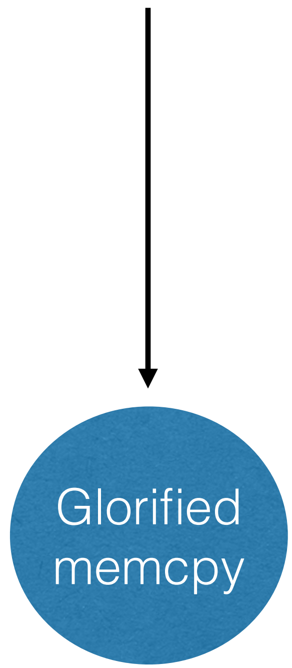
Run-DMA

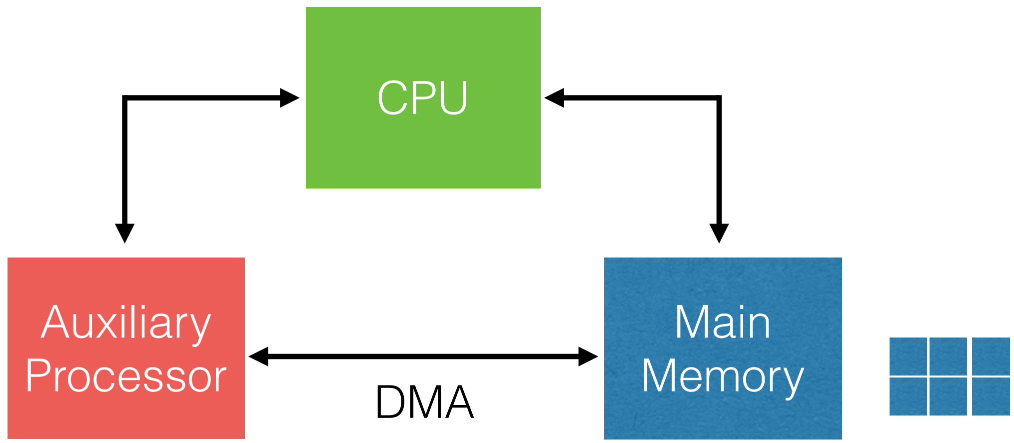
Michael Rushanan, Stephen Checkoway
Johns Hopkins University, University of Illinois at Chicago

Introduction

- Arbitrary computation using *Direct Memory Access engine*
- Access all resources of the device
- Implement the following as an example:
  - Brainfuck
  - Rootkit



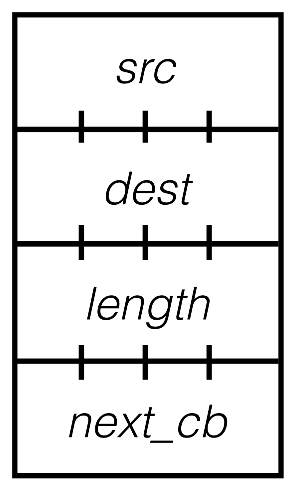
Direct Memory Access



- Offload task of copying memory to/from auxiliary processors (e.g., NIC, GPU, etc)
- Free CPU to do more interesting work

DMA Engine

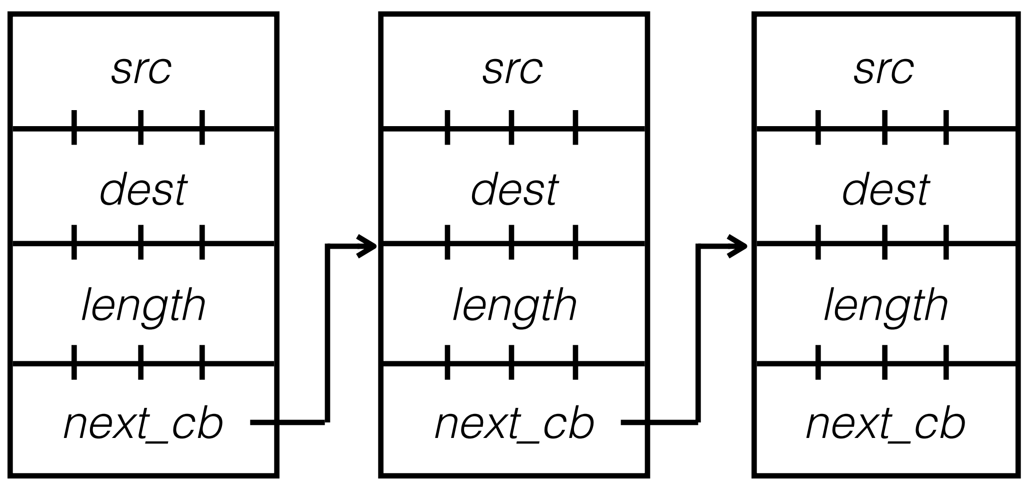
- CPU configures DMA transfer by setting control registers
- Control registers specify transfer operation



Control Block Structure

Control Blocking Chaining

- Scatter/gather DMA can transfer to/from multiple memory areas in a single transaction
- Configure a sequence of control blocks

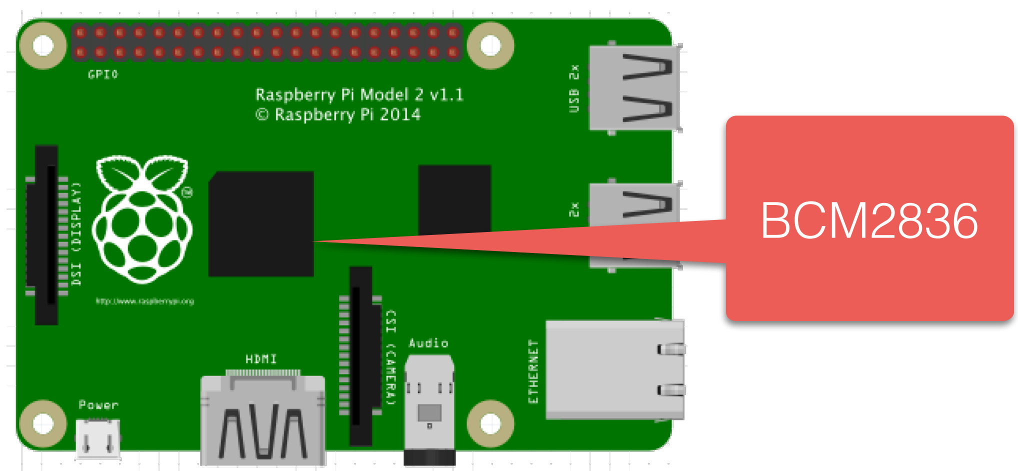


Required DMA Properties

- Perform memory-to-memory copies
- Programmed by loading address of control blocks
- Supports scatter/gather mode

