
# Reverse Engineering x86 Processor Microcode

**Vancouver, Canada**, August 18, 2017

Philipp Koppe, Benjamin Kollenda, Marc Fyrbiak, Christian Kison,
Robert Gawlik, Christof Paar, Thorsten Holz

Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum

# x86 CPUs are prone to errors

# x86 CPUs are prone to errors


# x86 CPUs are prone to errors



# x86 CPUs are prone to errors




# x86 ISA is complex

| Hex | Mnemonics |
|---|---|
| C3 | ret |

# x86 ISA is complex

| Hex | Mnemonics |
|---|---|
| C3 | ret |
| 48 b8 88 77 66 55<br>44 33 22 11 | movabs rax,0x1122334455667788 |

# x86 ISA is complex

| Hex | Mnemonics |
|---|---|
| C3 | ret |
| 48 b8 88 77 66 55<br>44 33 22 11 | movabs rax,0x1122334455667788 |
| 64 ff 03 | DWORD PTR fs:[ebx] |

# x86 ISA is complex

| Hex | Mnemonics |
|---|---|
| C3 | ret |
| 48 b8 88 77 66 55<br>44 33 22 11 | movabs rax,0x1122334455667788 |
| 64 ff 03 | DWORD PTR fs:[ebx] |
| 64 67 66 f0 ff 07 | lock inc WORD PTR fs:[bx] |

# x86 ISA is complex

| Hex | Mnemonics |
|---|---|
| C3 | ret |
| 48 b8 88 77 66 55<br>44 33 22 11 | movabs rax,0x1122334455667788 |
| 64 ff 03 | DWORD PTR fs:[ebx] |
| 64 67 66 f0 ff 07 | lock inc WORD PTR fs:[bx] |
| 2e c4 e2 71 96 84<br>be 34 23 12 01 | vfmaddsub132ps xmm0, xmm1,<br>xmmword ptr cs:<br>[esi + edi * 4 + 0x11223344] |

# Micro Ops

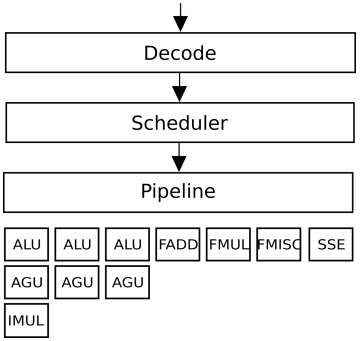
```
pop [ebx]
```

# Micro Ops

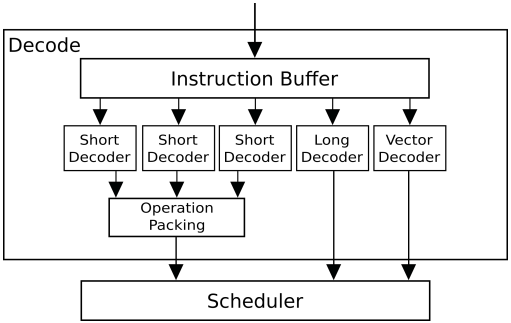
```
pop [ebx]
```

```
load temp, [esp]
store [ebx], temp
add esp, 4
```

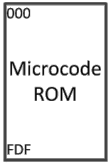
# x86 Instruction Decoding



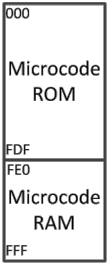
# x86 Instruction Decoding



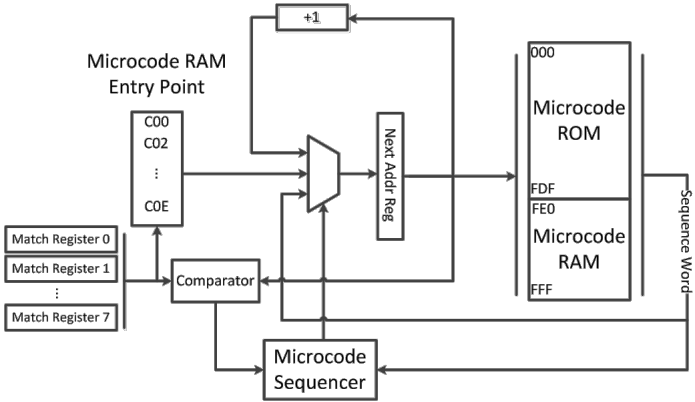
# Microcode Engine (Vector Decoder)



