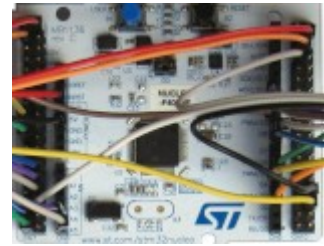


# Mikroprozessortechnik

Prof. Dr. Michael Lipp



# C++-Wissen

# Klassenattribute (Versuch 2)

- **Attribute können mit `static` als Klassenattribute deklariert werden**
  - Klassenattribute gibt es nur einmal pro Klasse
  - In Methoden kann auf Klassenattribute wie auf normale Attribute zugegriffen werden
  - Wird das Klassenattribut in der Methode eines Objekts geändert, ist die Änderung für alle Objekte sichtbar
  - Klassenattribute haben den gleichen Lebenszyklus wie globale Variablen (vom Beginn des Programms bis zum Ende) sind aber weniger „gefährlich“ da ihre Sichtbarkeit mit `private` auf die Methoden der Klasse beschränkt werden kann

Live Coding

# Klassenattribute

- **Deklaration**

```
class Test {  
    static int classAttribute;  
};
```

- **Definition (mit Initialisierung) in der cpp-Datei**

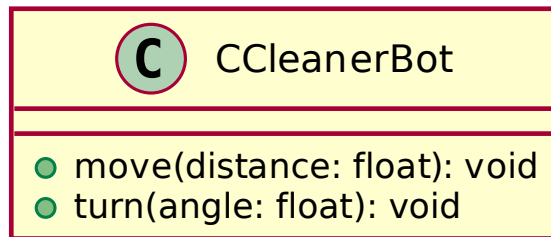
```
int Test::classAttribute = 10;
```



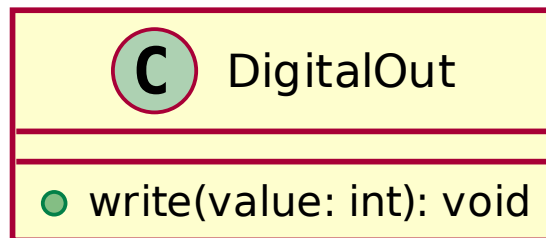
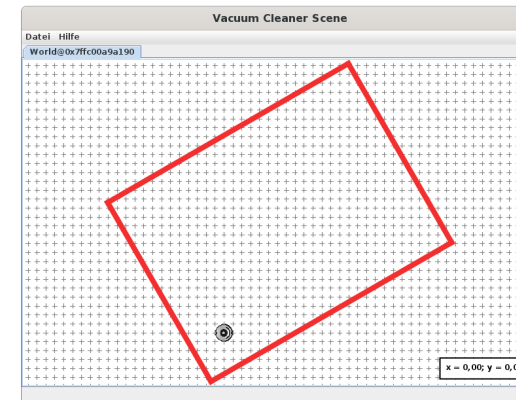
# GPIO – Details

# Peripherie-Ansteuerung

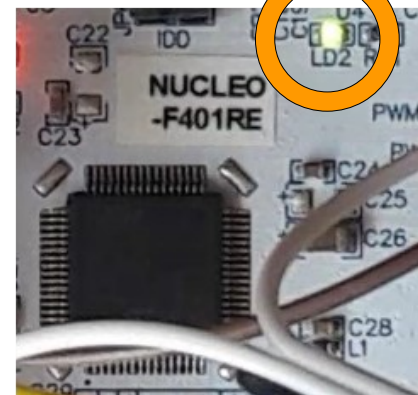
- Vom Modell zur virtuellen/realen Welt



World



???



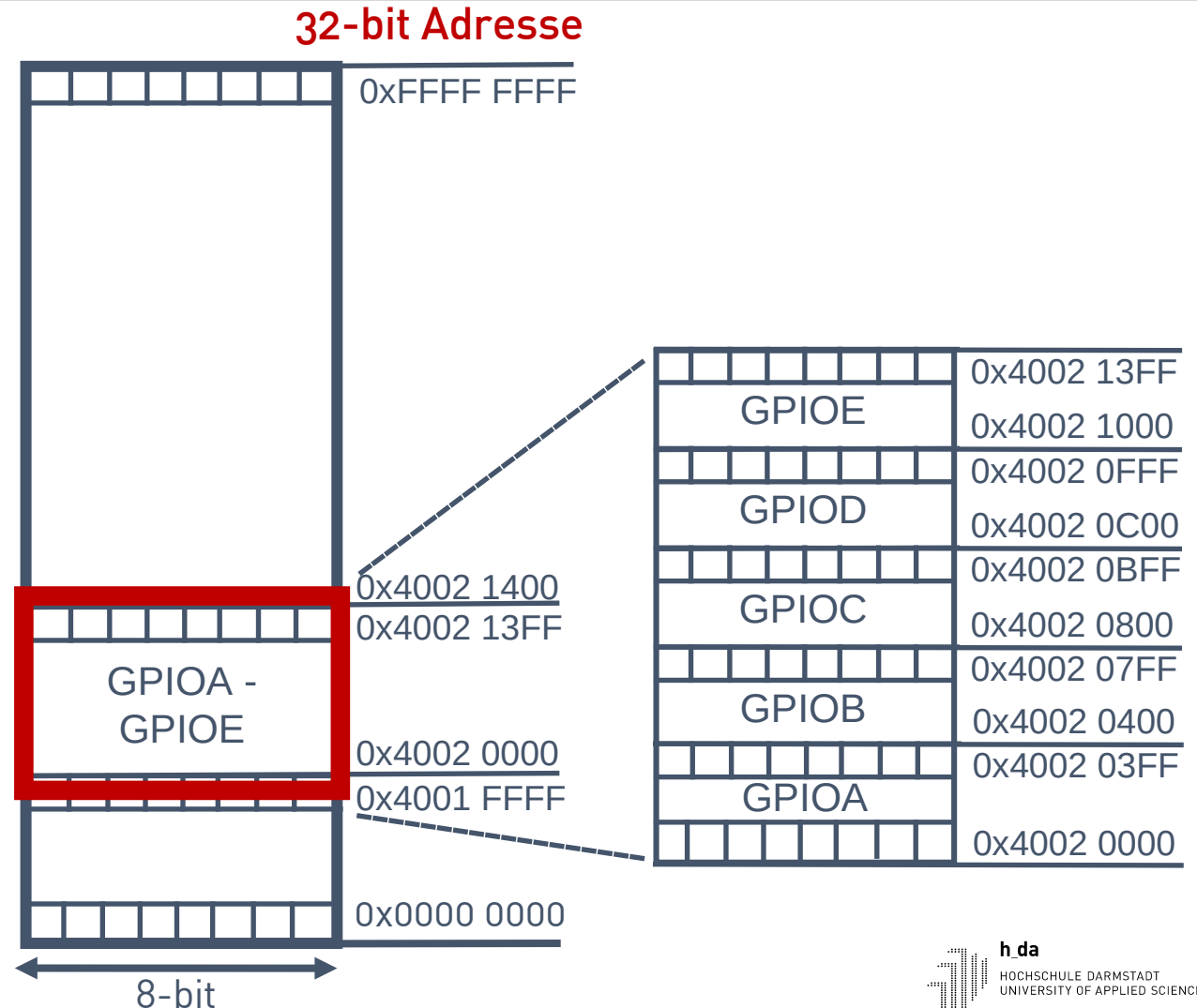
Es gibt eine Klasse DigitalOut, damit kann ich die Pins ansteuern

