Basic-PEARL auf dem MC68000 Design-Module

Prof. Dr.-Ing. W. Gerth, Hannover

Zusammenfassung:

Es wird ein komplettes Basic-PEARL Softwaresystem für den MC68000-Mikrorechner beschrieben. Durch virtuelle Codierung der Übersetzer und einen halbvirtuellen Laufzeitcode läßt sich die komplette Systemsoftware resident im E-PROM-Bereich z.B. des MEX68KDM unterbringen.

Summary:

In the paper a software-package for the MEX68KDM design module is described, which enables the user to compile and execute programs written in Basic-PEARL on the single-board computer itself. Due to the virtual coding of the compiler and a semi-virtual run-time-code the total package fits into the E-PROM storage of the board.

1. Einleitung

In einem früheren Beitrag [1] wurde ein Konzept für ein Basic-PEARL System für den Prozeßrechner Krantz Mulby 3/35 beschrieben, welches seit Anfang 1980 realisiert ist und von vornherein für eine spätere Anpassung an einen noch auszuwählenden Mikrocomputer geplant war. Ziel des hier beschriebenen Ansatzes ist die "intelligente Steckkarte", bei der das gesamte Software-Paket im Festwertspeicher auf dem Board Platz findet. Bei der Auswahl eines geeigneten Mikroprozessors wurde das Schwergewicht auf die Leistungsfähigkeit bei der Objekt adressierung gelegt. Treibende Kraft des Projektes ist ausschließlich der Wunsch nach einer preiswerten Anwendung von Basic-PEARL im Institutsbereich.

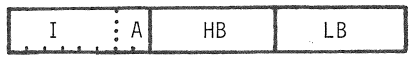
2. Randbedingungen

Bei Beginn der Arbeiten im Frühjahr 1981 existierte lediglich das lauffähige Mulby-PEARL und ein Modul MEX68KDM von Motorola. Nicht einmal ein MC68000 Assembler war vorhanden. Für die Durchführung des Projektes standen weder Mittel noch Mitarbeiter zur Verfügung, und es mußte auf Diplomkandidaten [2],[3] der Elektrotechnik zurückgegriffen werden. Für das Projekt von Nachteil ist dabei, daß bei 6 Monaten Gesamtdauer mehrere Monate für die Einarbeitung verlorengehen.

3. Aufbau der Übersetzer

3.1 Konzept der virtuellen Codierung

Zunächst mußte ein Assembler für den MC68000 geschrieben werden, der direkt auch für Gastrechner brauchbar ist. Schon in [1] wurde ein virtueller Code beschrieben, der sich recht effektiv durch einen statusfreien ca. 2KB großen Interpreter (VCP = virtual code processor) exekutieren läßt. Jede VCP-Instruktion besteht aus 1 bis 3 Bytes:



(1)

Das 2-bit-Feld A bezeichnet die Adressierungsarten direkt, indirekt (Memory), Index 1, Index 2. Die Bytes HB und LB bilden zusammen den Adreßteil. Es werden virtuelle Adressen errechnet, deren vorderstes Bit eines von zwei Basisregistern anwählt (User-Data-Block oder Code-Bereich). Die Befehlsgruppe I = 0 wird durch HB untergliedert. LB ist dann der Bezeichner für eine von 30 möglichen Listen oder ein 8-bit Displacement. Der VCP-Befehlsvorrat ist das Ergebnis einer mehrjährigen lockeren Evolutionsstrategie, wobei ein neuer virtueller Befehl dann als "effizient" angesehen wurde, wenn er bei der Formulierung des Compilers zehnmal mehr Bytes einsparen konnte als er selbst im Interpreter an Maschinencode verbraucht. Dabei entstand eine Akkumulatormaschine mit 64 bit-Arithmetik, zwei Indexregistern und automatisch (d.h. VCP-intern)

verwalteten Listen veränderlicher Struktur. Besonders platzsparend wirkt sich ein Satz virtueller Exceptions aus. Damit können u.a. frei definierbare weitere virtuelle Befehle - die dann intern selbst virtuell codiert sind - und die syntaktische Überprüfung bei erkannter Regel behandelt werden.

3.2 Assembler

Der zweiphasige Assembler benötigt immerhin ca. 6 KB virtuellen Code, was auf die für einen Mikroprozessor relativ komplizierte Semantik zurückzuführen ist. Mit dem VCP-codierten Assembler wurde nun der MC68000-VCP übersetzt, wodurch sowohl der Assembler als auch der Basic-PEARL-Compiler des Mulby schon wenige Wochen nach Beginn der Arbeiten auf dem Design-Module ablauffähig waren. Als Nachsetzer für den einphasigen Compiler verarbeitet der Assembler später auch die virtuellen Anteile des Laufzeitcodes (s. 5.1).

