Application Note AN048

Floating Point Arithmetik auf der MODULAR-4/486

Autor: JD Datei: AN048.DOC (4 Seiten)

Folgende Punkte sind bei der Erstellung von MODULAR-4 Echtzeit-Programmen mit Borland-C bei Verwendung von Fließkommaberechnungen zu beachten (alle Angaben beziehen sich auf 486DX CPUs):

1. Die Datei MATH.H muß eingebunden werden.
2. Bei der Initialisierung der PDT von Tasks mit Floating Point Arithmetik muß der entsprechende Coprozessor-Typ eingestellt werden.
3. Compiler-Optionen bzgl.Optimierung (nach Größe oder Geschwindigkeit) müssen abgestellt sein.
4. Compiler-Option bzgl. Floating Point auf EMULATION setzen.
5. Die Startup-Codes müssen aus dem Verzeichnis BC31 eingebunden werden (für Borland C 3.1).
6. Compiler-Option bzgl. Stack (Entry/Exit-Code) muß auf Standard Stack Frame eingestellt werden.
7. In **keiner** Taskfunktion (also einer in der PDT eingetragenen Funktion, die mit ml8rt_entry oder ml8rt_entry_function beginnt, also auch in der auto_init) darf eine Floating Point Operation durchgeführt werden. Dazu gehören auch einfache Zuweisungen der Art x=1.234. Soll eine solche Operation in einer Taskfunktion ausgeführt werden, so ist eine separate Unterfunktion zu schreiben, in der die Operationen durchgeführt werden können. Diese Unterfunktion ist eine **lokale** Taskfunktion, also eine, die nicht in die PDT eingetragen wird. Siehe dazu MODULAR-4 Handbuch Kapitel 7: Verwenden von Fließkommaoperationen.
8. Wenn in der Hauptprozedur einer Interrupt-Task mit Floating Point gearbeitet wird und andere Tasks ebenfalls die FPU benutzen, so müssen am Beginn der Interrupt-Hauptprozedur die Register der FPU gesichert werden (mit der Funktion ml8rt_store_487). Daraufhin kann die Interrupt-Task die FPU nutzen. Am Ende der Prozedur muß der alte Registerzustand mit ml8rt_restore_487 wiederhergestellt werden.
9. Benutzen mehrere NI-Tasks die FPU, so darf während der Installierung einer solchen Task keine andere ausgeführt werden. Deshalb sollten in diesem Fall zuerst alle Tasks geladen werden ohne sie zu aktivieren (Installationsflag 12 = 0). Danach können alle Tasks mit ml8_wakeup_task aktiviert werden.

Meldung von Fehlern durch die FPU

In der derzeitigen Version des Betriebssystems (3A.03) meldet der Coprozessor erkannte Fehler nicht an die CPU weiter. Deshalb kann das Betriebssystem nicht darauf reagieren. Die FPU besitzt jedoch die Möglichkeit, alle Fehler, die sie erzeugt, weiterzumelden.

Mit der folgenden C-Prozedur wird die FPU so initialisiert, daß dies geschieht.

```
#pragma inline                      // force compiler to run TASM

void Init_FPU()
{
      unsigned int control_word;
      unsigned int mask;

      mask = 0x00yz;                // für y, z siehe Tabelle
      asm push eax;                 // save eax register
      asm mov eax, CR0;             // load CPU-Register CR0 into eax
      asm or eax, 32;               // set bit #5 (NE)
      asm mov CR0, eax;             // write eax into CPU-Register CR0
      asm fstcw control_word;       // load FPU-Control-register
      control_word &= ~mask;        // unmask exception mask
      asm fldcw control_word;       // write new exception mask into FPU
      asm pop eax                   // restore eax register
}
```

Das NE-Bit im CR0-Register entscheidet, wie die CPU auf FPU-Exceptions reagiert, entweder mit INT16 (NE=1) oder mit INT13 (NE=0, DOS-Mode, General Protection Fault).
In den Bits der lokalen Variablen *mask* wird festgelegt, welche FPU-Fehler gemeldet werden sollen:

Bit	Bedeutung
0	ungültiger Befehl
1	denormaler Operand
2	Division durch Null
3	Überlauf
4	Unterlauf
5	Genauigkeitsverlust
6..15	=0 setzen

Soll ein Fehler aus dieser Tabelle signalisiert werden, muß das entsprechende Bit in *mask* =1 gesetzt werden, wenn z.B. ein Über- und Unterlauf registriert werden soll, muß mask = 0x18 gesetzt werden.

