Übungen zu Organisation und Architektur von Rechnern
SS 2002
Serie 5 — Musterlösung

Aufgabe 14 (10 Punkte)

Die in der Aufgabenstellung geforderte Aufrufkonvention für die Funktion `FAC` wird wie folgt realisiert:

Der aufrufende Kontext übergibt den Parameter über den Stack. Bei Eintritt in die Funktion `FAC` liegt auf der Spitze des Stack also die Rücksprungadresse und darunter das Argument. Um im weiteren Verlauf elegant auf das Argumentframe zugreifen zu können, wird mit Hilfe der `LINK` Instruktion die initiale Adresse des Stackpointers `SP` im Register `A6` gesichert. Dadurch kann unabhängig von der späteren Position des `SP` immer mittels `8(A6)` auf das Argument zugegriffen werden. Als Seiteneffekt sichert `LINK` den aktuellen Wert von `A6` auf dem Stack, damit er beim Verlassen der Funktion wiederhergestellt werden kann.

Um die Anforderung, alle Register nach dem Funktionsaufruf unverändert zu hinterlassen, erfüllen zu können, werden anschließend unter Verwendung der Anweisung `MOVEM` alle durch die Funktion verwendeten Register auf dem Stack gesichert.

Am Ende der Funktion wird der Rückgabewert in das Argumentframe geschrieben und es werden die Register wieder mit Hilfe von `MOVEM` in ihren ursprünglichen Zustand zurückversetzt. Der abschließende `UNLK` Befehl wirkt komplementär zum `LINK` und holt den alten Wert des Registers `A6` vom Stack.

Nach dem Rücksprung mittels `RTS` befindet sich also, wie in der Aufgabenstellung gefordert, nur noch das Ergebnis der Berechnung oben auf dem Stack.

Nach selbigem Schema muß die Funktion `MULU_L` modifiziert werden. Hier der vollständige Quelltext:

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

START:          MOVE.L  #12,-(SP)       push the argument on the stack
                JSR     FAC             call factorial
                MOVE.L  (SP)+,D0        pop the result from the stack
                BREAK

                MOVE.L  #13,-(SP)
                JSR     FAC
                MOVE.L  (SP)+,D0
                BREAK


********************************************************************************
*
* function:   (SP) FAC( (SP))
*
* description:
*   computes the factorial of the number that was top of the runtime stack
*   right before calling the function. The result is returned via the very
*   same stack location.
*
* registers affected: none
*

FAC:            LINK    A6,#0           save A6 and set it to its stack location
                MOVEM.L D0-D1,-(SP)     save the registers used
                MOVE.L  8(A6),D0        fetch n

                CMPI    #0,D0           n == 0?
                BNE     ELSE
                MOVE.L  #1,D0           res (D0) = 1
                BRA     DONE
ELSE:           MOVE.L  D0,D1
                SUBI    #1,D1
```

```
                MOVE.L  D1,-(SP)        push (n-1)
                JSR     FAC             call factorial
                MOVE.L  D0,-(SP)
                JSR     MULU_L
                MOVE.L  (SP)+,D0        higher long of the result
                BEQ     OK
                MOVE.L  #0,D0           signalize an error
                BRA     DONE
OK:             MOVE.L  (SP)+,D0        lower long of the result

DONE:           MOVE.L  D0,8(A6)        save the result in the argument frame
                MOVEM.L (SP)+,D0-D1     restore the registers used
                UNLK    A6              restore A6
                RTS


********************************************************************************
*
* function:   (SP1), (SP2) MULU_L( (SP1), (SP2))
*
* description:
*   multiplies two long formats given in (SP1),(SP2). The result is returned
*   in (SP2),(SP1), which hold the higher and the lower long format of the
*   result, respectively.
*
* registers affected: none
*

MULU_L:         LINK    A6,#0
                MOVEM.L D0-D7,-(SP)
                MOVE.L  12(A6),D0
                MOVE.L  8(A6),D1

                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)
                MOVE.L  D1,D3
                MULU    D0,D3           D3 = LW( D0) * LW( D1)

                MULU    D1,D4           D4 = HW( D0) * LW( D1)
                MULU    D0,D5           D5 = LW( D0) * HW( D1)
                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 = $0000     HW(D4+D5)
                ANDI.L  #$FFFF0000,D5   D5 = LW(D4+D5) $0000
                ADD.L   D5,D3           LL( D0.L * D1.L)
                ADDX.L  D4,D2           HL( D0.L * D1.L)