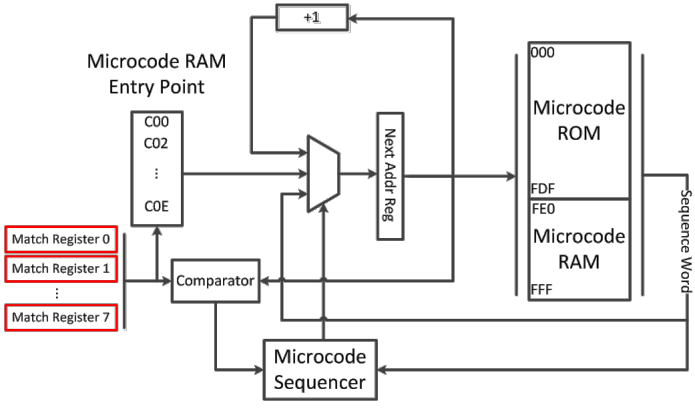
# Microcode Engine (Vector Decoder)



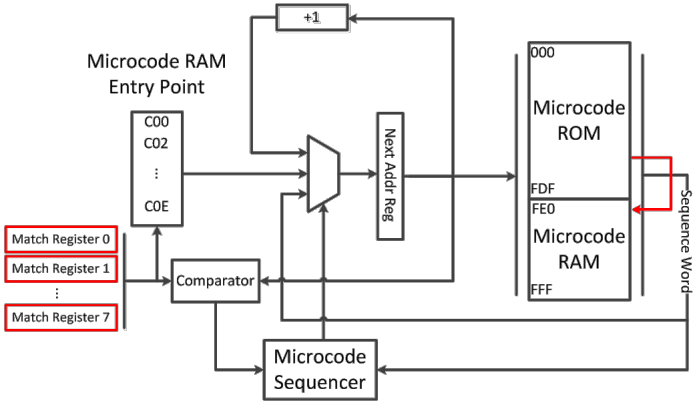
# Microcode Engine (Vector Decoder)



# Microcode Engine (Vector Decoder)



# Microcode Engine (Vector Decoder)



## Research Questions

- How does the microcode update mechanism work?

## Research Questions

- How does the microcode update mechanism work?
- How can we analyze the microcode encoding and meaning?

## Research Questions

- How does the microcode update mechanism work?
- How can we analyze the microcode encoding and meaning?
- Can we load our own microprograms into the CPU?

## Research Questions

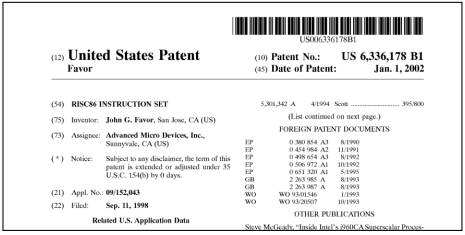
- How does the microcode update mechanism work?
- How can we analyze the microcode encoding and meaning?
- Can we load our own microprograms into the CPU?
- Are there security implications?

## Research Questions

- How does the microcode update mechanism work?
- How can we analyze the microcode encoding and meaning?
- Can we load our own microprograms into the CPU?
- Are there security implications?

- AMD K8 and K10 processor families

# Related Work



US006336178B1

(12) **United States Patent**
**Favor**

(10) **Patent No.: US 6,336,178 B1**
(45) **Date of Patent: Jan. 1, 2002**

(54) **RISC86 INSTRUCTION SET**

(75) Inventor: **John G. Favor**, San Jose, CA (US)

(73) Assignee: **Advanced Micro Devices, Inc.**, Sunnyvale, CA (US)

( * ) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/152,043**

(22) Filed: **Sep. 11, 1998**

**Related U.S. Application Data**

5,301,342 A 4/1994 Scott ..... 395/800

(List continued on next page.)

