Mikrocontroller -Theorie und Anwendung unter Verwendung der Software

KEIL - µvision

(Einführung in die Assembler- und C- Programmierung des EST-Compuboards mit dem Mikrocontroller 80C535 der Firma Infineon)

Als Unterrichtsmanuskript verfasst und weiterentwickelt von

Gunthard Kraus, Oberstudienrat
Elektronikschule Tettnang

Version „2005-2“

Vorgenommene Änderungen:
Fehlerkorrekturen, neue „LCD_CTRL_new.c“-Datei, Projekt „PWM-Eisenbahn-Fahrtregler“, Kapitel-Nummerierungen

01.03.2005

Inhaltsverzeichnis

1. Allgemeine Grundlagen: Zahlensysteme und Spannungspegel

In der gesamten Digital- und Computertechnik arbeitet man mit nur zwei verschiedenen Spannungswerten, nämlich

Low- und **High-Pegel**

Low-Pegel bedeutet eine Spannung zwischen **Null Volt und +0,8V**, High-Pegel (bisher) eine Spannung zwischen **+3,5V und +5V**. (Im Zuge der Energiesparmaßnahmen wird jetzt bei modernen Computern die höchste positive Spannung auf **+3,3V** begrenzt).

Diesen beiden Spannungspegeln werden nun die beiden Ziffern **"Null" und "Eins**" zugeordnet, um damit im **Zweiersystem ("Dualsystem")** arbeiten und rechnen zu können.

Bei der heute üblichen "positiven Logik" gilt:	**Low-Pegel entspricht der Null** **High-Pegel entspricht der Eins des Dualsystems.**

Nun wollen wir klären, wie unsere gewohnten Dezimalzahlen mit "Low" und "High" dargestellt werden. Dazu braucht man sich nur am Aufbau unseres Zehnersystems orientieren:

a) Es existieren **10 Ziffern** (Achtung : von **Null bis Neun**...)

b) Man zählt diese zehn Ziffern **solange hoch**, bis alle **"aufgebraucht** sind" (also: von 0.....9)

c) Dann fängt man eine **neue Stelle** an und kann erneut die vorhandenen zehn Ziffern hochzählen:

10 11 12 13 14 15 16 17 18 19

d) Geht der Vorrat wieder aus, dann wird einfach die "**Zehnerstelle" um 1 erhöht** und der Zählvorgang von 0 bis 9 kann erneut beginnen.

20 21 22 .....28 29

e) So kann es bis 99 weitergehen. Dann wird natürlich eine neue, dritte Stelle (mit dem Namen "Hunderter") erforderlich usw.

Genauso gehen wir jetzt beim Dualsystem vor und fangen immer, wenn der Zeichenvorrat ausgeht, eine neue Stelle an. Diese Stellen heißen natürlich nicht "Zehner" oder "Hunderter", sondern sie bilden **"Zweier-Potenzen" (1 - 2 - 4 - 8 - 16 - 32 - 64 - 128 - usw.).**

Vergleich:	Zahl im Dezimalsystem	Zahl im Dualsystem

Zahl im Dezimalsystem	Zahl im Dualsystem

Umwandlung von Dualzahlen in Dezimalzahlen

Dazu muss man einfach nur die durch **eine "1" belegten Stellen in der Dualzahl** (entsprechend ihren Bedeutungen als **Zweierpotenzen**) zusammenzählen.

Beispiel: 110011011

Umwandlung von Dezimalzahlen in Dualzahlen

Hier gibt es mehrere Methoden, nämlich a) die **Zerlegung in Zweierpotenzen** und
b) die **Divisionsmethode.**
c) die Benützung eines **Taschenrechners** mit dieser Funktion..

Zu a): Die Zerlegung bietet sich besonders bei nicht zu großen Dezimalzahlen an.

Beispiel: die Dezimalzahl 147 lässt sich zerlegen in

Zu b) Die **vorgegebene Dezimalzahl wird solange durch 2 geteilt**, bis man beim Punkt
"Null Rest Eins"
ankommt. Die "Divisionsreste" ergeben dann -- von unten nach oben gelesen -- die Dualzahl!

Für obiges Beispiel:

Hexadezimalzahlen

Bei den **reinen Dualzahlen** erhält man bereits bei **kleinen Zahlenwerten recht lange Ausdrücke**. Deshalb geht man meist zum **Sechzehnersystem ("Hexadezimalsystem**") über und arbeitet mit den

. **Dezimalziffern von Null bis 15 in einem neuen System.**

Dadurch werden jeweils vier Dualstellen zusammengefasst und durch eine einzige "HEX-Ziffer" ausgedrückt.

So gewinnt man beträchtlich an Übersichtlichkeit -- auch bei großen Zahlenwerten!

Allerdings muss man nun für die Ziffern von 10 .....15 neue Bezeichnungen erfinden....:

Dezimalzahl:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Hexadezimalzahl																			

Zusammenhang zwischen Dualzahlen und Hexadezimalzahlen

Immer daran denken**: eine** Hex-Ziffer besteht aus **vier Dualstellen** und erfordert deshalb **vier Leitungen zur Übertragung**.

Umwandlungen zwischen diesen beiden System werden also ganz einfach, wenn man die Dualstellen in "Vierergruppen" zusammenfasst!

Beispiel: Drücken Sie die Dualzahl 11001110101010B als Hex-Zahl aus.

Lösung:

Beispiel: Wie viel Leitungen sind zur Übertragung der Hex-Zahl FD2AH nötig und welche logischen Pegel misst man bei Übertragung dieser Zahl?

Lösung:

Umwandlung von Hexadezimalzahlen in Dezimalzahlen

Auch hier müssen die einzelnen Ziffern der Hex-Zahl entsprechend ihrer Bedeutung als "Sechzehner-Potenz" zusammenaddiert werden.

Beispiel: 2A3FH

Umwandlung von Dezimalzahlen in Hexadezimalzahlen

Entweder benützt man wieder die **Divisionsmethode** (wobei man diesmal **durch 16 teilen** muß) oder man wählt den **Umweg über das Dualsystem** (Siehe oben, ist auch übersichtlicher, da man gleich gratis die Informationen über die Pegel der einzelnen Leitungen mitgeliefert bekommt....).

