Wolfgang Renner
Gimbacher Weg 34
D-65779 Kelkheim
www.institut-wolfgang-renner.de
institut-wolfgang-renner@t-online.de
Telefon:
06195 4161

Kelkheim den 03. August 2015

Projektbeschreibung: „Doktorarbeit Zwo“ (Zweiter Versuch)

Eine Erweiterung der X86 - Prozessorarchitektur auf 128 Bit. Diese ermöglicht eine X86-32|64|128|256|512 harmonisierte Assemblerprogrammierung, die auch Basis einer kleinen Dozententätigkeit an der Goethe Universität Frankfurt sein soll.

In dieser Datei beschreibe ich den X86-32|64|128|256|512 Bit Prozessor mit Registersatz, Adressierungsmoden, Assemblerquellcodestil und weiteren technischen Beschreibungen. Mit dieser Datei wird zum einen mein DspLib-Projekt tiefergehend beschrieben und zum anderen soll sie als Grundlage für einen Dialog mit Intel und AMD sowie der Universität Frankfurt, dem BMBF und dem Hessischen Kultusministerium über eine Anstellung als Doktorand mit selbstgestellter Thematik und selbstbestimmter Lehrmöglichkeit dienen.

Inhaltsverzeichnis

1 Einleitung

Der vorliegende Text ist lediglich ein Provisorium ! Gesetztenfalls ich bekäme eine Doktorandenstelle, welche kompatibel mit der fortgeführten Projektentwicklung vom Bildungsunternehmen „Schloss Fünfeck“ wäre, dann würde dieser Text zu einer vollwertigeren Doktorarbeit ausgebaut werden. Dieser Text soll dann Grundlage einer kleinen Lehrtätigkeit an der Universität sein und wäre ein Manual zur konsistenten Assembler-Programmierung von X86 Prozessoren für den 32|64|128||256|512-Bit Modus. Der Modus 128 könnte in einer bald kommenden Neuauflage vom X86 Prozessor erscheinen. Das muss Intel oder AMD entscheiden. Ich kann nur einen Erweiterungsvorschlag formulieren. Der Modus 256 und 512 passt problemlos in die hier vorgestellte Systematik. Jedoch habe ich bei DspLib für Pointer usw. lediglich 16 Bytes in Strukturen vorgesehen. Für 256 bzw. 512 Bit Registergröße müssten dann jeweils 32 oder 64 Bytes pro Pointer usw. eingerichtet werden. Die Erweiterung auf 128 Bit macht noch echten Sinn. Darüberhinaus dürften erst irgendwelche neuartigen „Quantencomputer“ derartig riesige Adressräume usw. rechtfertigen können.

1.1 Vorgeschichte

Nachdem mein erster Versuch 1990 eine Doktorarbeit zu schreiben, aufgrund verschiedenartiger Ursachen ein krankheitsbedingtes vorzeitiges Ende nahm, habe ich mich dann letztlich in eine Privatlehrertätigkeit zurückgezogen und begleitend meine DspLib-Bibliothek entwickelt. Der Anfang war 1988 eine „AMIGA“-Funktion, die einem Fortran-Compiler mit M68000 Assemblerquellcode beilag. Diese Funktion ermöglichte den Zugriff vom Fortran auf die C-Style Amiga-Betriebssystemfunktionen. Ich habe 1980 als Schüler in der Odenwaldschule auf einem Apple II Computer in Basic programmieren gelernt. Es gab da so Craigs, die schnell bewegte Spiele in Assembler programmierten. Da konnte ich nicht mithalten und wollte auch immer die höheren Weihen der superschnellen Assemblerprogrammierung erreichen. Als ich mir 1987 einen Amiga kaufte und das DspLib Projekt seine Anregung fand, lernte ich nun die Assemblerprogrammierung, um aus der AMIGA-Funktion eine wesentlich aufgewertete Programmierschnittstelle zu machen. Um 1990 war das Programmieren in Assembler noch immer, für hochperformante Software, eine gute Empfehlung. Durch den Einzug der Hochleistungsgraphikkarten, fand diese Motivation ein Ende. Moderne Spiele werden in Standard C programmiert. Es werden nur noch Graphikkartenfunktionen aufgerufen, die dann die eigentliche Bilderzeugung schneller machen, als jeder noch so gute Assemblercode für den Hauptprozessor.

Ich hatte ab 1998 meinen M68000 Amiga-DspLib Code auf einen X86-32 Windows PC portiert. Dann habe ich eine X86-64 Erweiterung berücksichtigt. Noch ein wenig später habe ich auch noch eine X86-128 Erweiterung in den von mir formulierten Assemblerstil eingeführt. DspLib ist nun in einem Stil geschrieben, der die Übersetzung in den X86-32, X86-64 und X86-128 Prozessormodus ohne Quellcodeänderung harmonisch ermöglichen können müsste. „Müsste“ deswegen, da ich bis zum heutigen Tage auf einem veralteten X86-32 Pentium-III-Einzelkern PC arbeite. Ein Wechsel auf einen modernen Multikern X86-64 PC geht erst, wenn mein Code komplett Threadsafe ist. Das heißt, alle Threads mit ihren Speicherzugriffsrechten sauber mit den dafür vorgesehenen Windows Systemfunktionen getrennt sind. Daran arbeite ich mit Hochdruck. Bei diesem Durchfilzen wird wiederum der ganze Aufbau modernisiert. Das kostet Zeit ohne Ende. Einen neuen PC kann ich dabei vorerst überhaupt nicht einsetzen.

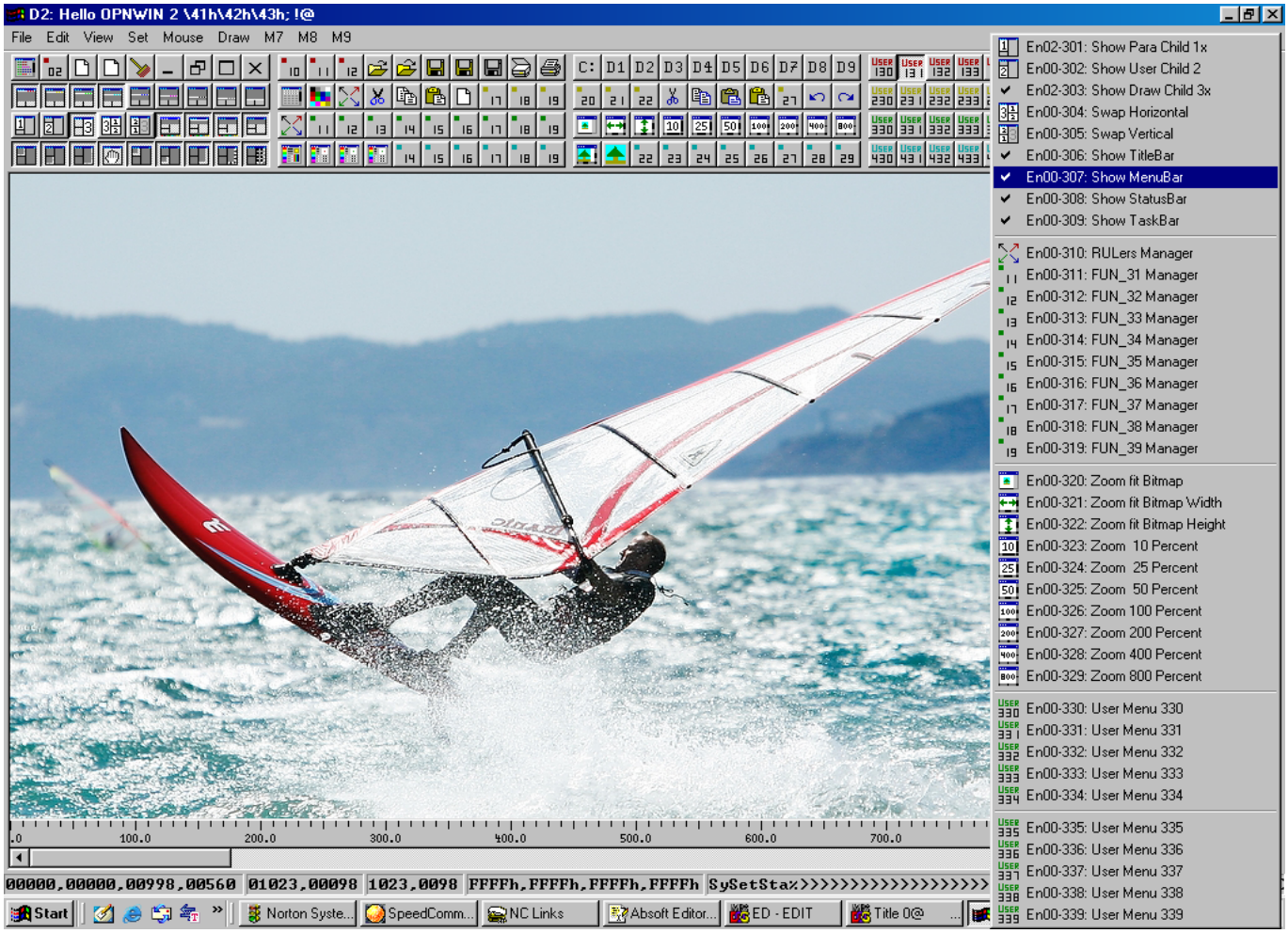
Meine DspLib-Bibliothek ermöglicht numerische Operationen mit Millionen von Stellen falls erwünscht. Diese „Extended Precision“ Operationen kann man nur in Assembler vernünftig programmieren. Da müssen beispielsweise Carry-Flags korrekt weitergegeben werden. Und da muss um jeden Prozessortakt gekämpft werden. Die graphische Bildbearbeitungsschnittstelle von DspLib beherrscht bis zu acht Farbkanäle mit 8, 16 und 32 Bit/Kanal, sowie zusätzlich transparent-monochrome Overlays und weitere Hilfslayer. Es mag sein, das moderne Graphikkarten solche Operationen hardwaremäßig anbieten. Hat man eine Software-Alternative, dann kann die Funktionalität auch auf weniger guten PC's benutzt werden, wenn auch langsamer. In diesem Fall kommt doch noch mal die klassische Bildbearbeitung mit Assembler-Bit-Popeleien sinnvoll zur Anwendung. Große Abschnitte von DspLib würde ich nicht gerne in C programmieren müssen. Für viele dieser Konstruktionen ist die Assemblerprogrammierung definitiv überlegen.

Das größte Problem der Assembler-Programmierung ist die Prozessorabhängigkeit und damit nicht vorhandene Portierbarkeit. Aus Sicht der Assemblerprogrammierer ist es außerordentlich unerwünscht, wenn ständig neue Prozessorarchitekturen „durch die Gassen getrieben werden“. Deshalb sind Alternativen wie der AlphaPC von DEC, Intels IA64 (Intanium), Moderolas M604 PowerPC Prozessoren usw. nicht sehr beliebt gewesen. Der hohe Marktanteil von X86-Computern und der kompatiblen Weiterentwicklung sind da wesentlich angenehmer und werterhaltender als komplette Neuformulierungen. Der Nachteil ist, dass der X86-Prozessor wie ein gewachsenes mittelalterliches Dorf viele parallele Funktionen besitzt (z.B. Fließkommazahlen in X87, MMX und XMM), die alle auf dem Chip berücksichtigt sein müssen, um die Rückwärtskompatibilität zu sichern. Das kostet insbesondere viel Strom, wodurch der X86 auf Mobilgeräten gegenüber von Alternativen (ARM-Architektur) das Nachsehen hatte.

Mein wissenschaftlich orientiertes Programmiersystem ist nichts, was typischerweise auf Handys genutzt wird. Bei netzgespeisten PC's sind einige Watt mehr Stromverbrauch unkritisch und dort konnte der X86-Prozessor seine Vorherrschaft behaupten. Mit dem Vorschlag den X86 auf 128 Bit zu erweitern, wird Neuland betreten. In dieser vorliegenden Ausarbeitung gebe ich eine Empfehlung, die eine X86-128 Bit Architektur definiert, die sich problemlos weiter zu einem X86-256 und X86-512 Prozessor ausbauen liese. Eine X86-128 Architektur spricht insbesondere wissenschaftliche Supercomputer an. Auch normale Endverbraucher PC's mit dem hier vorgestelltem 128 Bit Modus, dürfte Computerfreaks usw. interessieren. Handys werden damit sicherlich nicht bestückt.

1.2 Struktur vom X86-16|32|64

Die Entwicklung vom X86-Prozessor begann irgendwann um 1970 mit einer 16 Bit Variante. Um 1990 kam dann eine 32 Bit Erweiterung die schließlich um 2005 auf 64 Bit aufgebohrt wurde. Gesetztenfalls aus diesem Text wird eine Doktorarbeit, dann wird ein genauerer Rückblick hier erscheinen. Gegenwärtig ist dieser Text lediglich ein Teil der Projektbeschreibung für meine „Schloss Fünfeck“ Projekt-CD, mit der Zielsetzung einen Dialog zu meinem Fall endlich in die Wege zu leiten.



1.3 Struktur vom X86-128|256|512

Mein Programmiersystem „DspLib“ für Fortran und C auf X86-Computern ist komplett in Assembler geschrieben. Um einen längerfristigen Fortbestand zu ermöglichen, berücksichtigte ich eine X86-128 Variante, die sich zu X86-32|64|128 Bit Modulen aus dem gleichen Assemblerquellcode compilieren lassen müsste. Beim Aufsetzen des vorliegenden Textes musste ich mir darüber Rechenschaft geben, wie denn der beschriebene X86-128 vom Befehlscode her funktionieren soll. Nochmal mit irgendwelchen Prefixen zu patchen (wie beim X86-16|32|64) halte ich für unangebracht. Besser wird der Bezug der Assemblerbefehle zum binären Objektmodul komplett neu spezifiziert:

Gegenüber sind einige Zeilen Assemblercode für den X86-512 Modus mit einem die Übersetzung beschreibenden Listing. Das Arbeitsregister EAX heißt je nach Länge AH, AL, AX, EAX, RAX; SAX, TAX, PAX. Allgemein bedeutet LAX, die größte Länge im benutzten Modus. Im X86-128 Modus wird also aus LAX genau genommen SAX. Das wird in Preprocessing-Includedateien für den jeweiligen Modus definiert. Als Speichergrößendefinition wird BVAL, WVAL, EVAL (DVAL ?), RVAL, SVAL, TVAL, PVAL bzw. LVAL benutzt. Als Kurzform geht auch BV, WV, EV (DV ?), RV, SV, TV, PV bzw. LV. Zur Spezifikation vom „Addressing-Displacement“ kann BDIS, WDIS, EDIS, RDIS, SDIS, TDIS, PDIS bzw. LDIS (BD, WD, ED, RD, SD, TD, PD bzw. LD) benutzt werden. Üblicherweise wählt der Assembler die kürzest mögliche Displacementkonstante automatisch aus. Eine manuelle Festlegung ist zwar möglich aber überflüssig. Genauso verhält es sich mit den Konstanten: Diese können mit BCST, WCST, ECST (DCST ?), RCST, SCST, TCST, PCST bzw. LCST (BC, WC, EC (DC ?), RC, SC, TC, PC bzw. LC) auf eine gewünschte Länge gebracht werden. Falls zu klein gewählt, entsteht eine Fehlermeldung beim Assemblieren. Auch hier ist die automatische Wahl des Kürzesten der übliche Standard. Der neue X86-128 Modus erlaubt als „Scaling-Factor“ die Werte 10, 1, 2, 4, 8, 16, 32 und 64. Es gibt 32+1 Integerregister: LAX, LDX, LCX, LDI, LSI, LBX, LBP, LSP, L08, L09, L10, . . . L30, L31 und LIP. Ebenso gibt es 32 XMM Register.

Wie später genauer erklärt, werden die Adressierungsformen von den vier Bytes A0, A1, A2 und A3 beschrieben. Dort passen die 32 Register mit der umfangreichen Variation an Größen, Displacements und Zahlenwert-Konstanten in einfacher Systematik rein. Die klassischen Adressierungsformen sind einfach eine Teilmenge der vielfältigen neuen Möglichkeiten. Auch die binäre Darstellung der eigentlichen Operationen wird komplett neu definiert: Es gibt bis zu vier Bytes (P0, P1, P2 und P3), die eine Hufmann-Struktur wie bei dem Unicode UTF-08 System bilden. Häufige und einfache Opcodes mögen mit einem Byte beschrieben sein. Seltenere und kompliziertere Opcodes bekommen zusätzliche Bytes. Die Auswahl geht ähnlich wie bei normalen Kompressionsalgorithmen, wo eine Häufigkeitsanalyse die Zuordnung beeinflusst. Wenn man das sorgfältig macht, dann ist dass eine komplexe Aufgabe, die nur in Zusammenarbeit mit Intel und AMD in guter Qualität gelöst werden könnte. Es würde auch vom Arbeitsaufwand jenseits meiner biographischen Reichweite stehen.

Der Assembler-Quellcode sieht aus wie der bisherige Code ! Gerade um X86-128 Code auch für X86-64 bzw. X86-32 kompatibel zu machen, muss das explizite Anwenden der neuen Erweiterungen vermieden werden: Es gibt nur die ersten acht Register LAX, LDX, LCX, LDI, LSI, LBX, LBP und LSP sowie LIP. Die Register L08,...,L31 sind für den 32 Bit Modus verboten. Ebenso L16,...,L31 im 64 Bit Modus (Sowie im 1024 Bit Modus). Ein Register das zu groß spezifiziert wurde, z.B. RAX im 32 Bit Modus, wird auf EAX beim Assemblieren reduziert. Wird es für den 128 Bit Modus assembliert, dann bleibt es der 64 Bit Registerausschnitt RAX. Um im Assemblerlisting alle Werte darstellen zu können, muss eine mehrzeilige Darstellung gewählt werden: In der ersten Zeile ist der Offset vom Opcode in dezimaler und hexadezimaler Form als Erstes zu sehen. Dann kommen bis zu vier Opcode-Bytes und danach bis zu vier Addressing-Bytes. Schließlich kommt der aktive Assembler Quellcode Text. In der zweiten Zeile erscheint das Displacement. Meistens gibt's das garnicht. Oder es hat nur wenige Bytes. Das gleiche gilt für die Zahlenwertkonstante, die in der dritten Zeile mit dem Kommentarende erscheint. Sollte das Displacement oder die Konstante mehr als 16 Bytes beanspruchen, dann werden weitere Zeilen eingefügt. Danach wird eine Trennlinie zum nächsten Opcode eingefügt. Dieser Listingaufbau geht bis zum X86-512 ohne Probleme !

```
;#########################################################################################################################
LabelXMM: pmov123456 SVAL[ldx+lcx*16+BDIS 012345678h],xs12     ; 12345678901234567890123456789012345678901234567
;=========================================================================================================================
Label000: mov        LVAL[ldx+lcx*10],eax                      ; Kommentar (0)
Label001: mov        BVAL[ldx+lcx*01+BDIS 12h],BCST 000000013h ; Kommentar (1)
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
Label002: mov        WVAL[ldx+lcx*02+WDIS 12h],WCST 000000013h ; Kommentar (2)
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
Label003: mov        EVAL[ldx+lcx*04+EDIS 12h],ECST 000000013h ; Kommentar (3)
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
Label004: mov        RVAL[ldx+lcx*08+RDIS 12h],RCST 000000013h ; Kommentar (4)
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Label005: mov        SVAL[ldx+lcx*16+SDIS 12h],SCST 000000013h ; Kommentar (5)
Label006: mov        TVAL[ldx+lcx*32+TDIS 12h],TCST 000000013h ; Kommentar (6)
Label007: mov        PVAL[ldx+lcx*64+PDIS 12h],PCST 000000013h ; Kommentar (7)
Label008: mov        LVAL[ldx+lcx*64+LDIS 12h],LCST 000000013h ; Kommentar (8)
;*************************************************************************************************************************
;1234567890 44556677 P0P1P2P3 A0A1A2A3 [OfOpsAd] Aktiver Quellcode vor dem Kommentarzeichen (;)
;D0 D1 D2D3 D4D5D6D7 D8D9DADB DCDDDEDF [Dis0-15]
;C0 C1 C2C3 C4C5C6C7 C8C9CACB CCCDCECF [Cst0-15] Beschreibungen  nach  dem Kommentarzeichen (;)
;=========================================================================================================================
 0000000000 00000000 P0P1     E7600102 [OfOpsAd] Label000: mov        LVAL[ldx+lcx*10],eax
                                       [Dis0-15] ; Kommentar (0)

 0000000006 00000006 P0P1     252?2122 [OfOpsAd] Label001: mov        BVAL[ldx+lcx*01+BDIS 12h],BCST 000000013h
 12                                    [Dis0-15]
 13                                    [Cst0-15] ; Kommentar (1)
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
 0000000014 0000000E P0P1     454?4142 [OffOpAd] Label002: mov        WVAL[ldx+lcx*02+WDIS 12h],WCST 000000013h
 12 00                                 [Dis0-15]
 13 00                                 [Cst0-15] ; Kommentar (2)
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
 0000000024 00000018 P0P1     656?6162 [OfOpsAd] Label003: mov        EVAL[ldx+lcx*04+EDIS 12h],ECST 000000013h
 12 00 0000                            [Dis0-15]
 13 00 0000                            [Cst0-15] ; Kommentar (3)
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
 0000000038 00000026 P0P1     858?8182 [OffOpAd] Label004: mov        RVAL[ldx+lcx*08+RDIS 12h],RCST 000000013h
 12 00 0000 00000000                   [Dis0-15]
 13 00 0000 00000000                   [Cst0-15] ; Kommentar (4)
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 0000000060 0000003C P0P1     A5A?A1A2 [OffOpAd] Label005: mov        SVAL[ldx+lcx*16+SDIS 12h],SCST 000000013h
 12 00 0000 00000000 00000000 00000000 [Dis0-15]
 13 00 0000 00000000 00000000 00000000 [Cst0-15] ; Kommentar (5)

 0000000098 00000062 P0P1     C5C?C1C2 [OffOpAd] Label006: mov        TVAL[ldx+lcx*32+TDIS 12h],TCST 000000013h
 12 00 0000 00000000 00000000 00000000 [Dis0-15]
 00 00 0000 00000000 00000000 00000000 [Di16-31]
 13 00 0000 00000000 00000000 00000000 [Cst0-15] ; Kommentar (6)
 00 00 0000 00000000 00000000 00000000 [Cs16-31]

 0000000168 000000A8 P0P1     E5E?E1E2 [OffOpAd] Label007: mov        PVAL[ldx+lcx*64+PDIS 12h],PCST 000000013h
 12 00 0000 00000000 00000000 00000000 [Dis0-15]
 00 00 0000 00000000 00000000 00000000 [Di16-31]
 00 00 0000 00000000 00000000 00000000 [Di32-47]
 00 00 0000 00000000 00000000 00000000 [Di48-63]
 13 00 0000 00000000 00000000 00000000 [Cst0-15] ; Kommentar (7)
 00 00 0000 00000000 00000000 00000000 [Cs16-31]
 00 00 0000 00000000 00000000 00000000 [Cs32-47]
 00 00 0000 00000000 00000000 00000000 [Cs48-63]
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Label008: mov        LVAL[ldx+lcx*64+LDIS 12h],LCST 000000013h ; Kommentar (8)
12345678901234567890123456789012345678901234567890123456789012_;_12345678901234567890123456789012345678901234567
;=========================================================================================================================
;1234567890 44556677 P0P1P2P3 A0A1A2A3 [OfOpsAd] 12345678901234567890123456789012345678901234567890123456789012_
 D0 D1 D2D3 D4D5D6D7 D8D9DADB DCDDDEDF [Dis0-15]
 D0 D1 D2D3 D4D5D6D7 D8D9DADB DCDDDEDF [Di16-31]
 C0 C1 C2C3 C4C5C6C7 C8C9CACB CCCDCECF [Cst0-15] ;_12345678901234567890123456789012345678901234567 ( Kommentar )
 C0 C1 C2C3 C4C5C6C7 C8C9CACB CCCDCECF [Cs16-31]
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;2345678_123456789012345  equ  12345678_123456789012345+LS*000 ; #VAL: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;=========================================================================================================================
 -12345678901234567890123456789012345678901234567890 [Dec] 12345678_123456789012345  equ  12345678_123456789012345+LS*000
  1111 2222 3333 4444 5555 6666 7777 8888h [Hex] ; #VAL: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
 -12345678901234567890123456789012345678901234567890 [Dec] 12345678_123456789012345  equ  12345678_123456789012345+LS*000
  12345678901234567890123456789012345678901234567890 [D16]
  1111 2222 3333 4444 5555 6666 7777 8888h [Hex] ; #VAL: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
  1111 2222 3333 4444 5555 6666 7777 8888h [H16]
```

1.4 Struktur vom X86-1024|2048|4096|8192|16384|32768|65536|131072

Die Maschinenbefehle vom X86 Prozessor bestehen typischerweise aus einem Registeroperand und einem Hauptspeicheroperand. Dabei kann der Zieloperand oder der unveränderte Quelloperand das direkte Register sein. Das Register muss ausgewählt werden und hat eine frei wählbare Größe (Size). Bei der X86-32 Architektur gibt es 8 Arbeitsregister (EAX,EDX,ECX,EDI,ESI,EBX,EBP,ESP). Bei der X86-64 Architektur gibt es 16 Arbeitsregister (RAX,RDX,..,RBP,RSP;R08,R09,..,R14,R15).

