The
Culture of Hackers

Arne Köller

26.08.2022

What do you think of when you think of hacking?

“Understanding systems for what they really are, not just what people say they are.”

Two brief talking points

1. 3 Arrays + 1 While-Loop = Computer
2. Hacking then and now

1 3 Arrays + 1 While-Loop = Computer

Q: What happens when you run a program on your computer, smartphone, … ?

Question to the audience: Why would you want to know this in the first place?

My personal answer: Intellectually satisfying + understanding inner workings allows me to be a better programmer

A:

- Multiple things happen (of course)
- Depends on what is meant by “run” (whole program lifecycle?, just entry point?)
- Let’s try answering the entry point question like a hacker!

1 3 Arrays + 1 While-Loop = Computer

test.c

```c
#include <stdio.h>

int main(){

    int a = 5;
    int b = 10;
    int c = a + b;

    printf("%d\n", c);

    return 0;

}
```

“create executable”

$ gcc test.c –o test

test ELF-Binary (executable)

```
0000000000001100 <__do_global_dtors_aux>:
    1100:       f3 0f 1e fa             endbr64
    1104:       80 3d 05 2f 00 00 00    cmpb   $0x0,0x2f05(%rip)        # 4010 <__TMC_END__>
    110b:       75 2b                   jne    1138 <__do_global_dtors_aux+0x38>
    110d:       55                      push   %rbp
    110e:       48 83 3d e2 2e 00 00    cmpq   $0x0,0x2ee2(%rip)        # 3ff8 <__cxa_finalize@GLIBC_2.2.5>
    1115:       00
    1116:       48 89 e5                mov    %rsp,%rbp
    1119:       74 0c                   je     1127 <__do_global_dtors_aux+0x27>
    111b:       48 8b 3d e6 2e 00 00    mov    0x2ee6(%rip),%rdi        # 4008 <__dso_handle>
    1122:       e8 19 ff ff ff          call   1040 <__cxa_finalize@plt>
    1127:       e8 64 ff ff ff          call   1090 <deregister_tm_clones>
    112c:       c6 05 dd 2e 00 00 01    movb   $0x1,0x2edd(%rip)        # 4010 <__TMC_END__>
    1133:       5d                      pop    %rbp
    1134:       c3                      ret
    1135:       0f 1f 00                nopl   (%rax)
    1138:       c3                      ret
    1139:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)

0000000000001140 <frame_dummy>:
    1140:       f3 0f 1e fa             endbr64
    1144:       e9 77 ff ff ff          jmp    10c0 <register_tm_clones>

0000000000001149 <main>:
    1149:       f3 0f 1e fa             endbr64
    114d:       55                      push   %rbp
    114e:       48 89 e5                mov    %rsp,%rbp
    1151:       48 83 ec 10             sub    $0x10,%rsp
    1155:       c7 45 f4 05 00 00 00    movl   $0x5,-0xc(%rbp)
    115c:       c7 45 f8 0a 00 00 00    movl   $0xa,-0x8(%rbp)
    1163:       8b 55 f4                mov    -0xc(%rbp),%edx
    1166:       8b 45 f8                mov    -0x8(%rbp),%eax
    1169:       01 d0                   add    %edx,%eax
    116b:       89 45 fc                mov    %eax,-0x4(%rbp)
    116e:       8b 45 fc                mov    -0x4(%rbp),%eax
    1171:       89 c6                   mov    %eax,%esi
    1173:       48 8d 05 8a 0e 00 00    lea    0xe8a(%rip),%rax        # 2004 <_IO_stdin_used+0x4>
    117a:       48 89 c7                mov    %rax,%rdi
    117d:       b8 00 00 00 00          mov    $0x0,%eax
    1182:       e8 c9 fe ff ff          call   1050 <printf@plt>
    1187:       b8 00 00 00 00          mov    $0x0,%eax
    118c:       c9                      leave
    118d:       c3                      ret

Disassembly of section .fini:

0000000000001190 <_fini>:
    1190:       f3 0f 1e fa             endbr64
    1194:       48 83 ec 08             sub    $0x8,%rsp
    1198:       48 83 c4 08             add    $0x8,%rsp
    119c:       c3                      ret
```