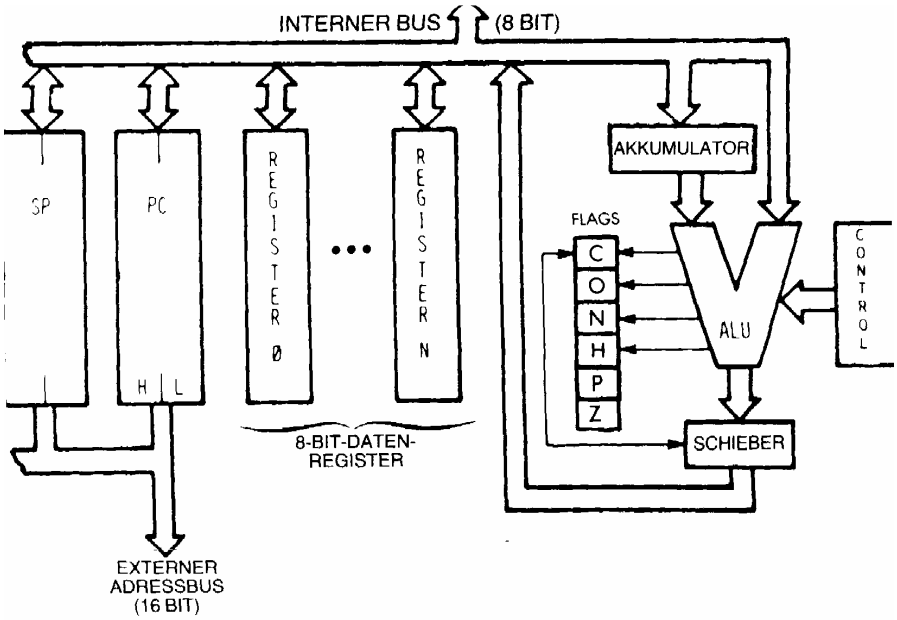
Beispiel: Wandeln Sie die Dezimalzahl 517 in eine Hexadezimalzahl um.

2. Grundlagen der Mikrocontrollertechnik

2.1. Aufbau eines Mikroprozessor-Systems

Ein **Mikroprozessor-System** stellt den "**Minimalaufbau einer Datenverarbeitungsanlage**" (für kleine Leistung) mit allen dafür erforderlichen Bausteinen dar. Der **Mikroprozessor** bildet darin nur die sogenannte **CPU (= central processing unit = Zentrale Recheneinheit**)

Die **Bearbeitung der Daten** (z. B. Addieren, Subtrahieren, Multiplizieren, Dividieren, Logisch Verknüpfen durch UND und ODER, Inkrementieren, Dekrementieren, usw. usw.) **erfolgt immer in der CPU**. Alle übrigen Bausteine sind hierbei nur ein notwendiges Übel....
So sieht es in einer CPU aus:



Aktuelle Daten, die noch in Bearbeitung sind , befinden sich entweder in einem der **Arbeitsregister (**z. B. Akkumulator, B-Register usw.) oder in einer **"Schnellablage**" (meist eine **kleine Registerbank** mit Platz für 8 Bytes, also 8 Zahlen). Größere anfallende Datenmengen speichert man immer in einem **RAM** (= random access memory = Schreib- Lese-Speicher.)

Die **Daten** werden innerhalb des Systems als **Dual- bzw. Hexadezimalzahlen** auf dem sogenannten **"Datenbus"** transportiert.

Üblich sind:
- **-- bei einfachen Bediencontrollern eine Datenbusbreite von 4 Leitungen (= 4 Bit), also Zahlen von 0....7**
- **-- bei Standard-Systemen (8051-Familie, PIC, ATMEL...) 8 Bit Datenbusbreite, also Zahlen von 0....255**
- **-- für erhöhte Anforderungen: 16 Bit Datenbusbreite, also Zahlen von 0.....65535**

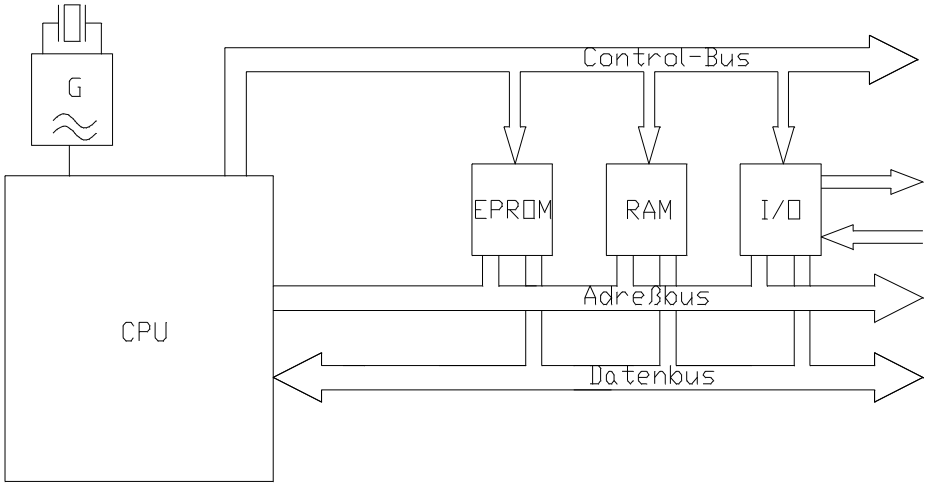
Für die Speicherung von **Programmen, Tabellen** usw. dient ein **"Festwertspeicher" (meist ein ROM = read only memory oder ein "EPROM" = electronic programmable read only memory),**

Achtung: Jeder in einem der beiden Speicher abgelegten Wert muss unter einer **bestimmten Adresse** (meist eine 16 - Bit - Zahl) zu finden sein . Deshalb sind **alle Speicherbausteine** nicht nur an den Datenbus, sondern auch an einen **Adressbus** angeschlossen und werden in ihren Funktionen (Schreiben oder Auslesen) über einen **Kontrollbus** gesteuert.

