An introduction to Firmware Analysis

Stefan Widmann

stefan.widmann@gmx.de

CCC Regensburg

30C3 in Hamburg

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Contents

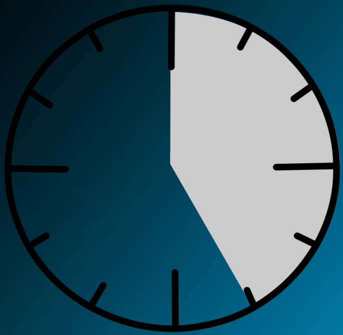
- Motivation
- Prerequisites
- Obtaining a Firmware image
- Analyzing the Firmware image
- Modifying the Firmware image

Motivation

- Interoperablility
- Fixing bugs because the manufacturer doesn't
- Forensics

Prerequisites

- Knowledge about embedded system's architectures
- Good knowledge of assembler languages
- Don't rely on decompilers
- To practice: Compile code on an embedded platform, then analyze the assembly output
- Device programmer?
- Time, time, time



Obtaining a Firmware image
- non-invasive -

- Download plain binary from manufacturer
- Download extracted binary from internet
- Download Bootdisk / USB / CD boot images, extract using Winrar (Windows) or mount the image (Linux)
- .bin / .hex / .s19 / .mot / .rom / .raw
- Convert non-bin-files to bin (e.g. hex2bin)

Obtaining a Firmware image
- non-invasive -

- Download an updater from manufacturer
  - (.exe → Windows)
- Updater types:
  - Selfextracting archive
  - Installer (like InstallShield, ...)
  - Updater containing an image
  - Updater downloading an image
  - Packed updater (UPX, PECompact, ...)

Obtaining a Firmware image
- non-invasive -

- Selfextracting archive:
  - „RARSFX“ signatures → unrar, WinRAR
  - „PK“ signatures → rename to .zip, unzip
- Installer (e.g. InstallShield)
  - Special unpacker (hard to use)
  - Let it install
    - Plain image
    - Updater

Obtaining a Firmware image
- non-invasive -

- Updater containing an image
  - Search image in executable (Hexeditor)
  - Writing a file → ProcessMonitor
- Updater downloading an image
  - Download to file → ProcessMonitor
  - Download to RAM → Debugger → Dump

Obtaining a Firmware image
- non-invasive -

- Packed Updater
  - Standard UPX → upx -d to unpack
  - Modified UPX → special unpacker
  - Other packers → special unpacker

Obtaining a Firmware image
- non-invasive -

- Problem: Compressed images
  - Normally unpacked before writing to the device (in RAM) → Debugger → Dump
  - Hard: FW is sent compressed to device → invasive techniques

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Obtaining a Firmware image
- non-invasive -

- Sniffing update transfer
  - WinXP: TraceSPTI (IDE / SATA / USB)
  - Linux: Wireshark (USB)
  - Various other tools
  - Problem: Image has to be reconstructed

Obtaining a Firmware image - invasive? -

- Serial interfaces
  - Embedded Linux? → serial console
  - JTAG
  - More information: 27C3 talk

    „JTAG/Serial/FLASH/PCB Embedded Reverse Engineering Tools and Techniques“

    https://www.youtube.com/watch?v=pCeedinviN0

Obtaining a Firmware image
- invasive -

- FW in memory devices
  - (E)PROM (27...)
  - EEPROM / FLASH (28... / 29... / 39... / 49...)
  - Serial FLASH (25..., sometimes even 24...)
    - Becoming standard
    - Cheap readers / programmers

Obtaining a Firmware image
- invasive -

- FW in chip-internal memories
  - Proprietary interfaces → try using IDE
  - JTAG
  - Bootloaders (in ROM)
  - Microprobing
    - More information: 29C3 talk „Low-Cost Chip Microprobing“ https://www.youtube.com/watch?v=b_MsQRpwRlw

Obtaining a Firmware image - invasive -

- CPLDs and FPGAs
  - CPLDs: internal EEPROM
  - FPGAs: internal SRAM, external serial FLASH
  - are sold to be reverse-engineer-proof

We have the image!

Congratulations!

You've got a FW binary in front of you!

But what's next?

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Analyzing the FW binary

- Finding out processor / controller type
  - Datasheets?
  - Internet?
  - Trial and error, trying n different disassemblers
    - Specific disassembler
    - IDA
    - ODA (OnlineDisAssembler)

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Analyzing the FW binary

```
0e 43 4e 93
04 34 3c 40
07 00 7f e3
5e 53 4c 93
03 34 5e e3
7f e3 5c 53
0d 12 b0 12
34 12 3a 41
6d b3 02 24
7d e3 5b 53
5d b3 02 24
7f e3 5e 53
30 41
```