Wir modellieren hier das Objekt aus der Reellen Welt

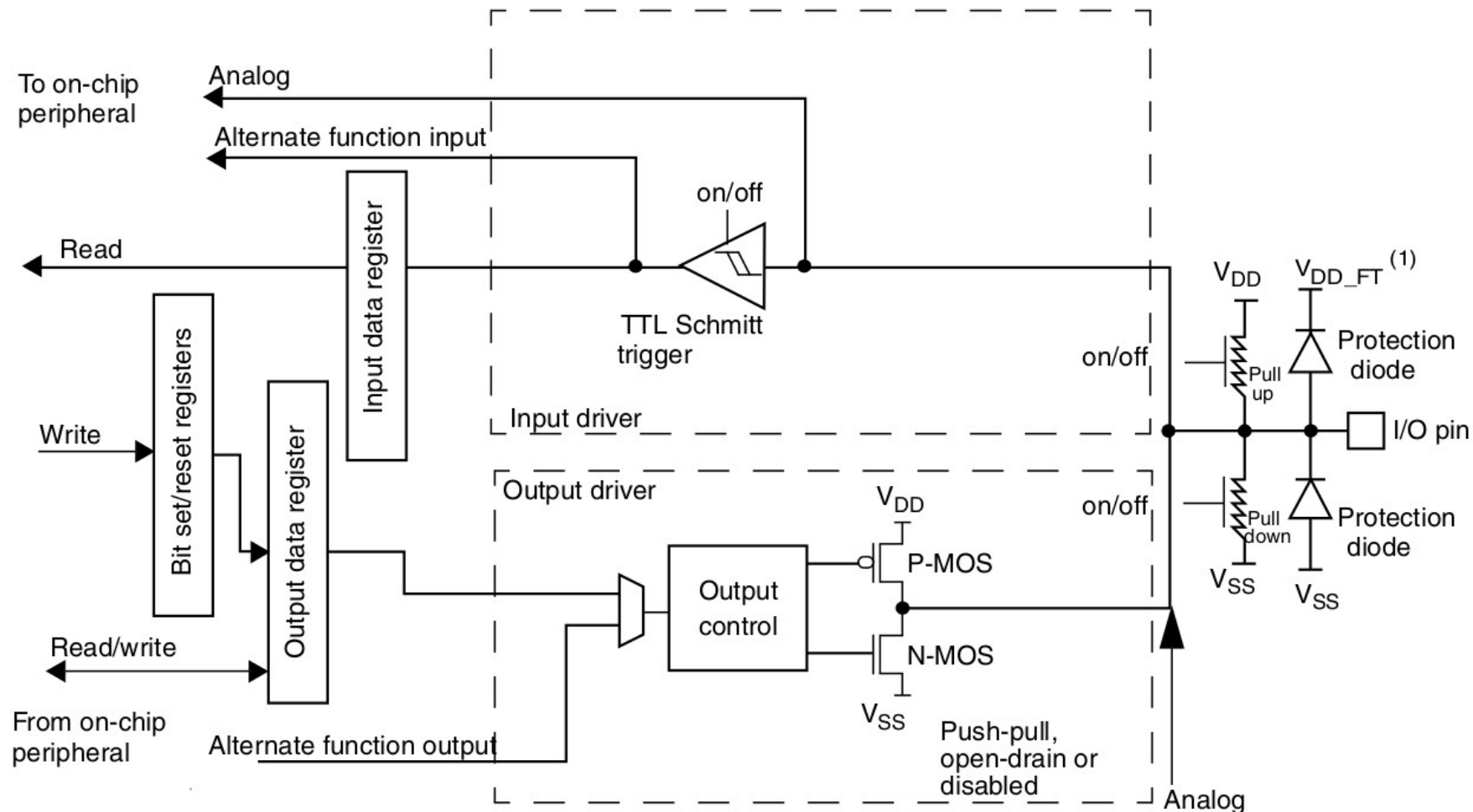
# Peripherie-Ansteuerung

- **Technische Realisierung**

- Memory-Mapped-I/O
- Nicht hinter jeder Speicheradresse befindet sich Speicher
- Beim Lesen/Schreiben bestimmter Bereiche werden Register (Gruppen von Flip-Flops) adressiert
- Die Zustände der Flip-Flops steuern nachgelagerte digitale Logik



# Wdh.: GPIO Pins sind konfigurierbar






# GPIOx Registersatz

Zu jedem GPIO Port gehört diese Registersatz.  
Note - Ein Pin ist nur Pin aber ein Port besteht aus mehrere Pins

z.B. BusOut Gruppierung ist ein Software Gruppierung deshalb können wir hier beliebige Pins gruppieren. Aber auf dem Mikrokontroller Hardware sind schon manche Pins gruppiert. Die werden gemeinsam von der Register gesteuert