- Programs are usually written in High-Level Languages, Computer executes Machine Code (“Binary”)

1 3 Arrays + 1 While-Loop = Computer

shell:

```
arne@localhost:~/fun$ ./test
```

BACKTRACE

bash, userspace

- ...
- execute_command_internal — Execute the command parsed into COMMAND struct
- execute_simple_command — checks if input is built-in (e.g. echo), calls execute_disk_command if not
- execute_disk_command — forks, connects pipes, command lookup in $PATH, call shell_execve
- shell_execve — calls execve and handles error case
- execve — syscall to kernel: executes the program referred to by parameters (“loader”)

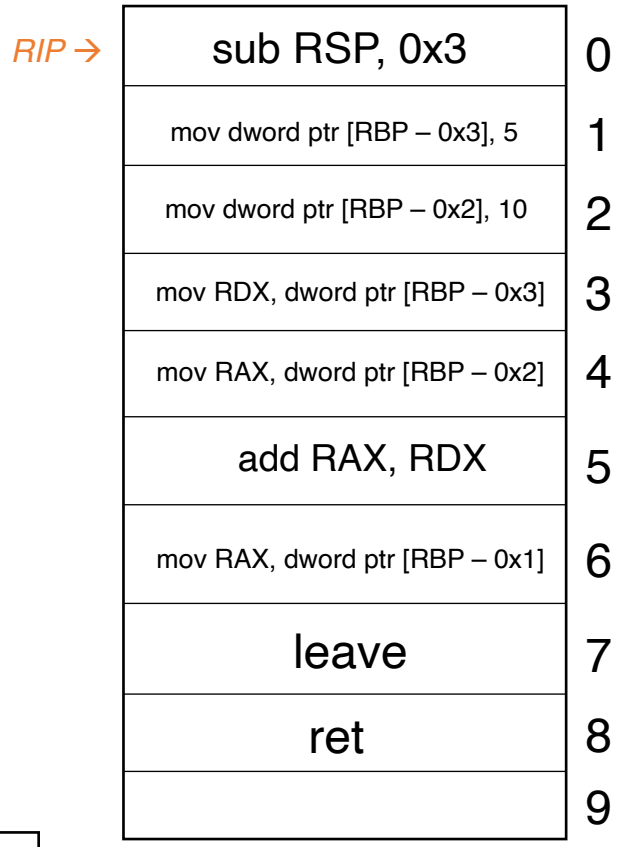
```
arne@localhost:~/fun$ strace test
execve("/usr/bin/test", ["test"], 0x7ffcddaa4740 /* 27 vars */) = 0
brk(NULL)                               = 0x5560fbf38000
```

1 3 Arrays + 1 While-Loop = Computer

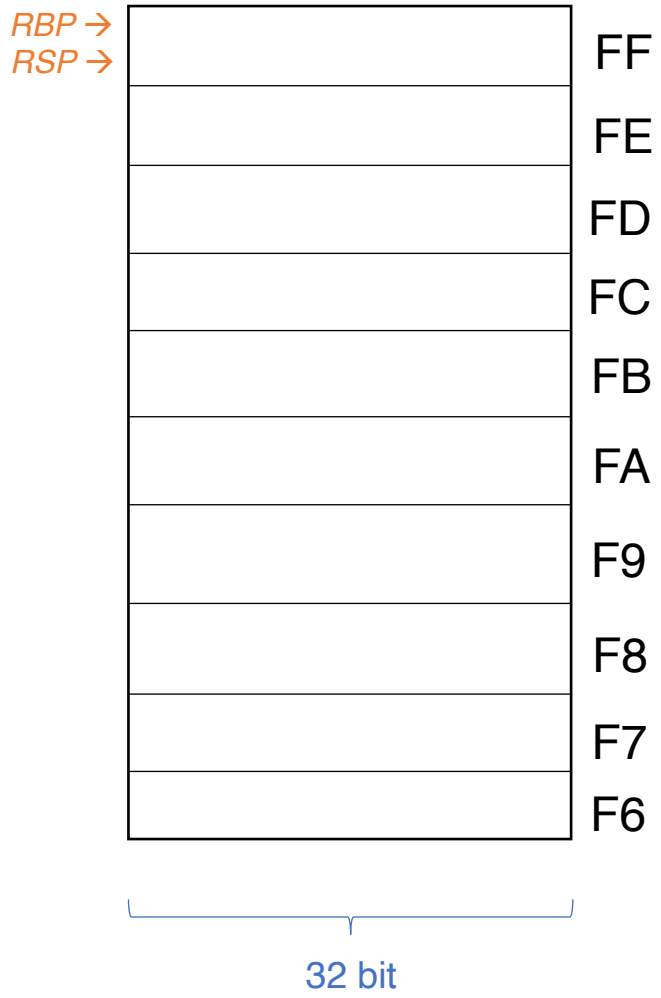
While-Loop

```
while 1:
    opcode = decode(RAM[RIP])
    switch opcode:
        case OP_ADD:  …
        case OP_SUB:  …
        case OP_MOV:  …
        …
    RIP = RIP + 1
```