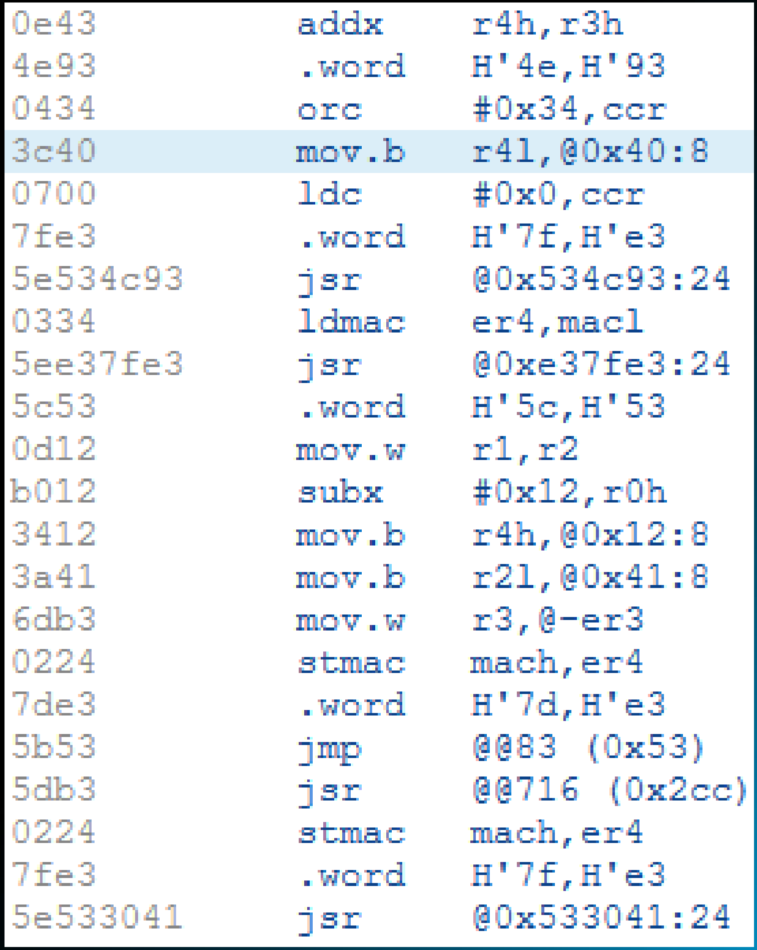
```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Analyzing the FW binary

```
0e 43 4e 93
04 34 3c 40
07 00 7f e3
5e 53 4c 93
03 34 5e e3
7f e3 5c 53
0d 12 b0 12
34 12 3a 41
6d b3 02 24
7d e3 5b 53
5d b3 02 24
7f e3 5e 53
30 41
```

H8s?

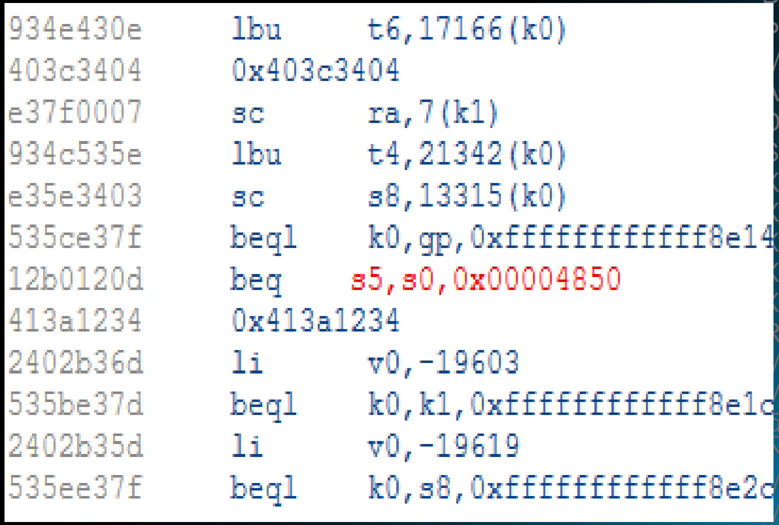


```
0e43        addx     r4h,r3h
4e93        .word    H'4e,H'93
0434        orc      #0x34,ccr
3c40        mov.b    r4l,@0x40:8
0700        ldc      #0x0,ccr
7fe3        .word    H'7f,H'e3
5e534c93    jsr      @0x534c93:24
0334        ldmac    er4,macl
5ee37fe3    jsr      @0xe37fe3:24
5c53        .word    H'5c,H'53
0d12        mov.w    r1,r2
b012        subx     #0x12,r0h
3412        mov.b    r4h,@0x12:8
3a41        mov.b    r2l,@0x41:8
6db3        mov.w    r3,@-er3
0224        stmac    mach,er4
7de3        .word    H'7d,H'e3
5b53        jmp      @@83 (0x53)
5db3        jsr      @@716 (0x2cc)
0224        stmac    mach,er4
7fe3        .word    H'7f,H'e3
5e533041    jsr      @0x533041:24
```

Analyzing the FW binary

```
0e 43 4e 93
04 34 3c 40
07 00 7f e3
5e 53 4c 93
03 34 5e e3
7f e3 5c 53
0d 12 b0 12
34 12 3a 41
6d b3 02 24
7d e3 5b 53
5d b3 02 24
7f e3 5e 53
30 41
```

MIPS?



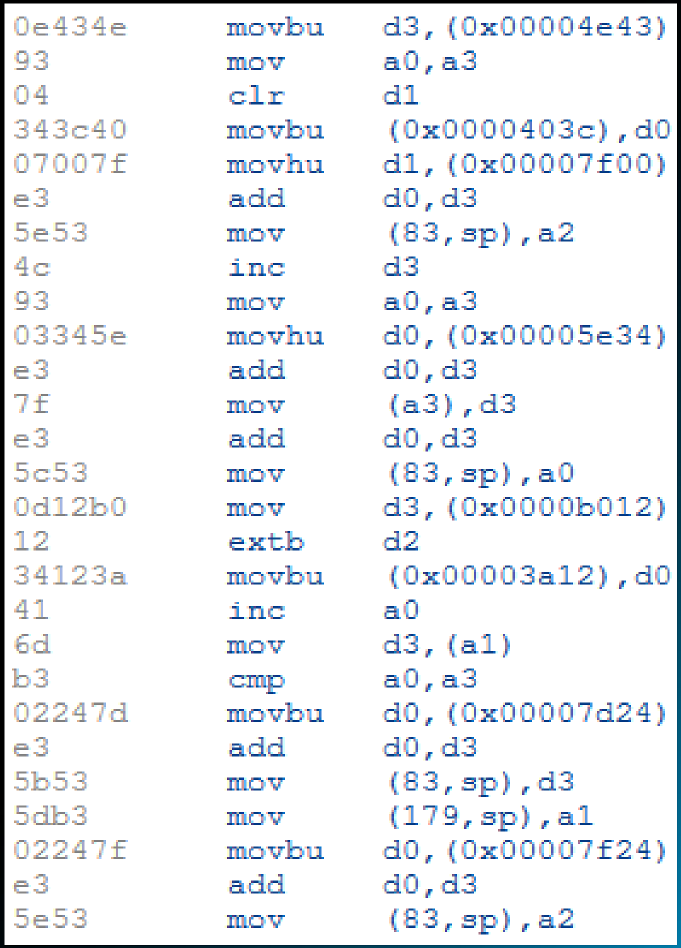
```
934e430e    lbu     t6,17166(k0)
403c3404    0x403c3404
e37f0007    sc      ra,7(k1)
934c535e    lbu     t4,21342(k0)
e35e3403    sc      s8,13315(k0)
535ce37f    beql    k0,gp,0xfffffffffffff8e14
12b0120d    beq     s5,s0,0x00004850
413a1234    0x413a1234
2402b36d    li      v0,-19603
535be37d    beql    k0,k1,0xfffffffffffff8e1c
2402b35d    li      v0,-19619
535ee37f    beql    k0,s8,0xfffffffffffff8e2c
```

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Analyzing the FW binary

