

Problem 1. *C's "short-circuit evaluation" for conditionals.

* Generate x86-64 assembly code for the following boxed fragments of C code, obeying C's semantics for && and ||. You are given the location of the variables. If the condition evaluates to TRUE (respectively, FALSE), have the generated code branch to the label Ltrue (respectively, Lfalse).

```
1  if((year%4==0&&year%100!=0)||year%400==0)
2  year in %rdi
3
4  # year%4==0
5  movl %rdi, %eax # number must be stored at %rax
6  movl $4, %edx # divisor
7  divl %edx # quotient in rax, remainder in rdx
8  movl %edx, %eax
9  testl %eax, %eax # %eax & %eax
10 jne Lfalse1
11 movl $1, %ebx # must be true now
12
13 # year%100!=0
14 movl %rdi, %eax
15 movl $100, %edx
16 divl %edx
17 movl %edx, %eax
18 text %eax, %ebx # result 2 & result 1
19 je Ltrue2
20 movl $0, %ebx # must be false now
21
22
23 Lfalse1:
24
25 # year%400==0
26 movl %rdi, %eax
27 movl $400, %edx
28 divl %edx
29 movl %edx, %eax
30 testl %eax, %eax
31 jne Lfalse1
32
33
34 Ltrue2:
35 Ltrue:
36 // if block
```

```

37
38 Lfalse2:
39 Lfalse:
40 // else
41

```

```

1  if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n')
2  s (char *) in %rdi
3  n (int) in %rbx
4
5  # s[n] != ' '
6  movlsq %rbx, %rax #[Q: check when to use what type (q or l)]
7  addq %rdi, %rax
8  movzbq (%rax), %rax #[Q: use ]
9  cmpb $32, %al
10 jne Lfalse
11
12 # s[n] != '\t'
13 movlsq %rbx, %rax #[Q: check when to use what type (q or l)]
14 addq %rdi, %rax
15 movzbq (%rax), %rax #[Q: use ]
16 cmpb $9, %al
17 je Lfalse
18
19 # s[n] != '\n'
20 movlsq %rbx, %rax #[Q: check when to use what type (q or l)]
21 addq %rdi, %rax
22 movzbq (%rax), %rax #[Q: use ]
23 cmpb $10, %al
24 je Lfalse
25
26 Ltrue:
27
28 Lfalse:
29
30

```

```

1  if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
2  p (char *) in %r11
3  allocbuf (static char *) in %r12
4

```

```

5  # p >= allocbuf
6  movq %r12, %rax
7  compq %r11, %rax
8  jb Lfalse
9
10 # p < allocbuf + ALLOCSIZE
11 movq %r12, %rax
12 addq $ALLOCSIZE, %rax
13 compq %r11, %rax
14 jnb Lfalse
15
16 Ltrue:
17
18 Lfalse:

```

```

1  if (p >= p->s.ptr && (bp > p || bp < p->s.ptr))
2  p (Mystery *) in %r8
3  bp (Mystery *) in %r9
4
5  # p >= p->s.ptr
6  movq %r8, %rax
7  movq (%rax), %rax
8  cmpq %r8, %rax
9  jb Lfalse
10
11 # bp > p
12 movq %r8, %rax
13 cmpq %r9, %rax
14 jbe Ltrue
15
16 # bp < p->s.ptr
17 movq %r8, %rax
18 movq (%rax), %rax
19 cmpq %r9, %rax
20 jnb Lfalse
21
22 Ltrue:
23
24 Lfalse:

```

Problem 2. *Stack frame for a complicated procedure.

```
1 extern int _foo1(int a, int b, int c, int d, bool w, bool x, bool y, bool z);
2 extern int _foo2(int a, int b, int c, int d, bool w, bool x);
3 extern int _foo3(int a, int b, int c, int d);
4
5 int foo(int argc, int argv[]) {
6     switch (argc) {
7         case 4: return _foo3(argv[0], argv[1], argv[2], argv[3]);
8         case 6: return _foo2(argv[0], argv[1], argv[2], argv[3], argv[4], argv[5]);
9         case 8: return _foo1(argv[0], argv[1], argv[2], argv[3], argv[4], argv[5],
10            argv[6], argv[7]);
11         default: return -1;
```

