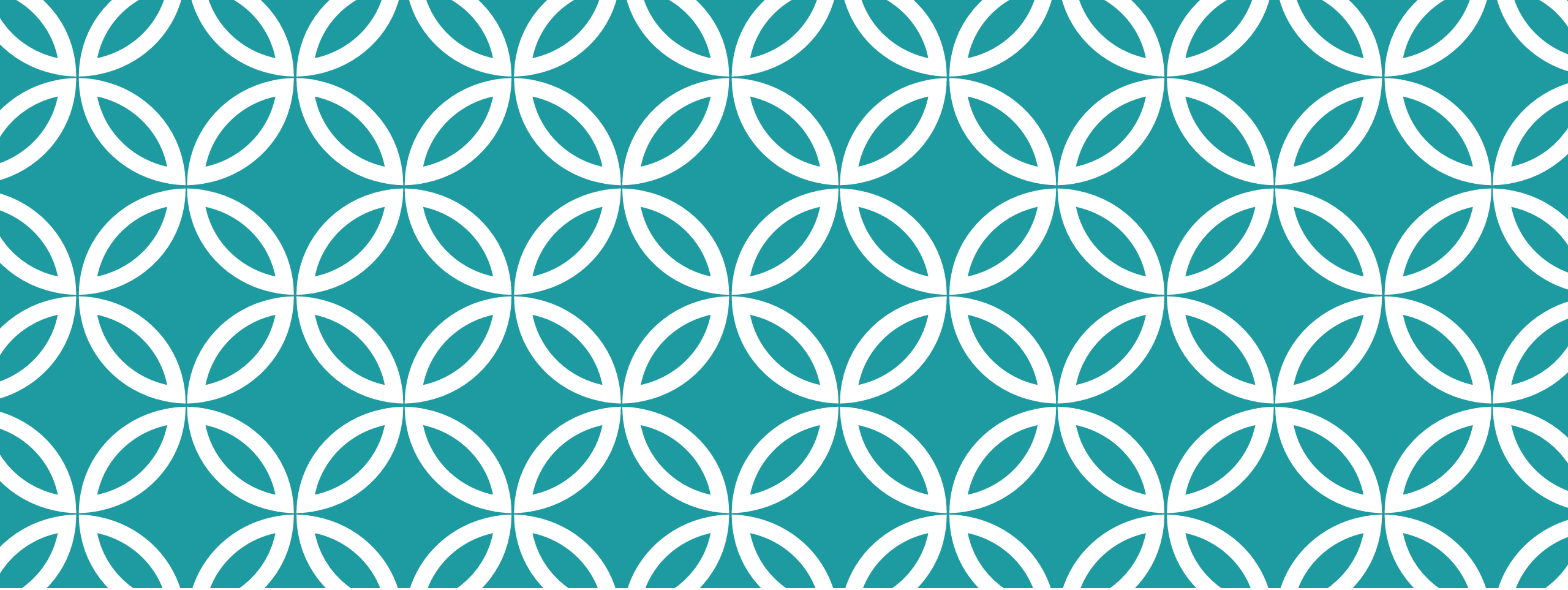
0X1F4A9

Intel X86 ASM Codegolf

INHALT

- Vorstellung
- Worum geht es?
- Inspiration
- Assembler Crashkurs
- Relevanz
- Optimierungsbeispiele
- Lösung
- Links



WORUM GEHT ES?

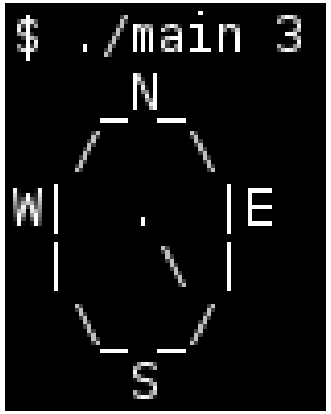
Intel X86 ASM Codegolf

CODE GOLF

- „A recreational computer programming competition“
- Ziel: Möglichst kleines Binary
- Input/Output ist vorgegeben
- Compiler Optionen sind vorgegeben

