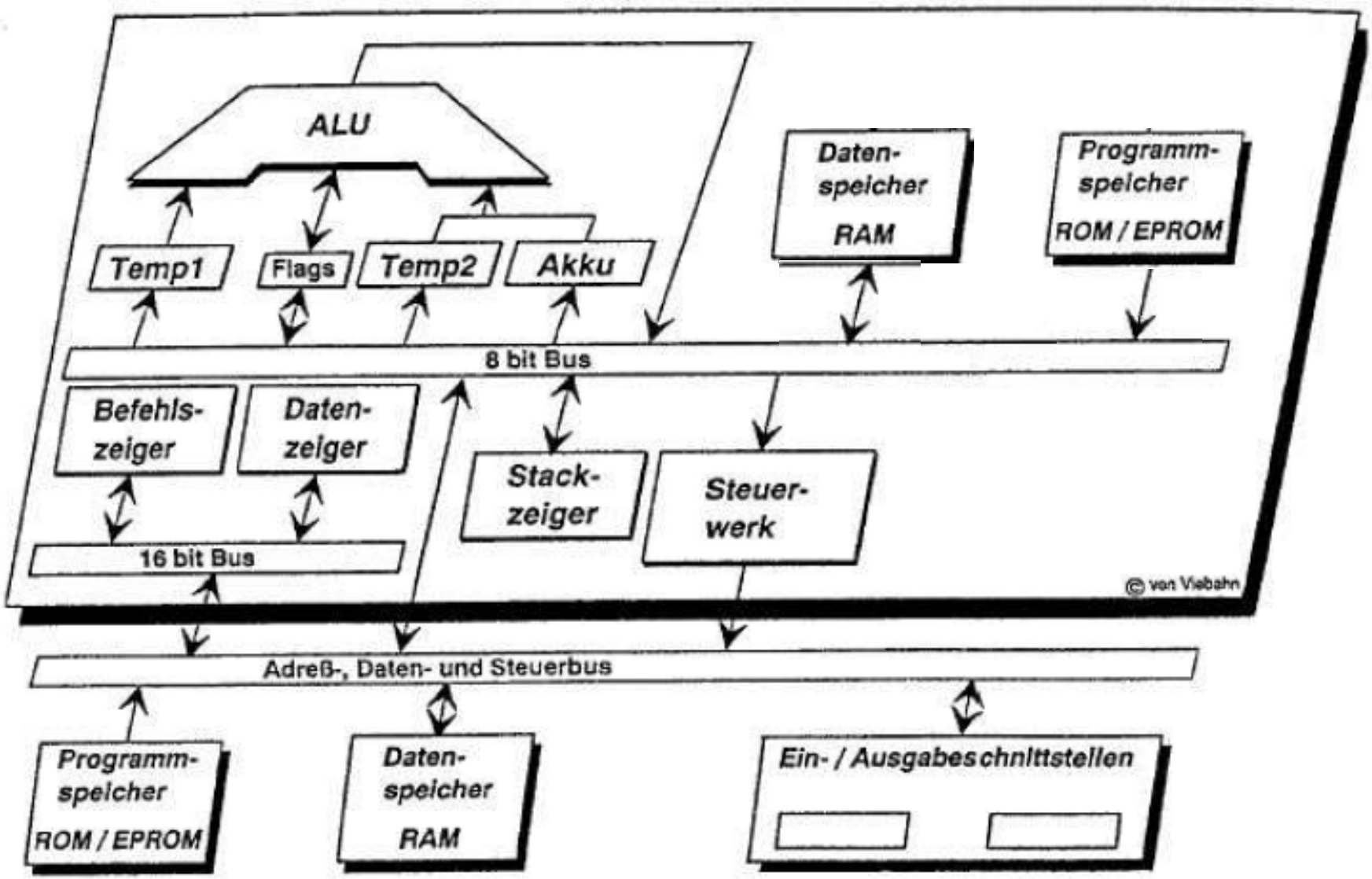
Kapitel 10

Adressräume

80c515c Systemstruktur



80c515c Systemstruktur

Programmspeicher

- Einige Modelltypen des 80c515c besitzen einen **internen** Programmspeicher
  - Im Prozessorchip integriert
  - Größe 64 kB ROM
  - Kann deaktiviert werden.
  - One-Time-Write (OTW) Funktion ermöglicht Programmierung (nur einige Modelltypen)
- **Externer** Programmspeicher
  - Optional
  - ROM-Baustein(e) am externen Systembus angeschlossen
  - Bis zu 64 kB max.
- **Entweder** ein interner Programmspeicher **oder** ein externer Programmspeicher wird benutzt.
  - $\overline{EA}$ Pin = 5V : Steuerwerk benutzt den internen Programmspeicher.
  - $\overline{EA}$ Pin = Masse : Steuerwerk benutzt den externen Programmspeicher

Datenspeicher

- **Interner Datenspeicher**
  - Standard
  - Im Prozessorchip integriert
  - 256 By SRAM
- **Externer Datenspeicher (XRAM)** – 2 Varianten
  1. Im 80c515c Prozessorchip integriert
     - 2 kB RAM
     - Adressbereich 0xF800 … 0xFFFF
     - Aktiviert mit Konfigurationsbit XMAP0 = '0' (siehe User's Manual 3.4.1)
  2. Externe Baustein(e)
     - Am externen Systembus angeschlossen
     - Bis 64 kB max.
     - Aktiviert mit Konfigurationsbit XMAP0 = '1' (siehe User's Manual 3.4.1)
  - Adressierung des XRAM ist für beide Varianten gleich
  - Auch der Betrieb der Kombination integriertes XRAM und extern angeschlossenens XRAM ist möglich.
- Separate Befehle zum Zugriff auf den internen 256 Byte großen Datenspeicher und das XRAM.
  - Die Befehle zum Zugriff auf den internen Datenspeicher haben wir bereits behandelt.

Adressierung des internen Datenspeichers

Die Zugriffsmechanismen auf Bytes im Internen Datenspeicher wurde im Kapitel *8051 Basisbefehle* behandelt.

Special Function Registers

- Der 8051 Prozesssor, insbesondere das Steuerwerk, enthält viele Speicherelemente (Latches und Hardwareregister) für die Steuerung des gesamten Prozessors, z.B.
  - B-Register für die MUL und DIV Befehle
  - Ports
  - Konfigurationsdaten wie z.B. SYSCON (s.u.)
- Die Informationen in diesen Speicherelementen müssen zum großen Teil auch per Maschinenbefehl zugreifbar sein.
- Zu diesem Zweck wird für diese Informationen ein 128 Byte großer Adressbereich, die sog. **Special Function Registers (SFRs)** zur Verfügung gestellt.
  - Diese 128 SFRs sollen über eine 8 Bit Adresse wie Bytes im internen 8051 Datenspeicher adressiert werden können.

Konflikt : 256 Bytes Datenspeicher **+** 128 Bytes SFRs sollen über 256 Adresswerte angesprochen werden

Lösung : Die **Zugriffsart** wird zusätzlich für die Auswahl zwischen Datenspeicher und SFRs eingesetzt :

- Den **SFRs** wird der Adressbereich 0x80 ... 0xFF zugewiesen, **und** die SFRs sind nur erreichbar über **direkte** Adressspezifikationen (z.B.: MOV A,0x81).
- Die **oberen 128 Bytes des integrierten Datenspeichers** (Addressen 80H ... 0FFH) können nur mit **indirekter** Adressierung angesprochen werden. (MOV R0,#81 MOV A,@R0)
- Die **unteren 128 Bytes des integrierten Datenspeichers** (Adressen 0x00 ... 0x7F) können sowohl mit direkter als auch indirekter Adressierung angesprochen werden.

SFR Adressierung

	direkte Adressierung	indirekte Adressierung
0x80 … 0xFF	SFRs	Datensp.
0x00 … 0x7F	Datensp.	Datensp.

- Beispiele :

```
MOV   A,0xF0      ;B-Reg in denAkku

MOV   R1,0xF0     ;Inhalt von DS 0xF0
MOV   A,@R1       ;in den Akku

MOV   A,0x40      ;Inhalt von DS 0x40
                  ;in den Akku

MOV   R1,0x40     ;Inhalt von DS 0x40
MOV   A,@R1       ;in den Akku
```

- Die Adressen der Special Function Registers sind fest in der Hardware verdrahtet.
  - z.B. das B-Register hat die Adresse 0xF0
  - Die KEIL Entwicklungsumgebung enthält bereits Symbole für alle SFRs.