(a) Show the layout of the stack frame for foo. Indicate each of the four areas of the stack frame, how much each area takes, and the total size of the stack frame. [Q: what are 4 areas of stack frame? Aren't they describing assembly code blocks rather than stack frame of the function itself?]

```
1 Prologue:
2 old rbp --4
3 (vars all in register)
4
5 Pre-call:
6 1-8 parameters: put all in stack (n*4)
7
8 Post-return:
9 (NA)
10
11 Epilogue:
12 ret addr --8
```

(b) Generate x86-64 assembly code for foo, following the code generation templates discussed earlier. Remember that you do not know the internal structure of the procedures foo1, _foo2, and _foo3.

```
1 # int foo(int argc, int argv[]) {
2 # argc in rdi, argv in rsi
3 pushq %rbp
4 movq %rsp, %rbp
5 pushq %rbx
6 movl %rsi, %rax # make space rsi for future arguments
7 # switch (argc) {
8 #     case 4: return _foo3(argv[0], argv[1], argv[2], argv[3]);
```

```

9  cmpl $4, %rdi
10 jne Lpass1
11 movl (%rax), %rdi #arg0
12 movl 4(%rax), %rsi #arg1
13 movl 8(%rax), %rdx #arg2
14 movl 12(%rax), %rcx #arg3
15 call Q
16
17
18 Lpass1:
19 #    case 6: return _foo2(argv[0], argv[1], argv[2], argv[3], argv[4], argv[5]);
20 cmpl $6, %rdi
21 jne Lpass2
22 movl (%rax), %rdi #arg0
23 movl 4(%rax), %rsi #arg1
24 movl 8(%rax), %rdx #arg2
25 movl 12(%rax), %rcx #arg3
26 movl 16(%rax), %r8 #arg4
27 movl 20(%rax), %r9 #arg5
28 call Q
29
30 Lpass2:
31 #    case 8: return _foo1(argv[0], argv[1], argv[2], argv[3], argv[4], argv[5],
32 #    argv[6], argv[7]);
32 cmpl $8, %rdi
33 jne Lpass2
34 movl (%rax), %rdi #arg0
35 movl 4(%rax), %rsi #arg1
36 movl 8(%rax), %rdx #arg2
37 movl 12(%rax), %rcx #arg3
38 movl 16(%rax), %r8 #arg4
39 movl 20(%rax), %r9 #arg5
40 movl 24(%rax), %r11
41 movl %r11, %rbp #arg6
42 movl 28(%rax), %r11
43 movl %r11, %rbp #arg7
44 call Q
45
46 Lpass3:
47 #    default: return -1;
48 movq $-1, %rax
49 popq %rbx
50 popq %rbp
51 retq

```

(c) How can you make the generated code more compact?

```

1  # int foo(int argc, int argv[]) {
2  # argc in rdi, argv in rsi
3  pushq %rbp
4  movq %rsp, %rbp
5  pushq %rbx
6  movl %rsi, %rax # make space rsi for future arguments
7  movl %rdi, %rbx
8
9  # switch (argc) {
10 #   case 4: return _foo3(argv[0], argv[1], argv[2], argv[3]);
11 movl (%rax), %rdi #arg0
12 movl 4(%rax), %rsi #arg1
13 movl 8(%rax), %rdx #arg2
14 movl 12(%rax), %rcx #arg3
15 cmpl $4, %rbx
16 jne Lpass1
17 call Q
18
19
20 Lpass1:
21 #   case 6: return _foo2(argv[0], argv[1], argv[2], argv[3], argv[4], argv[5]);
22 movl 16(%rax), %r8 #arg4
23 movl 20(%rax), %r9 #arg5
24 cmpl $6, %rbx
25 jne Lpass2
26 call Q
27
28 Lpass2:
29 #   case 8: return _foo1(argv[0], argv[1], argv[2], argv[3], argv[4], argv[5],
30 #   argv[6], argv[7]);
31 movl 24(%rax), %r11
32 movl %r11, %rbp #arg6
33 movl 28(%rax), %r11
34 movl %r11, %rbp #arg7
35 cmpl $8, %rbx
36 jne Lpass3
37 call Q
38
39 Lpass3:
40 #   default: return -1;
41 movq $-1, %rax
42 popq %rbx
43 popq %rbp
44 retq

```

Problem 3. Register allocation, straight-line code.