AUFGABE

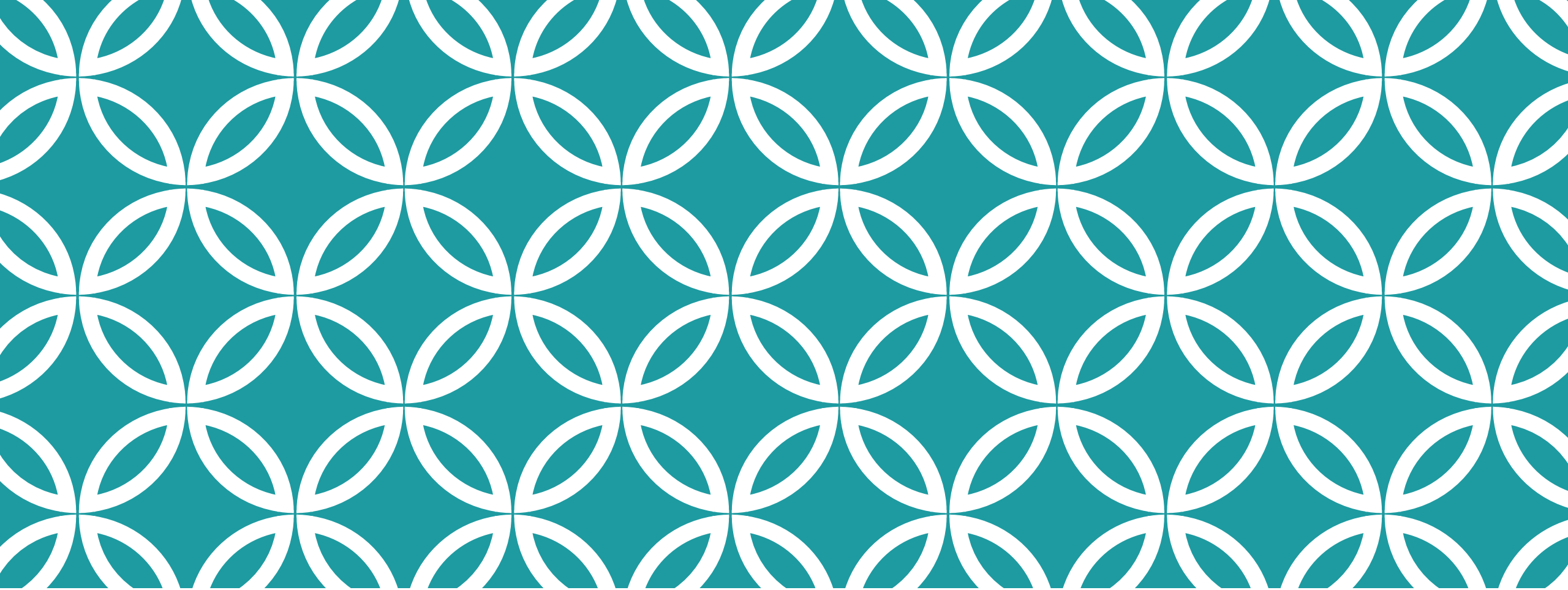
- ASCII-Kompass ausgeben
- Intel X86 Assembly
- Himmelsrichtung ist abhängig von Eingabe auf argv[1] (Wertebereich 0-7)
- Keine Libraries verwenden (printf…), nur Syscalls
- Makefile zum Kompilieren mit NASM steht zur Verfügung
- Testscript zum Verifizieren der Lösung steht zur Verfügung



ASSEMBLE – TEST – REPEAT

1. Mit der einfachsten funktionierenden Lösung beginnen
2. Etwas ändern
3. Assemblieren, Grösse überprüfen
4. JMP 1

```
$ vim main.s
$ make
nasm -f elf32 -O0 main.s
ld -m elf_i386 -s -O0 -o main
main.o
$ python test.py
Success!
Binary size is 460 bytes.
```



INSPIRATION

Intel X86 ASM Codegolf

REAL MEN KNOW ASSEMBLY

"…back when men were men and wrote their own device drivers…"

