

```
1: /* scheme/make_sob_vector.asm
2:  * Takes V1, ..., Vn, n, on the stack. Places in R0 the address
3:  * of a newly-allocated pointer to a Scheme vector.
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: MAKE_SOB_VECTOR:
9:     PUSH(FP);
10:    MOV(FP, SP);
11:    PUSH(R1);
12:    PUSH(R2);
13:    PUSH(R3);
14:    MOV(R0, FPARG(0));
15:    ADD(R0, IMM(2));
16:    PUSH(R0);
17:    CALL(MALLOC);
18:    DROP(1);
19:    MOV(IND(R0), IMM(T_VECTOR));
20:    MOV(INDD(R0, 1), FPARG(0));
21:    MOV(R1, FP);
22:    MOV(R2, FPARG(0));
23:    ADD(R2, IMM(3));
24:    MUL(R2, IMM(WORD_SIZE));
25:    SUB(R1, R2);
26:    MOV(R2, R0);
27:    ADD(R2, IMM(2));
28:    MOV(R3, FPARG(0));
29: L_MSV_LOOP:
30:    CMP(R3, IMM(0));
31:    JUMP_EQ(L_MSV_EXIT);
32:    MOV(IND(R2), REF(R1));
33:    ADD(R1, IMM(WORD_SIZE));
34:    INCR(R2);
35:    DECR(R3);
36:    JUMP(L_MSV_LOOP);
37: L_MSV_EXIT:
38:    POP(R3);
39:    POP(R2);
40:    POP(R1);
41:    POP(FP);
42:    RETURN;
43:
44:
```

```
1: /* abs.asm
2:  * Computes the absolute value of its argument:
3:  *   R0 <- | ARG[0] |
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: ABS:
9:   MOV(R0, STARG(0));
10:  CMP(R0, IMM(0));
11:  JUMP_LT(L_ABS_N);
12:  RETURN;
13: L_ABS_N:
14:  MOV(R1, R0);
15:  MOV(R0, IMM(0));
16:  SUB(R0, R1);
17:  RETURN;
```

```
1: /* ack.asm
2:  * Computes Ackermann's function: R0 <- ACK(ARG[0], ARG[1])
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: ACK:
8:   MOV(R1, STARG(0)); /* ARG[0] = A */
9:   MOV(R2, STARG(1)); /* ARG[1] = B */
10:  CMP(R1, IMM(0));
11:  JUMP_EQ(L_ACK_0_B);
12:  CMP(R2, IMM(0));
13:  JUMP_EQ(L_ACK_A_0);
14:  DECR(R2);
15:  PUSH(R2);
16:  PUSH(R1);
17:  CALL(ACK);
18:  POP(R1);
19:  POP(R1);
20:  MOV(STARG(1), R0);
21:  DECR(STARG(0));
22:  JUMP(ACK); /* tail-call optimization */
23:
24: L_ACK_A_0:
25:   MOV(STARG(1), IMM(1));
26:   DECR(STARG(0));
27:   JUMP(ACK); /* tail-call optimization */
28:
29: L_ACK_0_B:
30:   MOV(R0, R2);
31:   INCR(R0);
32:   RETURN;
```

```
1: /* char/char_in_range.asm
2:  * R0 <- (ARG[1] <= ARG[0] <= ARG[2])
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: CHAR_IN_RANGE:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    PUSH(R1);
11:    MOV(R1, FPARG(0));
12:    CMP(FPARG(1), R1);
13:    JUMP_GT(L_CIR_FALSE);
14:    CMP(R1, FPARG(2));
15:    JUMP_GT(L_CIR_FALSE);
16:    MOV(R0, IMM(1));
17:    JUMP(L_CIR_EXIT);
18: L_CIR_FALSE:
19:    MOV(R0, IMM(0));
20: L_CIR_EXIT:
21:    POP(R1);
22:    POP(FP);
23:    RETURN;
```

```
1: /* char/char_to_digit.asm
2:  * '0' -> 0, ..., '9' -> 9
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: CHAR_TO_DIGIT:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    MOV(R0, FPARG(0));
11:    SUB(R0, IMM('0'));
12:    POP(FP);
13:    RETURN;
```

```
1: /* char/char_to_lc.asm
2:  * R0 <- to_lc(ARG[0])
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: CHAR_TO_LC:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    PUSH(R1);
11:    MOV(R1, FPARG(0));
12:    PUSH(R1);
13:    CALL(L_IS_CHAR_UC);
14:    POP(R1);
15:    CMP(R0, IMM(0));
16:    JUMP_EQ(L_CHAR_TO_LC);
17:    MOV(R0, FPARG(0));
18:    ADD(R0, IMM('a' - 'A'));
19:    JUMP(L_CTL_C_EXIT);
20: L_CHAR_TO_LC:
21:    MOV(R0, FPARG(0));
22: L_CTL_C_EXIT:
23:    POP(R1);
24:    POP(FP);
25:    RETURN;
```

```
1: /* char/char_to_uc.asm
2:  * R0 <- to_uc(ARG[0])
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: CHAR_TO_UC:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    PUSH(R1);
11:    MOV(R1, FPARG(0));
12:    PUSH(R1);
13:    CALL(L_IS_CHAR_LC);
14:    POP(R1);
15:    CMP(R0, IMM(0));
16:    JUMP_EQ(L_CTUC);
17:    MOV(R0, FPARG(0));
18:    SUB(R0, IMM('a' - 'A'));
19:    JUMP(L_CTUC_EXIT);
20: L_CTUC:
21:    MOV(R0, FPARG(0));
22: L_CTUC_EXIT:
23:    POP(R1);
24:    POP(FP);
25:    RETURN;
```

```
1: /* char/digit_to_char.asm
2:  * 0 -> '0', ..., 9 -> '9'
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: DIGIT_TO_CHAR:
8:     MOV(R0, STARG(0));
9:     ADD(R0, IMM('0'));
10:    RETURN;
11:
```



```
1: /* fact.asm
2:  * Compute the factorial function recursively: R0 <- factorial(ARG0)
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: FACT:
8:     MOV(R1, STARG(0));
9:     CMP(R1, IMM(0));
10:    JUMP_EQ(L_FACT_ZERO);
11:    DECR(R1);
12:    PUSH(R1);
13:    CALL(FACT);
14:    POP(R1); /* pop arg (N-1) to fact */
15:    MUL(R0, STARG(0));
16:    RETURN;
17: L_FACT_ZERO:
18:    MOV(R0, IMM(1));
19:    RETURN;
```

```
1: /* fib.asm
2:  * Compute Fibonacci numbers recursively: R0 <- FIB(ARG[0])
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: FIB:
8:     MOV(R1, STARG(0));
9:     CMP(R1, IMM(2));
10:    JUMP_GE(L_FIB_REC);
11:    MOV(R0, R1);
12:    RETURN;
13:
14: L_FIB_REC:
15:    DECR(R1);
16:    PUSH(R1);
17:    CALL(FIB);
18:    POP(R1);
19:    PUSH(R0);
20:    DECR(R1);
21:    PUSH(R1);
22:    CALL(FIB);
23:    POP(R1);
24:    POP(R1);
25:    ADD(R0, R1);
26:    RETURN
27:
```

```
1: /* io/getchar.asm
2:  * Read a char from stdin
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: GETCHAR:
8:     IN(R0, IMM(1));
9:     RETURN;
```

```
1: /* char/is_char_uc.asm
2:  * R0 <- ('a' <= ARG[0] <= 'z') || ('A' <= ARG[0] <= 'Z')
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: IS_CHAR_ALPHABETIC:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    PUSH(R1);
11:    MOV(R1, FPARG(0));
12:    PUSH(R1);
13:    CALLL(IS_CHAR_UC);
14:    POP(R1);
15:    CMP(R0, IMM(0));
16:    JUMP_NE(L_ICAT_EXIT);
17:    MOV(R1, FPARG(0));
18:    PUSH(R1);
19:    CALLL(IS_CHAR_LC);
20:    POP(R1);
21:    JUMP(L_ICAT_EXIT);
22: L_ICAT_EXIT:
23:    POP(R1);
24:    POP(FP);
25:    RETURN;
```

```
1: /* char/is_char_lc.asm
2:  * R0 <- ('a' <= ARG[0] <= 'z')
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: IS_CHAR_LC:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    PUSH(R1);
11:    MOV(R1, FPARG(0));
12:    PUSH(IMM('z'));
13:    PUSH(IMM('a'));
14:    PUSH(R1);
15:    CALL(CHAR_IN_RANGE);
16:    POP(R1);
17:    POP(R1);
18:    POP(R1);
19:    POP(R1);
20:    POP(FP);
21:    RETURN;
```

```
1: /* char/is_char_uc.asm
2:  * R0 <- ('A' <= ARG[0] <= 'Z')
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: IS_CHAR_UC:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    PUSH(R1);
11:    MOV(R1, FPARG(0));
12:    PUSH(IMM('Z'));
13:    PUSH(IMM('A'));
14:    PUSH(R1);
15:    CALL(CHAR_IN_RANGE);
16:    POP(R1);
17:    POP(R1);
18:    POP(R1);
19:    POP(R1);
20:    POP(FP);
21:    RETURN;
```

```
1: /* char/is_char_white_space.asm
2:  * Returns 1 if argument is a whitespace char, 0 otherwise.
