# Systemy komputerowe

Lista zadań nr 4

Na zajęcia zdalne 30 marca – 2 kwietnia 2020

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 książki 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(int16_t 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 liczy ta funkcja?

```
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.** Zastąp instrukcje `pushq %reg1` oraz `popq %reg2` równoważnymi ciągami instrukcji jawnie operującymi na stosie.

**Zadanie 5.** 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? Jaka jest rola rejestru %rbp w tym kodzie?

```
1  foo:    pushq   %rbp
2          movq    %rsp, %rbp
3          movq    %rdi, -24(%rbp)
4          movq    %rsi, -32(%rbp)
5          movq    $0, -8(%rbp)
6          jmp     .L2
7  .L3:    movq    -8(%rbp), %rax
8          leaq    (%rax,%rax), %rdx
9          movq    -24(%rbp), %rax
10         addq    %rdx, %rax
11         movzwl  (%rax), %eax
12         leal    (%rax,%rax), %edx
13         movq    -8(%rbp), %rax
14         leaq    (%rax,%rax), %rcx
15         movq    -24(%rbp), %rax
16         addq    %rcx, %rax
17         movw    %dx, (%rax)
18         addq    $1, -8(%rbp)
19 .L2:    movq    -8(%rbp), %rax
20         cmpq    -32(%rbp), %rax
21         jb      .L3
22         nop
23         popq    %rbp
24         ret
```

**Wskazówka:** Instrukcja nop to tzw. 'no operation', nie ma efektu poza przejściem do wykonania kolejnej instrukcji kodu

**Zadanie 6.** Poniżej znajduje się kod funkcji rekurencyjnej o nieznanej sygnaturze. Przetłumacz tę funkcję na język C, odkryj jej sygnaturę i odpowiedz, jaką wartość ona liczy.

```
1  reccur:
2          pushq   %rbp
3          movq    %rsp, %rbp
4          subq    $16, %rsp
5          movl    %edi, -4(%rbp)
6          cmpl    $0, -4(%rbp)
7          jne     .L2
8          movl    $1, %eax
9          jmp     .L3
8  .L2:    movl    -4(%rbp), %eax
9          subl    $1, %eax
10         movl    %eax, %edi
11         call    reccur
12         imull   -4(%rbp), %eax
13 .L3:    leave
14         ret
```

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

**Zadanie 7.** 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

```
1  bar:    pushq   %rbp
2          movq    %rsp, %rbp
3              .....
4          movl    16(%rbp), %eax
5          popq    %rbp
6          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?