Tutorial 2

Erstellen, Kompilieren und Ausführen eines Assembler-Programms

ph v/2010/03

In dieser Aufgabe wiederholen wir das Anlegen von Datasets (Allocate) sowie das Füllen mit Daten unter Verwendung des ISPF-Editors, und Sie lernen kennen, wie man ein Assembler-Programm unter z/OS schreibt, kompiliert und ausführt.

Hinweis: Dieses Tutorial wurde unter Verwendung der Benutzer-ID "PRAKT20" erstellt. In allen Dateinamen müssen Sie "PRAKT20" durch ihre eigene Benutzer-ID ersetzen.

Aufgabe: Arbeiten Sie nachfolgendes Tutorial durch.

1. Einrichten der Entwicklungsumgebung

Wir müssen zunächst noch 2 Datasets anlegen. Der eine Dataset (PRAKT20.TEST.ASSEM) soll den Quellkode des Assembler-Programms und die beiden notwendigen Macros SAVEREG, EXITREG, die von dem Assembler-Programm aufgerufen werden, aufnehmen. Der zweite Dataset (PRAKT20.TEST.LOAD) beinhaltet die ausführbare Datei. Einen Dataset haben wir schon in der letzten Aufgabe erstellt: Den "PRAKT20.TEST.CNTL", welcher das JCL-Script ASSSTA02 enthält und zum Kompilieren benutzt wird.

Aufgabe: Legen Sie den Dataset PRAKT20.TEST.ASSEM ("PRAKT20" durch Ihre Benutzer-ID ersetzen) an. Verwenden Sie die gleichen Parameter wie im Tutorial zur Aufgabe 1.
*Legen Sie den Dataset PRAKT20.TEST.LOAD ("PRAKT20" durch Ihre Benutzer-ID ersetzen) an, welcher die ausführbare Datei nach dem Kompilieren aufnehmen soll. Verwenden Sie wieder die Ihnen bekannten Parameter, mit einem Unterschied: Statt im Dateiformat (Record format) "**F**ixed **B**lock" soll dieser Dataset im Dateiformat "Undefined" erstellt werden. Dazu ist an der dafür vorgesehenen Stelle ein "U" als Parameter anzugeben.*

2. Erstellen des Quelltextes des Assembler-Programms

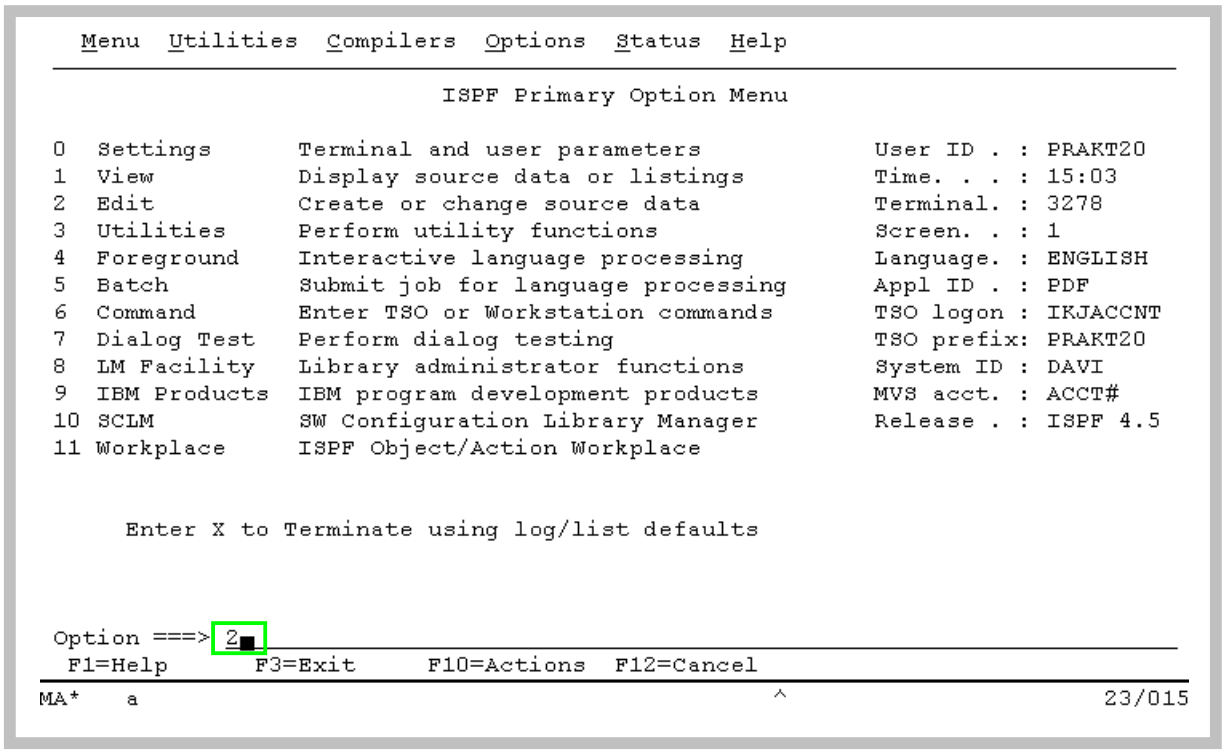

Abbildung 1: "ISPF Primary Option Bildschirm"

Wir haben bisher die Utilities-Funktion benutzt, um unsere Entwicklungsumgebung anzulegen. Hierzu haben wir drei Partitioned Datasets angelegt. Jetzt wollen wir diesen Speicherplatz benutzen, um ein Programm zu schreiben, zu übersetzen und auszuführen.

Dies geschieht mit Hilfe der "Edit"-Funktion. Wie in Abbildung 1 demonstriert, geben wir eine "2" in die Kommandozeile des "ISPF Primary Option Menu" ein und betätigen die Eingabetaste.

