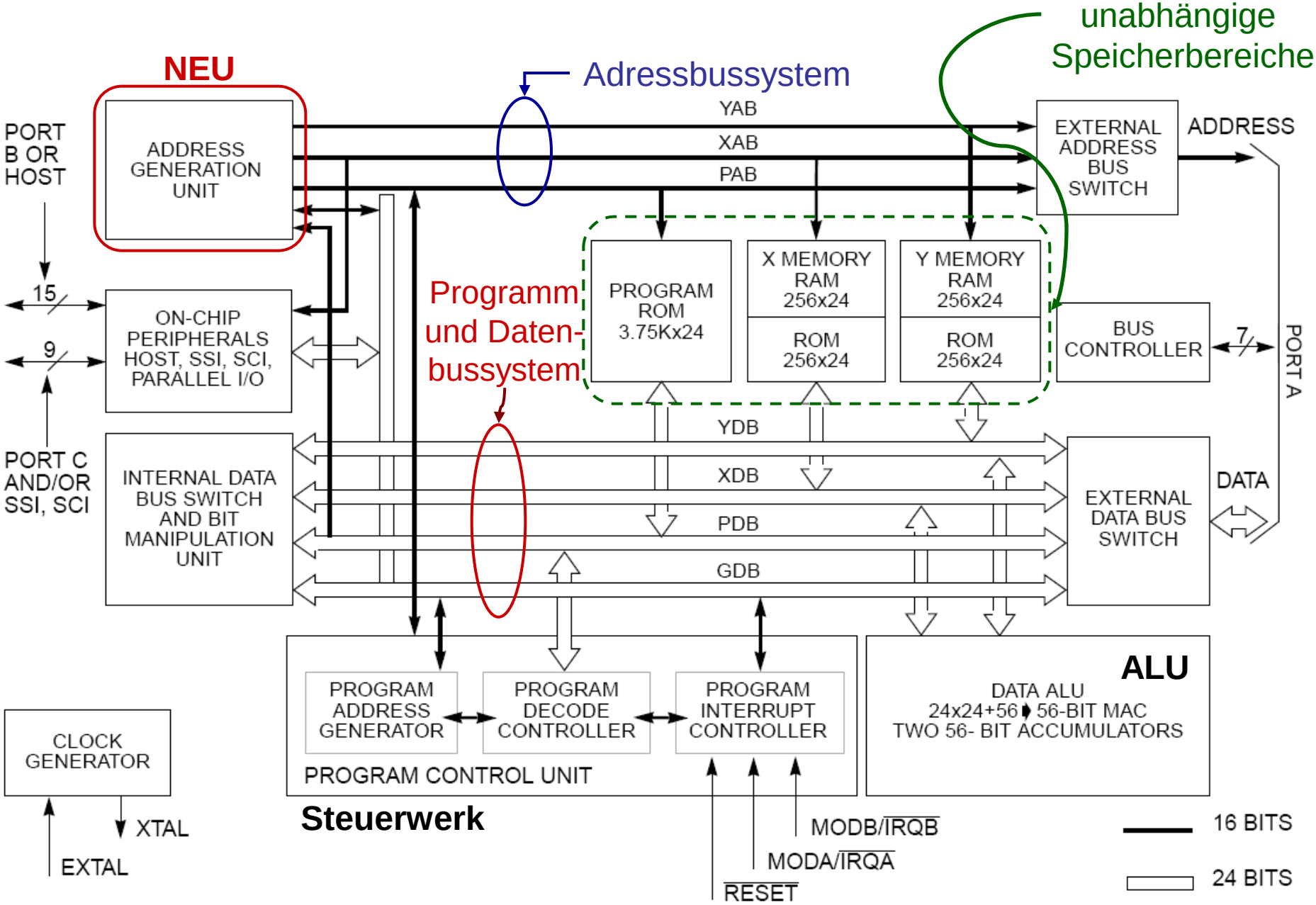
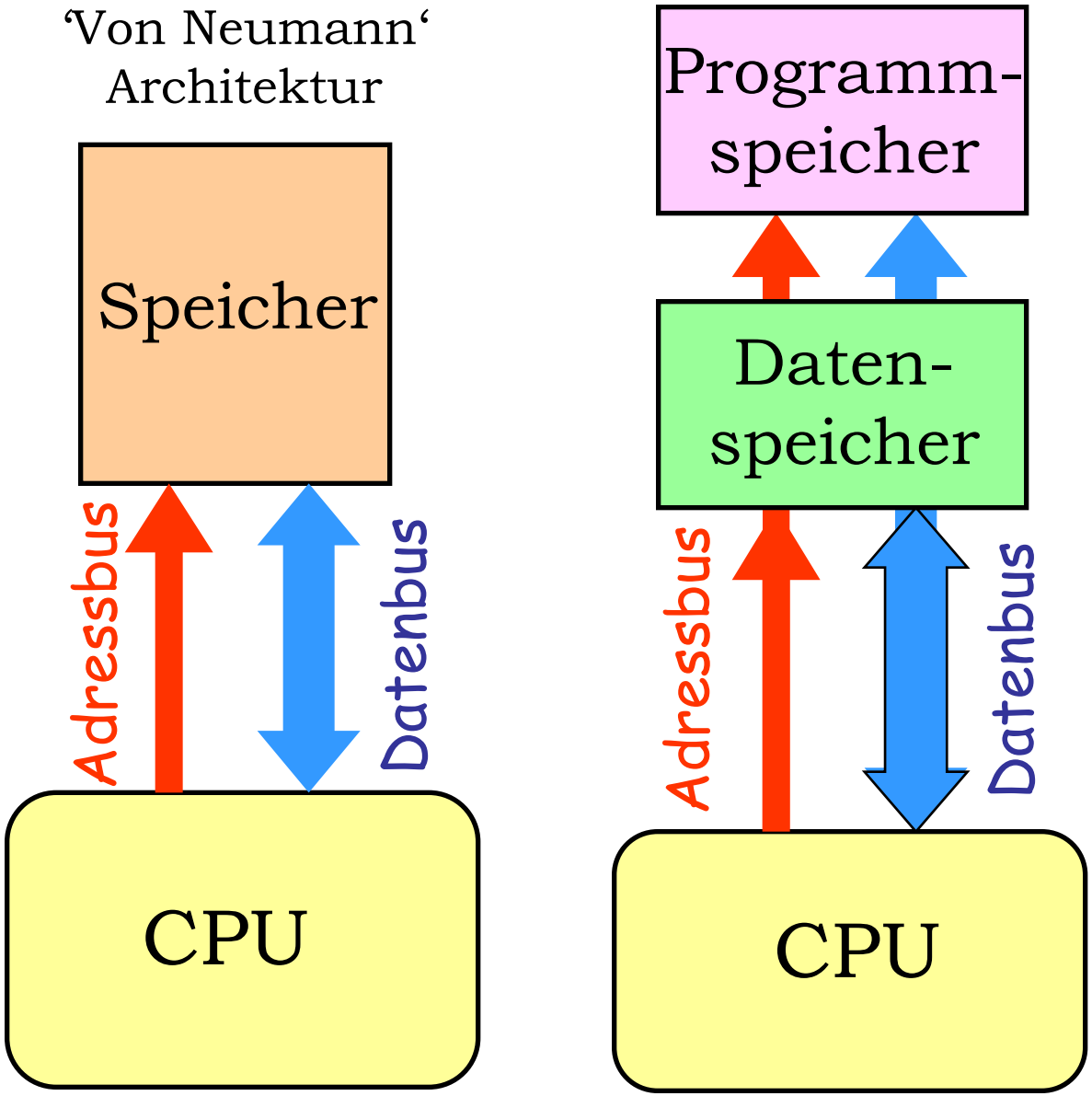
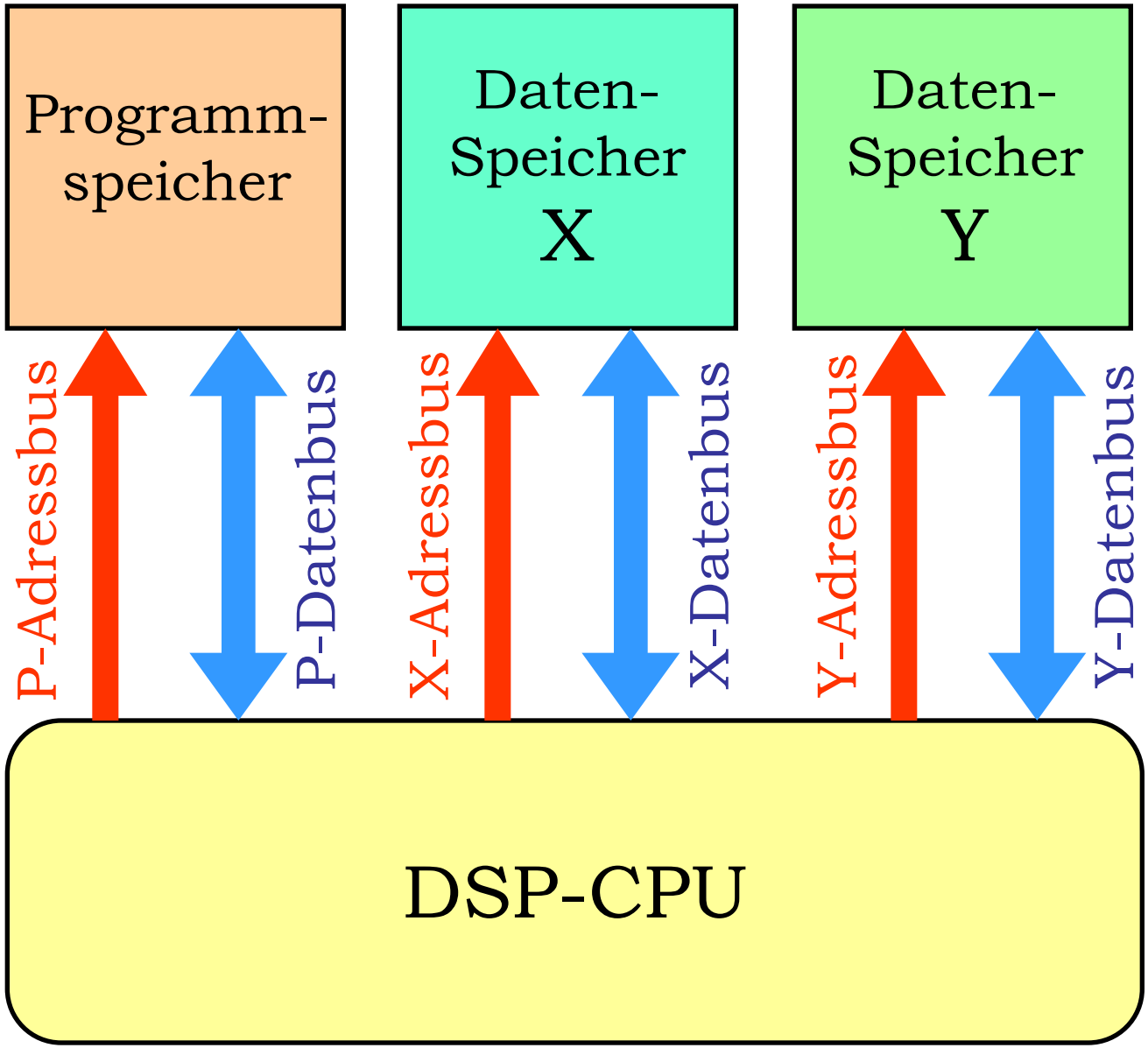
Übersichtsblockdiagramm des DSP 56000