Perform register allocation by graph coloring for the following code.

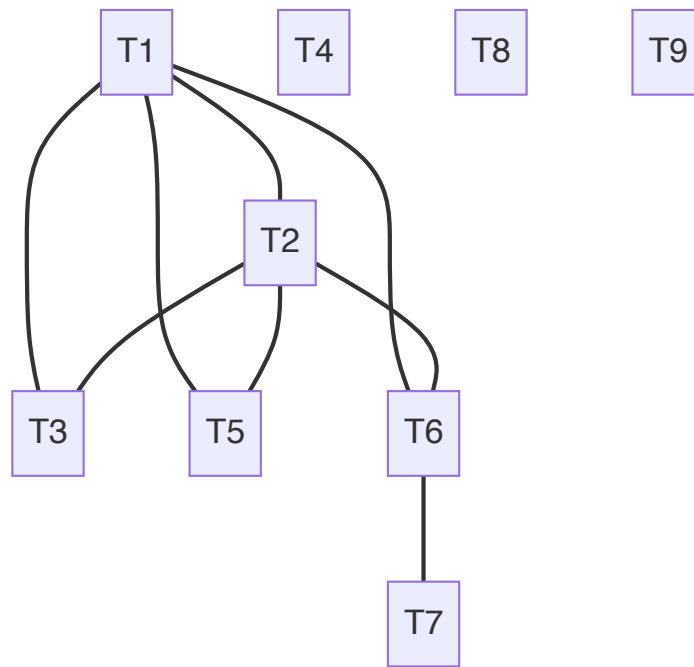
```
x = 2
y = 4;
w = x + y
z = x + 1
u = x * y
x = z * 2
```

(1)

(a) Rewrite it with symbolic registers substituted for the variables.

```
1  T1 := x
2  {1}
3  T2 := y
4  {1, 2}
5  T3 := T1 + T2
6  {1, 2, 3}
7  T4 := w
8  {1, 2, 3}
9  T4 = T3
10 {1, 2}
11 T5 := T1 + 1
12 {1, 2, 5}
13 T6 := z
14 {1, 2, 5}
15 T6 = T5
16 {1, 2, 6}
17 T7 := T1 * T2
18 {6, 7}
19 T8 := u
20 {6, 7}
21 T8 = T7
22 {6}
23 T9 := T6 * 2
24 {9}
25 T1 = T9
```

(b) Draw the interference graph for the rewritten code.



(c) Show an allocation for it with three registers, assuming that variables y and w are dead on exit from this code.

```

1  T1 := x      1: a
2  {1}
3  T2 := y      1: a, 2: b
4  {1, 2}
5  T3 := T1 + T2  1: a, 2: b, 3: c
6  {1, 2, 3}
7  T4 := w
8  {1, 2, 3}
9  T4 = T3      1: a, 2: b, 4: c
10 {1, 2}
11 T5 := T1 + 1  1: a, 2: b, 5: c
12 {1, 2, 5}
13 T6 := z
14 {1, 2, 5}
15 T6 = T5      1: a, 2: b, 6: c
16 {1, 2, 6}
17 T7 := T1 * T2  1: a, 7: b, 6: c
18 {6, 7}
19 T8 := u
20 {6, 7}
21 T8 = T7      1: a, 8: b, 6: c
22 {6, 7}

```



```

22 {0}
23 T9 := T6 * 2    9: a, 8: b, 6: c
24 {9}
25 T1 = T9        9: a (x), 8: b (u), 6: c (z)

```

Problem 4. *Register allocation with control flow.

