

# Mikroprozessorsysteme TK / Rechnerarchitektur

Haiko Wick

Skript zur Vorlesung von Herrn Hinsberger

Letzte Aktualisierung 21.07.2006 (RL 3)

**Kommentare und Anregungen: [mptkra@haikowick.de](mailto:mptkra@haikowick.de)**

# Inhaltsverzeichnis

<b>1 Grundlagen</b>	<b>4</b>
1.1 Mikroprozessor & Mikrocomputer . . . . .	4
1.1.1 Motivation . . . . .	4
1.2 Aufbau eines Mikrocomputers . . . . .	4
1.3 CISC & RISC-Prozessoren . . . . .	5
1.3.1 CISC - Complex Instruction Set Computer . . . . .	5
1.3.2 RISC - Reduced Instruction Set Computer . . . . .	5
1.4 Pipelining . . . . .	6
1.4.1 Befehlsabarbeitung beim CISC . . . . .	6
1.4.2 Befehlsabarbeitung beim RISC . . . . .	7
1.4.3 Hemmnisse beim Pipelining (Pipeline Interlocks) . . . . .	7
<b>2 Der 8086</b>	<b>8</b>
2.1 Blockschaltbild 8086 . . . . .	8
2.1.1 Grundsätzlich . . . . .	8
2.1.2 Execution Unit . . . . .	9
2.1.3 ALU (Arithmetic Logic Unit) . . . . .	9
2.1.4 Steuerwerk . . . . .	9
2.1.5 Bus Interface Unit . . . . .	9
2.1.6 Befehlswarteschlange (Instruction Queue) . . . . .	9
2.2 Der 8086 Registersatz . . . . .	10
2.2.1 Allgemeine Register / Universalregister . . . . .	10
2.2.2 Zeiger & Indexregister . . . . .	10
2.2.3 Segmentregister . . . . .	11
2.2.4 Instruction Pointer . . . . .	11
2.2.5 Flag Register (Control Status Word) . . . . .	11
2.2.6 Zusammenhängende Register . . . . .	12
2.2.7 20 Bit Adresse . . . . .	12
<b>3 80186 <math>\mu P</math> (Intel)</b>	<b>14</b>
3.1 Kenndaten . . . . .	14
3.2 Blockschaltbild . . . . .	15
3.2.1 Taktgenerator . . . . .	15

3.3	Die BIU und die Kontrolle des Daten- und Adressbusses . . . . .	15
3.3.1	Besonderheit . . . . .	16
3.3.2	Busleitungen . . . . .	16
3.4	Der 80186 im Minimum-Mode . . . . .	16
3.4.1	Erklärung der verwendeten Anschlüsse . . . . .	16
3.4.2	Ablauf eines Datenaustauschs zwischen $\mu P$ und RAM (allgemein) . . . . .	16
3.4.3	Zeitlicher Ablauf eines Schreibvorgangs am Datenbus . . . . .	17
3.4.4	Lesen von 8 Bit / 16 Bit aus byteweise organisierten Speicherbausteinen . . . . .	18
3.5	Der Peripheral Control Block (PCB) . . . . .	19
3.6	Relocation Register . . . . .	19
3.7	Die CS-Einheit . . . . .	20
3.7.1	Erweiterung . . . . .	20
3.8	CS-Speichereinteilung beim 80186 (RAM) . . . . .	21
3.8.1	Die $\overline{LCS}$ -Leitung und das LMCS-Register . . . . .	21
3.8.2	Die $\overline{UCS}$ -Leitung und das UMCS-Register . . . . .	21
3.8.3	$\overline{MCS0}:\overline{MCS3}$ und die Register MPCS und MMCS . . . . .	22
3.8.4	Beispiel: Aufbau RAM (1 MB) . . . . .	22
3.8.5	Upper Memory . . . . .	23
3.8.6	Beispiel: Fortsetzung . . . . .	24
3.9	Die CS-Leistungen und die Peripherie . . . . .	25
3.9.1	PACS-Register . . . . .	25
3.9.2	Die PCS-Leitungen und das MPCS Register . . . . .	26
3.9.3	Ready Control . . . . .	27
3.9.4	Ready-Control (Grundsätzliches) . . . . .	27
3.9.5	External Ready Control . . . . .	27
3.9.6	Anschluß - Ready Control . . . . .	29
3.9.7	Beispiel: Ready-Anschluss einer asynchronen Speicherbank . . . . .	29
3.10	Direct Memory Access (DMA) . . . . .	30
3.10.1	Ablauf DMA . . . . .	31
3.10.2	Mögliche DMA-Übertragungsrichtungen . . . . .	31
3.10.3	Die DMA-Einheit des 80186 . . . . .	32
3.11	Register DMA-Einheit (80186) . . . . .	32
3.11.1	Offsetadressen im PCB . . . . .	33
3.11.2	Die DMA-Register . . . . .	33
3.11.3	Programmierung des DMA-Control-Register . . . . .	33
3.12	Interrupt-Control . . . . .	36
3.12.1	Polling . . . . .	36
3.12.2	Interrupts . . . . .	36
3.12.3	Interrupt-Quellen (80186) . . . . .	36
3.12.4	IR-Prioritäten . . . . .	37
3.12.5	Interrupt Nesting . . . . .	38

3.12.6	Ablauf eines IR (grundsätzlich) . . . . .	38
3.12.7	Aufbau der IR-Vektortabelle . . . . .	38
3.12.8	IR-Kaskadierung im Master Mode . . . . .	39
3.12.9	Ablauf eines IR im Kaskade-Mode . . . . .	40
3.12.10	Anfordern der IR-Vektornummer . . . . .	40
3.13	Der 80186 IR-Controller im Slave-Mode . . . . .	41
3.13.1	Ablauf . . . . .	41
3.13.2	Übersicht über den 80186-IR-Controller und dessen Konfiguration . . . . .	42
3.14	Programmierung der PCB Register . . . . .	43
3.14.1	Programmierung . . . . .	44
3.15	Die TIMER/COUNTER Einheit des 80186 . . . . .	44
3.15.1	Die Modus Control Register der Zählereinheit (INTEL 80186) . . . . .	45
3.15.2	Anwendungsbeispiele Timer . . . . .	46

# Kapitel 1

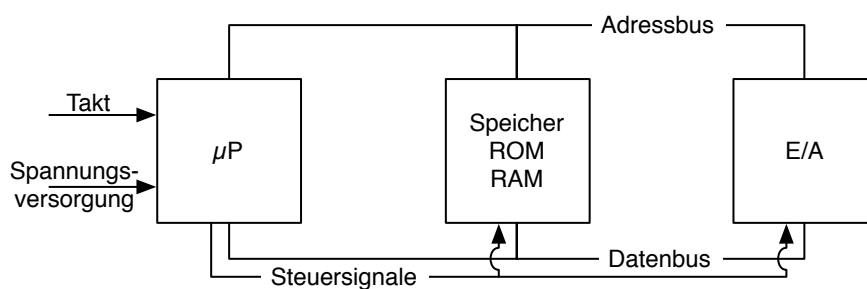
## Grundlagen

### 1.1 Mikroprozessor & Mikrocomputer

#### 1.1.1 Motivation

- Zum Lösen bestimmter Berechnungen mussten spezifische Schaltungen gebaut werden, die nur eine bestimmte Aufgabe erfüllen konnten.
  - addieren / subtrahieren
  - multiplizieren
  - logische Verknüpfungen
- Solche Schaltungen, die aus Verknüpfungsgliedern und FlipFlops aufgebaut werden, waren außerordentlich teuer.
- Idee einer programmierbaren “Universalschaltung”.

### 1.2 Aufbau eines Mikrocomputers



- zentrale Einheit des  $\mu$ -Computers ist der  $\mu$ -Prozessor.
  - Rechnerwerk / Steuerwerk
  - interne Register
- Speicher und E/A-Einheiten (E/A-Einheit benötigt zum Anschluss von externen Geräten (Peripherie))

## 1.3 CISC & RISC-Prozessoren

### 1.3.1 CISC - Complex Instruction Set Computer

- komplexe Befehle
  - Abarbeitung eines Befehls mit Hilfe von mehreren Mikrooperationen im Mikroprogramm (physikalische Ausführung im Prozessor / Register)
  - Das Mikroprogramm befindet sich im Mikrocode ROM
  - Befehlsdeko­der springt entsprechend des abzu­arbei­ten­den Befehls an die entsprechende Position im Mikrocode ROM

#### Vorteile CISC

- Einem Prozessor können neue Befehle zugeordnet werden (Erweiterung des Befehlssatz)
  - ⇒ Zusammensetzung aus geeigneten Abschnitten des  $\mu$ -Codes
  - ⇒ Flexibilität
- Kompatibilität
- Bei neuen Prozessoren kann der Befehlssatz von Vorgängen auf “Softwareebene” nachgebildet werden.
- Befehlssatz kann den Anforderungen einer Applikation angepasst werden.

⇒ Folge: Immer komplexere Befehle

⇒ Kostet die Komplexität der Befehle nicht zu viel Performance?

*Damalige Antwort:*

- sehr langsame Hauptspeicher
- während eines Hauptspeicherzugriffs konnte der Mikroprozessor 10mal oder mehr auf seine internen Register zugreifen.
- Begrenzender Faktor daher der Hauptspeicherzugriff und nicht die Abarbeitung des Mikroprogramms.
- ⇒ Genügend Zeit um Mikroprogramme auszuführen
- Prozessor und Systembus daher ungefähr gleich ausgelastet.

*Heutige Antwort:*

- Durch Caches und schnellere Hauptspeicherzugriffe hat sich dies geändert.
- Daher ist der Systembus nicht ausgelastet
- ⇒ Mikroprozessor langsamste Stelle im System

### 1.3.2 RISC - Reduced Instruction Set Computer

- einfache Befehle sind mit möglichst hohem Tempo abzuarbeiten
- Verzicht auf Mikroprogrammierung
- ausschließlich Befehle, die sich auf die Hardware direkt beziehen werden verwendet
- Prinzipiell kann der Befehl eines RISC Prozessors mit einer Mikrooperation beim CISC Prozessor verglichen werden.
- **Frage:** Verlust durch das Einsparen der komplexen Befehle?

## IBM Untersuchung (Ende der 70er Jahre)

- 80% aller Berechnungen werden mit nur 20% aller Befehle abgehandelt.
- Es werden sehr viele Berechnungen mit den gleichen Befehlsfolgen durchgeführt
- $\Rightarrow$  Ausführungszeit für komplexe Befehle größer als für eine gleichwertige Folge einzelner hardwarenaher Befehle.
  - Reduktion des Befehlssatzes
  - wichtigste Befehle in Hardware realisiert (“fest verdrahtet”)
  - mehrere getrennte interne Bussysteme
  - voneinander unabhängige Verarbeitungseinheiten
  - **Pipelining**

## 1.4 Pipelining

- Verbesserung der Performance durch Erhöhung des Befehlsdurchsatzes
- Parallele Ausführung mehrerer Instruktionen
- Abarbeitung eines Befehles (grundsätzlich):
  1. Instruction Fetch - *Einlesen des Befehls*
  2. Decode - *Dekodieren des Befehls*
  3. Operand Fetch - *Einlesen der Operanden*
  4. Execute - *Ausführen des Befehls*
  5. Write Back - *Zurückschreiben der Ergebnisse*

### 1.4.1 Befehlsabarbeitung beim CISC

Instruction Set	Decode	Operand Set	Execute	Write Back	
B1	-	-	-	-	1. Takt
-	B1	-	-	-	2. Takt
-	-	B1	-	-	3. Takt
-	-	-	B1	-	4. Takt
-	-	-	-	B1	5. Takt
B2	-	-	-	-	6. Takt
-	B2	-	-	-	7. Takt
-	-	B2	-	-	8. Takt
-	-	-	B2	-	9. Takt
-	-	-	-	B2	10. Takt

- Befehle werden sequentiell abgearbeitet
- Pipeline daher schlecht ausgelastet (Ineffizient)
- **Idee:** Pipelining
- zeitlich parallele Ausführung mehrerer Befehle in unterschiedlichen Verarbeitungsschritten
- Somit verbesserte Auslastung der Pipeline (Idealfall: 100%) nach einer kurzen Anlaufzeit (Latenz)
- Idealfall somit: Pro Takteinheit ein Befehl abgearbeitet.

### 1.4.2 Befehlsabarbeitung beim RISC

Instruction Set	Decode	Operand Set	Execute	Write Back	
B1	-	-	-	-	1. Takt, Latenz
B2	B1	-	-	-	2. Takt, Latenz
B3	B2	B1	-	-	3. Takt, Latenz
B4	B3	B2	B1	-	4. Takt, Latenz
B5	B4	B3	B2	B1	5. Takt, volle Auslastung
B6	B5	B4	B3	B2	6. Takt, volle Auslastung
B7	B6	B5	B4	B3	7. Takt, volle Auslastung
B8	B7	B6	B5	B4	8. Takt, volle Auslastung
B9	B8	B7	B6	B5	9. Takt, volle Auslastung
B10	B9	B8	B7	B6	10. Takt, volle Auslastung

### 1.4.3 Hemmnisse beim Pipelining (Pipeline Interlocks)

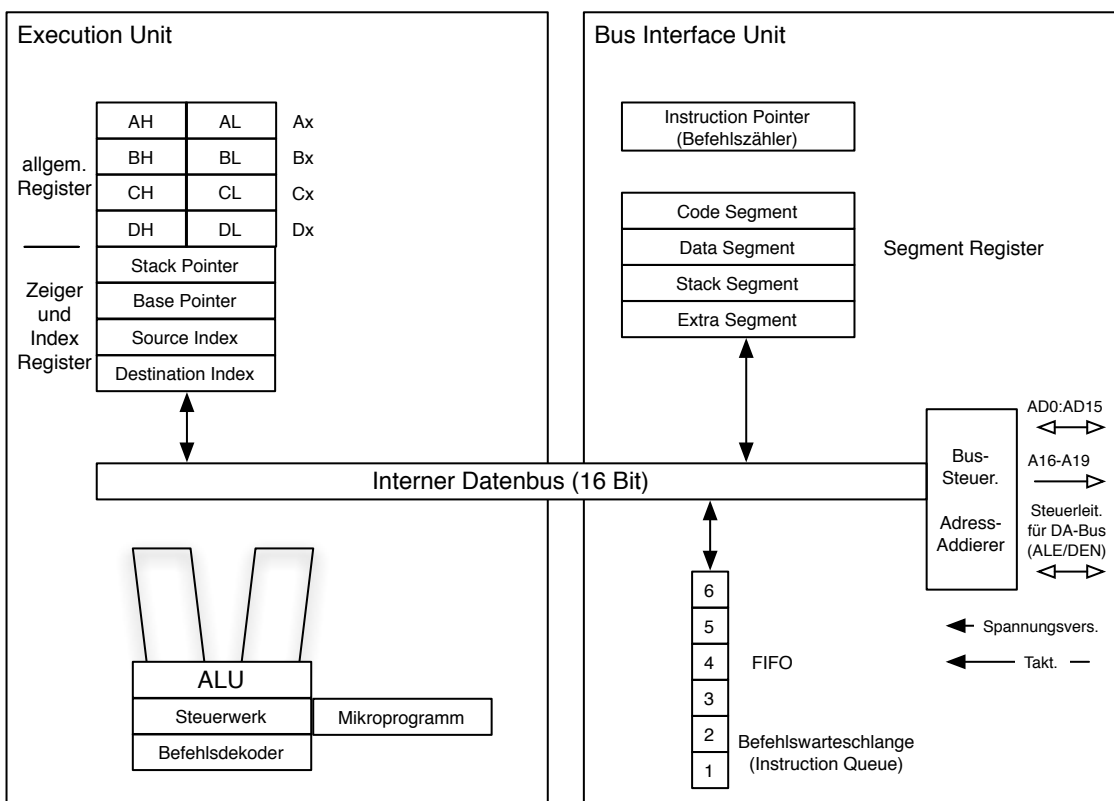
- A** Bei Hauptspeicherzugriffen dauert die Zuführung der Operanden ggf. mehrere Takte (Takteinheiten). Somit verschiebt sich die komplette weitere Verarbeitung von Schritten, da die Pipeline komplett blockiert wird.
- B** Mehrere Befehle aus unterschiedlichen Verarbeitungsstufen greifen gleichzeitig auf gleiche Baugruppen zu.  
⇒ Ressourcenkonflikte.
- C** Programmsprünge. Erfolgt in Schritt B2 ein Sprung zum Befehl B4 wurde B3 bereits bis zum Execute verarbeitet und muss wieder rückabgewickelt werden.
- D** Datenabhängigkeiten: Ein Befehl benötigt Operanden, die sich noch in Verarbeitung eines vorhergehenden Befehls steckt. Wird z.B. in B3 das Ergebnis von B2 als Operand benötigt, ist dies nicht möglich, bevor B2 nicht verarbeitet wurde. Die CPU wartet an dieser Stelle, bis B2 abgearbeitet ist und der Operand geladen werden kann.

