

Architektury systemów komputerowych

Lista zadań nr 5

Na zajęcia 6 kwietnia 2022

Uwaga! Należy być przygotowanym do wyjaśnienia semantyki każdej instrukcji, która pojawia się w treści zadania. W tym celu posłuż się dokumentacją: [x86 and amd64 instruction reference](http://www.felixcloutier.com/x86/)¹. W szczególności trzeba wiedzieć jak dana instrukcja korzysta z rejestru flag «EFLAGS» tam, gdzie obliczenia zależą od jego wartości.

W trakcie tłumaczeniu kodu z assemblera x86-64 do języka C należy trzymać się następujących wytycznych:

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

Zadanie 1. Zaimplementuj poniższą funkcję w assemblerze x86-64. Wartości «x» i «y» typu «uint64_t» są przekazywane przez rejestry `%rdi` i `%rsi`, a wynik zwracany w rejestrze `%rax`. Najpierw rozwiąż zadanie używając instrukcji skoku warunkowego. Potem przepisuj je używając instrukcji «`sbb`».

$$addu(x, y) = \begin{cases} \text{ULONG_MAX} & \text{dla } x + y \geq \text{ULONG_MAX} \\ x + y & \text{w p.p.} \end{cases}$$

Wskazówka! Rozwiązanie wzorcowe składa się z 3 instrukcji bez «`ret`».

Zadanie 2. Zaimplementuj funkcję zdefiniowaną poniżej w assemblerze x86-64. Taka procedura w języku C miałaby sygnaturę «`long cmp(uint64_t x, uint64_t y)`».

$$cmp(x, y) = \begin{cases} -1 & \text{gdy } x < y \\ 1 & \text{gdy } x > y \\ 0 & \text{gdy } x = y \end{cases}$$

Wskazówka: Rozwiązanie wzorcowe ma cztery wiersze (bez `ret`). Użyj instrukcji `adc`, `sbb` i `neg`.

Zadanie 3. Zapisz w języku C funkcję o sygnaturze «`int puzzle(long x, unsigned n)`» której kod w assemblerze podano niżej. Zakładamy, że parametr «n» jest nie większy niż 64. Przedstaw jednym zdaniem co robi ta procedura.

1	puzzle: testl %esi, %esi	8	sarq %rdi
2	je .L4	9	incl %edx
3	xorl %edx, %edx	10	cmpl %edx, %esi
4	xorl %eax, %eax	11	jne .L3
5	.L3: movl %edi, %ecx	12	ret
6	andl \$1, %ecx	13	.L4: movl %esi, %eax
7	addl %ecx, %eax	14	ret

Uwaga! Instrukcja zapisująca młodszą połowę 64-bitowego rejestru ustawia na 0 jego starszą połowę (brzydota x86-64).

Zadanie 4. Poniżej zamieszczono kod procedury o sygnaturze «`long puzzle2(char *s, char *d)`». Wyznacz *bloki podstawowe* oraz narysuj *graf przepływu sterowania*. Przetłumacz tę procedurę na język C, a następnie jednym zdaniem powiedz co ona robi.

¹<http://www.felixcloutier.com/x86/>

```

1 puzzle2:                                10      cmpb  %c1, %r9b
2      movq  %rdi, %rax                    11      jne   .L2
3  .L3:  movb  (%rax), %r9b                 12      movq  %r8, %rax
4      leaq  1(%rax), %r8                  13      jmp   .L3
5      movq  %rsi, %rdx                    14  .L4:  subq  %rdi, %rax
6  .L2:  movb  (%rdx), %c1                  15      ret
7      incq  %rdx
8      testb %c1, %c1
9      je    .L4

```

Zadanie 5 (2). Poniżej widnieje kod funkcji o sygnaturze «uint32_t puzzle3(uint32_t n, uint32_t d)». Wyznacz bloki podstawowe oraz narysuj graf przepływu sterowania, po czym przetłumacz tę funkcję na język C. Na podstawie ustępu „Mixing C and Assembly Language” strony [GNU Assembler Examples²](http://cs.lmu.edu/~ray/notes/gasexamples/) napisz i zaprezentuj działanie programu, który pomógł Ci odpowiedzieć na pytanie co ta funkcja robi.

```

1 puzzle3:                                11      orl   %ecx, %eax
2      movl  %edi, %edi                    12      movq  %r8, %rdi
3      salq  $32, %rsi                    13  .L2:  shr   %ecx
4      movl  $32, %edx                     14      decl  %edx
5      movl  $0x80000000, %ecx             15      jne   .L3
6      xorl  %eax, %eax                    16      ret
7  .L3:  addq  %rdi, %rdi
8      movq  %rdi, %r8
9      subq  %rsi, %r8
10     js    .L2

```

Zadanie 6 (2). Poniżej zamieszczono kod rekurencyjnej procedury o sygnaturze «int puzzle4(long *a, long v, uint64_t s, uint64_t e)». Wyznacz bloki podstawowe oraz narysuj graf przepływu sterowania. Przetłumacz tę procedurę na język C, a następnie jednym zdaniem powiedz co ona robi.

```

1 puzzle4:                                11      cmpq  %rsi, %r8
2      movq  %rcx, %rax                    12      jg    .L11
3      subq  %rdx, %rax                    13      leaq  1(%rax), %rdx
4      shrq  %rax                          14      call  puzzle4
5      addq  %rdx, %rax                    15  .L10:  ret
6      cmpq  %rdx, %rcx                    16  .L11:  leaq  -1(%rax), %rcx
7      jb    .L5                           17      call  puzzle4
8      movq  (%rdi,%rax,8), %r8             18      ret
9      cmpq  %rsi, %r8                     19  .L5:  movl  $-1, %eax
10     je    .L10                          20      ret

```

Wskazówka: Z reguły procedurę «puzzle4» woła się następująco: «i = puzzle4(a, v, 0, n - 1)».

Zadanie 7 (2). Poniższy kod w asemblerze otrzymano w wyniku deasemblacji funkcji zadeklarowanej jako «long switch_prob(long x, long n)». Zapisz w języku C kod odpowiadający tej funkcji.

1 400590 <switch_prob>:			
2 400590: 48 83	subq \$0x3c,%rsi		
3 400594: 48 83 fe 05	cmpq \$0x5,%rsi		
4 400598: 77 29	ja *0x4005c3		
5 40059a: ff 24 f5 f8 06 40 00	jmpq *0x4006f8(%rsi,8)		Zrzut pamięci przechowującej tablicę skoków:
6 4005a1: 48 8d 04 fd 00 00 00 00	lea 0x0(,%rdi,8),%rax		
7 4005a9: c3	retq		
8 4005aa: 48 89 f8	movq %rdi,%rax		18 (gdb) x/6gx 0x4006f8
9 4005ad: 48 c1 f8 03	sarq \$0x3,%rax		19 0x4006f8: 0x4005a1
10 4005b1: c3	retq		20 0x400700: 0x4005a1
11 4005b2: 48 89 f8	movq %rdi,%rax		21 0x400708: 0x4005b2
12 4005b5: 48 c1 e0 04	shlq \$0x4,%rax		22 0x400710: 0x4005c3
13 4005b9: 48 29 f8	subq %rdi,%rax		23 0x400718: 0x4005aa
14 4005bc: 48 89 c7	movq %rax,%rdi		24 0x400720: 0x4005bf
15 4005bf: 48 0f af ff	imulq %rdi,%rdi		
16 4005c3: 48 8d 47 4b	leaq 0x4b(%rdi),%rax		
17 4005c7: c3	retq		

²<http://cs.lmu.edu/~ray/notes/gasexamples/>