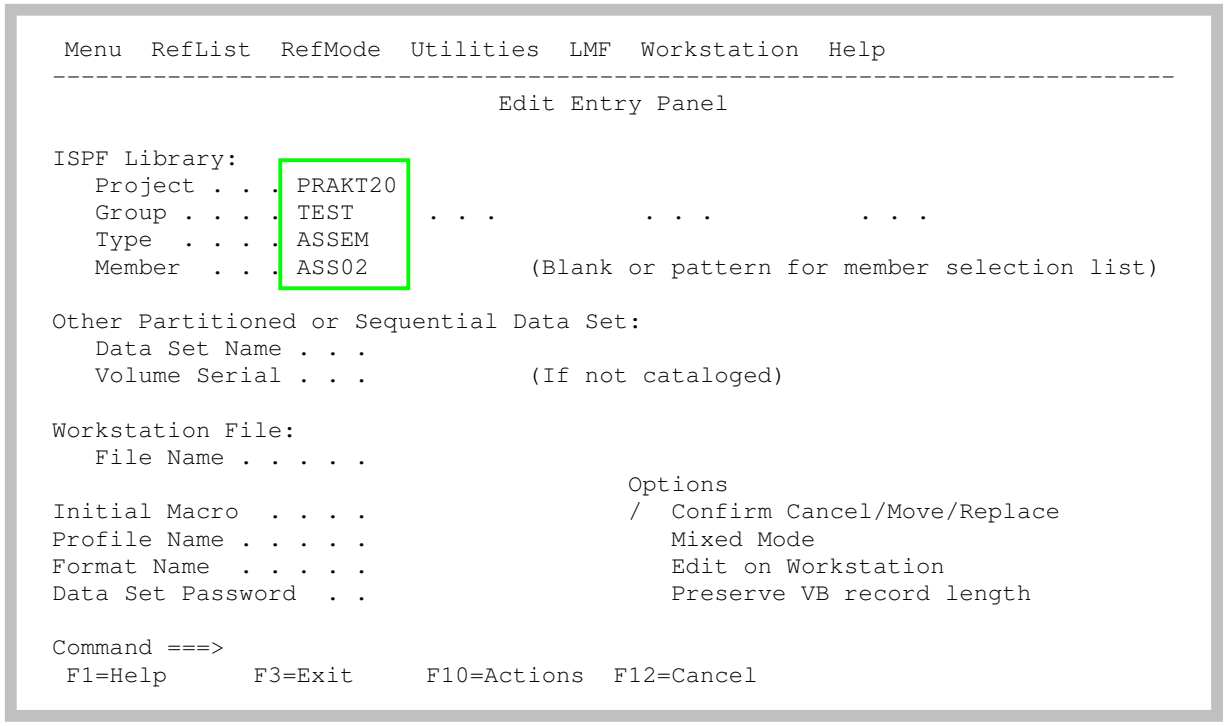

Abbildung 2: "Edit Entry"-Bildschirm

Wir wollen zuerst das Quellprogramm mit Hilfe des ISPF-Editors erstellen. Der "Edit Entry"-Bildschirm fordert uns auf, den Namen des zu editierenden Programms einzugeben (s. Abbildung 2).

Unser Quellprogramm soll als eine (von potentiell mehreren) Files in dem für Quellprogramme von uns vorgesehenen Partitioned Dataset PRAKT20.TEST.ASSEM gespeichert werden.

Wir bezeichnen unseren Member als ASS02. Der volle Name dieses Members ist PRAKT20.TEST.ASSEM(ASS02). Wir geben diese Werte in die dafür vorgesehenen Felder des "Edit Entry"-Bildschirmes ein. Es ist also, wie in Abbildung 2 gezeigt, ihre Benutzer-ID ins Feld "***Project***", "TEST" ins Feld "***Group***", "ASSEM" ins Feld "***Type***" sowie "ASS02" ins Feld "***Member***" einzutragen. Anschließend betätigen Sie die Eingabetaste.

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       PRAKT20.TEST.ASSEM(ASS02) - 01.00               Columns 00001 00072
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
Command ===>                                                 Scroll ===> PAGE
 F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abbildung 3: ISPF-Editor

Abbildung 3 zeigt die Oberfläche des ISPF-Editors.

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       PRAKT20.TEST.ASSEM(ASS02) - 01.00               Columns 00001 00072
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
'''''' ASS02    CSECT
'''''' * Formalismus fuer Assembler-Code:
'''''' * Angaben in den Stellen 1 - 71, Fortsetzungszeichen '*' in Stelle 72,
'''''' * Optional: Nummerierung in Stellen 73 - 80
'''''' * Kommentar-Zeile: '*' in Stelle 1 gefolgt vom Kommentar bis 71
'''''' * Assemblercode-Zeile: Optional: Referenzname in 1 - 8
'''''' *                       Zwingend: Operation ab 10
'''''' *                       Optional: Parameter mit mind. 1 Blank hinter
'''''' *                                 Operation, ueblich ab 16
'''''' *                       Optional: Kommentar mit mind. 1 Blank hinter
'''''' *                                 Parametern bis 71
'''''' ASS02    AMODE 24    Adressierungsmodus: 24 Bits, max 16 MB
'''''' ASS02    RMODE 24    Resident: Unterhalb 16 MB
'''''' * Die Aufloesung der Makros ist nach der Umwandlung in der
'''''' * Assembler-Liste zu sehen
''''''           SAVEREG     Makro: Einstieg und Retten Register
Command ===>                                                 Scroll ===> PAGE
 F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abbildung 4: ISPF-Editor mit Assembler-Programm, Panel 1/3

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       PRAKT20.TEST.ASSEM(ASS02) - 01.00               Columns 00001 00072
001700 ********************************************************************
001800           OPEN  (PRINTOUT,(OUTPUT)) Makro: Open DCB PRINTOUT
001900           PUT   PRINTOUT,PRINTREC   Makro: Schreibe Text
002000           CLOSE PRINTOUT            Makro: Close DCB PRINTOUT
002100 *
002200 * Bei Ende des Programms wird vom System der Inhalt von Register 15
002300 * als Return-Code verwendet, der in der JCL als Condition-Code
002400 * abgefragt werden kann.
002500           SR    R15,R15             Subtract Register --> R15 = 0
002600 *                                   ---> Fehlerfreies Ende
002700           B     ENDE                Verzweige unbedingt nach ENDE
002800 ********************************************************************
002900 ENDE      EQU   *     Bezeichnet die Programmstelle fuer den Assembler
003000 *                     mit Namen
003100           EXITREG     Makro: Laden der Register mit dem Inhalt zur
003200 *                            Zeit des Programmbeginns und Ruecksprung
003300 *                            zum System
003400 *****     Definitionen
003500 *         DC = Declare Constant
Command ===>                                                Scroll ===> PAGE
 F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abbildung 5: ISPF-Editor mit Assembler-Programm, Panel 2/3

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       PRAKT20.TEST.ASSEM(ASS02) - 01.02               Columns 00001 00072
003600 PRINTREC DC    Cl80'Hallo Welt, unser erstes TSO-Programm in ASSEMBLER'
003700 PRINTOUT DCB   DDNAME=SYSPRINT, Makro:  Data Control Block             *
003800                DSORG=PS,                                               *
003900                MACRF=(PM),                                             *
004000                LRECL=80
004100           END
****** **************************** Bottom of Data ****************************
Command ===>                                                Scroll ===> PAGE
 F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abbildung 6: ISPF-Editor mit Assembler-Programm, Panel 3/3

Wir schreiben das in der Abbildung 4 bis Abbildung 6 gezeigte Assembler-Programm ASS02. Letzteres enthält Data Management Macros und Assembler Instruction Statements, die einzeln erklärt werden sollen.