# Kapitel 2

## Der 8086

### 2.1 Blockschaltbild 8086

Blockschaltbild 8086



#### 2.1.1 Grundsätzlich

- zwei Einheiten: BIU / EU
- weitgehend unabhängig voneinander
- ⇒ Erhöhen der Verarbeitungsgeschwindigkeit

### 2.1.2 Execution Unit

- Dekodieren der Befehle (Befehlsdekoder)
- Ausführen der zugehörigen Mikrooperationen unter Verwendung der zugehörigen Register (Mikroprogramm)
- keine direkte Verbindung zum externen DA-Bus

### 2.1.3 ALU (Arithmetic Logic Unit)

- führt bitweise logische Operationen aus
- führt bitweise arithmetische Operationen aus
  - Inkrementieren / Dekrementieren
  - Addieren / Subtrahieren
  - Multiplikation / Division
  - Schiebebefehle / Rotationsbefehle
  - AND / OR / EXOR

### 2.1.4 Steuerwerk

- übernimmt die dekodierten Befehle
- liest die Operanden ein
- koordiniert die zeitlichen Abläufe in den übrigen Komponenten des  $\mu P$ .
- Ausnahmesituationen (Interrupts / Programmsprünge)

### 2.1.5 Bus Interface Unit

- stellt Verbindung zum externen DA Bus her (Bussteuerung)
- Bussteuerung: Umsetzen des Daten- und Adressbetriebs (ALE/DEN)
- Breite des externen Datenbus  $\Rightarrow$  16 Bit (Word)
- Breite des externen Adressbus  $\Rightarrow$  20 Bit
- Multiplexen von Daten und Adressen (Reduktion der externen Anschlüsse)
- BIU stellt Datenaustausch mit angeschlossener Peripherie sicher (zusammen mit Chip Select  $\Rightarrow$  7 CS-Leitungen zur Freigabe angeschlossener Peripherie)
- Befehlsaustausch zwischen  $\mu P$  und Speicher
- $\Rightarrow$  Instruction Queue (Befehlswarteschlange) bildet einen Puffer zwischen BIU und EU

### 2.1.6 Befehlswarteschlange (Instruction Queue)

- besteht auf 6 Registern a 8 Bit
- sofern die BWS nicht komplett gefüllt ist sorgt die BIU dafür, dass die nachfolgenden Befehle eingelesen werden.
- $\Rightarrow$  Nutzen:

- Wenn die EU den letzten Befehl abgearbeitet hat muss nicht erst auf die externen Speicherzugriffe gewartet werden um den nächsten Befehl einzulesen.
- BIU und EU arbeiten weitestgehend unabhängig
- $\Rightarrow$  IQ dient als Puffer (Synchronisation)
- $\Rightarrow$  **Problematik:** IQ berücksichtigt keine Sprungbefehle  $\rightarrow$  Befehle im IQ, die nicht abgearbeitet werden müssen. Möglichkeit, dass trotz des IQ auf Befehle gewartet werden muss.

## 2.2 Der 8086 Registersatz

- 8 16-Bit Register (Universal-, Index- und Zeigerregister)
  - die vier Universalregister sind byteweise adressierbar (Kompatibilität zu anderen Prozessoren, deren Universalregister 8 Bit umfassten)
- Befehlsregister (Instruction Pointer)
- Status Flag Register
- vier 16-Bit Segment-Register

### 2.2.1 Allgemeine Register / Universalregister

- Ax / Akkumulator
  - Beinhaltet einen der Operanden bei vielen Berechnungen der ALU
  - viele Operationen werden ausschließlich über den Akkumulator durchgeführt (Multiplikation, Division, Shift- und Rotationsbefehle)
- Bx / Basisregister
  - wird zur indirekten Adressierung genutzt (Basisregister im Datensegment). Das Basisregister kann zur temporären Speicherung von Daten und als Zeiger auf die Basis von Datenobjekten (beispielsweise den Beginn eines Feldes) bei indirekter Adressierung verwendet werden.
- Cx / Zählerregister
  - Schleifen / Wiederholungsoperationen
- Dx / Datenregister
  - 16-Bit Multiplikationen / Divisionen (Operand + Ergebnis)
  - Bei indirekter Adressierung der Peripherie enthält dieses Register die Nummer des verwendeten Ein-/Ausgabekanals

### 2.2.2 Zeiger & Indexregister

- Offsetadressen (Adressbildung)
  - Stack Pointer / Base Pointer
    - \* Stackadressierung
    - \* Stack Pointer zeigt auf den zuletzt abgelegten Befehl im Stack
    - \* Base Pointer zeigt auf Daten im Stacksegment
  - Source-/Destination Index
    - \* Adressierung von Daten im Speicher
    - \* SI  $\Rightarrow$  Quelladresse
    - \* DI  $\Rightarrow$  Zieladresse
      - Offset beim Zugriff auf das Extrasegment

### 2.2.3 Segmentregister

- momentane Aufteilung des RAM in definierte funktionale Blöcke (logisch)
- Basisadressen der einzelnen Blöcke / Segmente
- Code Segment Register (CS)
  - Operationeller Code (Befehle)
- Data Segment Register (DS)
  - Datenblock
- Stack Segment Register (SS)
  - Stack
- Extra Segment Register
  - für spezielle String Operationen
  - bei Nutzung zweier unabhängiger Datenblöcke

### 2.2.4 Instruction Pointer

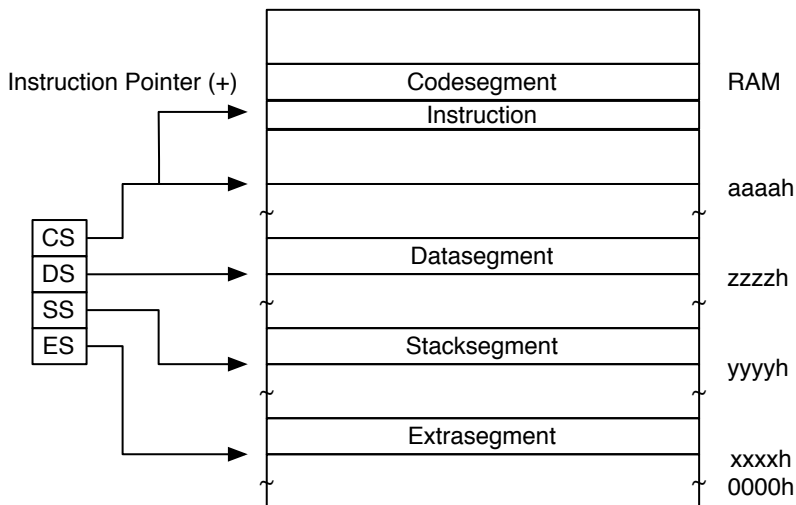
- Bildet den Offset der Adresse, die auf den aktuellen abzuarbeitenden Befehl zeigt.

### 2.2.5 Flag Register (Control Status Word)

- Verarbeitungszustand nach jeder Operation (Statusinformationen)
- Carry Flag (CF) bei arithmetischem Übertrag gesetzt
- Parity Flag (PF) Ist die Anzahl der Einsen im Codeword gerade, ist dieses Bit gesetzt.
- Auxiliary Flag (AF) Übertrag in der Wortmitte bei Dezimalberechnungen (BCD).
- Zero Flag (ZF) ist gesetzt, wenn das Ergebnis einer Operation Null ist.
- Sign Flag (SF) Vorzeichenflag (negativ = 1)
- Trap Single Step Flag (TF) Schrittweise abarbeiten der Befehle (TF=1) ⇒ Debugging
- Interrupt Enable Flag (IF) Steuert Funktionen von maskierbaren Interrupts (IF=0 ⇒ maskierte Interrupts werden ignoriert)
- Direction Flag (DF) Stringoperationen
  - bei Autoinkrement gelöscht
  - bei Autodrekrement gesetzt
- Overflow Flag (OF) in Verbindung mit dem Sign-Flag und dem Carry-Flag und signalisiert einen Überlauf des Ergebnisses

## 2.2.6 Zusammenhängende Register

1. Stack Pointer + Stacksegment
  - Adresse des letzten Befehls im Stack
2. Base Pointer + Stacksegment
  - Adresse eines bestimmten Befehls im Stack
3. Destination Index + Extra-Segment
  - Adressieren der Operanden bei bestimmten Operationen (String-Operationen)
4. Destination Index + Datensegment
  - Adressierung eines Ziels / Ergebnis bei Operationen / MOV-Befehl
5. Source Index + Datensegment
  - Adressierung einer Datenquelle
6. Basisregister + Datensegment
  - indirekte Adressierung
7. Instruction Pointer + Codesegment
  - Zeigt auf Adresse des aktuell abzuarbeitenden Befehls



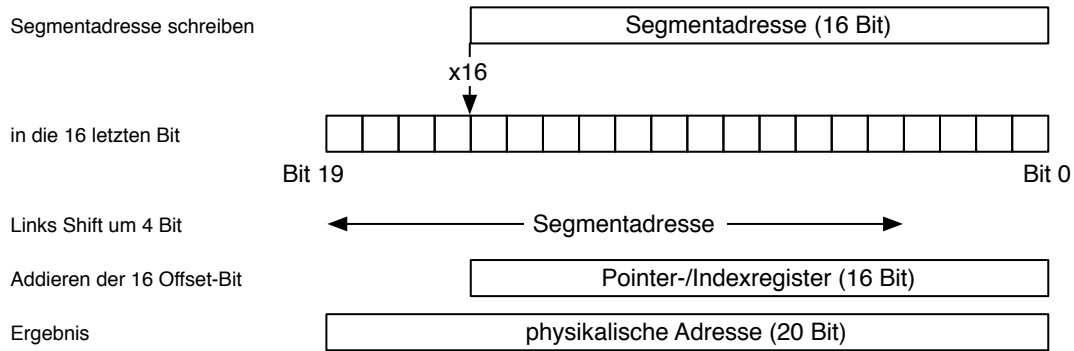
- Segmentregister teilt den RAM in gleichgroße Blöcke auf (nur für eine bestimmte Dauer)
- Größe  $\Rightarrow$  64 kByte
- Segmente können gleich sein bzw. sich überdecken

## 2.2.7 20 Bit Adresse

(Bildung der physikalischen RAM-Adresse)

- 8086/80186 hat 1 MByte RAM, die über 20 Adressleitungen adressiert werden ( $2^{20} = 1M$ )
- $\Rightarrow$  Frage: Wie erhält man aus 16-Bit Registern eine 20-Bit Adresse?
- $\Rightarrow$  Systematische Addition von Segment- und Index- bzw. Pointerregister

Zu der 16-Bit Adresse, die in den Segmentregistern, steht wird der Offset aus den Pointern- bzw. Indexregistern addiert. Zuvor wird allerdings die 16-Bit-Adresse um 4 Bits links geschiftet, also mit 0en aufgefüllt um Platz zu schaffen.



## Kapitel 3

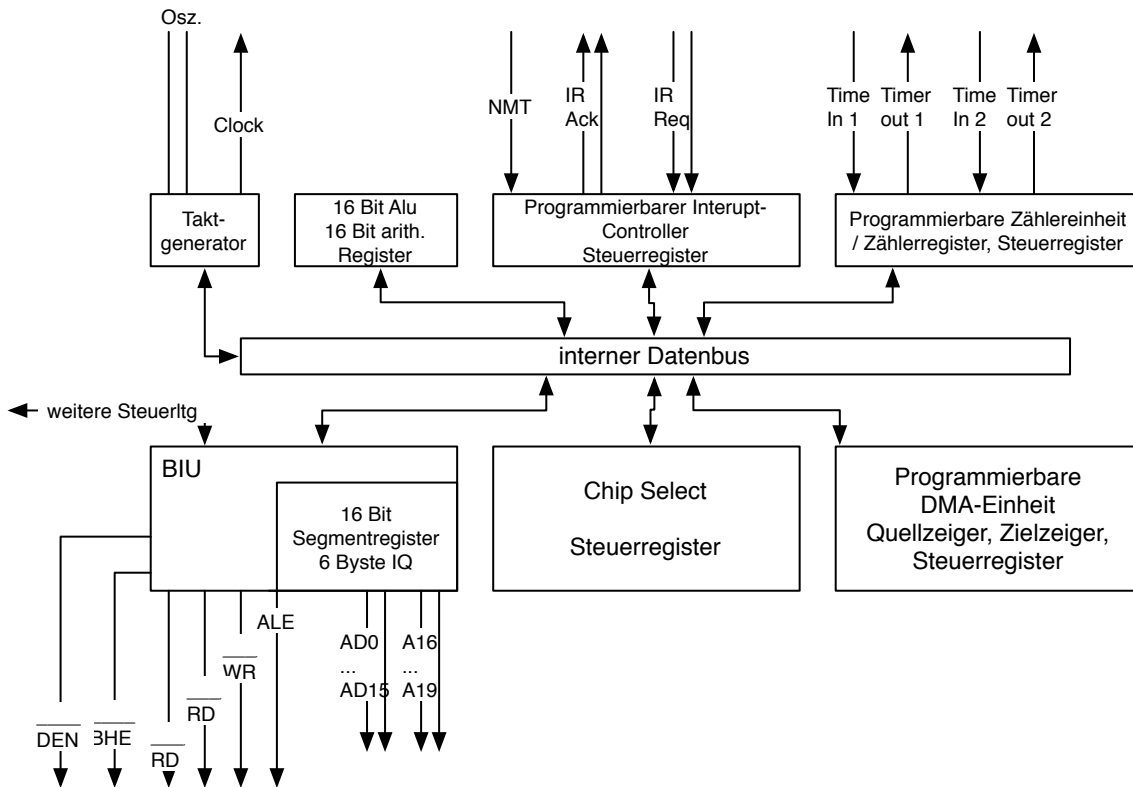
# 80186 $\mu P$ (Intel)

- Vorteile wegen Robustheit (Mean time between failure MTBF)
- palmähnliche Engeräte
- CISC
- $\Rightarrow$  keine parallele Abarbeitung der Befehle (Pipelining)

### 3.1 Kenndaten

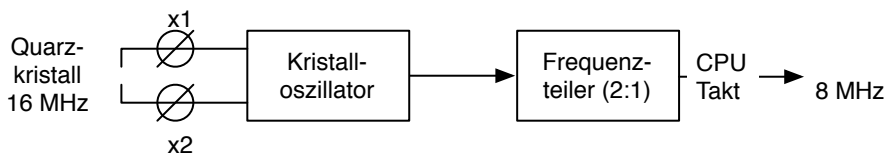
- MTBF > 15 Jahre
- Temperaturbereich  $-155$  Grad C bis  $125$  Grad C
- geringe Kosten
- 16 Bit Datenbreite
- Taktung im MHz-Bereich
- 1 MB RAM

### 3.2 Blockschaltbild



#### 3.2.1 Taktgenerator

- versorgt alle internen Komponenten mit dem Takt
- versorgt Peripherie mit Takt (clock out)



(Quarz: 16MHz/20MHz/40MHz)

- TG besteht aus Kristalloszillator und Frequenzteiler
- CPU Takt steht zur Synchronisation an CLOCKOUT zur Verfügung
- Zählregister Eingangsfrequenz  $f_E = \frac{1}{8}$  Frequenzkristall  $\Rightarrow \frac{1}{4}$  CPU-Takt

### 3.3 Die BIU und die Kontrolle des Daten- und Adressbusses

Adressbreite: 20Bit  
 Datenbreite: 16Bit

### 3.3.1 Besonderheit

Adress- und Datenbus sind gemultiplext  $\Rightarrow$  verwenden gleiche Anschlüsse.

