
# RONPAS-Compiler-Handbuch

Autor: Ronald Daleske

Stand: 24.10.2022

# Inhaltsverzeichnis

# RONPAS-Compiler - Sprachreferenz und Zusatzbefehle

## Schlüsselworte/Befehle nach Namen sortiert

| Bezeichnung | Beschreibung | Befehls-Gruppe |
|---|---|---|
| **BitHigh** | Abfrage, ob das angegebene Bit den Wert 1 hat | **Bit-Test, -Setz und Rücksetz-Befehle** |
| **BitLow** | Abfrage, ob das angegebene Bit den Wert 0 hat | **Bit-Test, -Setz und Rücksetz-Befehle** |
| **ByteToChar** | Konvertierung von Byte in Char | **Funktionen zur Typ-Konvertierung** |
| **ByteToDWord** | Konvertierung eines Bytes in ein DWord | **Funktionen zur Typ-Konvertierung** |
| **ByteToWord** | Konvertierung eines Bytes in ein DWord | **Funktionen zur Typ-Konvertierung** |
| **CharToByte** | Konvertierung von Char in Byte | **Funktionen zur Typ-Konvertierung** |
| **ClearBit** | Setzt das angegebene Bit auf 0 | **Bit-Test, -Setz und Rücksetz-Befehle** |
| **CONST** | Beginn der Definition der Konstante(n). | **PASCAL-Sprachreferenz** |
| **DEC** | Dekrementieren des Variablenwertes um 1 | **Elementare Operationen** |
| **DisableInts** | Sperren von Interrupts | **System-Befehle** |
| **DWordToByte** | Kopieren eines Bytes aus einem DWord | **Funktionen zur Typ-Konvertierung** |
| **DWHigh** | Kopieren des höherwertigen Words aus einem DWord | **Funktionen zur Typ-Konvertierung** |
| **DWLow** | Kopieren des niederwertigen Words aus einem DWord | **Funktionen zur Typ-Konvertierung** |
| **EnableInts** | Erlauben von Interrupts | **System-Befehle** |
| **END** | Ende von: PROGRAM, PROCEDURE oder FUNCTION | **PASCAL-Sprachreferenz** |
| **High** | Kopieren des höherwertigen Bytes aus einem Word | **Funktionen zur Typ-Konvertierung** |
| **IF THEN** | IF-THEN-(ELSE)-Auswahl | **PASCAL-Sprachreferenz** |
| **INC** | Inkrementieren des Variablenwertes um 1 | **Elementare Operationen** |
| **Length** | Ermittlung der Länge einer String | **STRING-Befehle** |
| **Low** | Kopieren des niederwertigen Bytes aus einem Word | **Funktionen zur Typ-Konvertierung** |
| **NOP** | NOP - No Operation - 1 Takt Assembler "Füllbefehl" tut nichts | **Elementare Operationen** |
| **PROGRAM** | kennzeichnet den Beginn eines PASCAL-Programm-Quelltextes | **PASCAL-Sprachreferenz** |
| **SetBit** | Setzt das angegebene Bit auf 1 | **Bit-Test, -Setz und Rücksetz-Befehle** |
| **SHL** | Shift Left - Linksverschieben des Variablenwertes um ein Bit | **Elementare Operationen** |
| **SHR** | Shift Right - Rechtserschieben des Variablenwertes um ein Bit | **Elementare Operationen** |
| **WordToDWord** | Konvertierung ein Word in ein DWord | **Funktionen zur Typ-Konvertierung** |

## Befehle nach Gruppen sortiert

### PASCAL-Sprachreferenz

| Bezeichnung | Beschreibung |
|---|---|
| **IF THEN** | IF-THEN-(ELSE)-Auswahl |
| **CONST** | Beginn der Definition der Konstante(n). |
| **PROGRAM** | PROGRAM kennzeichnet den Beginn eines PASCAL-Programm-Quelltextes. |

## Bit-Test, -Setz und Rücksetz-Befehle

Die Bit-Befehle ermöglichen das direkte Ansprechen einzelner PINs (Ein-, Ausgabe und Programmverzweigung).

| Bezeichnung | Beschreibung |
|---|---|
| **BitHigh** | Abfrage, ob das angegebene Bit den Wert 1 hat |
| **BitLow** | Abfrage, ob das angegebene Bit den Wert 0 hat |
| **SetBit** | Setzt das angegebene Bit auf 1 |
| **ClearBit** | Setzt das angegebene Bit auf 0 |

## Elementare Operationen

Die Elementaren Operationen werden mit einem oder sehr wenigen Assemblerbefehle umgesetzt und sind daher sehr schnell.

| Bezeichnung | Beschreibung |
|---|---|
| **DEC** | Dekrementieren des Variablenwertes um 1 |
| **INC** | Inkrementieren des Variablenwertes um 1 |
| **NOP** | NOP - No Operation - 1 Takt Assembler "Füllbefehl" tut nichts |
| **SHL** | Shift Left - Linksverschieben des Variablenwertes um ein Bit |
| **SHR** | Shift Right - Rechtserschieben des Variablenwertes um ein Bit |

## STRING-Befehle

Bearbeiten von Zeichenketten.

| Bezeichnung | Beschreibung |
|---|---|
| **Length** | Ermittlung der Länge einer String |

## Funktionen zur Typ-Konvertierung

Einer der großen Vorteile der Programmiersprache PASCAL ist die starke Typisierung (strong typing). Dadurch werden viele Fehler, die bei der Entwicklung des Quelltextes gemacht werden, schon beim Compilieren erkannt.

Ist es erforderlich, dass während der Abarbeitung des Programms Daten (von einer Konstante oder Variable) in eine andere Variablen eines anderen Typs kopiert werden, so sind dafür eine **Funktionen zur Typ-Konvertierung** erforderlich.