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: IS_CHAR_WHITE_SPACE:
8:     MOV(R0, STARG(0));
9:     CMP(R0, IMM(' '));
10:    JUMP_LE(L_IS_CHAR_WHITE_SPACE_T);
11:    MOV(R0, IMM(0));
12:    RETURN;
13: L_IS_CHAR_WHITE_SPACE_T:
14:    MOV(R0, IMM(1));
15:    RETURN;
```

```
1: /* is_even.asm
2:  * Tests whether its argument is even
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: IS_EVEN:
8:     MOV(R0, STARG(0));
9:     AND(R0, IMM(1));
10:    CMP(R0, IMM(0));
11:    JUMP_EQ(L_IS_EVEN_T);
12:    MOV(R0, IMM(0));
13:    RETURN;
14: L_IS_EVEN_T:
15:    MOV(R0, IMM(1));
16:    RETURN;
```



```
1: /* is_negative.asm
2:  * Tests whether its argument is negative
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: IS_NEGATIVE:
8:     CMP(STARG(0), IMM(0));
9:     JUMP_LT(L_IS_NEGATIVE_T);
10:    MOV(R0, IMM(0));
11:    RETURN;
12: L_IS_NEGATIVE_T:
13:    MOV(R0, IMM(1));
14:    RETURN;
```

```
1: /* is_odd.asm
2:  * Tests whether its argument is odd
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: IS_ODD:
8:     MOV(R0, STARG(0));
9:     AND(R0, IMM(1));
10:    CMP(R0, IMM(0));
11:    JUMP_EQ(L_IS_ODD_F);
12:    MOV(R0, IMM(1));
13:    RETURN;
14: L_IS_ODD_F:
15:    MOV(R0, IMM(0));
16:    RETURN;
```

```
1: /* is_positive.asm
2:  * Tests whether its argument is positive
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: IS_POSITIVE:
8:     CMP(STARG(0), IMM(0));
9:     JUMP_GT(L_IS_POSITIVE_T);
10:    MOV(R0, IMM(0));
11:    RETURN;
12: L_IS_POSITIVE_T:
13:    MOV(R0, IMM(1));
14:    RETURN;
```

```
1: /* scheme/is_sob_bool.asm
2:  * Take pointers to a Scheme object, and places in R0 either 0 or 1
3:  * (long, not Scheme integer objects or Scheme boolean objects),
4:  * depending on whether the argument is Boolean.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: IS_SOB_BOOL:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  MOV(R0, FPARG(0));
13:  CMP(IND(R0), T_BOOL);
14:  JUMP_EQ(L_IS_SOB_BOOL_TRUE);
15:  MOV(R0, IMM(0));
16:  JUMP(L_IS_SOB_BOOL_EXIT);
17: L_IS_SOB_BOOL_TRUE:
18:  MOV(R0, IMM(1));
19: L_IS_SOB_BOOL_EXIT:
20:  POP(FP);
21:  RETURN;
```

```
1: /* scheme/is_sob_char.asm
2:  * Take pointers to a Scheme object, and places in R0 either 0 or 1
3:  * (long, not Scheme integer objects or Scheme boolean objects),
4:  * depending on whether the argument is char.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: IS_SOB_CHAR:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  MOV(R0, FPARG(0));
13:  CMP(IND(R0), T_CHAR);
14:  JUMP_EQ(L_IS_SOB_CHAR_TRUE);
15:  MOV(R0, IMM(0));
16:  JUMP(L_IS_SOB_CHAR_EXIT);
17: L_IS_SOB_CHAR_TRUE:
18:  MOV(R0, IMM(1));
19: L_IS_SOB_CHAR_EXIT:
20:  POP(FP);
21:  RETURN;
22:
```

```
1: /* scheme/is_sob_closure.asm
2:  * Take pointers to a Scheme object, and places in R0 either 0 or 1
3:  * (long, not Scheme integer objects or Scheme boolean objects),
4:  * depending on whether the argument is a closure.
5:  *
6:  * Programmer: Mayer Goldberg, 2012
7:  */
8:
9: IS_SOB_CLOSURE:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  MOV(R0, FPARG(0));
13:  CMP(IND(R0), T_CLOSURE);
14:  JUMP_EQ(L_IS_SOB_CLOSURE_TRUE);
15:  MOV(R0, IMM(0));
16:  JUMP(L_IS_SOB_CLOSURE_EXIT);
17: L_IS_SOB_CLOSURE_TRUE:
18:  MOV(R0, IMM(1));
19: L_IS_SOB_CLOSURE_EXIT:
20:  POP(FP);
21:  RETURN;
22:
23:
```

```
1: /* scheme/is_sob_integer.asm
2:  * Take pointers to a Scheme object, and places in R0 either 0 or 1
3:  * (long, not Scheme integer objects or Scheme boolean objects),
4:  * depending on whether the argument is integer.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: IS_SOB_INTEGER:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  MOV(R0, FPARG(0));
13:  CMP(IND(R0), T_INTEGER);
14:  JUMP_EQ(L_IS_SOB_INTEGER_TRUE);
15:  MOV(R0, IMM(0));
16:  JUMP(L_IS_SOB_INTEGER_EXIT);
17: L_IS_SOB_INTEGER_TRUE:
18:  MOV(R0, IMM(1));
19: L_IS_SOB_INTEGER_EXIT:
20:  POP(FP);
21:  RETURN;
22:
23:
```

```
1: /* scheme/is_sob_nil.asm
2:  * Take pointers to a Scheme object, and places in R0 either 0 or 1
3:  * (long, not Scheme integer objects or Scheme boolean objects),
4:  * depending on whether the argument is nil.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: IS_SOB_NIL:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  MOV(R0, FPARG(0));
13:  CMP(IND(R0), T_NIL);
14:  JUMP_EQ(L_IS_SOB_NIL_TRUE);
15:  MOV(R0, IMM(0));
16:  JUMP(L_IS_SOB_NIL_EXIT);
17: L_IS_SOB_NIL_TRUE:
18:  MOV(R0, IMM(1));
19: L_IS_SOB_NIL_EXIT:
20:  POP(FP);
21:  RETURN;
22:
23:
24:
25:
```



```
1: /* scheme/is_sob_pair.asm
2:  * Take pointers to a Scheme object, and places in R0 either 0 or 1
3:  * (long, not Scheme integer objects or Scheme boolean objects),
4:  * depending on whether the argument is a pair.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: IS_SOB_PAIR:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  MOV(R0, FPARG(0));
13:  CMP(IND(R0), T_PAIR);
14:  JUMP_EQ(L_IS_SOB_PAIR_TRUE);
15:  MOV(R0, IMM(0));
16:  JUMP(L_IS_SOB_PAIR_EXIT);
17: L_IS_SOB_PAIR_TRUE:
18:  MOV(R0, IMM(1));
19: L_IS_SOB_PAIR_EXIT:
20:  POP(FP);
21:  RETURN;
22:
23:
24:
```

```
1: /* scheme/is_sob_void.asm
2:  * Take pointers to a Scheme object, and places in R0 either 0 or 1
3:  * (long, not Scheme integer objects or Scheme boolean objects),
4:  * depending on whether the argument is void.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: IS_SOB_VOID:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  MOV(R0, FPARG(0));
13:  CMP(IND(R0), T_VOID);
14:  JUMP_EQ(L_IS_SOB_VOID_TRUE);
15:  MOV(R0, IMM(0));
16:  JUMP(L_IS_SOB_VOID_EXIT);