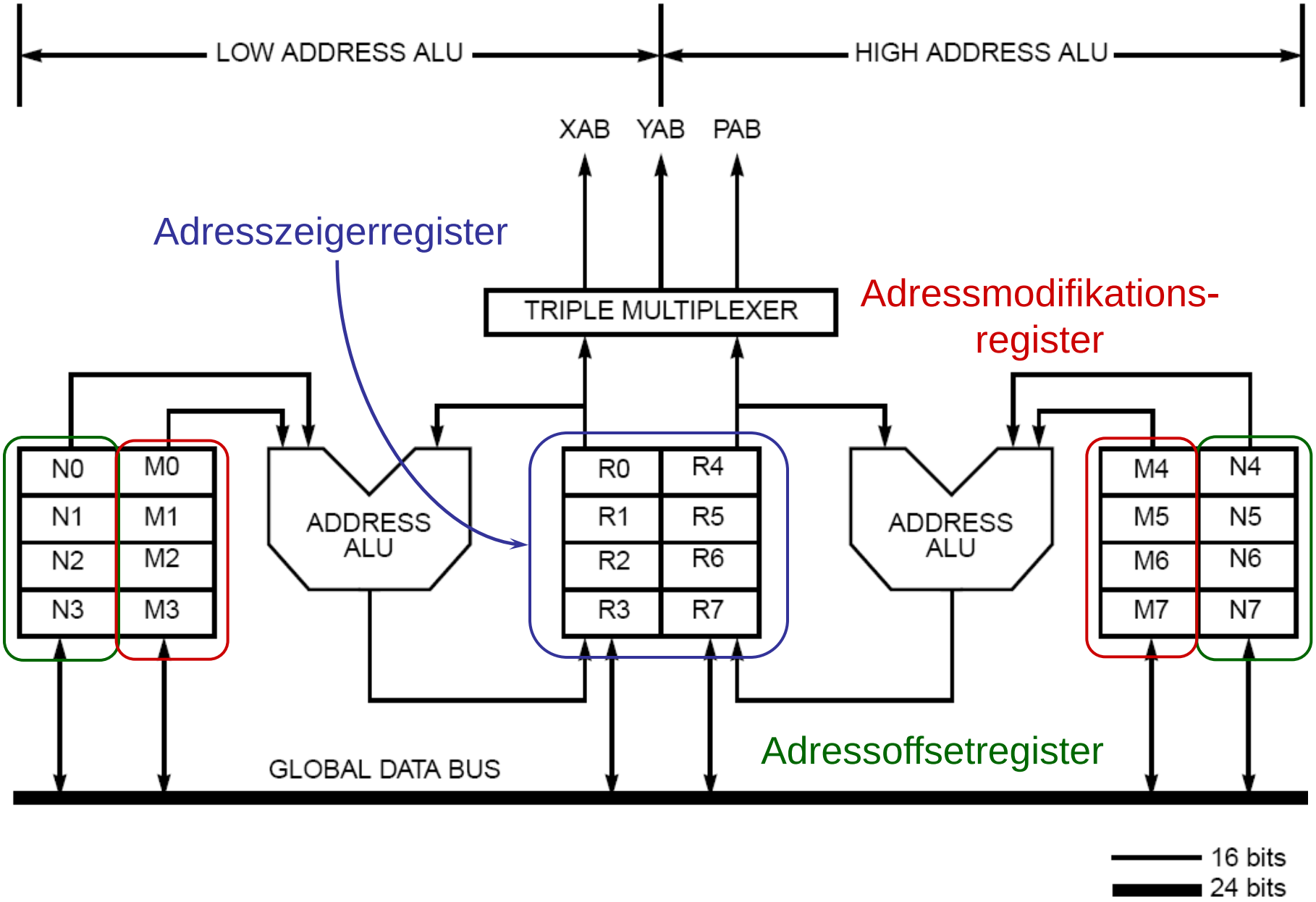
Mikroprozessor- und Mikro-controller-Speicherorganisation



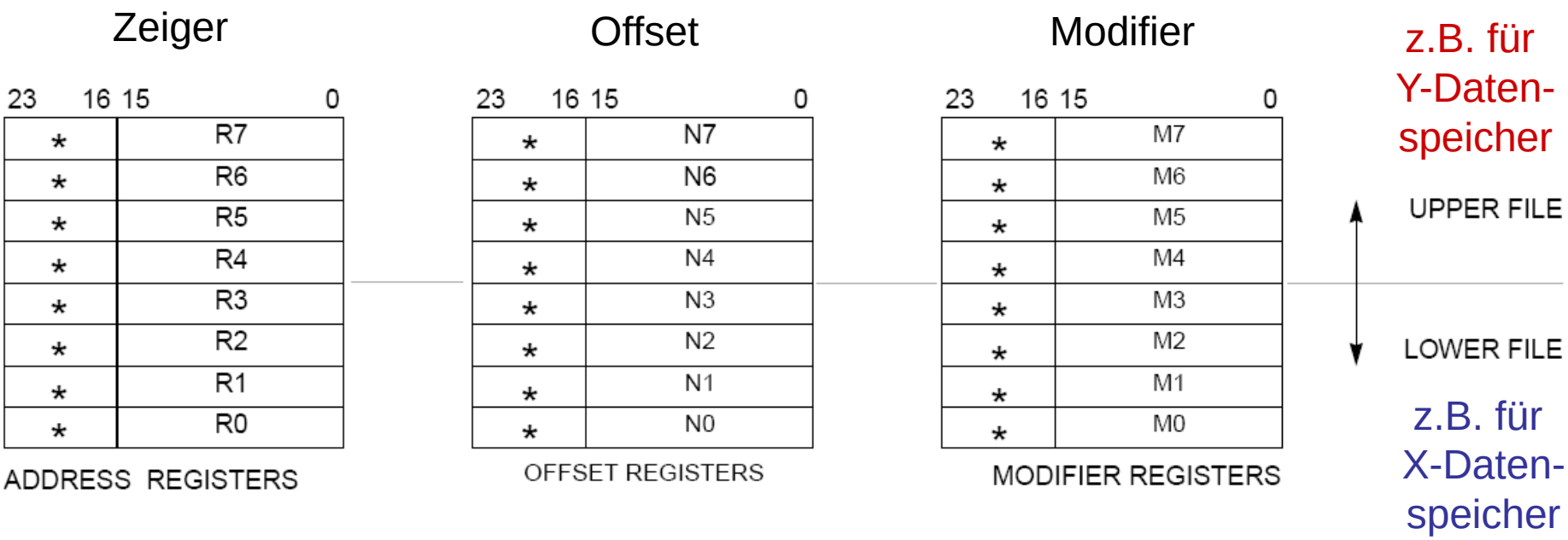
Harvard-Architektur eines digitalen Signalprozessors



