MC

Adressierungsarten	Register	MOV A, R0 ohne @	„Arbeitsregister" R0–R7	1Byte
	direkte	MOV P1, R0 (→ 90h)	adressiert int. DS (on-chip-RAM: 00–7F)	2Byte
	indirekte	MOVX A, @R0 (Zeiger)	Wert des Registers = Adresse (Berechnung zur Laufzeit)	1Byte
	unmittelbare	MOV A, #0Ch (Festwert)	Operand ist unmittelbar im Befehl codiert	2Byte
	indizierte	MOVC A, @ A+PC (oder DPTR)	Effektive Adresse = AKKU (Offset) + Basisregister (nur auslesen)	1Byte

Ziel/Quelle PS: A+DPTR, A+PC, #Festwert   ED: MOVX @R1, A   ID: sonst

Sichern push: SP inkrementieren, auf Stack sichern   pop: lesen, SP dekrementieren

Interrupt Polling: (Abfrage) CPU fragt in regelmäßigen Abständen die Peripherie (z.B. Flags) ab

Interrupt: Ereignis, unterbricht momentanen Programmablauf, ruft dem Ereignis zugeordnetes neues Programm auf
→ pro MZ werden die ext. Interrupts einmal abgefragt

Interrupt-Latenz: Zeit zw. Auftreten der Interruptbedingung + dem Ausführen der ISR ; min: 3,25µs (abhängig: Prio-Ebene in Bearbeitung, Abtasten (2S P2), Polling Sequenz, Sprung zur ISR, 1. Befehl der ISR (min. 1 Instruktion wird ausgeführt))

A/D Prinzip der sukzessiven Approximation (Wägeverfahren) [Annäherung von MSB -> LSB]

$$U_{LSB} = \frac{U_{ref}}{2^n} = \frac{5V}{2^{12}} = 1{,}22mV \qquad Z_{max} = 2^n - 1 \qquad U_{A,max} = U_{ref} \cdot \frac{U_{A,max}}{U_{A,max+1}} = Z_{max} \cdot U_{LSB}$$

Start-ADC-Taste: SCONV=1; ADCI=1, SCONV=0

Quantisierungsstufe (Genauigkeit) $\pm \frac{1}{2} \cdot U_{LSB}$ Wandlungsergebnis $\frac{x}{2^n} = \frac{U_e}{U_{ref}}$ $U_{erg} = x \cdot \frac{U_{ref}}{2^n}$

ADCDATAL: untere Bits
ADCDATAH: höhere Bits, + Kanalkennung

Harvard: Programme + Daten befinden sich in getrennten Speichern, die über getrennte Adress- + Datenbusse angesprochen werden (DSP)

Von Neumann: Programme + Daten werden über einen gemeinsamen Adress- + Datenbus angesprochen → es können entweder Befehle oder Daten geholt werden (Intel 8051)

Taktquellen d. Timer: C/$\overline{T}$=0: Intervalltimer; erzeugen von Zeitintervallen (on-chip-Oszillator)
C/$\overline{T}$=1: Ereigniszähler (für T0: P3.4, für T1: P3.5) -> 1-0-Übergang inkrementiert Zähler [Abtasten bei Z5]
Gate=1: Timer aktiviert über $\overline{INT1}$ ($\overline{INT0}$ bei T0)
$\overline{INT1}$: ext. Eingang -> Timer läuft, wenn =0

$\overline{EA}$: (Externer Zugriff); Freigabe des ext. PS; $\overline{EA}$=1 -> intern, $\overline{EA}$=0 -> extern
RxD (Seriell-Parallel), TxD (Parallel-Seriell-Wandlung) Serielle Schnittstelle } Port3
$\overline{INT0}$, $\overline{INT1}$ Interrupt-Eingang   T0, T1 Zählereingabe
$\overline{RD}$ Lesefreigabe $\overline{WR}$ Schreibfreigabe Ansteuerung d. ext DS (RAM)
Port2: Adress (high)   Port0: Adress (low) / Daten   Port1: 3. Timer