### 3.3.2 Busleitungen

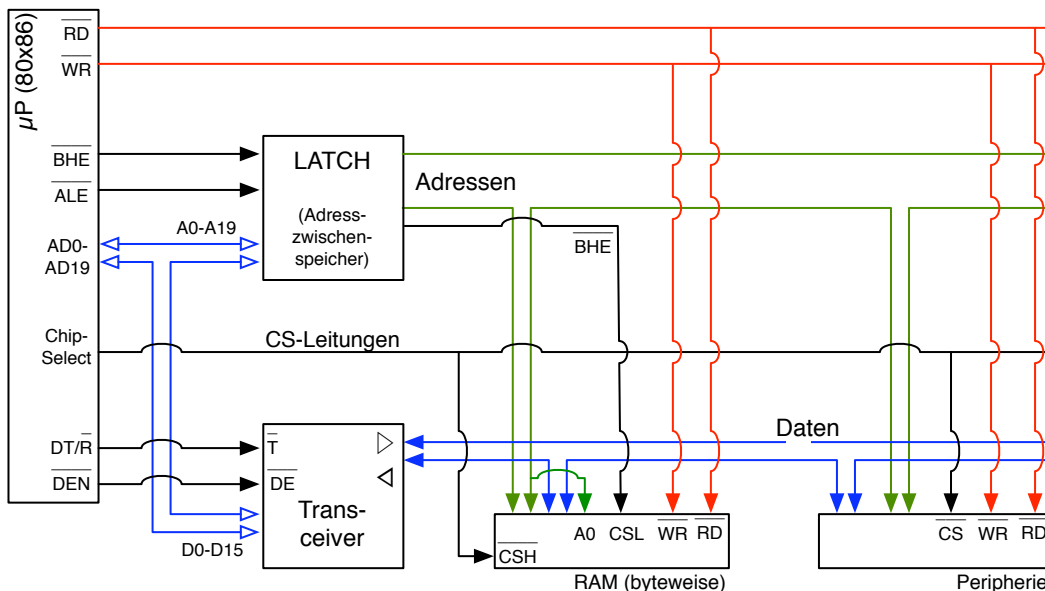
- AD0 bis AD15 (16 Daten- und Adressleitungen)
- AD16 bis AD19 (4 reine Adressleitungen)

Zum Lesen/Schreiben einer Operation bzw. eines Operanden müssen die Adressen an dem Speicherbaustein anliegen.

$\Rightarrow$  Frage: Wie können Daten über eine Leitung gelesen bzw. geschrieben werden, auf der zur gleichen Zeit Adressinformationen geführt wird?

$\Rightarrow$  Antwort: unmöglich.  $\Rightarrow$  zusätzliche Verschaltung nötig

## 3.4 Der 80186 im Minimum-Mode



### 3.4.1 Erklärung der verwendeten Anschlüsse

$\overline{RD}$   $\rightarrow$  Signalisiert, dass RAM/Peripherie Daten lesen soll.

$\overline{WR}$   $\rightarrow$  Signalisiert, dass RAM/Peripherie Daten schreiben soll.

DT /  $\overline{R}$   $\rightarrow$  Data Transmit / Receive  $\Rightarrow$  Übertragungsrichtung.

$\overline{DEN}$   $\rightarrow$  Data Enable  $\Rightarrow$  Signalisiert die Gültigkeit der Daten. ( $\rightarrow$  Einschwingperiode)

ALE  $\rightarrow$  Adress Latch Enable  $\Rightarrow$  Gültigkeit der Adressen. (Ob Adresse oder Daten am Bus anliegen)

$\overline{BHE}/A_0$   $\rightarrow$  16 Bit Speicherzugriff in byteweise organisierten Speichern.

### 3.4.2 Ablauf eines Datenaustauschs zwischen $\mu P$ und RAM (allgemein)

- $\mu P$  legt Adressen auf den Datenadressbus
- Adresslatch übernimmt die Adressen

- ALE-Steuerleitung gesetzt  $\Rightarrow$  Gültigkeit der Adressen
- Latch gibt die Adressen auf den "externen Adressbus" frei
- $\Rightarrow$  Adresse liegt an den Speicherbausteinen an
- 8 Bit / 16 Bit Datenaustausch mit Hilfe der Steuerleitungen  $\overline{BHE}/A_0$
- $\mu P$  nimmt die Adressen von Datenbus an

Schreibvorgang

- $\mu P$  setzt Daten auf Datenbus
- $\mu P$  setzt DT-Leitung (Senden)
- nach Einschwingperiode sind die Daten gültig  $\Rightarrow$  DEN gesetzt
- Transceiver gibt die Daten frei auf "externen Datenbus"
- $\mu P$  setzt  $\overline{RD}$ -Leitung
- RAM übernimmt Daten

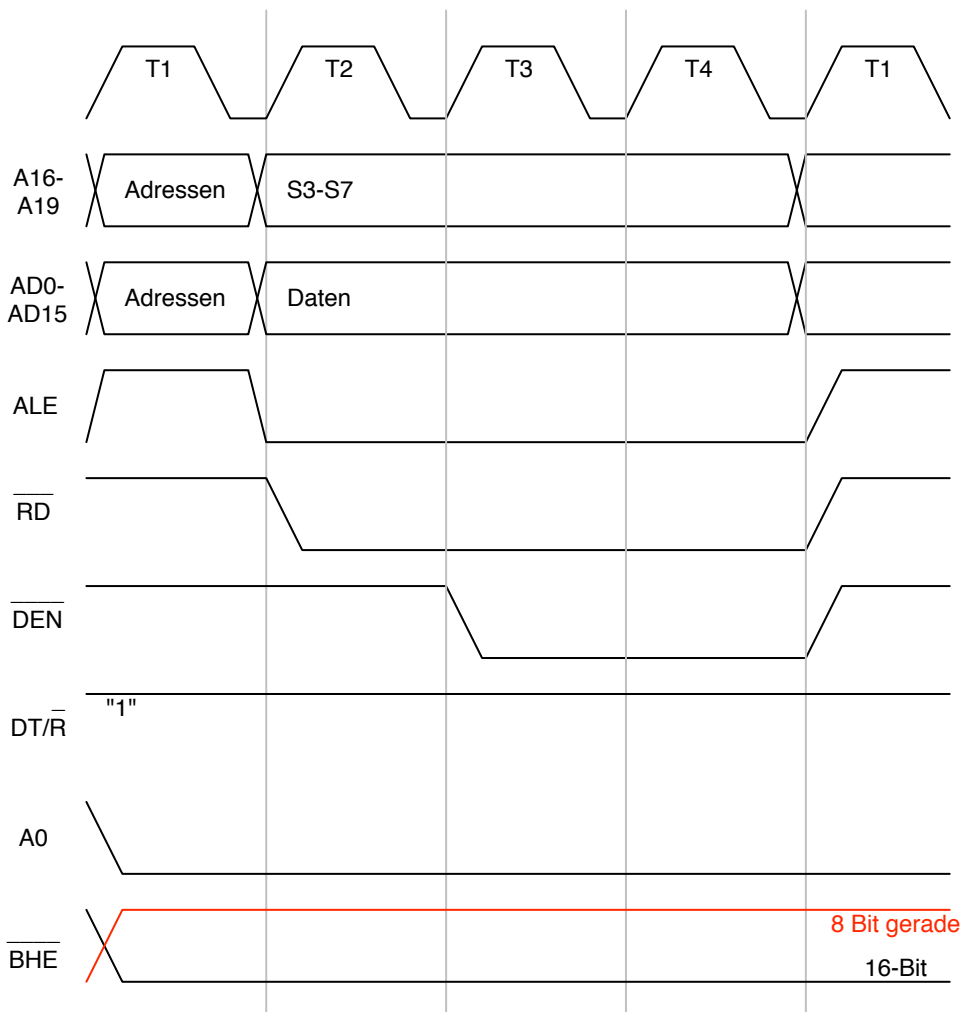
Lesevorgang

- $\mu P$  setzt  $\overline{R}$  (Receive)
- $\Rightarrow$  Transceiver (Daten werden in Richtung  $\mu P$  gesendet)
- $\mu P$  setzt  $\overline{WR}$  (Write)
- $\Rightarrow$  signalisiert, dass Daten vom RAM in Richtung  $\mu P$  übertragen werden sollen
- $\Rightarrow$  adressierte Speicherzelle legt Daten auf den Datenbus
- $\mu P$  übernimmt die Daten

$\mu P$  löscht alle Steuersignale und das Latch wird geflashed

- Datenaustausch mit Peripherie erfolgt ebenfalls mit Hilfe der Steuerleitung  $\overline{WR}/\overline{RD}$ ,  $DT/\overline{R}$  sowie mit Hilfe der CS-Leitung (Daten werden ebenfalls über DA-Bus und Transceiver geführt)

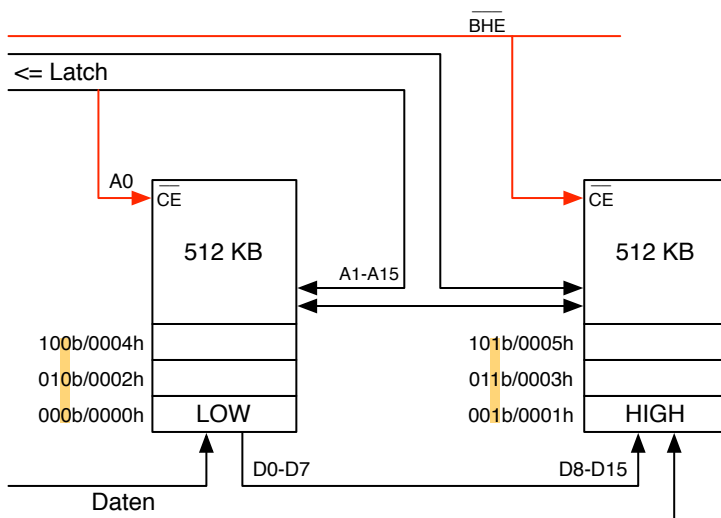
### 3.4.3 Zeitlicher Ablauf eines Schreibvorgangs am Datenbus



- Maschinenzyklus besteht aus 4 Takten
- Takt 1
  - Die Leitungen  $AD_0 : AD_{15}$  sowie  $A_{16} : A_{19}$  sind entsprechend der zu adressierenden Speicherzelle gesetzt
  - ALE-Signal kennzeichnet die Gültigkeit der Adressen am DA-Bus  $\Rightarrow$  Adressen werden in das Latch übernommen.
  - ALE-Signal wird gelöscht
  - Adresse liegt an den Speicherkomponenten an
- Takt 2
  - Daten werden auf die Leitungen  $AD_0 : AD_{15}$  gelegt
  - Bei Umschaltvorgängen von  $\overline{R}$  nach DT (bzw. umgekehrt) kann es zu Einschwingvorgängen kommen  $\Rightarrow$  Daten sind nicht unmittelbar gültig
  - $\overline{RD}$  wird gesetzt
- Takt 3/4
  - Daten gültig
  - Während  $\overline{DEN}$  gesetzt erfolgt die Datenübernahme durch den Speicher (Transceiver gibt Daten in den I/O-Bereich)
  - DT-Signal gesetzt  $\rightarrow$  Signalisiert Übertragungsrichtung von  $\mu P$  zum RAM

### 3.4.4 Lesen von 8 Bit / 16 Bit aus byteweise organisierten Speicherbausteinen

- Speicherzelle hat eine Größe von 8 Bit
- Beim 16-Bit Zugriff müssen zwei Speicherzellen gleichzeitig angesprochen werden können
- $\Rightarrow$  Systematischer Aufbau des RAM erforderlich (Vergabe der Adressen)
- $\Rightarrow$  Steuermöglichkeit zum Lesen/Schreiben von wahlweise einer bzw. zweier Speicherzellen



- 2 Speicherblöcke (High-, Low-Bereich)
  - $\rightarrow$  Low-Bereich  $\Rightarrow$  gerade Adressen

- $\rightarrow$  High-Bereich  $\Rightarrow$  ungerade Adressen
- $A_0$  und  $\overline{BHE}$  dienen als Steuerleitungen, die mit den CS-Eingängen der Speicherbausteine verbunden sind. (CS-Eingang unter Spannung  $\Rightarrow$  gesperrt)
- Adressen der aufeinander folgenden Speicherzellen in den beiden Speicherbausteinen unterscheiden sich ausschließlich am letzten Bit ( $A_0$ )  $\Rightarrow$  Steuerfunktion von  $A_0$ 
  - $A_0 = 0 \Rightarrow$  Low-Block durchgesteuert
  - $A_0 = 1 \Rightarrow$  High-Block durchgesteuert
  - (Binär:  $2^0 \Rightarrow$  gerade Adressen,  $2^1 \Rightarrow$  ungerade Adressen)

**Steuerfunktionen**

$\overline{BHE}$	$A_0$	Funktion
0	0	16 Bit Wort
0	1	ungerade Adresse / 8 Bit (High)
1	0	gerade Adresse / 8 Bit (Low)
1	1	beide Speicherblöcke gesperrt

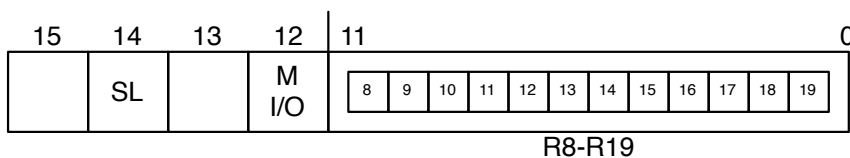
$\overline{BHE}$  und  $A_0 \Rightarrow$  physikalische Zustände an CS-Eingängen

- ohne Steuermöglichkeit von  $A_0$  und  $\overline{BHE}$ 
  - $\Rightarrow$  2 Bustypen sind nötig um 16-Bit zu lesen/schreiben (8 Takte (für einen Vorgang werden 4 Takte benötigt, siehe oben))
  - $\Rightarrow$  Busbreite von 8 Bit wäre ausreichend

**3.5 Der Peripheral Control Block (PCB)**

- kann entweder im I/O-Bereich oder im Main Memory (RAM) angelget werden.
- Beinhaltet programmierbare Register für jedes On-Chip Peripheral (Timer, CS-Einheit, Interrupt, DMA)
- Relocation-Register (RELREG)  $\Rightarrow$  Startadresse für den PCB
- Jedes Steuerregister hat einen definierten Offset  $\Rightarrow$  Adresse für ein bestimmtes Control-Register ergibt sich aus der Addition der Startadresse (RELREG) und dem zugehörigen Offset
- Offsetadressen für die On-Chip Peripherals des 80186  $\Rightarrow$  siehe Kopie aus den Intel-Specs.

**3.6 Relocation Register**



**R8 : R19**

- oberen Adressbits der PCB-Basisadresse (Startadresse) (niederwertigere Bits sind zu 0 definiert)

**Bit 12**

- indiziert, ob der PCB im RAM bzw. im externen Speicherbereich liegt

**Bit 14**

- indiziert, ob Interrupt-Controller im Master- bzw. Slave-Modus operiert.