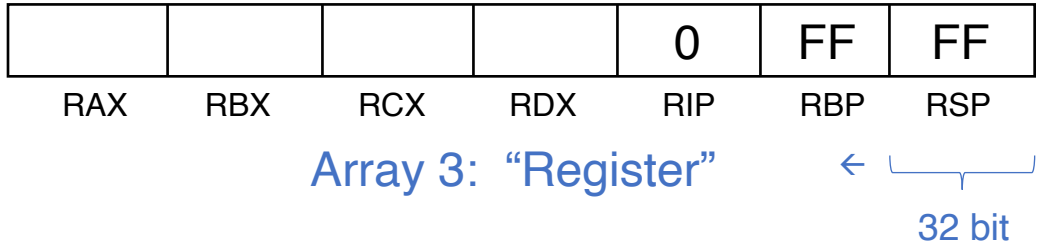
Array 1: "RAM"



Array 2: "Stack"



Array 3: "Register"

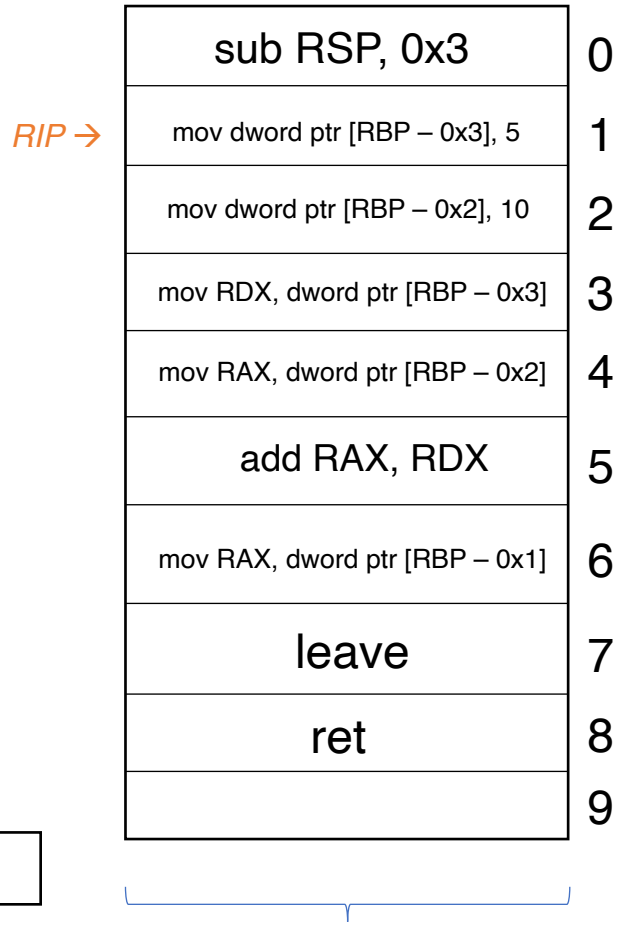


1 3 Arrays + 1 While-Loop = Computer

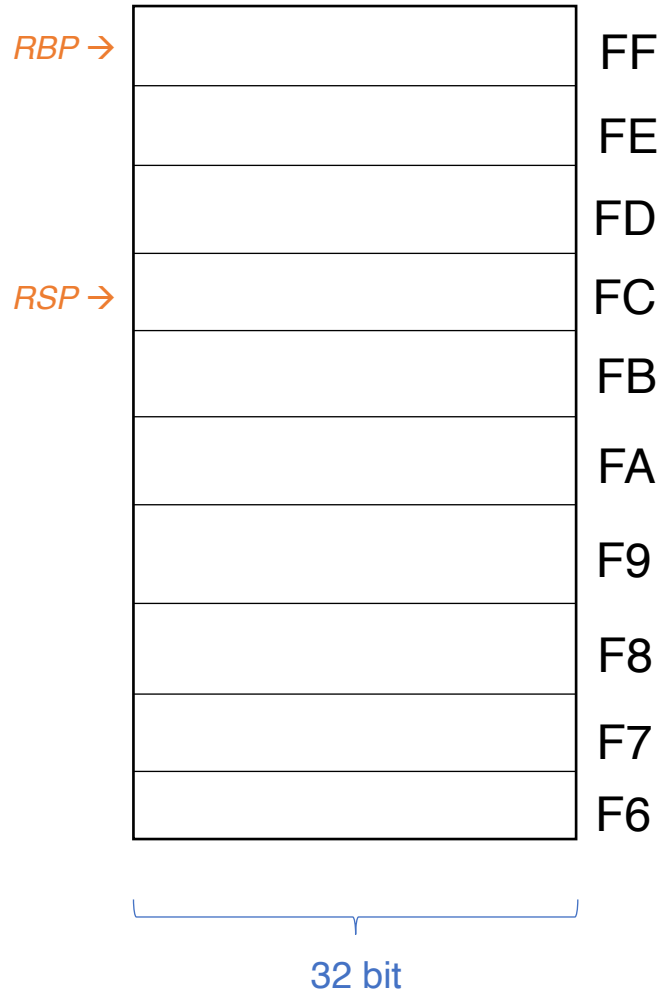
While-Loop

```
while 1:
    opcode = decode(RAM[RIP])
    switch opcode:
        case OP_ADD:  …
        case OP_SUB:  …
        case OP_MOV:  …
        …