Mit der Auswahl der richtigen Funktion kann gezielt auf die Besonderheiten der einzelnen Typen eingegangen werden.

| Bezeichnung | Quell-Typ | Ziel-Typ |
|---|---|---|
| **ByteToChar** | Byte | Char |
| **ByteToWord** | Byte | Word |

| | | |
|---|---|---|
| **ByteToDWord** | Byte | DWord |
| **CharToByte** | Char | Byte |
| **High** | Word | Byte |
| **Low** | Word | Byte |
| **WordToDWord** | Word | DWord |
| **DWordToByte** | DWord | Word |
| **DWHigh** | DWord | Word |
| **DWLow** | DWord | Word |

## System-Befehle

Die System-Befehle sind spezielle Befehle zur Steuerung der Hardware des Microcontrollers. Meist werden hier Assembler-Befehle gekapselt.

| Bezeichnung | Beschreibung |
|---|---|
| **DisableInts** | Sperren von Interrupts |
| **EnableInts** | Erlauben von Interrupts |

# BitHigh

Die Funktion **BitHigh** testet, ob das angegebene Bit den Binärwert 1 hat. Wenn ja, ist das Ergebnis TRUE. BitHigh wird ausschließlich in Abfragen genutzt um den Zustand eines Bits auszuwerten.

## Syntax

**BitHigh ( < das_Bit > ) ;**

## Parameter

das_Bit = Der Parameter muss als Bit im Vereinbarungsteil vorher vereinbart worden sein (z.B. **Eingang[PinC,0] : Bit;**)

## Beispiel

```
1   PROGRAM test_b2w;
2
3   DEVICE = mega8;
4
5   VAR
6     PORTC, DDRC, PinC : ByteConReg;
7
8     Eingang[PinC,0] : Bit;
9     Ausgang[PORTC,1] : Bit;
10
11  BEGIN
12    DDRC:=%11111110;
13
14    if BitHigh(Eingang) then
15      ClearBit(Ausgang);
16    else
17      SetBit(Ausgang);
18    endif;
19
20
21  END test_b2w.
```

Das Beispiel liest das Bit 0 des Registers C (als "Eingang" definiert) ein und testet, ob es den Binärert 1 (High) hat.

Wenn ja, wird das Bit 1 des Registers C (als "Ausgang" definiert) gelöscht. Hat das Bit 0 des Registers C den Binärwert 0, so wird die else-Schleife der if-Abfrage angesprungen und das Bit "Ausgang" wird gesetzt.

## siehe auch

**BitLow**, **ClearBit**, **SetBit**

# BitLow

Die Funktion **BitLow** testet, ob das angegebene Bit den Binärwert 0 hat. Wenn ja, ist das Ergebnis TRUE. BitLow wird ausschließlich in Abfragen genutzt um den Zustand eines Bits auszuwerten.

## Syntax

**BitLow ( < das_Bit > ) ;**

## Parameter

das_Bit = Der Parameter muss als Bit im Vereinbarungsteil vorher vereinbart worden sein (z.B. **Eingang[PinC,0] : Bit;**)

## Beispiel

```
1   PROGRAM test_b2w;
2
3   DEVICE = mega8;
4
5   VAR
6     PORTC, DDRC, PinC : ByteConReg;
7
8     Eingang[PinC,0] : Bit;
9     Ausgang[PORTC,1] : Bit;
10
11  BEGIN
12    DDRC:=%11111110;
13
14    if BitLow(Eingang) then
15      ClearBit(Ausgang);
16    else
17      SetBit(Ausgang);
18    endif;
19
20  END test_b2w.
21
```

Das Beispiel liest das Bit 0 des Registers C (als "Eingang" definiert) ein und testet, ob es den Binärert 0 (Low) hat.

Wenn ja, wird das Bit 1 des Registers C (als "Ausgang" definiert) gelöscht. Hat das Bit 0 des Registers C den Binärwert 1, so wird die else-Schleife der if-Abfrage angesprungen und das Bit "Ausgang" wird gesetzt.

## siehe auch

**BitHigh**, **ClearBit**, **SetBit**

# ByteToChar

Die Funktion **ByteToChar** ist eine reine Typkonvertierung. Das Byte wird in ein Char umgewandelt.

## Syntax

**< Variable vom Typ Char > := ByteToChar ( < Variable oder Konstante vom Typ Byte > ) ;**

## Parameter

< Variable oder Konstante vom Typ Byte >

## Beispiel

```
1   PROGRAM test_ByteToChar;
2
3   DEVICE = mega644P;
4
5   VAR
6     b1 : Byte;
7     c1 : Char;
8
9   BEGIN
10
11    b1 := $41;
12    c1 := ByteToChar(b1);
13
14  END test_ByteToChar.
```

In Zeile 11 wird der Bytevariablen "b1" der Bytewert $41 zugewiesen.

In Zeile 12 wird der Typ Byte (mit Hilfe der Fuktion ByteToChar) in den Typ Char umgewandelt und in die Variable "c1" geschrieben. Der Bytewert $41 entspricht dem Zeichen "A".

## siehe auch

**CharToByte**

# ByteToDWord

Die Funktion **ByteToDWord** liest eine Byte-Konstante oder -Variable und kopiert diese in das niederwertigste Byte der DWord Variablen als Rückgabewert. Die restlichen 3 Byte der DWord Variablen (MSB) werden auf $00 gesetzt.

## Syntax

**< Variable von Typ DWord > := ByteToDWord ( < Name der Byte-Konstante oder -Variable > ) ;**

## Parameter

< Name der Byte-Konstante oder -Variable >

## Beispiel

```
1  PROGRAM test_b2d;
2
3  DEVICE = mega8;
4
5  VAR
6
7    b1 : Byte;
8    d1 : DWord;
9
10 BEGIN
11
12   b1:=2;
13   d1:=ByteToDWord(b1);
14
15 END test_b2d.
```

In Zeile 14 wird der Wert der Byte-Variable "b1" gelesen und in die DWord-Variable "d1" geschrieben. Der Wert der DWord-Variablen ist danach $00000002.