Für die "**Verbindung zur Außenwelt**" (z. B. zur Ansteuerung irgendwelcher Schaltungen, zur Ausgabe von Ergebnissen an einen PC oder ein Display, zum Einlesen von Messwerten usw.) sind **I / O -Bausteine** erforderlich.
Ihre Aus- und Eingänge tragen den Namen

Ports

und weisen natürlich dieselbe Breite wie der Datenbus auf!
Damit kommt man zu folgendem **Übersichtsschaltplan für ein Mikroprozessorsystem**:



Unterschied zwischen Mikroprozessor und Mikrocontroller:

Beim Mikrocontroller sind alle besprochenen Baugruppen auf einem einzigen Chip integriert, während ein Mikroprozessor nur die CPU darstellt.

Der Mikrocontroller ist aus diesem Grund für sich allein -- also ohne Zusatzbausteine -- lauf- und arbeitsfähig. Wegen der Kommunikation mit seiner Umwelt und zur Bewältigung größerer Aufgaben weist er immer folgende Eigenschaften auf:

--- Er besitzt einen **kleinen Datenspeicher** (also ein **RAM**, meist mit 128 oder 256 Byte) **auf dem Chip**.
Der Grundbaustein ist auch gegen Aufpreis mit einem kleinen **PROM** oder **EPROM** von 4.....8 Kilobyte Speicher-Kapazität (für die Anwendungsprogramme) -- ebenfalls **auf dem Chip** -- lieferbar

--- Er kann durch **externe Speicher** (ROM und RAM) in seiner Speicherkapazität beträchtlich **erweitert** werden.
(Üblich sind max. 64 Kilobyte externer RAM- und 64 Kilobyte externer EPROM-Speicherplatz, entsprechend einer Adressbus-Breite von 16 Bit).

--- Mehrere **bidirektionale Ports** ermöglichen die **Ein- und Ausgabe** von Daten, Messwerten, Informationen und Steuersignalen.

--- Eine oder mehrere **serielle Schnittstellen** dienen zur Programm- oder Datenübertragung vom Chip zu einem PC bzw. zur Kommunikation mehrerer Kontroller untereinander.

--- Integrierte **Timer und Counter** sind für Zählaufgaben oder Zeitfunktionen vorgesehen.

--- **Interrupt-Eingänge** mit mehreren Prioritätsebenen ermöglichen Programmunterbrechungen bzw. die spontane Reaktion auf besondere Ereignisse.

--- Die Weiterentwicklungen enthalten **A/D-Wandler** mit bis zu 12 umschaltbaren Analogeingängen sowie **Interface-Bausteine für verschiedene Bussystem (I^{2}C, CAN**) usw.

--- **Arithmetik- und Logik-Aufgaben** sind durch einfache Befehle lösbar.

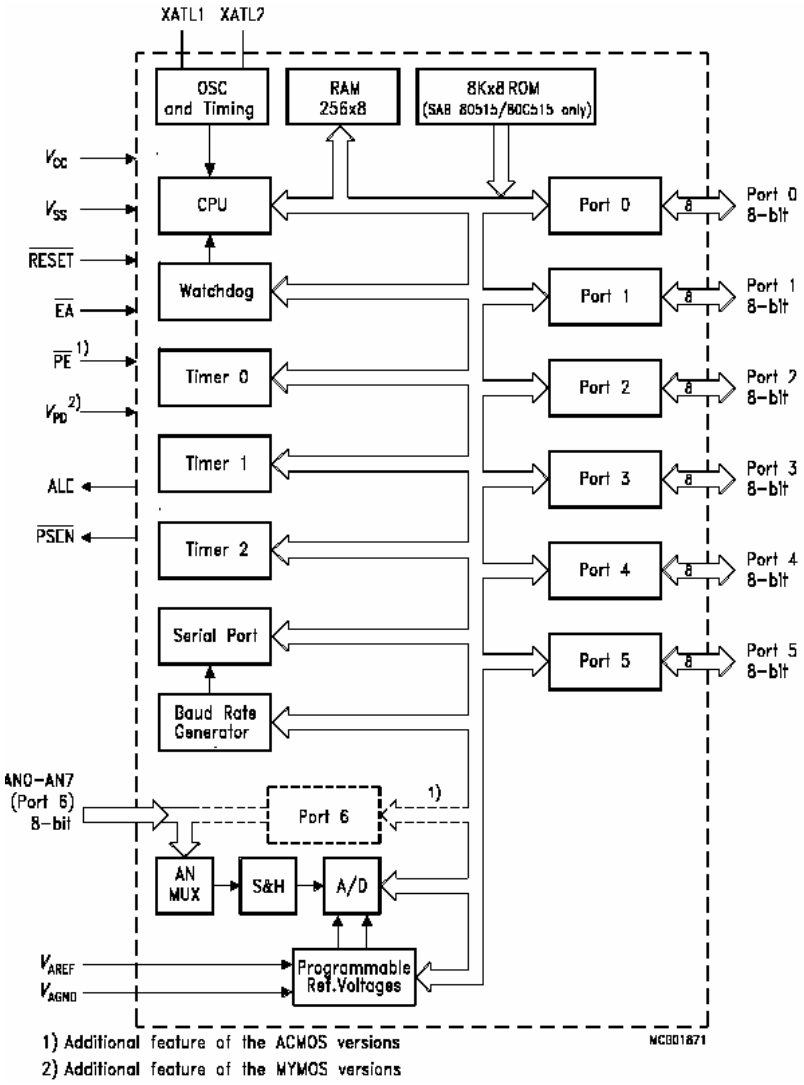
--- Neueste Entwicklung ist eine **zusätzliche Arithmetik-Einheit** (Addition, Subtraktion, Multiplikation, Division) **für 16-Bit-Zahlen** auf dem Chip! (Typ: 80C537).

--- **Capture- und Compare-Einheiten** ermöglichen die Vereinfachung von Zeit- oder Frequenzmessungen sowie die **Digital-Analog-Wandlung mittels Pulsbreitenmodulation**.