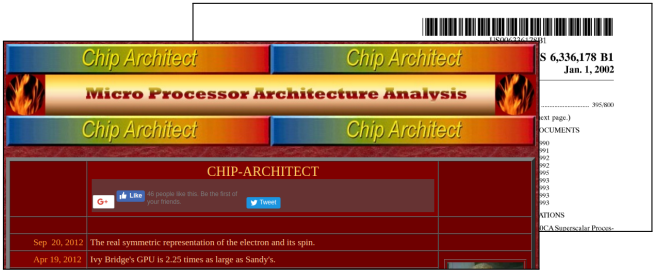
FOREIGN PATENT DOCUMENTS

| | | |
|---|---|---|
| EP | 0 380 854 A3 | 8/1990 |
| EP | 0 454 984 A2 | 11/1991 |
| EP | 0 498 654 A3 | 8/1992 |
| EP | 0 506 972 A1 | 10/1992 |
| EP | 0 651 320 A1 | 5/1995 |
| GB | 2 263 985 A | 8/1993 |
| GB | 2 263 987 A | 8/1993 |
| WO | WO 93/01546 | 1/1993 |
| WO | WO 93/20507 | 10/1993 |

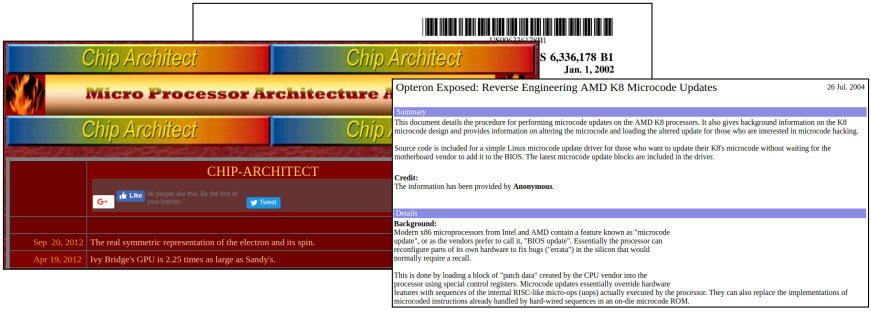
OTHER PUBLICATIONS

Steve McGeady, "Inside Intel's i960CA Superscalar Proces-

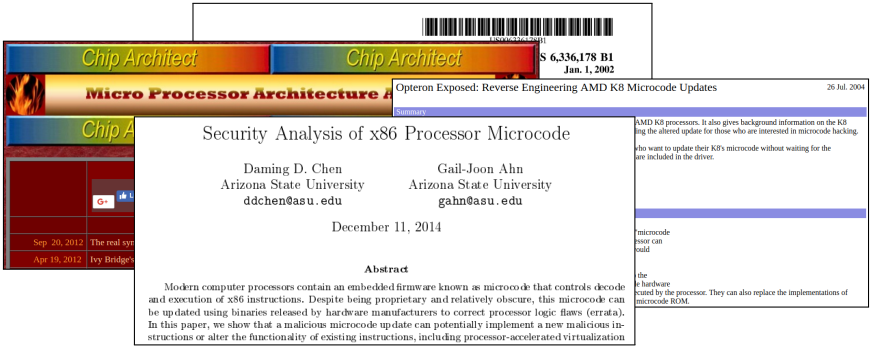
# Related Work



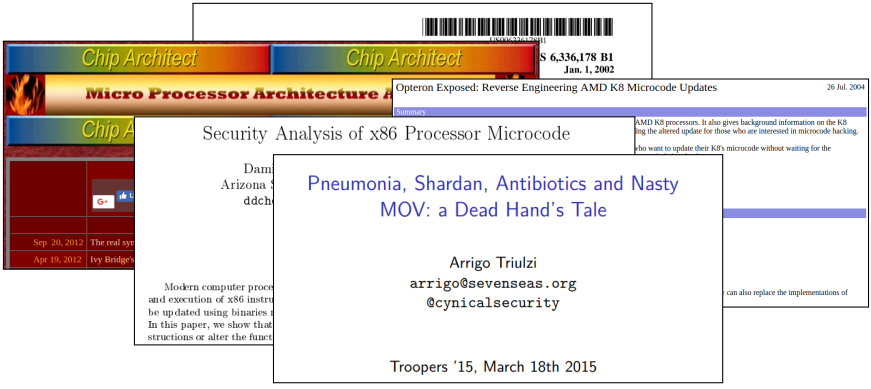
# Related Work



# Related Work



# Related Work



# Microcode Update Mechanism

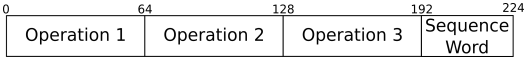
- Kernel mode
- Load microcode update into RAM
- Write virtual address to MSR `0xC0010020`
- Microcode patches not persistent

