Flare-On 4: Challenge 11 Solution - covfefe.exe

Challenge Author: Nick Harbour (@nickharbour)

The covfefe.exe is a very small binary, weighing in at a meager 20KB. A lot of Flare-On Challenges will try form their complexity out of multiple layers of obfuscation and protection. `covfefe.exe` is an attempt to craft an intricate and clever single layer of protection.

When you run the binary from a command prompt you are presented with a simple password prompt, as shown in Figure 1.

```
C:\FlareOn>covfefe.exe
Welcome to the magic box.
Enter the password:
```

Figure 1: covfefe.exe Prompt

If you enter an incorrect password, you are presented with an obnoxious error message such as “The password is incorrect. Try reverse engineering the binary to discover the correct one.” or “Consider installing a disassembler such as IDA Pro, and try again, only harder next time.” If we look at the ASCII or Unicode strings in the covfefe binary we do not see these messages or the prompt we saw on screen. Figure 2 shows all the ASCII strings (minimum length 4) in covfefe.exe.

```
!This program cannot be run in DOS mode.
Richezh
.text
`.rdata
@.data
printf
scanf
srand
rand
time
msvcrt.dll
```

Figure 2: Strings in the covfefe.exe binary

To solve this challenge we should follow the snarky failure message’s advice and load the binary in IDA

Pro (or your other favorite disassembler). IDA correctly detects only three functions in the entire binary. If we look at the code at the entry point (address 401190) we see a function that calls a few imports from msvcrt.dll (time, srand, and rand) and then calls internal function 401070 before returning. Figure 3 shows the start function in its entirety.

```
.text:00401190 public start
.text:00401190 start proc near
.text:00401190  push    ebp
.text:00401191  mov     ebp, esp
.text:00401193  push    0                   ; Time
.text:00401195  call    time
.text:0040119A  add     esp, 4
.text:0040119D  push    eax                 ; Seed
.text:0040119E  call    srand
.text:004011A3  add     esp, 4
.text:004011A6  call    rand
.text:004011AB  mov     ecx, 4
.text:004011B0  imul    ecx, 110h
.text:004011B6  cdq
.text:004011B7  idiv    dword_403008[ecx]
.text:004011BD  mov     eax, 4
.text:004011C2  imul    ecx, eax, 111h
.text:004011C8  mov     dword_403008[ecx], edx
.text:004011CE  push    463h
.text:004011D3  push    1100h
.text:004011D8  push    offset dword_403008
.text:004011DD  call    sub_401070
.text:004011E2  add     esp, 0Ch
.text:004011E5  xor     eax, eax
.text:004011E7  pop     ebp
.text:004011E8  retn
.text:004011E8 start endp
```

Figure 3: Start Function

Inside the start function you'll notice three references to an array in global memory space at address 403008. We will discuss this array later, but for now just note that a pointer to it is being passed along with two fixed values (463h and 1100h) to function 401070. Let's examine that function.

Function 401070 is composed primarily of a loop. The constraint on this loop is the code block at address 40107C shown in Figure 4.

```
.text:0040107C loc_40107C:
.text:0040107C      mov     ecx, [ebp+var_8]
.text:0040107F      add     ecx, 3
.text:00401082      cmp     ecx, [ebp+arg_4]
.text:00401085      jg      loc_4
```

Figure 4: Function 401070 Loop Constraint

In this code fragment, what IDA Pro has labeled as arg_4 is the second parameter passed into this function, which was the constant 1100h. This value is the constraint that ends the loop if the loop counter in var_8 exceeds. There is another way to end the loop though. The code at address 4010CB compares some value with 0FFFFFFFFh (a.k.a. -1) and exits the loop if it matches.

You may also notice in this function that we see calls to printf and scanf. This is the only location in the program that calls them and both calls only write and read one character at a time, but they both occur inside the loop. What we are looking at in this function is the Virtual Machine apparatus for the real code that will be executing inside the VM the program creates. The I/O mechanism is primitive and only supports reading or writing one byte at a time but it is effective enough for the purpose of this challenge.

The block of code at 40108B executes every loop cycle and is responsible for calling the third and final function in the program. The first portion of this code block is shown in Figure 5.

```
.text:0040108B mov     edx, [ebp+var_8]
.text:0040108E mov     eax, [ebp+arg_0]
.text:00401091 mov     ecx, [eax+edx*4+8]
.text:00401095 push    ecx
.text:00401096 mov     edx, [ebp+var_8]
.text:00401099 mov     eax, [ebp+arg_0]
.text:0040109C mov     ecx, [eax+edx*4+4]
.text:004010A0 push    ecx
.text:004010A1 mov     edx, [ebp+var_8]
.text:004010A4 mov     eax, [ebp+arg_0]
.text:004010A7 mov     ecx, [eax+edx*4]
.text:004010AA push    ecx
.text:004010AB mov     edx, [ebp+arg_0]
.text:004010AE push    edx
.text:004010AF call    sub_401000
```

Figure 5: Portion of code block from 40108B

Careful inspection of this code block will show that the function 401000 is being passed a total of four arguments. The first argument is a pointer to the global memory array that was passed into this function (at offset 403008). The second, third, and fourth argument are DWORD elements from this array starting at the location of our loop counter var_8. So each time through the loop this function is passed 3 DWORDs from this array, as well as a pointer to the array itself. This function is very small and not difficult to reverse engineer. In Figure 6 I have chosen to represent this function with the output of the hex-rays decompiler plugin for IDA Pro, for simplicity.

```
int __cdecl sub_401000(int program[], int a, int b, int c)
{
  int result; // eax

  result = program[b] - program[a];
  program[b] = result;
  if ( c )
  {
    result = b;
    LOBYTE(result) = program[b] <= 0;
  }
  else
  {
    LOBYTE(result) = 0;
  }
  return result;
}
```

Figure 6: Decompilation of 401000

The function uses the second, third, and fourth arguments (which I have labeled in the snippet as a, b, and c) as indices into the array that was passed in as the first argument (which I have labeled as "program" in the snippet). The first action of this function is to subtract program[a] from program[b] and store the result in program[b]. If the result of that subtraction is less than or equal to zero, the program returns True, otherwise it returns False. This function is the implementation of the one and only instruction in our VM: Subtract and Branch if Less than or Equal to Zero, also known as "subleq" for short. This is a form of One Instruction Set Computer (OISC) that was invented to produce cheap hardware but is also quite useful for code obfuscation.

If we examine the function the parent function (401070) once again and focus on the code after the call to function 401000 beginning at offset 4010B7 we can see how the program uses the return value. In Figure 7 we can see the disassembly for the relevant code.

```
.text:004010B7      movzx   eax, al
.text:004010BA      test    eax, eax
.text:004010BC      jz      short loc_4010E0
.text:004010BE      mov     ecx, [ebp+var_8]
.text:004010C1      mov     edx, [ebp+program]
.text:004010C4      mov     eax, [edx+ecx*4+8]
.text:004010C8      mov     [ebp+var_C], eax
.text:004010CB      cmp     [ebp+var_C], 0FFFFFFFFh
.text:004010CF      jnz     short loc_4010D8
.text:004010D1      mov     al, 1
.text:004010D3      jmp     loc_401185
.text:004010D8 ; ---------------------------------------------------
.text:004010D8
.text:004010D8 loc_4010D8:                  ; CODE XREF: sub_401070+5F↑j
.text:004010D8      mov     ecx, [ebp+var_C]
.text:004010DB      mov     [ebp+var_8], ecx
.text:004010E0 ; ---------------------------------------------------
.text:004010E0
.text:004010E0 loc_4010E0:                  ; CODE XREF: sub_401070+4C↑j
.text:004010E0      mov     edx, [ebp+var_8]
.text:004010E3      add     edx, 3
.text:004010E6      mov     [ebp+var_8], edx
```

Figure 7: Code after call to 401000

The local variable labeled var_8 is what we noted earlier as being the loop counter, which at this point we can call the program counter since we now have some understanding of the data that this is looping over. If you have researched the subleq instruction by this point you will have noticed that the each subleq instruction takes three operands: the right side of the subtraction, the destination and left side of the subtraction, and the third operand is the location to jump to if the result is less than or equal to zero. If the return value from the subleq instruction handler function 401000 is zero then the code jumps to location 4010E0 which simply increments the program counter by three to point to the next instruction. If it was not zero, then third operand of the instruction is examined. If it is -1 then the loop terminates, indicating a program Halt operation. If it is any other value then it is used as the new program counter by copying it into the var_8 local variable at line 4010DB.

Each time the instruction loop occurs there is an opportunity for the program to display and read a character of input. The code block at 4010E9 determines if a 1 is stored in the program at offset 4. If it is then block of code at line 4010FA will print to the console the character value stored at offset 2 (converted from a DWORD to a CHAR). Input works in the same way, with the code block at line

401138 checking the value at offset 3 for the value 1, and reading a CHAR from standard input into offset 1 in the array if so. Offsets 3 and 4 in the program act as signal flags to let the VM know that the subleq program is requesting an input or output operation, and offsets 1 and 2 in the program array act as the holding locations for the data.

For simplicity of coding, all the strings in the subleq program are stored as a series of DWORDs, since there is currently no mechanism in the subleq VM implemented here to address anything other than a DWORD. This explains why none of our prompts or fail strings show up when we run the strings command.

Once you have an understanding of how the x86 portion of this challenge works you can begin working on the difficult part: understanding the subleq program itself. You will need to reverse engineer the subleq code to understand what it is doing with your input and what input will be required to get a success message.

Since the data that the subleq program operates on must also be in the same array as the operands for the subleq instruction itself (note that there is no need to store an opcode since there is only one instruction) we cannot attempt to blindly disassemble every set of three DWORDs and assume it is going to be a valid subleq instruction. A better approach would be to instrument the binary with a scriptable debugger to print out every subleq instruction as it is executed. Examining the stack to inspect the value of the program counter in var_8 as well as the arguments passed to function 401000 should be sufficient for this task. It would also be wise to dereference each operand and show what the value is that is pointed to by that operand. For example, if the program counter is 1130 and the three DWORDs we find in the program array at that location are as follows:

```
program[1130] = 1142
program[1131] = 0
program[1132] = 0
```

If the subleq VM executes at this program location then the instruction executed is effectively:

```
subleq 1142, 0, 0
```

You might be inclined to read this as “subtract 1142 from 0 and jump to 0 if the result is negative or zero” but you’d be wrong. All the operands in this machine are indirect addresses. This should really read as “subtract program[1142] from program[0] and jump to 0 if the result is negative or zero”. For this reason it is important to include the dereferenced value of each cell in the array for the line of disassembly you dump form your debugger (or pintool!) If we look at the value in cell 1142 in the

program array you will find the number 5. The value at memory cell 0 is also initially 0 but a key component of the subleq architecture is constantly changing the value of memory cell 0. It is a special cell and the closest thing to a register in the architecture. You can get a feel for constructing basic operations using nothing but subtraction in subleq through the documentation available on Wikipedia: https://en.wikipedia.org/wiki/One_instruction_set_computer#Subtract_and_branch_if_less_than_or_equal_to_zero

With our instrumented debugger printing out the offset, subleq instruction operands and their dereferenced values (in the form of a comment showing what is being subtracted from what) we should be able to trace through the execution of the program. Figure 8 shows the first several instructions executed by this program when disassembled in this fashion.

```
1123: subleq 0, 0, 1127    // 0 - 0
1127: subleq 1126, 1126, 0    // 0 - 0
1130: subleq 1142, 0, 0    // 5 - 0
1133: subleq 0, 1126, 0    // -5 - 0
1136: subleq 0, 0, 0    // -5 - -5
1139: subleq 0, 0, 1143    // 0 - 0
1143: subleq 1179, 1179, 0    // 0 - 0
1146: subleq 161, 0, 0    // 157 - 0
1149: subleq 0, 1179, 0    // -157 - 0
1152: subleq 0, 0, 0    // -157 - -157
1155: subleq 1180, 1180, 0    // 0 - 0
1158: subleq 161, 0, 0    // 157 - 0
1161: subleq 0, 1180, 0    // -157 - 0
1164: subleq 0, 0, 0    // -157 - -157
1167: subleq 1186, 1186, 0    // 0 - 0
1170: subleq 161, 0, 0    // 157 - 0
1173: subleq 0, 1186, 0    // -157 - 0
1176: subleq 0, 0, 0    // -157 - -157
1179: subleq 157, 157, 0    // 0 - 0
```

Figure 8: Disassembled subleq instructions

The Subleq code in this program is self-modifying in certain areas and depends heavily on values that may not be easily known through static analysis, so the technique of analyzing the instructions from dynamic analysis should be more straightforward than a purely static approach. That being said, I look forward to seeing the creative approaches to this taken by the challenge winners in the many solution blogs that will be published in the coming weeks.

The first two instructions executed in the program illustrate a technique that is used heavily in this program to access a literal value. Since the subleq architecture does not support any operand type other than indirect cell references, the covfefe program gets around this by jumping over a memory cell that it will then reference as data. Take a look again at those first two lines:

```
1123: subleq 0, 0, 1127   // 0 - 0
1127: subleq 1126, 1126, 0   // 0 - 0
```

The first instruction simply jumps to offset 1127 (because 0 - 0 will always be <= 0). This means it jumps OVER the byte 1126 because the cells 1123, 1124, and 1125 are part of that subleq instruction and the next instruction to execute starts at 1127. The contents of 1126 are used in the second instruction. This second instruction is the first in a four instruction sequence which is effectively a MOV instruction (you can see its implementation in the Wikipedia article cited above).

The covfefe subleq VM was built up by creating macros such as MOV for easy to use pseudo instructions. By applying the technique in reverse you can start to unravel the challenge. If you find every sequence that fits this pattern, you can replace it with a MOV(a, b) in your disassembly. In Figure 9 you can see the macros used to form the MOV instruction.

```
.macro MOV(%a,%b) // Move %a to %b
{
        MOVJMP(%a,%b,Z)
}
.macro MOVJMP(%a,%b,%c)   // Move %a to %b then jump to %c
{
        subleq %b, %b
        subleq %a, Z
        subleq Z, %b
        subleq Z, Z, %c
}
```

Figure 9: Implementation of the MOV pseudo-instruction

Another thing that causes issue with attempting to disassemble the subleq code is self-modification. To get around issues of working with pointers, covfefe makes use of techniques such as the following macro, which is used to jump to a location specified by a value in a particular cell. To accomplish this jump the cell is simply copied to the third operand of a subleq instruction that immediately follows the MOV macro.

```
.macro INDIRECT_JMP(%a) // jump to the location stored in %a
```

```
{
.start_context
            MOV(%a, _INDIRECT_JMP_tailjump+2)
_INDIRECT_JMP_tailjump: subleq Z, Z, Z
.end_context
}
```

Figure 10: Indirect Jump Macro

In Figure 10 the "subleq Z, Z, Z" (Z being a macro for 0 because zero is so special in subleq) the third operand "Z" will be replaced with the contents of some other cell. Note that this is a macro and not a function, so this code will be pasted into the program directly when it is used. You may see this many times throughout the program. For example, if we knew cell 150 had the value 200 in it, and we see this macro being used, it would result in the instruction "subleq 0, 0, 200" being executed which would jump unconditionally to program location 200.

The subleq architecture does not contain registers or any concept of a stack. To allow function calls though, a stack system was implemented in subleq for covfefe. It is a very limited stack as it only has to support a few arguments and return pointers, but it works exactly like you'd expect, except backwards. That's right, the stack is backwards. The stack is only 4 cells: 157-160. The stack pointer value is stored in cell 161. To demonstration the usage of the custom stack, please examine the first few lines of code in the high level macro language build on subleq for this challenge in Figure 11.

```
stack:   dd 0,0,0,0
sp:      dd stack
// … code removed for demonstration …
start:  PUSH_PTR_REF(Prompt, sp)
        CALL(printstring, sp)
        POP_IGNORE(1,sp)

        PUSH_LITERAL(inputbuf_end-inputbuf, sp)
        PUSH_PTR_REF(inputbuf, sp)
        CALL(readinputstring,sp)
        POP_IGNORE(2,sp)
```

Figure 11: Displaying prompt and reading input in high level subleq macro language

In this example, the stack is initialized to all zeros and the stack pointer "sp" is a pointer to the stack cell. Every command in the program that is going to use the stack needs to be explicitly passed the "sp" label name because it is not inherent in the architecture and is entirely a high-level construction. This code is responsible for displaying the prompt to the user and collecting their input. The implementation of the printstring and readinputstring functions are not shown in this example.

The complete high level implementation of the program is provided in

Appendix 1: Complete High-Level Subleq Program Source. This reference does not contain the implementation of the macros shown but should give you a high level understanding of how the program works from a high level. I have also provided two forms of the expanded macro versions in Appendix 2 and Appendix 3.

Appendix 2 shows the code from Appendix 1 after it has been run through the macro processor. It is a good intermediate to understanding what you see in the disassembled subleq and how it relates to the high level logic described in Appendix 1. Appendix 3 is the actual array and associated enum that was used to build this binary. Each line represents one line from the source in Appendix 2 assembled into data. Only data lines containing actual data though are represented, and for this reason it is much harder to read, but is a definitive reference for understanding what you are seeing in the subleq program array. Every value in the array is referenceable by address in the comment at the end of the line, which shows the address of the data and the subleq source code for that line.

Once you have an understanding of how to read the subleq code and a method to piece together high level functionality from its lone primitive you can begin to focus on the actual logic of the key verification. Luckily, this is the home stretch where the challenge gets easy. The key verification algorithm is quite simple. If we implemented this entirely in C this would have been a good first or second stage Flare-On level.

In the program, you can find the encoded form of the correct key stored at array offset 3740. The program is not decoding this string but rather encoding the user input and comparing that with the encoded form of the answer key. After each encoded digit of the input, if it doesn't match the correct key then a pass bit in the array is set to zero. This value is initialized to 1 and stays that value unless an incorrect digit is encountered. This pass bit is tucked away at offset 3757 in the array and, once discovered, could be used to easily brute force the correct password one digit at a time.

If we take the algorithm found in the program and produce its inverse, we will have a function that can actually decode the answer key. The python script shown in Figure 12 can be used to decode the correct answer key found at offset 3740 in binary.

```
answerkey = [220810, 188179, 193934, 182430, 211227, 182413, 193947,
224668, 222742, 213152, 186267, 182430, 188172, 224653, 192010,
209407]

def decypher(inputcypher):
    retval = ''
    for cypherval in inputcypher:
```

```
        nextbyte = (cypherval & 0x7F) ^ 0x7F
        thisbyte = (cypherval >> 7) / 15
        retval += chr(thisbyte)
        retval += chr(nextbyte)
    return retval
```

Figure 12: Decoding script

Running this function with the answer key input at a python prompt will yield the following output:

```
>>> decypher(answerkey)
'subleq_and_reductio_ad_absurdum\x00'
```

Now if we enter this into the covfefe.exe prompt, we will learn the correct key to enter into the CTFd gameboard to advance to the next level!

```
Welcome to the magic box.
Enter the password: subleq_and_reductio_ad_absurdum
The password is correct. Good Job. You win Flare-On 4.
Your key to victory is: subleq_and_reductio_ad_absurdum@flare-on.com
```

Please forgive the part about “You win Flare-On 4.” This was supposed to be the final challenge until the development of what became the final challenge got a little out of hand, as you will see in the next solution in this series.