Macros

Eine Macro-Definition besteht aus einer Folge von Statements, die mit einem Macro-Befehl aufgerufen werden kann. Beim Aufruf werden normalerweise Assembler-Befehle generiert und verarbeitet. Der Assembler liefert eine **Macro-Definition** mit

- **Macro-Namen**,
- **Parameter**, die im Macro benutzt werden,
- **Befehlsfolge**, die generiert wird, wenn der Macro-Befehl in dem Quell-Programm erscheint.

Jede Macro-Definition verfügt über ein Macro Definition Header Statement (MACRO), ein Macro-Befehls-Prototyp Statement, ein oder mehrere Assembler Statements, ein Macro Definition End Statement (MEND).

Die Macro Definition Header und End Statements (MACRO, MEND) zeigen dem Assembler den Beginn und das Ende einer Macro-Definition an (s. Abbildung 7, 1)

Das Macro-Befehls-Prototype Statement kennzeichnet das Macro (s. Abbildung 7, 2) und deklariert seine Parameter (s. Abbildung 7, 3). In dem Operanden-Feld des Macro-Befehls können Werte den Parametern, die für die gerufene Macro-Definition deklariert sind, zugewiesen werden (s. Abbildung 7, 4).

Der Body einer Macro-Definition (s. Abbildung 7, 5) enthält die Statements, die beim Aufruf des Macros generiert werden. Diese Statements heißen Modell-Statements. Sie werden gewöhnlich mit Bedingungs- oder anderen Verarbeitungs-Statements eingefügt.

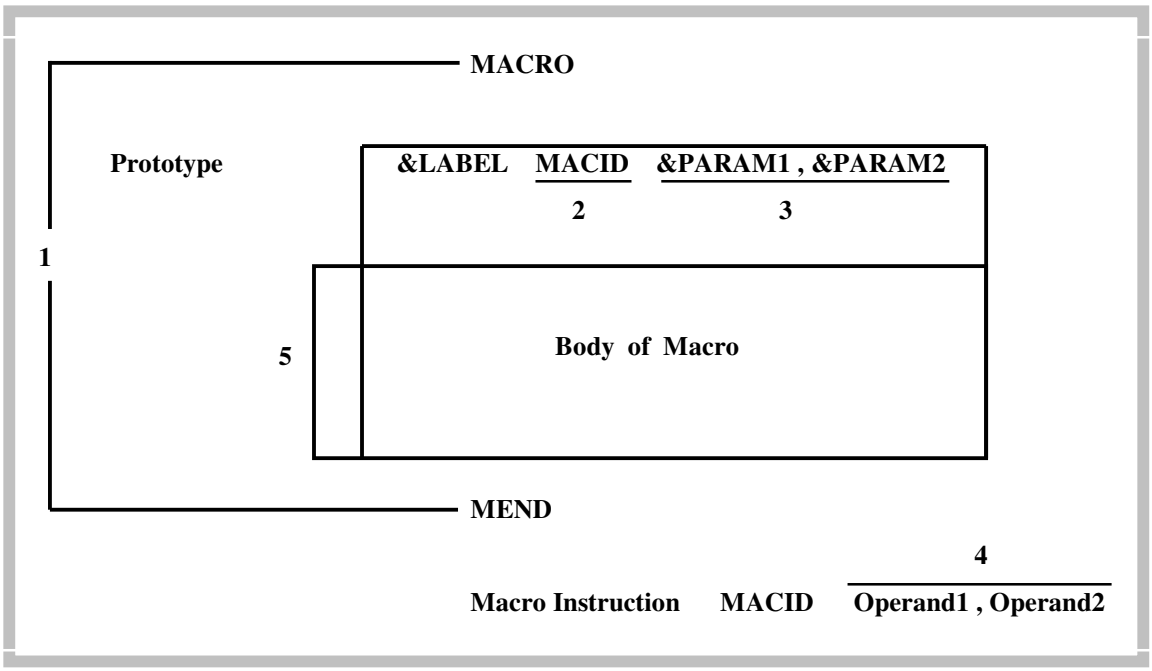

Abbildung 7

Das **OPEN Macro** schließt die spezifizierten Data Control Blocks ab und bereitet die Verarbeitung der Datasets, die darin identifiziert sind, vor. Input Labels werden untersucht und Output Labels erzeugt. Die Steuerung wird den Exit-Routinen, wie in der Data Control Block Exit List beschrieben, übergeben. Es kann eine bestimmte Anzahl von Data Control Block-Adressen und damit verbundene Optionen (Verarbeitungs-Methode, Verarbeitungs-Modus) in dem OPEN Macro angegeben werden. Die Standard-Form des OPEN Macro´s kann wie folgt dargestellt werden:

[label1] **OPEN** **(dcb address[, [(options)] [, …]])**
[, TYPE=J]
[, MODE=24 | 31]

dcb address: Spezifiziert die Adresse der Data Control Blocks für die Datasets, die für die Verarbeitung vorbereitet werden sollen.

options: Geben an, welcher Device-Type und Zugriffsmethode verwendet werden sollen. Die Optionen sind (Option 1): EXTEND, INOUT, OUTPUT, OUTIN, OUTINX, RDBACK, UPDAT; (Option 2): LEAVE, REREAD, DISP

TYPE=J: Man kann OPEN TYPE=J codieren, um festzulegen, dass für jeden Data Control Block, auf den verwiesen wird, ein Job File Control Block (JFCB) für die Nutzung während der Initialisierung vorgesehen ist. Ein JFCB stellt eine interne Repräsentation von Informationen in einem DD Statement dar.

MODE=24|31: Wenn OPEN MODE=31 codiert ist, so wird eine Long Form Parameter-Liste spezifiziert, die 31 Bit-Adressen enthalten kann. Das Programm muss nicht im 31 Bit-Modus ausgeführt werden, um MODE=31 in dem OPEN Macro zu verwenden. Dieser Parameter legt die Form der Parameter-Liste fest, nicht den Adressierungs-Modus des Programms. Der Standard MODE=24 spezifiziert eine Short Form Parameter-Liste mit 24 Bit-Adressen. MODE=31 ist nicht erlaubt, wenn TYPE=J codiert ist.

