

Macoun



ARM-Assembler für iOS-Entwickler

Klaus M. Rodewig
@cocoanehead

Referent

- Klugscheisser :)
- Autor
 - ***hint*** Auflage 3 erscheint im Dezember ***hint***



Is -la



Grundlagen

ARM

- ARM == „Advanced RISC Machines“
- CPU in iPhone, iPad, iPod Touch (uvm.)
- Reduced Instruction Set Computer
 - Complex Instruction Set Computer (Intel x86)
- 64 Bit ab iPhone 5S

Today: 32 Bit only!

Abgrenzung

- x86-Assembler
 - komplexer Befehlssatz (CISC)
 - Special Purpose Register
 - Offset-Hölle Realmode vs. Protected Mode
- iPhone-Simulator

Programmiersprachen

- Programmiersprache der **1. Generation**
 - Maschinensprache
- Programmiersprache der **2. Generation**
 - Assembler
- Programmiersprache der **3. Generation**
 - Objective-C, C, uvm.



Vom Code zur Binärdatei

3. Generation

```
[foo initWithBar:23];
```

2. Generation

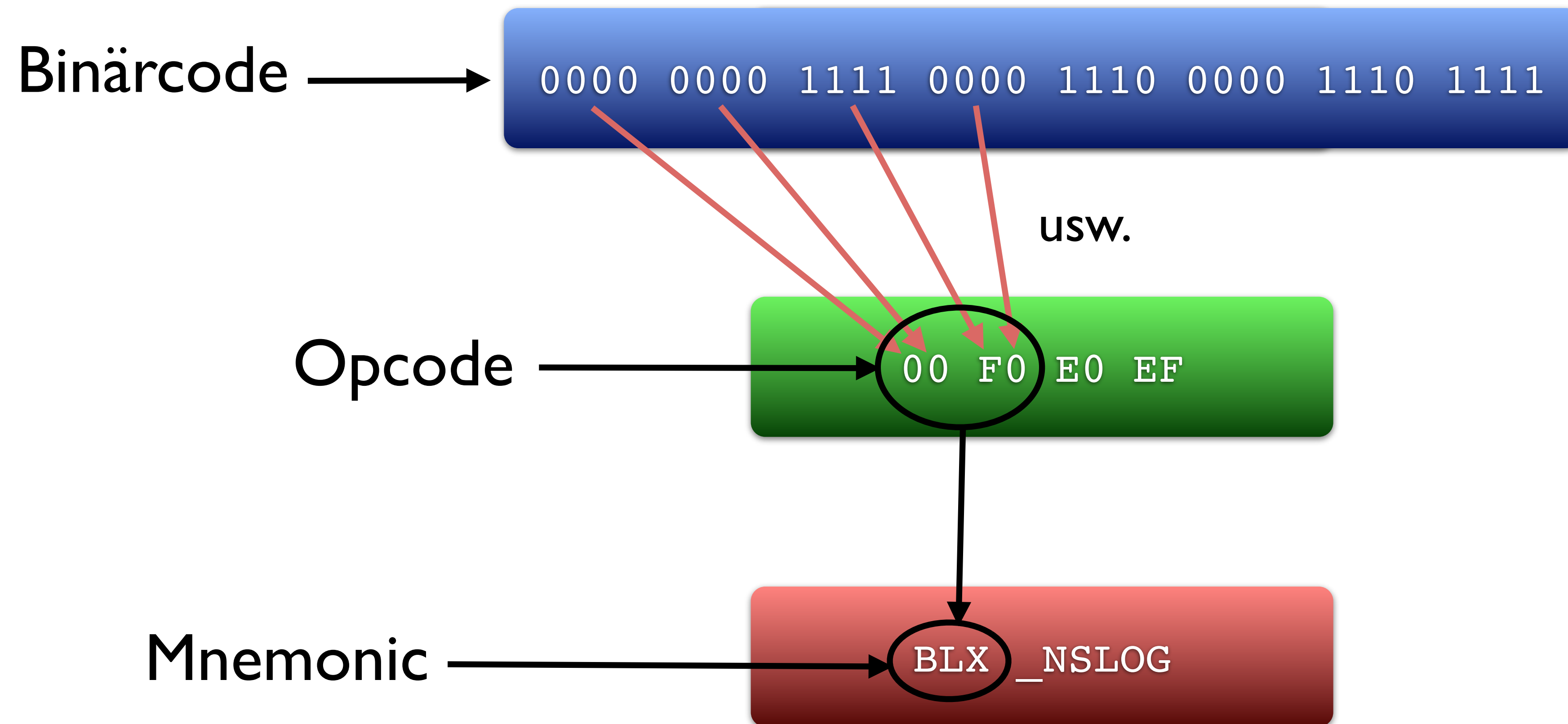
```
MOV R1,PC
```

1. Generation

```
0101101011101101
```

* Ersatzflüssigkeit

Opcodes und Mnemonics



Warum ist Hex cool?

- Die CPU arbeitet im Dualsystem (2)
- Menschen denken im Dezimalsystem (10)
- 10 lässt sich nicht als Potenz von 2 darstellen
- 16 (Hex) ist 2^4
- Jede Hex-Zahl lässt sich mit vier Binärzahlen darstellen
- Jede vierstellige Binärzahl lässt sich mit einer Hex-Zahl darstellen