```
0e 43 4e 93
04 34 3c 40
07 00 7f e3
5e 53 4c 93
03 34 5e e3
7f e3 5c 53
0d 12 b0 12
34 12 3a 41
6d b3 02 24
7d e3 5b 53
5d b3 02 24
7f e3 5e 53
30 41
```

MN103?



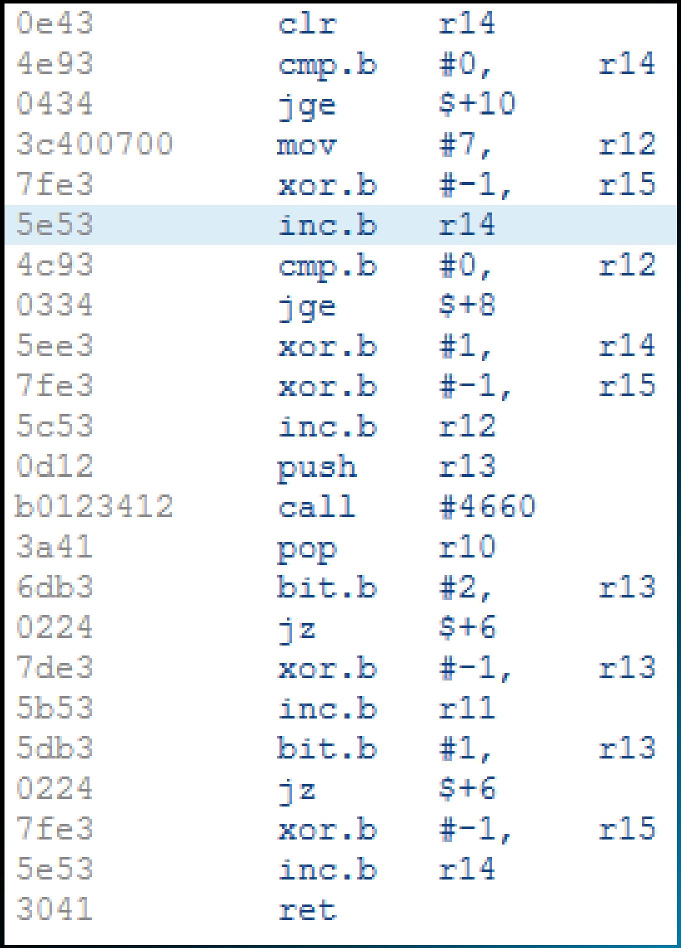
```
0e434e     movbu   d3,(0x00004e43)
93         mov     a0,a3
04         clr     d1
343c40     movbu   (0x0000403c),d0
07007f     movhu   d1,(0x00007f00)
e3         add     d0,d3
5e53       mov     (83,sp),a2
4c         inc     d3
93         mov     a0,a3
03345e     movhu   d0,(0x00005e34)
e3         add     d0,d3
7f         mov     (a3),d3
e3         add     d0,d3
5c53       mov     (83,sp),a0
0d12b0     mov     d3,(0x0000b012)
12         extb    d2
34123a     movbu   (0x00003a12),d0
41         inc     a0
6d         mov     d3,(a1)
b3         cmp     a0,a3
02247d     movbu   d0,(0x00007d24)
e3         add     d0,d3
5b53       mov     (83,sp),d3
5db3       mov     (179,sp),a1
02247f     movbu   d0,(0x00007f24)
e3         add     d0,d3
5e53       mov     (83,sp),a2
```

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Analyzing the FW binary

```
0e 43 4e 93
04 34 3c 40
07 00 7f e3
5e 53 4c 93
03 34 5e e3
7f e3 5c 53
0d 12 b0 12
34 12 3a 41
6d b3 02 24
7d e3 5b 53
5d b3 02 24
7f e3 5e 53
30 41
```

MSP430?



