Architektury systemów komputerowych 2016

Lista zadań nr 1

Na zajęcia 29 luty – 3 marzec 2016

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

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

Zadanie 2. Podaj fragment kodu, który obliczy ilość zapalonych bitów w zmiennej x.

Zadanie 3. Jaki jest rozmiar, w bajtach, poniższych struktur przy założeniu, 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 jego użycie jest niezbędne?

Zadanie 5. Dla procesora wszystkie wskaźniki mają typ `void *`. W trakcie generowania asemblera kompilator musi przetłumaczyć wszystkie operacje na wskaźnikach. Mamy funkcje o sygnaturze:

```
T load<sizeof(T)>(void *Ptr);       // z adresu Ptr załaduj zmienną typu T
store<sizeof(T)>(void *Ptr, T Val); // zachowaj wartość Val pod adres Ptr
```

... gdzie `T` jest typem maszynowym. W poniższym kodzie przetłumacz wszystkie wskaźniki do `void *` i zastąp wszystkie instrukcje dostępu do pamięci odwołaniami do funkcji `load` i `store`.

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

Zadanie 6. Ponieważ wszystkie instrukcje procesora posiadają, w uproszczeniu, co najwyżej dwa argumenty, to wszystkie niejawne wartości tymczasowe muszą zostać nazwane. W poniższym kodzie:

- zmień indeksowanie tablic na obliczenia na wskaźnikach tj. `a[i]` $=$ `*(a + i)`,
- przypisz wszystkim wynikom tymczasowym zmienne.

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

Zadanie 7. W trakcie generowania asemblera kompilator musi uprościć wszystkie wyrażenia kontroli przepływu – tj. `for`, `while`, `if` – do kodu przypominającego instrukcje procesora. W poniższym kodzie zastąp wyrażenie:

- `for` przy pomocy `while`,
- `while` przy pomocy `if` i `goto`,
- `if (b) then A else B` przy pomocy `goto` i `if (b) goto label`, gdzie b jest wartością typu `bool`.

```
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--;
    }
  }
}
```

Zadanie 8. Być może jest to zaskakujące, ale poniższy kod jest prawidłowy i czasem korzysta się z tej techniki. Wytłumacz co robi? Przyjmijmy, że `goto` może przyjąć oprócz etykiety wartość i można zrobić tablicę etykiet. Jak zatem można przetłumaczyć poniższe wyrażenie `switch`?

```
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);
  }
}
```