BLX _NSLOG

Umrechnung leichtgemacht

Binär	Hexadezimal	Dezimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

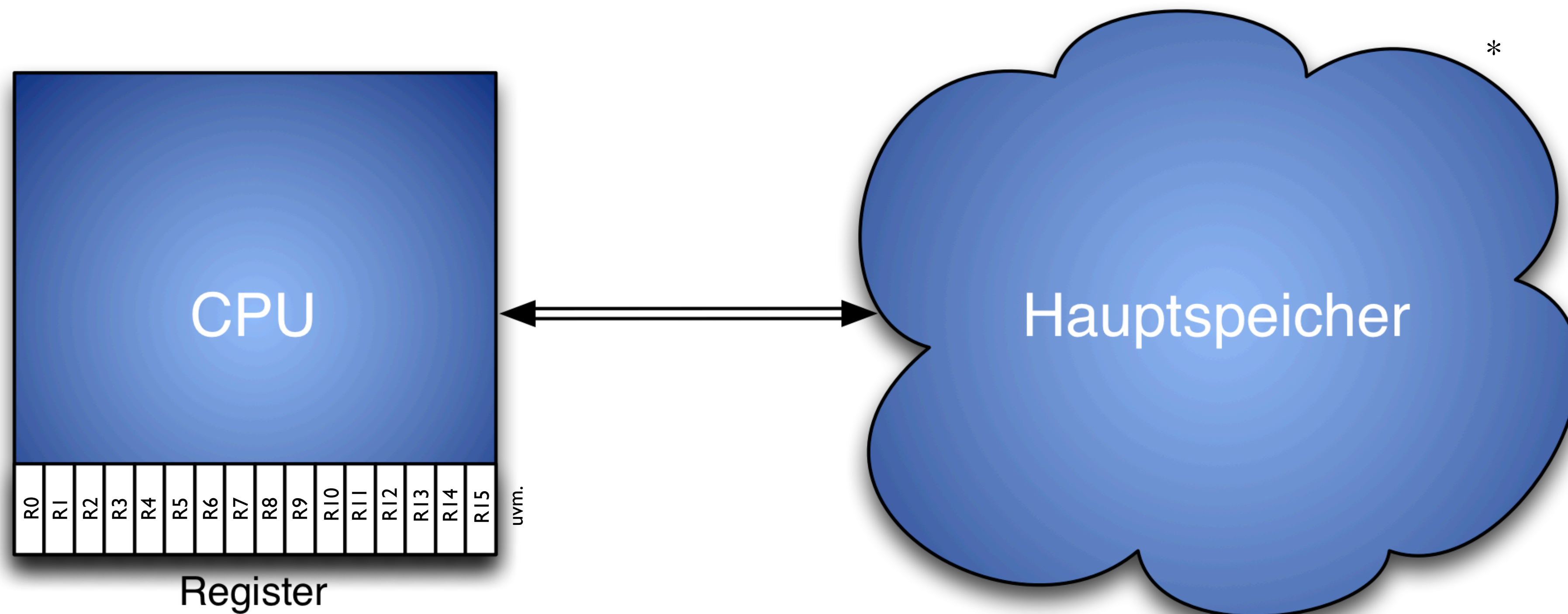
ARM-Assembler

Mnemonic	Bedeutung
ADD	Addition
SUB	Subtraktion
MOV	Bewegen von Daten
CMP	Vergleich
BLX	Sprung mit Wechsel des Befehlssatzes
LDR	Wert aus Register laden
STR	Wert in Register speichern
PUSH	Auf Stack schreiben
POP	Vom Stack lesen
...	...

Operanden

Syntax	Bedeutung
R0	Im Register R0 gespeicherte Adresse
[R0]	Inhalt der im Register R0 gespeicherten Adresse
#23	Die Zahl 23
:lower:	Die unteren 16 Bit des Operanden
:upper:	Die oberen 16 Bit des Operanden

Register



* Ersatzflüssigkeit

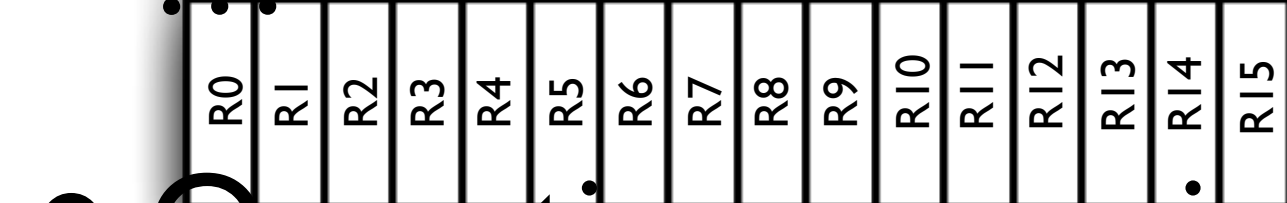
Register

- 15 general purpose register (R0-R14)

- R15 == PC

- CPSR (Current Program Status Register)

-



- Operationen nur in Registern möglich, nicht im Speicher



* Ersatzflüssigkeit

General Purpose Register

Register	Eigenschaften
R0-R3	Parameter und Rückgabewerte
R4-R6	frei verfügbar, erhalten Werte über Funktionsaufrufe
R7	Frame Pointer & Link Register (!= ARM Specs)
R8	Analog R4-R6
R9	iOS 2.x: reserviert. iOS >= 3.x: verfügbar
R10-R11	Analog R4-R6
R12	Intra Procedure Scratch Register (IP)
R13	Stack Pointer (SP)
R14	Link Register (LR) - Rücksprungadresse
R15	Program Counter (PC)

D0-D31:VFP - Floating Point Register

Current Program Status Register

- Status-Register
- Flags enthalten Informationen zur aktuellen Ausführung
 - N: negatives Ergebnis einer Berechnung
 - Z: Ergebnis ist Null
 - T: Thumb oder nicht
 - ...

Befehlssatz

- ARMv7
 - 32 Bit, 8 Bit == 1 Byte
 - Little Endian Byte Order
- Thumb-Mode
 - Instruktionen in 16 Bit
 - geringere Code-Größe (~35% kleiner)

Eierköpfe aka Byte Order

Little Endian

1101 1110	1010 1101	1011 1110	1110 1111
D E	A D	B E	E F

Big Endian

1110 1111	1011 1110	1010 1101	1101 1110
E F	B E	A D	D E

Stack

```
MOV R0, 0xDEADBEEF
MOV R1, 0x0F00CAFE
MOV R2, 0x41414141
PUSH R0
PUSH R1
PUSH R2
POP R0
POP R1
POP R2
...
```

R15 (PC)

R13 (SP)

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

Stack

MOV R0,0xDEADBEEF

MOV R1,0x0F00CAFE

MOV R2,0x41414141

PUSH R0

PUSH R1

PUSH R2

POP R0

POP R1

POP R2

...

