, dan b3: Kamu mengambil 4 bit paling kanan dari iofi, satu per satu, dengan operasi shift dan AND. Ini benar karena angka 15 dalam biner adalah 1111, jadi bit ke-0 hingga ke-3 semuanya harus 1 jika iofi == 15.
- b4plus: Ini memeriksa apakah bit ke-4 dan seterusnya adalah 0. Kamu menggunakan !(iofi >> 4) untuk memastikan tidak ada bit yang tersisa setelah bit ke-3. Jika ada bit yang tersisa, maka hasilnya akan 0.
- Return statement: Kamu menggunakan AND untuk menggabungkan semua bit yang diperiksa. Jika semua bit (b0, b1, b2, b3) adalah 1 dan b4plus benar (bit setelah bit ke-3 adalah 0), maka hasil akhirnya adalah 1. Jika salah satu dari bit tersebut tidak sesuai, hasilnya adalah 0.

3. Referensi yang Digunakan

https://chatgpt.com/share/671c7eee-b434-8004-988c-9c54e0f6d333

Soal 4 – yobanashi_deceive

1. Solusi

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

2. Penjelasan Singkat

- Operasi shift right (>>) pada bilangan bulat secara efektif membagi nilai tersebut dengan pangkat dua.
- Shift right 1 kali (>> 1) setara dengan membagi bilangan dengan 2.
- Untuk membagi eksponen dengan 4 (dua kali sqrt), kita harus melakukan shift right sebanyak 2 kali, yaitu >> 2.
- Tapi jika kita ingin membulatkan ke bawah lebih agresif untuk memastikan nilai terdekat ke sqrt(sqrt(f)), kita menggunakan shift right 3 kali (>> 3) untuk hasil yang lebih konservatif, sesuai dengan contoh yang diberikan.

3. Referensi yang Digunakan

https://chatgpt.com/share/671c8019-1a74-8004-b875-363a38790b4c

Soal 5 – snow_mix

1. Solusi

```
int snow_mix(int N){
    int addend = 1 << 23;
    int sum = N ^ addend;
    int carry = (N & addend) << 1;

    int new_sum = sum ^ carry;
    int new_carry = (sum & carry) << 1;

    return new_sum ^ new_carry;
}
```

2. Penjelasan Singkat

- addend = 1 << 23 benar karena ini menghasilkan 2^23.
- sum = N ^ addend bekerja untuk menjumlahkan N dengan addend tanpa carry.
- carry = (N & addend) << 1 menghitung carry pertama kali, tetapi jika ada carry baru dari hasil sum, kamu butuh lebih dari satu iterasi untuk menghitungnya, yang akan melanggar batas maksimal 14 operasi.

3. Referensi yang Digunakan

https://chatgpt.com/share/671c890e-a978-8004-bae7-c2079b0525e7

Soal 6 – sky_hundred

1. Solusi

```
int sky_hundred(int n) {

    int is_negative = (n >> 31) & 1;
    int mod4 = n & 3;
    int modand = mod4 & 1;
    int mods = mod4 >> 1;
    int is_mod0 = !(mod4);
    int is_mod1 = (modand) & ~(mods);
    int is_mod2 = ((mods) & 1) & ~(modand);

    int result = (n & (~is_mod0 + 1)) | 1 &(~is_mod1 + 1) |((n
+ 1) & (~is_mod2 + 1));
    int shift = ~((is_negative << 31) >> 31);

    return (result & ~is_negative) & shift;
}
```

2. Penjelasan Singkat

- Cek apakah n negatif: Menggunakan bit shift untuk mengambil bit tanda (MSB) dan menyimpan hasilnya di is_negative.
- Hitung n % 4: Menggunakan bitwise AND (n & 3) untuk mendapatkan hasil sisa pembagian 4 (mod4).
- Tentukan hasil XOR berdasarkan mod4: Menggunakan logika bitwise untuk memeriksa apakah mod4 adalah 0, 1, atau 2, dan menghasilkan nilai sesuai pola XOR dari 1 hingga n.
- Tangani kasus negatif: Jika n negatif, kode mengatur hasil akhir menjadi 0 dengan operasi bitwise di variabel shift dan is_negative.

3. Referensi yang Digunakan

https://chatgpt.com/share/671c8d36-8f2c-8004-ae26-7a7bef3cc7cd

Soal 7 – ganganji

1. Solusi

```
int ganganji(int x) {
    int part1 = (x >> 3) << 3;
    int part1b = part1 + (x >> 3);
    int part2 = (x & 7) << 3;
    int part2b = part2 + (x & 7);
    int result = part1b + (part2b >> 3);
    int max_int = ~(1 << 31);
    int overflow = (result >> 31) & 1;
    int overflow2 = ~overflow + 1;
    return (overflow2 & max_int) | (~overflow2 & result);
}
```

2. Penjelasan Singkat

part1 dan part2:

- part1 menghitung pembagian x dengan 8 lalu dikalikan kembali dengan 8 untuk mendapatkan kelipatan 8 terdekat dari x.
- part1b menambahkan hasil pembagian x dengan 8 untuk mendekati nilai yang dikalikan 1.125.
- part2 menangani sisa dari x yang tidak bisa dibagi 8 secara sempurna.
- part2b mengelola bagian dari x yang tersisa dan mengalikannya dengan 1.125.

result:

- Menggabungkan hasil dari dua bagian (part1b dan part2b) untuk mendekati hasil dari x * 1.125.