17: L_IS_SOB_VOID_TRUE:
18:  MOV(R0, IMM(1));
19: L_IS_SOB_VOID_EXIT:
20:  POP(FP);
21:  RETURN;
22:
23:
24:
```

```
1: /* is_zero.asm
2:  * R0 <- 1, 0 -- depending on whether ARG[0] is zero
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: IS_ZERO:
8:     CMP(STARG(0), IMM(0));
9:     JUMP_NE(L_IS_ZERO_F);
10:    MOV(R0, IMM(1));
11:    RETURN;
12: L_IS_ZERO_F:
13:    MOV(R0, IMM(0));
14:    RETURN;
```

```
1: /* left_string.asm
2:  * Copy the left N chars in a string:
3:  * R0 = dest <- left_string(dest, src, N))
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: LEFT_STRING:
9:     MOV(R0, STARG(0));
10:    MOV(R1, R0);
11:    MOV(R2, STARG(1));
12:    MOV(R3, STARG(2));
13: L_LEFT_STRING_1:
14:    CMP(R3, IMM('\0'));
15:    JUMP_EQ(L_LEFT_STRING_2);
16:    MOV(IND(R1), IND(R2));
17:    INCR(R1);
18:    INCR(R2);
19:    DECR(R3);
20:    JUMP(L_LEFT_STRING_1);
21: L_LEFT_STRING_2:
22:    MOV(IND(R1), IMM('\0'));
23:    RETURN;
24:
```

```
1: /* scheme/make_sob_bool.asm
2:  * Takes 0 or 1 as an argument, and places in R0 either #f or #t
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: MAKE_SOB_BOOL:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    PUSH(IMM(2));
11:    CALL(MALLOC);
12:    DROP(1);
13:    MOV(IND(R0), T_BOOL);
14:    MOV(INDD(R0,1), FPARG(0));
15:    POP(FP);
16:    RETURN;
17:
```

```
1: /* scheme/make_sob_char.asm
2:  * Takes an integer 0 <= n < 256 as an argument, and places
3:  * in R0 the corresponding character object
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: MAKE_SOB_CHAR:
9:     PUSH(FP);
10:    MOV(FP, SP);
11:    PUSH(IMM(2));
12:    CALL(MALLOC);
13:    DROP(1);
14:    MOV(IND(R0), T_CHAR);
15:    MOV(INDD(R0, 1), FPARG(0));
16:    POP(FP);
17:    RETURN;
18:
```

```
1: /* scheme/make_sob_closure.asm
2:  * Take pointers to an environment and some code (address of a label),
3:  * and place the corresponding Scheme closure in R0
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: MAKE_SOB_CLOSURE:
9:     PUSH(FP);
10:    MOV(FP, SP);
11:    PUSH(IMM(3));
12:    CALL(MALLOC);
13:    DROP(1);
14:    MOV(IND(R0), IMM(T_CLOSURE));
15:    MOV(INDD(R0, 1), FPARG(0));
16:    MOV(INDD(R0, 2), FPARG(1));
17:    POP(FP);
18:    RETURN;
19:
```

```
1: /* scheme/make_sob_integer.asm
2:  * Takes an integer, and place the corresponding Scheme object in R0
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: MAKE_SOB_INTEGER:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    PUSH(IMM(2));
11:    CALL(MALLOC);
12:    DROP(1);
13:    MOV(IND(R0), T_INTEGER);
14:    MOV(INDD(R0, 1), FPARG(0));
15:    POP(FP);
16:    RETURN;
```



```
1: /* scheme/make_sob_nil.asm
2:  * Create a nil -- () object, and place it in R0
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: MAKE_SOB_NIL:
8:     PUSH( IMM(1) );
9:     CALL( MALLOC );
10:    DROP( 1 );
11:    MOV( IND( R0 ), T_NIL );
12:    RETURN;
```

```
1: /* scheme/make_sob_pair.asm
2:  * Take pointers to two sexprs, and place the corresponding
3:  * Scheme pair in R0
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: MAKE_SOB_PAIR:
9:     PUSH(FP);
10:    MOV(FP, SP);
11:    PUSH(IMM(3));
12:    CALL(MALLOC);
13:    DROP(1);
14:    MOV(IND(R0), T_PAIR);
15:    MOV(INDD(R0, 1), FPARG(0));
16:    MOV(INDD(R0, 2), FPARG(1));
17:    POP(FP);
18:    RETURN;
19:
```

```
1: /* scheme/make_sob_string.asm
2:  * Takes CHAR1, ..., CHARn, n, on the stack. Places in R0 the address
3:  * of a newly-allocated pointer to a Scheme string.
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: MAKE_SOB_STRING:
9:     PUSH(FP);
10:    MOV(FP, SP);
11:    PUSH(R1);
12:    PUSH(R2);
13:    PUSH(R3);
14:    MOV(R0, FPARG(0));
15:    ADD(R0, IMM(2));
16:    PUSH(R0);
17:    CALL(MALLOC);
18:    DROP(1);
19:    MOV(IND(R0), IMM(T_STRING));
20:    MOV(INDD(R0, 1), FPARG(0));
21:    MOV(R1, FP);
22:    MOV(R2, FPARG(0));
23:    ADD(R2, IMM(3));
24:    SUB(R1, R2);
25:    MOV(R2, R0);
26:    ADD(R2, IMM(2));
27:    MOV(R3, FPARG(0));
28: L_MSS_LOOP:
29:    CMP(R3, IMM(0));
30:    JUMP_EQ(L_MSS_EXIT);
31:    MOV(IND(R2), STACK(R1));
32:    INCR(R1);
33:    INCR(R2);
34:    DECR(R3);
35:    JUMP(L_MSS_LOOP);
36: L_MSS_EXIT:
37:    POP(R3);
38:    POP(R2);
39:    POP(R1);
40:    POP(FP);
41:    RETURN;
42:
```

```
1: /* scheme/make_sob_vector.asm
2:  * Takes V1, ..., Vn, n, on the stack. Places in R0 the address
3:  * of a newly-allocated pointer to a Scheme vector.
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: MAKE_SOB_VECTOR:
9:     PUSH(FP);
10:    MOV(FP, SP);
11:    PUSH(R1);
12:    PUSH(R2);
13:    PUSH(R3);
14:    MOV(R0, FPARG(0));
15:    ADD(R0, IMM(2));
16:    PUSH(R0);
17:    CALL(MALLOC);
18:    DROP(1);
19:    MOV(IND(R0), IMM(T_VECTOR));
20:    MOV(INDD(R0, 1), FPARG(0));
21:    MOV(R1, FP);
22:    MOV(R2, FPARG(0));
23:    ADD(R2, IMM(3));
24:    SUB(R1, R2);
25:    MOV(R2, R0);
26:    ADD(R2, IMM(2));
27:    MOV(R3, FPARG(0));
28: L_MSV_LOOP:
29:    CMP(R3, IMM(0));
30:    JUMP_EQ(L_MSV_EXIT);
31:    MOV(IND(R2), STACK(R1));
32:    INCR(R1);
33:    INCR(R2);
34:    DECR(R3);
35:    JUMP(L_MSV_LOOP);
36: L_MSV_EXIT:
37:    POP(R3);
38:    POP(R2);
39:    POP(R1);
40:    POP(FP);
41:    RETURN;
42:
43:
```

```
1: /* scheme/make_sob_void.asm
2:  * Create a #<void> object, and place it in R0
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: MAKE_SOB_VOID:
8:     PUSH( IMM(1) );
9:     CALL( MALLOC );
10:    DROP( 1 );
11:    MOV( IND( R0 ), T_VOID );
12:    RETURN;
```

```
1: /* system/malloc.asm
2:  * A stub for a more intelligent memory allocation code
3:  * that shall be re-written later...