2.2. Übersichtsschaltplan unseres Controllers (80C535)

Dieser Baustein stellt eine Weiterentwicklung der “INTEL-Weltstandard- Mikrocontroller-Familie 8051” durch die Firma Infineon dar. Auf den “8051-Kern” werden immer mehr Zusatzfunktionen und Baugruppen “aufgepfropft”. Dadurch lassen sich ältere Programme auch noch bei Neu- und Weiterentwicklungen verwenden.

Ähnliche Sachen gibt es von Philips (z. B. 80C552 mit I^2C-Bus-Steuerung auf dem Chip).



Eigenschaften des 80C535 - Controllerbausteins:

a) Auch mit **8k x 8Bit ROM** auf dem Chip erhältlich (80C515)

b) **256 x 8 Bit RAM** auf dem Chip

c) **Sechs digitale 8 Bit- I / O Ports** sowie ein Eingangsport für Analog- oder Digitalsignale

d) **Drei 16 Bit - Timer / Zähler**

e) Reload, capture und compare-Optionen

f) **Serielle Schnittstelle (voll duplex)**

g) **12 Interrupt - Vektoren, 4 Prioritätsebenen**

h) **8 Bit - A / D -Wandler mit 8 gemultiplexten Eingängen** und programmierbarer interner Referenz

l) **16 Bit - Watchdog - Timer**

j) Boolescher Prozessor für **Arithmetik- und Logik- Aufgaben**

k) **256 “Bit-adressierbare” Speicherplätze**

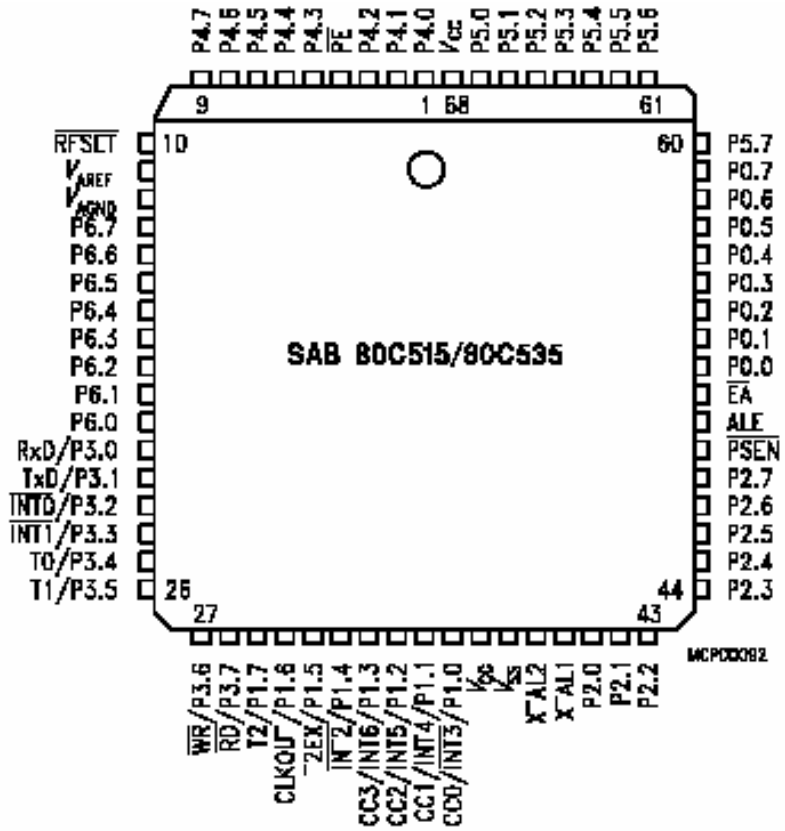
l) Befehls-Ausführungszeit für die meisten Befehle: **1 Mikrosekunde** bei einer Taktoszillatorfrequenz von 12 MHz

m) **4 Mikrosekunden Rechenzeit für Multiplikation oder Division** (von 8 Bit - Zahlen)

n) **Externe Speicher bis zu 64 Kilobyte Umfang adressierbar**

o) **Idle- und power-down-mode**

Anschluss-Belegung



3. Speicherorganisation

3.1. Grundlagen

Der Mikrocontroller soll bekanntlich **Daten** verarbeiten. Sie müssen eingelesen, abgespeichert, aus dem Speicher geholt, bearbeitet, wieder ausgegeben oder neu gespeichert werden. Deshalb benötigt man für die Datenspeicherung immer einen

Schreib - Lese - Speicher (also ein RAM)

Bei den **Programmen** gibt es aber **zwei Möglichkeiten**:

1) **Alle Befehle eines Programms** (Fachausdruck: Code-Bytes) **werden in ein EPROM "eingebrannt**" und der Controller beginnt sofort nach einem Reset oder nach dem Einschalten des Gerätes mit der Ausführung des ersten Befehls. Zum Ändern des Programms muss dann (leider) ein neues EPROM gebrannt und gegen das vorhandene ausgetauscht werden.
   Frisch anfallende Daten haben im EPROM nichts zu suchen und können auch während des Betriebes nicht in das EPROM geschrieben werden (denn dafür haben wir das RAM).
   Man bezeichnet diese Speicherorganisation (mit getrennten Programm- und Datenspeichern) als

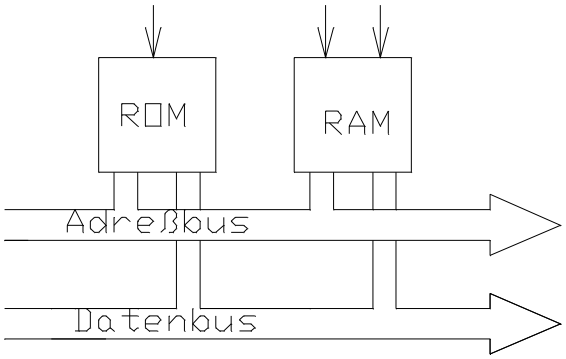
   Harvard-Architektur.

So werden die meisten Mikrocontroller in der Praxis betrieben.