Adressberechnung im DSP 5600x



Programmiermodell der AGU



Die wichtigste Adressierungsart: Adreßregister indirekt

Beispiele für indirekte Registeradressierung

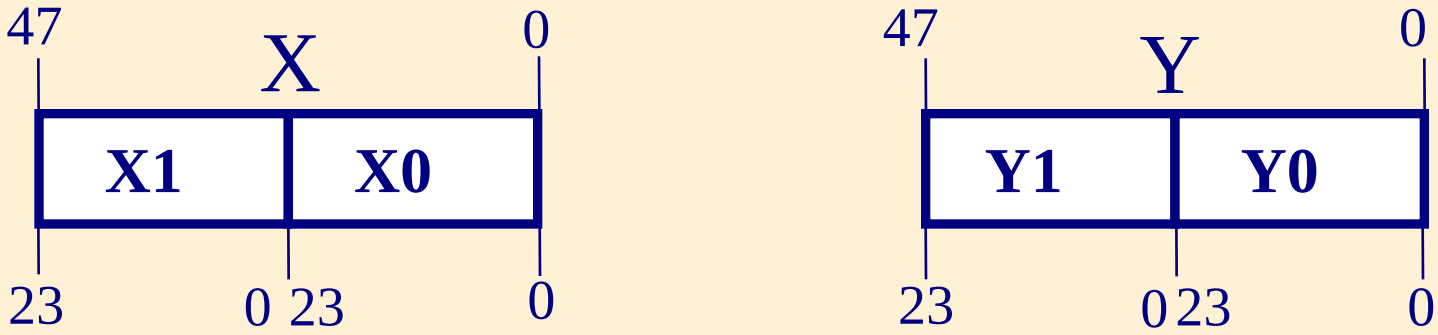
(Rn)	indirekt ohne Pointermanipulation
(Rn) +	Postinkrement
(Rn) –	Postdekrement
– (Rn)	Prädekrement
(Rn)+Nn	Postinkrement mit Offset
(Rn)–Nn	Postdekrement mit Offset
(Rn+Nn)	„indexed“ durch Offset: EA <= Inhalt von Rn+Nn

EA: Effektive Adresse

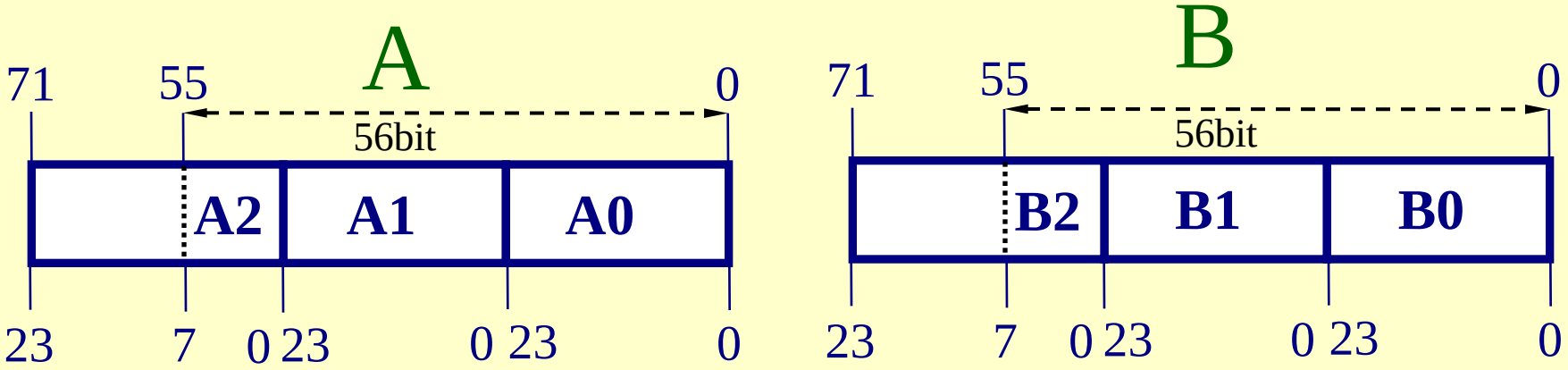
Anwendungsbeispiele in der Übung

Das ALU-Programmiermodell des DSP5600x

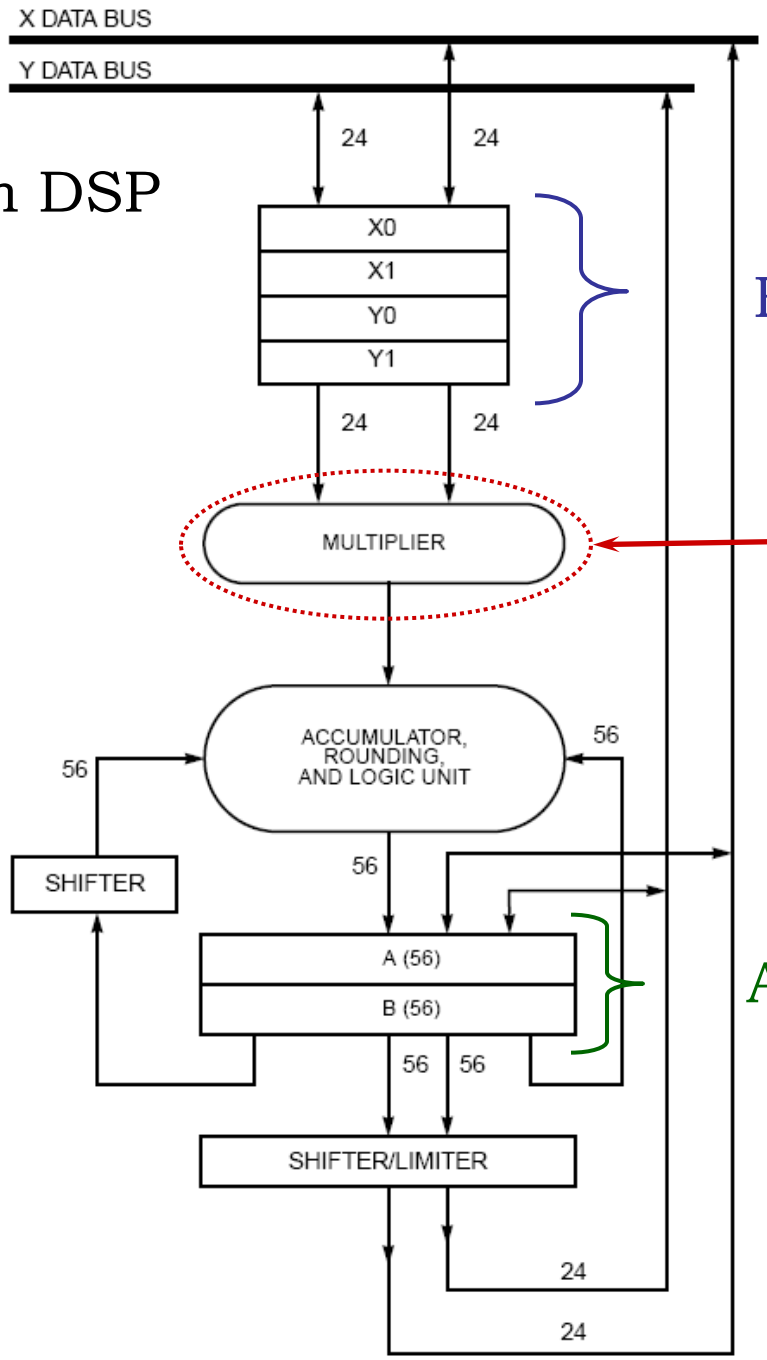
Eingaberegister (2-fach für komplexe FFT)