Das **CLOSE Macro** erzeugt Output Dataset Labels und erlaubt es, Volumes zu platzieren. Die Felder des DCB und DCBE werden unter den Bedingungen vor dem OPEN Macro-Ruf gespeichert, und der Dataset wird von dem verarbeitenden Programm getrennt. Nach einem CLOSE für verschiedene Datasets zeigt ein Return Code von "4" an, dass eines der Datasets VSAM oder Nicht-VSAM nicht erfolgreich geschlossen wurde. Die Standard-Form des Close Macro´s ist:

[label1] **CLOSE** **(dcb address[, [(options)] [, …]])**
[, TYPE=T]
[, MODE=24 | 31]

dcb address: Spezifiziert die Adresse der Data Control Blocks für den geöffneten Dataset, der geschlossen werden soll.

options: Jede dieser Optionen zeigt der Platteneinheit an, wenn der Dataset geschlossen wird. Diese Optionen werden generell mit dem Type=T für Datasets auf Magnetband verwendet. Die Optionen sind: REREAD, LEAVE, REWIND, FREE, DISP.

TYPE=T: Es wird CLOSE TYPE=T für temporär geschlossene Datasets auf Magnetband und Direct Access Volumes, die mit Basic Sequential Access Method (BSAM) verarbeitet werden, codiert.

MODE=24|31: Wenn CLOSE MODE=31 codiert ist, so wird eine Long Form Parameter-Liste spezifiziert, die 31 Bit-Adressen enthalten kann. Das Programm muss nicht im 31 Bit-Modus ausgeführt werden, um MODE=31 in dem CLOSE Macro zu verwenden.

Das **PUT Macro** schreibt einen Record in einen Index-sequentiellen Dataset. Im Move Mode bewegt das PUT Macro einen logischen Record in einen Output-Puffer, von dem er geschrieben wird. Wenn der Locate Mode spezifiziert wird, ist die Adresse des nächsten verfügbaren Output-Puffer-Segments im Register 1 nach einem PUT Macro verfügbar. Das Format des PUT Macro´s:

[label] **PUT** **dcb address**

[, area address]

dcb address: Spezifiziert die Adresse des Data Control Block für den geöffneten Index-sequentiellen Dataset.

area address: Legt die Adresse des Area, das den zu schreibenden Record enthält, fest. Es kann entweder Move oder Locate Mode mit QISAM (Queued Index Sequential Access Method) benutzt werden, sie dürfen aber nicht gemischt in dem spezifizierten Data Control Block auftreten.

> **Locate Mode**: Wenn dieser Mode im Data Control Block gewählt ist, muss die **area address** weggelassen werden. Das System gibt die Adresse des nächsten verfügbaren Puffers im Register 1 zurück. Das ist der Puffer, in den der nächste Record bewegt werden sollte. Der Record wird nicht geschrieben bis ein anderer PUT Macro für denselben DCB oder ein CLOSE Macro gerufen wird.
>
> **Move Mode**: Wenn dieser Mode im DCB gewählt ist, muss die **area address** die Adresse in dem Programm, das den zu schreibenden Record enthält, angegeben werden. Das System bewegt den Record von dem Area zu einem Output Puffer, bevor die Steuerung zurückgegeben wird. Wenn die **area address** weggelassen wird, nimmt das System an, dass Register 0 die Area-Adresse enthält.

Der **DCB (Data Control Block)** für einen QISAM Dataset wird während der Übersetzung eines Programms aufgebaut. Es müssen DSORG und MACRF in dem **DCB** Macro codiert sein. Das Format des DCB Macro ist in "DFSMS/MVS V1R5 Macro Instructions for Data Sets" (z/OS Collection) angegeben.

Assembler Instruction Statements

Die in dem Assembler-Programm ASS02 verwendeten Befehle sind**: AMODE, RMODE, USING, EQU, DC, END.**

Der **AMODE**-Befehl spezifiziert den Address Mode, der mit den Control Sections verbunden ist. Er hat folgende Form:

```
name  AMODE    24
               31
               ANY
```

name: Kann sein

- ein gewöhnliches Symbol,
- ein variables Symbol, dem ein Character String mit einem Wert zugeordnet ist, der für ein gewöhnliches Symbol gültig ist,
- ein Sequence Symbol.

24: Spezifiziert, dass 24 Bit-Adressierungs-Mode mit einer Control Section oder Entry Point verbunden werden soll.

31: Spezifiziert, dass 31 Bit-Adressierungs-Mode mit einer Control Section oder Entry Point verbunden werden soll.

ANY: Der Control Point oder Entry Point ist nicht sensitiv für den Adressierungs-Mode, in dem er eingegeben wird.

AMODE kann irgendwo im Assembler-Programm festgelegt werden. Im Programm sind mehrere AMODE-Befehle möglich, zwei AMODE-Befehle dürfen nicht dasselbe Name-Feld besitzen. Die Spezifikation von AMODE 24 und RMODE ANY oder für dasselbe Name-Feld ist nicht erlaubt, alle anderen Kombinationen sind erlaubt. AMODE oder RMODE dürfen nicht für eine unbenannte Common Control Section spezifiziert werden.

Der **RMODE**-Befehl legt den Residence Mode fest, der verbunden wird mit Control Sections. RMODE hat folgende Synthax:

```
name  AMODE    24
               ANY
```

name: Stellt das Name-Feld, das den Residence Mode mit einer Control Section verbindet, dar.

24: Legt fest, dass ein Residence Mode von 24 mit der Control Section verbunden werden soll.

ANY: Spezifiziert, dass ein Residence Mode von entweder 24 oder 31 mit der Control Section verbunden werden soll. Die Control Section kann oberhalb oder unterhalb von 16 MByte liegen. Für den RMODE-Befehl gelten Eigenschaften analog AMODE bezüglich eines Assembler-Programms.

Der **USING**-Befehl bestimmt eine Base-Adresse mit Bereich und weist ein oder mehrere Base-Register zu. Wenn das Base-Register mit der Base-Adresse geladen wird, dann ist die Adressierbarkeit einer Control Section festgelegt. Um den USING-Befehl richtig zu nutzen, sollte man wissen,

welche Speicherplätze in einer Control Section durch den USING-Befehl adressierbar werden,

wo in einem Quell-Modul kann man implizit Adressen in Befehlsoperanden bezüglich dieser adressierbaren Speicherplätze benutzen.

Der USING-Befehl hat drei Formate:

1) Das Format spezifiziert eine Basis-Adresse, einen optionalen Bereich und ein oder mehrere Basis-Register. Das Format heißt "ordinary USING instruction".
2) Das Format bestimmt eine Basis-Adresse, einen optionalen Bereich, ein oder mehrere Basis-Register und ein USING Label, das als symbolischer Qualifier verwendet werden kann. Dieses Format heißt "labeled USING instruction".
3) Das Format legt eine Basis-Adresse, einen optionalen Bereich und einen relativierbaren Ausdruck anstatt eines oder mehrerer Basis-Register fest. Das Format heißt "dependent USING instruction". Wenn ein Label auch spezifiziert wird, nennt sich das Format "labeled dependent USING instruction".

