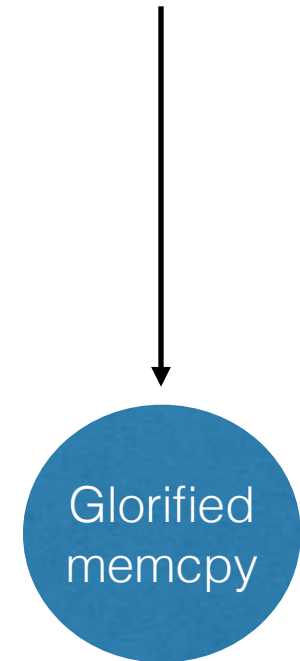


Run-DMA

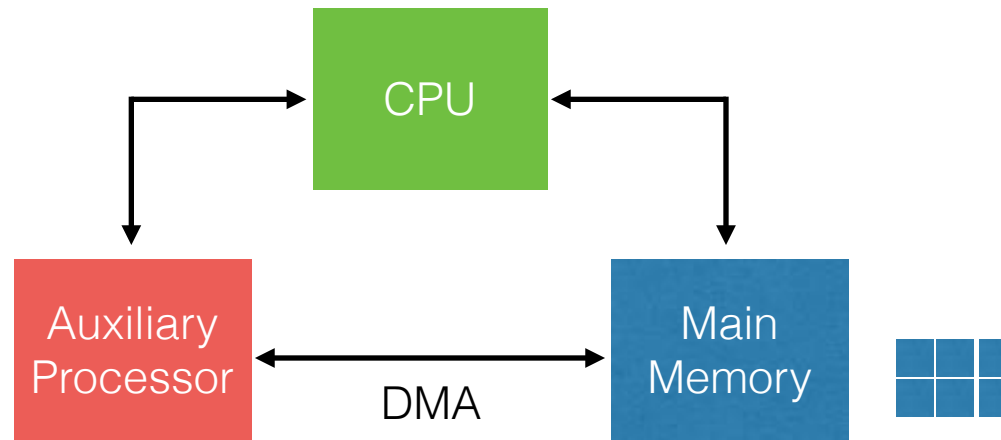
Michael Rushanan, Stephen Checkoway
Johns Hopkins University, University of Illinois at Chicago

Introduction

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



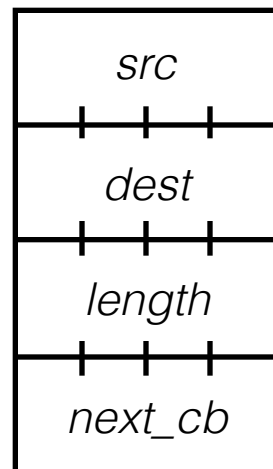
Direct Memory Access



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

DMA Engine

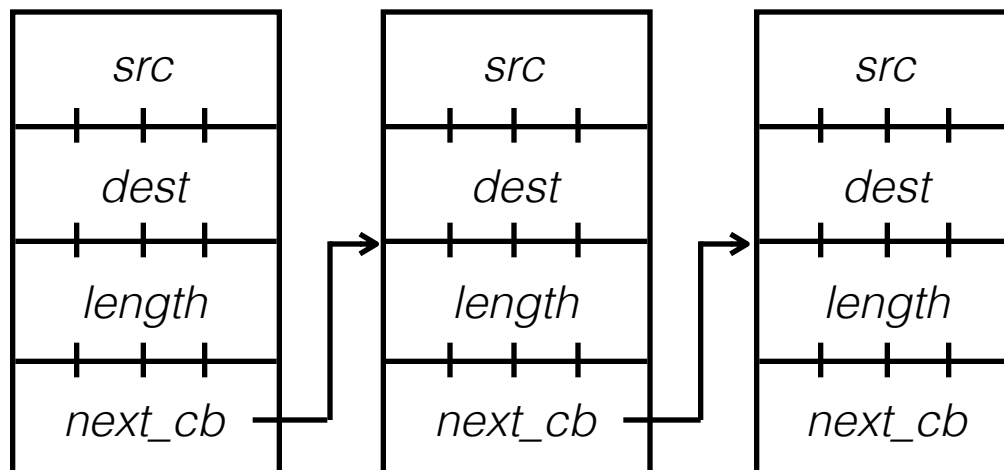
- CPU configures DMA transfer by setting control registers
- Control registers specify transfer operation