Akkumulatoren (2-fach für komplexe FFT)



Die Daten-ALU im DSP

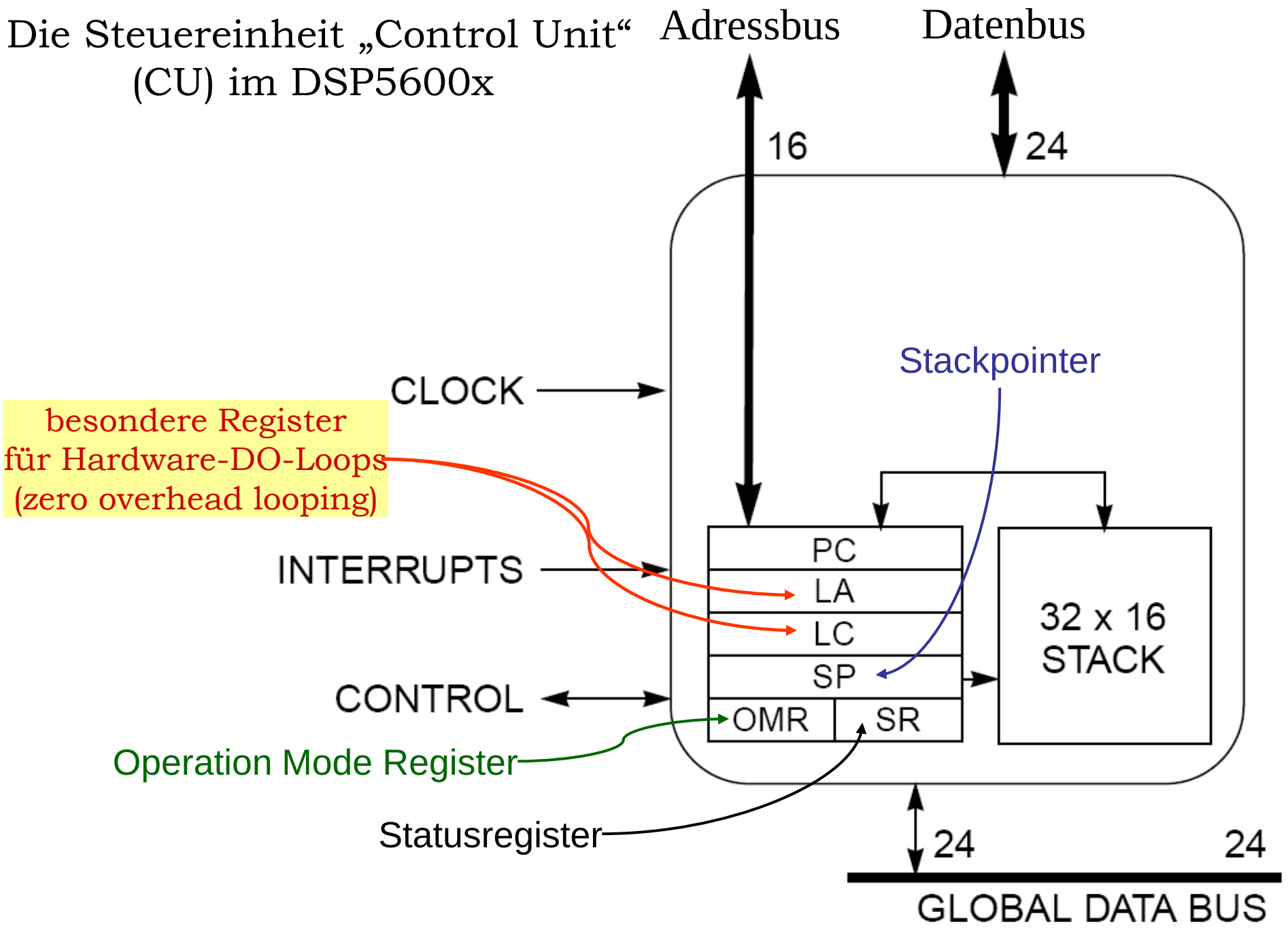


Eingaberegister

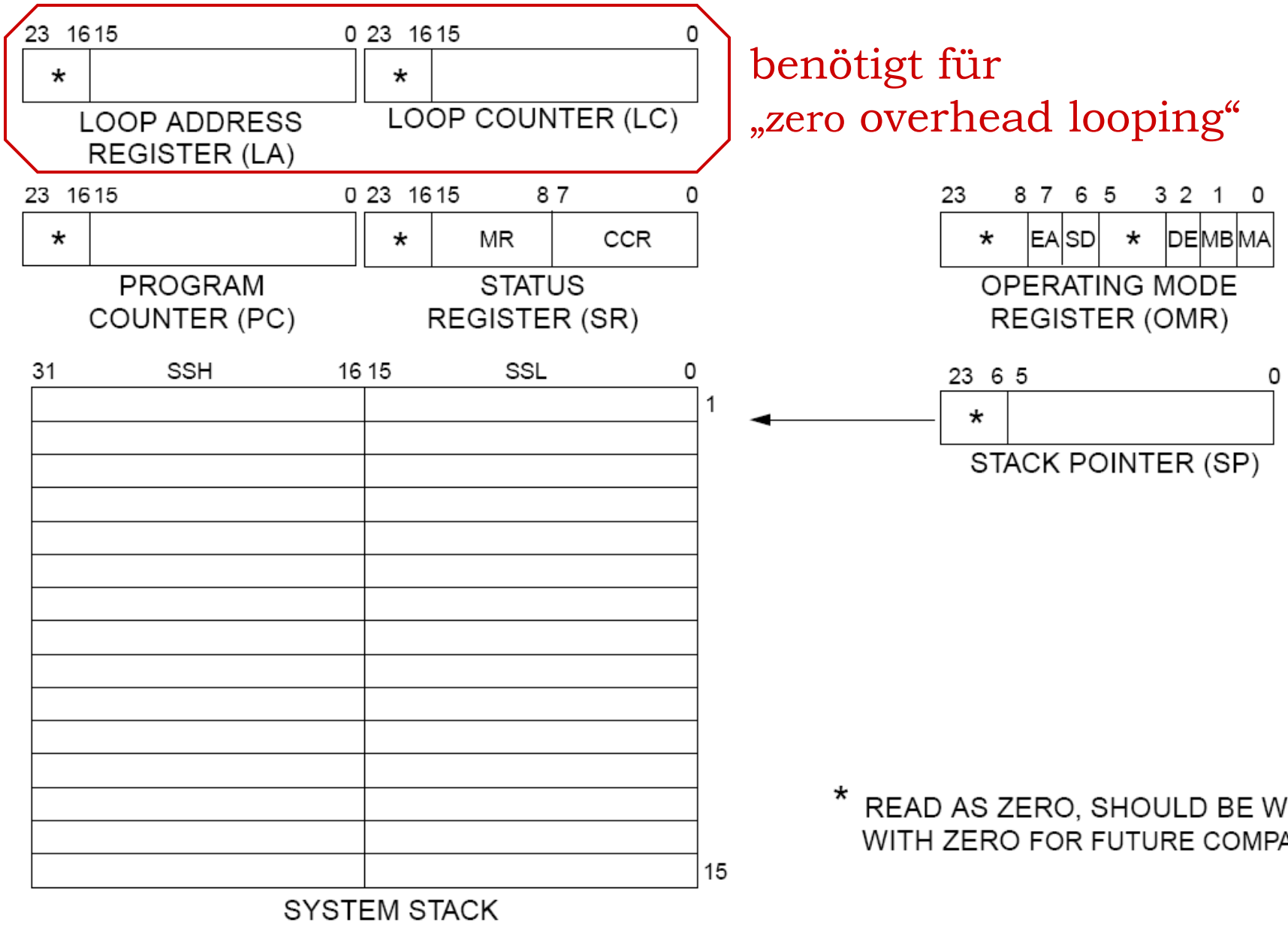
Parallelmultiplizierer

Akkumulatoren

Die Steuereinheit „Control Unit“ (CU) im DSP5600x



Das Programmiermodell der CU des DSP5600x



Nahezu alle Algorithmen der DSV erfordern die Abarbeitung von Schleifen, z.B. über genau N MAC-Operationen