Der **DC**-Befehl wird benutzt, um die Daten-Konstanten zu definieren, die für die Programm-Ausführung gebraucht werden. Dieser Befehl veranlasst den Assembler, die binäre Darstellung der Daten-Konstanten des Assembler Quell-Moduls zu erzeugen. Dieser Vorgang erfolgt zur Übersetzungszeit. Folgende Typen von Konstanten können von dem DC-Befehl generiert werden: Address, Binary, Character, Decimal, Fixed-point, Floating-point, Graphic, Hexadecimal (s. HLASM V1R3 Language Reference, z/OS Collection).

Das Format sieht folgendermaßen aus:

symbol DC operand

symbol ist eines der folgenden:

- ein **ordinary Symbol**,
- ein **variable Symbol**, das einem Character String mit einem gültigen Wert zugewiesen wurde,
- ein **sequence Symbol**.

operand besitzt vier Subfelder**,** die ersten drei Subfelder beschreiben die Konstante, das vierte liefert den nominellen Wert der Konstanten:

duplication_factor type modifier nominal value

duplication_factor: Bewirkt, dass der nominal_value bereitgestellt wird, der die Anzahl (durch diesen Faktor dargestellt) erzeugen soll.

type: Bestimmt den Typ der Konstanten, die nominal_value darstellt.

modifier: Beschreibt die Länge, die Wichtung und den Exponenten von nominal_value.

nominal_value: Definiert den Wert der Konstanten.

Der **END**-Befehl wird verwendet, um das Ende des Assembler-Programms anzugeben. Es kann auch eine Adresse im Operanden-Feld verwendet werden, an welche die Steuerung nach dem Laden des Programms übergeben wird. Der END-Befehl muss immer das letzte Statement im Quellprogramm bilden, Format:

sequence_symbol END expression language

sequence_symbol: Ist ein Sequence Symbol

expression: Spezifiziert den Punkt, an dem die Steuerung übergeben werden kann, wenn das Laden des Objekt-Programms fertig ist. Dieser Punkt ist gewöhnlich die Adresse des ersten ausführbaren Befehls im Programm.

language: Ist eine Marke für die Nutzung durch den Sprach-Übersetzer, der Assembler-Code erzeugt. Der Operand hat drei Sub-Operanden. Die Werte in diesem Operand werden in die Zeichen 53 - 71 des Record-Endes kopiert.

Der **EQU**-Befehl weist absolute oder relative Werte den Symbolen zu. Er wird benutzt, um

- einzelne absolute Werte den Symbolen zuzuweisen,
- die Werte von vorher definierten Symbolen oder Ausdrücken neuen Symbolen zuzuordnen,
- Ausdrücke zu berechnen, deren Werte zur Codierungszeit unbekannt oder schwer zu berechnen sind. Der Wert der Ausdrücke wird dann einem Symbol zugewiesen.

Die Synthax des EQU-Befehls ist:

symbol EQU expression_1
,expression_2 , expression_3

symbol: Ist eines der folgenden**:**

- Ein ordinary Symbol,
- Ein variable Symbol, das einem Zeichen-String mit einem dafür gültigen Wert zugewiesen wurde.

expression_1: Stellt einen Wert dar, den der Assembler dem Symbol in dem Name-Feld zuweist**.** Expression_1 kann irgendeinen Wert annehmen, der für einen Assembler-Ausdruck erlaubt ist: absolute (einschließlich negativ), relocatable oder complex relocatable. Der Assembler verwendet diesen als 4 Byte-Wert. Symbole in expression_1 müssen nicht vorher definiert werden. Wenn jedoch ein Symbol nicht vorher definiert wird, ist der Wert von expression_1 dem Symbol in dem name-Feld zur Übersetzungszeit nicht zugewiesen und kann demzufolge während der Übersetzung nicht benutzt werden.

expression_2: Repräsentiert einen Wert, den der Assembler dem Symbol in dem name-Feld als "length attribute value" zuordnet. Der Wert ist optional, aber muss, wenn er angegeben wird, absolut in dem Bereich 0 ... 65.535 liegen. Er überschreibt das normale length-Attribut von expression_1. Alle Symbole, die in expression_2 erscheinen, müssen vorher definiert werden.

expression_3: Ist ein Wert, den der Assembler dem Symbol in dem name-Feld als "type attribute value" zuweist. Der Wert ist optional, wenn er aber angegeben wird, muss er einen absoluten Wert im Bereich 0 ... 255 haben. Er überschreibt das normale type-Attribut, das von expression_1 zugewiesen wird. Alle Symbole, die in expression_3 erscheinen, müssen vorher definiert sein.

Durch Betätigen der F3-Taste kehren wir zum vorherigen Bildschirm zurück. Unser Programm wird automatisch abgespeichert (saved).

```
  Menu  RefList  RefMode  Utilities  LMF  Workstation  Help
------------------------------------------------------------------------------
                              Edit Entry Panel            Member ASS02 saved

ISPF Library:
   Project . . . PRAKT20
   Group . . . . TEST     . . .          . . .          . . .
   Type  . . . . ASSEM
   Member  . . .                 (Blank or pattern for member selection list)

Other Partitioned or Sequential Data Set:
   Data Set Name . . .
   Volume Serial . . .           (If not cataloged)

Workstation File:
   File Name . . . . .
                                       Options
Initial Macro  . . . .                 /  Confirm Cancel/Move/Replace
Profile Name . . . . .                    Mixed Mode
Format Name  . . . . .                    Edit on Workstation
Data Set Password  . .                    Preserve VB record length

Command ===>
 F1=Help      F3=Exit     F10=Actions  F12=Cancel
```

Abbildung 8: "Edit Entry Panel"-Bildschirm

Rechts oben erscheint die Meldung, dass unser Member abgespeichert wurde (s. Abbildung 8).