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

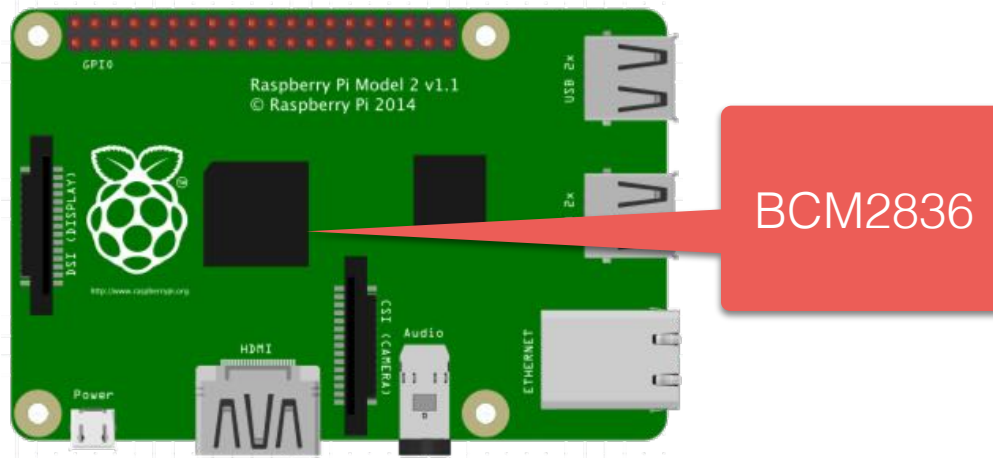


Required DMA Properties

- 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

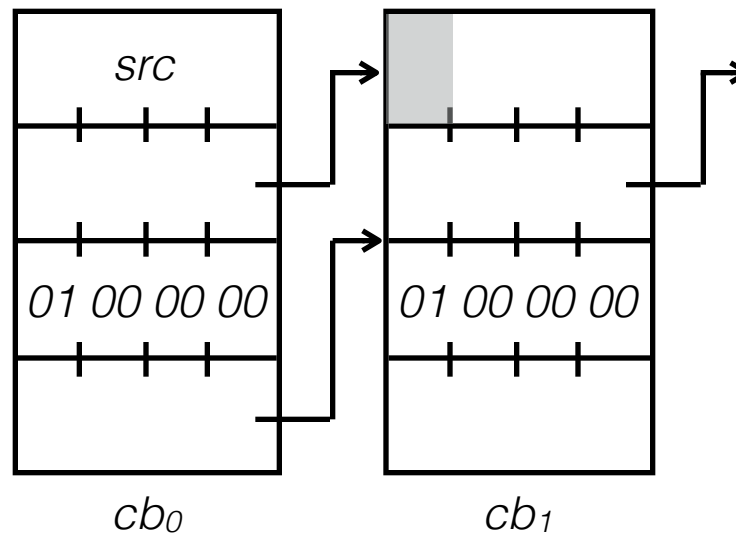
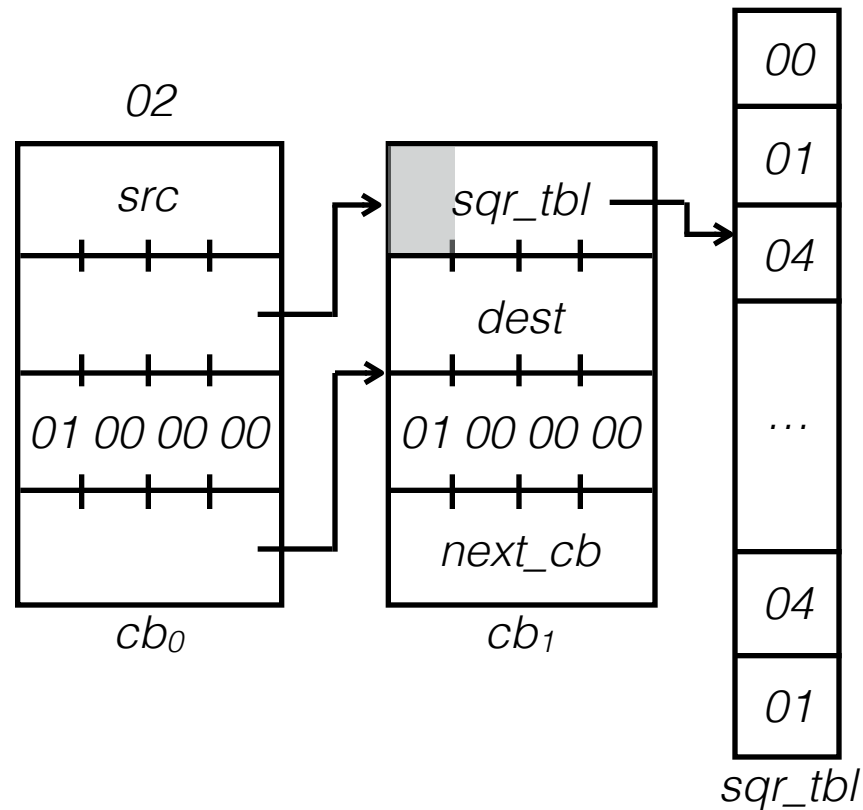


Table Lookups



Basic Building Blocks

Unary
Functions

Lookup value in table and
store to memory

$y = f(x)$

Variable
Dereferencing

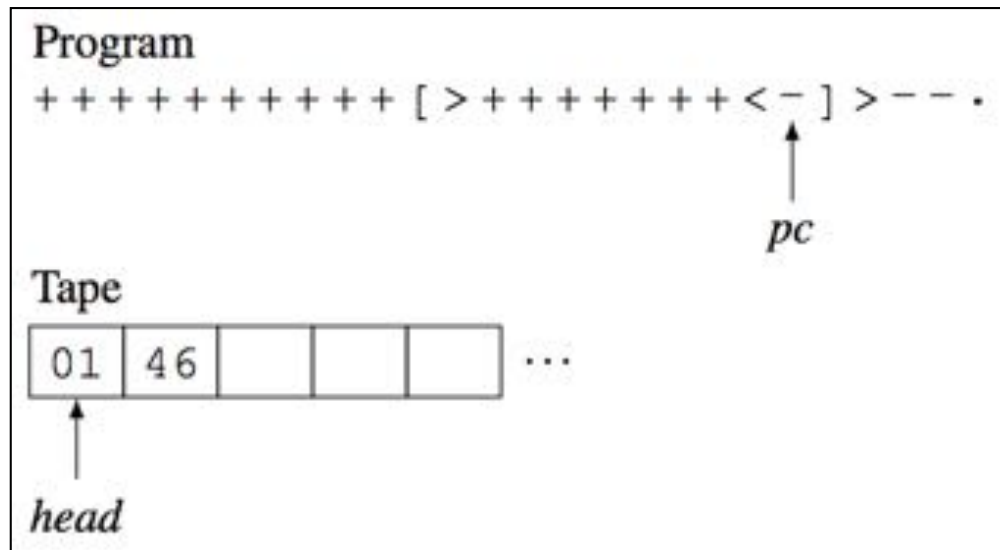
Copy value pointed to into
src/dest of subsequent
control block

