Systemsicherheit - 2. Übung

Dennis Rotärmel, Niklas Entschladen, Tobias Ratajczyk, Gruppe Q

May 12, 2019

1 Aufgabe 1

a)

Folgender Assembly-Code führt die Berechnung aus:

```
1         imul ecx, ecx ;ecx*ecx
2         sub ecx, ebx ;ecx*ecx-ebx
3         shl eax, 1 ;eax*2
4         add eax, ecx ;ecx*ecx-ebx+eax*2
5         add ebx, 0xaaaa ;ebx+0xaaaa
6         xor eax, ebx ;(ecx*ecx-ebx+eax*2)^(ebx+0xaaaa)
```

Listing 1.1: Assembler Code "Erstes Programm"

Die Ausgabe des Programms lautet: `12345`.

b)

Folgender Assembly-Code führt den Primzahltest aus, wobei die Teilbarkeit durch die Zahlen von 2 bis 31 geprüft wird:

```
1         ; ----- DO NOT MODIFY THE DATA SECTION -----
2         section .data
3
4         string_choice: db "Choose a number: ", 0
5         string_success: db "%d is prime.", 0xa, 0
6         string_failure: db "%d is not prime.", 0xa, 0
7         string_error: db "Invalid input. Exiting...", 0xa, 0
8         input_number: db "%d", 0
9
10        ; ----- DO NOT MODIFY THE BSS SECTION -----
11        section .bss
12        choice: resd 1
13
14        section .text
15
16        global main
17        ;; tell NASM that printf and scanf are symbols
18        ;; defined in another module
19        extern scanf, printf
20
```

```
21          ; ---- Your code goes into the TODO snippets ----
22
23          main:
24          ;-------------------------------
25          ; TODO
26          ; Ask user for input of a number (use string_choice)
27          push string_choice
28          call printf
29          add esp, 4
30          ;-------------------------------
31
32          ;-------------------------------
33          ; TODO
34          ; Read user input (use input_number)
35          push choice ;Trage uebergegebenen Wert in "choice" ein
36          push input_number
37          call scanf
38          add esp, 8
39          ;-------------------------------
40
41          ;-------------------------------
42          ; TODO
43          ; Check if user input is within valid range
44          mov eax, [choice]       ; Bewege "choice" in EAX
45          cmp eax, 2                      ; Fehlermeldung, falls choise<2
46          jb error
47          cmp eax, 1000           ; Fehlermeldung, falls choise>1000
48          ja error
49          mov esi, [choice]
50          ;-------------------------------
51
52          ;-------------------------------
53          ; DO NOT MODIFY THIS
54          ; Call is_prime subroutine
55          mov eax, [choice]  ; Load user input
56          push eax           ; Pass user's input via the stack
57          call is_prime      ; Call subroutine
58          add esp, 4         ; Cleanup stack
59
60          ; Check result (which is in eax)
61          cmp eax, 1
62          je success
63
64          ; If number is not prime, print failure message
```

```
65          push esi
66          push string_failure
67          call printf
68          add esp, 8
69          ret
70
71          ; If number is prime, print success message
72          success:
73          push esi
74          push string_success
75          call printf
76          add esp, 8
77          ret
78          ;-------------------------------
79
80          ;-------------------------------
81          ; TODO
82          ; Print error message (use string_error)
83          error:
84          push string_error
85          call printf
86          add esp, 4
87          ret
88          ;-------------------------------
89
90          ;-------------------------------
91          ; TODO
92          ; The function is_prime should contain your primality test.
93          ; Return 1 if the argument (passed via the stack) is prime, else 0.
94          ; Mark instructions belonging to the prologue and epilogue.
95          set_eax_true:
96          mov eax, 1
97          leave
98          retn
99          set_eax_false:
100         mov eax, 0
101         leave
102         retn
103         is_prime:
104         ; prologue
105         push ebp
106         mov ebp, esp
107         sub esp, 4h
108         ; end of prologue
```

```
109
110         ; function body
111         mov eax, [ebp+8]
112         xor edx, edx
113         mov ebx, 2
114         div ebx
115         cmp edx, 0
116         jz set_eax_false
117         mov eax, [ebp+8]
118         xor edx, edx
119         mov ebx, 3
120         div ebx
121         cmp edx, 0
122         jz set_eax_false
123         mov eax, [ebp+8]
124         xor edx, edx
125         mov ebx, 5
126         div ebx
127         cmp edx, 0
128         jz set_eax_false
129         mov eax, [ebp+8]
130         xor edx, edx
131         mov ebx, 7
132         div ebx
133         cmp edx, 0
134         jz set_eax_false
135         mov eax, [ebp+8]
136         xor edx, edx
137         mov ebx, 11
138         div ebx
139         cmp edx, 0
140         jz set_eax_false
141         mov eax, [ebp+8]
142         xor edx, edx
143         mov ebx, 13
144         div ebx
145         cmp edx, 0
146         jz set_eax_false
147         mov eax, [ebp+8]
148         xor edx, edx
149         mov ebx, 17
150         div ebx
151         cmp edx, 0
152         jz set_eax_false
```

```
153         mov eax, [ebp+8]
154         xor edx, edx
155         mov ebx, 19
156         div ebx
157         cmp edx, 0
158         jz set_eax_false
159         mov eax, [ebp+8]
160         xor edx, edx
161         mov ebx, 23
162         div ebx
163         cmp edx, 0
164         jz set_eax_false
165         mov eax, [ebp+8]
166         xor edx, edx
167         mov ebx, 29
168         div ebx
169         cmp edx, 0
170         jz set_eax_false
171         mov eax, [ebp+8]
172         xor edx, edx
173         mov ebx, 31
174         div ebx
175         cmp edx, 0
176         jz set_eax_false
177         call set_eax_true
178         ; end of function body
179
180         ; epilogue
181         leave
182         retn
183         ; end of epilogue / end of function
184         ;-------------------------------
```