# Microcode Update File Format

| B↓ Bit→ | 0 31 | 32 63 |
|---|---|---|
| 0 | date | patch ID |
| 8 | patch block \| len \| init | checksum |
| 16 | northbridge ID | southbridge ID |
| 24 | CPUID | magic value |
| 32 | match register 0 | match register 1 |
| 40 | match register 2 | match register 3 |
| 48 | match register 4 | match register 5 |
| 54 | match register 6 | match register 7 |
| 64 | triad 0, microinstruction 0 | |
| 72 | triad 0, microinstruction 1 | |
| 80 | triad 0, microinstruction 2 | |
| 88 | triad 0, sequence word | triad 1 ... |

# Microcode Update File Format

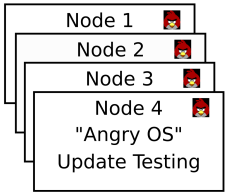
| B↓ Bit→ | 0 31 | 32 63 |
|---|---|---|
| 0 | date | patch ID |
| 8 | patch block \| len \| init | checksum |
| 16 | northbridge ID | southbridge ID |
| 24 | CPUID | magic value |
| 32 | match register 0 | match register 1 |
| 40 | match register 2 | match register 3 |
| 48 | match register 4 | match register 5 |
| 54 | match register 6 | match register 7 |
| 64 | triad 0, microinstruction 0 | |
| 72 | triad 0, microinstruction 1 | |
| 80 | triad 0, microinstruction 2 | |
| 88 | triad 0, sequence word | triad 1 ... |



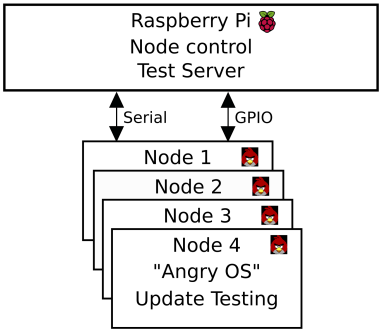
## Reverse Engineering Setting

- Unknown instruction set analysis
- Black box model with oracle
- Feed inputs, filter and observe outputs
- Infer structure, encoding, meaning

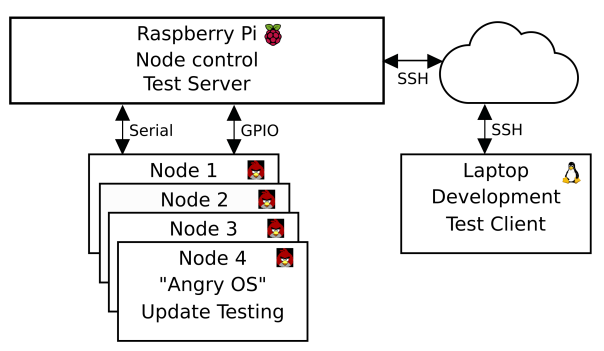
# Framework



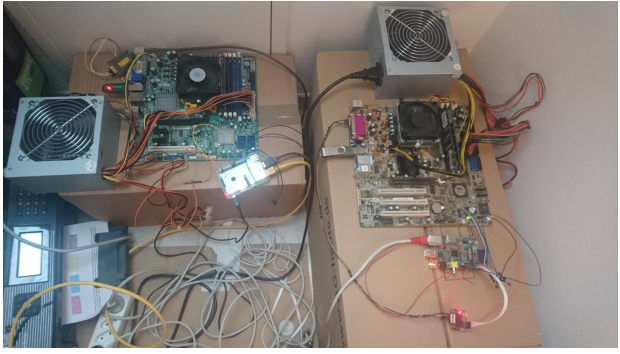
# Framework


Raspberry Pi
Node control
Test Server
Serial
GPIO
Node 1
Node 2
Node 3
Node 4
"Angry OS"
Update Testing

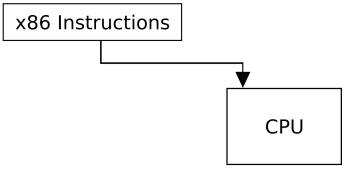
# Framework


Raspberry Pi
Node control
Test Server
SSH
Serial
GPIO
SSH
Node 1
Node 2
Node 3
Node 4
"Angry OS"
Update Testing
Laptop
Development
Test Client

