Frontal Attack: Leaking Control-Flow in SGX via the CPU Frontend

Ivan Puddu, Moritz Schneider, Miro Haller, Srdjan Čapkun
Intel SGX

- Enclave
  - App 1
  - Untrusted App 1
  - Untrusted App 1 Memory

- Application 2
- Application 3

- Operating System
- Hypervisor
- Hardware (CPU)

TCB
Intel SGX

• How can you extract secrets from an enclave?
Victim code patterns

\[
\text{secret} == 1
\]

- True
  - \text{var1} = 1 + \text{var1}
  - \text{var2} = 1 + \text{var2}
  - return

- False
  - \text{var1} = 2 + \text{var1}
  - \text{var2} = 2 + \text{var2}
  - return
Related attacks

```python
secret == 1

var1 = 1 + var1
var2 = 1 + var2
return

var1 = 2 + var1
var2 = 2 + var2
return
```
Related attacks

• Cache attacks [41,9,10,11]
Related attacks

• Cache attacks [41,9,10,11]
Related attacks

- Cache attacks [41,9,10,11]
Related attacks

• Cache attacks [41,9,10,11]
Related attacks

- Cache attacks [41, 9, 10, 11]
Related attacks

• Cache attacks [41,9,10,11]
Related attacks

- Cache attacks [41,9,10,11]
- BPU attacks [14,19,20]
Related attacks

• Cache attacks [41,9,10,11]
• BPU attacks [14,19,20]
Related attacks

- Cache attacks [41,9,10,11]
- BPU attacks [14,19,20]
Related attacks

• Cache attacks [41,9,10,11]
• BPU attacks [14,19,20]
• Nemesis [21]
Related attacks

• Cache attacks [41,9,10,11]
• BPU attacks [14,19,20]
• Nemesis [21]
Related attacks

• Cache attacks [41,9,10,11]
• BPU attacks [14,19,20]
• Nemesis [21]

var1 = 2 + var1
var2 = 1 + var2
return

var1 = 20 * var1
var2 = 1 + var2
return

var1 = 2 + var1
var2 = 2 + var2
return

secret == 1
Related attacks

• Cache attacks [41,9,10,11]
• BPU attacks [14,19,20]
• Nemesis [21]
Related attacks

- Cache attacks [41, 9, 10, 11]
- BPU attacks [14, 19, 20]
- Nemesis [21]
Related attacks

- Cache attacks [41,9,10,11]
- BPU attacks [14,19,20]
- Nemesis [21]
- CopyCat [23]
Related attacks

- Cache attacks [41,9,10,11]
- BPU attacks [14,19,20]
- Nemesis [21]
- CopyCat [23]
Related attacks

- Cache attacks [41,9,10,11]
- BPU attacks [14,19,20]
- Nemesis [21]
- CopyCat [23]
Frontal Attack

secret == 1

True

var1 = 1 + var1

var2 = 1 + var2

return

False

var1 = 2 + var1

var2 = 2 + var2

return
Frontal Attack

secret == 1

True

var1 = 1 + var1
var2 = 1 + var2
return

False

var1 = 2 + var1
var2 = 2 + var2
return

Interrupt execution
Frontal Attack

secret == 1

True

var1 = 1 + var1

var2 = 1 + var2

return

False

var1 = 2 + var1

var2 = 2 + var2

return
Frontal Attack

secret == 1

- True
  - var1 = 1 + var1
  - var2 = 1 + var2
  - return

- False
  - var1 = 2 + var1
  - var2 = 2 + var2
  - return

Interrupt execution
Frontal Attack

secret == 1

True

var1 = 1 + var1
var2 = 1 + var2
return

False

var1 = 2 + var1
var2 = 2 + var2
return
Frontal Attack

```
var1 = 1 + var1
var2 = 1 + var2
return
```

```
var1 = 2 + var1
var2 = 2 + var2
return
```
Exploited libraries
Exploited libraries

• Intel IPP Cryptography library
Exploited libraries

• Intel IPP Cryptography library
• Big number comparison contains a balanced branch
Exploited libraries

• Intel IPP Cryptography library
  • Big number comparison contains a balanced branch
Exploited libraries

- Intel IPP Cryptography library
- Big number comparison contains a balanced branch
  - Leak comparison results (25% of the time with high confidence)
Exploited libraries
Exploited libraries