Table 3
Special Function Registers - Functional Blocks

Block	Symbol	Name	Address	Contents after Reset
CPU	ACC	Accumulator	**$E0_H$** 1)	00_H
	B	B-Register	**$F0_H$** 1)	00_H
	DPH	Data Pointer, High Byte	83_H	00_H
	DPL	Data Pointer, Low Byte	82_H	00_H
	DPSEL	Data Pointer Select Register	92_H	$XXXXX000_B$ 3)
	PSW	Program Status Word Register	**$D0_H$** 1)	00_H
	SP	Stack Pointer	81_H	07_H
	SYSCON2)	System Control Register	$B1_H$	$X010XX01_B$ 3)
A/D-Converter	ADCON0 2)	A/D Converter Control Register 0	**$D8_H$** 1)	00_H
	ADCON1	A/D Converter Control Register 1	DC_H	$0XXXX000_B$ 3)
	ADDATH	A/D Converter Data Register High Byte	$D9_H$	00_H
	ADDATL	A/D Converter Data Register Low Byte	DA_H 4)	$00XXXXXX_B$ 3)
Interrupt System	IEN0 2)	Interrupt Enable Register 0	**$A8_H$** 1)	00_H
	IEN1 2)	Interrupt Enable Register 1	**$B8_H$** 1)	00_H
	IEN2	Interrupt Enable Register 2	$9A_H$	$XX00X00X_B$ 3)
	IP0 2)	Interrupt Priority Register 0	$A9_H$	00_H
	IP1	Interrupt Priority Register 1	$B9_H$	$0X000000_B$ 3)
	TCON 2)	Timer Control Register	**88_H** 1)	00_H
	T2CON 2)	Timer 2 Control Register	**$C8_H$** 1)	00_H
	SCON 2)	Serial Channel Control Register	**98_H** 1)	00_H
	IRCON	Interrupt Request Control Register	**$C0_H$** 1)	00_H
XRAM	XPAGE	Page Address Register for Extended on-chip XRAM and CAN Controller	91_H	00_H
	SYSCON2)	System Control Register	$B1_H$	$X010XX01_B$ 3)
Ports	P0	Port 0	**80_H** 1)	FF_H
	P1	Port 1	**90_H** 1)	FF_H
	P2	Port 2	**$A0_H$** 1)	FF_H
	P3	Port 3	**$B0_H$** 1	FF_H
	P4	Port 4	**$E8_H$** 1)	FF_H
	P5	Port 5	**$F8_H$** 1)	FF_H
	DIR5	Port 5 Direction Register	**$F8_H$** 1) 4)	FF_H
	P6	Port 6, Analog/Digital Input	DB_H	–
	P7	Port 7	FA_H	$XXXXXXX1_B$ 3)

Programm Status Wort – PSW

Special Function Register PSW (Address $D0_H$) **Reset Value : 00_H**

Bit No.	MSB							LSB	
	$D7_H$	$D6_H$	$D5_H$	$D4_H$	$D3_H$	$D2_H$	$D1_H$	$D0_H$	
$D0_H$	CY	AC	F0	RS1	RS0	OV	F1	P	PSW

Bit	Function
CY	Carry Flag Used by arithmetic instruction.
AC	Auxiliary Carry Flag Used by instructions which execute BCD operations.
F0	General Purpose Flag
RS1 RS0	Register Bank select control bits These bits are used to select one of the four register banks.
OV	Overflow Flag Used by arithmetic instruction.
F1	General Purpose Flag
P	Parity Flag Set/cleared by hardware after each instruction to indicate an odd/even number of "one" bits in the accumulator, i.e. even parity.

RS1	RS0	Function
0	0	Bank 0 selected, data address 00_H-07_H
0	1	Bank 1 selected, data address 08_H-$0F_H$
1	0	Bank 2 selected, data address 10_H-17_H
1	1	Bank 3 selected, data address 18_H-$1F_H$

80c515c User's Manual Seite 2 – 4

Übungsaufgabe AR1 :

Kopieren Sie den Inhalt des PSW in die Adresse 0x99 des internen Datenspeichers !

Direkt adressierbarer Datenspeicheradressraum

	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
F...	B-Reg															
E...	ACC															
D...	PSW															
C...																
B...		SYSCON														
A...																
9...			DPSEL													
8...			DPL	DPH												
7...																
6...																
5...																
4...																
3...																
2...																
1...	Register Bank 2								Register Bank 3							
0...	Register Bank 0								Register Bank 1							

- 0x00 ... 0x7F : Datenspeicher
- 0x80 ... 0xFF : Special Function Registers (SFRs)
- Bit-adressierbare Bereiche

General Purpose Register

- Register sind häufig in schnellerer Technologie implementiert als der Hauptspeicher (nicht beim 8051).
- Mit Registern ergibt sich kompakterer Code, denn die Registeradresse ist bereits Teil des OpCodes.
  - z.B.: MOV A, R7 benötigt 1 Byte
  - MOV A, 37 benötigt 2 Bytes
- Beim 8051 sind die Register in den internen Datenspeicher eingeblendet (gemapped)
- Im 80c515c gibt es 4 * 8 Register : (R0 … R7) in 4 **Register-Bänken**
  - Datenspeicher-Adresse 0x00 ... 0x07 : **Register Bank 0** – R0 ... R7
  - Datenspeicher-Adresse 0x08 ... 0x0F : **Register Bank 1** – R0 ... R7
  - Datenspeicher-Adresse 0x10 ... 0x17 : **Register Bank 2** – R0 ... R7
  - Datenspeicher-Adresse 0x18 ... 0x1F : **Register Bank 3** – R0 ... R7
- **Genau eine** Registerbank ist zu jedem Zeitpunkt **aktiv.**
- Die aktive Register Bank wird spezifiziert durch **PSW Bits 3, 4 (RS1, RS0)**
  - Änderung der PSW Bits 3, 4 (z.B. mit einem MOV Befehl) schaltet auf eine andere Registerbank um.
  - PSW ist das SFR mit Adresse 0xD0
- Sinnvolle Verwendung von mehreren Register Bänken
  - im Fall von mehreren parallel laufenden Programmkomponenten, z.B. Interrupt Routinen
- Was tut der Befehl : MOV R1,0 ?

Bitadressierbarer Bereich

- Die kleinste **ansprechbare Einheit** in Computerspeichern ist ein Byte.
- Darüber hinaus sind beim 8051 Prozessor die **Bits** einiger ausgewählter Bytes auch individuell **adressierbar**.
- Dafür sind spezielle Befehle, die **Bitbefehle** vorgesehen, z.B. SETB, CLR, ...
- Beim 8051 gibt es **256 direkt adressierbare Bits**.
  - Die Bitadresse (badr) in Bitbefehlen ist also 8 Bits groß
    - **Bits 7 ... 3** der Bitadresse identifizieren das Byte, in dem das Bit enthalten ist.
      - Es gibt also 2^5 = 32 Bytes im internen Datenspeicher, die bit-adressierbar sind.
    - **Bits 2 ... 0** spezifizieren die Bitposition innerhalb des Bytes.
  - Von den 256 Bits sind die 128 Bits in den Datenspeicheradressen 0x20 bis 0x2F beliebig verwendbar, die anderen 128 Bits sind fest vorgegebene Bits im SFR Bereich, meist Konfigurationsbits.
- **Bitadresse < 0x80** (00 … 7F) : Bit befindet sich im **Adressbereich 0x20 ... 0x2F** (16 Bytes * 8 Bits = 128 Bits)

  z.B.: Bitadresse badr = 121 = 0x79 = 0111 1001 B = 0 1 1 1 1 0 0 1

  | | | |→ Bitposition im Byte

  | | → **x** bei Byteadresse 0x2**x**

  | → Bitadresse < 0x80

badr : 7 6 5 4 3 2 1 0 | 15 14 13 12 11 10 9 8 | 23 22 21 20 .... | 127 126 125 124 123 122 121 120

Byte 0x20 | Byte 0x21 | Byte 0x22 ..... | Byte 0x2F

- Berechnung : Bitadresse dividiert durch 8 = Byteposition Rest Bitposition im Byte
  - z.B.: 121 : 8 = 15 Rest 1

- **Bitadresse ≥ 0x80** : Bit befindet sich im **SFR Adressbereich**.

  z.B.: Bitadresse badr = 129 = 0x81 = 1 0 0 0 0 | 0 0 1

  | | |→ Bitposition im Byte

  | | → 1 0 0 0 0 || 0 0 0 = 0x80

  | → Bitadresse ≥ 0x80

  - Byte Adresse = (Bitadresse Bits 7 ... 3) || 000B
  - Bitadressierbare SFRs sind also : 0x80, 0x88, 0x90, 0x98, 0xA0, …. 0xF0, 0xF8

- Ein Operand, der ein Bit in einem Befehl (z.B.: SETB badr) identifiziert, kann folgendermaßen dargestellt werden :
  - Bitnummer : Ein Wert 0 … 255 (z.B. SETB 37)
  - Ein Symbol, dem die Bitnummer zugewiesen wurde : MyBit1 (MyBit1 EQU 37)
    - Den Bits in den SFRs (128 … 255) sind in der KEIL Entwicklungsumgebung bereits Symbole zugewiesen.
      - z.B.: SETB OV setzt das Overflow Flag
  - Eine Marke, die das Bit in einem Bitadressraum deklariert : MyBit2 (MyBit2 DBIT 1)
  - Byteadresse.Bitnummer :
    - 0x27.5 : Bit 5 in Byte 0x27
    - MyByte.3
    - PSW.7 (=Carry Flag)

Bit-Befehle

- **Set/Reset Bit Befehle** :
  - **SETB** badr ; Setzt das adressierte Bit *(badr) ← 1*
  - **CLR** badr ; Löscht das adressierte Bit *(badr) ← 0*
  - **SETB** C ; Setzt das Carry Flag im PSW *(CY) ← 1*
  - **CLR** C ; Löscht das Carry Flag im PSW *(CY) ← 0*
- **Komplement Bit Befehle** :
  - **CPL** badr ; Invertiert das adressierte Bit *(badr) ← $(\overline{badr})$*
  - **CPL** C ; Invertiert das Carry Flag im PSW *(CY) ← $(\overline{CY})$*
- **Kopierbefehle** :
  - **MOV** C,badr ;Kopiert das adressierte Bit in das Carry Flag *(CY) ← (badr)*
  - **MOV** badr,C ; Kopiert das Carry Flag in das adressierte Bit *(badr) ← (CY)*

Bit-Befehle (Forts.)