Appendix 1: Complete High-Level Subleq Program Source

```
// Flare-On 4: Challenge 11 - covfefe
.include "macros.subleq"

///////////////////////////////////////////
Z:      dd 0        // do not change this
INPUT:  dd 0        // do not change this
OUTPUT: dd 0        // do not change this
INPUT_READY: dd 0   // do not change this
OUTPUT_READY: dd 0  // do not change this
Prompt: dd 'W','e','l','c','o','m','e',' ','t','o',' ','t','h','e','
','m','a','g','i','c',' ','b','o','x','.',10,'E','n','t','e','r',' ','t','h','e','
','p','a','s','s','w','o','r','d',':',' ',0
rickroll: dd
'h','t','t','p',':','/','/','b','i','t','l','y','.','c','o','m','/','9','8','K','8'
,'e','H',0
retval: dd 0
inputbuf: dd
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
inputbuf_end:
stack:  dd 0,0,0,0
sp:     dd stack
you_entered: dd 'Y','o','u',' ','E','n','t','e','r','e','d',':',' ',0
key:    dd 291
winstring: dd 'T','h','e',' ','p','a','s','s','w','o','r','d',' ','i','s','
','c','o','r','r','e','c','t','.',' ','G','o','o','d',' ','J','o','b','.','
','Y','o','u',' ','w','i','n',' ','F','l','a','r','e','-','O','n','
','4','.',10,'Y','o','u','r',' ','k','e','y',' ','t','o','
','v','i','c','t','o','r','y',' ','i','s',':',' ',0
atflareon: dd '@','f','l','a','r','e','-','o','n','.','c','o','m',10,0
num_failstrings: dd failstring1-failstring_array
failstring_selector: dd 3
failstring_array: dd failstring1, failstring2, failstring3, failstring4,
failstring5, failstring6, failstring7, failstring8
failstring1: dd 'T','h','e',' ','p','a','s','s','w','o','r','d',' ','i','s','
','i','n','c','o','r','r','e','c','t','.',' ','T','r','y','
','r','e','v','e','r','s','e',' ','e','n','g','i','n','e','e','r','i','n','g','
','t','h','e',' ','b','i','n','a','r','y',' ','t','o','
','d','i','s','c','o','v','e','r',' ','t','h','e',' ','c','o','r','r','e','c','t','
','o','n','e','.',10,0
failstring2: dd 'I','n','c','o','r','r','e','c','t','
','p','a','s','s','w','o','r','d','.',' ','H','a','v','e',' ','y','o','u','
','t','r','i','e','d',' ','t','u','r','n','i','n','g',' ','y','o','u','r','
','c','o','m','p','u','t','e','r',' ','o','f','f',' ','a','n','d',' ','o','n','
','a','g','a','i','n','?',10,0
```

```
failstring3: dd 'T','h','a','t',' ','i','s',' ','t','h','e','
','w','r','o','n','g',' ','p','a','s','s','w','o','r','d','.','
','D','e','l','e','t','e',' ','t','h','e','
','C',':','\','W','i','n','d','o','w','s','\','S','y','s','t','e','m','3','2','
','f','o','l','d','e','r',' ','t','o',' ','m','a','k','e',' ','y','o','u','r','
','c','o','m','p','u','t','e','r',' ','r','u','n',' ','f','a','s','t','e','r',',','
','a','n','d',' ','t','r','y',' ','a','g','a','i','n','.',10,0
failstring4: dd 'P','a','s','s','w','o','r','d',' ','F','a','i','l','e','d','.','
','Y','o','u',' ','h','a','v','e',' ','u','s','e','d','
','i','n','s','u','f','f','i','c','i','e','n','t','
','c','o','m','p','u','t','e','r',' ','s','c','i','e','n','c','e','.','
','H','a','v','e',' ','y','o','u',' ','c','o','n','s','i','d','e','r','e','d','
','a',' ','c','a','r','e','e','r',' ','i','n',' ','s','a','l','e','s','
','i','n','s','t','e','a','d','?',10,0
failstring5: dd 'I','n','c','o','r','r','e','c','t','
','p','a','s','s','w','o','r','d','.',' ','Y','o','u','r','
','a','c','c','o','u','n','t',' ','s','h','o','u','l','d','
','p','r','o','b','a','b','l','y',' ','b','e','
','d','e','l','e','t','e','d','.',10,0
failstring6: dd 'C','o','m','p','l','e','t','e','
','p','a','s','s','w','o','r','d',' ','f','a','i','l','u','r','e','.','
','T','r','y',' ','a','g','a','i','n',' ','o','r',' ','w','a','i','t','
','u','n','t','i','l',' ','n','e','x','t',' ','y','e','a','r',' ','a','n','d','
','h','o','p','e',',',' ','i','n',' ','v','a','i','n',',',' ','f','o','r','
','a','n',' ','e','v','e','n','t',' ','e','a','s','i','e','r','
','c','h','a','l','l','e','n','g','e','.',10,0
failstring7: dd 'T','h','a','t',' ','i','s',' ','n','o','t',' ','t','h','e','
','p','a','s','s','w','o','r','d','.',' ','Y','o','u',' ','d','i','d','
','n','o','t',' ','s','o','l','v','e',' ','t','h','e','
','c','h','a','l','l','e','n','g','e','.',' ','C','o','n','s','i','d','e','r','
','i','n','s','t','a','l','l','i','n','g',' ','a','
','d','i','s','a','s','s','e','m','b','l','e','r',' ','s','u','c','h',' ','a','s','
','I','D','A',' ','P','r','o',',',' ','a','n','d',' ','t','r','y','
','a','g','a','i','n',',',' ','o','n','l','y',' ','h','a','r','d','e','r','
','n','e','x','t',' ','t','i','m','e','.',10,0
failstring8: dd 'Y','o','u',' ','d','i','d',' ','n','o','t','
','g','u','e','s','s',' ','t','h','e',' ','p','a','s','s','w','o','r','d','.','
','C','o','n','s','i','d','e','r',' ','b','r','u','t','e','-','f','o','r','c','e','
','g','u','e','s','s','i','n','g',' ','u','s','i','n','g',' ','a','
','s','c','r','i','p','t',',',' ','a','n','d',' ','l','e','t','t','i','n','g','
','i','t',' ','r','u','n',' ','f','o','r',' ','a',' ','f','e','w','
','t','r','i','l','l','i','o','n',' ','y','e','a','r','s','.',10,0
////////////////////////////////////////////
start:  PUSH_PTR_REF(Prompt, sp)
        CALL(printstring, sp)
        POP_IGNORE(1,sp)

        PUSH_LITERAL(inputbuf_end-inputbuf, sp)
        PUSH_PTR_REF(inputbuf, sp)
```

```
        CALL(readinputstring,sp)
        POP_IGNORE(2,sp)

        PUSH_PTR_REF(inputbuf, sp)
        CALL(check_string, sp)
        GET_FUNCTION_RETURN_VALUE(sp, retval)

        BNEZ(retval,correct_key)
        DECLARE_VARIABLE(failstring,0)
        DECLARE_VARIABLE(tmp,0)

        PTR_REF(failstring_array, tmp)
        ADD(failstring_selector, tmp)
        DEREF_SRC_MOV(tmp, failstring)

        PUSH(failstring, sp)
        CALL(printstring, sp)
        POP_IGNORE(1,sp)

        JMP(ending)
correct_key:
        PUSH_PTR_REF(winstring, sp)
        CALL(printstring, sp)
        POP_IGNORE(1,sp)

        PUSH_PTR_REF(inputbuf, sp)
        CALL(printstring, sp)
        POP_IGNORE(1,sp)

        PUSH_PTR_REF(atflareon, sp)
        CALL(printstring, sp)
        POP_IGNORE(1,sp)
ending:
        HALT()

FUNCTION(check_string, sp)
        DECLARE_FUNCTION_ARGUMENT(_string,-2)
        FORLOOP_START(_i,_answerkey_end-_answerkey,check_loop)
            DEREF_SRC_MOV(_string, _chara)
            BNEZ(_chara, _check_loop_continue)
            SETZ(_passbit)
            FORLOOP_BREAK(check_loop)
_check_loop_continue:
            INC(_string)
            DEREF_SRC_MOV(_string, _charb)
            MUL(_fifteen,_chara)
            SHL(7,_chara)
            XOR(_xormask, _charb)
            OR(_charb, _chara)
```

```
            DEREF_SRC_MOV(_answerkey_ptr,_charb)
            SUB(_chara,_charb)
            BEZ(_charb,_bytesmatched)
            SETZ(_passbit)
            JMP(_check_loop_tail)
_bytesmatched:
_check_loop_tail:
            INC(_string)
            INC(_answerkey_ptr)
        FORLOOP_END(_i,check_loop)
        SET_FUNCTION_RETURN_VALUE(_passbit)
        RET(sp)
_i:     dd 0
_chara: dd 0
_charb: dd 0
_fifteen: dd 15
_xormask: dd 0x7F

// subleq_and_reductio_ad_absurdum@flare-on.com
_answerkey: dd 220810, 188179, 193934, 182430, 211227, 182413, 193947, 224668,
222742, 213152, 186267, 182430, 188172, 224653, 192010, 209407

_answerkey_end:
_answerkey_ptr: dd _answerkey
_passbit:   dd 1
END_FUNCTION()

FUNCTION(printstring, sp)
        DECLARE_FUNCTION_ARGUMENT(_string,-2)
        DECLARE_VARIABLE(_char,0)
        LOOP_START(print_loop)
            DEREF_SRC_MOV(_string, _char)
            BNEZ(_char, _print_loop_continue)
            LOOP_BREAK(print_loop)
_print_loop_continue:
            OUT(_char)
            INC(_string)
        LOOP_END(print_loop)
        RET(sp)
END_FUNCTION()

FUNCTION(readinputstring, sp)
        DECLARE_FUNCTION_ARGUMENT(_bufptr,-2)
        DECLARE_FUNCTION_ARGUMENT(_buffersize,-3)
        DECLARE_VARIABLE(_i,0)
        DECLARE_VARIABLE(_newline,10)
        DECLARE_VARIABLE(_tmp,0)
        FORLOOP_VAR_START(_i,_buffersize,readinputstring_loop)
            IN(_tmp)
```

```
            BNEQ(_tmp,_newline,_not_newline)
            SETZ(_tmp)
            DEREF_DST_MOV(_tmp,_bufptr)
            JMP(_readinputstring_finished)
_not_newline:
            DEREF_DST_MOV(_tmp,_bufptr)
            INC(_bufptr)
        FORLOOP_END(_i,readinputstring_loop)
_readinputstring_finished:
        RET(sp)
END_FUNCTION()
```