• ARM mBedTLS
Exploited libraries

• ARM mBedTLS
  • Montgomery multiplication
Exploited libraries

- ARM mBedTLS
- Montgomery multiplication
Exploited libraries

• ARM mBedTLS
  • Montgomery multiplication
    • Leak 83% of subtractions
Exploited libraries

• ARM mBedTLS
  • Montgomery multiplication
  • Leak 83% of subtractions
• RSA key generation
Exploited libraries

- ARM mBedTLS
  - Montgomery multiplication
    - Leak 83% of subtractions
  - RSA key generation
    - full key recovery on 65% of the runs
Exploited libraries
Exploited libraries

- Verified on all SGX architectures up to 10th gen
Exploited libraries

• Verified on all SGX architectures up to 10th gen
• Simultaneous multi-threading (SMT) *disabled*
Profiling code

secret == 1

.else:

0x00: add %rax, %rax
0x03: mov %rax, -8(%rsp)
0x08: add %rax, %rax
0x0b: mov %rax, -8(%rsp)
0x10: add %rax, %rax
0x13: mov %rax, -8(%rsp)
0x18: add %rax, %rax
0x1b: mov %rax, -8(%rsp)
0x20: add %rax, %rax
0x23: mov %rax, -8(%rsp)
0x28: add %rax, %rax
0x2b: mov %rax, -8(%rsp)
...
0x190: ret

0x00: add %rax, %rax
0x03: mov %rax, -8(%rsp)
0x08: add %rax, %rax
0x0b: mov %rax, -8(%rsp)
0x10: add %rax, %rax
0x13: mov %rax, -8(%rsp)
0x18: add %rax, %rax
0x1b: mov %rax, -8(%rsp)
0x20: add %rax, %rax
0x23: mov %rax, -8(%rsp)
0x28: add %rax, %rax
0x2b: mov %rax, -8(%rsp)
...
0x190: ret
Proiling code

```
.x25
 0x00: add %rax, %rax
 0x03: mov %rax, -8(%rsp)
 0x08: add %rax, %rax
 0x0b: mov %rax, -8(%rsp)
 0x10: add %rax, %rax
 0x13: mov %rax, -8(%rsp)
 0x18: add %rax, %rax
 0x1b: mov %rax, -8(%rsp)
 0x20: add %rax, %rax
 0x23: mov %rax, -8(%rsp)
 0x28: add %rax, %rax
 0x2b: mov %rax, -8(%rsp)
 0x190: ret
```

```
 0x00: add %rax, %rax
 0x03: mov %rax, -8(%rsp)
 0x08: add %rax, %rax
 0x0b: mov %rax, -8(%rsp)
 0x10: add %rax, %rax
 0x13: mov %rax, -8(%rsp)
 0x18: add %rax, %rax
 0x1b: mov %rax, -8(%rsp)
 0x20: add %rax, %rax
 0x23: mov %rax, -8(%rsp)
 0x28: add %rax, %rax
 0x2b: mov %rax, -8(%rsp)
 0x190: ret
```

secret == 1

..
Profiling code

.secret == 1

.else:

.align X

0x00: add %rax, %rax
0x03: mov %rax, -8(%rsp)
0x08: add %rax, %rax
0x0b: mov %rax, -8(%rsp)
0x10: add %rax, %rax
0x13: mov %rax, -8(%rsp)
0x18: add %rax, %rax
0x1b: mov %rax, -8(%rsp)
0x20: add %rax, %rax
0x23: mov %rax, -8(%rsp)
0x28: add %rax, %rax
0x2b: mov %rax, -8(%rsp)
...
0x190: ret

0x00: add %rax, %rax
0x03: mov %rax, -8(%rsp)
0x08: add %rax, %rax
0x0b: mov %rax, -8(%rsp)
0x10: add %rax, %rax
0x13: mov %rax, -8(%rsp)
0x18: add %rax, %rax
0x1b: mov %rax, -8(%rsp)
0x20: add %rax, %rax
0x23: mov %rax, -8(%rsp)
0x28: add %rax, %rax
0x2b: mov %rax, -8(%rsp)
...
0x190: ret
Profiling code

```assembly
.align X
0x00: add %rax, %rax
0x03: mov %rax, -8(%rsp)
0x08: add %rax, %rax
0x0b: mov %rax, -8(%rsp)
0x10: add %rax, %rax
0x13: mov %rax, -8(%rsp)
0x18: add %rax, %rax
0x1b: mov %rax, -8(%rsp)
0x20: add %rax, %rax
0x23: mov %rax, -8(%rsp)
0x28: add %rax, %rax
0x2b: mov %rax, -8(%rsp)
...
0x190: ret

.align Y
0x00: add %rax, %rax
0x03: mov %rax, -8(%rsp)
0x08: add %rax, %rax
0x0b: mov %rax, -8(%rsp)
0x10: add %rax, %rax
0x13: mov %rax, -8(%rsp)
0x18: add %rax, %rax
0x1b: mov %rax, -8(%rsp)
0x20: add %rax, %rax
0x23: mov %rax, -8(%rsp)
0x28: add %rax, %rax
0x2b: mov %rax, -8(%rsp)
...
0x190: ret
```
Analyzing the results
Analyzing the results
Analyzing the results