⇒ Reset ⇒ 0000 0000 1111 1111<sub>(2)</sub> = 00FF<sub>h</sub>

### 3.7 Die CS-Einheit

(programmierbare Control-Register im PCB)

- 2 Aufgaben:
  1. Einteilung des Speicherblocks (RAM)  
⇒ 6 CS-Leitungen für die Einteilung des Speicherraums (physikalisch)
  2. Freigabesignale für zusätzlich angeschlossene Peripherie  
⇒ 7 CS-Leitungen (Optional: umprogrammieren zu Adressleitungen)
- 1. Unterteilung des Speichers in Speicherbänke (Speicherverschränkung / Interleaved Memory)  
Motivation: Verschaltung / Ansteuerung des RAM ausschließlich über  $\overline{BHE}$  und  $A_0$  als CS-Leitungen setzt voraus, dass man 2x512 kb Chips verwendet!
  - Speicherchips mit geringerer Kapazität sind preisgünstiger
  - Schnellere Zugriffszeiten bei kleineren Speicherbausteinen⇒ physikalische Aufteilung des Hauptspeichers
- Chips werden nicht mehr an den kompletten Adressbus angeschlossen  
⇒ nur niederwertige Adressbits ( $< A_{10}$ )
- ⇒ an mehreren Speicherzellen liegen ggf. die gleichen Adressen an.
- ⇒ Welche der Chips letztlich für den Speicherzugriff durchgesteuert sind wird durch die CS-Leitungen bestimmt
- ⇒ Die CS-Leitungen werden entsprechend zu den höherwertigen Adressbits gesteuert ( $> A_{10}$ )  
⇒ CS-Logik (integriert im 80186) ⇒ Umkodierung der höherwertigen Adressbits

⇒ **Problematik:**

- Die Entwicklung der Zugriffszeiten für Speichereinheiten steigt nicht proportional mit der Entwicklung der Geschwindigkeit der Prozessoren.
- ⇒ Divergenz zwischen Prozessor- und “Speicherleistung”
- ⇒ ineffizient weil die Performance des Prozessors nicht mehr ausgenutzt werden kann

#### 3.7.1 Erweiterung

(nicht 80186-Systeme)

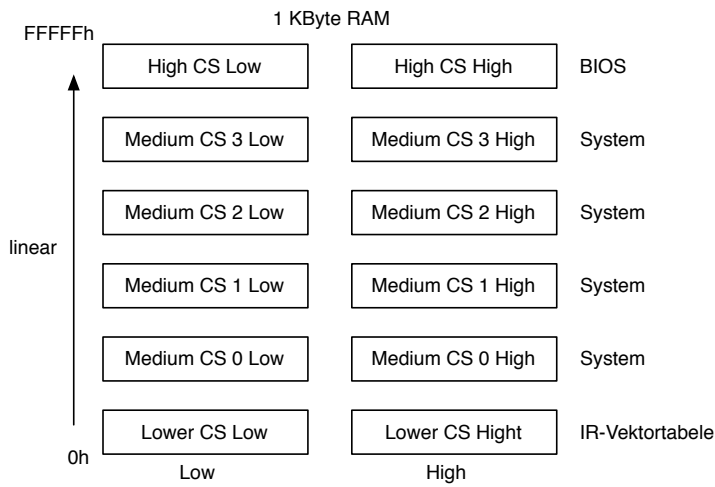
- Aufeinanderfolgende Adresse werden auf unterschiedlichen Speicherbänken

Adresse	xh	(x+1)h	(x+2)h	(x+3)h
Befehl/Operand	1	2	3	4
Speicherbank	1	2	3	4

⇒ Zugriffszeiten reduziert (Aufeinanderfolgende Befehle/Operanden befinden sich in der Regel in aufeinanderfolgenden Adressen)

### 3.8 CS-Speichereinteilung beim 80186 (RAM)

(physikalische)

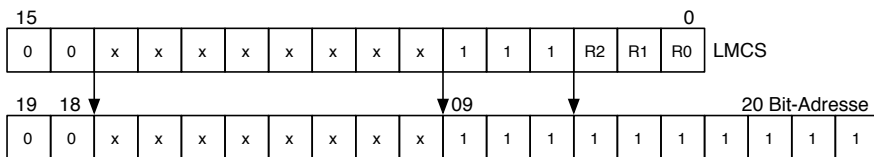


- Für alle Speicherbänke steht eine CS-Leitung zur Verfügung
- Jede CS-Leitung ist für einen definierten Adressbereich aktiv
- Adressräume können den CS-Leitungen zugemessen werden → Programmieren der zugehörigen Register im PCB!
- Nachfolgende Register werden verwendet um die CS-Leitungen zu programmieren.

#### 3.8.1 Die $\overline{LCS}$ -Leitung und das LMCS-Register

- adressierter Bereich beginnt per Definition mit der Adresse 0h
- LMCS legt die Blockgröße des “Lower Memory” (unterer Speicherbereich) fest.

#### LMCS-Register

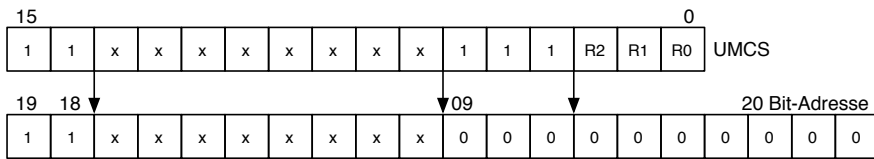


⇒ LMCS-Register definiert die obere Adresse des LM-Bereiches. ⇒ max. Länge von 256 kB (LM-Bereich beinhaltet die IR-Vektortabelle)

#### 3.8.2 Die $\overline{UCS}$ -Leitung und das UMCS-Register

- adressierter Bereich beginnt per Definition mit der Adresse FFFFh
- UMCS legt die Blockgröße des “Upper Memory” (oberer Speicherbereich) fest.

### UMCS-Register



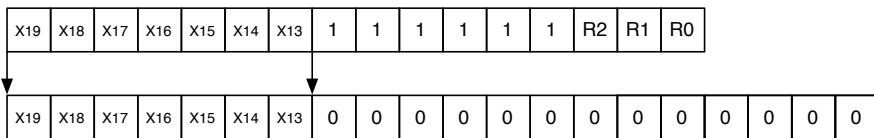
⇒ UMCS-Register definiert die untere Adresse des UM-Bereiches. (Startadresse des UM) ⇒ max. Länge von 256 kB (UM-Bereich beinhaltet das BIOS)

Nach Reset beginnt der  $\mu P$  bei Adresse FFFF0H!

### 3.8.3 $\overline{MCS0}:\overline{MCS3}$ und die Register MPCS und MMCS

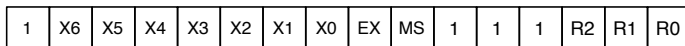
- Anfangsadresse im MMCS Register programmierbar
- Blocklänge wird im MPCS Register festgelegt

#### Das MMCS-Register



R2-R0 Ready-Bits für MCS-Leitungen

#### Das MPCS-Register



R2-R0 Ready Bits für  $\overline{PCS4} : \overline{PCS6}$

X0 : X6	Einzelblocklänge	Gesamtblocklänge
0000001	2k	8k
0000010	4k	16k
0000100	8k	32k
0001000	16k	64k
0010000	32k	128k
0100000	64k	256k
1000000	128k	512k

### 3.8.4 Beispiel: Aufbau RAM (1 MB)

Upper Memory:  $2 \cdot 128k = 256k$

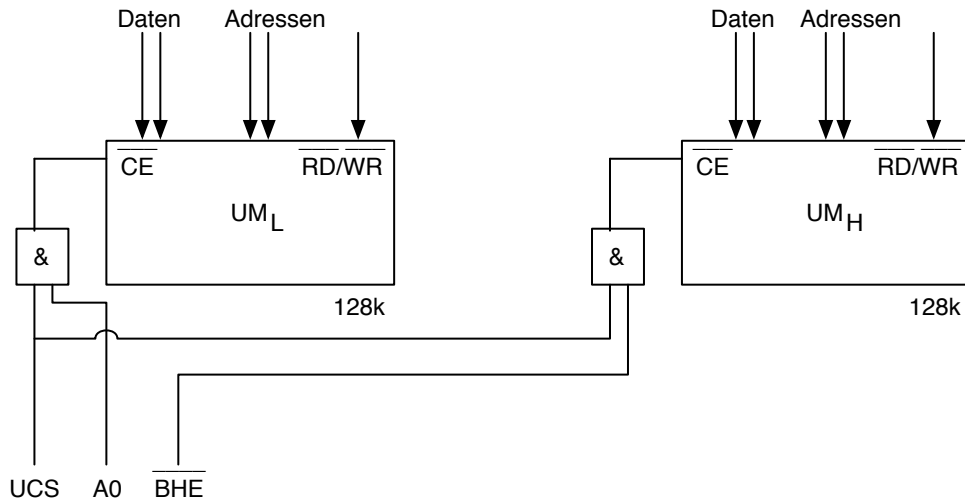
Lower Memory:  $2 \cdot 128k = 256k$

Mod Memory  $4 \cdot 2 \cdot 64 = 512k$

⇒ Gesamtspeicher: 1024k

( $2^{16} = 1,048567$  M physikalisch adressierbar)

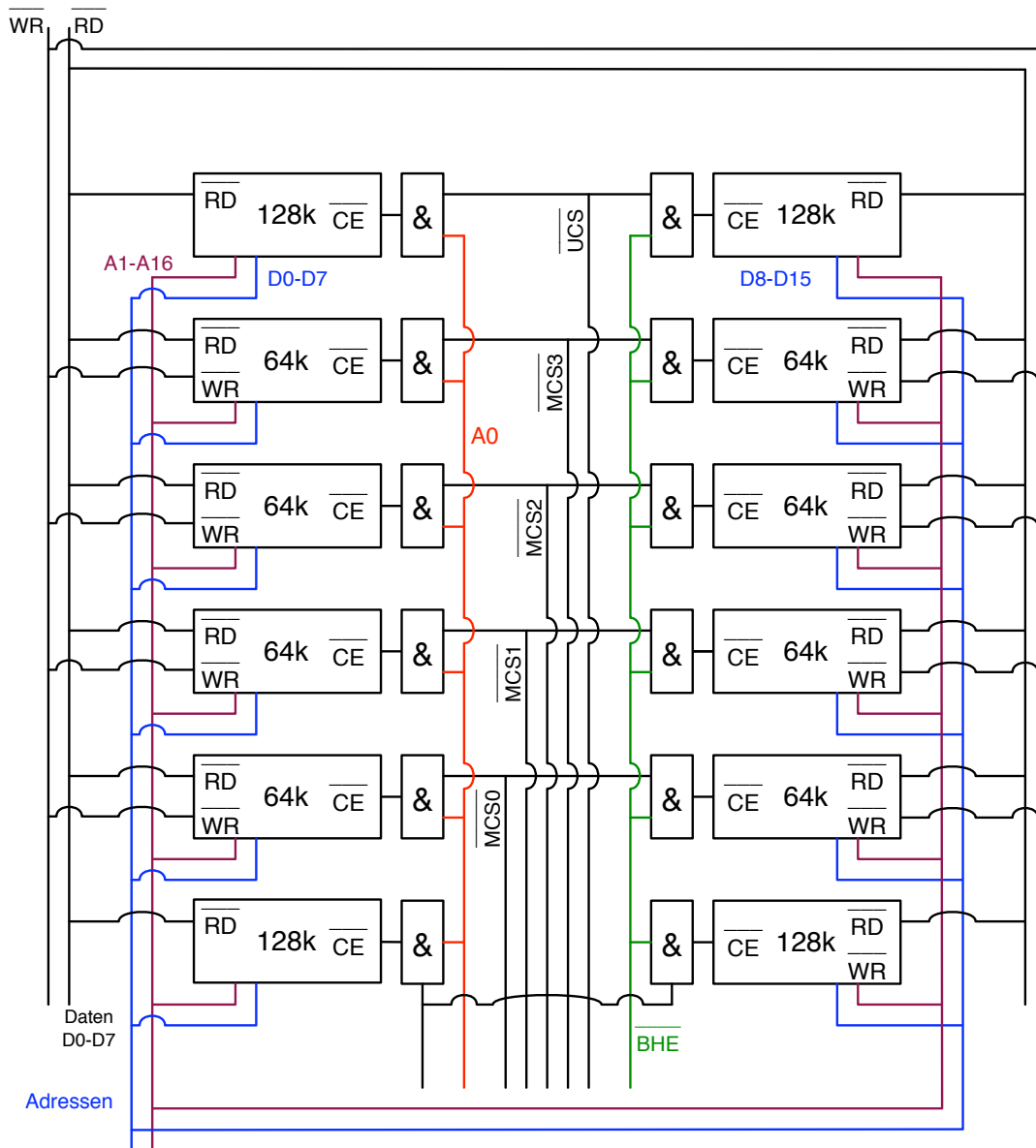
### 3.8.5 Upper Memory



Verschaltung von CS-Leitungen und  $A_0 / \overline{BHE}$

CS-Ltg	$\overline{BHE}$	$A_0$	Funktion
L	L	L	16 Bit Zugriff
L	L	H	8 Bit ungerade (obere)
L	H	L	8 Bit gerade (untere)
L	H	H	gesperrt
H	L	L	gesperrt
H	L	H	gesperrt
H	H	L	gesperrt
H	H	H	gesperrt

Nur wenn CS-Leitung und  $\overline{BHE}/A_0$  auf Masse liegen  $\Rightarrow$  Speicherzugriff.

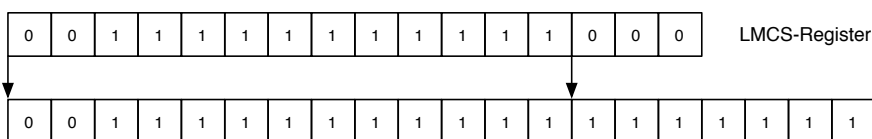


### 3.8.6 Beispiel: Fortsetzung

#### Programmierung der Register

unterer Speicherbereich (Lower Memory)

$\Rightarrow \overline{LCS} \Rightarrow LMCS$

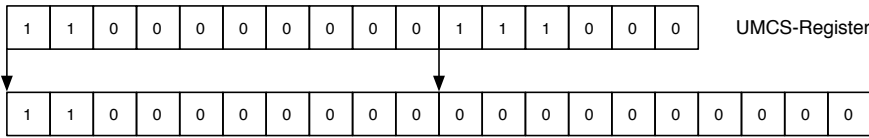


20 Bit obere Adresse des unteren Speicherbereichs

hex: 3FF8 (LMCS Register), 00000 (Startadresse), 3FFFF (obere Adresse)

oberer Speicherbereich (Upper Memory)

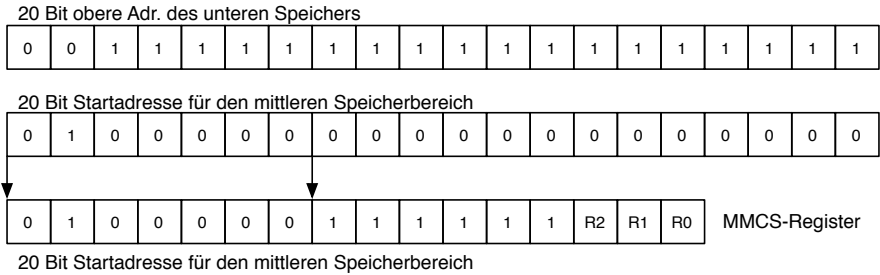
$\Rightarrow \overline{UCS} \Rightarrow UMCS$



20 Bit Startadresse des oberen Speicherbereichs

hex: C038 (UMCS Register), C0000 (Startadresse), FFFFF (obere Adresse)

mittlerer Speicherbereich (Mid Memory)



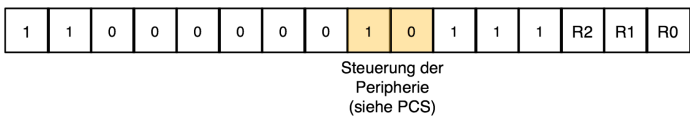
20 Bit Startadresse für den mittleren Speicherbereich

hex: 41F8 (MMCS Register), 40000 (Startadresse) , 40000 + 7FFF = BFFFF (obere Adresse)