Beispiel: Korrelation $C_{xy}(k) = \sum_{i=0}^{N-1} x(i) \cdot y(k+i)$

DSP_1_F2,5,7

Eine typische Software-Schleife könnte wie folgt aussehen:

FOR i=1 to N

1. Befehl ;bei jedem Befehl muss geprüft werden, ob die Schleife schon
. ;zu Ende ist und, falls ja, ob die Zahl N von Schleifen schon
. ;abgearbeitet ist => beides erfordert zusätzliche Maschinenzyklen

letzter Befehl

NEXT i

Im DSP sind **Hardware-DO-LOOPs** implementiert, so dass z.B. eine Korrelation mit N=1000 Abtastwerten wie folgt programmiert werden kann:

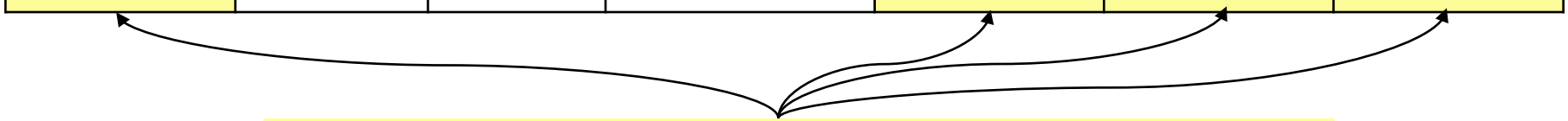
```
        DO #1000, END1
        MAC X0,Y0,A     X:(R0)+,X0    Y:(R4)+,Y0
END1
```

parallele Datentransfers mit 'Pointer-Update'

;das Korrelationsergebnis steht im Akku A

;zudem sind die 3 Bussysteme der Harvard-Architektur in Aktion

PC Program Counter	**SP** Stack Pointer	**SS** System Stack	**OMR** Operation Mode Reg.	**SR** Status Register	**LA** Loop Adresse	**LC** Loop Counter



kommen auf den STACK bei jedem DO-LOOP-Start

Details der Abarbeitung eines DO-LOOPs

```
DO      #N,     END1
        ⇓       ⇓
        LC      LA      ;letzter Befehl im LOOP
```

PC ⇒ **Stack** **;**markiert den LOOP-Anfang
SR ⇒ **Stack** ;Status – hier nicht von Belang
LA ⇒ **Stack** **;**für Verschachtelung (Nesting)
LC ⇒ **Stack** **;**für Verschachtelung (Nesting)

Folgende Schritte laufen im LOOP bei jedem OP-Code-Holen ab:

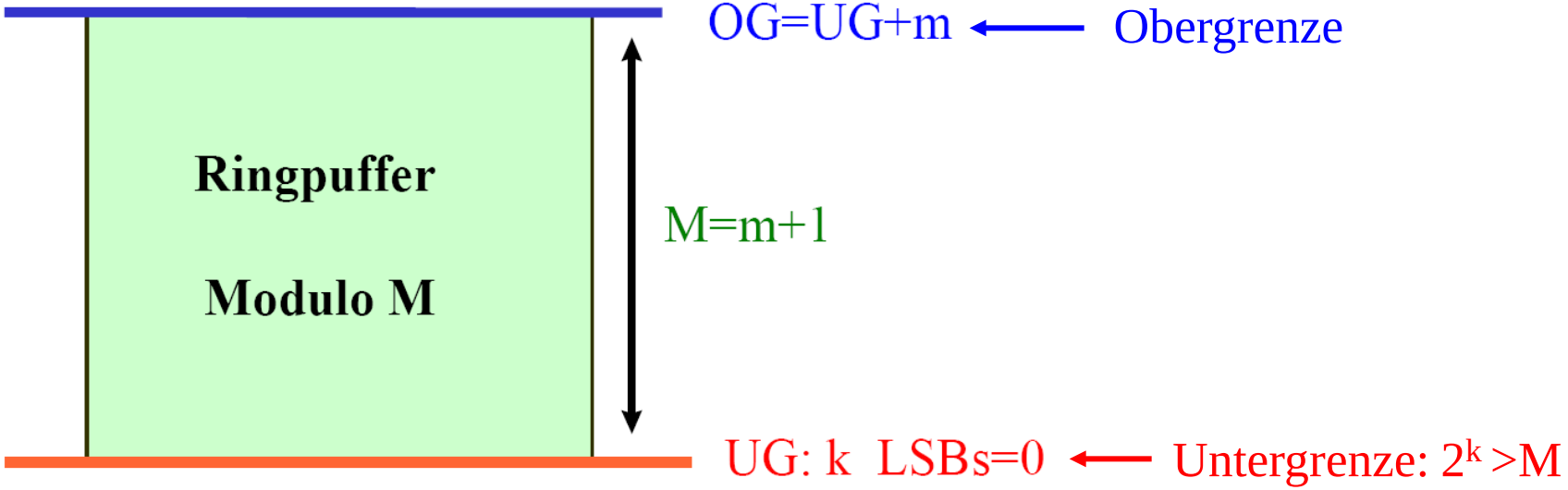
Vergleich: **PC ⇔ LA**
wenn **PC = LA** ;letzter LOOP-Befehl liegt vor
wenn **LC > 1** ;LC erniedrigen und weitermachen
LC:=LC-1

PC vom Stack holen und 1. LOOP-Befehl ausführen

Dieser Ablauf wiederholt sich bis
LC = 1 ;dann erfolgt noch genau ein Durchlauf

danach: **LA, LC, SR** vom Stack zurückholen und Programm bei **PC:=LA+1** fortsetzen

Register M_n	**Funktion**
0000	**Reverse-Carry ⇒ für FFT**
0001	**Modulo 2: 0,1, 0,1, 0,1…**
0002	**Modulo 3: 0,1,2, 0,1,2, 0,1,2…**
m	**Modulo(m+1): 0,1,2…m, 0,1,2…m, 0…**
…..	…..
32.767	**Modulo 32.768**
….. …..	**reserviert**
65.535 **$FFFF**	**Modulo 65.536 (linear)** ⇒ nach Reset



Beispiel zum Anlegen und Verwenden eines Ringpuffers

UG ist nicht identisch mit dem Pointer Rn, sondern erfordert Nullen in den unteren k Bits, mit $2^k > M$.

Registerbelegung:

```
MOVE #5,M2
MOVE #2,N2
MOVE #35,R2
```

$M=6 < 2^3=8 \Rightarrow k=3$ =>UG=xxxx000

Der Ringpuffer soll mit (R2)+N2 durchlaufen werden, d.h.:

```
MOVE    X0,X:(R2)+N2        ;wiederholte Instruktion
```

Der Pointer R2 liegt im Bereich $2^5=32....2^6=64$

Buffer-Untergrenze: $2^5=32$ (100000b) Obergrenze OG: 32+5 = 37

Ablauf: X0→X:(35), dann (R2)+N2 =>R2=37

X0→X:(37), dann (R2)+N2 =>R2=39: Buffer-Überlauf

Aktion: Modulus wird abgezogen: (R2)+N2-M => R2=37+2-6=33

X0→X:(33), dann (R2)+N2 =>R2=35

X0→X:(35), dann (R2)+N2 =>R2=37

weitere Erläuterungen zu Ringpufferoperationen

UG erfordert Nullen in den unteren k Bits, mit $2^k > M$.