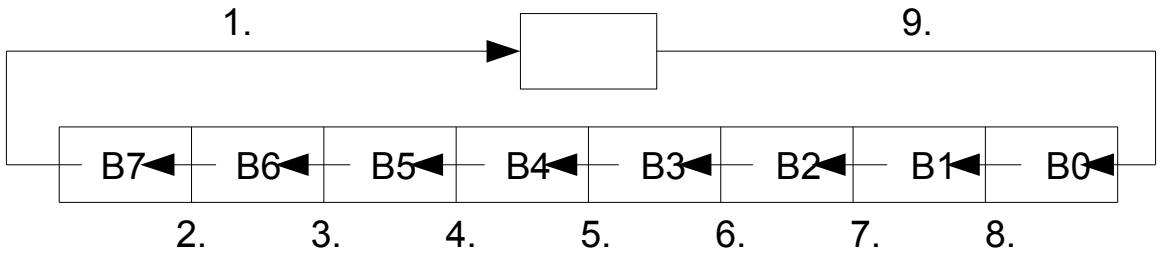
- **Logische Befehle** :
  - **ANL** C,badr ; AND-verknüpft das Carry Flag und das ;adressierte Bit und speichert das Ergebnis in ;das Carry Flag — *(CY) ← (CY) AND (badr)*
  - **ORL** C,badr ; OR-verknüpft das Carry Flag und das ;adressierte Bit und speichert das Ergebnis in ;das Carry Flag — *(CY) ← (CY) OR (badr)*
  - **ANL** C,/badr ; AND-verknüpft das Carry Flag und den ;invertierten Wert des adressierten Bits und ;speichert das Ergebnis in das Carry Flag — *(CY) ← (CY) AND $\overline{(badr)}$*
  - **ORL** C,/badr ; OR-verknüpft das Carry Flag und den ;invertierten Wert des adressierten Bits und ;speichert das Ergebnis in das Carry Flag — *(CY) ← (CY) OR $\overline{(badr)}$*

Bit-Befehle (Forts.)

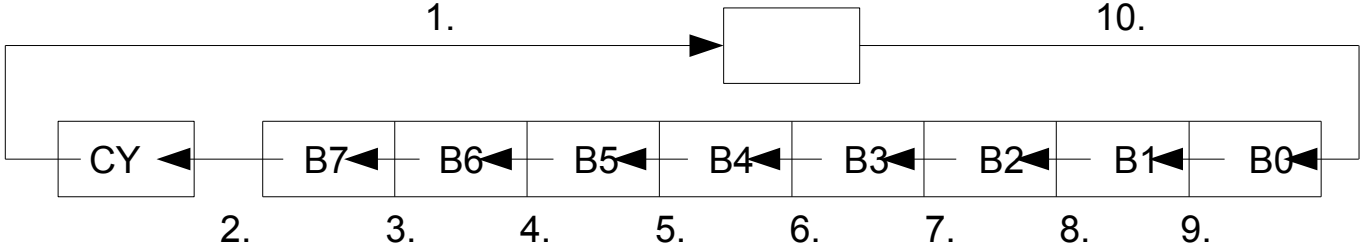
- **Sprungbefehle** :
  - **JB** badr,rel ; Springt, wenn das adressierte Bit = 1 ist. *IF (badr) == 1 THEN (PC) ← (PC) ± rel*
  - **JBC** badr,rel ; Springt, wenn das adressierte Bit = 1 ist, ;und löscht das addressierte Bit. *IF (badr) == 1 THEN { (PC) ← (PC) ± rel; (badr) ← 0 }*
  - **JNB** badr,rel ; Springt, wenn das adressierte Bit = 0 ist. *IF (badr) == 0 THEN (PC) ← (PC) ± rel*
  - **JC** rel ; Springt, wenn das Carry Flag im PSW = 1 ist. *IF (CY) == 1 THEN (PC) ← (PC) ± rel*
  - **JNC** rel ; Springt, wenn das Carry Flag im PSW = 0 ist. *IF (CY) == 0 THEN (PC) ← (PC) ± rel*
  - Als erste Aktion bei der Ausführung eines Befehls setzt der 8051 Prozessor den Befehlszähler (PC) auf die Adresse des OpCodes des nächst folgenden Befehls.
    - Bei den Sprungbefehlen muss das Sprungziel (rel) also relativ zur Adresse dieses OpCodes angegeben sein.
    - Bei Verwendung einer Marke als Sprungziel erledigt das der Assembler automatisch.

Bit-Befehle (Forts.)

- **Rotationsbefehle**
  - **RR A / RL A** ; Rotiert den Akkumulator um eine Bitposition nach rechts/links
    - z.B.: **RL A**



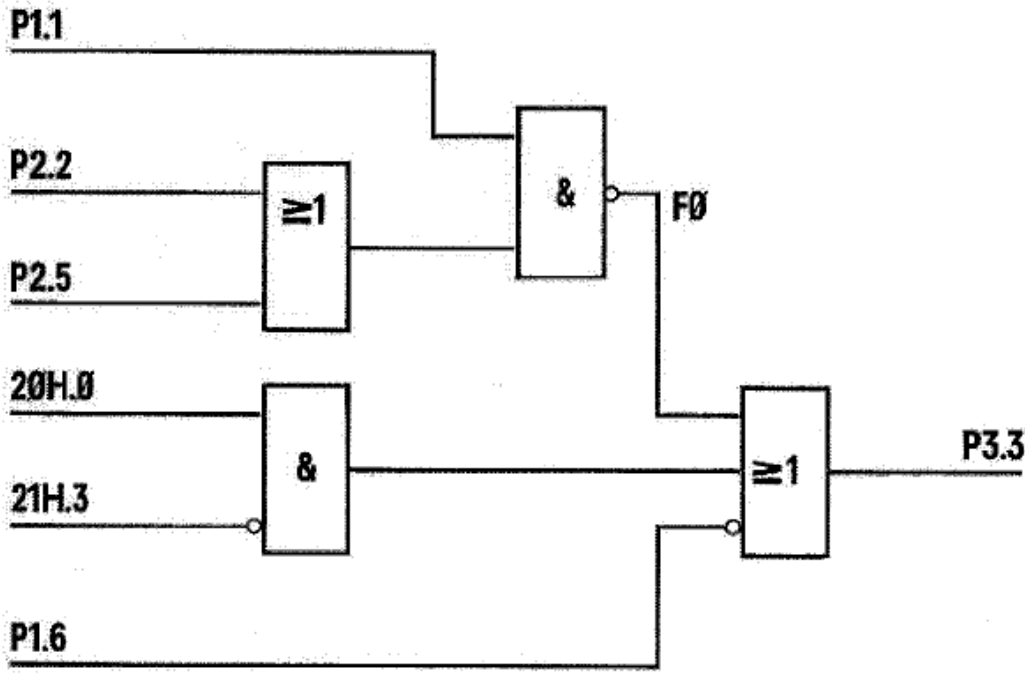
  - **RRC A / RLC A** ; Rotiert den Akkumulator samt Carry Flag um eine Bitposition nach rechts/links
    - z.B.: **RLC A**



Übungsaufgabe AR2 :

1. Vertauschen Sie das höherwertige und das niederwertige Halbbyte des Akkus mit Hilfe von Rotierbefehlen?
2. Kehren Sie die Reihenfolge der Bits in R1 um (B7→B0, B6→B1, ... , B1→B6, B0→B7) und schreiben Sie das Ergebnis in R2 !

Logikschaltungen abbilden




```
MOV   C,P2.2     ; P2.2 ins Carry einlesen
ORL   C, P2.5    ; ODER mit P2.5
ANL   C, P1.1    ; UND mit P1.1
CPL   C          ; Ergebnis negieren
MOV   FØ,C       ; Ergebnis zwischenspeichern
MOV   C, 20H.Ø   ; Bit 20H.Ø ins Carry einlesen
ANL   C,/21H.3   ; UND mit neg. Bit 21H.3
ORL   C, FØ      ; ODER mit Bit Bit FØ
ORL   C,/P1.6    ; ODER mit neg. Bit P1.6
MOV   P3.3,C     ; Endergebnis an P3.3 ausgeben
```

