CustomProcessingUnit: Reverse Engineering and Customization of Intel Microcode

WOOT 2023

Pietro Borrello
Sapienza University of Rome

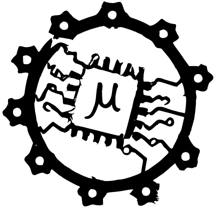
Catherine Easdon
Dynatrace Research & Graz University of Technology

Martin Schwarzl
Graz University of Technology

Roland Czerny
Graz University of Technology

Michael Schwarz
CISPA Helmholtz Center for Information Security

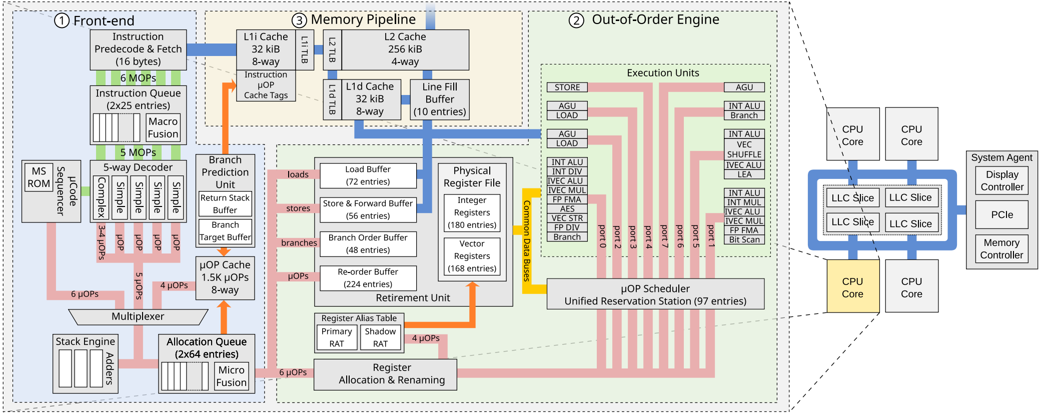
CustomProcessingUnit



The first CPU µcode Software Framework

- µcode Static analysis
- µcode Dynamic analysis

How do CPUs work?



Pietro Borrello (🐦@borrello_pietro)

Mark Ermolov, Maxim Goryachy & Dmitry Sklyarov Results

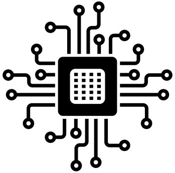
- Red Unlock of Atom Goldmont (GLM) CPUs
- Extraction and reverse engineering of GLM µcode format
- Discovery of undocumented control instructions to access internal buffers

udbgrd and udbgwr

Two secret instructions that can access:

- System agent
- URAM
- Staging buffer
- I/O ports
- Power supply unit
- CRBUS

Control Registers Bus

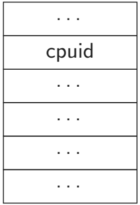


CPU interacts with its internal components through the `CRBUS`

- MSRs → CRBUS addr
- Control and Status registers
- Post Silicon Validation features

What can you do with access to microarchitectural buffers?

Microcoded Instuctions 101



...

cpuid

...

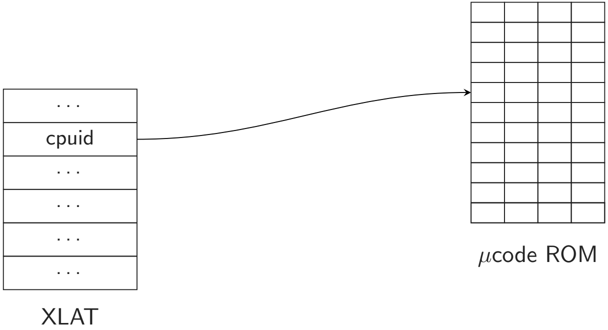
...

...

