記号処理・第12回

2017年7月11日

今回の内容

12.1 文 (Statement) に対するコード生成 12–1
12.2 プログラム全体に対するコード生成の例 12–5
12.3 Minimum C コンパイラの最終的なプログラム 12–6
12.4 情報処理実習室の Linux 環境での実行 12–20

12.1 文 (Statement) に対するコード生成

前回は、Minimum C の式 $\langle Exp \rangle$ の値を計算して特定のレジスタへ格納する命令の生成について解説しました。この節では、Minimum C の文 $\langle Statement \rangle$ に対して、コンパイラがどうのような命令の列を生成すればよいかについて解説します。以下の説明では、式 $\langle Exp \rangle$ の値を計算してレジスタ Ri へ格納するために生成される命令の列を $\boxed{Exp \to \mathtt{R}i}$ で表し、また、文 *Statement* に対して生成される命令の列を $\boxed{Statement}$ で表すことにします。

メモ

代入文

$\boxed{Exp \to \mathtt{R1}}$
ST R1, Ri, d

代入文 *Ident* = *Exp*; に対しては、右のような命令の列を生成します。まず $\boxed{Exp \to \mathtt{R1}}$ によって右辺の式 *Exp* を計算した結果をレジスタ R1 に置くような命令列を生成し、次に、スコープリストを検索して、変数 *Ident* の置かれているアドレス R$i + d$ を調べ、レジスタ R1 の内容を R$i + d$ に格納するような ST 命令を置きます。

メモ

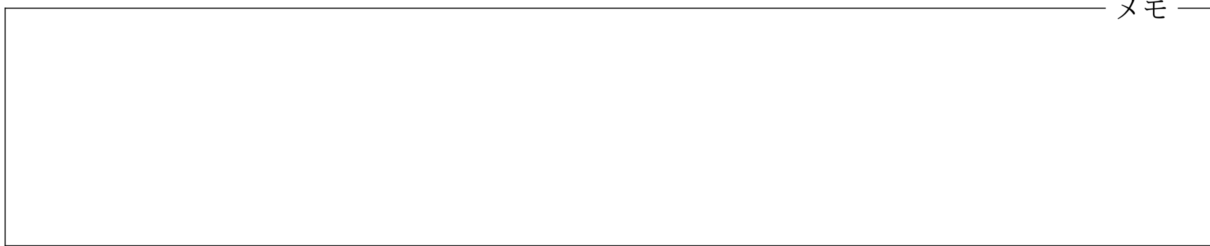
関数呼び出し文

$\boxed{Exp_1 \to \mathtt{R1}}$
PUSH R1
$\boxed{Exp_2 \to \mathtt{R1}}$
PUSH R1
⋮
$\boxed{Exp_n \to \mathtt{R1}}$
PUSH R1
CALL R0, L
ADDI R14, R14, $4n$

関数呼び出し文 *Ident* (Exp_1, Exp_2, ..., Exp_n); に対しては、前回解説した式中での関数呼び出しに対する命令の生成方法と同様な機械語命令を生成します。

ただし、関数呼び出し文が実行される場合は、仮想レジスタが使用されている可能性はないので、仮想レジスタをスタックへ退避する必要はありません。また、関数の返り値も利用されることがありません

ので、関数の返り値 R1 を他のレジスタにコピーする必要もなくなります。結局、右のような命令の列を生成すれば十分です。



return 文

```
[Exp → R1]
ADD R14, R13, R0
POP R13
POP R15
```

return *Exp*; に対しては、右のような命令の列を生成します。まず [*Exp* → R1] によって関数の戻り値 *Exp* を計算した結果をレジスタ R1 に置くような命令列を生成します。これに続く ADD 命令では、フレームポインタ (R13) をスタックポインタ (R14) にコピーし、その関数定義の本体の実行が開始された時点でののスタックポインタの値を復元させます。この後、スタック上に PUSH されていた古いフレームポインタの値が復元され、さらに、スタック上のリターンアドレスがプログラムカウンタ (R15) に POP されて関数の呼び出し元に制御が戻されます。

ただし、return 文が置かれている位置から参照可能な局所変数が (関数の仮引数を除けば) まったく存在しない場合には、ADD 命令が実行される時点でのスタックポインタとフレームポインタの値はすでに等しいはずなので、このような場合には、ADD 命令の生成は省略することができます。この省略が可能かどうかは、その時のスコープリストを調べれば決定することができます。

input 文

```
IN R1
ST R1, Ri, d
```

input *Ident*; に対しては、代入文と同様に、まずスコープリストを検索し、変数 *Ident* の置かれているアドレス R$i+d$ を得て、右のような 2 つの命令を生成します。IN 命令が実行されると、入力された数値がレジスタ R1 へ格納され、続く ST 命令によって、その値が変数 *Ident* に格納されます。

print 文

`print` Exp; に対しては、右のような命令の列を生成します。代入文や return 文の場合と同様に、まず [$\mathit{Exp} \to$ R1] によって Exp を計算した結果をレジスタ R1 に置くような命令列を生成し、続いて、R1 の内容を出力する OUT 命令を置きます。

```
[Exp → R1]
OUT R1
```

メモ

if 文

if（Exp_1 Rel Exp_2）$\mathit{Statement}$ の形の else の伴わない if 文に対しては、右のような命令の列を生成します。ただし L は、この if 文によって生成された命令の次の命令が置かれるアドレスとします。また、JPx は、条件が成立しなかった場合にのみ分岐を引き起すような条件付き分岐命令で、Rel が何であるかに応じて次のような命令の内の 1 つを選びます。

```
    [Exp1 → R1]
    [Exp2 → R2]
    SUB R0, R1, R2
    JPx R0, L
    [Statement]
L :
```