Bei meinem Vorschlag für einen X86-128 gibt es 32 Arbeitsregister (SAX,SDX,..,S30,S31), die auch als 256 und 512 Bit Variante verfügbar wären. Für einen X86-1024 müsste wiederum eine neue Architektur begonnen werden. Das Adressbyte (A0) = <MSz|Acces> ist obligatorisch und enthält meistens die Speicheroperandengröße <MSz| und eine Zugriffsformenauswahl |Acces>. Das Adressbyte [A1] := <RSz|RRRRR> ist optional und enthält die Spezifikationen vom Registeroperanden. Das Adressbyte [A2] := <DSz|BBBBB> ist optional und enthält eine Displacementgröße (für die Displacementbytes im Codemodul) und das Basisregister für den Basispunkt vom Hauptspeicherzugriff. Das Adressbyte [A3] := <Scl|JJJJJ> ist optional und enthält einen Scalingfactorselector und das Indexregister für ein bestimmtes Vektorelement. Bei der X86-128 Architektur gibt es jeweils 3 Bits für die Größe und 5 Bits für die Registerauswahl. Bei der X86-1024 Architektur bräuchten wir jeweils 4 Bits für die Größe. Damit verblieben nur noch 4 Bits für die Registerauswahl.

```
;*****************************************************************************************************************
;*                                                                                                               *
;* +--------+--------------------+--------------------+--------------------+--------------------+--------+ *
;* | <Size| | 0000 0001 0010 0011 : 0100 0101 0110 0111 | 1000 1001 1010 1011 : 1100 1101 1110 1111 | General| *
;* |  Bits  |    8    8   16   32 :   64  128  256  512 | 1024 2048 4096 8192 : 2^14 2^15 2^16 2^17 | Long   | *
;* |  Bytes |    1    1    2    4 :    8   16   32   64 |  128  256  512 1024 : 2048 4096 8192 2^14 | Regist | *
;* |--------+--------------------+--------------------+--------------------+--------------------+--------+ *
;* | |RRRR> |   AH   AL   AX  EAX :  RAX  SAX  TAX  PAX | K0AX K1AX K2AX K3AX : K4AX K5AX K6AX K7AX | LAX 00 | *
;* | |0001> |   DH   DL   DX  EDX :  RDX  SDX  TDX  PDX | K0DX K1DX K2DX K3DX : K4DX K5DX K6DX K7DX | LDX 01 | *
;* | |0010> |   CH   CL   CX  ECX :  RCX  SCX  TCX  PCX | K0CX K1CX K2CX K3CX : K4CX K5CX K6CX K7CX | LCX 02 | *
;* |  -   - | -   -   -   -   -   :   -   -   -   -   - | -   -   -   -   -   :   -   -   -   -   - | -   -- | *
;* | |0011> |  H03  B03   DI  EDI :  RDI  SDI  TDI  PDI | K0DI K1DI K2DI K3DI : K4DI K5DI K6DI K7DI | LDI 03 | *
;* | |0100> |  H04  B05   SI  ESI :  RSI  SSI  TSI  PSI | K0SI K1SI K2SI K3SI : K4SI K5SI K6SI K7SI | LSI 04 | *
;* | |0101> |   BH   BL   BX  EBX :  RBX  SBX  TBX  PBX | K0BX K1BX K2BX K3BX : K4BX K5BX K6BX K7BX | LBX 05 | *
;* | |0110> |  H06  B06   BP  EBP :  RBP  SBP  TBP  PBP | K0BP K1BP K2BP K3BP : K4BP K5BP K6BP K7BP | LBP 06 | *
;* | |0111> |  H07  B07   SP  ESP :  RSP  SSP  TSP  PSP | K0SP K1SP K2SP K3SP : K4SP K5SP K6SP K7SP | LSP 07 | *
;* |- - - - + - - - - - - - - - - + - - - - - - - - - - + - - - - - - - - - - + - - - - - - - - - - + - - - -+ *
;* | |1000> |  H08  B08  W08  E08 :  R08  S08  T08  P08 | K008 K108 K208 K308 : K408 K508 K608 K708 | L08 08 | *
;* | |1001> |  H09  B09  W09  E09 :  R09  S09  T09  P09 | K009 K109 K209 K309 : K409 K509 K609 K709 | L09 09 | *
;* | |1010> |  H10  B10  W10  E10 :  R10  S10  T10  P10 | K010 K110 K210 K310 : K410 K510 K610 K710 | L10 10 | *
;* | |1011> |  H11  B11  W11  E11 :  R11  S11  T11  P11 | K011 K111 K211 K311 : K411 K511 K611 K711 | L11 11 | *
;* |  -   - | -   -   -   -   -   :   -   -   -   -   - | -   -   -   -   -   :   -   -   -   -   - | -   -  | *
;* | |1100> |  H12  B12  W12  E12 :  R12  S12  T12  P12 | K012 K112 K212 K312 : K412 K512 K612 K712 | L12 12 | *
;* | |1101> |  H13  B13  W13  E13 :  R13  S13  T13  P13 | K013 K113 K213 K313 : K413 K513 K613 K713 | L13 13 | *
;* | |1110> |  H14  B14  W14  E14 :  R14  S14  T14  P14 | K014 K114 K214 K314 : K414 K514 K614 K714 | L14 14 | *
;* | |1111> |  H15  B15  W15  E15 :  R15  S15  T15  P15 | K015 K115 K215 K315 : K415 K515 K615 K715 | L15 15 | *
;* |--------+--------------------+--------------------+--------------------+--------------------+--------+ *
;* | Memory | BVAL BVAL WVAL EVAL : RVAL SVAL TVAL PVAL | K0VL K1VL K2VL K3VL : K4VL K5VL K6VL K7VL |        | *
;* | Displa | BDIS BDIS WDIS EDIS : RDIS SDIS TDIS PDIS | K0DS K1DS K2DS K3DS : K4DS K5DS K6DS K7DS |        | *
;* | Consta | BCST BCST WCST ECST : RCST SCST TCST PCST | K0CS K1CS K2CS K3VL : K4CS K5CS K6CS K7CS |        | *
;* |--------+--------------------+--------------------+--------------------+--------------------+--------+ *
;* | 10000> |  H16  B16  W16  E16 :  R16  S16  T16  P16 | (A0) := <MSiz|Accs>   <MemoSize|Access  > | L16 16 | *
;* | 10001> |  H17  B17  W17  E17 :  R17  S17  T17  P17 | [A1] := <RSiz|RRRR>   <RegiSize|Register> | L17 17 | *
;* | 10010> |  H18  B18  W18  E18 :  R18  S18  T18  P18 | [A2] := <DSiz|BBBB>   <DispSize|BasePntr> | L18 18 | *
;* | 10011> |  H19  B19  W19  E19 :  R19  S19  T19  P19 | [A3] := <Scal|JJJJ>   <Scaleing|Index(J)> | L19 19 | *
;* |  -   - | -   -   -   -   -   :   -   -   -   -   - | -   -   -   -   -   -   -   -   -   -   - | -   -  | *
;* | 10100> |  H20  B20  W20  E20 :  R20  S20  T20  P20 |                                           | L20 20 | *
;* | 10101> |  H21  B21  W21  E21 :  R21  S21  T21  P21 |                                           | L21 21 | *
;* | 10110> |  H22  B22  W22  E22 :  R22  S22  T22  P22 |                                           | L22 22 | *
;* | 10111> |  H23  B23  W23  E23 :  R23  S23  T23  P23 |                                           | L23 23 | *
;* |- - - - + - - - - - - - - - - + - - - - - - - - - - + - - - - - - - - - - - - - - - - - - - - - - + - - - -+ *
;* | 11000> |  H24  B24  W24  E24 :  R24  S24  T24  P24 | (A0) := <MSz|Acces>                       | L24 24 | *
;* | 11001> |  H25  B25  W25  E25 :  R25  S25  T25  P25 | [A1] := <RSz|RRRRR>                       | L25 25 | *
;* | 11010> |  H26  B26  W26  E26 :  R26  S26  T26  P26 | [A2] := <DSz|BBBBB>                       | L26 26 | *
;* | 11011> |  H27  B27  W27  E27 :  R27  S27  T27  P27 | [A3] := <Scl|JJJJJ>                       | L27 27 | *
;* |  -   - | -   -   -   -   -   :   -   -   -   -   - | -   -   -   -   -   -   -   -   -   -   - | -   -  | *
;* | 11100> |  H28  B28  W28  E28 :  R28  S28  T28  P28 |                                           | L28 28 | *
;* | 11101> |  H29  B29  W29  E29 :  R29  S29  T29  P29 |                                           | L29 29 | *
;* | 11110> |  H30  B30  W30  E30 :  R30  S30  T30  P30 |                                           | L30 30 | *
;* | 11111> |  H31  B31  W31  E31 :  R31  S31  T31  P31 |                                           | L31 31 | *
;* |--------+--------------------+--------------------+-------------------------------------------+--------+ *
;*                                                                                                               *
;*****************************************************************************************************************
```

Für positionsunabhängigen Code brauchen wir den InstructionPointer (LIP) als Basisregister für Speicherzugriffe in DATA-Segmenten. Beispielsweise gibt es folgende Zugriffsformen:

```
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
         mov       eax,[lip+dat_AnyValue]                  ; Kommentar
         mov       RVAL[lip+dat_AnyValue],rax              ; Kommentar
         mov       RVAL[lip+dat_AnyValue],012345678h       ; Kommentar
;* -     -    -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
         mov       eax,[lip+lcx*04+dat_AnyValue]           ; Kommentar
         mov       RVAL[lip+lcx*08+dat_AnyValue],rax       ; Kommentar
         mov       RVAL[lip+lcx*08+dat_AnyValue],-1234567890 ; Kommentar
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Man könnte LIB in dem neuen Adressierungsschema als Register 15 festlegen (|BBBB> := |1111>). Dann würde aber L15 verloren gehen. Besser ist es einen speziellen Accesmodus einzuführen, bei dem gilt: <MSiz|Accs> := <MSiz|1111> und <DSiz|Base> := <DSiz|Accs>. Das wird noch genauer illustriert. In dieser Weise lassen sich alle Universalregister LAX,LDX,..,LSP;L08,..,L15;L16,..,L31 vollständig ohne Probleme spezifizieren. Der neue Accesmode hat immer LIP als Basisregister. Deswegen bleibt |BBBB> frei für 16|32 verschiedene Accesmoden bei denen LIP erscheint. Die neuen Register L08,L09,...,L14,L15 sowie L16,L17,...,L30,L31 sollten nur mit Zurückhaltung genutzt werden. Hochsprachen wie Fortran und C können freizügiger diese neuen Register nutzen, da die Register jeweils prozessorspezifisch und betriebsmodusgemäß (32|64|128...) ausgeschöpft werden können.

```
;*****************************************************************************************************************
;*                                                                                                               *
;* +--------+--------------------+--------------------+--------------------+--------------------+--------+ *
;* | <Size| | 0000 0001 0010 0011 : 0100 0101 0110 0111 | 1000 1001 1010 1011 : 1100 1101 1110 1111 | General| *
;* |  Bits  |    8    8   16   32 :   64  128  256  512 | 1024 2048 4096 8192 : 2^14 2^15 2^16 2^17 | Long   | *
;* |  Bytes |    1    1    2    4 :    8   16   32   64 |  128  256  512 1024 : 2048 4096 8192 2^14 | Regist | *
;* |--------+--------------------+--------------------+--------------------+--------------------+--------+ *
;* | |RRRR> | XH0  XB0  XW0  XE0  : XR0  XS0  XT0  XP0  | Y00  Y10  Y20  Y30  : Y40  Y50  Y60  Y70  | ZM0 32 | *
;* | |0001> | XH1  XB1  XW1  XE1  : XR1  XS1  XT1  XP1  | Y01  Y11  Y21  Y31  : Y41  Y51  Y61  Y71  | ZM1 33 | *
;* | |0010> | XH2  XB2  XW2  XE2  : XR2  XS2  XT2  XP2  | Y02  Y12  Y22  Y32  : Y42  Y52  Y62  Y72  | ZM2 34 | *
;* | |0011> | XH3  XB3  XW3  XE3  : XR3  XS3  XT3  XP3  | Y03  Y13  Y23  Y33  : Y43  Y53  Y63  Y73  | ZM3 35 | *
;* |  -   - | -   -   -   -   :   -   -   -   -  | -   -   -   -   :   -   -   -   -  | -   -  | *
;* | |0100> | XH4  XB4  XW4  XE4  : XR4  XS4  XT4  XP4  | Y04  Y14  Y24  Y34  : Y44  Y54  Y64  Y74  | ZM4 36 | *
;* | |0101> | XH5  XB5  XW5  XE5  : XR5  XS5  XT5  XP5  | Y05  Y15  Y25  Y35  : Y45  Y55  Y65  Y75  | ZM5 37 | *
;* | |0110> | XH6  XB6  XW6  XE6  : XR6  XS6  XT6  XP6  | Y06  Y16  Y26  Y36  : Y46  Y56  Y66  Y76  | ZM6 38 | *
;* | |0111> | XH7  XB7  XW7  XE7  : XR7  XS7  XT7  XP7  | Y07  Y17  Y27  Y37  : Y47  Y57  Y67  Y77  | ZM7 39 | *
;* |- - - - + - - - - - - - - - + - - - - - - - - - + - - - - - - - - - + - - - - - - - - - + - - - -+ *
;* | |1000> | XH8  XB8  XW8  XE8  : XR8  XS8  XT8  XP8  | Y08  Y18  Y28  Y38  : Y48  Y58  Y68  Y78  | ZM8 40 | *
;* | |1001> | XH9  XB9  XW9  XE9  : XR9  XS9  XT9  XP9  | Y09  Y19  Y29  Y39  : Y49  Y59  Y69  Y79  | ZM9 41 | *
;* | |1010> | XH10 XB10 XW10 XE10 : XR10 XS10 XT10 XP10 | Y010 Y110 Y210 Y310 : Y410 Y510 Y610 Y710 | Z10 42 | *
;* | |1011> | XH11 XB11 XW11 XE11 : XR11 XS11 XT11 XP11 | Y011 Y111 Y211 Y311 : Y411 Y511 Y611 Y711 | Z11 43 | *
;* |  -   - | -   -   -   -   :   -   -   -   -  | -   -   -   -   :   -   -   -   -  | -   -  | *
;* | |1100> | XH12 XB12 XW12 XE12 : XR12 XS12 XT12 XP12 | Y012 Y112 Y212 Y312 : Y412 Y512 Y612 Y712 | Z12 44 | *
;* | |1101> | XH13 XB13 XW13 XE13 : XR13 XS13 XT13 XP13 | Y013 Y113 Y213 Y313 : Y413 Y513 Y613 Y713 | Z13 45 | *
;* | |1110> | XH14 XB14 XW14 XE14 : XR14 XS14 XT14 XP14 | Y014 Y114 Y214 Y314 : Y414 Y514 Y614 Y714 | Z14 46 | *
;* | |1111> | XH15 XB15 XW15 XE15 : XR15 XS15 XT15 XP15 | Y015 Y115 Y215 Y315 : Y415 Y515 Y615 Y715 | Z15 47 | *
;* |--------+--------------------+--------------------+--------------------+--------------------+--------+ *
;* | 10000> | XH16 XB16 XW16 XE16 : XR16 XS16 XT16 XP16 | XS0  <=>  XMM0     : XS8  <=>  XMM8     | Z16 48 | *
;* | 10001> | XH17 XB17 XW17 XE17 : XR17 XS17 XT17 XP17 | XS1  <=>  XMM1     : XS9  <=>  XMM9     | Z17 49 | *
;* | 10010> | XH18 XB18 XW18 XE18 : XR18 XS18 XT18 XP18 | XS2  <=>  XMM2     : XS10 <=>  XMM10    | Z18 50 | *
;* | 10011> | XH19 XB19 XW19 XE19 : XR19 XS19 XT19 XP19 | XS3  <=>  XMM3     : XS11 <=>  XMM11    | Z19 51 | *
;* |  -   - | -   -   -   -   :   -   -   -   -  | -   -   -   -   :   -   -   -   -  | -   -  | *
;* | 10100> | XH20 XB20 XW20 XE20 : XR20 XS20 XT20 XP20 | XS4  <=>  XMM4     : XS12 <=>  XMM12    | Z20 52 | *
;* | 10101> | XH21 XB21 XW21 XE21 : XR21 XS21 XT21 XP21 | XS5  <=>  XMM5     : XS13 <=>  XMM13    | Z21 53 | *
;* | 10110> | XH22 XB22 XW22 XE22 : XR22 XS22 XT22 XP22 | XS6  <=>  XMM6     : XS14 <=>  XMM14    | Z22 54 | *
;* | 10111> | XH23 XB23 XW23 XE23 : XR23 XS23 XT23 XP23 | XS7  <=>  XMM7     : XS15 <=>  XMM15    | Z23 55 | *
;* |- - - - + - - - - - - - - - + - - - - - - - - - + - - - - - - - - - + - - - - - - - - - + - - - -+ *
;* | 11000> | XH24 XB24 XW24 XE24 : XR24 XS24 XT24 XP24 | XS16 <=>  XMM16    : XS24 <=>  XMM24    | Z24 56 | *
;* | 11001> | XH25 XB25 XW25 XE25 : XR25 XS25 XT25 XP25 | XS17 <=>  XMM17    : XS25 <=>  XMM25    | Z25 57 | *
;* | 11010> | XH26 XB26 XW26 XE26 : XR26 XS26 XT26 XP26 | XS18 <=>  XMM18    : XS26 <=>  XMM26    | Z26 58 | *
;* | 11011> | XH27 XB27 XW27 XE27 : XR27 XS27 XT27 XP27 | XS19 <=>  XMM19    : XS27 <=>  XMM27    | Z27 59 | *
;* |  -   - | -   -   -   -   :   -   -   -   -  | -   -   -   -   :   -   -   -   -  | -   -  | *
;* | 11100> | XH28 XB28 XW28 XE28 : XR28 XS28 XT28 XP28 | XS20 <=>  XMM20    : XS28 <=>  XMM28    | Z28 60 | *
;* | 11101> | XH29 XB29 XW29 XE29 : XR29 XS29 XT29 XP29 | XS21 <=>  XMM21    : XS29 <=>  XMM29    | Z29 61 | *
;* | 11110> | XH30 XB30 XW30 XE30 : XR30 XS30 XT30 XP30 | XS22 <=>  XMM22    : XS30 <=>  XMM30    | Z30 62 | *
;* | 11111> | XH31 XB31 XW31 XE31 : XR31 XS31 XT31 XP31 | XS23 <=>  XMM23    : XS31 <=>  XMM31    | Z31 63 | *
;* |--------+--------------------+--------------------+--------------------+--------------------+--------+ *
;* | |Accs> |  H64  B64   IP  EIP :  RIP  SIP  TIP  PIP | K0IP K1IP K2IP K3IP : K4IP K5IP K6IP K7IP | LIP 64 | *
;* |--------+--------------------+--------------------+----------------------------------------+--------+ *
;*                                                                                                               *
;*****************************************************************************************************************
```

1.5 Übersicht der Datenregister

Der X86-32 hat acht Integerregister (EAX,EDX,ECX,EDI,ESI,EBX,EBP,ESP) sowie den InstructionPointer (EIP), welcher die Leseposition beim Programmablauf beschreibt. Das Register EAX ist das Hauptdatenregister. Man nennt es auch das Akkumulatorregister, da dort die Rechenergebnisse gesammelt werden. Das Datenregister EDX erweitert die Registerbreite auf 64 Bit für Multiplikationen und Divisionen und kann ansonsten für beliebige Daten- und Adressierungsoperationen genutzt werden. Das Register ECX dient in einigen Spezialoperationen (LOOP[cc], MOVS[sz], usw.) als Zählerregister (Counter) und kann ansonsten beliebig benutzt werden. Die Register EDI und ESI dienen in den Spezialoperationen (MOVS[sz] usw.) als Destination und Source Pointer oder auch allgemein als Indexregister. Bei meiner DspLib-Entwicklung in Assembler, wurde mir schnell klar, dass die Register ESI und EDI besser als Basisregister von Quelldaten- und Zieldatenbereichen zu nutzen sind. Als Array-Index bleibt meist EDX und ECX eher verfügbar. Das Register EBX wird bei mir grundsätzlich als Basis-Pointer auf einen großen gemeinsamen statischen Hauptspeicherplatz genutzt und wird niemals umgeladen. Ansonsten könnte EBX auch beliebig genutzt werden. EBP zeigt normalerweise auf lokale (Subroutinen- und Threadprivate) Speicherbereiche, die vom Stapelspeicher temporär genommen werden. Der Stapelspeicher wird vom StackPointer ESP verwaltet. Neben diesen Universalregistern gibt es noch spezielle Rechenregister von der X87-Fließkommaeinheit ST0,ST1,...,ST7 sowie den Vektorrechenregistern XMM0,XMM1,...,XMM7. Außerdem gibt es noch viele Systemprogrammierer Register, die vom normalen Anwendungsprogrammierer nicht benutzt werden.

Beim X86-64 wurden die Integerregister auf 64 Bit erweitert und heißen in dieser Zugriffsgröße RAX,RDX,RCX,RDI,RSI,RBX,RBP,RSP und RIP. Zusätzlich wurden acht zusätzliche Integerregister R08,R09,...,R14,R15 und Vektorregister XMM8,...,XMM15 eingeführt.

Bei dem in diesem Text von mir vorgeschlagenen X86-128 werden die Integerregister auf 128 Bit erweitert und ein komplett neues Schema der Operationscodierung und Adresscodierung eingeführt, welches eine weitere Erweiterung auf 256 und 512 Bit enthält. Außerdem können 16 weitere Integerregister bedient werden, deren 128 Bit Form nun lauten: SAX,SDX,...,SSP;S08,S09,...,S15;S16,S17,... S30,S31 und SIP. Ab dem X86-1024 würden nur noch 4 Bit zur Registerauswahl verbleiben, wodurch die Registerzahl wieder wie beim X86-64 auf 16 beschränkt wäre. Das Register EAX (32 Bit Modus), RAX (64 Bit Modus), SAX (128 Bit Modus), usw. heißt LAX wenn es als Pointer usw. immer in der vollen verfügbaren Registergröße genutzt wird. In der folgenden Tabelle wird kurz erklärt, wie die Registergröße (Siz) mit 3 Bits in den Adressbytes beschrieben wird:

Register Addressbyte	Size <Siz>	0 <000>	1 <001>	2 <010>	3 <011>	4 <100>	5 <101>	6 <110>	7 <111>	Kommentar
MyOutLib Selections:	Bytes Bits:	1 8	1 8	2 16	4 32	8 64	16 128	32 256	64 512	
Memory Operandsize	Full Short	HVAL HV	BVAL BV	WVAL WV	EVAL EV	RVAL RV	SVAL SV	TVAL TV	PVAL PV	EVAL = DVAL EV = DV

In der folgenden Tabelle wird gezeigt, wie aus 3 Bits der Scalingfactor gewählt wird. Es gibt nun alle Scalings in Zweierpotenzen (1,2,4,8,16,32,64) und den Wert 10 für X87 TEMP-Real Felder:

IndexRegister Scaling Value	Scale <Scl>	0 <000>	1 <001>	2 <010>	3 <011>	4 <100>	5 <101>	6 <110>	7 <111>	Kommentar Inside AddressByte A3
Potence Value RegisterType		10	2**0 1 AL,AH	2**1 2 AX	2**2 4 EAX	2**3 8 RAX	2**4 16 SAX	2**5 32 TAX	2**6 64 PAX	How to compute Array Steppings X86 General Purpose
Integers Floatings RegisterType		TEMP ST0	BYTE (INT1) XB0	WORD INT2 XW0	EVAL SNGL XE0	RVAL DBLE XR0	SVAL (QUAD) XS0	TVAL (EX32) XT0	PVAL (EX64) XP0	X86 Single Integers X87 Single Floatings XMM Packed Floatings

Beim X86-1024 würden jeweils 4 Bits für die Operandengröße und den Scalingselector freigehalten werden. Es gäbe nur 16 Register wie beim X86-64. Der Scalingfactor könnte folgende Werte annehmen: 10;1,2,4,8,16,32,64;128,256,512,1024,2048,4096,8192,16384. Vermutlich wäre es am besten den X86-128 gleich in der X86-1024 Version zu formulieren und damit (ein für allemal ?) eine ausreichend erweiterbare Konfiguration festzulegen. Das heißt, ab dem X86-64 gibt es dauerhaft nur die Register LAX,...,L15 und XMM0,..,XMM15. Der InstructionPointer wird als eigenständiger Accessmodus von <MSiz|Accs> gewählt. In meiner Outlib-Funktion dürfte dann LIB die Nummer 64 bekommen und die X87-Fließkommaregister die Nummern 80,81,...,87 bzw. 50h,51h,..,57h. Die Outlib-Registerselektoren 65,66,...,79 könnten beispielsweise für die neuen XMM-FlagRegister usw. genutzt werden.

Register Addressbyte	Size <Siz>	0 <000>	1 <001>	2 <010>	3 <011>	4 <100>	5 <101>	6 <110>	7 <111>	Kommentar
MyOutLib Selections:	Bytes Bits:	1 8	1 8	2 16	4 32	8 64	16 128	32 256	64 512	
<0000|00:ST>	000	AH	AL	AX	EAX	RAX	SAX	TAX	PAX	Accumulator Register LAX
<0000|01:ST>	001	DH	DL	DX	EDX	RDX	SDX	TDX	PDX	Data extend Register LDX
<0000|02:ST>	002	CH	CL	CX	ECX	RCX	SCX	TCX	PCX	Counter etc. Register LCX
<0000|03:ST>	003	H03	B03	DI	EDI	RDI	SDI	TDI	PDI	Destination(I) Pointer LDI
<0000|04:ST>	004	H04	B04	SI	ESI	RSI	SSI	TSI	PSI	Source(Index) Pointer LSI
<0000|05:ST>	005	BH	BL	BX	EBX	RBX	SBX	TBX	PBX	Global Base Pointer LBX
<0000|06:ST>	006	H06	B06	BP	EBP	RBP	SBP	TBP	PBP	Local Base Pointer LBP
<0000|07:ST>	007	H07	B07	SP	ESP	RSP	SSP	TSP	PSP	Local Stack Pointer LSP
<0000|08:ST>	008	H08	B08	W08	E08	R08	S08	T08	P08	(Old) Integer Register L08
<0000|09:ST>	009	H09	B09	W09	E09	R09	S09	T09	P09	(Old) Integer Register L09
<0000|0A:ST>	010	H10	B10	W10	E10	R10	S10	T10	P10	(Old) Integer Register L10
<0000|0B:ST>	011	H11	B11	W11	E11	R11	S11	T11	P11	(Old) Integer Register L11
<0000|0C:ST>	012	H12	B12	W12	E12	R12	S12	T12	P12	(Old) Integer Register L12
<0000|0D:ST>	013	H13	B13	W13	E13	R13	S13	T13	P13	(Old) Integer Register L13
<0000|0E:ST>	014	H14	B14	W14	E14	R14	S14	T14	P14	(Old) Integer Register L14
<0000|0F:ST>	015	H15	B15	W15	E15	R15	S15	T15	P15	(Old) Integer Register L15
<0000|10:ST>	016	H16	B16	W16	E16	R16	S16	T16	P16	(New) Integer Register L16
<0000|11:ST>	017	H17	B17	W17	E17	R17	S17	T17	P17	(New) Integer Register L17
<0000|12:ST>	018	H18	B18	W18	E18	R18	S18	T18	P18	(New) Integer Register L18
<0000|13:ST>	019	H19	B19	W19	E19	R19	S19	T19	P19	(New) Integer Register L19
<0000|14:ST>	020	H20	B20	W20	E20	R20	S20	T20	P20	(New) Integer Register L20
<0000|15:ST>	021	H21	B21	W21	E21	R21	S21	T21	P21	(New) Integer Register L21
<0000|16:ST>	022	H22	B22	W22	E22	R22	S22	T22	P22	(New) Integer Register L22
<0000|17:ST>	023	H23	B23	W23	E23	R23	S23	T23	P23	(New) Integer Register L23
<0000|18:ST>	024	H24	B24	W24	E24	R24	S24	T24	P24	(New) Integer Register L24
<0000|19:ST>	025	H25	B25	W25	E25	R25	S25	T25	P25	(New) Integer Register L25
<0000|1A:ST>	026	H26	B26	W26	E26	R26	S26	T26	P26	(New) Integer Register L26
<0000|1B:ST>	027	H27	B27	W27	E27	R27	S27	T27	P27	(New) Integer Register L27
<0000|1C:ST>	028	H28	B28	W28	E28	R28	S28	T28	P28	(New) Integer Register L28
<0000|1D:ST>	029	H29	B29	W29	E29	R29	S29	T29	P29	(New) Integer Register L29
<0000|1E:ST>	030	H30	B30	W30	E30	R30	S30	T30	P30	(New) Integer Register L30
<0000|1F:ST>	031	H31	B31	W31	E31	R31	S31	T31	P31	(New) Integer Register L31
Memory Operandsize	Full Short	HVAL HV	BVAL BV	WVAL WV	EVAL EV	RVAL RV	SVAL SV	TVAL TV	PVAL PV	EVAL = DVAL EV = DV
XMM-Packed Int/Floating	Size <Siz>	0 <000>	1 <001>	2 <010>	3 <011>	4 <100>	5 <101>	6 <110>	7 <111>	Packed Integer or Packed Floatings
<0000|20:ST>	32	XH0	XB0	XW0	XE0	XR0	XS0	XT0	XP0	(Old) Packed Register XMM0
<0000|21:ST>	33	XH1	XB1	XW1	XE1	XR1	XS1	XT1	XP1	(Old) Packed Register XMM1
<0000|22:ST>	34	XH2	XB2	XW2	XE2	XR2	XS2	XT2	XP2	(Old) Packed Register XMM2
<0000|23:ST>	35	XH3	XB3	XW3	XE3	XR3	XS3	XT3	XP3	(Old) Packed Register XMM3
<0000|24:ST>	36	XH4	XB4	XW4	XE4	XR4	XS4	XT4	XP4	(Old) Packed Register XMM4
<0000|25:ST>	37	XH5	XB5	XW5	XE5	XR5	XS5	XT5	XP5	(Old) Packed Register XMM5
<0000|26:ST>	38	XH6	XB6	XW6	XE6	XR6	XS6	XT6	XP6	(Old) Packed Register XMM6
<0000|27:ST>	39	XH7	XB7	XW7	XE7	XR7	XS7	XT7	XP7	(Old) Packed Register XMM7
<0000|28:ST>	40	XH8	XB8	XW8	XE8	XR8	XS8	XT8	XP8	(Old) Packed Register XMM8
<0000|29:ST>	41	XH9	XB9	XW9	XE9	XR9	XS9	XT9	XP9	(Old) Packed Register XMM9
<0000|2A:ST>	42	XH10	XB10	XW10	XE10	XR10	XS10	XT10	XP10	(Old) Packed Register XMM10
<0000|2B:ST>	43	XH11	XB11	XW11	XE11	XR11	XS11	XT11	XP11	(Old) Packed Register XMM11
<0000|2C:ST>	44	XH12	XB12	XW12	XE12	XR12	XS12	XT12	XP12	(Old) Packed Register XMM12
<0000|2D:ST>	45	XH13	XB13	XW13	XE13	XR13	XS13	XT13	XP13	(Old) Packed Register XMM13
<0000|2E:ST>	46	XH14	XB14	XW14	XE14	XR14	XS14	XT14	XP14	(Old) Packed Register XMM14
<0000|2F:ST>	47	XH15	XB15	XW15	XE15	XR15	XS15	XT15	XP15	(Old) Packed Register XMM15
<0000|30:ST>	48	XH16	XB16	XW16	XE16	XR16	XS16	XT16	XP16	(New) Packed Register XMM16
<0000|31:ST>	49	XH17	XB17	XW17	XE17	XR17	XS17	XT17	XP17	(New) Packed Register XMM17
<0000|32:ST>	50	XH18	XB18	XW18	XE18	XR18	XS18	XT18	XP18	(New) Packed Register XMM18
<0000|33:ST>	51	XH19	XB19	XW19	XE19	XR19	XS19	XT19	XP19	(New) Packed Register XMM19
<0000|34:ST>	52	XH20	XB20	XW20	XE20	XR20	XS20	XT20	XP20	(New) Packed Register XMM20
<0000|35:ST>	53	XH21	XB21	XW21	XE21	XR21	XS21	XT21	XP21	(New) Packed Register XMM21
<0000|36:ST>	54	XH22	XB22	XW22	XE22	XR22	XS22	XT22	XP22	(New) Packed Register XMM22
<0000|37:ST>	55	XH23	XB23	XW23	XE23	XR23	XS23	XT23	XP23	(New) Packed Register XMM23
<0000|38:ST>	56	XH24	XB24	XW24	XE24	XR24	XS24	XT24	XP24	(New) Packed Register XMM24
<0000|39:ST>	57	XH25	XB25	XW25	XE25	XR25	XS25	XT25	XP25	(New) Packed Register XMM25
<0000|3A:ST>	58	XH26	XB26	XW26	XE26	XR26	XS26	XT26	XP26	(New) Packed Register XMM26
<0000|3B:ST>	59	XH27	XB27	XW27	XE27	XR27	XS27	XT27	XP27	(New) Packed Register XMM27
<0000|3C:ST>	60	XH28	XB28	XW28	XE28	XR28	XS28	XT28	XP28	(New) Packed Register XMM28
<0000|3D:ST>	61	XH29	XB29	XW29	XE29	XR29	XS29	XT29	XP29	(New) Packed Register XMM29
<0000|3E:ST>	62	XH30	XB30	XW30	XE30	XR30	XS30	XT30	XP30	(New) Packed Register XMM30
<0000|3F:ST>	63	XH31	XB31	XW31	XE31	XR31	XS31	XT31	XP31	(New) Packed Register XMM31
<0000|40:ST>	64	H64	B64	IP	EIP	RIP	SIP	TIP	PIP	(Cod) Instruction Pointer LIP
<0000|41:ST>	65	.. 79								XMM PackedResults FlagRegister
<0000|50:ST>	80	ST0								(X87) Floatingpoint Register ST0
<0000|51:ST>	81	ST1								(X87) Floatingpoint Register ST1
<0000|56:ST>	86	ST6								(X87) Floatingpoint Register ST6
<0000|57:ST>	87	ST7								(X87) Floatingpoint Register ST7

1.6 Übersicht der X86-Adressierung

Als X86-Befehle bezeichnet man die Operationen, die sich ausschließlich mit den Hauptintegerregister (LAX,..,L15;LIP) betätigen. Der Zieloperand und der Quelloperand bekommen ab dem X86-128 immer eine eigenständige Größenangabe (<RSiz| und <ASiz|). Das gilt auch für Speicherzugriffe, bei denen die Operandengröße durch <MSiz| gegeben wird. Vom X86-32 kennt man Operationen wie MOVZX und MOVSX die einen Zahlenwert durch Zero- oder Sign-eXtension von einer kleineren Bitzahl zu einer größeren Bitzahl erweitern. Dieses Prinzip wird beim X86-128 verallgemeinert:

```
%* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Label001: mov        lax, dh                                  ; fetch LAX := SignExtend( DH  ) ! 512 <= 008 Bit
          mov        lax, dl                                  ; fetch LAX := SignExtend( DL  ) ! 512 <= 008 Bit
          mov        lax, dx                                  ; fetch LAX := SignExtend( DX  ) ! 512 <= 016 Bit
          mov        lax, edx                                 ; fetch LAX := SignExtend( EDX ) ! 512 <= 032 Bit
          mov        lax, rdx                                 ; fetch LAX := SignExtend( RDX ) ! 512 <= 064 Bit
          mov        lax, sdx                                 ; fetch LAX := SignExtend( SDX ) ! 512 <= 128 Bit
          mov        lax, tdx                                 ; fetch LAX := SignExtend( TDX ) ! 512 <= 256 Bit
          mov        lax, pdx                                 ; fetch LAX := SignExtend( PDX ) ! 512 <= 512 Bit
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
Label002: push       ah                                       ; push: STK := SignExtend( AH  ) ! 128 BitModus !
          push       al                                       ; push: STK := SignExtend( AL  ) ! 128 <= 008 Bit
          push       ax                                       ; push: STK := SignExtend( AX  ) ! 128 <= 016 Bit
          push       eax                                      ; push: STK := SignExtend( EAX ) ! 128 <= 032 Bit
          push       rax                                      ; push: STK := SignExtend( RAX ) ! 128 <= 064 Bit
          push       lax                                      ; push: STK := SignExtend( SAX ) ! 128 <= 128 Bit
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Label003: push       BCST 001                                 ; push: STK := SignExtend(BCst ) ! 128 <= 008 Bit
          push       WCST 001                                 ; push: STK := SignExtend(WCst ) ! 128 <= 016 Bit
          push       ECST 001                                 ; push: STK := SignExtend(ECst ) ! 128 <= 032 Bit
          push       RCST 001                                 ; push: STK := SignExtend(RCst ) ! 128 <= 064 Bit
          push       SCST 001                                 ; push: STK := SignExtend(SCst ) ! 128 <= 128 Bit
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
          push       BVAL[lip+dat_BytVALue]                   ; push: STK := SignExtend(BVal ) ! 128 <= 008 Bit
          push       WVAL[lip+dat_WrdVALue]                   ; push: STK := SignExtend(WVal ) ! 128 <= 016 Bit
          push       EVAL[lip+dat_ExtVALue]                   ; push: STK := SignExtend(EVal ) ! 128 <= 032 Bit
          push       RVAL[lip+dat_RegVALue]                   ; push: STK := SignExtend(RVal ) ! 128 <= 064 Bit
          push       SVAL[lip+dat_SexVALue]                   ; push: STK := SignExtend(SVal ) ! 128 <= 128 Bit
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Label004: push       LVAL[lip+dat_LongVALu]                   ; push: STK := SignExtend(SVal ) ! 128 <= 128 Bit
          movzx      eax, dl                                  ; fetch EAX := ZeroExtend( DL)
          movsx      eax, BVAL[ldx+dat_ByteValu]              ; fetch EAX := SignExtend( dat_ByteValu )
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
          add        eax, BVAL[ldx+dat_ByteValu]              ; fetch EAX := SignExtend( dat_ByteValu )
          addsx      eax, BVAL[ldx+dat_ByteValu]              ; fetch EAX := SignExtend( dat_ByteValu )
          addzx      eax, BVAL[ldx+dat_ByteValu]              ; fetch EAX := ZeroExtend( dat_ByteValu )
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
          mov        eax,[lip+000+dat_StepSize]               ; fetch LAX := StepSize
          mul        DVAL[lip+000+dat_StepCntr]               ; mult: LAX := StepSize * StepCntr
          mov        edx,[lbx+lax+any_ValuArry]               ; fetch EDX := ValuArray[ StepCounter ]
%* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Wenn der Zieloperand größer als der Quelloperand ist, dann wird standardmäßig SignExtended, das heißt, das höchste Bit der Quelle füllt alle höheren Bits vom Ziel. Bei PUSH BCST 001 wird dieses Prinzip schon jetzt genutzt. Wenn der Zieloperand kleiner als der Quelloperand ist, dann werden die oberen Bits aus der Quelle genommen. Also MOV AL , EDX kopiert die Bits 24...31 von EDX nach AL. Wenn das Operationsziel ein Register ist, welches kleiner als die Betriebsmodusbreite ist (z.B. 128 Bit) und größergleich 32 Bit, dann wird das Resultat ZeroExtended bis die maximale Registerbreite erreicht ist. Der X86-64 macht es in dieser Weise, wie ich glaube gelesen zu haben. Man kann dann beispielsweise Adress-Offsets als EVAL's (32 Bit) berechnen. Das Resultat ist trotzdem ein gültiger 64 (128) Bit Wert. Das gilt jedoch nur, solange bei der Multiplikation das hohe Register EDX komplett Null bleibt. Solange das höchste Bit von EAX auch Null bleibt, wäre es egal, ob eine Zero- oder Sign-Extension ausgeführt wird. Also, solange der als EVAL berechnete Offset kleiner als 2.147.483.648 bleibt, ist der Erweiterungsmodus für eine vorzeichenlose Multiplikation egal. Wenn negative Offsets entstehen sollen, müsste SignExtended werden. Der Stapelspeicher (Stack) hat eine Schrittweite (Granularität) gemäß dem Betriebsmodus: Beim X86-128 Modus also 16 Bytes, die mit jedem PUSH und POP auf und vom Stapel genommen werden. Ist der Operand kleiner, wird SignExtendet. Es wäre denkbar, dass es ein PUSHZX (ZeroExtend) gibt, welches PUSH BCST 255 als 0000000255 und nicht als -000000001 auf Maschinenbreite extended abspeichert.

Label:	(OP)	(A0)[A1][A2]	(???|Acces)	[RSz|RRRRR]	[ASz|AAAAA]	[Scl|JJJJJ]	Displacement	Constant
Old:	add	ah , dh	(000|00000)	[000|00000]	[000|00001]			
Old:	add	al , dl	(000|00000)	[001|00000]	[001|00001]			
Old:	add	ax , dx	(000|00000)	[010|00000]	[010|00001]			
Old:	add	eax, edx	(000|00000)	[011|00000]	[011|00001]			
Old:	add	rax, rdx	(000|00000)	[100|00000]	[100|00001]			
New:	add	sax, sdx	(000|00000)	[101|00000]	[101|00001]			
New:	add	tax, tdx	(000|00000)	[110|00000]	[110|00001]			
New:	add	pax, pdx	(000|00000)	[111|00000]	[111|00001]			
Label:	(OP)	(A0)[A1]	(CSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Old:	add	ah , BCST 1	(000|00001)	[000|00000]				
Old:	add	al , BCST 12	(001|00001)	[001|00000]				[C0]
Old:	add	ax , WCST 12	(010|00001)	[010|00000]				[C0][C1]
Old:	add	eax, ECST 12	(011|00001)	[011|00000]				[C0][C1][C2][C3]
Old:	add	rax, RCST 12	(100|00001)	[100|00000]				[C0][C1]. . .[C7]
New:	add	sax, SCST 12	(101|00001)	[101|00000]				[C0][C1]. . .[15]
New:	add	tax, TCST 12	(110|00001)	[110|00000]				[C0][C1]. . .[31]
New:	add	pax, PCST 12	(111|00001)	[111|00000]				[C0][C1]. . .[63]
Label:	(OP)	(A0)[A1][A2]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Old:	add	ah ,BVAL[ldx]	(000|00010)	[000|00000]	[000|00001]			
Old:	add	al ,BVAL[ldx+BDIS 12]	(001|00010)	[001|00000]	[001|00001]		[D0]	
Old:	add	ax ,WVAL[ldx+WDIS 12]	(010|00010)	[010|00000]	[010|00001]		[D0][D1]	
Old:	add	eax,EVAL[ldx+EDIS 12]	(011|00010)	[011|00000]	[011|00001]		[D0][D1][D2][D3]	
Old:	add	rax,RVAL[ldx+RDIS 12]	(100|00010)	[100|00000]	[100|00001]		[D0][D1]. . .[D7]	
New:	add	sax,SVAL[ldx+SDIS 12]	(101|00010)	[101|00000]	[101|00001]		[D0][D1]. . .[15]	
New:	add	tax,TVAL[ldx+TDIS 12]	(110|00010)	[110|00000]	[110|00001]		[D0][D1]. . .[31]	
New:	add	pax,PVAL[ldx+PDIS 12]	(111|00010)	[111|00000]	[111|00001]		[D0][D1]. . .[63]	
Label:	(OP)	(A0)[A1][A2][A3]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Old:	add	ah ,BV[ldx+lcx*10]	(000|00011)	[000|00000]	[000|00001]	[000|00010]		
Old:	add	al ,BV[ldx+lcx*01+BD 12]	(001|00011)	[001|00000]	[001|00001]	[001|00010]	[D0]	
Old:	add	ax ,WV[ldx+lcx*02+WD 12]	(010|00011)	[010|00000]	[010|00001]	[010|00010]	[D0][D1]	
Old:	add	eax,EV[ldx+lcx*04+ED 12]	(011|00011)	[011|00000]	[011|00001]	[011|00010]	[D0][D1][D2][D3]	
Old:	add	rax,RV[ldx+lcx*08+RD 12]	(100|00011)	[100|00000]	[100|00001]	[100|00010]	[D0][D1]. . .[D7]	
New:	add	sax,SV[ldx+lcx*16+SD 12]	(101|00011)	[101|00000]	[101|00001]	[101|00010]	[D0][D1]. . .[15]	
New:	add	tax,TV[ldx+lcx*32+TD 12]	(110|00011)	[110|00000]	[110|00001]	[110|00010]	[D0][D1]. . .[31]	
New:	add	pax,PV[ldx+lcx*64+PD 12]	(111|00011)	[111|00000]	[111|00001]	[111|00010]	[D0][D1]. . .[63]	