    RIP = RIP + 1
```

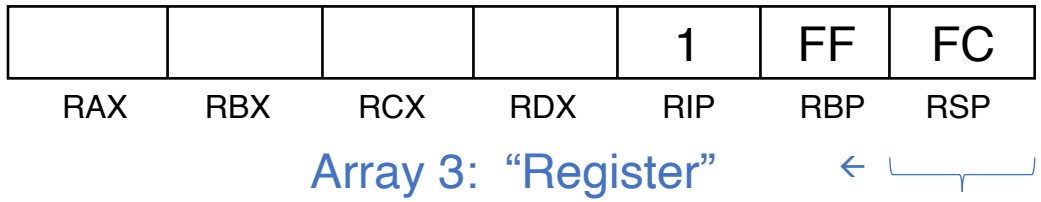
Array 1: "RAM"



Array 2: "Stack"



Array 3: "Register"



1 3 Arrays + 1 While-Loop = Computer

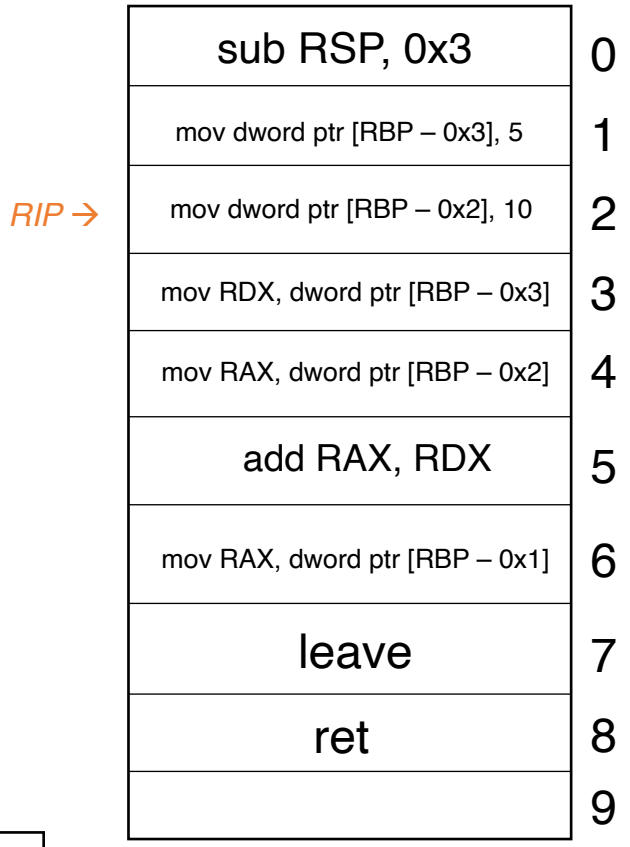
While-Loop

```
while 1:
    opcode = decode(RAM[RIP])
    switch opcode:
        case OP_ADD:  …
        case OP_SUB:  …
        case OP_MOV:  …
        …
    RIP = RIP + 1
```