==	JPNE	<	JPGE	>	JPLE
!=	JPE	>=	JPLT	<=	JPGT

if 文の条件 Exp_1 Rel Exp_2 が成り立つ場合は、この JPx 命令は分岐を引き起しませんので、続く $\mathit{Statement}$ の部分に対して生成された命令が実行されます。条件 Exp_1 Rel Exp_2 が成り立たない場合は、JPx 命令は分岐を引き起し、この部分の命令は実行されません。この JPx 命令が生成される時点では、$\mathit{Statement}$ の構文はまだ読み込まれていないため、分岐先のアドレス L は分かりません。そこで、$\mathit{Statement}$ の部分に対する命令の生成が完了した時点で、JPx 命令の L の部分を遡って設定します。

メモ

同様に、if（Exp_1 Rel Exp_2）$\mathit{Statement}_1$ else $\mathit{Statement}_2$ の形の else を伴う if 文の場合は、右のような命令の列を生成します。

L_2 は、この else 付きの if 文によって生成された命令の次の命令が置かれるアドレスです。JPx 命令は else のない場合と同じように選びます。

if 文の条件 Exp_1 Rel Exp_2 が成り立つ場合は、この JPx 命令は分岐を引き起しませんので、続く $\mathit{Statement}_1$ の部分に対して生成された

```
     [Exp1 → R1]
     [Exp2 → R2]
     SUB R0, R1, R2
     JPx R0, L1
     [Statement1]
     JP R0, L2
L1 : [Statement2]
L2 :
```

命令が実行され、その後、無条件分岐命令 JP R0, L_2 によって、L_2 に分岐します。条件 Exp_1 Rel Exp_2 が成り立たない場合は、JPx 命令は L_1 への分岐を引き起し、$Statement_2$ の部分に対して生成された命令が実行されます。

メモ

while 文

while（Exp_1 Rel Exp_2）$Statement$ の形の構文に対しては、右のような命令の列を生成します。JPx 命令としては if 文の場合と同様に、条件 Exp_1 Rel Exp_2 が成立しなかった場合に分岐を行うような命令を選びます。while 文の場合も JPx 命令が生成される時点では、L_2 のアドレスは決まりませんので、JPx 命令の L_2 の値は $Statement$ に対する命令の生成が完了した時点で設定しなければなりません。

```
L1: [Exp1 → R1]
    [Exp2 → R2]
    SUB R0, R1, R2
    JPx R0, L2
    [Statement]
    JP R0, L1
L2:
```

while 文の条件 Exp_1 Rel Exp_2 が成り立つ場合は、JPx 命令は分岐を引き起しませんので、続く $Statement$ の部分に対して生成された命令が実行され、その後、無条件分岐命令 JP R0, L_1 によって、この while 文に対して生成された命令の列の先頭に戻ります。条件 Exp_1 Rel Exp_2 が成り立たない場合は、JPx 命令は L_2 への分岐を引き起し、この while 文の実行を終えます。

メモ

ブロック

ブロック { int $Ident_1$, ..., $Ident_m$; $Statement_1$ ... $Statement_n$ } に対しては右のような命令を生成します。まず、そのブロックの先頭で宣言されている局所変数をスタック上に割り当てるために、スタックポインタ (R14) を局所変数の数 m に応じて減じるような命令を生成します。続いてブロック中の各 $Statement$ に対して順に命令を生成し、最後にスタックポインタを元に戻し局所変数のための領域を解放する命令を生成します。ただし、このブロックで新たに宣言された局所変数が無い場合は、先頭の SUBI 命令と最後の ADDI 命令は省略することができます。

```
SUBI R14, R14, 4m
[Statment1]
    ⋮
[Statmentn]
ADDI R14, R14, 4m
```

メモ

12.2　プログラム全体に対するコード生成の例

以下は、入力された 2 つの自然数の最大公約数をユークリッドの互除法によって計算して出力するプログラムに対して生成されたオブジェクトプログラムです。オブジェクトプログラムの先頭には、(1) フレームポインタ R13 の初期化、(2) 大域変数領域の先頭を示す R11 の初期化、(3) 関数 main の呼び出し、(4) プログラム全体の終了、の 4 つの命令が置かれ、それに続いて、ソースプログラム中で定義された各関数定義に対応する命令が生成されています。

```
                          0 ADD R13, R14, R0
                          4 LDI R11,216
                          8 CALL R0,148
                         12 EXIT R1

gcd (x, y)               16 PUSH R13
                         20 ADD R13, R14, R0
{
    int z;               24 SUBI R14, R14, 4

    if (x < y) {         28 LD R1, R13, 12
                         32 LD R2, R13, 8
                         36 SUB R0, R1, R2
                         40 JPGE R0, 68
        z = x;           44 LD R1, R13, 12
                         48 ST R1, R13, -4
        x = y;           52 LD R1, R13, 8
                         56 ST R1, R13, 12
        y = z;           60 LD R1, R13, -4
                         64 ST R1, R13, 8
    }

    while (y!= 0) {      68 LD R1, R13, 8
                         72 LDI R2, 0
                         76 SUB R0, R1, R2
                         80 JPE R0,120
        z = y;           84 LD R1, R13, 8
                         88 ST R1, R13, -4
        y = x % y;       92 LD R1, R13, 12
                         96 LD R2, R13, 8
                        100 MOD R1, R1, R2
                        104 ST R1, R13, 8
        x = z;          108 LD R1, R13, -4
                        112 ST R1, R13, 12
    }                   116 JP R0, 68
    return x;           120 LD R1, R13, 12
                        124 ADD R14, R13, R0
                        128 POP R13
                        132 POP R15
}                       136 ADDI R14, R14, 4
                        140 POP R13
                        144 POP R15

main ()                 148 PUSH R13
                        152 ADD R13, R14, R0
{
    int a, b;           156 SUBI R14, R14, 8
    input a;            160 IN R1
                        164 ST R1, R13, -4
    input b;            168 IN R1
                        172 ST R1, R13, -8
    print gcd(a, b);    176 LD R1, R13, -4
                        180 PUSH R1
                        184 LD R1, R13, -8
                        188 PUSH R1
                        192 CALL R0, 16
                        196 ADDI R14, R14, 8
                        200 OUT R1
}                       204 ADDI R14, R14, 8
                        208 POP R13
                        212 POP R15
```