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

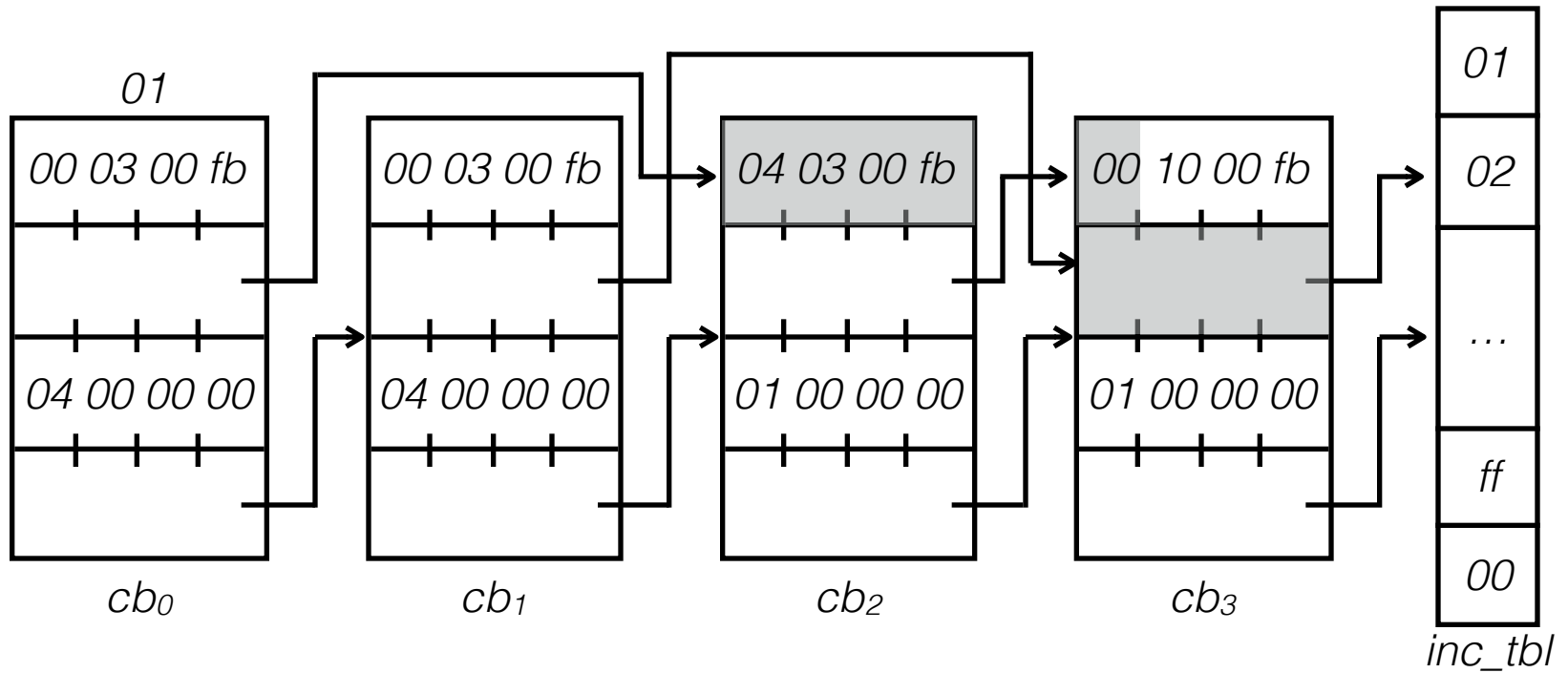
BrainFuck

[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);