AFRLn / AFRHn	0x.... ..27 0x.... ..20
LCKn / LKK	0x.... ..1F 0x.... ..1C
Bsn / BRn	0x.... ..1B 0x.... ..18
ODRn	0x.... ..14
IDRn	0x.... ..10
PUPDRn	0x.... ..0F 0x.... ..0C
OSPEEDn	0x.... ..0B 0x.... ..08
OTn	0x.... ..04
MODERn	0x.... ..03 0x.... ..00

32-bit

# GPIOx Registersatz

Ein Port bei unsere ARM Prozessor besteht aus 16 Pin. Ein Register hat immer 32 Bit. Hier werden alle Bits benutzt weil jeder Pin hat zwei Bit aus dem Register. Wir können hier für jede einzelne Pin modus einstellen und zwar mit 2 Bit.

Möglichkeiten mit 2 bit ist 4. Wie ?  $2^2 = 4$  verschiedene Zustände annehmen

z.B.

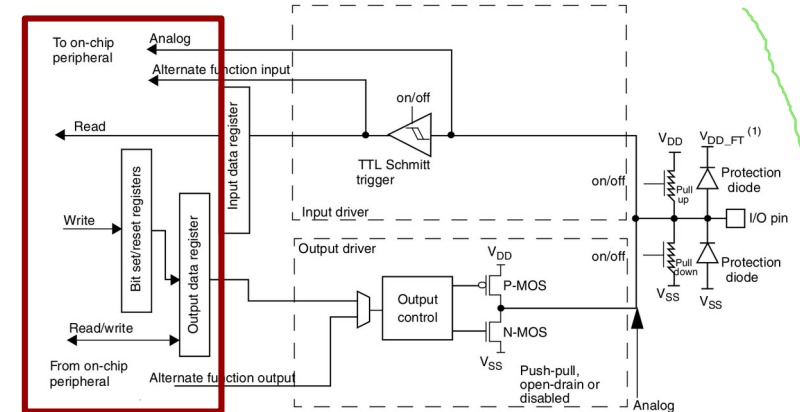
## 8.4.1 GPIO port mode register (GPIOx\_MODER) (x = A..E and H)

Wenn ich die Mikrokontroller einschalte dann hat es der Wert 00000 überall. Es ist kein gute Idee unsere Mikrokontroller auf Ausgang Modus zum einschalten weil das kann z.B. mit hohe Strom den Transistor zerstören

Address offset: 0x00

Reset values:

- 0x0C00 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits  $2y:2y+1$  **MODERy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

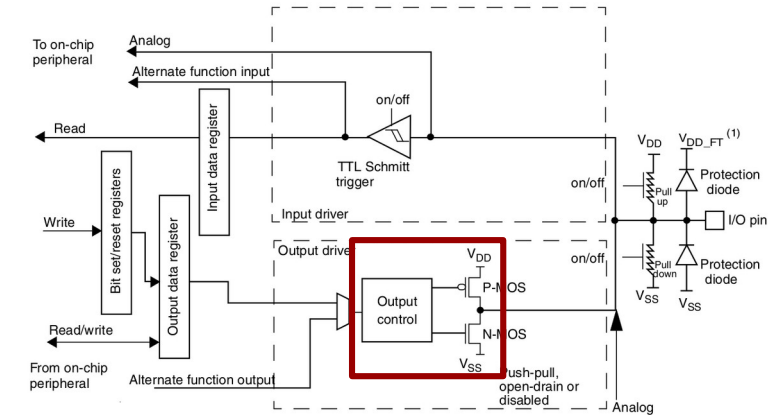
- 00: Input (reset state)
- 01: General purpose output mode
- 10: Alternate function mode
- 11: Analog mode

Quelle: [F401-RM], 8.4

# GPIOx Registersatz

- 8.4.2 GPIO port output type register (GPIOx\_OTYPER)**  
**(x = A..E and H)** Hier kann ich für jeder Pin einstellen ob er in diese Push-Pull oder Open-Drain konfiguration verwendet werden soll  
 Address offset: 0x04  
 Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw



Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OT<sub>y</sub>**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the output type of the I/O port.

0: Output push-pull (reset state)

1: Output open-drain

# GPIOx Registersatz

## 8.4.3 GPIO port output speed register (GPIOx\_OSPEEDR) (x = A..E and H)

Address offset: 0x08

Reset values:

- 0x0C00 0000 for port A
- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **OSPEEDRy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output speed.

00: Low speed

01: Medium speed

10: High speed

11: Very high speed

Ich kann nicht nur Eingang und Ausgang sondern kann ich auch die Geschwindigkeit einstellen.

Maximum output high to low level fall time and output low to high level rise time,  $V_{DD} \geq 2.7 \text{ V}$

00, $C_L = 50 \text{ pF}$	100 ns
01, $C_L = 50 \text{ pF}$	10 ns
10, $C_L = 40 \text{ pF}$	6 ns
11, $C_L = 30 \text{ pF}$	4 ns

Wenn ich schnell umlade dann fließt höhere Strom als wenn ich langsam umlade

# GPIOx Registersatz

## 8.4.4 GPIO port pull-up/pull-down register (GPIOx\_PUPDR) (x = A..E and H)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