メモ

12.3 Minimum C コンパイラの最終的なプログラム

前回までに作成した構文解析部にコード生成部を追加することで Minimum C コンパイラが完成します。字句解析部 scan.h や scan.c は変更する必要はありません。コード生成を支援するために gen.h と gen.c という 2 つのファイルが新たに用意されています。

コード生成のための補助プログラム (gen.h, gen.c)

gen.h

```
 1 /*
 2  *      gen.h
 3  */
 4
 5 #include <limits.h>
 6
 7 /* コンパイラが生成可能な最大の機械語命令数 */
 8 #define MAXPROGLEN  (100000)
 9
10 /* MVM の各レジスタの識別番号 */
11 #define MVM_R0  (0)
12 #define MVM_R1  (1)
13 #define MVM_R2  (2)
14 #define MVM_R3  (3)
15 #define MVM_R4  (4)
16 #define MVM_R5  (5)
17 #define MVM_R6  (6)
18 #define MVM_R7  (7)
19 #define MVM_R8  (8)
20 #define MVM_R9  (9)
21 #define MVM_R10 (10)
22 #define MVM_GP  (11)
23 #define MVM_RA  (12)
24 #define MVM_FP  (13)
25 #define MVM_SP  (14)
26 #define MVM_PC  (15)
27
28 /* MVM の各機械語命令の上位 8 bit */
29 #define MVM_LD          (0x00)
30 #define MVM_ST          (0x10)
31 #define MVM_LDB         (0x20)
32 #define MVM_LDUB        (0x30)
33 #define MVM_STB         (0x40)
34 #define MVM_LDI         (0x80)
35 #define MVM_LDHI        (0x81)
36 #define MVM_POP         (0x82)
37 #define MVM_PUSH        (0x83)
38 #define MVM_CALL        (0x84)
39 #define MVM_JPL         (0x85)
40 #define MVM_IN          (0x88)
41 #define MVM_OUT         (0x89)
42 #define MVM_INQ         (0x8a)
43 #define MVM_OUTQ        (0x8b)
44 #define MVM_OUTS        (0x8c)
45 #define MVM_EXIT        (0x8f)
46 #define MVM_JP          (0x90)
47 #define MVM_JPE         (0x92)
48 #define MVM_JPNE        (0x93)
49 #define MVM_JPGT        (0x94)
50 #define MVM_JPLE        (0x95)
51 #define MVM_JPLT        (0x96)
52 #define MVM_JPGE        (0x97)
53 #define MVM_JPGTU       (0x98)
54 #define MVM_JPLEU       (0x99)
55 #define MVM_JPC         (0x9a)
```

```
 56 #define MVM_JPNC        (0x9b)
 57 #define MVM_JPNEG       (0x9c)
 58 #define MVM_JPNNEG      (0x9d)
 59 #define MVM_JPV         (0x9e)
 60 #define MVM_JPNV        (0x9f)
 61 #define MVM_ADD         (0xa0)
 62 #define MVM_ADDC        (0xa1)
 63 #define MVM_SUB         (0xa2)
 64 #define MVM_SUBC        (0xa3)
 65 #define MVM_AND         (0xa4)
 66 #define MVM_OR          (0xa5)
 67 #define MVM_XOR         (0xa6)
 68 #define MVM_NXOR        (0xa7)
 69 #define MVM_SLL         (0xa8)
 70 #define MVM_SRL         (0xa9)
 71 #define MVM_SRA         (0xaa)
 72 #define MVM_ADDI        (0xb0)
 73 #define MVM_ADDIC       (0xb1)
 74 #define MVM_SUBI        (0xb2)
 75 #define MVM_SUBIC       (0xb3)
 76 #define MVM_ANDI        (0xb4)
 77 #define MVM_ORI         (0xb5)
 78 #define MVM_XORI        (0xb6)
 79 #define MVM_NXORI       (0xb7)
 80 #define MVM_SLLI        (0xb8)
 81 #define MVM_SRLI        (0xb9)
 82 #define MVM_SRAI        (0xba)
 83 #define MVM_MUL         (0xc0)
 84 #define MVM_DIV         (0xc1)
 85 #define MVM_MOD         (0xc2)
 86 #define MVM_MULU        (0xc4)
 87 #define MVM_DIVU        (0xc5)
 88 #define MVM_MODU        (0xc6)
 89
 90 #if UINT_MAX >= 0xffffffff
 91 typedef unsigned int    Word;
 92 typedef int             SWord;
 93 typedef int             Offset;
 94 typedef unsigned int    Address;
 95 #else
 96 typedef unsigned long   Word;
 97 typedef long            SWord;
 98 typedef long            Offset;
 99 typedef unsigned long   Address;
100 #endif
101
102 extern Address  CurrentPC;
103 extern void     PutInst(Word op, Word i, Word j, SWord dk);
104 extern void     FixInst(Address pc, Word d);
105 extern void     FixInstChain(Address pc, Word d);
106 extern void     InitInst(void);
107 extern int      WriteObjectFile(char * filename);
108
```