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       PRAKT20.TEST.ASSEM(ASS02) - 01.04               Columns 00001 00072
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000100 ASS02    CSECT
000200 * Formalismus fuer Assembler-Code:
000300 * Angaben in den Stellen 1 - 71, Fortsetzungszeichen '*' in Stelle 72,
000400 * Optional: Nummerierung in Stellen 73 - 80
000500 * Kommentar-Zeile: '*' in Stelle 1 gefolgt vom Kommentar bis 71
000600 * Assemblercode-Zeile: Optional: Referenzname in 1 - 8
000700 *                       Zwingend: Operation ab 10
000800 *                       Optional: Parameter mit mind. 1 Blank hinter
000900 *                                 Operation, ueblich ab 16
001000 *                       Optional: Kommentar mit mind. 1 Blank hinter
001100 *                                 Parametern bis 71
001200 ASS02    AMODE 24    Adressierungsmodus: 24 Bits, max 16 MB
001300 ASS02    RMODE 24    Resident: Unterhalb 16 MB
001400 * Die Aufloesung der Makros ist nach der Umwandlung in der
001500 * Assembler-Liste zu sehen
001600          SAVEREG     Makro: Einstieg und Retten Register
Command ===>                                                 Scroll ===> PAGE
 F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abbildung 9: Assembler-Programm mit Zeilennummern

Für das Übersetzen, Verbinden und Ausführen des Assembler-Quellprogramms werden noch zwei weitere Programme benötigt: SAVEREG, EXITREG. Beide Macros werden in dem JCL-Script ASSSTA02 für &SYSUID.LIB (s. Abbildung 11) benötigt und vom Quellprogramm aufgerufen. SAVEREG sorgt für das Retten der Register und die Verkettung der Register-SAVEAREAs beim Programmaufruf. EXITREG stellt sicher, dass am Programmende das Restore der Register aus der verketteten Register-SAVEAREA und der Rücksprung in das aufrufende Programm (in diesem Fall das Betriebssystem) erfolgt. Beide Macros (s. Abbildung 17 - Abbildung 19 sowie Abbildung 20) müssen neben ASS02 als Members in den Dataset PRAKT20.TEST.ASSEM gestellt werden.

3. Erstellen und Ausführung des JCL-Scriptes

```
  Menu  RefList  RefMode  Utilities  LMF  Workstation  Help
--------------------------------------------------------------------------------
                              Edit Entry Panel

ISPF Library:
   Project . . . PRAKT20
   Group . . . . TEST     . . .          . . .          . . .
   Type  . . . . CNTL
   Member  . . . ASSSTA02        (Blank or pattern for member selection list)

Other Partitioned or Sequential Data Set:
   Data Set Name . . .
   Volume Serial . . .           (If not cataloged)

Workstation File:
   File Name . . . . .
                                       Options
Initial Macro  . . . .                 /  Confirm Cancel/Move/Replace
Profile Name . . . . .                    Mixed Mode
Format Name  . . . . .                    Edit on Workstation
Data Set Password  . .                    Preserve VB record length

Command ===>
 F1=Help      F3=Exit     F10=Actions  F12=Cancel
```

Abbildung 10: "Edit Entry Panel"-Bildschirm

Der nächste Schritt besteht im Editieren des JCL-Scripts ASSSTA02. Wir legen alle "Compile Scripts" als Members in dem von uns dafür vorgesehenen Partitioned Dataset PRAKT20.TEST.CNTL ab. ASSSTA02 dient dazu, unser Quell-Programm zu übersetzen, zu linken und das ausführbare Maschinenprogramm (Binary) abzuspeichern.

Wir geben als Typ "CNTL" sowie als Member "ASSSTA02" ein und betätigen die Eingabetaste (s. Abbildung 10).

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       PRAKT20.TEST.CNTL(ASSSTA02) - 01.03            Columns 00001 00072
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000010 //PRAKT20D JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID,
000020 //         REGION=4M
000030 //STEP1  EXEC ASMACL
000031 //C.SYSLIB DD
000040 //         DD  DSN=&SYSUID..TEST.ASSEM,DISP=SHR
000041 //C.SYSIN  DD  DSN=&SYSUID..TEST.ASSEM(ASS02),DISP=SHR
000050 //L.SYSLMOD DD DSN=&SYSUID..TEST.LOAD,DISP=SHR
000060 //L.SYSIN  DD  *
000070  NAME ASS02(R)
000080 /*
****** **************************** Bottom of Data ****************************




Command ===>                                                 Scroll ===> PAGE
 F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abbildung 11: JCL-Script

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       PRAKT20.TEST.CNTL(ASSSTA02) - 01.03            Columns 00001 00072
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000010 //PRAKT20D JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID,
000020 //         REGION=4M
000030 //STEP1  EXEC ASMACL
000031 //C.SYSLIB DD
000040 //         DD  DSN=&SYSUID..TEST.ASSEM,DISP=SHR
000041 //C.SYSIN  DD  DSN=&SYSUID..TEST.ASSEM(ASS02),DISP=SHR
000050 //L.SYSLMOD DD DSN=&SYSUID..TEST.LOAD,DISP=SHR
000060 //L.SYSIN  DD  *
000070  NAME ASS02(R)
000080 /*
****** **************************** Bottom of Data ****************************




Command ===> SUB                                             Scroll ===> PAGE
 F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abbildung 12: Ausführung des JCL-Scriptes

Unser Compile- und Link-Script kann nun ausgeführt werden. Wir geben, wie in Abbildung 12 gezeigt, auf der Kommandozeile "SUB" (für Submit) ein und betätigen die Eingabetaste.

TSO Subsystem	JES Subsystem	USS Subsystem	CICS Subsystem	DB2 Subsystem	andere .....
z/OS Kernel					

Das "**J**ob **E**ntry **S**ubsystem" (**JES**) des zOS-Betriebssystems dient dazu, Stapelverarbeitungsaufträge (Jobs) auf die einzelnen CPU's zu verteilen und der Reihe nach abzuarbeiten. Jobs werden dem "JES"-Subsystem in der Form von JCL-Scripten zugeführt, wobei deren erstes JCL-Statement ein JOB-Statement sein muß. PRAKT20.TEST.CNTL(ASSSTA02) ist ein derartiges Script. Das Kommando "SUB" (Submit) bewirkt, dass PRAKT20.TEST.CNTL(ASSSTA02) in die Warteschlange der von JES abzuarbeitenden Aufträge eingereiht wird.