Appendix 2: Complete subleq Instruction and Data Dump

```
Z:      dd 0
INPUT:  dd 0
OUTPUT: dd 0
INPUT_READY: dd 0
OUTPUT_READY: dd 0
Prompt: dd 'W','e','l','c','o','m','e',' ','t','o',' ','t','h','e','
','m','a','g','i','c',' ','b','o','x','.',10,'E','n','t','e','r',' ','t','h','e','
','p','a','s','s','w','o','r','d',':',' ',0
rickroll: dd
'h','t','t','p',':','/','/','b','i','t','l','y','.','c','o','m','/','9','8','K','8'
,'e','H',0
retval: dd 0
inputbuf: dd
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
inputbuf_end:
stack:  dd 0,0,0,0
sp:     dd stack
you_entered: dd 'Y','o','u',' ','E','n','t','e','r','e','d',':',' ',0
key:    dd 291
winstring: dd 'T','h','e',' ','p','a','s','s','w','o','r','d',' ','i','s','
','c','o','r','r','e','c','t','.',' ','G','o','o','d',' ','J','o','b','.','
','Y','o','u',' ','w','i','n',' ','F','l','a','r','e','-','O','n','
','4','.',10,'Y','o','u','r',' ','k','e','y',' ','t','o','
','v','i','c','t','o','r','y',' ','i','s',':',' ',0
atflareon: dd '@','f','l','a','r','e','-','o','n','.','c','o','m',10,0
num_failstrings: dd failstring1-failstring_array
failstring_selector: dd 3
failstring_array: dd failstring1, failstring2, failstring3, failstring4,
failstring5, failstring6, failstring7, failstring8
failstring1: dd 'T','h','e',' ','p','a','s','s','w','o','r','d',' ','i','s','
','i','n','c','o','r','r','e','c','t','.',' ','T','r','y','
','r','e','v','e','r','s','e',' ','e','n','g','i','n','e','e','r','i','n','g','
','t','h','e',' ','b','i','n','a','r','y',' ','t','o','
','d','i','s','c','o','v','e','r',' ','t','h','e',' ','c','o','r','r','e','c','t','
','o','n','e','.',10,0
failstring2: dd 'I','n','c','o','r','r','e','c','t','
','p','a','s','s','w','o','r','d','.',' ','H','a','v','e',' ','y','o','u','
','t','r','i','e','d',' ','t','u','r','n','i','n','g',' ','y','o','u','r','
','c','o','m','p','u','t','e','r',' ','o','f','f',' ','a','n','d',' ','o','n','
','a','g','a','i','n','?',10,0
failstring3: dd 'T','h','a','t',' ','i','s',' ','t','h','e','
','w','r','o','n','g',' ','p','a','s','s','w','o','r','d','.','
','D','e','l','e','t','e',' ','t','h','e','
','C',':','\','W','i','n','d','o','w','s','\','S','y','s','t','e','m','3','2','
```

```
','f','o','l','d','e','r',' ','t','o',' ','m','a','k','e',' ','y','o','u','r','
','c','o','m','p','u','t','e','r',' ','r','u','n',' ','f','a','s','t','e','r',',','
','a','n','d',' ','t','r','y',' ','a','g','a','i','n','.',10,0
failstring4: dd 'P','a','s','s','w','o','r','d',' ','F','a','i','l','e','d','.','
','Y','o','u',' ','h','a','v','e',' ','u','s','e','d','
','i','n','s','u','f','f','i','c','i','e','n','t','
','c','o','m','p','u','t','e','r',' ','s','c','i','e','n','c','e','.','
','H','a','v','e',' ','y','o','u',' ','c','o','n','s','i','d','e','r','e','d','
','a',' ','c','a','r','e','e','r',' ','i','n',' ','s','a','l','e','s','
','i','n','s','t','e','a','d','?',10,0
failstring5: dd 'I','n','c','o','r','r','e','c','t','
','p','a','s','s','w','o','r','d','.',' ','Y','o','u','r','
','a','c','c','o','u','n','t',' ','s','h','o','u','l','d','
','p','r','o','b','a','b','l','y',' ','b','e','
','d','e','l','e','t','e','d','.',10,0
failstring6: dd 'C','o','m','p','l','e','t','e','
','p','a','s','s','w','o','r','d',' ','f','a','i','l','u','r','e','.','
','T','r','y',' ','a','g','a','i','n',' ','o','r',' ','w','a','i','t','
','u','n','t','i','l',' ','n','e','x','t',' ','y','e','a','r',' ','a','n','d','
','h','o','p','e',',',' ','i','n',' ','v','a','i','n',',',' ','f','o','r','
','a','n',' ','e','v','e','n',' ','e','a','s','i','e','r','
','c','h','a','l','l','e','n','g','e','.',10,0
failstring7: dd 'T','h','a','t',' ','i','s',' ','n','o','t',' ','t','h','e','
','p','a','s','s','w','o','r','d','.',' ','Y','o','u',' ','d','i','d','
','n','o','t',' ','s','o','l','v','e',' ','t','h','e','
','c','h','a','l','l','e','n','g','e','.',' ','C','o','n','s','i','d','e','r','
','i','n','s','t','a','l','l','i','n','g',' ','a','
','d','i','s','a','s','s','e','m','b','l','e','r',' ','s','u','c','h',' ','a','s','
','I','D','A',' ','P','r','o',',',' ','a','n','d',' ','t','r','y','
','a','g','a','i','n',',',' ','o','n','l','y',' ','h','a','r','d','e','r','
','n','e','x','t',' ','t','i','m','e','.',10,0
failstring8: dd 'Y','o','u',' ','d','i','d',' ','n','o','t','
','g','u','e','s','s',' ','t','h','e',' ','p','a','s','s','w','o','r','d','.','
','C','o','n','s','i','d','e','r',' ','b','r','u','t','e','-','f','o','r','c','e','
','g','u','e','s','s','i','n','g',' ','u','s','i','n','g',' ','a','
','s','c','r','i','p','t',',',' ','a','n','d',' ','l','e','t','t','i','n','g','
','i','t',' ','r','u','n',' ','f','o','r',' ','a',' ','f','e','w','
','t','r','i','l','l','i','o','n',' ','y','e','a','r','s','.',10,0
start:
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd Prompt
```

```
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,  sp
        subleq Z, Z
.end_context
.end_context
.start_context
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd _CALL_retloc
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq   sp, Z
```

```
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,   sp
        subleq Z, Z
.end_context
.end_context
        subleq Z, Z, printstring
_CALL_retloc:
.end_context
.start_context
        subleq _SUB_LITERAL_val, sp
        subleq Z, Z, _SUB_LITERAL_end
_SUB_LITERAL_val:   dd 1
_SUB_LITERAL_end:
.end_context
.start_context
        subleq Z, Z, $+4
_PUSH_LITERAL_tmp:  dd inputbuf_end-inputbuf
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PUSH_LITERAL_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
```

```
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,  sp
        subleq Z, Z
.end_context
.end_context
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd inputbuf
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,  sp
        subleq Z, Z
.end_context
.end_context
```

```
.start_context
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd _CALL_retloc
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,  sp
        subleq Z, Z
.end_context
.end_context
        subleq Z, Z, readinputstring
_CALL_retloc:
.end_context
.start_context
        subleq _SUB_LITERAL_val, sp
        subleq Z, Z, _SUB_LITERAL_end
_SUB_LITERAL_val:   dd 2
_SUB_LITERAL_end:
.end_context
```

```
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd inputbuf
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,  sp
        subleq Z, Z
.end_context
.end_context
.start_context
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
```

```
_SETVALUE_tmp:  dd _CALL_retloc
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,   sp
        subleq Z, Z
.end_context
.end_context
        subleq Z, Z, check_string
_CALL_retloc:
.end_context
.start_context
        subleq Z, Z, _DEC_sub
_DEC_ONE:   dd 1
_DEC_sub:
        subleq _DEC_ONE, sp
.end_context
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq sp, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq  retval,  retval
_DEREF_target:  subleq Z, Z
                subleq Z,  retval
                subleq Z, Z
.end_context
```

```
.start_context
.start_context
            subleq retval, Z, _BEZ_true
            subleq Z, Z, _BEZ_false
_BEZ_true:  subleq Z, Z
            subleq Z, retval, _BNEZ_false
_BEZ_false:
.end_context
        subleq Z, Z, correct_key
_BNEZ_false:
.end_context
        subleq Z, Z, $+4
failstring: dd 0
        subleq Z, Z, $+4
tmp:    dd 0
.start_context
        subleq   tmp,   tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,   tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd failstring_array
_SETVALUE_end:
.end_context
        subleq failstring_selector, Z
        subleq Z,  tmp
        subleq Z, Z
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq tmp, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq  failstring,  failstring
_DEREF_target:  subleq Z, Z
                subleq Z,  failstring
                subleq Z, Z
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
```

```
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq failstring, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,  sp
        subleq Z, Z
.end_context
.start_context
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd _CALL_retloc
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
```

```
        subleq _INC_ONE, Z
        subleq Z,   sp
        subleq Z, Z
.end_context
.end_context
        subleq Z, Z, printstring
_CALL_retloc:
.end_context
.start_context
        subleq _SUB_LITERAL_val, sp
        subleq Z, Z, _SUB_LITERAL_end
_SUB_LITERAL_val:   dd 1
_SUB_LITERAL_end:
.end_context
        subleq Z, Z, ending
correct_key:
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd winstring
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
```

```
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,  sp
        subleq Z, Z
.end_context
.end_context
.start_context
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd _CALL_retloc
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,   sp
        subleq Z, Z
.end_context
.end_context
        subleq Z, Z, printstring
_CALL_retloc:
.end_context
```

```
.start_context
        subleq _SUB_LITERAL_val, sp
        subleq Z, Z, _SUB_LITERAL_end
_SUB_LITERAL_val:   dd 1
_SUB_LITERAL_end:
.end_context
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd inputbuf
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,  sp
        subleq Z, Z
.end_context
.end_context
.start_context
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
```

```
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd _CALL_retloc
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,   sp
        subleq Z, Z
.end_context
.end_context
        subleq Z, Z, printstring
_CALL_retloc:
.end_context
.start_context
        subleq _SUB_LITERAL_val, sp
        subleq Z, Z, _SUB_LITERAL_end
_SUB_LITERAL_val:   dd 1
_SUB_LITERAL_end:
.end_context
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
```

```
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd atflareon
_SETVALUE_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq  sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,  sp
        subleq Z, Z
.end_context
.end_context
.start_context
.start_context
        subleq Z, Z, $+4
_PTR_REF_tmp:   dd 0
.start_context
        subleq  _PTR_REF_tmp,  _PTR_REF_tmp
        subleq _SETVALUE_tmp, Z
        subleq Z,  _PTR_REF_tmp
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd _CALL_retloc
_SETVALUE_end:
.end_context
.start_context
```

```
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq   sp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _PTR_REF_tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z,   sp
        subleq Z, Z
.end_context
.end_context
        subleq Z, Z, printstring
_CALL_retloc:
.end_context
.start_context
        subleq _SUB_LITERAL_val, sp
        subleq Z, Z, _SUB_LITERAL_end
_SUB_LITERAL_val:   dd 1
_SUB_LITERAL_end:
.end_context
ending:
        subleq Z, Z, -1
check_string:
.start_context
        subleq Z, Z, $+4
_bp:    dd 0
        subleq _bp, _bp
        subleq  sp, Z
        subleq Z, _bp
        subleq Z, Z, Z
        subleq Z, Z, $+4
_string:    dd 0
.start_context
        subleq Z, Z, $+4
```

```
_DECLARE_FUNCTION_ARGUMENT_bp_offset:   dd -2
        subleq Z, Z, $+4
_DECLARE_FUNCTION_ARGUMENT_tmp: dd 0
        subleq _DECLARE_FUNCTION_ARGUMENT_tmp, _DECLARE_FUNCTION_ARGUMENT_tmp
        subleq _DECLARE_FUNCTION_ARGUMENT_bp_offset, Z
        subleq Z, _DECLARE_FUNCTION_ARGUMENT_tmp
        subleq Z, Z, Z
        subleq _bp, Z
        subleq Z, _DECLARE_FUNCTION_ARGUMENT_tmp
        subleq Z, Z
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq _DECLARE_FUNCTION_ARGUMENT_tmp, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq _string, _string
_DEREF_target:  subleq Z, Z
                subleq Z, _string
                subleq Z, Z
.end_context
.end_context
_FORLOOP_START_LOOPID_check_loop:
.start_context
        subleq Z, Z, _FORLOOP_START_compare
_FORLOOP_START_numloops: dd _answerkey_end-_answerkey
_FORLOOP_START_compare:
.start_context
        subleq _BGEQ_tmp, _BGEQ_tmp
        subleq _i, Z
        subleq Z, _BGEQ_tmp
        subleq Z, Z, Z
        subleq _FORLOOP_START_numloops, _BGEQ_tmp
.start_context
            subleq _BGEQ_tmp, Z, _BLZ_false
            subleq Z, Z, _BGEQ_false
_BLZ_false: subleq Z, Z
.end_context
        subleq Z, Z, _FORLOOP_END_LOOPID_check_loop
_BGEQ_tmp:  dd 0
_BGEQ_false:
.end_context
.end_context
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq _string, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq  _chara,  _chara
_DEREF_target:  subleq Z, Z
```

```
                subleq Z,  _chara
                subleq Z, Z
.end_context
.start_context
.start_context
            subleq _chara, Z, _BEZ_true
            subleq Z, Z, _BEZ_false
_BEZ_true:  subleq Z, Z
            subleq Z, _chara, _BNEZ_false
_BEZ_false:
.end_context
        subleq Z, Z,  _check_loop_continue
_BNEZ_false:
.end_context
        subleq _passbit, _passbit, 0
        subleq Z, Z, _FORLOOP_END_LOOPID_check_loop
_check_loop_continue:
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _string
        subleq Z, Z
.end_context
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq _string, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq  _charb,  _charb
_DEREF_target:  subleq Z, Z
                subleq Z,  _charb
                subleq Z, Z
.end_context
.start_context
        subleq _MUL_i, _MUL_i, 0
        subleq _MUL_result, _MUL_result, 0
.start_context
            subleq _chara, Z, _BLZ_false
            subleq Z, Z, _MUL_b_neg
_BLZ_false: subleq Z, Z
.end_context
        subleq Z, Z, _MUL_b_pos
_MUL_i:     dd 0
_MUL_result: dd 0
_MUL_b_pos:
.start_context
        subleq _BEQ_tmp, _BEQ_tmp
```

```
        subleq _chara, Z
        subleq Z, _BEQ_tmp
        subleq Z, Z, Z
        subleq  _MUL_i, _BEQ_tmp
.start_context
.start_context
            subleq _BEQ_tmp, Z, _BEZ_true
            subleq Z, Z, _BEZ_false
_BEZ_true:  subleq Z, Z
            subleq Z, _BEQ_tmp, _BNEZ_false
_BEZ_false:
.end_context
        subleq Z, Z, _BEQ_false
_BNEZ_false:
.end_context
        subleq Z, Z,  _MUL_finished
_BEQ_tmp:   dd 0
_BEQ_false:
.end_context
        subleq _fifteen, Z
        subleq Z, _MUL_result
        subleq Z, Z
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _MUL_i
        subleq Z, Z
.end_context
        subleq Z, Z, _MUL_b_pos
_MUL_b_neg:
.start_context
        subleq _BEQ_tmp, _BEQ_tmp
        subleq _chara, Z
        subleq Z, _BEQ_tmp
        subleq Z, Z, Z
        subleq _MUL_i, _BEQ_tmp
.start_context
.start_context
            subleq _BEQ_tmp, Z, _BEZ_true
            subleq Z, Z, _BEZ_false
_BEZ_true:  subleq Z, Z
            subleq Z, _BEQ_tmp, _BNEZ_false
_BEZ_false:
.end_context
        subleq Z, Z, _BEQ_false
_BNEZ_false:
.end_context
```

```
        subleq Z, Z, _MUL_finished
_BEQ_tmp:   dd 0
_BEQ_false:
.end_context
        subleq _fifteen, _MUL_result
.start_context
        subleq Z, Z, _DEC_sub
_DEC_ONE:   dd 1
_DEC_sub:
        subleq _DEC_ONE, _MUL_i
.end_context
        subleq Z, Z, _MUL_b_neg
_MUL_finished:
        subleq _chara, _chara
        subleq _MUL_result, Z
        subleq Z, _chara
        subleq Z, Z, Z
.end_context
.start_context
        subleq Z, Z, $+4
_i: dd 0
        subleq _i, _i, 0
_FORLOOP_START_LOOPID__SHL_loop:
.start_context
        subleq Z, Z, _FORLOOP_START_compare
_FORLOOP_START_numloops: dd 7
_FORLOOP_START_compare:
.start_context
        subleq _BGEQ_tmp, _BGEQ_tmp
        subleq _i, Z
        subleq Z, _BGEQ_tmp
        subleq Z, Z, Z
        subleq _FORLOOP_START_numloops, _BGEQ_tmp
.start_context
            subleq _BGEQ_tmp, Z, _BLZ_false
            subleq Z, Z, _BGEQ_false
_BLZ_false: subleq Z, Z
.end_context
        subleq Z, Z, _FORLOOP_END_LOOPID__SHL_loop
_BGEQ_tmp:  dd 0
_BGEQ_false:
.end_context
.end_context
        subleq _chara, Z
        subleq _chara, Z
        subleq _chara, _chara
        subleq Z, _chara
        subleq Z, Z
.start_context
```

```
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _i
        subleq Z, Z
.end_context
        subleq Z, Z, _FORLOOP_START_LOOPID__SHL_loop
_FORLOOP_END_LOOPID__SHL_loop:
.end_context
.start_context
        subleq _BITWISE_OPERATOR_tmp_a, _BITWISE_OPERATOR_tmp_a
        subleq _xormask, Z
        subleq Z, _BITWISE_OPERATOR_tmp_a
        subleq Z, Z, Z
        subleq _BITWISE_OPERATOR_tmp_b, _BITWISE_OPERATOR_tmp_b
        subleq  _charb, Z
        subleq Z, _BITWISE_OPERATOR_tmp_b
        subleq Z, Z, Z
        subleq _BITWISE_OPERATOR_i, _BITWISE_OPERATOR_i, 0
_FORLOOP_START_LOOPID__BITWISE_OPERATOR_loop:
.start_context
        subleq Z, Z, _FORLOOP_START_compare
_FORLOOP_START_numloops: dd 32
_FORLOOP_START_compare:
.start_context
        subleq _BGEQ_tmp, _BGEQ_tmp
        subleq _BITWISE_OPERATOR_i, Z
        subleq Z, _BGEQ_tmp
        subleq Z, Z, Z
        subleq _FORLOOP_START_numloops, _BGEQ_tmp
.start_context
            subleq _BGEQ_tmp, Z, _BLZ_false
            subleq Z, Z, _BGEQ_false
_BLZ_false: subleq Z, Z
.end_context
        subleq Z, Z, _FORLOOP_END_LOOPID__BITWISE_OPERATOR_loop
_BGEQ_tmp:  dd 0
_BGEQ_false:
.end_context
.end_context
.start_context
.start_context
            subleq _BITWISE_OPERATOR_tmp_a, Z, _BLZ_false
            subleq Z, Z, _GETMSB_return1
_BLZ_false: subleq Z, Z
.end_context
_GETMSB_return0:
        subleq _BITWISE_OPERATOR_msb_a, _BITWISE_OPERATOR_msb_a
```

```
        subleq _GETMSB_zero, Z
        subleq Z, _BITWISE_OPERATOR_msb_a
        subleq Z, Z, _GETMSB_finished
_GETMSB_return1:
        subleq _BITWISE_OPERATOR_msb_a, _BITWISE_OPERATOR_msb_a
        subleq _GETMSB_one, Z
        subleq Z, _BITWISE_OPERATOR_msb_a
        subleq Z, Z, _GETMSB_finished
_GETMSB_one: dd 1
_GETMSB_zero: dd 0
_GETMSB_finished:
.end_context
.start_context
.start_context
            subleq _BITWISE_OPERATOR_tmp_b, Z, _BLZ_false
            subleq Z, Z, _GETMSB_return1
_BLZ_false: subleq Z, Z
.end_context
_GETMSB_return0:
        subleq _BITWISE_OPERATOR_msb_b, _BITWISE_OPERATOR_msb_b
        subleq _GETMSB_zero, Z
        subleq Z, _BITWISE_OPERATOR_msb_b
        subleq Z, Z, _GETMSB_finished
_GETMSB_return1:
        subleq _BITWISE_OPERATOR_msb_b, _BITWISE_OPERATOR_msb_b
        subleq _GETMSB_one, Z
        subleq Z, _BITWISE_OPERATOR_msb_b
        subleq Z, Z, _GETMSB_finished
_GETMSB_one: dd 1
_GETMSB_zero: dd 0
_GETMSB_finished:
.end_context
.start_context
        subleq _BITWISE_JMP_tmp, _BITWISE_JMP_tmp
        subleq _BITWISE_OPERATOR_msb_a, Z
        subleq Z, _BITWISE_JMP_tmp
        subleq Z, Z, Z
        subleq _BITWISE_OPERATOR_msb_b, Z
        subleq Z, _BITWISE_JMP_tmp
        subleq Z, Z
.start_context
            subleq _BITWISE_JMP_tmp, Z, _BEZ_true
            subleq Z, Z, _BEZ_false
_BEZ_true:  subleq Z, Z
            subleq Z, _BITWISE_JMP_tmp, _BITWISE_OPERATOR_none
_BEZ_false:
.end_context
.start_context
        subleq Z, Z, _DEC_sub
```

```
_DEC_ONE:    dd 1
_DEC_sub:
        subleq _DEC_ONE, _BITWISE_JMP_tmp
.end_context
.start_context
            subleq _BITWISE_JMP_tmp, Z, _BEZ_true
            subleq Z, Z, _BEZ_false
_BEZ_true:  subleq Z, Z
            subleq Z, _BITWISE_JMP_tmp, _BITWISE_OPERATOR_onlyone
_BEZ_false:
.end_context
        subleq Z, Z, _BITWISE_OPERATOR_both
_BITWISE_JMP_tmp:   dd 0
.end_context
_BITWISE_OPERATOR_tmp_a: dd 0
_BITWISE_OPERATOR_tmp_b: dd 0
_BITWISE_OPERATOR_msb_a: dd 0
_BITWISE_OPERATOR_msb_b: dd 0
_BITWISE_OPERATOR_result: dd 0
_BO_ZERO: dd 0
_BO_ONE:  dd 1
_BITWISE_OPERATOR_i:    dd 0
_BITWISE_OPERATOR_none:
        subleq _BITWISE_OPERATOR_msb_a, _BITWISE_OPERATOR_msb_a
        subleq _BO_ZERO, Z
        subleq Z, _BITWISE_OPERATOR_msb_a
        subleq Z, Z, _BITWISE_OPERATOR_insert_new_bit
_BITWISE_OPERATOR_onlyone:
        subleq _BITWISE_OPERATOR_msb_a, _BITWISE_OPERATOR_msb_a
        subleq _BO_ONE, Z
        subleq Z, _BITWISE_OPERATOR_msb_a
        subleq Z, Z, _BITWISE_OPERATOR_insert_new_bit
_BITWISE_OPERATOR_both:
        subleq _BITWISE_OPERATOR_msb_a, _BITWISE_OPERATOR_msb_a
        subleq _BO_ZERO, Z
        subleq Z, _BITWISE_OPERATOR_msb_a
        subleq Z, Z, Z
_BITWISE_OPERATOR_insert_new_bit:
        subleq _BITWISE_OPERATOR_result, Z
        subleq _BITWISE_OPERATOR_result, Z
        subleq _BITWISE_OPERATOR_result, _BITWISE_OPERATOR_result
        subleq Z, _BITWISE_OPERATOR_result
        subleq Z, Z
        subleq _BITWISE_OPERATOR_msb_a, Z
        subleq Z, _BITWISE_OPERATOR_result
        subleq Z, Z
        subleq _BITWISE_OPERATOR_tmp_a, Z
        subleq _BITWISE_OPERATOR_tmp_a, Z
        subleq _BITWISE_OPERATOR_tmp_a, _BITWISE_OPERATOR_tmp_a
```

```
        subleq Z, _BITWISE_OPERATOR_tmp_a
        subleq Z, Z
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _BITWISE_OPERATOR_tmp_a
        subleq Z, Z
.end_context
        subleq _BITWISE_OPERATOR_tmp_b, Z
        subleq _BITWISE_OPERATOR_tmp_b, Z
        subleq _BITWISE_OPERATOR_tmp_b, _BITWISE_OPERATOR_tmp_b
        subleq Z, _BITWISE_OPERATOR_tmp_b
        subleq Z, Z
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _BITWISE_OPERATOR_tmp_b
        subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _BITWISE_OPERATOR_i
        subleq Z, Z
.end_context
        subleq Z, Z, _FORLOOP_START_LOOPID__BITWISE_OPERATOR_loop
_FORLOOP_END_LOOPID__BITWISE_OPERATOR_loop:
        subleq  _charb,  _charb
        subleq _BITWISE_OPERATOR_result, Z
        subleq Z,  _charb
        subleq Z, Z, Z
.end_context
.start_context
        subleq _BITWISE_OPERATOR_tmp_a, _BITWISE_OPERATOR_tmp_a
        subleq _charb, Z
        subleq Z, _BITWISE_OPERATOR_tmp_a
        subleq Z, Z, Z
        subleq _BITWISE_OPERATOR_tmp_b, _BITWISE_OPERATOR_tmp_b
        subleq  _chara, Z
        subleq Z, _BITWISE_OPERATOR_tmp_b
        subleq Z, Z, Z
        subleq _BITWISE_OPERATOR_i, _BITWISE_OPERATOR_i, 0
_FORLOOP_START_LOOPID__BITWISE_OPERATOR_loop:
```

```
.start_context
        subleq Z, Z, _FORLOOP_START_compare
_FORLOOP_START_numloops: dd 32
_FORLOOP_START_compare:
.start_context
        subleq _BGEQ_tmp, _BGEQ_tmp
        subleq _BITWISE_OPERATOR_i, Z
        subleq Z, _BGEQ_tmp
        subleq Z, Z, Z
        subleq _FORLOOP_START_numloops, _BGEQ_tmp
.start_context
            subleq _BGEQ_tmp, Z, _BLZ_false
            subleq Z, Z, _BGEQ_false
_BLZ_false: subleq Z, Z
.end_context
        subleq Z, Z, _FORLOOP_END_LOOPID__BITWISE_OPERATOR_loop
_BGEQ_tmp:  dd 0
_BGEQ_false:
.end_context
.end_context
.start_context
.start_context
            subleq _BITWISE_OPERATOR_tmp_a, Z, _BLZ_false
            subleq Z, Z, _GETMSB_return1
_BLZ_false: subleq Z, Z
.end_context
_GETMSB_return0:
        subleq _BITWISE_OPERATOR_msb_a, _BITWISE_OPERATOR_msb_a
        subleq _GETMSB_zero, Z
        subleq Z, _BITWISE_OPERATOR_msb_a
        subleq Z, Z, _GETMSB_finished
_GETMSB_return1:
        subleq _BITWISE_OPERATOR_msb_a, _BITWISE_OPERATOR_msb_a
        subleq _GETMSB_one, Z
        subleq Z, _BITWISE_OPERATOR_msb_a
        subleq Z, Z, _GETMSB_finished
_GETMSB_one: dd 1
_GETMSB_zero: dd 0
_GETMSB_finished:
.end_context
.start_context
.start_context
            subleq _BITWISE_OPERATOR_tmp_b, Z, _BLZ_false
            subleq Z, Z, _GETMSB_return1
_BLZ_false: subleq Z, Z
.end_context
_GETMSB_return0:
        subleq _BITWISE_OPERATOR_msb_b, _BITWISE_OPERATOR_msb_b
        subleq _GETMSB_zero, Z
```

```
        subleq Z, _BITWISE_OPERATOR_msb_b
        subleq Z, Z, _GETMSB_finished
_GETMSB_return1:
        subleq _BITWISE_OPERATOR_msb_b, _BITWISE_OPERATOR_msb_b
        subleq _GETMSB_one, Z
        subleq Z, _BITWISE_OPERATOR_msb_b
        subleq Z, Z, _GETMSB_finished
_GETMSB_one: dd 1
_GETMSB_zero: dd 0
_GETMSB_finished:
.end_context
.start_context
        subleq _BITWISE_JMP_tmp, _BITWISE_JMP_tmp
        subleq _BITWISE_OPERATOR_msb_a, Z
        subleq Z, _BITWISE_JMP_tmp
        subleq Z, Z, Z
        subleq _BITWISE_OPERATOR_msb_b, Z
        subleq Z, _BITWISE_JMP_tmp
        subleq Z, Z
.start_context
            subleq _BITWISE_JMP_tmp, Z, _BEZ_true
            subleq Z, Z, _BEZ_false
_BEZ_true:  subleq Z, Z
            subleq Z, _BITWISE_JMP_tmp, _BITWISE_OPERATOR_none
_BEZ_false:
.end_context
.start_context
        subleq Z, Z, _DEC_sub
_DEC_ONE:   dd 1
_DEC_sub:
        subleq _DEC_ONE, _BITWISE_JMP_tmp
.end_context
.start_context
            subleq _BITWISE_JMP_tmp, Z, _BEZ_true
            subleq Z, Z, _BEZ_false
_BEZ_true:  subleq Z, Z
            subleq Z, _BITWISE_JMP_tmp, _BITWISE_OPERATOR_onlyone
_BEZ_false:
.end_context
        subleq Z, Z, _BITWISE_OPERATOR_both
_BITWISE_JMP_tmp:   dd 0
.end_context
_BITWISE_OPERATOR_tmp_a: dd 0
_BITWISE_OPERATOR_tmp_b: dd 0
_BITWISE_OPERATOR_msb_a: dd 0
_BITWISE_OPERATOR_msb_b: dd 0
_BITWISE_OPERATOR_result: dd 0
_BO_ZERO: dd 0
_BO_ONE:  dd 1
```

```
_BITWISE_OPERATOR_i:    dd 0
_BITWISE_OPERATOR_none:
        subleq _BITWISE_OPERATOR_msb_a, _BITWISE_OPERATOR_msb_a
        subleq _BO_ZERO, Z
        subleq Z, _BITWISE_OPERATOR_msb_a
        subleq Z, Z, _BITWISE_OPERATOR_insert_new_bit
_BITWISE_OPERATOR_onlyone:
        subleq _BITWISE_OPERATOR_msb_a, _BITWISE_OPERATOR_msb_a
        subleq _BO_ONE, Z
        subleq Z, _BITWISE_OPERATOR_msb_a
        subleq Z, Z, _BITWISE_OPERATOR_insert_new_bit
_BITWISE_OPERATOR_both:
        subleq _BITWISE_OPERATOR_msb_a, _BITWISE_OPERATOR_msb_a
        subleq _BO_ONE, Z
        subleq Z, _BITWISE_OPERATOR_msb_a
        subleq Z, Z, Z
_BITWISE_OPERATOR_insert_new_bit:
        subleq _BITWISE_OPERATOR_result, Z
        subleq _BITWISE_OPERATOR_result, Z
        subleq _BITWISE_OPERATOR_result, _BITWISE_OPERATOR_result
        subleq Z, _BITWISE_OPERATOR_result
        subleq Z, Z
        subleq _BITWISE_OPERATOR_msb_a, Z
        subleq Z, _BITWISE_OPERATOR_result
        subleq Z, Z
        subleq _BITWISE_OPERATOR_tmp_a, Z
        subleq _BITWISE_OPERATOR_tmp_a, Z
        subleq _BITWISE_OPERATOR_tmp_a, _BITWISE_OPERATOR_tmp_a
        subleq Z, _BITWISE_OPERATOR_tmp_a
        subleq Z, Z
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _BITWISE_OPERATOR_tmp_a
        subleq Z, Z
.end_context
        subleq _BITWISE_OPERATOR_tmp_b, Z
        subleq _BITWISE_OPERATOR_tmp_b, Z
        subleq _BITWISE_OPERATOR_tmp_b, _BITWISE_OPERATOR_tmp_b
        subleq Z, _BITWISE_OPERATOR_tmp_b
        subleq Z, Z
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _BITWISE_OPERATOR_tmp_b
```

```
        subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _BITWISE_OPERATOR_i
        subleq Z, Z
.end_context
        subleq Z, Z, _FORLOOP_START_LOOPID__BITWISE_OPERATOR_loop
_FORLOOP_END_LOOPID__BITWISE_OPERATOR_loop:
        subleq  _chara,  _chara
        subleq _BITWISE_OPERATOR_result, Z
        subleq Z,  _chara
        subleq Z, Z, Z
.end_context
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq _answerkey_ptr, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq _charb, _charb
_DEREF_target:  subleq Z, Z
                subleq Z, _charb
                subleq Z, Z
.end_context
        subleq _chara, _charb
.start_context
            subleq _charb, Z, _BEZ_true
            subleq Z, Z, _BEZ_false
_BEZ_true:  subleq Z, Z
            subleq Z, _charb, _bytesmatched
_BEZ_false:
.end_context
        subleq _passbit, _passbit, 0
        subleq Z, Z, _check_loop_tail
_bytesmatched:
_check_loop_tail:
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _string
        subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
```

```
_INC_ONE:    dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _answerkey_ptr
        subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:    dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _i
        subleq Z, Z
.end_context
        subleq Z, Z, _FORLOOP_START_LOOPID_check_loop
_FORLOOP_END_LOOPID_check_loop:
.start_context
        subleq Z, Z, $+4
_SET_FUNCTION_RETURN_VALUE_tmp: dd 0
        subleq _SET_FUNCTION_RETURN_VALUE_tmp, _SET_FUNCTION_RETURN_VALUE_tmp
        subleq _bp, Z
        subleq Z, _SET_FUNCTION_RETURN_VALUE_tmp
        subleq Z, Z, Z
.start_context
        subleq _SUB_LITERAL_val, _SET_FUNCTION_RETURN_VALUE_tmp
        subleq Z, Z, _SUB_LITERAL_end
_SUB_LITERAL_val:    dd 2
_SUB_LITERAL_end:
.end_context
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq _SET_FUNCTION_RETURN_VALUE_tmp, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq _SET_FUNCTION_RETURN_VALUE_tmp, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq _SET_FUNCTION_RETURN_VALUE_tmp, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:    subleq Z, Z
                subleq _passbit, Z
_DEREF_DST_MOV_y:    subleq Z, Z
                subleq Z, Z
.end_context
.end_context
.start_context
```

```
.start_context
        subleq Z, Z, _DEC_sub
_DEC_ONE:   dd 1
_DEC_sub:
        subleq _DEC_ONE, sp
.end_context
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq sp, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq  _RET_tailjump+2,  _RET_tailjump+2
_DEREF_target:  subleq Z, Z
                subleq Z,  _RET_tailjump+2
                subleq Z, Z
.end_context
_RET_tailjump:  subleq Z, Z, Z
.end_context
_i:     dd 0
_chara: dd 0
_charb: dd 0
_fifteen: dd 15
_xormask: dd 0x7F
_answerkey: dd 220810, 188179, 193934, 182430, 211227, 182413, 193947, 224668,
222742, 213152, 186267, 182430, 188172, 224653, 192010, 209407
_answerkey_end:
_answerkey_ptr: dd _answerkey
_passbit:   dd 1
.end_context
printstring:
.start_context
        subleq Z, Z, $+4
_bp:    dd 0
        subleq _bp, _bp
        subleq  sp, Z
        subleq Z, _bp
        subleq Z, Z, Z
        subleq Z, Z, $+4
_string:    dd 0
.start_context
        subleq Z, Z, $+4
_DECLARE_FUNCTION_ARGUMENT_bp_offset:   dd -2
        subleq Z, Z, $+4
_DECLARE_FUNCTION_ARGUMENT_tmp: dd 0
        subleq _DECLARE_FUNCTION_ARGUMENT_tmp, _DECLARE_FUNCTION_ARGUMENT_tmp
        subleq _DECLARE_FUNCTION_ARGUMENT_bp_offset, Z
        subleq Z, _DECLARE_FUNCTION_ARGUMENT_tmp
        subleq Z, Z, Z
        subleq _bp, Z
```

```
        subleq Z, _DECLARE_FUNCTION_ARGUMENT_tmp
        subleq Z, Z
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq _DECLARE_FUNCTION_ARGUMENT_tmp, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq _string, _string
_DEREF_target:  subleq Z, Z
                subleq Z, _string
                subleq Z, Z
.end_context
.end_context
        subleq Z, Z, $+4
_char:  dd 0
_LOOP_START_LOOPID_print_loop:
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq _string, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq  _char,  _char
_DEREF_target:  subleq Z, Z
                subleq Z,  _char
                subleq Z, Z
.end_context
.start_context
.start_context
            subleq _char, Z, _BEZ_true
            subleq Z, Z, _BEZ_false
_BEZ_true:  subleq Z, Z
            subleq Z, _char, _BNEZ_false
_BEZ_false:
.end_context
        subleq Z, Z,  _print_loop_continue
_BNEZ_false:
.end_context
        subleq Z, Z, _LOOP_END_LOOPID_print_loop
_print_loop_continue:
        subleq OUTPUT, OUTPUT
        subleq _char, Z
        subleq Z, OUTPUT
        subleq Z, Z, Z
.start_context
        subleq   OUTPUT_READY,   OUTPUT_READY
        subleq _SETVALUE_tmp, Z
        subleq Z,   OUTPUT_READY
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
```

```
_SETVALUE_tmp:  dd 1
_SETVALUE_end:
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _string
        subleq Z, Z
.end_context
        subleq Z, Z, _LOOP_START_LOOPID_print_loop
_LOOP_END_LOOPID_print_loop:
.start_context
.start_context
        subleq Z, Z, _DEC_sub
_DEC_ONE:   dd 1
_DEC_sub:
        subleq _DEC_ONE, sp
.end_context
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq sp, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq  _RET_tailjump+2,  _RET_tailjump+2
_DEREF_target:  subleq Z, Z
                subleq Z,  _RET_tailjump+2
                subleq Z, Z
.end_context
_RET_tailjump:  subleq Z, Z, Z
.end_context
.end_context
readinputstring:
.start_context
        subleq Z, Z, $+4
_bp:    dd 0
        subleq _bp, _bp
        subleq  sp, Z
        subleq Z, _bp
        subleq Z, Z, Z
        subleq Z, Z, $+4
_bufptr:    dd 0
.start_context
        subleq Z, Z, $+4
_DECLARE_FUNCTION_ARGUMENT_bp_offset:   dd -2
        subleq Z, Z, $+4
_DECLARE_FUNCTION_ARGUMENT_tmp: dd 0
        subleq _DECLARE_FUNCTION_ARGUMENT_tmp, _DECLARE_FUNCTION_ARGUMENT_tmp
```

```
        subleq _DECLARE_FUNCTION_ARGUMENT_bp_offset, Z
        subleq Z, _DECLARE_FUNCTION_ARGUMENT_tmp
        subleq Z, Z, Z
        subleq _bp, Z
        subleq Z, _DECLARE_FUNCTION_ARGUMENT_tmp
        subleq Z, Z
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq _DECLARE_FUNCTION_ARGUMENT_tmp, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq _bufptr, _bufptr
_DEREF_target:  subleq Z, Z
                subleq Z, _bufptr
                subleq Z, Z
.end_context
.end_context
        subleq Z, Z, $+4
_buffersize:    dd 0
.start_context
        subleq Z, Z, $+4
_DECLARE_FUNCTION_ARGUMENT_bp_offset:   dd -3
        subleq Z, Z, $+4
_DECLARE_FUNCTION_ARGUMENT_tmp: dd 0
        subleq _DECLARE_FUNCTION_ARGUMENT_tmp, _DECLARE_FUNCTION_ARGUMENT_tmp
        subleq _DECLARE_FUNCTION_ARGUMENT_bp_offset, Z
        subleq Z, _DECLARE_FUNCTION_ARGUMENT_tmp
        subleq Z, Z, Z
        subleq _bp, Z
        subleq Z, _DECLARE_FUNCTION_ARGUMENT_tmp
        subleq Z, Z
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq _DECLARE_FUNCTION_ARGUMENT_tmp, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq _buffersize, _buffersize
_DEREF_target:  subleq Z, Z
                subleq Z, _buffersize
                subleq Z, Z
.end_context
.end_context
        subleq Z, Z, $+4
_i: dd 0
        subleq Z, Z, $+4
_newline:   dd 10
        subleq Z, Z, $+4
_tmp:   dd 0
_FORLOOP_START_LOOPID_readinputstring_loop:
```

```
.start_context
        subleq Z, Z, _FORLOOP_START_compare
_FORLOOP_START_compare:
.start_context
        subleq _BGEQ_tmp, _BGEQ_tmp
        subleq _i, Z
        subleq Z, _BGEQ_tmp
        subleq Z, Z, Z
        subleq _buffersize, _BGEQ_tmp
.start_context
            subleq _BGEQ_tmp, Z, _BLZ_false
            subleq Z, Z, _BGEQ_false
_BLZ_false: subleq Z, Z
.end_context
        subleq Z, Z, _FORLOOP_END_LOOPID_readinputstring_loop
_BGEQ_tmp:  dd 0
_BGEQ_false:
.end_context
.end_context
.start_context
        subleq   INPUT_READY,   INPUT_READY
        subleq _SETVALUE_tmp, Z
        subleq Z,   INPUT_READY
        subleq Z, Z, Z
        subleq Z, Z, _SETVALUE_end
_SETVALUE_tmp:  dd 1
_SETVALUE_end:
.end_context
        subleq _tmp, _tmp
        subleq INPUT, Z
        subleq Z, _tmp
        subleq Z, Z, Z
.start_context
        subleq _BNEQ_tmp, _BNEQ_tmp
        subleq _tmp, Z
        subleq Z, _BNEQ_tmp
        subleq Z, Z, Z
        subleq _newline, _BNEQ_tmp
.start_context
            subleq _BNEQ_tmp, Z, _BEZ_true
            subleq Z, Z, _BEZ_false
_BEZ_true:  subleq Z, Z
            subleq Z, _BNEQ_tmp, _BNEQ_false
_BEZ_false:
.end_context
        subleq Z, Z, _not_newline
_BNEQ_tmp:  dd 0
_BNEQ_false:
.end_context
```

```
        subleq _tmp, _tmp, 0
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq _bufptr, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq _bufptr, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq _bufptr, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
        subleq Z, Z, _readinputstring_finished
_not_newline:
.start_context
        subleq  _DEREF_DST_MOV_x,  _DEREF_DST_MOV_x
        subleq _bufptr, Z
        subleq Z,  _DEREF_DST_MOV_x
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_x+1,  _DEREF_DST_MOV_x+1
        subleq _bufptr, Z
        subleq Z,  _DEREF_DST_MOV_x+1
        subleq Z, Z, Z
        subleq  _DEREF_DST_MOV_y+1,  _DEREF_DST_MOV_y+1
        subleq _bufptr, Z
        subleq Z,  _DEREF_DST_MOV_y+1
        subleq Z, Z, Z
_DEREF_DST_MOV_x:   subleq Z, Z
                subleq _tmp, Z
_DEREF_DST_MOV_y:   subleq Z, Z
                subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
_INC_ONE:   dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _bufptr
        subleq Z, Z
.end_context
.start_context
        subleq Z, Z, _INC_add
```

```
_INC_ONE:    dd 1
_INC_add:
        subleq _INC_ONE, Z
        subleq Z, _i
        subleq Z, Z
.end_context
        subleq Z, Z, _FORLOOP_START_LOOPID_readinputstring_loop
_FORLOOP_END_LOOPID_readinputstring_loop:
_readinputstring_finished:
.start_context
.start_context
        subleq Z, Z, _DEC_sub
_DEC_ONE:    dd 1
_DEC_sub:
        subleq _DEC_ONE, sp
.end_context
.start_context
        subleq _DEREF_target, _DEREF_target
        subleq sp, Z
        subleq Z, _DEREF_target
        subleq Z, Z, Z
                subleq  _RET_tailjump+2,  _RET_tailjump+2
_DEREF_target:  subleq Z, Z
                subleq Z,  _RET_tailjump+2
                subleq Z, Z
.end_context
_RET_tailjump:  subleq Z, Z, Z
.end_context
.end_context
```