gen.c

```
  1 /*
  2  *      gen.c
  3  */
  4
  5 #include <stdio.h>
  6 #include <stdlib.h>
  7 #include <string.h>
  8 #include "scan.h"
  9 #include "gen.h"
 10
```

```
11 static Word     Mem[MAXPROGLEN];
12
13 Address         CurrentPC;
14
15 /* PutInst(): CurrentPC の位置に指定された命令を生成し、CurrentPC を
16               進める。 */
17
18 void PutInst(Word op, Word i, Word j, SWord dk)
19 {
20     long waddr = CurrentPC / sizeof(Word);
21
22     if (waddr >= MAXPROGLEN)
23         Error("機械語プログラムが大きくなり過ぎます");
24     CurrentPC += sizeof(Word);
25     switch (op) {
26         case MVM_LD: case MVM_ST: case MVM_LDB: case MVM_LDUB: case MVM_STB:
27             Mem[waddr] = (op << 24) | (i << 24) | (j << 20) | (dk & 0xfffff);
28             break;
29         case MVM_LDI: case MVM_LDHI: case MVM_POP: case MVM_PUSH:
30         case MVM_CALL: case MVM_JPL: case MVM_IN: case MVM_OUT:
31         case MVM_INQ: case MVM_OUTQ: case MVM_OUTS: case MVM_EXIT:
32         case MVM_JP: case MVM_JPE: case MVM_JPNE:
33         case MVM_JPGT: case MVM_JPLE: case MVM_JPLT: case MVM_JPGE:
34         case MVM_JPGTU: case MVM_JPLEU: case MVM_JPC: case MVM_JPNC:
35         case MVM_JPNEG: case MVM_JPNNEG: case MVM_JPV: case MVM_JPNV:
36             Mem[waddr] = (op << 24) | (i << 20) | (dk & 0xfffff);
37             break;
38         case MVM_ADD: case MVM_ADDC: case MVM_SUB: case MVM_SUBC:
39         case MVM_AND: case MVM_OR: case MVM_XOR: case MVM_NXOR:
40         case MVM_SLL: case MVM_SRL: case MVM_SRA:
41         case MVM_ADDI: case MVM_ADDIC: case MVM_SUBI: case MVM_SUBIC:
42         case MVM_ANDI: case MVM_ORI: case MVM_XORI: case MVM_NXORI:
43         case MVM_SLLI: case MVM_SRLI: case MVM_SRAI:
44         case MVM_MUL: case MVM_DIV: case MVM_MOD:
45         case MVM_MULU: case MVM_DIVU: case MVM_MODU:
46             Mem[waddr] = (op << 24) | (i << 20) | (j << 16) | (dk & 0xffff);
47         default:
48             break;
49     }
50 }
51
52 /* FixInst(): 指定された位置の命令のアドレス部分(下位20bit)を修正する。 */
53
54 void FixInst(Address pc, Address d)
55 {
56     long waddr = pc / sizeof(Word);
57
58     Mem[waddr] = (Mem[waddr] & ~0xfffff) | d;
59 }
60
61 /* FixInstChain(): 指定された位置から始るリストをたどりながら、リスト中の
62                    命令のアドレス部分(下位20bit)を修正する。 */
63
64 void FixInstChain(Address pc, Address d)
65 {
66     Address next;
67
68     while (pc) {
69         next = Mem[pc / sizeof(Word)] & 0xfffff;
70         FixInst(pc, d);
71         pc = next;
72     }
73 }
74
75 void InitInst(void)
```

```
76 {
77     CurrentPC = 0;
78 }
79
80 /* WriteObjectFile(): コンパイラのメモリ中に生成された命令の列をファイルに
81                       格納する。 */
82
83 int WriteObjectFile(char * filename)
84 {
85     FILE *out = fopen(filename, "wb");
86     long waddr;
87
88     if (out == NULL)
89         return -1;
90     for (waddr = 0; waddr < CurrentPC / sizeof(Word); waddr ++) {
91         putc((Mem[waddr] >> 24) & 0xff, out);
92         putc((Mem[waddr] >> 16) & 0xff, out);
93         putc((Mem[waddr] >> 8) & 0xff, out);
94         putc(Mem[waddr] & 0xff, out);
95     }
96     fclose(out);
97     return 0;
98 }
```

コード生成部を追加した構文解析部のプログラム (**parse.c**)