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: MALLOC:
9:     PUSH(FP);
10:    MOV(FP, SP);
11:    PUSH(R1);
12:    MOV(R1, FPARG(0));
13:    MOV(R0, ADDR(0));
14:    ADD(ADDR(0), R1);
15:    POP(R1);
16:    POP(FP);
17:    RETURN;
18:
```

```
1: /* mid_string.asm
2:  * Copy the middle N chars in a string, from position Pos:
3:  * R0 = dest <- mid_string(dest, src, Pos, N))
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: MID_STRING:
9:     MOV(R3, STARG(3));
10:    MOV(R2, STARG(1));
11:    ADD(R2, STARG(2));
12:    MOV(R1, STARG(0));
13:    PUSH(R3);
14:    PUSH(R2);
15:    PUSH(R1);
16:    CALL(LEFT_STRING);
17:    POP(R1);
18:    POP(R1);
19:    POP(R1);
20:    RETURN;
```

```
1: /* io/newline.asm
2:  * Print a newline character to stdout
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: NEWLINE:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    PUSH(R1);
11:    PUSH(IMM('\n'));
12:    CALL(PUTCHAR);
13:    POP(R1);
14:    POP(R1);
15:    POP(FP);
16:    RETURN;
```



```
1: /* number_to_string.asm
2:  * Takes a pointer to a dest string and an integer, and
3:  * writes in the destination the string representation
4:  * of the number
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: NUMBER_TO_STRING:
10:  MOV(R1, STARG(0));
11:  MOV(R2, R1);
12:  MOV(R3, STARG(1));
13:  CMP(R3, IMM(0));
14:  JUMP_EQ(L_NTS_0);
15:  JUMP_LT(L_NTS_N);
16:  PUSH(R3);
17: L_NTS_3:
18:  CALL(L_NTS_1);
19:  POP(R2);
20:  MOV(R0, R1);
21:  RETURN;
22: L_NTS_1:
23:  PUSH(FP);
24:  MOV(FP, SP);
25:  MOV(R4, FPARG(0));
26:  CMP(R4, IMM(0));
27:  JUMP_EQ(L_NTS_2);
28:  MOV(R5, R4);
29:  DIV(R4, 10);
30:  REM(R5, 10);
31:  PUSH(R5);
32:  PUSH(R4);
33:  CALL(L_NTS_1);
34:  POP(R4);
35:  POP(R5);
36:  ADD(R5, IMM('0'));
37:  MOV(IND(R0), R5);
38:  INCR(R0);
39:  POP(FP);
40:  RETURN;
41: L_NTS_2:
42:  MOV(R0, R2);
43:  POP(FP);
44:  RETURN;
45: L_NTS_0:
46:  MOV(IND(R2), IMM('0'));
47:  INCR(R2);
48:  MOV(IND(R2), IMM('\0'));
49:  RETURN;
50: L_NTS_N:
51:  MOV(IND(R2), IMM('-'));
52:  INCR(R2);
53:  MOV(R4, IMM(0));
54:  SUB(R4, R3);
55:  PUSH(R4);
56:  JUMP(L_NTS_3);
```

```
1: /* power.asm
2:  * Computes the integer power: R0 <- ARG[0] ^ ARG[1]
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: POWER:
8:     MOV(R1, STARG(0)); /* A */
9:     MOV(R2, STARG(1)); /* B */
10:    CMP(R2, IMM(0));
11:    JUMP_EQ(L_POWER_A_0);
12:    MOV(R3, R2);
13:    AND(R3, IMM(1));
14:    CMP(R3, IMM(0));
15:    JUMP_EQ(L_POWER_A_EVEN);
16:    SHR(R2, IMM(1));
17:    PUSH(R2);
18:    PUSH(R1);
19:    CALL_L(POWER);
20:    POP(R1);
21:    POP(R1);
22:    MUL(R0, R0);
23:    MUL(R0, STARG(0));
24:    RETURN;
25:
26: L_POWER_A_0:
27:     MOV(R0, IMM(1));
28:     RETURN;
29:
30: L_POWER_A_EVEN:
31:     SHR(R2, IMM(1));
32:     PUSH(R2);
33:     PUSH(R1);
34:     CALL_L(POWER);
35:     POP(R1);
36:     POP(R1);
37:     MUL(R0, R0);
38:     RETURN;
```

```
1: /* io/putchar.asm
2:  * Print a char to stdout
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: PUTCHAR:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    MOV(R0, FPARG(0));
11:    OUT(IMM(2), R0);
12:    POP(FP);
13:    RETURN;
```

```
1: /* io/readline.asm
2:  * Read chars until the end of the line or the end of the file,
3:  * and return a pointer to a null-terminated string. This
4:  * routine calls MALLOC to allocate its own memory for the
5:  * string.
6:  *
7:  * Programmer: Mayer Goldberg, 2010
8:  */
9:
10: READLINE:
11:     PUSH(FP);
12:     MOV(FP, SP);
13:     PUSH(R1);
14:     PUSH(R2);
15:     PUSH(IMM(''));
16:     PUSH(IMM(0));
17:     CALL(READLINE_LOOP);
18:     POP(R1);
19:     POP(R1);
20:     MOV(IND(R0), R1);
21:     POP(R2);
22:     POP(R1);
23:     POP(FP);
24:     RETURN;
25: READLINE_LOOP:
26:     IN(R1, IMM(1)); /* read a char from stdin */
27:     CMP(R1, IMM('\n'));
28:     JUMP_EQ(READLINE_DONE);
29:     CMP(R1, IMM(-1));
30:     JUMP_EQ(READLINE_DONE);
31:     MOV(STARG(1), R1);
32:     MOV(R1, STARG(0));
33:     INCR(R1);
34:     PUSH(IMM(''));
35:     PUSH(R1);
36:     CALL(READLINE_LOOP);
37:     POP(R1); /* INDEX */
38:     POP(R2); /* CH */
39:     ADD(R1, R0);
40:     MOV(IND(R1), R2);
41:     RETURN;
42: READLINE_DONE:
43:     MOV(R1, STARG(0)); /* NUMBER OF CHARS */
44:     INCR(R1); /* MAKE ROOM FOR '\0' */
45:     PUSH(R1);
46:     CALL(MALLOC);
47:     POP(R1);
48:     MOV(STARG(1), IMM('\0'));
49:     RETURN;
50:
```

```
1: /* right_string.asm
2:  * Copy the right N chars in a string:
3:  * R0 = dest <- right_string(dest, src, N))
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: RIGHT_STRING:
9:     MOV(R1, STARG(1));
10:    PUSH(R1);
11:    CALL(STRLEN);
12:    POP(R1);
13:    MOV(R3, STARG(2));
14:    SUB(R0, R3);
15:    MOV(R2, STARG(1));
16:    MOV(R1, STARG(0));
17:    PUSH(R3);
18:    PUSH(R0);
19:    PUSH(R2);
20:    PUSH(R1);
21:    CALL(MID_STRING);
22:    POP(R1);
23:    POP(R1);
24:    POP(R1);
25:    POP(R1);
26:    RETURN;
```

```
1: /* signum.asm
2:  * Compute the signum function on its arg:
3:  * R0 <- 1, 0, -1 -- depending on the sign of ARG[0]
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: SIGNUM:
9:     CMP(STARG(0), IMM(0));
10:    JUMP_GT(L_SIGNUM_P);
11:    JUMP_LT(L_SIGNUM_N);
12:    MOV(R0, IMM(0));
13:    RETURN;
14: L_SIGNUM_P:
15:    MOV(R0, IMM(1));
16:    RETURN;
17: L_SIGNUM_N:
18:    MOV(R0, IMM(-1));
19:    RETURN;
```

```
1: /* strcat.asm
2:  * Equivalent to strcat in C:
3:  * R0 = dest <- strcat(dest, src)
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: STRCAT:
9:     MOV(R0, STARG(0));
10:    MOV(R1, R0);
11:    MOV(R2, STARG(1));
12: L_STRCAT_1:
13:    CMP(IND(R1), IMM('\0'));
14:    JUMP_EQ(L_STRCAT_2);
15:    INCR(R1);
16:    JUMP(L_STRCAT_1);
17: L_STRCAT_2:
18:    CMP(IND(R2), IMM('\0'));
19:    JUMP_EQ(L_STRCAT_3);
20:    MOV(IND(R1), IND(R2));
21:    INCR(R1);
22:    INCR(R2);
23:    JUMP(L_STRCAT_2);
24: L_STRCAT_3:
25:    MOV(IND(R1), IMM('\0'));
26:    RETURN;
```

```
1: /* strcpy.asm
2:  * Equivalent to strcat in C:
3:  * R0 = dest <- strcpy(dest, src)
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: STRCPY:
9:     MOV(R0, STARG(0));
10:    MOV(R1, R0);
11:    MOV(R2, STARG(1));
12: L_STRCPY_1:
13:    CMP(IND(R2), IMM('\0'));
14:    JUMP_EQ(L_STRCPY_2);
15:    MOV(IND(R1), IND(R2));
16:    INCR(R1);
17:    INCR(R2);
18:    JUMP(L_STRCPY_1);
19: L_STRCPY_2:
20:    MOV(IND(R1), IMM('\0'));
21:    RETURN;
```



```
1: /* string_reverse.asm
2:  * Takes a pointer to a null-terminated string,
3:  * and reverses it in place.