                MOVE.L  D2,8(A6)        store D2
                MOVE.L  D3,12(A6)       store D3
                MOVEM.L (SP)+,D0-D7
                UNLK    A6
                RTS
```

Aufgabe 15 (10 Punkte)

Für die Funktionen `F` und `G` des Roller-Coasters wird das gleiche Programmgerüst wie für `FAC` in Aufgabe 14 verwendet. Da der `LINK` Befehl den aktuellen Wert des Registers `A6` auf dem Stack speichert, entsteht als Seiteneffekt von `LINK` eine Verkettung der Argumentframes. Dazu muß vor dem Aufruf einer Funktion das Register `A6` auf das zu der aufgerufenen Funktion übergeordnete Argumentframe zeigen. Folgende Fälle sind dabei zu unterscheiden:

a) *Aufruf einer Funktion, die lokal zur aktuellen Funktion definiert ist:*

   In diesem Fall zeigt `A6` bereits auf das übergeordnete Frame, da die aktuelle Funktion auch die übergeordnete ist.

Dies tritt im vorliegenden Beispiel auf, wenn die Funktion `G` innerhalb der Funktion `F` aufgerufen wird (Zeilen 36 f. im Quellcode).

b) *Aufruf einer Funktion, die auf gleicher Ebene definiert ist:*

Hier ist die der aktuellen Funktion übergeordnete Funktion auch die übergeordnete Funktion der aufzurufenden Funktion. Daher muß `A6` einmal dereferenziert werden, um auf das übergeordnete Frame zu zeigen.

Dieser Fall tritt im vorliegenden Beispiel ein, wenn die Funktion `G` sich selbst aufruft (Zeilen 91 f. im Quellcode).

c) *Aufruf einer Funktion, zu der die aktuelle Funktion lokal definiert ist:*

Da die aufgerufene Funktion der aufrufenden übergeordnet ist, muß `A6` mehrfach dereferenziert werden, abhängig davon, wie viele Ebenen die aufgerufene Funktion über der aufrufenden definiert wurde.

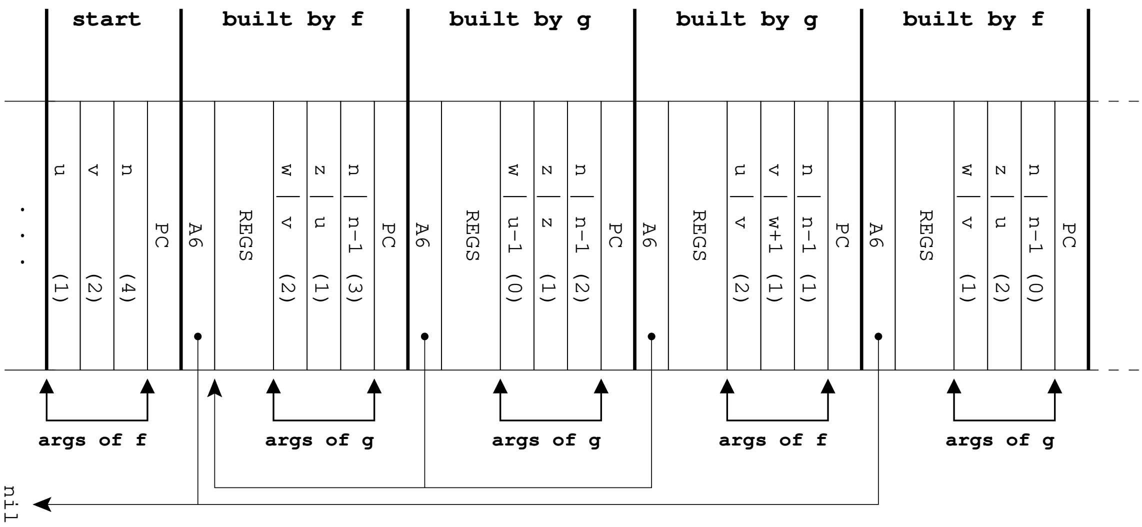
Beim Roller-Coaster ist dies dann der Fall, wenn die Funktion `G` die Funktion `F` aufruft. `F` ist eine Ebene über `G` definiert und somit muß `A6` zweimal dereferenziert werden (Zeilen 79 ff. im Quellcode).

Diese Regeln sind vollkommen analog zu den in der Vorlesung vorgestellten Regeln für die `SECDH-2` Maschine.

In den Fällen b) und c) ist sicherzustellen, daß die aufrufende Funktion den Wert von `A6` (auf dem Stack) sichert, bevor sie ihn für die aufzurufende Funktion modifiziert (z. B. Zeile 75), und ihn nach der Rückkehr aus der Unterfunktion wiederherstellt (z. B. Zeile 83). Ansonsten hätte sie anschließend zum einen keine Möglichkeit mehr, über `A6` auf ihr eigenes Argumentframe zuzugreifen, und zum anderen würde `UNLK` den `SP` auf einen falschen Wert setzen.

Um Zugriffe auf relativ freie Variablen zu realisieren, muß die durch `A6` referenzierte verkettet Liste von Argumentframes durchlaufen werden, bis die jeweilige Bindungsebene der Variablen erreicht ist (vgl. Zeilen 63 ff. im Quellcode).

Beim Aufruf von `F` mit den Argumenten 1, 2, 4 entsteht folgende Struktur auf dem Laufzeitstack:



Hier der Quellcode des Roller-Coasters:

```
1                ORG      $0
2                DC.L     $8000         stack pointer value after a reset
3                DC.L     START         program counter value after a reset
4
5   START:       MOVE.L   #1,-(SP)      put 1 into arg. frame for F
6                MOVE.L   #2,-(SP)      put 2 into arg. frame for F
7                MOVE.L   #20,-(SP)     put 20 into arg. frame for F
8                LEA.L    $0,A6         initialize frame pointer
9                                       * (there is no higher arg. frame yet)
```

```
                JSR     F               call F( 1, 2, 20)
                MOVE.L  (SP)+,D0        put result into D0
                BREAK


******************************************************************************
*
* function: (SP1) F( (SP1), (SP2), (SP3))
*
* description:
*   Expects an argument frame with three arguments on the stack.
*   The register A6 should contain the frame pointer of the actual environment.
*
* registers affected: none
*

F:              LINK    A6,#0           store pointer to arg. frame in A6
                MOVEM.L D0-D2,-(SP)     save registers
                MOVE.L  16(A6),D0       fetch u from actual arg. frame
                MOVE.L  12(A6),D1       fetch v from actual arg. frame
                MOVE.L  8(A6),D2        fetch n from actual arg. frame

                SUBI.L  #1,D2           n = n - 1
                MOVE.L  D1,-(SP)        put v into arg. frame for G
                MOVE.L  D0,-(SP)        put u into arg. frame for G
                MOVE.L  D2,-(SP)        put n into arg. frame for G
                                        * no frame pointer adjustment needed
                JSR     G               call G( v, u, n)
                MOVE.L  (SP)+,D0        put result into D0
                MOVE.L  D0,16(A6)       write result into arg. frame (1st entry)

                MOVEM.L (SP)+,D0-D2     restore registers
                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


******************************************************************************
*
* function: (SP1) G( (SP1), (SP2), (SP3))
*
* description:
*   Expects an argument frame with three arguments on the stack.
*   The register A6 should contain the frame pointer of the actual environment.
*
* registers affected: none
*

G:              LINK    A6,#0           store pointer to arg. frame in A6
                MOVEM.L D0-D4,-(SP)     save registers
                MOVE.L  16(A6),D0       fetch w from actual arg. frame
                MOVE.L  12(A6),D1       fetch z from actual arg. frame
                MOVE.L  8(A6),D2        fetch n from actual arg. frame
                MOVEA.L (A6),A0         fetch pointer to arg. frame of F
                MOVE.L  16(A0),D3       fetch u from arg. frame of F
                MOVE.L  12(A0),D4       fetch v from arg. frame of F

                CMPI.L  #0,D2           (n == 0) ?
                BLE     G_TAIL
                SUBI.L  #1,D2           n = n - 1
                CMP.L   D3,D0           (w > u) ?
                BGT     G_G

G_F:            ADDI.L  #1,D0           w = w + 1
                MOVE.L  A6,-(SP)        save A6
                MOVE.L  D4,-(SP)        put v into arg. frame for F
                MOVE.L  D0,-(SP)        put w into arg. frame for F
                MOVE.L  D2,-(SP)        put n into arg. frame for F
                MOVEA.L (A6),A6         adjust frame pointer for F
                MOVEA.L (A6),A6         adjust frame pointer for F
                JSR     F               call F( v, w, n)
                MOVE.L  (SP)+,D0        res = F( ...)
                MOVE.L  (SP)+,A6        restore A6
```

```
                JMP     G_END

G_G:            SUBI.L  #1,D3           u = u - 1
                MOVE.L  A6,-(SP)        save A6
                MOVE.L  D3,-(SP)        put u into arg. frame for G
                MOVE.L  D1,-(SP)        put z into arg. frame for G
                MOVE.L  D2,-(SP)        put n into arg. frame for G
                MOVEA.L (A6),A6         adjust frame pointer for G
                JSR     G               call G( u, z, n)
                MOVE.L  (SP)+,D0        res = G( ...)
                MOVE.L  (SP)+,A6        restore A6
                JMP     G_END

G_TAIL:         ADD.L   D1,D0           res = w + z

G_END:          MOVE.L  D0,16(A6)       write result into arg. frame (1st entry)
                MOVEM.L (SP)+,D0-D4     restore registers
                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
```