Label:	(OP)	(A0)[A1][A2]	(MSz|Acces)	[CSz|Const]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Old:	add	BVAL[ldx],019	(000|00100)	[000|01101]	[000|00001]			
Old:	add	BVAL[ldx+BDIS 12],BCST 13	(001|00100)	[001|01101]	[001|00001]		[D0]	[C0]
Old:	add	WVAL[ldx+WDIS 12],WCST 13	(010|00100)	[010|01101]	[010|00001]		[D0][D1]	[C0][C1]
Old:	add	EVAL[ldx+EDIS 12],ECST 13	(011|00100)	[011|01101]	[011|00001]		[D0][D1][D2][D3]	[C0][C1][C2][C3]
Old:	add	RVAL[ldx+RDIS 12],RCST 13	(100|00100)	[100|01101]	[100|00001]		[D0][D1]. . .[D7]	[C0][C1]. . .[C7]
New:	add	SVAL[ldx+SDIS 12],SCST 13	(101|00100)	[101|01101]	[101|00001]		[D0][D1]. . .[15]	[C0][C1]. . .[15]
New:	add	TVAL[ldx+TDIS 12],TCST 13	(110|00100)	[110|01101]	[110|00001]		[D0][D1]. . .[31]	[C0][C1]. . .[31]
New:	add	PVAL[ldx+PDIS 12],PCST 13	(111|00100)	[111|01101]	[111|00001]		[D0][D1]. . .[63]	[C0][C1]. . .[63]
Label:	(OP)	(A0)[A1][A2][A3]	(MSz|Acces)	[CSz|Const]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Old:	add	BV[ldx+lcx*10],0001	(000|00101)	[000|01101]	[000|00001]	[000|00010]		
Old:	add	BV[ldx+lcx*01+BD 12],BC 13	(001|00101)	[001|00000]	[001|00001]	[001|00010]	[D0]	[C0]
Old:	add	WV[ldx+lcx*02+WD 12],WC 13	(010|00101)	[010|00000]	[010|00001]	[010|00010]	[D0][D1]	[C0][C1]
Old:	add	EV[ldx+lcx*04+ED 12],EC 13	(011|00101)	[011|00000]	[011|00001]	[011|00010]	[D0][D1][D2][D3]	[C0][C1][C2][C3]
Old:	add	RV[ldx+lcx*08+RD 12],RC 13	(100|00101)	[100|00000]	[100|00001]	[100|00010]	[D0][D1]. . .[D7]	[C0][C1]. . .[C7]
New:	add	SV[ldx+lcx*16+SD 12],SC 13	(101|00101)	[101|00000]	[101|00001]	[101|00010]	[D0][D1]. . .[15]	[C0][C1]. . .[15]
New:	add	TV[ldx+lcx*32+TD 12],TC 13	(110|00101)	[110|00000]	[110|00001]	[110|00010]	[D0][D1]. . .[31]	[C0][C1]. . .[31]
New:	add	PV[ldx+lcx*64+PD 12],PC 13	(111|00101)	[111|00000]	[111|00001]	[111|00010]	[D0][D1]. . .[63]	[C0][C1]. . .[63]
Label:	(OP)	(A0)[A1][A2]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Old:	add	BVAL[ldx],ah	(000|00110)	[000|00000]	[000|00001]			
Old:	add	BVAL[ldx+BDIS 12],al	(001|00110)	[001|00000]	[001|00001]		[D0]	
Old:	add	WVAL[ldx+WDIS 12],ax	(010|00110)	[010|00000]	[010|00001]		[D0][D1]	
Old:	add	EVAL[ldx+EDIS 12],eax	(011|00110)	[011|00000]	[011|00001]		[D0][D1][D2][D3]	
Old:	add	RVAL[ldx+RDIS 12],rax	(100|00110)	[100|00000]	[100|00001]		[D0][D1]. . .[D7]	
New:	add	SVAL[ldx+SDIS 12],sax	(101|00110)	[101|00000]	[101|00001]		[D0][D1]. . .[15]	
New:	add	TVAL[ldx+TDIS 12],tax	(110|00110)	[110|00000]	[110|00001]		[D0][D1]. . .[31]	
New:	add	PVAL[ldx+PDIS 12],pax	(111|00110)	[111|00000]	[111|00001]		[D0][D1]. . .[63]	
Label:	(OP)	(A0)[A1][A2][A3]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Old:	add	BV[ldx+lcx*10],ah	(000|00111)	[000|00000]	[000|00001]	[000|00010]		
Old:	add	BV[ldx+lcx*01+BD 12],al	(001|00111)	[001|00000]	[001|00001]	[001|00010]	[D0]	
Old:	add	WV[ldx+lcx*02+WD 12],ax	(010|00111)	[010|00000]	[010|00001]	[010|00010]	[D0][D1]	
Old:	add	EV[ldx+lcx*04+ED 12],eax	(011|00111)	[011|00000]	[011|00001]	[011|00010]	[D0][D1][D2][D3]	
Old:	add	RV[ldx+lcx*08+RD 12],rax	(100|00111)	[100|00000]	[100|00001]	[100|00010]	[D0][D1]. . .[D7]	
New:	add	SV[ldx+lcx*16+SD 12],sax	(101|00111)	[101|00000]	[101|00001]	[101|00010]	[D0][D1]. . .[15]	
New:	add	TV[ldx+lcx*32+TD 12],tax	(110|00111)	[110|00000]	[110|00001]	[110|00010]	[D0][D1]. . .[31]	
New:	add	PV[ldx+lcx*64+PD 12],pax	(111|00111)	[111|00000]	[111|00001]	[111|00010]	[D0][D1]. . .[63]	

Label:	(OP)	(A0)[A1][A2]	(MSiz|Accs)	[RSiz|RRRR]	[DSiz|Accs]	[Scal|JJJJ]	Displacement	Constant
Lab:	add	pax, eip	(0000|1111)	[0111|0000]	[0011|0000]			
Lab:	add	eip, al	(0000|1111)	[0001|0000]	[0011|0001]			
Lab:	add	eax,MS[lip+DS 12]	(MSiz|1111)	[0000|0000]	[DSiz|0010]		[D0][D1]. . .[DS]	
Lab:	add	eax,MS[lip+lcx*SC+ DS 12]	(MSiz|1111)	[0000|0000]	[DSiz|0011]	[Scal|0010]	[D0][D1]. . .[DS]	
Lab:	add	MSiz[lip+DSiz 12],CS Cnst	(MSiz|1111)	[CSiz|Cnst]	[DSiz|0100]		[D0][D1]. . .[DS]	[C0][C1]. . .[CS]
Lab:	add	MSiz[lip+lcx+SC+Dis],Cnst	(MSiz|1111)	[CSiz|Cnst]	[DSiz|0101]	[Scal|0010]	[D0][D1]. . .[DS]	[C0][C1]. . .[CS]
Lab:	add	RVAL[lip+DSiz 12],rdi	(0100|1111)	[0100|0011]	[DSiz|0110]		[D0][D1]. . .[DS]	
Lab:	add	SVAL[ldx+lcx+SC+Dis],w13	(0101|1111)	[0010|1101]	[DSiz|0111]	[Scal|0010]	[D0][D1]. . .[DS]	

1.7 Übersicht der X87-Adressierung

Der X86-Mikroprozessor war ursprünglich ein 8/16-Bit Prozessor, nur für Integerzahlen. Man kann mit reiner Integerarithmetik auch per Software Fließkommazahlen bearbeiten. Die MATLIB-Funktionen von DspLib machen genau diese Abbildungen einer Fließkommarechnung mit beliebiger Präzision auf die X86 Integeroperationen ADD, SUB, MUL, DIV usw. Das geht prima ! Selbst komplizierte Operationen wie das schriftliche Ziehen von Quadrat- und Kubikwurzeln und transzendente Funktionen wie EXP(X), LOG(X), SIN(X), usw. lassen sich auf reine Integeroperationen zurückführen. Jedoch ist der Rechenzeitaufwand groß. Will man komplexe numerische Operation, wie große Matrizenrechnungen usw. durchführen, dann müssen Millionen Fließkommaoperationen durchgeführt werden. Das wäre mit einer Integerarithmetik ziemlich zeitintensiv. Deswegen wurde mit der X87 Prozessorerweiterung eine schnelle Fließkommaeinheit zum X86 dazu gefügt. In den folgenden Tabellen werden die möglichen Adressierungsformen den Adressbytes (A0)[A2][A3] zugeordnet. Diese Zuordnung ist vorläufig und prinzipiell. Eine sorgfältige Umsetzung auf die neue X86-128 Architektur wird noch kleine Änderungen bringen. Die X86-128 Architektur kann dann problemlos auf 256 und 512 Bit erweitert werden, da dann lediglich in den <Siz|-Teilen die Werte auch <110| und <111| erreichen können. Ein X86-1024 müsste wieder komplett neu entworfen werden. Oder man beschränkt die Registerzahl auf 16: |RRRR> = |0000>...|1111> für LAX,LDX,LCX,...,L14,L15 und hat dann für <Size| vier Bits, die einen X86-32|64||128|256|512||1024|2048|4096|8192|16384|32768|65536|131072 ermöglichen würde.

Label:	(OP)[OP]	(A0)[A2]	(MSiz|Accs)	[RSiz|RRRR]	[DSiz|BBBB]	[Scal|JJJJ]	Displacement	Cst X86-1024
Lab00:	fild	WVAL[ldx]	(0010|0000)		[0000|0001]			
Lab01:	fild	WVAL[ldx+BDIS 12]	(0010|0000)		[0001|0001]		[D0]	
Lab02:	fild	WVAL[ldx+WDIS 12]	(0010|0000)		[0010|0001]		[D0][D1]	
Lab03:	fild	WVAL[ldx+EDIS 12]	(0010|0000)		[0011|0001]		[D0][D1][D2][D3]	
Lab04:	fild	WVAL[ldx+RDIS 12]	(0010|0000)		[0100|0001]		[D0][D1]. . .[D7]	
Lab05:	fild	WVAL[ldx+SDIS 12]	(0010|0000)		[0101|0001]		[D0][D1]. . .[15]	
Lab06:	fild	WVAL[ldx+TDIS 12]	(0010|0000)		[0110|0001]		[D0][D1]. . .[31]	
Lab07:	fild	WVAL[ldx+PDIS 12]	(0010|0000)		[0111|0001]		[D0][D1]. . .[63]	
Label:	(OP)[OP]	(A0)[A2][A3]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Lab10:	fild	WVAL[ldx+lcx*10]	(010|00001)		[000|00001]	[000|00010]		
Lab11:	fild	WVAL[ldx+lcx*01+BD 12]	(010|00001)		[001|00001]	[001|00010]	[D0]	
Lab12:	fild	WVAL[ldx+lcx*02+WD 12]	(010|00001)		[010|00001]	[010|00010]	[D0][D1]	
Lab13:	fild	WVAL[ldx+lcx*04+ED 12]	(010|00001)		[011|00001]	[011|00010]	[D0][D1][D2][D3]	
Lab14:	fild	WVAL[ldx+lcx*08+RD 12]	(010|00001)		[100|00001]	[100|00010]	[D0][D1]. . .[D7]	
Lab15:	fild	WVAL[ldx+lcx*16+SD 12]	(010|00001)		[101|00001]	[101|00010]	[D0][D1]. . .[15]	
Lab16:	fild	WVAL[ldx+lcx*32+TD 12]	(010|00001)		[110|00001]	[110|00010]	[D0][D1]. . .[31]	
Lab17:	fild	WVAL[ldx+lcx*64+PD 12]	(010|00001)		[111|00001]	[111|00010]	[D0][D1]. . .[63]	
Label:	(OP)[OP]	(A0)[A2]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Lab20:	fild	EVAL[ldx]	(011|00010)		[000|00001]			
Lab21:	fild	EVAL[ldx+BDIS 12]	(011|00010)		[001|00001]		[D0]	
Lab22:	fild	EVAL[ldx+WDIS 12]	(011|00010)		[010|00001]		[D0][D1]	
Lab23:	fild	EVAL[ldx+EDIS 12]	(011|00010)		[011|00001]		[D0][D1][D2][D3]	
Lab24:	fild	EVAL[ldx+RDIS 12]	(011|00010)		[100|00001]		[D0][D1]. . .[D7]	
Lab25:	fild	EVAL[ldx+SDIS 12]	(011|00010)		[101|00001]		[D0][D1]. . .[15]	
Lab26:	fild	EVAL[ldx+TDIS 12]	(011|00010)		[110|00001]		[D0][D1]. . .[31]	
Lab27:	fild	EVAL[ldx+PDIS 12]	(011|00010)		[111|00001]		[D0][D1]. . .[63]	
Label:	(OP)[OP]	(A0)[A2][A3]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Lab30:	fild	EVAL[ldx+lcx*10]	(011|00011)		[000|00001]	[000|00010]		
Lab31:	fild	EVAL[ldx+lcx*01+BD 12]	(011|00011)		[001|00001]	[001|00010]	[D0]	
Lab32:	fild	EVAL[ldx+lcx*02+WD 12]	(011|00011)		[010|00001]	[010|00010]	[D0][D1]	
Lab33:	fild	EVAL[ldx+lcx*04+ED 12]	(011|00011)		[011|00001]	[011|00010]	[D0][D1][D2][D3]	
Lab34:	fild	EVAL[ldx+lcx*08+RD 12]	(011|00011)		[100|00001]	[100|00010]	[D0][D1]. . .[D7]	
Lab35:	fild	EVAL[ldx+lcx*16+SD 12]	(011|00011)		[101|00001]	[101|00010]	[D0][D1]. . .[15]	
Lab36:	fild	EVAL[ldx+lcx*32+TD 12]	(011|00011)		[110|00001]	[110|00010]	[D0][D1]. . .[31]	
Lab37:	fild	EVAL[ldx+lcx*64+PD 12]	(011|00011)		[111|00001]	[111|00010]	[D0][D1]. . .[63]	
Label:	(OP)[OP]	(A0)[A2]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Lab40:	fild	RVAL[ldx]	(100|00100)		[000|00001]			
Lab41:	fild	RVAL[ldx+BDIS 12]	(100|00100)		[001|00001]		[D0]	
Lab42:	fild	RVAL[ldx+WDIS 12]	(100|00100)		[010|00001]		[D0][D1]	
Lab43:	fild	RVAL[ldx+EDIS 12]	(100|00100)		[011|00001]		[D0][D1][D2][D3]	
Lab44:	fild	RVAL[ldx+RDIS 12]	(100|00100)		[100|00001]		[D0][D1]. . .[D7]	
Lab45:	fild	RVAL[ldx+SDIS 12]	(100|00100)		[101|00001]		[D0][D1]. . .[15]	
Lab46:	fild	RVAL[ldx+TDIS 12]	(100|00100)		[110|00001]		[D0][D1]. . .[31]	
Lab47:	fild	RVAL[ldx+PDIS 12]	(100|00100)		[111|00001]		[D0][D1]. . .[63]	
Label:	(OP)[OP]	(A0)[A2][A3]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Lab50:	fild	RVAL[ldx+lcx*10]	(100|00101)		[000|00001]	[000|00010]		
Lab51:	fild	RVAL[ldx+lcx*01+BD 12]	(100|00101)		[001|00001]	[001|00010]	[D0]	
Lab52:	fild	RVAL[ldx+lcx*02+WD 12]	(100|00101)		[010|00001]	[010|00010]	[D0][D1]	
Lab53:	fild	RVAL[ldx+lcx*04+ED 12]	(100|00101)		[011|00001]	[011|00010]	[D0][D1][D2][D3]	
Lab54:	fild	RVAL[ldx+lcx*08+RD 12]	(100|00101)		[100|00001]	[100|00010]	[D0][D1]. . .[D7]	
Lab55:	fild	RVAL[ldx+lcx*16+SD 12]	(100|00101)		[101|00001]	[101|00010]	[D0][D1]. . .[15]	
Lab56:	fild	RVAL[ldx+lcx*32+TD 12]	(100|00101)		[110|00001]	[110|00010]	[D0][D1]. . .[31]	
Lab57:	fild	RVAL[ldx+lcx*64+PD 12]	(100|00101)		[111|00001]	[111|00010]	[D0][D1]. . .[63]	

Die folgende Tabelle zeigt die Zahlenformate, die von der X87 Fließkommaeinheit bearbeitet werden können. Es sind jedoch immer nur Einzelwerte (Skalare). Erst mit der frühen 64 Bit MMX-Erweiterung wurde das Prinzip der gepackten Vektoroperationen eingeführt. Mit der 128 Bit XMM-Erweiterung gab es dann eigenständige Vektordatenregister (ursprünglich XMM0...XMM7).

Full X87 Operand	Siz <S>	Bits used	Fortran [I]VALUE	IX [I]	XBN 8	XWN 16	XEN 32	XRN 64	XSN 128	XTN 256	XPN 512	Expo Bits	Frac Bits	Maximal possible Value Decimal Representation	Decimal Precision
WVAL	2	16	INT*2	I		1	2	4	8	16	32		15	+/-32767	4.5
EVAL	3	32	INT*4	I			1	2	4	8	16		31	+/-2147483647	9.3
RVAL	4	64	INT*8	J				1	2	4	8		63	+9.223372036E+000000018	18.9
SNGL	3	32	REAL*4	S			1	2	4	8	16	8	23	+3.402823669E+000000038	6.9
DBLE	4	64	REAL*8	D				1	2	4	8	11	52	+1.797693143E+000000308	15.6
TEMP	5	80	REAL*10	T					?	?	?	15	63	+1.189731581E+000004932	18.9

Label:	(OP)[OP]	(A0)[A2]	(RRRR|Accs)	[RSiz|RRRR]	[DSiz|BBBB]	[Scal|JJJJ]	Displacement	Cst X86-1024
Lab00:	fld	st0	(0000|0000)					
Lab01:	fld	st1	(0001|0000)					
Lab02:	fld	st2	(0010|0000)					
Lab03:	fld	st3	(0011|0000)					
Lab04:	fld	st4	(0100|0000)					
Lab05:	fld	st5	(0101|0000)					
Lab06:	fld	st6	(0110|0000)					
Lab07:	fld	st7	(0111|0000)					

Label:	(OP)[OP]	(A0)[A2]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Lab20:	fld	SNGL[ldx]	(011|00010)		[000|00001]			
Lab21:	fld	SNGL[ldx+BDIS 12]	(011|00010)		[001|00001]		[D0]	
Lab22:	fld	SNGL[ldx+WDIS 12]	(011|00010)		[010|00001]		[D0][D1]	
Lab23:	fld	SNGL[ldx+EDIS 12]	(011|00010)		[011|00001]		[D0][D1][D2][D3]	
Lab24:	fld	SNGL[ldx+RDIS 12]	(011|00010)		[100|00001]		[D0][D1]. . .[D7]	
Lab25:	fld	SNGL[ldx+SDIS 12]	(011|00010)		[101|00001]		[D0][D1]. . .[15]	
Lab26:	fld	SNGL[ldx+TDIS 12]	(011|00010)		[110|00001]		[D0][D1]. . .[31]	
Lab27:	fld	SNGL[ldx+PDIS 12]	(011|00010)		[111|00001]		[D0][D1]. . .[63]	
Label:	(OP)[OP]	(A0)[A2][A3]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Lab30:	fld	SNGL[ldx+lcx*10]	(011|00011)		[000|00001]	[000|00010]		
Lab31:	fld	SNGL[ldx+lcx*01+BD 12]	(011|00011)		[001|00001]	[001|00010]	[D0]	
Lab32:	fld	SNGL[ldx+lcx*02+WD 12]	(011|00011)		[010|00001]	[010|00010]	[D0][D1]	
Lab33:	fld	SNGL[ldx+lcx*04+ED 12]	(011|00011)		[011|00001]	[011|00010]	[D0][D1][D2][D3]	
Lab34:	fld	SNGL[ldx+lcx*08+RD 12]	(011|00011)		[100|00001]	[100|00010]	[D0][D1]. . .[D7]	
Lab35:	fld	SNGL[ldx+lcx*16+SD 12]	(011|00011)		[101|00001]	[101|00010]	[D0][D1]. . .[15]	
Lab36:	fld	SNGL[ldx+lcx*32+TD 12]	(011|00011)		[110|00001]	[110|00010]	[D0][D1]. . .[31]	
Lab37:	fld	SNGL[ldx+lcx*64+PD 12]	(011|00011)		[111|00001]	[111|00010]	[D0][D1]. . .[63]	

Label:	(OP)[OP]	(A0)[A2]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Lab40:	fld	DBLE[ldx]	(100|00100)		[000|00001]			
Lab41:	fld	DBLE[ldx+BDIS 12]	(100|00100)		[001|00001]		[D0]	
Lab42:	fld	DBLE[ldx+WDIS 12]	(100|00100)		[010|00001]		[D0][D1]	
Lab43:	fld	DBLE[ldx+EDIS 12]	(100|00100)		[011|00001]		[D0][D1][D2][D3]	
Lab44:	fld	DBLE[ldx+RDIS 12]	(100|00100)		[100|00001]		[D0][D1]. . .[D7]	
Lab45:	fld	DBLE[ldx+SDIS 12]	(100|00100)		[101|00001]		[D0][D1]. . .[15]	
Lab46:	fld	DBLE[ldx+TDIS 12]	(100|00100)		[110|00001]		[D0][D1]. . .[31]	
Lab47:	fld	DBLE[ldx+PDIS 12]	(100|00100)		[111|00001]		[D0][D1]. . .[63]	
Label:	(OP)[OP]	(A0)[A2][A3]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Lab50:	fld	DBLE[ldx+lcx*10]	(100|00101)		[000|00001]	[000|00010]		
Lab51:	fld	DBLE[ldx+lcx*01+BD 12]	(100|00101)		[001|00001]	[001|00010]	[D0]	
Lab52:	fld	DBLE[ldx+lcx*02+WD 12]	(100|00101)		[010|00001]	[010|00010]	[D0][D1]	
Lab53:	fld	DBLE[ldx+lcx*04+ED 12]	(100|00101)		[011|00001]	[011|00010]	[D0][D1][D2][D3]	
Lab54:	fld	DBLE[ldx+lcx*08+RD 12]	(100|00101)		[100|00001]	[100|00010]	[D0][D1]. . .[D7]	
Lab55:	fld	DBLE[ldx+lcx*16+SD 12]	(100|00101)		[101|00001]	[101|00010]	[D0][D1]. . .[15]	
Lab56:	fld	DBLE[ldx+lcx*32+TD 12]	(100|00101)		[110|00001]	[110|00010]	[D0][D1]. . .[31]	
Lab57:	fld	DBLE[ldx+lcx*64+PD 12]	(100|00101)		[111|00001]	[111|00010]	[D0][D1]. . .[63]	

Label:	(OP)	(A0)[A1][A2]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Lab60:	fld	TEMP[ldx]	(101|00110)		[000|00001]			
Lab61:	fld	TEMP[ldx+BDIS 12]	(101|00110)		[001|00001]		[D0]	
Lab62:	fld	TEMP[ldx+WDIS 12]	(101|00110)		[010|00001]		[D0][D1]	
Lab63:	fld	TEMP[ldx+EDIS 12]	(101|00110)		[011|00001]		[D0][D1][D2][D3]	
Lab64:	fld	TEMP[ldx+RDIS 12]	(101|00110)		[100|00001]		[D0][D1]. . .[D7]	
Lab65:	fld	TEMP[ldx+SDIS 12]	(101|00110)		[101|00001]		[D0][D1]. . .[15]	
Lab66:	fld	TEMP[ldx+TDIS 12]	(101|00110)		[110|00001]		[D0][D1]. . .[31]	
Lab67:	fld	TEMP[ldx+PDIS 12]	(101|00110)		[111|00001]		[D0][D1]. . .[63]	
Label:	(OP)[OP]	(A0)[A2][A3]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
Lab70:	fld	TEMP[ldx+lcx*10]	(101|00111)		[000|00001]	[000|00010]		
Lab71:	fld	TEMP[ldx+lcx*01+BD 12]	(101|00111)		[001|00001]	[001|00010]	[D0]	
Lab72:	fld	TEMP[ldx+lcx*02+WD 12]	(101|00111)		[010|00001]	[010|00010]	[D0][D1]	
Lab73:	fld	TEMP[ldx+lcx*04+ED 12]	(101|00111)		[011|00001]	[011|00010]	[D0][D1][D2][D3]	
Lab74:	fld	TEMP[ldx+lcx*08+RD 12]	(101|00111)		[100|00001]	[100|00010]	[D0][D1]. . .[D7]	
Lab75:	fld	TEMP[ldx+lcx*16+SD 12]	(101|00111)		[101|00001]	[101|00010]	[D0][D1]. . .[15]	
Lab76:	fld	TEMP[ldx+lcx*32+TD 12]	(101|00111)		[110|00001]	[110|00010]	[D0][D1]. . .[31]	
Lab77:	fld	TEMP[ldx+lcx*64+PD 12]	(101|00111)		[111|00001]	[111|00010]	[D0][D1]. . .[63]	

1.8 Übersicht der XMM-Adressierung

Die XMM-Register können vektorartige Zahlenpakete speichern und verarbeiten. Erstmalig in den MMX (MultiMediaXtension) Befehlen eingeführt. Anfänglich als Überlagerung der (80 Bit) X87-Fließkommaregister und dem neuen Begriff der „Packed“ (PADD,..) Operationen. Später wurden acht neue und eigenständige 128 Bit Register (XMM0,..,XMM7) für diesen Zweck eingeführt, welche nach und nach mit Operationen garniert wurden. Mit der hier eingeführten X86-128 usw. Architektur sollen alle skalar- und vektorartigen Rechenoperationen als Integer- und Fließkommazahlen eine „vollständige“ und systematische Abbildung auf die XMM-Register bekommen. Damit könnte langfristig die parallele X87-Einheit abgeschafft werden und damit der X86 Prozessor seinen Charakter vom „mittelalterlichen Dorf“ zu einem gestrafften „Universalboliden“ wechseln. Betrachtet man beispielsweise die X86-512 Architektur, dann hätten wir auch 512 Bit große XMM-Register. Diese können in 64 Bytes, 32 Words, 16 EVAL/SNGL, acht RVAL/DBLE, vier SVAL/QUAD, zwei TVAL/EX32 oder einen PVAL/EX64 geteilt werden. Beim X86-128 (XMM schon jetzt) hätten wir 16 Bytes, acht Words, vier EVAL/SNGL, zwei RVAL/DBLE oder einen SVAL/QUAD Operanden zur Verfügung:

```
+-----------------------------------------------------------------------------------------------------------------------+
|                                                                                      P00                              |
|                                                              T00                         T01                          |
|                              S00                              |                  S01    ....   S03                  |
|               R00             |               R01             |             R02         ....         R07              |
|      E00      |      E01      |      E02      |      E03      |      E04      |         ....        |      E15      |
|  W00  |  W01  |  W02  |  W03  |  W04  |  W05  |  W06  |  W07  |  W08  |  W09  |  W10     ....    W29  |  W30  |  W31  |
|B00|B01|B02|B03|B04|B05|B06|B07|B08|B09|B10|B11|B12|B13|B14|B15|B16|B17|B18|B19|B20|B21|  ....  |B58|B59|B60|B61|B62|B63|
|007|015|023|031|039|047|055|063|071|079|087|095|103|111|119|127|135|143|151|159|167|175|  ....  |471|479|487|495|503|511|
+---------------------------------------------------------------+-------------------------------------------------------+
```