← R15 (PC)

R13 (SP) →

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

Stack

MOV R0,0xDEADBEEF

MOV R1,0x0F00CAFE

MOV R2,0x41414141

PUSH R0

PUSH R1

PUSH R2

POP R0

POP R1

POP R2

...

← R15 (PC)

R13 (SP) →

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

Stack

MOV R0,0xDEADBEEF

MOV R1,0x0F00CAFE

MOV R2,0x41414141

PUSH R0

PUSH R1

PUSH R2

POP R0

POP R1

POP R2

...

← R15 (PC)

R13 (SP) →

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

Stack

```
MOV R0,0xDEADBEEF
MOV R1,0x0F00CAFE
MOV R2,0x41414141
PUSH R0
PUSH R1
PUSH R2
POP R0
POP R1
POP R2
...
```

R13 (SP)

0xDEADBEEF

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

uninitialized

R15 (PC)

Stack

```
MOV R0,0xDEADBEEF
MOV R1,0x0F00CAFE
MOV R2,0x41414141
PUSH R0
PUSH R1
PUSH R2
POP R0
POP R1
POP R2
...
```

← R15 (PC)

R13 (SP) →

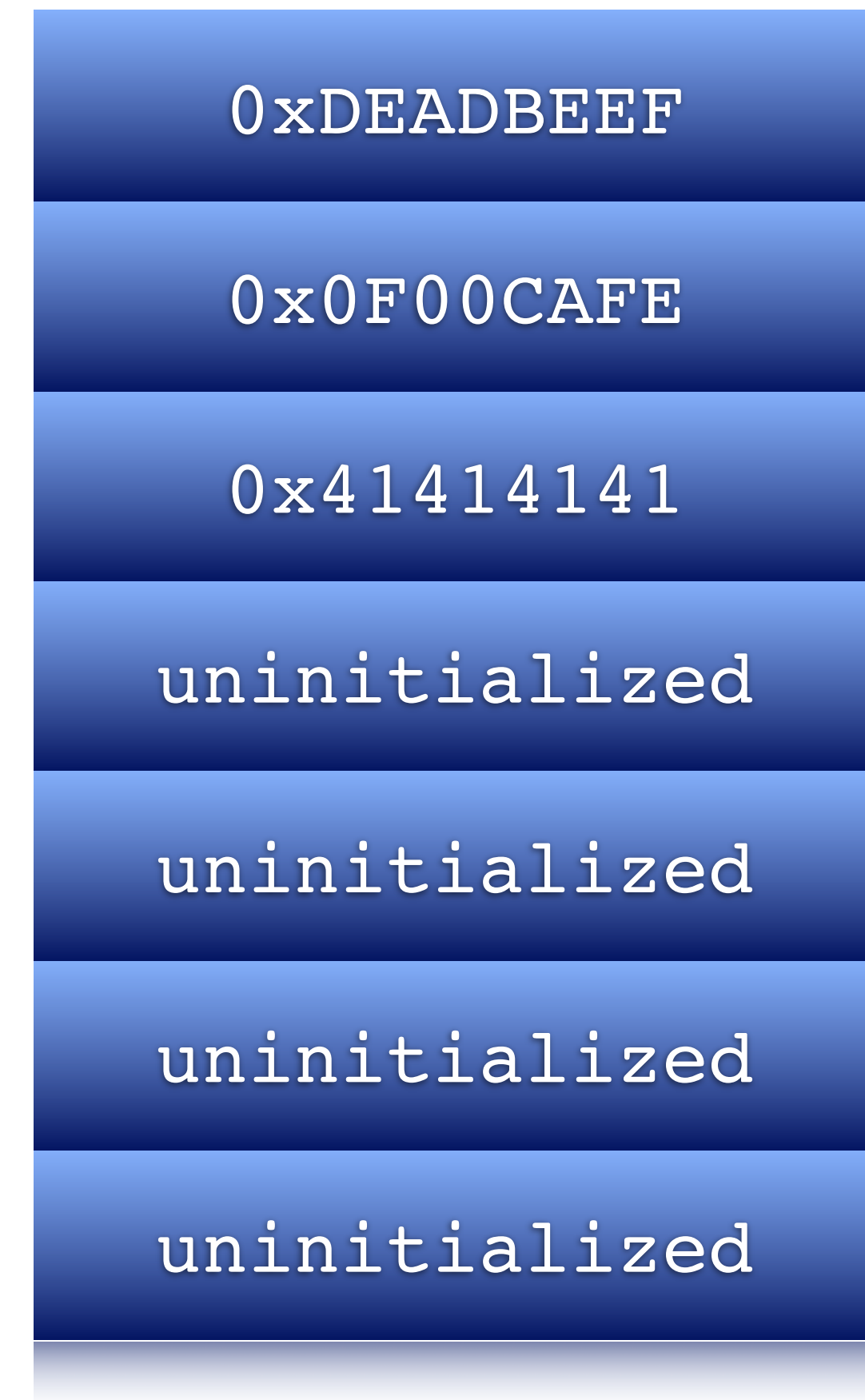


Stack

```
MOV R0,0xDEADBEEF
MOV R1,0x0F00CAFE
MOV R2,0x41414141
PUSH R0
PUSH R1
PUSH R2
POP R0
POP R1
POP R2
...
```

← R15 (PC)