```
0e43        clr     r14
4e93        cmp.b   #0,     r14
0434        jge     $+10
3c400700    mov     #7,     r12
7fe3        xor.b   #-1,    r15
5e53        inc.b   r14
4c93        cmp.b   #0,     r12
0334        jge     $+8
5ee3        xor.b   #1,     r14
7fe3        xor.b   #-1,    r15
5c53        inc.b   r12
0d12        push    r13
b0123412    call    #4660
3a41        pop     r10
6db3        bit.b   #2,     r13
0224        jz      $+6
7de3        xor.b   #-1,    r13
5b53        inc.b   r11
5db3        bit.b   #1,     r13
0224        jz      $+6
7fe3        xor.b   #-1,    r15
5e53        inc.b   r14
3041        ret
```

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Analyzing the FW binary

- Offset in file often != offset in address space
- No real problem with relative adressing
- Big problem in absolute adressing
- Entry point unknown
- Interrupt vectors unknown
- Subroutine calls do not make sense
- → Load offset must be found out

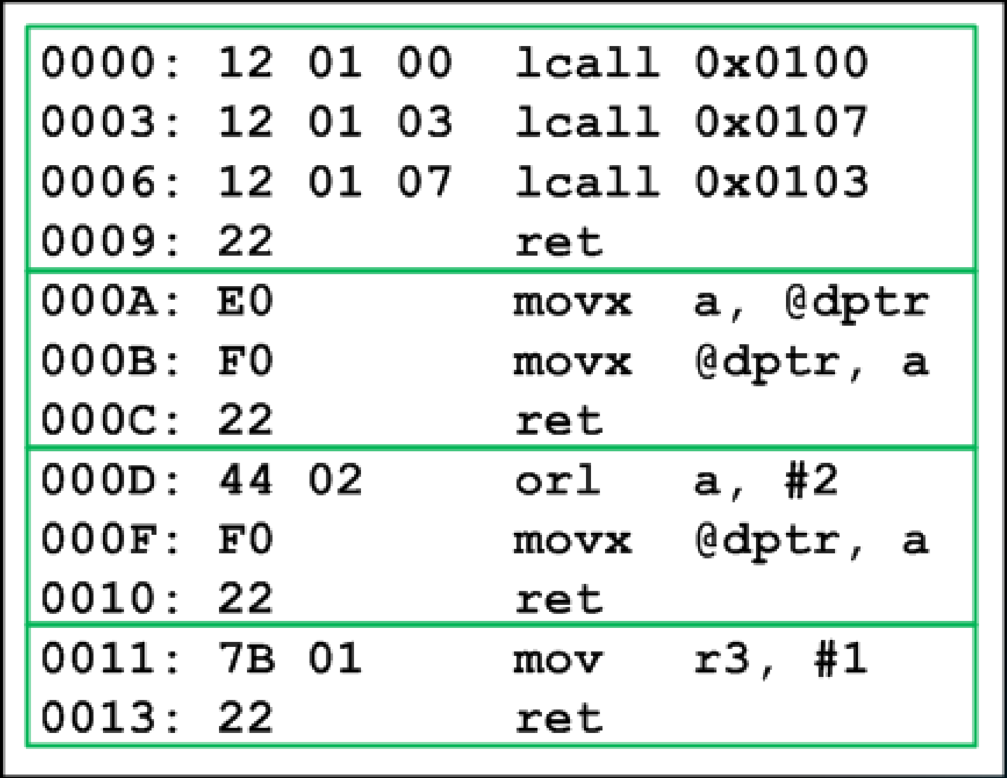
```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Analyzing the FW binary

- Determination of load offset:
  - Method „call distance search“
  - Select closely located subroutines addresses
  - Decide to use either return instructions or function entry sequences
  - Build search string containing wildcards

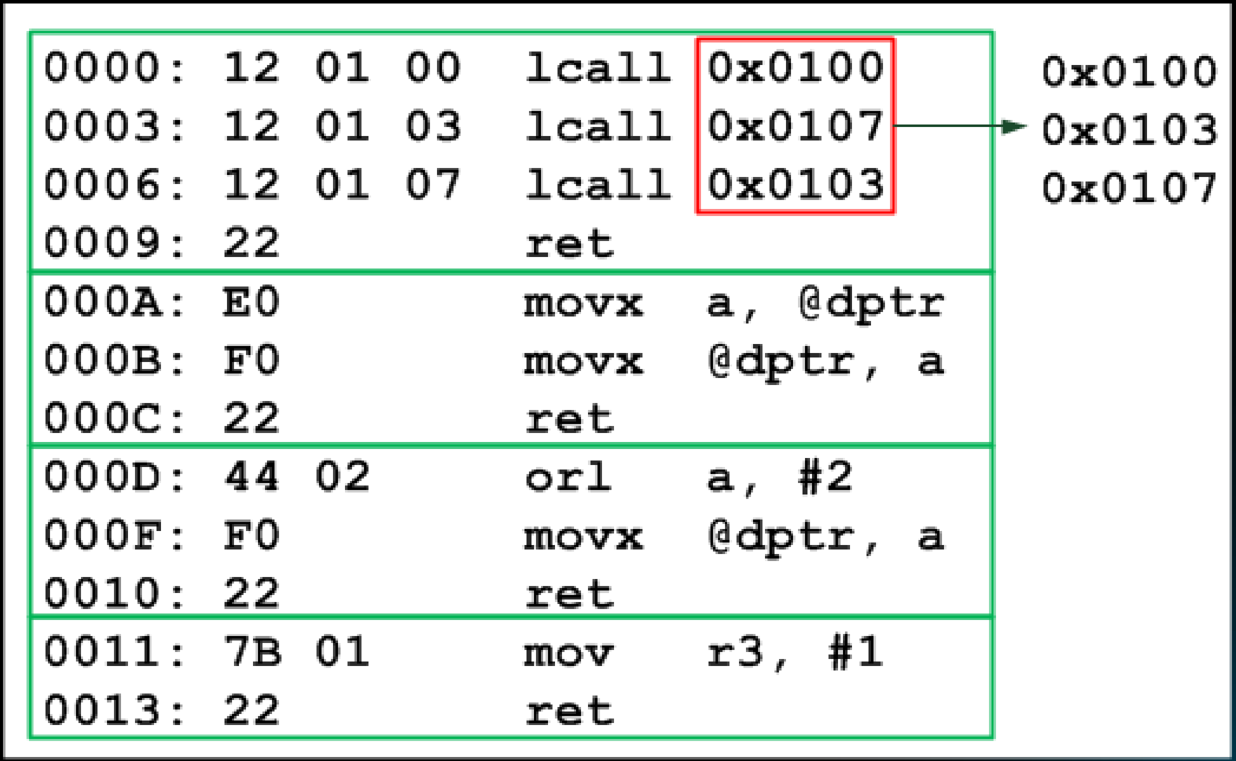
```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Analyzing the FW binary



