Architektury systemów komputerowych 2017

Lista zadań nr 1

Na zajęcia 1 – 2 marzec 2017

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, j 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 poniższych struktur (w bajtach) 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?

```
struct A {
  int8_t a;
  void *b;
  int8_t c;
  int16_t d;
};
```

```
struct B {
  uint16_t a;
  double b;
  void *c;
};
```

Zadanie 4. Jakie jest działanie słowa kluczowego `volatile` w stosunku do zmiennych? Kiedy programiści muszą go użyć, by program zachowywał się poprawnie?

Zadanie 5. Z punktu widzenia procesora wszystkie wskaźniki mają taki sam typ, który zachowuje się jak «`void *`». W trakcie generowania kodu wynikowego kompilator musi przetłumaczyć operacje na wskaźnikach na instrukcje proste. Sygnatury instrukcji dostępu do pamięci dla typu «`int8_t`» podano niżej. Dla innych typów maszynowych sygnatury wyglądają analogicznie.

```
int8_t load_I8(void *ptr); // załaduj wartość typu «int8_t» spod adresu «ptr»
void   store_I8(void *ptr, int8_t val); // zapisz wartość «val» pod adres «ptr»
```

Przetłumacz poniższy kod tak, by nie występowały w nim operatory wyłuskania «`*x`», wyboru pola struktury «`x->k`» i «`x.k`» oraz indeksowania tablic «`a[i]`». Innymi słowy – należy zrzutować wskaźniki «`us`» i «`vs`» na typ «`void *`», a następnie posługując się operatorami arytmetycznymi obliczyć wskaźniki dla instrukcji `load` i `store`.

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

Zadanie 6. Ponieważ instrukcje procesora posiadają co najwyżej dwa argumenty, to w trakcie generowania kodu wynikowego kompilator tłumaczy wszystkie złożone wyrażenia do postaci «`c = a + b`» lub «`x = *ptr`» wprowadzając zmienne tymczasowe.

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

Dla powyższych instrukcji wykonaj, krok po kroku, następujący szereg transformacji:

- przetłumacz indeksowanie tablic na obliczenia na wskaźnikach: `a[i]` $\rightarrow$ `*(a + i)`,
- przetłumacz operatory z przypisaniem: `a += b` $\rightarrow$ `a = a + b`,
- przetłumacz operatory inkrementacji i dekrementacji: `a++` $\rightarrow$ `a = a + 1`,
- przypisz wszystkim wynikom pośrednim zmienne tymczasowe.

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

Zadanie 7. W trakcie generowania kodu wynikowego kompilator musi uprościć wszystkie złożone instrukcje sterujące do instrukcji procesora, tj. skoków warunkowych i bezwarunkowych.

```
void insertion_sort(int arr[], int length) {
  int j, temp;
  for (int i = 0; i < length; i++) {
    j = i;
    while (j > 0 && arr[j] < arr[j-1]) {
      temp = arr[j];
      arr[j] = arr[j-1];
      arr[j-1] = temp;
      j--;
    }
  }
}
```

Dla powyższej procedury wykonaj, krok po kroku, następujący szereg transformacji:

- «`for`» $\rightarrow$ «`while`» oraz «`while`» $\rightarrow$ «`if`» i «`goto`»,
- «`if (cond) then A else B`» $\rightarrow$ «`if (b) goto label`», gdzie b jest wartością logiczną.

Następnie dla powyższej procedury narysuj **graf kontroli 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`»?

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

Kompilator `GCC` dopuszcza by instrukcja «`goto`» przyjmowała wartość wskaźnika i można było zdefiniować tablicę etykiet[1]. Przetłumacz powyższą procedurę tak, by nie występowały w niej złożone instrukcje sterujące.

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