3.3 Compiler

Eine Änderung des Codegenerators hätte im Prinzip genügt, um den vorhandenen Einphasencompiler [1] anzupassen. Allerdings erfordert die optimale Ausnutzung des Mikroprozessors - vor allem wegen des großen Adreßbereiches - nun doch einige zeitraubende Änderungen im Compilerkern.
Die Steuerung bei der verzahnten semantischen und der syntaktischen Analyse wird wahlweise von der Eingabezeichenkette oder von der Produktionsregel abgeleitet. (Fest programmiertes Verfahren)
Die PEARL-Objektbezeichner sind linear in verschiedenen, unterschiedlich strukturierten Listen untergebracht. So wird z.B. nach REQUEST/RELEASE zunächst nur in der Liste der Semaphorbezeichner gesucht. Eventuelle zeitliche Nachteile bei der linearen Suche werden,soweit möglich, durch semantische Vorsortierung gemildert. Die 29 Listen diffundieren zunächst frei in den Nutzerdatenblock und werden - automatisch durch die virtuelle Hardware - bei Bedarf reorganisiert und komprimiert. Im Grenzfall ist der angebotene Nutzerdatenblock lückenlos dicht ausgenutzt. Es genügen 18 KB virtuellen Code, wobei das Lexikon und die Tabellen der erlaubten Verknüpfungen bei dyadischen Operationen mit enthalten sind. Die Compilerausgabe ist ASCII-Text, der den halbvirtuellen Laufzeitcode bestimmt.

3.4 Effektivität der virtuellen Codierung

Für den Assembler ergibt sich ein Speicherplatzverhältnis von 3 : 1 für die Relation virtuell/reell. Hier offenbart sich, daß der VCP-Code zum Schreiben von Assemblern überdimensioniert ist. Natürlich wird die Geschwindigkeit negativ beeinflußt: Grob geschätzt werden bei typischen Systemprogrammen mit Ausdrücken in den Operandenfeldern etwa 700 - 700 Zeilen pro Minute (4MHz clock) verarbeitet.
Bei der Realisierung des Compilers wird ein Zahlenverhältnis von 9 : 1 für die Relation virtuell/reell erreicht, für das Paket Compiler + Assembler gar der Wert 12 : 1. Da der VCP reentrant ist - auch seine internen Variablen liegen im Nutzerdatenblock - sind auch Assembler und Compiler mehrbenutzerfähig.

Insgesamt ist festzustellen, daß der MC68000 wegen seiner 8 Adreßregister und einer echten Adreßrechnung im 24-Bit-Adreßraum für die Realisierung des VCP besser geeignet ist als die meisten seiner 16-Bit-Konkurrenten. Auch gegenüber dem Mulby 3/35, der mit seinen 16 Registern ebenfalls eine effektive VCP-Abbildung gestattet, ergibt sich selbst bei 4 MHz-Betrieb ein zeitlicher Vorteil, so daß die in [1] ermittelten zeitlichen Meßwerte sicher unterboten werden - beim Betrieb mit 8 MHz sogar ganz erheblich.

4. Betriebssystem

4.1 Allgemeine Überlegungen

Alle Betriebssystemfunktionen, die frei von Wartezuständen sind, müssen auch den Prozessen 1. Art (Interruptprozesse) zur Verfügung stehen. Die Hardware des MC68000 bietet dazu mit der Verarbeitung von TRAP - und Illegal-Befehlen scheinbar gute Voraussetzungen, da diese auch im Interruptlevel des Prozessors wirksam sind. Einige BS-Funktionen können auch in Programmschleifen sinnvoll sein (z.B. REQUEST/RELEASE, Speicheranforderung) und werden deswegen über die 15 freien TRAP-Vektoren angeschlossen. Bei den restlichen (z.B. TERMINATE) können die zusätzlichen Zyklen bei der Dekodierung des (illegalen) Befehls hingenommen werden.Leider hatten die Konstrukteure des Schaltkreises offenbar andere Vorstellungen vom Aufbau eines Multitask-Echtzeitbetriebssystems. Nicht nur Interrupts, sondern auch die Exceptions (z.B. TRAP, ill. Befehl) gelten als "privilegierte" Prozesse, bei denen ein anderer Stack (das Adreß-