R13 (SP) →

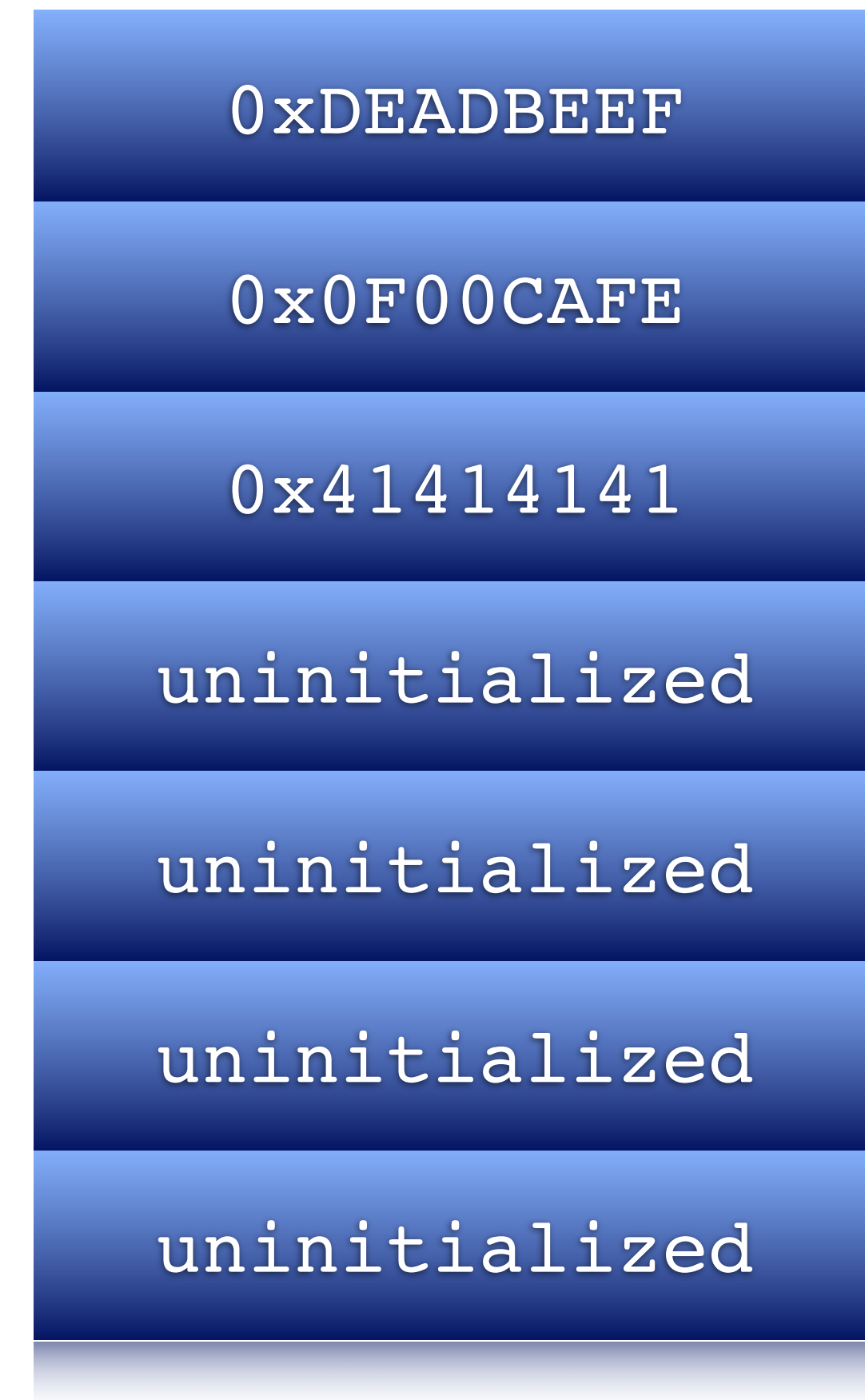


Stack

```
MOV R0,0xDEADBEEF
MOV R1,0x0F00CAFE
MOV R2,0x41414141
PUSH R0
PUSH R1
PUSH R2
POP R0
POP R1
POP R2
...
```

← R15 (PC)