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: STRING_REVERSE:
9:     MOV(R1, STARG(0));
10:    PUSH(R1);
11:    CALL(STRLEN);
12:    POP(R2);
13:    DECR(R0);
14:    ADD(R0, STARG(0));
15:    MOV(R1, STARG(0));
16: L_STRING_REVERSE_0:
17:    CMP(R1, R0);
18:    JUMP_GE(L_STRING_REVERSE_1);
19:    MOV(R2, IND(R1));
20:    MOV(IND(R1), IND(R0));
21:    MOV(IND(R0), R2);
22:    DECR(R0);
23:    INCR(R1);
24:    JUMP(L_STRING_REVERSE_0);
25: L_STRING_REVERSE_1:
26:    MOV(R0, STARG(0));
27:    RETURN;
```

```
1: /* string_to_lc.asm
2:  * Convert the string to lowercase
3:  * R0 = dest <- string_to_lc(dest)
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: STRING_TO_LC:
9:     MOV(R1, STARG(0));
10:    L_STR_TO_LC_1:
11:        CMP(IND(R1), IMM('\0'));
12:        JUMP_EQ(L_STR_TO_LC_2);
13:        PUSH(R1);
14:        PUSH(IND(R1));
15:        CALL(CHAR_TO_LC);
16:        POP(R1);
17:        POP(R1);
18:        MOV(IND(R1), R0);
19:        INCR(R1);
20:        JUMP(L_STR_TO_LC_1);
21:    L_STR_TO_LC_2:
22:        MOV(R0, STARG(0));
23:        RETURN;
```

```
1: /* string_to_number.asm
2:  * Converts a source string to a number. Similar to
3:  * atoi in C.
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: STRING_TO_NUMBER:
9:     MOV(R1, STARG(0));
10:    MOV(R0, IMM(0));
11: L_STRING_TO_NUMBER_0:
12:    CMP(IND(R1), IMM('\0'));
13:    JUMP_EQ(L_STRING_TO_NUMBER_1);
14:    MOV(R2, IND(R1));
15:    SUB(R2, IMM('0'));
16:    MUL(R0, IMM(10));
17:    ADD(R0, R2);
18:    INCR(R1);
19:    JUMP(L_STRING_TO_NUMBER_0);
20: L_STRING_TO_NUMBER_1:
21:    RETURN;
```

```
1: /* string_to_uc.asm
2:  * Convert the string to uppercase
3:  * R0 = dest <- string_to_uc(dest)
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: STRING_TO_UC:
9:     MOV(R1, STARG(0));
10:    L_STR_TO_UC_1:
11:        CMP(IND(R1), IMM('\0'));
12:        JUMP_EQ(L_STR_TO_UC_2);
13:        PUSH(R1);
14:        PUSH(IND(R1));
15:        CALL(CHAR_TO_UC);
16:        POP(R1);
17:        POP(R1);
18:        MOV(IND(R1), R0);
19:        INCR(R1);
20:        JUMP(L_STR_TO_UC_1);
21:    L_STR_TO_UC_2:
22:        MOV(R0, STARG(0));
23:        RETURN;
24:
```

```
1: /* strlen.asm
2:  * Takes a pointer to a null-terminated string,
3:  * and returns its length.
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: STRLEN:
9:     MOV(R1, STARG(0));
10:    MOV(R0, IMM(0));
11: L_STRLEN_LOOP:
12:    CMP(IND(R1), IMM('\0'));
13:    JUMP_EQ(L_STRLEN_END);
14:    INCR(R1);
15:    INCR(R0);
16:    JUMP(L_STRLEN_LOOP);
17: L_STRLEN_END:
18:    RETURN;
```

```
1: /* io/tab.asm
2:  * Print a tab character to stdout
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: TAB:
8:     PUSH( IMM( '\t' ) );
9:     CALL( PUTCHAR );
10:    POP( R0 );
11:    RETURN;
```

```
1: /* io/write.asm
2:  * Takes a pointer to a null-terminated string,
3:  * and prints it to STDOUT.
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: WRITE:
9:     PUSH(FP);
10:    MOV(FP, SP);
11:    PUSH(R1);
12:    PUSH(R2);
13:    MOV(R1, FPARG(0));
14: L_WRITE0:
15:    MOV(R2, IND(R1));
16:    CMP(R2, IMM('\0'));
17:    JUMP_EQ(L_WRITE_END);
18:    PUSH(R2);
19:    CALLL(PUTCHAR);
20:    POP(R2);
21:    INCR(R1);
22:    JUMP(L_WRITE0);
23: L_WRITE_END:
24:    POP(R2);
25:    POP(R1);
26:    POP(FP);
27:    RETURN;
28:
```

```
1: /* io/write_integer.asm
2:  * Print a decimal representation of an integer argument to stdout
3:  *
4:  * Programmer: Mayer Goldberg, 2010
5:  */
6:
7: WRITE_INTEGER:
8:     PUSH(FP);
9:     MOV(FP, SP);
10:    PUSH(R1);
11:    MOV(R0, FPARG(0));
12:    CMP(R0, IMM(0));
13:    JUMP_EQ(L_WI_0);
14:    JUMP_LT(L_WI_N);
15:    PUSH(R0);
16:    CALL(L_WI_LOOP);
17:    POP(R1);
18:    JUMP(L_WI_EX);
19: L_WI_LOOP:
20:    PUSH(FP);
21:    MOV(FP, SP);
22:    MOV(R0, FPARG(0));
23:    CMP(R0, IMM(0));
24:    JUMP_EQ(L_WI_LOOP_END);
25:    REM(R0, IMM(10));
26:    PUSH(R0);
27:    MOV(R0, FPARG(0));
28:    DIV(R0, IMM(10));
29:    PUSH(R0);
30:    CALL(L_WI_LOOP);
31:    POP(R0);
32:    POP(R0);
33:    ADD(R0, IMM('0'));
34:    PUSH(R0);
35:    CALL(PUTCHAR);
36:    POP(R0);
37: L_WI_LOOP_END:
38:    POP(FP);
39:    RETURN;
40: L_WI_N:
41:    PUSH(IMM('-'));
42:    CALL(PUTCHAR);
43:    POP(R1);
44:    MOV(R0, FPARG(0));
45:    MOV(R1, IMM(0));
46:    SUB(R1, R0);
47:    PUSH(R1);
48:    CALL(WRITE_INTEGER);
49:    POP(R1);
50:    JUMP(L_WI_EX);
51: L_WI_0:
52:    PUSH(IMM('0'));
53:    CALL(PUTCHAR);
54:    POP(R0);
55:    JUMP(L_WI_EX);
56: L_WI_EX:
57:    POP(R1);
58:    POP(FP);
59:    RETURN;
```



```
1: /* scheme/write_sob.asm
2:  * Take a pointer to a Scheme object, and
3:  * prints (to stdout) the character representation