- Success rates depend on the branch alignment
Analyzing the results

• Success rates depend on the branch alignment

• Same success rates every 16 bytes
Analyzing the results

- Success rates depend on the branch alignment
- Same success rates every 16 bytes
- 50% success rate in the diagonals
What happens to the pipeline as we interrupt?

0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret

L1 - Instruction $

48 83 C0 01
48 89 44 24 10
48 83 C3 01
48 89 5C 24 08
C3
What happens to the pipeline as we interrupt?

IP
0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret

L1 - Instruction $
4883C00148894424104883C30148895C
2408C3...

Instruction Fetch & PreDecode
Instruction Queue
Decoder
Allocation Queue

μOP Scheduler
Execution Units
 Rename / Allocate / Retire

Instruction
Status
What happens to the pipeline as we interrupt?

0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret

L1 - Instruction $ 4883C00148894424104883C30148895C 2408C3 ...

Instruction Fetch & PreDecode 4883C00148894424104883C30148895C

Instruction Queue Decoder Allocation Queue

Execution Units μOP Scheduler Rename / Allocate / Retire

IP
What happens to the pipeline as we interrupt?

L1 - Instruction $:

4883C00148894424104883C30148895C
2408C3...

Instruction Queue
Decoder
Allocation Queue

Instruction Fetch & PreDecode:

4883C00148894424104883C30148895C

Renamed / Allocated / Retired

Decoder
Instruction Queue
Allocation Queue

Instruction Queue
Decoder
Allocation Queue

μOP Scheduler
Execution Units

Backend

IP

0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret
What happens to the pipeline as we interrupt?

Instruction Queue:
- 48 48 48
- 83 89 83
- C0 44 C3
- 01 24 01
- 10

Decoder:

Allocation Queue:

IP:
- 0x10: add $1, %rax
- 0x14: mov %rax, (var1)
- 0x19: add $1, %rbx
- 0x1d: mov %rbx, (var2)
- 0x22: ret

L1 - Instruction $:
- 4883C00148894424104883C30148895C
- 2408C3...

Backend:
- Execution Units
- μOP Scheduler
- Rename / Allocate / Retire

Instruction Fetch & PreDecode:
- 48895C 2408C3...
What happens to the pipeline as we interrupt?

0x10: `add $1, %rax`
0x14: `mov %rax, (var1)`
0x19: `add $1, %rbx`
0x1d: `mov %rbx, (var2)`
0x22: `ret`

L1 - Instruction $:

```
4883C00148894424104883C30148895C2408C3...
```

Instruction Queue:

| 48 | 48 | 48 |
| 83 | 89 | 83 |
| C0 | 44 | C3 |
| 01 | 24 | 01 |
| 10 |

Decoder:

```
48895C2408C3...
```

Allocation Queue:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What happens to the pipeline as we interrupt?

Instruction Queue

48  C3
89
5C
24
08

Decoder

48  48  48
83  89  83
C0  44  C3
01  24  01
10

Allocation Queue

L1 - Instruction $4883C00148894424104883C30148895C2408C3...

0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret

IP
What happens to the pipeline as we interrupt?

Frontend

```
0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret
```

```
4883C00148894424104883C30148895C
2408C3...
```

Instruction Queue

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>48 C3</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td></td>
</tr>
<tr>
<td>5C</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td></td>
</tr>
</tbody>
</table>

Decoder

- `add μOP`
- `mov μOP`
- `add μOP`

Allocation Queue

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
</table>

Backend

- Execution Units
- μOP Scheduler
- Rename / Allocate / Retire
What happens to the pipeline as we interrupt?