Diese Operanden können entweder als Integer- oder als Fließkommazahlen interpretiert werden. Dabei entsteht bis zum X86-512 folgender Zoo aus Datenformaten und deren Zahlenwertbereichen:

Full XMM Operand	Siz <S>	Bits used	Fortran [I]VALUE	IX [I]	XBN 8	XWN 16	XEN 32	XRN 64	XSN 128	XTN 256	XPN 512	Expo Bits	Frac Bits	Maximal possible Value Decimal Representation	Decimal Precision
Bytes	1	8	INT*1	I	1	2	4	8	16	32	64		8	255	2.4
Words	2	16	INT*2	I		1	2	4	8	16	32		16	65535	4.8
EVals	3	32	INT*4	I			1	2	4	8	16		32	4294967295	9.6
RVals	4	64	INT*8	J				1	2	4	8		64	+1.844674407E+000000019	19.2
SVals	5	128	INT*16	K					1	2	4		128	+3.402823669E+000000038	38.5
TVals	6	256	INT*32	L						1	2		256	+1.157920892E+000000077	77.1
PVals	7	512	INT*64	M							4		512	+1.340780793E+000000154	154.1
SNGL	3	32	REAL*4	S			1	2	4	8	16	8	23	+3.402823669E+000000038	6.9
DBLE	4	64	REAL*8	D				1	2	4	8	11	52	+1.797693143E+000000308	15.6
TEMP	?	80	REAL*10	T					?	?	?	15	63	+1.189731581E+000004932	18.9
QUAD	5	128	REAL*16	Q					1	2	4	15	112	+1.189731581E+000004932	33.7
EX32	6	256	REAL*32	U						1	2	31	223	+4.226686143E+323228496	67.1
EX64	7	512	REAL*64	V							1	31	479	+4.226686143E+323228496	144.2
EX128		1024	CHAR*128	W								31	991	+4.226686143E+323228496	298.3
EX256		2048	CHAR*256	X								31	2015	+4.226686143E+323228496	606.5
EX512		4096	CHAR*512	Y								31	4043	+4.226686143E+323228496	1223.0
EX1024		8192	CHAR*1024	Z								31	8159	+4.226686143E+323228496	2456.1

Der Fließkommatyp „QUAD“ hat genau wie „TEMP“ 15 Bits als Exponent und alle folgenden EXtended Precision Typen 31 Bits für den Exponenten. Dieses System ist in DspLib per Software realisiert ! Der QUAD-Wert wäre für schnelle und komplexe Rechnungen als Hardware-Typus wünschenswert. Er passt bestens in die klassischen XMM-Register. Der Operationscode gibt vor, ob beispielsweise Integer Words oder Fließkomma DBLEs bearbeitet werden sollen. Die Operandengrößen geben die Zahl der zu bearbeitenden Vektorelemente an: Wird beispielsweise als Ziel XE0 (32 Bit) gewählt und Byteoperanden, dann werden 4 Bytes bearbeitet und geändert. Ist der Quelloperand größer als der Zieloperand, dann werden höherliegende Registerinhalte als Quelle genutzt. Ist der Zieloperand größer als der Quelloperand, dann werden höherliegende Registerinhalte als Ziel genutzt. Der kleinere Wert gibt die zu bearbeitende Vektorausschnittslänge an. Als Operationen sollen alle Grundrechenarten immer möglich sein. Für die Fließkommaoperationen sollten auch Wurzeln, Potenzen und transzendente Funktionen verfügbar sein. Auch müssen etliche Zahlenformattransformationen verfügbar sein. Es sollte Integerregister geben, in denen die Rechenresultate geflaged werden. Also ein ZeroFlag-Register, ein SignFlag-Register usw. welches in ein normales Register wie LAX geladen werden kann. Von dort können mittels Bit-Tests Ergebnisprüfungen ausgewertet werden.

Lab	(OP)[OP]	(A0)[A1][A2]	(???|Acces)	[RSz|RRRRR]	[ASz|AAAAA]	[Scl|JJJJJ]	Displacement	Constant
L:	paddhi	xh0, xh1	(000|00000)	[000|00000]	[000|00001]			
L:	paddbi	xb0, xb1	(000|00000)	[001|00000]	[001|00001]			
L:	paddwi	xw0, xw1	(000|00000)	[010|00000]	[010|00001]			
L:	paddei	xe0, xe1	(000|00000)	[011|00000]	[011|00001]			
L:	paddri	xr0, xr1	(000|00000)	[100|00000]	[100|00001]			
L:	paddsi	xm0, xm1	(000|00000)	[101|00000]	[101|00001]			
L:	paddti	ym0, ym1	(000|00000)	[110|00000]	[110|00001]			
L:	paddpi	zm0, zm1	(000|00000)	[111|00000]	[111|00001]			
Label:	(OP)[OP]	(A0)[A1]	(CSz|Acces)	[RSz|RRRRR]	[Siz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
L:	paddhi	xh0, BCST 1	(000|00001)	[000|00000]				
L:	paddbi	xb0, BCST 12	(001|00001)	[001|00000]				[C0]
L:	paddwi	xw0, WCST 12	(010|00001)	[010|00000]				[C0][C1]
L:	paddei	xe0, ECST 12	(011|00001)	[011|00000]				[C0][C1][C2][C3]
L:	paddri	xr0, RCST 12	(100|00001)	[100|00000]				[C0][C1]. . .[C7]
L:	paddsi	xs0, SCST 12	(101|00001)	[101|00000]				[C0][C1]. . .[15]
L:	paddti	xt0, TCST 12	(110|00001)	[110|00000]				[C0][C1]. . .[31]
L:	paddpi	xp0, PCST 12	(111|00001)	[111|00000]				[C0][C1]. . .[63]
Label:	(OP)	(A0)[A1][A2]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
L:	paddhi	xh0,BVAL[ldx]	(000|00010)	[000|00000]	[000|00001]			
L:	paddbi	xb0,BVAL[ldx+BDIS 12]	(001|00010)	[001|00000]	[001|00001]		[D0]	
L:	paddwi	xw0,WVAL[ldx+WDIS 12]	(010|00010)	[010|00000]	[010|00001]		[D0][D1]	
L:	paddei	xe0,EVAL[ldx+EDIS 12]	(011|00010)	[011|00000]	[011|00001]		[D0][D1][D2][D3]	
L:	paddri	xr0,RVAL[ldx+RDIS 12]	(100|00010)	[100|00000]	[100|00001]		[D0][D1]. . .[D7]	
L:	paddsi	xs0,SVAL[ldx+SDIS 12]	(101|00010)	[101|00000]	[101|00001]		[D0][D1]. . .[15]	
L:	paddti	xt0,TVAL[ldx+TDIS 12]	(110|00010)	[110|00000]	[110|00001]		[D0][D1]. . .[31]	
L:	paddpi	xp0,PVAL[ldx+PDIS 12]	(111|00010)	[111|00000]	[111|00001]		[D0][D1]. . .[63]	
Label:	(OP)	(A0)[A1][A2][A3]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
L:	paddhi	xh0,BV[ldx+lcx*10]	(000|00011)	[000|00000]	[000|00001]	[000|00010]		
L:	paddbi	xb0,BV[ldx+lcx*01+BD 12]	(001|00011)	[001|00000]	[001|00001]	[001|00010]	[D0]	
L:	paddwi	xw0,WV[ldx+lcx*02+WD 12]	(010|00011)	[010|00000]	[010|00001]	[010|00010]	[D0][D1]	
L:	paddei	xe0,EV[ldx+lcx*04+ED 12]	(011|00011)	[011|00000]	[011|00001]	[011|00010]	[D0][D1][D2][D3]	
L:	paddri	xr0,RV[ldx+lcx*08+RD 12]	(100|00011)	[100|00000]	[100|00001]	[100|00010]	[D0][D1]. . .[D7]	
L:	paddsi	xs0,SV[ldx+lcx*16+SD 12]	(101|00011)	[101|00000]	[101|00001]	[101|00010]	[D0][D1]. . .[15]	
L:	paddti	xt0,TV[ldx+lcx*32+TD 12]	(110|00011)	[110|00000]	[110|00001]	[110|00010]	[D0][D1]. . .[31]	
L:	paddpi	xp0,PV[ldx+lcx*64+PD 12]	(111|00011)	[111|00000]	[111|00001]	[111|00010]	[D0][D1]. . .[63]	

Label:	(OP)	(A0)[A1][A2]	(MSz|Acces)	[CSz|Const]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
L:	paddhi	BVAL[ldx],019	(000|00100)	[000|01101]	[000|00001]			
L:	paddbi	BVAL[ldx+BDIS 12],BCST 13	(001|00100)	[001|01101]	[001|00001]		[D0]	[C0]
L:	paddwi	WVAL[ldx+WDIS 12],WCST 13	(010|00100)	[010|01101]	[010|00001]		[D0][D1]	[C0][C1]
L:	paddei	EVAL[ldx+EDIS 12],ECST 13	(011|00100)	[011|01101]	[011|00001]		[D0][D1][D2][D3]	[C0][C1][C2][C3]
L:	paddri	RVAL[ldx+RDIS 12],RCST 13	(100|00100)	[100|01101]	[100|00001]		[D0][D1]. . .[D7]	[C0][C1]. . .[C7]
L:	paddsi	SVAL[ldx+SDIS 12],SCST 13	(101|00100)	[101|01101]	[101|00001]		[D0][D1]. . .[15]	[C0][C1]. . .[15]
L:	paddti	TVAL[ldx+TDIS 12],TCST 13	(110|00100)	[110|01101]	[110|00001]		[D0][D1]. . .[31]	[C0][C1]. . .[31]
L:	paddpi	PVAL[ldx+PDIS 12],PCST 13	(111|00100)	[111|01101]	[111|00001]		[D0][D1]. . .[63]	[C0][C1]. . .[63]
Label:	(OP)[OP]	(A0)[A1][A2][A3]	(MSz|Acces)	[CSz|Const]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
L	paddhi	BV[ldx+lcx*10],0001	(000|00101)	[000|01101]	[000|00001]	[000|00010]		
L	paddbi	BV[ldx+lcx*01+BD 12],BC 13	(001|00101)	[001|00000]	[001|00001]	[001|00010]	[D0]	[C0]
L	paddwi	WV[ldx+lcx*02+WD 12],WC 13	(010|00101)	[010|00000]	[010|00001]	[010|00010]	[D0][D1]	[C0][C1]
L	paddei	EV[ldx+lcx*04+ED 12],EC 13	(011|00101)	[011|00000]	[011|00001]	[011|00010]	[D0][D1][D2][D3]	[C0][C1][C2][C3]
L	paddri	RV[ldx+lcx*08+RD 12],RC 13	(100|00101)	[100|00000]	[100|00001]	[100|00010]	[D0][D1]. . .[D7]	[C0][C1]. . .[C7]
L	paddsi	SV[ldx+lcx*16+SD 12],SC 13	(101|00101)	[101|00000]	[101|00001]	[101|00010]	[D0][D1]. . .[15]	[C0][C1]. . .[15]
L	paddti	TV[ldx+lcx*32+TD 12],TC 13	(110|00101)	[110|00000]	[110|00001]	[110|00010]	[D0][D1]. . .[31]	[C0][C1]. . .[31]
L	paddpi	PV[ldx+lcx*64+PD 12],PC 13	(111|00101)	[111|00000]	[111|00001]	[111|00010]	[D0][D1]. . .[63]	[C0][C1]. . .[63]
Label:	(OP)[OP]	(A0)[A1][A2]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
L:	paddhi	BVAL[ldx],xb0	(000|00110)	[000|00000]	[000|00001]			
L:	paddbi	BVAL[ldx+BDIS 12],xb0	(001|00110)	[001|00000]	[001|00001]		[D0]	
L:	paddwi	WVAL[ldx+WDIS 12],xw0	(010|00110)	[010|00000]	[010|00001]		[D0][D1]	
L:	paddei	EVAL[ldx+EDIS 12],xe0	(011|00110)	[011|00000]	[011|00001]		[D0][D1][D2][D3]	
L:	paddri	RVAL[ldx+RDIS 12],xr0	(100|00110)	[100|00000]	[100|00001]		[D0][D1]. . .[D7]	
L:	paddsi	SVAL[ldx+SDIS 12],xs0	(101|00110)	[101|00000]	[101|00001]		[D0][D1]. . .[15]	
L:	paddti	TVAL[ldx+TDIS 12],xt0	(110|00110)	[110|00000]	[110|00001]		[D0][D1]. . .[31]	
L:	paddpi	PVAL[ldx+PDIS 12],xp0	(111|00110)	[111|00000]	[111|00001]		[D0][D1]. . .[63]	
Label:	(OP)[OP]	(A0)[A1][A2][A3]	(MSz|Acces)	[RSz|RRRRR]	[DSz|BBBBB]	[Scl|JJJJJ]	Displacement	Constant
L:	paddhi	BV[ldx+lcx*10],xb0	(000|00111)	[000|00000]	[000|00001]	[000|00010]		
L:	paddbi	BV[ldx+lcx*01+BD 12],xb0	(001|00111)	[001|00000]	[001|00001]	[001|00010]	[D0]	
L:	paddwi	WV[ldx+lcx*02+WD 12],xw0	(010|00111)	[010|00000]	[010|00001]	[010|00010]	[D0][D1]	
L:	paddei	EV[ldx+lcx*04+ED 12],xe0	(011|00111)	[011|00000]	[011|00001]	[011|00010]	[D0][D1][D2][D3]	
L:	paddri	RV[ldx+lcx*08+RD 12],xr0	(100|00111)	[100|00000]	[100|00001]	[100|00010]	[D0][D1]. . .[D7]	
L:	paddsi	SV[ldx+lcx*16+SD 12],xs0	(101|00111)	[101|00000]	[101|00001]	[101|00010]	[D0][D1]. . .[15]	
L:	paddti	TV[ldx+lcx*32+TD 12],xt0	(110|00111)	[110|00000]	[110|00001]	[110|00010]	[D0][D1]. . .[31]	
L:	paddpi	PV[ldx+lcx*64+PD 12],xp0	(111|00111)	[111|00000]	[111|00001]	[111|00010]	[D0][D1]. . .[63]	

Label:	(OP)	(A0)[A1][A2]	(MSiz|Accs)	[RSiz|RRRR]	[DSiz|Accs]	[Scal|JJJJ]	Displacement	Constant
L:	paddbw	xs0,MSiz[lip+DSiz 00012]	(MSiz|1111)	[RSiz|RRRR]	[DSiz|0010]		[D0][D1]. . .[DS]	
L:	paddbw	xs0,[lip+lcx*Scal+DS 12]	(MSiz|1111)	[RSiz|RRRR]	[DSiz|0011]	[Scal|0010]	[D0][D1]. . .[DS]	
L:	paddbw	MSiz[lip+DSiz 12],CS Cst	(MSiz|1111)	[CSiz|CNST]	[DSiz|0100]		[D0][D1]. . .[DS]	[C0][C1]. . .[CS]
L:	paddbw	MSiz[lip+lcx+SC+Dis],Cst	(MSiz|1111)	[CSiz|CNST]	[DSiz|0101]	[Scal|0010]	[D0][D1]. . .[DS]	[C0][C1]. . .[CS]
L:	paddbw	SVAL[lip+0000000012],xs0	(0101|1111)	[RSiz|RRRR]	[DSiz|0110]		[D0][D1]. . .[DS]	
L:	paddbw	SVAL[lip+lcx+SC+012],xs0	(0101|1111)	[RSiz|RRRR]	[DSiz|0111]	[Scal|0010]	[D0][D1]. . .[DS]	

2 Illustrierende Quellcodebeispiele

Es gibt hier einige Quellcodeschnipsel, die zeigen, wie DspLib in Assembler programmiert wird und aus Fortran und C heraus genutzt werden kann. Ein wesentlicher Teil einer kommenden Informatik-Doktorarbeit wäre, die ausgereifte Form zu dokumentieren. Das wäre ein Kompendium zu meinem DspLib-System, welches auch Basis der numerischen „Schloss Fünfeck“ Buchserie wäre. Daraus könnte sich ein, im universitären Umfeld, gerne und allgemein genutztes Programmiersystem entwickeln.

2.1 Einige Daten-Strukturen bei Assembler-Programmierung

Ein Computer verarbeitet Daten, welche in gespeicherter Form vorliegen. Am nächsten zum Prozessor sind die Register (LAX,LDX,...,L31;LIP). Dann kommt der Hauptspeicher welcher mit den „Memory“-Operanden angesprochen wird. Dann kommen Massenspeicher wie die Festplatten, Bandspeicher, Server und Clouds. Der Assembler-Programmierer spricht mit seinen Speicherzugriffen den sogenannten „virtuellen“ Speicher an, dessen maximale Größe von der Registerweite abhängt: Zum Beispiel vier Gigabyte beim 32 Bit Modus. Dieser virtuelle Speicher wird teilweise im physikalischen RAM gespeichert. Selten genutzte Speicherbereiche werden in die Auslagerungsdatei auf die Festplatte „weggelegt“, um möglichst viel physikalischen Speicher für schnelle Zugriffe frei zu halten. Als Assembler-Programmierer benutzen wir typischerweise folgende Speichergebiete: Erstens die zum Codemodul gehörigen Data-Segmente [LIP] und allokierten statischen Speicher [LBX]. Diese Speicherbereiche sind typischerweise statisch und geteilt. Deswegen müssen Schreibzugriffe bei Multithreading-Programmen sorgfältig kontrolliert werden. Schließlich gibt es den lokalen Speicher, der Teil des threadprivaten Stapelspeicher ist [LBP] und im Thread frei beschrieben werden kann.

```
%*=============================================================================================================
Label001: mov        eax,[lip+dat_EValue01]                  ; fetch EAX := EValue01 from DATA 1 Memory (LIP)
Label002: mov        eax,[lbx+sta_EValue01]                  ; fetch EAX := EValue01 from Static Memory (LBX)
Label003: mov        eax,[lbp+loc_EValue01]                  ; fetch EAX := EValue01 from Local  Memory (LBP)
Label004: mov        eax,[ldx+any_EValue01]                  ; fetch EAX := EValue01 from Any    Memory (ANY)
%*=============================================================================================================

;#############################################################################################################
BegDATA1                                                     ; Begin DATA 1 segment
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ALIGN     64                                                 ; Alignes Data segment to PVAL boundary
%*=============================================================================================================
dat_PValue01  dpi    01111222233334444555566667777888h       ; PVAL: Fixed sized Value with 512 Bits&  [00000]
dat_PValue02  dpi   -12345678901234567890123456789012345678901  ; PVAL: Fixed sized Value with 064 Bytes  [+0064]
%* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
dat_TValue01  dti    01111222233334444555566667777888811112222333344445555666677778888h               ; [+0128]
dat_TValue02  dti   -12345678901234567890123456789012345678901234567890123456789012345678901234567890  ; [+0160]
%* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
dat_SValue01  dsi    01111222233334444555566667777888h       ; SVAL: Fixed sized Value with 128 Bits&  [+0192]
dat_SValue02  dsi   -12345678901234567890123456789012345678901  ; SVAL: Fixed sized Value with 016 Bytes  [+0208]
%* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
dat_RValue01  dri    01111222233334444h                      ; RVAL: Fixed sized Value with 064 Bits&  [+0224]
dat_RValue02  dq    -12345678901234567890                    ; RVAL: Fixed sized Value with 008 Bytes  [+0232]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
dat_EValue01  dei    011112222h                              ; EVAL: Fixed sized Value with 032 Bits&  [+0240]
dat_EValue02  dd    -1234567890                              ; DVAL: Fixed sized Value with 004 Bytes  [+0244]
%*-------------------------------------------------------------------------------------------------------------
dat_WValue01  dwi    01111h                                  ; WVAL: Fixed sized Value with 016 Bits&  [+0248]
dat_WValue02  dw    -12345                                   ; WORD: Fixed sized Value with 002 Bytes  [+0250]
%*=============================================================================================================
dat_BValue01  dbi    011h                                    ; BVAL: Fixed sized Value with 008 Bits&  [+0252]
dat_BValue02  db    'ABC'                                    ; BYTE: Fixed sized Value with 001 Bytes  [+0253]
%*************************************************************************************************************
dat_QFloat01  dsf   +1.12345678901234567890123456789012345E-1234 ; QUAD: Fixed sized Float with 128 Bits&  [#0256]
dat_QFloat02  dsf   -1.12345678901234567890123456789012345E+1234 ; QUAD: Fixed sized Float with 016 Bytes  [+0016]
%* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
dat_DFloat01  drf   +1.1234567890123456E-123                ; DBLE: Fixed sized Float with 064 Bits&  [+0032]
dat_DFloat02  drf   -1.1234567890123456E+123                ; DBLE: Fixed sized Float with 008 Bytes  [+0040]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
dat_SFloat01  def   +1.1234567E-12                          ; SNGL: Fixed sized Float with 032 Bits&  [+0048]
dat_SFloat02  def   -1.1234567E+12                          ; SNGL: Fixed sized Float with 004 Bytes  [+0052]
%*-------------------------------------------------------------------------------------------------------------
dat_TmpFlt01  dsf   +1.1234567890123456789E-1234            ; TEMP: Fixed sized Float with 080 Bits&  [+0056]
dat_TmpFlt02  dsf   -1.1234567890123456789E+1234            ; TEMP: Fixed sized Float with 010 Bytes  [+0066]
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
EndDATA1                                                     ; Ended DATA 1 segment
;#############################################################################################################
```

```
;#########################################################################################################################
sta_PValue01              equ  mbk_sta+0000+000               ; PVAL: Fixed sized Value with 512 Bits&  [00000]
sta_PValue02              equ  sta_PValue01+64                ; PVAL: Fixed sized Value with 064 Bytes  [+0064]
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
sta_TValue01              equ  sta_PValue02+64                ; TVAL: Fixed sized Value with 256 Bits&  [+0128]
sta_TValue02              equ  sta_TValue01+32                ; TVAL: Fixed sized Value with 032 Bytes  [+0160]
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
sta_SValue01              equ  sta_TValue02+32                ; SVAL: Fixed sized Value with 128 Bits&  [+0192]
sta_SValue02              equ  sta_SValue01+16                ; SVAL: Fixed sized Value with 016 Bytes  [+0208]
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
sta_RValue01              equ  sta_SValue02+16                ; RVAL: Fixed sized Value with 064 Bits&  [+0224]
sta_RValue02              equ  sta_RValue01+08                ; RVAL: Fixed sized Value with 008 Bytes  [+0232]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
sta_EValue01              equ  sta_RValue02+08                ; EVAL: Fixed sized Value with 032 Bits&  [+0240]
sta_EValue02              equ  sta_EValue01+04                ; DVAL: Fixed sized Value with 004 Bytes  [+0244]
%*-------------------------------------------------------------------------------------------------------------------------
sta_WValue01              equ  sta_EValue02+04                ; WVAL: Fixed sized Value with 016 Bits&  [+0248]
sta_WValue02              equ  sta_WValue01+02                ; WORD: Fixed sized Value with 002 Bytes  [+0250]
%*=========================================================================================================================
sta_BValue01              equ  mbk_sta+0000+252               ; BVAL: Fixed sized Value with 008 Bits&  [#0252]
sta_BValue02              equ  sta_BValue01+01                ; BYTE: Fixed sized Value with 001 Bytes  [+0001]
sta_BValue03              equ  sta_BValue02+01                ; BYTE: Fixed sized Value with 001 Bytes  [+0002]
sta_BValue04              equ  sta_BValue03+01                ; BYTE: Fixed sized Value with 001 Bytes  [+0003]
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sta_LValue01              equ  mbk_sta+0256+000               ; LVAL: Any Value [00256 00256 00256 00256 00256]
sta_LValue02              equ  sta_LValue01+LS                ; PNTR: Any Value [+0004 +0008 +0016 +0032 +0064]
sta_LValue03              equ  sta_LValue02+LS                ; HNDL: Any Value [+0008 +0016 +0032 +0064 +0128]
sta_LValue04              equ  sta_LValue03+LS                ; ADDR: Any Value [+0012 +0024 +0048 +0096 +0192]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
sta_ByteSize              equ  sta_LValue04+LS                ; PARA: Structure [00272 00288 00320 00384 00512]
;#########################################################################################################################

;#########################################################################################################################
any_PValue01              equ  000+00000000+000               ; PVAL: Fixed sized Value with 512 Bits&  [00000]
any_PValue02              equ  any_PValue01+64                ; PVAL: Fixed sized Value with 064 Bytes  [+0064]
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
any_TValue01              equ  any_PValue02+64                ; TVAL: Fixed sized Value with 256 Bits&  [+0128]
any_TValue02              equ  any_TValue01+32                ; TVAL: Fixed sized Value with 032 Bytes  [+0160]
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
any_SValue01              equ  any_TValue02+32                ; SVAL: Fixed sized Value with 128 Bits&  [+0192]
any_SValue02              equ  any_SValue01+16                ; SVAL: Fixed sized Value with 016 Bytes  [+0208]
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
any_RValue01              equ  any_SValue02+16                ; RVAL: Fixed sized Value with 064 Bits&  [+0224]
any_RValue02              equ  any_RValue01+08                ; RVAL: Fixed sized Value with 008 Bytes  [+0232]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
any_EValue01              equ  any_RValue02+08                ; EVAL: Fixed sized Value with 032 Bits&  [+0240]
any_EValue02              equ  any_EValue01+04                ; DVAL: Fixed sized Value with 004 Bytes  [+0244]
%*-------------------------------------------------------------------------------------------------------------------------
any_WValue01              equ  any_EValue02+04                ; WVAL: Fixed sized Value with 016 Bits&  [+0248]
any_WValue02              equ  any_WValue01+02                ; WORD: Fixed sized Value with 002 Bytes  [+0250]
%*=========================================================================================================================
any_BValue01              equ  000+00000000+252               ; BVAL: Fixed sized Value with 008 Bits&  [#0252]
any_BValue02              equ  any_BValue01+01                ; BYTE: Fixed sized Value with 001 Bytes  [+0001]
any_BValue03              equ  any_BValue02+01                ; BYTE: Fixed sized Value with 001 Bytes  [+0002]
any_BValue04              equ  any_BValue03+01                ; BYTE: Fixed sized Value with 001 Bytes  [+0003]
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
any_LValue01              equ  000+00000256+000               ; LVAL: Any Value [00256 00256 00256 00256 00256]
any_LValue02              equ  any_LValue01+LS                ; PNTR: Any Value [+0004 +0008 +0016 +0032 +0064]
any_LValue03              equ  any_LValue02+LS                ; HNDL: Any Value [+0008 +0016 +0032 +0064 +0128]
any_LValue04              equ  any_LValue03+LS                ; ADDR: Any Value [+0012 +0024 +0048 +0096 +0192]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
any_ByteSize              equ  any_LValue04+LS                ; PARA: Structure [00272 00288 00320 00384 00512]
;#########################################################################################################################

;#########################################################################################################################
;23456789_12345678901234  equ  123456789_12345678901234+LngSiz ; #VAL: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;* WinCon.h                 SMALLRECT                          ; TYPE: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ! - - - - -
SmallRect_Left            equ  000000000                     ; WORD: 00000 00000 00000 00000 00000 ! SHORT
SmallRect_Topp            equ  SmallRect_Left+02             ; WORD: 00002 00002 00002 00002 00002 ! SHORT
SmallRect_Rght            equ  SmallRect_Topp+02             ; WORD: 00004 00004 00004 00004 00004 ! SHORT
SmallRect_Bott            equ  SmallRect_Rght+02             ; WORD: 00006 00006 00006 00006 00006 ! SHORT
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ! - - - - -
SMALLRECT                 equ  SmallRect_Bott+02             ; PARA: 00008 00008 00008 00008 00008 ! PARAMETER
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;23456789_12345678901234  equ  123456789_12345678901234+LngSiz ; #VAL: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;#########################################################################################################################
```