Array 3: "Register"

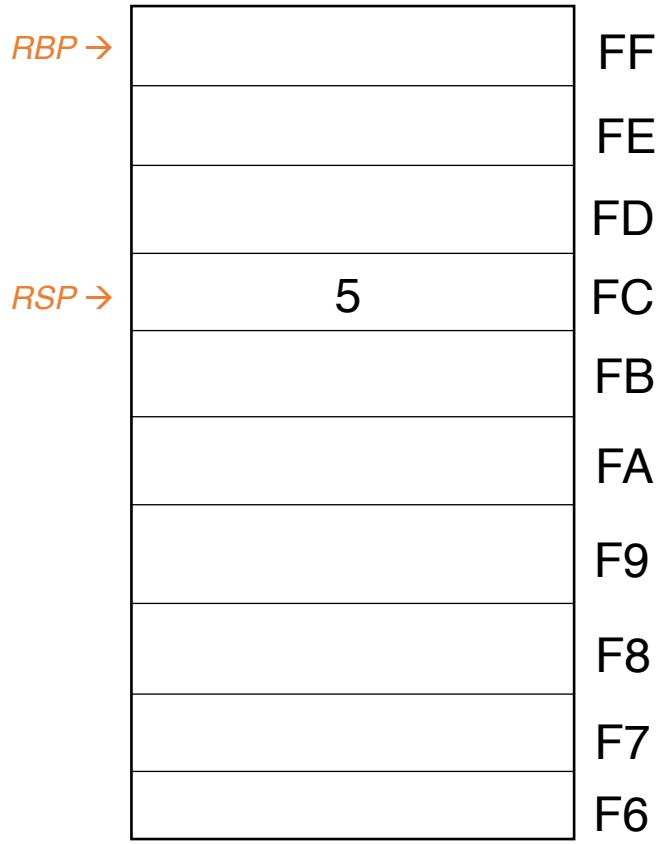
32 bit

Array 1: "RAM"



32 bit

Array 2: "Stack"



32 bit

1 3 Arrays + 1 While-Loop = Computer

While-Loop

```
while 1:
    opcode = decode(RAM[RIP])
    switch opcode:
        case OP_ADD:  …
        case OP_SUB:  …
        case OP_MOV:  …
        …
    RIP = RIP + 1
```

Array 1: "RAM"

	sub RSP, 0x3	0
	mov dword ptr [RBP – 0x3], 5	1
	mov dword ptr [RBP – 0x2], 10	2
RIP →	mov RDX, dword ptr [RBP – 0x3]	3
	mov RAX, dword ptr [RBP – 0x2]	4
	add RAX, RDX	5
	mov RAX, dword ptr [RBP – 0x1]	6
	leave	7
	ret	8
		9

32 bit

Array 2: "Stack"

RBP →		FF
		FE
	10	FD
RSP →	5	FC
		FB
		FA
		F9
		F8
		F7
		F6

32 bit

Array 3: "Register"

RAX	RBX	RCX	RDX	RIP	RBP	RSP
				3	FF	FC

← 32 bit

1 3 Arrays + 1 While-Loop = Computer

While-Loop

```
while 1:
    opcode = decode(RAM[RIP])
    switch opcode:
        case OP_ADD:  …
        case OP_SUB:  …
        case OP_MOV:  …
        …
    RIP = RIP + 1
```