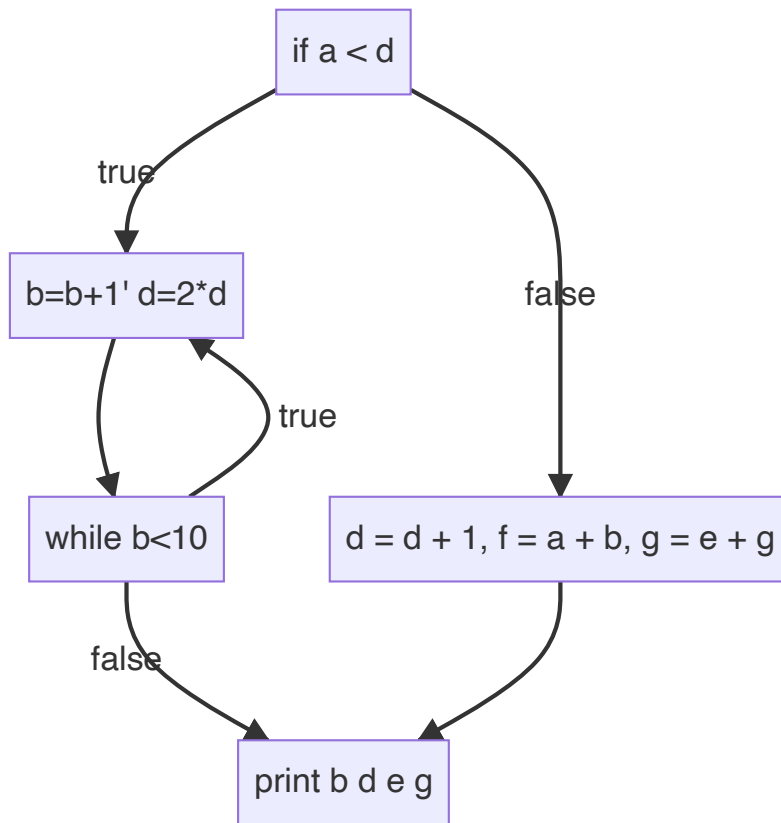
* Perform register allocation by graph coloring for the following program

```

1  a = 2;
2  b = 3;
3  d = c;
4  e = a;
5  g = c + 1;
6  if (a < d) {
7      do {
8          b = b + 1;
9          d = 2 * d;
10     } while (b < 10);
11 } else {
12     d = d+1;
13 f = a + b;
14 g = e + g; }
15 print(b, d, e, g);

```

(a) Draw the control flow graph for this program.



(b) Rewrite it with symbolic registers substituted for the variables.

```

1  1a 2b 3c 4d 5e 7g 12f
2
3  T1 := a = 2;
4  T2 := b = 3;
5  T3 := c;
6  T4 := d;
7  T4 := T3;
8  T5 := e;
9  T5 = T1
10 T6 := T3 + 1;
11 T7 := g;
12 T7 = T6
13
14 cmpq T1, T4
15 jnb Lelse
16
17 Lhead:
18 T8 := T2 + 1
19 T2 = T8

```

```

19  i2 = i0
20  T9 := 2 * T4
21  T4 = T9
22
23  cmp1 T2, $10
24  jb Lhead # while
25  jump Lend #exit while & if
26
27  Lelse:
28  T10 := T4 + 1
29  T4 = T10
30  T11 := T1 + T2
31  T12 := f
32  T12 = T11
33  T12 := T5 + T7
34  T13 := g
35  T13 = T12
36
37  Lend:
38  print(T2, T4, T5, T7);