Bits 2y:2y+1 **PUPDRy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

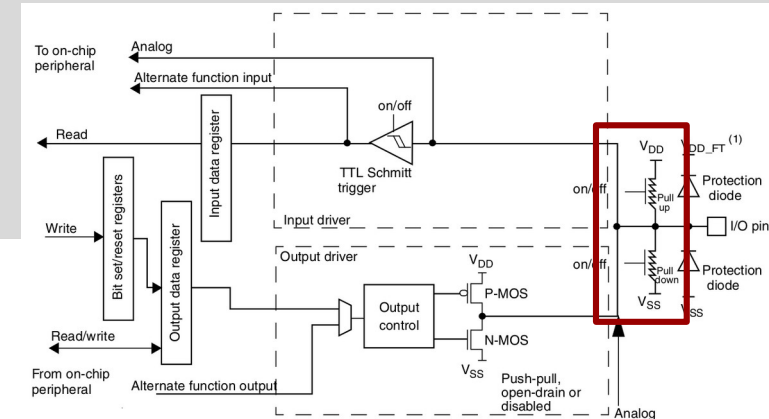
01: Pull-up

10: Pull-down

11: Reserved

Hier wieder 2 Bit für jeder Pin. Ich brauche hier 2 Bit obwohl es gibt hier nur 3 sinnvolle Zustände. Beide Pull Up und Pull-down geschaltet macht wenig Sinn und deshalb steht hier Reserved

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw



# GPIOx Registersatz

Ich lese hier die Werte ein

## 8.4.5 GPIO port input data register (GPIOx\_IDR) (x = A..E and H)

Address offset: 0x10

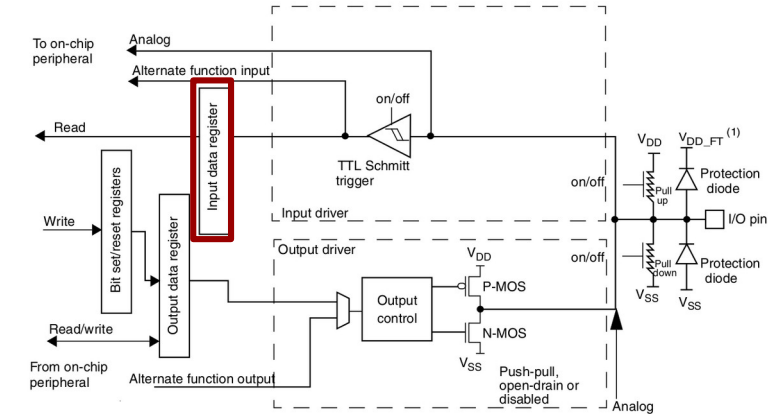
Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.



# GPIOx Registersatz

## 8.4.6 GPIO port output data register (GPIOx\_ODR) (x = A..E and H)

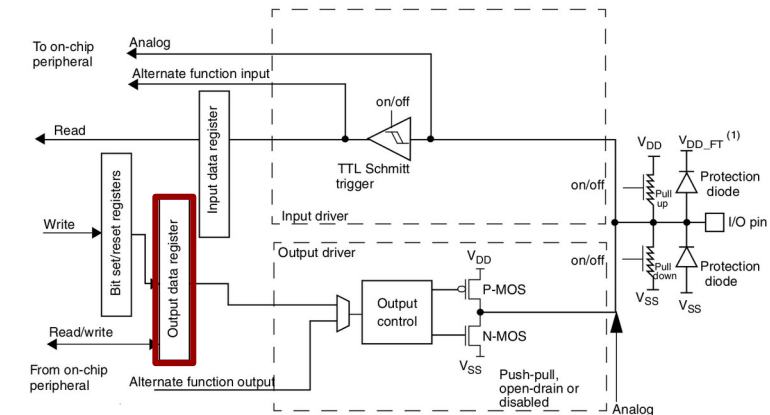
Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)



# GPIOx Registersatz

## 8.4.7 GPIO port bit set/reset register (GPIOx\_BSRR) (x = A..E and H)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy**: Port x reset bit y (y = 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx bit

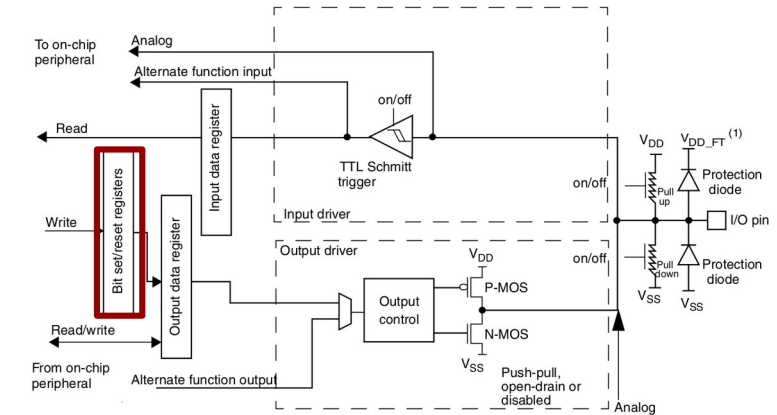
*Note: If both BSx and BRx are set, BSx has priority.*

Bits 15:0 **BSy**: Port x set bit y (y= 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit





# GPIOx Registersatz

## 8.4.8 GPIO port configuration lock register (GPIOx\_LCKR) (x = A..E and H)

Ich verhindere hier das die Konfiguration von der Pin verändern kann

This register is used to lock the configuration of the port bits when a correct write sequence is applied to bit 16 (LCKK). The value of bits [15:0] is used to lock the configuration of the GPIO. During the write sequence, the value of LCKR[15:0] must not change. When the LOCK sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next MCU or peripheral reset.

*Note: A specific write sequence is used to write to the GPIOx\_LCKR register. Only word access (32-bit long) is allowed during this write sequence.*

Each lock bit freezes a specific configuration register (control and alternate function registers).

Address offset: 0x1C

Reset value: 0x0000 0000

Access: 32-bit word only, read/write register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															LCKK
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

# GPIOx Registersatz

Bits 31:17 Reserved, must be kept at reset value.

Bit 16 **LCKK[16]**: Lock key

This bit can be read any time. It can only be modified using the lock key write sequence.

0: Port configuration lock key not active

1: Port configuration lock key active. The GPIOx\_LCKR register is locked until an MCU reset or a peripheral reset occurs.