第 10 回の parse1.c から変更された行には * が付してあります。

parse.c

```
   1 /*
*  2  *      parse.c
   3  */
   4
   5 #include <stdio.h>
   6 #include <stdlib.h>
   7 #include <string.h>
   8 #include "scan.h"
*  9 #include "gen.h"
  10
  11 typedef enum { Var, Fun, Unknown } ObjectType;
  12
  13 typedef struct Object {
  14     ObjectType      type;       /* オブジェクトの種類(変数/関数/未定義) */
  15     char            name[MC_MAXIDLEN+1];        /* 変数名/関数名 */
  16     int             reg;        /* 変数アドレスの基準となるレジスタ番号
  17                                    関数の場合は R0、大域変数の場合は GP (R11)、
  18                                    関数の仮引数や局所変数の場合は FP (R13) */
  19     Offset          offset;     /* reg からのオフセット(相対位置) */
  20     struct Object * next;       /* 同じスコープの次のオブジェクト */
  21 } Object;
  22
  23 typedef struct Scope {
  24     struct Scope *  outer;      /* 一つ外側のスコープ */
  25     Object *        objects;    /* このスコープのオブジェクトのリスト */
  26     Offset          offset;     /* 新しい変数の次の割り当て(相対)位置 */
  27 } Scope;
  28
  29 static Scope * GlobalScope;     /* 大域(一番外側の)スコープ */
  30 static Scope * CurrentScope;    /* 現在コンパイル中のスコープ */
  31
  32 /* FindObject() : 指定のスコープのオブジェクリストからオブジェクトを探し
  33                   出す。 見つからない場合は 0 を返す。 */
  34
```

```
35 static Object * FindObject(Scope *scope, char *name)
36 {
37     Object *obj;
38
39     for (obj = scope->objects; obj != (Object *)0; obj = obj->next) {
40         if (strcmp(name, obj->name) == 0)
41             return obj;
42     }
43     return (Object *)0;
44 }
45
46 /* NewObject() : 新しいオブジェクトを作成する。 */
47
48 static Object * NewObject(ObjectType type, char *name)
49 {
50     Object *obj;
51
52     obj = (Object *) malloc(sizeof(Object));
53     if (obj == (Object *)0)
54         Error("コンパイルするためのメモリが足りません");
55     strcpy(obj->name, name);
56     obj->type = type;
57     obj->next = (Object *)0;
58     return obj;
59 }
60
61 /* NewVar() : 新しい変数を現在のスコープに追加する。*/
62
63 static Object * NewVar(char *name)
64 {
65     Object *obj;
66
67     if (FindObject(CurrentScope, name) != (Object *)0)
68         Error("変数名または引数名が衝突しています");
69
70     obj = NewObject(Var, name);
71     obj->next = CurrentScope->objects;
72
73     if (CurrentScope->outer == (Scope *)0) { /* 大域変数 */
74         obj->reg = MVM_GP;
75         obj->offset = CurrentScope->offset;
76         CurrentScope->offset += sizeof(Word);
77     }
78     else if (CurrentScope->outer->outer == (Scope *)0) { /* 仮引数 */
79         obj->reg = MVM_FP;
80         obj->offset = 0;       /* 後で FunDef() が設定する */
81     }
82     else { /* 局所変数 */
83         obj->reg = MVM_FP;
84         CurrentScope->offset -= sizeof(Word);
85         obj->offset = CurrentScope->offset;
86     }
87
88     CurrentScope->objects = obj;
89     return obj;
90 }
91
92 /* UseObject() : オブジェクトを探し出す。 見つからなければ、
93                  type=Unknown として大域スコープにオブジェクトを作る。 */
94
95 static Object * UseObject(char *name)
96 {
97     Scope *scope;
```

```
 98     Object *obj;
 99
100     /* 内側のスコープから大域スコープに向って順にオブジェクトを探す */
101     for (scope = CurrentScope; scope != (Scope *)0; scope = scope->outer) {
102         obj = FindObject(scope, name);
103         if (obj != (Object *)0)
104             return obj;
105     }
106
107     /* 大域スコープに新しいオブジェクトを作る */
108     obj = NewObject(Unknown, name);
109     obj->reg = MVM_R0;
110     obj->offset = 0;
111     obj->next = GlobalScope->objects;
112     GlobalScope->objects = obj;
113     return obj;
114 }
115
116 /* NewScope() : 新しいスコープを作成し、それを CurrentScope とする。 */
117
118 static void NewScope(Offset offset)
119 {
120     Scope * scope;
121
122     scope = (Scope *) malloc(sizeof(Scope));
123     if (scope == (Scope *)0)
124         Error("コンパイルするためのメモリが足りません");
125     scope->outer = CurrentScope;
126     scope->objects = (Object *)0;
127     scope->offset = offset;
128     CurrentScope = scope;
129 }
130
131 /* ExitScope() : CurrentScope を廃棄し、一つ外側のスコープに出る。 */
132
133 static void ExitScope(void)
134 {
135     char message[100+MC_MAXIDLEN];
136     Scope *cur = CurrentScope;
137     Object *obj, *next;
138
139     /* 未定義関数のチェック */
140     for (obj = cur->objects; obj != (Object *)0; obj = next) {
141         if (obj->type == Unknown) {
142             sprintf(message, "関数 %s が未定義です", obj->name);
143             Error(message);
144         }
145         next = obj->next;
146         free((void *)obj);
147     }
148
149     CurrentScope = CurrentScope->outer;
150     free((void *)cur);
151 }
152
*153 #define N_EXPREG 4      /* 式の値の計算に使用する物理レジスタの総数 */
154
*155 /* RealReg(): 仮想レジスタ vreg が割り当てられるべき物理レジスタ番号 */
*156 #define RealReg(vreg)   (MVM_R1 + (vreg) % N_EXPREG)
157
*158 static int CurrentVRegTop = 0;  /* 使用中の仮想レジスタの最大の番号 */
159
*160 /* ActivateVReg(): 仮想レジスタ vreg を使用可能な状態にする。 物理レジス
```

```