```

(c) Draw the interference graph for the rewritten code.

```

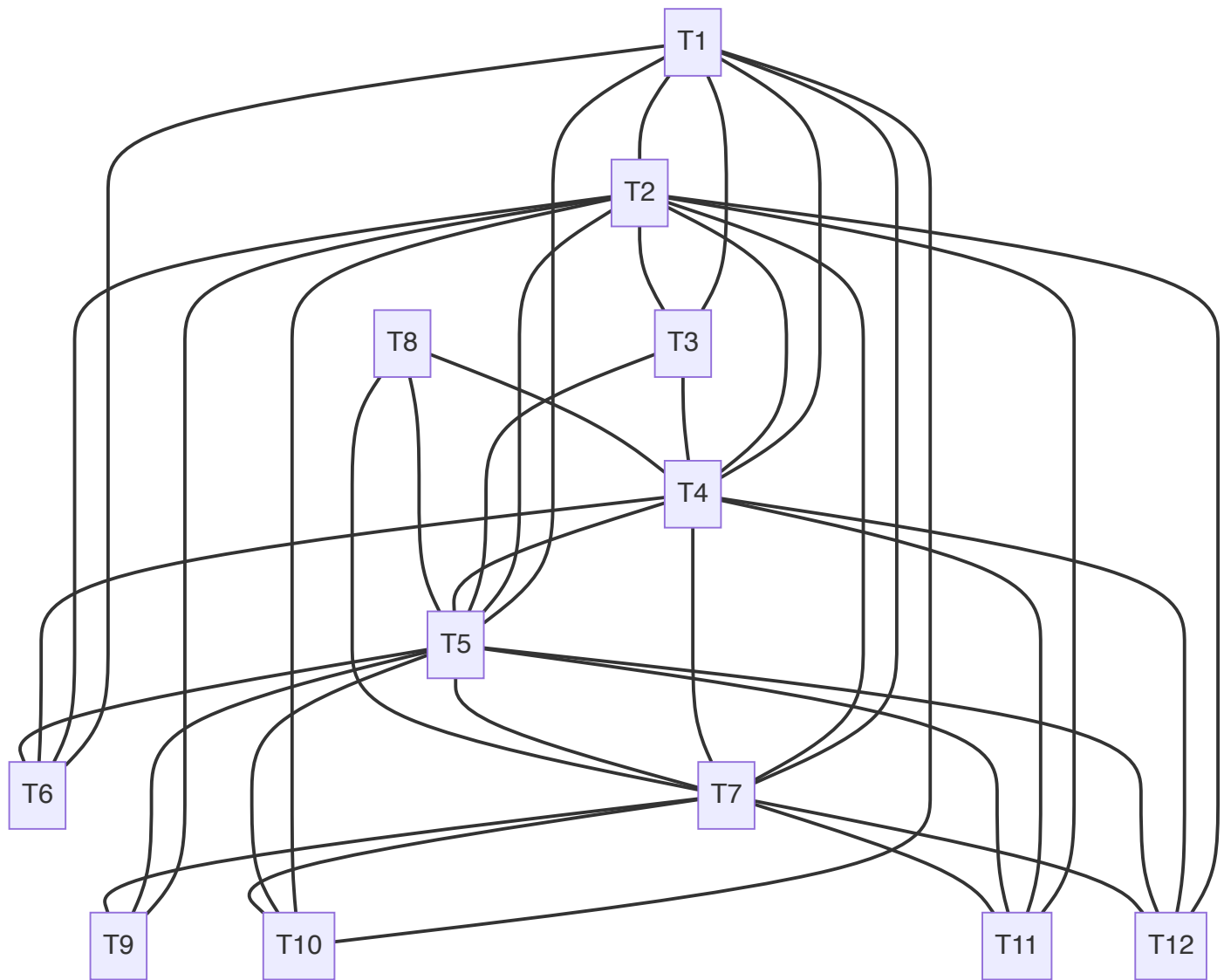
1  1a 2b 3c 4d 5e 7g 12f
2
3  T1 := a = 2;
4  {1}
5  T2 := b = 3;
6  {1, 2}
7  T3 := c;
8  {1, 2, 3}
9  T4 := d;
10 {1, 2, 3}
11 T4 := T3;
12 {1, 2, 3, 4}
13 T5 := e;
14 {1, 2, 3, 4}
15 T5 = T1
16 {1, 2, 3, 4, 5}
17 T6 := T3 + 1;
18 {1, 2, 4, 5, 6}
19 T7 := g;
20 {1, 2, 4, 5, 6}
21 T7 = T6
22