Registerbelegung:

```
MOVE #5,M2
MOVE #2,N2
MOVE #35,R2          ;Pointer R2 liegt im Bereich 32...40
MOVE X0,X:(R2)+N2    ;wiederholte Instruktion
```

M=6 < 2^3=8 ⇒ k=3=> UG=xxxx000, d.h. 0,8,16,24,32,40,48,56,64…

```
0000 000        0
0001 000        8
0010 000        16
0011 000        24
0100 000        32
0101 000        40
0110 000        48
```

Buffer-Grenzen bei R2=35: UG: 2^5=32 => OG: 32+5=37

nicht verwendbare Adressen: 38, 39

Ablauf wenn R2=39: X0→X:(39), dann (R2)+N2 =>R2=41 (Überlauf)

Aktion: Modulus wird abgezogen: (R2)+N2-M => R2=39+2-6=35

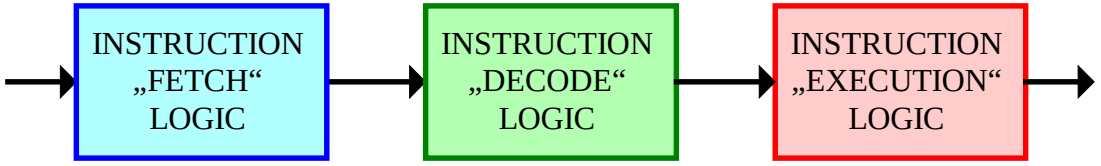
X0→X:(35), dann (R2)+N2 => R2=37

X0→X:(37), dann (R2)+N2 => R2=39

(R2)+N2-M => R2=37+2-6=33

X0→X:(33), dann (R2)+N2 => R2=35

Instruktionspipelining in einem DSP5600x

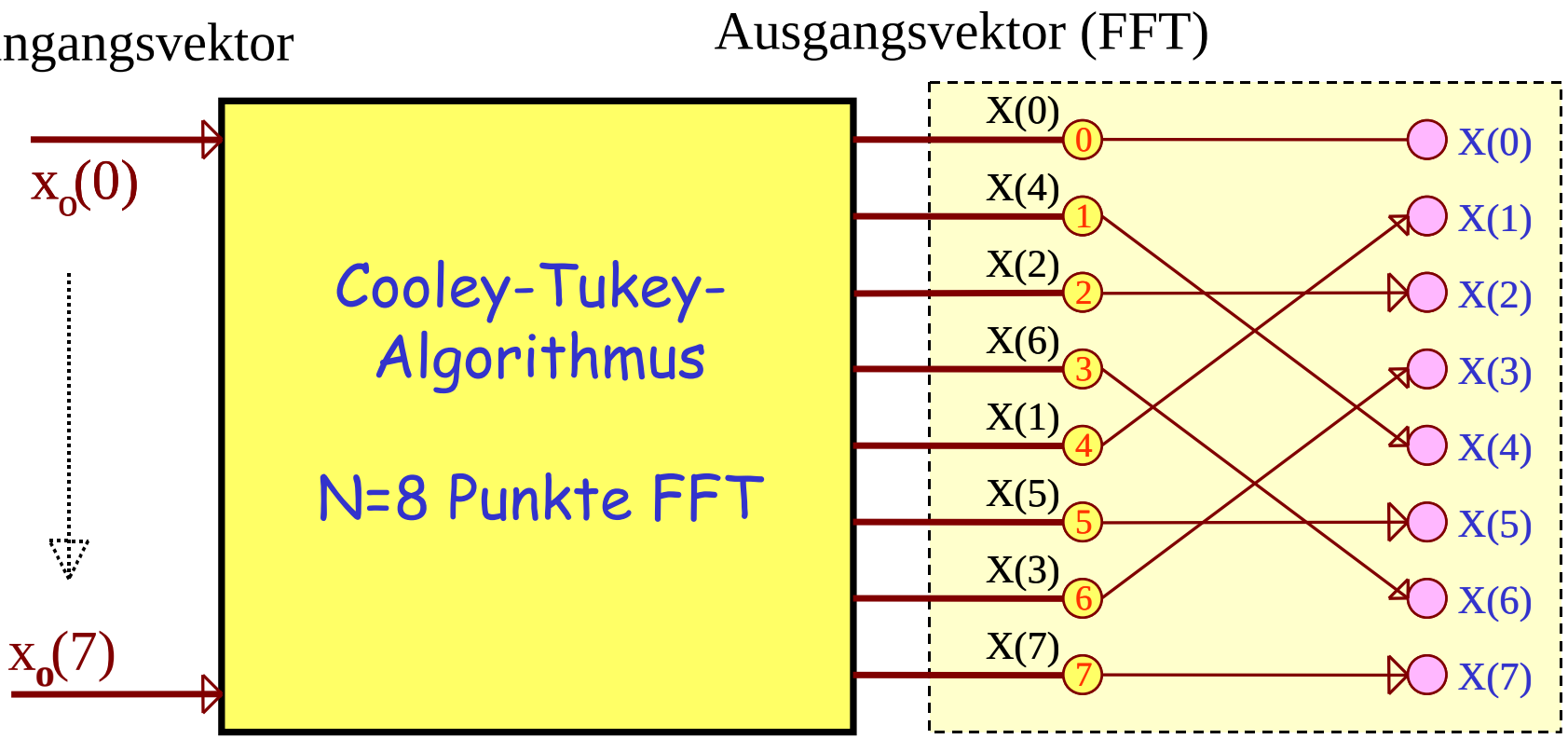


I_1	MACR	X0, Y1, A	X: (R0) +, X0	Y: (R4) +, Y1
I_2	CLR	A	X0, X: (R0) +	A, Y: (R4) -
I_3	MAC	X0, Y1, A	X: (R0) +, X0	Y: (R4) +, Y1

$$X0 \cdot Y1 = 5 \cdot 2^{-23} \cdot 8 \cdot 2^{-23} = 40 \cdot 2^{-46}$$
$$= 80 \cdot 2^{-47} = 50h \cdot 2^{-47} \Rightarrow A0{=}000050$$

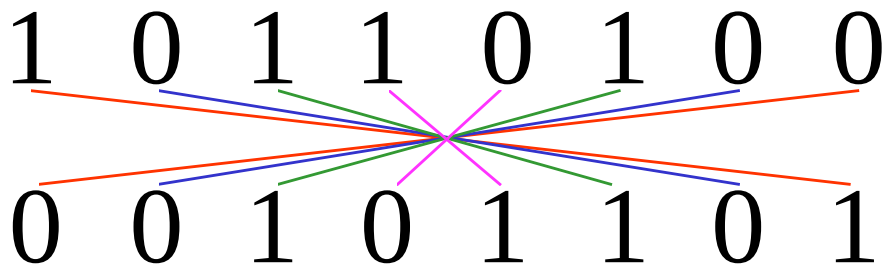
INSTRUCTION FETCH INSTRUCTION DECODE INSTRUCTION EXECUTION		I_1	I_2 I_1	I_3 I_2 I_1	I_4 I_3 I_2	I_5 I_4 I_3
parallel operations	initial conditions			**↓ 1 fertig**	**↓ 2 fertig**	**↓ 3 fertig**
address update (AGU)	R0=$0005 R4=$0008	———	———→ ———→	R0=5+1 R4=8+1	R0=6+1 R4=9-1	R0=7+1 R4=8+1
execution within the (DATA ALU)	A: A2=$00 A1=$000066 A0=$000000 X0=$400000 Y1=$000077	——— ——— ——— ——— ——— ———	———→ ———→ ———→ ———→ ——→ ———→	A: A2=$00 A1=$0000A2 A0=$000000 X0=$000005 Y1=$000008	A: A2=$00 A1=$000000 A0=$000000 X0=$000005 Y1=$000008	A: A2=$00 A1=$000000 A0=$000050 X0=$000007 Y1=$000008
X memory contents at address $0005 $0006 $0007	DATA $000005 $000006 $000007	———	———→	$000005 $000006 $000007	$000005 $000005 $000007	$000005 $000005 $000007
Y memory contents at address $0008 $0009	DATA $000008 $000009	———	———→	$000008 $000009	$000008 $0000A2	$000008 $0000A2

