Übungen zu Organisation und Architektur von Rechnern
SS 2002
Serie 6 — Musterlösung

Aufgabe 16 (4 Punkte)

Der DUART-Dokumentation läßt sich entnehmen, daß die Initialisierung aus folgenden Schreiboperationen besteht:

a) `%00010000` an die Adresse `$effc01+4` (CRA) schreiben.

b) `%00000011` an die Adresse `$effc01+0` (MR1A) schreiben.

c) `%00000000` an die Adresse `$effc01+0` (MR2A) schreiben.

d) `%10111011` an die Adresse `$effc01+2` (CSRA) schreiben.

e) `%00000100` an die Adresse `$effc01+4` (CRA) schreiben.

Um diese `MOVE`-Befehle elegant hinschreiben zu können, bietet es sich an, mit Hilfe von `EQU`-Anweisungen Bezeichner für die Basisadresse des DUART (`$effc01`) und die erforderlichen Offsets zu definieren.

Aufgabe 17 (3 Punkte)

Die Funktion `PUT_CHAR` erwartet als Argument einen Byte-Wert auf dem Stack. Bei der Übergabe über den Stack ist jedoch zu berücksichtigen, daß beim MC68k der `SP` immer auf Wort-Adressen zeigen muß. Angenommen das Argument für `PUT_CHAR` wird im Byte-Format auf den Stack geschoben:

```
MOVE.B  #$21,-(SP)
JSR     PUT_CHAR                                          .
```

Die `MOVE`-Anweisung dekrementiert `SP` dann nicht nur um 1, sondern um 2, damit wieder eine legale Adresse herauskommt. Entsprechend muß am Ende von `PUT_CHAR` die Rücksprungadresse nicht nur um ein Byte, sondern um ein Wort verschoben werden und der `SP` entsprechend hochgesetzt werden:

```
MOVE.L  (SP),2(SP)      kill argument frame: move return address
ADDQ.L  #2,SP                                adjust SP         .
        RTS
```

Die Musterlösung umgeht dieses Problem, indem alle Argumente stets als Long-Werte auf den Stack geschoben werden. Innerhalb der Funktion werden von einem solchen Long-Wert bei Bedarf nur das niederwertige Byte oder Wort berücksichtigt.

Aufgabe 18 (3 Punkte)

Die Funktion `PUT_STR` erhält als Parameter die Startadresse einer Byte-Sequenz. Die an dieser und nachfolgender Adressen vorgefundenen Bytes werden mit Hilfe der Funktion `PUT_CHAR` solange ausgeben, bis an einer Adresse der Byte-Wert `$00` auftritt. Es ist darauf zu achten, daß dieses Endsymbol selbst *nicht* ausgegeben wird!

Aufgabe 19 (10 Punkte)

Die Funktion `HTOA` wurde so konzipiert, daß sie drei Argumente benötigt: Die zu konvertierende Hex-Zahl, sowie die niedrigste und die höchste Adresse des Speicherbereichs, in dem die zu generierende Zeichenkette abgelegt werden soll. Alternativ wäre es möglich, auf die Übergabe einer der beiden Adressen zu verzichten. Die Vorgehensweise der Musterlösung hat jedoch zwei Vorteile: Zum einen kann `HTOA` feststellen, ob der Speicherbereich eine ausreichende Größe hat. Zum anderen kann bezüglich des aufrufenden Kontextes offen bleiben, ob `HTOA` den Speicherbereich von vorne oder von hinten aus beschreibt.

Mit Hilfe der folgenden Assembler-Anweisungen ist es möglich auf elegante Art und Weise ein Byte-Feld der Größe 100 anzulegen, wobei die Label `BUFFER_F` und `BUFFER_L` die Adressen des ersten bzw. des letzten Bytes bezeichnen:

```
BUFFER_F        DS.B    99
BUFFER_L        DS.B    1                                        .
```

Achtung: Falls im Anschluß an diese Zeilen noch Instruktionen stehen, muß darauf geachtet werden, daß der reservierte Speicherbereich eine gerade Anzahl von Bytes umfaßt! Andernfalls würde beim Assemblieren der `LC` (location counter) für die nachfolgende Instruktion u. U. nicht mehr auf eine Wort-Adresse zeigen, und bei der Ausführung des Maschinencodes käme es zu einem Busfehler (falsches Alignment).

Einziger Rückgabewert von `HTOA` ist die Startadresse der generierten Zeichenkette, es könnte also gleich im Anschluß die Funktion `PUT_STR` aufgerufen werden.

Beispiel: Die Funktion `HTOA` wird mit den beiden Adressen `$00001000` und `$0000100f` aufgerufen, der Speicherbereich umfaßt also 16 Bytes. Die gewählte Implementierung geht so vor, daß sie diesen Speicherbereich von hinten aus beschreibt. Die 10 Zeichen (`'$'`, acht Hex-Ziffern und `$00`) werden also auf den höchstwertigsten Adressen abgelegt und die Adresse `$00001006` zurückgeliefert.

