Arnold-Assembler

Die meisten meiner Assembler-Programme sind mit dem Arnold-Assembler übersetzbar. Der Assembler von Alfred Arnold ist ein universeller Makro-Cross-Assembler für eine Vielzahl von Mikroprozessoren und -Controllern. Außerdem ist er komplett kostenlos.

Allerdings erzeugt der Arnold-Assembler erzeugt nicht direkt Binär-Dateien oder auch Hex-Dateien, sondern *.p-Zwischencode-Dateien. Diese müssen mit einem weiteren Programm p2bin.exe erst zu einer Binär-Datei umgeformt werden.

Deswegen liegt in den Downloadpaketen oft eine kleine Batch-Datei as.cmd bei, mit der ein Assemblerquelltext in eine Binärdatei umgewandelt wird:

```
as.exe -cpu Z80 -L file.asm
p2bin.exe -r $-$ file.p
del file.p
```

Hinweise

Der Assembler ist recht kompatibel zum M80 und anderen Assemblern. Nur Zeichenketten müssen in „..“ statt Hochkommas geschrieben werden. Einzelne Charakter-Zeichen bleiben in Hochkommas.

Kleine Perl-Programme erleichtern die Arbeit

- convasm.pl konvertiert SYPS K1520 - U880 - Syntax nach Zilog-Syntax
- convida.pl Konvertiert IDAPro-Z80-ASM-Dateien in brauchbares Format für den Arnold-Assembler

Beim **Z8-Prozessor** muss beachtet werden, dass der Assembler automatisch versucht, die Registernutzung zu optimieren. Um 100% originalen Code zu erreichen, ist es hilfreich, RP auf ein ungenutzte Adresse zu setzen:

```
    assume RP:0C0h        ; keine Optimierung durch AS!
```

Beim **Z80-Prozessor** kann man eine alternative Syntax für Hex-Zahlen aktivieren. Das erfolgt mit

```
     INTSYNTAX +0xhex
```

Dann sind Hex-Zahlen in C-Notation zulässig, z.B. 0xc000

Eine weitere Variante ist

```
INTSYNTAX +$hex, +%bin
```

Das erlaubt $efb0, %00100111 zusätzlich zur Suffix-Notation 0EFBh.

Diverses

statt equ EVAL oder .SET

SET und EQU erlauben die Definition typenloser Konstanten, d.h. sie werden keinem Segment zugeordnet und ihre Verwendung erzeugt in kei- nem Fall eine Warnung wegen Segment-Vermischung. Während EQU Konstan- ten definiert, die nicht wieder (mit EQU) geändert werden können, er- laubt SET die Definition von Variablen.

Einige Prozessoren besitzen leider bereits selber einen SET-Befehl. Bei diesen muß EVAL anstelle von SET verwendet werden, falls sich der Maschinenbefehl nicht durch die andere Anzahl der Argumente erkennen läßt. Alternativ ist es auch immer möglich, durch einen vorangestell- ten Punkt (.SET anstelle SET) explizit den Pseudobefehl aufzurufen.

Ausdrücke in IF .. ELSEIF .. ENDIF

```
!!   log. XOR
||   log. OR
&&   log. AND
~~   log. NOT
>>   log. Rechtsschieben
<<   log. Linksschieben
=    Gleichheit
==   Alias für =
<>   Ungleichheit
!=   Alias für <>

MOD    #
SHL    <<
SHR  >>
```

PAGE –> NEWPAGE

Symbolnamen

Wann immer der Name eines Symboles mit einem Punkt (.) anfängt, wird das Symbol nicht mit diesem Namen in der Symboltabelle abgelegt. Stattdessen wird der Name des zuletzt definierten Symbols ohne vorangestellten Punkt davor gehängt. Auf diese Weise nehmen Symbole, deren Name nicht mit einem Punkt anfängt, quasi die Rolle von 'Bereichsgrenzen' ein und Symbole, deren Name mit einem Punkt anfängt, können in jedem Be- reich neu verwendet werden.

Standard-Funktionen

```
; AS-Funktionen
hi              function x,(x>>8)&255
lo              function x, x&255
setlength   function text,len,substr(text+'                    ',0,len)
```

hi(), lo() liefern das obere bzw. untere Byte eines Words
setlength() z.B. für DATE, damit hier immer eine konstante Länge bleibt: db setlength(DATE,10)

individuell z.B. zur Ausgabepositionierung im Bildwiederholspeicher

```
; Z9001 bws(zeile 0..23, spalte 0..39) analog print_at
bws     function z,s,z*40+s+0EC00h
bwsc        function z,s,z*40+s+0E800h
```

Makros

Beim Aufruf eines Makros werden die beim Aufruf angegebenen Parameter- namen überall textuell im Befehlsblock eingesetzt und der sich so er- gebene Assemblercode wird normal assembliert. Das funktioniert auch beim Ersetzen des Befehls

```
LOGOP   MACRO    LINST
    LD    A,B
    LINST    D    ;;apply logical instruction to B and D
    LD    B,A    ;;and result to B
    LD    A,C
    LINST    E    ;;apply logical instruction to C and E
    LD    C,A    ;;and result to C
    RET
    ENDM
LOGOP   XOR
```

Wichtig ist, daß der Assembler alle Parameterna- men im case-sensitiven Modus in Großbuchstaben umsetzt, in Strings aber nie eine Umwandlung in Großbuchstaben erfolgt. Die Makroparame- ternamen müssen in den Stringkonstanten daher groß geschrieben wer- den.

Der Unter- strich erlaubt es, einzelne Makroparameternamen zu einem Symbol zusam- menzuketten, z.B. CALL part1_part2

Ein etwas verstecktes (und mit Vorsicht zu nutzendes) Feature ist, Symbolnamen aus String-Variablen zusammenzubauen, indem man den Namen des Strings mit geschweiften Klammern in den Symbolnamen einbaut. So kann man z.B. den Namen eines Symbols anhand des Wertes eines anderen Symbols festlegen:

```
cnt     .set    cnt+1
temp        equ    "\{CNT}"
        jrnz    skip{temp}
        .
        .
skip{temp}: nop
```

Beispiele:

```
; ich möchte im Arnold-Assembler für Z80-Code Zeichenketten Byte für Byte
mit
; 0A8h XOR-verknüpfen, also statt
; DB CR,"Test",0
; die Bytefolge A5, FC, CD, DB, DC, A8 erzeugen.

    cpu    z80
```

```
CR  equ    0dh

DX      macro X,{NOEXPAND}
    IFNB X
        if EXPRTYPE(x) = 2
            irpc y,x
              DB    'Y' ! 0A8h
                ENDM
            else
                DB  (X) ! 0A8h
        endif
        shift
            DX ALLARGS
        endif
    endm

; Codierte Texte erzeugen:
    DX    CR,"Test",0

    end
```

s.a. FORTH für den KC85/2-4, dort gibt es ein umfangreicheres header-Makro, das den FORTH-Header erzeugt.

ASIDE-Assembler-Makros

```
; Anpassung Arnold-Assembler
    cpu    z80

DEFM    macro    x
    IFNB X
               DB  (X)
        shift
            DEFM ALLARGS
        endif
    endm

DEFS    macro    a,b
    ifb    b
        ds    a
    else
        db    a dup(b)
    endif
    endm

     INTSYNTAX +$hex        ; # --> $

hi              function x,(x>>8)&255
```

```
lo              function x, x&255
```