*161                     タがすでに一杯ならば、一部をスタックに退避して物理レ
*162                     ジスタを vreg として使用できるようにする。 また、vreg
*163                     がすでにスタック上に退避されている場合はスタック上か
*164                     ら物理レジスタに復帰する。 */
 165
*166 static void ActivateVReg(int vreg)
*167 {
*168     while (CurrentVRegTop < vreg) {
*169         CurrentVRegTop ++;
*170         if (N_EXPREG <= CurrentVRegTop)
*171             PutInst(MVM_PUSH, RealReg(CurrentVRegTop), 0, 0);
*172     }
 173
*174     while (vreg <= CurrentVRegTop - N_EXPREG) {
*175         PutInst(MVM_POP, RealReg(CurrentVRegTop), 0, 0);
*176         CurrentVRegTop --;
*177     }
*178 }
 179
*180 /* SaveVReg(): 関数呼び出しに先だって、vreg より小さい番号の仮想レジスタの
*181                 ために使われている物理レジスタをすべてスタックに退避する。 */
 182
*183 static void SaveVReg(int vreg)
*184 {
*185     int r, oldest = CurrentVRegTop - N_EXPREG + 1;
 186
*187     if (oldest < 0)
*188         oldest = 0;
*189     for (r = oldest; r < vreg; r ++)
*190         PutInst(MVM_PUSH, RealReg(r), 0, 0);
*191     CurrentVRegTop = 0;
*192 }
 193
*194 /* RestoreVReg(): SaveVReg(vreg) によって仮想レジスタがスタックに退避され
*195                   た後の状況に整合するように CurrentVRegTop を設定する。
*196                   RestoreVReg() は関数呼び出しの後に呼び出される。 */
 197
*198 static void RestoreVReg(int vreg)
*199 {
*200     CurrentVRegTop = vreg + N_EXPREG - 1;
*201 }
 202
*203 static void Exp(int vreg);
 204 static void Block(void);
 205
 206
*207 /* FunArgs(): 関数呼び出しの実引数部を構文解析し、左の引数から順番に式の
*208               値を計算してスタックに PUSH する命令群を生成する。
*209               FunArgs() の返り値は実引数の個数。 */
 210
*211 static int FunArgs(void)
 212 {
*213     int n = 0;
 214
 215     GetToken();
 216     if (token != mc_rparen) {
*217         Exp(0);                  /* 最初の実引数 */
*218         PutInst(MVM_PUSH, RealReg(0), 0, 0);
*219         n ++;
 220         while (token == mc_comma) {
 221             GetToken();
*222             Exp(0);              /* 2番目以降の実引数 */
*223             PutInst(MVM_PUSH, RealReg(0), 0, 0);
*224             n ++;
```

```
 225          }
 226          if (token != mc_rparen)
 227              Error("関数呼び出しの引数リストに ) がありません");
 228      }
 229      GetToken();
*230      return n;
 231  }
 232
*233  /* Factor(): <Factor> を構文解析し、その値を計算した結果を仮想レジスタ
*234                vreg に置くような命令を生成する。 */
 235
*236  static void Factor(int vreg)
 237  {
*238      int n;
*239      Address l1;
 240      Object *obj;
 241
 242      switch (token) {
 243      case mc_ident:
 244          /* 変数: <Ident>、または、関数呼び出し: <Ident> <FunArgs> */
 245          obj = UseObject(ident);
 246          GetToken();
 247          if (token == mc_lparen) {
*248              /* 関数呼び出し: <Ident> <FunArgs> */
 249              if (obj->type == Var)
 250                  Error("変数に対する関数呼び出しはできません");
*251              SaveVReg(vreg);
*252              n = FunArgs();
*253              l1 = CurrentPC;
*254              PutInst(MVM_CALL, obj->reg, 0, obj->offset);
*255              if (obj->type == Unknown) {
*256                  /* 未定義の関数なので、この CALL 命令をリストに追加する。
*257                     この関数が定義された時点で CALL 命令のアドレス部分を
*258                     設定する。 */
*259                  obj->offset = l1;
*260              }
*261              if (n > 0)
*262                  PutInst(MVM_ADDI, MVM_SP, MVM_SP, n*sizeof(Word));
*263              if (RealReg(vreg) != RealReg(0))
*264                  PutInst(MVM_ADD, RealReg(vreg), RealReg(0), MVM_R0);
*265              RestoreVReg(vreg);
 266          }
*267          else if (obj->type == Var) {
*268              /* 変数: <Ident> */
*269              ActivateVReg(vreg);
*270              PutInst(MVM_LD, RealReg(vreg), obj->reg, obj->offset);
*271          }
*272          else
 273              Error("宣言されていない変数は使用できません");
 274          break;
 275      case mc_number:
 276          /* 定数: <Number> */
*277          ActivateVReg(vreg);
*278          if (number < (1 << 19)) {
*279              PutInst(MVM_LDI, RealReg(vreg), 0, number);
*280          }
*281          else {
*282              PutInst(MVM_LDHI, RealReg(vreg), 0, number >> 16);
*283              PutInst(MVM_ADDI, RealReg(vreg), RealReg(vreg),
*284                      number & 0xffff);
*285          }
 286          GetToken();
 287          break;
 288      case mc_lparen:
```

```
 289         /* 括弧で囲まれた式: ( <Exp> ) */
 290         GetToken();
*291         Exp(vreg);
 292         if (token == mc_rparen)
 293             GetToken();
 294         else
 295             Error("閉じ括弧があるべき位置にありません");
 296         break;
 297     default:
 298         Error("式に文法誤りがあります");
 299         break;
 300     }
 301 }
 302
*303 /* Term(): <Term> を構文解析し、その値を計算した結果を仮想レジスタ vreg
*304            に置くような命令を生成する。 */
 305
*306 static void Term(int vreg)
 307 {
*308     Token op;
 309
*310     Factor(vreg);
 311     while (token == mc_mult || token == mc_div || token == mc_mod) {
