Systemy komputerowe

Lista zadań nr 3

Na zajęcia 14 – 17 marca 2019

Przy tłumaczeniu kodu w asemblerze `x86-64` do języka `C` należy trzymać się następujących wytycznych:

- Nazwy wprowadzonych zmiennych muszą opisywać ich zastosowanie, np. `result` zamiast `rax`.
- Instrukcja `goto` jest zabroniona w finalnym rozwiązaniu. Należy używać instrukcji sterowania `if`, `for` lub `while`, preferując użycie `for`.

Sposób przekazywania argumentów, zwracania wartości funkcji i dyscyplina stosu utrzymywana przez kod asemblerowy zgodna jest z konwencjami z rodziału 3 CSAPP3e. Innymi słowy kod jest zgodny z System-V/x86-64 ABI.

Zadanie 1. Rejestry `%reg1%` i `%reg2` są tego samego rozmiaru. Wykaż, że niezależnie od zapisanych w nich wartości, interpretowanych jako liczby ze znakiem, instrukcja `cmp %reg1, %reg2` ustawia flagi tak, że `setl %reg3` zadziała zgodnie z oczekiwaniami. Podobnie, wykaż, że jeśli te wartości intepretujemy jako liczby bez znaku, to `setb %reg3` zadziała zgodnie z oczekiwaniami. Wywnioskuj stąd, że pozostałe instrukcje rodziny `set` działają stosownie do swoich sufiksów.

Zadanie 2. Poniżej znajduje się kod funkcji o sygnaturze «`void who(short v[], size_t n)`». Przetłumacz go na język C i odpowiedz, jaki jest efekt jego wykonania. Czy znajomość sygnatury jest istotna?

```
1  who:    subq    $1, %rsi
2          movl    $0, %eax
3  .L2:    cmpq    %rsi, %rax
4          jnb     .L4
5          leaq    (%rdi,%rax,2), %rcx
6          movzwl  (%rcx), %r8d
7          leaq    (%rdi,%rsi,2), %rdx
8          movzwl  (%rdx), %r9d
9          movw    %r9w, (%rcx)
10         movw    %r8w, (%rdx)
11         addq    $1, %rax
12         subq    $1, %rsi
13         jmp     .L2
14 .L4:    ret
```

Zadanie 3. Poniżej znajduje się kod funkcji o sygnaturze «`bool zonk(char* a, char* b)`», jako argumenty przyjmującej C-owe łańcuchy znaków. Przetłumacz ją na język C (bez instrukcji `goto`). Jaką wartość powinna liczyć ta funkcja? Zauważ, że dla pewnych poprawnych argumentów jej działanie jest niezdefiniowane i napraw poniższy kod.

```
1  zonk:   movl    $0, %ecx
2  .L2:    movslq  %ecx, %rax
3          movzbl  (%rdi,%rax), %edx
4          testb   %dl, %dl
5          je      .L6
6          cmpb    %dl, (%rsi,%rax)
7          jne     .L5
8          addl    $1, %ecx
9          jmp     .L2
10 .L6:    orb     (%rsi,%rax), %dl
11         sete    %al
12         ret
13 .L5:    movl    $0, %eax
14         ret
```

Zadanie 4. Poniżej znajduje się kod funkcji o sygnaturze «`foo(int16_t v[], size_t n)`». Przetłumacz ją na język C. Narysuj ramkę stosu tej funkcji i wytłumacz, jaka jest rola poszczególnych komórek ramki oraz jak jej zawartość zmienia się w trakcie działania. Jaki jest efekt ma ten kod?

```
foo:    pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -24(%rbp)
        movq    %rsi, -32(%rbp)
        movq    $0, -8(%rbp)
        jmp     .L2
.L3:    movq    -8(%rbp), %rax
        leaq    (%rax,%rax), %rdx
        movq    -24(%rbp), %rax
        addq    %rdx, %rax
        movzwl  (%rax), %eax
        leal    (%rax,%rax), %edx
        movq    -8(%rbp), %rax
        leaq    (%rax,%rax), %rcx
        movq    -24(%rbp), %rax
        addq    %rcx, %rax
        movw    %dx, (%rax)
        addq    $1, -8(%rbp)
.L2:    movq    -8(%rbp), %rax
        cmpq    -32(%rbp), %rax
        jb      .L3
        nop
        popq    %rbp
        ret
```