## siehe auch

**DWLow**, **DWHigh**, **DWordToByte**, **WordToDWord**

# ByteToWord

Die Funktion **ByteToWord** liest eine Byte-Konstante oder -Variable und kopiert diese in das niederwertige Byte der Word Variablen als Rückgabewert. Das höherwertige Byte der Word Variablen (MSB) wird auf $00 gesetzt.

## Syntax

**< Variable von Typ Word > := ByteToWord ( < Name der Byte-Konstante oder -Variable > ) ;**

## Parameter

< Name der Byte-Konstante oder -Variable >

## Beispiel

```
1  PROGRAM test_b2w;
2
3  DEVICE = mega8;
4
5  VAR
6
7    b1 : Byte;
8    w1 : Word;
9
10 BEGIN
11
12   b1:=$47;
13   w1:=ByteToWord(b1);
14
15 END test_b2w.
```

In Zeile 13 wird der Wert der Byte-Variable "b1" gelesen und in die Word-Variable "w1" geschrieben. Der Wert der Word-Variablen ist danach $0047.

## siehe auch

**DWLow**, **DWHigh**, **DWordToByte**, **WordToDWord**, **ByteToDWord**

# CharToByte

Die Funktion **CharToByte** ist eine reine Typkonvertierung. Die Variable oder Konstante Char wird in den Typ Byte umgewandelt.

## Syntax

**< Variable vom Typ Byte > := CharToByte ( < Variable oder Konstante vom Typ Char > ) ;**

## Parameter

< Variable oder Konstante vom Typ Char >

## Beispiel

```
1   PROGRAM test_CharToByte;
2
3   DEVICE = mega644P;
4
5   VAR
6     b1 : Byte;
7     c1 : Char;
8
9   BEGIN
10
11    c1 := 'C';
12    b1 := CharToByte(c1);
13
14  END test_CharToByte.
```

In Zeile 11 wird der Char-Variablen "c1" der Konstanten-Wert "C" zugewiesen.

In Zeile 12 wird durch die Funktion CharToByte der Inhalt der Char-Variablen "c1" gelesen und in die Byte-Variable "b1" geschrieben.

Das Zeichen "C" entspricht dem Bytewert $43, der jetzt in der Variablen "b1" steht.

## siehe auch

**ByteToChar**

# ClearBit

Die Prozedur **ClearBit** löscht das angegebene Bit (setzt das Bit auf den Binärwert 0).

## Syntax

**ClearBit ( < das_Bit > ) ;**

## Parameter

das_Bit = Der Parameter muss als Bit im Vereinbarungsteil vorher vereinbart worden sein (z.B. **Ausgang[PORTC,1] : Bit;**)

## Beispiel

```
1   PROGRAM test_b2w;
2
3   DEVICE = mega8;
4
5   VAR
6     PORTC, DDRC, PinC : ByteConReg;
7
8     Eingang[PinC,0] : Bit;
9     Ausgang[PORTC,1] : Bit;
10
11  BEGIN
12    DDRC:=%11111110;
13
14    if BitHigh(Eingang) then
15      ClearBit(Ausgang);
16    else
17      SetBit(Ausgang);
18    endif;
19
20
21  END test_b2w.
```

Die Prozedur ClearBit löscht das Bit 1 (wird auf den Binärwert 0 gesetzt) in PORTC (Ausgang).

## siehe auch

**BitLow**, **BitHigh**, **SetBit**

# CONST

**Befehl aus der Gruppe:** PASCAL-Sprachreferenz

Das Schlüsselwort **CONST** wird im Definitions-Bereich genutzt und kennzeichnet den Beginn der Definition der Konstante(n).

## Syntax (CONST)

**CONST**

**<Konstanten-Definition>**

**<Konstanten-Definition>**

**...**

Nach dem Schlüsselwort **CONST** folgt die Liste der Konstanten-Definitionen. Die Liste kann minimal 0 und maximal 9999 Elemente enthalten. Bei der Anzahl diesen Elementen zählen die Konstanten und Variablen zusammen.

## Syntax (<Konstanten-Definition>)

**<Konstanten-Definition>**

**<einzelne Konstante> | <Konstanten-Array>**

In der <Konstanten-Definition> können entweder eine <einzelne Konstante> oder ein <Konstanten-Array> definiert werden.

## Syntax (<einzelne Konstante>)

**<einzelne Konstante>**

**<Name der Konstante> : <Typ der Konstante> = <Konstanten-Term> ;**

Die <einzelne Konstante> besteht aus dem <Namen der Konstante>. Der Name muss mit einem Buchstaben beginnen und muss eindeutig sein (darf noch nicht vorher in einer anderen Deklaration genutzt worden sein).

Danach folgt das Zeichen : und der <Typ der Konstante>.

Als <Typ der Konstante> sind folgende Typen möglich:

- Byte
- Word
- DWord
- Boolean
- Char

Danach folgt das Zeichen = der <Konstanten-Term> und das Zeichen ; .

**Syntax (<Konstanten-Term>)**

**<Konstanten-Term>**

**<Konstanten-Wert oder Konstanten-Bezeichner> ( <Operator> <Konstanten-Wert oder Konstanten-Bezeichner> )**

**<Operator> -> + - * /**

Im einfachsten Fall wird mit dem <Konstanten-Term> ein Konstanten-Wert (z.B. 50) definiert. Der Wertebereich des Konstanten-Wertes muss dem gewählten Typ entsprechen (z.B. bei Byte von 0..255).

Es kann auch ein Konstanten-Bezeichner angegeben werden. Das ist eine Konstante, die vorher bereits definiert wurde. Wichtig ist, dass der Typ des Konstanten-Bezeichners mit dem aktuell definierten Typ übereinstimmt.