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT       PRAKT20.TEST.CNTL(ASSSTA02) - 01.03            Columns 00001 00072
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000010 //PRAKT20D JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID,
000020 //         REGION=4M
000030 //STEP1  EXEC ASMACL
000031 //C.SYSLIB DD
000040 //         DD  DSN=&SYSUID..TEST.ASSEM,DISP=SHR
000041 //C.SYSIN  DD  DSN=&SYSUID..TEST.ASSEM(ASS02),DISP=SHR
000050 //L.SYSLMOD DD DSN=&SYSUID..TEST.LOAD,DISP=SHR
000060 //L.SYSIN  DD  *
000070  NAME ASS02(R)
000080 /*
****** **************************** Bottom of Data ****************************


IKJ56250I JOB PRAKT20D(JOB08337) SUBMITTED
***
```

Abbildung 13: Meldung "JOB PRAKT20C(JOB05141) SUBMITTED"

Der JCL-Kommando-Interpreter überprüft die Syntax des Scripts. Falls er keinen Fehler findet, übergibt (submitted) er den Job zur Abarbeitung an das JES-Subsystem. Die Meldung oberhalb der Kommandozeile besagt, dass dies hier der Fall ist (s. Abbildung 13). Der Job erhält die Nummer 08337. Diese Nummer kann z.B. vom Systemadministrator benutzt werden, um den Status der Verarbeitung dieses Jobs abzufragen.

Wir warten einige Sekunden und betätigen anschließend die Eingabetaste. Erscheint keine Meldung, hat JES das JCL-Script noch nicht endgültig abgearbeitet. Wir warten erneut einige Sekunden und betätigen die Eingabetaste; wir wiederholen dies notfalls mehrfach, bis eine Statusmeldung, so ähnlich wie in Abbildung 14 dargestellt ist, ausgegeben wird.

```
14.17.47 JOB08337 $HASP165 PRAKT20D ENDED AT N1  MAXCC=0 CN(INTERNAL)
***
```

Abbildung 14: Statusmeldung nach Abarbeitung des JCL-Scriptes

"MAXCC-0" ist eine Erfolgsmeldung: Die Übersetzung ist erfolgreich durchgeführt worden. "MAXCC-4" ist ebenfalls OK, alles andere besagt, dass ein Fehler aufgetreten ist. Das übersetzte Programm ist nun ausführungsfertig in dem File

PRAKT20.TEST.LOAD(ASS02) abgespeichert.

4. Ausführung des Assembler-Programms

```
  Menu  Utilities  Compilers  Options  Status  Help
--------------------------------------------------------------------------------
                          ISPF Primary Option Menu

0  Settings      Terminal and user parameters           User ID . : PRAKT20
1  View          Display source data or listings        Time. . . : 14:27
2  Edit          Create or change source data           Terminal. : 3278
3  Utilities     Perform utility functions              Screen. . : 1
4  Foreground    Interactive language processing        Language. : ENGLISH
5  Batch         Submit job for language processing     Appl ID . : PDF
6  Command       Enter TSO or Workstation commands      TSO logon : IKJACCNT
7  Dialog Test   Perform dialog testing                 TSO prefix: PRAKT20
8  LM Facility   Library administrator functions        System ID : DAVI
9  IBM Products  IBM program development products       MVS acct. : ACCT#
10 SCLM          SW Configuration Library Manager       Release . : ISPF 4.5
11 Workplace     ISPF Object/Action Workplace

