Übungen zu Organisation und Architektur von Rechnern
SS 2002
Serie 7 — Musterlösung

Aufgabe 20 – 22 (10 + 10 + 20 Punkte)

Der **Assembler-Code** für die Aufgaben 20 – 22:

```
1  ********************************************************************************
2  ********************************************************************************
3  **
4  **  Interpreter for the SECDH-2 machine
5  **
6  ********************************************************************************
7  ********************************************************************************
8
9
10 ********************************************************************************
11 *
12 * Here, we define the opcodes for the abstract machine.
13 * Since we want to use them as offsets into an opcode table, they have to be
14 * aligned to long, i.e., they have to be a multiple of 4.
15 * The last entry should be LAST_OP which is used to determine the size of the
16 * opcode table only.
17 *
18
19 PUSH_S          EQU     $00000000
20 PUSH_ES         EQU     $00000004
21 MKFRAME         EQU     $00000008
22 FREE            EQU     $0000000c
23 BRC             EQU     $00000010
24 CALL            EQU     $00000014
25 RET             EQU     $00000018
26 AP              EQU     $0000001c
27 PLUS            EQU     $00000020
28 MINUS           EQU     $00000024
29 MULT            EQU     $00000028
30 EQ              EQU     $0000002c
31 GT              EQU     $00000030
32 ARGS            EQU     $00000034
33 LINK            EQU     $00000038
34 CREATE_CLOS     EQU     $0000003c
35 DONE            EQU     $00000040
36 LAST_OP         EQU     $00000044
37
38
39 ********************************************************************************
40
41                 ORG     $0
42                 DC.L    $8000           Stack pointer value after a reset
43                 DC.L    START           Program counter value after a reset
44
45
46 ********************************************************************************
47 *
48 * Here, we reserve some space for the opcode table
49 *
50
51 OP_TAB          DS.L    LAST_OP
52
53
54 ********************************************************************************
55 *
56 * Here, we reserve memory space for the stacks S, E, and D.
57 * The labels x_STACK_T point to the stack bottom:
58 *
59
```

```
60  STACK_REG       DS.L    100
61  S_STACK_T       DS.L    100
62  E_STACK_T       DS.L    100
63  D_STACK_T       DS.L    1
64
65
66  ********************************************************************************
67  *
68  * In the following part of the memory we keep our programs, i.e., the portion
69  * of the heap denoted by C
70  *
71
72  P_FAC           DC.L    ARGS,1,MKFRAME,1,PUSH_ES,0,0,PUSH_S,0,EQ,BRC,P_T,P_F,FREE,1,RET
73  P_T             DC.L    PUSH_S,1,RET
74  P_F             DC.L    PUSH_S,1,PUSH_ES,0,0,MINUS,LINK,0,CALL,P_FAC,PUSH_ES,0,0,MULT,RET
75
76  P_FAC_PRG       DC.L    PUSH_S,4,LINK,-1,CALL,P_FAC,DONE
77  * ^^^^^^^ Example
78
79
80  P_Y             DC.L    ARGS,1,MKFRAME,1,CREATE_CLOS,-1,P_A,LINK,-1,CALL,P_A,FREE,1,RET
81  P_A             DC.L    ARGS,1,MKFRAME,1,CREATE_CLOS,-1,P_A2,PUSH_ES,1,0,AP,1,FREE,1,RET
82  P_A2            DC.L    ARGS,1,MKFRAME,1,PUSH_ES,0,0,PUSH_ES,1,0,PUSH_ES,1,0,AP,1,AP,1,FREE,1,RET
83
84  P_FACP          DC.L    ARGS,1,MKFRAME,1,CREATE_CLOS,-1,P_FACP2,FREE,1,RET
85  P_FACP2         DC.L    ARGS,1,MKFRAME,1,PUSH_ES,0,0,PUSH_S,0,EQ,BRC,P_TP,P_FP,FREE,1,RET
86  P_TP            DC.L    PUSH_S,1,RET
87  P_FP            DC.L    PUSH_S,1,PUSH_ES,0,0,MINUS,PUSH_ES,1,0,AP,1,PUSH_ES,0,0,MULT,RET
88
89  P_FACP_PRG      DC.L    PUSH_S,4,CREATE_CLOS,-1,P_FACP,LINK,-1,CALL,P_Y,AP,1,DONE