Der <Konstanten-Term> kann aber auch etwas komplexer sein. Unter Umständen ist es sinnvoll, während der Definition einer Konstante gleich eine kleine Berechnung durchzuführen (z.B. wenn der Teiler eines Timers in Abhängigkeit von der vorher definierten Prozessortaktfrequenz berechnet werden soll). Dann folgt nach dem <Konstanten-Wert oder Konstanten-Bezeichner> ein <Operator> (+ - * /).

Die Länge des Terms ist auf maximal 50 Elemente (Wert, Operator, Bezeichner) begrenzt.

**Achtung!!!**

**Mathematische Vorrangregeln (Punktrechnung geht vor Strichrechnung) werden an dieser Stelle nicht beachtet. In diesem Fall müssen**

**Zwischenkonstanten gebildet werden.**

**Das folgende Beispiel:**

**TestKonstante : DWord = 20 + 40 * 4;**

**ergibt 3200. Das ist falsch und muss folgendermaßen definiert werden:**

**TestKonstante1 : DWord = 40 * 4;**

**TestKonstante : DWord = 20 + TestKonstante1;**

**ergibt 180 (das ist richtig).**

**Hinweis:** Die eigentliche Berechnung des <Konstanten-Terms> erfolgt innerhalb des Compilers mit einer Variablen mit einem sehr großen Wertebereich (LongInt). Erst beim Ergebnis wird geprüft, ob der Ziel-Wertebereich eingehalten wird. So ist folgende Definition möglich:

**Prozessortakt : Word = 20000;**

**Faktor : Word = Prozessortakt * 100 / 12345;**

Auch wenn der Wetebereich der Word-Konstante "Faktor" (max. 65535) während der Berechnung (genauer nach der Multiplikation mit 100) kurzzeitig überschritten wird, ist das Ergebnis wieder im Wetebereich des Typs Word. Damit ist die Berechnung ohne Compilerfehler möglich.

## Beispiel (einzelne Konstante)

```
1  PROGRAM test_const;
2
3  DEVICE = mega8;
4
5  CONST
6
7    c_byte : Word = 50;
8    d_byte : Word = c_byte;
9    e_byte : Word = 20000*7/5;
10   xb : Boolean = false;
11   f_byte : dWord = %101 * $40;
12
13   Prozessortakt : Word = 20000;
14   Faktor : Word = Prozessortakt * 100 / 12345;
15
16 BEGIN
17
18 END test_const.
```

## siehe auch

**PROGRAM**

# DEC (Decrement)

Die Prozedur **DEC** verringert den Wert der (in der Klammer angegebenen) Variable um eins. Das ist gleichbedeutend mit dem arithmetischen Ausdruck:

**Variable := Variable - 1;**

Wird mit dieser Operation der Wertebereich der Variablen unterschritten, entspricht das Ergebnis dem letzten Wert des Wertebereiches.

Beispiel: Variable hat den Typ Byte und den Wert 0, dann ist das Ergebnis von DEC(Variable) gleich 255.

## Syntax

**DEC ( < Variable vom Typ Byte, Char, Word oder DWord > ) ;**

## Parameter

< Name der Variable >

## Beispiel

```
1   PROGRAM UNO_DEC;
2
3   DEVICE = mega328p;
4
5   {$I mega328P_Register.INC}
6
7
8   CONST
9     b1 : Byte = 214;
10
11
12  VAR
13
14    b2 : Byte;
15
16
17  BEGIN
18
19    b2:=b1;
20
21    DEC(b2);
22
23
24
25  END UNO_DEC.
26
```

In Zeile 19 wird der Byte-Variablen **b2** der Inhalt der Konstante **b1** mit dem Wert 214 zugewiesen.

Nach der **DEC** Procedur in Zeile 21 ist der Wert der Variablen **b2** gleich 213.

## siehe auch

**INC**

# DisableInts

Die Prozedur **DisableInts** sperrt die Ausführung von Interrupts des Mikrocontrollers.

Hinweis: Dieser Befehl wird durch Ausführen der Assembleranweisung "CLI" umgesetzt. CLI löscht das globale Interrupt-Flag des Microkontrollers.

Nach dem Rücksetzen des Microkontrollers ist das Interrupt-Flag gelöscht (kein Interrupt wird ausgeführt).

## Syntax

**DisableInts**

## Parameter

keine

## Beispiel

```
1   PROGRAM test_ISR;
2
3   DEVICE = mega8;
4
5   {$I mega8_Register.INC}
6
7   procedure InitPorts;
8   begin
9     DDRB:=  %11111111;
10    DDRC:=  %11111111;
11    DDRD:=  %11111100;
12  end InitPorts;
13
14  BEGIN
15
16    InitPorts;
17
18    EnableInts;
19
20    LOOP
21    ENDLOOP;
22
23    DisableInts;
24
25  END test_ISR.
```

Ab der Zeile 18 wird die Abarbeitung von Interrupts (mit EnableInts) auf diesem Microcontroller aktiviert.

Ab der Zeile 23 werden Interrupts (mit DisabeInts) auf diesem Microcontroller gesperrt.

## siehe auch

**EnableInts**

# DWHigh

Die Funktion **DWHigh** liest das höherwertige Word aus dem angegebenen DWord und übergibt es als Rückgabewert.

## Syntax

**< Variable von Typ Word > := DWHigh (< Name der DWord-Konstante oder -Variable > ) ;**

## Parameter

< Name der DWord-Konstante oder -Variable > Der Name eines DWords, aus dem das höherwertige Word gelesen werden soll. Das DWord kann eine lokale oder globale Variable oder eine Konstante sein.

## Beispiel

```
1  PROGRAM test_DWHigh;
2
3  DEVICE = mega8;
4
5  CONST
6    dw1 : DWord = $98765431;
7
8  VAR
9    w1 : Word;
10
11 BEGIN
12
13   w1:=DWHigh(dw1);
14
15 END test_DWHigh.
```

In Zeile 14 wird das höherwertige Word aus der DWord Konstante "dw1" gelesen. Der Wert der Word-Variablen "w1" ist danach $9876.