Appendix 3: C Array of Subleq program with source line comments

```
enum program_labels_t {
    SUBLEQ_LABEL_ending = 2471 ,
    SUBLEQ_LABEL_correct_key = 1955 ,
    SUBLEQ_LABEL_OUTPUT_READY = 4 ,
    SUBLEQ_LABEL_readinputstring = 3955 ,
    SUBLEQ_LABEL_tmp = 1750 ,
    SUBLEQ_LABEL_num_failstrings = 272 ,
    SUBLEQ_LABEL_start = 1123 ,
    SUBLEQ_LABEL_printstring = 3758 ,
    SUBLEQ_LABEL_you_entered = 162 ,
    SUBLEQ_LABEL_failstring5 = 676 ,
    SUBLEQ_LABEL_check_string = 2474 ,
    SUBLEQ_LABEL_failstring8 = 999 ,
    SUBLEQ_LABEL_failstring6 = 738 ,
    SUBLEQ_LABEL_failstring7 = 849 ,
    SUBLEQ_LABEL_failstring4 = 566 ,
    SUBLEQ_LABEL_inputbuf = 77 ,
    SUBLEQ_LABEL_failstring2 = 374 ,
    SUBLEQ_LABEL_failstring3 = 450 ,
    SUBLEQ_LABEL_failstring1 = 282 ,
    SUBLEQ_LABEL_key = 176 ,
    SUBLEQ_LABEL_OUTPUT = 2 ,
    SUBLEQ_LABEL_INPUT = 1 ,
    SUBLEQ_LABEL_Z = 0 ,
    SUBLEQ_LABEL_stack = 157 ,
    SUBLEQ_LABEL_INPUT_READY = 3 ,
    SUBLEQ_LABEL_Prompt = 5 ,
    SUBLEQ_LABEL_rickroll = 52 ,
    SUBLEQ_LABEL_sp = 161 ,
    SUBLEQ_LABEL_winstring = 177 ,
    SUBLEQ_LABEL_failstring = 1746 ,
    SUBLEQ_LABEL_inputbuf_end = 157 ,
    SUBLEQ_LABEL_failstring_selector = 273 ,
    SUBLEQ_LABEL_retval = 76 ,
    SUBLEQ_LABEL_failstring_array = 274 ,
    SUBLEQ_LABEL_atflareon = 257
};

int program[] = {
    0 ,     // (0000) Z: dd 0
    0 ,     // (0001) INPUT: dd 0
    0 ,     // (0002) OUTPUT: dd 0
    0 ,     // (0003) INPUT_READY: dd 0
    0 ,     // (0004) OUTPUT_READY: dd 0
    87, 101, 108, 99, 111, 109, 101, 32, 116, 111, 32, 116, 104, 101, 32, 109, 97,
103, 105, 99, 32, 98, 111, 120, 46, 10, 69, 110, 116, 101, 114, 32, 116, 104, 101,
```

```
32, 112, 97, 115, 115, 119, 111, 114, 100, 58, 32, 0 ,    // (0005) Prompt: dd 87,
101, 108, 99, 111, 109, 101, 32, 116, 111, 32, 116, 104, 101, 32, 109, 97, 103,
105, 99, 32, 98, 111, 120, 46, 10, 69, 110, 116, 101, 114, 32, 116, 104, 101, 32,
112, 97, 115, 115, 119, 111, 114, 100, 58, 32, 0
    104, 116, 116, 112, 58, 47, 47, 98, 105, 116, 108, 121, 46, 99, 111, 109, 47,
57, 56, 75, 56, 101, 72, 0 ,      // (0052) rickroll: dd 104, 116, 116, 112, 58,
47, 47, 98, 105, 116, 108, 121, 46, 99, 111, 109, 47, 57, 56, 75, 56, 101, 72, 0
    0 ,     // (0076) retval: dd 0
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
// (0077) inputbuf: dd 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0
    0, 0, 0, 0 ,    // (0157) stack: dd 0, 0, 0, 0
    157 ,   // (0161) sp: dd 157
    89, 111, 117, 32, 69, 110, 116, 101, 114, 101, 100, 58, 32, 0 ,     // (0162)
you_entered: dd 89, 111, 117, 32, 69, 110, 116, 101, 114, 101, 100, 58, 32, 0
    291 ,   // (0176) key: dd 291
    84, 104, 101, 32, 112, 97, 115, 115, 119, 111, 114, 100, 32, 105, 115, 32, 99,
111, 114, 114, 101, 99, 116, 46, 32, 71, 111, 111, 100, 32, 74, 111, 98, 46, 32,
89, 111, 117, 32, 119, 105, 110, 32, 70, 108, 97, 114, 101, 45, 79, 110, 32, 52,
46, 10, 89, 111, 117, 114, 32, 107, 101, 121, 32, 116, 111, 32, 118, 105, 99, 116,
111, 114, 121, 32, 105, 115, 58, 32, 0 ,    // (0177) winstring: dd 84, 104, 101,
32, 112, 97, 115, 115, 119, 111, 114, 100, 32, 105, 115, 32, 99, 111, 114, 114,
101, 99, 116, 46, 32, 71, 111, 111, 100, 32, 74, 111, 98, 46, 32, 89, 111, 117, 32,
119, 105, 110, 32, 70, 108, 97, 114, 101, 45, 79, 110, 32, 52, 46, 10, 89, 111,
117, 114, 32, 107, 101, 121, 32, 116, 111, 32, 118, 105, 99, 116, 111, 114, 121,
32, 105, 115, 58, 32, 0
    64, 102, 108, 97, 114, 101, 45, 111, 110, 46, 99, 111, 109, 10, 0 ,     //
(0257) atflareon: dd 64, 102, 108, 97, 114, 101, 45, 111, 110, 46, 99, 111, 109,
10, 0
    8 ,     // (0272) num_failstrings: dd 8
    3 ,     // (0273) failstring_selector: dd 3
    282, 374, 450, 566, 676, 738, 849, 999 ,    // (0274) failstring_array: dd 282,
374, 450, 566, 676, 738, 849, 999
    84, 104, 101, 32, 112, 97, 115, 115, 119, 111, 114, 100, 32, 105, 115, 32, 105,
110, 99, 111, 114, 114, 101, 99, 116, 46, 32, 84, 114, 121, 32, 114, 101, 118, 101,
114, 115, 101, 32, 101, 110, 103, 105, 110, 101, 101, 114, 105, 110, 103, 32, 116,
104, 101, 32, 98, 105, 110, 97, 114, 121, 32, 116, 111, 32, 100, 105, 115, 99, 111,
118, 101, 114, 32, 116, 104, 101, 32, 99, 111, 114, 114, 101, 99, 116, 32, 111,
110, 101, 46, 10, 0 ,    // (0282) failstring1: dd 84, 104, 101, 32, 112, 97, 115,
115, 119, 111, 114, 100, 32, 105, 115, 32, 105, 110, 99, 111, 114, 114, 101, 99,
116, 46, 32, 84, 114, 121, 32, 114, 101, 118, 101, 114, 115, 101, 32, 101, 110,
103, 105, 110, 101, 101, 114, 105, 110, 103, 32, 116, 104, 101, 32, 98, 105, 110,
97, 114, 121, 32, 116, 111, 32, 100, 105, 115, 99, 111, 118, 101, 114, 32, 116,
104, 101, 32, 99, 111, 114, 114, 101, 99, 116, 32, 111, 110, 101, 46, 10, 0
```

```
    73, 110, 99, 111, 114, 114, 101, 99, 116, 32, 112, 97, 115, 115, 119, 111, 114,
100, 46, 32, 72, 97, 118, 101, 32, 121, 111, 117, 32, 116, 114, 105, 101, 100, 32,
116, 117, 114, 110, 105, 110, 103, 32, 121, 111, 117, 114, 32, 99, 111, 109, 112,
117, 116, 101, 114, 32, 111, 102, 102, 32, 97, 110, 100, 32, 111, 110, 32, 97, 103,
97, 105, 110, 63, 10, 0 ,      // (0374) failstring2: dd 73, 110, 99, 111, 114,
114, 101, 99, 116, 32, 112, 97, 115, 115, 119, 111, 114, 100, 46, 32, 72, 97, 118,
101, 32, 121, 111, 117, 32, 116, 114, 105, 101, 100, 32, 116, 117, 114, 110, 105,
110, 103, 32, 121, 111, 117, 114, 32, 99, 111, 109, 112, 117, 116, 101, 114, 32,
111, 102, 102, 32, 97, 110, 100, 32, 111, 110, 32, 97, 103, 97, 105, 110, 63, 10, 0
    84, 104, 97, 116, 32, 105, 115, 32, 116, 104, 101, 32, 119, 114, 111, 110, 103,
32, 112, 97, 115, 115, 119, 111, 114, 100, 46, 32, 68, 101, 108, 101, 116, 101, 32,
116, 104, 101, 32, 67, 58, 92, 87, 105, 110, 100, 111, 119, 115, 92, 83, 121, 115,
116, 101, 109, 51, 50, 32, 102, 111, 108, 100, 101, 114, 32, 116, 111, 32, 109, 97,
107, 101, 32, 121, 111, 117, 114, 32, 99, 111, 109, 112, 117, 116, 101, 114, 32,
114, 117, 110, 32, 102, 97, 115, 116, 101, 114, 44, 32, 97, 110, 100, 32, 116, 114,
121, 32, 97, 103, 97, 105, 110, 46, 10, 0 ,     // (0450) failstring3: dd 84, 104,
97, 116, 32, 105, 115, 32, 116, 104, 101, 32, 119, 114, 111, 110, 103, 32, 112, 97,
115, 115, 119, 111, 114, 100, 46, 32, 68, 101, 108, 101, 116, 101, 32, 116, 104,
101, 32, 67, 58, 92, 87, 105, 110, 100, 111, 119, 115, 92, 83, 121, 115, 116, 101,
109, 51, 50, 32, 102, 111, 108, 100, 101, 114, 32, 116, 111, 32, 109, 97, 107, 101,
32, 121, 111, 117, 114, 32, 99, 111, 109, 112, 117, 116, 101, 114, 32, 114, 117,
110, 32, 102, 97, 115, 116, 101, 114, 44, 32, 97, 110, 100, 32, 116, 114, 121, 32,
97, 103, 97, 105, 110, 46, 10, 0
    80, 97, 115, 115, 119, 111, 114, 100, 32, 70, 97, 105, 108, 101, 100, 46, 32,
89, 111, 117, 32, 104, 97, 118, 101, 32, 117, 115, 101, 100, 32, 105, 110, 115,
117, 102, 102, 105, 99, 105, 101, 110, 116, 32, 99, 111, 109, 112, 117, 116, 101,
114, 32, 115, 99, 105, 101, 110, 99, 101, 46, 32, 72, 97, 118, 101, 32, 121, 111,
117, 32, 99, 111, 110, 115, 105, 100, 101, 114, 101, 100, 32, 97, 32, 99, 97, 114,
101, 101, 114, 32, 105, 110, 32, 115, 97, 108, 101, 115, 32, 105, 110, 115, 116,
101, 97, 100, 63, 10, 0 ,     // (0566) failstring4: dd 80, 97, 115, 115, 119, 111,
114, 100, 32, 70, 97, 105, 108, 101, 100, 46, 32, 89, 111, 117, 32, 104, 97, 118,
101, 32, 117, 115, 101, 100, 32, 105, 110, 115, 117, 102, 102, 105, 99, 105, 101,
110, 116, 32, 99, 111, 109, 112, 117, 116, 101, 114, 32, 115, 99, 105, 101, 110,
99, 101, 46, 32, 72, 97, 118, 101, 32, 121, 111, 117, 32, 99, 111, 110, 115, 105,
100, 101, 114, 101, 100, 32, 97, 32, 99, 97, 114, 101, 101, 114, 32, 105, 110, 32,
115, 97, 108, 101, 115, 32, 105, 110, 115, 116, 101, 97, 100, 63, 10, 0
    73, 110, 99, 111, 114, 114, 101, 99, 116, 32, 112, 97, 115, 115, 119, 111, 114,
100, 46, 32, 89, 111, 117, 114, 32, 97, 99, 99, 111, 117, 110, 116, 32, 115, 104,
111, 117, 108, 100, 32, 112, 114, 111, 98, 97, 98, 108, 121, 32, 98, 101, 32, 100,
101, 108, 101, 116, 101, 100, 46, 10, 0 ,      // (0676) failstring5: dd 73, 110,
99, 111, 114, 114, 101, 99, 116, 32, 112, 97, 115, 115, 119, 111, 114, 100, 46, 32,
89, 111, 117, 114, 32, 97, 99, 99, 111, 117, 110, 116, 32, 115, 104, 111, 117, 108,
100, 32, 112, 114, 111, 98, 97, 98, 108, 121, 32, 98, 101, 32, 100, 101, 108, 101,
116, 101, 100, 46, 10, 0
    67, 111, 109, 112, 108, 101, 116, 101, 32, 112, 97, 115, 115, 119, 111, 114,
100, 32, 102, 97, 105, 108, 117, 114, 101, 46, 32, 84, 114, 121, 32, 97, 103, 97,
105, 110, 32, 111, 114, 32, 119, 97, 105, 116, 32, 117, 110, 116, 105, 108, 32,
110, 101, 120, 116, 32, 121, 101, 97, 114, 32, 97, 110, 100, 32, 104, 111, 112,
101, 44, 32, 105, 110, 32, 118, 97, 105, 110, 44, 32, 102, 111, 114, 32, 97, 110,
```

```
32, 101, 118, 101, 110, 32, 101, 97, 115, 105, 101, 114, 32, 99, 104, 97, 108, 108,
101, 110, 103, 101, 46, 10, 0 ,    // (0738) failstring6: dd 67, 111, 109, 112,
108, 101, 116, 101, 32, 112, 97, 115, 115, 119, 111, 114, 100, 32, 102, 97, 105,
108, 117, 114, 101, 46, 32, 84, 114, 121, 32, 97, 103, 97, 105, 110, 32, 111, 114,
32, 119, 97, 105, 116, 32, 117, 110, 116, 105, 108, 32, 110, 101, 120, 116, 32,
121, 101, 97, 114, 32, 97, 110, 100, 32, 104, 111, 112, 101, 44, 32, 105, 110, 32,
118, 97, 105, 110, 44, 32, 102, 111, 114, 32, 97, 110, 32, 101, 118, 101, 110, 32,
101, 97, 115, 105, 101, 114, 32, 99, 104, 97, 108, 108, 101, 110, 103, 101, 46, 10,
0
    84, 104, 97, 116, 32, 105, 115, 32, 110, 111, 116, 32, 116, 104, 101, 32, 112,
97, 115, 115, 119, 111, 114, 100, 46, 32, 89, 111, 117, 32, 100, 105, 100, 32, 110,
111, 116, 32, 115, 111, 108, 118, 101, 32, 116, 104, 101, 32, 99, 104, 97, 108,
108, 101, 110, 103, 101, 46, 32, 67, 111, 110, 115, 105, 100, 101, 114, 32, 105,
110, 115, 116, 97, 108, 108, 105, 110, 103, 32, 97, 32, 100, 105, 115, 97, 115,
115, 101, 109, 98, 108, 101, 114, 32, 115, 117, 99, 104, 32, 97, 115, 32, 73, 68,
65, 32, 80, 114, 111, 44, 32, 97, 110, 100, 32, 116, 114, 121, 32, 97, 103, 97,
105, 110, 44, 32, 111, 110, 108, 121, 32, 104, 97, 114, 100, 101, 114, 32, 110,
101, 120, 116, 32, 116, 105, 109, 101, 46, 10, 0 ,    // (0849) failstring7: dd 84,
104, 97, 116, 32, 105, 115, 32, 110, 111, 116, 32, 116, 104, 101, 32, 112, 97, 115,
115, 119, 111, 114, 100, 46, 32, 89, 111, 117, 32, 100, 105, 100, 32, 110, 111,
116, 32, 115, 111, 108, 118, 101, 32, 116, 104, 101, 32, 99, 104, 97, 108, 108,
101, 110, 103, 101, 46, 32, 67, 111, 110, 115, 105, 100, 101, 114, 32, 105, 110,
115, 116, 97, 108, 108, 105, 110, 103, 32, 97, 32, 100, 105, 115, 97, 115, 115,
101, 109, 98, 108, 101, 114, 32, 115, 117, 99, 104, 32, 97, 115, 32, 73, 68, 65,
32, 80, 114, 111, 44, 32, 97, 110, 100, 32, 116, 114, 121, 32, 97, 103, 97, 105,
110, 44, 32, 111, 110, 108, 121, 32, 104, 97, 114, 100, 101, 114, 32, 110, 101,
120, 116, 32, 116, 105, 109, 101, 46, 10, 0
    89, 111, 117, 32, 100, 105, 100, 32, 110, 111, 116, 32, 103, 117, 101, 115,
115, 32, 116, 104, 101, 32, 112, 97, 115, 115, 119, 111, 114, 100, 46, 32, 67, 111,
110, 115, 105, 100, 101, 114, 32, 98, 114, 117, 116, 101, 45, 102, 111, 114, 99,
101, 32, 103, 117, 101, 115, 115, 105, 110, 103, 32, 117, 115, 105, 110, 103, 32,
97, 32, 115, 99, 114, 105, 112, 116, 44, 32, 97, 110, 100, 32, 108, 101, 116, 116,
105, 110, 103, 32, 105, 116, 32, 114, 117, 110, 32, 102, 111, 114, 32, 97, 32, 102,
101, 119, 32, 116, 114, 105, 108, 108, 105, 111, 110, 32, 121, 101, 97, 114, 115,
46, 10, 0 ,    // (0999) failstring8: dd 89, 111, 117, 32, 100, 105, 100, 32, 110,
111, 116, 32, 103, 117, 101, 115, 115, 32, 116, 104, 101, 32, 112, 97, 115, 115,
119, 111, 114, 100, 46, 32, 67, 111, 110, 115, 105, 100, 101, 114, 32, 98, 114,
117, 116, 101, 45, 102, 111, 114, 99, 101, 32, 103, 117, 101, 115, 115, 105, 110,
103, 32, 117, 115, 105, 110, 103, 32, 97, 32, 115, 99, 114, 105, 112, 116, 44, 32,
97, 110, 100, 32, 108, 101, 116, 116, 105, 110, 103, 32, 105, 116, 32, 114, 117,
110, 32, 102, 111, 114, 32, 97, 32, 102, 101, 119, 32, 116, 114, 105, 108, 108,
105, 111, 110, 32, 121, 101, 97, 114, 115, 46, 10, 0
    0, 0, 1127 ,    // (1123) start: subleq 0, 0, 1127
    0 ,     // (1126) _PTR_REF_tmp: dd 0
    1126, 1126, 0 ,     // (1127) subleq 1126, 1126, 0
    1142, 0, 0 ,    // (1130) subleq 1142, 0, 0
    0, 1126, 0 ,    // (1133) subleq 0, 1126, 0
    0, 0, 0 ,   // (1136) subleq 0, 0, 0
    0, 0, 1143 ,    // (1139) subleq 0, 0, 1143
```

```
    5 ,     // (1142) _SETVALUE_tmp: dd 5
    1179, 1179, 0 ,     // (1143) subleq 1179, 1179, 0
    161, 0, 0 ,     // (1146) subleq 161, 0, 0
    0, 1179, 0 ,    // (1149) subleq 0, 1179, 0
    0, 0, 0 ,   // (1152) subleq 0, 0, 0
    1180, 1180, 0 ,     // (1155) subleq 1180, 1180, 0
    161, 0, 0 ,     // (1158) subleq 161, 0, 0
    0, 1180, 0 ,    // (1161) subleq 0, 1180, 0
    0, 0, 0 ,   // (1164) subleq 0, 0, 0
    1186, 1186, 0 ,     // (1167) subleq 1186, 1186, 0
    161, 0, 0 ,     // (1170) subleq 161, 0, 0
    0, 1186, 0 ,    // (1173) subleq 0, 1186, 0
    0, 0, 0 ,   // (1176) subleq 0, 0, 0
    0, 0, 0 ,   // (1179) _DEREF_DST_MOV_x: subleq 0, 0, 0
    1126, 0, 0 ,    // (1182) subleq 1126, 0, 0
    0, 0, 0 ,   // (1185) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (1188) subleq 0, 0, 0
    0, 0, 1195 ,    // (1191) subleq 0, 0, 1195
    1 ,     // (1194) _INC_ONE: dd 1
    1194, 0, 0 ,    // (1195) _INC_add: subleq 1194, 0, 0
    0, 161, 0 ,     // (1198) subleq 0, 161, 0
    0, 0, 0 ,   // (1201) subleq 0, 0, 0
    0, 0, 1208 ,    // (1204) subleq 0, 0, 1208
    0 ,     // (1207) _PTR_REF_tmp: dd 0
    1207, 1207, 0 ,     // (1208) subleq 1207, 1207, 0
    1223, 0, 0 ,    // (1211) subleq 1223, 0, 0
    0, 1207, 0 ,    // (1214) subleq 0, 1207, 0
    0, 0, 0 ,   // (1217) subleq 0, 0, 0
    0, 0, 1224 ,    // (1220) subleq 0, 0, 1224
    1288 ,      // (1223) _SETVALUE_tmp: dd 1288
    1260, 1260, 0 ,     // (1224) subleq 1260, 1260, 0
    161, 0, 0 ,     // (1227) subleq 161, 0, 0
    0, 1260, 0 ,    // (1230) subleq 0, 1260, 0
    0, 0, 0 ,   // (1233) subleq 0, 0, 0
    1261, 1261, 0 ,     // (1236) subleq 1261, 1261, 0
    161, 0, 0 ,     // (1239) subleq 161, 0, 0
    0, 1261, 0 ,    // (1242) subleq 0, 1261, 0
    0, 0, 0 ,   // (1245) subleq 0, 0, 0
    1267, 1267, 0 ,     // (1248) subleq 1267, 1267, 0
    161, 0, 0 ,     // (1251) subleq 161, 0, 0
    0, 1267, 0 ,    // (1254) subleq 0, 1267, 0
    0, 0, 0 ,   // (1257) subleq 0, 0, 0
    0, 0, 0 ,   // (1260) _DEREF_DST_MOV_x: subleq 0, 0, 0
    1207, 0, 0 ,    // (1263) subleq 1207, 0, 0
    0, 0, 0 ,   // (1266) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (1269) subleq 0, 0, 0
    0, 0, 1276 ,    // (1272) subleq 0, 0, 1276
    1 ,     // (1275) _INC_ONE: dd 1
    1275, 0, 0 ,    // (1276) _INC_add: subleq 1275, 0, 0
```

```
    0, 161, 0 ,     // (1279) subleq 0, 161, 0
    0, 0, 0 ,   // (1282) subleq 0, 0, 0
    0, 0, 3758 ,    // (1285) subleq 0, 0, 3758
    1294, 161, 0 ,      // (1288) subleq 1294, 161, 0
    0, 0, 1295 ,    // (1291) subleq 0, 0, 1295
    1 ,     // (1294) _SUB_LITERAL_val: dd 1
    0, 0, 1299 ,    // (1295) subleq 0, 0, 1299
    80 ,    // (1298) _PUSH_LITERAL_tmp: dd 80
    1335, 1335, 0 ,     // (1299) subleq 1335, 1335, 0
    161, 0, 0 ,     // (1302) subleq 161, 0, 0
    0, 1335, 0 ,    // (1305) subleq 0, 1335, 0
    0, 0, 0 ,   // (1308) subleq 0, 0, 0
    1336, 1336, 0 ,     // (1311) subleq 1336, 1336, 0
    161, 0, 0 ,     // (1314) subleq 161, 0, 0
    0, 1336, 0 ,    // (1317) subleq 0, 1336, 0
    0, 0, 0 ,   // (1320) subleq 0, 0, 0
    1342, 1342, 0 ,     // (1323) subleq 1342, 1342, 0
    161, 0, 0 ,     // (1326) subleq 161, 0, 0
    0, 1342, 0 ,    // (1329) subleq 0, 1342, 0
    0, 0, 0 ,   // (1332) subleq 0, 0, 0
    0, 0, 0 ,   // (1335) _DEREF_DST_MOV_x: subleq 0, 0, 0
    1298, 0, 0 ,    // (1338) subleq 1298, 0, 0
    0, 0, 0 ,   // (1341) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (1344) subleq 0, 0, 0
    0, 0, 1351 ,    // (1347) subleq 0, 0, 1351
    1 ,     // (1350) _INC_ONE: dd 1
    1350, 0, 0 ,    // (1351) _INC_add: subleq 1350, 0, 0
    0, 161, 0 ,     // (1354) subleq 0, 161, 0
    0, 0, 0 ,   // (1357) subleq 0, 0, 0
    0, 0, 1364 ,    // (1360) subleq 0, 0, 1364
    0 ,     // (1363) _PTR_REF_tmp: dd 0
    1363, 1363, 0 ,     // (1364) subleq 1363, 1363, 0
    1379, 0, 0 ,    // (1367) subleq 1379, 0, 0
    0, 1363, 0 ,    // (1370) subleq 0, 1363, 0
    0, 0, 0 ,   // (1373) subleq 0, 0, 0
    0, 0, 1380 ,    // (1376) subleq 0, 0, 1380
    77 ,    // (1379) _SETVALUE_tmp: dd 77
    1416, 1416, 0 ,     // (1380) subleq 1416, 1416, 0
    161, 0, 0 ,     // (1383) subleq 161, 0, 0
    0, 1416, 0 ,    // (1386) subleq 0, 1416, 0
    0, 0, 0 ,   // (1389) subleq 0, 0, 0
    1417, 1417, 0 ,     // (1392) subleq 1417, 1417, 0
    161, 0, 0 ,     // (1395) subleq 161, 0, 0
    0, 1417, 0 ,    // (1398) subleq 0, 1417, 0
    0, 0, 0 ,   // (1401) subleq 0, 0, 0
    1423, 1423, 0 ,     // (1404) subleq 1423, 1423, 0
    161, 0, 0 ,     // (1407) subleq 161, 0, 0
    0, 1423, 0 ,    // (1410) subleq 0, 1423, 0
    0, 0, 0 ,   // (1413) subleq 0, 0, 0
```

```
    0, 0, 0 ,   // (1416) _DEREF_DST_MOV_x: subleq 0, 0, 0
    1363, 0, 0 ,    // (1419) subleq 1363, 0, 0
    0, 0, 0 ,   // (1422) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (1425) subleq 0, 0, 0
    0, 0, 1432 ,    // (1428) subleq 0, 0, 1432
    1 ,     // (1431) _INC_ONE: dd 1
    1431, 0, 0 ,    // (1432) _INC_add: subleq 1431, 0, 0
    0, 161, 0 ,     // (1435) subleq 0, 161, 0
    0, 0, 0 ,   // (1438) subleq 0, 0, 0