- Linus Torvalds

REAL MEN KNOW ASSEMBLY

assembly code

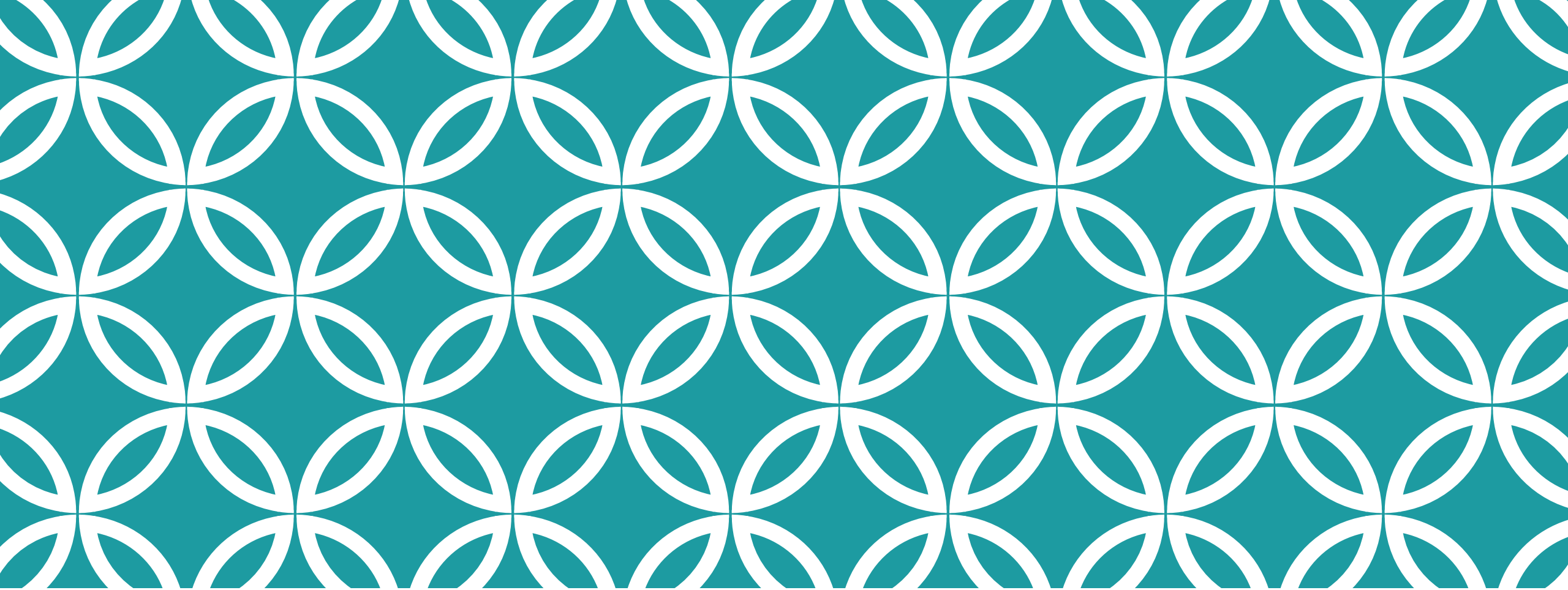
"…back when men were men and wrote ~~their own device drivers~~…"

- ~~Linus Torvalds~~

0x1F4A9

INSPIRATION

- The Art of Programming: «real men know assembly»
- Vollständige Kontrolle der CPU/HW
- Bufferoverflows / Shellcodes @ InfSi 2
- Computer Engineering 1 @ HSR
- Mikrocontroller Projekte / Hackerspace: www.coredump.ch



ASSEMBLER CRASHKURS

Intel X86 ASM Codegolf

ASSEMBLER

x86 CPU Register:

- General Purpose: EAX, EBX, ECX, EDX, ESI, EDI
- Stack Manipulation: ESP, EBP

Befehle:

- MOV eax, 0x42 // Zahl ins Register laden
- INT 0x80 // Interrupt vektor 0x80 auslösen
- ADD eax, 10 // Addition von 10 zu eax. Resultat in eax
- SUB eax, 10 // Subtraktion von 10 von eax. Resultat in eax
- INC eax // Increment (++eax)
- PUSH eax // Push to Stack
- POP eax // Pop from Stack
- AND, XOR, OR…

SYSCALLS

Nutzen des Kernels

- Nummer des Syscalls in eax register laden
- Argumente in weitere Register laden
- Interrupt aufrufen (INT 0x80)

Linux system calls:

%eax	*Name*	*%ebx*	*%ecx*	*%edx*	
1	sys_exit	int	-	-	-
3	sys_read	unsigned int	char *	size_t	-
4	sys_write	unsigned int	const char *		

http://www.digilife.be/quickreferences/qrc/linux%20system%20call%20quick%20reference.pdf

$: man 2 intro

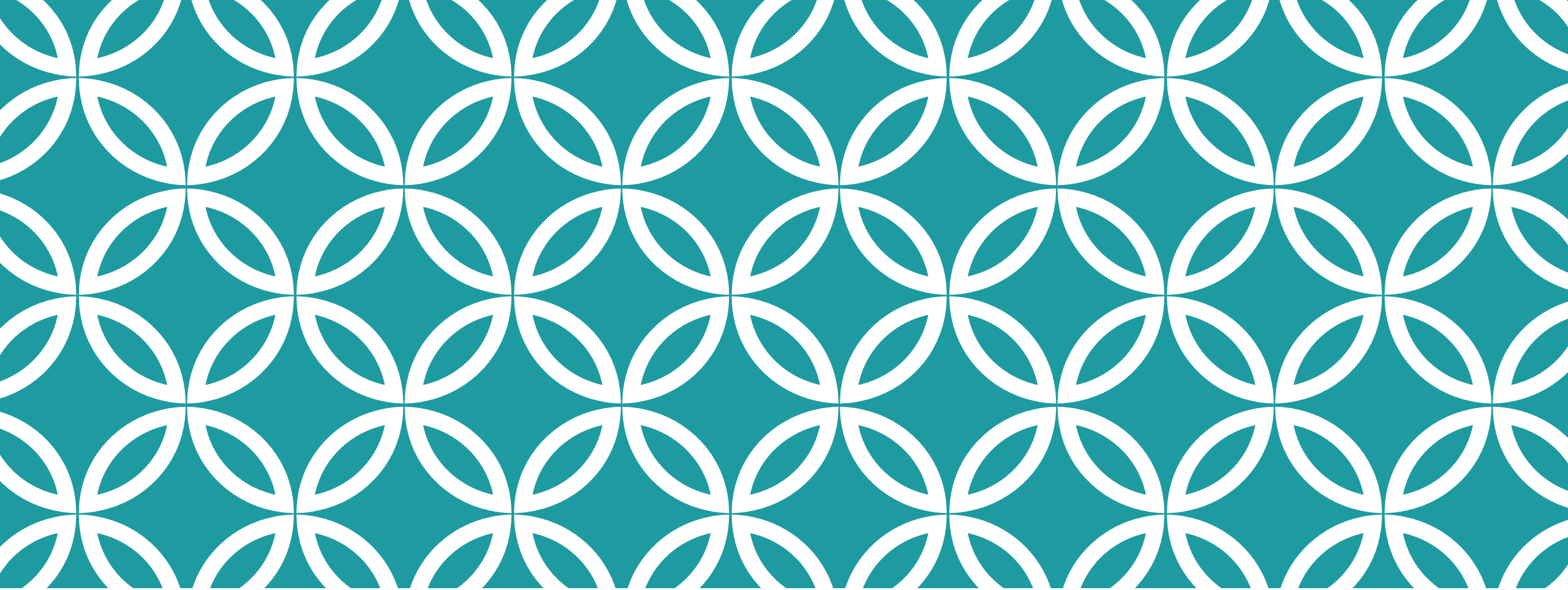
HELLO WORLD: WRITE()

C:

```
ssize_t write(int fd, const void *buf, size_t count);
```

Assembly:

```
MOV   edx, 0x04          // Länge des String laden
MOV   ecx, msg           // char * in ecx laden
MOV   ebx, 0x01          // Filedeskriptor laden (1: stdout)
MOV   eax, 0x01          // Nummer des syscalls laden (sys_write)
INT   0x80               // Kernelaufruf
```



RELEVANZ

Intel X86 ASM Codegolf

RELEVANZ?