*312         op = token;
 313         GetToken();
*314         Factor(vreg+1);
*315         ActivateVReg(vreg);
*316         switch (op) {
*317         case mc_mult:
*318             PutInst(MVM_MUL, RealReg(vreg), RealReg(vreg), RealReg(vreg+1));
*319             break;
*320         case mc_div:
*321             PutInst(MVM_DIV, RealReg(vreg), RealReg(vreg), RealReg(vreg+1));
*322             break;
*323         case mc_mod:
*324             PutInst(MVM_MOD, RealReg(vreg), RealReg(vreg), RealReg(vreg+1));
*325             break;
*326         }
 327     }
 328 }
 329
*330 /* Exp(): <Exp> を構文解析し、その値を計算した結果を仮想レジスタ vreg に置
*331           くような命令を生成する。 */
 332
*333 static void Exp(int vreg)
 334 {
*335     Token op = mc_plus;
 336
 337     if (token == mc_plus || token == mc_minus) {
*338         op = token;
_339         GetToken();
 340     }
*341     Term(vreg);
*342     if (op == mc_minus)
*343         PutInst(MVM_SUB, RealReg(vreg), MVM_R0, RealReg(vreg));
 344     while (token == mc_plus || token == mc_minus) {
*345         op = token;
 346         GetToken();
*347         Term(vreg+1);
*348         ActivateVReg(vreg);
*349         if (op == mc_minus)
*350             PutInst(MVM_SUB, RealReg(vreg), RealReg(vreg), RealReg(vreg+1));
*351         else
*352             PutInst(MVM_ADD, RealReg(vreg), RealReg(vreg), RealReg(vreg+1));
```

```
 353     }
 354 }
 355
*356 /* Cond(): <Cond> を構文解析し、その条件が成り立たない場合に分岐を行うよう
*357            な命令を生成する。 分岐命令が置かれたアドレスを返す。 */
 358
*359 static Address Cond(void)
 360 {
*361     Address l;
*362     Word op;
*363     Token rel;
 364
*365     Exp(0);
 366     if (mc_eq <= token && token <= mc_ge) {
*367         rel = token;
 368         GetToken();
*369         Exp(1);
*370         ActivateVReg(0);
*371         PutInst(MVM_SUB, MVM_R0, RealReg(0), RealReg(1));
*372         l = CurrentPC;
*373         switch (rel) {
*374             case mc_eq: op = MVM_JPNE; break;
*375             case mc_neq: op = MVM_JPE; break;
*376             case mc_gt: op = MVM_JPLE; break;
*377             case mc_ge: op = MVM_JPLT; break;
*378             case mc_lt: op = MVM_JPGE; break;
*379             case mc_le: op = MVM_JPGT; break;
*380         }
*381         PutInst(op, MVM_R0, 0, 0);
*382         return l;
 383     }
 384     else
 385         Error("条件式に文法誤りがあります");
 386 }
 387
*388 /* Statement(): <Statement> を構文解析し、対応する命令を生成する。*/
 389
 390 static void Statement(void)
 391 {
*392     int n;
*393     Address l1, l2;
 394     Object *obj;
 395
 396     switch (token) {
 397     case mc_ident:
 398         /* <Ident> = <Exp>; | <Ident> <FunArgs>; */
 399         obj = UseObject(ident);
 400         GetToken();
 401         switch (token) {
 402         case mc_assign:
 403             if (obj->type != Var)
 404                 Error("宣言されていない変数への代入はできません");
 405             GetToken();
*406             Exp(0);
*407             PutInst(MVM_ST, RealReg(0), obj->reg, obj->offset);
 408             if (token != mc_semicolon)
 409                 Error("代入文に ; がありません");
 410             GetToken();
 411             break;
 412         case mc_lparen:
 413             if (obj->type == Var)
 414                 Error("変数に対する関数呼び出しはできません");
*415             n = FunArgs();
*416             l1 = CurrentPC;
```

```
*417            PutInst(MVM_CALL, obj->reg, 0, obj->offset);
*418            if (obj->type == Unknown)
*419                obj->offset = l1;
*420            if (n > 0)
*421                PutInst(MVM_ADDI, MVM_SP, MVM_SP, n*sizeof(Word));
 422            if (token != mc_semicolon)
 423                Error("関数呼び出し文に ; がありません");
 424            GetToken();
 425            break;
 426        default:
 427            Error("代入文の =、または、関数呼び出し文の ( がありません");
 428            break;
 429        }
 430        break;
 431    case mc_return:
 432        /* return <Exp>; */
 433        GetToken();
*434        Exp(0);
*435        if (CurrentScope->offset < 0)
*436            PutInst(MVM_ADD, MVM_SP, MVM_FP, MVM_R0);
*437        PutInst(MVM_POP, MVM_FP, 0, 0);
*438        PutInst(MVM_POP, MVM_PC, 0, 0);
 439        if (token != mc_semicolon)
 440            Error("return 文に ; がありません");
 441        GetToken();
 442        break;
 443    case mc_input:
 444        /* input <Ident>; */
 445        GetToken();
 446        if (token != mc_ident)
 447            Error("input 文に変数名がありません");
 448        obj = UseObject(ident);
 449        if (obj->type != Var)
 450            Error("input 文の変数が宣言されていません");
*451        PutInst(MVM_IN, MVM_R1, 0, 0);
*452        PutInst(MVM_ST, MVM_R1, obj->reg, obj->offset);
 453        GetToken();
 454        if (token != mc_semicolon)
 455            Error("input 文に ; がありません");
 456        GetToken();
