All the codes and results in this document are available in the Github repository.

While

Initial code:

```
export fn add1(reg u64 arg) -> reg u64
{
    reg u64 z;
    reg bool temp;
    z = 0;
    temp = arg > 0;
    while(temp){
        z += 1;
        arg -= 1;
        temp = arg > 0;
    }
    return z;
}
```

After adding #init_msf and #update_msf:

```
export
fn add1 (reg u64 arg) -> (reg u64) {
  reg u64 msf;
  reg u64 z;
  reg bool temp;

  msf = #init_msf();
  z = 0;
  temp = arg > 0;
  while (temp) {
    msf = #update_msf(temp, msf);
    z = z + 1;
    arg = arg - 1;
    temp = arg > 0;
  }
  msf = #update_msf((! temp), msf);
  return (z);
}
```

The generated assembly code:

```
   .att_syntax
   .text
```

```
   .p2align 5
   .globl  _add1
   .globl  add1
_add1:
add1:
   movq  $0, %rax
   cmpq  $0, %rdi
   jmp   Ladd1$1
Ladd1$2:
   incq  %rax
   addq  $-1, %rdi
   cmpq  $0, %rdi
Ladd1$1:
   jnbe  Ladd1$2
   ret
```

If

Input code:

```
fn double_64(reg u64 inp) -> reg u64
{
    reg u64 dupl;
    dupl = 2 * inp;
    return dupl;
}

export fn add1(reg u64 arg) -> reg u64
{
    reg u64 z;
    reg bool temp;
    temp = arg > 0;
    if temp{
        z = double_64(arg);
    }
    else{
        z = double_64(arg);
        z = z + 1;
    }
    return z;
}
```

after slh update msf:

```
fn double_64 (reg u64 msf.102, reg u64 inp.174) -> (reg u64, reg u64) {
  reg u64 dupl.175;
```

```
3
4   dupl.175 = (((64u) 2) *64u inp.174); /* u64 */
5   return (msf.102, dupl.175);
6 }
7
8 export
9 fn add1 (reg u64 arg.171) -> (reg u64) {
10  reg u64 msf.102;
11  reg u64 z.172;
12  reg bool temp.173;
13
14  msf.102 = #init_msf();
15  temp.173 = (arg.171 >u ((64u) 0));
16  if temp.173 {
17    msf.102 = #update_msf(temp.173, msf.102);
18    (msf.102, z.172) = double_64(msf.102, arg.171);
19  } else {
20    msf.102 = #update_msf((! temp.173), msf.102);
21    (msf.102, z.172) = double_64(msf.102, arg.171);
22    z.172 = (z.172 +64u ((64u) 1));
23  }
24  return (z.172);
25 }
```

generated assembly code:

```
1    .att_syntax
2    .text
3    .p2align 5
4    .globl  _add1
5    .globl  add1
6 _add1:
7 add1:
8    movq %rsp, %rsi
9    andq $-8, %rsp
10   lfence
11   movq $0, %rax
12   cmpq $0, %rdi
13   jnbe Ladd1$1
14   movq $-1, %rcx
15   cmovnbe %rcx, %rax
16   call Ldouble_64$1
17 Ladd1$4:
18   incq %rax
19   jmp  Ladd1$2
20 Ladd1$1:
21   movq $-1, %rcx
```

```
   cmovbe  %rcx, %rax
   call  Ldouble_64$1
Ladd1$3:
Ladd1$2:
   movq  %rsi, %rsp
   ret
Ldouble_64$1:
   imulq $2, %rdi, %rax
   ret
```

Function call inside while

initial code:

```
fn double_64(reg u64 inp) -> reg u64
{
    reg u64 dupl;
    dupl = 2 * inp;
    return dupl;
}
export fn add1(reg u64 arg) -> reg u64
{
    reg u64 z;
    reg bool temp;
    z = 0;
    temp = arg > 0;
    while(temp){
        z += 1;
        z = double_64(z);
        arg -= 1;
        temp = arg > 0;
    }
    return z;
}
```

after slh_gen:

```
fn double_64 (reg u64 inp) -> (reg u64) {
  reg u64 dupl;

  dupl = (((64u) 2) *64u inp); /* u64 */
  return (dupl);
}

export
fn add1 (reg u64 arg) -> (reg u64) {
```

```
10    reg u64 z;
11    reg bool temp;
12
13    z = ((64u) 0); /* u64 */
14    temp = (arg >u ((64u) 0)); /* bool */
15    while (temp) {
16      z = (z +64u ((64u) 1)); /* u64 */
17      z = double_64(z);
18      arg = (arg -64u ((64u) 1)); /* u64 */
19      temp = (arg >u ((64u) 0)); /* bool */
20    }
21    return (z);
22  }
```