0x10: `add $1, %rax`
0x14: `mov %rax, (var1)`
0x19: `add $1, %rbx`
0x1d: `mov %rbx, (var2)`
0x22: `ret`

L1 - Instruction $4883C00148894424104883C30148895C2408C3...

Instruction Queue:
- 48 C3
- 89
- 5C
- 24
- 08

Decoder
- `add μOP`
- `mov μOP`
- `add μOP`

Allocation Queue

Frontend
- Instruction Fetch & PreDecode

Backend
- Execution Units
- μOP Scheduler

Rename / Allocate / Retire

IP

4883C00148894424104883C30148895C2408C3...
What happens to the pipeline as we interrupt?

IP

0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret

L1 - Instruction $

4883C00148894424104883C30148895C
2408C3...

Instruction Fetch & PreDecode

Instruction Queue

Decoder

mov μOP
ret μOP

Allocation Queue

add μOP
mov μOP
add μOP

Backend

μOP Scheduler

Execution Units

Rename / Allocate / Retire

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What happens to the pipeline as we interrupt?

### L1 - Instruction $\$1$

- **0x10:** `add $1, %rax`
- **0x14:** `mov %rax, (var1)`
- **0x19:** `add $1, %rbx`
- **0x1d:** `mov %rbx, (var2)`
- **0x22:** `ret`

### Instruction Fetch & PreDecode

```
4883C00148894424104883C30148895C
2408C3...
```

### Execution Units

- `mov %rax, (var1)`
- `mov %rbx, (var2)`

### μOP Scheduler

- `add $1, %rax`
- `mov %rax, (var1)`
- `add $1, %rbx`
- `mov %rbx, (var2)`
- `ret`

### Rename / Allocate / Retire

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add</code></td>
<td>Pending</td>
</tr>
<tr>
<td><code>mov</code></td>
<td>Pending</td>
</tr>
<tr>
<td><code>add</code></td>
<td>Pending</td>
</tr>
</tbody>
</table>
What happens to the pipeline as we interrupt?

```
0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret
```

L1 - Instruction $

```
4883C00148894424104883C30148895C
2408C3...
```

IP
What happens to the pipeline as we interrupt?

0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret

4883C00148894424104883C30148895C2408C3...
What happens to the pipeline as we interrupt?

**Instruction Queue**
- Add $1, %rax
- Mov %rax, (var1)
- Add $1, %rbx
- Mov %rbx, (var2)
- Ret

**Instruction Fetch & PreDecode**
- L1 - Instruction $4883C00148894424104883C30148895C2408C3...

**Execution Units**
- Add mov add

**μOP Scheduler**
- Mov ret

**Rename / Allocate / Retire**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Executing</td>
</tr>
<tr>
<td>Mov</td>
<td>Executing</td>
</tr>
<tr>
<td>Add</td>
<td>Executing</td>
</tr>
<tr>
<td>Mov</td>
<td>Executing</td>
</tr>
<tr>
<td>Ret</td>
<td>Executing</td>
</tr>
</tbody>
</table>
What happens to the pipeline as we interrupt?

Instruction Queue

Decoder

Allocation Queue

L1 - Instruction $4883C00148894424104883C30148895C 2408C3...

0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret
What happens to the pipeline as we interrupt?

```
0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret
```

L1 - Instruction $4883C00148894424104883C30148895C 2408C3...

Instruction Fetch & PreDecode

Instruction Queue

Decoder

Allocation Queue

Execution Units
add mov

μOP Scheduler
mov ret

Rename / Allocate / Retire

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Executing</td>
</tr>
<tr>
<td>mov</td>
<td>Executing</td>
</tr>
<tr>
<td>add</td>
<td>Completed</td>
</tr>
<tr>
<td>mov</td>
<td>Executing</td>
</tr>
<tr>
<td>ret</td>
<td>Executing</td>
</tr>
</tbody>
</table>
What happens to the pipeline as we interrupt?