Beispiel: Boolescher Prozessor SAB 8051

Bild 52: logische Verknüpfungen durch Bitoperationen

80c515c Adressräume

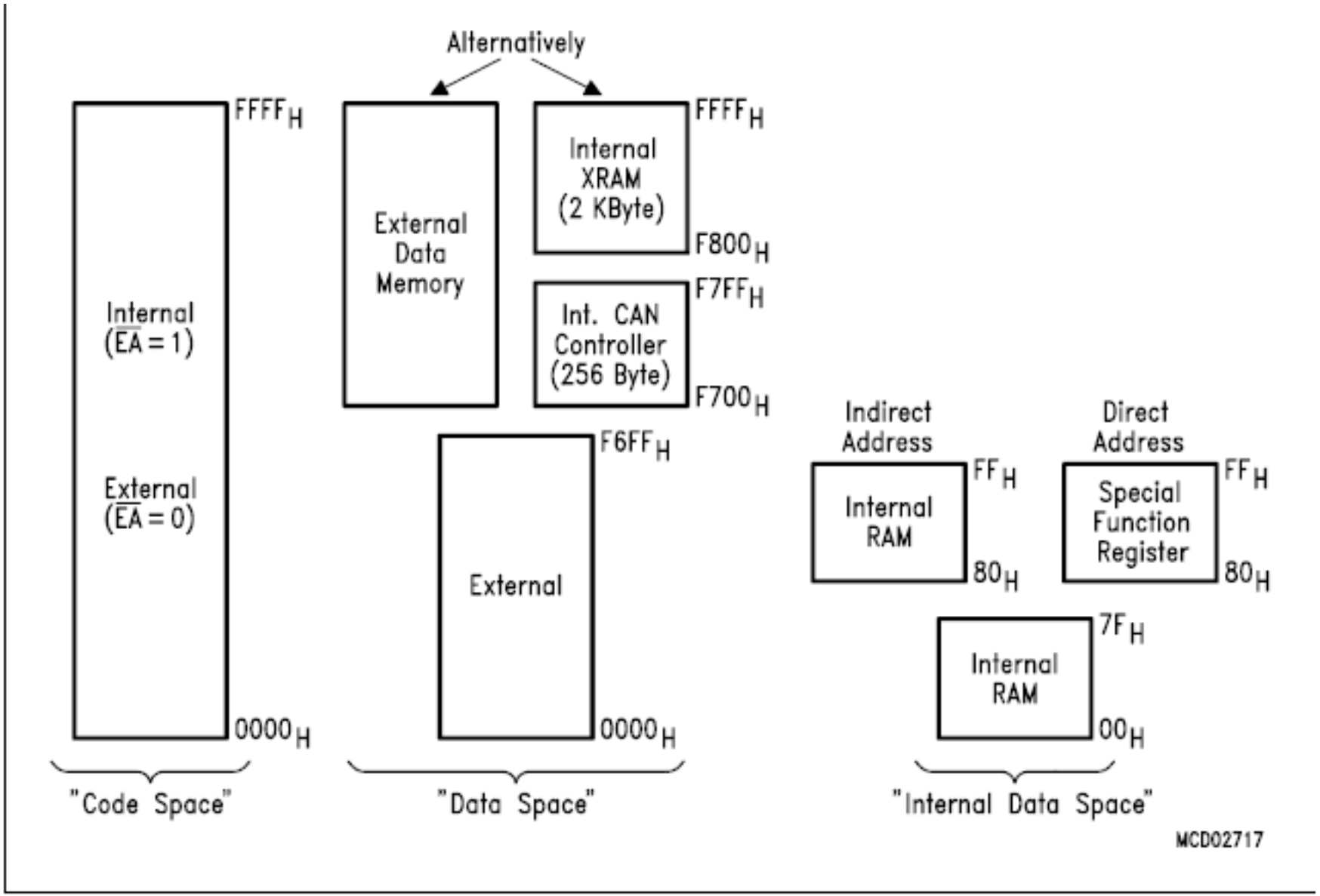
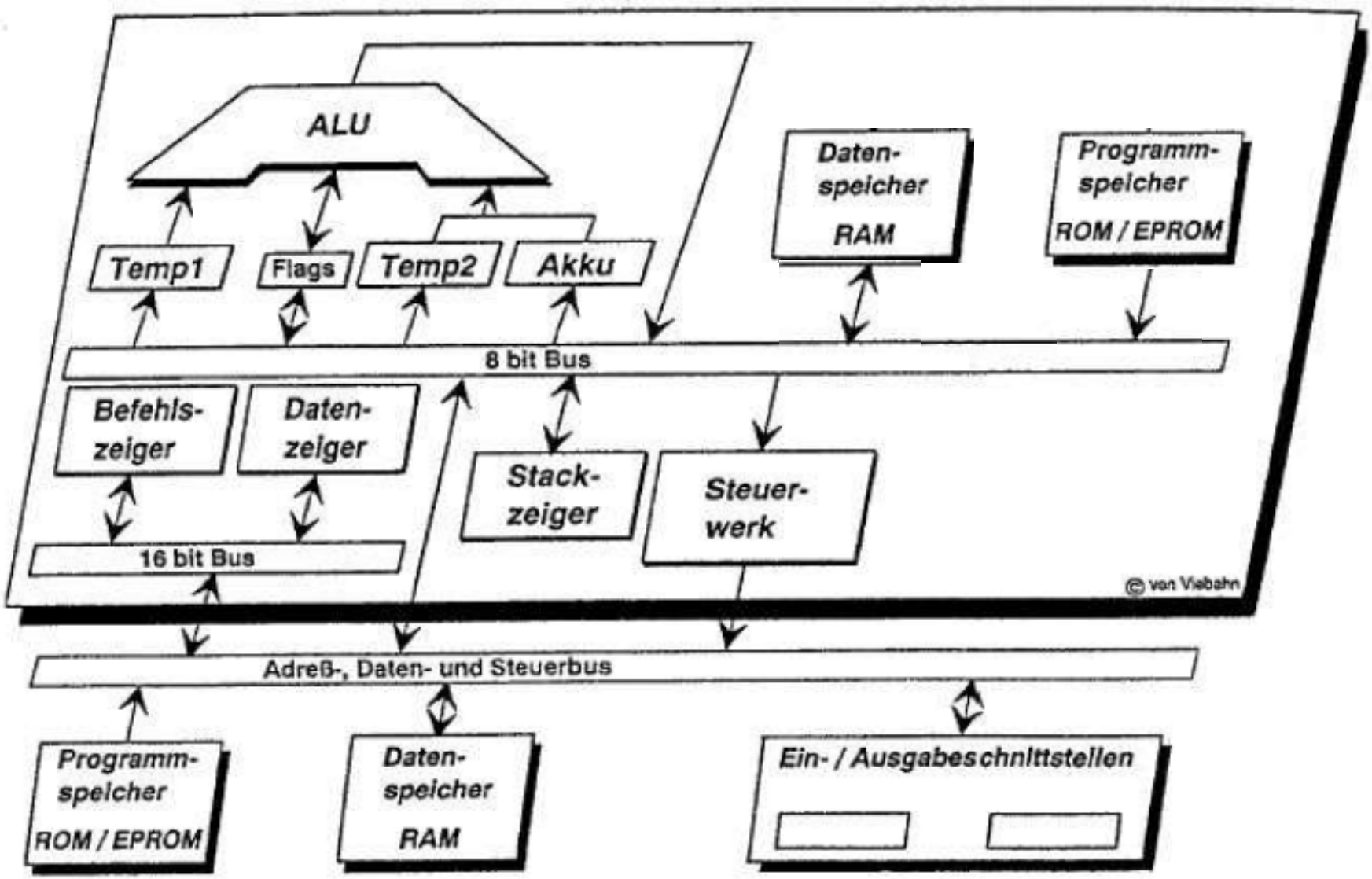


Figure 5
C515C Memory Map

80c515c Systemstruktur



Adressraum des Externen Datenspeichers (XRAM)

- Der Adressraum für das XRAM ist 64kB groß.
  - 16 Bit breite Adressen werden benötigt : Adressen 0x0000 … 0xFFFF
    - Also auch ein 16 Bit breites Adressregister in der Steuerwerkshardware : **der Datenzeiger DPTR**
      - Die Bytes an Adresse 0x83 und 0x82 im Bereich der Special Function Registers (SFR) stellen den DPTR dar.
- Für den Zugriff zum XRAM gibt es genau zwei Maschinenbefehle :
  - **MOVX A,@DPTR** und **MOVX @DPTR,A** transportieren ein Byte zwischen dem Datenspeicher und dem Akku.
  - Diese Art der Speicheranbindung wird **Load/Store Architektur** genannt
    - Bei der Load/Store Architektur werden nur Daten aus dem Speicher gelesen bzw. in den Speicher geschrieben.
    - Während der Verarbeitung werden die Daten intern im Prozessor gehalten
      - meistens in den General Purpose Registern oder im internen Datenspeicher wie beim 8051
- Der veraltete Mechanismus über XPAGE und R0/R1 ist nur noch aus Kompatibilitätsgründen vorhanden.
- Der 80c515c Prozessor besitzt ein **integriertes** XRAM
  - Im Adressbereich 0xF800 bis 0xFFFF, also 2 kB.
  - Wird über das Konfigurationsbit **XMAP0 = 0** aktiviert; XMAP0 ist Bit 0 im SFR SYSCON (0xB1),
  - Ist nach RESET deaktiviert (XMAP0 = 1).
- Es kann auch ein **externes** XRAM angeschlossen werden
  - Bestehend aus ein oder mehreren Speicherbausteinen.
  - Verfügbarer Adressbereich : 0x0000 bis 0xFFFF, wenn das integrierte XRAM deaktiviert ist (XMAP0 = 1)
    0x0000 bis 0xF6FF, wenn das integrierte XRAM nicht deaktiviert ist (XMAP0 = 0)