## siehe auch

**DWLow**, **DWordToByte**, **ByteToDWord**, **WordToDWord**

# DWLow

Die Funktion **DWLow** liest das niederwertige Word aus dem angegebenen DWord und übergibt es als Rückgabewert.

## Syntax

**< Variable von Typ Word > := DWLow ( < Name des DWord > ) ;**

## Parameter

< Name des DWord > Der Name eines DWords aus dem das niederwertige Word gelesen werden soll. Das DWord kann eine lokale oder globale Variable oder eine Konstante sein.

## Beispiel

```
1   PROGRAM test_DWLow;
2
3   DEVICE = mega8;
4
5   CONST
6     dw1 : DWord = 98765431;
7
8   VAR
9     w1 : Word;
10
11  BEGIN
12
13    w1:=DWLow(dw1);
14
15  END test_DWLow.
```

In Zeile 13 wird das niederwertige Word aus der DWord Konstante "dw1" gelesen. Der Wert der Word-Variablen "w1" nach DWLow ist $5431.

## siehe auch

**DWHigh**, **DWordToByte**, **ByteToDWord**, **WordToDWord**

# DWordToByte

Die Funktion **DWordToByte** liest ein Byte an der angegebenen Position aus dem angegebenen DWord und übergibt es als Rückgabewert.

## Syntax

**< Variable von Typ Byte > := DWordToByte ( < Position im DWord 0..3 > < Name des DWord > ) ;**

## Parameter

< Position im DWord 0..3 > Ein DWord besteht aus 4 Bytes. Hier wird das Byte angegeben, das gelesen werden soll. Byte 0 ist das niederwertigste Byte und Byte 3 ist das höchstwertige Byte.

< Name des DWord > Der Name eines DWords aus dem das Byte gelesen werden soll. Das DWord kann eine lokale oder globale Variable oder eine Konstante sein.

## Beispiel

```
1  PROGRAM DWord;
2  // Testprogramm fÃ¼r DWord-Proceduren
3
4  DEVICE = mega644P;
5
6  CONST
7    dw1 : DWord = 98765431;
8
9  VAR
10   b1 : Byte;
11
12 BEGIN
13
14   b1:=DWordToByte(0,dw1);
15
16 END DWord.
```

In Zeile 14 wird mit DWordToByte ein Byte aus der DWord-Konstante "dw1" an der Stelle "0" gelesen. Das gelesene Byte mit dem Inhalt $31 wird in die Bytevariable "b1" geschrieben.

## siehe auch

**DWHigh**, **DWLow**, **ByteToDWord**, **WordToDWord**

# EnableInts

Die Prozedur **EnableInts** erlaubt die Ausführung von Interrupts des Mikrocontrollers. Hinweis: Dieser Befehl wird durch Ausführen der Assembleranweisung "SEI" umgesetzt. SEI setzt das globale Interrupt-Flag.

## Syntax

**EnableInts**

## Parameter

keine

## Beispiel

```
1  PROGRAM test_ISR;
2
3  DEVICE = mega8;
4
5  {$I mega8_Register.INC}
6
7  procedure InitPorts;
8  begin
9    DDRB:=  %11111111;
10   DDRC:=  %11111111;
11   DDRD:=  %11111100;
12 end InitPorts;
13
14 BEGIN
15
16   InitPorts;
17
18   EnableInts;
19
20   LOOP
21   ENDLOOP;
22
23   DisableInts;
24
25 END test_ISR.
```

Ab der Zeile 18 wird die Abarbeitung von Interrupts (mit EnableInts) auf diesem Microcontroller aktiviert.

Ab der Zeile 23 werden Interrupts (mit DisabeInts) auf diesem Microcontroller gesperrt.

## siehe auch

**DisableInts**

# END

**Befehl aus der Gruppe:** PASCAL-Sprachreferenz

Das Schlüsselwort **END** kennzeichnet das Ende folgender Blöcke innerhalb eines PASCAL-Programm-Quelltextes.

- PROGRAM
- PROCEDURE
- FUNCTION

## Syntax (PROGRAM)

**PROGRAM <Program-Name> ;**

**<Definitions-Bereich>**

**BEGIN**

**<Implementations-Bereich>**

**END <Program-Name> .**

## Syntax (PROCEDURE)

**PROCEDURE <Prozedur-Name> (<Übergabe-Parameter>) ;**

**<Definitions-Bereich>**

**BEGIN**

**<Implementations-Bereich>**

**END <Prozedur-Name> ;**

## Syntax (FUNCTION)

**FUNCTION <Funktion-Name> (<Übergabe-Parameter>) : <Rückgabe-Parameter> ;**

**<Definitions-Bereich>**

**BEGIN**

**<Implementations-Bereich>**

**END <Prozedur-Name> ;**

## Beispiel (PROGRAM)

```
1  PROGRAM PRO;
2
3  DEVICE = mega8;
4
5  {$I mega8_Register.INC}
6
7  BEGIN
8
9  END PRO.
```

## siehe auch

**PROGRAM**

# High

Die Funktion **High** liest das höherwertige Byte einer Word-Kostante oder Word-Variable und übergibt es als Rückgabewert.

## Syntax

**< Variable von Typ Byte > := High ( < Name der Word-Konstante oder -Variable > ) ;**

## Parameter

< Name der Word-Konstante oder -Variable >

## Beispiel

```
1   PROGRAM test_High;
2
3   DEVICE = mega8;
4
5   CONST
6     q1 : Word = $9876;
7
8   VAR
9     a2 : Byte;
10
11  BEGIN
12
13    a2:=High(q1);
14
15  END test_High.
```

In Zeile 13 wird durch die Funktion High das höherwertige Byte der Word Konstante "q1" gelesen und in die Byte-Variable "a2" geschrieben.
Der Wert der Byte-Variablen "a2" nach dieser Operation ist $98.