2) **Falls überhaupt, dann ist nur ein kleines EPROM vorhanden**.In ihm befindet sich **nur** das sogenannte **Betriebssystem ("Monitorprogramm**", oder beim PC das "**BIOS" =** bidirectional input-output system).
   Damit ist der Controller mit einer "Grund-Intelligenz" ausgestattet und man **muss das zu bearbeitende Programm nach dem Einschalten des Gerätes von irgendeinem Datenträger in das RAM einlesen**. Natürlich muss der Programmierer nun sehr sorgfältig darauf achten, dass sich Codebytes und Datenbytes nie in die Haare geraten können, da sie ja **hintereinander im RAM-Speicher angeordnet** sind (so macht es jeder **PC**). Diese Methode heißt

   von Neumann - Architektur

Harvard - Architektur



Kennzeichen:

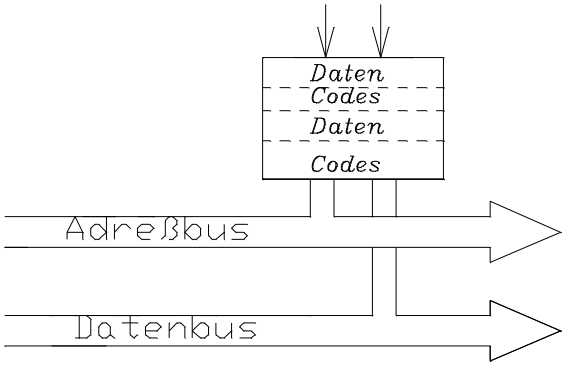
a) **Getrennte Speicher.**

b) Daten werden durch "**#RD**" (= read nicht) ausgelesen und durch "**#WR**" (= write nicht) im RAM gespeichert.

c) **Programmcodes** werden durch das Signal "**#PSEN**" (= program store enable nicht) ausgelesen.

d) **Gemeinsamer Adress- bzw. Datenbus**

Von Neumann - Architektur



Kennzeichen:

a) Nur **ein gemeinsamer** Speicher **(RAM).**

b) Daten und Programmcodes liegen **hintereinander im Speicher** und müssen anhand ihrer **Adressen** unterschieden werden.

c) Deshalb **keine Unterschiede bei den Schreib- und Lesebefehlen für Daten bzw. Programme**

Umfang des Daten- und Adressbereiches:
Bei **8-Bit**-Prozessoren weist der **Datenbus** natürlich **8 Leitungen und damit eine Breite von 8 Bit =1 Byte** auf. Deshalb stellt der Inhalt jeder Speicherzelle eine Hexadezimal-Zahl zwischen

00H = Null und **FF**H = 255

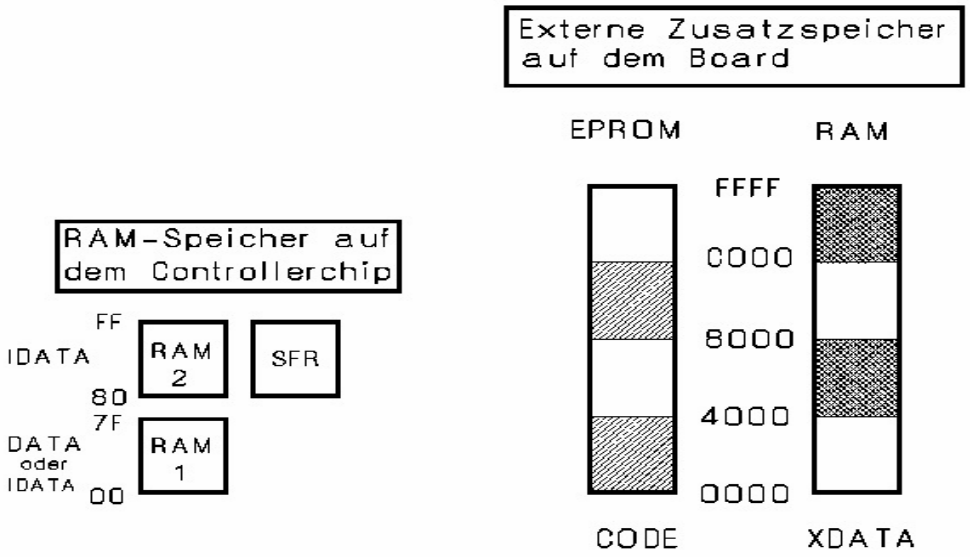
dar.

Der **Adressbus** (zur Auswahl der Speicherplätze außerhalb des Controllerchips) besitzt eine Breite von **16 Bit = 2 Byte.** Das ergibt einen ansprechbaren Adressraum von Adresse

0000H = Null bis **FFFFH = 65535**

(Also umfasst der maximal extern adressierbare Speicher bei der MCS51-Familie ca. **64 KiloByte** -- **sowohl beim Programmspeicher wie auch beim Datenspeicher).**

3.2. Speicherorganisation beim EST-Compuboard 80C535



Auf dem Controllerchip befinden sich die **„Spezial-Funktions-Register“** im Adressbereich von **80h bis FFh**. Sie sind den unterschiedlichen Baugruppen wie Timer, Schnittstelle, Interrupt usw. zugeordnet und werden direkt adressiert. Sie dürfen vom Anwender nicht zur Ablage irgendwelcher Daten benützt werden.

Name: **SFR**

Zusätzlich gibt es noch ein **RAM2** unter **denselben Adressen wie die SFR**, also
von 80h bis FFh (= ebenfalls 128 Bytes). Es ist nur ***indirekt adressierbar***, um Verwechslungen mit SFR zu vermeiden.

Name**: IDATA SEGMENT**.
Der für uns wichtigste Datenspeicher ist das **RAM 1 im Adressbereich von 00h bis 7Fh** (= 128 Bytes). Es wird üblicherweise direkt adressiert.

Name: **DATA-SEGMENT**.
(Er darf aber auch indirekt adressiert werden und zählt dann zum IDATA-SEGMENT).

Außerhalb des Chips wird auf dem Board ein **externer Programmspeicher (ROM oder EPROM)** mit maximal 64 Kilobyte Kapazität angeordnet. Der Anwender hat auf die gespeicherten Programmcodes normalerweise keinen Zugriff (Ausnahme: mit dem Befehl „MOVC A,@DPTR+A“ kann man ein Codebyte aus dem EPROM in den Akku holen und dort wie ein Datenbyte behandeln. Das wird z.B. zum Auslesen von gespeicherten Tabellen eingesetzt).