MPCS Register

$\Rightarrow$  Blocklänge des mittleren Speicherbereiches

- 512 kB (insgesamt)



hex: C0B8 h

- obere Adresse des mittleren Speicherraums  
 Startadresse + Blocklänge = obere Adresse  
 $40000h + 7FFFh = BFFFFh$
- (20 Bit Startadresse des oberen Speicherbereiches: C0000h > BFFFFh  $\Rightarrow$  es gibt keine Adresskonflikte)

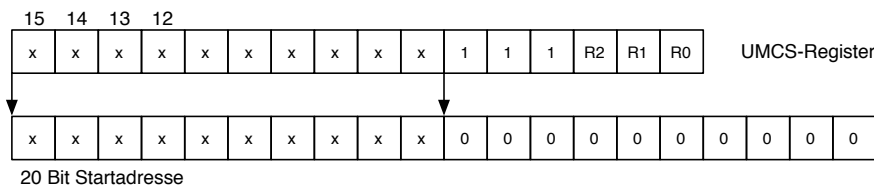
### 3.9 Die CS-Leistungen und die Peripherie

- 7 CS-Leitungen für Peripherie ( $\overline{PCS0} : 6$ )
- MPCS-Register stellt zusätzliche Steuerfunktionalitäten zur Verfügung
- CS-Leitungen unterteilen 7 zusammenhängende Speicherblöcke (nicht RAM)
- Jeder der Blöcke umfasst 128 Byte
- Prinzipiell kann der Speicherblock im I/O Bereich bzw. im RAM liegen

#### 3.9.1 PACS-Register

- PACS-definiert die Startadresse (Base) des Peripherie-Speicherblocks
- Kleinste Adresse, die erlaubt ist, ist 1024 (1k-Grenze)

	Active Range	
	Startadresse	Endadresse
$\overline{PCS0}$	Base	Base + 127
$\overline{PCS1}$	B+128	B+256
$\overline{PCS2}$	B+256	B+384
$\overline{PCS3}$	B+384	B+512
$\overline{PCS4}$	B+512	B+640
$\overline{PCS5}$	B+640	B+768
$\overline{PCS6}$	B+768	B+895



- niederwertige Bits der Startadresse sind zu 0 definiert
- Besonderheit: Der I/O Bereich beim 80186 System wird nur mit 16 Bit adressiert.  
⇒ Bit 12 bis Bit 15 sind zu 0 definiert.
- $R_0$  bis  $R_2$  ⇒ Ready Bits zur Festlegung von Wait States für die Leitungen  $\overline{PCS0} : \overline{PCS3}$

### 3.9.2 Die PCS-Leitungen und das MPCS Register

- Für Peripherie-CS-Leitungen relevante Bits im MPCS-Register:
  - Ex
  - MS
  - R0:R2
- Ready Bits zur Festlegung von Wait States für die Leitungen  $\overline{PCS4}$  bis  $\overline{PCS6}$
- Ex-Bit legt die Funktion von  $\overline{PCS5}$  und  $\overline{PCS6}$  fest.
  - wenn  $Ex = 0$  ⇒  $\overline{PCS5}$  und  $\overline{PCS6}$  stehen nicht mehr als CS-Leitungen zur Verfügung
  - ⇒ diese beiden Leitungen übertragen Adressinformationen.  
 $\overline{PCS5} \rightarrow A1$   
 $\overline{PCS6} \rightarrow A2$
  - CS-Leitungen:  $\overline{PCS0} : \overline{PCS4}$
- $\overline{PCS5} : \overline{PCS6}$  Können direkt mit den Adresseingängen der Peripherie verbunden  
⇒ Möglichkeit zur Auswahl von Betriebsfunktionen der Peripherie
- $Ex=1$  ⇒ Alle CS-Leitungen stehen zur Auswahl von Peripheriekomponenten zur Verfügung
- MS-Bit legt fest ob die Peripherie im Main Memory oder im I/O Bereich liegt
  - $MS = 0$  ⇒ Peripherie liegt im I/O-Bereich
  - ⇒ PCS-Leitungen geben Komponenten frei
  - $MS = 1$  ⇒ Peripherie Adressbereich liegt im Main Memory (RAM)
  - ⇒ Freigabe der Komponenten über LCS, UCS,  $\overline{MCS0} : \overline{MCS3}$

- $\Rightarrow$  PCS-Leitungen bedeutungslos
- Ready-Bits ( $R_0 : R_2$ )  
 (wenn Peripherie im I/O-Bereich liegt  $\Rightarrow MS = 0$ )  $\Rightarrow$  legt die Wait States für die Komponenten fest, die über  $\overline{PCS4} : \overline{PCS6}$  angesprochen werden.

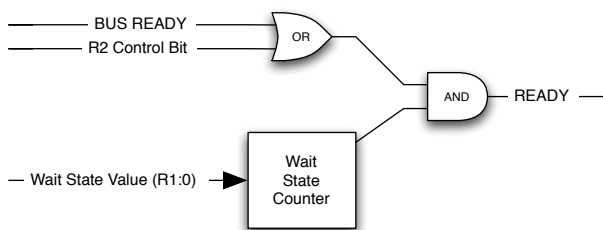
### 3.9.3 Ready Control

- Speicherbausteine / Peripheriekomponenten mit längerer Zugriffszeit
- Zugriff innerhalb eines Buszyklus (4 Takte) nicht möglich
- $\Rightarrow$  Einfügen zusätzlicher "Wartetakte"  
 $\Rightarrow$  Wait States
- Möglichkeiten zur Ready-Generierung:
  - Extern (ARDY + SRDY - Eingänge am  $\mu P$ )
  - Intern (Programmierung der Ready Bits in den Control Registern der CS Logik)

#### Programmieren von Wait States

- Möglichkeit eine bestehende Anzahl von WS für eine Komponente zu definieren
- $\Rightarrow$  Ready-Bits ( $R_2:R_0$ ) im Control Register der zugehörigen CS-Leitung  
 $\Rightarrow R_1:R_0$  legen die Anzahl der Wait States fest  
 $\Rightarrow R_2$  bestimmt, ob zusätzlich externe Ready Signale verarbeitet oder ignoriert werden
  - $R_2 = 0 \Rightarrow$  externes RDY-Signal wird verarbeitet
  - $R_2 = 1 \Rightarrow$  externes RDY-Signal wird ignoriert

### 3.9.4 Ready-Control (Grundsätzliches)



R2	R1	R0	no. of wait states
0	0	0	0 + no of exist. wait states
0	0	1	1 + no of exist. wait states
0	1	0	2 + no of exist. wait states
0	1	1	3 + no of exist. wait states
1	0	0	0
1	0	1	1
1	1	0	2
1	1	1	3

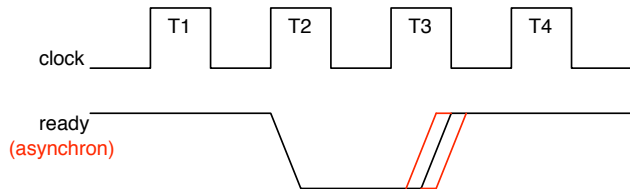
### 3.9.5 External Ready Control

- Komponente liefert ein Signal an den  $\mu P$ , das die Gültigkeit der Daten signalisiert  
 $\Rightarrow$  READY-Signal (high active)
- Buszyklus besteht aus 4 Takten  
 $\Rightarrow$  Auswirkung des Ready Signals:

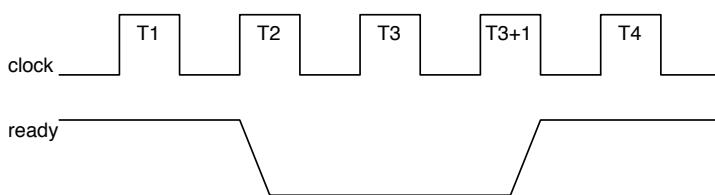
- liegt der RDY-Eingang zur Zeit des Flankenanstiegs von Takt 3 auf low, so wird der Folgetakt wiederholt,

⇒ Komponente bestimmt die Anzahl von Takten für einen Buszyklus

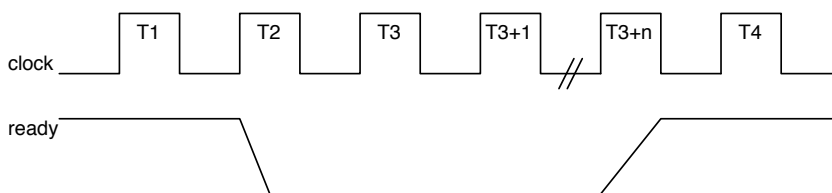
#### ohne wait states



#### mit 1 wait state



#### mit n wait state



## Buszyklus

**Takt 1** Adressen werden an die Komponenten gelegt

**Takt 2** Beginn Datenaustausch

⇒ Komponente zieht das RDY-Signal auf Low

⇒ wenn Einschwingvorgänge in den Komponenten abgeschlossen sind das RDY auf High Pegel gesetzt.

**Takt 3 a)** Ready-Signal mit ansteigender Flanke von Takt 3 wieder auf High Pegel ⇒ kein Wait State

**b)** RDY-Signal mit ansteigender Flanke von Takt 3 noch auf low

⇒ Einfügen von Wait-States

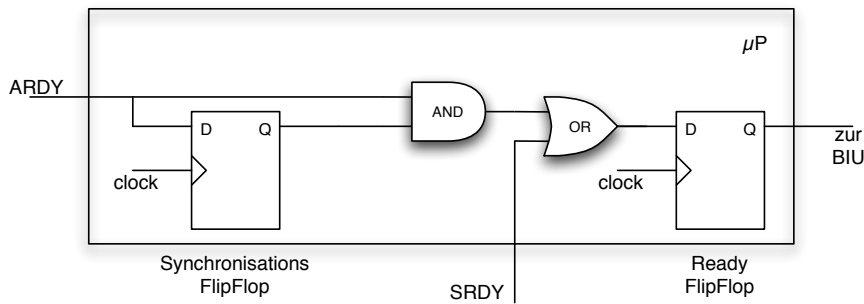
⇒ Ist mit ansteigender Flanke des Folgetaktes das RDY-Signal immer noch low, so wird ein weiterer Wait State eingefügt.

⇒ es können beliebig viele Wait States eingefügt werden

## Integration

- 2 RDY-Eingänge beim 80186
  - ⇒ A RDY ⇒ asynchroner Eingang
  - ⇒ S RDY ⇒ synchroner Eingang
- Da das RDY Signal auf die ansteigende Flanke ausgewertet wird, kommt der Synchronisation eine übergeordnete Rolle zu.

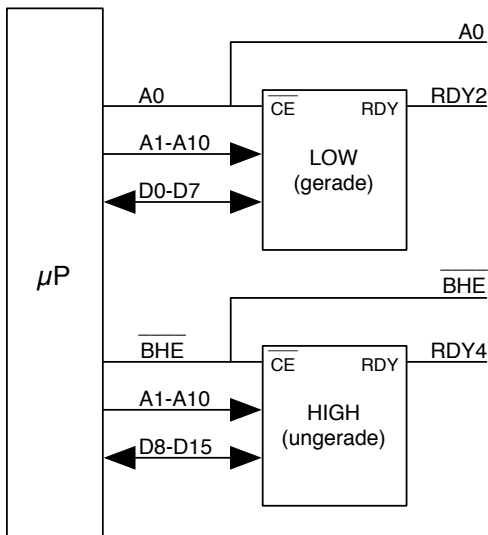
### 3.9.6 Anschluß - Ready Control



- Synchronisations FlipFlop synchronisiert Ready-Signal auf den Takt (auf ansteigende Flanke)
- Ready FlipFlop vergleicht Ready Zustand mit ansteigender Flanke des Taktes
- Ready Logic ist direkt mit BIU verbunden.
- ausschließlich 1 Eingang kontaktiert  
 ⇒ SRDY für bereits synchronisierte Signale  
 ⇒ ARDY für asynchrone
- nicht benutzer Eingang ist mit Masse zu verbinden.

### 3.9.7 Beispiel: Ready-Anschluss einer asynchronen Speicherbank

#### 1. Schritt Analyse



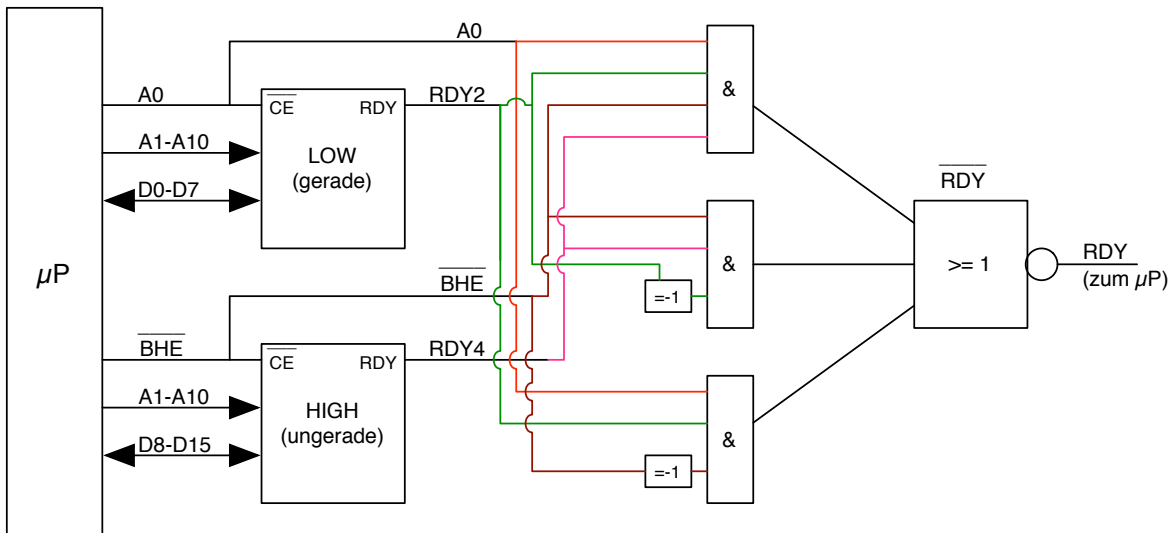
2. Schritt Wahrheitstabelle

	$\overline{BHE}$	A0	$\overline{RDY_2}$	$\overline{RDY_4}$	$\overline{RDY}$	
kein Zugriff	0	0	0	0	0	unrelevant
	0	0	0	1	0	
	0	0	1	0	0	
	0	0	1	1	0	
8 Bit gerade	0	1	0	0	0	unrelevant
	0	1	0	1	0	
	0	1	1	0	1	
	0	1	1	1	1	
8 Bit ungerade	1	0	0	0	0	unrelevant
	1	0	0	1	1	
	1	0	1	0	0	
	1	0	1	1	1	
16 Bit	1	1	0	0	0	Alle Steuerleitungen relevant!
	1	1	0	1	0	
	1	1	1	0	0	
	1	1	1	1	1	

3. Schritt resultierende Gleichung

$$RDY = (A_0 \wedge RDY_2 \wedge \overline{BHE}) \vee (\overline{BHE} \wedge \overline{A_0} \wedge RDY_4) \vee (\overline{BHE} \wedge A_0 \wedge \overline{RDY_2} \wedge \overline{RDY_4})$$

4. Schritt Schaltung basierend auf Gleichungen



(Low-Pegel == logisch "1")

3.10 Direct Memory Access (DMA)

- In vielen Fällen werden große Datenblöcke zwischen Speicherkomponenten (z.B. zwischen I/O und RAM) ausgetauscht.
- würde die CPU den Datentransfer abwickeln, so wäre die Performance drastisch eingeschränkt.
  - ⇒ DMA erlaubt den Datenaustausch zweier Komponenten *ohne* Steuerung durch die CPU
  - ⇒ DMA-Controller übernimmt die Steuerung des D-A-Busses: RD, WR,  $\overline{BHE}$ , A0, CS-Leitungen