LOCK key write sequence:

WR LCKR[16] = '1' + LCKR[15:0]

WR LCKR[16] = '0' + LCKR[15:0]

WR LCKR[16] = '1' + LCKR[15:0]

RD LCKR

RD LCKR[16] = '1' (this read operation is optional but it confirms that the lock is active)

*Note: During the LOCK key write sequence, the value of LCK[15:0] must not change.*

*Any error in the lock sequence aborts the lock.*

*After the first lock sequence on any bit of the port, any read access on the LCKK bit will return '1' until the next CPU reset.*

Bits 15:0 **LCKy**: Port x lock bit y (y= 0..15)

These bits are read/write but can only be written when the LCKK bit is '0'.

0: Port configuration not locked

1: Port configuration locked

# GPIOx Registersatz

## 8.4.9 GPIO alternate function low register (GPIOx\_AFRL) (x = A..E and H)

Address offset: 0x20

Hier braucht man für jeder Pin 4 Bit

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **AFRLy**: Alternate function selection for port x bit y (y = 0..7)

These bits are written by software to configure alternate function I/Os

AFRLy selection:

0000: AF0

1000: AF8

0001: AF1

1001: AF9

0010: AF2

1010: AF10

0011: AF3

1011: AF11

0100: AF4

1100: AF12

0101: AF5

1101: AF13

0110: AF6

1110: AF14

0111: AF7

1111: AF15

# GPIOx Registersatz

## 8.4.10 GPIO alternate function high register (GPIOx\_AFRH) (x = A..E and H)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **AFRHy**: Alternate function selection for port x bit y (y = 8..15)

These bits are written by software to configure alternate function I/Os

AFRHy selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

# GPIOx Registersatz

- **Analyse: Wie greifen die Mbed GPIO-Klassen auf die Register zu?**
  - Mit Klicken und F3 (springe zum Quelltext der ausgewählten Funktion) oder im Debugger „Step into“
  - `DigitalIn::read()`
  - `DigitalIn::mode(...)`
  - `DigitalOut::write(...)`
  - `DigitalOut::mode(...)`

Live Coding

# Datenverarbeitung – „physikalisch“

# Speicher-/Registermanipulation mit $\mu$ C

- **GPIO-Funktionen werden über Lesen/Schreiben von „Speicherinhalten“ zugänglich gemacht**
- **Bislang (Informatik/GIT) Speicher eher abstrakt behandelt**
  - Es gibt Speicher
  - Speicherbereiche sind über Aliase (Variablennamen) erreichbar (typisch für Stack, vom Compiler verwaltet)
  - Speicherbereiche können explizit angefordert werden (Heap) und sind über Zeiger erreichbar
  - Bislang nicht genutzt: Mit Zeigern auf „beliebige“ Adressen zugreifen
- **... aber wie funktioniert es eigentlich?**

# Von Neumann-Architektur

- **Allgemeine „Grundarchitektur“**
  - ... mit Varianten (später)
- **Aufbau und Struktur eines von ‚Neumann‘ Rechners**
  - John von Neumann (1903 - 1957) entwarf 1946 ein Konzept zur Auslegung einer universellen Rechnerarchitektur. Selbst heutige Architekturen orientieren sich noch an der Struktur dieses klassischen Universalrechners.

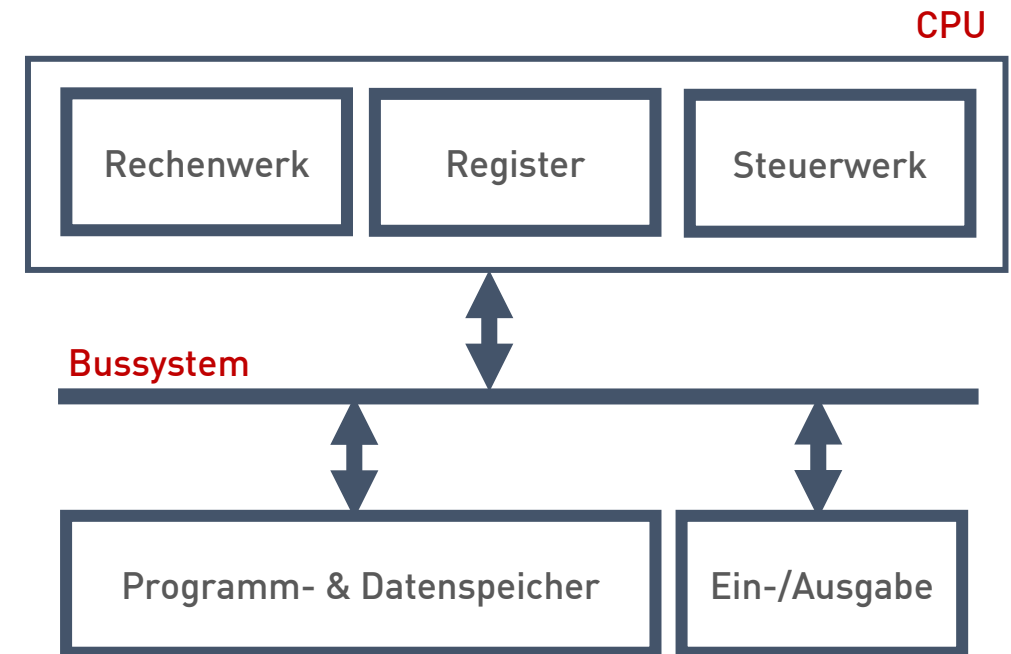


# Von Neumann-Architektur

- **Aufbau und Struktur eines von ‚Neumann‘ Rechner**

- Zentrale Verarbeitungseinheit
  - engl.: Central Processing Unit (CPU)
- Speicher
  - engl.: Memory (Unit)
- Ein- und Ausgabe System
  - engl.: Input/Output Unit
- Bussystem

Der Rechner besteht aus alle diese 4 Dinge.