Name: **CODE SEGMENT**.

Ein zusätzlicher **externer Schreib - Lesespeicher (RAM)** mit ebenfalls maximal 64 Kilobyte Speicherkapazität dient zur Speicherung großer Datenmengen. Er wird über einen „MOVX“-Befehl und indirekte Adressierung mittels des „Datapointers“ (= DPTR) beschrieben oder ausgelesen.

Name: **XDATA SEGMENT**.

Bitte beachten:
Bei unserem Compuboard ist (wegen seiner Konzeption als Entwicklungssystem) folgende Aufteilung fest vorgegeben und kann nicht geändert werden:
Wir verwenden ein EPROM „27C256“ mit einer Speicherkapazität von nur **32 Kilobyte**, bei dem zwei Blöcke mit je 16 Kilobyte in die Bereiche **0000h bis 3FFFh** bzw. **8000h bis BFFFh** gelegt wurden.

Auch das **externe RAM** weist hier nur eine Größe von **32 Kilobyte** auf und ist in zwei Blöcke geteilt. Der erste Block reicht von **4000h bis 7FFFh**, der zweite von **C000h bis FFFFh**.

Wichtig: Auf die **XDATA-Segmente** kann hier nicht nur mit dem „**MOVX“-**Befehl zugegriffen werden, um Daten zu speichern und zu lesen. Wegen der verwendeten Adresslogik lassen sich hier auch **Programmcodes** unterbringen und dadurch diese Bereiche als **CODE Segmente** nutzen (= „von Neumann - Architektur“).

Erläuterung zur Adressierung:

a) Der Datentransport auf dem Controllerchip erfolgt immer mit einem „**MOVE“-Befehl**, der folgende Form aufweisen muss:

```
MOV   Ziel, Quelle
```

„**Direkte Adressierung**“ bedeutet nun einfach, dass in diesem Befehl **direkt die Adressen von Ziel und Quelle** stehen müssen. Ein Beispiel wäre die Übergabe des Akkuinhaltes an Port1:

```
MOV   P1,A
```

„**Indirekte Adressierung**“ liegt dann vor, wenn die eigentliche **Zieladresse** in einem **weiteren Register** enthalten ist und im „MOVE“-Befehl auf dieses Register als „Auskunftsbüro“ verwiesen wird.
Beispiel: die Zieladresse sei CAh. Sie wird im Register R7 bereitgestellt. Damit soll der Akkuinhalt im Register CAh abgelegt werden.

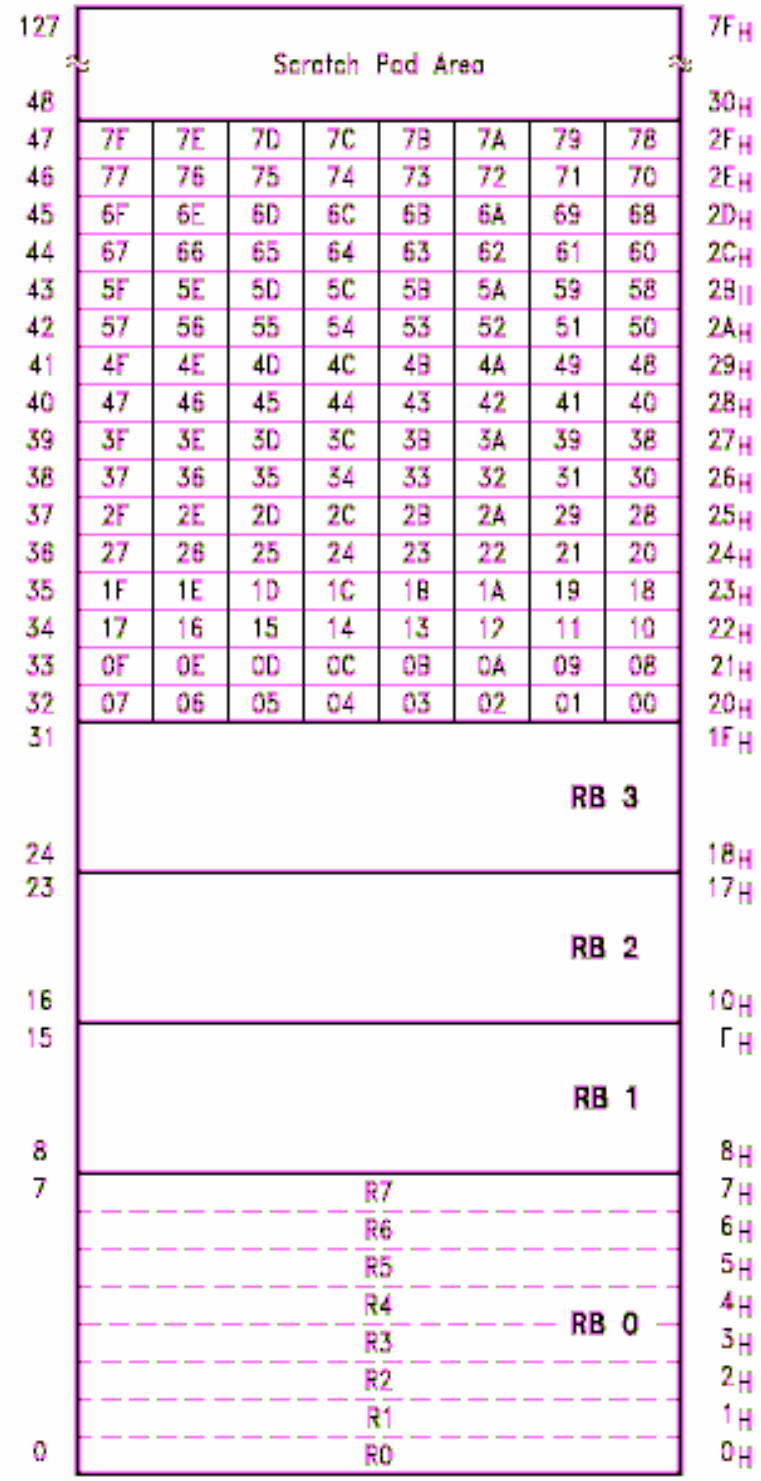
```
MOV R7,#0CAh        ;Adresse CAh nach R7
MOV @R7,A           ;Akkuinhalt nach CAh schreiben
```

b) Beim Datenverkehr mit dem **externen RAM** ist nur **indirekte Adressierung** zulässig, da die erforderliche 16-Bit-RAM-Adresse in den **Datapointer** geladen werden muss. Außerdem lautet nun der Befehl „**MOVX**“ (= steht für „move extern“).