Target Device

- Raspberry Pi 2 single-board computer



- Other Potential DMA Engines:
  - Intel 8237 (e.g., legacy IBM PC/ATs)
  - Cell multi-core microprocessor (e.g., PS3)

DMA Gadgets

- DMA "programs" require self-modifying constructs
    - Overwrite members of later control blocks

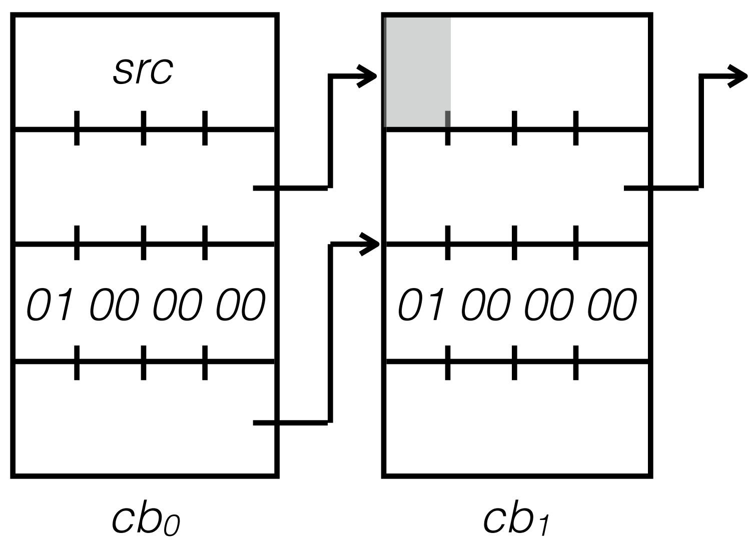
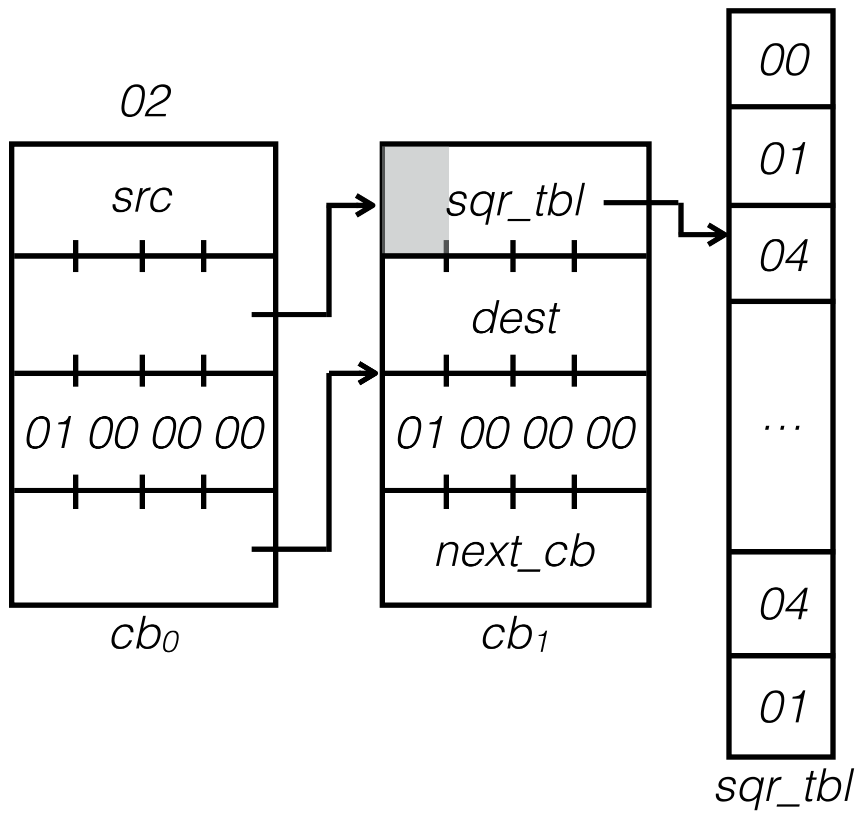


Table Lookups



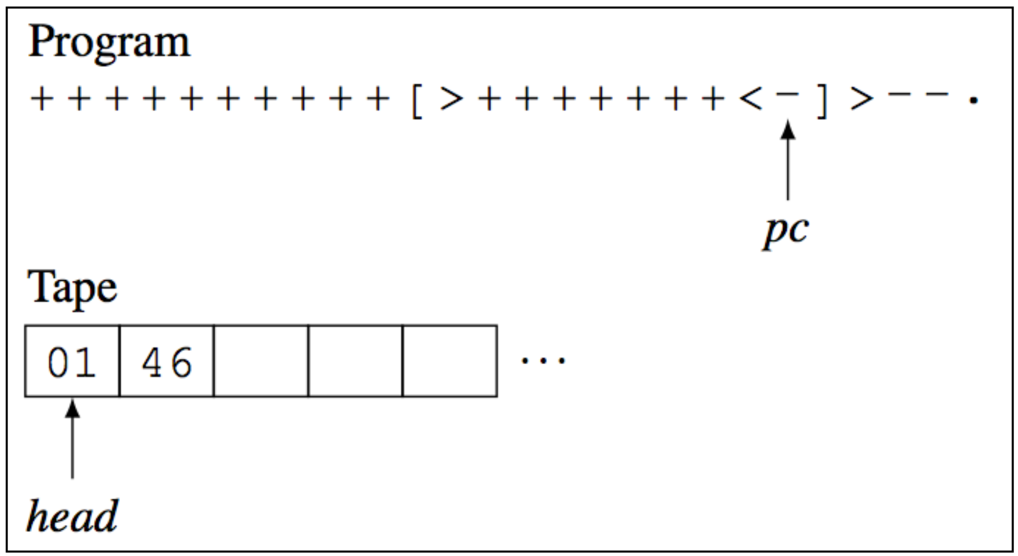
Basic Building Blocks

Unary Functions	Lookup value in table and store to memory	$y = f(x)$
Variable Dereferencing	Copy value pointed to into src/dest of subsequent control block	*x

Basic Building Blocks

Conditional Goto	Address of a control block written to the next_cb member of a trampoline
Switch	Offset table with entries that are offsets into an address table
Memory-mapped I/O Registers	Loop over memory-mapped flag or status register

