Report Assignment 2

Assembly Forensics

Group Member

Platzer Andreas	11771913
Reindl Hanner	01532129
Manuel Sammer	11903022

Getting Ready

We used the GNU Tools ARM Embedded arm-none-eabi binary command line executeables to compile the assembler program.

Nothing happened with the Raspberry Pi as predicted.

More Blinking!

What does it do?

It blinks! To be more precise: It sends SOS in morse code very fast, in an endless loop.

“*.*.*....***.***.***....*.*.*.....” repeat!

* => One “dot_length” unit, LED ON

. => One “dot_length” unit, LED OFF

How to get the Symbol Table and Assembler Code

Assembler Code:

```
"C:\Program Files (x86)\GNU Tools ARM Embedded\8 2019-q3-update\bin\arm-none-eabi-objdump" -d
kernel.elf > kernel_dump.asm
```

Symbol Table:

```
"C:\Program Files (x86)\GNU Tools ARM Embedded\8 2019-q3-update\bin\arm-none-eabi-objdump" -t
kernel.elf > kernel_dump_symbols.txt
```

List of used Assembler Functions and other interesting bites:

ldr rN, [ADRESS]	It's dereferencing the adress inside the [] brakets and loads the value of that address into the register with the number N.
str r3, [r2]	Take the value of r2, dereference it and save the value of r2 into the dereferenced space.
mov r1, #0	Save DEC Value 0 into register r1
orr r2, r1, 0x4000	Logical Or operation. Use r1 and the value (e.g 0x4000) and do a logical or operation. Save value in r2.
pc	Special Register called program counter.
b[cond] label	Branch with added condition. Jump to given label inside the code when condition is true. Else continue with the next assembly function.
rsb rN, rX, rY, Operator	Subtract rX from rY modified by Operator and save it into rN. For example: rsb r1, r1, r1, lsl #3 => r1 = 8*r1 – r1
add rN, rX , Value	Add rX and Value and save it in rN

Condition Code	Meaning	Flags Tested
EQ	Equal (==)	Z == 1
NE	Not Equal (!=)	Z == 0
GT	Signed >	(Z==0)&&(N==V)
LT	Signed <	N != V
GE	Signed >=	N == V
LE	Signed <=	(Z==1) || (N!=V)
CS or HS	U. Higher or Same	C == 1
CC or LO	U. Lower	C == 0
MI	Negative -	N == 1
PL	Positive +	N == 0
AL	Always executed	-
NV	Never executed	-
VS	S. Overflow	V == 1
VC	No Overflow	V == 0
HI	U. Higher	(C==1)&&(Z==0)
LS	U. Lower or same	(C==0)||(Z==1)

Source: https://azeria-labs.com/assembly-basics-cheatsheet/

Explanation of Code with the Flowchart

See additional image called flowchart.jpg which provides an overview of the whole program with explanations of the code.

Based on this flowchart we answered the following questions and created our pseudocode.

Which parts switch the LED?

Offset 0x8030 adjusts gpio[1] with the value from the logical or operation at offset 0x8028 declaring the LED as an output.

Offset 0x8048 & 0x8104 & 0x81c4 save the value 0x10000 in gpio[10] which sets the Port to low (Clear). Which turns on the LED.

Offset 0x8078 & 0x8138 & 0x81f4 saves the value 0x10000 in gpio[7] which sets the Port to high (Set).
Which turns off the LED.

Which parts implement the timing between blinks?

The code uses simple loops to adjust the timing between blinks. This can be seen when you look at the flowchart.

The following Offsets are used to time how long a LED is shining in case dot_length > 0:

- 0x804c to 0x8074 (1 dot_length unit)
- 0x8108 to 0x8134 (3 dot_length units)
- 0x81c8 to 0x81f0 (1 dot_length unit)

The following Offsets are used to time how long a LED is dark in case dot_length > 0:

- 0x807c to 0x80a4 (1 dot_length unit)
- 0x813c to 0x8164 (1 dot_length unit)
- 0x81f8 to 0x8220 (1 dot_length unit)

The following Offsets are responsible to repeat the LED going on and off for X times
in case morser > 0:

- 0x8034 to 0x8044 & 0x80a8 to 0x80bc (3 Times)
- 0x80f0 to 0x8100 & 0x8168 to 0x817c (3 Times)
- 0x81b0 to 0x81c0 & 0x8224 to 0x8238 (3 Times)

The following Offsets are responsible to take a break between the characters and after a word:

- 0x80c0 to 0x80ec (3*dot_length)
- 0x8180 to 0x81ac (3*dot_length)
- 0x823c to 0x8268 (7*dot_length)

The following Offset is responsible for an endless loop:

- 0x8250 & 0x826c

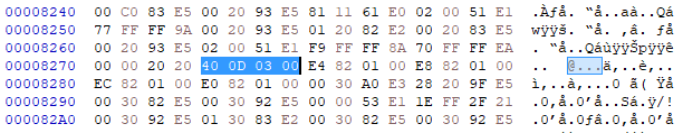
Pseudocode

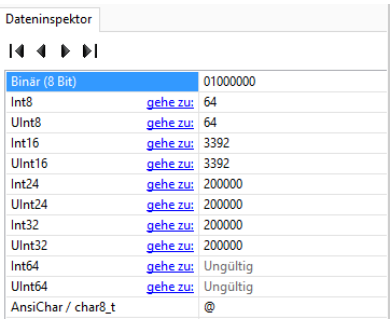
You can find the pseudocode in the pseudocode.txt file

Too much Blinking!

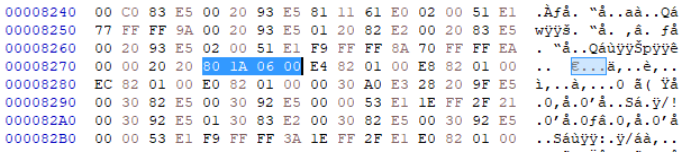
At the offset 0x8274, there is a DEC value of 200000. This one gets saved once into the memory address of “dot_length” (0x000182e4) (see flowchart) which gave us the idea of simply editing the value at the offset 0x8274 to 400000 via HxD Editor which worked instantly.

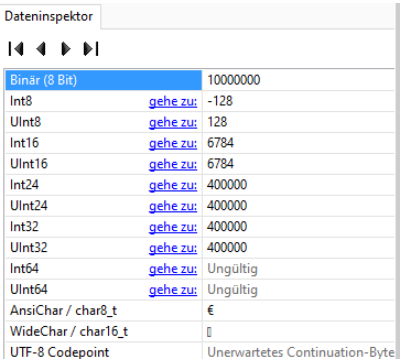
Before edit:





Edited:





Pitfalls

While working on the assembler code we forgot to look first for loops, jumps etc.

This made it quite hard and we took quite a long time to get the full picture of the program. It got a lot clearer after creating a first prototype for the flowchart which then got adjusted in the final version.

With the flowchart it was quite easy to understand what was going on.