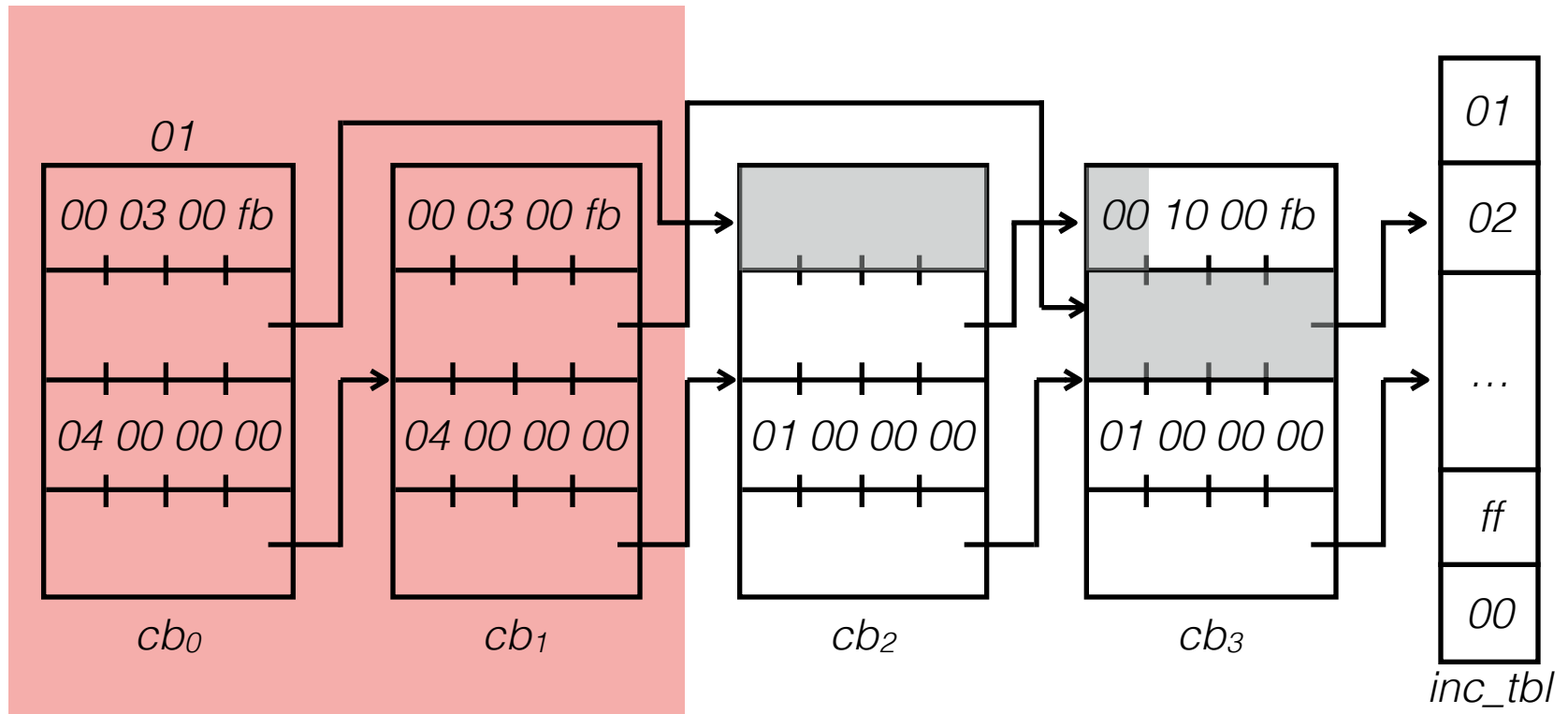
Interpreter Implementation

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

Increment

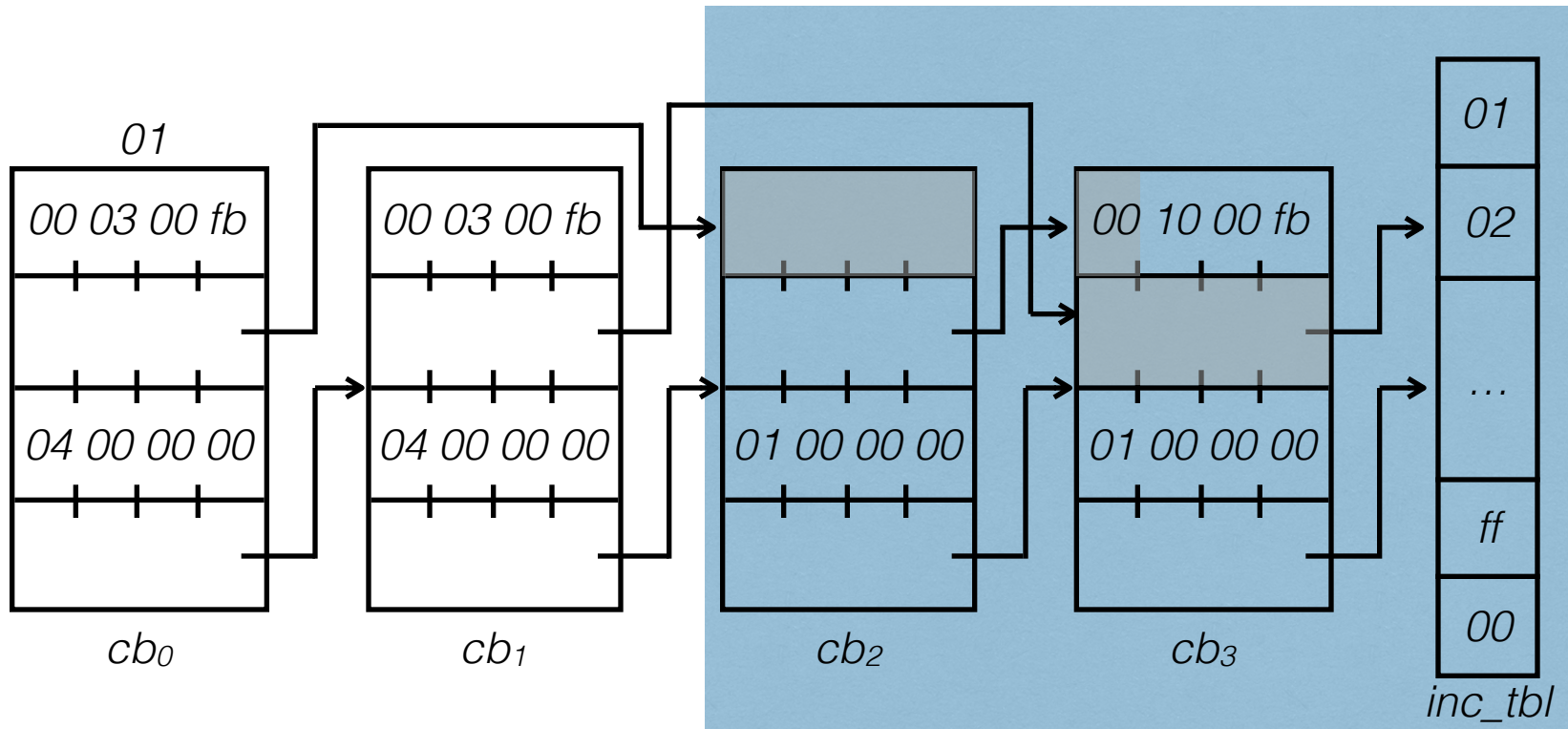


Increment



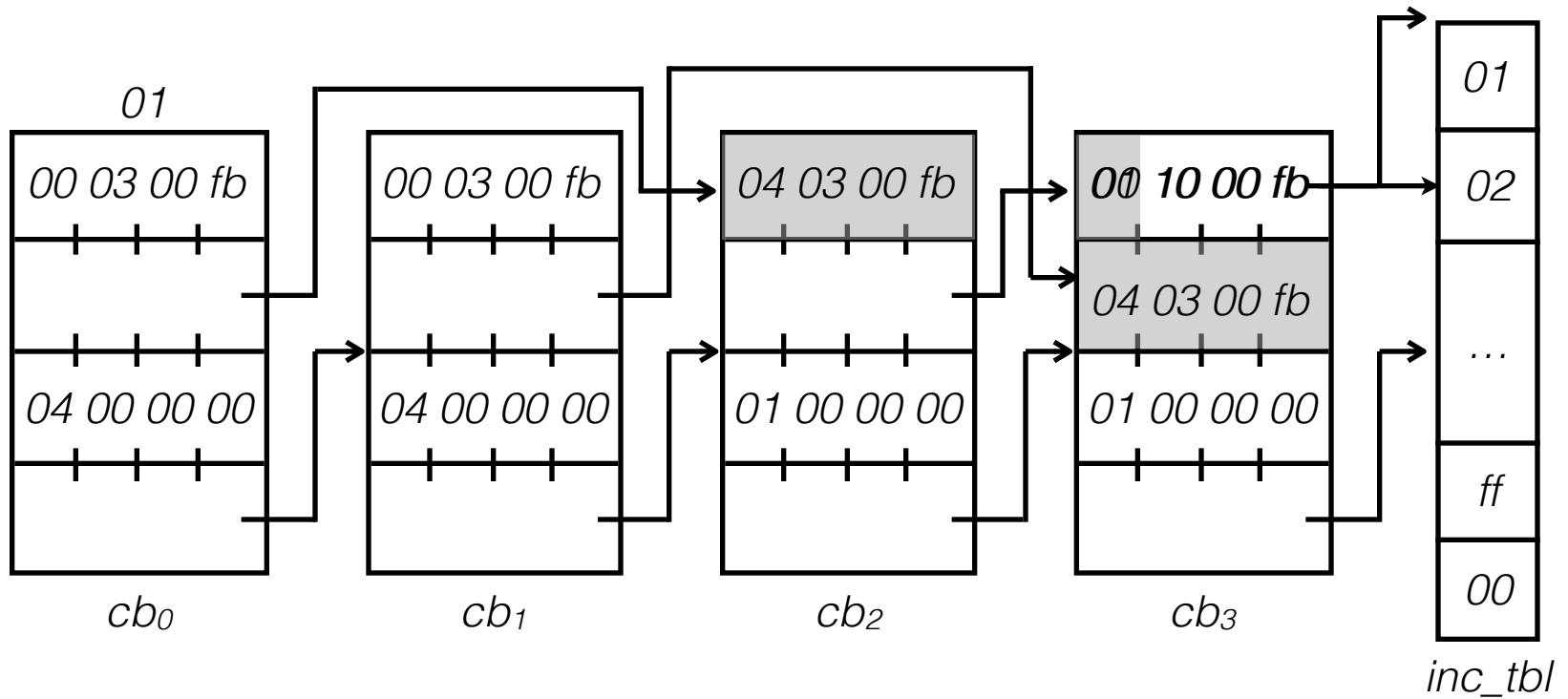
Variable Dereference

Increment

