Universität Mannheim
Lehrstuhl für Praktische Informatik IV
Prof. Dr. W. Effelsberg
Christoph Kuhmünch, Gerald Kühne

Praktische Informatik II
SS 2000

Übungsblatt 7 - Musterlösung

.

Aufgabe 1: Statusregister [4 Punkte]

Erläutern Sie die Bedeutung der Flags N (Negativ), Z (Zero), V (Overflow) und C (Carry) im Statusregister des M68000. Geben Sie für jedes Flag eine Sequenz von M68000-Assemblerbefehlen an, nach deren Ablauf das entsprechende Flag gesetzt ist (kein direkter Zugriff auf das Statusregister!).

Lösung:

<u>N (negative) Flag</u>

Gibt an, ob das Ergebnis der letzten arithmetischen Operation negativ (als Zweikomplement betrachtet) war, also das MSB des Ergebnisses gesetzt ist. N = MSB(result);

MOVE.B #129,D0

<u>Z (zero) Flag</u>

Gibt an, ob das Ergebnis der letzten arithmetischen Operation == 0 war.

MOVE.B #3,D0
SUB.B D0, D0

<u>V (overflow) Flag</u>

Wird gesetzt, wenn bei einer Berechung ein Überlauf stattfand, also der Zahlenbereich der arithmetischen Einheit überschritten wurde. Dies geschieht beispielsweise, wenn zwei positive Zahlen addiert werden, das Ergebnis dann aber negativ ist. Entsprechendes gilt für die Addition zweier negativer Zahlen und ein daraus resultierendes positives Ergebnis. Ergebnisse mit denen das Setzen des V-Flags einhergeht sollten i.a. verworfen werden, da es nicht das ist, was man eigentlich berechnen wollte.

MOVE.B #120,D0	01111000	(= 120)	
ADD.B D0,D0	<u>+01111000</u>	(= 120)	
	11110000	(= -16)	(auch N-Flag gesetzt)

<u>C (carry) Flag</u>

Wird gesetzt, wenn bei einer Berechung ein Übertrag stattfand, bei der Verwendung solcher Ergebnisse sollte man wie auch bei V-Flag vorsichtig sein. Bei Shift-Operationen wird das zuletzt aus dem Wort „gefallene“ Bit im C-Bit gespeichert.

MOVE.B #129,D0	10000001	(= -127)	
ADD.B D0,D0	<u>+10000001</u>	(= -127)	
	1|00000010	(= 2)	(auch V-Flag gesetzt)
	C		

Aufgabe 2: Adressierungsarten [7 Punkte]

Gegeben sei folgendes M68000-Assemblerprogramm:

```
          MOVEA.L M2,A0
          MOVE.W #2,D0
          MOVE.B (A0),D1
          MOVE.W -(A0),D2
          MOVE.L (A0)+,D3
          MOVE.B -1(A0,D0),D4
          MOVE.W M0,D5
          BREAK

M0        DC.B   0      DC = define constant
          DC.B   1      DC.B = Byte-Konstante (8 Bit)
M1        DC.B   2
          DC.B   3
          DC.B   4
          DC.B   5
M2        DC.L   M1     DC.L = Longword-Konstante (32 Bit)
```

Geben Sie den Inhalt der Register D0, ..., D5 in hexadezimaler Schreibweise nach Programmablauf an und vermerken Sie, mit welcher Adressierungsart der entsprechende Quelloperand gewonnen wurde.

Lösung:

MOVEA.L M2,A0 Lädt den Speicherinhalt an der Stelle des Labels M2 als Longword ins Adressregister A0, dort steht aber grade M1, also befindet sich die Adresse des Labels M1 in A0.