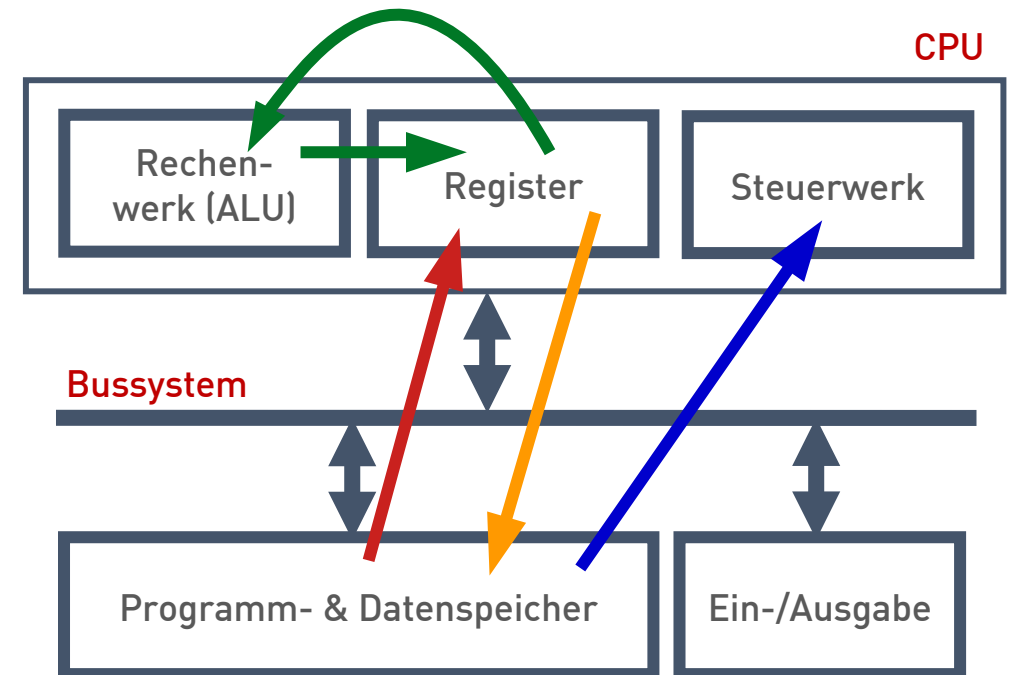


Die einfachste I/O system ist GPIO Pins

# Von Neumann-Architektur

- **Befehl aus Speicher holen**
- **Je nach Befehl...**
  - Daten aus Speicher oder von I/O-Register in CPU lesen
  - Daten verarbeiten
  - Daten von CPU in Speicher schreiben
- **Neuen Befehl holen usw.**

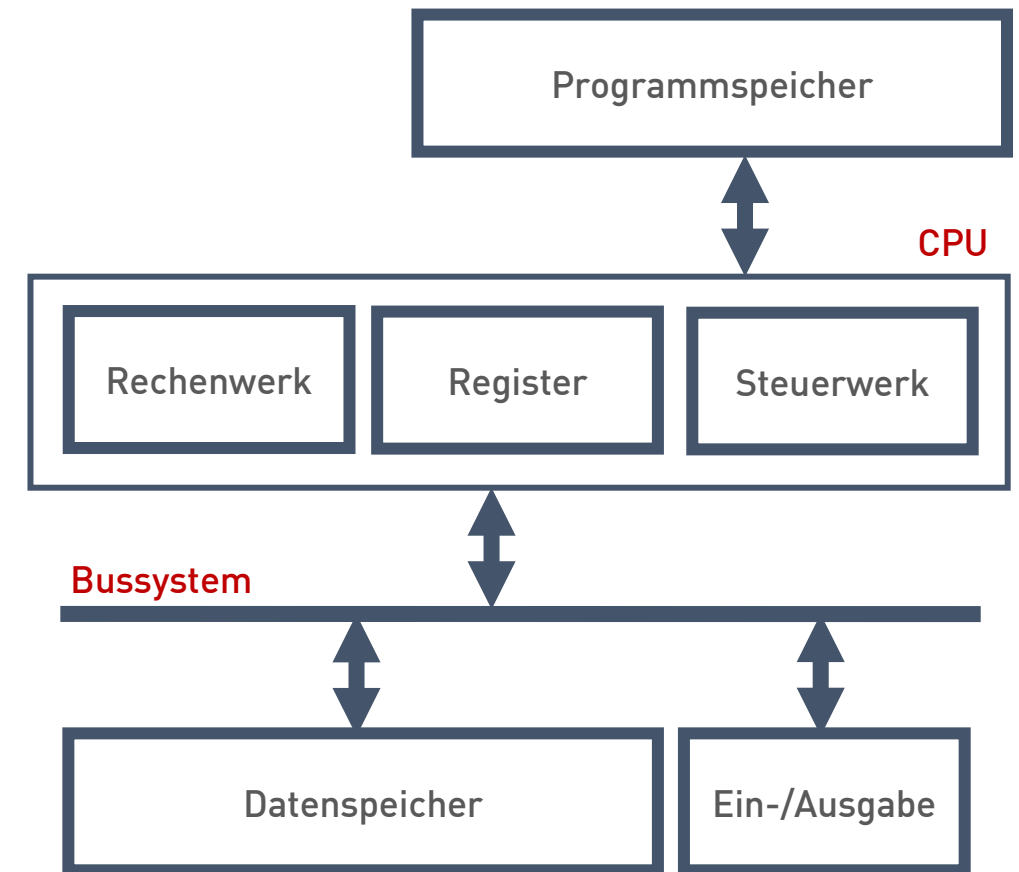
Der Steuerwerk der Teil des CPU holt der Befehl aus dem Speicher. Das ist ein sogenannte Maschinen Befehl. z.B. was soll ich nächstes tun ?