4:  * of that object.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: WRITE_SOB:
10:  MOV(R0, STARG(0));
11:  MOV(R0, IND(R0));
12:  CMP(R0, IMM(T_VOID));
13:  JUMP_EQ(WRITE_SOB_VOID);
14:  CMP(R0, IMM(T_NIL));
15:  JUMP_EQ(WRITE_SOB_NIL);
16:  CMP(R0, IMM(T_BOOL));
17:  JUMP_EQ(WRITE_SOB_BOOL);
18:  CMP(R0, IMM(T_CHAR));
19:  JUMP_EQ(WRITE_SOB_CHAR);
20:  CMP(R0, IMM(T_INTEGER));
21:  JUMP_EQ(WRITE_SOB_INTEGER);
22:  CMP(R0, IMM(T_STRING));
23:  JUMP_EQ(WRITE_SOB_STRING);
24:  CMP(R0, IMM(T_SYMBOL));
25:  JUMP_EQ(WRITE_SOB_SYMBOL);
26:  CMP(R0, IMM(T_PAIR));
27:  JUMP_EQ(WRITE_SOB_PAIR);
28:  CMP(R0, IMM(T_VECTOR));
29:  JUMP_EQ(WRITE_SOB_VECTOR);
30:  CMP(R0, IMM(T_CLOSURE));
31:  JUMP_EQ(WRITE_SOB_CLOSURE);
32:  PUSH(R0);
33:  PUSH(IMM('\n'));
34:  CALL(PUTCHAR);
35:  PUSH(IMM('f'));
36:  CALL(PUTCHAR);
37:  PUSH(IMM('a'));
38:  CALL(PUTCHAR);
39:  PUSH(IMM('t'));
40:  CALL(PUTCHAR);
41:  PUSH(IMM('a'));
42:  CALL(PUTCHAR);
43:  PUSH(IMM('l'));
44:  CALL(PUTCHAR);
45:  PUSH(IMM(' '));
46:  CALL(PUTCHAR);
47:  PUSH(IMM('e'));
48:  CALL(PUTCHAR);
49:  PUSH(IMM('r'));
50:  CALL(PUTCHAR);
51:  PUSH(IMM('r'));
52:  CALL(PUTCHAR);
53:  PUSH(IMM('o'));
54:  CALL(PUTCHAR);
55:  PUSH(IMM('r'));
56:  CALL(PUTCHAR);
57:  PUSH(IMM(':'));
58:  CALL(PUTCHAR);
59:  PUSH(IMM(' '));
60:  CALL(PUTCHAR);
61:  PUSH(IMM('C'));
62:  CALL(PUTCHAR);
63:  PUSH(IMM('o'));
64:  CALL(PUTCHAR);
```

```
65:  PUSH( IMM( 'r' ) );
66:  CALL( PUTCHAR );
67:  PUSH( IMM( 'r' ) );
68:  CALL( PUTCHAR );
69:  PUSH( IMM( 'u' ) );
70:  CALL( PUTCHAR );
71:  PUSH( IMM( 'p' ) );
72:  CALL( PUTCHAR );
73:  PUSH( IMM( 't' ) );
74:  CALL( PUTCHAR );
75:  PUSH( IMM( ' ' ) );
76:  CALL( PUTCHAR );
77:  PUSH( IMM( 's' ) );
78:  CALL( PUTCHAR );
79:  PUSH( IMM( 'e' ) );
80:  CALL( PUTCHAR );
81:  PUSH( IMM( 'x' ) );
82:  CALL( PUTCHAR );
83:  PUSH( IMM( 'p' ) );
84:  CALL( PUTCHAR );
85:  PUSH( IMM( 'r' ) );
86:  CALL( PUTCHAR );
87:  PUSH( IMM( ' ' ) );
88:  CALL( PUTCHAR );
89:  PUSH( IMM( 't' ) );
90:  CALL( PUTCHAR );
91:  PUSH( IMM( 'y' ) );
92:  CALL( PUTCHAR );
93:  PUSH( IMM( 'p' ) );
94:  CALL( PUTCHAR );
95:  PUSH( IMM( 'e' ) );
96:  CALL( PUTCHAR );
97:  PUSH( IMM( ':' ) );
98:  CALL( PUTCHAR );
99:  PUSH( IMM( ' ' ) );
100: CALL( PUTCHAR );
101: DROP( 34 );
102: CALL( WRITE_INTEGER );
103: DROP( 1 );
104: CALL( NEWLINE );
105: HALT;
```

```
1: /* scheme/write_sob_bool.asm
2:  * Take a pointer to a Scheme Boolean object, and
3:  * prints (to stdout) the character representation
4:  * of that object.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: WRITE_SOB_BOOL:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  MOV(R0, FPARG(0));
13:  MOV(R0, INDD(R0, 1));
14:  CMP(R0, IMM(0));
15:  JUMP_EQ(L_WRITE_SOB_BOOL_FALSE);
16:  PUSH(IMM('#'));
17:  CALL(PUTCHAR);
18:  PUSH(IMM('t'));
19:  CALL(PUTCHAR);
20:  DROP(2);
21:  JUMP(L_WRITE_SOB_BOOL_EXIT);
22: L_WRITE_SOB_BOOL_FALSE:
23:  PUSH(IMM('#'));
24:  CALL(PUTCHAR);
25:  PUSH(IMM('f'));
26:  CALL(PUTCHAR);
27:  DROP(2);
28: L_WRITE_SOB_BOOL_EXIT:
29:  POP(FP);
30:  RETURN;
31:
```

```
1: /* scheme/write_sob_char.asm
2:  * Take a pointer to a Scheme char object, and
3:  * prints (to stdout) the character representation
4:  * of that object.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: WRITE_SOB_CHAR:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  PUSH(R1);
13:  PUSH( IMM( '#' ) );
14:  CALL( PUTCHAR );
15:  PUSH( IMM( '\\ ' ) );
16:  CALL( PUTCHAR );
17:  DROP( 2 );
18:  MOV( R0, FPARG( 0 ) );
19:  MOV( R0, INDD( R0, 1 ) );
20:  CMP( R0, IMM( '\\n ' ) );
21:  JUMP_EQ( L_WRITE_SOB_CHAR_NEWLINE );
22:  CMP( R0, IMM( '\\t ' ) );
23:  JUMP_EQ( L_WRITE_SOB_CHAR_TAB );
24:  CMP( R0, IMM( '\\f ' ) );
25:  JUMP_EQ( L_WRITE_SOB_CHAR_PAGE );
26:  CMP( R0, IMM( '\\r ' ) );
27:  JUMP_EQ( L_WRITE_SOB_CHAR_RETURN );
28:  CMP( R0, IMM( ' ' ) );
29:  JUMP_EQ( L_WRITE_SOB_CHAR_SPACE );
30:  JUMP_LT( L_WRITE_SOB_CHAR_OCTAL );
31:  PUSH( R0 );
32:  CALL( PUTCHAR );
33:  DROP( 1 );
34:  JUMP( L_WRITE_SOB_CHAR_EXIT );
35: L_WRITE_SOB_CHAR_OCTAL:
36:  MOV( R1, R0 );
37:  REM( R1, IMM( 8 ) );
38:  PUSH( R1 );
39:  DIV( R0, IMM( 8 ) );
40:  MOV( R1, R0 );
41:  REM( R1, IMM( 8 ) );
42:  PUSH( R1 );
43:  DIV( R0, IMM( 8 ) );
44:  REM( R0, IMM( 8 ) );
45:  PUSH( R0 );
46:  CALL( WRITE_INTEGER );
47:  DROP( 1 );
48:  CALL( WRITE_INTEGER );
49:  DROP( 1 );
50:  CALL( WRITE_INTEGER );
51:  DROP( 1 );
52:  JUMP( L_WRITE_SOB_CHAR_EXIT );
53: L_WRITE_SOB_CHAR_RETURN:
54:  PUSH( IMM( 'r ' ) );
55:  CALL( PUTCHAR );
56:  PUSH( IMM( 'e ' ) );
57:  CALL( PUTCHAR );
58:  PUSH( IMM( 't ' ) );
59:  CALL( PUTCHAR );
60:  PUSH( IMM( 'u ' ) );
61:  CALL( PUTCHAR );
62:  PUSH( IMM( 'r ' ) );
63:  CALL( PUTCHAR );