BrainFuck



BrainFuck

+	increment the cell pointed to by head	++*ptr;
-	decrement the cell pointed to by head	--*ptr;
>	increment head to point to the next cell	++ptr;
<	decrement head to point to the previous cell	--ptr;

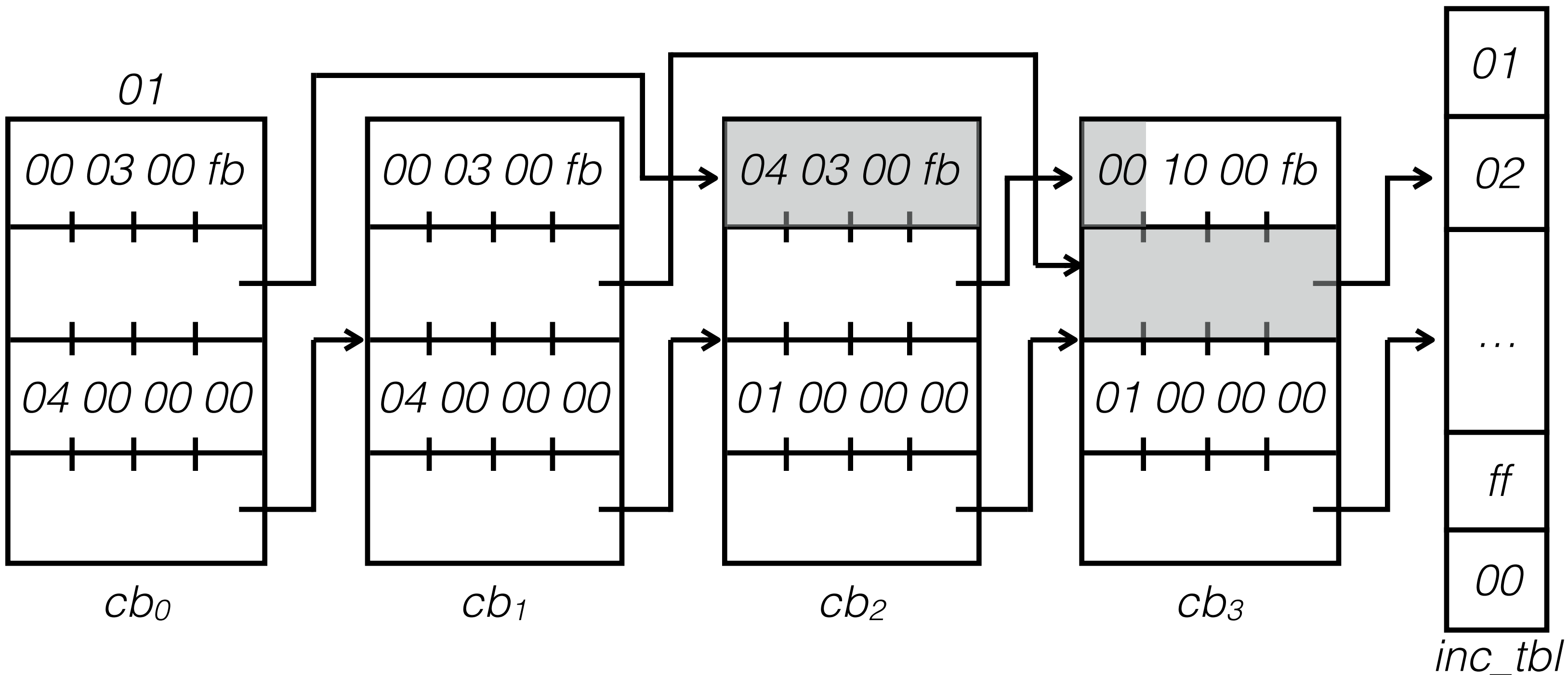
BrainFuck

[	if the cell pointed to by head is nonzero, execute next instruction; otherwise, jump to the instruction following]	while (*ptr) {
]	if the cell pointed to by head is zero, execute next instruction; otherwise, jump to the instruction following [	}
,	store the input to the cell pointed to by head	*ptr=getchar();
.	output the cell pointed to by head	putchar(*ptr);