# Harvard-Architektur

- **Getrennte Pfade für Programm und Daten**
  - Holen des nächsten Befehls aus Programmspeicher parallel zu Daten lesen bzw. schreiben
- **(ARM: „Kreuzschienenverteiler“ für noch mehr Parallelität – Details z. B. Modul Embedded Systems)**

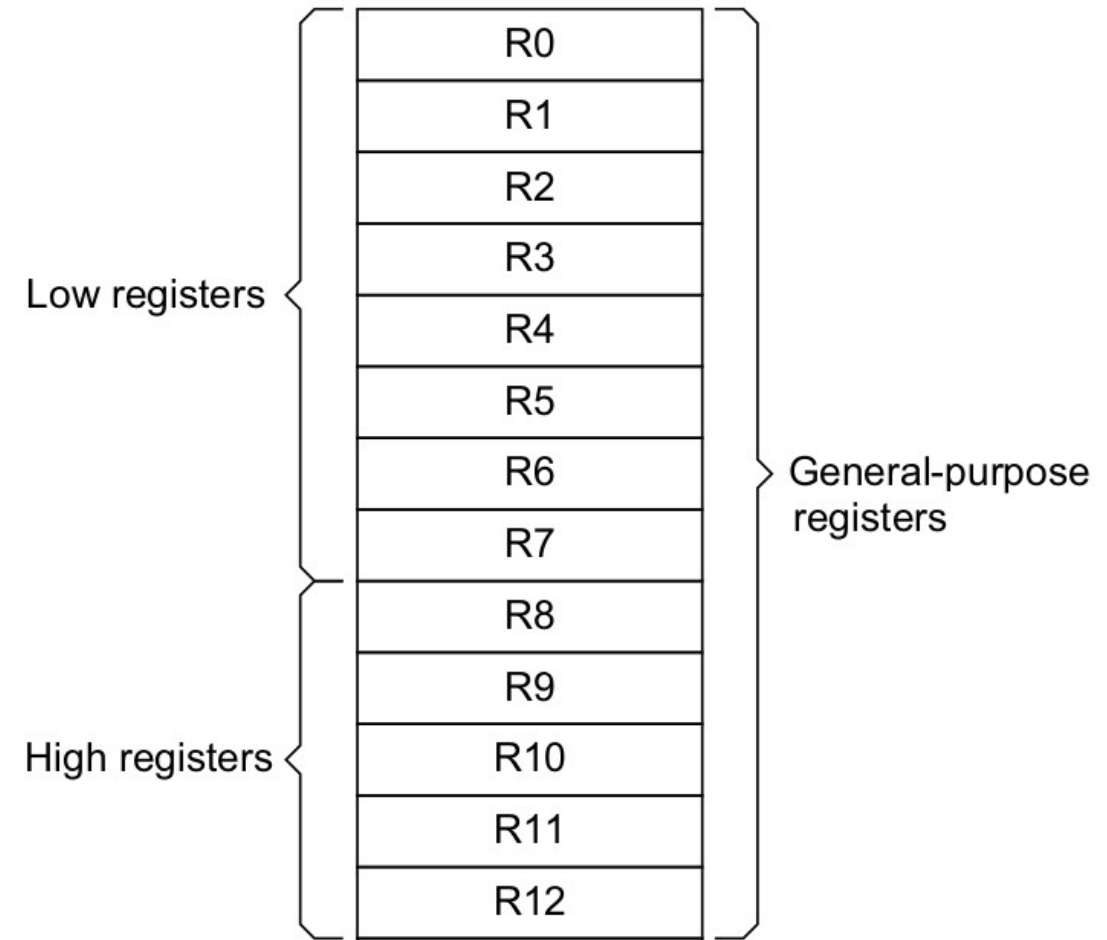
Dadurch wird man schneller aber braucht man mehr Hardware (mehr Leitungen)



# Aufbau der CPU: Register

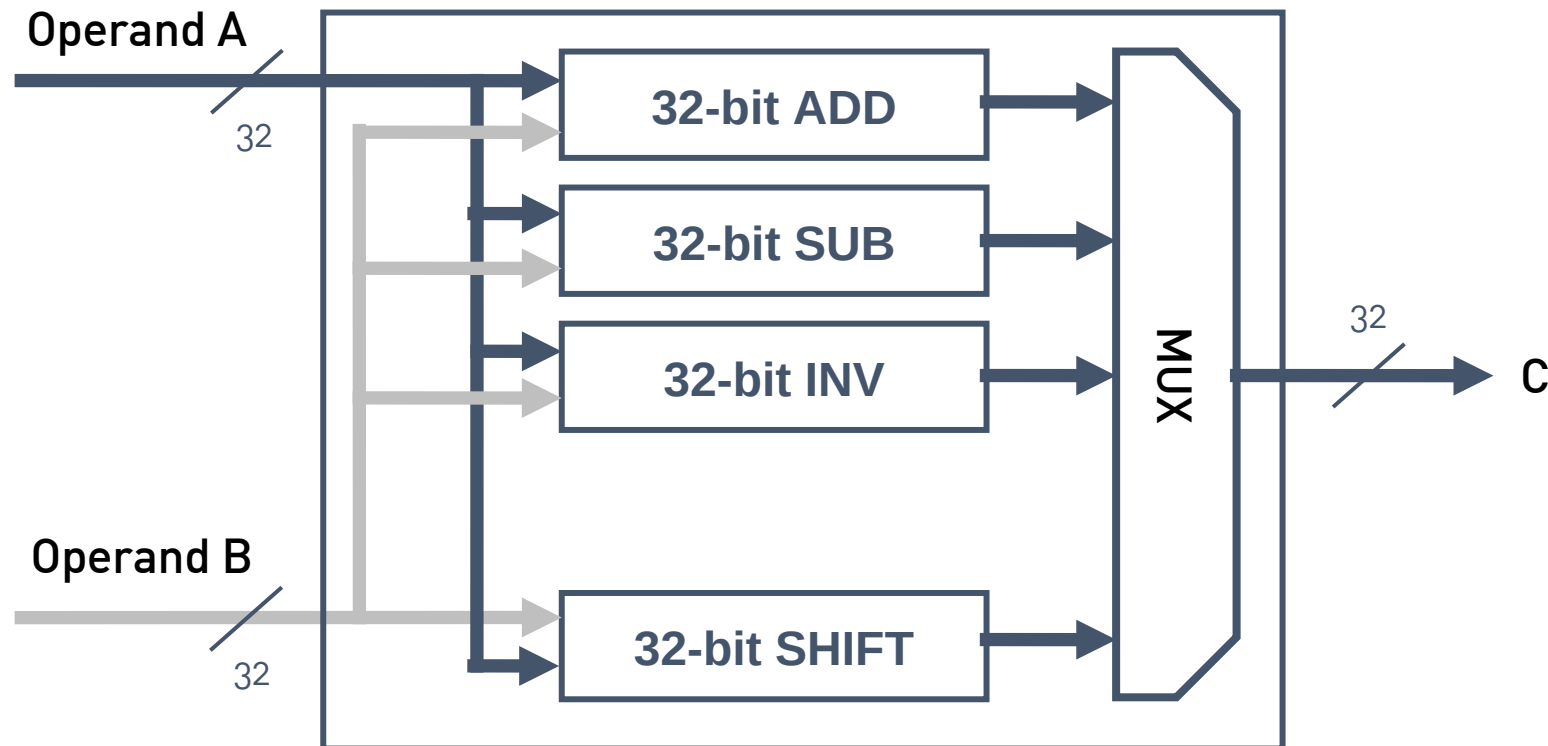
Unsere ARM Prozessor hat kurzzeit Gedächtnis und besteht aus 16 Registern davon:-