```
0000: 12 01 00   lcall 0x0100
0003: 12 01 03   lcall 0x0107
0006: 12 01 07   lcall 0x0103
0009: 22         ret
000A: E0         movx  a, @dptr
000B: F0         movx  @dptr, a
000C: 22         ret
000D: 44 02      orl   a, #2
000F: F0         movx  @dptr, a
0010: 22         ret
0011: 7B 01      mov   r3, #1
0013: 22         ret
```

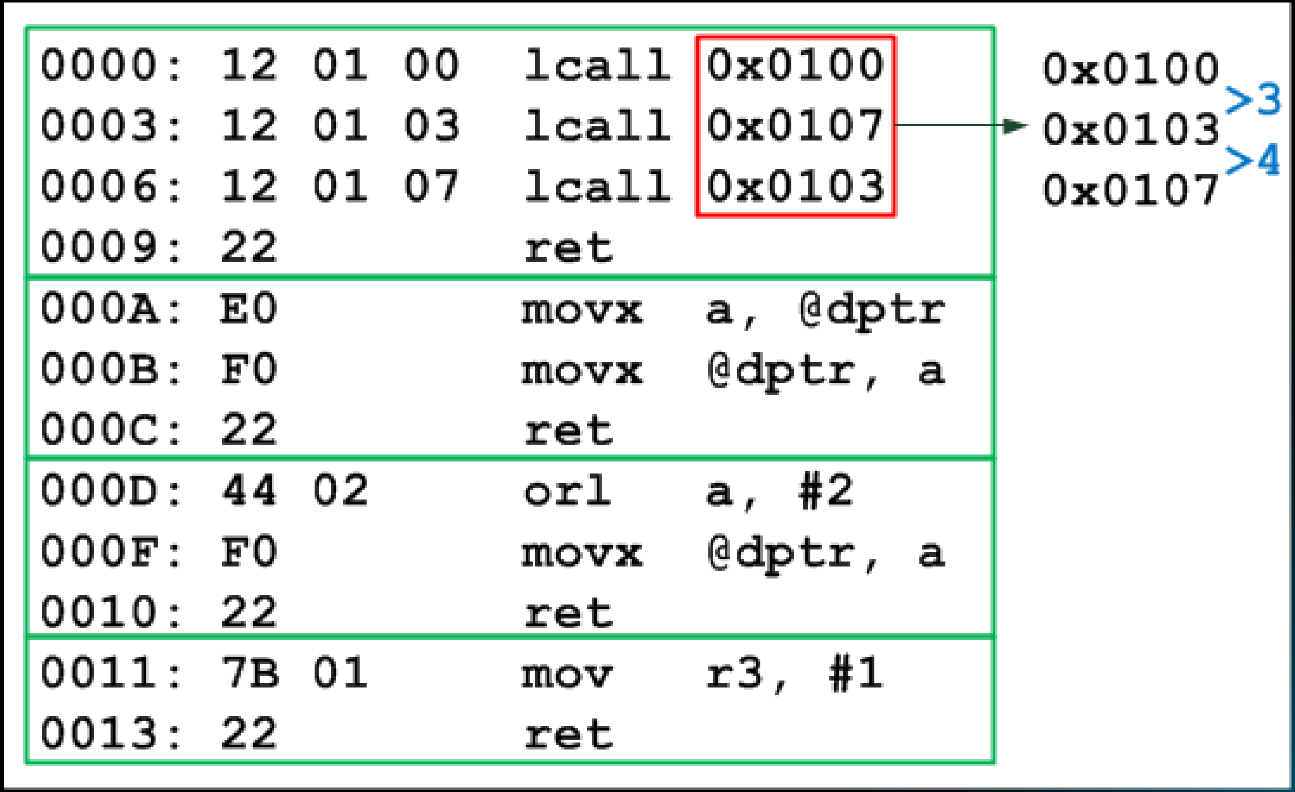
Analyzing the FW binary



```
0000: 12 01 00   lcall 0x0100
0003: 12 01 03   lcall 0x0107
0006: 12 01 07   lcall 0x0103
0009: 22         ret
000A: E0         movx  a, @dptr
000B: F0         movx  @dptr, a
000C: 22         ret
000D: 44 02      orl   a, #2
000F: F0         movx  @dptr, a
0010: 22         ret
0011: 7B 01      mov   r3, #1
0013: 22         ret
```