R13 (SP) →

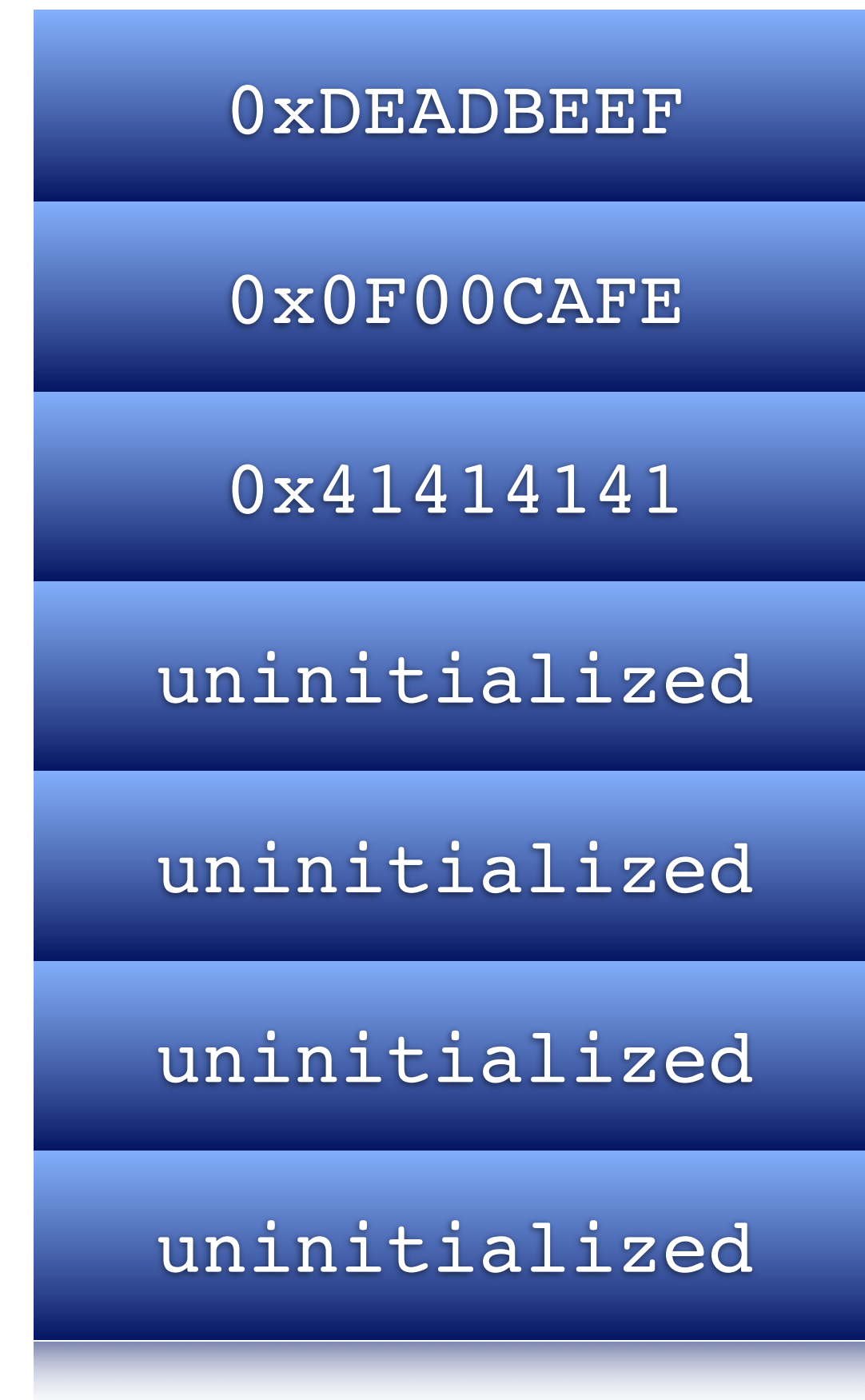


Stack

```
MOV R0,0xDEADBEEF
MOV R1,0x0F00CAFE
MOV R2,0x41414141
PUSH R0
PUSH R1
PUSH R2
POP R0
POP R1
POP R2
...
```

← R15 (PC)

R13 (SP) →



Stack

```
MOV R0,0xDEADBEEF
MOV R1,0x0F00CAFE
MOV R2,0x41414141
PUSH R0
PUSH R1
PUSH R2
POP R0
POP R1
POP R2
...
```

R13 (SP)
→

0xDEADBEEF

0x0F00CAFE

0x41414141

uninitialized

uninitialized

uninitialized

uninitialized

← R15 (PC)

Prolog

- Inhalt des Link Register (LR) wird auf den Stack geschrieben
- Inhalt des Frame Pointer wird auf den Stack geschrieben
- Framepointer wird auf Stack Pointer (SP) gesetzt
- Schreibt die Werte aller zu speichenden Register auf den Stack
- Reserviert Speicher im Stack

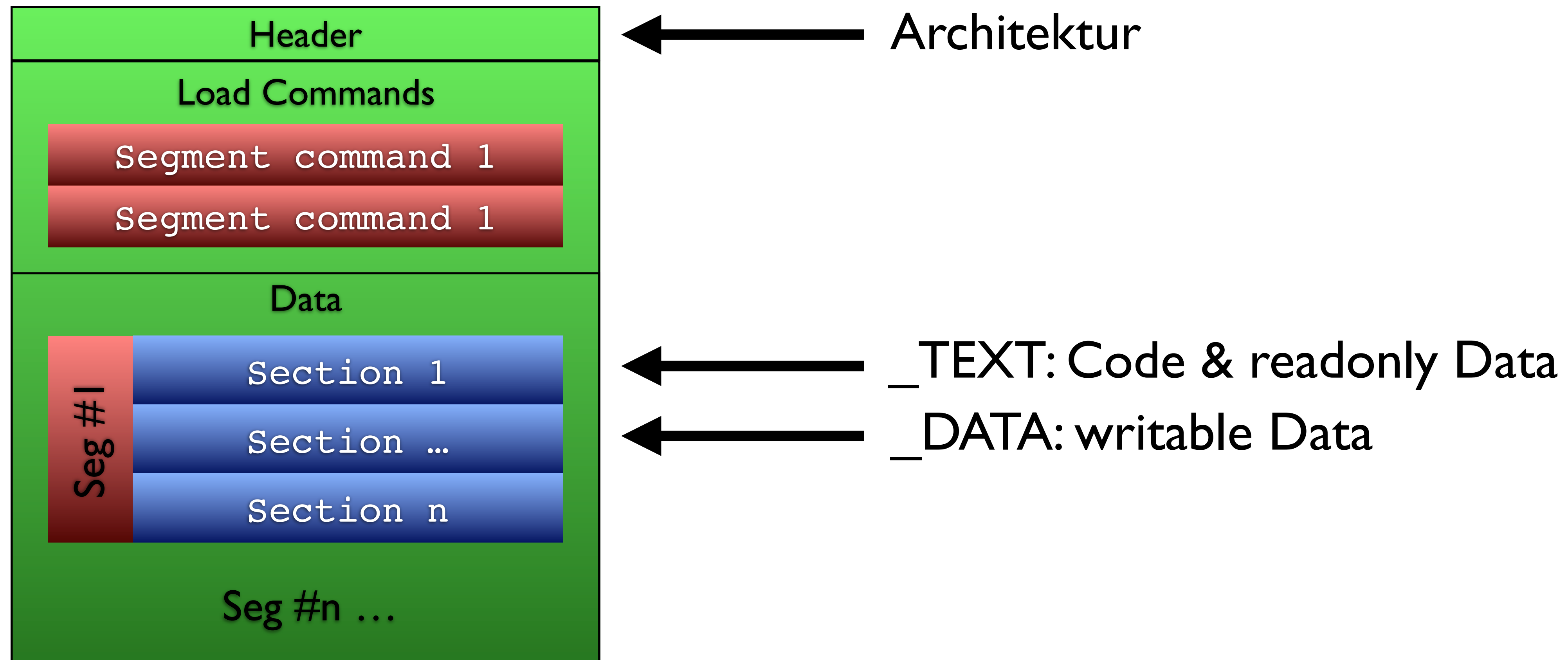
Epilog

- Lokalen Speicher freigeben
- gesicherte Register aus Stack wiederherstellen
 - Frame Pointer
 - Link Register

Mach-O

- Mach-O („Mach object“)
- Binärformat (ABI) auf Mac OS X und iOS
- Code und Daten

Aufbau Mach-O



Praxis

Foobar

```
#include <stdio.h>

int main()
{
    printf("Foobar!");
    return 0;
}
```




D3mo!!^ölf

Erkenntnisse

- Assembler ist cool
- Assembler ist plattformspezifisch
- Alle Tools in Xcode und OS X enthalten



```
[theAudience setBenefit:nil];
```

Objective-C Messaging

- Kommunikation von Objekten über Nachrichtenaustausch
- Abstraktion von Funktionsaufrufen

