Übungen zu Organisation und Architektur von Rechnern
SS 2002
Serie 4 — Musterlösung

Aufgabe 12 (10 Punkte)

Da der Befehlssatz des MC68000 keine Multiplikation für Langworte vorsieht, können auf direktem Wege nur Fakultäten berechnet werden, die maximal Wortlänge haben. Dabei stellt sich das Problem, einen Überlauf während der Berechnung festzustellen. Ein Überlauf ist immer dann gegeben, wenn ein Zwischenergebnis nicht mehr im Wortformat darstellbar ist. Folgender Ansatz wäre naheliegend:

```
MULU.W   D0,D1
CMP.L    #$10000,D1
BPL      OVERFLOW                                .
```

Dabei ist aber zu bedenken, das der `CMP` Befehl seine Argumente immer als vorzeichenbehaftet interpretiert. Somit würden also Werte in D1 die größer als `$80000000` sind aufgrund des gesetzten höchstwertigen Bits als negativ interpretiert und damit der Overflow durch obigen Ansatz nicht erkannt. Eine andere Möglichkeit wäre, zu überprüfen, ob im oberen Wort ein von 0 verschiedener Wert steht:

```
MULU.W   D0,D1
SWAP     D1          swap D1 to perform test on higher word
TST.W    D1          word non zero => overflow
BNZ      OVERFLOW
SWAP     D1          restore D1                  .
```

In dieser Musterlösung kommt eine dritte Variante zur Anwendung, die weniger Rechenzyklen benötigt:

```
MULU     D0,D1
BMI      OVERFLOW    result negative => overflow
CMP.L    #$10000,D1
BPL      OVERFLOW                                .
```

Hier werden Ergebnisse, die als negative Zahlen zu interpretieren sind, im Zuge der `BMI` Instruktion als Überlauf erkannt und ansonsten der ursprüngliche Ansatz verwendet.

Insgesamt ergibt sich folgender Code:

```
                ORG     $0
                DC.L    $8000           Stack pointer value after a reset
                DC.L    START           Program counter value after a reset

START:          MOVE.L  #6,D0
                JSR     FAC_W           call FAC_W( 6)
                BREAK


******************************************************************************
*
* function:   D0 FAC_W( D0)
*
* description:
*   computes the factorial of the number given in D0 and returns
*   the result in D0 as well. Returns 0, iff an overflow (wrt. to word format)
*   is detected.
*
* registers affected: D0, D1
*

FAC_W:          MOVE.L  #1,D1           initialize the accumulator D1
                CMPI    #0,D0           if ( n eq 0)
```

```
                BEQ     DONE_W          return( D1)

LOOP_W:         MULU    D0,D1           multiply the accu with n
                BNE     OVERFLOW        result still in word format? (negativ)
                CMP.L   #$10000,D1
                BPL     OVERFLOW        result still in word format? (positiv)
                SUBI    #1,D0           decrement n
                BNE     LOOP_W          until ( n eq 0)

DONE_W:         MOVE.L  D1,D0           return the accu in D0
                RTS

OVERFLOW:       MOVE.L  #0,D0
                RTS
```

Aufgabe 13 (10 Punkte)

Um bei der Berechnung der Fakultät das volle Langwortformat ausschöpfen zu können, muß eine Funktion zur Multiplikation zweier Langworte geschrieben werden. Die hier vorgestellte Lösung führt die Langwort-Multiplikation auf die im MC86000-Befehlssatz vorhande Wort-Multiplikation zurück.

Zwei Langworte a und b können in jeweils zwei Worte a_{hi}, a_{lo} und b_{hi}, b_{lo} zerlegt werden:

$$a = 2^{16} \cdot a_{\mathrm{hi}} + a_{\mathrm{lo}} \qquad , \qquad b = 2^{16} \cdot b_{\mathrm{hi}} + b_{\mathrm{lo}} \qquad .$$

Bei Multiplikation von a und b ergibt sich dann:

$$\begin{aligned} a \cdot b &= (2^{16} \cdot a_{\mathrm{hi}} + a_{\mathrm{lo}})(2^{16} \cdot b_{\mathrm{hi}} + b_{\mathrm{lo}}) \\ &= 2^{32} \cdot a_{\mathrm{hi}} b_{\mathrm{hi}} + 2^{16} \cdot a_{\mathrm{hi}} b_{\mathrm{lo}} + 2^{16} \cdot a_{\mathrm{lo}} b_{\mathrm{hi}} + a_{\mathrm{lo}} b_{\mathrm{lo}} \end{aligned}$$