```
0x0100
0x0103
0x0107
```

Analyzing the FW binary

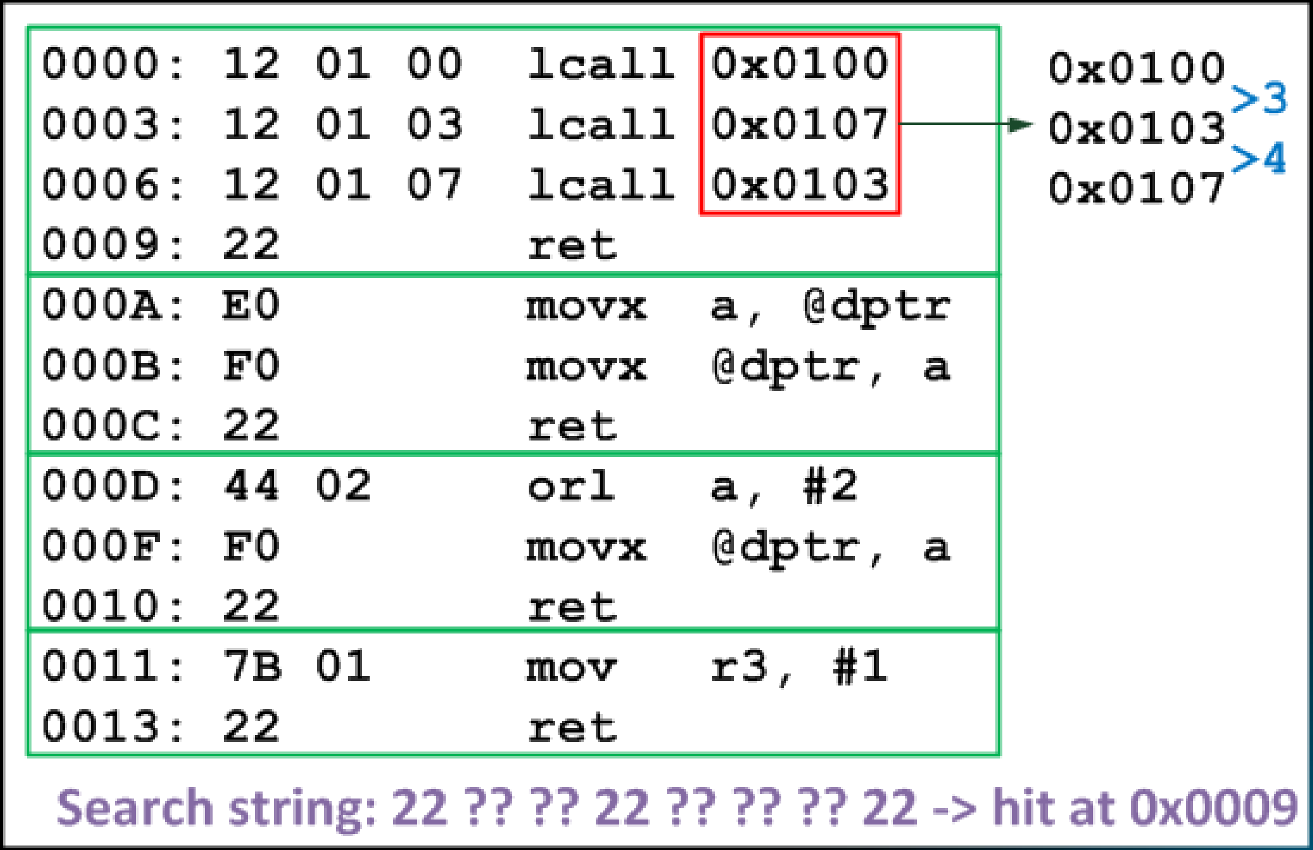


```
0000: 12 01 00   lcall 0x0100
0003: 12 01 03   lcall 0x0107
0006: 12 01 07   lcall 0x0103
0009: 22         ret
000A: E0         movx  a, @dptr
000B: F0         movx  @dptr, a
000C: 22         ret
000D: 44 02      orl   a, #2
000F: F0         movx  @dptr, a
0010: 22         ret
0011: 7B 01      mov   r3, #1
0013: 22         ret
```

```
0x0100
       >3
0x0103
       >4
0x0107
```

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Analyzing the FW binary

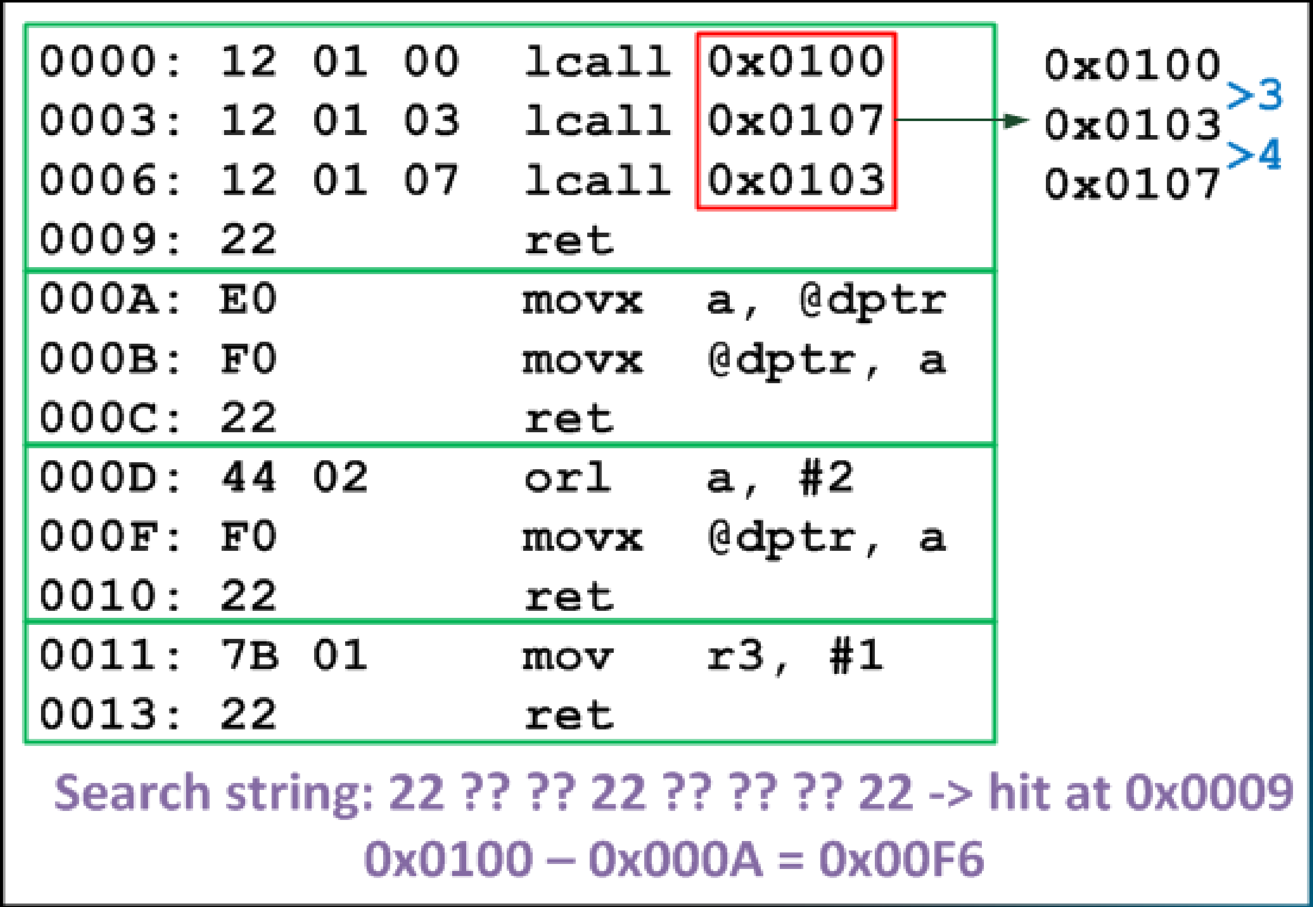