Vereinfachter Signalflussgraph für eine 8-Punkte FFT



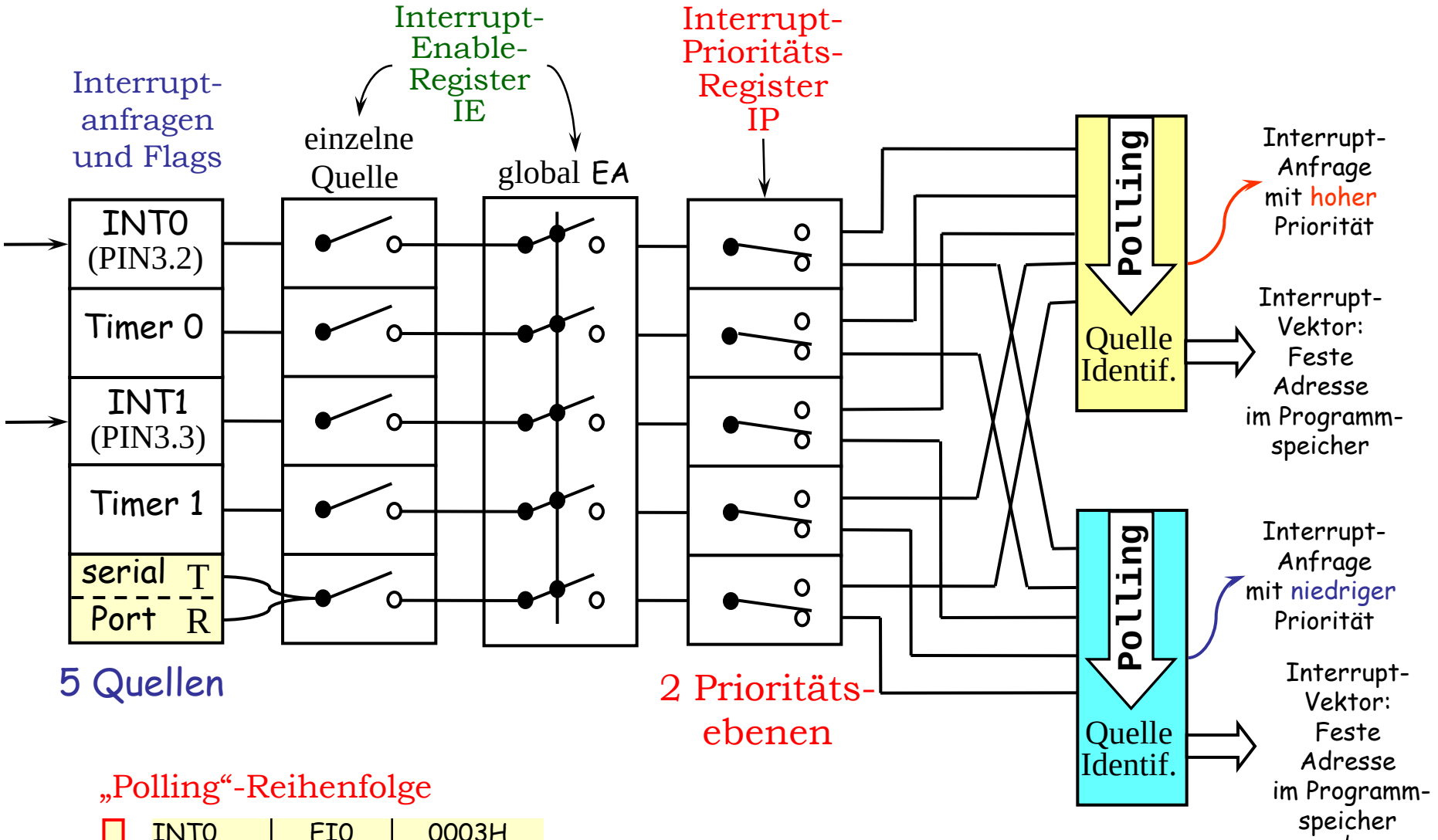
Umordnen mit Reverse-Carry

M_n=$0000



DSP_1 =>F20

Übersicht eines Interruptsystems am Beispiel des 8051-Mikrocontrollers



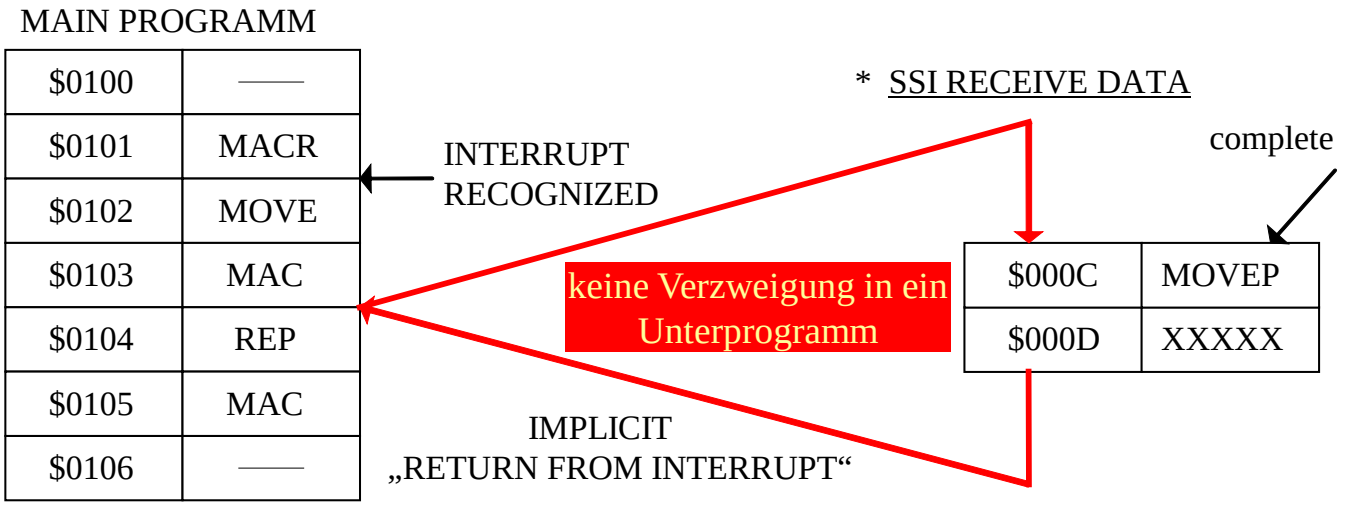
„Polling"-Reihenfolge

INT0	EI0	0003H
Timer 0	T0	000BH
INT1	EI1	0013H
Timer 1	T1	001BH
seriell	TI, RI	0023H

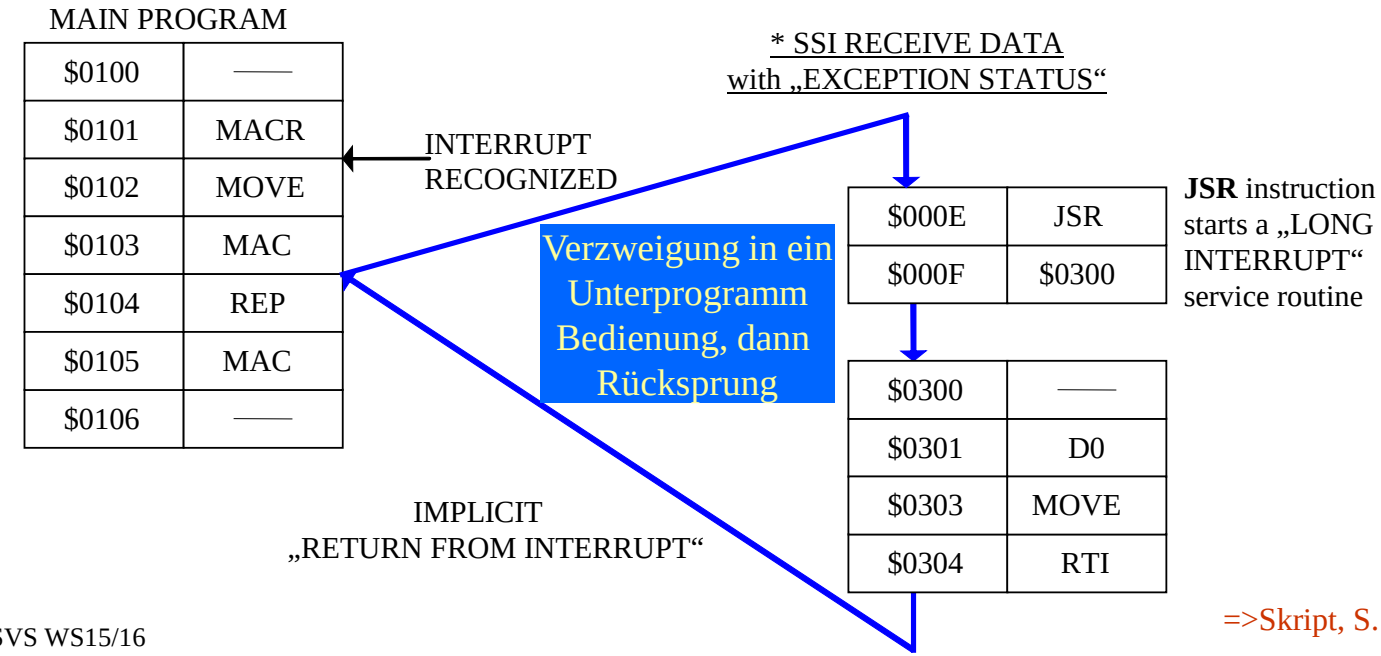
5 feste Interrupt-Vektor-adressen im Programmspeicher