Array 1: "RAM"

	sub RSP, 0x3	0
	mov dword ptr [RBP – 0x3], 5	1
	mov dword ptr [RBP – 0x2], 10	2
	mov RDX, dword ptr [RBP – 0x3]	3
RIP →	mov RAX, dword ptr [RBP – 0x2]	4
	add RAX, RDX	5
	mov RAX, dword ptr [RBP – 0x1]	6
	leave	7
	ret	8
		9

32 bit

Array 2: "Stack"

RBP →		FF
		FE
	10	FD
RSP →	5	FC
		FB
		FA
		F9
		F8
		F7
		F6

32 bit

Array 3: "Register"

RAX	RBX	RCX	RDX	RIP	RBP	RSP
			5	4	FF	FC

← 32 bit

1 3 Arrays + 1 While-Loop = Computer

While-Loop

```
while 1:
    opcode = decode(RAM[RIP])
    switch opcode:
        case OP_ADD:  …
        case OP_SUB:  …
        case OP_MOV:  …
        …
    RIP = RIP + 1
```

10			5	5	FF	FC
RAX	RBX	RCX	RDX	RIP	RBP	RSP

Array 3: "Register"

32 bit

Array 1: "RAM"

	sub RSP, 0x3	0
	mov dword ptr [RBP – 0x3], 5	1
	mov dword ptr [RBP – 0x2], 10	2
	mov RDX, dword ptr [RBP – 0x3]	3
	mov RAX, dword ptr [RBP – 0x2]	4
RIP →	add RAX, RDX	5
	mov RAX, dword ptr [RBP – 0x1]	6
	leave	7
	ret	8
		9

32 bit

Array 2: "Stack"

RBP →		FF
		FE
	10	FD
RSP →	5	FC
		FB
		FA
		F9
		F8
		F7
		F6

32 bit

1 3 Arrays + 1 While-Loop = Computer

While-Loop

```
while 1:
    opcode = decode(RAM[RIP])
    switch opcode:
        case OP_ADD:  …
        case OP_SUB:  …
        case OP_MOV:  …
        …
    RIP = RIP + 1
```

Array 1: "RAM"

	sub RSP, 0x3	0
	mov dword ptr [RBP – 0x3], 5	1
	mov dword ptr [RBP – 0x2], 10	2
	mov RDX, dword ptr [RBP – 0x3]	3
	mov RAX, dword ptr [RBP – 0x2]	4
	add RAX, RDX	5
RIP →	mov RAX, dword ptr [RBP – 0x1]	6
	leave	7
	ret	8
		9

32 bit

Array 2: "Stack"

RBP →		FF
		FE
	10	FD
RSP →	5	FC
		FB
		FA
		F9
		F8
		F7
		F6

32 bit

Array 3: "Register"

RAX	RBX	RCX	RDX	RIP	RBP	RSP
15			5	6	FF	FC

← 32 bit

1 3 Arrays + 1 While-Loop = Computer

While-Loop

```
while 1:
    opcode = decode(RAM[RIP])
    switch opcode:
        case OP_ADD:  …
        case OP_SUB:  …
        case OP_MOV:  …
        …
    RIP = RIP + 1
```

Array 1: "RAM"

	sub RSP, 0x3	0
	mov dword ptr [RBP – 0x3], 5	1
	mov dword ptr [RBP – 0x2], 10	2
	mov RDX, dword ptr [RBP – 0x3]	3
	mov RAX, dword ptr [RBP – 0x2]	4
	add RAX, RDX	5
	mov RAX, dword ptr [RBP – 0x1]	6
RIP →	leave	7
	ret	8
		9

32 bit

Array 2: "Stack"

RBP →		FF
	15	FE
	10	FD
RSP →	5	FC
		FB
		FA
		F9
		F8
		F7
		F6

32 bit

Array 3: "Register"

RAX	RBX	RCX	RDX	RIP	RBP	RSP
15			5	7	FF	FC

← 32 bit

1 3 Arrays + 1 While-Loop = Computer

While-Loop

```
while 1:
    opcode = decode(RAM[RIP])
    switch opcode:
        case OP_ADD:  …
        case OP_SUB:  …
        case OP_MOV:  …
        …
    RIP = RIP + 1
```