register A7 ist zweimal vorhanden) von der Hardware benutzt wird. Das hat die recht unangenehme Konsequenz, daß Prozesse 2. Art (Tasks) vom Dispatcher sowohl im privilegierten als auch im unprivilegierten Mode angetroffen werden können. Es müssen also Teile des "Systemstacks" dem aktuellen Taskzustand hinzugeschlagen werden und bei Taskwechsel ausgetauscht werden. Durch Vermeidung von tiefgestaffelten Unterprogrammaufrufen innerhalb der BS-Funktionen läßt sich der Inhalt des Systemstacks allerdings vom Umfang her begrenzen.

4.2 Belegung der Interruptebenen

Der Dispatcherprozeß (DP) wurde, wie schon in [1], [4] beschrieben, als innenmaskierbarer niedrigstprioritierter Prozeß 1. Art realisiert. Damit lassen sich drei Ziele erreichen:

* Alle Interruptprozesse (mit Ausnahme des DP) sind geschlossen, d.h. sie kehren rasch an die Unterbrechungsstelle zurück.
* Man benötigt keine Bearbeitungsschlangen für BS-Aufträge und damit auch keinen "Kern-Mode" des BS. Jedes Stück Code wird also ausschließlich auf Task- oder auf Hardware-Interrupt-Ebene exekutiert.
* Durch die Innenmaskierung des DP können Abschnitte in BS-Funktionen für den DP unteilbar gemacht werden, ohne daß Hardware-Interrupts behindert werden.

Die 8 Ebenen wurden wie folgt zugeordnet (GE = Grundebene):

IR7:	Urstart und Rücksetzfunktion
IR6:	Zeitäquidistanter Zeittakt (1,2,5,10 msec)
IR5:	I/O, ACIAS, PIAS etc.
IR4:	I/O "
IR3:	I/O "
IR2:	Prozeßinterrupts
IR1:	Dispatcherprozeß mit Innenmaske
GE:	System- und Nutzertasks, Softwareprioritäten

4.3 Dispatcherhardware

Mit dem käuflichen Board MEX68KDM ist die Realisierung des DP in der gewünschten Weise nicht möglich. Durch Verwendung des auf dem Board vorhandenen I/O-Adreßdekoders genügen aber zwei hinzugefügte Schaltkreise, um zwei Flip-Flops getrennt durch Hardwarebefehle setzen und rücksetzen zu können. Über eine Undbedingung wird der Eingang für IR1 angesteuert. Damit ergeben sich vier Dispatcherbefehle:

TST	$ 3FF22	Enable Dispatcher
TST	$ 3FF20	Disable Dispatcher
TST	$ 3FF02	Dispatcher alert
TST	$ 3FF00	Dispatcher acknowledge

4.4 Speicherverwaltung

Beim Laden wird jeder Task ein lagefester Taskkopf beigegeben, der ihre Einkettung in die nach Prioritäten geordnete Dispatcherkette ermöglicht. Es wird von unten nach oben Programmcode geladen, während die dynamisch belegten Abschnitte nach dem First-fit-Prinzip von oben nach unten durch eine BS-Funktion vergeben werden. Es gibt 3 unterschiedliche Einsatzmöglichkeiten solcher Abschnitte:

* Lokaler Taskarbeitsspeicher für taskeigene Variablen, für Variable des Laufzeitinterpreters und zum Retten der Taskzustände (Registerinhalte etc.)
* Prozedurarbeitsspeicher. Er wird unter dem Bezeichner der aufrufenden Task verwaltet (z.B. auch für Compiler und Assembler)
* Kommunikationselement. Es wird für die Ankettung an echte oder simulierte E/A-Warteschlangen mit wechselnden Taskbezeichnern benutzt.

4.5 Ein-/Ausgabe

Auch hier konnte im Kern die schon in [1], [4] beschriebene Struktur übernommen werden:
Zu jedem E/A-Baustein mit asynchroner Sende-/Empfangsbereitschaft wird eine Warteschlange aus Kommunikationselementen (KE) angelegt. Eine E/A-willige Task bietet ein von ihr be- und versorgtes KE einer zentralen E/A-Funktionsroutine an, die das KE an die zum Quellen-/Zielort gehörende hochprioritierte Systemtask weiterreicht und bei Eingabeaufträgen den Auftraggeber blockiert. Die durch die E/A-Funktionsroutine aktivierte E/A-Systemtask inspiziert die Warteschlange und bereitet den E/A-Baustein vor. Anschließend hält sie sich mit SUSPEND selbst an und überläßt den eigentlichen Datentransfer dem zugehörigen Interruptprozeß. Dieser Interruptprozeß ruft bei Satzende oder Irregularitäten die CONTINUE-Funktion auf, um die E/A-Systemtask freizugeben, die dann gegebenenfalls den Auftraggeber entblockiert und das nächste

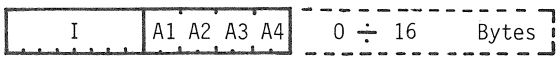
Element der Warteschlange bearbeitet oder sich durch TERMINATE zur Ruhe setzt.