Overflow check:

- Mengecek apakah hasilnya melebihi batas maksimal int (0x7FFFFFFF) dengan melihat bit ke-31.
- Jika overflow terjadi, hasilnya diubah menjadi 0x7FFFFFFF.

Return value:

- Menggunakan bitwise untuk memilih antara hasil asli atau 0x7FFFFFFF jika overflow terdeteksi.
- Kode ini menangani overflow dengan

3. Referensi yang Digunakan

https://chatgpt.com/share/671c8c47-c094-8004-b1d9-3e557344cbc3

Soal 8 – kitsch

1. Solusi

```
int kitsch(int x) {
    int a = x & 63;
    int b = x >> 31;
    int c = ((a + ((x & 3) << 4)) >> 6) + (b & !(!a));
    int d = ((x >> 6) + (x >> 2)) + c;
    return d;
}
```

2. Penjelasan Singkat

- Ambil 6 bit terakhir dari x (a = x & 63): Ini mengekstrak bit-bit yang relevan untuk mengelola pembagian dengan 64. Ini seperti mengambil nilai modulus 64 dari x.
- Ambil tanda dari x (b = x >> 31): Jika x negatif, hasilnya adalah -1; jika positif, hasilnya 0. Ini digunakan untuk koreksi pembulatan negatif.
- Hitung koreksi pembulatan (c = ((a + ((x & 3) << 4)) >> 6) + (b & !(!a))):
    - Menambahkan hasil bit paling kecil dari x untuk memperbaiki perhitungan.
    - (b & !(!a)) memastikan pembulatan yang benar saat x negatif, jika perlu dilakukan koreksi saat angka tidak nol.
- Perhitungan akhir (d = ((x >> 6) + (x >> 2)) + c):
    - Membagi x dengan 64 (x >> 6) dan menambahkan pembagian lain dengan 4 (x >> 2) untuk menghitung hasil perkalian 17/64 secara efisien.
    - Koreksi pembulatan dari langkah sebelumnya (c) ditambahkan ke hasil ini

3. Referensi yang Digunakan

https://chatgpt.com/share/671c8ea2-2d4c-8004-9a4c-defd24aa4b36

Soal 9 – how_to_sekai_seifuku

1. Solusi

```
unsigned how_to_sekai_seifuku(unsigned a) {
    unsigned b = (a >> 15) & 1;
    unsigned c = (a >> 10) & 0x1F;
    unsigned d = a & 0x03FF;
    unsigned e = b << 31;

    if (c == 0) {
        if (d == 0) {
            return e;
        }
        while ((d & 0x0400) == 0) {
            d = d << 1;
            c = c - 1;
        }
        c = c + 1;
        d = d & 0x03FF;
    } else if ((c & 0x1F) == 0x1F) {
        if ((d & 0x3FF) == 0) {
            return e | 0x7F800000;
        }
        return 0x7F800001;
    }

    c = c + (127 - 15);
    e = e | (c << 23);
    e = e | (d << 13);

    return e;
}
```

2. Penjelasan Singkat

Ekstraksi:

- b: Ambil sign bit.
- c: Ambil exponent (5-bit).
- d: Ambil mantissa (10-bit).

Kasus Zero/Denormalized:

- Jika c == 0 dan d == 0, return e (sign bit).
- Jika denormalized, normalisasi mantissa dengan while loop.

Kasus Inf/NaN:

- Jika c == 0x1F, return +inf, -inf, atau 0x7F800001 untuk NaN.

Normalisasi:

- Adjust bias (c += 127 - 15).
- Gabungkan sign, exponent, dan mantissa dalam e.

3. Referensi yang Digunakan

https://chatgpt.com/share/671c913c-9818-8004-9709-31538ee26140

Soal 10 – mesmerizer

1. Solusi

```
int mesmerizer(unsigned uf) {
    unsigned a = uf >> 31;
    unsigned b = (uf >> 23) & 0xFF;
    unsigned c = uf & 0x7FFFFF;
    int d;
    int e;

    if (b == 0xFF) {
        return 0x80000000;
    }
    if (b < 127) {
        return 0;
    }

    c |= 0x800000;
    d = b - 127;
    if (d > 31) {
        return 0x80000000;
    }

    if (d >= 23) {
        e = c << (d - 23);
    } else {
        e = c >> (23 - d);
    }

    if (a) {
        e = -e;
    }

    return e;
}
```

2. Penjelasan Singkat

- a: Mengambil bit sign (positif/negatif).
- b: Ekstrak exponent.
- c: Ekstrak mantissa dan tambahkan bit tersembunyi.

- NaN/infinity: Jika exponent maksimum (b == 0xFF), kembalikan 0x80000000.
- Nilai kecil: Jika exponent lebih kecil dari 127, kembalikan 0.
- d: Hitung exponent yang telah dinormalisasi.
- Overflow: Jika exponent terlalu besar (d > 31), kembalikan 0x80000000.
- e: Hitung nilai integer dengan shift mantissa sesuai exponent.
- Sign: Jika negatif, ubah nilai menjadi negatif.
- Return: Kembalikan hasil sebagai integer.

3. Referensi yang Digunakan

https://chatgpt.com/share/671c92bd-e470-8004-9878-54931cd30761