Wskazówka: Instrukcja nop to tzw. ńo operation", nie ma efektu poza przejściem do wykonania kolejnej instrukcji kodu

Zadanie 5. Poniżej znajduje się kod funkcji rekurencyjnej o nieznanej sygnaturze. Przetłumacz tę funkcję na język C, odkryj jej sygnaturę i odpowiedz, jaką wartość ona liczy.

```
reccur:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        cmpl    $0, -4(%rbp)
        jne     .L2
        movl    $1, %eax
        jmp     .L3
.L2:    movl    -4(%rbp), %eax
        subl    $1, %eax
        movl    %eax, %edi
        call    reccur
        imull   -4(%rbp), %eax
.L3:    leave
        ret
```

Wskazówka Instrukcja leave podstawia %rbp pod %rsp wykonuje popq %rbp.

Zadanie 6. Dana jest funkcja o sygnaturze postaci «`int32_t bar(int32_t a1,...,int32_t an)`», gdzie `n` jest nieznane. Jaka jest minimalna wartość `n`, jeżeli wiadomo, że funkcja zwraca wartość jednego ze swoich argumentów, a jej kod wygląda tak

```
bar:    pushq   %rbp
        movq    %rsp, %rbp
            .....
        movl    16(%rbp), %eax
        popq    %rbp
        ret
```

Napisz szkic kodu asemblerowego wywołującego funkcję `bar` z liczbą parametrów równą takiemu minimalnemu `n`. Zadbaj o poprawne przekazanie argumentów do funkcji. Jak zmieni się napisany przez Ciebie kod, gdy `n` będzie większe?

Zadanie 7. Dana jest funkcja o sygnaturze «`int16_t you(int8_t index)`» i fragmencie kodu podanym poniżej. Funkcja ta została skompilowana z flagą `-O0`, a jej kod asemblerowy również jest podany. Nieznana jest natomiast funkcja «`int16_t set_secret(void)`». Jaki argument należy podać wywołując `you`, by odkryć wartość sekretu?

```
int16_t you(int8_t index) {
struct {
   int16_t tbl[5];
   int8_t tmp;
   int16_t secret;
} str;

str.secret = set_secret();
...
return str.tbl[index];
}
```

```
you:    pushq   %rbp
        movq    %rsp, %rbp
        subq    $24, %rsp
        movl    %edi, %eax
        movb    %al, -20(%rbp)
        call    set_secret
        movw    %ax, -2(%rbp)
        ...
        movsbl  -20(%rbp), %eax
        cltq
        movzwl  -14(%rbp,%rax,2), %eax
        leave
        ret
```

Wskazówka: Instrukcja cltq rozszerza rejestr %eax do %rax zachowując znak. Pamiętaj, że zadeklarowane zmienne muszą być odpowiednio wyrównane.

Zadanie 8. Przeczytaj poniższy kod w języku C i odpowiadający mu kod w asemblerze, a następnie wywnioskuj jakie są wartości stałych «A» i «B».

```
1  typedef struct {
2    int32_t x[A][B];
3    int64_t y;
4  } str1;
5
6  typedef struct {
7    int8_t array[B];
8    int32_t t;
9    int16_t s[A];
10   int64_t u;
11 } str2;
12
13 void set_val(str1 *p, str2 *q) {
14   int64_t v1 = q->t;
15   int64_t v2 = q->u;
16   p->y = v1 + v2;
17 }
```

```
18 set_val:
19   movslq 8(%rsi),%rax
20   addq   32(%rsi),%rax
21   movq   %rax,184(%rdi)
22   ret
```

Wskazówka: Deklaracja int32_t x[A][B] powoduje, że x będzie A-elementową tablicą wartości typu int32_t [B]. Pamiętaj o wyrównaniu pól w strukturach.