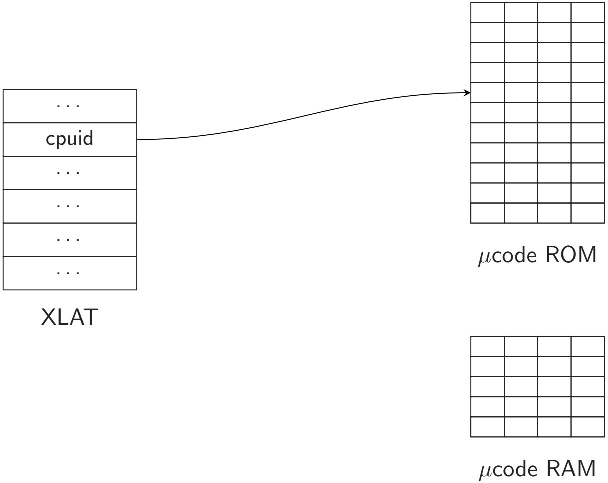
...

XLAT

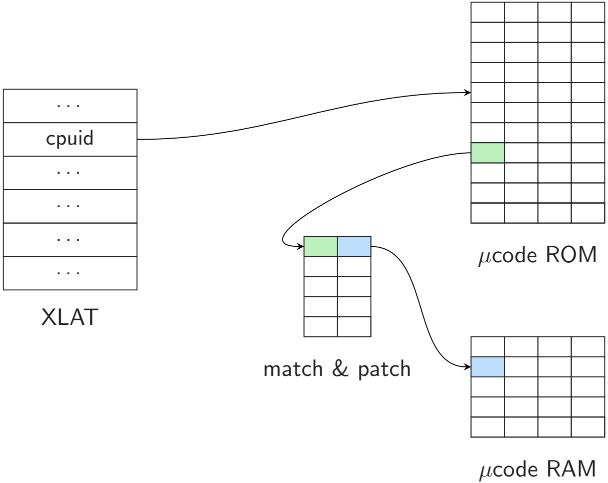
Microcoded Instuctions 101



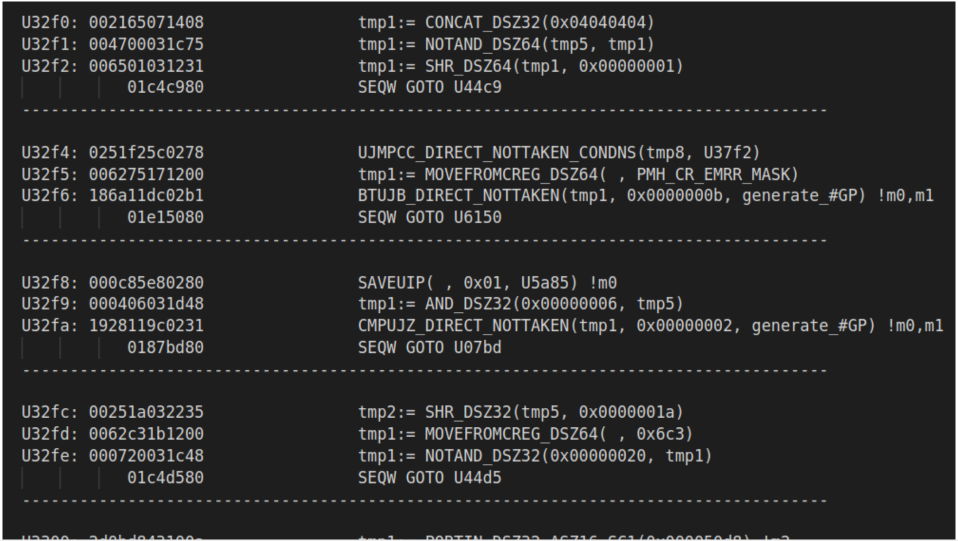
Microcoded Instuctions 101



Microcoded Instuctions 101

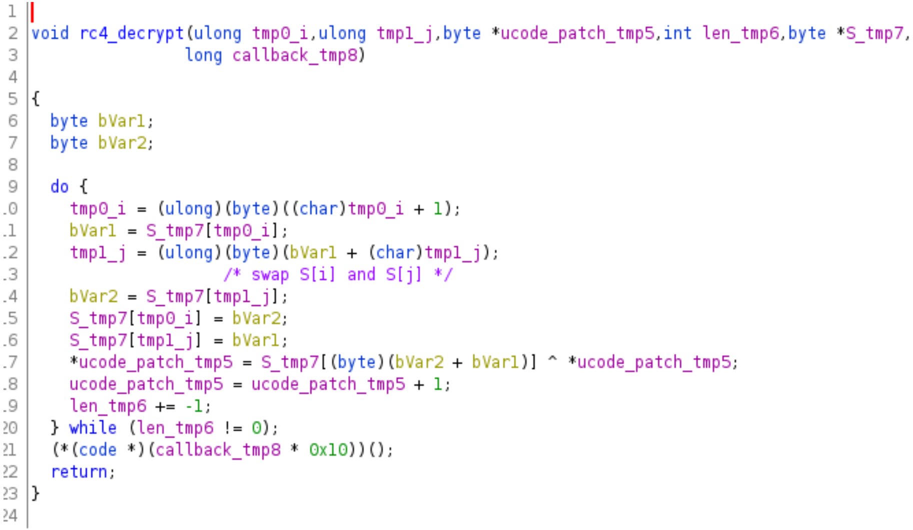


Building a Ghidra µcode Decompiler



```
U32f0: 002165071408              tmp1:= CONCAT_DSZ32(0x04040404)
U32f1: 004700031c75              tmp1:= NOTAND_DSZ64(tmp5, tmp1)
U32f2: 006501031231              tmp1:= SHR_DSZ64(tmp1, 0x00000001)
|   |   |   01c4c980             SEQW GOTO U44c9
--------------------------------------------------------------------------------------

U32f4: 0251f25c0278              UJMPCC_DIRECT_NOTTAKEN_CONDNS(tmp8, U37f2)
U32f5: 006275171200              tmp1:= MOVEFROMCREG_DSZ64( , PMH_CR_EMRR_MASK)
U32f6: 186a11dc02b1              BTUJB_DIRECT_NOTTAKEN(tmp1, 0x0000000b, generate_#GP) !m0,m1
|   |   |   01e15080             SEQW GOTO U6150
--------------------------------------------------------------------------------------

U32f8: 000c85e80280              SAVEUIP( , 0x01, U5a85) !m0
U32f9: 000406031d48              tmp1:= AND_DSZ32(0x00000006, tmp5)
U32fa: 1928119c0231              CMPUJZ_DIRECT_NOTTAKEN(tmp1, 0x00000002, generate_#GP) !m0,m1
|   |   |   0187bd80             SEQW GOTO U07bd
--------------------------------------------------------------------------------------