80c515c Adressräume

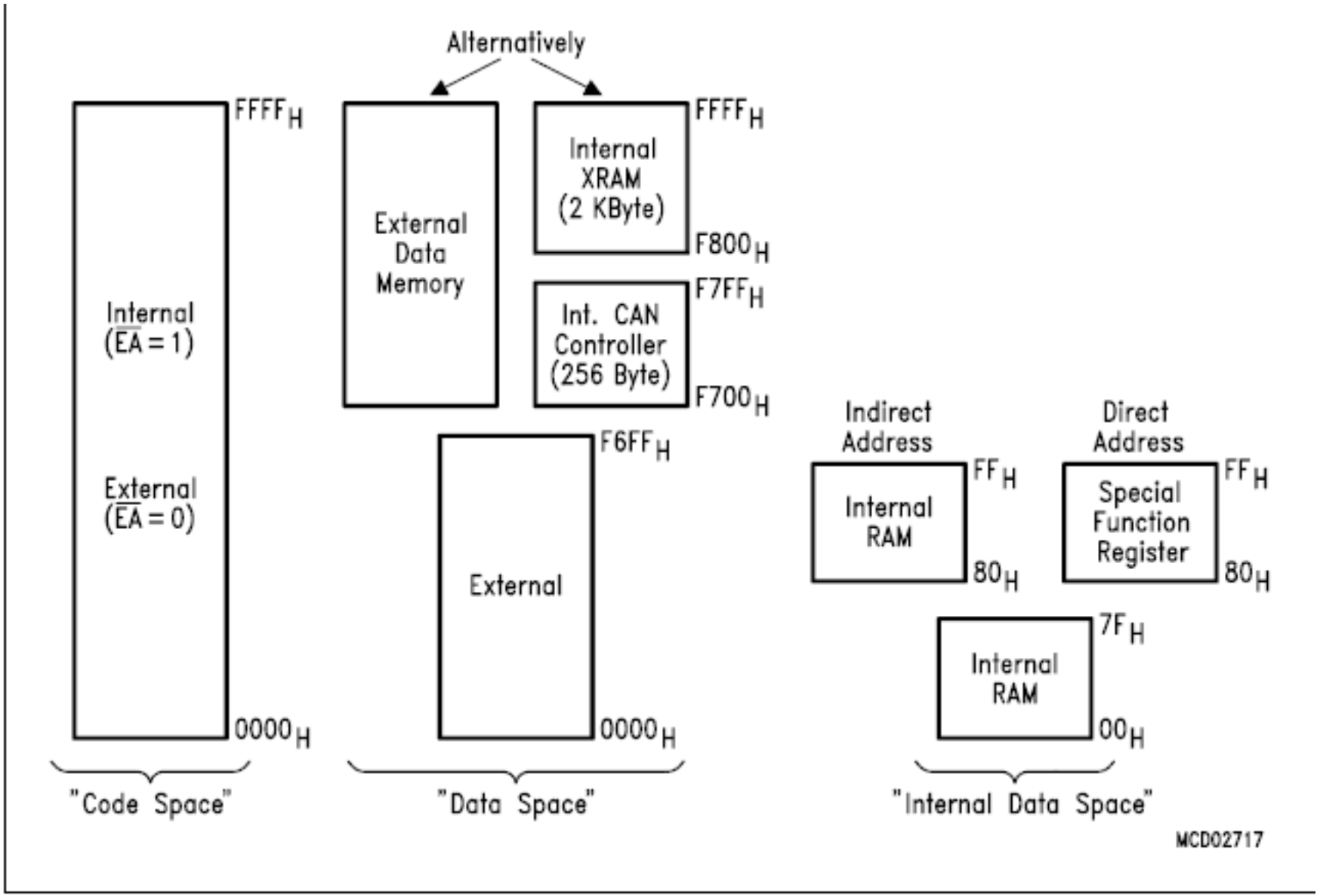


Figure 5
C515C Memory Map

Befehle zum Zugriff auf den DPTR

- **MOV DPTR,#c16**
  - OpCode : 0x90
  - Befehl is 3 Bytes groß.
  - Lädt den 16 Bit großen **konstanten** Wert c16 in das DPTR Register (SFR Adressen 0x83 und 0x82).
    - Schlüsselwort ist DPTR.
  - Beispiel 1 : **MOV DPTR,#0x1234**
  - Beispiel 2 :
    - im externen Datenspeicher :

```
            XSEG    AT   0x4321
ExtBy       DS      1                ;reserviert 1 Byte
```

    - im Programm

```
            MOV     DPTR,#ExtBy      ;lädt den Adresswert von ExtBy in den DPTR
```

    - Dem Symbol ExtBy ist ein konstanter **16 Bit** großer Adresswert in der Symboltabelle zugeordnet.

- **INC DPTR**
  - Inkrementiert den 16 Bit großen DPTR um 1.

Befehle zum Zugriff auf den DPTR (Forts.)

- **MOV DPH,Op2** und **MOV DPL,Op2**
  - Lädt einen 8 Bit Wert in das HIGH Byte (0x83) bzw. das LOW Byte (0x82) des DPTR Registers
  - Regulärer MOV Befehl
    - DPH und DPL sind reguläre SFR Adressen
    - Op2 kann daher sein : #c8, A, dadr, Rn, @Ri.
  - Mit diesen Befehlen wird typischerweise ein **variabler** Adresswert in den DPTR geladen.
  - Beispiel :
    - im XRAM :

```
ExtBy   DS      10          ;reserviert 10 Bytes
```

    - im Programm

```
        MOV     R6,#HIGH ExtBy      ;lädt HIGH Byte der Adresse von ExtBy (Bits 15 … 8)
        MOV     R7,#LOW ExtBy       ;lädt LOW Byte der Adresse von ExtBy (Bits 7 … 0)
        ….....                      ;Modifikation der Adresse in R6, R7
        MOV     DPH, R6             ;lädt den DPTR mit dem 'variablen' Adresswert
        MOV     DPL, R7
```

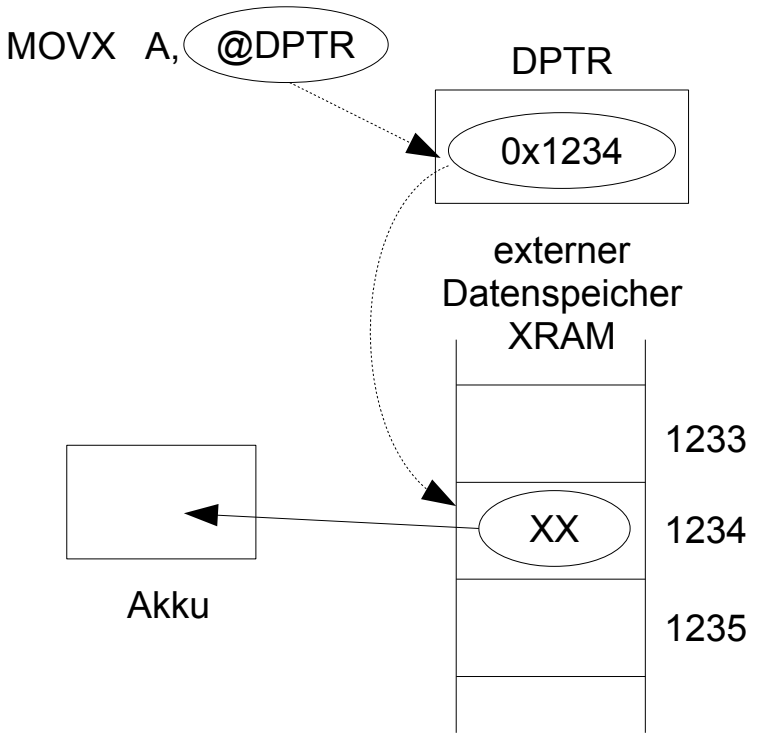
      - HIGH und LOW identifizieren die beiden Bytes eines **konstanten** 16 Bit Wertes
      - Deshalb muss vor HIGH und LOW immer das Zeichen # stehen.

Befehle zum Zugriff auf den externen Datenspeicher

- **MOVX Zieloperand,Quelloperand**
  - Überträgt ein Byte zwischen dem externen Datenspeicher (XRAM) und dem Akkumulator

OpCode	Ziel-Op.	Quell-Op.	Beispiel
E0	A	@DPTR	MOVX A,@DPTR
F0	@DPTR	A	MOVX @DPTR,A

  - Das Zeichen @ weist auf die indirekte Adressierung hin.



Übungsaufgaben AR4 :

1. Schreiben Sie in die externe Speicherstelle 0E8H den Wert 0AAH.
   (Wie Aufgabe 1.2.1, jedoch mit **externem** anstelle von internem Speicher.)
2. Laden Sie den 2-Byte-Wert bei Adresse 0x23 im internen Datenspeicher in das DPTR Register.
3. Im externen Datenspeicher bei der Marke ***Text:*** steht ein 10 Bytes langer Textstring. In R7 steht ein Indexwert in den Textstring (R7 = ein Wert zwischen 0 und 9). Laden Sie das DPTR Register so, dass es auf das indizierte Zeichen zeigt.

DPSEL

- Der 80c515c Prozessor besitzt nicht nur ein DPTR wie ältere 8051s, sondern **8** DPTR Register
  - Zu jedem Zeitpunkt ist genau eines der 8 DPTR Register aktiv und kann in Maschinenbefehlen verwendet werden.
  - Bits 2 bis Bit 0 des SFR Registers **DPSEL** (Adresse 0x92) identifiziert das gerade aktive DPTR Register.
- Die parallele Verwendung von mehreren DPTR Registern erzeugt effektiveren Code.