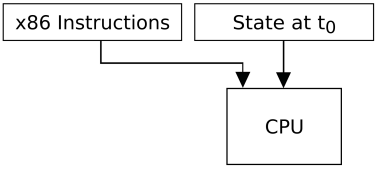
# Framework



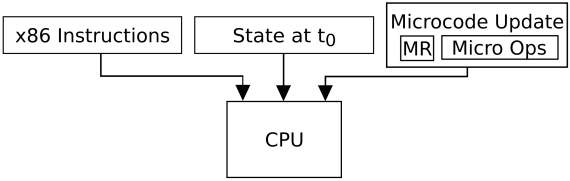
# Processor Oracle



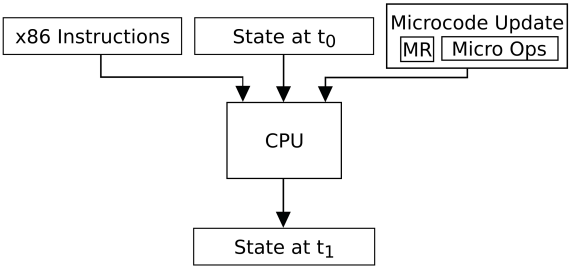
# Processor Oracle



# Processor Oracle



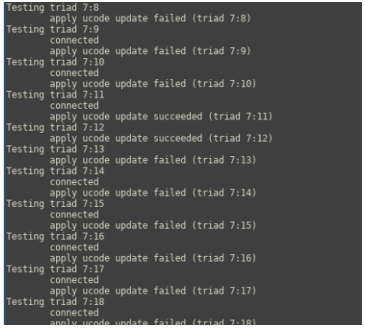
# Processor Oracle



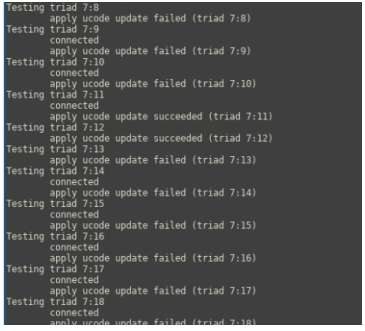
# Analysis - Heatmaps

| ROM Address | vector instruction |
|---|---|
| 0x900 - 0x913 | - |
| 0x900 - 0x913 | - |
| 0x914 - 0x917 | rep_cmps_mem8 |
| 0x918 - 0x95f | - |
| 0x960 | mul_mem16 |
| 0x961 | idiv |
| 0x962 | mul_reg16 |
| 0x963 | - |
| 0x964 | imul_mem16 |
| 0x965 | bound |
| 0x966 | imul_reg16 |
| 0x967 | - |
| 0x968 | bts_imm |
| 0x969 - 0x971 | - |
| 0x972 - 0x973 | div |
| 0x974 - 0x975 | - |
| 0x976 - 0x977 | idiv |
| 0x978 | - |
| 0x979 - 0x97a | idiv |
| 0x97b - 0x9a7 | - |
| 0x9a8 | btr_imm |
| 0x9a9 - 0x9ad | - |
| 0x9ae | mfence |
| 0x9af - 09ff | - |

