

# Architektury systemów komputerowych

## Lista zadań nr 6

Na zajęcia 8–9 kwietnia 2019

**UWAGA!** Przy tłumaczeniu kodu z asemblera x86–64 do języka C należy trzymać się poniższych wytycznych:

- Używaj złożonych wyrażeń minimalizując liczbę zmiennych tymczasowych.
- Nazwy wprowadzonych zmiennych muszą opisywać ich zastosowanie, np. «result» zamiast «%rax».
- Należy używać instrukcji sterowania «if», «for», «while» i «switch»; «goto» jest zabronione!
- Pętle «while» należy przetłumaczyć do pętli «for», jeśli poprawia to czytelność kodu.

**Zadanie 1.** 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     int x[A][B];
3     long y;
4 } str1;
5
6 typedef struct {
7     char array[B];
8     int t;
9     short s[A];
10    long u;
11 } str2;
12 void set_val(str1 *p, str2 *q) {
13     long v1 = q->t;
14     long v2 = q->u;
15     p->y = v1 + v2;
16 }
17
18 1 set_val:
19 2     movslq 8(%rsi),%rax
20 3     addq   32(%rsi),%rax
21 4     movq   %rax,184(%rdi)
22 5     ret
```

**Zadanie 2.** Przeczytaj poniższy kod w języku C i odpowiadający mu kod w asemblerze, a następnie wywnioskuj jakie są wartości stałych «R», «S» i «T».

```
1 long A[R][S][T];
2
3 long store_elem(long i, long j,
4                 long k, long *dest)
5 {
6     *dest = A[i][j][k];
7     return sizeof(A);
8 }
9 store_elem:
10 leaq  (%rsi,%rsi,2),%rax
11 leaq  (%rsi,%rax,4),%rax
12 movq  %rdi,%rsi
13 salq  $6,%rsi
14 addq  %rsi,%rdi
15 addq  %rax,%rdi
16 addq  %rdi,%rdx
17 movq  A(,%rdx,8),%rax
18 movq  %rax,(%rcx)
19 movq  $3640,%rax
20 ret
```

**Zadanie 3.** Przeczytaj poniższy kod w języku C i odpowiadający mu kod w asemblerze, a następnie wywnioskuj jaka jest wartość stałej «CNT» i jak wygląda definicja struktury «a\_struct».

```
1 typedef struct {
2     int first;
3     a_struct a[CNT];
4     int last;
5 } b_struct;
6
7 void test (long i, b_struct *bp) {
8     int n = bp->first + bp->last;
9     a_struct *ap = &bp->a[i];
10    ap->x[ap->idx] = n;
11 }
12 test:
13 movl  0x120(%rsi),%ecx
14 addl  (%rsi),%ecx
15 leaq  (%rdi,%rdi,4),%rax
16 leaq  (%rsi,%rax,8),%rax
17 movq  0x8(%rax),%rdx
18 movslq %ecx,%rcx
19 movq  %rcx,0x10(%rax,%rdx,8)
20 retq
```

**Zadanie 4.** Przeczytaj definicję unii «elem» i wyznacz jej rozmiar w bajtach. Następnie przepisz procedurę «proc» na kod w języku C.

```

1 union elem {
2     struct {
3         long *p;
4         long y;
5     } e1;
6     struct {
7         long x;
8         union elem *next;
9     } e2;
10 };
11 proc:
12     movq 8(%rdi),%rax
13     movq (%rax),%rdx
14     movq (%rdx),%rdx
15     subq 8(%rax),%rdx
16     movq %rdx,(%rdi)
17     ret

```

**Zadanie 5 (2).** Przeczytaj definicje struktur «SA» i «SB», po czym przeanalizuj kod procedur o sygnaturach «SB eval(SA s)» i «long wrap(long x, long y, long z)». Następnie zapisz w języku C kod odpowiadający procedurom «eval» i «wrap». Narysuj diagram przedstawiający zawartość rekordu aktywacji procedury «wrap» w momencie wywołania funkcji «eval».