## siehe auch

**Low**, **ByteToWord**

# IF THEN

**Befehl aus der Gruppe:** PASCAL-Sprachreferenz

Die bedingte Anweisung **IF THEN** führt eine Gruppe von <Anweisungen> nur aus, wenn die oben angegebene <Bedingung> erfüllt ist. Die Erweiterung dieser bedingten Anweisung durch einen **ELSE** Abschnitt, gestattet die Abarbeitung einer weiteren Gruppe von <Anweisungen>, wenn die <Bedingung> **nicht** erfüllt ist.

## Syntax 1 (ohne ELSE)

**IF <Bedingung> THEN**

**<0..n Anweisungen>**

**ENDIF;**

## Syntax 2 (mit ELSE)

**IF <Bedingung> THEN**

**<0..n Anweisungen>**

**ELSE**

**<0..n Anweisungen>**

**ENDIF;**

## Parameter

<Bedingung> = BOOLscher Ausdruck (Wahr oder Falsch)

<0..n Anweisungen> = Gruppe von einzelnen Anweisungen, Verzweigungen oder weiteren bedingten Anweisungen

## Beispiel 1 (ohne ELSE)

```
1   PROGRAM test_if_then;
2
3   DEVICE = mega1284P;
4
5   {$I mega1284_Register.INC}
6
7   VAR
8     Kursor_Schleife : DWord;
9     Ergebnis : Boolean;
10
11  BEGIN
12
13    Ergebnis:=false;
14
15    if Kursor_Schleife=100000 then
16      Ergebnis:=true;
17    endif;
18
19  END test_if_then.
```

Die Bedingung dieser bedingten **IF THEN** Anweisung besteht aus dem Vergleich der Variable "Kursor_Schleife" mit der Konstanten 100.000. Ist das Ergebnis dieses Vergleichs wahr (true), so werden die Anweisungen zwischen den Schlüsselworten "then" und "endif" ausgeführt. In diesem Beispiel wäre das "Ergebnis:=true;".

Ist das Ergebnis dieses Vergleichs falsch (false), so wird mit der Abarbeitung der Anweisungen nach dem "endif" fortgesetzt.

## Beispiel 2 (mit ELSE)

```
1   PROGRAM test_if_then;
2
3   DEVICE = mega8;
4
5   VAR
6     Zahl1, Zahl2, Zahl3 : Byte;
7
8   BEGIN
9
10    Zahl1 := 5;
11    Zahl2 := 6;
12
13    if Zahl1>Zahl2 then
14      Zahl3:=1;
15    else
16      Zahl3:=2;
17    endif;
18
19  END test_if_then.
```

Die Bedingung dieser bedingten **IF THEN ELSE**Anweisung besteht aus einem Vergleich der Byte-Variablen "Zahl1" und "Zahl2". Ist die Bedingung erfüllt, dass heißt, ist das Ergebnis des Vergleichs "Zahl1>Zahl2" wahr (true), so wird der Abschnitt zwischen den Anweisungen "then" und "else" abgearbeitet. In diesem Beispiel würde die Anweisung "Zahl3:=1;" ausgeführt. Der Abschnitt nach dem "else" bis zum "endif;" wird übersprungen.

Ist das Ergebnis des Vergleichs "Zahl1>Zahl2" falsch (false), so wird mit der Abarbeitung der Anweisungen nach dem "else" bis zum "endif" fortgesetzt. In diesem Beispiel würde die Anweisung "Zahl3:=2;" ausgeführt. Der Abschnitt zwischen dem "then" und dem "else" wird übersprungen.

## siehe auch

# INC (Increment)

Die Prozedur **INC** erhöht den Wert der (in der Klammer angegebenen) Variable um eins. Das ist gleichbedeutend mit dem arithmetischen Ausdruck:

**Variable := Variable + 1;**

Wird mit dieser Operation der Wertebereich der Variablen überschritten, ist das Ergebnis 0.

Beispiel: Variable hat den Typ Byte und den Wert 255, dann ist das Ergebnis von INC(Variable) gleich 0.

## Syntax

**INC ( < Variable vom Typ Byte, Char, Word oder DWord > ) ;**

## Parameter

< Name der Variable >

## Beispiel

```
1   PROGRAM UNO_INC;
2
3   DEVICE = mega328p;
4
5   {$I mega328P_Register.INC}
6
7
8   CONST
9     b1 : Byte = 5;
10
11
12  VAR
13
14    b2 : Byte;
15
16
17  BEGIN
18
19    b2:=b1;
20
21    INC(b2);
22
23
24
25  END UNO_INC.
26
```

In Zeile 19 wird der Byte-Variablen **b2** der Inhalt der Konstante **b1** mit dem Wert 5 zugewiesen.

Nach der **INC** Procedur in Zeile 21 ist der Wert der Variablen **b2** gleich 6.

## siehe auch

**DEC**

# Length

Die Funktion **Length** ermittelt die Länge der angegebenen Zeichenkette (String) und übergibt sie als Rückgabewert.

## Syntax

**< Variable von Typ Byte > := Length( < Variable von Typ String > ) ;**

## Parameter

< Variable von Typ String >

## Beispiel

```
1   PROGRAM test_string;
2
3   DEVICE = mega644P;
4
5   VAR
6     Write_String : STRING[20];
7     String_Laenge : Byte;
8
9   BEGIN
10
11    Write_String := 'Hallo Welt!!!';
12    String_Laenge := Length(Write_String);
13
14  END test_string.
```

In der Zeile 11 wird der String-Variablen "Write_String" mit der maximale Länge von 20 Zeichen die Zeichenkette "Hallo Welt!!!" zugewiesen.