# Analysis - Brute Force



# Analysis - Brute Force

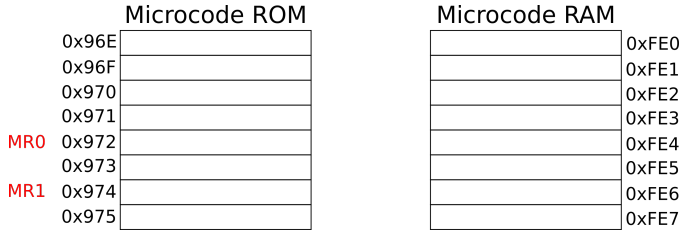



```
add eax, imm16
```

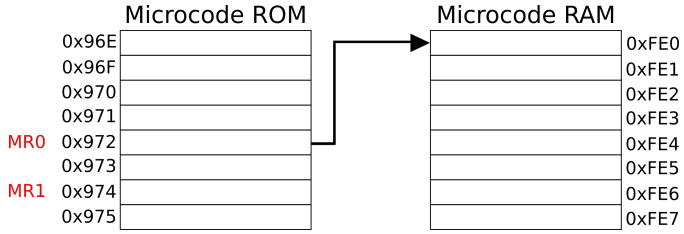
# Analysis - Automated Tests

```
6004  Testing (insn:offset:b1:b2 DIV:0x972:799:0)
6005  Current test:
6006  MOV ECX, EAX
6007  0 001100000 00 000001 0001100011111 011 000000 1 0000000 0000000000001000
6008  ? OOOOOOOOO ?? DDDDDD ???????????? ZZZ ?????? R ??????? IIIIIIIIIIIIIIII
6009      apply ucode update succeeded t1:t2 5315:1686
6010      eax deadbeef ebx 00000101 ecx deadbeef edx 00000103
6011      esi 00000104 edi 00000105 ebp 00000106 esp 0014579c
6012      efl 00000082
6013  Testing (insn:offset:b1:b2 DIV:0x972:800:0)
6014  Current test:
6015  MOV ECX, EAX
6016  0 001100000 00 000001 0001100100000 011 000000 1 0000000 0000000000001000
6017  ? OOOOOOOOO ?? DDDDDD ???????????? ZZZ ?????? R ??????? IIIIIIIIIIIIIIII
6018      apply ucode update failed (insn:offset:b1:b2 DIV:0x972:800:0)
6019  Testing (insn:offset:b1:b2 DIV:0x972:801:0)
6020      connected
6021  Current test:
6022  MOV ECX, EAX
6023  0 001100000 00 000001 0001100100001 011 000000 1 0000000 0000000000001000
6024  ? OOOOOOOOO ?? DDDDDD ???????????? ZZZ ?????? R ??????? IIIIIIIIIIIIIIII
6025      apply ucode update failed (insn:offset:b1:b2 DIV:0x972:801:0)
6026  Testing (insn:offset:b1:b2 DIV:0x972:802:0)
6027      connected
6028  Current test:
6029  MOV ECX, EAX
6030  0 001100000 00 000001 0001100100010 011 000000 1 0000000 0000000000001000
6031  ? OOOOOOOOO ?? DDDDDD ???????????? ZZZ ?????? R ??????? IIIIIIIIIIIIIIII
6032      apply ucode update failed (insn:offset:b1:b2 DIV:0x972:802:0)
6033  Testing (insn:offset:b1:b2 DIV:0x972:803:0)
```