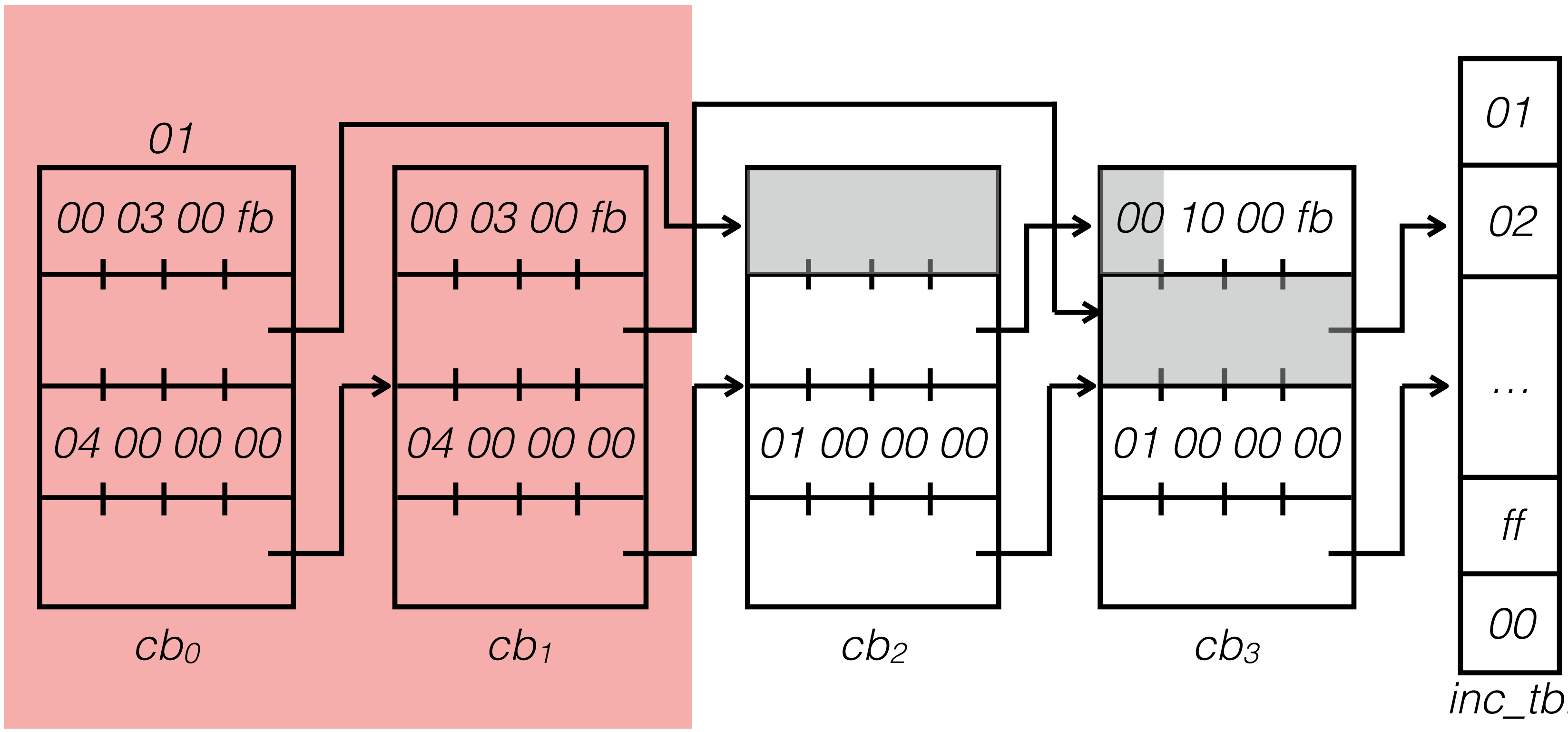
Interpreter Implementation

- 8 gadgets corresponding to BrainFuck instructions
- Dispatch
- Increment word and decrement word
- Fetch Next instruction (i.e., increment PC and dispatch)

Increment

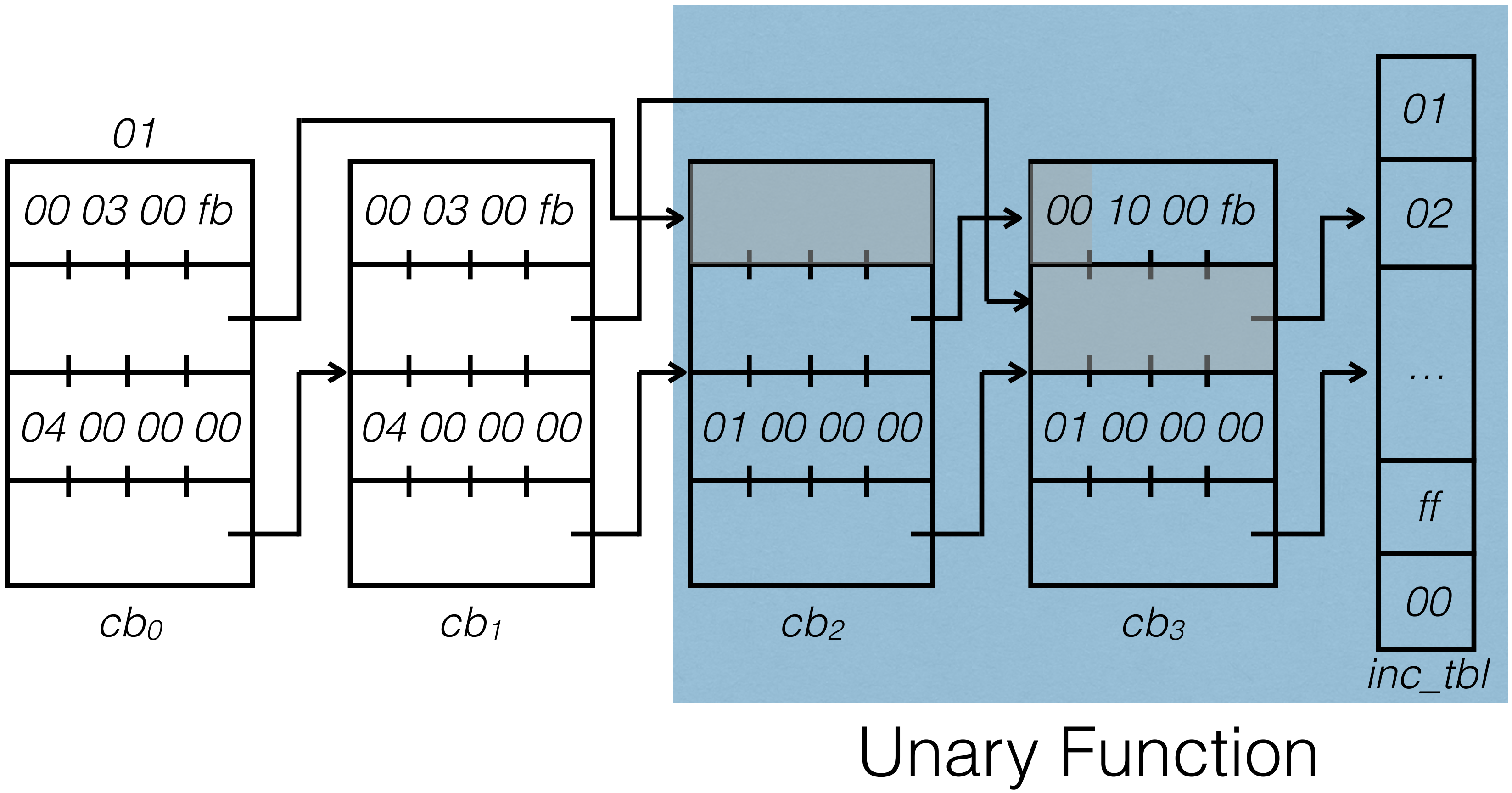

01
00 03 00 fb
04 00 00 00
cb0
00 03 00 fb
04 00 00 00
cb1
04 03 00 fb
01 00 00 00
cb2
00 10 00 fb
01 00 00 00
cb3
01
02
…
ff
00
inc_tbl