Senden einer Nachricht

OOP-Ebene:

```
NSString *fooBar = @"23";
```

Funktionsebene:

```
id temp = objc_msgSend([NSString class], @selector(alloc));  
NSString *fooBar = objc_msgSend(temp, @selector initWithCString:), "23");
```

Calling convention

- Empfänger einer Nachricht: R0
- Argumente: R1-R3
- Rückgabewert: R0



D3mo!!^drölf

Dekompilierung: Objective-C

```
- (BOOL)application:(UIApplication *)inApplication
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    self.managedObjectContext.persistentStoreCoordinator =
    self.storeCoordinator;
    self.window.rootViewController = self.viewController;
    [self.window makeKeyAndVisible];
    return YES;
}
```

Dekompilierung: C

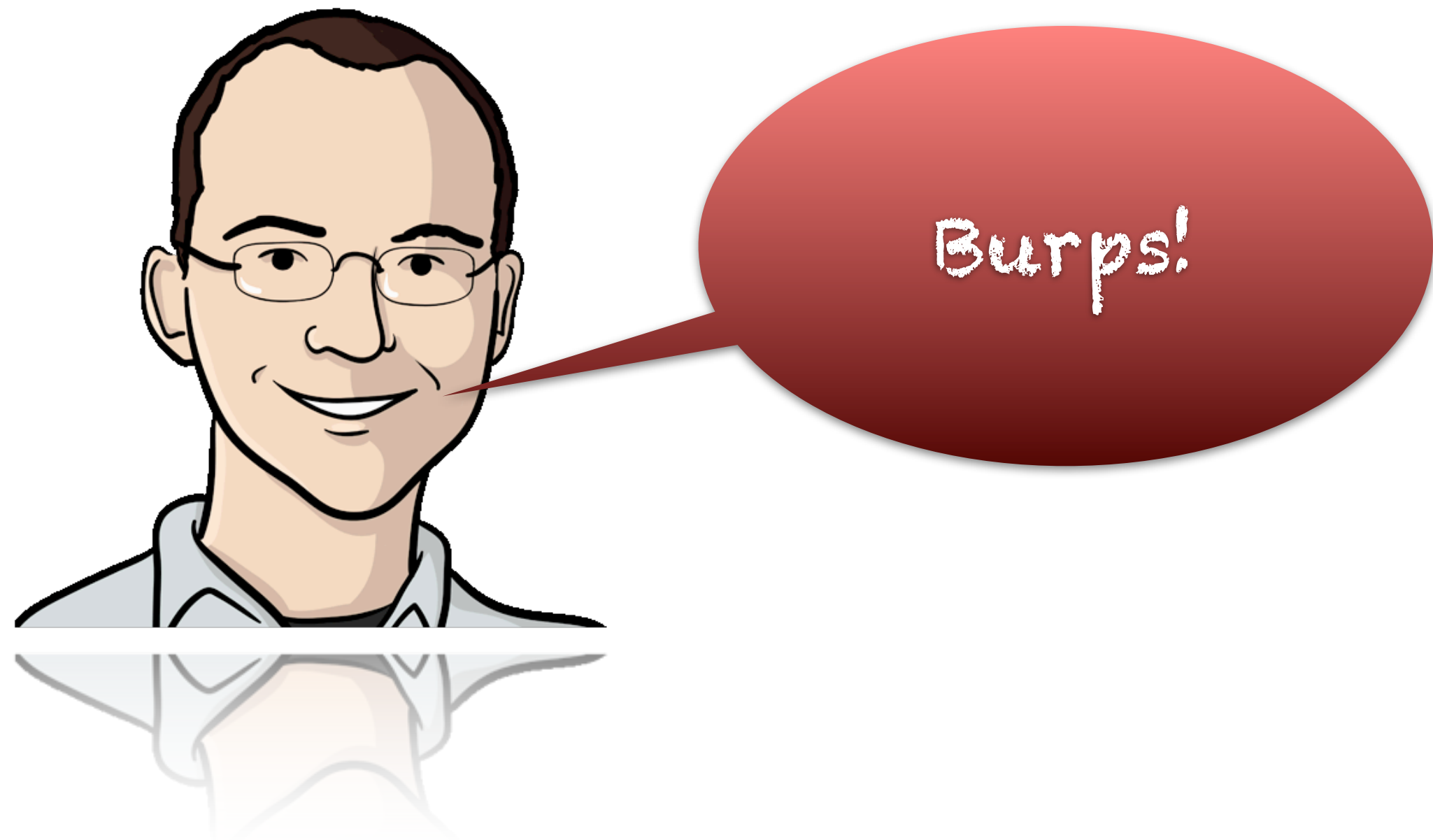
```
char __cdecl  
__PhotoDiaryAppDelegate_application_didFinishLaunchingWithOptions__(struct  
PhotoDiaryAppDelegate *self, SEL a2, id a3, id a4)  
{  
    ...
```


Praxis

- !(Apps in Assembler programmieren)
- Optimierung und Analyse
- Beispiel: Dead Store Elimination

Dead Store Elimination

```
void bar(int x) {  
    printf("Bar!\n");  
    char foobar[10];  
    int i;  
    for (i=0; i<sizeof(foobar); ++i)  
        foobar[i]=x++;  
    memset(foobar,0,sizeof(foobar));  
}
```