90  * ^^^^^^^ Exercise 22d
91
92
93  P_K             DC.L    ARGS,1,MKFRAME,1,CREATE_CLOS,-1,P_KI,FREE,1,RET
94  P_KI            DC.L    ARGS,1,MKFRAME,1,PUSH_ES,1,0,FREE,1,RET
95
96  P_AP3           DC.L    ARGS,1,MKFRAME,1,PUSH_S,3,PUSH_ES,0,0,AP,1,FREE,1,RET
97
98  P_K_PRG2        DC.L    PUSH_S,2,LINK,-1,CALL,P_K,LINK,-1,CALL,P_AP3,DONE
99  * ^^^^^^^ Exercise 22c
100
101 P_K_PRG         DC.L    PUSH_S,3,PUSH_S,2,LINK,-1,CALL,P_K,AP,1,DONE
102 * ^^^^^^^ Exercise 22b
103
104
105 P_RC_G          DC.L    ARGS,2,MKFRAME,2,PUSH_ES,1,1,PUSH_ES,0,1,GT,BRC,P_RC_GT,P_RC_GF,FREE,2,RET
106 P_RC_GT         DC.L    PUSH_ES,0,0,PUSH_S,1,PUSH_ES,1,1,MINUS,LINK,0,CALL,P_RC_G,RET
107 P_RC_GF         DC.L    PUSH_S,1,PUSH_ES,0,1,PLUS,PUSH_ES,1,0,LINK,1,CALL,P_RC_F,RET
108 P_RC_F          DC.L    ARGS,2,MKFRAME,2,PUSH_ES,0,1,PUSH_ES,0,0,LINK,-1,CALL,P_RC_G,FREE,2,RET
109 P_RC_PRG        DC.L    PUSH_S,2,PUSH_S,1,LINK,-1,CALL,P_RC_F,DONE
110 * ^^^^^^^ Exercise 22a
111
112
113 ********************************************************************************
114 *
115 * Here, we reserve a heap area for storing the closures:
116 *
117
118 HEAP_START      DS.L    1000
119 HEAP_STOP       DS.L    1
120
121
122 ********************************************************************************
123 ********************************************************************************
124 *
125 * Here, the implementation of the SEMCD-H2 interpreter starts:
126 *
127
128 ********************************************************************************
129 *
130 * First, we define some string constants:
131 *
132
133 START_STR       DC.B    'starting secdh-2 interpreter...',0
```

```
134 DONE_STR        DC.B    'finished',0,0
135 RES_STR         DC.B    'the result is: ',0
136
137
138 ********************************************************************************
139 *
140 * Now, the main program:
141 *
142
143 START:          JSR     INIT_OP_TAB             initialize opcode table
144                 JSR     INIT_IO                 initialize DUART
145
146                 JSR     CLEAR_SCREEN
147                 MOVE.L  #START_STR,-(SP)
148                 JSR     PUT_STR
149
150                 MOVE.L  #P_FACP_PRG,-(SP)       push program pointer on stack
151                 JSR     EXEC                    execute program
152
153                 MOVE.L  #DONE_STR,-(SP)
154                 JSR     PUT_STR
155                 JSR     NEW_LINE
156                 MOVE.L  #RES_STR,-(SP)
157                 JSR     PUT_STR
158                 JSR     PUT_OBJ                 print result (still on stack)
159                 JSR     NEW_LINE
160                 BREAK
161
162 *  second run -- this time in debugging mode
163
164                 JSR     CLEAR_SCREEN
165                 MOVE.L  #START_STR,-(SP)
166                 JSR     PUT_STR
167
168                 MOVE.L  #P_FACP_PRG,-(SP)       push program pointer on stack
169                 JSR     EXEC_IO                 execute program
170
171                 MOVE.L  #DONE_STR,-(SP)
172                 JSR     PUT_STR
173                 JSR     NEW_LINE
174                 MOVE.L  #RES_STR,-(SP)
175                 JSR     PUT_STR
176                 JSR     PUT_OBJ                 print result (still on stack)
177                 JSR     NEW_LINE
178                 BREAK
179
180
181 ********************************************************************************
182 *
183 * function:  void INIT_OP_TAB()
184 *
185 * description:
186 *   initializes the opcode table at OP_TAB with pointers to the subroutines
187 *   I_xxx that implement the operations xxx.