- **13 General-purpose registers**
  - 32-bit
  - Für beliebige Daten
- **R13 – R15 (und weitere) sind „special purpose“**
  - Werden später im Detail behandelt



# Aufbau der CPU: ALU (Prinzip)

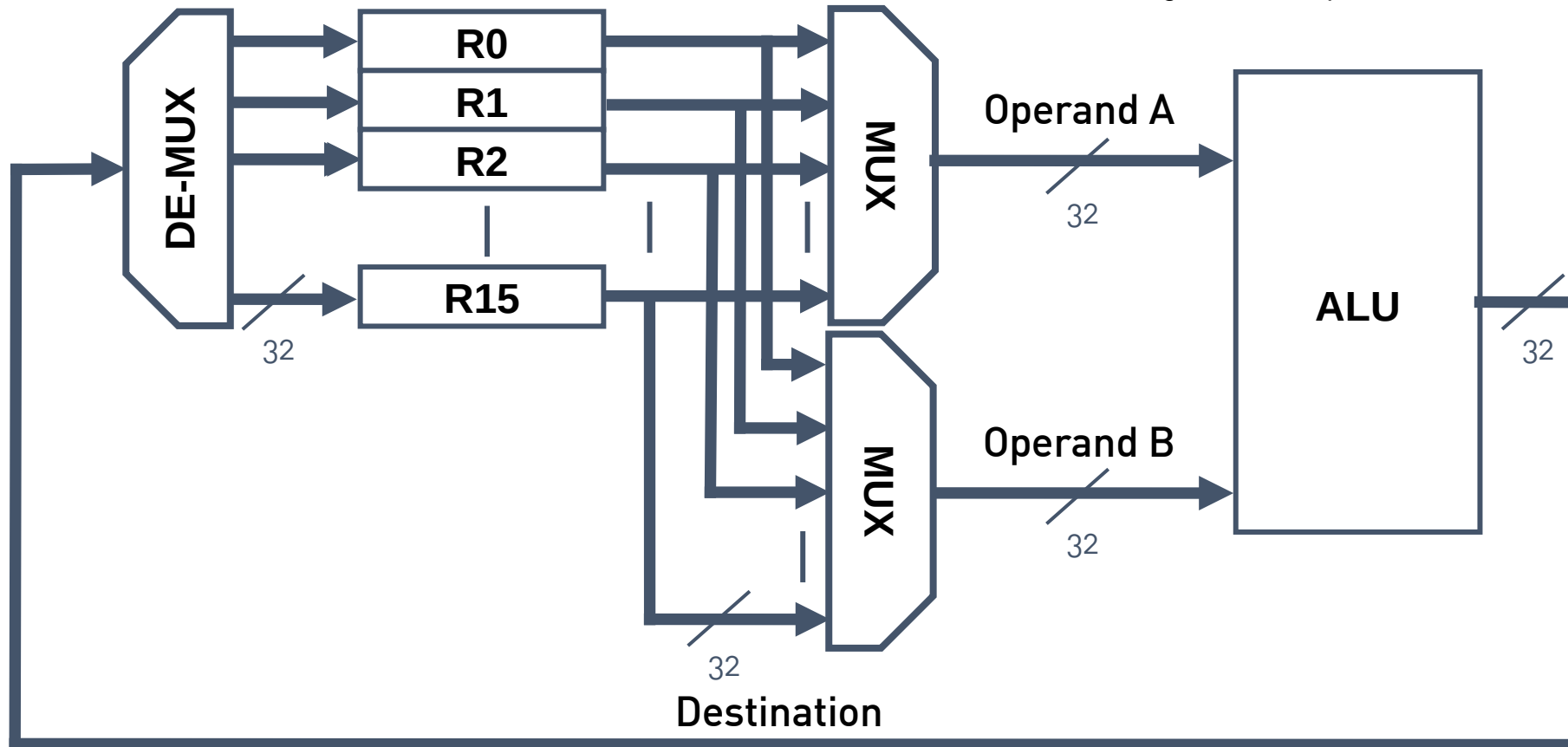
ALU - Algorithmic Logic Unit



# Aufbau der CPU: Datenverarbeitung

Die Register sind mit der ALU verbunden

Die Datenverarbeitung besteht darin die Werte aus dem 2 Register zum verknüpfen und das Ergebnis in einem 3te Registern zum speichern



Das Ergebnis der Verknüpfung je nach dem was wir mit dem ALU ausgewählt haben. Das Ergebnis wird wieder durch den Demultiplexer in den Registern wieder gespeichert.

# Aufbau der CPU: Hochsprache

- **Alle Anweisungen einer Hochsprache (z. B. C/C++) müssen in Operationen mit den 32-bit Registern und der ALU (Maschinensprache) übersetzt werden**
- **Übersetzung ist Aufgabe des Compilers**
- **In besonderen Fällen können Programmierer effizientere Lösungen finden**
  - Assembler-Programmierung (Programmierung in Maschinensprache)
  - (Details später)