Tritt in der FPU ein Fehler auf, der wie beschrieben demaskiert worden ist, reagiert die CPU mit einem Interrupt 16 (10h). Das Betriebssystem OsX würde darauf mit einem Service-Request zum PC reagieren.

Um eine gesonderte Behandlung dieses Fehlers zu erreichen, stehen zwei Möglichkeiten zur Verfügung.

1. Die Programmierung einer DI-Task, die unter Interrupt 16 installiert wird, und deren Hauptprozedur folglich beim Auftreten des Interrupts aufgerufen wird.
2. Soll keine eigene Task für diesen Fall installiert werden, so besteht folgende Alternative:

   Mit der Funktion ml8rt_set_int_vec(unsigned int int_nr, void *func) kann der Interruptvektor des INT16 auf eine eigene Interrupt-Service-Routine gesetzt werden. Diese muß folgende Deklaration haben:

   ```
   void interrupt my_service_func(void);
   ```

   Die Funktion ml8rt_set_int_vec ist bisher nicht im ML8-Handbuch beschrieben, sie ist jedoch in der ML8RTBIB implementiert und steht auch in der Header-Datei.

   Um beim Auftreten eines FPU-Interrupts mit einer eigenen Funktion reagieren zu können, sind folgende Schritte erforderlich:

   1. Initialisierung der FPU wie oben beschrieben
   2. Deklaration einer eigenen Interrupt-Service-Routine (s.o.)
   3. Aufruf: `ml8rt_set_int_vec(16, my_service_func);`

Pascal

Für Fehler, die im Coprozessor entstehen, gilt in Pascal dasselbe wie in "C". Insbesondere die oben aufgelisteten Hinweise 7. bis 9. sind auch unter Pascal zu befolgen. Die erforderlichen Compiler-Einstellungen entnehmen Sie dem MODULAR-4 Handbuch (Kapitel 7.6.3 und 7.6.4).

Der Coprozessor ist auch in Pascal separat zu initialisieren, wenn er erkannte Fehler an die CPU melden soll. Zu beachten ist jedoch, daß der Borland Turbo Pascal 7.0 Compiler die oben aufgeführte Prozedur Init_FPU nicht annimmt, da er keine 486-Befehle und -Register kennt (z.B. eax und CR0). Diese Funktion muß deshalb als Objekt-File (mit C oder Assembler erzeugt) in ein PASCAL-Programm einbezogen werden.

Die Behandlung von Coprozessor-Fehlern kann auf zwei Arten erfolgen:

- Installierung einer Interrupt-Task unter Interrupt 16 (10h)
- Schreiben einer Interrupt-Service-Routine, die wie folgt zu deklarieren ist:

  ```
  PROCEDURE my_service_func; INTERRUPT;
  ```

  Zum Installieren der ISR muß die Funktion ml8rt_set_int_vec aufgerufen werden:

  ```
  ml8rt_set_int_vec(16, @my_service_func);
  ```

  Diese Funktion ist bisher nicht im ML8-Handbuch beschrieben, sie ist jedoch in der ML8RTBIB implementiert.

Hinweis: Der Pascal-Datentyp *real* benutzt die Floating Point Einheit nicht. Auf Laufzeitfehler, die bei Operationen mit *real* Variablen entstehen, wird genauso reagiert, wie bei Operationen mit Integer-Variablen (siehe unten, eine Division durch Null führt jedoch -vermutlich durch Rundungs-Ungenauigkeiten- bei *real*-Variablen nicht zu einem Fehler).

Integer-Arithmetik

Auf Fehler bei Integer-Operationen reagiert die CPU sowohl unter C als auch unter Pascal mit Interrupt 0 (z.B. bei einer Division durch Null oder einem Überlauf). Die Fehlerbehandlung kann auf eine der zwei oben beschriebenen Arten erfolgen.

Wenn bei Operationen andere Interrupts (MODULAR-4/486 Handbuch, Anhang D) auftreten, kann die Fehlerbehandlung unter dem entsprechenden Interrupt genauso ausgeführt werden.