188 *
189 * registers affected: none
190 *
191
192 INIT_OP_TAB:    MOVE.L  A0,-(SP)                save A0
193                 MOVE.L  #OP_TAB,A0
194                 MOVE.L  #I_PUSH_S,PUSH_S(A0)
195                 MOVE.L  #I_PUSH_ES,PUSH_ES(A0)
196                 MOVE.L  #I_LINK,LINK(A0)
197                 MOVE.L  #I_MKFRAME,MKFRAME(A0)
198                 MOVE.L  #I_FREE,FREE(A0)
199                 MOVE.L  #I_BRC,BRC(A0)
200                 MOVE.L  #I_CALL,CALL(A0)
201                 MOVE.L  #I_ARGS,ARGS(A0)
202                 MOVE.L  #I_RET,RET(A0)
203                 MOVE.L  #I_AP,AP(A0)
204                 MOVE.L  #I_CREATE_CLOS,CREATE_CLOS(A0)
205                 MOVE.L  #I_PLUS,PLUS(A0)
206                 MOVE.L  #I_MINUS,MINUS(A0)
207                 MOVE.L  #I_MULT,MULT(A0)
```

```
208                 MOVE.L  #I_EQ,EQ(A0)
209                 MOVE.L  #I_GT,GT(A0)
210                 MOVE.L  #I_DONE,DONE(A0)
211                 MOVE.L  (SP)+,A0                restore A0
212                 RTS
213
214
215 ********************************************************************************
216 *
217 * function:  (SP1) DEREF( (SP1), (SP2))
218 *
219 * description:
220 *   dereferences the pointer (SP1) (SP2) many times and returns the resulting
221 *   pointer. It is assumed that (SP2) >= #0 holds and that all pointers to be
222 *   followed are within the legal range.
223 *
224 * registers affected: none
225 *
226
227 DEREF:          LINK    A6,#0
228                 MOVEM.L D0/A0,-(SP)     save A0 and D0
229                 MOVE.L  8(A6),D0        fetch sp2
230                 MOVE.L  12(A6),A0       fetch sp1
231
232                 CMPI.L  #0,D0
233                 BEQ     DREF_DONE
234 DR_LOOP:        MOVE.L  (A0),A0         deref once
235                 SUBI.L  #1,D0
236                 BNE     DR_LOOP
237                 MOVE.L  A0,12(A6)       write result on SP1
238
239 DREF_DONE:      MOVEM.L (SP)+,D0/A0     restore A0 and D0
240                 UNLK    A6              restore A6
241                 MOVE.L  (SP),4(SP)      eliminate argument frame
242                 ADDQ.L  #4,SP
243                 RTS
244
245
246 ********************************************************************************
247 *
248 * function:  (SP1) COPY_ENV( (SP1), (SP2))
249 *
250 * description:
251 *   Expects (SP2) to point to an environment which is t be copied into the
252 *   memory starting at (SP1). On return, (SP1) points to the memory cell right
253 *   after the last cell copied.