- Überblick DMA-Transfer (grundsätzlich)
  - ⇒ DMA beginnt mit einem DMA-Request (DRQ)
    - \* von Datenquelle angefordert (source synchronised)
    - \* von Datensenke angefordert (destination synchronised)
  - ⇒ 3 Möglichkeiten zum starten eines DMA
    1. extern (DRQ-Leitung)
    2. durch Timer ausgelöst
    3. Programmieren der zugehörigen Control-Register

### 3.10.1 Ablauf DMA

#### 1.) DRQ

- DRQ ist auf die fallende Flanke des CPU Taktes gesampelt
- 4 Takte sind nötig bevor der DMA-Controller die BIU steuern kann

#### 2.) DMA-Zyklus

- besteht immer aus 2 Buszyklen (8 Takte)
  1. Buszyklus: fetch-cycle (Zugriffszyklus = Lesevorgang )
    - ⇒ Daten werden von Quelle eingelesen
  2. Buszyklus: deposite-cycle (Ablegezyklus ⇒ Schreibvorgang)
    - ⇒ Daten werden zum Ziel geschrieben
- Lese-Schreibvorgänge laufen wie gewohnt ab:
  - T1: Übernahme der Adresse
  - T3/4: Daten werden übernommen (BIU)  
Steuerleitungen ALE, RD, WR,  $\overline{BHE}$ ,  $A_0$ , CS-Leitungen

#### 3.) DMA-Folge

- Source synchronised oder destination synchronised
  - a) source synchronised
    - DRQ low vor fallender Flanke von T4 des fetch cycles ⇒ keine DMA-Folge.
    - DRQ high bei fallender Flanke von T4 des fetch cycles ⇒ neuer DMA-Zyklus nach T4 des deposite cycles.
  - b) destination synchronised
    - ⇒ Unterschied: Nach Beendigung eines DMA-Zyklusses werden immer 2 Takte (Idle States) eingefügt.
    - ⇒ entsprechend wird das DRQ-Signal um 2 Takte verschoben auf T2 des deposite cycles ausgewertet.
      - DRQ low vor fallender Flanke von T4 des fetch cycles ⇒ keine DMA-Folge.
      - DRQ high bei fallender Flanke von T4 des fetch cycles ⇒ neuer DMA-Zyklus nach T4 des deposite cycles.

### 3.10.2 Mögliche DMA-Übertragungsrichtungen

- vom Hauptspeicher nach I/O
- von I/O nach Hauptspeicher
- innerhalb des Hauptspeichers
- innerhalb des I/O

(Möglichkeit zur Übetragung von 8- oder 16-Bit Worten)

### 3.10.3 Die DMA-Einheit des 80186

- Es stehen 2 voneinander unabhängige DMA-Kanäle zur Verfügung ( $CH_0, CH_1$ )  $\Rightarrow$  2 externe DRQ-Eingänge für DMA-Anforderungen
- jeder Kanal hat eigenständige Control-Register zur Steuerung der DMA-Abläufe ( $\Rightarrow$  PCB)

#### Eigenschaften

- zur Adressierung von Datenquelle und Datensenke stehen 2 x 20 Bit Register zur Verfügung (Source- bzw. Destination Pointer)
- Möglichkeit nach Ablauf eines DMA-Zyklusses Source- und Destination Pointer zu inkrementieren bzw. dekrementieren.
- Anzahl der DMA-Zyklen kann im Transfercount Register (TC) festgelegt werden. (Nach Ablauf eines DMA-Zyklus wird dieses Register um 1 dekrementiert)
- Nach Beendigung eines DMA Zyklus kann ein IR ausgelöst werden (IR-Vektornummer 10/11)

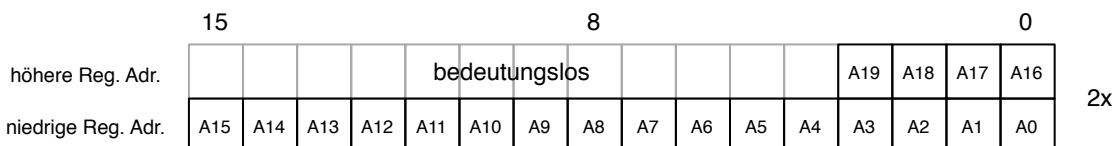
#### Arten eines DRQ

1. Extern: DRQ-Leitung (source synch./dest. synch.)
  2. Timer 2: wenn max. Zählwert erreicht (option)  
 $\Rightarrow$  Vor Request müssen alle DMA-Control-Register programmiert sein
  3. von DMA-Einheit selbst: durch Setzen bestimmter Bits im Control Register des DMA-Kanals  
 $\Rightarrow \text{Syn} = 0/0$   
 $\Rightarrow \text{ST}/\text{STOP} = 1$
- vor Beginn des DMA-Transfers müssen TC, SA, DA festgelegt werden, da der DMA-Transfer unmittelbar nach Programmierung des Control-Registers gestartet wird.

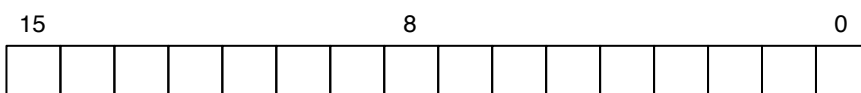
### 3.11 Register DMA-Einheit (80186)

(für einen Kanal) Organisiert in 16 Bit

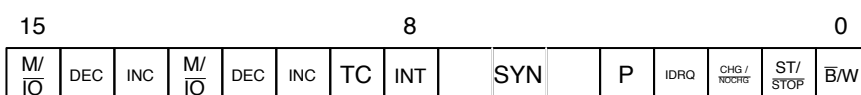
Source-Pointer (Dest. Pointer analog!):



16 Bit Transfer-Count  $\Rightarrow$  max. Counter-Value:  $2^{16} = 65536$



16 Bt Control-Register



### 3.11.1 Offsetadressen im PCB

	DMA 0	DMA 1
Control	CAh	DAh
Transfer-Count	C8h	D8h
Source - höhere Reg. Adr.	C2h	D2h
Source - niedrige Reg. Adr.	C0h	D0h
Dest. - höhere Reg. Adr.	C6h	D6h
Dest. - niedrige Reg. Adr.	C4h	D4h

### 3.11.2 Die DMA-Register

- Wie die Control Register aller anderen On-Chip Peripherals des 80186 sind die DMA-Register im PCB angesiedelt.

#### Source-/Destination Pointer

- Angabe von Quell- bzw. Zieladresse des DMA (20-Bit)
- PCB (im RAM) ist wortweise aufgebaut (16-Bit / 8-Bit High/Low)
  - niedrige Registeradresse: Bits 0:15 repräsentieren die Adressbits  $A_0 : A_{15}$
  - höhere Registeradresse: Bits 0:3 repräsentieren die Adressbits  $A_{16} : A_{19}$ . Die Bits 4-15 der höheren Registeradresse sind bedeutungslos.

#### Transfercount

- legt die Anzahl (Größe) eines DMA-Transfers fest.
- wird nach jedem DMA-Zyklus dekrementiert
- je nach Betriebsart wird der DMA-Transfer mit Erreichen eines TC-Wertes von 0 fortgesetzt oder beendet (=j TC-Bit im Control-Register)
- Max. Zählwert: 65536

#### Das Control-Register

- legt Betriebsart des zugehörigen DMA Kanals fest
- kann während eines laufenden DMA-Transfers umprogrammiert werden, was allerdings dessen Parameter verändert.

### 3.11.3 Programmierung des DMA-Control-Register

⇒ setzen der zugehörigen Steuerbits des Registers

$\overline{B}/W$  Beschreibung:

- legt fest, ob bei einem DMA-Transfer Bytes oder Words (16 Bit) ausgetauscht werden.
- $\overline{B}/W = 0 \Rightarrow$  Byte-Transfer
- $\overline{B}/W = 1 \Rightarrow$  Word-Transfer

$ST/\overline{STOP}$  Beschreibung:

- $ST/\overline{STOP} = 0 \Rightarrow$  DMA-Transfer kann nicht gestartet werden (auch bei "externen" DRQ's)
- $ST/\overline{STOP} = 1 \Rightarrow$  DMA-Kanal ist für DMA-Transfer bereit.

$CHG/\overline{NCHG}$  Beschreibung:

- bezieht sich auf  $ST/\overline{STOP}$
- Bit kann nur beschrieben werden
- wird das Control Register gelesen, liefert dieses Bit immer 0.
- wird zur Initialisierung des Control Registers benötigt.
- $CHG/\overline{NCHG} = 0 \Rightarrow$  augenblicklicher Wert von  $ST/\overline{STOP}$  wird bei der Initialisierung übernommen
- $CHG/\overline{NCHG} = 1 \Rightarrow$  augenblicklicher Wert von  $ST/\overline{STOP}$  wird bei der Initialisierung mit dem Initialisierungswert überschrieben.

$IDRQ$  Beschreibung:

- Timer 2 DMA Request
- wird benutzt um DMA-Request von T2 zuzulassen, falls dieser den im Max-Zählerregister (T2) programmierten Wert erreicht hat.
- $IDRQ = 1 \Rightarrow$  T2-DMA-Request wird verarbeitet
- $IDRQ = 0 \Rightarrow$  T2-DMA-Request wird ignoriert

$P$  Beschreibung:

- Priority-Bit
- Repräsentiert Priorität gegenüber eines möglichen DMA-Transfers des anderen DMA-Kanals
- Hat einer der beiden DMA-Kanäle Priorität gegenüber dem Anderen, so werden dessen DMA-Transfers vorher ausgeführt.
- Bei gleicher Priorität werden die DMA-Zyklen der beiden DMA-Kanäle abwechselnd abgearbeitet
- $P = 1 \Rightarrow$  hohe Priorität
- $P = 0 \Rightarrow$  niedrige Priorität

DMA-CH <sub>0</sub>	DMA-CH <sub>1</sub>	Abarbeitung DMA-Zyklen
0	0	abwechselnd
0	1	DMA-Zyklus von CH <sub>1</sub> vor denen von CH <sub>0</sub>
1	0	DMA-Zyklus von CH <sub>0</sub> vor denen von CH <sub>1</sub>
1	1	abwechselnd

$SYN$  Beschreibung:

- 2 Bit (6,7)
- gibt an, ob der DMA-Transfer Source- oder Destination synchronised ist
- $SYN$  0/0
  - unsynchronisierter DMA (DRQ  $\Rightarrow$  "intern")
  - DRQ erfolgt mit Programmieren des Control-Registers
  - TC-Bit hat keine Bedeutung
  - DMA-Transfer wird immer beendet, wenn das TC-Register den Wert 0 hat.
- $SYN$  0/1
  - Source-synchronised (DRQ-Leitung wurde von der Datenquelle gesetzt)
- $SYN$  1/0
  - Destination-synchronised (DRQ-Leitung wurde von der Datensenke gesetzt)
- $SYN$  1/1 nicht definiert.

$INT$  Beschreibung:

- wird benutzt um Programmunterbrechungen mit Ablauf des TC-Registers (TC = 0) zuzulassen oder nicht.

- $INT = 0 \Rightarrow$  keine Programmunterbrechungen zugelassen
- $INT = 1 \Rightarrow$  Interrupt, wenn das TC-Register den Wert 0 aufweist.

*TC* Beschreibung:

- bestimmt über die Beendigung eines DMA-Transfers mit Ablauf des TC-Registers (TC-Register = 0)
- $TC = 0 \Rightarrow$  DMA-Transfer wird fortgesetzt unabhängig ob das TC-Register den Wert 0 aufweist oder eben nicht.
- $TC = 1 \Rightarrow$  DMA-Transfer wird beendet, wenn das TC-Register den Wert 0 aufweist.

*INC* Beschreibung:

- legt fest, ob nach Beendigung eines DMA-Zyklus Source- bzw. Destination Pointer um 1 inkrementiert wird oder nicht.
- $INC = 1 \Rightarrow$  nach Ablauf eines DMA-Zyklus wird Source- bzw. Destination Pointer um 1 inkrementiert.
- $INC = 0 \Rightarrow$  bedeutungslos

*DEC* Beschreibung:

- legt fest, ob nach Beendigung eines DMA-Zyklus Source- bzw. Destination Pointer um 1 dekrementiert wird oder nicht.
- $DEC = 1 \Rightarrow$  nach Ablauf eines DMA-Zyklus wird Source- bzw. Destination Pointer um 1 dekrementiert.
- $DEC = 0 \Rightarrow$  bedeutungslos

- **Zusammenhang zwischen  $INC/DEC$  und  $\overline{B}/W$**

- $\overline{B}/W = 0 \Rightarrow$  Byte Transfer  
 $\Rightarrow$  Source- bzw. Destination Pointer wird um 1 inkrementiert bzw. dekrementiert.
- $\overline{B}/W = 1 \Rightarrow$  Word Transfer (16 Bit)  
 $\Rightarrow$  Source- bzw. Destination Pointer wird um 2 inkrementiert bzw. dekrementiert.

- **Möglichkeit**

$INC = DEC = 1$

- Wert für Source- bzw. Destination Pointer bleibt konstant

$M/\overline{IO}$  Beschreibung:

- Zeigt an, ob Datenquelle bzw. Datensenke im Hauptspeicher (RAM) oder im I/O-Bereich angesiedelt ist.
- $M/\overline{IO} = 0 \Rightarrow$  Source- bzw. Destination befindet sich im I/O.
- $M/\overline{IO} = 1 \Rightarrow$  Source- bzw. Destination befindet sich im RAM.

### Bemerkung

Die Bits  $INC$ ,  $DEC$ ,  $M/\overline{IO}$  gibt es im Control-Wort 2x

- Bits 10:12 beziehen sich auf die Datensenke
- Bits 13:15 beziehen sich auf die Datenquelle

## 3.12 Interrupt-Control

### Allgemein

- Mikroprozessoren stehen in Kommunikation zu extern angeschlossenen Funktionseinheiten (Peripherie/-Geräte)
- Diese Peripherie benötigt Anteil am Processing der CPU zu unterschiedlichen teils nicht vorhersagbaren Zeitpunkten
- Zwei Möglichkeiten
  1. Polling
  2. Interrupts

### 3.12.1 Polling

- CPU fragt angeschlossene Geräte periodisch ab, ob eine Anforderung vorliegt
  - ⇒ Folglich muss dafür CPU-Performance zur Verfügung gestellt werden.
  - ⇒ Verringerung des Systemdurchsatzes

### 3.12.2 Interrupts

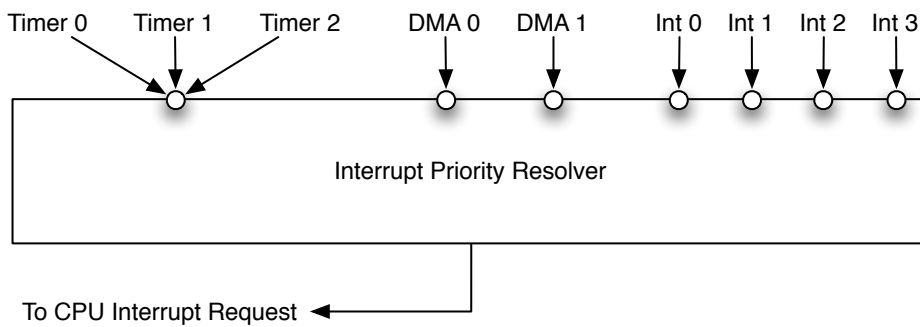
- Angeschlossene Einheiten signalisieren ihre Anforderungen an die CPU
  - ⇒ keine Abfrage der angeschlossenen Einheiten notwendig.
- CPU unterbricht das aktuelle Programm.
- Zwischenspeichern der zugehörigen Operanden und Adressen.
- Abarbeitung der mit der Interrupt-Quelle verbundenen Sequenz.
- Nach Beendigung der IR-Sequenz wird das zuvor unterbrochene Programm fortgesetzt.