Increment

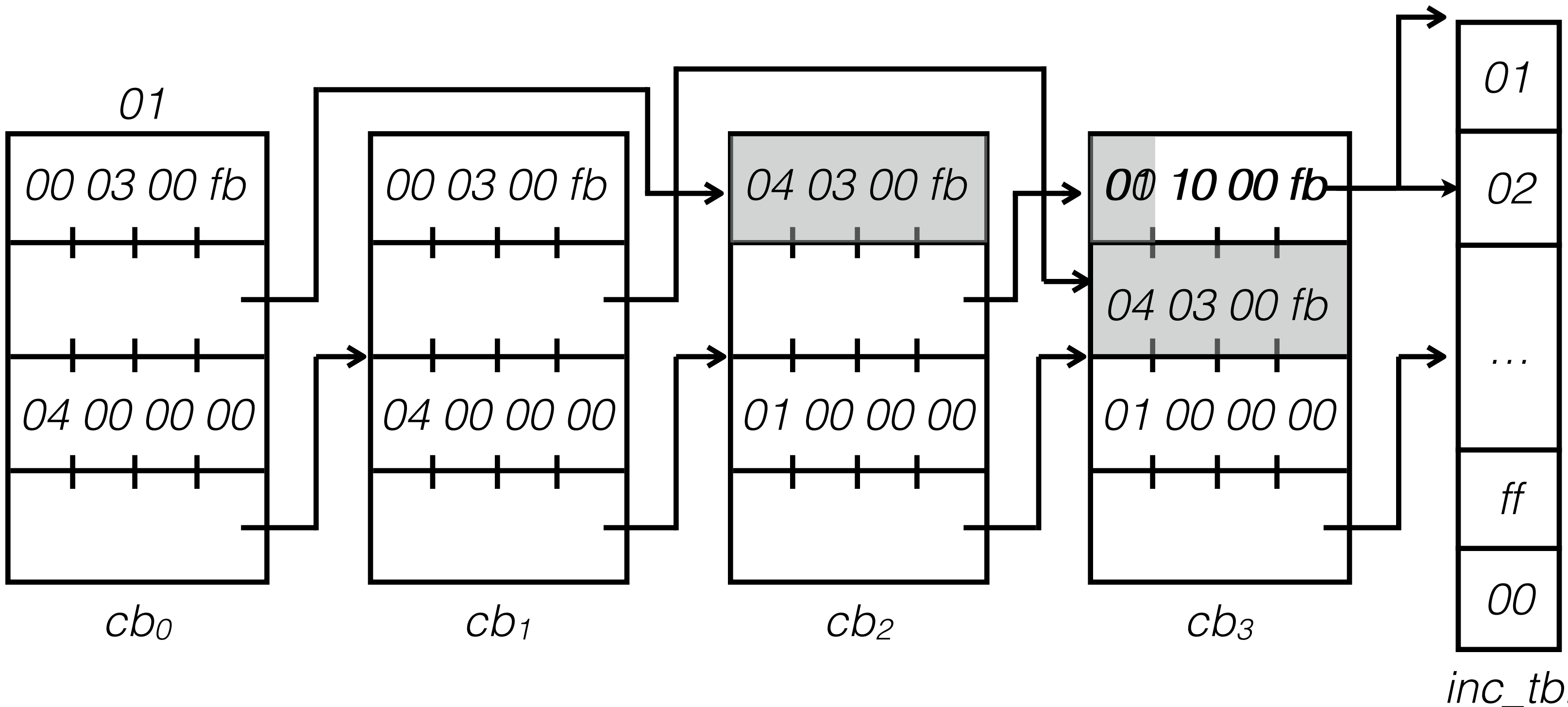


Variable Dereference

Increment

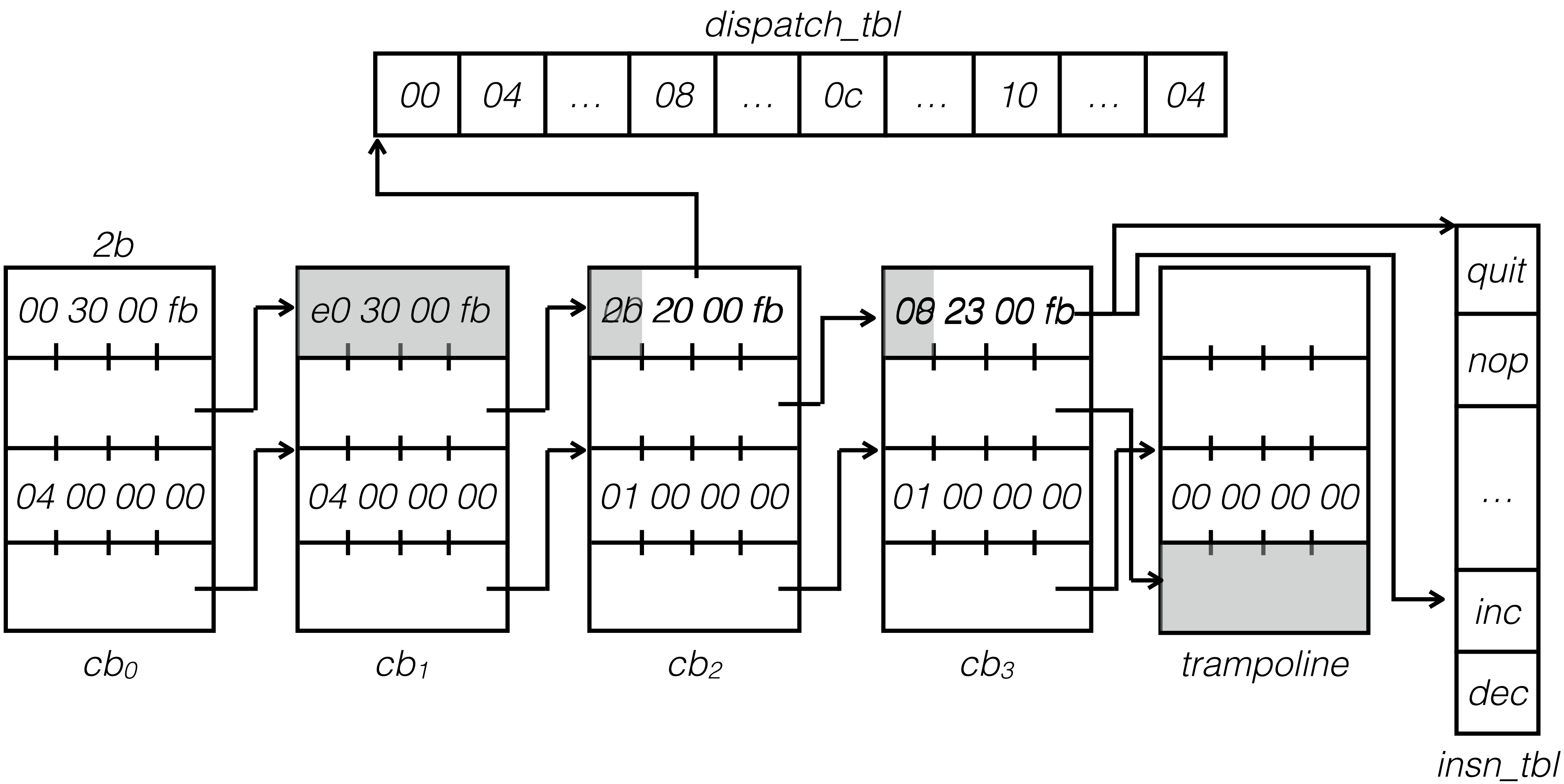

01
00 03 00 fb
04 00 00 00
cb0
00 03 00 fb
04 00 00 00
cb1
01 00 00 00
cb2
00 10 00 fb
01 00 00 00
cb3
01
02
...
ff
00
inc_tbl
Unary Function