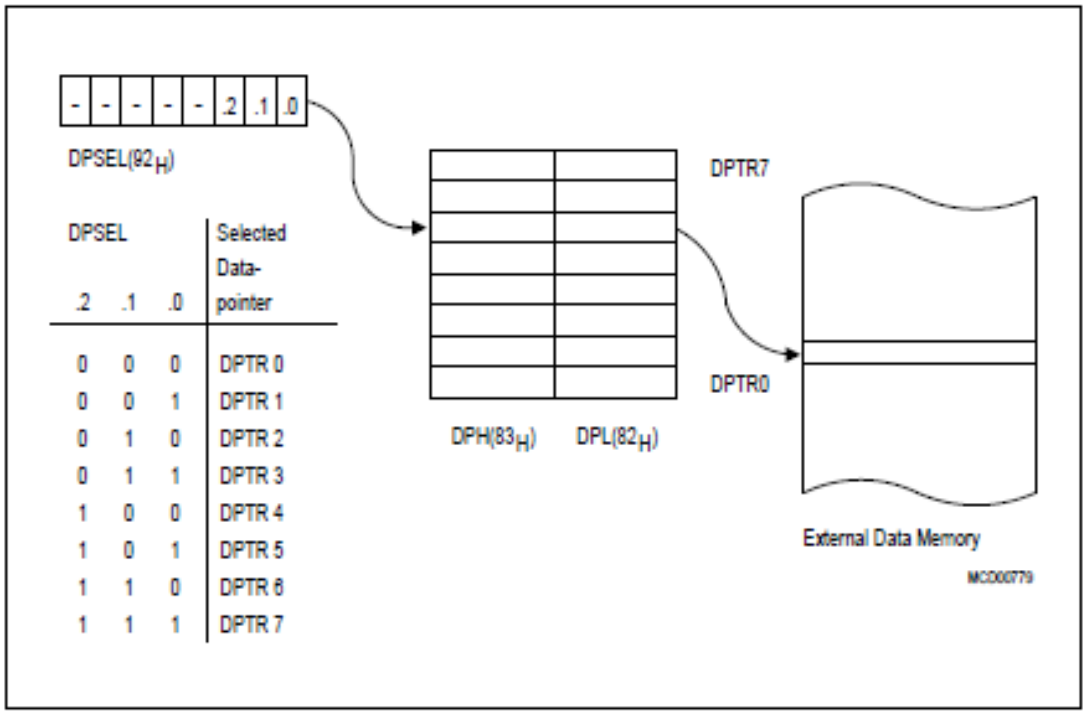
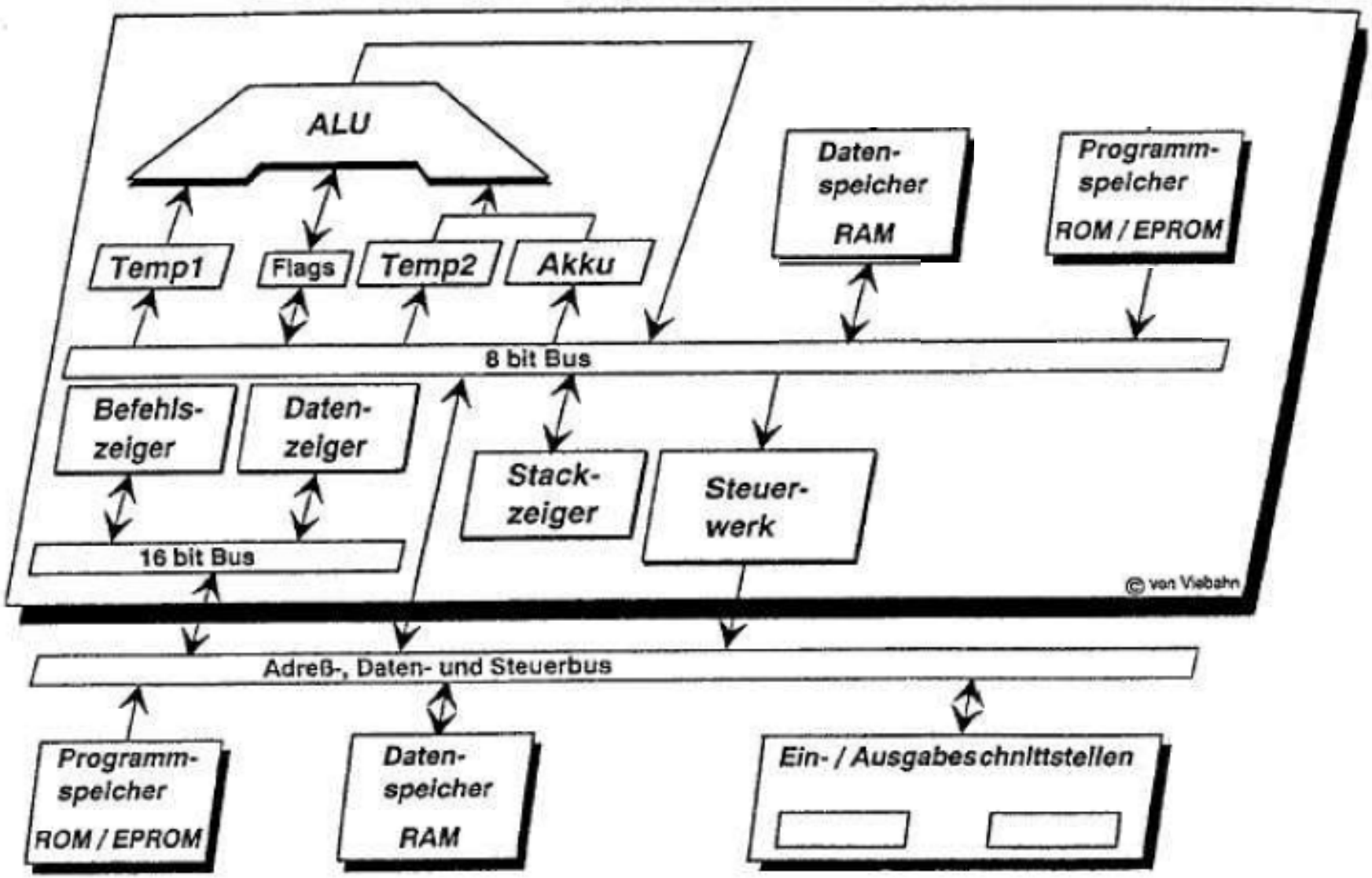


Figure 4-3
Accessing of External Data Memory via Multiple Datapointers

- Beispiel : Eine Kopierschleife

```
        MOV     DPSEL,#1        ;Initialisiere
        MOV     DPTR,#Ziel      ;DPTR1
        MOV     DPSEL,#0
        MOV     DPTR,#Quelle    ;DPTR0
Loop:
        MOVX    A,@DPTR         ;Lies Byte
        INC     DPTR
        MOV     DPSEL,#1
        MOVX    @DPTR,A         ;Schreibe Byte
        INC     DPTR
        MOV     DPSEL,#0
        ……                      ;Endbedingung
        JZ      Loop
```

80c515c Systemstruktur



Adressraum des Programmspeichers

- Der Adressraum für den Programmspeicher ist 64kB groß.
  - 16 Bit breite Adresse werden benötigt : Adressen 0x0000 … 0xFFFF
- Bei **Instruction Fetches** wird das **Befehlszähler-Register (PC)** für die Addressierung verwendet.
- **Konstante Daten** im Programmspeicher.
  - Konstante Daten können nicht im Datenspeicher stehen. Datenspeicher ist nach dem Einschalten leer.
  - Bei Zugriffen durch das Programm wird das **DPTR Register** für die Addressierung verwendet.
    - Zugriff mit dem Befehl **MOVC**.
- Der 80c515c Prozessor besitzt einen **integrierten** Programmspeicher.
  - im Adressbereich von 0 bis 64k -1 (0x0000 bis 0xFFFF).
  - kann über den Prozessorpin $\overline{EA}$ deaktiviert werden.
- Alternativ kann ein **externer** Programmspeicher angeschlossen werden
  - bestehend aus einem oder mehreren Speicherbausteinen
  - Adressbereich von 0 bis 64k -1 (0x0000 bis 0xFFFF)
  - wird aktiviert durch $\overline{EA}$ an Masse ($\overline{EA}$ = 0).
    - $\overline{EA}$ kann kein Konfigurationsbit in einem SFR sein, denn der Prozessor muss direkt nach RESET wissen, ob er schon den ersten Befehl aus dem externen oder internen Programmspeicher holen soll.