Im folgenden wird eine Subroutine mit lokalem StapelRahmenSpeicher (StackFrameMemory) betrachtet: Um die variable Registerbreite von 4 bis 64 Bytes möglichst elegant berücksichtigen zu können, hat sich folgendes Prinzip empfohlen: Alle Variablen mit fester Größe bilden den Anfang einer Datenstruktur. Diese sollte auf ein ganzzahliges Vielfaches von 64 enden. Danach kommen alle Pointer, Handles und sonstige LongVALues, die gemäß des Bitmodus 4,8,16,32 oder 64 Bytes groß sind. Damit skaliert die Strukturgröße nur am Ende mit dem Betriebsmodus. Der lokale BasePointer LBP wird üblicherweise auf die Position nach dem PUSHen der Register gesetzt. Der lokale Stapelrahmen geht dann ins Negative. Der Vorteil ist, dass dann einfach MOV lsp, lbp am Ende genutzt werden kann. Der Nachteil sind die negativen Offsets (Displacements), die im Listing schwer zu kontrollieren sind. Setzt man den Stapelzeiger auf das untere Stapelspeicherende, dann bekommen wir nur positive Offsets. Beim X86-32|64 haben wir einen Stapel der auf 4 oder 8 Bytes „aligned“ ist. Jedoch verlangen die 128 Bit XMM-Register einen auf 16 Bytes ausgerichteten Speicherzugriff. Deswegen wird das AND lbp, -16 eingesetzt. Der Nachteil dabei ist, dass nun nicht mehr mit dem BasePointer LBP auf die externen Argumente im Stackbereich zugegriffen werden kann. Es braucht dafür „blöde“ Hilfskonstruktionen die ab dem X86-128 entfallen würden. Beim X86-128 haben die Hauptregister und XMM-Register die gleiche Bitgröße und können daher beliebig auf den Stack gepushed werden. Alle BasePointer sollten 64 Byte-aligned sein, falls LVAL's auftreten.

```
;*****************************************************************************************************************
;* MyLocFun My own Local memory demonstration Function                                                           *
;*                                                                                                               *
;*===============================================================================================================*
;* MyLocFun takes as register input:                          LBX = mbk_   Pointer                               *
;*                                                            LCX = Old    Offset                                *
;* MyLocFun returns:    -                                     EAX = Code   Supposed                              *
;*                                                                                                               *
;* MyLocFun changes the registers:                            LAX,LDX,LCX,LDI,LSI,LBX,LBP,LSP                    *
;*****************************************************************************************************************
loc_PValue01              equ  000+00000000+000                ; PVAL: Fixed sized Value with 512 Bits&  [00000]
loc_PValue02              equ  loc_PValue01+64                 ; PVAL: Fixed sized Value with 064 Bytes  [+0064]
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
loc_TValue01              equ  loc_PValue02+64                 ; TVAL: Fixed sized Value with 256 Bits&  [+0128]
loc_TValue02              equ  loc_TValue01+32                 ; TVAL: Fixed sized Value with 032 Bytes  [+0160]
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
loc_SValue01              equ  loc_TValue02+32                 ; SVAL: Fixed sized Value with 128 Bits&  [+0192]
loc_SValue02              equ  loc_SValue01+16                 ; SVAL: Fixed sized Value with 016 Bytes  [+0208]
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
loc_RValue01              equ  loc_SValue02+16                 ; RVAL: Fixed sized Value with 064 Bits&  [+0224]
loc_RValue02              equ  loc_RValue01+08                 ; RVAL: Fixed sized Value with 008 Bytes  [+0232]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
loc_EValue01              equ  loc_RValue02+08                 ; EVAL: Fixed sized Value with 032 Bits&  [+0240]
loc_EValue02              equ  loc_EValue01+04                 ; DVAL: Fixed sized Value with 004 Bytes  [+0244]
%*---------------------------------------------------------------------------------------------------------------*
loc_WValue01              equ  loc_EValue02+04                 ; WVAL: Fixed sized Value with 016 Bits&  [+0248]
loc_WValue02              equ  loc_WValue01+02                 ; WORD: Fixed sized Value with 002 Bytes  [+0250]
%*===============================================================================================================*
loc_BValue01              equ  000+00000000+252                ; BVAL: Fixed sized Value with 008 Bits&  [#0252]
loc_BValue02              equ  loc_BValue01+01                 ; BYTE: Fixed sized Value with 001 Bytes  [+0001]
loc_BValue03              equ  loc_BValue02+01                 ; BYTE: Fixed sized Value with 001 Bytes  [+0002]
loc_BValue04              equ  loc_BValue03+01                 ; BYTE: Fixed sized Value with 001 Bytes  [+0003]
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
loc_SFBasPtr              equ  000+00000256+000                ; PNTR: SFrBasPtr [00256 00256 00256 00256 00256]
loc_SFTopPtr              equ  loc_SFBasPtr+LS                 ; PNTR: SFrTopPtr [+0004 +0008 +0016 +0032 +0064]
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
loc_LValue01              equ  loc_SFTopPtr+LS                 ; HNDL: Any Value [+0008 +0016 +0032 +0064 +0128]
loc_LValue02              equ  loc_LValue01+LS                 ; ADDR: Any Value [+0012 +0024 +0048 +0096 +0192]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
loc_SFBytSiz              equ  loc_LValue02+LS                 ; PARA: Structure [00272 00288 00320 00384 00512]
;#################################################################################################################
LOC_StackLAX              equ  000+00000000+000                ; LVAL: Stack LAX [00000 +0000 +0000 +0000 00000]
LOC_StackLDX              equ  LOC_StackLAX+LS                 ; LVAL: Stack LDX [+0004 +0008 +0016 +0032 +0064]
LOC_StackLCX              equ  LOC_StackLDX+LS                 ; LVAL: Stack LCX [+0008 +0016 +0032 +0064 +0128]
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
LOC_StackLDI              equ  LOC_StackLCX+LS                 ; PNTR: Stack LDI [+0012 +0024 +0048 +0096 +0192]
LOC_StackLSI              equ  LOC_StackLDI+LS                 ; PNTR: Stack LSI [+0016 +0032 +0064 +0128 +0256]
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
LOC_StackLBX              equ  LOC_StackLSI+LS                 ; PNTR: Stack LBX [+0020 +0040 +0080 +0160 +0512]
LOC_StackLBP              equ  LOC_StackLBX+LS                 ; PNTR: Stack LBP [+0024 +0048 +0096 +0192 +0384]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LOC_StackLIP              equ  LOC_StackLBP+LS                 ; PNTR: Stack LIP [+0028 +0056 +0112 +0224 +0448]
;*===============================================================================================================*
LOC_ExtArg01              equ  LOC_StackLIP+LS                 ; PNTR: ExARG(01) [+0032 +0064 +0128 +0256 +0512]
LOC_ExtArg02              equ  LOC_ExtArg01+LS                 ; LVAL: ExARG(02) [+0036 +0072 +0144 +0288 +0576]
;#################################################################################################################
```

Um externe Argumente auch innerhalb der Funktion adressierbar zu machen, könnte man beispielsweise das Register LSI auf die Adresse von StackLAX setzen. Falls das Register andersweitig nicht benötigt wird, wäre ein leichter Zugriff möglich. Alternativ könnte man die Adressen/Inhalte der externen Argumente in interne LVAL's kopieren. Jedoch bleibt dieses Prozedere ein Ärgernis, welches ab dem X86-128 verschwinden würde. Soll der Code rückwärtskompatibel sein, müssen diese Krücken benutzt werden, solange XMM oder X87 (64|80 Bit) Werte im Stackframe gespeichert werden.

```
;*=============================================================================================================
LOC_ExtArg01             equ  LOC_StackLIP+LS               ; PNTR: ExARG(01) [+0032 +0064 +0128 +0256 +0512]
LOC_ExtArg02             equ  LOC_ExtArg01+LS               ; LVAL: ExARG(02) [+0036 +0072 +0144 +0288 +0576]
;#############################################################################################################
MyPUBLIC  MyLocFun                                          ; Called from: MyCopStr,MyAddStr,MyDivStr, etc. !
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MyLocFun: push       lbp                                    ; Save: LBP => TOP[+0024 +0048 +0096 +0192 +0384]
          push       lbx                                    ; Save: LBX => TOP[+0020 +0040 +0080 +0160 +0320]
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          push       lsi                                    ; Save: LSI => TOP[+0016 +0032 +0064 +0128 +0256]
          push       ldi                                    ; Save: LDI => TOP[+0012 +0024 +0048 +0096 +0192]
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          push       lcx                                    ; Save: LCX => TOP[+0008 +0016 +0032 +0064 +0128]
          push       ldx                                    ; Save: LDX => TOP[+0004 +0008 +0016 +0032 +0064]
          push       lax                                    ; Save: LAX => TOP[00000 00000 00000 00000 00000]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LocFun@1: lea        lbp,[lsp-loc_SFBytSiz]                 ; Fetch LBP := TOP_LSP - Stack Frame Bytes Size
;* unused and        lbp, BC -016                           ; Andi: LBP &= <FFFF:FFFF|FFFF..FFFF|FFFF:FFF0>hx
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
;* unused mov        PNTR[lbp+loc_SFTopPtr],lsp             ; Store LSP => Stack Frame Topp Pointer
;* unused mov        PNTR[lbp+loc_SFBasPtr],lbp             ; Store LBP => Stack Frame Base Pointer
;*-------------------------------------------------------------------------------------------------------------
;* unused mov        lax,[lsp+LOC_ExtArg01]                 ; fetch LAX := ExternalArgument(01)
;* unused mov        LVAL[lbp+loc_IntArg01],lax             ; store LAX => InternalArgument(01)
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
;* unused mov        lax,[lsp+LOC_ExtArg02]                 ; fetch LAX := ExternalArgument(02)
;* unused mov        LVAL[lbp+loc_IntArg02],lax             ; store LAX => InternalArgument(02)
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LocFun@2: mov        lsp, lbp                               ; fetch LSP := Stack Frame Base Pointer
;*=============================================================================================================
LocFun@9: mov        eax,'LFu@'                             ; fetch EAX := LocalFunc @(beg)
;         mov        edx, 000003Dh                          ; 0000B-EAX-NN-DVAL-NwLn-B0Char
;         call            M1OutLib                          ; Write Out
;%        jmp        LJMP LocFunGT                          ; Goto: End
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

;#############################################################################################################
LocFunGT: mov        ecx,+00000001                          ; fetch ECX := +1 for "GT"
;% next   jmp        BJMP LocFunZ0                          ; condition to return "GT"
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
LocFunZ0: mov        eax,'LFuZ'                             ; fetch EAX := LocalFunc Z(end)
;         mov        edx, 000003Dh                          ; 0000B-EAX-NN-DVAL-NwLn-B0Char
;         call            M1OutLib                          ; Write Out
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;*        mov        edx, 00001500                          ; fetch EDX := 1500 MilliSecond
;*        call            S1MSleep                          ; call: SyMillisecondSleep(EDX)
;*=============================================================================================================
LocFunZ6: test       ecx, ecx                               ; Test: ECX as return condition
;*-------------------------------------------------------------------------------------------------------------
;* unused mov        lsp,[lbp+loc_SFTopPtr]                 ; Fetch LSP := (InStore) Stack Frame Topp Pointer
LocFunZ7: lea        lsp,[lbp+loc_SFBytSiz]                 ; Fetch LSP := BAS_LBP + Stack Frame Bytes Size
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LocFunZ8: pop        lax                                    ; fetch LAX := TOP[00000 00000 00000 00000 00000]
          pop        ldx                                    ; fetch LDX := TOP[+0004 +0008 +0016 +0032 +0064]
          pop        lcx                                    ; fetch LCX := TOP[+0008 +0016 +0032 +0064 +0128]
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          pop        ldi                                    ; fetch LDI := TOP[+0012 +0024 +0048 +0096 +0192]
          pop        lsi                                    ; fetch LSI := TOP[+0016 +0032 +0064 +0128 +0256]
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          pop        lbx                                    ; fetch LBX := TOP[+0020 +0040 +0080 +0160 +0320]
          pop        lbp                                    ; fetch LBP := TOP[+0024 +0048 +0096 +0192 +0384]
;*-------------------------------------------------------------------------------------------------------------
LocFunZ9: retn       LngSiz * 002                           ; fetch LIP := TOP[+0028 +0056 +0112 +0224 +0448]
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
EmLocFun: mov        DVAL[lbx+mbk_ErrorNum],edx             ; store EDX => mbk_ErrorNumbers
          lea        ldx,[lip+dat_F1LocFun]                 ; fetch LDX := MyLocFun ErrMssg
          mov        PNTR[lbx+mbk_F1FunNam],ldx             ; store LDX => FuncName Pointer
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;*        mov        eax, Value1                            ; fetch EAX := EMssgBox Value 1
;*        mov        ecx, Value2                            ; fetch ECX := EMssgBox Value 2
;*        mov        edx, Value3                            ; fetch EDX := EMssgBox Value 3
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
E1MsgBox: jmp        LJMP EMssgBox                          ; Error MessageBox
;************************************************************************************** end MyLocFun
```