Beispiel: der Akkuinhalt soll unter der externen Adresse 7000h gespeichert werden.

```
MOV DPTR,#7000h     ;Speicheradresse 7000h in Datapointer schreiben
MOVX @DPTR,A        ;Akkuinhalt unter 7000h im RAM speichern
```

Organisation von RAM1 auf dem Controllerchip:



Ab **Adresse 30h bis Adresse 7Fh**: 80 freie Speicherplätze, üblicherweise als **DATA-SEGMENT** eingesetzt (aber auch indirekt adressierbar).

Bitadressierbarer Bereich =16 Register zu je 8 Bit. (Registeradressen von **20h bis 2Fh**)

Vier Registerbänke zu je 8 Registern im Adressbereich von 00h bis 1Fh (Sie heißen stets „R0....R7“).

Die Umschaltung auf eine andere Registerbank erfordert stets **zwei Maßnahmen:**

a) eine **Direktive** innerhalb der Definition des CODE-SEGMENTES. Darin muss die Nummer der gewünschten Bank angegeben werden (z. B. „USING 1“) und

b) einen **Assemblerbefehl** innerhalb des eigentlichen Code-Segments, durch den die Umschaltung der Bank tatsächlich ausgelöst wird (Dazu müssen im PSW-Register bestimmte Bits gesetzt oder gelöscht werden).

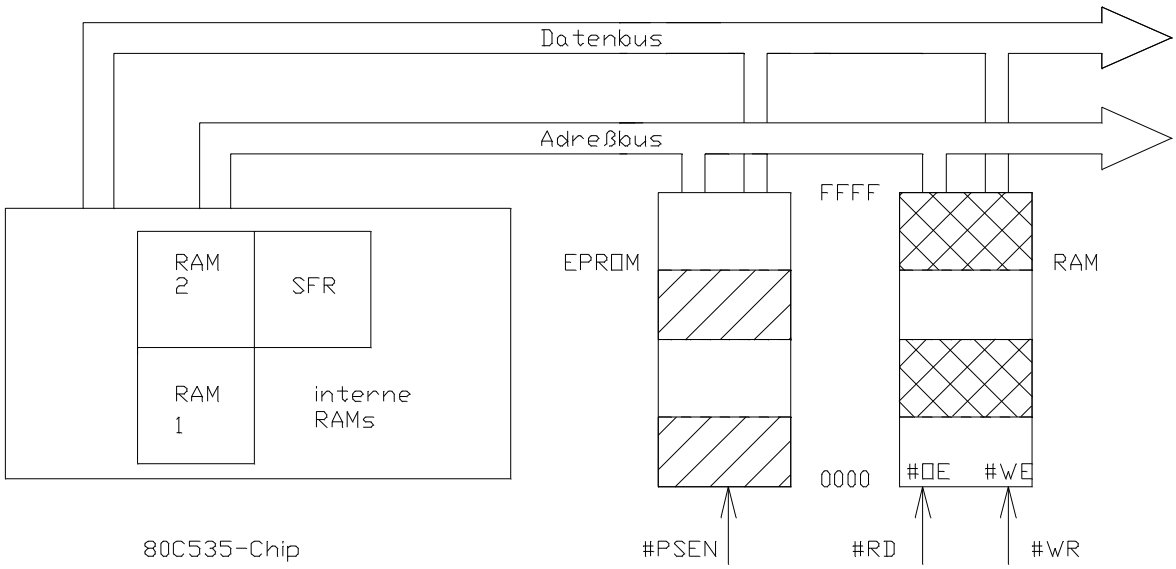
Weitere Information: **Alle Adressen und Bezeichnungen der Spezial - Funktions - Register beim Controller 80C535**

Table 4-1
Special Function Registers

	Symbol	Name	Address
*	P0	Port 0	80_H
	SP	Stack pointer	81_H
	DPL	Data pointer, low byte	82_H
	DPH	Data pointer, high byte	83_H
	PCON	Power control register	87_H
*	TCON	Timer control register	88_H
	TMOD	Timer mode register	89_H
	TL0	Timer 0, low byte	$8A_H$
	TL1	Timer 1, low byte	$8B_H$
	TH0	Timer 0, high byte	$8C_H$
	TH1	Timer 1, high byte	$8D_H$
*	P1	Port 1	90_H
*	SCON	Serial channel control register	98_H
	SBUF	Serial channel buffer register	99_H
*	P2	Port 2	$0A0_H$
*	IEN0	Interrupt enable register 0	$0A8_H$
	IP0	Interrupt priority register 0	$0A9_H$
*	P3	Port 3	$0B0_H$
*	IEN1	Interrupt enable register 1	$0B8_H$
	IP1	Interrupt priority register 1	$0B9_H$
*	IRCON	Interrupt request control register	$0C0_H$
	CCEN	Compare/capture enable register	$0C1_H$
	CCL1	Compare/capture register 1, low byte	$0C2_H$
	CCH1	Compare/capture register 1, high byte	$0C3_H$
	CCL2	Compare/capture register 2, low byte	$0C4_H$
	CCH2	Compare/capture register 2, high byte	$0C5_H$
	CCL3	Compare/capture register 3, low byte	$0C6_H$
	CCH3	Compare/capture register 3, high byte	$0C7_H$
*	T2CON	Timer 2 control register	$0C8_H$
	CRCL	Compare/reload/capture register, low byte	$0CA_H$
	CRCH	Compare/reload/capture register, high byte	$0CB_H$
	TL2	Timer 2, low byte	$0CC_H$
	TH2	Timer 2, high byte	$0CD_H$
*	PSW	Program status word register	$0D0_H$
*	ADCON	A/D converter control register	$0D8_H$
	ADDAT	A/D converter data register	$0D9_H$
	DAPR	D/A converter program register	$0DA_H$
	P6	Port 6	$0DB_H$ 1)
*	ACC	Accumulator	$0E0_H$
*	P4	Port 4	$0E8_H$
*	B	B register	$0F0_H$
*	P5	Port 5	$0F8_H$