254 *
255 * registers affected: none
256 *
257
258 COPY_ENV:       LINK    A6,#0
259                 MOVEM.L A0-A2/D0-D1,-(SP)
260                 MOVE.L  8(A6),A0        fetch sp2 (env ptr)
261                 MOVE.L  12(A6),A1       fetch sp1 (heap loc)
262
263 CE_LOOP:        MOVE.L  (A0)+,A2        store link pointer of env in A2
264                 CMPA.L  #0,A2           is this the nil-link?
265                 BEQ     CE_DONE
266                 MOVE.L  (A0)+,D0        store frame size in D0
267                 MOVE.L  D0,D1           compute addr of next frame in D1:
268                 MULU.W  #4,D1             frame-size * 4
269                 ADD.L   A1,D1             + actual addr + 8 (link / size entry)
270                 ADDI.L  #8,D1           now D1 holds addr of next heap frame
271                 MOVE.L  D1,(A1)+        store this ptr in heap
272                 MOVE.L  D0,(A1)+        store frame size in heap
273 CE_FR_LOOP:     CMPI.L  #0,D0
274                 BEQ     CE_FR_DONE
275                 MOVE.L  (A0)+,(A1)+     copy next entry into heap
276                 SUBI.L  #1,D0
277                 BRA     CE_FR_LOOP
278 CE_FR_DONE:     MOVE.L  A2,A0           make link ptr (in e) the new env ptr
279                 BRA     CE_LOOP
280
281 CE_DONE:        MOVE.L  #0,(A1)+        write terminating NULL frame
```

```
282
283                 MOVE.L  A1,12(A6)       write final heap ptr into SP1
284                 MOVEM.L (SP)+,A0-A2/D0-D1
285                 UNLK    A6              restore A6
286                 MOVE.L  (SP),4(SP)      eliminate argument frame
287                 ADDQ.L  #4,SP
288                 RTS
289
290
291 ********************************************************************************
292 *
293 * function:  (SP) EXEC( (SP))
294 *
295 * description:
296 *   This is the main execution cycle which initializes the SECDH-2 interpreter,
297 *   and starts the interpretation at (SP).
298 *   Basically, it repeatedly reads the opcode behind the P_C, looks up the
299 *   address of the code that implements the operation (in the opcode table), and
300 *   jumps to that code.
301 *
302 *   The last instruction should be "DONE" which leads to the jump address I_DONE
303 *   and thus exits the loop EXEC_LOOP.
304 *   The result found on top of P_S is returned via (SP).
305 *
306 *   NOTE: A6 serves two purposes here. During the program interpretation, it
307 *         serves as pointer into the heap P_H; otherwise, it is used as frame
308 *         pointer.
309 *
310 * registers affected: none
311 *
312
313 EXEC:           LINK    A6,#0
314                 MOVEM.L D0-D7/A0-A5,-(SP)
315                 MOVE.L  8(A6),A0        fetch SP (program ptr)
316
317                 MOVE.L  A6,-(SP)        save frame pointer
318
319                 MOVE.L  #S_STACK_T,A2   init P_S (A2)
320                 MOVE.L  #E_STACK_T,A3   init P_E (A3)
321                 MOVE.L  #0,-(A3)        push nil-link to P_E
322                 MOVE.L  A0,A4           init P_C (A4) with prg to be evaluated
323                 MOVE.L  #D_STACK_T,A5   init P_D (A5)
324                 MOVE.L  #HEAP_START,A6  init P_H (A6)
325                 MOVE.L  #OP_TAB,A0
326
327 EXEC_LOOP:      MOVE.L  (A4)+,D0        load opcode into D0 and increment P_C
328                 MOVE.L  (A0,D0.L),A1    dispatch opcode into instruction address
329                 JSR     (A1)
330                 CMPA.L  #I_DONE,A1
331                 BNE     EXEC_LOOP
332
333                 MOVE.L  (SP)+,A6        restore frame pointer
334
335                 MOVE.L  (A2)+,8(A6)     move result (top of P_S) into SP
336                 MOVEM.L (SP)+,D0-D7/A0-A5
337                 UNLK    A6
338                 RTS
339
340
341 ********************************************************************************
342 *
343 * function:  (SP) EXEC_IO( (SP))
344 *
345 * description:
346 *   Same as EXEC together with I/O.