Für manche Synchronisationsaufgaben, etwa nach Art des Producer-consumer-Schemas, bietet sich die vorgesehene Möglichkeit namentlich definierbarer fiktiver DATIONs vom Typ INOUT als echte Alternative zu Semaphorkonstruktionen an.
Damit langsame Peripherie bei der Ausgabe nicht den freien Speicher restlos zustopfen kann [4], wird die Anzahl der KE nach einem einfachen Kriterium für jede Task begrenzt.
Die E/A-Interruptprozesse lassen sich zusätzlich nutzen, um eine Bedientask dem entsprechenden Terminal zugänglich zu machen.

5. Laufzeitsystem

5.1 Laufzeitcode

Der vom Compiler produzierte Code besteht, wie in [1], [4], auch für den MC68000 aus realen (z.B. BRA, ADD, ADD.L, OR) und virtuellen Befehlen (z.B. Floating -Operationen, REQUEST, PUT). Die vom Compiler automatisch generierten Umschaltbefehle benötigen nur je 1 Wort (TRAP bzw. virtueller 1 Wort-Befehl).Im Gegensatz zu [1] und [4] wurden hier jedoch 4 Adressierungsarten festgelegt, um auch in dem großen Adreßbereich mit kurzem Code auszukommen.
Die virtuellen Laufzeitbefehle sind nun bis zu 18 Bytes lang:



(2)

Bis zu 4 Operanden sind jeweils möglich. Die Felder A1 bis A4 spezifizieren die jeweilige Adressierungsart:

00: Local Workspace relative, 16 bit Displacement:
Skalare Variable auf PROC/TASK-Ebene.

01: Program counter relative, 16 bit Displacement:
lokale Sprungziele, Konstanten, Format-Labels etc. begrenzt evtl. die Codelänge einer Task/Prozedur.

10: Indirekt (32 Bit) Local Workspace relative:
Feldelemente, IDENT-Objekte.

11: Long direct, 32 bit:
Prozeduraufrufe, globale Variablen und Formate.

Bei realen Befehlen wird die Adressierungsart 10 durch einen eingeschobenen MOVEA.L-Befehl zum Laden eines Adreßregisters und anschließende indirekte Registeradressierung ersetzt.
Zwei Adreßregister sind dauerhaft als Basisregister belegt (Local Workspace und Task-Workspace).
Alle Variablen des Laufzeitinterpreters sind ebenfalls in diesen Bereichen untergebracht. Reentrance und sogar Rekursivität von Prozeduren ergibt sich ohne weitere Maßnahmen durch die Struktur.
Bei formatierten E/A-Anweisungen wird ein zusätzlicher Variablenblock für die internen Operationen des Interpreters bereits vom Compiler an den Bereich der gerade "lebendigen" lokalen PEARL-Variablen angehängt. Auf diese Weise dürfen sogar z.B. in einer PUT-Liste Ausdrücke stehen, die durch Funktionsaufrufe auf eine "eingeschachtelte" weitere PUT oder GET-Anweisung führen.

5.2 Speicherbelegung MEX68KDM-PEARL

$00000 ÷ $004FF	Exception Vektoren Lagefeste Taskköpfe der ROM-Tasks, Zentrale OS-Daten
$00500 ÷ $07FFF	Freies, werkseitig bestücktes RAM
$08000 ÷ $1FFFF	Nachrüstbares freies RAM
$20000 ÷ $23FFF	EPROM-Gruppe 1 (16KB) Nucleus, Urstart, Systemtasks
$24000 ÷ $2BFFF	EPROM-Gruppen 2 und 3 (32KB) Compiler, Assembler, VCP
$2C000 ÷ $2FFFF	EPROM-Gruppe 4 (16 KB) Laufzeitinterpreter, Bedientask
$3FF00 ÷ $3FFFF	Memory-mapped I/O