```

1 typedef struct A {
2     long u[2];
3     long *v;
4 } SA;
5
6 typedef struct B {
7     long p[2];
8     long q;
9 } SB;
10 eval:
11     movq %rdi, %rax
12     movq 16(%rsp), %rcx
13     movq 24(%rsp), %rdx
14     movq (%rdx), %rsi
15     movq %rcx, %rdx
16     imulq %rsi, %rdx
17     movq %rdx, (%rdi)
18     movq 8(%rsp), %rdx
19     movq %rdx, %rdi
20     subq %rsi, %rdi
21     movq %rdi, 8(%rax)
22     subq %rcx, %rdx
23     movq %rdx, 16(%rax)
24     ret
25 wrap:
26     subq $72, %rsp
27     movq %rdx, (%rsp)
28     movq %rsp, %rdx
29     leaq 8(%rsp), %rax
30     pushq %rdx
31     pushq %rsi
32     pushq %rdi
33     movq %rax, %rdi
34     call eval
35     movq 40(%rsp), %rax
36     addq 32(%rsp), %rax
37     imulq 48(%rsp), %rax
38     addq $96, %rsp
39     ret

```

**Zadanie 6.** Poniżej widniej kod procedury o sygnaturze «float puzzle6(struct P \*, float)». Wyznacz definicję typu «struct P». Przetłumacz tę procedurę na język C i wyjaśnij jednym zdaniem co robi.

```

1 puzzle6:
2     movq (%rdi), %rdx
3     leaq 8(%rdi), %rcx
4     xorl %eax, %eax
5     vxorps %xmm1, %xmm1, %xmm1
6     vmovss .LC1(%rip), %xmm2
7 .L2: cmpq %rdx, %rax
8     jge .L5
9     vfmadd231ss (%rcx,%rax,4), %xmm2, %xmm1
10    incq %rax
11    vmulss %xmm0, %xmm2, %xmm2
12    jmp .L2
13 .L5: vmovaps %xmm1, %xmm0
14    ret
15
16 .LC1: .long 0x3f800000

```

**Zadanie 7.** Poniższy kod otrzymano w wyniku deasemblacji funkcji o sygnaturze «float puzzle7(float, float)». Przetłumacz tę procedurę na język C, po czym jednym zdaniem powiedz co ona robi.

```

1 puzzle7:
2     vmulss    .LC2(%rip), %xmm0, %xmm6
3     vroundss $1, %xmm6, %xmm6, %xmm6
4     vmulss    .LC3(%rip), %xmm6, %xmm6
5     vsubss    %xmm6, %xmm0, %xmm6
6     vcomiss   .LC4(%rip), %xmm6
7     jb       .L2
8     vsubss    .LC3(%rip), %xmm6, %xmm6
9 .L2:  vmovaps   %xmm6, %xmm5
10    vmovaps   %xmm6, %xmm0
11    vmovss    .LC0(%rip), %xmm3
12    vmovss    .LC1(%rip), %xmm4
13 .L4:  vmovaps   %xmm6, %xmm2
14    vxorps    .LC5(%rip), %xmm2, %xmm2
15    vmulss    %xmm2, %xmm6, %xmm2
16    vmulss    %xmm2, %xmm5, %xmm5
17    vaddss    .LC1(%rip), %xmm3, %xmm2
18    vmulss    %xmm2, %xmm3, %xmm2
19    vmulss    %xmm2, %xmm4, %xmm4
20    vaddss    .LC0(%rip), %xmm3, %xmm3
21    vdivss    %xmm4, %xmm5, %xmm2
22    vaddss    %xmm2, %xmm0, %xmm0
23    vandps    .LC6(%rip), %xmm2, %xmm2
24    vcomiss   %xmm1, %xmm2
25    ja       .L4
26    ret
27 .LC0: .single 2.0
28 .LC1: .single 1.0
29 .LC2: .single 0.159154936 # 1/(2*pi)
30 .LC3: .single 6.283185482 # 2*pi
31 .LC4: .single 3.141592741 # pi
32 .LC5: .long  0x80000000, 0, 0, 0
33 .LC6: .long  0x7fffffff, 0, 0, 0

```

**Wskazówka:** Znaczenie pierwszego operandu instrukcji «vroundsd» jest wyjaśnione w tabeli 4-18 zawartej w dokumencie „Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 2: Instruction Set Reference, A-Z” pod opisem instrukcji «ROUNDPD».