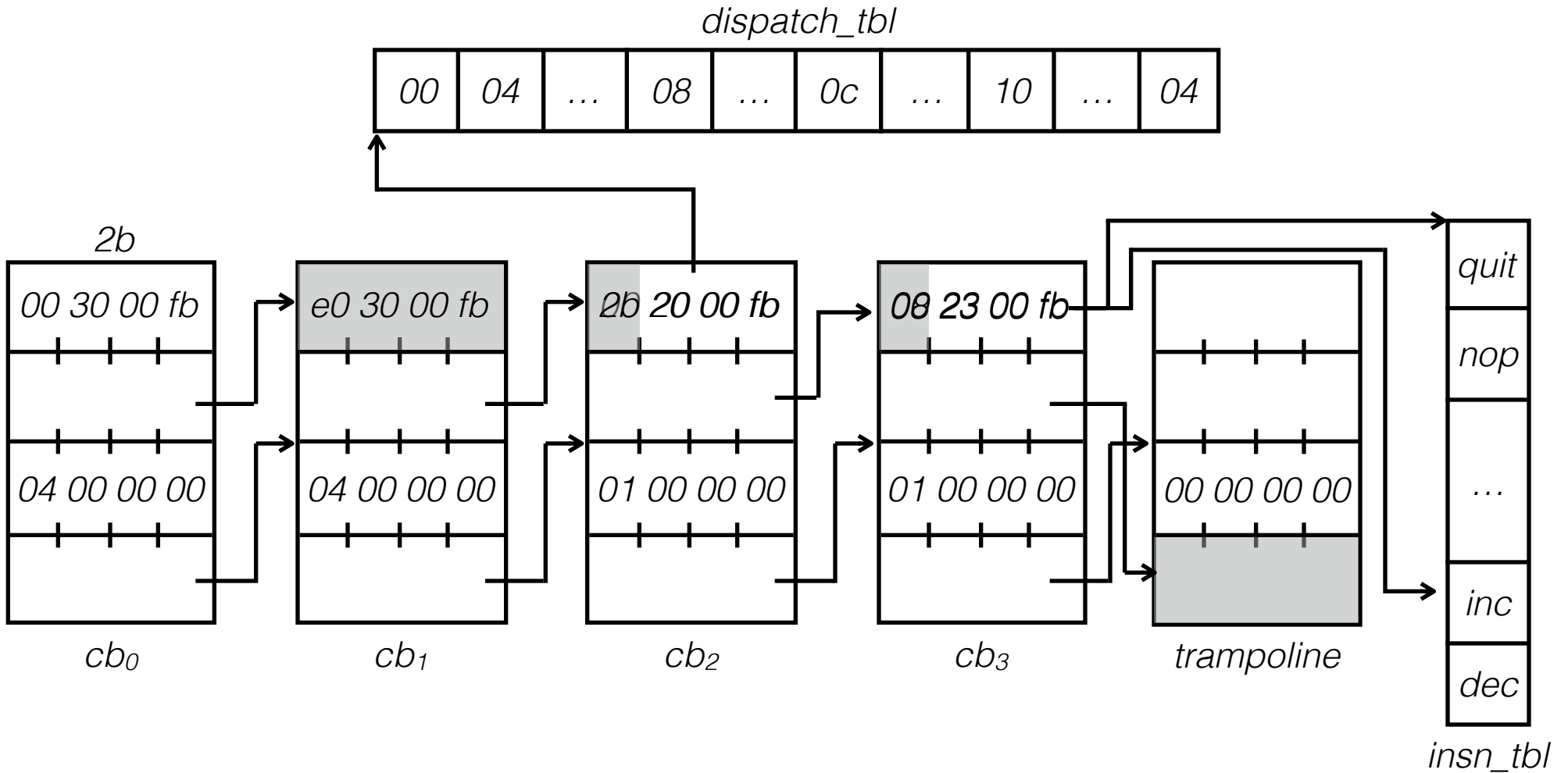


Unary Function

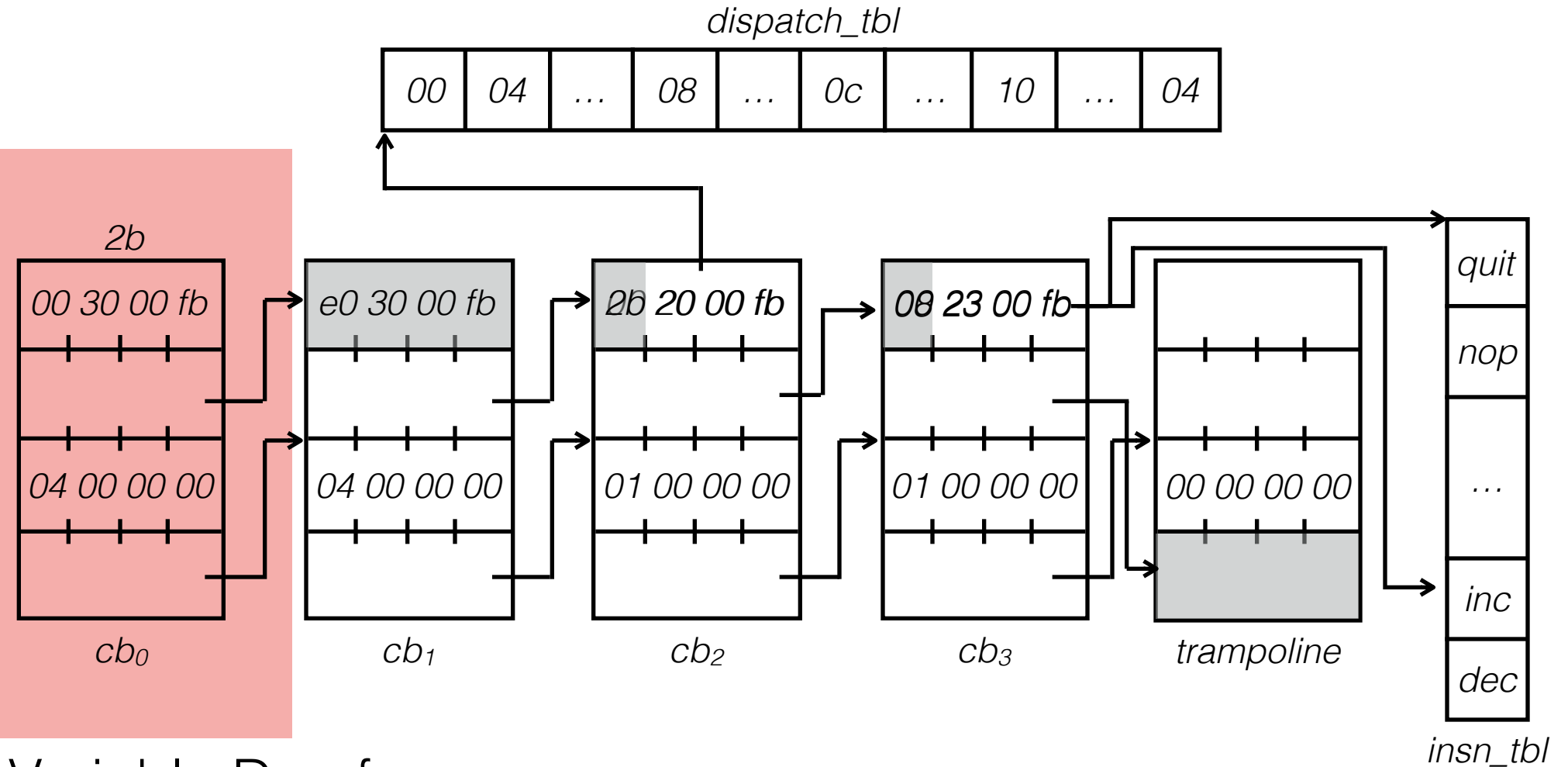
Increment



Dispatch

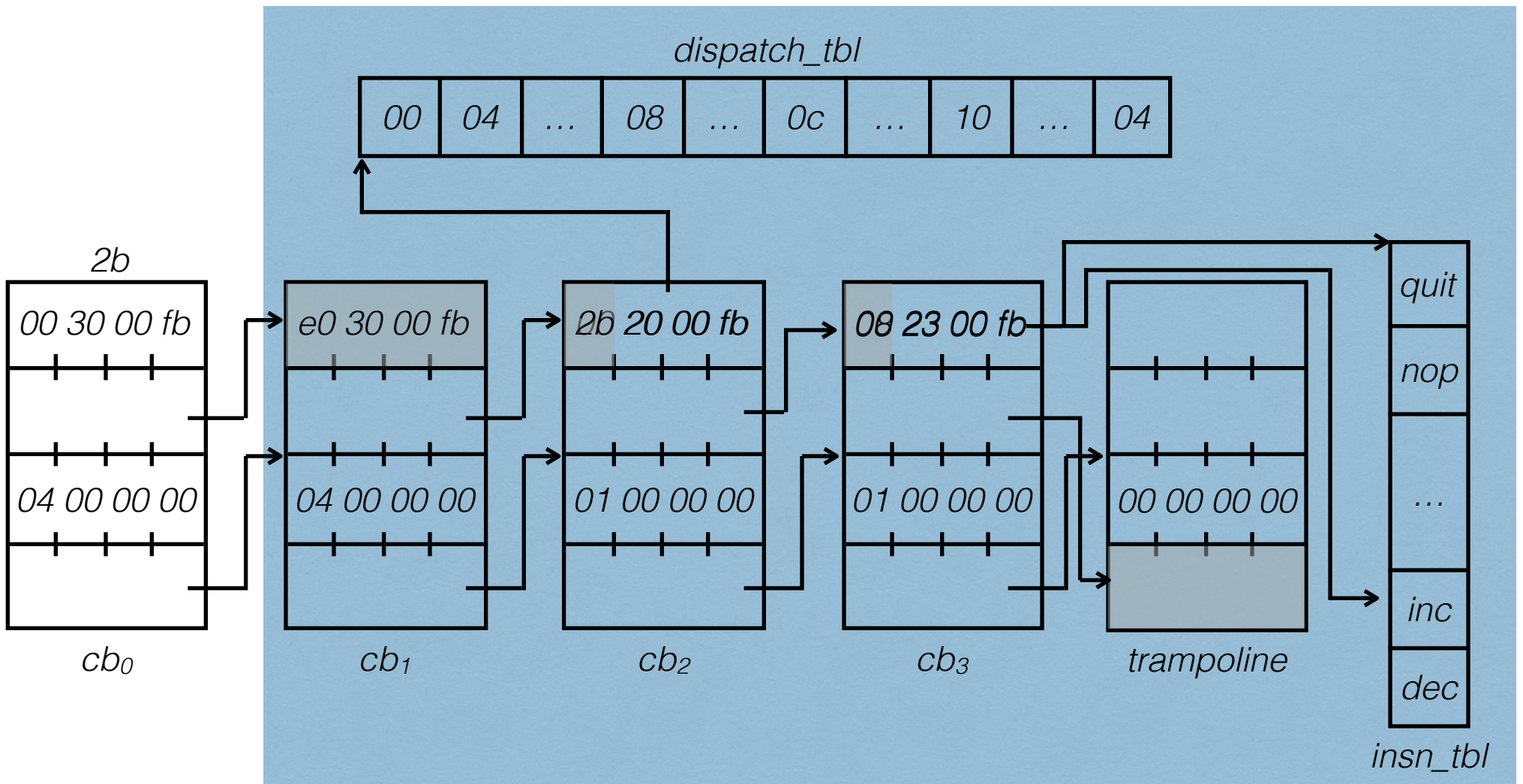


Dispatch



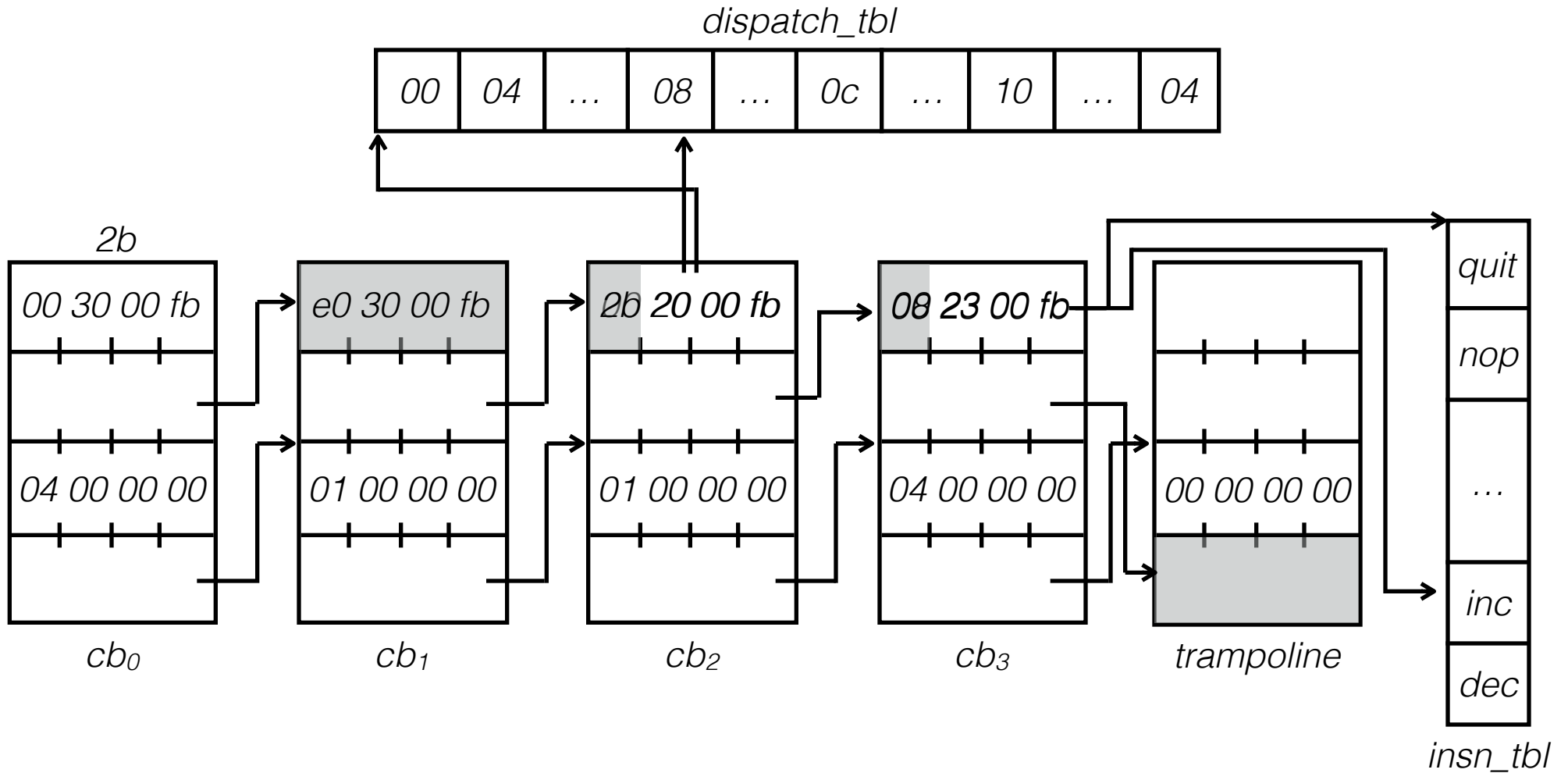
Variable Dereference

Dispatch

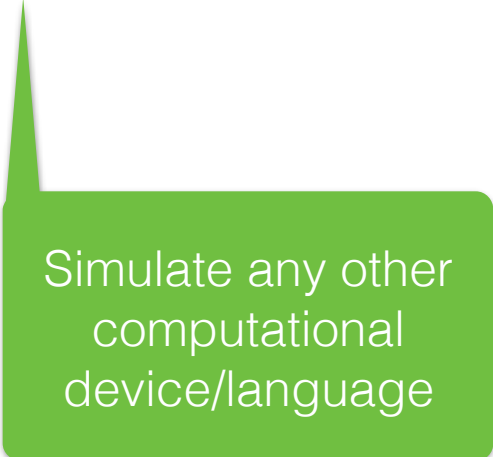