80c515c Adressräume

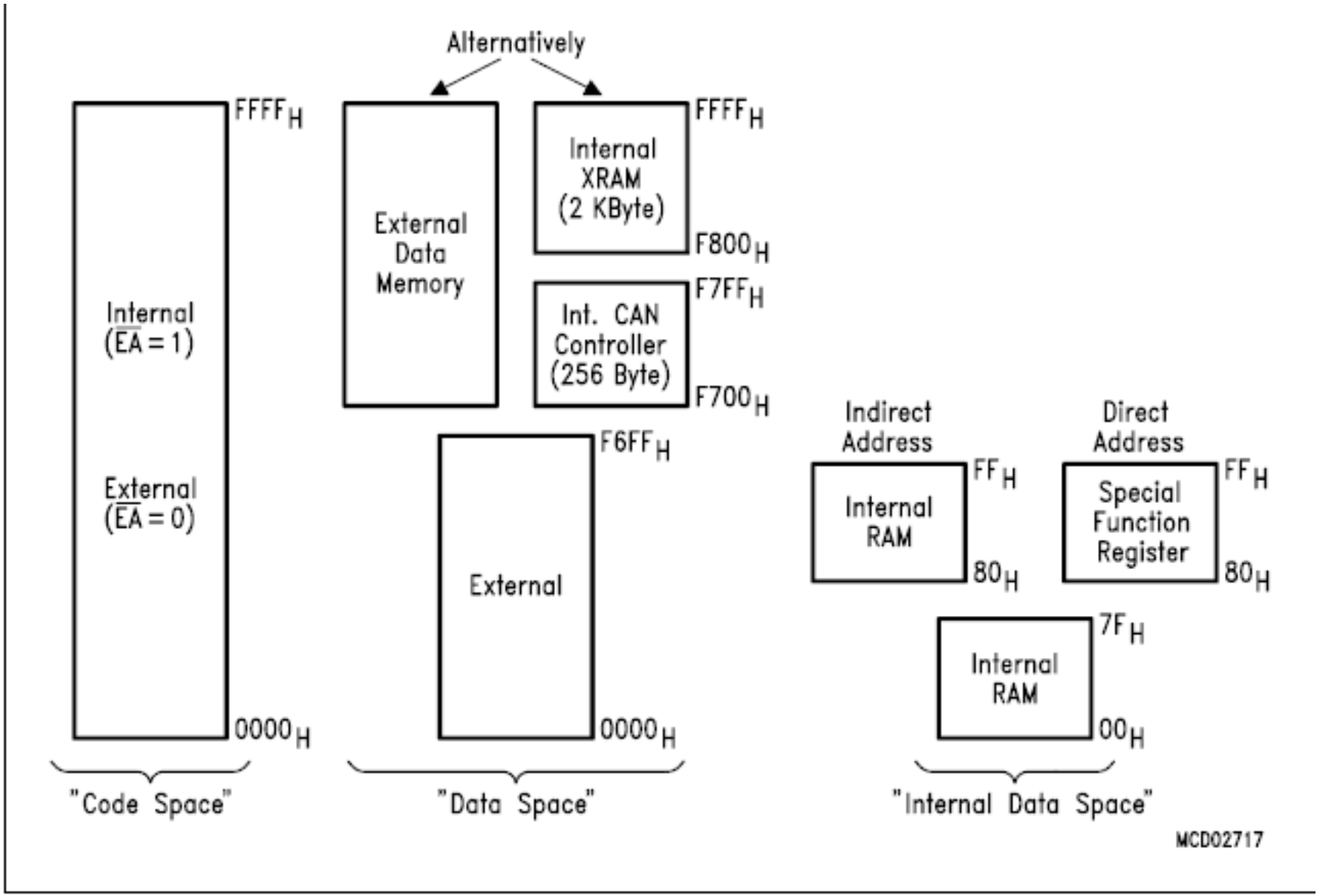
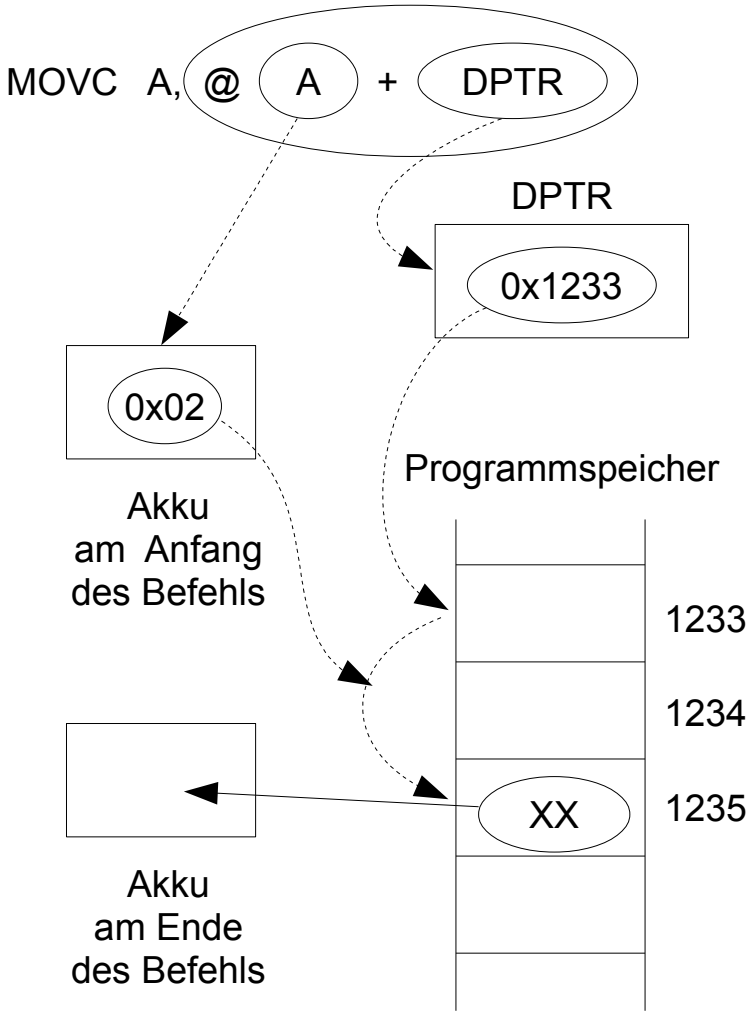


Figure 5
C515C Memory Map

Befehle zum Zugriff auf den Programmspeicher



- Die Befehle zum Aufsetzen des DPTR sind im vorigen Abschnitt beschrieben.
- **MOVC Zieloperand,Quelloperand**
  - Überträgt ein Byte vom Programmspeicher in den Akkumulator

OpCode	Ziel-Op.	Quell-Op.	Beispiel
93	A	@A+DPTR	MOVC A,@A+DPTR
83	A	@A+PC	MOVC A,@A+PC

- Bytes werden nur gelesen, nicht geschrieben
- **Indirekter** und **indizierter** Zugriff
  - Indexwert im Akkumulator

Übungsaufgabe AR5 :

Schreiben Sie eine Assembler-Routine, die den Text « Dies ist Text » aus dem Programmspeicher in den externen Datenspeicher ab Adresse 0x1000 kopiert.

```
Neuer Pseudobefehl :  DB                      ;Definiere Byte(s)
z.B.:        Marke1:  DB  0x12, 0x34          ;legt 2 Bytes im Programmspeicher an
             Marke2:  DB  'Press a Key', 0    ;Null-terminierter String, 12 Bytes lang
```

Indizierter Sprungbefehl : JMP @A + DPTR

- Die Summe der Werte im DPTR Register und im Akkumulator ergeben die Zieladresse des JMP Befehls.
- Bei allen anderen Sprungbefehlen ist das Sprungziel als konstanter Wert definiert.
  - Der konstante Wert der Zieladresse liegt also schon beim Assemblieren fest.
- Beim Befehl JMP @A+DPTR wird das Sprungziel zur Laufzeit des Programms festgelegt.
  - Mit der Befehlssequenz

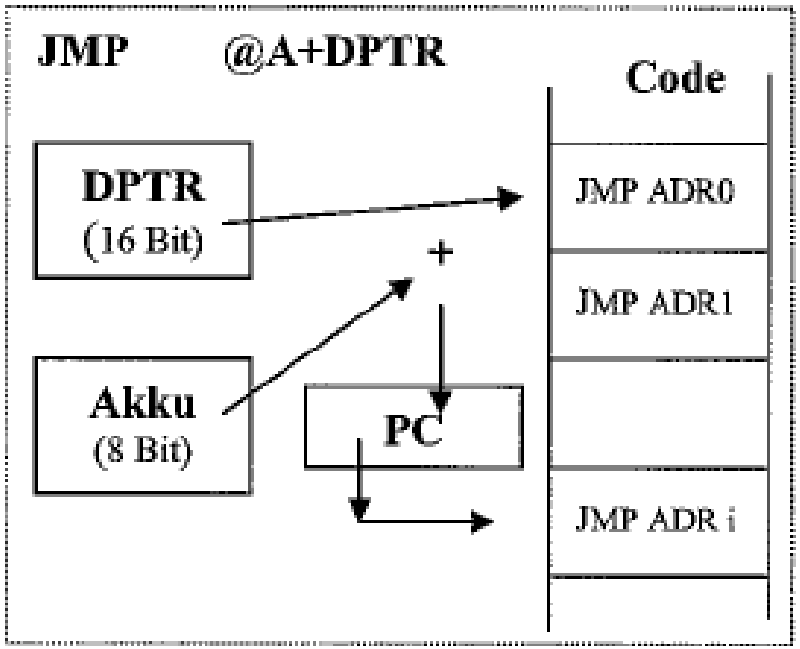
```
MOV     DPH,VarZiel
MOV     DPL,VarZiel+1
MOV     A,#0
JMP     @A+DPTR
```

    erfolgt ein Sprung zu der an der internen Datenspeicheradresse VarZiel abgelegten Programmspeicheradresse.

- Kann zur effizienten Realisierung des CASE-Kostrukts in C verwendet werden

```
switch (Schlüssel)
    {
        CASE 1 : Routine1();
        CASE 2 : Routine2();
        CASE 3 : Routine3();
    }
```

- Der Wert im DPTR Register (mit Akku = 0) zeigt auf den Code für den ersten Fall (Index = 0)
- Mit einem anderen Wert im Akkumulator kann der Code für die anderen Fälle angesprungen werden.



Operationen des externen Systembusses

- Zur Steuerung von Operationen auf dem externen Systembus benutzt der Prozessor die drei Steuerleitungen Read (**$\overline{\text{RD}}$**), Write (**$\overline{\text{WR}}$**) und Program Storage ENable (**$\overline{\text{PSEN}}$**).
- Der Prozessor führt eine Leseoperation durch, indem er
  - die Adresse aus dem Befehlszeiger-Register (PC) oder dem DPTR auf den Adressbus stellt,
  - die $\overline{\text{RD}}$ oder $\overline{\text{PSEN}}$ Steuerleitung aktiviert,
  - nach entsprechender Wartezeit das Byte vom Datenbus übernimmt.
- Der Prozessor führt eine Schreiboperation durch, indem er
  - die Adresse aus dem DPTR auf den Adressbus stellt,
  - das zu schreibende Byte auf den Datenbus stellt,
  - die $\overline{\text{WR}}$ Steuerleitung aktiviert.

Speichertyp	Operation	Verwendete Adresse	Steuerleitung
XRAM	MOVX A,@DPTR	DPTR	$\overline{\text{RD}}$
XRAM	MOVX @DPTR,A	DPTR	$\overline{\text{WR}}$
Programmspeicher	MOVC A,@A+DPTR	A+DPTR	$\overline{\text{PSEN}}$
Programmspeicher	Instruction Fetch	PC	$\overline{\text{PSEN}}$

- Mit der Steuerleitung $\overline{\text{RD}}$ wird aus dem XRAM, mit der Steuerleitung $\overline{\text{PSEN}}$ aus dem Programmspeicher gelesen.
  - Die $\overline{\text{PSEN}}$ Leitung vom Prozessor ist mit der $\overline{\text{RD}}$ Anschlussleitung des Programmspeichers verbunden.
  - So können 2 mal 64kB (Datenspeicher und Programmspeicher) mit einer 16 Bit Adresse unterschieden werden.