In Zeile 12 wird durch die Funktion "Length" die aktuell belegte Länge der String-Variablen ermittelt und in die Byte-Variable "String_Laenge" geschrieben. Der Wert der Byte-Variablen "String_Laenge" nach dieser Operation ist 13 (Anzahl der Zeichen von "Hallo Welt!!!").

## siehe auch

**ReadAddrArray**, **WriteAddrArray**

## Low

Die Funktion **Low** liest das niederwertige Byte einer Word-Kostante oder -Variable und übergibt es als Rückgabewert.

## Syntax

**< Variable von Typ Byte > := Low ( < Name der Word-Konstante oder -Variable > ) ;**

## Parameter

< Name der Word-Konstante oder -Variable >

## Beispiel

```
1   PROGRAM test_Low;
2
3   DEVICE = mega8;
4
5   VAR
6     b1 : Byte;
7     w1 : Word;
8
9   BEGIN
10
11    w1:=$8415;
12    b1:=Low(w1);
13
14  END test_Low.
```

In Zeile 12 wird durch die Fuktion "Low" das niederwertige Byte aus der Word-Variablen "w1" gelesen und in die Byte-Variable "b1" geschrieben.
Der Wert der Byte-Variablen "b1" nach dieser Operation ist $15.

## siehe auch

**High**, **DWLow**, **DWordToByte**, **ByteToWord**, **WordToDWord**

# NOP (No Operation)

Die **NOP** Anweisung generiert den Assembler-Befehl **NOP**.

Durch die **NOP** Anweisung werden keinerlei Änderungen im Programmumfeld vorgenommen.

Die Abarbeitungsdauer beträgt 1 Takt. Somit können definierte Taktzeiten generiert werden (z.B. 3 NOP Anweisungen gleich 3 Takte Verzögerung).

## Syntax

**NOP ;**

## Beispiel

```
1   PROGRAM UNO_NOP;
2
3   DEVICE = mega328p;
4
5   {$I mega328P_Register.INC}
6
7   BEGIN
8
9     NOP;
10
11  END UNO_NOP.
```

In Zeile 9 der NOP Befehl. Keine weiteren Änderungen.

```
1   ; NOP
2           NOP
```

In der Assembler-Ansicht des RONPAS-Compilers ist der generierte Assembler-Befehl - NOP sichtbar.

# PROGRAM

**Befehl aus der Gruppe:** PASCAL-Sprachreferenz

Das Schlüsselwort **PROGRAM** kennzeichnet den Beginn eines PASCAL-Programm-Quelltextes.

## Syntax

**PROGRAM <Program-Name> ;**

**<Definitions-Bereich>**

**BEGIN**

**<Implementations-Bereich>**

**END <Program-Name> .**

## Beispiel

```
1  PROGRAM PRO;
2
3  DEVICE = mega8;
4
5  {$I mega8_Register.INC}
6
7  BEGIN
8
9  END PRO.
```

Zeile 1 zeigt den Beginn des PASCAL-Programm-Quelltextes mit dem Wort **PROGRAM**.

## siehe auch

**END**

# SetBit

Die Prozedur **SetBit** setzt das angegebene Bit auf den Binärwert 1.

Alle anderen Bits der angegebenen Variable werden nicht verändert.

## Syntax

**SetBit ( < das_Bit > ) ;**

## Parameter

das_Bit = Der Parameter muss als Bit im Vereinbarungsteil vorher vereinbart worden sein (z.B. **Ausgang[PORTC,1] : Bit;**)

## Beispiel

```
1   PROGRAM test_sb;
2
3   DEVICE = mega8;
4
5   {$I mega8_Register.INC}
6
7   VAR
8
9     d1 : DWord;
10
11    ein_DWord[d1,30] : Bit;
12
13  BEGIN
14
15    SetBit(ein_DWord);
16
17  END test_sb.
```

Die Prozedur SetBit setzt (in Zeile 15) das Bit 30 der Variable "ein_DWord" auf den Binärwert 1.

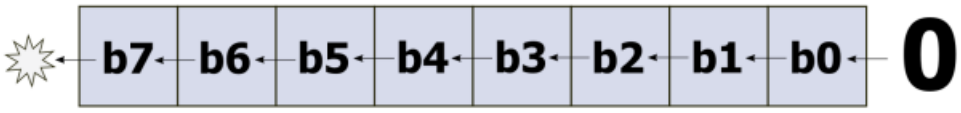
## siehe auch

**ClearBit**, **BitLow**, **BitHigh**

# SHL (Shift Left)

Die Prozedur **SHL** schiebt eine logische 0 in das Bit b0 der Variablen und verschiebt alle Bitstellen der Variable um ein Bit nach links. Der Inhalt des höchstwertigen Bits geht verloren.

Beispiel für SHL auf eine Byte-Variable:




Für anderen Typen (Word, DWord) läuft es entsprechend mit mehreren Stellen.

Mathematisch entspricht die Operation einer binären Multiplikation mit 2.

## Syntax

**SHL ( < Variable vom Typ Byte, Char, Word oder DWord > ) ;**

## Parameter

< Name der Variable >

## Beispiel

```
1   PROGRAM UNO_SHL;
2
3   DEVICE = mega328p;
4
5   {$I mega328P_Register.INC}
6
7
8   CONST
9     b1 : Byte = %00111010;
10
11
12  VAR
13
14    b2 : Byte;
15
16
17  BEGIN
18
19    b2:=b1;
20
21    SHL(b2);
22
23
24
25  END UNO_SHL.
26
```

In Zeile 19 wird der Byte-Variablen **b2** der Inhalt der Konstante **b1** mit dem Binär-Wert 00111010 (3AH) zugewiesen.

Nach der Verschiebeoperation **SHL** in Zeile 21 ist der Binär-Wert der Variablen **b2** gleich 01110100 (74H).