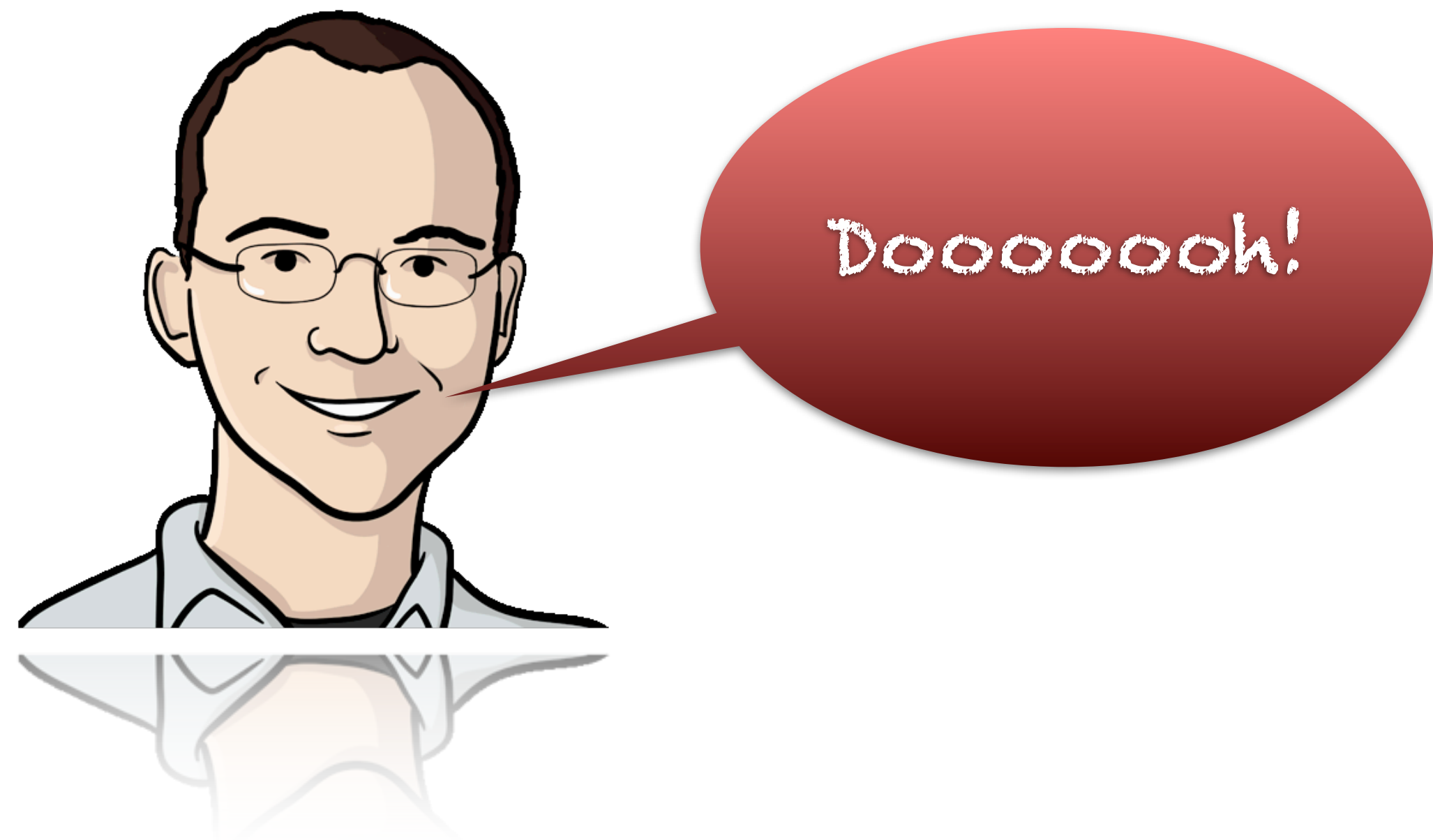


Dead Store Elimination

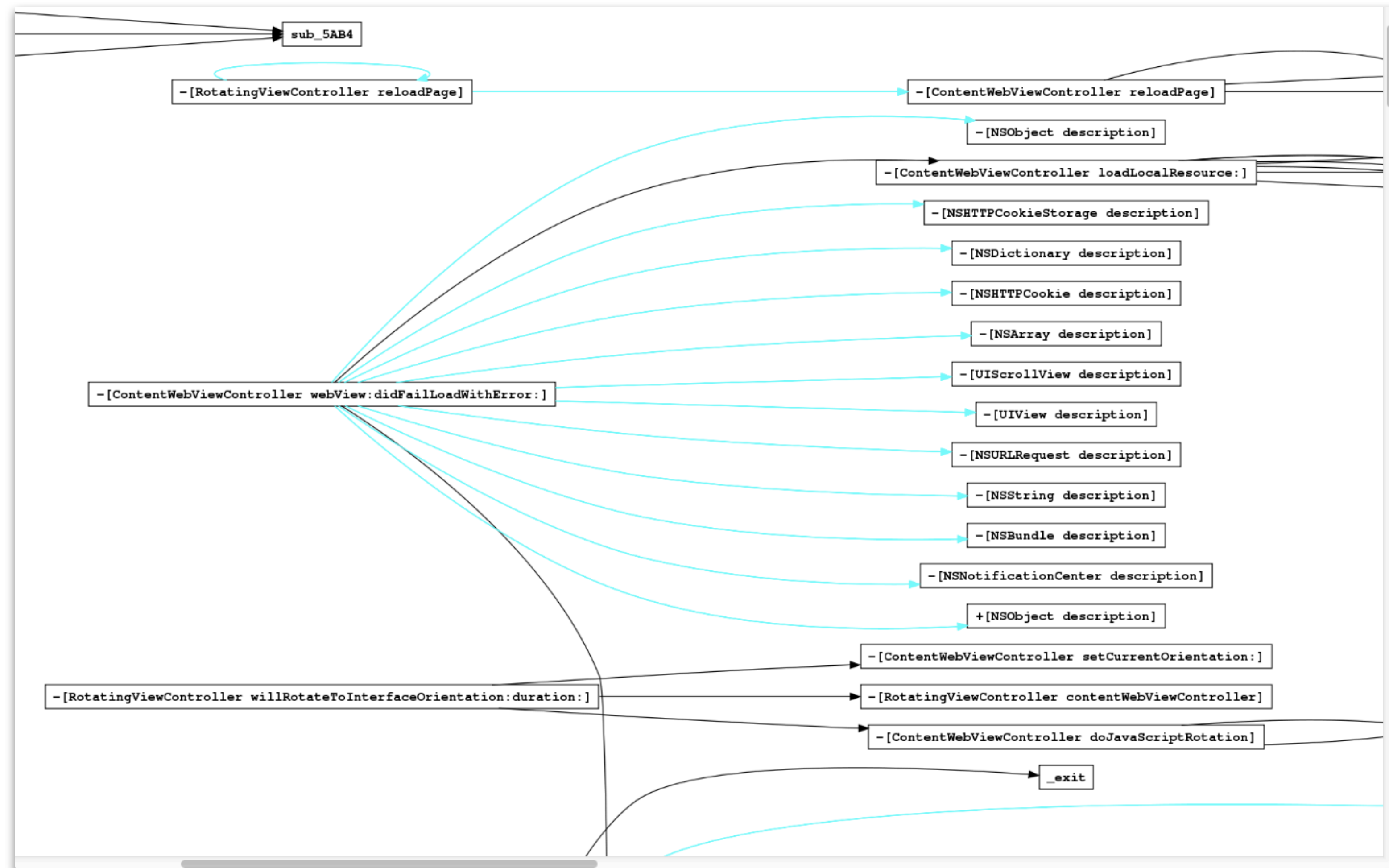
```
void bar(int x) {  
    printf("Bar!\n");  
    char foobar[10];  
    int i;  
    for (i=0; i<sizeof(foobar); ++i)  
        foobar[i]=x++;  
    memset(foobar,0,sizeof(foobar));  
    asm volatile("" : : "r"(&foobar) : "memory");  
}
```