Array 1: "RAM"

	sub RSP, 0x3	0
	mov dword ptr [RBP – 0x3], 5	1
	mov dword ptr [RBP – 0x2], 10	2
	mov RDX, dword ptr [RBP – 0x3]	3
	mov RAX, dword ptr [RBP – 0x2]	4
	add RAX, RDX	5
	mov RAX, dword ptr [RBP – 0x1]	6
	leave	7
RIP →	ret	8
		9

32 bit

Array 2: "Stack"

RBP →		FF
	15	FE
	10	FD
RSP →	5	FC
		FB
		FA
		F9
		F8
		F7
		F6

32 bit

Array 3: "Register"

RAX	RBX	RCX	RDX	RIP	RBP	RSP
15			5	8	FF	FC

← 32 bit

1 3 Arrays + 1 While-Loop = Computer

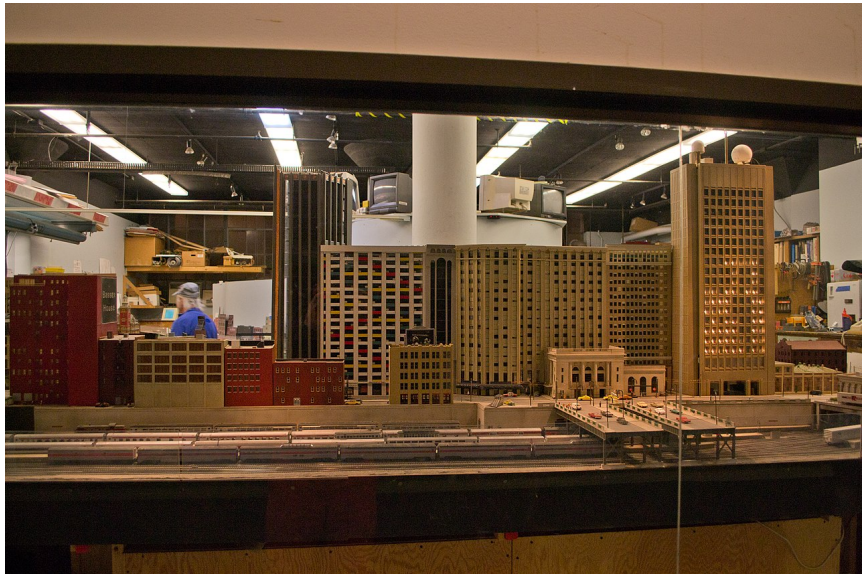
1 3 Arrays + 1 While-Loop = Computer

- Hacking something often starts by Reverse-Engineering existing systems
- Reverse-Engineering can be used to gain intimate knowledge of an existing system
- Intimate knowledge will allow to understand system's design decisions and limitations
- Understanding will allow to manipulate/improve system or to create a better system

Hacking: Being highly competent (theory + hands-on) in a certain discipline by deeply understanding what's going on

Not Hacking: Superficially learning XY because you wish to exploit a system (technical or social)

2 Hacking now and then



Model des TMRC, Cambridge, MA, 1960s

Homebrew Computer Club, Menlo Park, CA,1978

2 Hacking now and then

Steven Levy, 1984, “Hacker Ethics”:

- Access to information should be unlimited and total
- All information should be free
- Mistrust authority - promote decentralization
- Hackers should be judged by their hacking, not bogus criteria such as degrees, age, race, or position

2 Hacking now and then

The iPhone Wiki

Main page | Discussion | Read | View source | View history | Search The iPhone Wiki

Main page
Community portal
Current events
Recent changes
Random page
Help
Miscellaneous
Ground rules
Timeline
Tools
What links here
Related changes
Special pages
Printable version
Permanent link
Page information

The iPhone Wiki

Welcome to the iPhone Wiki

This is a conglomerate work of everything done by everyone on Apple's amazing iDevices. *Anyone* can contribute here, just ask an administrator for an account. Currently there are 4,560 users, with 3,275 articles (and 14,061 key pages).