Optimierung:

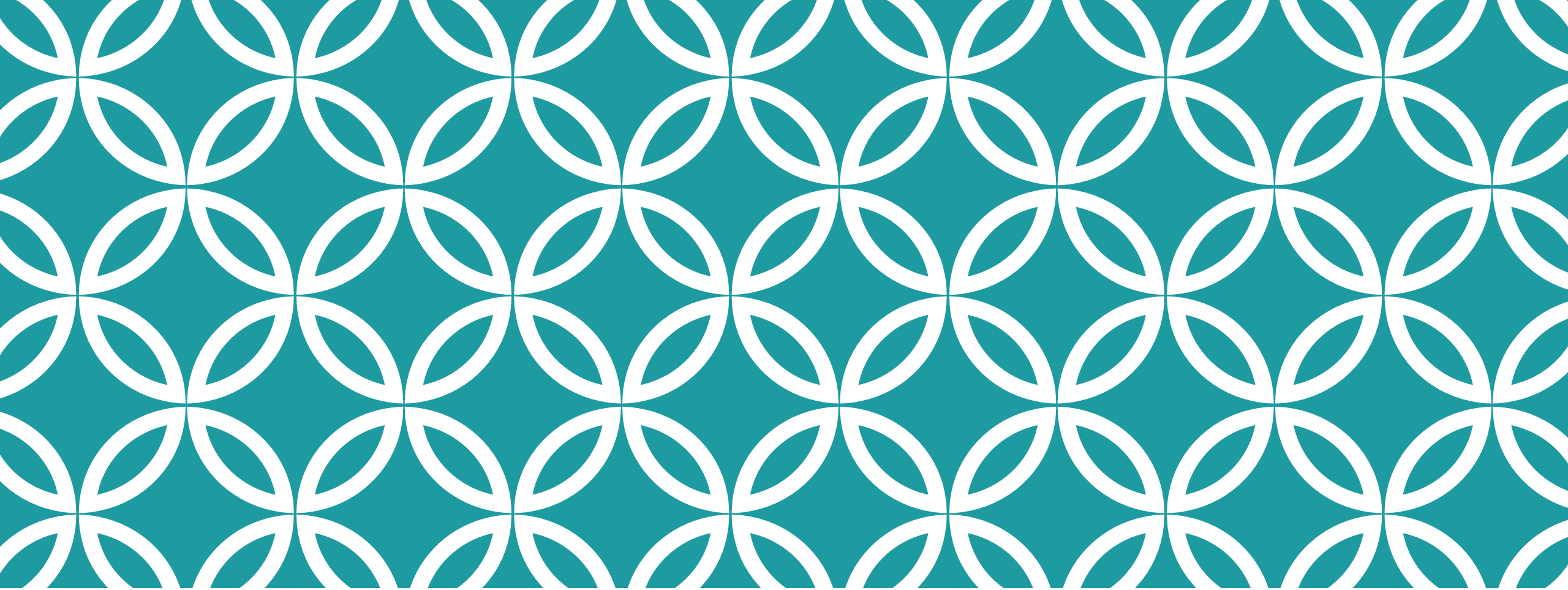
- Speed, AES-NI Verschlüsselung Speedup 10x
- Codegrösse (embedded)
- Realtime (embedded / Anzahl Instruktionszyklen)

Gut zu Wissen

- Compilerbau (z.B ++i, i++)

Hacking

- Smashing the Stack / Shellcode http://shell-storm.org/shellcode/
- Reverse Engineering (Firmware, Treiber, etc.)



OPTIMIERUNGSBEISPIELE

Intel X86 ASM Codegolf

DEBUGGING?