Der **Assembler-Code** für die Aufgaben 16 – 19:

```
                ORG     $0
                DC.L    $8000           stack pointer value after a reset
                DC.L    START           program counter value after a reset

START:          JSR     INIT_IO         initialize the DUART
                BREAK
                MOVE.L  #$1fa79ad9,-(SP)
                JSR     PUT_HEX         call PUT_HEX
                BREAK


********************************************************************************
********************************************************************************
*
* INPUT / OUTPUT stuff
*

DUART           EQU     $effc01
MR1A            EQU     0
MR2A            EQU     0
SRA             EQU     2
CSRA            EQU     2
CRA             EQU     4
TBA             EQU     6

BUFFER_F        DS.B    $ff             buffer containing $100 bytes with start
BUFFER_L        DS.B    1               ... and end address
                * caution: should be a multiple of word in order to avoid
                * alignment problems in subsequent statements!!!


********************************************************************************
*
* function: void INIT_IO( )
*
* description:
*   Initializes the Dual UART Device
*
* registers affected: none
*

INIT_IO:        MOVEM.L A0,-(SP)                save registers on stack
                LEA     DUART,A0
                MOVE.B  #%00010000,CRA(A0)      reset MRxA pointer
                MOVE.B  #%00000011,MR1A(A0)     8 data bits
                MOVE.B  #%00000000,MR2A(A0)     normal mode
                MOVE.B  #%10111011,CSRA(A0)     set clock to 9600
                MOVE.B  #%00000100,CRA(A0)      enable Tx
                MOVEM.L (SP)+,A0                restore registers
                RTS


```

```
********************************************************************************
*
* function: void PUT_CHAR( (SP))
*
* description:
*   Prints the character that is passed via the stack (lowest byte of the long)
*
* registers affected: none
*


PUT_CHAR:       LINK    A6,#0           store pointer to argument frame in A6
                MOVEM.L D0,-(SP)        save registers on stack
                MOVE.L  8(A6),D0        fetch character

PC_LOOP:        BTST    #2,SRA+DUART    check whether port is ready
                BEQ.S   PC_LOOP
                MOVE.B  D0,TBA+DUART    write data to output

                MOVEM.L (SP)+,D0        restore registers
                UNLK    A6              restore A6
                MOVE.L  (SP),4(SP)      kill argument frame: move return address
                ADDQ.L  #4,SP                                adjust SP
                RTS


********************************************************************************
*
* function: void PUT_STR( (SP))
*
* description:
*   Prints the string (sequence of bytes) whose starting address is passed via
*   the stack (long)
*
* registers affected: none
*


PUT_STR:        LINK    A6,#0           store pointer to argument frame in A6
                MOVEM.L A0/D0,-(SP)     save registers on stack
                MOVE.L  8(A6),A0        fetch address of string

PS_LOOP:        MOVE.B  (A0)+,D0        fetch first character of string
                CMPI.B  #0,D0           is this the terminating 0?
                BEQ     PS_DONE
                MOVE.L  D0,-(SP)
                JSR     PUT_CHAR
                BRA     PS_LOOP

PS_DONE:        MOVEM.L (SP)+,A0/D0     restore registers
                UNLK    A6              restore A6
                MOVE.L  (SP),4(SP)      kill argument frame: move return address
                ADDQ.L  #4,SP                                adjust SP
                RTS


********************************************************************************
*
* function: void PUT_HEX( (SP))
*
* description:
*   Prints the hex number that is passed via the stack (long)
*
* registers affected: none
*

PUT_HEX:        LINK    A6,#0           store pointer to argument frame in A6
                MOVEM.L D0,-(SP)        save registers on stack
                MOVE.L  8(A6),D0        fetch hex number

                MOVE.L  #BUFFER_F,-(SP) push pointer to start of buffer
                MOVE.L  #BUFFER_L,-(SP) push pointer to end of buffer
                MOVE.L  D0,-(SP)        push hex number
                JSR     HTOA            create ascii-rep in buffer and
```

```
                                        * return ptr to result on stack
                JSR     PUT_STR         print buffer starting at returned ptr

                MOVEM.L (SP)+,D0        restore registers
                UNLK    A6              restore A6
                MOVE.L  (SP),4(SP)      kill argument frame: move return address
                ADDQ.L  #4,SP                                 adjust SP
                RTS


********************************************************************************
*
* function: (SP1) HTOA( (SP1), (SP2), (SP3))
*
* description:
*   Converts the unsigned hex number (SP3) into an ascii string which is put
*   into the buffer whose begin and end is pointed at by (SP1) and (SP2)
*   respectively. The starting address within the buffer is returned via the
*   stack (SP1).
*
* registers affected: none
*

HTOA:           LINK    A6,#0           store pointer to argument frame in A6
                MOVEM.L D0-D2/A0-A1,-(SP)
                MOVE.L  8(A6),D0        fetch hex number
                MOVE.L  12(A6),A1       fetch pointer to end of buffer
                MOVE.L  16(A6),A0       fetch pointer to start of buffer

                MOVE.B  #$00,(A1)       write terminating 0 into buffer
                MOVE.L  #8,D2           convert 8 digits

HTOA_LOOP:      MOVE.L  D0,D1
                ANDI.L  #$F,D1          separate last digit
                ADDI.B  #$30,D1         add ’0’ in order to get a char ’0’, ...
                CMPI.L  #$39,D1         is char > ’9’ ?
                BLE     HTOA2
                ADD.B   #$7,D1          add (’A’ - (’9’ + 1))
HTOA2:          CMPA.L  A1,A0           end of buffer reached?
                BEQ     HTOA_OOB
                MOVE.B  D1,-(A1)        write char into buffer
                LSR.L   #4,D0           shift addr 4 Bit right
                SUBI.L  #1,D2
                BNE     HTOA_LOOP

                CMPA.L  A1,A0           end of buffer reached?
                BEQ     HTOA_OOB
                MOVE.B  #$24,-(A1)      write ’$’ into buffer
                MOVE.L  A1,16(A6)       write result over buffer arg!
                JMP     HTOA_DONE

HTOA_OOB:       MOVE.L  12(A6),16(A6)   out of bounds: return empty string

HTOA_DONE:      MOVEM.L (SP)+,D0-D2/A0-A1
                UNLK    A6              restore A6
                MOVE.L  (SP),8(SP)      kill last two entries of arg.frame:
                ADDQ.L  #8,SP             move return address, adjust SP
                RTS


END
```