```

```

23 {1, 2, 4, 5, 7}
24 cmpq T1, T4
25 jnb Lelse
26
27 Lhead:
28 {2, 4, 5, 7}
29 T8 := T2 + 1
30 {8, 4, 5, 7}
31 T2 = T8
32 {2, 4, 5, 7}
33 T9 := 2 * T4
34 {2, 5, 7, 9}
35 T4 = T9
36
37 {2, 4, 5, 7}
38 cmpl T2, $10
39 jb Lhead # while
40 jump Lend #exit while & if
41
42 Lelse:
43 {1, 2, 4, 5, 7}
44 T10 := T4 + 1
45 {1, 2, 5, 7, 10}
46 T4 = T10
47 {1, 2, 4, 5, 7}
48 T11 := T1 + T2
49 {2, 4, 5, 7, 11}
50 T12 := f
51 {2, 4, 5, 7, 11}
52 T12 = T11
53 {2, 4, 5, 7}
54 T12 := T5 + T7
55 {2, 4, 5, 7, 12}
56 T13 := g
57 {2, 4, 5, 7, 12}
58 T13 = T12
59
60 Lend:
61 {2,4,5,7}
62 print(T2, T4, T5, T7);

```



(d) Show an allocation for it with five registers. (You will need to perform coalescing.)

```

1  1a 2b 3c 4d 5e 7g 12f
2
3  T1 := a = 2;
4  {1}                1: x
5  T2 := b = 3;
6  {1, 2}             1: x, 2: y
7  T3 := c;
8  {1, 2, 3}          1: x, 2: y 3: z
9  T4 := d;
10 {1, 2, 3}

```

```

11  T4 := T3;
12  {1, 2, 3, 4} 1: x, 2: y 3: z, 4: w
13  T5 := e;
14  {1, 2, 3, 4}
15  T5 = T1
16  {1, 2, 3, 4, 5} 1: x, 2: y 3: z, 4: w, 5: v
17  T6 := T3 + 1;
18  {1, 2, 4, 5, 6} 1: x, 2: y 6: z, 4: w, 5: v
19  T7 := g;
20  {1, 2, 4, 5, 6}
21  T7 = T6
22
23  {1, 2, 4, 5, 7} 1: x, 2: y 7: z, 4: w, 5: v
24  cmpq T1, T4
25  jnb Lelse
26
27  Lhead:
28  {2, 4, 5, 7} 1: x, 2: y 7: z, 4: w, 5: v
29  T8 := T2 + 1
30  {8, 4, 5, 7} 1: x, 8: y 7: z, 4: w, 5: v
31  T2 = T8
32  {2, 4, 5, 7} 1: x, 2: y 7: z, 4: w, 5: v
33  T9 := 2 * T4
34  {2, 5, 7, 9} 1: x, 2: y 7: z, 9: w, 5: v
35  T4 = T9
36
37  {2, 4, 5, 7} 1: x, 2: y 7: z, 4: w, 5: v
38  cmpl T2, $10
39  jb Lhead # while
40  jump Lend #exit while & if
41
42  Lelse:
43  {1, 2, 4, 5, 7} 1: x, 2: y 7: z, 4: w, 5: v
44  T10 := T4 + 1
45  {1, 2, 5, 7, 10} 1: x, 2: y 7: z, 10: w, 5: v
46  T4 = T10
47  {1, 2, 4, 5, 7} 1: x, 2: y 7: z, 4: w, 5: v
48  T11 := T1 + T2
49  {2, 4, 5, 7, 11} 11: x, 2: y 7: z, 4: w, 5: v
50  T12 := f
51  {2, 4, 5, 7, 11}
52  T12 = T11
53  {2, 4, 5, 7}
54  T12 := T5 + T7
55  {2, 4, 5, 7, 12} 12: x, 2: y 7: z, 4: w, 5: v
56  T13 := q

```

```
57 {2, 4, 5, 7, 12}
58 T13 = T12
59
60 Lend:
61 {2,4,5,7}
62 print(T2, T4, T5, T7);
```