U32fc: 00251a032235              tmp2:= SHR_DSZ32(tmp5, 0x0000001a)
U32fd: 0062c31b1200              tmp1:= MOVEFROMCREG_DSZ64( , 0x6c3)
U32fe: 000720031c48              tmp1:= NOTAND_DSZ32(0x00000020, tmp1)
|   |   |   01c4d580             SEQW GOTO U44d5
--------------------------------------------------------------------------------------

U3300: 2d0bd843100a              tmp1:= PORTIN_DSZ32_ASZ16_SC1(0x000050d8) !m2
```

Building a Ghidra µcode Decompiler



```
void rc4_decrypt(ulong tmp0_i,ulong tmp1_j,byte *ucode_patch_tmp5,int len_tmp6,byte *S_tmp7,
                long callback_tmp8)

{
  byte bVar1;
  byte bVar2;

  do {
    tmp0_i = (ulong)(byte)((char)tmp0_i + 1);
    bVar1 = S_tmp7[tmp0_i];
    tmp1_j = (ulong)(byte)(bVar1 + (char)tmp1_j);
                    /* swap S[i] and S[j] */
    bVar2 = S_tmp7[tmp1_j];
    S_tmp7[tmp0_i] = bVar2;
    S_tmp7[tmp1_j] = bVar1;
    *ucode_patch_tmp5 = S_tmp7[(byte)(bVar2 + bVar1)] ^ *ucode_patch_tmp5;
    ucode_patch_tmp5 = ucode_patch_tmp5 + 1;
    len_tmp6 += -1;
  } while (len_tmp6 != 0);
  (*(code *)(callback_tmp8 * 0x10))();
  return;
}
```

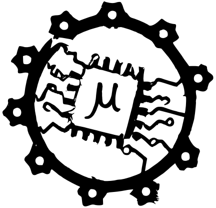
Accessing the μcode

Reverse engineer how the CPU itself updates μcode

- Observe patterns of CRBUS accesses
- Reproduce the same accesses using the undocumented instructions

→ With the undocumented instructions we can control μcode!