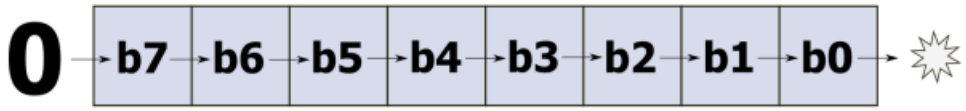
## siehe auch

**SHR**

# SHR (Shift Right)

Die Prozedur **SHR** schiebt eine logische 0 in das höchstwertige Bit (hier Bit b7) der Variablen und verschiebt alle Bitstellen der Variable um ein Bit nach rechts. Der Inhalt des niederwertigsten Bits geht verloren.

Beispiel für SHR auf eine Byte-Variable:




Für anderen Typen (Word, DWord) läuft es entsprechend mit mehreren Stellen.

Mathematisch entspricht die Operation einer binären Division durch 2.

## Syntax

**SHR ( < Variable vom Typ Byte, Char, Word oder DWord > ) ;**

## Parameter

< Name der Variable >

## Beispiel

```
1   PROGRAM UNO_SHR;
2
3   DEVICE = mega328p;
4
5   {$I mega328P_Register.INC}
6
7
8   CONST
9     b1 : Byte = %00101011;
10
11
12  VAR
13
14    b2 : Byte;
15
16
17  BEGIN
18
19    b2:=b1;
20
21    SHR(b2);
22
23
24
25  END UNO_SHR.
```

In Zeile 19 wird der Byte-Variablen **b2** der Inhalt der Konstante **b1** mit dem Binär-Wert 00101011 (2BH) zugewiesen.

Nach der Verschiebeoperation **SHL** in Zeile 21 ist der Binär-Wert der Variablen **b2** gleich 00010101 (15H).

## siehe auch

**SHL**

# WHILE DO

**Befehl aus der Gruppe:** PASCAL-Sprachreferenz

Die **WHILE-DO**-Schleife führt eine Gruppe von <Anweisungen> nur aus, wenn die zwischen WHILE und DO angegebene <Bedingung> erfüllt ist.

Abgeschlossen wird die **WHILE-DO**-Schleife mit **ENDWHILE;**. Ist das Ende der Schleife erreicht, wird geprüft, ob die <Bedingung> noch erfüllt ist. Wenn ja, so wird die Schleife noch einmal durchlaufen.

Ist die <Bedingung> nicht mehr erfüllt, so wird mit der Abarbeitung der nächsten <Anweisung> nach dem **ENDWHILE;** fortgesetzt.

Ist die <Bedingung> zwischen WHILE und DO zum Zeitpunkt der Programmabarbeitung nicht erfüllt, so wird mit der nächsten Anweisung nach dem **ENDWHILE;** fortgesetzt.

Im Gegensatz zur **FOR TO DO**-Schleife benötigt die **WHILE-DO**-Schleife keine Zählvariable.

## Syntax

**WHILE <Bedingung> DO**

**<0..n Anweisungen>**

**ENDWHILE;**

## Parameter

<Bedingung> = BOOLscher Ausdruck (Wahr oder Falsch)

<0..n Anweisungen> = Gruppe von einzelnen Anweisungen, Verzweigungen oder weiteren bedingten Anweisungen

## Beispiel

```
1   PROGRAM test_while_do;
2
3   DEVICE = mega1284P;
4
5   {$I mega1284_Register.INC}
6
7   VAR
8     Kursor_Schleife : DWord;
9
10  BEGIN
11
12    Kursor_Schleife:=5;
13
14    while Kursor_Schleife<100 do
15      Kursor_Schleife:=Kursor_Schleife+10;
16    endwhile;
17
18  END test_while_do.
```

Die <Bedingung> für diese **WHILE-DO**-Schleife ist "Kursor_Schleife<100" (Zeile 14). Das bedeutet, solange das Ergebnis des Vergleichs wahr ist, wird die Schleife abgearbeitet.

Wichtig ist, dass die Bedingung beim Eintritt in die Schleife bereits erfüllt sein muß.

Daher wird in diesem Beispiel in Zeile 12 mit "Kursor_Schleife:=5;" die Eintrittsbedingung in die Schleife festgelegt.

Mit jedem Schleifendurchlauf wird der Wert der Variable "Kursor_Schleife" in Zeile 15 mit "Kursor_Schleife:=Kursor_Schleife+10;" verändert. Nach jedem Schleifendurchlauf ist er: 15, 25, 35..95.

Nach dem 10. Schleifendurchlauf ist er 105 und die <Bedingung> "Kursor_Schleife<100" ist nicht mehr erfüllt. Damit wird die Schleife nach dem 10. Schleifendurchlauf verlassen.

**Hinweis:** Wird die <Bedingung> nie unwahr (z.B. wenn die <Bedingung> "Kursor_Schleife<>100" heißen würde), dann wird die Schleife auch nie verlassen (Endlosschleife).

## siehe auch

**IF_THEN**

# WordToDWord

Die Funktion **WordToDWord** liest eine Word-Konstante oder -Variable und kopiert diese in den niederwertigen Teil der DWord Variablen als Rückgabewert. Der höherwertige Teil der DWord Variablen wird auf $00 gesetzt.

## Syntax

**< Variable von Typ DWord > := WordToDWord ( < Name der Word-Konstante oder -Variable > ) ;**

## Parameter

< Name der Word-Konstante oder -Variable >

## Beispiel

```
1  PROGRAM DWord;
2
3  DEVICE = mega8;
4
5  {$I mega8_Register.INC}
6
7  CONST
8    c1 : Word = 89;
9
10 VAR
11   dw1 : DWord;
12
13 BEGIN
14
15   dw1:=WordToDWord(c1);
16
17 END DWord.
```

In Zeile 14 wird die Word-Konstante "c1" in die DWord-Variable "dw1" geschrieben.

Der Wert der DWord-Variablen nach der Operation ist $00000089 (Hexadezimal).

## siehe auch

**DWLow**, **DWHigh**, **DWordToByte**, **ByteToDWord**