RS 12 (HA) zum 25.01.2013

Paul Bienkowski, Hans Ole Hatzel

24. Januar 2013

1. a) `MEM[0x0000 0100] = 0x0000 CB02`

b) `MEM[0x0000 0104] = 0x0000 00BF`

c) `MEM[0x0000 0108] = 0x0024 6000`

d) `MEM[0x0000 010C] = 0x0000 0043`

e) `%ecx = 0x0000 000B`

f) `%eax = 0x0000 00FC`

2. Da ein beliebiges Bit per XOR mit sich selbst kombiniert immer 0 ergibt, lässt sich auch einfach schreiben:

```
xor %eax, %eax
```

3. Um den Program-Counter zu erhalten, wird ein Hilfssprung als Funktionsaufruf ausgeführt. Der `call`-Befehl schreibt dabei den Programmzähler auf den Stack und springt zum Label (`jmp`). Danach kann man den Zähler manuell vom Stack holen:

```
    call helper_label:
helper_label:
    pop %eax
```

4.
```
f2c:
    pushl   %ebp                ; standard-setup
    movl    %esp, %ebp

    movl    8(%ebp), %eax       ; eax = f (first and only argument)
    subl    $0x20, %eax         ; eax = (f - 32)
    imull   $0x8E, %eax         ; eax = (f - 32) * 142
    sarl    $8, %eax            ; eax = (f - 32) * 142 / 256

    movl    %ebp, %esp          ; standard-return
    popl    %ebp
    ret
```

5. a)

```
myst:
    pushl   %ebx                ; save ebx on stack
    subl    $24, %esp           ; esp -= 24 (6 words)
    movl    32(%esp), %ebx      ; ebx = first parameter
    movl    36(%esp), %edx      ; edx = second parameter
    movl    $1, %eax            ; eax = 1
    testl   %edx, %edx          ; ZF = (edx == 0)
    je      .L2                 ; jump to L2 if edx is 0
    subl    $1, %edx            ; edx--
    movl    %edx,  4(%esp)      ; second paramter = edx
    movl    %ebx, (%esp)        ; first parameter = ebx
    call    myst                ; recursive call: myst(edx, ebx)
    imull   %ebx, %eax          ; eax *= ebx
.L2:
    addl    $24, %esp           ; reset stack pointer
    popl    %ebx                ; reset ebx from stack
    ret                         ; return to caller
```

Dies ergibt folgenden C-Code:

```
int myst(int a, int b) {
    int c = 1;
    if(b == 0) {
        return 1;
    }
    b--;
    c = myst(a, b) * a;
    return c;
}
```

Eine noch kompaktere Schreibweise wäre:

```
int myst(int a, int b) {
    if(b == 0) return 1;
    return myst(a, b - 1) * a;
}
```

b) Es ist klar zu erkennen, dass das Unterprogramm/die Funktion die Potenz a^b berechnet.

c)

b = 2
a = 5
%eip main
%ebx main
b = 1
a = 5
%eax myst-1
%eip myst-1
%ebx myst-1
b = 0
a = 5
%eax myst-2
%eip myst-3
%ebx myst-3
0
0
0

d) Die Abbruchbedingung wird nicht sofort erreicht. Bei 16 bit Wortbreite werden $2^{16} - 3 = 65533$ Rekursionsaufrufe (nach *integer underflow*) gestartet. Bei jedem Aufruf wird der Stack jeweils um 5 Worte (10 Byte) größer, damit wird ein Stack von 655.33 KB benötigt. Ist der Stack dafür zu klein, wird ein *stack overflow* vermutlich vom Kernel erkannt und das Programm abbrechen. Ansonsten wird das Ergebnis von 3^{65533} berechnet (etwa $1,5 \cdot 10^{31267}$), welches selbst mehrmals überläuft und somit $3^{65533} \bmod 2^{16} = 55827$ ergibt.