    0, 0, 1445 ,    // (1441) subleq 0, 0, 1445
    0 ,     // (1444) _PTR_REF_tmp: dd 0
    1444, 1444, 0 ,     // (1445) subleq 1444, 1444, 0
    1460, 0, 0 ,    // (1448) subleq 1460, 0, 0
    0, 1444, 0 ,    // (1451) subleq 0, 1444, 0
    0, 0, 0 ,   // (1454) subleq 0, 0, 0
    0, 0, 1461 ,    // (1457) subleq 0, 0, 1461
    1525 ,      // (1460) _SETVALUE_tmp: dd 1525
    1497, 1497, 0 ,     // (1461) subleq 1497, 1497, 0
    161, 0, 0 ,     // (1464) subleq 161, 0, 0
    0, 1497, 0 ,    // (1467) subleq 0, 1497, 0
    0, 0, 0 ,   // (1470) subleq 0, 0, 0
    1498, 1498, 0 ,     // (1473) subleq 1498, 1498, 0
    161, 0, 0 ,     // (1476) subleq 161, 0, 0
    0, 1498, 0 ,    // (1479) subleq 0, 1498, 0
    0, 0, 0 ,   // (1482) subleq 0, 0, 0
    1504, 1504, 0 ,     // (1485) subleq 1504, 1504, 0
    161, 0, 0 ,     // (1488) subleq 161, 0, 0
    0, 1504, 0 ,    // (1491) subleq 0, 1504, 0
    0, 0, 0 ,   // (1494) subleq 0, 0, 0
    0, 0, 0 ,   // (1497) _DEREF_DST_MOV_x: subleq 0, 0, 0
    1444, 0, 0 ,    // (1500) subleq 1444, 0, 0
    0, 0, 0 ,   // (1503) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (1506) subleq 0, 0, 0
    0, 0, 1513 ,    // (1509) subleq 0, 0, 1513
    1 ,     // (1512) _INC_ONE: dd 1
    1512, 0, 0 ,    // (1513) _INC_add: subleq 1512, 0, 0
    0, 161, 0 ,     // (1516) subleq 0, 161, 0
    0, 0, 0 ,   // (1519) subleq 0, 0, 0
    0, 0, 3955 ,    // (1522) subleq 0, 0, 3955
    1531, 161, 0 ,      // (1525) subleq 1531, 161, 0
    0, 0, 1532 ,    // (1528) subleq 0, 0, 1532
    2 ,     // (1531) _SUB_LITERAL_val: dd 2
    0, 0, 1536 ,    // (1532) subleq 0, 0, 1536
    0 ,     // (1535) _PTR_REF_tmp: dd 0
    1535, 1535, 0 ,     // (1536) subleq 1535, 1535, 0
    1551, 0, 0 ,    // (1539) subleq 1551, 0, 0
    0, 1535, 0 ,    // (1542) subleq 0, 1535, 0
    0, 0, 0 ,   // (1545) subleq 0, 0, 0
    0, 0, 1552 ,    // (1548) subleq 0, 0, 1552
```

```
    77 ,    // (1551) _SETVALUE_tmp: dd 77
    1588, 1588, 0 ,     // (1552) subleq 1588, 1588, 0
    161, 0, 0 ,     // (1555) subleq 161, 0, 0
    0, 1588, 0 ,    // (1558) subleq 0, 1588, 0
    0, 0, 0 ,   // (1561) subleq 0, 0, 0
    1589, 1589, 0 ,     // (1564) subleq 1589, 1589, 0
    161, 0, 0 ,     // (1567) subleq 161, 0, 0
    0, 1589, 0 ,    // (1570) subleq 0, 1589, 0
    0, 0, 0 ,   // (1573) subleq 0, 0, 0
    1595, 1595, 0 ,     // (1576) subleq 1595, 1595, 0
    161, 0, 0 ,     // (1579) subleq 161, 0, 0
    0, 1595, 0 ,    // (1582) subleq 0, 1595, 0
    0, 0, 0 ,   // (1585) subleq 0, 0, 0
    0, 0, 0 ,   // (1588) _DEREF_DST_MOV_x: subleq 0, 0, 0
    1535, 0, 0 ,    // (1591) subleq 1535, 0, 0
    0, 0, 0 ,   // (1594) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (1597) subleq 0, 0, 0
    0, 0, 1604 ,    // (1600) subleq 0, 0, 1604
    1 ,     // (1603) _INC_ONE: dd 1
    1603, 0, 0 ,    // (1604) _INC_add: subleq 1603, 0, 0
    0, 161, 0 ,     // (1607) subleq 0, 161, 0
    0, 0, 0 ,   // (1610) subleq 0, 0, 0
    0, 0, 1617 ,    // (1613) subleq 0, 0, 1617
    0 ,     // (1616) _PTR_REF_tmp: dd 0
    1616, 1616, 0 ,     // (1617) subleq 1616, 1616, 0
    1632, 0, 0 ,    // (1620) subleq 1632, 0, 0
    0, 1616, 0 ,    // (1623) subleq 0, 1616, 0
    0, 0, 0 ,   // (1626) subleq 0, 0, 0
    0, 0, 1633 ,    // (1629) subleq 0, 0, 1633
    1697 ,      // (1632) _SETVALUE_tmp: dd 1697
    1669, 1669, 0 ,     // (1633) subleq 1669, 1669, 0
    161, 0, 0 ,     // (1636) subleq 161, 0, 0
    0, 1669, 0 ,    // (1639) subleq 0, 1669, 0
    0, 0, 0 ,   // (1642) subleq 0, 0, 0
    1670, 1670, 0 ,     // (1645) subleq 1670, 1670, 0
    161, 0, 0 ,     // (1648) subleq 161, 0, 0
    0, 1670, 0 ,    // (1651) subleq 0, 1670, 0
    0, 0, 0 ,   // (1654) subleq 0, 0, 0
    1676, 1676, 0 ,     // (1657) subleq 1676, 1676, 0
    161, 0, 0 ,     // (1660) subleq 161, 0, 0
    0, 1676, 0 ,    // (1663) subleq 0, 1676, 0
    0, 0, 0 ,   // (1666) subleq 0, 0, 0
    0, 0, 0 ,   // (1669) _DEREF_DST_MOV_x: subleq 0, 0, 0
    1616, 0, 0 ,    // (1672) subleq 1616, 0, 0
    0, 0, 0 ,   // (1675) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (1678) subleq 0, 0, 0
    0, 0, 1685 ,    // (1681) subleq 0, 0, 1685
    1 ,     // (1684) _INC_ONE: dd 1
    1684, 0, 0 ,    // (1685) _INC_add: subleq 1684, 0, 0
```

```
    0, 161, 0 ,      // (1688) subleq 0, 161, 0
    0, 0, 0 ,   // (1691) subleq 0, 0, 0
    0, 0, 2474 ,     // (1694) subleq 0, 0, 2474
    0, 0, 1701 ,     // (1697) subleq 0, 0, 1701
    1 ,      // (1700) _DEC_ONE: dd 1
    1700, 161, 0 ,       // (1701) _DEC_sub: subleq 1700, 161, 0
    1719, 1719, 0 ,      // (1704) subleq 1719, 1719, 0
    161, 0, 0 ,      // (1707) subleq 161, 0, 0
    0, 1719, 0 ,     // (1710) subleq 0, 1719, 0
    0, 0, 0 ,   // (1713) subleq 0, 0, 0
    76, 76, 0 ,      // (1716) subleq 76, 76, 0
    0, 0, 0 ,   // (1719) _DEREF_target: subleq 0, 0, 0
    0, 76, 0 ,       // (1722) subleq 0, 76, 0
    0, 0, 0 ,   // (1725) subleq 0, 0, 0
    76, 0, 1734 ,    // (1728) subleq 76, 0, 1734
    0, 0, 1740 ,     // (1731) subleq 0, 0, 1740
    0, 0, 0 ,   // (1734) _BEZ_true: subleq 0, 0, 0
    0, 76, 1743 ,    // (1737) subleq 0, 76, 1743
    0, 0, 1955 ,     // (1740) subleq 0, 0, 1955
    0, 0, 1747 ,     // (1743) subleq 0, 0, 1747
    0 ,      // (1746) failstring: dd 0
    0, 0, 1751 ,     // (1747) subleq 0, 0, 1751
    0 ,      // (1750) tmp: dd 0
    1750, 1750, 0 ,      // (1751) subleq 1750, 1750, 0
    1766, 0, 0 ,     // (1754) subleq 1766, 0, 0
    0, 1750, 0 ,     // (1757) subleq 0, 1750, 0
    0, 0, 0 ,   // (1760) subleq 0, 0, 0
    0, 0, 1767 ,     // (1763) subleq 0, 0, 1767
    274 ,    // (1766) _SETVALUE_tmp: dd 274
    273, 0, 0 ,      // (1767) subleq 273, 0, 0
    0, 1750, 0 ,     // (1770) subleq 0, 1750, 0
    0, 0, 0 ,   // (1773) subleq 0, 0, 0
    1791, 1791, 0 ,      // (1776) subleq 1791, 1791, 0
    1750, 0, 0 ,     // (1779) subleq 1750, 0, 0
    0, 1791, 0 ,     // (1782) subleq 0, 1791, 0
    0, 0, 0 ,   // (1785) subleq 0, 0, 0
    1746, 1746, 0 ,      // (1788) subleq 1746, 1746, 0
    0, 0, 0 ,   // (1791) _DEREF_target: subleq 0, 0, 0
    0, 1746, 0 ,     // (1794) subleq 0, 1746, 0
    0, 0, 0 ,   // (1797) subleq 0, 0, 0
    1836, 1836, 0 ,      // (1800) subleq 1836, 1836, 0
    161, 0, 0 ,      // (1803) subleq 161, 0, 0
    0, 1836, 0 ,     // (1806) subleq 0, 1836, 0
    0, 0, 0 ,   // (1809) subleq 0, 0, 0
    1837, 1837, 0 ,      // (1812) subleq 1837, 1837, 0
    161, 0, 0 ,      // (1815) subleq 161, 0, 0
    0, 1837, 0 ,     // (1818) subleq 0, 1837, 0
    0, 0, 0 ,   // (1821) subleq 0, 0, 0
    1843, 1843, 0 ,      // (1824) subleq 1843, 1843, 0
```

```
    161, 0, 0 ,     // (1827) subleq 161, 0, 0
    0, 1843, 0 ,    // (1830) subleq 0, 1843, 0
    0, 0, 0 ,   // (1833) subleq 0, 0, 0
    0, 0, 0 ,   // (1836) _DEREF_DST_MOV_x: subleq 0, 0, 0
    1746, 0, 0 ,    // (1839) subleq 1746, 0, 0
    0, 0, 0 ,   // (1842) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (1845) subleq 0, 0, 0
    0, 0, 1852 ,    // (1848) subleq 0, 0, 1852
    1 ,     // (1851) _INC_ONE: dd 1
    1851, 0, 0 ,    // (1852) _INC_add: subleq 1851, 0, 0
    0, 161, 0 ,     // (1855) subleq 0, 161, 0
    0, 0, 0 ,   // (1858) subleq 0, 0, 0
    0, 0, 1865 ,    // (1861) subleq 0, 0, 1865
    0 ,     // (1864) _PTR_REF_tmp: dd 0
    1864, 1864, 0 ,     // (1865) subleq 1864, 1864, 0
    1880, 0, 0 ,    // (1868) subleq 1880, 0, 0
    0, 1864, 0 ,    // (1871) subleq 0, 1864, 0
    0, 0, 0 ,   // (1874) subleq 0, 0, 0
    0, 0, 1881 ,    // (1877) subleq 0, 0, 1881
    1945 ,      // (1880) _SETVALUE_tmp: dd 1945
    1917, 1917, 0 ,     // (1881) subleq 1917, 1917, 0
    161, 0, 0 ,     // (1884) subleq 161, 0, 0
    0, 1917, 0 ,    // (1887) subleq 0, 1917, 0
    0, 0, 0 ,   // (1890) subleq 0, 0, 0
    1918, 1918, 0 ,     // (1893) subleq 1918, 1918, 0
    161, 0, 0 ,     // (1896) subleq 161, 0, 0
    0, 1918, 0 ,    // (1899) subleq 0, 1918, 0
    0, 0, 0 ,   // (1902) subleq 0, 0, 0
    1924, 1924, 0 ,     // (1905) subleq 1924, 1924, 0
    161, 0, 0 ,     // (1908) subleq 161, 0, 0
    0, 1924, 0 ,    // (1911) subleq 0, 1924, 0
    0, 0, 0 ,   // (1914) subleq 0, 0, 0
    0, 0, 0 ,   // (1917) _DEREF_DST_MOV_x: subleq 0, 0, 0
    1864, 0, 0 ,    // (1920) subleq 1864, 0, 0
    0, 0, 0 ,   // (1923) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (1926) subleq 0, 0, 0
    0, 0, 1933 ,    // (1929) subleq 0, 0, 1933
    1 ,     // (1932) _INC_ONE: dd 1
    1932, 0, 0 ,    // (1933) _INC_add: subleq 1932, 0, 0
    0, 161, 0 ,     // (1936) subleq 0, 161, 0
    0, 0, 0 ,   // (1939) subleq 0, 0, 0
    0, 0, 3758 ,    // (1942) subleq 0, 0, 3758
    1951, 161, 0 ,      // (1945) subleq 1951, 161, 0
    0, 0, 1952 ,    // (1948) subleq 0, 0, 1952
    1 ,     // (1951) _SUB_LITERAL_val: dd 1
    0, 0, 2471 ,    // (1952) subleq 0, 0, 2471
    0, 0, 1959 ,    // (1955) correct_key: subleq 0, 0, 1959
    0 ,     // (1958) _PTR_REF_tmp: dd 0
    1958, 1958, 0 ,     // (1959) subleq 1958, 1958, 0
```

```
    1974, 0, 0 ,    // (1962) subleq 1974, 0, 0
    0, 1958, 0 ,    // (1965) subleq 0, 1958, 0
    0, 0, 0 ,   // (1968) subleq 0, 0, 0
    0, 0, 1975 ,    // (1971) subleq 0, 0, 1975
    177 ,   // (1974) _SETVALUE_tmp: dd 177
    2011, 2011, 0 ,     // (1975) subleq 2011, 2011, 0
    161, 0, 0 ,     // (1978) subleq 161, 0, 0
    0, 2011, 0 ,    // (1981) subleq 0, 2011, 0
    0, 0, 0 ,   // (1984) subleq 0, 0, 0
    2012, 2012, 0 ,     // (1987) subleq 2012, 2012, 0
    161, 0, 0 ,     // (1990) subleq 161, 0, 0
    0, 2012, 0 ,    // (1993) subleq 0, 2012, 0
    0, 0, 0 ,   // (1996) subleq 0, 0, 0
    2018, 2018, 0 ,     // (1999) subleq 2018, 2018, 0
    161, 0, 0 ,     // (2002) subleq 161, 0, 0
    0, 2018, 0 ,    // (2005) subleq 0, 2018, 0
    0, 0, 0 ,   // (2008) subleq 0, 0, 0
    0, 0, 0 ,   // (2011) _DEREF_DST_MOV_x: subleq 0, 0, 0
    1958, 0, 0 ,    // (2014) subleq 1958, 0, 0
    0, 0, 0 ,   // (2017) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (2020) subleq 0, 0, 0
    0, 0, 2027 ,    // (2023) subleq 0, 0, 2027
    1 ,     // (2026) _INC_ONE: dd 1
    2026, 0, 0 ,    // (2027) _INC_add: subleq 2026, 0, 0
    0, 161, 0 ,     // (2030) subleq 0, 161, 0
    0, 0, 0 ,   // (2033) subleq 0, 0, 0
    0, 0, 2040 ,    // (2036) subleq 0, 0, 2040
    0 ,     // (2039) _PTR_REF_tmp: dd 0
    2039, 2039, 0 ,     // (2040) subleq 2039, 2039, 0
    2055, 0, 0 ,    // (2043) subleq 2055, 0, 0
    0, 2039, 0 ,    // (2046) subleq 0, 2039, 0
    0, 0, 0 ,   // (2049) subleq 0, 0, 0
    0, 0, 2056 ,    // (2052) subleq 0, 0, 2056
    2120 ,      // (2055) _SETVALUE_tmp: dd 2120
    2092, 2092, 0 ,     // (2056) subleq 2092, 2092, 0
    161, 0, 0 ,     // (2059) subleq 161, 0, 0
    0, 2092, 0 ,    // (2062) subleq 0, 2092, 0
    0, 0, 0 ,   // (2065) subleq 0, 0, 0
    2093, 2093, 0 ,     // (2068) subleq 2093, 2093, 0
    161, 0, 0 ,     // (2071) subleq 161, 0, 0
    0, 2093, 0 ,    // (2074) subleq 0, 2093, 0
    0, 0, 0 ,   // (2077) subleq 0, 0, 0
    2099, 2099, 0 ,     // (2080) subleq 2099, 2099, 0
    161, 0, 0 ,     // (2083) subleq 161, 0, 0
    0, 2099, 0 ,    // (2086) subleq 0, 2099, 0
    0, 0, 0 ,   // (2089) subleq 0, 0, 0
    0, 0, 0 ,   // (2092) _DEREF_DST_MOV_x: subleq 0, 0, 0
    2039, 0, 0 ,    // (2095) subleq 2039, 0, 0
    0, 0, 0 ,   // (2098) _DEREF_DST_MOV_y: subleq 0, 0, 0
```

```
    0, 0, 0 ,   // (2101) subleq 0, 0, 0
    0, 0, 2108 ,    // (2104) subleq 0, 0, 2108
    1 ,     // (2107) _INC_ONE: dd 1
    2107, 0, 0 ,    // (2108) _INC_add: subleq 2107, 0, 0
    0, 161, 0 ,     // (2111) subleq 0, 161, 0
    0, 0, 0 ,   // (2114) subleq 0, 0, 0
    0, 0, 3758 ,    // (2117) subleq 0, 0, 3758
    2126, 161, 0 ,      // (2120) subleq 2126, 161, 0
    0, 0, 2127 ,    // (2123) subleq 0, 0, 2127
    1 ,     // (2126) _SUB_LITERAL_val: dd 1
    0, 0, 2131 ,    // (2127) subleq 0, 0, 2131
    0 ,     // (2130) _PTR_REF_tmp: dd 0
    2130, 2130, 0 ,     // (2131) subleq 2130, 2130, 0
    2146, 0, 0 ,    // (2134) subleq 2146, 0, 0
    0, 2130, 0 ,    // (2137) subleq 0, 2130, 0
    0, 0, 0 ,   // (2140) subleq 0, 0, 0
    0, 0, 2147 ,    // (2143) subleq 0, 0, 2147
    77 ,    // (2146) _SETVALUE_tmp: dd 77
    2183, 2183, 0 ,     // (2147) subleq 2183, 2183, 0
    161, 0, 0 ,     // (2150) subleq 161, 0, 0
    0, 2183, 0 ,    // (2153) subleq 0, 2183, 0
    0, 0, 0 ,   // (2156) subleq 0, 0, 0
    2184, 2184, 0 ,     // (2159) subleq 2184, 2184, 0
    161, 0, 0 ,     // (2162) subleq 161, 0, 0
    0, 2184, 0 ,    // (2165) subleq 0, 2184, 0
    0, 0, 0 ,   // (2168) subleq 0, 0, 0
    2190, 2190, 0 ,     // (2171) subleq 2190, 2190, 0
    161, 0, 0 ,     // (2174) subleq 161, 0, 0
    0, 2190, 0 ,    // (2177) subleq 0, 2190, 0
    0, 0, 0 ,   // (2180) subleq 0, 0, 0
    0, 0, 0 ,   // (2183) _DEREF_DST_MOV_x: subleq 0, 0, 0
    2130, 0, 0 ,    // (2186) subleq 2130, 0, 0
    0, 0, 0 ,   // (2189) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (2192) subleq 0, 0, 0
    0, 0, 2199 ,    // (2195) subleq 0, 0, 2199
    1 ,     // (2198) _INC_ONE: dd 1
    2198, 0, 0 ,    // (2199) _INC_add: subleq 2198, 0, 0
    0, 161, 0 ,     // (2202) subleq 0, 161, 0
    0, 0, 0 ,   // (2205) subleq 0, 0, 0
    0, 0, 2212 ,    // (2208) subleq 0, 0, 2212
    0 ,     // (2211) _PTR_REF_tmp: dd 0
    2211, 2211, 0 ,     // (2212) subleq 2211, 2211, 0
    2227, 0, 0 ,    // (2215) subleq 2227, 0, 0
    0, 2211, 0 ,    // (2218) subleq 0, 2211, 0
    0, 0, 0 ,   // (2221) subleq 0, 0, 0
    0, 0, 2228 ,    // (2224) subleq 0, 0, 2228
    2292 ,      // (2227) _SETVALUE_tmp: dd 2292
    2264, 2264, 0 ,     // (2228) subleq 2264, 2264, 0
    161, 0, 0 ,     // (2231) subleq 161, 0, 0
```

```
    0, 2264, 0 ,    // (2234) subleq 0, 2264, 0
    0, 0, 0 ,   // (2237) subleq 0, 0, 0
    2265, 2265, 0 ,     // (2240) subleq 2265, 2265, 0
    161, 0, 0 ,     // (2243) subleq 161, 0, 0
    0, 2265, 0 ,    // (2246) subleq 0, 2265, 0
    0, 0, 0 ,   // (2249) subleq 0, 0, 0
    2271, 2271, 0 ,     // (2252) subleq 2271, 2271, 0
    161, 0, 0 ,     // (2255) subleq 161, 0, 0
    0, 2271, 0 ,    // (2258) subleq 0, 2271, 0
    0, 0, 0 ,   // (2261) subleq 0, 0, 0
    0, 0, 0 ,   // (2264) _DEREF_DST_MOV_x: subleq 0, 0, 0
    2211, 0, 0 ,    // (2267) subleq 2211, 0, 0
    0, 0, 0 ,   // (2270) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (2273) subleq 0, 0, 0
    0, 0, 2280 ,    // (2276) subleq 0, 0, 2280
    1 ,     // (2279) _INC_ONE: dd 1
    2279, 0, 0 ,    // (2280) _INC_add: subleq 2279, 0, 0
    0, 161, 0 ,     // (2283) subleq 0, 161, 0
    0, 0, 0 ,   // (2286) subleq 0, 0, 0
    0, 0, 3758 ,    // (2289) subleq 0, 0, 3758
    2298, 161, 0 ,      // (2292) subleq 2298, 161, 0
    0, 0, 2299 ,    // (2295) subleq 0, 0, 2299
    1 ,     // (2298) _SUB_LITERAL_val: dd 1
    0, 0, 2303 ,    // (2299) subleq 0, 0, 2303
    0 ,     // (2302) _PTR_REF_tmp: dd 0
    2302, 2302, 0 ,     // (2303) subleq 2302, 2302, 0
    2318, 0, 0 ,    // (2306) subleq 2318, 0, 0
    0, 2302, 0 ,    // (2309) subleq 0, 2302, 0
    0, 0, 0 ,   // (2312) subleq 0, 0, 0
    0, 0, 2319 ,    // (2315) subleq 0, 0, 2319
    257 ,   // (2318) _SETVALUE_tmp: dd 257
    2355, 2355, 0 ,     // (2319) subleq 2355, 2355, 0
    161, 0, 0 ,     // (2322) subleq 161, 0, 0
    0, 2355, 0 ,    // (2325) subleq 0, 2355, 0
    0, 0, 0 ,   // (2328) subleq 0, 0, 0
    2356, 2356, 0 ,     // (2331) subleq 2356, 2356, 0
    161, 0, 0 ,     // (2334) subleq 161, 0, 0
    0, 2356, 0 ,    // (2337) subleq 0, 2356, 0
    0, 0, 0 ,   // (2340) subleq 0, 0, 0
    2362, 2362, 0 ,     // (2343) subleq 2362, 2362, 0
    161, 0, 0 ,     // (2346) subleq 161, 0, 0
    0, 2362, 0 ,    // (2349) subleq 0, 2362, 0
    0, 0, 0 ,   // (2352) subleq 0, 0, 0
    0, 0, 0 ,   // (2355) _DEREF_DST_MOV_x: subleq 0, 0, 0
    2302, 0, 0 ,    // (2358) subleq 2302, 0, 0
    0, 0, 0 ,   // (2361) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (2364) subleq 0, 0, 0
    0, 0, 2371 ,    // (2367) subleq 0, 0, 2371
    1 ,     // (2370) _INC_ONE: dd 1
```

```
    2370, 0, 0 ,    // (2371) _INC_add: subleq 2370, 0, 0
    0, 161, 0 ,     // (2374) subleq 0, 161, 0
    0, 0, 0 ,   // (2377) subleq 0, 0, 0
    0, 0, 2384 ,    // (2380) subleq 0, 0, 2384
    0 ,     // (2383) _PTR_REF_tmp: dd 0
    2383, 2383, 0 ,     // (2384) subleq 2383, 2383, 0
    2399, 0, 0 ,    // (2387) subleq 2399, 0, 0
    0, 2383, 0 ,    // (2390) subleq 0, 2383, 0
    0, 0, 0 ,   // (2393) subleq 0, 0, 0
    0, 0, 2400 ,    // (2396) subleq 0, 0, 2400
    2464 ,      // (2399) _SETVALUE_tmp: dd 2464
    2436, 2436, 0 ,     // (2400) subleq 2436, 2436, 0
    161, 0, 0 ,     // (2403) subleq 161, 0, 0
    0, 2436, 0 ,    // (2406) subleq 0, 2436, 0
    0, 0, 0 ,   // (2409) subleq 0, 0, 0
    2437, 2437, 0 ,     // (2412) subleq 2437, 2437, 0
    161, 0, 0 ,     // (2415) subleq 161, 0, 0
    0, 2437, 0 ,    // (2418) subleq 0, 2437, 0
    0, 0, 0 ,   // (2421) subleq 0, 0, 0
    2443, 2443, 0 ,     // (2424) subleq 2443, 2443, 0
    161, 0, 0 ,     // (2427) subleq 161, 0, 0
    0, 2443, 0 ,    // (2430) subleq 0, 2443, 0
    0, 0, 0 ,   // (2433) subleq 0, 0, 0
    0, 0, 0 ,   // (2436) _DEREF_DST_MOV_x: subleq 0, 0, 0
    2383, 0, 0 ,    // (2439) subleq 2383, 0, 0
    0, 0, 0 ,   // (2442) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (2445) subleq 0, 0, 0
    0, 0, 2452 ,    // (2448) subleq 0, 0, 2452
