Kolokwium MRJP 4.12.2014

Zadanie 1. Przetłumacz poniższą funkcję na LLVM

```
.method public static f1([II)V
  .limit stack 3
  .limit locals 5
  0: iconst_0
  1: istore_2
  2: iload_2
  3: istore_3
Label4:
  4: iload_3
  5: iload_1
  6: if_icmpge Label47
  9: iload_2
  10: iload_1
  11: if_icmpge Label20
  14: aload_0
  15: iload_2
  16: iaload
  17: goto Label21
Label20:
  20: iconst_0
Label21:
  21: istore 4
  23: iload_2
  24: iinc 2 1
  27: iload_1
  28: if_icmpge Label36
  31: iload 4
  33: ifeq Label44
Label36:
  36: aload_0
  37: iload_3
  38: iinc 3 1
  41: iload 4
  43: iastore
Label44:
  44: goto Label4
Label47:
  47: return
.end method
```

Zadanie 2. Przetłumacz poniższą funkcję na asembler x86, przyjmując standardowy protokół wywołania funkcji.

```
define void @f2(i32* %t, i32 %n) {
  %1 = alloca i32*
  %2 = alloca i32
  %i = alloca i32
  %j = alloca i32
  %x = alloca i32
  store i32* %t, i32** %1
  store i32 %n, i32* %2
  %3 = load i32* %2
  store i32 %3, i32* %j
  store i32 %3, i32* %i
  br label %4

; <label>:4
  %5 = load i32* %i
  %6 = icmp sgt i32 %5, 0
  br i1 %6, label %7, label %20

; <label>:7
  %8 = load i32* %i
  %9 = add  i32 %8, -1
  store i32 %9, i32* %i
  %10 = load i32** %1
  %11 = getelementptr i32* %10, i32 %9
  %12 = load i32* %11
  store i32 %12, i32* %x
  %13 = icmp ne i32 %12, 0
  br i1 %13, label %14, label %4

; <label>:14
  %15 = load i32* %x
  %16 = load i32* %j
  %17 = add  i32 %16, -1
  store i32 %17, i32* %j
  %18 = load i32** %1
  %19 = getelementptr i32* %18, i32 %17
  store i32 %15, i32* %19
  br label %4

; <label>:20
  ret void
}
```

Zadanie 3.
Przetłumacz poniższą funkcję na JVM

```
    .globl  f3
f3:
.LFB2:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $16, %esp
    movl    12(%ebp), %eax
    movl    %eax, -4(%ebp)
    movl    $0, -8(%ebp)
    jmp .L13
.L20:
    movl    -8(%ebp), %eax
    leal    0(,%eax,4), %edx
    movl    8(%ebp), %eax
    addl    %edx, %eax
    movl    (%eax), %eax
    testl   %eax, %eax
    jne .L14
    movl    $0, -12(%ebp)
    jmp .L15
.L17:
    movl    -4(%ebp), %eax
    leal    0(,%eax,4), %edx
    movl    8(%ebp), %eax
    addl    %edx, %eax
    movl    (%eax), %eax
    movl    %eax, -12(%ebp)
    cmpl    $0, -12(%ebp)
    je  .L15
    jmp .L16
.L15:
    subl    $1, -4(%ebp)
    movl    -4(%ebp), %eax
    cmpl    -8(%ebp), %eax
    jg  .L17
.L16:
    cmpl    $0, -12(%ebp)
    jne .L18
    jmp .L12
.L18:
    movl    -8(%ebp), %eax
    leal    0(,%eax,4), %edx
    movl    8(%ebp), %eax
    addl    %eax, %edx
    movl    -12(%ebp), %eax
    movl    %eax, (%edx)
    movl    -4(%ebp), %eax
    leal    0(,%eax,4), %edx
    movl    8(%ebp), %eax
    addl    %edx, %eax
    movl    $0, (%eax)
.L14:
    addl    $1, -8(%ebp)
.L13:
    movl    -8(%ebp), %eax
    cmpl    12(%ebp), %eax
    jl  .L20
.L12:
    leave
    ret
```

Uwagi:
* `0(,%eax,4)` w notacji Intel zapisalibyśmy jako `[4*EAX]`
* w zadaniu 2 dopuszczalne jest użycie notacji AT&T lub Intel.
* każde zadanie proszę oddawać na osobnej kartce; zadania są niezależne, ale warto przeczytać wszystkie przed rozpoczęciem rozwiązywania