### L1 - Instruction $\$

```
4883C00148894424104883C30148895C
2408C3...
```

### Frontend
- **Instruction Queue**
- **Decoder**
- **Allocation Queue**

### Backend
- **Execution Units**
  - add
  - mov

- **μOP Scheduler**
  - mov
  - ret

### Table

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Executing</td>
</tr>
<tr>
<td>mov</td>
<td>Executing</td>
</tr>
<tr>
<td>add</td>
<td>Completed</td>
</tr>
<tr>
<td>mov</td>
<td>Executing</td>
</tr>
<tr>
<td>ret</td>
<td>Executing</td>
</tr>
</tbody>
</table>
What happens to the pipeline as we interrupt?

```
0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret
```

L1 - Instruction $

```
4883C00148894424104883C30148895C
2408C3...
```

Instruction Fetch & PreDecode

Execution Units

- add

μOP Scheduler

Rename / Allocate / Retire

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Executing</td>
</tr>
</tbody>
</table>
What happens to the pipeline as we interrupt?

L1 - Instruction $ 

4883C00148894424104883C30148895C 2408C3...

0x10: \text{add} \ $1, \ %rax 
0x14: \text{mov} \ %rax, \ (\text{var}1) 
0x19: \text{add} \ $1, \ %rbx 
0x1d: \text{mov} \ %rbx, \ (\text{var}2) 
0x22: \text{ret}
What happens to the pipeline as we interrupt?

L1 - Instruction $:

4883C00148894424104883C30148895C
2408C3...

0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret

Frontend:
Instruction Fetch & PreDecode
Instruction Queue
Decoder
Allocation Queue

Backend:
μOP Scheduler
Execution Units
Rename / Allocate / Retire

Interrupt
What happens to the pipeline as we interrupt?

```
0x2b: add $2, %rax
0x2f: mov %rax, (var1)
0x34: add $2, %rbx
0x38: mov %rbx, (var2)
0x3d: ret
```

```
...4883C00148
89424104883C30148895C2408C3...
```

### Instruction Queue

### Decoder

### Allocation Queue

### L1 - Instruction $
What happens to the pipeline as we interrupt?

```plaintext
IP
else:
0x2b: add $2, %rax
0x2f: mov %rax, (var1)
0x34: add $2, %rbx
0x38: mov %rbx, (var2)
0x3d: ret

.L1 - Instruction $:

894424104883C30148895C2408C3...

...4883C00148

Else:
```
What happens to the pipeline as we interrupt?

```
IP
0x2b: add $2, %rax
0x2f: mov %rax, (var1)
0x34: add $2, %rbx
0x38: mov %rbx, (var2)
0x3d: ret
```

```
L1 - Instruction $

...4883C00148
894424104883C30148895C2408C3...
```

Frontend
- Instruction Fetch & PreDecode
- Instruction Queue
- Decoder
- Allocation Queue

Backend
- Execution Units
- μOP Scheduler
- Rename / Allocate / Retire

```
.example:
```
```
```
```
```
```
```
```
```
```
```
```
```
What happens to the pipeline as we interrupt?

```
IP
0x2b: add $2, %rax
0x2f: mov %rax, (var1)
0x34: add $2, %rbx
0x38: mov %rbx, (var2)
0x3d: ret
```

```
L1 - Instruction $

...4883C00148
894424104883C30148895C2408C3...
```

```
Instruction Queue
48
83
C0
01
```

```
Decoder
```

```
Allocation Queue
```

```
Backend

Execution Units

μOP Scheduler

Rename / Allocate / Retire
```

```
Frontend

Instruction Fetch & PreDecode
48894424104883C30148895C2408C3...
```
What happens to the pipeline as we interrupt?

```
0x2b: add $2, %rax
0x2f: mov %rax, (var1)
0x34: add $2, %rbx
0x38: mov %rbx, (var2)
0x3d: ret
```

Instruction Queue:
- 48
- 83
- C0
- 01

Decoder:

Allocation Queue:

L1 - Instruction $:
```
...4883C00148
894424104883C30148895C2408C3...
```

Frontend:

Instruction Fetch & PreDecode:
```
48894424104883C30148895C2408C3...
```

Backend:

Execution Units

\(\mu\)OP Scheduler

Rename / Allocate / Retire

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constraint satisfaction
What happens to the pipeline as we interrupt?

```
0x2b: add $2, %rax
0x2f: mov %rax, (var1)
0x34: add $2, %rbx
0x38: mov %rbx, (var2)
0x3d: ret
```

L1 - Instruction $

...4883C00148
894424104883C30148895C2408C3...