### 3.12.3 Interrupt-Quellen (80186)

- interne IR-Quellen: Timer (3x), DMA (2x)
- externe IR-Quellen: INT 0:3 ; NMI (Non-Maskable Interrupt / Anschluss von Peripherie)

### Interrupt-Masking

- Es gibt Umstände, die es erfordern, einzelne oder alle IR-Eingänge zu sperren. Diesen Vorgang nennt man IR-Masking.
- Maskierung wird über das Programmieren der zugehörigen Register im PCB vorgenommen
- IR, die über den NMI-Eingang angefordert werden, haben höchste Priorität und setzen sich direkt durch (Non-Maskable IR)
- ⇒ globale Maskierung aller IR-Eingänge INT 0:3 erfolgt durch Setzen des IR-Enable Flags (IF) im Prozessor Status Wort.



### Interrupt-Prioritäten

Interrupt Name	Relative Priority
Timer 0	0 (a)
Timer 1	0 (b)
Timer 2	0 (c)
DMA 0	1
DMA 1	2
INT 0	3
INT 1	4
INT 2	5
INT 3	6

Tabelle 3.1: Default Interrupt Priorities

#### 3.12.4 IR-Prioritäten

- Jeder IR-Quelle wird eine Priorität zugewiesen  
 ⇒ Reihenfolge der Abarbeitung der IR  
 ⇒ es ist möglich, dass ein IR von einem weiteren IR höherer Priorität unterbrochen wird (IR-Nesting)
- Klassifizierung der IR-Prioritäten in 8 Stufen:  
 ⇒ Prioritäten = 0 ⇒ höchste Priorität ⇒ Prioritäten = 7 ⇒ niedrigste Priorität
- Priorität nach Initialisierung ⇒ siehe Folie
- Timer teilen sich einen gemeinsamen IR-Eingang  
 ⇒ relative Priorität ⇒ Timer 0 mit höchster Priorität und Timer 2 mit niedrigster Priorität.
- Priorität für jede IR-Quelle ist programmierbar
- erlaubte Prioritäten liegen zwischen 0 und 6
- IR-Quellen dürfen gleiche Priorität haben

#### Priority Mask Register

- Maskiert alle IR-Quellen, deren Priorität niedriger ist, als die spezifizierte Priorität im Priority-Mask-Register. (Nach der Initialisierung hat das Priorität-Mask-Register den Wert 7 ⇒ Alle IR-Quellen sind unmaskiert).

### 3.12.5 Interrupt Nesting

- einfachste Möglichkeit  $\Rightarrow$  ohne IR-Nesting
  - $\Rightarrow$  IR werden sequenziell verarbeitet unabhängig von ihrer Priorität. (in der Reihenfolge wie die zugehörigen IRQ (Interrupt Request) aufgetreten sind.
  - $\Rightarrow$  ggf. große Wartezeit für IRQ hoher Priorität, da die vorherigen IRQ vorher abgearbeitet werden müssen.
  - $\Rightarrow$  Unterbrechung von IR-Service Routinen (ISR) für IRQ höherer Priorität nötig (IR-Nesting)
- Hier existieren 2 Regeln:
  1. Ein IRQ kann keine ISR unterbrechen, deren Priorität höher ist
  2. Ein IRQ kann seine eigene ISR nicht unterbrechen

### 3.12.6 Ablauf eines IR (grundsätzlich)

1. IRQ von einer bestimmten Quelle (Komponente)
  2. Entfüllt der IRQ die Prioritätsbedingung?
  3. Der aktuell abzuarbeitende Befehl wird zu Ende gebracht
  4. Das Control-Register des  $\mu P$  wird auf den Stack "gerettet". Das Code-Segment-Register und der Instruction Pointer werden ebenfalls auf dem Stack zwischengespeichert (Rückkehradresse in das unterbrochene Programm)
  5. Aus der Kennzahl, die mit der IR-Quelle fest verknüpft ist ergibt sich die Adresse in der IR-Vektortabelle, in der die Startadresse für die zugehörige ISR gespeichert ist.
  6. ISR wird verarbeitet
  7. Control Register sowie CS-Register und IP werden vom Stack geladen
    - $\Rightarrow$  Unterbrochenes Programm wird fortgesetzt.
- In beschriebener Weise werden alle unterbrochenen Programme (auch unterbrochene ISR) auf den Stack zwischengespeichert, während die höchstrangige ISR abgearbeitet wird.
  - Nach Beendigung einer ISR werden die unterbrochenen Programme (ISR) in umgekehrter Reihenfolge zu Ende verarbeitet.

Problematik: Dimensionierung des Stack ( $\Rightarrow$  Stack overflow)

### 3.12.7 Aufbau der IR-Vektortabelle

- IR-Vektortabelle beinhaltet die Startadressen der ISR aller IR-Quellen
- Startadresse repräsentiert durch den Wert von CS-Register und IP  
[ $CS + IP \Rightarrow 32\text{Bit} \Rightarrow 4\text{Byte}$ ]
- $\Rightarrow$  daher Multiplikation der Kennzahl mit 4. Es werden immer 4 Byte zum Repräsentieren der Startadresse benötigt.
- IR-Vektortabelle befindet sich im Lower Chip Memory des RAM und beginnt per Definition mit der Adresse 0h

Vektornummer-Kennzahl	Vektortadr. physik.	Interrupt Quelle
0	0	Divide Error ( $\mu P$ selbst)
1	4	Single Step (Debugging)
2	8	NMI
3	12	Breakpoint IR
4	16	IR on Overflow (Stack)
5	20	Array Bounds Check
6	24	unused
7	28	ESC Opcode
8	32	Timer 0
9	36	Timer 1
10	40	Timer 2
11	44	DMA 0
12	48	DMA 1
13	52	INT 0
14	56	INT 1
15	60	INT 2
16	64	INT 3

### IR-Vektortabelle (Master-Mode / keine Kaskade)

#### Die IR-Vektortabelle im RAM

IRV 3	CS-ISR 3	00016h
	IP-ISR 3	00014h
IRV 2	CS-ISR 2	00012h
	IP-ISR 2	00010h
IRV 1	CS-ISR 1	00008h
	IP-ISR 1	00006h
IRV 0	CS-ISR 0	00004h
	IP-ISR 0	00002h
		00000h

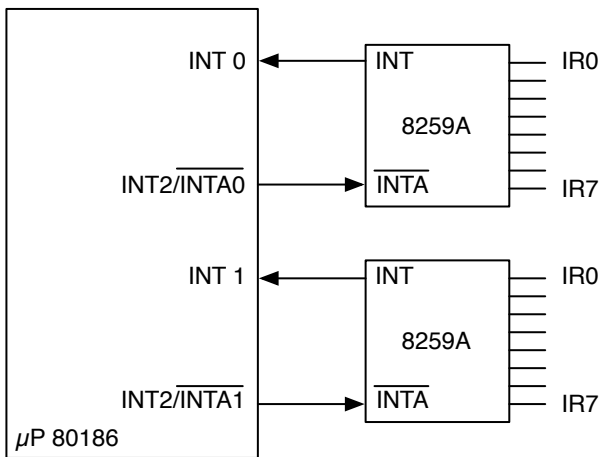
Zusammenhang: Interrupt  $\Rightarrow$  IR-Kennzahl (\*4)  $\Rightarrow$  Adr. IR-Vektor in Tabelle  $\Rightarrow$  Startadresse der ISR

### 3.12.8 IR-Kaskadierung im Master Mode

Grund: 4 externe Anschlüsse (INT 0:3) reichen nicht aus

$\Rightarrow$  Möglichkeit max. zwei externe IR-Controller mit jeweils 8 weiteren IR-Eingängen mit dem  $\mu P$  zu verschalten. (Intel 8259A)

$\Rightarrow$  16 IR-Quellen können angeschlossen werden



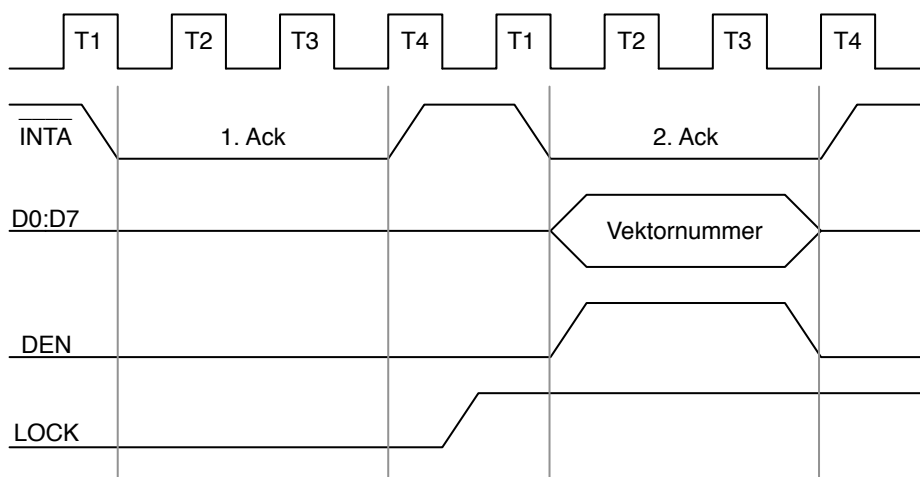
- Im Cascade-Mode sind nur die Pins INT 0 und INT 1 Interrupt Eingänge
- Die Pins INT 2 und INT 3 dienen als IR-Quittierungssignale für die Pins INT 0 und INT 1

### 3.12.9 Ablauf eines IR im Kaskade-Mode

Problematik:

- Zuordnung einer IR-Kennzahl zum externen Eingang am  $\mu P$  nicht mehr möglich
- 16 mögliche IR Quellen teilen sich 2 Eingänge am  $\mu P$
- $\Rightarrow$  Der physikalische IR-Eingang am  $\mu P$  kann zur Identifizierung der IR-Quelle nicht mehr dienen.
- $\Rightarrow$  Lösung Nachdem der  $\mu P$  den IRQ quittiert hat, wird die Kennzahl von dem entsprechenden IR-Controller über den Datenbus angefordert.

### 3.12.10 Anfordern der IR-Vektornummer

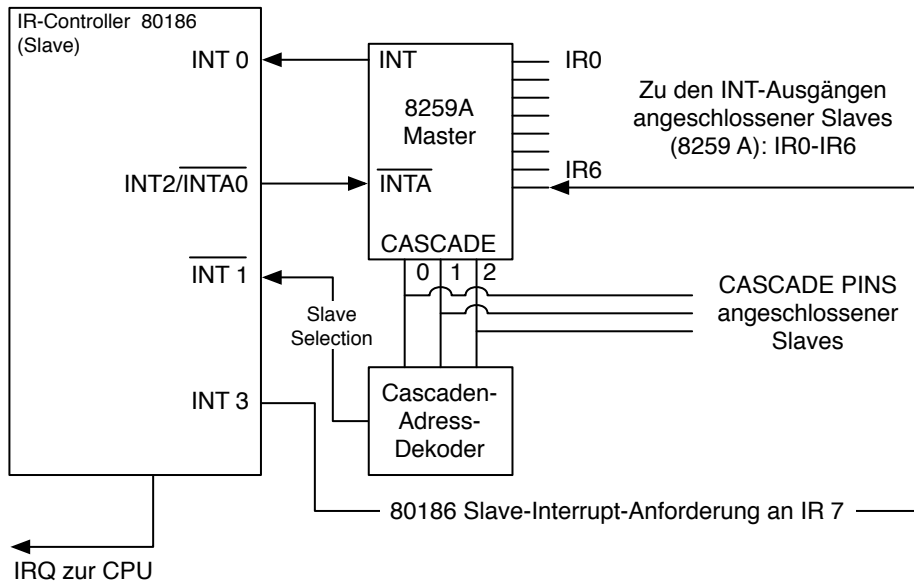


1. Ack quittiert den IRQ
  2. Ack: 8259 A legt die entsprechende Vektornummer auf den Datenbus
- Lock sperrt den Datenbus für andere Benutzer

- die Abarbeitung auftretender IR bleibt grundsätzlich gleich.

- Unterschied: Bevor der nächste Befehl des unterbrochenen Programms und der Prozessorstatus auf den Stack "gerettet" wird, wird die Kennzahl des IR-Eingangs über den Datenbus (D0:7) übertragen.

### 3.13 Der 80186 IR-Controller im Slave-Mode

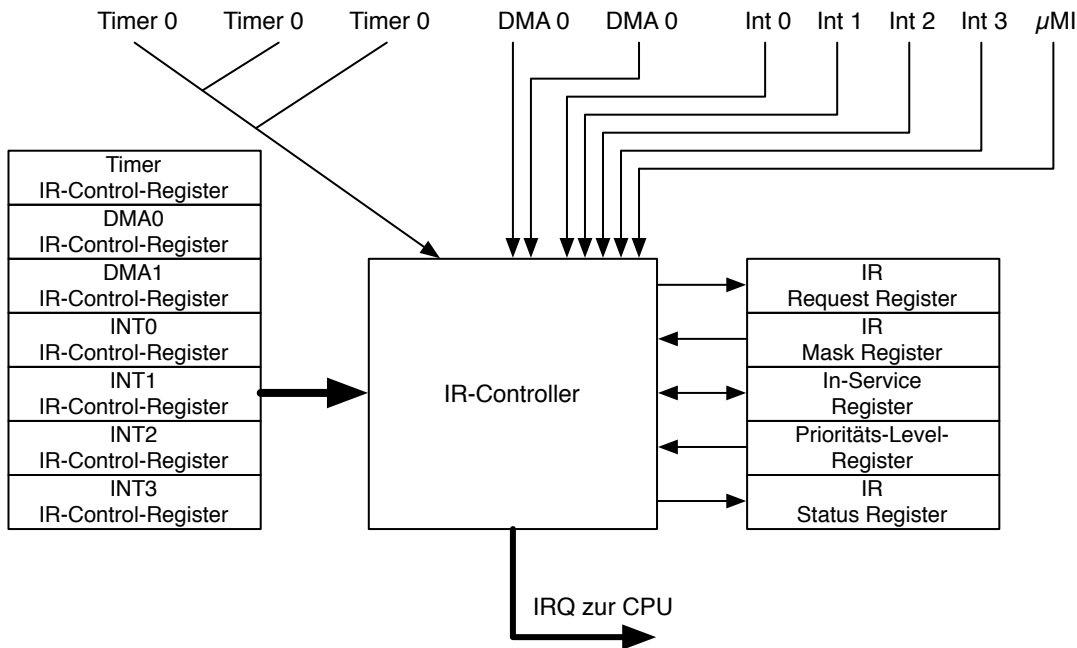


- Möglichkeit, den externen IR-Controller (8259A) als Master zu programmieren.
  - alle IRQ werden vom externen Controller abgewickelt
  - IRQs des  $\mu P$  müssen ebenfalls über den externen Controller angefordert werden
- Möglichkeit, an die einzelnen Eingänge des Master Controllers jeweils einen weiteren externen IR-Controller (Slave Mode) anzuschließen
  - 1 Anschluss am Master-Controller muss für den  $\mu P$  bereitgestellt werden
  - 7 frei Anschlüsse für weitere Slave-Controller mit jeweils 8 Interrupt-Anschlüssen  $\Rightarrow$  maximale Anzahl von IR-Quellen: 56