Increment

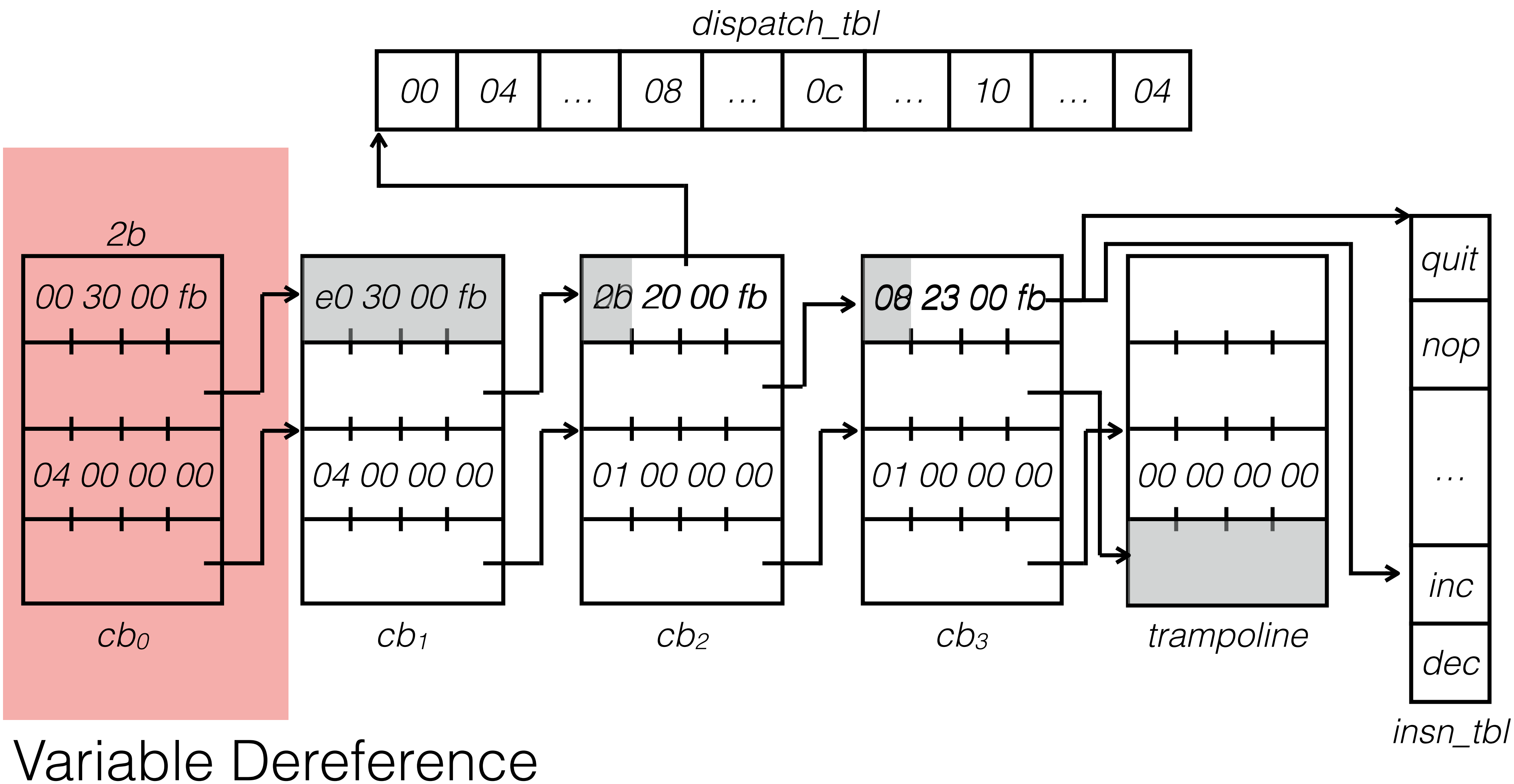


01
00 03 00 fb
04 00 00 00
cb0
00 03 00 fb
04 00 00 00
cb1
04 03 00 fb
01 00 00 00
cb2
00 10 00 fb
04 03 00 fb
01 00 00 00
cb3
01
02
…
ff
00
inc_tbl