### Backend

- **Execution Units**
- **μOP Scheduler**
- **Rename / Allocate / Retire**

### Frontend

- **Instruction Fetch & PreDecode**
- **Instruction Queue**
- **Decoder**
- **Allocation Queue**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Pending</td>
</tr>
</tbody>
</table>

### Example

```
0x2b: add $2, %rax
0x2f: mov %rax, (var1)
0x34: add $2, %rbx
0x38: mov %rbx, (var2)
0x3d: ret
```
Comparison after each interrupt

First instructions to reach this stage after each interrupt

0x10:  add $1, %rax
0x14:  mov %rax, (var1)
0x19:  add $1, %rbx
0x1d:  mov %rbx, (var2)
0x22:  ret

0x2b:  add $2, %rax
0x2f:  mov %rax, (var1)
0x34:  add $2, %rbx
0x38:  mov %rbx, (var2)
0x3d:  ret
Comparison after each interrupt

First instructions to reach this stage after each interrupt:

- IP: 0x10: \texttt{add} \ $1$, \%rax
- IP: 0x14: \texttt{mov} \%rax, \ (\texttt{var1})
- IP: 0x19: \texttt{add} \ $1$, \%rbx
- IP: 0x1d: \texttt{mov} \%rbx, \ (\texttt{var2})
- IP: 0x22: \texttt{ret}

- IP: 0x2b: \texttt{add} \ $2$, \%rax
- IP: 0x2f: \texttt{mov} \%rax, \ (\texttt{var1})
- IP: 0x34: \texttt{add} \ $2$, \%rbx
- IP: 0x38: \texttt{mov} \%rbx, \ (\texttt{var2})
- IP: 0x3d: \texttt{ret}
Comparison after each interrupt

First instructions to reach this stage after each interrupt

Renamed / Allocate / Retire

If branch

Else branch

Add mov add

Add
Comparison after each interrupt

First instructions to reach this stage after each interrupt

0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret

0x2b: add $2, %rax
0x2f: mov %rax, (var1)
0x34: add $2, %rbx
0x38: mov %rbx, (var2)
0x3d: ret
Comparison after each interrupt

0x10: add $1, %rax
0x14: mov %rax, (var1)
0x19: add $1, %rbx
0x1d: mov %rbx, (var2)
0x22: ret

0x2b: add $2, %rax
0x2f: mov %rax, (var1)
0x34: add $2, %rbx
0x38: mov %rbx, (var2)
0x3d: ret

First instructions to reach this stage after each interrupt
Comparison after each interrupt

Rename / Allocate / Retire

**IP**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10</td>
<td>add $1, %rax</td>
<td>0x14</td>
<td>mov %rax, (var1)</td>
</tr>
<tr>
<td>0x19</td>
<td>add $1, %rbx</td>
<td>0x1d</td>
<td>mov %rbx, (var2)</td>
</tr>
<tr>
<td>0x22</td>
<td>ret</td>
<td>0x2b</td>
<td>add $2, %rax</td>
</tr>
<tr>
<td>0x2f</td>
<td>mov %rax, (var1)</td>
<td>0x34</td>
<td>add $2, %rbx</td>
</tr>
<tr>
<td>0x38</td>
<td>mov %rbx, (var2)</td>
<td>0x3d</td>
<td>ret</td>
</tr>
</tbody>
</table>

First instructions to reach this stage after each interrupt
Comparison after each interrupt

First instructions to reach this stage after each interrupt

<table>
<thead>
<tr>
<th>If branch</th>
<th>Else branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
</tr>
<tr>
<td>mov</td>
<td>mov add mov ret</td>
</tr>
<tr>
<td>add</td>
<td>add mov ret</td>
</tr>
<tr>
<td>mov ret</td>
<td>mov ret</td>
</tr>
</tbody>
</table>
Defenses
Defenses

• Principled approach:
Defenses

• Principled approach:
  • Avoiding secret-dependent branches
Defenses

• Principled approach:
  • Avoiding secret-dependent branches
• Ad-hoc:
Defenses

• Principled approach:
  • Avoiding secret-dependent branches

• Ad-hoc:
  • Align all branches to the same offset (mod 16)
Defenses

• Principled approach:
  • Avoiding secret-dependent branches
• Ad-hoc:
  • Align all branches to the same offset (mod 16)
  • Space overhead usually < 0.5 %
Defenses

• Principled approach:
  • Avoiding secret-dependent branches

• Ad-hoc:
  • Align all branches to the same offset (mod 16)
  • Space overhead usually < 0.5 %
  • Performance overhead between -5.5% and 30%