1 . Hausübung

Gruppe: - Ulf Gebhardt (rbg: hu56nifa)
- Michael Scholz (rbg: mi48azih)

Aufgabe 1 : Reverse-Engineering:

a)

linkes Programm

%eax	%ebx	%ecx	%edx
7	4	24	0

rechtes Programm

%eax	%ebx	%ecx	%edx
7	4	24	12

b)

linkes Programm

```
.data
[….]

.text
movl a, %eax    | 8    Takte
movl b, %ebx    | 8    Takte
movl d, %ecx    | 8    Takte
addl $2, %eax   | 4    Takte
addl $1, %ebx   | 4    Takte
addl e, %ecx    | 9    Takte
div c           | 159  Takte
                200 Takte
```

rechtes Programm

```
.data
[...]

.text
movl a, %eax        | 8 Takte
movl b, %ebx        | 8 Takte
movl d, %ecx        | 8 Takte
movl e, %edx        | 8 Takte
addl $2, %eax       | 4 Takte
incl %ebx           | 3 Takte
addl %edx, %ecx     | 3 Takte
shrl $1, %eax       | 2 Takte
                    44 Takte
```

c)

Mit ***shll $1, Reg*** kann man eine Multiplikation mit dem Faktor zwei schnell ausführen. Der Befehl schiebt die angegebene Anzahl an Bits (in diesem Fall 1) nach links, wobei rechts Nullen aufgefüllt werden und die links herausfallenden Bits verlorengehen. Dies entspricht einer Multiplikation mit dem Faktor 2^1, also 2.

Aufgabe 2 : Parity-Bit:

```
.data
intout:
.string "Wert %d\n"

.text

.globl main

main:

	# Wert fuer die Berechnung
	movl $13, %eax			# 13 == 0b1101
	movl %eax, %ecx			# Kopiere %eax in %ecx
	xor %ebx, %ebx			# Initialisiere counter %ebx mit 0. "xor" braucht
					# hierbei weniger Takte als "movl"


	# erstes Bit
	shrl $3, %ecx			# Schiebe Bits um drei Stellen nach rechts. Aus 1101
					# wird 0001.
	movl %ecx, %ebx			# Verschiebe in counter

	# zweites Bit
	movl %eax, %ecx			# Hole Ursprungszahl aus %eax
	shrl $2, %ecx			# Schiebe Bits um zwei Stellen nach rechts. Aus 1101
					# wird 0011.
	and $0b0001, %ecx		# Maskieren: Somit werden alle Bits, die uns nicht
					# interesieren, genullt.
	add %ecx, %ebx			# counter += %ecx

	#drittes Bit
	movl %eax, %ecx			# Hole Ursprungszahl aus %eax
	shrl $1, %ecx			# Schiebe Bits um eine Stelle nach rechts. Aus 1101
					# wird 0110.
	and $0b0001, %ecx		# Maskieren: Somit werden alle Bits, die uns nicht
					# interesieren, genullt.
	add %ecx, %ebx			# counter += %ecx

	#viertes Bit
	movl %eax, %ecx			# Hole Ursprungszahl aus %eax
	and $0b0001, %ecx		# Maskieren: Somit werden alle Bits, die uns nicht
					# interesieren, genullt.
	add %ecx, %ebx			# counter += %ecx



	# %ebx nochmals maskieren, da es nur auf des lsb ankommt. Ist dieses gesetzt, so
	# ist die Zahl ungerade. Ist es nicht gesetzt, so ist die Zahl gerade.

	and $0b0001, %ebx

	shll $1, %eax			# Schiebe Bits um eine Stelle nach links, um eine "0"
					# oder "1" anhängen zu können.
	add %ebx, %eax			# %eax += %ebx.


# Wert im %eax ausgeben
pushl %eax
pushl $intout
call printf


# Exit
movl $1, %eax
int $0x80
```