Dispatch

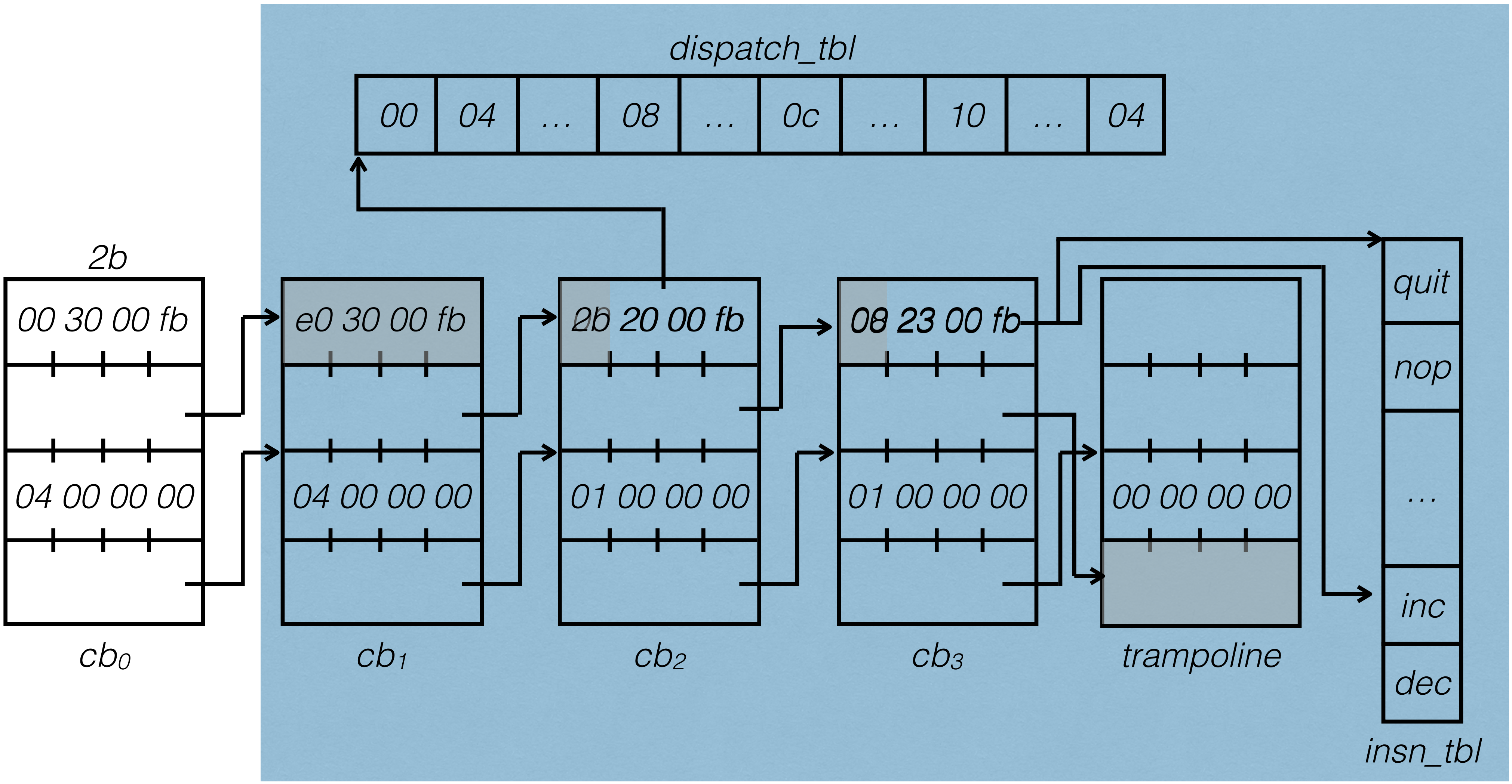

dispatch_tbl
00
04
…
08
…
0c
…
10
…
04
2b
00 30 00 fb
04 00 00 00
e0 30 00 fb
04 00 00 00
2b 20 00 fb
01 00 00 00
08 23 00 fb
01 00 00 00
00 00 00 00
quit
nop
…
inc
dec
cb0
cb1
cb2
cb3
trampoline
insn_tbl

Dispatch



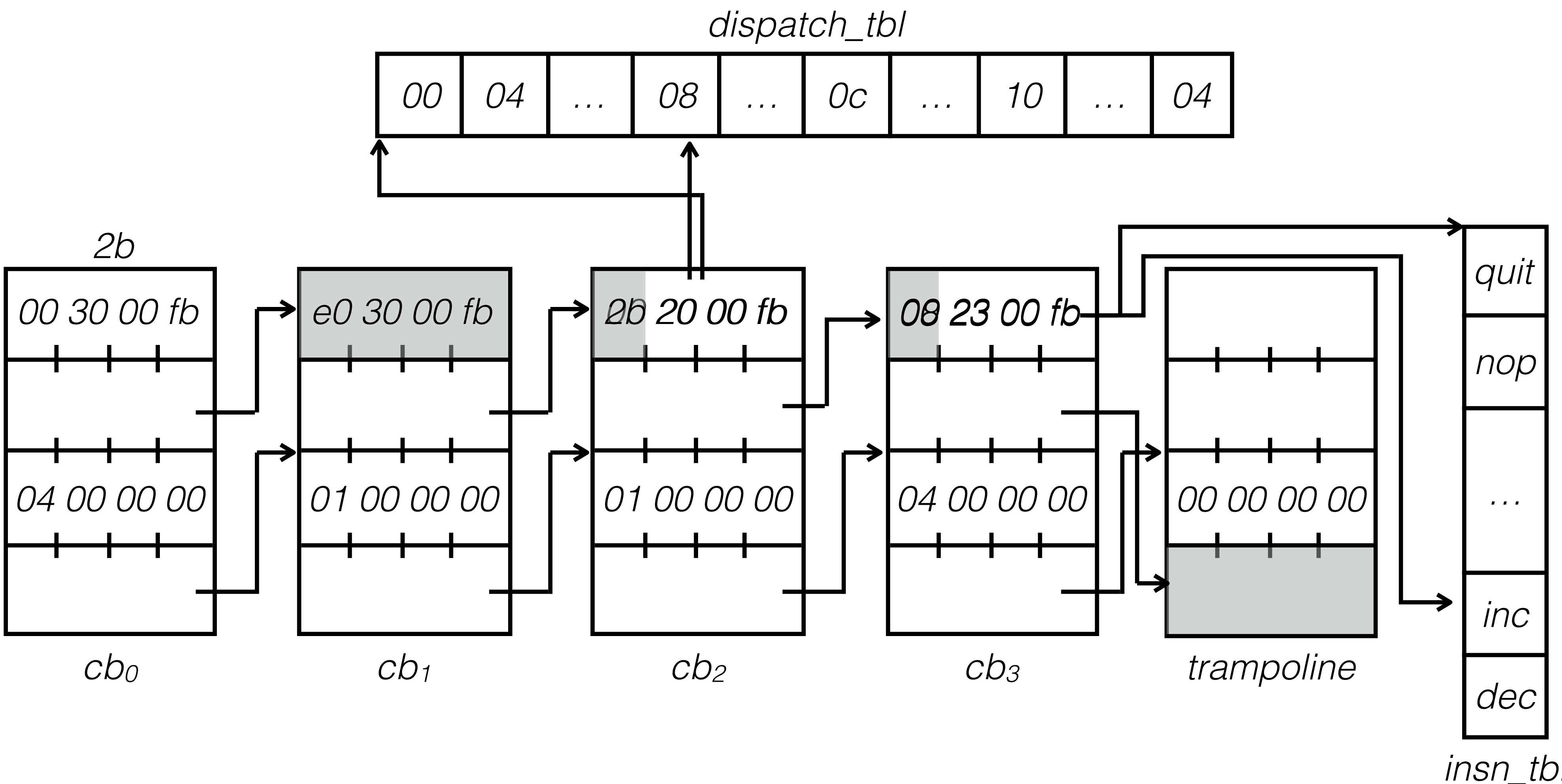
dispatch_tbl
00
04
...
08
...
0c
...
10
...
04
2b
00 30 00 fb
04 00 00 00
cb0
e0 30 00 fb
04 00 00 00
cb1
2b 20 00 fb
01 00 00 00
cb2
08 23 00 fb
01 00 00 00
cb3
00 00 00 00
trampoline
quit
nop
...
inc
dec
insn_tbl
Variable Dereference