The first µcode Framework



Leveraging udbgrd/wr we can patch µcode via software

- Completely observe CPU behavior
- Completely control CPU behavior
- All within a BIOS or kernel module

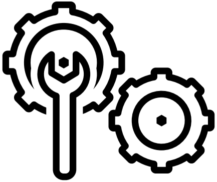
μcode Framework

Patch μcode

Hook μcode

Trace μcode

µcode patches



We can customize the CPU's behavior.

- Change microcoded instructions
- Add functionalities to the CPU

Case Studies

Improve CPU security and performance through µcode customization

- x86 Pointer Authentication Codes
- Fast Breakpoints
- Constant Time Hardware Division

µcode hooks

Install µcode hooks to observe events.

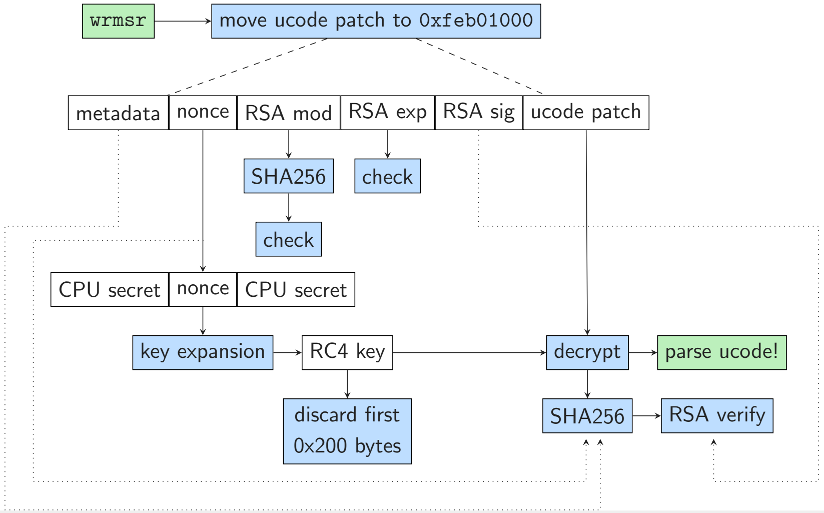
- Setup Match & Patch to execute custom µcode at certain events
- Resume execution

µcode traces

Trace µcode execution leveraging µcode hooks.

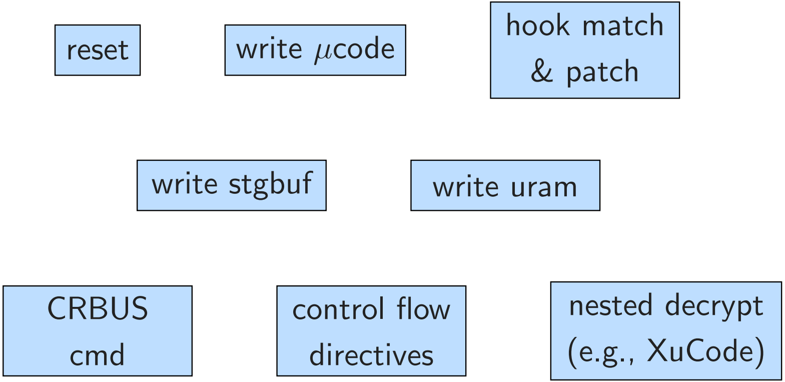
- Setup a hook for every possible µop
- Reconstruct µops executed

GLM µcode update algorithm



Parsing µcode updates

A µcode update is bytecode: the CPU interprets commands from the µcode update



reset

write μcode

hook match & patch

write stgbuf

write uram

CRBUS cmd

control flow directives

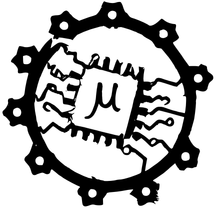
nested decrypt (e.g., XuCode)

µcode decryptor

- Create a parser for µcode updates
- Automatically collect existing µcode (s) for GLM
- Decrypt all GLM updates

`github.com/pietroborrello/CustomProcessingUnit/ucode_collection`

Conclusion



- Deepen understanding of modern CPUs with μcode access
- Develop a static and dynamic analysis framework for μcode:
  - μcode decompiler
  - μcode assembler
  - μcode patcher
  - μcode tracer
- Let's control our CPUs!

`github.com/pietroborrello/CustomProcessingUnit`