Listing 1.2: Assembler Code "Primzahltest"

2 Aufgabe 2

a)

Bei der Ausführung des Programmes wird der jeweils i-te Buchstabe um den Wert i-1 erhöht (siehe ASCII-Tabelle). Danach wird der daraus entstehende String mit folgendem String vergleichen: "HPFRV". Daraus lässt sich schlussfolgern, dass das Schlüsselwort "HODOR" lautet. Die Eingabe des Wortes bestätigt dies. Bis auf den Befehl "break verif_key" und die dazugehörigen step- und continue-Anweisungen wurden keine weiteren Befehle benötigt.

b)

```
1   #include <stdio.h>
2
3   int verify_key(char *str){
4   char key[5] = "HPFRV";
5
6   for(int i=0; i<5; i++){
7   if (str[i]!=key[i]){
8   printf("Key is not valid :(\n");
9   return 0;
10  }
11  }
12  printf("Key is valid! Whoop whoop :)\n");
13  return 0;
14  }
15
16  int main(){
17  char str[5];
18  printf("Enter serial (5 capital letters): ");
19  scanf("%s", str);
20
21  for(int i=0; i<5; i++){
22  str[i]=str[i]+i;
23  }
24
25  return verify_key(str);
```

```
26              }
```

Listing 2.1: Crackme-Code

3 Aufgabe 3

a)

- **Data Movement**:
- `mov esi, 4` → Speichert den Wert 4 im esi-Register
- `push 2` → Bewegt den Wert 2 auf den Stack
- `push esi` → Bewegt den Wert des esi-Registers (4) auf den Stack
- `mov eax, dword ptr [esp]` → Der Wert, auf den der esp zeigt (4), wird in das eax-Register geladen → eax = 4
- `lea eax, [esp + eax * 2 + 4]` → Die Adresse des esp, addiert mit dem doppelten Wert des eax und der Konstanten 4, wird in das eax-Register geladen → eax = esp+12
- `sub eax, 8` → Vom eax-register wird der Wert 8 abgezogen → eax = esp+4
- `mov eax, dword ptr [eax]` → In das eax wird der Wert an der Stelle [esp+4] geladen → eax = 2
- `pop ebx` → Der ursprüngliche Wert des esi-Registers wird in das ebx-Register geschrieben und vom Stack entfernt → ebx = 4
- `add esp, 4` → Der esp wird 4 Bytes nach oben verschoben (bezüglich des Stacks) → Der Stack wird aufgeräumt
- `add eax, ebx` → eax = 2 + 4 = 6
- **Arithmetic and Logic**:
  - `xor eax, eax` $\hat{=}$ eax $\oplus$ eax = 0
  - `add eax, 1234h` $\hat{=}$ eax + 4660 = 0 + 4660 = 4660
  - `ror eax, 16` $\hat{=}$ $0001001000110100_2 \rightarrow 0011010000010010_2 \hat{=} 13330_{10}$
  - `or eax, 55h` $\hat{=} 0011010000010010_2 \vee 0000000001010101_2 = 0011010001010111_2 \hat{=} 13399_{10}$
  - `inc eax` $\hat{=}$ = eax + 1 = 13400
  - `shl ax, 8` $\hat{=} 0011010001011000_2 \rightarrow 0011010001011000_2$ (ax = 0, somit keine Änderung)

  - `mov al, 78h` $\hat{=} 0011010001011000_2 \rightarrow 0011010001110100_2 \hat{=} 13428_{10}$
  - Damit ist am Ende der Wert 13428 im `eax` Register

- **Control Flow**:
  - `mov eax, 1h` $\hat{=}$ eax $= 00\ldots0001$
  - `neg eax` $\hat{=}$ eax $= 11\ldots1111$
  - `mov ebx, FFFFFFF8h`
  - `cmp eax, ebx`
  - `jg true` $\rightarrow$ ebx ist größer als eax (da Vorzeichen beachtet)
  - `mov eax, 0` wird ausgeführt
  - damit ist im eax der Wert 0 am Ende.

b)

`ja` ist ein unsigned-Vergleich, damit wird beim vergleichen das Vorzeichen nicht beachtet. In diesem Fall ist der Wert im `eax` größer und es wird `mov eax, 1` ausgeführt.