Dispatch



dispatch_tbl
00
04
…
08
…
0c
…
10
…
04
2b
00 30 00 fb
04 00 00 00
cb0
e0 30 00 fb
04 00 00 00
cb1
2b 20 00 fb
01 00 00 00
cb2
08 23 00 fb
01 00 00 00
cb3
00 00 00 00
trampoline
quit
nop
…
inc
dec
insn_tbl
Switch

Dispatch


dispatch_tbl
00
04
…
08
…
0c
…
10
…
04
2b
00 30 00 fb
04 00 00 00
e0 30 00 fb
01 00 00 00
2b 20 00 fb
01 00 00 00
08 23 00 fb
04 00 00 00
00 00 00 00
quit
nop
…
inc
dec
cb0
cb1
cb2
cb3
trampoline
insn_tbl

Turing-Complete

Simulate any other computational device/language

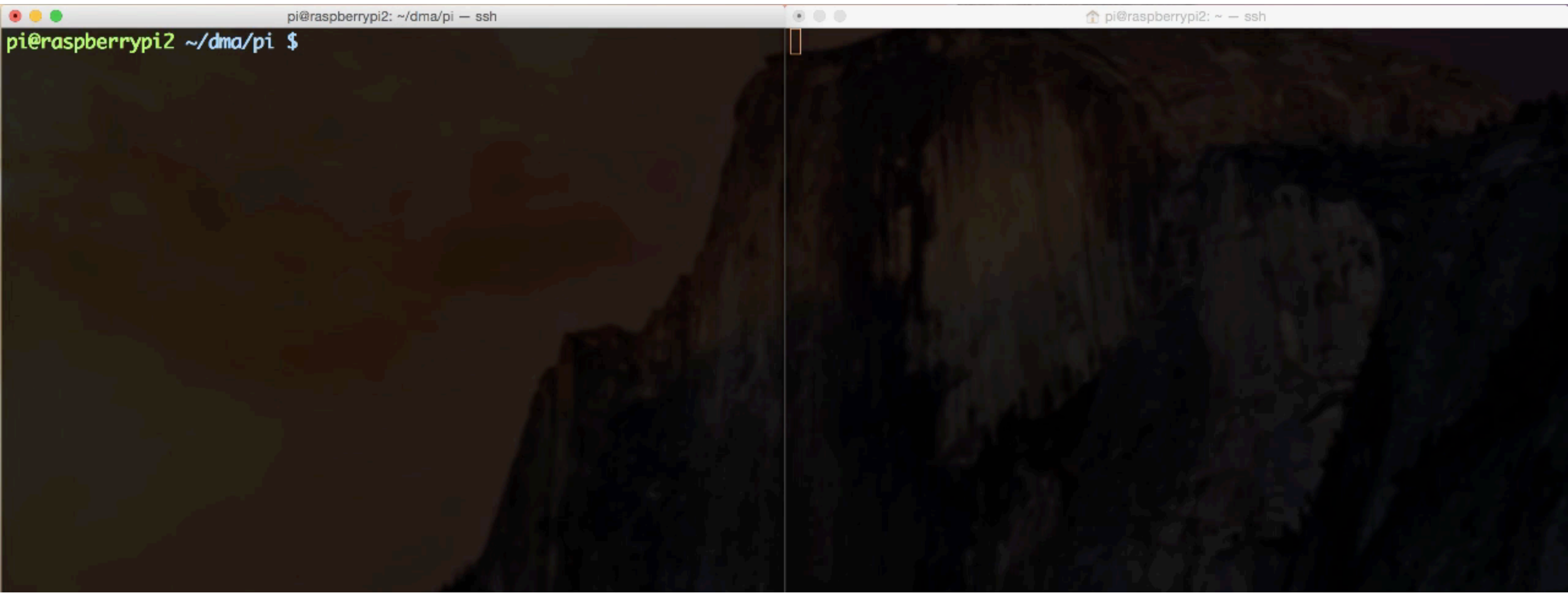
- BrainFuck is Turing-complete
- We implemented BrainFuck with DMA gadgets
- Thus DMA gadgets are Turing-complete

Resource-Complete

Access all resources of system from within the language

- DMA has access to memory-mapped IO registers
- Thus DMA gadgets are resource-complete

Hello World



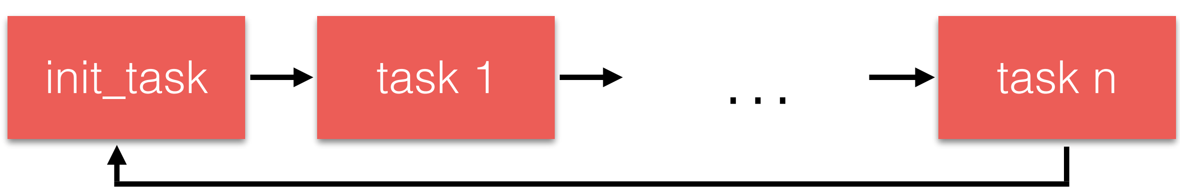
https://github.com/stevecheckoway/rundma

More Gadgets

- Binary functions
  - $f : \{0,1\}^8 \times \{0,1\}^8 \rightarrow \{0,1\}^8$
- Relational operators
  - Equality (e.g., =)
  - Inequality (e.g., <)

Raspbian Rootkit

- Raspbian Linux
- task_structs hold information about a process
    - pointer to cred structure (e.g., UID of process)
    - pointer to next structure



DMA Performance

Gadget	Control Blocks
inc/dec	4
inc/dec word	4 + 2 trampolines
dispatch	33
right/left	26
left/right condition	2
I/O	5

Total DMA Transfers

Program	Control Blocks
Interpreter	148
Hello World	36356
Rootkit	20

DMA Malware

- DMA Malware
  - Code running on auxilary processor/external device with DMA access
  - Example: firewire, thunderbolt, NIC, GPU
- Main difference of our work:
  - DMA gadgets run entirely on DMA engine
  - No additional processors

Countermeasures

- Input/out memory management (Duflot, 2011)
- Peripheral firmware load-time integrity (Stewin, 2012)
- Anomaly detection systems (Duflot, 2011)
- Bus agent runtime monitors (Stewin, 2013)

Conclusion

- Everything non-trivial ends up being Turing-complete
  - Parsing file formats
  - Page Tables
- DMA Engine is yet another example
- We need to consider specialized hardware

Questions?