64:  PUSH( IMM( 'n ' ) );
```

```
65:    CALL( PUTCHAR );
66:    DROP( 6 );
67:    JUMP( L_WRITE_SOB_CHAR_EXIT );
68: L_WRITE_SOB_CHAR_PAGE:
69:    PUSH( IMM( 'p' ) );
70:    CALL( PUTCHAR );
71:    PUSH( IMM( 'a' ) );
72:    CALL( PUTCHAR );
73:    PUSH( IMM( 'g' ) );
74:    CALL( PUTCHAR );
75:    PUSH( IMM( 'e' ) );
76:    CALL( PUTCHAR );
77:    DROP( 4 );
78:    JUMP( L_WRITE_SOB_CHAR_EXIT );
79: L_WRITE_SOB_CHAR_TAB:
80:    PUSH( IMM( 't' ) );
81:    CALL( PUTCHAR );
82:    PUSH( IMM( 'a' ) );
83:    CALL( PUTCHAR );
84:    PUSH( IMM( 'b' ) );
85:    CALL( PUTCHAR );
86:    DROP( 3 );
87:    JUMP( L_WRITE_SOB_CHAR_EXIT );
88: L_WRITE_SOB_CHAR_NEWLINE:
89:    PUSH( IMM( 'n' ) );
90:    CALL( PUTCHAR );
91:    PUSH( IMM( 'e' ) );
92:    CALL( PUTCHAR );
93:    PUSH( IMM( 'w' ) );
94:    CALL( PUTCHAR );
95:    PUSH( IMM( 'l' ) );
96:    CALL( PUTCHAR );
97:    PUSH( IMM( 'i' ) );
98:    CALL( PUTCHAR );
99:    PUSH( IMM( 'n' ) );
100:   CALL( PUTCHAR );
101:   PUSH( IMM( 'e' ) );
102:   CALL( PUTCHAR );
103:   DROP( 7 );
104:   JUMP( L_WRITE_SOB_CHAR_EXIT );
105: L_WRITE_SOB_CHAR_SPACE:
106:   PUSH( IMM( 's' ) );
107:   CALL( PUTCHAR );
108:   PUSH( IMM( 'p' ) );
109:   CALL( PUTCHAR );
110:   PUSH( IMM( 'a' ) );
111:   CALL( PUTCHAR );
112:   PUSH( IMM( 'c' ) );
113:   CALL( PUTCHAR );
114:   PUSH( IMM( 'e' ) );
115:   CALL( PUTCHAR );
116:   DROP( 5 );
117: L_WRITE_SOB_CHAR_EXIT:
118:   POP( R1 );
119:   POP( FP );
120:   RETURN;
121:
```

```
1: /* scheme/write_sob_closure.asm
2:  * Take a pointer to a Scheme closure object, and
3:  * prints (to stdout) the character representation
4:  * of that object.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: WRITE_SOB_CLOSURE:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  PUSH(IMM('#'));
13:  CALL(PUTCHAR);
14:  PUSH(IMM('<'));
15:  CALL(PUTCHAR);
16:  PUSH(IMM('c'));
17:  CALL(PUTCHAR);
18:  PUSH(IMM('l'));
19:  CALL(PUTCHAR);
20:  PUSH(IMM('o'));
21:  CALL(PUTCHAR);
22:  PUSH(IMM('s'));
23:  CALL(PUTCHAR);
24:  PUSH(IMM('u'));
25:  CALL(PUTCHAR);
26:  PUSH(IMM('r'));
27:  CALL(PUTCHAR);
28:  PUSH(IMM('e'));
29:  CALL(PUTCHAR);
30:  PUSH(IMM(' '));
31:  CALL(PUTCHAR);
32:  PUSH(IMM('a'));
33:  CALL(PUTCHAR);
34:  PUSH(IMM('t'));
35:  CALL(PUTCHAR);
36:  PUSH(IMM(' '));
37:  CALL(PUTCHAR);
38:  DROP(13);
39:  PUSH(FPARG(0));
40:  CALL(WRITE_INTEGER);
41:  DROP(1);
42:  PUSH(IMM(' '));
43:  CALL(PUTCHAR);
44:  PUSH(IMM('e'));
45:  CALL(PUTCHAR);
46:  PUSH(IMM('n'));
47:  CALL(PUTCHAR);
48:  PUSH(IMM('v'));
49:  CALL(PUTCHAR);
50:  PUSH(IMM(':'));
51:  CALL(PUTCHAR);
52:  PUSH(IMM(' '));
53:  CALL(PUTCHAR);
54:  DROP(6);
55:  MOV(R0, FPARG(0));
56:  PUSH(INDD(R0, 1));
57:  CALL(WRITE_INTEGER);
58:  DROP(1);
59:  PUSH(IMM(' '));
60:  CALL(PUTCHAR);
61:  PUSH(IMM('c'));
62:  CALL(PUTCHAR);
63:  PUSH(IMM('o'));
64:  CALL(PUTCHAR);
```

```
65:  PUSH( IMM( 'd' ) );
66:  CALL( PUTCHAR );
67:  PUSH( IMM( 'e' ) );
68:  CALL( PUTCHAR );
69:  PUSH( IMM( ':' ) );
70:  CALL( PUTCHAR );
71:  PUSH( IMM( ' ' ) );
72:  CALL( PUTCHAR );
73:  DROP( 7 );
74:  MOV( R0, FPARG( 0 ) );
75:  PUSH( INDD( R0, 2 ) );
76:  CALL( WRITE_INTEGER );
77:  DROP( 1 );
78:  PUSH( IMM( '>' ) );
79:  CALL( PUTCHAR );
80:  DROP( 1 );
81:  POP( FP );
82:  RETURN;
83:
```

```
1: /* scheme/write_sob_integer.asm
2:  * Take a pointer to a Scheme integer object, and
3:  * prints (to stdout) the character representation
4:  * of that object.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: WRITE_SOB_INTEGER:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  MOV(R0, FPARG(0));
13:  MOV(R0, INDD(R0, 1));
14:  PUSH(R0);
15:  CALL(WRITE_INTEGER);
16:  DROP(1);
17:  POP(FP);
18:  RETURN;
19:
```



```
1: /* scheme/write_sob_nil.asm
2:  * Take a pointer to a Scheme nil object, and
3:  * prints (to stdout) the character representation
4:  * of that object.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: WRITE_SOB_NIL:
10:  PUSH( IMM( ' ( ' ) ) );
11:  CALL( PUTCHAR );
12:  PUSH( IMM( ' ) ' ) );
13:  CALL( PUTCHAR );
14:  DROP( 2 );
15:  RETURN;
16:
```

```
1: /* scheme/write_sob_pair.asm
2:  * Take a pointer to a Scheme pair object, and
3:  * prints (to stdout) the character representation
4:  * of that object.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: WRITE_SOB_PAIR:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  MOV(R0, FPARG(0));
13:  PUSH(INDD(R0, 2));
14:  PUSH(INDD(R0, 1));
15:  PUSH(IMM('('));
16:  CALL(PUTCHAR);
17:  DROP(1);
18:  CALL(WRITE_SOB);
19:  DROP(1);
20:  PUSH(IMM(' '));
21:  CALL(PUTCHAR);
22:  PUSH(IMM('.')');
23:  CALL(PUTCHAR);
24:  PUSH(IMM(' '));
25:  CALL(PUTCHAR);
26:  DROP(3);
27:  CALL(WRITE_SOB);
28:  DROP(1);
29:  PUSH(IMM(')')));
