Architektury systemów komputerowych

Lista zadań nr 1

Na zajęcia 4 marca 2021

W zadaniu 1 i 2 wolno używać **wyłącznie** instrukcji przypisania, operatorów bitowych, dodawania, odejmowania i przesunięć bitowych. Wszystkie zmienne mają typ `uint32_t`. Można używać zmiennych tymczasowych.

Zadanie 1. Zmienne i, k spełniają warunek $0 \leq i,\ k \leq 31$. Napisz fragment kodu, który skopiuje i-ty bit zmiennej x na pozycję k-tą.

Zadanie 2. Napisz fragment kodu, który wyznaczy liczbę zapalonych bitów w zmiennej x.

UWAGA! Oczekiwana złożoność to $O(\log n)$, gdzie n to liczba bitów w słowie. Posłuż się strategią *„dziel i zwyciężaj"*.

Zadanie 3. Podaj rozmiar w bajtach poniższych struktur przyjmując, że wskaźnik jest 32-bitowy. Pod jakim przesunięciem, względem początku struktury, znajdują się poszczególne pola? Jak zreorganizować pola struktury, by zajmowała mniej miejsca? Z czego wynika takie zachowanie kompilatora?

```
1 struct A {
2   int8_t a;
3   void *b;
4   int8_t c;
5   int16_t d;
6 };
```

```
1 struct B {
2   uint16_t a;
3   double b;
4   void *c;
5 };
```

Wskazówka: Użyj kompilatora, aby się dowiedzieć jaki jest rozmiar powyższych struktur – przyda się słowo kluczowe «`sizeof`».

Zadanie 4. Rozważamy słowa kluczowe ze standardu C11 (a nie C++). Jakie jest działanie «`volatile`» w stosunku do zmiennych? Kiedy programiści muszą go użyć, by program zachowywał się poprawnie? Jaki jest skutek użycia «`static`» w stosunku do zmiennych globalnych, zmiennych lokalnych i procedur? Kiedy należy go używać? Jaką rolę pełni «`restrict`» odnośnie typów wskaźnikowych?

Wskazówka: W przypadku «`volatile`» nie chodzi o wyłączenie optymalizacji!

Zadanie 5. Zmienne «a», «b» i «c» to wskaźniki na tablice elementów typu «`uint32_t`». Przetłumacz, krok po kroku, poniższe dwie instrukcje złożone zapisane w języku C na **kod trójkowy**:

```
s += b[j+1] + b[--j];          a[i++] -= *b * (c[j*2] + 1);
```

UWAGA! Przyjmujemy, że wszystkie wyrażenia są obliczane od lewej do prawej.

Zadanie 6. Z punktu widzenia procesora wszystkie wskaźniki są tożsame z liczbami całkowitymi. W trakcie generowania kodu wynikowego kompilator musi przetłumaczyć instrukcje wyboru pola struktury lub wariantu unii «`x->k`» i «`x.k`» oraz indeksowania tablic «`a[i]`» na prostsze instrukcje.

Przetłumacz, krok po kroku, poniższą instrukcję zapisaną w języku C na kod trójkowy. Trzeba pozbyć się typów złożonych, wykonać odpowiednie obliczenia na wskaźnikach, a wszystkie dostępy do pamięci realizować wyłącznie instrukcjami «`x:=*y`» lub «`*x:=y`». Zmienne «`us`» i «`vs`» są typu «`struct A *`» (patrz zad. 3).

```
vs->d = us[1].a + us[j].c;
```

Zadanie 7. Przetłumacz, krok po kroku, poniższą procedurę napisaną w języku C na kod trójkowy:

```
1  void insertion_sort(int arr[], int length) {
2    int j, temp;
3    for (int i = 0; i < length; i++) {
4      j = i;
5      while (j > 0 && arr[j] < arr[j-1]) {
6        temp = arr[j];
7        arr[j] = arr[j-1];
8        arr[j-1] = temp;
9        j--;
10     }
11   }
12 }
```

Następnie oznacz **bloki podstawowe** i narysuj **graf przepływu sterowania** (ang. *control flow graph*).

Zadanie 8. Być może jest to zaskakujące, ale poniższy kod jest poprawny i w dodatku czasami korzysta się z tej niskopoziomowej techniki optymalizacji. Co robi procedura «`secret`»?

```
1  void secret(uint8_t *to, uint8_t *from, size_t count) {
2    size_t n = (count + 7) / 8;
3    switch (count % 8) {
4    case 0: do { *to++ = *from++;
5    case 7:      *to++ = *from++;
6    case 6:      *to++ = *from++;
7    case 5:      *to++ = *from++;
8    case 4:      *to++ = *from++;
9    case 3:      *to++ = *from++;
10   case 2:      *to++ = *from++;
11   case 1:      *to++ = *from++;
12             } while (--n > 0);
13   }
14 }
```

Kompilator `GCC` dopuszcza by instrukcja «`goto`» przyjmowała wyrażenie obliczające adres skoku. Dodatkowo umożliwia definiowanie tablic etykiet[1]. Przetłumacz powyższą procedurę tak, by korzystała wyłącznie z instrukcji «`goto`».

[1] `https://gcc.gnu.org/onlinedocs/gcc/Labels-as-Values.html`