Aspekte der Interruptverarbeitung in einem DSP

Prinzip der ‘FAST-INTERRUPT’- Bedienung



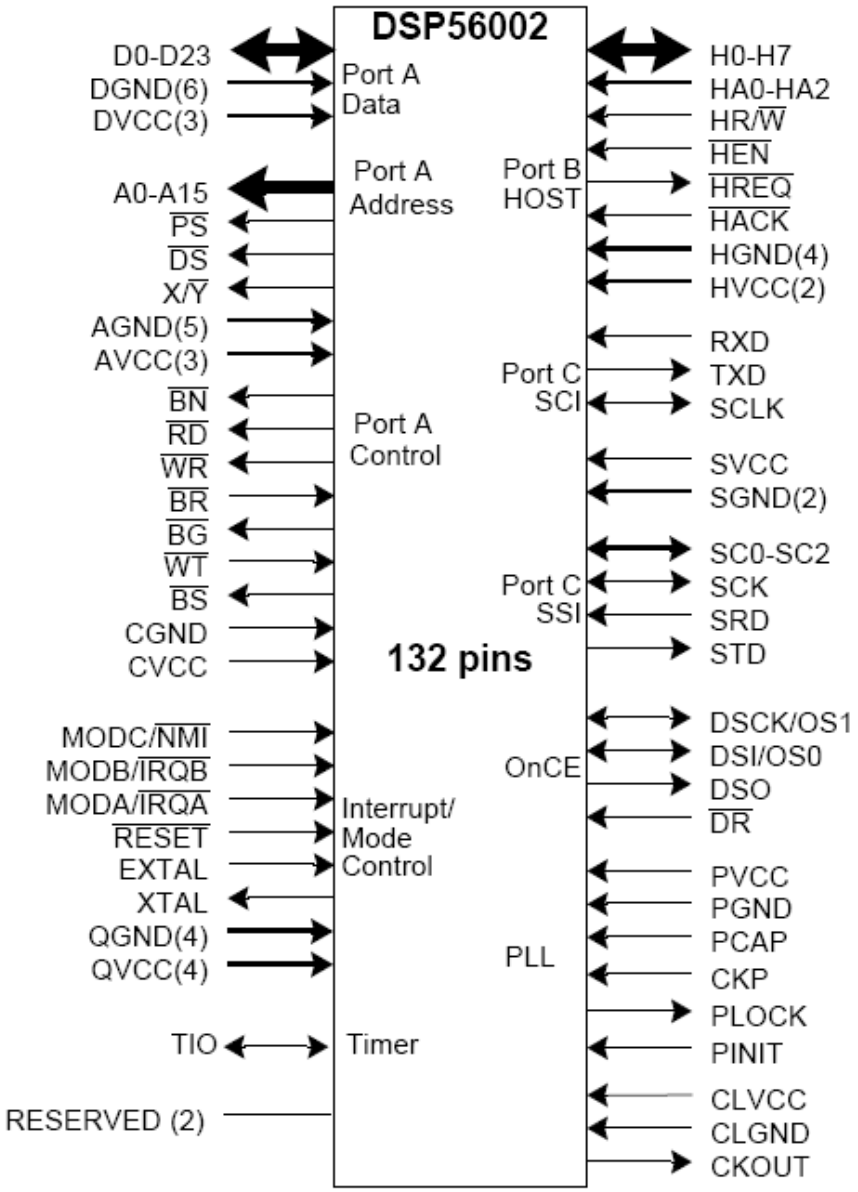
Beispiel einer ‘LONG-INTERRUPT’-Bedienung

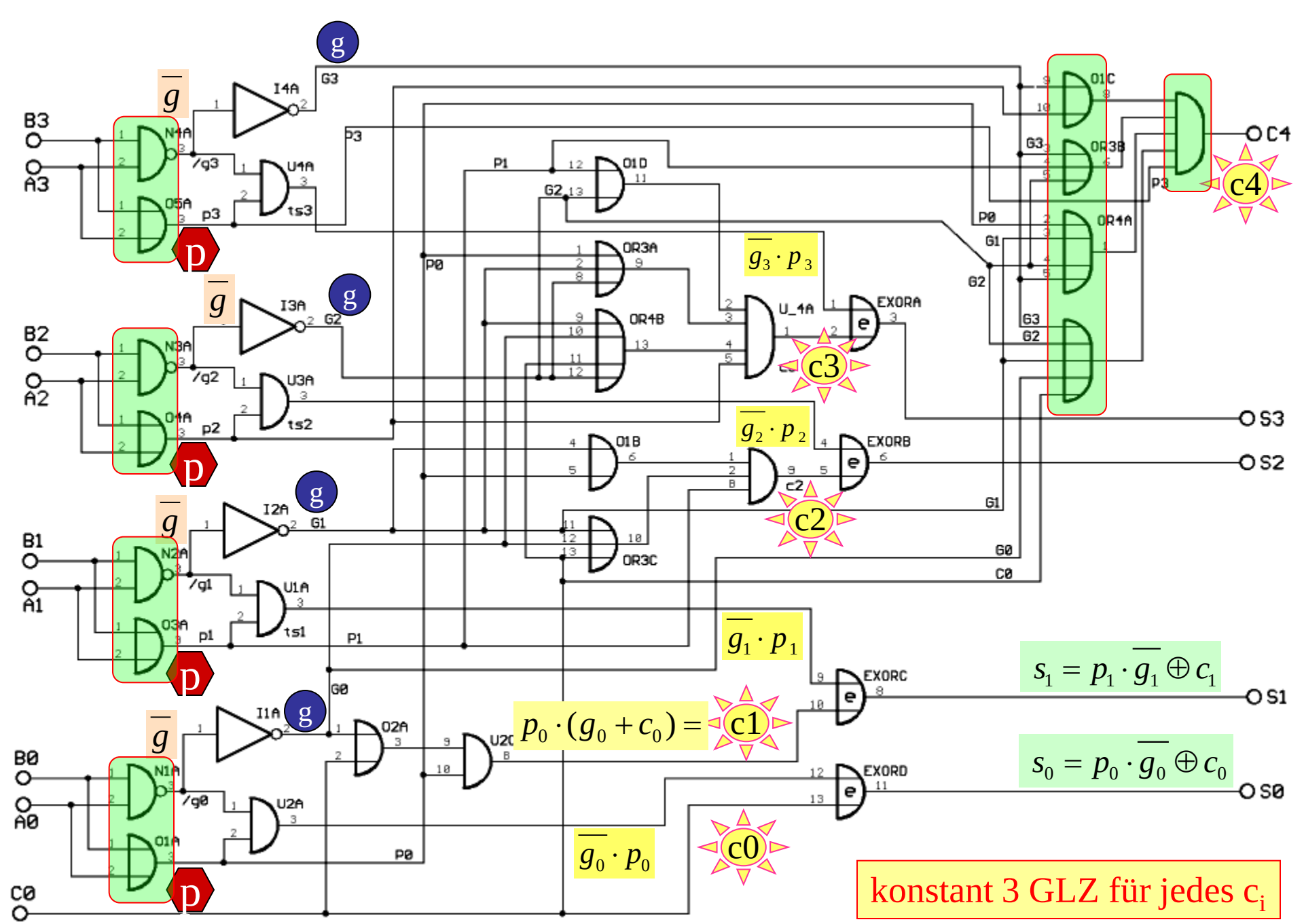


=>Skript, S. 56

DSP-Pinanordnung und Funktionsbeschreibung

Functional Group	Number of Pins
Port A Data Bus	24
Port A Address	19
Port A Bus Control	7
Port B Host Interface	15
Port C Synchronous Comm. Interface	3
Port C Synchronous Serial Interface	6
Interrupt and Mode Control	4
PLL and Clock	7
On-chip Emulation (OnCE)	4
Power (VCC)	16
Ground (GND)	24
Timer	1
Reserved	2
Total (for the PGA package)	132




B3
A3
B2
A2
B1
A1
B0
A0
C0
g
p
I4A
I3A
I2A
I1A
N4A
N3A
N2A
N1A
O5A
O4A
O3A
O1A
/g3
/g2
/g1
/g0
U4A
U3A
U1A
U2A
ts3
ts2
ts1
p3
p2
p1
p0
G3
G2
G1
G0
P3
P2
P1
P0
O1D
OR3A
OR4B
O1B
OR3C
O2A
U2C
U_4A
EXORA
EXORB
EXORC
EXORD
c3
c2
c1
c0
O1C
OR3B
OR4A
C4
S3
S2
S1
S0
$\overline{g_3} \cdot p_3$
$\overline{g_2} \cdot p_2$
$\overline{g_1} \cdot p_1$
$\overline{g_0} \cdot p_0$
$p_0 \cdot (g_0 + c_0) =$
$s_1 = p_1 \cdot \overline{g_1} \oplus c_1$
$s_0 = p_0 \cdot \overline{g_0} \oplus c_0$
konstant 3 GLZ für jedes c_i