```
0000: 12 01 00   lcall 0x0100
0003: 12 01 03   lcall 0x0107
0006: 12 01 07   lcall 0x0103
0009: 22         ret
000A: E0         movx  a, @dptr
000B: F0         movx  @dptr, a
000C: 22         ret
000D: 44 02      orl   a, #2
000F: F0         movx  @dptr, a
0010: 22         ret
0011: 7B 01      mov   r3, #1
0013: 22         ret
```

```
0x0100
      >3
0x0103
      >4
0x0107
```

Search string: 22 ?? ?? 22 ?? ?? ?? 22 -> hit at 0x0009

Analyzing the FW binary



```
0000: 12 01 00   lcall 0x0100
0003: 12 01 03   lcall 0x0107
0006: 12 01 07   lcall 0x0103
0009: 22         ret
000A: E0         movx  a, @dptr
000B: F0         movx  @dptr, a
000C: 22         ret
000D: 44 02      orl   a, #2
000F: F0         movx  @dptr, a
0010: 22         ret
0011: 7B 01      mov   r3, #1
0013: 22         ret
```

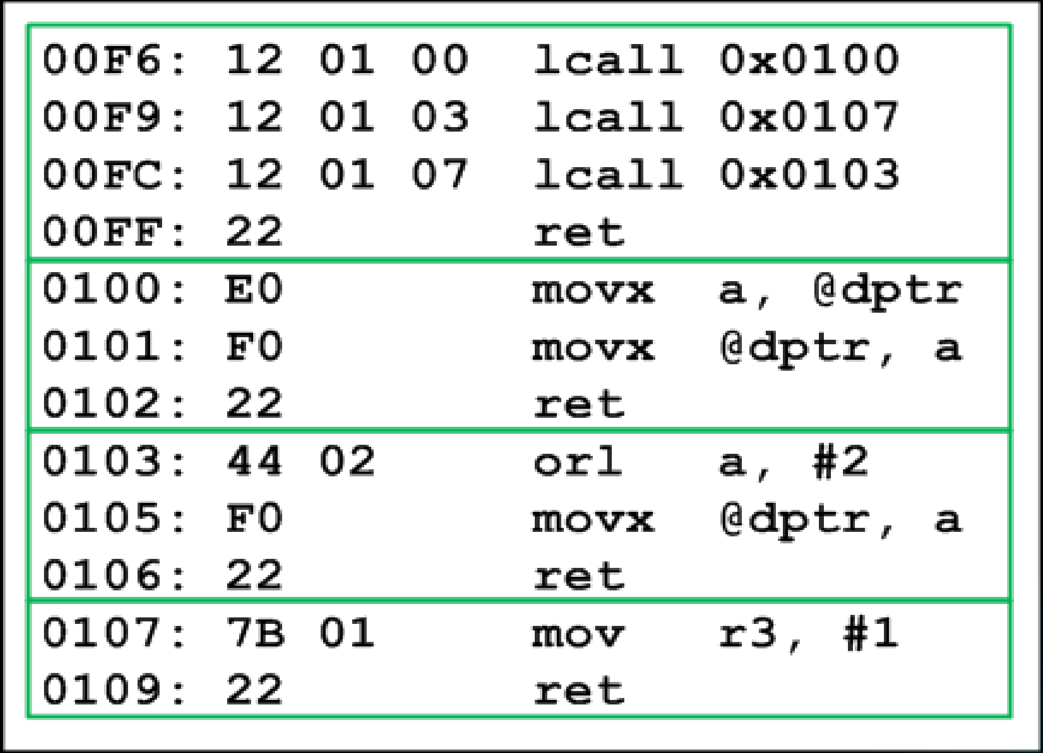
```
0x0100
       >3
0x0103
       >4
0x0107
```

Search string: 22 ?? ?? 22 ?? ?? ?? 22 -> hit at 0x0009

0x0100 – 0x000A = 0x00F6

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

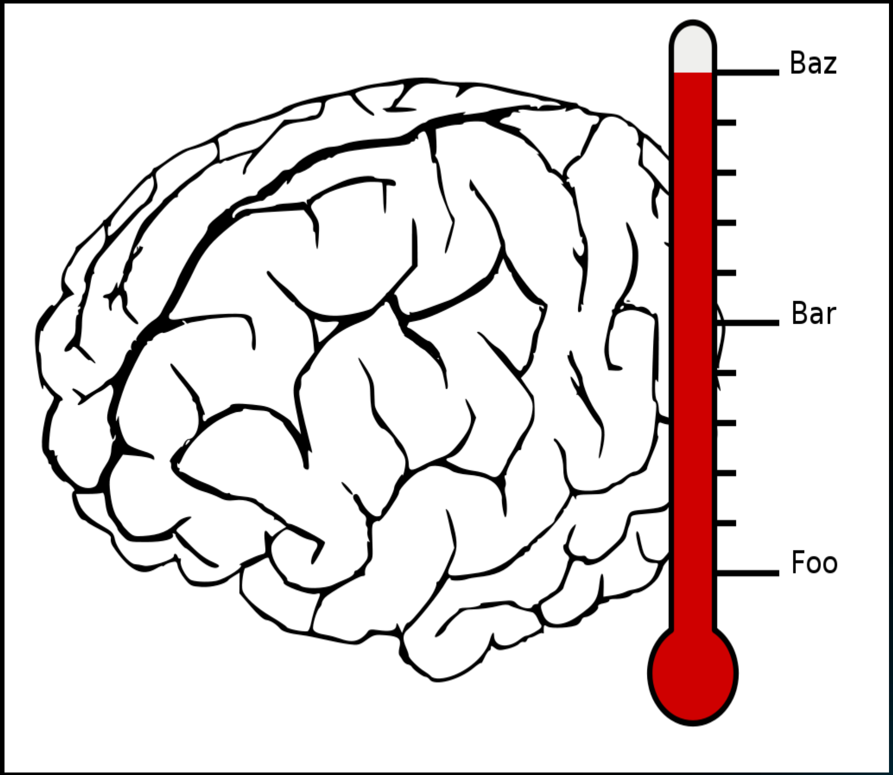
Analyzing the FW binary