347 *
348 * registers affected: none
349 *
350
351 EXEC_IO:        LINK    A6,#0
352                 MOVEM.L D0-D7/A0-A5,-(SP)
353                 MOVE.L  8(A6),A0        fetch SP (program ptr)
354
355                 MOVE.L  A6,-(SP)        save frame pointer
```

```
356
357                 MOVE.L  #S_STACK_T,A2   init P_S (A2)
358                 MOVE.L  #E_STACK_T,A3   init P_E (A3)
359                 MOVE.L  #0,-(A3)        push nil-link to P_E
360                 MOVE.L  A0,A4           init P_C (A4) with prg to be evaluated
361                 MOVE.L  #D_STACK_T,A5   init P_D (A5)
362                 MOVE.L  #HEAP_START,A6  init P_H (A6)
363                 MOVE.L  #OP_TAB,A0
364
365 EXEC_IO_LOOP:   MOVE.L  A2,-(SP)         *
366                 MOVE.L  #S_STACK_T,-(SP) *  push arguments for
367                 MOVE.L  A3,-(SP)         *     printing actual state
368                 MOVE.L  A4,-(SP)         *
369                 MOVE.L  A5,-(SP)         *
370                 MOVE.L  #D_STACK_T,-(SP) *
371                 JSR     PR_STATE
372                 JSR     GETC
373                 MOVE.L  (A4)+,D0        load opcode into D0 and increment P_C
374                 MOVE.L  (A0,D0.L),A1    dispatch opcode into instruction address
375                 JSR     (A1)
376                 CMPA.L  #I_DONE,A1
377                 BNE     EXEC_IO_LOOP
378
379                 MOVE.L  (SP)+,A6        restore frame pointer
380
381                 MOVE.L  (A2)+,8(A6)     move result (top of P_S) into SP
382                 MOVEM.L (SP)+,D0-D7/A0-A5
383                 UNLK    A6
384                 RTS
385
386
387 ********************************************************************************
388 *
389 * Now, the implementations of the individual operations follow.
390 *
391 ********************************************************************************
392 *
393 * function:  A2, A3, A4, A5, A6 I_xxx( A2, A3, A4, A5, A6)
394 *
395 * description:
396 *   Implements the SECDH-2 operation xxx.
397 *   A2 contains P_S, A3 contains P_E, A4 contains P_C, A5 contains P_D,
398 *   A6 contains P_H.
399 *
400 * registers which might be affected: D0-D7
401 *
402
403 I_PUSH_S:       MOVE.L  (A4)+,D0        load atom from code heap
404                 LSL.L   #2,D0
405                 ORI.L   #1,D0           mark as integer
406                 MOVE.L  D0,-(A2)        push atom on s
407                 RTS
408
409 I_PUSH_ES:      MOVE.L  A0,-(SP)        save A0
410                 MOVE.L  (A4)+,D0        fetch parameter i
411                 MOVE.L  (A4)+,D1        fetch parameter j
412                 MOVE.L  A3,-(SP)
413                 MOVE.L  D0,-(SP)
414                 JSR     DEREF           get i-th indirection starting at P_E
415                 MOVE.L  (SP)+,A0        fetch i-th indirection
416                 LSL.L   #2,D1           multiply D1 by 4
417                                         * (log. offset => byte offset)
418                 MOVE.L  8(A0,D1.L),-(A2)  pick j-th entry of the i-th env
419                                         * and push it on s
420                 MOVE.L  (SP)+,A0        restore A0
421                 RTS
422
423 I_MKFRAME:      MOVE.L  (A4)+,D0        fetch frame size parameter
424                 MOVE.L  (A2)+,D1        fetch link pointer for new env frame
425                 MOVE.L  D0,D2           preserve frame size in D2
426 I_MKF_LOOP:     CMPI.L  #0,D0
427                 BEQ     I_MKF_DONE
428                 MOVE.L  (A2)+,-(A3)     move top elem from s to e
429                 SUBI.L  #1,D0
```

```
                BRA     I_MKF_LOOP
I_MKF_DONE:     MOVE.L  D2,-(A3)        push frame size on stack
                MOVE.L  D1,-(A3)        push link pointer on e
                RTS

I_FREE:         MOVE.L  (A4)+,D0        fetch parameter
                ADDI.L  #2,D0           +2 in order to skip link and size info
                MULU.W  #4,D0           multiply by length of entry (4 byte)
                ADD.L   D0,A3
                RTS

I_BRC:          MOVE.L  (A4)+,D0        fetch p_then
                MOVE.L  (A4)+,D1        fetch p_else
                MOVE.L  (A2)+,D2        fetch predicate from s
                MOVE.L  A4,-(A5)        push actual P_C to d-stack