Inline Assembler

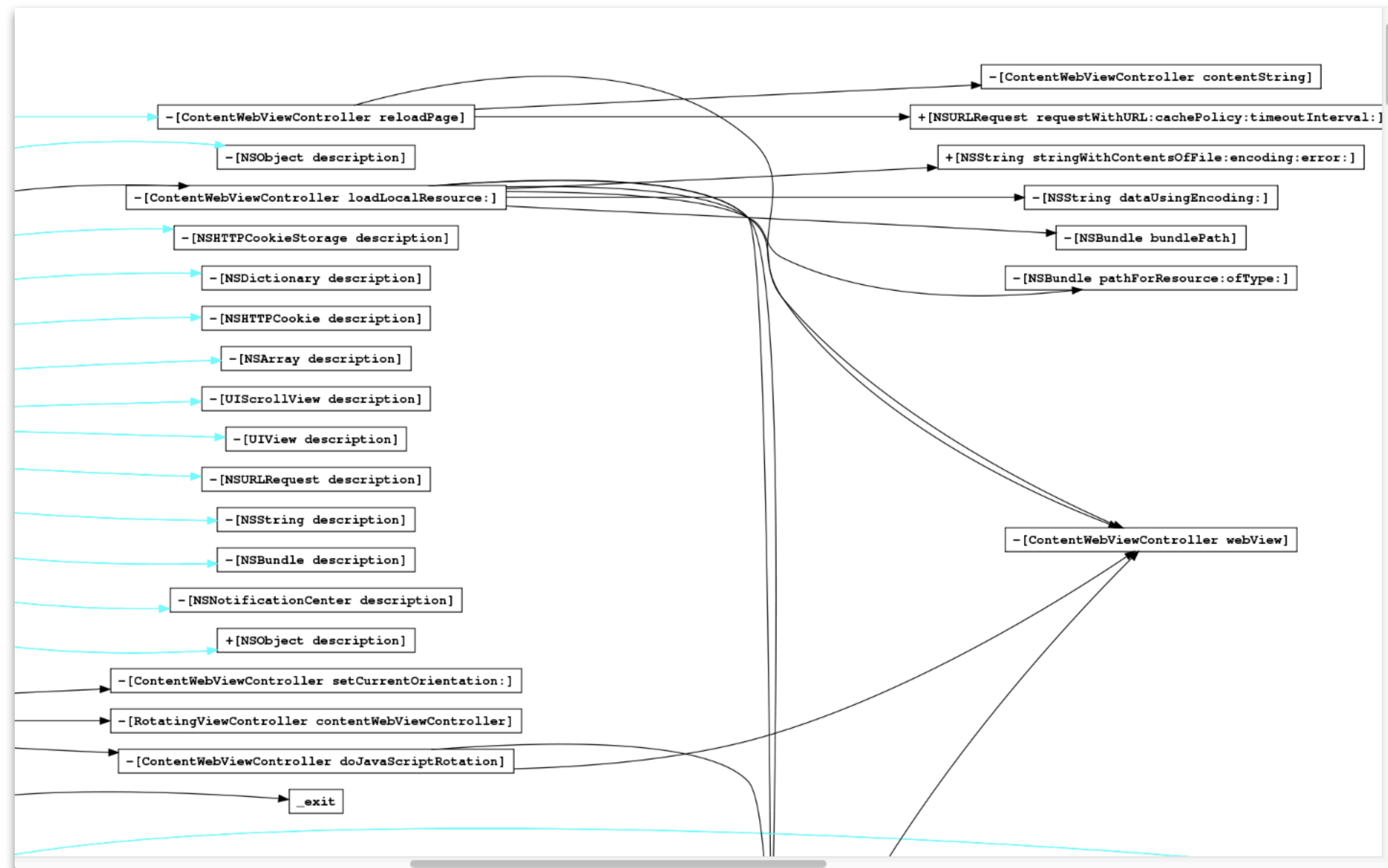
```
asm volatile (Code: Input : Output : Clobber);
```



Steve



Steve

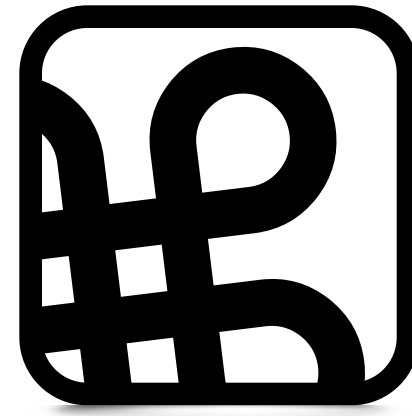


Literatur

- Apple: „Mac OS X ABI Mach-O File Format Reference“
- Apple: „iOS ABI Function Call Guide“
- ARM Infocenter (kostenlose Registrierung erforderlich)
- Peter Cockerell: „ARM Assembly Language Programming“
- Chris Eagle: „The IDA PRO Book“

Fragen?

No animals were harmed for this presentation.



Macoun