6. Einige Daten zur Implementierung

6.1 Platzbedarf EPROM-Inhalt:

Compiler:	18 KB
Assembler:	6 KB
VCP:	2,2 KB
Nucleus (ohne E/A-Tasks):	3,5 KB
Laufzeitinterpreter:	ca. 9 KB
Bedientask:	? (in Arbeit)

6.2 Eigenschaften auf Basic-PEARL-Ebene

Länge der Bezeichner: Max 6 Zeichen

FLOAT (max.55): intern (24+8) und (56+8) Bit
FIXED (max.31): intern 16 und 32 Bit
BIT (1 ÷ 32): intern 8,16,32 Bit
CHAR (1 ÷ 255)

Prozeduren immer reentrant und rekursiv.
Felder als Prozedurparameter nur im IDENT-Mode.
Implementierungslücken: Bei den ON-Blöcken sowie unvollständige Untergliederung von Dations

6.3 Compilerdaten

Max.ca. 2400 gleichzeitig "lebende" PEARL-Bezeichner.
Fehlertexte in-line mit Ortsmarkierung.
Keine Ausgabemöglichkeit von Cross-Referenz-Listen.
Zeilentracing vorbereitet.
Garantiert erreichbare Obergrenze der Codelänge pro Task/Prozedur: 32 KB.

Daten zur Übersetzungsgeschwindigkeit können nur als grober Anhaltspunkt dienen. Sie wurden bei Betrieb des MEX68KDM als Cross-Compiler für den Mulby 3/35 ermittelt und korrespondieren zu den Daten in [1] bzw. in [4]:

21 sek/1000 Worte Code (4MHz)
11 sek/1000 Worte Code (8MHz)

7. Ausblick

Die Arbeiten am Betriebssystemkern, Compiler und Assembler sind zur Zeit (Oktober 1981) in etwa abgeschlossen. Mit der Fertigstellung des Laufzeitinterpreters ist gegen Ende des Jahres zu rechnen, so daß Anfang 1982 die Zusammenschaltung der Einzelkomponenten ausgetestet werden kann. Das System wird mit dem werksseitigen 32 KB RAM bis zu 1500 Zeilen große Basic-PEARL-Programme verarbeiten können. Nach Vergrößerung des RAM auf 128 KB können schließlich bis zu 6000 Zeilen resident bearbeitet werden. Für sehr kleine Programme genügt ein Sichtgerät als Peripherie, ansonsten kann - ebenfalls noch preiswert - an die Verwendung zweier Kassettenrekorder gedacht werden: Auf Laufwerk 1 liegt der Quellentext, er wird einmal gelesen. Das Laufwerk 2 speichert den Assemblertext, der nur einmal gelesen zu werden braucht, denn die restlichen Schreib-/Leseoperationen lassen sich ohne Überlaufprobleme durch die simulierten Datenstationen nach dem producer-consumer-Schema erledigen. Dabei laufen Compiler und Assembler parallel. Schließlich wird zuletzt "bei lebendigem Assembler" geladen.

Zur Zeit sind Prozessor und EPROM-Bausteine (8Kx8) noch zu teuer, um gleich an eine Anwendung von PEARL im Hobby-Computer-Bereich zu denken. Dennoch hofft der Verfasser, daß die vorgestellte Spar-Implementierung in naher Zukunft einem größeren Anwenderkreis den Zugang zu PEARL ermöglicht.

8. Literatur

[1] Gerth, W.: "Ergebnisse einer Basic-PEARL-Implementierung für Kleinrechner" Fachtagung Prozeßrechner GI, GMR, VDI/VDE München, 1981.

[2] Mönster, J.: "Entwicklung eines Multitask-Echtzeitbetriebssystemes zur Implementierung von Basic-PEARL auf dem Mikrorechner MC68000". Diplomarbeit, Institut für Regelungstechnik, 1981.

[3] Kemmerling, M.: "Entwicklung und Aufbau des Laufzeitinterpreters zur Implementierung von Basic-PEARL auf dem Mikrorechner MC68000". Diplomarbeit Institut für Regelungstechnik, 1981.

[4] Gerth, W.: "Basic-PEARL für Mini- und Mikrorechner", PEARL-Rundschau Nr. 1, Band 2, April 1981.

Anschrift des Verfassers:
Prof. Dr.-Ing. W. Gerth,
Institut für Regelungstechnik
Universität Hannover
Appelstr. 11, 3000 Hannover 1
Telef. (0511) 7624512.