GDB is your friend!

GDB

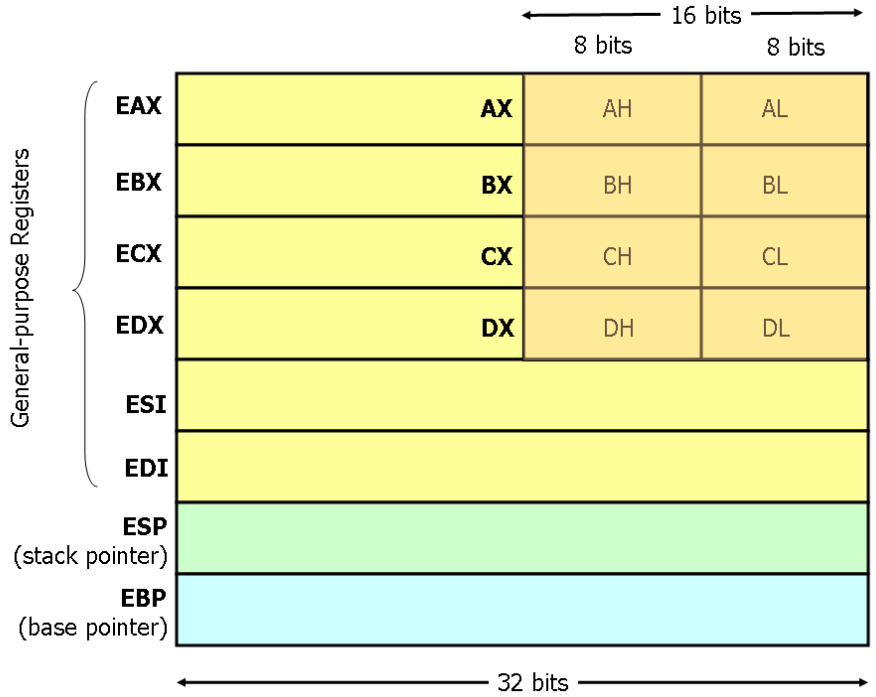
Befehl	Beispiel	Bedeutung
`break <addr>`	`break _start`	Breakpoint setzen
`run <param>`	`run 5`	Programm mit Parameter starten
`nexti` `ni`	`ni`	Nächste Instruktion
`info registers` `i r`	`i r`	Inhalte der CPU Register anzeigen (EAX, ESP…)
`examine /<num><fmt> <len> <addr>` `x /<num><fmt> <len> <addr>`	`x/4xb 0xCOFFEE00`	4 Bytes an Memory Adresse anzeigen
`disassemble` `disass`	`disass`	Nächste Instruktionen anzeigen
`help`	`help x`	Hilfe anzeigen

OPTIMIERUNG: STRATEGIEN

- Ansatz überdenken
  - Subroutinen und JMP vs Arithmetik
  - Hardcoden vs Berechnen
  - Redundanz vs Kompression
- Weniger Instruktionen
- Kleinere Instruktionen

REGISTERWAHL

- - EAX vs AX vs AL
  - AL ist das „Lower Byte“ von EAX
- EAX vs EBX vs ECX vs ...
  - EAX ist der Akkumulator, hat besonders kurze Arithmetik-Instruktionen