- Check out the most recent article changes.
- Get up to speed in the community (and learn about how jailbreaks work).
- Read (and edit) the constitution to understand what purpose this wiki serves.
- Read the timeline to see where we are.
- Read the community portal to find out what people request to be added/changed.
- Read the ground rules to know what you should and should not post in this wiki.
- If you have notes on something you did, post them here, no matter how ugly.
- If you see something ugly, work to make it pretty.
- If you have a fix for a problem people are having, post it here.

What are we about?

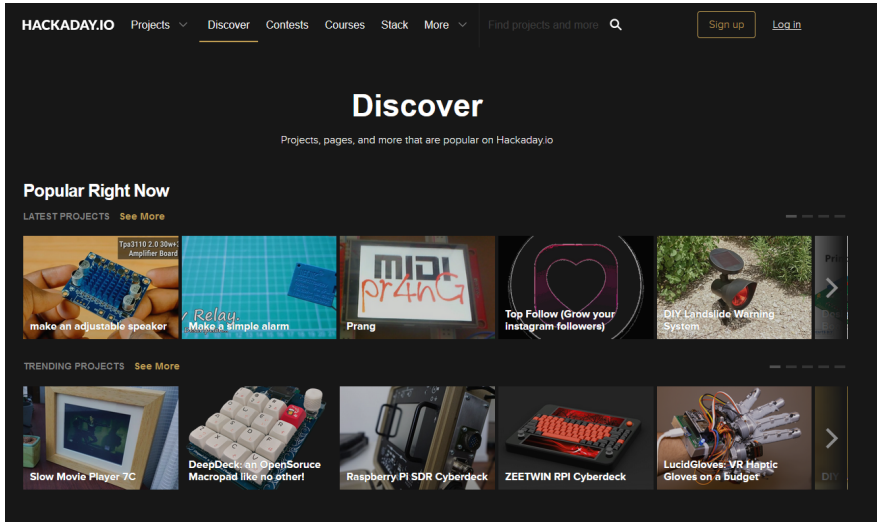
The iPhone Wiki is an unofficial wiki dedicated to collecting, storing and providing information on the internals of Apple's amazing iDevices. We hope to pass this information on to the next generation of hackers so that they can go forth into their forebears' footsteps and break the ridiculous bonds Apple has put on their amazing mobile devices.

Firmware Status

Product Line	Apple TV	HomePod	Apple Watch		iPad	iPad Air		iPad Pro	iPad mini		iPhone		iPod touch
Supported	Apple TV HD and newer	All models	Series 3	Series 4 and newer	5th generation and newer	iPad Air 2	3rd generation and newer	All models	iPad mini 4	5th generation and newer	6s / 6s Plus 7 / 7 Plus SE (1st generation)	iPhone 8 and newer	7th generation
Latest Public Firmwares	15.6 (19M65)		8.7.1 (19U67)	8.7 (19U66)	15.6.1 (19G82)								
Jailbreak availability	No												
Latest Beta Firmwares	16.0 beta 7 (20J5371a)		N/A	9.0 beta 7 (20R5359a)	16.1 beta (20B5027f)	N/A	16.1 beta (20B5027f)		N/A	16.1 beta (20B5027f)	N/A	16.0 beta 7 (20A5356a)	N/A

See Jailbreak for a complete list of devices and firmware versions and tools used to jailbreak those versions.

The iPhone Wiki, a community dedicated to reverse-engineering Apple products, 2007 - today



HACKADAY.IO Projects | Discover | Contests | Courses | Stack | More | Find projects and more | Sign up | Log in

Discover

Projects, pages, and more that are popular on Hackaday.io

Popular Right Now

LATEST PROJECTS See More

make an adjustable speaker | Make a simple alarm | Prang | Top Follow (Grow your instagram followers) | DIY Landslide Warning System

TRENDING PROJECTS See More

Slow Movie Player 7C | DeepDeck: an OpenSoruce Macropad like no other! | Raspberry Pi SDR Cyberdeck | ZEETWIN RPI Cyberdeck | LucidGloves: VR Haptic Gloves on a budget

hackaday.io, a community dedicated building open-source hardware products, 2022

2 Hacking now and then

Maker’s Fair in Rome, 2019

Any competent engineer, who is interested in figuring out the inner workings of a system, 200k years ago - today

Thank you for your attention!

Questions?