Es sind also erst einmal vier Wort-Multiplikationen (z. Bsp. $a_{\mathrm{hi}} b_{\mathrm{hi}}$) auszuführen. Anschließend müssen diese langwortigen Zwischenergebnisse noch mit den angegebenen Zweierpotenzen multipliziert (= Bit-verschiebung) und aufsummiert werden. Es ergibt sich das folgende Additions-Schema, wobei $r_0 = a_{\mathrm{hi}} b_{\mathrm{hi}}$, $r_1 = a_{\mathrm{hi}} b_{\mathrm{lo}}$, $r_2 = a_{\mathrm{lo}} b_{\mathrm{hi}}$ und $r_3 = a_{\mathrm{lo}} b_{\mathrm{lo}}$:

hi(r0)	lo(r0)		
	hi(r1)	lo(r1)	
	hi(r2)	lo(r2)	
		hi(r3)	lo(r3)
c0	c1	c2	c3

Dabei bezeichnen c_0, c_1 das obere Langwort und c_2, c_3 das untere Langwort des Ergebnisses.

Dieses Verfahren läßt sich direkt in eine Assembler-Funktion `MULU_L` umsetzen, wobei allerdings zu beachten ist, daß bei der Addition der Zwischenergebnisse die Überträge korrekt berücksichtigt werden.

In der Funktion `FAC` kann die Instruktion `MULU.W` durch einen Aufruf von `MULU_L` ersetzt werden, um die gewünschte Erweiterung der Fakultätsberechnung auf Langworte zu erreichen. Ein Overflow liegt nun immer dann vor, wenn das Zwischenergebnis größer als ein Langwort ist, d. h. das obere Langwort des Ergebnisses von Null verschieden ist.

Insgesamt ergibt sich folgender Code:

```
                ORG     $0
                DC.L    $8000           Stack pointer value after a reset
                DC.L    START           Program counter value after a reset

START:          MOVE.L  #$ffffffff,D0
```

```
                MOVE.L  #$ffffffff,D1
                JSR     MULU_L          call MULU_L( ...)
                BREAK
                MOVE.L  #6,D0
                JSR     FAC_L           call FAC_L( 6)
                BREAK
                MOVE.L  #9,D0
                JSR     FAC_L           call FAC_L( 9)
                BREAK
                MOVE.L  #12,D0
                JSR     FAC_L           call FAC_L( 12)
                BREAK
                MOVE.L  #13,D0
                JSR     FAC_L           call FAC_L( 13)
                BREAK


****************************************************************************
*
* function:   D0 FAC_L( D0)
*
* description:
*   computes the factorial of the number given in D0 and returns
*   the result in D0 as well. Returns 0, iff an overflow (wrt. to long format)
*   is detected.
*
* registers affected: D0-D5
*

FAC_L:          MOVE.L  #1,D1           initialize the accumulator D1
                CMPI    #0,D0           if ( n eq 0)
                BEQ     DONE_L          return( D1)
LOOP_L:         JSR     MULU_L          D2,D3 = MULU_L( D0, D1)
                CMPI.L  #0,D2           result still in long format?
                BNE     OVERFLOW
                MOVE.L  D3,D1           if so, move result into accu
                SUBI    #1,D0           decrement n
                BNE     LOOP_L          until ( n eq 0)

DONE_L:         MOVE.L  D1,D0           return the accu in D0
                RTS


****************************************************************************
*
* function:   D2,D3 MULU_L( D0, D1)
*
* description:
*   multiplies two unsigned long formats given in D0,D1. The result is returned
*   in D2,D3, which hold the higher and the lower long format of the result,
*   respectively.
*
* registers affected: D2-D5
*

*
* For reasons of convenience, the following notation will be used to denote
* the individual parts of the resulting double-long:
*
*                   |----||----||----||----|
*                   |    D2    ||    D3    |
*                   | r0 || r1 || r2 || r3 |
*

MULU_L:         MOVE.L  D0,D4
                SWAP    D4              D4 = HW( D0)
                MOVE.L  D1,D5
                SWAP    D5              D5 = HW( D1)
                MOVE.L  D4,D2
                MULU    D5,D2           D2 = HW( D0) * HW( D1) => affects r0 r1
                MOVE.L  D1,D3
                MULU    D0,D3           D3 = LW( D0) * LW( D1) => affects r2 r3

                MULU    D1,D4           D4 = HW( D0) * LW( D1) => affects r1 r2
```

```
                MULU    D0,D5           D5 = LW( D0) * HW( D1) => affects r1 r2
                ADD.L   D5,D4
                BCC     NO_INC_HW       do we have a carry affecting r0?
                ADD.L   #$10000,D2      increment r0
NO_INC_HW:      SWAP    D4
                MOVE.L  D4,D5
                ANDI.L  #$FFFF,D4       D4 = $0000     HW(D4+D5) => r1
                ANDI.L  #$FFFF0000,D5   D5 = LW(D4+D5) $0000     => r2
                ADD.L   D5,D3           compute     r2 r3
                ADDX.L  D4,D2           compute     r0 r1
                RTS
```