The Art of Picking Intel Registers: http://www.swansontec.com/sregisters.html

WIEDERVERWENDEN VON DATEN

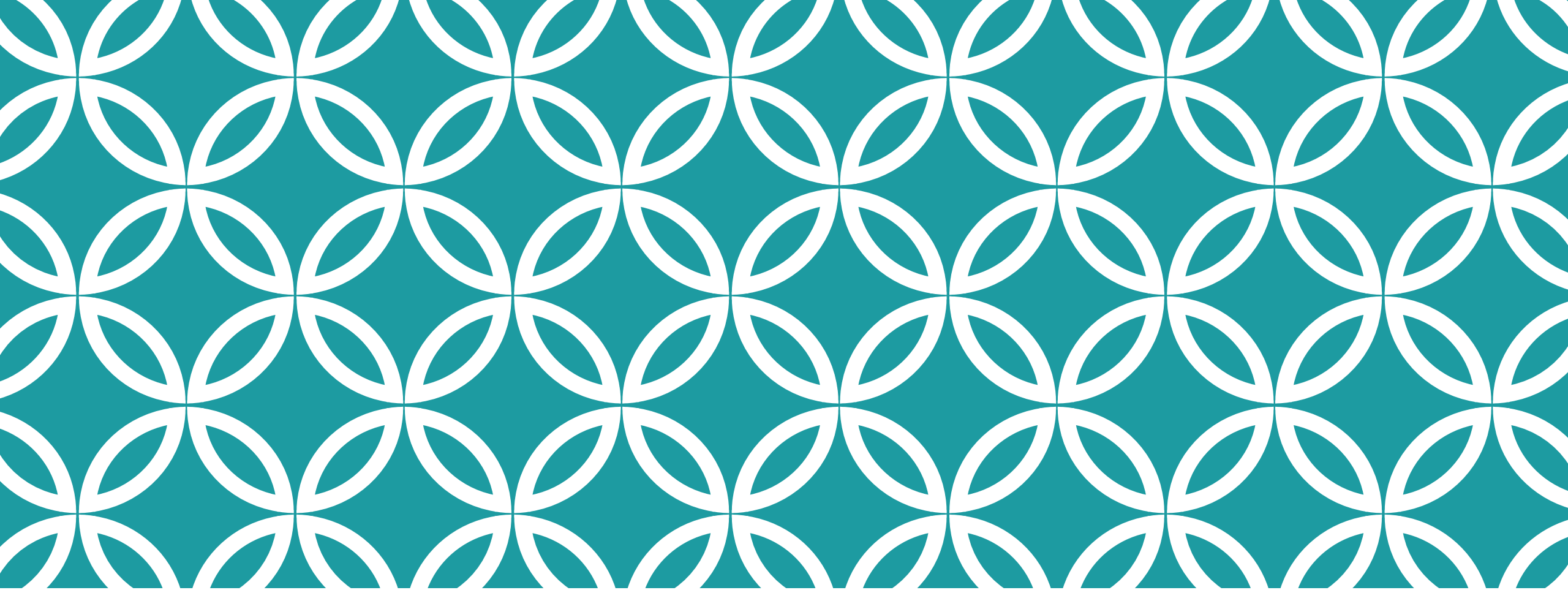
```
write_compass:
    mov     eax, 4                  ; Syscall: sys_write
    mov     ebx, 1                  ; File descriptor (stdout)
    mov     ecx, compass            ; Pointer to string
    mov     edx, len                ; Length of string
    int     0x80                    ; Invoke syscall

exit:
    mov     eax, 1                  ; Syscall: sys_exit
    dec     ebx                     ; Exit code 0
    int     0x80                    ; Invoke syscall
```

KÜRZERE INSTRUKTIONEN VERWENDEN

- ADD EBX, 1
  - 8 Byte
- INC EBX
  - 4 Byte

- MOV EAX, 0
  - 8 Byte
- XOR EAX, EAX
  - 4 Byte



LÖSUNG

Intel X86 ASM Codegolf

LÖSUNG

- … verraten wir natürlich nicht ;)
- Challenge an euch
- Unser aktueller Stand: 452 Bytes.
  Beat that!
- Vorläufige Aufgabenstellung:
  https://github.com/dbrgn/asm-codegolf

LINKS

- x86 Einführung http://www.cs.virginia.edu/~evans/cs216/guides/x86.html
- x86 Instruktions-Referenz: http://faydoc.tripod.com/cpu/index.htm
- Noch eine Referenz: http://ref.x86asm.net/
- The Art of Picking Intel Registers http://www.swansontec.com/sregisters.html
- Kurze x86 Instruktionen: http://www.xxeo.com/single-byte-or-small-x86-opcodes

VIELEN DANK FÜR DIE AUFMERKSAMKEIT

Habt ihr Fragen?