     Enter X to Terminate using log/list defaults


Option ===>
 F1=Help      F3=Exit     F10=Actions  F12=Cancel
```

Abbildung 15: "ISPF Primary Option Menu"-Bildschirm

Wir sind nun soweit, dass unser Programm ausgeführt werden kann. Durch mehrfaches Betätigen der F3-Taste kehren wir in das "ISPF Primary Option Menu" zurück (s. Abbildung 15). Auf der Kommandozeile geben wir den Befehl

tso call 'prakt20.test.load(ass02)'

ein und betätigen die Eingabetaste. "prakt20.test.load(ass02)" enthält das vom Compiler erzeugte Maschinenprogramm. "call" ist ein TSO-Kommando und ruft ein Programm auf. Wir sind aber im ISPF-Subsystem und nicht im TSO-Subsystem. "tso call" an Stelle von "call" bewirkt, dass der "call"-Befehl auch innerhalb des ISPF-Subsystems aufgerufen werden kann.

Wichtiger Hinweis:

Achten Sie darauf, daß Sie bei dem Befehl "tso call 'prakt20.test.load(ass02)'" die richtigen Hochkommas verwenden. Das Hochkomma, das auf den meisten Tastaturen über dem Zeichen "#" steht, ist das korrekte.

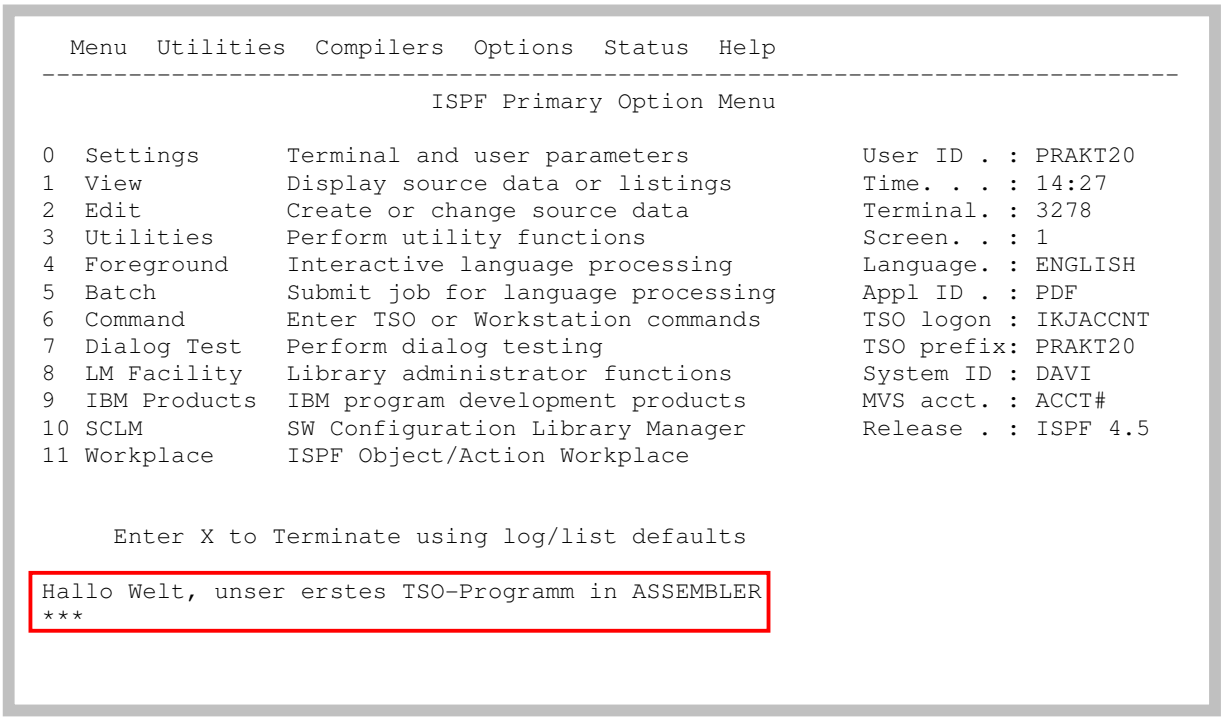

Abbildung 16: Ausgabe unseres Assembler-Programms

Abbildung 16 zeigt: Oberhalb der Kommandozeile erscheint die Ausgabe unseres Assembler-Programms.

Wir nehmen an, Ihnen fallen jetzt viele Möglichkeiten ein, ein aussagefähigeres Assembler-Programm zu schreiben. Sie können ein neues Quellprogramm PRAKT20.TEST.ASSEM(ASSxx) schreiben und hierfür ein neues JCL-Script PRAKT20.TEST.CNTL(ASSSTAxx) erzeugen,

Aufgabe: *Verfassen Sie ein eigenes funktionsfähiges Assembler-Programm (keine Modifikation des vorgegebenen Hallo-Welt-Programms) und legen Sie den Quellkode in PRAKT20.TEST.ASSEM(ASSxx) ab. Das angepaßte JCL-Script legen Sie bitte in PRAKT20.TEST.CNTL(ASSSTAxx) ab ("PRAKT20" ist bei beiden Datasets durch Ihre Benutzer-ID zu ersetzen). Erstellen Sie Print-Screens der ISPF-Fenster mit dem vollständigen Quellkode Ihres Programms sowie einen Print-Screen Ihres ISPF-Fensters mit der Ausgabe Ihres Programms. Schicken Sie die Print-Screens im Bitmap- oder JPEG-Format (pro Bild maximal 250 KByte) an die Mailadresse Ihres Betreuers.*

Anhang : Der Programmcode der Macros

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
VIEW       PRAKT20.TEST.ASSEM(SAVEREG) - 01.02            Columns 00001 00072
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000100          MACRO
000200          SAVEREG
000300 * REGISTERS AND EQUATES   ****************************************
000400 FIRST    EQU   *
000500 R0       EQU   0     REGISTERBEZEICHNUNGEN 0 - 15
000600 R1       EQU   1     WERDEN MIT R0 - R15 GLEICHGESETZT
000700 R2       EQU   2
000800 R3       EQU   3
000900 R4       EQU   4
001000 R5       EQU   5
001100 R6       EQU   6
001200 R7       EQU   7
001300 R8       EQU   8
001400 R9       EQU   9
001500 R10      EQU   10
001600 R11      EQU   11
Command ===>                                                Scroll ===> PAGE
 F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abbildung 17: Programmcode des Macros SAVEREG (Panel 1/3)

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
VIEW       PRAKT20.TEST.ASSEM(SAVEREG) - 01.02            Columns 00001 00072
001700 R12      EQU   12
001800 R13      EQU   13
001900 R14      EQU   14
002000 R15      EQU   15
002100 * STANDARD ENTRY PROCEDUR ****************************************
002200 * LINKAGE CONVENTIONEN IM MVS:
002300 *    DAS RUFENDE PROGRAMM STELLT ENTRY-POINT DES GERUFENEN PROGRAMMS
002400 *        IN REG 15
002500 *    DAS RUFENDE PROGRAMM STELLT ADR. DER EIGENEN SAVEAAREA IN REG 13
002600 *    DAS RUFENDE PROGRAMM STELLT DIE EIGENE RUECKSPRUNGADR. IN REG 14
002700 *    DAS RUFENDE PROGRAMM STELLT ADR. VON ZU UEBERGEBENDEN PARAMETERN
002800 *        IN REG 1
002900 *
003000 *    DAS RUFENDE PROGRAMM KANN DAS OPERATINGSYSTEM SEIN ÜÜ
003100 *
003200          STM   R14,R12,12(R13) STORE ALLE REGISTER AUSSER 13 IN DIE
003300 *                               SAVEAREA+12 DES RUFENDEN PROGRAMMS
003400          LR    R12,R15    LADE ENTRY-POINT DIESES PROGRAMMS NACH REG 12
003500          USING FIRST,R12  BENUTZE REG 12 ALS BASISREGISTER VOM BEGINN
Command ===>                                                Scroll ===> PAGE
 F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abbildung 18: Programmcode des Macros SAVEREG (Panel 2/3)

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
VIEW       PRAKT20.TEST.ASSEM(SAVEREG) - 01.02            Columns 00001 00072
003600 *                          DIES IST EINE ANWEISUNG AN DEN ASSEMBLER
003700           ST    R13,SAVE+4 STORE REG 13 (ADRESSE DER SAVEAEREA DES
003800 *                          RUFENDEN PROGRAMMS) IN DIE EIGENE SAVEAEREA+4
003900           LR    R15,R13    RETTE REG 13 NACH REG 15
004000           LA    R13,SAVE   LADE ADR. DER EIGENEN SAVEAEREA NACH REG 13
004100           ST    R13,8(R15) STORE REG 13 IN SAVEAEREA DES RUFENDEN
004200 *                          PROGRAMMS
004300 *                          JETZT SIND DIE SAVEAEREAS DES RUFENDEN UND
004400 *                          DES GERUFENEN (DIESES) PROGRAMMS VERKETTET
004500           LR    R2,R1      RETTE REG 1 (ADR. PARAMETER, FALLS
004600 *                          VORHANDEN), DA REG 1 BEI MACROS BENUTZT WIRD
004700           B     ANFANG     UEBERSPRINGEN EYE-CATCHER UND SAVEAREA
004800           DC    CL16'SAVEAREA'    EYE-CATCHER FUER DUMP
004900 SAVE      DC    18F'0'     SAVEAEREA
005000 ANFANG    EQU   *
005100           MEND
****** **************************** Bottom of Data ****************************


Command ===>                                                  Scroll ===> PAGE
 F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abbildung 19: Programmcode des Macros SAVEREG (Panel 3/3)

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
VIEW       PRAKT20.TEST.ASSEM(EXITREG) - 01.02             Columns 00001 00072
****** **************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000100           MACRO
000200           EXITREG
000300 * STANDARD EXIT  PROCEDUR ****************************************
000400           L     R13,SAVE+4 LADE ADR. SAVEAREA DES RUFENDEN PROGRAMMS
000500           L     R14,12(R13) LADE REG 14 MIT INHALT BEI AUFRUF
000600           LM    R0,R12,20(R13) LADE REGS 0 BIS 12 MIT INHALT BEI AUFRUF
000700 *                          DIESES PROGRAMS, REG 15 ENTHAELT
000800 *                          RETURN-CODE AUS DIESEM PROGRAMM
000900           BR    R14        SPRING ZUR RUECKSPRUNGADR. DES RUFENDEN
001000 *                          PROGRAMMS. DIES KANN DAS OPERATINGSYSTEM SEIN
001100           MEND
****** **************************** Bottom of Data ****************************




Command ===>                                                  Scroll ===> PAGE
 F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abbildung 20: Programmcode des Macros EXITREG