```
00F6: 12 01 00   lcall 0x0100
00F9: 12 01 03   lcall 0x0107
00FC: 12 01 07   lcall 0x0103
00FF: 22         ret
0100: E0         movx  a, @dptr
0101: F0         movx  @dptr, a
0102: 22         ret
0103: 44 02      orl   a, #2
0105: F0         movx  @dptr, a
0106: 22         ret
0107: 7B 01      mov   r3, #1
0109: 22         ret
```

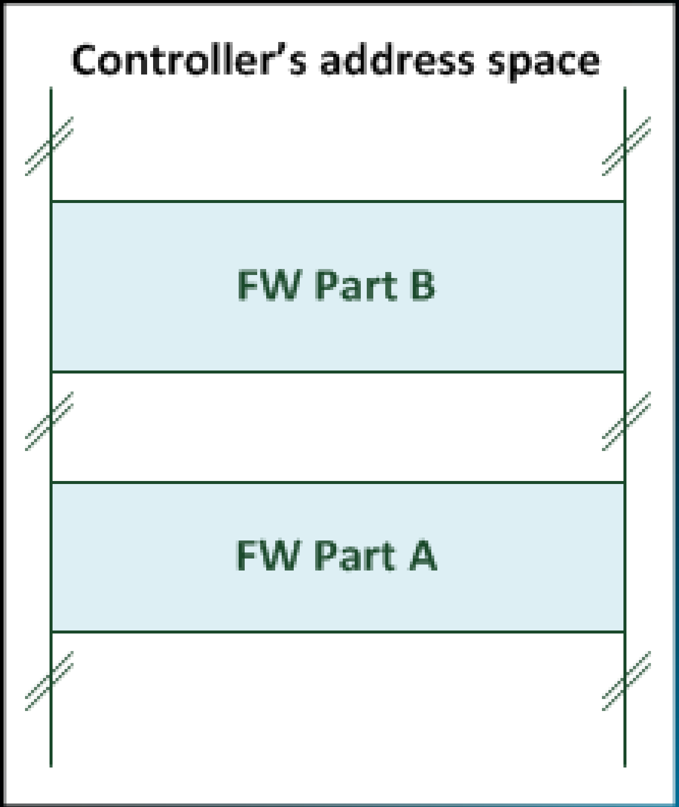
```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Analyzing the FW binary



Analyzing the FW binary

- Question: Is there additional FW?
- Jumps and calls to destinations outside the FW?
- e.g. chip-internal?
- See chapter „Modification“



Analyzing the FW binary

- Starting reverse engineering of the code
- Search for strings and references to the strings
- Search for very specific data references / operands
  - USB descriptor fields (lsusb -v...)
  - USB magics („USBC“ and „USBS“)
  - IDE / SATA / ATAPI ID strings
  - Typical communicated data blocks
  - Error codes (either strings or in code)

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Analyzing the FW binary

- Very interesting: Finding FW update sequences
- Allows non-invasive modifications
- e.g. chip erase and programming commands

Modifying the FW binary

Now we've learned a lot about our device and its FW...

Ready to modify it?

```
MOV    A,R0
SUBB   A,#7
MOV    R1,A
MOV    A,R2
SUBB   A,#0
MOV    R3,A
MOV    A,R1
MUL    AB
ADD    A,DPL
MOV    R1,A
MOV    A,B
ADDC   A,DPH
MOV    R3,A
RET
MOVX   A,@R0
MOV    DPH,A
INC    R0
MOVX   A,@R0
MOV    DPL,A
INC    R0
MOVX
A,@DPTR
POP    DPL
POP    DPH
MOVX
A,@DPTR
DEC    A
SUBB   A,R1
XCH    A,R0
XCH    A,R1
XCH    A,R3
XCH    A,R2
XCH    A,R3
RET
MOV    A,R7
MOVX
@DPTR,A
INC    DPTR
MOV    A,R6
MOVX
@DPTR,A
RET
SETB   RS0
MOV    P2,R2
MOVX   A,@R0
CLR    RS0
RET
```

Modifying the FW binary

- Be prepared to brick your device
- Integrity checks
  - SW based checksum calculation
  - HW based checksum calculation
  - Combination of both = more than one checksum
- Correct checksums
- Patch checksum algorithms

Modifying the FW binary

- Goals:
  - Correcting errors
  - Dumping additional memory regions
    - Find or implement memcpy routine
    - Write memory contents to output (buffers)
  - Gather more device internal information

Modifying the FW binary

- Inject the modified FW into device
  - Using the original updater (checksum check?)
  - Re-programming memory device / processor
  - Via serial interface (JTAG, proprietary)

The end

That's it for now...

```
mov ax, 0x10D0
xor ax, 0x2013
ax = ???
```

Links

- Hex-Rays IDA 5.0 Freeware version

  https://www.hex-rays.com/products/ida/support/download_freeware.shtml
- OnlineDisAssembler

  www.onlinedisassembler.com
- Process Monitor

  http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx

List of references

- Pictures on slides x to x are taken from www.onlinedisassembler.com screen output
- Cliparts are taken from www.openclipart.org