A0	D0	D1	D2	D3	D4	D5
<addr M1>	U	U	U	U	U	U

(U = unknown)

MOVE.W #2,D0 Laden eines **Sofortoperanden (immediate data);** der konstante Wert 2 wird ins Register D0 geladen.(W = 16 Bit Operand)

A0	D0	D1	D2	D3	D4	D5
<addr M1>	$UUUU0002	U	U	U	U	U

MOVE.B (A0),D1 **Adressregister indirekt;** der Inhalt der Speicherzelle deren Adresse in A0 steht wird ins Register D0 geladen(B ➔ als Byte). In A0 befindet sich die Adresse von M1, man lädt also die erste ab M1 abgelegte Byte-Konstante (= 2).

A0	D0	D1	D2	D3	D4	D5
<addr M1>	$UUUU0002	$UUUUUU02	U	U	U	U

MOVE.W –(A0), D2 **Adressregister indirekt mit Prädekrement**; A0 wird zunächst entsprechend der Operandenlänge (W = 2 Byte) erniedrigt, steht dann also auf der Adresse des Labels M0. Von dort wird ein 16 Bit Wert ins Register D2 geladen.

A0	D0	D1	D2	D3	D4	D5
<addr M0>	$UUUU0002	$UUUUUU02	$UUUU0001	U	U	U

MOVE.L (A0)+, D3 **Adressregister indirekt mit Postinkrement;** ein Longword wird von der Adresse vom Label M0 beginnend ins Register D3 geladen, dann wird entsprechend der Operandenlänge (L = 4 Byte) A0 erhöht.

A0	D0	D1	D2	D3	D4	D5
<addr M1> + 2	$UUUU0002	$UUUUUU02	$UUUU0001	$00010203	U	U

MOVE.B –1(A0,D0),D4 **Adressregister indirekt indiziert mit Distanz;** eine Adressrechnung findet statt, der Inhalt des Registers D0 als Word(=16 Bit) wird zu A0 addiert dann eins abgezogen(=Distanz oder displacement). Es ergibt sich eine errechnete Adresse <addr M2> - 1, von dort wird ein Bytewert ins Register D4 geladen.

Achtung: Die errechnete Adresse wird nicht in A0 zurückgeschrieben!

A0	D0	D1	D2	D3	D4	D5
<addr M1> + 2	$UUUU0002	$UUUUUU02	$UUUU0001	$00010203	$UUUUUU05	U

MOVE.W M0,D5 **Speicher absolut;** von der mit Label M0 angegebenen Speicheradresse wird ein 16 Bit-Wert ins Register D5 geladen.

A0	D0	D1	D2	D3	D4	D5
<addr M1> + 2	$UUUU0002	$UUUUUU02	$UUUU0001	$00010203	$UUUUUU05	$UUUU0001

Aufgabe 3: Assemblerprogrammierung [9 Punkte]

Gegeben sei folgendes M68000-Assemblerprogramm:

```
            MOVEQ #1,D0
SCHLEIFE    CMP #10,D0
            BGT ENDE
            MOVE D0,D6
            MULS D6,D6
            BSR AUSGABE
            ADDQ #1,D0
            BRA SCHLEIFE
ENDE        BREAK
```

Der Befehl `BSR AUSGABE` bewirkt den Aufruf eines Unterprogramms, das den Inhalt des Registers D6 auf den Bildschirm ausgibt.

(a) [4 Punkte] Geben Sie ein äquivalentes C-Programm an.

Lösung:

```
/* Ausgabe der Quadratzahlen von 1 ... 10 */
int main ()
{
   int i;

   for (i = 1; i <= 10; i++)
   printf ("%d\n", i * i);

   return 0;
}
```

(b) [5 Punkte] Ändern Sie das Assemblerprogramm so ab, dass die Summe der Quadrate der Zahlen von 1 bis 10 gebildet und in Register D6 angezeigt wird.

Lösung:

```
          MOVEQ     #1,D0
          MOVE.L    #0,D6
SCHLEIFE  CMP       #10,D0
          BGT       ENDE
          MOVE      D0,D5
          MULS      D5,D5
          ADD       D5,D6
          BSR       AUSGABE
          ADDQ      #1,D0
          BRA       SCHLEIFE
ENDE      BREAK
```