I/O Komponenten am externen Systembus des 8051

- I/O Komponenten können in gleicher Weise wie Datenspeicherbausteine am externen Systembus angeschlossen werden.
- Der Zugriff auf die internen Register einer I/O Komponente erfolgt mit dem MOVX Befehl, in gleicher Weise wie der Zugriff auf die Bytes in einem Datenspeicherbaustein.
- Die internen Register einer I/O Komponente sind in den Datenspeicheradressraum eingeblendet, deshalb spricht man von **memory mapped I/O**.
  - Ein Dekoder in der I/O Komponente selektiert das interne Register abhängig von der angelegten Adresse.
- Die Adressbereiche der Datenspeicherbausteine und I/O-Komponenten dürfen sich nicht überlappen.

Memory-Mapped I/O vs. Isolated I/O

- **Memory-Mapped I/O** : E/A-Schnittstellen- und Datenspeicherbausteine sind im selben Adressraum ansprechbar.
  - Es muss separate Adressbereiche im Datenspeicheradressraum für I/O-Schnittstellen- und Datenspeicherbausteine geben.
  - Zugriff mit MOVX Befehlen.
  - Der 8051 Prozessor benutzt die Memory-Mapped I/O Methode.
- **Isolated I/O** : Separate Adressräume für I/O-Schnittstellenbausteine und Datenspeicherbausteine.
  - Da eine Adresse sowohl ein Element im I/O-Baustein als auch ein Element im Datenspeicher ansprechen kann, wird zur Unterscheidung der beiden Adressräume eine zusätzliche Steuerleitung benützt : M/$\overline{IO}$
    - M/$\overline{IO}$ = 0 selektiert den I/O-Adressraum
    - M/$\overline{IO}$ = 1 selektiert den Datenspeicheradressraum
  - Zugriff zu Elementen des I/O-Schnittstellenbausteins mit **IN** und **OUT** Befehlen (bei x86 Prozessoren).
  - Die Prozessoren der x86 Familie z.B. benutzen die Isolated I/O Methode.

Memory-Mapped I/O vs. Isolated I/O (Forts.)

Im folgenden Bild wird jeweils ein von Neumann System mit gemeinsamem Programm- und Datenspeicher gezeigt.

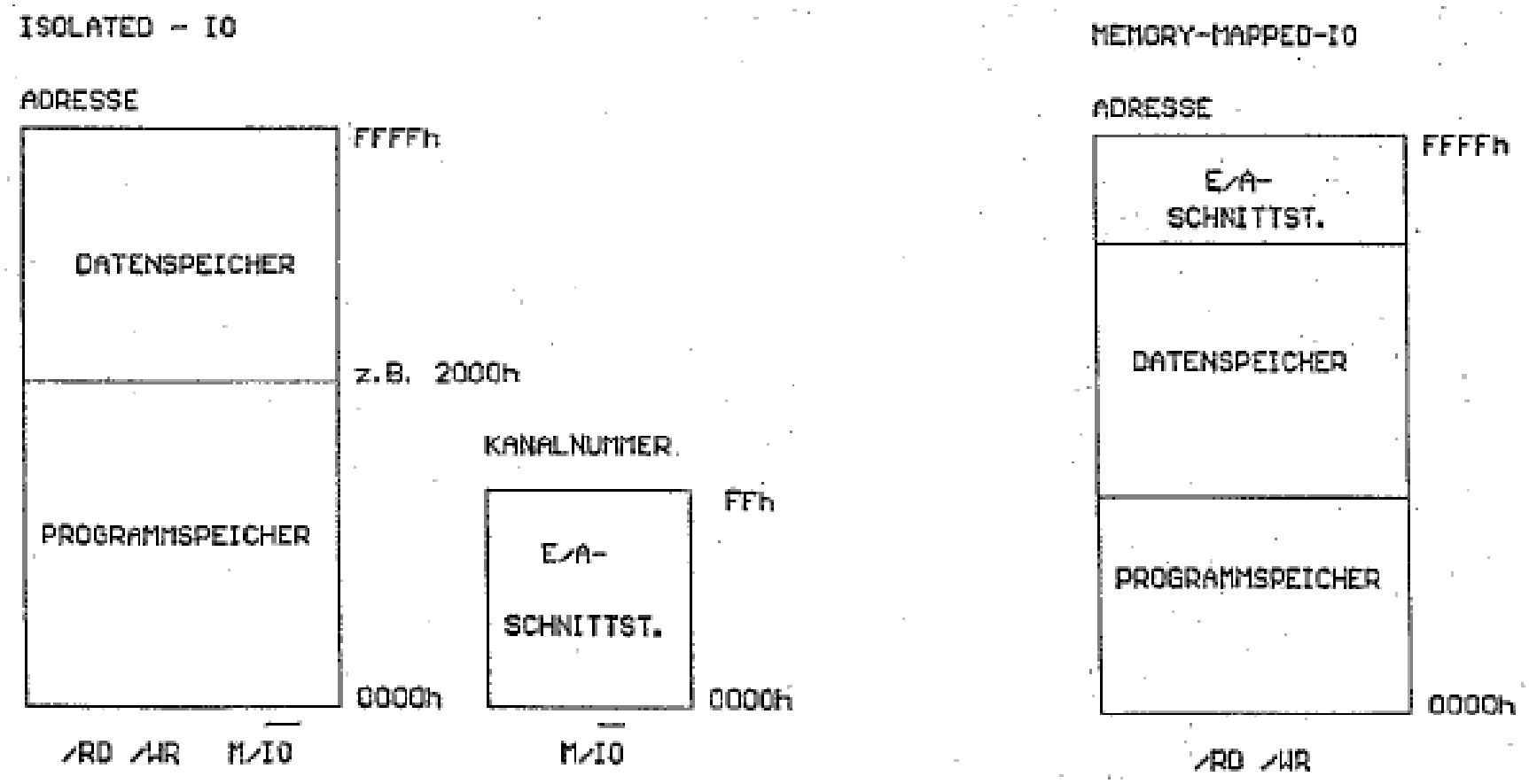


Bild 21: Adressraum für ISOLATED- und MEMORY-MAPPED-IO

Lösungen zu Übungsaufgaben

Übungsaufgabe AR1 :

```
MOV R0,#0x99        ;Zieladresse für indirekte Addressierung
MOV @R0,PSW         ;PSW (0xD0) direkt adressiert
                    ;0x99 indirekt adressiert
```

Übungsaufgabe AR2 : 1.

1. RR A ; B7 → B6 B0 → B7
   RR A ; → B5 → B6
   RR A ; → B4 → B5
   RR A ; → B3 → B4

Übungsaufgabe AR2 : 2.

```
        MOV     R0,#8       ;zählt 8 Schleifendurchläufe
Schleife: MOV   A,R1
          RRC   A           ;1. Durchlauf : B0→CY,  B1→B0
          MOV   R1,A
          MOV   A,R2
          RRL   A           ;1. Durchlauf : CY→B0,  B0→B1
          MOV   R2,A
          DJNZ  R0,Schleife
          MOV   A,R1
          RRC   A           ;Step 9
          MOV   R1,A
```

R1: Anfang: x 7 6 5 4 3 2 1 0 Step 1 : x 7 6 5 4 3 2 1 0

R2: 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 0

R1: Step 2 : 7 x 7 6 5 4 3 2 1 Step 3 : 6 7 x 7 6 5 4 3 2

R2: 6 5 4 3 2 1 0 0 1 5 4 3 2 1 0 0 1 2

…….

R1: Step 8 : 1 2 3 4 5 6 7 x 7 Step 9 : 0 1 2 3 4 5 6 7 x

R2: 0 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7

Übungsaufgabe AR3 :

Routine verwendet Big Endian Format

```
        PUSH    Op1
        PUSH    Op1+1           ;Op1 auf den Stack
        PUSH    Op1+2           ;MSB zuerst, LSB zuletzt
        PUSH    Op1+3

        MOV     R0,#Op2+3       ;R0 auf das LSB von Op2
        MOV     R1,#Erg+3       ;R1 auf das LSB von Erg
        CLR     CY
Loop:
        POP     ACC             ;Hole Op1 Byte, (LSB zuerst, MSB zuletzt)
        ADDC    A,@R0
        MOV     @R1,A
        DEC     R0
        DEC     R1
        CJNE    R1,#Erg-1,Loop  ;Durchlaufe die Schleife 4 mal
```

Übungsaufgaben AR4 :

1.
```
MOV DPTR,#0x00E8
MOV A,#0xAA          ;MOVX verlangt, dass das zu übertragende Byte im Akku steht
MOV @DPTR, A
```

2.
```
MOV DPH, 0x23        ;Big Endian für Adresse bei Adresse 0x23
MOV DPL,  0x24
```

3.
```
MOV     A, #LOW Text
ADD     A,R7
MOV     DPL, A

MOV     A, #HIGH Text
ADDC    A,#0         ;Addiere einen möglichen Carry aus der Addition des LOW Bytes
MOV     DPH, A
```

Übungsaufgabe AR5:

Im XRAM :

```
Ziel:   DS      20
```

Im Programm :

```
        MOV     R6,#HIGH  Ziel      ;Anfangswert
        MOV     R7,#LOW   Ziel

        MOV     R3,#0               ;Offset

Loop:   MOV     DPTR,#Cstr          ;Lade Quelladresse
        MOV     A,R3                ;Lade Offset
        MOVC    A,@A+DPTR           ;Lies Byte aus dem Programmspeicher
        JZ      Ende                ;Schleifenende, wenn Byte = 0x00

        MOV     DPH,R6              ;Lade Zieladresse
        MOV     DPL,R7

        MOVX    @DPTR,A             ;Schreibe Byte ins XRAM

        INC     DPTR                ;Inkrementiere Zieladresse
        MOV     R6,DPH              ;Rette Zieladresse
        MOV     R7,DPL
        INC     R3                  ;Inkrementiere den Offset

        SJMP    Loop

Cstr:   DB      'Dies ist Text', 0  ;Textstring
Ende:   .....
```