    1 ,     // (2451) _INC_ONE: dd 1
    2451, 0, 0 ,    // (2452) _INC_add: subleq 2451, 0, 0
    0, 161, 0 ,     // (2455) subleq 0, 161, 0
    0, 0, 0 ,   // (2458) subleq 0, 0, 0
    0, 0, 3758 ,    // (2461) subleq 0, 0, 3758
    2470, 161, 0 ,      // (2464) subleq 2470, 161, 0
    0, 0, 2471 ,    // (2467) subleq 0, 0, 2471
    1 ,     // (2470) _SUB_LITERAL_val: dd 1
    0, 0, -1 ,      // (2471) ending: subleq 0, 0, -1
    0, 0, 2478 ,    // (2474) check_string: subleq 0, 0, 2478
    0 ,     // (2477) _bp: dd 0
    2477, 2477, 0 ,     // (2478) subleq 2477, 2477, 0
    161, 0, 0 ,     // (2481) subleq 161, 0, 0
    0, 2477, 0 ,    // (2484) subleq 0, 2477, 0
    0, 0, 0 ,   // (2487) subleq 0, 0, 0
    0, 0, 2494 ,    // (2490) subleq 0, 0, 2494
    0 ,     // (2493) _string: dd 0
    0, 0, 2498 ,    // (2494) subleq 0, 0, 2498
    -2 ,    // (2497) _DECLARE_FUNCTION_ARGUMENT_bp_offset: dd -2
    0, 0, 2502 ,    // (2498) subleq 0, 0, 2502
    0 ,     // (2501) _DECLARE_FUNCTION_ARGUMENT_tmp: dd 0
```

```
    2501, 2501, 0 ,      // (2502) subleq 2501, 2501, 0
    2497, 0, 0 ,    // (2505) subleq 2497, 0, 0
    0, 2501, 0 ,    // (2508) subleq 0, 2501, 0
    0, 0, 0 ,   // (2511) subleq 0, 0, 0
    2477, 0, 0 ,    // (2514) subleq 2477, 0, 0
    0, 2501, 0 ,    // (2517) subleq 0, 2501, 0
    0, 0, 0 ,   // (2520) subleq 0, 0, 0
    2538, 2538, 0 ,      // (2523) subleq 2538, 2538, 0
    2501, 0, 0 ,    // (2526) subleq 2501, 0, 0
    0, 2538, 0 ,    // (2529) subleq 0, 2538, 0
    0, 0, 0 ,   // (2532) subleq 0, 0, 0
    2493, 2493, 0 ,      // (2535) subleq 2493, 2493, 0
    0, 0, 0 ,   // (2538) _DEREF_target: subleq 0, 0, 0
    0, 2493, 0 ,    // (2541) subleq 0, 2493, 0
    0, 0, 0 ,   // (2544) subleq 0, 0, 0
    0, 0, 2551 ,    // (2547) _FORLOOP_START_LOOPID_check_loop: subleq 0, 0, 2551
    16 ,    // (2550) _FORLOOP_START_numloops: dd 16
    2578, 2578, 0 ,      // (2551) _FORLOOP_START_compare: subleq 2578, 2578, 0
    3735, 0, 0 ,    // (2554) subleq 3735, 0, 0
    0, 2578, 0 ,    // (2557) subleq 0, 2578, 0
    0, 0, 0 ,   // (2560) subleq 0, 0, 0
    2550, 2578, 0 ,      // (2563) subleq 2550, 2578, 0
    2578, 0, 2572 ,      // (2566) subleq 2578, 0, 2572
    0, 0, 2579 ,    // (2569) subleq 0, 0, 2579
    0, 0, 0 ,   // (2572) _BLZ_false: subleq 0, 0, 0
    0, 0, 3630 ,    // (2575) subleq 0, 0, 3630
    0 ,     // (2578) _BGEQ_tmp: dd 0
    2594, 2594, 0 ,      // (2579) subleq 2594, 2594, 0
    2493, 0, 0 ,    // (2582) subleq 2493, 0, 0
    0, 2594, 0 ,    // (2585) subleq 0, 2594, 0
    0, 0, 0 ,   // (2588) subleq 0, 0, 0
    3736, 3736, 0 ,      // (2591) subleq 3736, 3736, 0
    0, 0, 0 ,   // (2594) _DEREF_target: subleq 0, 0, 0
    0, 3736, 0 ,    // (2597) subleq 0, 3736, 0
    0, 0, 0 ,   // (2600) subleq 0, 0, 0
    3736, 0, 2609 ,      // (2603) subleq 3736, 0, 2609
    0, 0, 2615 ,    // (2606) subleq 0, 0, 2615
    0, 0, 0 ,   // (2609) _BEZ_true: subleq 0, 0, 0
    0, 3736, 2618 ,      // (2612) subleq 0, 3736, 2618
    0, 0, 2624 ,    // (2615) subleq 0, 0, 2624
    3757, 3757, 0 ,      // (2618) subleq 3757, 3757, 0
    0, 0, 3630 ,    // (2621) subleq 0, 0, 3630
    0, 0, 2628 ,    // (2624) _check_loop_continue: subleq 0, 0, 2628
    1 ,     // (2627) _INC_ONE: dd 1
    2627, 0, 0 ,    // (2628) _INC_add: subleq 2627, 0, 0
    0, 2493, 0 ,    // (2631) subleq 0, 2493, 0
    0, 0, 0 ,   // (2634) subleq 0, 0, 0
    2652, 2652, 0 ,      // (2637) subleq 2652, 2652, 0
    2493, 0, 0 ,    // (2640) subleq 2493, 0, 0
```

```
    0, 2652, 0 ,    // (2643) subleq 0, 2652, 0
    0, 0, 0 ,   // (2646) subleq 0, 0, 0
    3737, 3737, 0 ,     // (2649) subleq 3737, 3737, 0
    0, 0, 0 ,   // (2652) _DEREF_target: subleq 0, 0, 0
    0, 3737, 0 ,    // (2655) subleq 0, 3737, 0
    0, 0, 0 ,   // (2658) subleq 0, 0, 0
    2679, 2679, 0 ,     // (2661) subleq 2679, 2679, 0
    2680, 2680, 0 ,     // (2664) subleq 2680, 2680, 0
    3736, 0, 2673 ,     // (2667) subleq 3736, 0, 2673
    0, 0, 2740 ,    // (2670) subleq 0, 0, 2740
    0, 0, 0 ,   // (2673) _BLZ_false: subleq 0, 0, 0
    0, 0, 2681 ,    // (2676) subleq 0, 0, 2681
    0 ,     // (2679) _MUL_i: dd 0
    0 ,     // (2680) _MUL_result: dd 0
    2714, 2714, 0 ,     // (2681) _MUL_b_pos: subleq 2714, 2714, 0
    3736, 0, 0 ,    // (2684) subleq 3736, 0, 0
    0, 2714, 0 ,    // (2687) subleq 0, 2714, 0
    0, 0, 0 ,   // (2690) subleq 0, 0, 0
    2679, 2714, 0 ,     // (2693) subleq 2679, 2714, 0
    2714, 0, 2702 ,     // (2696) subleq 2714, 0, 2702
    0, 0, 2708 ,    // (2699) subleq 0, 0, 2708
    0, 0, 0 ,   // (2702) _BEZ_true: subleq 0, 0, 0
    0, 2714, 2711 ,     // (2705) subleq 0, 2714, 2711
    0, 0, 2715 ,    // (2708) subleq 0, 0, 2715
    0, 0, 2787 ,    // (2711) subleq 0, 0, 2787
    0 ,     // (2714) _BEQ_tmp: dd 0
    3738, 0, 0 ,    // (2715) subleq 3738, 0, 0
    0, 2680, 0 ,    // (2718) subleq 0, 2680, 0
    0, 0, 0 ,   // (2721) subleq 0, 0, 0
    0, 0, 2728 ,    // (2724) subleq 0, 0, 2728
    1 ,     // (2727) _INC_ONE: dd 1
    2727, 0, 0 ,    // (2728) _INC_add: subleq 2727, 0, 0
    0, 2679, 0 ,    // (2731) subleq 0, 2679, 0
    0, 0, 0 ,   // (2734) subleq 0, 0, 0
    0, 0, 2681 ,    // (2737) subleq 0, 0, 2681
    2773, 2773, 0 ,     // (2740) _MUL_b_neg: subleq 2773, 2773, 0
    3736, 0, 0 ,    // (2743) subleq 3736, 0, 0
    0, 2773, 0 ,    // (2746) subleq 0, 2773, 0
    0, 0, 0 ,   // (2749) subleq 0, 0, 0
    2679, 2773, 0 ,     // (2752) subleq 2679, 2773, 0
    2773, 0, 2761 ,     // (2755) subleq 2773, 0, 2761
    0, 0, 2767 ,    // (2758) subleq 0, 0, 2767
    0, 0, 0 ,   // (2761) _BEZ_true: subleq 0, 0, 0
    0, 2773, 2770 ,     // (2764) subleq 0, 2773, 2770
    0, 0, 2774 ,    // (2767) subleq 0, 0, 2774
    0, 0, 2787 ,    // (2770) subleq 0, 0, 2787
    0 ,     // (2773) _BEQ_tmp: dd 0
    3738, 2680, 0 ,     // (2774) subleq 3738, 2680, 0
    0, 0, 2781 ,    // (2777) subleq 0, 0, 2781
```

```
    1 ,     // (2780) _DEC_ONE: dd 1
    2780, 2679, 0 ,     // (2781) _DEC_sub: subleq 2780, 2679, 0
    0, 0, 2740 ,    // (2784) subleq 0, 0, 2740
    3736, 3736, 0 ,     // (2787) _MUL_finished: subleq 3736, 3736, 0
    2680, 0, 0 ,    // (2790) subleq 2680, 0, 0
    0, 3736, 0 ,    // (2793) subleq 0, 3736, 0
    0, 0, 0 ,   // (2796) subleq 0, 0, 0
    0, 0, 2803 ,    // (2799) subleq 0, 0, 2803
    0 ,     // (2802) _i: dd 0
    2802, 2802, 0 ,     // (2803) subleq 2802, 2802, 0
    0, 0, 2810 ,    // (2806) _FORLOOP_START_LOOPID__SHL_loop: subleq 0, 0, 2810
    7 ,     // (2809) _FORLOOP_START_numloops: dd 7
    2837, 2837, 0 ,     // (2810) _FORLOOP_START_compare: subleq 2837, 2837, 0
    2802, 0, 0 ,    // (2813) subleq 2802, 0, 0
    0, 2837, 0 ,    // (2816) subleq 0, 2837, 0
    0, 0, 0 ,   // (2819) subleq 0, 0, 0
    2809, 2837, 0 ,     // (2822) subleq 2809, 2837, 0
    2837, 0, 2831 ,     // (2825) subleq 2837, 0, 2831
    0, 0, 2838 ,    // (2828) subleq 0, 0, 2838
    0, 0, 0 ,   // (2831) _BLZ_false: subleq 0, 0, 0
    0, 0, 2869 ,    // (2834) subleq 0, 0, 2869
    0 ,     // (2837) _BGEQ_tmp: dd 0
    3736, 0, 0 ,    // (2838) subleq 3736, 0, 0
    3736, 0, 0 ,    // (2841) subleq 3736, 0, 0
    3736, 3736, 0 ,     // (2844) subleq 3736, 3736, 0
    0, 3736, 0 ,    // (2847) subleq 0, 3736, 0
    0, 0, 0 ,   // (2850) subleq 0, 0, 0
    0, 0, 2857 ,    // (2853) subleq 0, 0, 2857
    1 ,     // (2856) _INC_ONE: dd 1
    2856, 0, 0 ,    // (2857) _INC_add: subleq 2856, 0, 0
    0, 2802, 0 ,    // (2860) subleq 0, 2802, 0
    0, 0, 0 ,   // (2863) subleq 0, 0, 0
    0, 0, 2806 ,    // (2866) subleq 0, 0, 2806
    3054, 3054, 0 ,     // (2869) subleq 3054, 3054, 0
    3739, 0, 0 ,    // (2872) subleq 3739, 0, 0
    0, 3054, 0 ,    // (2875) subleq 0, 3054, 0
    0, 0, 0 ,   // (2878) subleq 0, 0, 0
    3055, 3055, 0 ,     // (2881) subleq 3055, 3055, 0
    3737, 0, 0 ,    // (2884) subleq 3737, 0, 0
    0, 3055, 0 ,    // (2887) subleq 0, 3055, 0
    0, 0, 0 ,   // (2890) subleq 0, 0, 0
    3061, 3061, 0 ,     // (2893) subleq 3061, 3061, 0
    0, 0, 2900 ,    // (2896) _FORLOOP_START_LOOPID__BITWISE_OPERATOR_loop: subleq
0, 0, 2900
    32 ,    // (2899) _FORLOOP_START_numloops: dd 32
    2927, 2927, 0 ,     // (2900) _FORLOOP_START_compare: subleq 2927, 2927, 0
    3061, 0, 0 ,    // (2903) subleq 3061, 0, 0
    0, 2927, 0 ,    // (2906) subleq 0, 2927, 0
    0, 0, 0 ,   // (2909) subleq 0, 0, 0
```

```
    2899, 2927, 0 ,      // (2912) subleq 2899, 2927, 0
    2927, 0, 2921 ,      // (2915) subleq 2927, 0, 2921
    0, 0, 2928 ,    // (2918) subleq 0, 0, 2928
    0, 0, 0 ,   // (2921) _BLZ_false: subleq 0, 0, 0
    0, 0, 3194 ,    // (2924) subleq 0, 0, 3194
    0 ,     // (2927) _BGEQ_tmp: dd 0
    3054, 0, 2934 ,      // (2928) subleq 3054, 0, 2934
    0, 0, 2949 ,    // (2931) subleq 0, 0, 2949
    0, 0, 0 ,   // (2934) _BLZ_false: subleq 0, 0, 0
    3056, 3056, 0 ,      // (2937) _GETMSB_return0: subleq 3056, 3056, 0
    2962, 0, 0 ,    // (2940) subleq 2962, 0, 0
    0, 3056, 0 ,    // (2943) subleq 0, 3056, 0
    0, 0, 2963 ,    // (2946) subleq 0, 0, 2963
    3056, 3056, 0 ,      // (2949) _GETMSB_return1: subleq 3056, 3056, 0
    2961, 0, 0 ,    // (2952) subleq 2961, 0, 0
    0, 3056, 0 ,    // (2955) subleq 0, 3056, 0
    0, 0, 2963 ,    // (2958) subleq 0, 0, 2963
    1 ,     // (2961) _GETMSB_one: dd 1
    0 ,     // (2962) _GETMSB_zero: dd 0
    3055, 0, 2969 ,      // (2963) subleq 3055, 0, 2969
    0, 0, 2984 ,    // (2966) subleq 0, 0, 2984
    0, 0, 0 ,   // (2969) _BLZ_false: subleq 0, 0, 0
    3057, 3057, 0 ,      // (2972) _GETMSB_return0: subleq 3057, 3057, 0
    2997, 0, 0 ,    // (2975) subleq 2997, 0, 0
    0, 3057, 0 ,    // (2978) subleq 0, 3057, 0
    0, 0, 2998 ,    // (2981) subleq 0, 0, 2998
    3057, 3057, 0 ,      // (2984) _GETMSB_return1: subleq 3057, 3057, 0
    2996, 0, 0 ,    // (2987) subleq 2996, 0, 0
    0, 3057, 0 ,    // (2990) subleq 0, 3057, 0
    0, 0, 2998 ,    // (2993) subleq 0, 0, 2998
    1 ,     // (2996) _GETMSB_one: dd 1
    0 ,     // (2997) _GETMSB_zero: dd 0
    3053, 3053, 0 ,      // (2998) subleq 3053, 3053, 0
    3056, 0, 0 ,    // (3001) subleq 3056, 0, 0
    0, 3053, 0 ,    // (3004) subleq 0, 3053, 0
    0, 0, 0 ,   // (3007) subleq 0, 0, 0
    3057, 0, 0 ,    // (3010) subleq 3057, 0, 0
    0, 3053, 0 ,    // (3013) subleq 0, 3053, 0
    0, 0, 0 ,   // (3016) subleq 0, 0, 0
    3053, 0, 3025 ,      // (3019) subleq 3053, 0, 3025
    0, 0, 3031 ,    // (3022) subleq 0, 0, 3031
    0, 0, 0 ,   // (3025) _BEZ_true: subleq 0, 0, 0
    0, 3053, 3062 ,      // (3028) subleq 0, 3053, 3062
    0, 0, 3035 ,    // (3031) subleq 0, 0, 3035
    1 ,     // (3034) _DEC_ONE: dd 1
    3034, 3053, 0 ,      // (3035) _DEC_sub: subleq 3034, 3053, 0
    3053, 0, 3044 ,      // (3038) subleq 3053, 0, 3044
    0, 0, 3050 ,    // (3041) subleq 0, 0, 3050
    0, 0, 0 ,   // (3044) _BEZ_true: subleq 0, 0, 0
```

```
    0, 3053, 3074 ,     // (3047) subleq 0, 3053, 3074
    0, 0, 3086 ,    // (3050) subleq 0, 0, 3086
    0 ,     // (3053) _BITWISE_JMP_tmp: dd 0
    0 ,     // (3054) _BITWISE_OPERATOR_tmp_a: dd 0
    0 ,     // (3055) _BITWISE_OPERATOR_tmp_b: dd 0
    0 ,     // (3056) _BITWISE_OPERATOR_msb_a: dd 0
    0 ,     // (3057) _BITWISE_OPERATOR_msb_b: dd 0
    0 ,     // (3058) _BITWISE_OPERATOR_result: dd 0
    0 ,     // (3059) _BO_ZERO: dd 0
    1 ,     // (3060) _BO_ONE: dd 1
    0 ,     // (3061) _BITWISE_OPERATOR_i: dd 0
    3056, 3056, 0 ,     // (3062) _BITWISE_OPERATOR_none: subleq 3056, 3056, 0
    3059, 0, 0 ,    // (3065) subleq 3059, 0, 0
    0, 3056, 0 ,    // (3068) subleq 0, 3056, 0
    0, 0, 3098 ,    // (3071) subleq 0, 0, 3098
    3056, 3056, 0 ,     // (3074) _BITWISE_OPERATOR_onlyone: subleq 3056, 3056, 0
    3060, 0, 0 ,    // (3077) subleq 3060, 0, 0
    0, 3056, 0 ,    // (3080) subleq 0, 3056, 0
    0, 0, 3098 ,    // (3083) subleq 0, 0, 3098
    3056, 3056, 0 ,     // (3086) _BITWISE_OPERATOR_both: subleq 3056, 3056, 0
    3059, 0, 0 ,    // (3089) subleq 3059, 0, 0
    0, 3056, 0 ,    // (3092) subleq 0, 3056, 0
    0, 0, 0 ,   // (3095) subleq 0, 0, 0
    3058, 0, 0 ,    // (3098) _BITWISE_OPERATOR_insert_new_bit: subleq 3058, 0, 0
    3058, 0, 0 ,    // (3101) subleq 3058, 0, 0
    3058, 3058, 0 ,     // (3104) subleq 3058, 3058, 0
    0, 3058, 0 ,    // (3107) subleq 0, 3058, 0
    0, 0, 0 ,   // (3110) subleq 0, 0, 0
    3056, 0, 0 ,    // (3113) subleq 3056, 0, 0
    0, 3058, 0 ,    // (3116) subleq 0, 3058, 0
    0, 0, 0 ,   // (3119) subleq 0, 0, 0
    3054, 0, 0 ,    // (3122) subleq 3054, 0, 0
    3054, 0, 0 ,    // (3125) subleq 3054, 0, 0
    3054, 3054, 0 ,     // (3128) subleq 3054, 3054, 0
    0, 3054, 0 ,    // (3131) subleq 0, 3054, 0
    0, 0, 0 ,   // (3134) subleq 0, 0, 0
    0, 0, 3141 ,    // (3137) subleq 0, 0, 3141
    1 ,     // (3140) _INC_ONE: dd 1
    3140, 0, 0 ,    // (3141) _INC_add: subleq 3140, 0, 0
    0, 3054, 0 ,    // (3144) subleq 0, 3054, 0
    0, 0, 0 ,   // (3147) subleq 0, 0, 0
    3055, 0, 0 ,    // (3150) subleq 3055, 0, 0
    3055, 0, 0 ,    // (3153) subleq 3055, 0, 0
    3055, 3055, 0 ,     // (3156) subleq 3055, 3055, 0
    0, 3055, 0 ,    // (3159) subleq 0, 3055, 0
    0, 0, 0 ,   // (3162) subleq 0, 0, 0
    0, 0, 3169 ,    // (3165) subleq 0, 0, 3169
    1 ,     // (3168) _INC_ONE: dd 1
    3168, 0, 0 ,    // (3169) _INC_add: subleq 3168, 0, 0
```

```
    0, 3055, 0 ,    // (3172) subleq 0, 3055, 0
    0, 0, 0 ,   // (3175) subleq 0, 0, 0
    0, 0, 3182 ,    // (3178) subleq 0, 0, 3182
    1 ,     // (3181) _INC_ONE: dd 1
    3181, 0, 0 ,    // (3182) _INC_add: subleq 3181, 0, 0
    0, 3061, 0 ,    // (3185) subleq 0, 3061, 0
    0, 0, 0 ,   // (3188) subleq 0, 0, 0
    0, 0, 2896 ,    // (3191) subleq 0, 0, 2896
    3737, 3737, 0 ,     // (3194) _FORLOOP_END_LOOPID__BITWISE_OPERATOR_loop:
subleq 3737, 3737, 0
    3058, 0, 0 ,    // (3197) subleq 3058, 0, 0
    0, 3737, 0 ,    // (3200) subleq 0, 3737, 0
    0, 0, 0 ,   // (3203) subleq 0, 0, 0
    3391, 3391, 0 ,     // (3206) subleq 3391, 3391, 0
    3737, 0, 0 ,    // (3209) subleq 3737, 0, 0
    0, 3391, 0 ,    // (3212) subleq 0, 3391, 0
    0, 0, 0 ,   // (3215) subleq 0, 0, 0
    3392, 3392, 0 ,     // (3218) subleq 3392, 3392, 0
    3736, 0, 0 ,    // (3221) subleq 3736, 0, 0
    0, 3392, 0 ,    // (3224) subleq 0, 3392, 0
    0, 0, 0 ,   // (3227) subleq 0, 0, 0
    3398, 3398, 0 ,     // (3230) subleq 3398, 3398, 0
    0, 0, 3237 ,    // (3233) _FORLOOP_START_LOOPID__BITWISE_OPERATOR_loop: subleq
0, 0, 3237
    32 ,    // (3236) _FORLOOP_START_numloops: dd 32
    3264, 3264, 0 ,     // (3237) _FORLOOP_START_compare: subleq 3264, 3264, 0
    3398, 0, 0 ,    // (3240) subleq 3398, 0, 0
    0, 3264, 0 ,    // (3243) subleq 0, 3264, 0
    0, 0, 0 ,   // (3246) subleq 0, 0, 0
    3236, 3264, 0 ,     // (3249) subleq 3236, 3264, 0
    3264, 0, 3258 ,     // (3252) subleq 3264, 0, 3258
    0, 0, 3265 ,    // (3255) subleq 0, 0, 3265
    0, 0, 0 ,   // (3258) _BLZ_false: subleq 0, 0, 0
    0, 0, 3531 ,    // (3261) subleq 0, 0, 3531
    0 ,     // (3264) _BGEQ_tmp: dd 0
    3391, 0, 3271 ,     // (3265) subleq 3391, 0, 3271
    0, 0, 3286 ,    // (3268) subleq 0, 0, 3286
    0, 0, 0 ,   // (3271) _BLZ_false: subleq 0, 0, 0
    3393, 3393, 0 ,     // (3274) _GETMSB_return0: subleq 3393, 3393, 0
    3299, 0, 0 ,    // (3277) subleq 3299, 0, 0
    0, 3393, 0 ,    // (3280) subleq 0, 3393, 0
    0, 0, 3300 ,    // (3283) subleq 0, 0, 3300
    3393, 3393, 0 ,     // (3286) _GETMSB_return1: subleq 3393, 3393, 0
    3298, 0, 0 ,    // (3289) subleq 3298, 0, 0
    0, 3393, 0 ,    // (3292) subleq 0, 3393, 0
    0, 0, 3300 ,    // (3295) subleq 0, 0, 3300
    1 ,     // (3298) _GETMSB_one: dd 1
    0 ,     // (3299) _GETMSB_zero: dd 0
    3392, 0, 3306 ,     // (3300) subleq 3392, 0, 3306
```

```
    0, 0, 3321 ,    // (3303) subleq 0, 0, 3321
    0, 0, 0 ,   // (3306) _BLZ_false: subleq 0, 0, 0
    3394, 3394, 0 ,     // (3309) _GETMSB_return0: subleq 3394, 3394, 0
    3334, 0, 0 ,    // (3312) subleq 3334, 0, 0
    0, 3394, 0 ,    // (3315) subleq 0, 3394, 0
    0, 0, 3335 ,    // (3318) subleq 0, 0, 3335
    3394, 3394, 0 ,     // (3321) _GETMSB_return1: subleq 3394, 3394, 0
    3333, 0, 0 ,    // (3324) subleq 3333, 0, 0
    0, 3394, 0 ,    // (3327) subleq 0, 3394, 0
    0, 0, 3335 ,    // (3330) subleq 0, 0, 3335
    1 ,     // (3333) _GETMSB_one: dd 1
    0 ,     // (3334) _GETMSB_zero: dd 0
    3390, 3390, 0 ,     // (3335) subleq 3390, 3390, 0
    3393, 0, 0 ,    // (3338) subleq 3393, 0, 0
    0, 3390, 0 ,    // (3341) subleq 0, 3390, 0
    0, 0, 0 ,   // (3344) subleq 0, 0, 0
    3394, 0, 0 ,    // (3347) subleq 3394, 0, 0
    0, 3390, 0 ,    // (3350) subleq 0, 3390, 0
    0, 0, 0 ,   // (3353) subleq 0, 0, 0
    3390, 0, 3362 ,     // (3356) subleq 3390, 0, 3362
    0, 0, 3368 ,    // (3359) subleq 0, 0, 3368
    0, 0, 0 ,   // (3362) _BEZ_true: subleq 0, 0, 0
    0, 3390, 3399 ,     // (3365) subleq 0, 3390, 3399
    0, 0, 3372 ,    // (3368) subleq 0, 0, 3372
    1 ,     // (3371) _DEC_ONE: dd 1
    3371, 3390, 0 ,     // (3372) _DEC_sub: subleq 3371, 3390, 0
    3390, 0, 3381 ,     // (3375) subleq 3390, 0, 3381
    0, 0, 3387 ,    // (3378) subleq 0, 0, 3387
    0, 0, 0 ,   // (3381) _BEZ_true: subleq 0, 0, 0
    0, 3390, 3411 ,     // (3384) subleq 0, 3390, 3411
    0, 0, 3423 ,    // (3387) subleq 0, 0, 3423
    0 ,     // (3390) _BITWISE_JMP_tmp: dd 0