2.2 Kurzes Quellcodebeispiel für Assembler

Wie oben erwähnt, habe ich eine Systematik beschrieben, die Speicherobjekte in strukturierter Form für alle Bit Moden (32, 64, 128, 256 und 512) konsistent ermöglicht, ohne übermäßig Speicherplatz zu verschwenden. Lediglich bei dem großen statischen Hauptspeicherobjekt bleibt die Frage nach der maximalen Registerweite gegen unnötige Speicherplatzverschwendung ungeklärt. Ein weiteres Problem sind die Datenstrukturen vom Betriebssystem: Wenn man die C-Include Dateien vom Windows API betrachtet, dann sieht man, dass viele Strukturen in gedankenloser Form Pointer und kürzere Datenelemente gemischt beinhalten. Entsprechend schwierig wird es, diese Datenelemente auf ihre jeweilige Größe hin korrekt auszurichten. Bei Strukturen, die nur Bytes, Words und DWords enthalten wäre die Struktur für alle Betriebsmoden gleich. Bei Strukturen mit Pointern und Handles dürfte sich ein nervtötender Prozess ergeben, um all die Strukturen für X86-32 und X86-64 richtig in Assembler gemeinschaftlich zu konstruieren. Die Erweiterung auf 128 Bit usw. dürfte auch spannend werden. Mit dem X86-32 glaubte man wohl lange ausgedient zu haben. Dann kam der X86-64 und jetzt soll das Ganze auch noch für einem X86-128 usw. passend gemacht werden. Es ist durchaus sinnvoll eine Systematik vom X86-256|512 auch gleich bei der Strukturbildung zu berücksichtigen.

```
;#####################################################################################################
Label001: mov       DVAL[ldx+lcx*16+123_12345678], 011112222h ; store CST => Memory
Label001: mov       DVAL[ldx+lcx*16+123_12345678],-1234567890 ; store CST => Memory
Label001: mov       DVAL[ldx+lcx*16+123_12345678],eax         ; store EAX => Memory
;* -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
Label001: mov       DVAL[ldx+lcx*16+12345678_12345678901],eax ; store EAX => Memory
Label001: mov       DVAL[ldx+lcx*16+12345678_12345678901],eax ; store EAX => Memory
Label001: mov       eax,[ldx+lcx*16+12345678_123456789012345] ; store CST => Memory
Label001: padd123456 xm0,[ldx+lcx*16+12345678_123456789012345] ; store CST => Memory
;#####################################################################################################
;2345678_123456789012345  equ  12345678_123456789012345+LngSiz ; #VAL: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;* WindowsLibrary              STRUCTURE                       ; TYPE: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ! - - - - -
Structure_MemoryPlace1    equ  000000000                       ; PNTR: 00000 00000 00000 00000 00000 ! Pointer
Structure_MemoryPlace2    equ  Structure_MemoryPlace1+LS       ; PNTR: 00004 00008 00016 00032 00064 ! Pointer
;* -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -  !   -    -
Structure_PictureLeft     equ  Structure_MemoryPlace2+LS       ; EVAL: 00008 00016 00032 00064 00128 ! int
Structure_PictureTopp     equ  Structure_PictureLeft+04        ; EVAL: 00012 00020 00036 00068 00132 ! int
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ! - - - - -
STRUCTURE                 equ  Structure_PictureTopp+04        ; SIZE: 00016 00024 00040 00072 00136 ! PARAMETER
;*====================================================================================================
;* WinCon.h                    CONSOLE_SCREEN_BUFFER_INFO      ; TYPE: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ! - - - - -
CSBuInfo_wSizeX           equ  00000000                        ; WVAL: 00000 00000 00000 00000 00000 ! SHORT
CSBuInfo_wSizeY           equ  CSBuInfo_wSizeX+02              ; WVAL: 00002 00002 00002 00002 00002 ! SHORT
CSBuInfo_wCursorPosX      equ  CSBuInfo_wSizeY+02              ; WVAL: 00004 00004 00004 00004 00004 ! SHORT
CSBuInfo_wCursorPosY      equ  CSBuInfo_wCursorPosX+02         ; WVAL: 00006 00006 00006 00006 00006 ! SHORT
CSBuInfo_wAttributes      equ  CSBuInfo_wCursorPosY+02         ; WVAL: 00008 00008 00008 00008 00008 ! WORD
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   - ! -   -   -
CSBuInfo_wWindowLeft      equ  CSBuInfo_wAttributes+02         ; WVAL: 00010 00010 00010 00010 00010 ! SHORT
CSBuInfo_wWindowTopp      equ  CSBuInfo_wWindowLeft+02         ; WVAL: 00012 00012 00012 00012 00012 ! SHORT
CSBuInfo_wWindowRght      equ  CSBuInfo_wWindowTopp+02         ; WVAL: 00014 00014 00014 00014 00014 ! SHORT
CSBuInfo_wWindowBott      equ  CSBuInfo_wWindowRght+02         ; WVAL: 00016 00016 00016 00016 00016 ! SHORT
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   - ! -   -   -
CSBuInfo_wMaxWinSizX      equ  CSBuInfo_wWindowBott+02         ; WVAL: 00018 00018 00018 00018 00018 ! SHORT
CSBuInfo_wMaxWinSizY      equ  CSBuInfo_wMaxWinSizX+02         ; WVAL: 00020 00020 00020 00020 00020 ! SHORT
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ! - - - - -
CSBUINFO                  equ  CSBuInfo_wMaxWinSizY+02         ; SIZE: 00022 00022 00022 00022 00022 ! PARAMETER
;*====================================================================================================
;* WindowsLibrary              STRCNAME                        ; TYPE: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ! - - - - -
StrcName_123456789012345  equ  00000000                        ; PNTR: 00000 00000 00000 00000 00000 ! Pointer
StrcName_123456789012345  equ  StrcName_123456789012345+LngSiz ; PNTR: 00004 00008 00016 00032 00064 ! Pointer
;* -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -  !   -    -
StrcName_123456789012345  equ  StrcName_123456789012345+LngSiz ; PNTR: 00008 00016 00032 00064 00128 ! Pointer
StrcName_123456789012345  equ  StrcName_123456789012345+LngSiz ; PNTR: 00012 00024 00048 00096 00192 ! Pointer
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   - ! -   -   -
StrcName_123456789012345  equ  StrcName_123456789012345+LngSiz ; PNTR: 00016 00032 00064 00128 00256 ! Pointer
StrcName_123456789012345  equ  StrcName_123456789012345+LngSiz ; PNTR: 00020 00040 00080 00160 00320 ! Pointer
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - !- - - - -
StrcName_123456789012345  equ  StrcName_123456789012345+LngSiz ; PNTR: 00024 00048 00096 00192 00384 ! Pointer
StrcName_123456789012345  equ  StrcName_123456789012345+LngSiz ; PNTR: 00028 00056 00112 00224 00448 ! Pointer
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ! - - - - -
STRCNAME                  equ  StrcName_123456789012345+LngSiz ; SIZE: 00032 00064 00128 00256 00512 ! PARAMETER
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;2345678_123456789012345  equ  12345678_123456789012345+LngSiz ; #VAL: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;#####################################################################################################
```

```
;*****************************************************************************************************
;* MyRMBoms My own Read Memory Unicode   ByteOrderMark                                               *
;*                                                                                                   *
;*===================================================================================================*
;*                                                                                                   *
;*NaCtrUni   Coding     Single Char      Multiple  Chars     ByteOrderMark                          *
;*                                                                                                   *
;*<U> 0 ->   Latin1     0000...0000FFh                      BOM   Ignored                          *
;*    1 -> _ Latin1   _ 0000...0000FFh _ $0100  .. $U0FFFF _ BOM _ Testing                          *
;*    2 ->   UTF-08     0000...00007Fh   000080h...10FFFFh   BOM   Ignored                          *
;*    3 -> _ UTF-08   _ 0000...00007Fh _ 000080h...10FFFFh _ BOM _ Testing                          *
;*    4 ->   UTF-16LE   0000...00FFFFh   010000h...10FFFFh   BOM   Ignored                          *
;*    5 -> _ UTF-16LE _ 0000...00FFFFh _ 010000h...10FFFFh _ BOM _ Testing                          *
;*    6 ->   UTF-32LE   0000...10FFFFh   Unused              BOM   Ignored                          *
;*    7 -> _ UTF-32LE _ 0000...10FFFFh _ Unused            _ BOM _ Testing                          *
;*    8 ->   UTF-16BE   0000...00FFFFh   010000h...10FFFFh   BOM   Testing                          *
;*    9 ->   UTF-32BE   0000...10FFFFh   Unused              BOM   Testing                          *
;*                                                                                                   *
;*    An NASTRG BOM can be used to change coding. Finding an BOM makes:                              *
;*                                                                                                   *
;*    NaChrCod  :=  3   when  UTF-08   BOM found   otherwise                                         *
;*                  5   when  UTF-16LE  BOM found                                                    *
;*                  7   when  UTF-32LE  BOM found   NaChrCod == NaCtrUni !                           *
;*                  8   when  UTF-16BE  BOM found                                                    *
;*                  9   when  UTF-32BE  BOM found   Processed as given                               *
;*                                                                                                   *
;*===================================================================================================*
;*                                                                                                   *
;* <U> Coding    | Reads       $1234 : BOM   | Writes      $1234 : BOM                               *
;* --------------+-------------+-----+-------+-------------+-----+------                             *
;*  0  Latin1    | 00..0000FFh : Not : Not   | 00..0000FFh : Not : Not                               *
;*  1  Latin1    | 00..10FFFFh : Yes : Tests | 00..10FFFFh : Yes : Not                               *
;*  2  UTF-08    | 00..10FFFFh : Not : Not   | 00..10FFFFh : Not : Not                               *
;*  3  UTF-08    | 00..10FFFFh : Not : Tests | 00..10FFFFh : Not : Write                             *
;* - - - - - - - + - - - - - - + - - + - - - + - - - - - - + - - + - - -                             *
;*  4  UTF-16-LE | 00..10FFFFh : Not : Not   | 00..10FFFFh : Not : Not                               *
;*  5  UTF-16-LE | 00..10FFFFh : Not : Tests | 00..10FFFFh : Not : Write                             *
;*  6  UTF-32-LE | 00..10FFFFh : Not : Not   | 00..10FFFFh : Not : Not                               *
;*  7  UTF-32-LE | 00..10FFFFh : Not : Tests | 00..10FFFFh : Not : Write                             *
;* - - - - - - - + - - - - - - + - - + - - - + - - - - - - + - - + - - -                             *
;*  8  UTF-16-BE | 00..10FFFFh : Not : Tests | 00..10FFFFh : Not : Write                             *
;*  9  UTF-32-BE | 00..10FFFFh : Not : Tests | 00..10FFFFh : Not : Write                             *
;*                                                                                                   *
;*===================================================================================================*
;* MyRMBoms takes as register input:           LBX = mbk_   Pointer                                  *
;*                                             LCX = Old    Offset                                   *
;* MyRMBoms returns:    LCX = New Offset       EAX = Code   Supposed                                 *
;*                                                                                                   *
;* MyRMBoms changes the registers:             EAX = Code   NewFound                                 *
;*****************************************************************************************************
;MB_RdBytSiz              equ  000+LngSiz*10                  ; LVAL: StackInput ARG2 [036 0072 0144 0288 0576]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RMB_RdBytSiz              equ  000+LngSiz*09                  ; LVAL: StackInput ARG2 [036 0072 0144 0288 0576]
RMB_RdStrPtr              equ  000+LngSiz*08                  ; PNTR: StackInput ARG1 [032 0064 0128 0256 0512]
;*===================================================================================================
RMB_StackLIP              equ  000+LngSiz*07                  ; PNTR: StackFrame LIP  [028 0056 0112 0224 0448]
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
RMB_StackLBP              equ  000+LngSiz*06                  ; PNTR: StackFrame LBP  [024 0048 0096 0192 0384]
RMB_StackLBX              equ  000+LngSiz*05                  ; PNTR: StackFrame LBX  [020 0040 0080 0160 0320]
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
RMB_StackLSI              equ  000+LngSiz*04                  ; PNTR: StackFrame LSI  [016 0032 0064 0128 0256]
RMB_StackLDI              equ  000+LngSiz*03                  ; PNTR: StackFrame LDI  [012 0024 0048 0096 0192]
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
RMB_StackLCX              equ  000+LngSiz*02                  ; LVAL: StackFrame LCX  [008 0016 0032 0064 0128]
RMB_StackLDX              equ  000+LngSiz*01                  ; LVAL: StackFrame LDX  [004 0008 0016 0032 0064]
RMB_StackLAX              equ  000+000000                     ; LVAL: StackFrame LAX  [000 0000 0000 0000 0000]
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%b_hPenNewD              equ  000-00000032+000               ; DVAL: hPen<New>Destination             [00000]
;%b_hPenOldD              equ  rmb_hPenNewD+16                ; DVAL: hPen<Old>Destination             [+0016]
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
RMB_M0000016              equ  000-00000016+000               ; PARA: Reads Memory ByOrdMrk            [-0016]
;MB_M0000032              equ  000-00000032+000               ; PARA: Reads Memory ByOrdMrk            [-0032]
;MB_M0000064              equ  000-00000064+000               ; PARA: Reads Memory ByOrdMrk            [-0064]
;MB_M0000128              equ  000-00000128+000               ; PARA: Reads Memory ByOrdMrk            [-0128]
;####################################################################################################
MyPUBLIC  MyRMBoms                                            ; Called from: MyCopStr,MyAddStr,MyDivStr, etc. !
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MyRMBoms: push       lbp                                      ; Save: LBP    [lsp+LongSize*6]
          push       lbx                                      ; Save: LBX    [lsp+LongSize*5]
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
```

```
          push       lsi                                        ; Save: LSI    [lsp+LongSize*4]
          push       ldi                                        ; Save: LDI    [lsp+LongSize*3]
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          push       lcx                                        ; Save: LCX    [lsp+LongSize*2]
          push       ldx                                        ; Save: LDX    [lsp+LongSize*1]
          push       lax                                        ; Save: LAX    [lsp+LongSize*0]
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RMemBoms: mov        lbp, lsp                                   ; Fetch LBP := StackFrameTopPtr
          lea        lsp,[lbp+RMB_M0000016]                     ; Addi: LSP += StackFrameBottom
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;MemChar: mov        lbp, lsp                                   ; Fetch LBP := StackFrameTopPtr
;* unused and        lbp, BC -016                               ; Andi: LBP &= <F:F|FFFF:FFF0>h
;* unused mov        PNTR[lbp+rmb_WLMbkPtr],lbx                 ; Store LBX => MemorBlockBasPtr
;* -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
;* unused mov        PNTR[lbp+rmb_WLRmcPtr],lbp                 ; Store LBP => StackFrameBasPtr
;* unused mov        PNTR[lbp+rmb_RestoLSP],lsp                 ; Store LSP => StackRestoLspPtr
;* unused lea        lsp,[lbp+RMB_M0000128]                     ; Addi: LSP += StackFrameBottom
;*=========================================================================================================
RMBoms@9: mov        eax,'RMB@'                                 ; fetch EAX := ReaMemBom @(beg)
;         mov        edx, 000003Dh                              ; 0000B-EAX-NN-DVAL-NwLn-B0Char
;         call           M1OutLib                               ; Write Out
;%        jmp        LJMP RMBomsGT                              ; Goto: End
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;MBoms00: test       lbx, lbx                                   ; test: LBX == 00000000 ?
;%        je         BJMP RMBoms01                              ; EQ -> Error: Mystical
;% -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
;%        cmp        lbx,[lbx+mbk_WLMbkPtr]                     ; test: LBX == WLMbkPtr ?
;%        je         BJMP RMBoms06                              ; EQ -> Pretty Successi
;% next   jmp        BJMP RMBoms01                              ; NE -> Error: Mystical
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;MBoms01: mov        edx, EnMystic                              ; ErrorNumber "Mystical"
;%        mov        ecx, 00000001                              ; ErrorLabel: "RMBoms01"
;%        call           EmRMBoms                               ; ErrorMessag "MyRMBoms"
;%        jmp        BJMP RMBomsLT                              ; Return with "LessThan"
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
RMBoms06: mov        lax,[lbp+RMB_RdBytSiz]                     ; fetch LAX := ReadChar BytSize
          sub        lax,[lbp+RMB_StackLCX]                     ; subr: LAX -= ReadChar Offsets
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          cmp        lax, BCST 004                              ; test: LAX == 00000004 ?
          jge        BJMP RMBoms08                              ; GE -> Pretty Successi
;* next   jmp        BJMP RMBoms07                              ; LT -> Error: Mystical
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RMBoms07: mov        edx, EnMystic                              ; ErrorNumber "Mystical"
          mov        ecx, 00000007                              ; ErrorLabel: "RMBoms07"
          call           EmRMBoms                               ; ErrorMessag "MyRMBoms"
          jmp        BJMP RMBomsEQ                              ; Return with "LessThan"
;*---------------------------------------------------------------------------------------------------------
RMBoms08: mov        ldi,[lbp+RMB_RdStrPtr]                     ; fetch LDI := ReadStrg Pointer
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          test       ldi, ldi                                   ; test: LDI == 00000000 ?
          jne        BJMP RMBoms10                              ; NE -> Pretty Successi
;* next   jmp        BJMP RMBoms09                              ; EQ -> Error: Mystical
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RMBoms09: mov        edx, EnMystic                              ; ErrorNumber "Mystical"
          mov        ecx, 00000009                              ; ErrorLabel: "RMBoms09"
          call           EmRMBoms                               ; ErrorMessag "MyRMBoms"
;* next   jmp        BJMP RMBomsLT                              ; Return with "LessThan"
;*=========================================================================================================
RMBomsLT: mov        ecx,-00000001                              ; fetch ECX := -1 for "LT"
          jmp        LJMP RMBomsZ0                              ; condition to return "LT"
;*=========================================================================================================
RMBoms10: mov        eax,'RMB1'                                 ; fetch EAX := ReaMemBom 000010
;*        mov        edx, 000003Dh                              ; 0000B-EAX-NN-DVAL-NwLn-B0Char
;*        call           M1OutLib                               ; Write Out
;* debugs jmp        LJMP RMBomsGT                              ; Goto: End
;*---------------------------------------------------------------------------------------------------------
RMBoms1@: mov        eax,[lbp+RMB_StackLAX]                     ; fetch EAX := Supposed ChrCode
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RMBoms11: cmp        eax, BCST 001                              ; test: EAX == 00,01;GT ?
          jb         LJMP RMBomsGT                              ; 00 -> Proces Latin1BY
          je         BJMP RB300000                              ; 01 -> Proces Latin1BY TestBOM
;* next   jmp        BJMP RMBoms13                              ; GT -> Search Continue $u12345
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
RMBoms13: cmp        eax, BCST 003                              ; test: EAX == 02,03;GT ?
          jb         LJMP RMBomsGT                              ; 02 -> Proces UTF-08BY
          je         BJMP RB300000                              ; 03 -> Proces UTF-08BY TestBOM
;* next   jmp        BJMP RMBoms15                              ; GT -> Search Continue
;* -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
RMBoms15: cmp        eax, BCST 005                              ; test: EAX == 04,05;GT ?
          jb         LJMP RMBomsGT                              ; 04 -> Proces UTF-16LE
          je         BJMP RB300000                              ; 05 -> Proces UTF-16LE TestBOM
```

```
;* next   jmp        BJMP RMBoms17                          ; GT -> Search Continue
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
RMBoms17: cmp        eax, BCST 007                          ; test: EAX == 06,07;GT ?
          jb         LJMP RMBomsGT                          ; 06 -> Proces UTF-32LE
          je         BJMP RB300000                          ; 07 -> Proces UTF-32LE TestBOM
;* next   jmp        BJMP RMBoms19                          ; GT -> Search Continue
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
RMBoms19: cmp        eax, BCST 009                          ; test: EAX == 08,09;GT ?
          jb         BJMP RB300000                          ; 08 -> Proces UTF-16BE TestBOM
          je         BJMP RB300000                          ; 09 -> Proces UTF-32BE TestBOM
;* next   jmp        BJMP RMBoms1E                          ; GT -> Error: Mystical
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RMBoms1E: mov        edx, EnMystic                          ; ErrorNumber "Mystical"
          mov        ecx, 00000019                          ; ErrorLabel: "RMBoms19"
          call            EmRMBoms                          ; ErrorMessag "MyRMBoms"
          jmp        LJMP RMBomsLT                          ; Return with "LessThan"
;*=============================================================================================================
RMBomsEQ: xor        ecx, ecx                               ; fetch ECX := 00 for "EQ"
          jmp        LJMP RMBomsZ0                          ; condition to return "EQ"
;#############################################################################################################
;# 100000 Proces Latin1       Non  $Escapes                ;                                                  #
;# 200000 Proces Latin1       Uses $Escapes                ;                                                  #
;# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
;# 300000 Proces UTF-08       With ByOrMark                ; 0xEFh,0xBBh,0xBFh                               #
;# 400000 Proces UTF-16-LE    With ByOrMark                ; 0xFFh,0xFEh                                     #
;# 500000 Proces UTF-32-LE    With ByOrMark                ; 0xFFh,0xFEh,0x00h,0x00h                         #
;# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
;# 600000 Proces UTF-16-BE    With ByOrMark                ; 0xFEh,0xFFh                                     #
;# 700000 Proces UTF-32-BE    With ByOrMark                ; 0x00h,0x00h,0xFEh,0xFFh                         #
;#############################################################################################################
RB300000: mov        eax,'3000'                             ; fetch EAX := Read Boms 300000
;*        mov        edx, 000003Dh                          ; 0000B-EAX-NN-DVAL-NwLn-B0Char
;*        call            M1OutLib                          ; Write Out
;* debugs jmp        LJMP RMBomsGT                          ; Goto: End
;*-------------------------------------------------------------------------------------------------------------
RB30000@: mov        lcx,[lbp+RMB_StackLCX]                 ; fetch LCX := OldMemChr Offset
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RB300001: cmp        BVAL[ldi+lcx+000],0EFh                 ; test: Byt[0] === 00EFh ?
          jne        BJMP RB500000                          ; NE -> May be UTF-16|32
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
RB300002: cmp        BVAL[ldi+lcx+001],0BBh                 ; test: Byt[1] === 00BBh ?
          jne        LJMP RMBomsGT                          ; NE -> Is not ByOrdMark
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
RB300003: cmp        BVAL[ldi+lcx+002],0BFh                 ; test: Byt[2] === 00BFh ?
          jne        LJMP RMBomsGT                          ; NE -> Is not ByOrdMark
;* next   jmp        BJMP RB300008                          ; EQ -> Is now UTF-08
;*-------------------------------------------------------------------------------------------------------------
RB300008: mov        eax, 0000003                           ; fetch LAX := UTF-08 (Bytes) 3
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          add        lcx, BC  003                           ; addi: LCX := OldMemCharOffs+3
          mov        LVAL[lbp+RMB_StackLCX],lcx             ; store LCX => NewMemCharOffset
          mov        LVAL[lbp+RMB_StackLAX],lax             ; store LAX => NewMemCharCoding
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RB300009: jmp        LJMP RMBomsGT                          ; Goto: Return with GreaterThan
;#############################################################################################################
;# 500000 Proces UTF-16-LE    With ByOrMark                ; 0xFFh,0xFEh                                     #
;#############################################################################################################
RB500000: mov        eax,'5000'                             ; fetch EAX := Read Boms 500000
;*        mov        edx, 000003Dh                          ; 0000B-EAX-NN-DVAL-NwLn-B0Char
;*        call            M1OutLib                          ; Write Out
;* debugs jmp        LJMP RMBomsGT                          ; Goto: End
;*-------------------------------------------------------------------------------------------------------------
;* is Ok! mov        lcx,[lbp+RMB_StackLCX]                 ; fetch LCX := OldMemChr Offset
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RB500001: cmp        BVAL[ldi+lcx+000],0FFh                 ; test: Byt[0] === 00FFh ?
          jne        BJMP RB800000                          ; NE -> May be UTF-16-BE
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
RB500002: cmp        BVAL[ldi+lcx+001],0FEh                 ; test: Byt[1] === 00FEh ?
          jne        BJMP RMBomsGT                          ; NE -> Is not ByOrdMark
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
RB500003: cmp        BVAL[ldi+lcx+002],000h                 ; test: Byt[2] === 0000h ?
          je         BJMP RB700004                          ; EQ -> May be UTF-32-LE
;* next ! jmp        BJMP RB500008                          ; NE -> Now is UTF-16-LE
;*-------------------------------------------------------------------------------------------------------------
RB500008: mov        eax, 0000005                           ; fetch LAX := UTF-16-LE Word-5
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          add        lcx, BC  002                           ; addi: LCX := OldMemCharOffs+2
          mov        LVAL[lbp+RMB_StackLCX],lcx             ; store LCX => NewMemCharOffset
          mov        LVAL[lbp+RMB_StackLAX],lax             ; store LAX => NewMemCharCoding
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RB500009: jmp        BJMP RMBomsGT                          ; Goto: Return with GreaterThan
```

```
;#########################################################################################################################
;# 700000 Proces UTF-32-LE    With ByOrMark                  ; 0xFFh,0xFEh,0x00h,0x00h                                  #
;#########################################################################################################################
;B700000: mov        eax,'7000'                              ; fetch EAX := Read Boms 700000
;*        mov        edx, 000003Dh                           ; 0000B-EAX-NN-DVAL-NwLn-B0Char
;*        call            M1OutLib                           ; Write Out
;* debugs jmp        LJMP RMBomsGT                           ; Goto: End
;*-------------------------------------------------------------------------------------------------------------------------
;* is Ok! mov        lcx,[lbp+RMB_StackLCX]                  ; fetch LCX := OldMemCharOffset
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;B700001: cmp        BVAL[ldi+lcx+000],0FFh                  ; test: Byt[0] === 00FFh ?
;* done ! jne        BJMP RMBomsGT                           ; NE -> Is not ByOrdMark
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
;B700002: cmp        BVAL[ldi+lcx+001],0FEh                  ; test: Byt[1] === 00FEh ?
;* done ! jne        BJMP RMBomsGT                           ; NE -> Is not ByOrdMark
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
;B700003: cmp        BVAL[ldi+lcx+002],000h                  ; test: Byt[2] === 0000h ?
;* done ! jne        BJMP RMBomsGT                           ; NE -> Is not ByOrdMark
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
RB700004: cmp        BVAL[ldi+lcx+003],000h                  ; test: Byt[3] === 0000h ?
          jne        BJMP RB500008                           ; NE -> Is now UTF-16-LE
;* next   jmp        BJMP RB700008                           ; EQ -> Is now UTF-32-LE
;*-------------------------------------------------------------------------------------------------------------------------
RB700008: mov        eax, 0000007                            ; fetch LAX := UTF-32-LE DWord7
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          add        lcx, BC  004                            ; addi: LCX := OldMemCharOffs+4
          mov        LVAL[lbp+RMB_StackLCX],lcx              ; store LCX => NewMemCharOffset
          mov        LVAL[lbp+RMB_StackLAX],lax              ; store LAX => NewMemCharCoding
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RB700009: jmp        BJMP RMBomsGT                           ; Goto: Return with GreaterThan
;#########################################################################################################################
;# 800000 Proces UTF-16-BE    With ByOrMark                  ; 0xFEh,0xFFh                                              #
;#########################################################################################################################
RB800000: mov        eax,'8000'                              ; fetch EAX := Read Boms 800000
;*        mov        edx, 000003Dh                           ; 0000B-EAX-NN-DVAL-NwLn-B0Char
;*        call            M1OutLib                           ; Write Out
;* debugs jmp        LJMP RMBomsGT                           ; Goto: End
;*-------------------------------------------------------------------------------------------------------------------------
;* is Ok! mov        lcx,[lbp+RMB_StackLCX]                  ; fetch LCX := OldMemChr Offset
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RB800001: cmp        BVAL[ldi+lcx+000],0FEh                  ; test: Byt[0] === 00FEh ?
          jne        BJMP RB900000                           ; NE -> May be UTF-32-BE
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
RB800002: cmp        BVAL[ldi+lcx+001],0FFh                  ; test: Byt[1] === 00FFh ?
          jne        BJMP RMBomsGT                           ; NE -> Is not ByOrdMark
;* next ! jmp        BJMP RB800008                           ; EQ -> Is now UTF-16-BE
;*-------------------------------------------------------------------------------------------------------------------------
RB800008: mov        eax, 0000008                            ; fetch LAX := UTF-16-BE Word-8
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          add        lcx, BC  002                            ; addi: LCX := OldMemCharOffs+2
          mov        LVAL[lbp+RMB_StackLCX],lcx              ; store LCX => NewMemCharOffset
          mov        LVAL[lbp+RMB_StackLAX],lax              ; store LAX => NewMemCharCoding
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RB800009: jmp        BJMP RMBomsGT                           ; Goto: Return with GreaterThan
;#########################################################################################################################
;# 900000 Proces UTF-32-BE    ByteOrderMark                  ; 0x00h,0x00h,0xFEh,0xFFh                                  #
;#########################################################################################################################
RB900000: mov        eax,'9000'                              ; fetch EAX := Read Boms 900000
;*        mov        edx, 000003Dh                           ; 0000B-EAX-NN-DVAL-NwLn-B0Char
;*        call            M1OutLib                           ; Write Out
;* debugs jmp        LJMP RMBomsGT                           ; Goto: End
;*-------------------------------------------------------------------------------------------------------------------------
;* is Ok! mov        lcx,[lbp+RMB_StackLCX]                  ; fetch LCX := OldMemChr Offset
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RB900001: cmp        BVAL[ldi+lcx+000],000h                  ; test: Byt[0] === 0x00h ?
          jne        BJMP RMBomsGT                           ; NE -> Is not ByOrdMark
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
RB900002: cmp        BVAL[ldi+lcx+001],000h                  ; test: Byt[1] === 0x00h ?
          jne        BJMP RMBomsGT                           ; NE -> Is not ByOrdMark
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
RB900003: cmp        BVAL[ldi+lcx+002],0FEh                  ; test: Byt[2] === 0xFEh ?
          jne        BJMP RMBomsGT                           ; NE -> Is not ByOrdMark
;* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
RB900004: cmp        BVAL[ldi+lcx+003],0FFh                  ; test: Byt[3] === 0xFFh ?
          jne        BJMP RMBomsGT                           ; NE -> Is not ByOrdMark
;* next   jmp        BJMP RB900008                           ; EQ -> Is now UTF-32-BE
;*-------------------------------------------------------------------------------------------------------------------------
RB900008: mov        eax, 0000009                            ; fetch LAX := UTF-32-BE DWord9
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          add        lcx, BC  004                            ; addi: LCX := OldMemCharOffs+4
          mov        LVAL[lbp+RMB_StackLCX],lcx              ; store LCX => NewMemCharOffset
```

```
         mov        LVAL[lbp+RMB_StackLAX],lax              ; store LAX => NewMemCharCoding
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;B900009: jmp       BJMP RMBomsGT                           ; Goto: Return with GreaterThan
;###################################################################################################
RMBomsGT: mov       ecx,+00000001                           ; fetch ECX := +1 for "GT"
;% next   jmp       BJMP RMBomsZ0                           ; condition to return "GT"
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
RMBomsZ0: mov       eax,'RMBZ'                              ; fetch EAX := ReaMemBom Z(end)
;         mov       edx, 000003Dh                           ; 0000B-EAX-NN-DVAL-NwLn-B0Char
;         call          M1OutLib                            ; Write Out
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;*        mov       edx, 00001500                           ; fetch EDX := 1500 MilliSecond
;*        call          S1MSleep                            ; call: SyMillisecondSleep(EDX)
;*==================================================================================================
RMBomsZ6: test      ecx, ecx                                ; Generate ECX return condition
;*--------------------------------------------------------------------------------------------------
RMBomsZ7: mov       lsp, lbp                                ; Restore: LSP StackFramTopPntr
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
RMBomsZ8: pop       lax                                     ; Restore: LAX [lsp+LongSize*0]
          pop       ldx                                     ; Restore: LDX [lsp+LongSize*1]
          pop       lcx                                     ; Restore: LCX [lsp+LongSize*2]
;*  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          pop       ldi                                     ; Restore: LDI [lsp+LongSize*3]
          pop       lsi                                     ; Restore: LSI [lsp+LongSize*4]
;*  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
          pop       lbx                                     ; Restore: LBX [lsp+LongSize*6]
          pop       lbp                                     ; Restore: LBP [lsp+LongSize*5]
;*--------------------------------------------------------------------------------------------------
RMBomsZ9: retn      LngSiz * 002                            ; And retn LIP [lsp+LongSize*7]
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
EmRMBoms: mov       DVAL[lbx+mbk_ErrorNum],edx              ; store EDX => mbk_ErrorNumbers
          lea       ldx,[lip+dat_F1RMBoms]                  ; fetch LDX := MyRMBoms ErrMssg
          mov       PNTR[lbx+mbk_F1FunNam],ldx              ; store LDX => FuncName Pointer
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;*        mov       eax, Value1                             ; fetch EAX := EMssgBox Value 1
;*        mov       ecx, Value2                             ; fetch ECX := EMssgBox Value 2
;*        mov       edx, Value3                             ; fetch EDX := EMssgBox Value 3
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
E1MsgBox: jmp       LJMP EMssgBox                           ; Error MessageBox
;******************************************************************************************** end MyRMBoms

;###################################################################################################
;23456789012345678901234567890123452  equ  0x11112222_33334444h ; 123456789012345678901234567890123452  011112222h
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12345678901234567890123456789012345  equ -12345678901234567890 ; define 1234567890123456789012345678  011112222h
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
12345678901234567890123456789012345  equ -12345678901234567890 ; def 1234567890123456789012345678901  011112222h
;*  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
12345678901234567890123456789012345  equ -12345678901234567890 ; 12345678901234567890123456789012345  011112222h
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
12345678901234567890123456789012345  equ -12345678901234567890 ; 12345678901234567890123456789012345  011112222h
;* -    -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
12345678901234567890123456789012345  equ -12345678901234567890 ; 12345678901234567890123456789012345  011112222h
;###################################################################################################
;2345678_123456789012345  equ  12345678_123456789012345+LngSiz ; #VAL: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;* WindowsLibrary             STRUCTURE                     ; TYPE: Int32 Amd64 IA128 IA256 IA512 ! CPP-TYPE:
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ! - - - - -
Structure_MemoryPlace1    equ  000000000                    ; PNTR: 00000 00000 00000 00000 00000 ! Pointer
Structure_MemoryPlace2    equ  Structure_MemoryPlace1+LS    ; PNTR: 00004 00008 00016 00032 00064 ! Pointer
;* -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    - !    -    -
Structure_PictureLeft     equ  Structure_MemoryPlace2+LS    ; EVAL: 00008 00016 00032 00064 00128 ! int
Structure_PictureTopp     equ  Structure_PictureLeft+04     ; EVAL: 00012 00020 00036 00068 00132 ! int
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ! - - - - -
STRUCTURE                 equ  Structure_PictureTopp+04     ; SIZE: 00016 00024 00040 00072 00136 ! PARAMETER
;###################################################################################################
Label123: mov       eax,[ldx+Structure_MemoryPlace1]        ; fetch EAX := [ldx+Structure_MemoryPlace1]
          mov       eax, BC 123456789012345678901234567890123h ; oring EAX &= 123456789012345678901234567890123h
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          mov       eax, 12345678901234567890123456789012345h ; EAX&= 12345678901234567890123456789012345h
          mov       eax,[ldx+lcx*16+12345678_123456789012345] ; EAX:= [ldx+lcx*16+12345678_123456789012345]
;###################################################################################################
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;***************************************************************************************************
;*==================================================================================================
;*--------------------------------------------------------------------------------------------------
;* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
;* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
;* -    -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
;###################################################################################################
```

