Imię i nazwisko: TOMASZ WIERZBICKI Numer indeksu: 666

Architektury systemów komputerowych

21 czerwca 2016

czas trwania: 150–180 minut

Punkty	Ocena
90% – 100%	5.0
80% – 89%	4.5
70% – 79%	4.0
60% – 69%	3.5
50% – 59%	3.0
0% – 49%	2.0

Uwagi do zadań 1–2: Należy używać wyłącznie operatorów arytmetyczno-logicznych, przypisania, stałych i dodatkowych zmiennych. **Zabrania się** korzystania z instrukcji sterowania, w tym instrukcji i operatora warunkowego, oraz operatorów mnożenia, dzielenia i modulo. **Nie można** dopuścić do wystąpienia nadmiaru i niedomiaru!

Zadanie 1 (6). Przetłumacz poniższą funkcję na procedurę języka `C` o sygnaturze `int num(unsigned)`.

$$\mathrm{num}\,(n) = \begin{cases} i & \text{dla } n = 2 \cdot i \\ -i & \text{dla } n = 2 \cdot i + 1 \end{cases}$$

```
int num(unsigned n) {
    return ((n >> 1) ^ (-(n & 1))) + (n & 1);
}
```

Zadanie 2 (6). Przetłumacz poniższą funkcję na procedurę języka `C` o sygnaturze `int avg(int, int)`.

$$\mathrm{avg}\,(x, y) = \left\lfloor \frac{x + y}{2} \right\rfloor$$

```
int avg(int x, int y) {
    return (x >> 1) + (y >> 1) + (x & y & 1);
}
```

Zadanie 3 (8). Przetłumacz kod procedury `foobar` z asemblera x86-64 do języka C. **Należy posłużyć się** wysokopoziomowymi instrukcjami sterującymi. Używanie etykiet i instrukcji `goto` jest **niedozwolone**.

```
1  foobar:
2          mov    %rdi, %rax
3          xor    %rdx, %rdx
4  .L2:    cmp    $0, (%rax)
5          je     .L7
6          jg     .L3
7          dec    %rdx
8          jmp    .L4
9  .L3:    inc    %rdx
10 .L4:    add    $8, %rax
11         jmp    .L2
12 .L7:    mov    %rdx, (%rsi)
13         ret
```

```
long *foobar (long *x, long *y) {
    *y = 0;
    while (*x != 0)
        if (*(x++) > 0)
            (*y)++;
        else (*y)--;
    return x;
}
```

Zadanie 4 (10). Przeczytaj poniższy kod w języku C i odpowiadający mu kod w asemblerze x86-64, po czym wywnioskuj rozmiar struktur `strA` i `strB` oraz wartość stałych `N` i `M`. W kratce poniżej należy umieścić **zwięzły** opis wnioskowania prowadzący do odpowiedzi.

```
1  typedef struct {
2    int  s[M];
3    long z;
4  } strA;
5
6  typedef struct {
7    strA  t[N];
8    long  k;
9    short x;
10   short y;
11 } strB;
12
13 long foo(strB *bp, long i) {
14   return bp[bp->k].t[i].z;
15 }
```

```
1  foo:
2      movq   120(%rdi), %rax
3      movq   %rax, %rdx
4      salq   $7, %rdx
5      leaq   (%rdx,%rax,8), %rax
6      addq   %rax, %rdi
7      leaq   (%rsi,%rsi,2), %rax
8      movq   16(%rdi,%rax,8), %rax
9      ret
```

Na początku obliczeń %rdi = bp i %rsi = i.
Wiersz 2 asemblera: %rax = *(bp+120)
odpowiada obliczeniu bp[0].k = bp->k, zatem
offset zmiennej k w strukturze strB wynosi
120, a skoro sizeof(long) dzieli 120,
to nie ma w tym miejscu paddingu i
sizeof(strA) * N = 120.
Wiersz 4-6 asemblera:
%rdi += (128+8) * %rax = 136 * (bp->k)
odpowiada obliczeniu bp[bp->k], stąd
sizeof(strB) = 136.
W wierszach 7-8 asemblera wyliczamy adres
%rdi + 3*8 * %rsi + 16 = %rdi + 24*i + 16
potrzeby do obliczenia bp[bp->k].t[i].z,
więc sizeof(strA) = 24 a offset zmiennej
z w strukturze strA wynosi 16. Stąd
M = 4, a skoro sizeof(strA) * N = 120, to N = 5.