30:  CALL(PUTCHAR);
31:  DROP(1);
32:  POP(FP);
33:  RETURN;
34:
```

```
1: /* scheme/write_sob_string.asm
2:  * Take a pointer to a Scheme string object, and
3:  * prints (to stdout) the character representation
4:  * of that object.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: WRITE_SOB_STRING:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  PUSH(R1);
13:  PUSH(R2);
14:  PUSH(R3);
15:  PUSH(IMM('\\"'));
16:  CALL(PUTCHAR);
17:  DROP(1);
18:  MOV(R0, FPARG(0));
19:  MOV(R1, INDD(R0, 1));
20:  MOV(R2, R0);
21:  ADD(R2, IMM(2));
22: L_WSS_LOOP:
23:  CMP(R1, IMM(0));
24:  JUMP_EQ(L_WSS_EXIT);
25:  CMP(IND(R2), '\n');
26:  JUMP_EQ(L_WSS_NEWLINE);
27:  CMP(IND(R2), '\t');
28:  JUMP_EQ(L_WSS_TAB);
29:  CMP(IND(R2), '\f');
30:  JUMP_EQ(L_WSS_PAGE);
31:  CMP(IND(R2), '\r');
32:  JUMP_EQ(L_WSS_RETURN);
33:  CMP(IND(R2), '\\');
34:  JUMP_EQ(L_WSS_BACKSLASH);
35:  CMP(IND(R2), '\"');
36:  JUMP_EQ(L_WSS_DQUOTE);
37:  CMP(IND(R2), ' ');
38:  JUMP_LT(L_WSS_OCT_CHAR);
39:  PUSH(IND(R2));
40:  CALL(PUTCHAR);
41:  DROP(1);
42:  JUMP(L_WSS_LOOP_CONT);
43: L_WSS_DQUOTE:
44:  PUSH(IMM('\\'));
45:  CALL(PUTCHAR);
46:  PUSH(IMM('\\"'));
47:  CALL(PUTCHAR);
48:  DROP(2);
49:  JUMP(L_WSS_LOOP_CONT);
50: L_WSS_BACKSLASH:
51:  PUSH(IMM('\\'));
52:  CALL(PUTCHAR);
53:  PUSH(IMM('\\'));
54:  CALL(PUTCHAR);
55:  DROP(2);
56:  JUMP(L_WSS_LOOP_CONT);
57: L_WSS_RETURN:
58:  PUSH(IMM('\\'));
59:  CALL(PUTCHAR);
60:  PUSH(IMM('r'));
61:  CALL(PUTCHAR);
62:  DROP(2);
63:  JUMP(L_WSS_LOOP_CONT);
64: L_WSS_PAGE:
```

```
65:  PUSH( IMM( '\\\' ) );
66:  CALL( PUTCHAR );
67:  PUSH( IMM( 'f' ) );
68:  CALL( PUTCHAR );
69:  DROP( 2 );
70:  JUMP( L_WSS_LOOP_CONT );
71:  L_WSS_TAB:
72:  PUSH( IMM( '\\\' ) );
73:  CALL( PUTCHAR );
74:  PUSH( IMM( 't' ) );
75:  CALL( PUTCHAR );
76:  DROP( 2 );
77:  JUMP( L_WSS_LOOP_CONT );
78:  L_WSS_NEWLINE:
79:  PUSH( IMM( '\\\' ) );
80:  CALL( PUTCHAR );
81:  PUSH( IMM( 'n' ) );
82:  CALL( PUTCHAR );
83:  DROP( 2 );
84:  JUMP( L_WSS_LOOP_CONT );
85:  L_WSS_OCT_CHAR:
86:  MOV( R0, IND( R2 ) );
87:  MOV( R3, R0 );
88:  REM( R3, IMM( 8 ) );
89:  PUSH( R3 );
90:  DIV( R0, IMM( 8 ) );
91:  MOV( R3, R0 );
92:  REM( R3, IMM( 8 ) );
93:  PUSH( R3 );
94:  DIV( R0, IMM( 8 ) );
95:  REM( R0, IMM( 8 ) );
96:  PUSH( R0 );
97:  PUSH( IMM( '\\\' ) );
98:  CALL( PUTCHAR );
99:  DROP( 1 );
100: CALL( WRITE_INTEGER );
101: DROP( 1 );
102: CALL( WRITE_INTEGER );
103: DROP( 1 );
104: CALL( WRITE_INTEGER );
105: DROP( 1 );
106: L_WSS_LOOP_CONT:
107: INCR( R2 );
108: DECR( R1 );
109: JUMP( L_WSS_LOOP );
110: L_WSS_EXIT:
111: PUSH( IMM( '\\"' ) );
112: CALL( PUTCHAR );
113: DROP( 1 );
114: POP( R3 );
115: POP( R2 );
116: POP( R1 );
117: POP( FP );
118: RETURN;
119:
```

```
1: /* scheme/write_sob_vector.asm
2:  * Take a pointer to a Scheme vector object, and
3:  * prints (to stdout) the character representation
4:  * of that object.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: WRITE_SOB_VECTOR:
10:  PUSH(FP);
11:  MOV(FP, SP);
12:  PUSH(R1);
13:  PUSH(R2);
14:  PUSH(IMM('#'));
15:  CALL(PUTCHAR);
16:  MOV(R0, FPARG(0));
17:  PUSH(INDD(R0, 1));
18:  CALL(WRITE_INTEGER);
19:  PUSH(IMM('('));
20:  CALL(PUTCHAR);
21:  DROP(3);
22:  MOV(R0, FPARG(0));
23:  MOV(R1, INDD(R0, 1));
24:  CMP(R1, IMM(0));
25:  JUMP_EQ(L_WSV_EXIT);
26:  MOV(R2, R0);
27:  ADD(R2, IMM(2));
28:  PUSH(IND(R2));
29:  CALL(WRITE_SOB);
30:  DROP(1);
31:  INCR(R2);
32:  DECR(R1);
33: L_WSV_LOOP:
34:  CMP(R1, IMM(0));
35:  JUMP_EQ(L_WSV_EXIT);
36:  PUSH(IMM(' '));
37:  CALL(PUTCHAR);
38:  DROP(1);
39:  PUSH(IND(R2));
40:  CALL(WRITE_SOB);
41:  DROP(1);
42:  INCR(R2);
43:  DECR(R1);
44:  JUMP(L_WSV_LOOP);
45: L_WSV_EXIT:
46:  PUSH(IMM(')'));
47:  CALL(PUTCHAR);
48:  DROP(1);
49:  POP(R2);
50:  POP(R1);
51:  POP(FP);
52:  RETURN;
53:
```

```
1: /* scheme/write_sob_void.asm
2:  * Take a pointer to a Scheme void object, and
3:  * prints (to stdout) the character representation
4:  * of that object.
5:  *
6:  * Programmer: Mayer Goldberg, 2010
7:  */
8:
9: WRITE_SOB_VOID:
10:  PUSH( IMM( '#' ) );
11:  CALL( PUTCHAR );
12:  PUSH( IMM( '<' ) );
13:  CALL( PUTCHAR );
14:  PUSH( IMM( 'v' ) );
15:  CALL( PUTCHAR );
16:  PUSH( IMM( 'o' ) );
17:  CALL( PUTCHAR );
18:  PUSH( IMM( 'i' ) );
19:  CALL( PUTCHAR );
20:  PUSH( IMM( 'd' ) );
21:  CALL( PUTCHAR );
22:  PUSH( IMM( '>' ) );
23:  CALL( PUTCHAR );
24:  DROP( 7 );
25:  RETURN;
```

```
1: /* io/write.asm
2:  * Takes a pointer to a null-terminated string,
3:  * and prints it to STDOUT.
4:  *
5:  * Programmer: Mayer Goldberg, 2010
6:  */
7:
8: WRITELN:
9:     MOV(R0, STARG(0));
10:    PUSH(R0);
11:    CALL(WRITE);
12:    POP(R0);
13:    PUSH(IMM('\n'));
14:    CALL(PUTCHAR);
15:    POP(R0);
16:    RETURN;
```