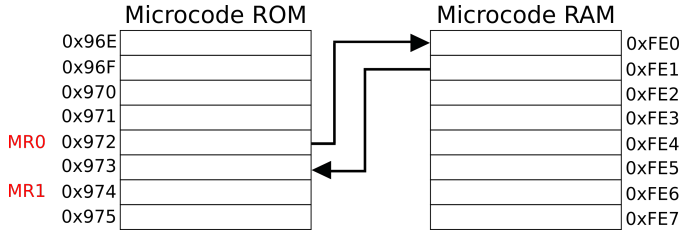
# Analysis - Infer Logic of ROM Triads



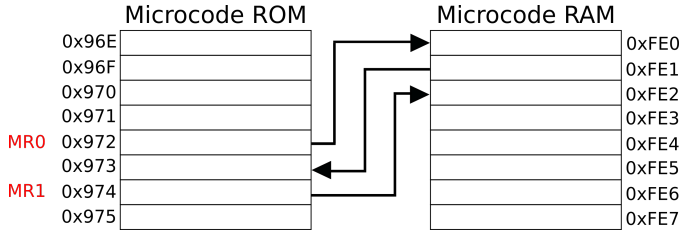
# Analysis - Infer Logic of ROM Triads



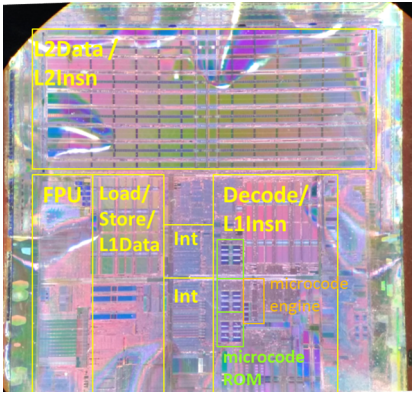
# Analysis - Infer Logic of ROM Triads



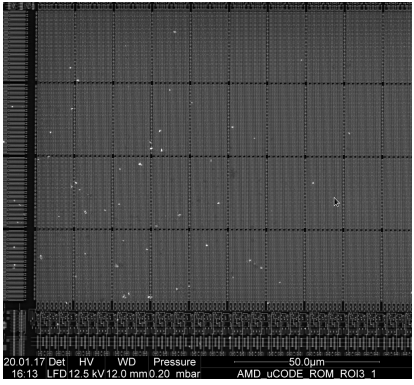
# Analysis - Infer Logic of ROM Triads



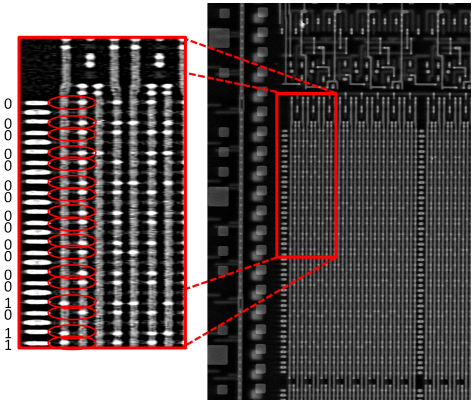
# Analysis - Hardware



# Analysis - Hardware



# Analysis - Hardware



# Results - Micro Ops

- Heatmaps

## Results - Micro Ops

- Heatmaps
- 29 Micro Ops
  - Logic, arithmetic, load, store
  - Write x86 program counter
  - Conditional microcode branch

## Results - Micro Ops

- Heatmaps
- 29 Micro Ops
  - Logic, arithmetic, load, store
  - Write x86 program counter
  - Conditional microcode branch
- Sequence word
  - Next triad, sequence complete, unconditional branch

## Results - Micro Ops

- Heatmaps
- 29 Micro Ops
  - Logic, arithmetic, load, store
  - Write x86 program counter
  - Conditional microcode branch
- Sequence word
  - Next triad, sequence complete, unconditional branch
- Substitution engine

# Results - Augment x86 instructions

- Jump back to ROM
  - `DIV`
- Emulate instruction logic
  - `IMUL, SHRD, CMPXCHG, ENTER`

# Our Microprograms

- Instrumentation

# Our Microprograms

- Instrumentation
- Remote microcode backdoors
  - Control flow hijack in browsers induced by microcode
  - Triggered remotely with ASM.JS or WebAssembly

# Our Microprograms

- Instrumentation
- Remote microcode backdoors
  - Control flow hijack in browsers induced by microcode
  - Triggered remotely with ASM.JS or WebAssembly
- Cryptographic microcode Trojans
  - Introduce timing side-channels in constant-time ECC implementation
  - Inject faults to enable fault attacks

# Sample Microprogram (simplified)

```
sub.Z t1d, eax
jcc EZF, 0x2
or t12d, eax, 0x8
```

# Sample Microprogram (simplified)

```
sub.Z t1d, eax
jcc EZF, 0x2
or t12d, eax, 0x8

div2 t15q, t24q, 0xd5
srl t13w, ax, 0x8
div1.C t19d, t12d, t56d
```

# Sample Microprogram (simplified)

```
sub.Z t1d, eax
jcc EZF, 0x2
or t12d, eax, 0x8

div2 t15q, t24q, 0xd5
srl t13w, ax, 0x8
div1.C t19d, t12d, t56d

mov t9d, t9d, regmd4
add.EP t56d, edx, t56d
jcc True, -0x800
```

# Sample Microprogram (simplified)

```
sub.Z t1d, eax
jcc EZF, 0x2
or t12d, eax, 0x8

div2 t15q, t24q, 0xd5
srl t13w, ax, 0x8
div1.C t19d, t12d, t56d

mov t9d, t9d, regmd4
add.EP t56d, edx, t56d
jcc True, -0x800

mov eax, eax
add t1d, pcd, 1
writePC t1d
```

## Demo

# DEMO

## Summary

- We built a framework for microcode reverse engineering
- We reverse engineered substantial parts of the encoding
- We implemented meaningful microprograms from scratch

`https://github.com/RUB-SysSec/microcode/`