#### 3.13.1 Ablauf

- Ein am Master auftretender IRQ wird an den  $\mu P$  über den Eingang INTO weitergeleitet.
- $\mu P$  quittiert über den Eingang  $\overline{INT2}/\overline{INTA}$
- Master setzt die Cascade Leitungen entsprechend der IR-Quelle mit dem 2. IR-Acknowledge (IACK)
- IR-Quelle (ext. angeschlossener IR-Controller im Slave Betrieb) liefert die Vektor-Kennzahl über den Datenbus (andere Slave-Controller sind inaktiv, da die Adressen auf den Cascade-Leitungen nicht zutreffen)
- IR wird verarbeitet (Kennzahl  $\Rightarrow$  Adresse in der IR-Vektortabelle  $\Rightarrow$  Startadresse der entsprechenden ISR  $\Rightarrow$  ISR wird abgearbeitet)
- Da der  $\mu P$  keine Eingänge zum Anschluss der Cascade-Leitungen hat, muss hierfür ein zusätzlicher Adressdekodierer vorgeschaltet werden.
- Bei der für den  $\mu P$  zugehörigen Cascade-Adresse liefert der Adressdecoder den INTA an den  $\mu P$  ( $\overline{INT1}$ )

### 3.13.2 Übersicht über den 80186-IR-Controller und dessen Konfiguration



Jede IR-Quelle hat ein zugehöriges IR-Control-Register zur Konfiguration und zur Wiedergabe des IR-Status gibt es 5 weitere Register (PCB)

#### Aufgaben der einzelnen IR-Control-Register

- Die Control-Register der internen IR Quellen (Timer / DMA)
  - 3 Bit zur festlegung der Priorität der IR-Quellen
  - 1 Bit zur Maskierung der IR-Quelle
- Die Control-Register der externen IR-Quelle
  - 3 Bit zur Festlegung der Priorität der zugehörigen IR-Quelle
  - 1 Bit zur Maskierung der IR-Quelle
  - LTM-Bit: Legt fest, ob der IR Pegel- oder Flankengesteuert ist.
- zusätzliche Bits in den Registern für die Quellen NT0/NT2 (benötigt bei Cascade-Mode)
  - ⇒ C-Bit indiziert Cascade-Mode
  - ⇒ SFNM-Bit
    - gelöscht ⇒ Es wird immer nur ein IR für jeden externen IR-Controller zugelassen (gleichzeitig)
    - gesetzt ⇒ Mehrere IR eines externen IR-Controllers zugelassen
    - → Special Fully Nested Mode

#### Control-Register zur Konfiguration und Zustandserfassung des IR-Controllers

##### 1. IR-Request Register

- Jeder IR-Quelle ist ein Bit zugeordnet
- ⇒ fordert eine IR-Quelle einen IR an, wird dies an der zugehörige Position im Register durch Setzen des Bits gekennzeichnet

2. IR-Mask-Register

- Jede Quelle ist durch ein bestimmtes Bit repräsentiert
- ⇒ Möglichkeit einzelne IR-Quellen zu sperren (maskieren)
- 

3. IR-Service-Register

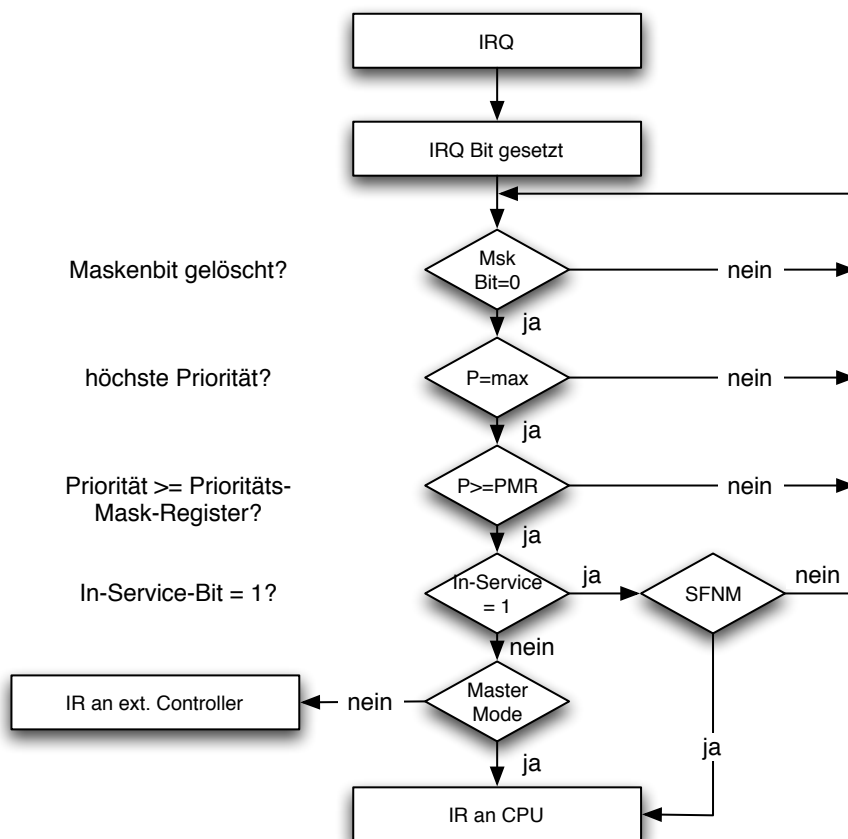
- Jede Quelle ist durch ein bestimmtes Bit repräsentiert
- Ist ein Bit gesetzt, so wird die ISR der zugehörigen Quelle gerade verarbeitet

4. Priority Level Register

- legt eine Priorität fest (Grenzpriorität) oberhalb welcher IRs zugelassen werden (R niedrigerer Priorität gesperrt)

5. IR Status Register

- für interne IR-Quellen
- Spezifiziert den Timer bei Timer Interrupts
- DHLT (DMA-HALT) ⇒ Stoppt den DMA bei NMI



### 3.14 Programmierung der PCB Register

- PCB befindet sich im I/O-Bereich (I/O-Bereich mit 16 Bit reserviert)
- ⇒ PCB-Register werden durch indirekte Adressierung angesprochen

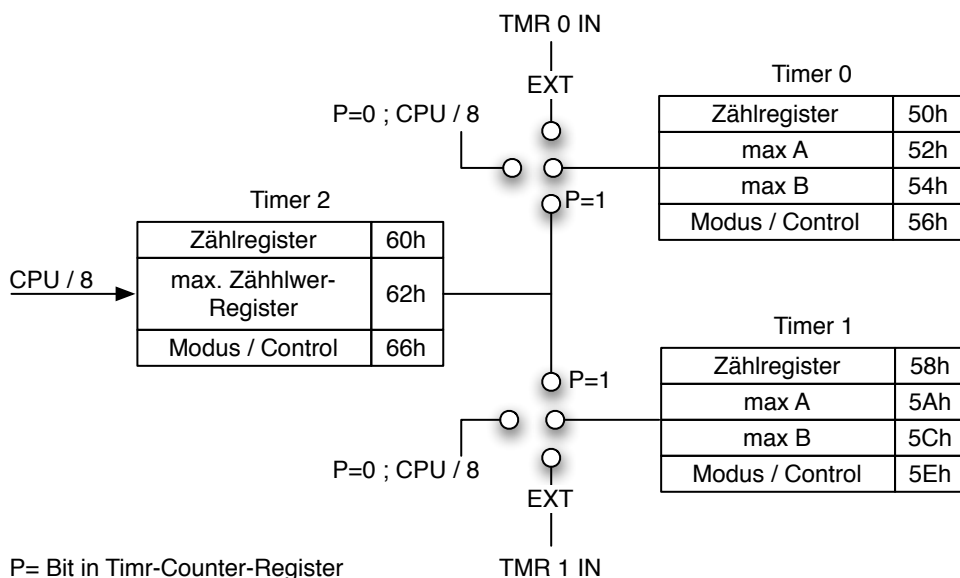
- ⇒ 2 Register (80186) nötig
  - Register, das den zu programmierenden Wert enthält (Ax) ⇒ Akku
  - Register, das die Zieladresse im PCB enthält (Dx)
- ⇒ Assembler-Befehle nötig
  - Schiebepfehl, der die entsprechenden Werte in die Register des  $\mu P$  (Ax, Dx) schreibt (MOV)
  - Befehl, der die Speicherzelle im PCB, die durch das Dx-Register adressiert ist, mit dem Wert aus dem Ax-Register beschreibt (OUT)

### 3.14.1 Programmierung

1. Schreibe den zu programmierenden Wert (x) in das Ax-Register:  
MOV AX, x
2. Schreibe die Adresse (y) der zu beschreibenden Speicherzelle im PCB nach DX:  
MOV DX, y
3. Schreibe den Wert aus Ax (x) an die Stelle aus Dx (y):  
OUT DX, AX

## 3.15 Die TIMER/COUNTER Einheit des 80186

- 3 Timer
- grundsätzlich zählen alle Timer ausschließlich definierte Ereignisse
- “serienschialtung” von Timern möglich
  - $T_2 \rightarrow T_0$
  - $T_2 \rightarrow T_1$
- zur Kontrolle stehen jedem Timer mehrere Register im PCB zur Verfügung
  1. Zählregister: enthält den aktuellen Zählwert
  2. Max. Zählregister: spezifiziert den maximalen Zählwert
  3. Modus/Control-Register: Steuerregister des Timers



- Eingänge der Timer
  - $T_2$  hat als Eingangssignal ausschließlich  $\frac{1}{8}$  CPU-Takt
  - $T_1/T_0$  können unterschiedliche Eingänge haben:
    1. Ausgang von  $T_2$  (immer dann, wenn  $T_2$  den maximalen Zählwert erreicht)
    2.  $\frac{1}{8}$  CPU-Takt
    3. externe Events
  - $\Rightarrow$  Programmierbar im Modus/Control-Register von  $T_1/T_0$  (P, EXT)

### 3.15.1 Die Modus Control Register der Zählereinheit (INTEL 80186)

T2 Modus Control

EN	INH	INT	0	0	0	0	0	0	0	0	MC	0	0	0	0	CONT
----	-----	-----	---	---	---	---	---	---	---	---	----	---	---	---	---	------

T0/T1 Modus Control

EN	INH	INT	RIU	0	0	0	0	0	0	0	MC	RTG	P	EXT	ALT	CONT
----	-----	-----	-----	---	---	---	---	---	---	---	----	-----	---	-----	-----	------

EN	Enable	Aktivieren (Starten des Timers)
INH	Inhibit	Im Zusammenhang zu EN: EN-Bit kann nur gesetzt werden, wenn zugleich auch das INH Bit gesetzt ist. Kann nur gesetzt werden; Beim Auslesen des Registers immer 0
INT	Interrupt	INT=1 $\Rightarrow$ IR wird ausgelöst, wenn der im Max-Zählwertregister definierte Wert erreicht ist.
RIU	Register In Use	Indiziert, welches der beiden Max. Zählwertregister beim aktuellen Zyklus verglichen wird: - RIU = 0 $\Rightarrow$ Max. Zählwertregister A - RIU = 1 $\Rightarrow$ Max. Zählwertregister B
MC	Max. Count	Indiziert, wenn der Max. Zählwert des entsprechenden Registers erreicht wurde. Dieses Bit darf ausschließlich von der Timer Einheit selbst gesetzt werden (Beim Programmieren: MC=0)
RTG	Retrigger	RTG=1 $\Rightarrow$ Aktueller Zählwert wird zurückgesetzt RTG=0 $\Rightarrow$ Fortsetzen mit dem aktuellen Zählwert (Bei EXT=1 hat dieses Bit keine Bedeutung)
P	Prescaler	P=0 $\Rightarrow$ T0 bzw. T1 Eingang auf $\frac{1}{8}$ CPU-Takt P=1 $\Rightarrow$ T0 bzw. T1 Eingang auf Eventausgang von T2 (Bei EXT=1 hat dieses Bit keine Bedeutung)
EXT	External	EXT=1 T0 bzw. T1 zählt externe Ereignisse (externer Eingang)
ALT	Alternate	Single Compare Mode vs. Dual Compare Mode ALT=0 $\Rightarrow$ Der aktuelle Zählwert wird ausschließlich mit Max. Zählregister A vergleichen. ALT=1 $\Rightarrow$ Der aktuelle Zählwert wird abwechselnd mit dem Max. Zählregister A und Max. Zählregister B verglichen (nach Ablauf von jedem Zyklus)
CONT	Continuous Mode	Entscheidet über das Fortfahren der Timer-Einheit, wenn der aktuelle Zählwert den Wert des entsprechenden Max. Zählwertregisters erreicht hat: CONT=0 $\Rightarrow$ Timer hält an, wenn Max. Zählwert erreicht ist CONT=1 $\Rightarrow$ Zähler läuft kontinuierlich durch. (Beginnt mit Erreichen des entsprechenden Max. Zählwertes mit Zählwert 0)

### 3.15.2 Anwendungsbeispiele Timer

#### 1. WATCHDOG-Timer

- überwacht den Fortgang eines Programmblocks.
- Der Programmblock darf eine definierte Zeitspanne nicht überschreiten, ansonsten erfolgt ein IR durch den Watchdog-Timer.
- Damit ist ausgeschlossen, dass das gesamte System durch eine Routine blockiert ist.
- Der Watchdog muss vom Programm konfiguriert, gestartet und gestoppt werden
- Wird der Timer nicht gestoppt, bevor er den maximalen Zählwert erreicht erfolgt der zugehörige IR

#### Beispiel

Beispiel: Ein Teil einer Routine darf max. 50ms dauern. Der Prozessortakt beträgt 8 MHz.  $T_2$  soll als "Watchdog" konfiguriert werden.

$$\frac{f}{8} = 1MHz \Rightarrow T = \frac{1}{f} = 10^{-6}s$$

$$\text{max. Zählwert: } \frac{50ms \cdot 10^{-3}s}{10^{-6}s} = 50.000$$

#### Programmierung der Register

##### Control-Register $T_2$

EN	INH	INT									MC				CONT			
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

- $T_2$  muss vor Beginn der zu überwachenden Befehlsfolge konfiguriert und gestartet werden
- Max. Zählwert (50.000) muss in das zugehörige Register geschrieben werden

#### 2. Anwendungsbeispiel

- Eine LED soll durch die Verschaltung von  $T_2$  und  $T_0$  im vorgegebenen Takt leuchten.



- Prozessortakt ist 2 MHz.
- $\Rightarrow$  Lösungsansatz

–  $T_2$  teilt die Prozessorfrequenz auf 1 kHz runter.  $T_0$  zählt abwechselnd bis 1000 bzw. 1500

$$f_0 = 1kHz$$

$$f_{CPU} = 2 \cdot 10^6 Hz$$

–  $\Rightarrow$  max. Zählwert von  $T_2 = \frac{f}{8 \cdot f_0} = \frac{2 \cdot 10^6 Hz}{8 \cdot 10^3 Hz} = 250 = FA_h$

–  $\Rightarrow$  max. Zählwert von  $T_0 : A = 1000 = 3E8_h, B = 1500 = 5DC_h$

–  $T_2$ Control

EN	INH	INT									MC				CONT			
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	

$C001_h$

–  $T_0$ Control

EN	INH	INT	RIU									MC	RTG	P	EXT	ALT	CONT
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

$C00B_h$

– Offset-Adressen der Register (PCB)

$T_2$ -Control:  $66_h$

$T_2$ -Compare:  $62_h$

$T_0$ -Control:  $56_h$

$T_0$ -Compare A:  $52_h$

$T_0$ -Compare B:  $54_h$

$\Rightarrow$  Startadresse des PCB (Relocation) liegt bei  $100_h$ .

- $T_2$  Max. Zähler = 250  
MOV Ax, FA<sub>h</sub>  
MOV Dx, 162<sub>h</sub>  
OUT Dx, Ax
- $T_2$  Control  
MOV Ax, C001<sub>h</sub>  
MOV Dx, 166<sub>h</sub>  
OUT Dx, Ax
- $T_0$  Compare A  
MOV Ax, 3E8<sub>h</sub>  
MOV Dx, 152<sub>h</sub>  
OUT Dx, Ax
- $T_0$  Compare B  
MOV Ax, 5DC<sub>h</sub>  
MOV Dx, 154<sub>h</sub>  
OUT Dx, Ax
- $T_0$ Control  
MOV Ax, C00B<sub>h</sub>  
MOV Dx, 156<sub>h</sub>  
OUT Dx, Ax