    0 ,     // (3391) _BITWISE_OPERATOR_tmp_a: dd 0
    0 ,     // (3392) _BITWISE_OPERATOR_tmp_b: dd 0
    0 ,     // (3393) _BITWISE_OPERATOR_msb_a: dd 0
    0 ,     // (3394) _BITWISE_OPERATOR_msb_b: dd 0
    0 ,     // (3395) _BITWISE_OPERATOR_result: dd 0
    0 ,     // (3396) _BO_ZERO: dd 0
    1 ,     // (3397) _BO_ONE: dd 1
    0 ,     // (3398) _BITWISE_OPERATOR_i: dd 0
    3393, 3393, 0 ,     // (3399) _BITWISE_OPERATOR_none: subleq 3393, 3393, 0
    3396, 0, 0 ,    // (3402) subleq 3396, 0, 0
    0, 3393, 0 ,    // (3405) subleq 0, 3393, 0
    0, 0, 3435 ,    // (3408) subleq 0, 0, 3435
    3393, 3393, 0 ,     // (3411) _BITWISE_OPERATOR_onlyone: subleq 3393, 3393, 0
    3397, 0, 0 ,    // (3414) subleq 3397, 0, 0
    0, 3393, 0 ,    // (3417) subleq 0, 3393, 0
    0, 0, 3435 ,    // (3420) subleq 0, 0, 3435
    3393, 3393, 0 ,     // (3423) _BITWISE_OPERATOR_both: subleq 3393, 3393, 0
```

```
    3397, 0, 0 ,    // (3426) subleq 3397, 0, 0
    0, 3393, 0 ,    // (3429) subleq 0, 3393, 0
    0, 0, 0 ,   // (3432) subleq 0, 0, 0
    3395, 0, 0 ,    // (3435) _BITWISE_OPERATOR_insert_new_bit: subleq 3395, 0, 0
    3395, 0, 0 ,    // (3438) subleq 3395, 0, 0
    3395, 3395, 0 ,     // (3441) subleq 3395, 3395, 0
    0, 3395, 0 ,    // (3444) subleq 0, 3395, 0
    0, 0, 0 ,   // (3447) subleq 0, 0, 0
    3393, 0, 0 ,    // (3450) subleq 3393, 0, 0
    0, 3395, 0 ,    // (3453) subleq 0, 3395, 0
    0, 0, 0 ,   // (3456) subleq 0, 0, 0
    3391, 0, 0 ,    // (3459) subleq 3391, 0, 0
    3391, 0, 0 ,    // (3462) subleq 3391, 0, 0
    3391, 3391, 0 ,     // (3465) subleq 3391, 3391, 0
    0, 3391, 0 ,    // (3468) subleq 0, 3391, 0
    0, 0, 0 ,   // (3471) subleq 0, 0, 0
    0, 0, 3478 ,    // (3474) subleq 0, 0, 3478
    1 ,     // (3477) _INC_ONE: dd 1
    3477, 0, 0 ,    // (3478) _INC_add: subleq 3477, 0, 0
    0, 3391, 0 ,    // (3481) subleq 0, 3391, 0
    0, 0, 0 ,   // (3484) subleq 0, 0, 0
    3392, 0, 0 ,    // (3487) subleq 3392, 0, 0
    3392, 0, 0 ,    // (3490) subleq 3392, 0, 0
    3392, 3392, 0 ,     // (3493) subleq 3392, 3392, 0
    0, 3392, 0 ,    // (3496) subleq 0, 3392, 0
    0, 0, 0 ,   // (3499) subleq 0, 0, 0
    0, 0, 3506 ,    // (3502) subleq 0, 0, 3506
    1 ,     // (3505) _INC_ONE: dd 1
    3505, 0, 0 ,    // (3506) _INC_add: subleq 3505, 0, 0
    0, 3392, 0 ,    // (3509) subleq 0, 3392, 0
    0, 0, 0 ,   // (3512) subleq 0, 0, 0
    0, 0, 3519 ,    // (3515) subleq 0, 0, 3519
    1 ,     // (3518) _INC_ONE: dd 1
    3518, 0, 0 ,    // (3519) _INC_add: subleq 3518, 0, 0
    0, 3398, 0 ,    // (3522) subleq 0, 3398, 0
    0, 0, 0 ,   // (3525) subleq 0, 0, 0
    0, 0, 3233 ,    // (3528) subleq 0, 0, 3233
    3736, 3736, 0 ,     // (3531) _FORLOOP_END_LOOPID__BITWISE_OPERATOR_loop:
subleq 3736, 3736, 0
    3395, 0, 0 ,    // (3534) subleq 3395, 0, 0
    0, 3736, 0 ,    // (3537) subleq 0, 3736, 0
    0, 0, 0 ,   // (3540) subleq 0, 0, 0
    3558, 3558, 0 ,     // (3543) subleq 3558, 3558, 0
    3756, 0, 0 ,    // (3546) subleq 3756, 0, 0
    0, 3558, 0 ,    // (3549) subleq 0, 3558, 0
    0, 0, 0 ,   // (3552) subleq 0, 0, 0
    3737, 3737, 0 ,     // (3555) subleq 3737, 3737, 0
    0, 0, 0 ,   // (3558) _DEREF_target: subleq 0, 0, 0
    0, 3737, 0 ,    // (3561) subleq 0, 3737, 0
```

```
    0, 0, 0 ,   // (3564) subleq 0, 0, 0
    3736, 3737, 0 ,     // (3567) subleq 3736, 3737, 0
    3737, 0, 3576 ,     // (3570) subleq 3737, 0, 3576
    0, 0, 3582 ,    // (3573) subleq 0, 0, 3582
    0, 0, 0 ,   // (3576) _BEZ_true: subleq 0, 0, 0
    0, 3737, 3588 ,     // (3579) subleq 0, 3737, 3588
    3757, 3757, 0 ,     // (3582) subleq 3757, 3757, 0
    0, 0, 3588 ,    // (3585) subleq 0, 0, 3588
    0, 0, 3592 ,    // (3588) _check_loop_tail: subleq 0, 0, 3592
    1 ,     // (3591) _INC_ONE: dd 1
    3591, 0, 0 ,    // (3592) _INC_add: subleq 3591, 0, 0
    0, 2493, 0 ,    // (3595) subleq 0, 2493, 0
    0, 0, 0 ,   // (3598) subleq 0, 0, 0
    0, 0, 3605 ,    // (3601) subleq 0, 0, 3605
    1 ,     // (3604) _INC_ONE: dd 1
    3604, 0, 0 ,    // (3605) _INC_add: subleq 3604, 0, 0
    0, 3756, 0 ,    // (3608) subleq 0, 3756, 0
    0, 0, 0 ,   // (3611) subleq 0, 0, 0
    0, 0, 3618 ,    // (3614) subleq 0, 0, 3618
    1 ,     // (3617) _INC_ONE: dd 1
    3617, 0, 0 ,    // (3618) _INC_add: subleq 3617, 0, 0
    0, 3735, 0 ,    // (3621) subleq 0, 3735, 0
    0, 0, 0 ,   // (3624) subleq 0, 0, 0
    0, 0, 2547 ,    // (3627) subleq 0, 0, 2547
    0, 0, 3634 ,    // (3630) _FORLOOP_END_LOOPID_check_loop: subleq 0, 0, 3634
    0 ,     // (3633) _SET_FUNCTION_RETURN_VALUE_tmp: dd 0
    3633, 3633, 0 ,     // (3634) subleq 3633, 3633, 0
    2477, 0, 0 ,    // (3637) subleq 2477, 0, 0
    0, 3633, 0 ,    // (3640) subleq 0, 3633, 0
    0, 0, 0 ,   // (3643) subleq 0, 0, 0
    3652, 3633, 0 ,     // (3646) subleq 3652, 3633, 0
    0, 0, 3653 ,    // (3649) subleq 0, 0, 3653
    2 ,     // (3652) _SUB_LITERAL_val: dd 2
    3689, 3689, 0 ,     // (3653) subleq 3689, 3689, 0
    3633, 0, 0 ,    // (3656) subleq 3633, 0, 0
    0, 3689, 0 ,    // (3659) subleq 0, 3689, 0
    0, 0, 0 ,   // (3662) subleq 0, 0, 0
    3690, 3690, 0 ,     // (3665) subleq 3690, 3690, 0
    3633, 0, 0 ,    // (3668) subleq 3633, 0, 0
    0, 3690, 0 ,    // (3671) subleq 0, 3690, 0
    0, 0, 0 ,   // (3674) subleq 0, 0, 0
    3696, 3696, 0 ,     // (3677) subleq 3696, 3696, 0
    3633, 0, 0 ,    // (3680) subleq 3633, 0, 0
    0, 3696, 0 ,    // (3683) subleq 0, 3696, 0
    0, 0, 0 ,   // (3686) subleq 0, 0, 0
    0, 0, 0 ,   // (3689) _DEREF_DST_MOV_x: subleq 0, 0, 0
    3757, 0, 0 ,    // (3692) subleq 3757, 0, 0
    0, 0, 0 ,   // (3695) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (3698) subleq 0, 0, 0
```

```
    0, 0, 3705 ,    // (3701) subleq 0, 0, 3705
    1 ,     // (3704) _DEC_ONE: dd 1
    3704, 161, 0 ,      // (3705) _DEC_sub: subleq 3704, 161, 0
    3723, 3723, 0 ,     // (3708) subleq 3723, 3723, 0
    161, 0, 0 ,     // (3711) subleq 161, 0, 0
    0, 3723, 0 ,    // (3714) subleq 0, 3723, 0
    0, 0, 0 ,   // (3717) subleq 0, 0, 0
    3734, 3734, 0 ,     // (3720) subleq 3734, 3734, 0
    0, 0, 0 ,   // (3723) _DEREF_target: subleq 0, 0, 0
    0, 3734, 0 ,    // (3726) subleq 0, 3734, 0
    0, 0, 0 ,   // (3729) subleq 0, 0, 0
    0, 0, 0 ,   // (3732) _RET_tailjump: subleq 0, 0, 0
    0 ,     // (3735) _i: dd 0
    0 ,     // (3736) _chara: dd 0
    0 ,     // (3737) _charb: dd 0
    15 ,    // (3738) _fifteen: dd 15
    127 ,   // (3739) _xormask: dd 127
    220810, 188179, 193934, 182430, 211227, 182413, 193947, 224668, 222742, 213152,
186267, 182430, 188172, 224653, 192010, 209407 ,    // (3740) _answerkey: dd
220810, 188179, 193934, 182430, 211227, 182413, 193947, 224668, 222742, 213152,
186267, 182430, 188172, 224653, 192010, 209407
    3740 ,      // (3756) _answerkey_ptr: dd 3740
    1 ,     // (3757) _passbit: dd 1
    0, 0, 3762 ,    // (3758) printstring: subleq 0, 0, 3762
    0 ,     // (3761) _bp: dd 0
    3761, 3761, 0 ,     // (3762) subleq 3761, 3761, 0
    161, 0, 0 ,     // (3765) subleq 161, 0, 0
    0, 3761, 0 ,    // (3768) subleq 0, 3761, 0
    0, 0, 0 ,   // (3771) subleq 0, 0, 0
    0, 0, 3778 ,    // (3774) subleq 0, 0, 3778
    0 ,     // (3777) _string: dd 0
    0, 0, 3782 ,    // (3778) subleq 0, 0, 3782
    -2 ,    // (3781) _DECLARE_FUNCTION_ARGUMENT_bp_offset: dd -2
    0, 0, 3786 ,    // (3782) subleq 0, 0, 3786
    0 ,     // (3785) _DECLARE_FUNCTION_ARGUMENT_tmp: dd 0
    3785, 3785, 0 ,     // (3786) subleq 3785, 3785, 0
    3781, 0, 0 ,    // (3789) subleq 3781, 0, 0
    0, 3785, 0 ,    // (3792) subleq 0, 3785, 0
    0, 0, 0 ,   // (3795) subleq 0, 0, 0
    3761, 0, 0 ,    // (3798) subleq 3761, 0, 0
    0, 3785, 0 ,    // (3801) subleq 0, 3785, 0
    0, 0, 0 ,   // (3804) subleq 0, 0, 0
    3822, 3822, 0 ,     // (3807) subleq 3822, 3822, 0
    3785, 0, 0 ,    // (3810) subleq 3785, 0, 0
    0, 3822, 0 ,    // (3813) subleq 0, 3822, 0
    0, 0, 0 ,   // (3816) subleq 0, 0, 0
    3777, 3777, 0 ,     // (3819) subleq 3777, 3777, 0
    0, 0, 0 ,   // (3822) _DEREF_target: subleq 0, 0, 0
    0, 3777, 0 ,    // (3825) subleq 0, 3777, 0
```

```
    0, 0, 0 ,   // (3828) subleq 0, 0, 0
    0, 0, 3835 ,    // (3831) subleq 0, 0, 3835
    0 ,     // (3834) _char: dd 0
    3850, 3850, 0 ,     // (3835) _LOOP_START_LOOPID_print_loop: subleq 3850, 3850,
0
    3777, 0, 0 ,    // (3838) subleq 3777, 0, 0
    0, 3850, 0 ,    // (3841) subleq 0, 3850, 0
    0, 0, 0 ,   // (3844) subleq 0, 0, 0
    3834, 3834, 0 ,     // (3847) subleq 3834, 3834, 0
    0, 0, 0 ,   // (3850) _DEREF_target: subleq 0, 0, 0
    0, 3834, 0 ,    // (3853) subleq 0, 3834, 0
    0, 0, 0 ,   // (3856) subleq 0, 0, 0
    3834, 0, 3865 ,     // (3859) subleq 3834, 0, 3865
    0, 0, 3871 ,    // (3862) subleq 0, 0, 3871
    0, 0, 0 ,   // (3865) _BEZ_true: subleq 0, 0, 0
    0, 3834, 3874 ,     // (3868) subleq 0, 3834, 3874
    0, 0, 3877 ,    // (3871) subleq 0, 0, 3877
    0, 0, 3921 ,    // (3874) subleq 0, 0, 3921
    2, 2, 0 ,   // (3877) _print_loop_continue: subleq 2, 2, 0
    3834, 0, 0 ,    // (3880) subleq 3834, 0, 0
    0, 2, 0 ,   // (3883) subleq 0, 2, 0
    0, 0, 0 ,   // (3886) subleq 0, 0, 0
    4, 4, 0 ,   // (3889) subleq 4, 4, 0
    3904, 0, 0 ,    // (3892) subleq 3904, 0, 0
    0, 4, 0 ,   // (3895) subleq 0, 4, 0
    0, 0, 0 ,   // (3898) subleq 0, 0, 0
    0, 0, 3905 ,    // (3901) subleq 0, 0, 3905
    1 ,     // (3904) _SETVALUE_tmp: dd 1
    0, 0, 3909 ,    // (3905) subleq 0, 0, 3909
    1 ,     // (3908) _INC_ONE: dd 1
    3908, 0, 0 ,    // (3909) _INC_add: subleq 3908, 0, 0
    0, 3777, 0 ,    // (3912) subleq 0, 3777, 0
    0, 0, 0 ,   // (3915) subleq 0, 0, 0
    0, 0, 3835 ,    // (3918) subleq 0, 0, 3835
    0, 0, 3925 ,    // (3921) _LOOP_END_LOOPID_print_loop: subleq 0, 0, 3925
    1 ,     // (3924) _DEC_ONE: dd 1
    3924, 161, 0 ,      // (3925) _DEC_sub: subleq 3924, 161, 0
    3943, 3943, 0 ,     // (3928) subleq 3943, 3943, 0
    161, 0, 0 ,     // (3931) subleq 161, 0, 0
    0, 3943, 0 ,    // (3934) subleq 0, 3943, 0
    0, 0, 0 ,   // (3937) subleq 0, 0, 0
    3954, 3954, 0 ,     // (3940) subleq 3954, 3954, 0
    0, 0, 0 ,   // (3943) _DEREF_target: subleq 0, 0, 0
    0, 3954, 0 ,    // (3946) subleq 0, 3954, 0
    0, 0, 0 ,   // (3949) subleq 0, 0, 0
    0, 0, 0 ,   // (3952) _RET_tailjump: subleq 0, 0, 0
    0, 0, 3959 ,    // (3955) readinputstring: subleq 0, 0, 3959
    0 ,     // (3958) _bp: dd 0
    3958, 3958, 0 ,     // (3959) subleq 3958, 3958, 0
```

```
    161, 0, 0 ,     // (3962) subleq 161, 0, 0
    0, 3958, 0 ,    // (3965) subleq 0, 3958, 0
    0, 0, 0 ,   // (3968) subleq 0, 0, 0
    0, 0, 3975 ,    // (3971) subleq 0, 0, 3975
    0 ,     // (3974) _bufptr: dd 0
    0, 0, 3979 ,    // (3975) subleq 0, 0, 3979
    -2 ,    // (3978) _DECLARE_FUNCTION_ARGUMENT_bp_offset: dd -2
    0, 0, 3983 ,    // (3979) subleq 0, 0, 3983
    0 ,     // (3982) _DECLARE_FUNCTION_ARGUMENT_tmp: dd 0
    3982, 3982, 0 ,     // (3983) subleq 3982, 3982, 0
    3978, 0, 0 ,    // (3986) subleq 3978, 0, 0
    0, 3982, 0 ,    // (3989) subleq 0, 3982, 0
    0, 0, 0 ,   // (3992) subleq 0, 0, 0
    3958, 0, 0 ,    // (3995) subleq 3958, 0, 0
    0, 3982, 0 ,    // (3998) subleq 0, 3982, 0
    0, 0, 0 ,   // (4001) subleq 0, 0, 0
    4019, 4019, 0 ,     // (4004) subleq 4019, 4019, 0
    3982, 0, 0 ,    // (4007) subleq 3982, 0, 0
    0, 4019, 0 ,    // (4010) subleq 0, 4019, 0
    0, 0, 0 ,   // (4013) subleq 0, 0, 0
    3974, 3974, 0 ,     // (4016) subleq 3974, 3974, 0
    0, 0, 0 ,   // (4019) _DEREF_target: subleq 0, 0, 0
    0, 3974, 0 ,    // (4022) subleq 0, 3974, 0
    0, 0, 0 ,   // (4025) subleq 0, 0, 0
    0, 0, 4032 ,    // (4028) subleq 0, 0, 4032
    0 ,     // (4031) _buffersize: dd 0
    0, 0, 4036 ,    // (4032) subleq 0, 0, 4036
    -3 ,    // (4035) _DECLARE_FUNCTION_ARGUMENT_bp_offset: dd -3
    0, 0, 4040 ,    // (4036) subleq 0, 0, 4040
    0 ,     // (4039) _DECLARE_FUNCTION_ARGUMENT_tmp: dd 0
    4039, 4039, 0 ,     // (4040) subleq 4039, 4039, 0
    4035, 0, 0 ,    // (4043) subleq 4035, 0, 0
    0, 4039, 0 ,    // (4046) subleq 0, 4039, 0
    0, 0, 0 ,   // (4049) subleq 0, 0, 0
    3958, 0, 0 ,    // (4052) subleq 3958, 0, 0
    0, 4039, 0 ,    // (4055) subleq 0, 4039, 0
    0, 0, 0 ,   // (4058) subleq 0, 0, 0
    4076, 4076, 0 ,     // (4061) subleq 4076, 4076, 0
    4039, 0, 0 ,    // (4064) subleq 4039, 0, 0
    0, 4076, 0 ,    // (4067) subleq 0, 4076, 0
    0, 0, 0 ,   // (4070) subleq 0, 0, 0
    4031, 4031, 0 ,     // (4073) subleq 4031, 4031, 0
    0, 0, 0 ,   // (4076) _DEREF_target: subleq 0, 0, 0
    0, 4031, 0 ,    // (4079) subleq 0, 4031, 0
    0, 0, 0 ,   // (4082) subleq 0, 0, 0
    0, 0, 4089 ,    // (4085) subleq 0, 0, 4089
    0 ,     // (4088) _i: dd 0
    0, 0, 4093 ,    // (4089) subleq 0, 0, 4093
    10 ,    // (4092) _newline: dd 10
```

```
    0, 0, 4097 ,    // (4093) subleq 0, 0, 4097
    0 ,     // (4096) _tmp: dd 0
    0, 0, 4100 ,    // (4097) _FORLOOP_START_LOOPID_readinputstring_loop: subleq 0,
0, 4100
    4127, 4127, 0 ,     // (4100) _FORLOOP_START_compare: subleq 4127, 4127, 0
    4088, 0, 0 ,    // (4103) subleq 4088, 0, 0
    0, 4127, 0 ,    // (4106) subleq 0, 4127, 0
    0, 0, 0 ,   // (4109) subleq 0, 0, 0
    4031, 4127, 0 ,     // (4112) subleq 4031, 4127, 0
    4127, 0, 4121 ,     // (4115) subleq 4127, 0, 4121
    0, 0, 4128 ,    // (4118) subleq 0, 0, 4128
    0, 0, 0 ,   // (4121) _BLZ_false: subleq 0, 0, 0
    0, 0, 4318 ,    // (4124) subleq 0, 0, 4318
    0 ,     // (4127) _BGEQ_tmp: dd 0
    3, 3, 0 ,   // (4128) subleq 3, 3, 0
    4143, 0, 0 ,    // (4131) subleq 4143, 0, 0
    0, 3, 0 ,   // (4134) subleq 0, 3, 0
    0, 0, 0 ,   // (4137) subleq 0, 0, 0
    0, 0, 4144 ,    // (4140) subleq 0, 0, 4144
    1 ,     // (4143) _SETVALUE_tmp: dd 1
    4096, 4096, 0 ,     // (4144) subleq 4096, 4096, 0
    1, 0, 0 ,   // (4147) subleq 1, 0, 0
    0, 4096, 0 ,    // (4150) subleq 0, 4096, 0
    0, 0, 0 ,   // (4153) subleq 0, 0, 0
    4186, 4186, 0 ,     // (4156) subleq 4186, 4186, 0
    4096, 0, 0 ,    // (4159) subleq 4096, 0, 0
    0, 4186, 0 ,    // (4162) subleq 0, 4186, 0
    0, 0, 0 ,   // (4165) subleq 0, 0, 0
    4092, 4186, 0 ,     // (4168) subleq 4092, 4186, 0
    4186, 0, 4177 ,     // (4171) subleq 4186, 0, 4177
    0, 0, 4183 ,    // (4174) subleq 0, 0, 4183
    0, 0, 0 ,   // (4177) _BEZ_true: subleq 0, 0, 0
    0, 4186, 4187 ,     // (4180) subleq 0, 4186, 4187
    0, 0, 4241 ,    // (4183) subleq 0, 0, 4241
    0 ,     // (4186) _BNEQ_tmp: dd 0
    4096, 4096, 0 ,     // (4187) subleq 4096, 4096, 0
    4226, 4226, 0 ,     // (4190) subleq 4226, 4226, 0
    3974, 0, 0 ,    // (4193) subleq 3974, 0, 0
    0, 4226, 0 ,    // (4196) subleq 0, 4226, 0
    0, 0, 0 ,   // (4199) subleq 0, 0, 0
    4227, 4227, 0 ,     // (4202) subleq 4227, 4227, 0
    3974, 0, 0 ,    // (4205) subleq 3974, 0, 0
    0, 4227, 0 ,    // (4208) subleq 0, 4227, 0
    0, 0, 0 ,   // (4211) subleq 0, 0, 0
    4233, 4233, 0 ,     // (4214) subleq 4233, 4233, 0
    3974, 0, 0 ,    // (4217) subleq 3974, 0, 0
    0, 4233, 0 ,    // (4220) subleq 0, 4233, 0
    0, 0, 0 ,   // (4223) subleq 0, 0, 0
    0, 0, 0 ,   // (4226) _DEREF_DST_MOV_x: subleq 0, 0, 0
```

```
    4096, 0, 0 ,    // (4229) subleq 4096, 0, 0
    0, 0, 0 ,   // (4232) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (4235) subleq 0, 0, 0
    0, 0, 4318 ,    // (4238) subleq 0, 0, 4318
    4277, 4277, 0 ,     // (4241) _not_newline: subleq 4277, 4277, 0
    3974, 0, 0 ,    // (4244) subleq 3974, 0, 0
    0, 4277, 0 ,    // (4247) subleq 0, 4277, 0
    0, 0, 0 ,   // (4250) subleq 0, 0, 0
    4278, 4278, 0 ,     // (4253) subleq 4278, 4278, 0
    3974, 0, 0 ,    // (4256) subleq 3974, 0, 0
    0, 4278, 0 ,    // (4259) subleq 0, 4278, 0
    0, 0, 0 ,   // (4262) subleq 0, 0, 0
    4284, 4284, 0 ,     // (4265) subleq 4284, 4284, 0
    3974, 0, 0 ,    // (4268) subleq 3974, 0, 0
    0, 4284, 0 ,    // (4271) subleq 0, 4284, 0
    0, 0, 0 ,   // (4274) subleq 0, 0, 0
    0, 0, 0 ,   // (4277) _DEREF_DST_MOV_x: subleq 0, 0, 0
    4096, 0, 0 ,    // (4280) subleq 4096, 0, 0
    0, 0, 0 ,   // (4283) _DEREF_DST_MOV_y: subleq 0, 0, 0
    0, 0, 0 ,   // (4286) subleq 0, 0, 0
    0, 0, 4293 ,    // (4289) subleq 0, 0, 4293
    1 ,     // (4292) _INC_ONE: dd 1
    4292, 0, 0 ,    // (4293) _INC_add: subleq 4292, 0, 0
    0, 3974, 0 ,    // (4296) subleq 0, 3974, 0
    0, 0, 0 ,   // (4299) subleq 0, 0, 0
    0, 0, 4306 ,    // (4302) subleq 0, 0, 4306
    1 ,     // (4305) _INC_ONE: dd 1
    4305, 0, 0 ,    // (4306) _INC_add: subleq 4305, 0, 0
    0, 4088, 0 ,    // (4309) subleq 0, 4088, 0
    0, 0, 0 ,   // (4312) subleq 0, 0, 0
    0, 0, 4097 ,    // (4315) subleq 0, 0, 4097
    0, 0, 4322 ,    // (4318) _FORLOOP_END_LOOPID_readinputstring_loop: subleq 0,
0, 4322
    1 ,     // (4321) _DEC_ONE: dd 1
    4321, 161, 0 ,      // (4322) _DEC_sub: subleq 4321, 161, 0
    4340, 4340, 0 ,     // (4325) subleq 4340, 4340, 0
    161, 0, 0 ,     // (4328) subleq 161, 0, 0
    0, 4340, 0 ,    // (4331) subleq 0, 4340, 0
    0, 0, 0 ,   // (4334) subleq 0, 0, 0
    4351, 4351, 0 ,     // (4337) subleq 4351, 4351, 0
    0, 0, 0 ,   // (4340) _DEREF_target: subleq 0, 0, 0
    0, 4351, 0 ,    // (4343) subleq 0, 4351, 0
    0, 0, 0 ,   // (4346) subleq 0, 0, 0
    0, 0, 0     // (4349) _RET_tailjump: subleq 0, 0, 0
};
```