4 Aufgabe 4

a)

```
1          int fkt_f(int a, int b, int c){
2          int d = 0;
3          if(a!=0){
4          d = fkt_g(a,b);
5          }
6          else{
7          d = a+b;
8          }
9          int e = c+d;
10         return e;
11         }
12         int fkt_g(int a, intb){
13         int f = 0;
14         if(f<b){
15         a = a+b;
16         f = 1;
17         }
18         return a;
19         }
```

Listing 4.1: Funktionen f und g

b)

Calling Convention: cdecl
Der Caller ruft erst die Subroutine auf (call ...) und gibt danach in der nächsten Instruktion wieder den Platz im Stack frei (add esp, ...). Außerdem werden sämtliche Paramter über den Stack übergeben.

- 1. Paramter: `[ebp+8]`
- 2. Paramter: `[ebp+0Ch]`
- 3. Paramter: `[ebp+10h]`

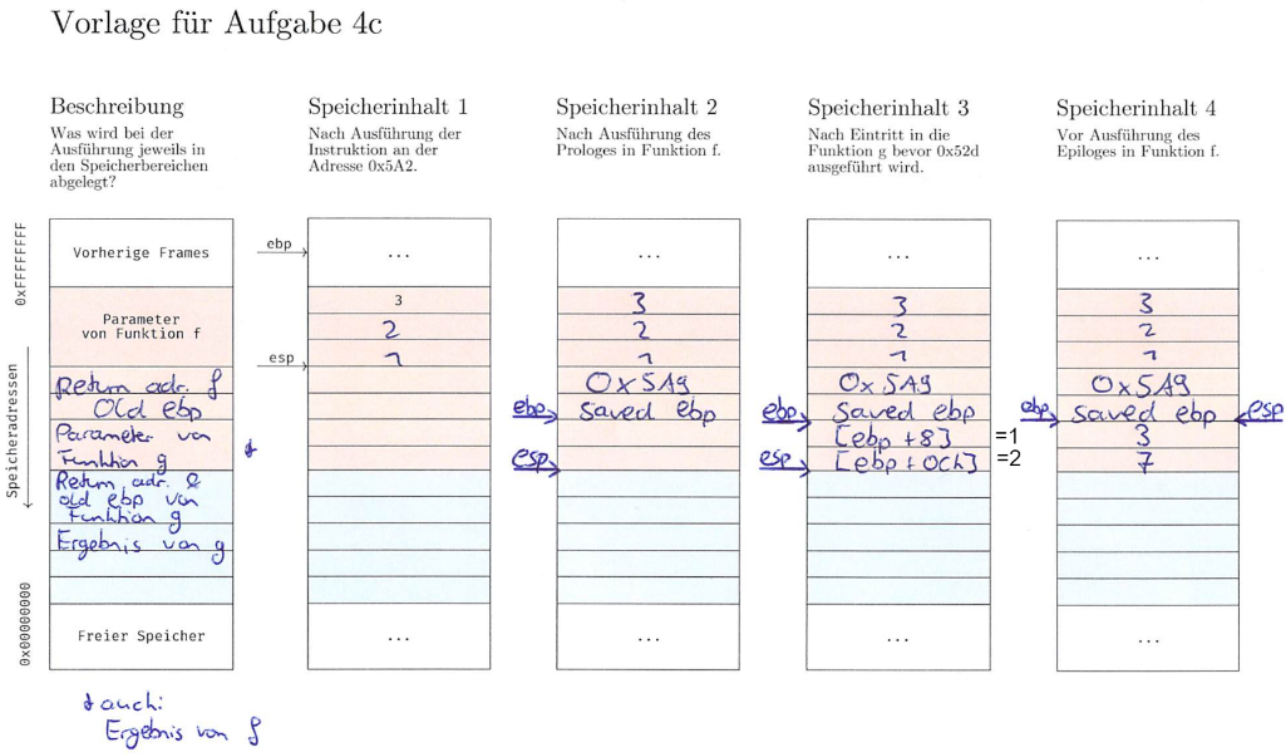

Figure 4.1: Zustände des Stacks