                CMPI.L  #7,D2
                BEQ     I_BRC_ELSE
                MOVE.L  D0,A4           set P_C to p_then
                RTS
I_BRC_ELSE:     MOVE.L  D1,A4           set P_C to p_else
                RTS

I_CALL:         MOVE.L  (A4)+,D0        load P_C from program code
                MOVE.L  A4,-(A5)        push actual P_C to d-stack
                MOVE.L  D0,A4           set P_C to new address
                RTS

I_RET:          MOVE.L  (A5)+,A4        retore P_C from d
                RTS

I_AP:           MOVEM.L A0-A1,-(SP)
                MOVE.L  (A4)+,D0        fetch parameter r
                MOVE.L  (A2)+,D1        fetch cls-ptr from s
                ANDI.L  #$FFFFFFFD,D1   eliminate closure tag
                MOVE.L  D1,A0
                MOVE.L  (A0)+,A1        fetch p_fun
                MOVE.L  4(A1),D1        fetch arity behind ARGS op
                CMP.L   D1,D0           check arity conformity
                BNE     ERROR
                MOVE.L  A0,-(A2)        push p_env to s
                MOVE.L  A4,-(A5)        push actual P_C to d-stack
                MOVE.L  A1,A4           set P_C to fun-ptr
                MOVEM.L (SP)+,A0-A1
                RTS

I_PLUS:         MOVE.L  (A2)+,D0        fetch first operand from s
                MOVE.L  (A2)+,D1        fetch second operand from s
                ADD.L   D1,D0
                EORI.L  #3,D0           correct tag
                MOVE.L  D0,-(A2)        push result
                RTS

I_MINUS:        MOVE.L  (A2)+,D0        fetch first operand from s
                MOVE.L  (A2)+,D1        fetch second operand from s
                SUB.L   D1,D0
                ORI.L   #1,D0           correct tag
                MOVE.L  D0,-(A2)        push result
                RTS

I_MULT:         MOVE.L  (A2)+,D0        fetch first operand from s
                ASR.L   #2,D0           eliminate tag
                MOVE.L  (A2)+,D1        fetch second operand from s
                ASR.L   #2,D1           eliminate tag
                MULS.W  D1,D0
                LSL.L   #2,D0
                ORI.L   #1,D0           set tag
                MOVE.L  D0,-(A2)        push result
                RTS

I_EQ:           MOVE.L  (A2)+,D0        fetch first operand from s
                MOVE.L  (A2)+,D1        fetch second operand from s
                CMP.L   D0,D1
                BEQ     I_EQ_TRUE
                MOVE.L  #7,-(A2)        push false
```

```
                RTS
I_EQ_TRUE:      MOVE.L  #3,-(A2)        push true
                RTS

I_GT:           MOVE.L  (A2)+,D0        fetch first operand from s
                MOVE.L  (A2)+,D1        fetch second operand from s
                CMP.L   D1,D0
                BGT     I_GT_TRUE
                MOVE.L  #7,-(A2)        push false
                RTS
I_GT_TRUE:      MOVE.L  #3,-(A2)        push true
                RTS

I_ARGS:         MOVE.L  (A4)+,D0        fetch parameter n and ignore it (-8
                RTS

I_LINK:         MOVE.L  (A4)+,D0        fetch parameter n
                ADDI.L  #1,D0           increment link parameter
                MOVE.L  A3,-(SP)
                MOVE.L  D0,-(SP)
                JSR     DEREF           get (n+1)-th indirection starting at P_E
                MOVE.L  (SP)+,-(A2)     push it on s
                RTS

I_CREATE_CLOS:  MOVE.L  A6,-(A2)        push closure ptr on s
                ORI.L   #2,(A2)         tag closure
                MOVE.L  (A4)+,D0        fetch parameter n
                MOVE.L  (A4)+,(A6)+     fetch parameter p_fun, push it onto heap
                ADDI.L  #1,D0           increment link parameter

                MOVE.L  A6,-(SP)        first arg for COPY_ENV

                MOVE.L  A3,-(SP)
                MOVE.L  D0,-(SP)
                JSR     DEREF           get n-th indirection starting at P_E
                                        * which is the second arg of COPY_ENV

                JSR     COPY_ENV        copy env into heap
                MOVE.L  (SP)+,A6        fetch new heap ptr (res of COPY_ENV)
                RTS

I_DONE:         RTS                     no operation

ERROR:          BREAK


*******************************************************************************

                INCLUDE io-pack.ss
```