16 Adressleitungen, falls ext. Speicher benutzt wird;   8 Datenleitungen, falls ext. Speicher benutzt wird

movx: erkennbar am fehlenden ALE-Signal
ALE (Adress Latch Enable); Freigabe des Adresszwischenspeichers
$\overline{PSEN}$ (Programmspeicherfreigabe) =0: Programmwort wird geholt; Ansteuerung des ext PS (ROM)
-> unterdrückt bei movx-Befehl [2.1] (=1)

UART Universal Asynchronous Receiver / Transmitter: kann gleichzeitig senden + empfangen
1 Startbit + 8 Datenbits + 1 Stoppbit   Baudrate $f_B = \frac{1}{\tau_B}$

asynchron: nicht kontinuierliche Wandlung -> Synchronisation über Start- + Stoppbit

Verzögerung

```
_Schleife0:  push 00              ; Inhalt von R0 auf Stack
             mov R0, #idelay0     ; Zählwert in R0
_Loop1    :  call Schleife1       ; UP für 500µs
             djnz R0, _Loop1      ; dekrementiere & zähle bis R0 = 0
             pop 00               ; Inhalt von R0 vom Stack
             ret                  ; Ende der Verzögerungsschleife

_Schleife1:  mov R2, #idelay1     ; Zählwert in R2 laden
_Loop     :  djnz R2, _Loop       ; dekrementiere & zähle bis R2=0
             ret                  ; Ende der Unterschleife
```

LCD-Controller

```
pLCD_Cont  equ  0FF100h ; Steuerdaten schreiben
pLCD_Data  equ  0FF101h ; Anzeigedaten schreiben
```

LCD-Ausgabe

Positionierung für Ausgabe

```
mov DPTR, #pLCD_Cont   ; DPTR mit Adresse Display-Steuerregister laden
mov A, #80h            ; AKKU mit 1/1-LCD-Display laden
movx @DPTR, A          ; AKKU nach Display-Steuerregister
call _Delay_K          ; kurze Zeitschleife aufrufen
```

Ausgabe auf Display

```
mov DPTR, #pLCD_Data   ; Datenpointer mit LCD-Daten-Adresse laden
call _Delay_K
mov A, #'H'            ; Zeichen "H"
movx @DPTR, A          ; Zeichen auf Display schreiben
call _Delay_K
```

```
String:
dB 'Text' ; Text
dB 00h    ; Endkennung
```

Stringausgabe

```
mov A, #80h
call _Set_Pos
mov DPTR, #String
call _LCD_Out
```

```
_LCD_Out:  push 01h
           mov R1, #pLCD_L      ; Low-Byte Datenpointer in R1
           mov P2, #pLCD_H      ; High-Byte Adressbus laden
_Label     clr A                ; Offset = 0
           movc A, @A + DPTR    ; Zeichen von String laden
           inc DPTR             ; auf nächstes Zeichen setzen
           jz _Fertig           ; bis Endkennung (00h)
           movx @R1, A          ; AKKU-Ausgabe auf Display
           call _Delay_K
           jmp _Label
           pop 01h
           ret
```

AD-Ausgabe

```
setb EA              ; Interrupts global freigeben
setb EADC            ; AD Interrupt freischalten

mov ADCCON1, #60h    ; Zeitverhalten des AD-Wandlers ; CLK = 1/4 CPU
mov ADCCON2, #00h    ; Kanal 0 als Quelle auswählen

clr ADFlag           ; Ausgabeflag löschen
setb SCONV           ; AD-Wandlung starten

mov pKanal_H, ADCDATAH ; aktuellen Wert der Wandlung abholen
mov pKanal_L, ADCDATAL
setb ADFlag          ; Ausgabeflag setzen
```

Maschinenzyklusfrequenz $f = \frac{1}{12} f_{osc}$ Pulsbreite $= \frac{12}{f_{osc}}$ $f_{max} = \frac{1}{6} f_{osc}$

Daten vom ext. DS, Adresse 0x0008h -> ACC : mov DPTR, #08h movx A, @DPTR

T0 Priorität erhöhen: setb PT0 ^ mov IP, #02h

Relative -, Absolute Adressierung, Absolute mit langer Adresse

Verzögerungszeit: $(7 + 2 \cdot idelay1) \cdot idelay0 + 7$ (_Loop, _Loop1)