The SFR's marked with an asterisk (*) are bit- and byte-addressable.
1) Additional feature of the ACMOS versions

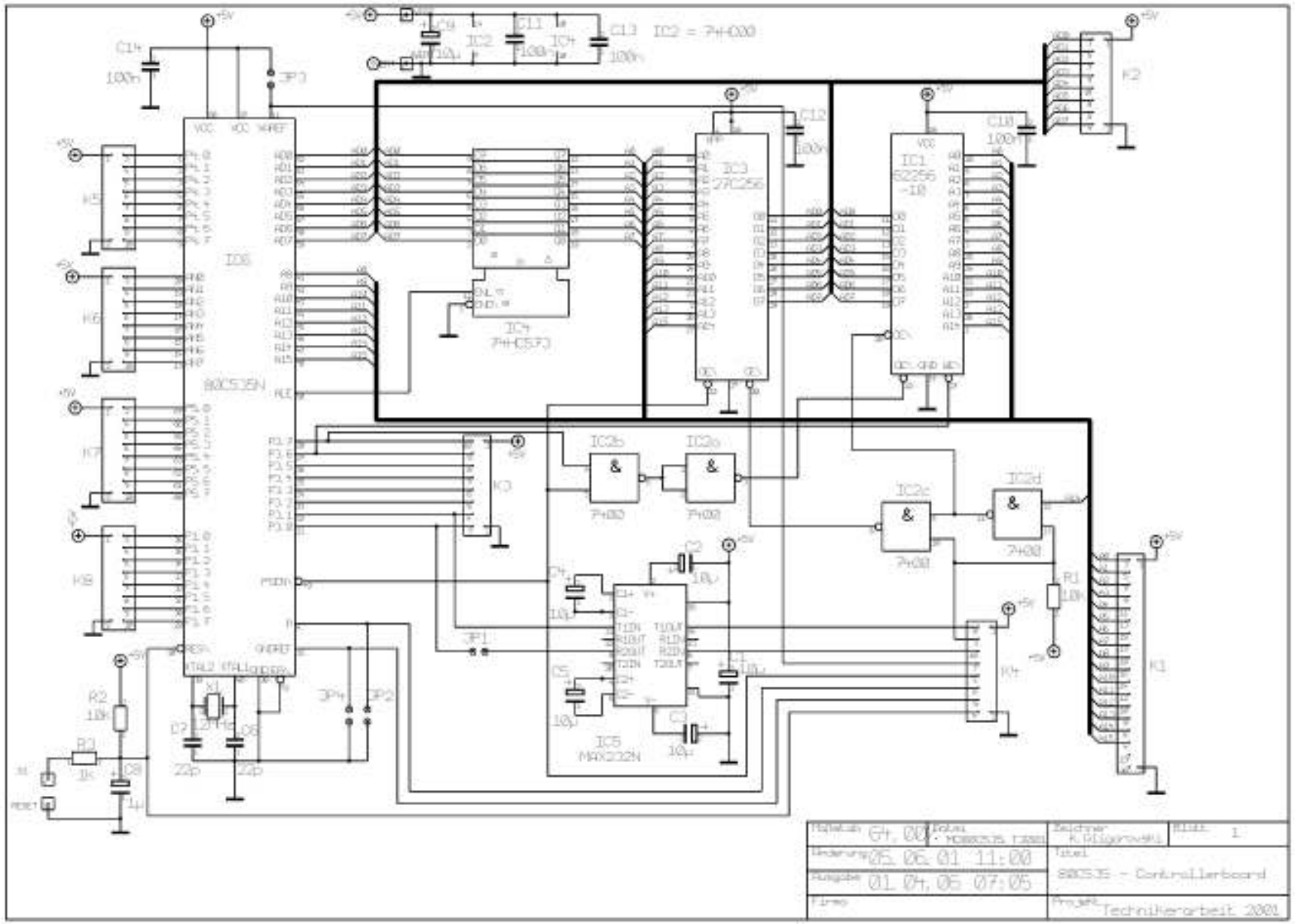
Zusammenfassung: Speicherorganisation des „EST-Compuboards 80C535“



Aufgabe: a) Markieren Sie in unterschiedlichen Farben den **CODE-, DATA-, XDATA-und IDATA-Bereich**.

b) Auf dem nächsten Blatt folgt der Stromlaufplan des verwendeten „EST-Compuboard 80C535“. Markieren Sie darin in Farbe alle im obigen Übersichtsschaltplan eingetragenen Baugruppen.

c) Skizzieren Sie in dieser unteren Blatthälfte den prinzipiellen Aufbau der Zusatzlogik, die für eine „von Neumann-Architektur“ in den Adressbereichen 4000h....7FFFh bzw. C000h....FFFFh sorgt.


IC2 = 74HC00
IC1 S2256 -10
IC3 27C256
IC4 74HC573
IC5 MAX232N
80C535N
IC5
IC2b
IC2a
IC2c
IC2d
74HC00
K1
K2
K3
K4
K5
K6
K7
K8
JP1
JP2
JP3
JP4
X1
R1 10K
R2 10K
R3 1K
C1 10µ
C2 10µ
C3 10µ
C4 10µ
C5 10µ
C6 22p
C7 22p
C8 1µ
C9 100n
C10 100n
C11 100n
C12 100n
C13 100n
C14 100n
+5V
Datum 07.06.01
Änderung 05.06.01 11:00
Ausgabe 01.04.05 07:05
Datei: MC80C535_T2001
Zeichner: K.Hilgarth
Blatt: 1
Titel: 80C535 - Controllerboard
Projekt: Technikerarbeit 2001