2.3 Kurzes Quellcodebeispiel für Fortran

DspLib-Programme werden typischerweise in Fortran geschrieben. Dabei soll der Fortran-Stil nahe am Fortran77 ausgerichtet sein, damit der Code auch mit alten frei downloadbaren Compilern genutzt werden kann. Die hier gezeigten Beispiele sind veraltet und unvollständig !

```
!*****************************************************************************
!*  Function  ERROR = DGAUSS(N1,N2A,N2AB,DMATR,DMATI,EPS,IFLAG,CONTRL,     *
!*                           I1NEW,I2NEW,RANG,DETR,DETI)                    *
!*****************************************************************************
      FUNCTION  DGAUSS(N1,N2A,N2AB,DMATR,DMATI,EPS,IFLAG,CONTRL,         & !
     &                 I1NEW,I2NEW,RANG,DETR,DETI)
      IMPLICIT  NONE
!     INCLUDE    "VODL.INC"
!*=================================================================== PAR
      INTEGER*4  OUTUNI  ,I1MAX,I2MAX
      PARAMETER( OUTUNI=9,I1MAX=200,I2MAX=200)
!*=================================================================== VAR
      INTEGER*4  DGAUSS,N1,N2A,N2AB,IFLAG,CONTRL,I12,I1,I2,J1,J2,        & !
     &           I1NEW(I1MAX),I2NEW(I2MAX),RANG
!*-------------------------------------------------------------------------
      REAL*8     DMATR(I1MAX,I2MAX),DMATI(I1MAX,I2MAX),DETR,DETI,        & !
     &           DAUX1,DAUX2,D1SGN,D2SGN,DF12R,DF12I,DFACR,DFACI,EPS
!*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DAT
      DATA  D1SGN,D2SGN                                                  & !
     &   /   1.D0, 1.D0 /
      DGAUSS = 555
!*############################################################################ EXE
      DO I1=1,I1MAX ; I1NEW(I1) = I1 ; ENDDO  ;  DETR = 1.D0
      DO I2=1,I2MAX ; I2NEW(I2) = I2 ; ENDDO  ;  DETI = 0.D0
!*-------------------------------------------------------------------------
      DO I12=1,MIN(N1,N2A)
!*=========================================================================
       DO I1=1,N1
        IF( (I1.LT.I12 .AND. CONTRL.LT.2) .OR. I1.EQ.I12) CYCLE
!* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        IF(IFLAG.EQ.0)THEN
          DFACR = DF12R * DMATR(I1,I12)
        ELSE
          DFACR = DF12R * DMATR(I1,I12) - DF12I * DMATI(I1,I12)
          DFACI = DF12I * DMATR(I1,I12) + DF12R * DMATI(I1,I12)
        ENDIF
!* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
        DO I2=I12,N2AB
         IF(IFLAG.EQ.0)THEN
           DMATR(I1,I2) = DMATR(I1,I2) - DFACR * DMATR(I12,I2)
         ELSE
           DAUX1 = DMATR(I12,I2) ; DAUX2 = DMATI(I12,I2)
           DMATR(I1,I2) = DMATR(I1,I2) - DFACR * DAUX1 + DFACI * DAUX2
           DMATI(I1,I2) = DMATI(I1,I2) -(DFACI * DAUX1 + DFACR * DAUX2)
         ENDIF
        ENDDO
!* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
       ENDDO
!* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
       DAUX1 = DETR
       DETR  = DETR * DMATR(I12,I12) -  DETI * DMATI(I12,I12)
       DETI  = DETI * DMATR(I12,I12) + DAUX1 * DMATI(I12,I12)
!*-------------------------------------------------------------------------
       IF(CONTRL.EQ.3)THEN
         DO I2=I12,N2AB
          IF(IFLAG.EQ.0)THEN
            DMATR(I12,I2) = DF12R * DMATR(I12,I2)
          ELSE
            DAUX1 = DMATR(I12,I2) ; DAUX2 = DMATI(I12,I2)
            DMATR(I12,I2) = DF12R * DAUX1 - DF12I * DAUX2
            DMATI(I12,I2) = DF12I * DAUX1 + DF12R * DAUX2
          ENDIF
         ENDDO
       ENDIF
!*=========================================================================
      ENDDO
!*-------------------------------------------------------------------------
      DAUX1 = D1SGN*D2SGN ; DETR = DETR*DAUX1 ; DETI = DETI*DAUX1
      RETURN
!*############################################################################ END
      END
!*************************************************************** END DGAUSS
```

```
!*****************************************************************************************************************
!*    IVALU1 = IS4(IVALU2,IVALU3,IVALU4,-111111,-222222,-333333,-444444) !!!                                     *
!*                                                                                                               *
!*    FUNCTION Integer Set4() serves for Fortran F77 compatible setting of                                       *
!*    multiple Integer VALUes per  line  without F90 compatible ";" usage.                                       *
!*****************************************************************************************************************
      FUNCTION      IS4(IVALU2,IVALU3,IVALU4,                                & !
     &           IDATA1,IDATA2,IDATA3,IDATA4)                                !!!
!################################################################################################################ DECLARATION
      IMPLICIT   NONE                                                       !!!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VARIABLES
      INTEGER*4     IS4,IVALU2,IVALU3,IVALU4,                               & !
     &           IDATA1,IDATA2,IDATA3,IDATA4                                !!!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COMMON
!%    COMMON /COMBL2/ DSUMR,DSUMI                                         !!!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% STATEMENT FUNCTIONS
!%    DMAT1R(I,J) = SQRT( (+10.D0)**J / DBLE(MAX(I,1)) )                  !!!
!################################################################################################################ EXECUTABLE
      IS4    = IDATA1                                                       !!!
      IVALU2 = IDATA2                                                       !!!
      IVALU3 = IDATA3                                                       !!!
      IVALU4 = IDATA4                                                       !!!
!################################################################################################################
      RETURN                                                                !!!
      END                                                                   !!!
!********************************************************************************************************** END IS4

!*****************************************************************************************************************
!*    SVALU1 = SS4(SVALU2,SVALU3,SVALU4,+11.000,-12.000,+13.000,-14.000) !!!                                     *
!*                                                                                                               *
!*    FUNCTION Single Set4() serves for Fortran F77 compatible setting of                                        *
!*    multiple Single VALUes per  line  without F90 compatible ";" usage.                                        *
!*****************************************************************************************************************
      FUNCTION     SS4(SVALU2,SVALU3,SVALU4,SDATA1,SDATA2,SDATA3,SDATA4) !!!
!################################################################################################################ DECLARATION
      IMPLICIT     NONE                                                     !!!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VARIABLES
      REAL*4        SS4,SVALU2,SVALU3,SVALU4,                               & !
     &           SDATA1,SDATA2,SDATA3,SDATA4                                !!!
!################################################################################################################ EXECUTABLE
10001 SS4    = SDATA1                                                       !!! SS4    = SNGL(+11.000)
!* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
10002 SVALU2 = SDATA2                                                       !!! SVALU2 = SNGL(-12.000)
!* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
10003 SVALU3 = SDATA3                                                       !!! SVALU3 = SNGL(+13.000)
!* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
10004 SVALU4 = SDATA4                                                       !!! SVALU4 = SNGL(-14.000)
!* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!*005 SVALU5 = SDATA5                                                      !!! SVALU5 = SNGL(-14.000)
!################################################################################################################
      RETURN                                                                !!!
      END                                                                   !!!
!********************************************************************************************************** END SS4

!################################################################################################################
1     STRG32 = '@             '                                             !!!
!* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
      ERROR1 = DL1(GETINP,000001,TOKCNT,PAGCNT,LINCNT,COLCNT,               & !
     &         1000910000,002560,000000,002560,NASOFF,NASCNT,               & !
     &         1060916168,000032,000000,000032,DASOFF,DASCNT,               & !
     &             000032,HLMEN3,000012,000012,000010,BASFND,               & !
     &             STRG32,'&p&n'                                          // & !
!* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
     &'&s HLMEN1:  MENU #.#.#.# Demos   xx                       &n&n'// & !
     &'&s 3.6.1.n  Menu 1.X.X.X Demos                             &n'// & !
     &'&s 3.6.2.n  Menu 2.X.X.X Demos                             &n'// & !
     &'&s 3.6.3.n  Menu 3.X.X.X Demos                            &n&n'// & !
     &'&E HLMEN3@                                                    ')  !!!
!################################################################################################################
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
!*****************************************************************************************************************
!*=============================================================================================================*
!*-------------------------------------------------------------------------------------------------------------*
!* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
!* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
!* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
!################################################################################################################
```

2.4 Kurzes Quellcodebeispiel für C/CPP und LaTeX

DspLib-Programme können auch im Umfeld von C/CPP genutzt werden. Ich habe dazu einfache Testprogramme geschrieben, welche funktionstüchtig waren. Gegenwärtig benutze ich nur Fortran zum Weiterentwickeln von DspLib. Deswegen sind die C-Beispiele hier nur von halbherziger Reife. Im gereiften Stadium sollen C Beispielprogramme ausgearbeitet werden, die in einem Fortran-artigen Stil geschrieben sind und ebenso leicht verstanden und für eigene Anwendungen weiterbearbeitet werden können. Beidesmal entfällt die Auseinandersetzung mit dem Windows API.

Um textuelle Veröffentlichungen zu schreiben ist pdfLaTeX sehr zu empfehlen. Mit Dsplib kann man Bilder im JPG- oder PNG-Format speichern und CSV-Datentextdateien erzeugen. Das lässt sich prima in LaTeX einbinden. Man kann aber auch beliebige andere Textsatzsystem benutzen.

```
//=========================================================================
//     error = DL1(OpnWin,winnum,lftedg,topedg,winwdt,winhgt,contrl,
//                 wtitle,bmifil,fnthnd,dummy1,winsrc,bmpwdt,bmphgt)
//=========================================================================
       winnum = 1 ; lftedg= 670 ; topedg=   0 ; winwdt=350 ; winhgt=250 ;
       contrl = 100000 ; *owtitl= "01 Window CPP @" ;
       fnthnd = 0      ; *cfgfil= "D1OpnWin.ini@" ;
       dummy1 = 0  ; winsrc = 0 ; bmpwdt = 1024 ; bmphgt = 1024 ;
       error = DL1(OpnWin,winnum,lftedg,topedg,winwdt,winhgt,contrl,
                 *owtitl,&MYNULL,fnthnd,dummy1,winsrc,bmpwdt,bmphgt);
//-------------------------------------------------------------------------
       dest  = 0;
       error = DL1(DrwLin,dest,20,20,220,135,100000,1);
//     printf("DrwLin: error = %10.8d %9.8x\n\r",error,error);
       error = DL1(DrwLin,dest,20,12,260,145,100000,2);
//     printf("DrwLin: error = %10.8d %9.8x\n\r",error,error);
       for(i=1;i<128;i++) error=DL1(DrwPxl,00,i*3+20,i*2+26,0,i) ;
       for(i=1;i<32;i++)  error=DL1(DrwLin,00,i*9+20,120,i*9+26,135,0,i);
       for(i=1;i<32;i++)  error=DL1(DrwFrm,00,i*9+20,140,i*9+26,155,0,i);
       for(i=1;i<32;i++)  error=DL1(DrwBlk,00,i*9+20,160,i*9+26,175,0,i);
       dest  = 0 ; x = 200 ; y = 100 ; contrl = 10000 ; apen = 11 ;
       fnthnd= 0 ; *txtstr = "Hallo CPP !@" ;
       error = DL1(DrwTxt,dest,x,y,contrl,apen,fnthnd,*txtstr);
//-------------------------------------------------------------------------
       for(k = 1 ; k <= 350 ; k++){
        contrl = 100000 ; fnthnd = -2 ;
        error = DL1(GetMsg,&msgnum,&gmdest,&gmcmit,&mousex,&mousey,&contrl,
                   &igmval,&sgmval,&dgmval,egmval,gmstrg,&fnthnd,&dlbcbk);
        dest = 0 ;

        if(msgnum != -10)
          printf(" GetMsg: %4d %8.6d %9.8x %4d CB()= %5d %5d %5d %5d\n",
          k,msgnum,msgnum,gmcmit,
          dlbcbk[0],dlbcbk[1],dlbcbk[2],dlbcbk[3]);
        for(i=1 ; i <=1000000*50 ; i++);
        if(msgnum == 110104) break;

        error = DL1(DrwLin,dest,20,20,220,135,100000,1);
        error = DL1(DrwFrm,dest,20,12,260,145,100000,2);
        sprintf(*valstr,"GetMsg = %4d %12.6d @",k,msgnum);
        error = DL1(DrwTxt,dest,10,165,10000,k,0,*valstr);

        if(k % 2 == 0) error = DL1(DrwLin,dest,k,175,k,195,100000,k);
                  else error = DL1(DrwLin,dest,k,190,k,200,100000,k);

        if(msgnum == 10101){
          winnum = 1 ; *bmifil = "C1ClsWin.bmi#" ; contrl = 100001 ;
          error = DL1(ClsWin,winnum,*bmifil,contrl);
          printf(" ClsWin: error = %10.8d %9.8x\n\r",error,error);
          error = DL1(GetInp,0,10,00,12,1,&i,*gistrg,"OK ? (RETURN) = @");}
        else if(msgnum == 110101 || msgnum == 110111){
          dspnum = 1 ; *bmifil = "C1ClsDsp.bmi#" ; contrl = 100001 ;
          error = DL1(ClsDsp,dspnum,*bmifil,contrl) ;
          printf(" ClsWin: error = %10.8d %9.8x\n\r",error,error);
          error = DL1(GetInp,0,10,00,12,1,&i,*gistrg,"OK ? (RETURN) = @");}
        else if(msgnum == 120101 || msgnum == 120111){
          dspnum = 2 ; *bmifil = "C2ClsDsp.bmi#" ; contrl = 0001 ;
          error = DL1(ClsDsp,dspnum,*bmifil,contrl) ;
          printf(" ClsWin: error = %10.8d %9.8x\n\r",error,error);
          error = DL1(GetInp,0,10,00,17,1,&i,*gistrg,"OK ? (RETURN) = @");}
        if(msgnum == 110601) error = DL1(SetABM,12,1000);
        if(msgnum == 110602) error = DL1(SetABM,12,1001); }
```

```
//*****************************************************************************************************************
//    SVALU1 = SS4(SVALU2,SVALU3,SVALU4,+11.000,-12.000,+13.000,-14.000) !!!                                      *
//                                                                                                                *
//    FUNCTION Single Set4() serves for Fortran F77 compatible setting of                                         *
//    multiple Single VALUes per  line  without F90 compatible ";" usage.                                         *
//*****************************************************************************************************************
      FUNCTION     SS4(SVALU2,SVALU3,SVALU4,SDATA1,SDATA2,SDATA3,SDATA4)  //
//################################################################################################## DECLARATION
      IMPLICIT     NONE                                                                            //
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VARIABLES
      REAL*4        SS4,SVALU2,SVALU3,SVALU4,                                                         //
     &           SDATA1,SDATA2,SDATA3,SDATA4                                                        //
//################################################################################################## EXECUTABLE
10001 SS4    = SDATA1                                                                               // SS4    = SNGL(+11.000)
// -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
10002 SVALU2 = SDATA2                                                                               // SVALU2 = SNGL(-12.000)
// -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
10003 SVALU3 = SDATA3                                                                               // SVALU3 = SNGL(+13.000)
// -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
10004 SVALU4 = SDATA4                                                                               // SVALU4 = SNGL(-14.000)
//#####################################################################################################################
      RETURN                                                                                        //
      END                                                                                           //
//************************************************************************************************ END SS4

//#####################################################################################################################
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//*********************************************************************************************************************
//=====================================================================================================================
//---------------------------------------------------------------------------------------------------------------------
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
// -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
// -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
// -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
//#####################################################################################################################

%=====================================================================================================================
\newpage\noindent
Das Qualitätsproblem ist ausschließlich ein Kostenproblem ! Im folgenden ist eine Rezeptur für einen sehr
hochwertigen Bubble Gum. Dessen Stückgröße sei 7.5 bis 8.0 Gramm. Dabei wurden 25 \% Gumbase benutzt.
\par\vspace{4mm}\noindent
%*-------------------------------------------------------------------------------------------------------------------
\setlength{\tabcolsep}{1.7mm}
\small\footnotesize%\scriptsize%\tiny
\begin{tabular}{|l||r|r|r|r||r|r|r|r|}\hline
%* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Zutatenkomponenten           & Anteil  & Masse & Preis & ProKilo & Charge & Tag  & Woche & Monat &  Pro Jahr  \\
\hline %*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Polyisobuten (+Isopren)  20 \% &  5 \% &  0,40 &       &         &  15 Kg &   60 &   300 &  1200 &  15.000 Kg \\
PolyvinylAcetat/Alkohol  28 \% &  7 \% &  0,56 &       &         &  21 Kg &   84 &   420 &  1680 &  21.000 Kg \\
Kolophonium-Harze        20 \% &  5 \% &  0,40 &       &         &  15 Kg &   60 &   300 &  1200 &  15.000 Kg \\
Wachse und Pflanzenöle   12 \% &  3 \% &  0,24 &       &         &   9 Kg &   36 &   180 &   720 &   9.000 Kg \\
Calciumkarbonat/Talkum   20 \% &  5 \% &  0,40 &       &         &  15 Kg &   60 &   300 &  1200 &  15.000 Kg \\
\hline %*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Puderzucker                  & 50 \% &  4,00 & 0,260 & 65 Cent & 150 Kg &  600 & 3.000 &12.000 & 150.000 Kg \\
Glukosesirup                 & 23 \% &  1,84 &       &         &  69 Kg &  276 & 1.380 & 5.520 &  69.000 Kg \\
Glycerin und Antioxidantien  &  1 \% &  0,08 &       &         &   3 Kg &   12 &    60 &   240 &   3.000 Kg \\
Aromen und Farben            &  1 \% &  0,08 &       &         &   3 Kg &   12 &    60 &   240 &   3.000 Kg \\
\hline %*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Summen     ( Kaumasse: 25 \% ) &100 \% &  8,00 &       &         & 300 Kg & 1200 & 6.000 &24.000 & 300.000 Kg \\
\hline\hline %*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Wachspapier ~ (Quadratmeter)   &       &  0,01 &       &         & 400 qm &      &       &       &            \\
Lohnkosten  ~ ~ (Euro)         &       &       &       &         &        &      &       &       &            \\
Betriebskosten  (Euro)         &       &       &       &         &        &      &       &       &            \\
\hline %*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Summen                         &       &       &       &         &        &      &       &       &            \\
%* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
\hline\end{tabular}\normalsize \\[4mm]
%#####################################################################################################################
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%*********************************************************************************************************************
%*=====================================================================================================================
%*---------------------------------------------------------------------------------------------------------------------
%* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
%* -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
%* -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
%* -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -     -
%#####################################################################################################################
```

3 Grundlagen der Assemblerprogrammierung

3.1 Zahlenformate

3.2 Operationen

3.3 Adressierungen

4 Praktisches Programmieren

4.1 Quellcodeformate

4.2 Deklarationen

4.3 Funktionen und Subroutinen

4.4 Verzweigungen

5 Und noch mehr für meine Doktorarbeit

5.1 Windows API

5.2 Threads und Speicherplatznutzung

5.3 Gemischte Programmierung in Fortran, C und Assembler

5.4 GUI Design im Rahmen von DspLib-Programmen