Switch

Dispatch



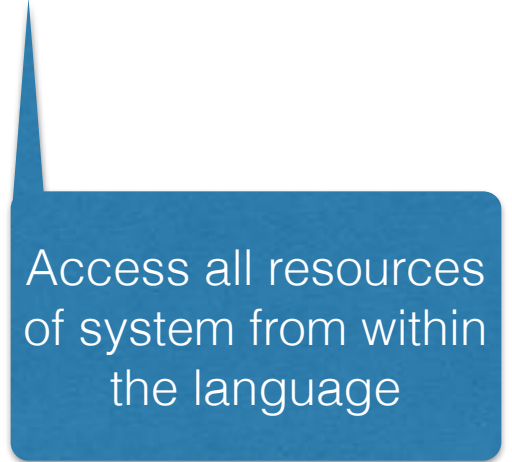
Turing-Complete

A green callout box with a white border and a pointed top-left corner, containing the text "Simulate any other computational device/language".

Simulate any other
computational
device/language

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

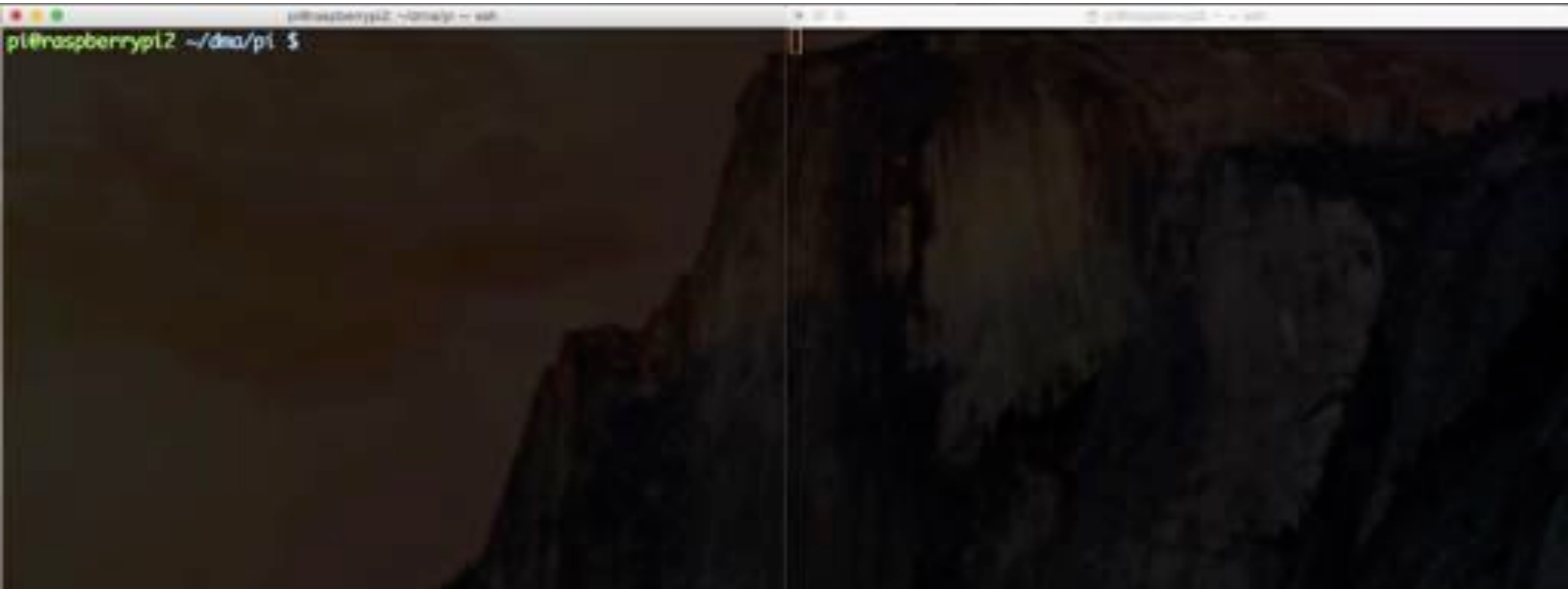
Resource-Complete



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



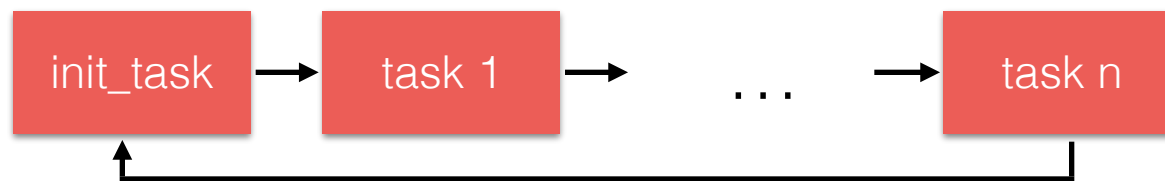
<https://github.com/stevecheckoway/rundma>

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 auxiliary 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 (Dufлот, 2011)
- Peripheral firmware load-time integrity (Stewin, 2012)
- Anomaly detection systems (Dufлот, 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?