 457        break;
 458    case mc_print:
 459        /* print <Exp>; */
 460        GetToken();
*461        Exp(0);
*462        PutInst(MVM_OUT, RealReg(0), 0, 0);
 463        if (token != mc_semicolon)
 464            Error("print 文に ; がありません");
 465        GetToken();
 466        break;
 467    case mc_if:
 468        /* if ( <Cond> ) <Statement> [ else <Statement> ] */
 469        GetToken();
 470        if (token != mc_lparen)
 471            Error("if の後に ( がありません");
 472        GetToken();
*473        l1 = Cond();
 474        if (token != mc_rparen)
 475            Error("if の条件式の後に ) がありません");
 476        GetToken();
 477        Statement();
 478        if (token == mc_else) {
*479            l2 = CurrentPC;
*480            PutInst(MVM_JP, MVM_R0, 0, 0);
```

```
*481             FixInst(l1, CurrentPC);
 482             GetToken();
 483             Statement();
*484             FixInst(l2, CurrentPC);
 485         }
*486         else
*487             FixInst(l1, CurrentPC);
 488         break;
 489     case mc_while:
 490         /* while ( <Cond> ) <Statement> */
 491         GetToken();
 492         if (token != mc_lparen)
 493             Error("while の後に ( がありません");
 494         GetToken();
*495         l1 = CurrentPC;
*496         l2 = Cond();
 497         if (token != mc_rparen)
 498             Error("while の条件式の後に ) がありません");
 499         GetToken();
 500         Statement();
*501         PutInst(MVM_JP, MVM_R0, 0, l1);
*502         FixInst(l2, CurrentPC);
 503         break;
 504     case mc_lcb:
 505         /* "{" [ <VarDef> ] { <Statement> } "}" */
 506         Block();
 507         break;
 508     case mc_semicolon:
 509         Error("空文は許されていません");
 510         break;
 511     default:
 512         Error("文を始めるべき記号に誤りがあります");
 513         break;
 514     }
 515 }
 516
*517 /* VarDef(): <VarDef> を構文解析し、CurrentScope のオブジェクトリストを
*518              作成する。 */
 519
*520 static int VarDef(void)
 521 {
*522     int n = 0;
 523
 524     GetToken();
 525     if (token != mc_ident)
 526         Error("宣言される変数名がありません");
 527     NewVar(ident);
*528     n ++;
 529     GetToken();
 530     while (token == mc_comma) {
 531         GetToken();
 532         if (token == mc_ident) {
 533             NewVar(ident);
*534             n ++;
 535             GetToken();
 536         }
 537     }
 538     if (token != mc_semicolon)
 539         Error("変数宣言に , か ; がありません");
 540     GetToken();
*541     return n;
 542 }
 543
*544 /* FunDef(): <FunDef> を構文解析し、関数定義に対応する命令を生成する。 */
 545
 546 static void FunDef(void)
```

```
 547 {
 548     Offset offset;
 549     Object *obj;
 550
 551     obj = UseObject(ident);
 552     if (obj->type == Var)
 553         Error("変数名を関数名として再定義することはできません");
 554     if (obj->type == Fun)
 555         Error("関数を再定義することはできません");
 556     obj->type = Fun;
 557     GetToken();
 558
*559     /* この関数に対してすでに生成された CALL 命令のアドレス部分を設定する。 */
*560     FixInstChain(obj->offset, CurrentPC);
 561
*562     /* この関数オブジェクトのアドレスを設定する。 */
*563     obj->offset = CurrentPC;
 564
*565     /* 新しいスコープを追加し、仮引数をそのオブジェクトリストに登録する。 */
*566     NewScope((Offset)0);
 567     if (token != mc_lparen)
 568         Error("関数定義の引数リストに ( がありません");
 569     GetToken();
 570     if (token != mc_rparen) {
 571         if (token != mc_ident)
 572             Error("関数定義の引数リストに文法誤りがあります");
 573         obj = NewVar(ident);
 574         GetToken();
 575         while (token == mc_comma) {
 576             GetToken();
 577             if (token != mc_ident)
 578                 Error("関数定義の引数リストに文法誤りがあります");
 579             obj = NewVar(ident);
 580             GetToken();
 581         }
 582         if (token != mc_rparen)
 583             Error("関数定義の引数リストに ) がありません");
 584     }
 585     GetToken();
 586
*587     /* 仮引数のアドレス(offset)を計算して設定する。 */
 588     offset = 2 * sizeof(Word);  /* FP から最後の引数までのオフセット */
 589     /* 最後の引数から順に、引数のオフセットを設定 */
 590     for (obj = CurrentScope->objects; obj != (Object *)0; obj = obj->next) {
 591         obj->offset = offset;
 592         offset += sizeof(Word);
 593     }
 594
*595     /* 命令を生成する */
*596     PutInst(MVM_PUSH, MVM_FP, 0, 0);
*597     PutInst(MVM_ADD, MVM_FP, MVM_SP, MVM_R0);
 598     Block();
*599     PutInst(MVM_POP, MVM_FP, 0, 0);
*600     PutInst(MVM_POP, MVM_PC, 0, 0);
 601     ExitScope();
 602 }
 603
*604 /* Block(): <Block> を構文解析し、対応する命令を生成する。 */
 605
 606 static void Block(void)
 607 {
*608     int n = 0;
 609
 610     if (token != mc_lcb)
 611         Error("ブロックが { で始まっていません");
```

```
*612     /* 新しいスコープを追加し、局所変数をそのオブジェクトリストに登録する。 */
 613     NewScope(CurrentScope->offset);
 614     GetToken();
*615     if (token == mc_int) {
*616         n = VarDef();
*617         /* 局所変数の数に見合う領域をスタック上に確保する。 */
*618         PutInst(MVM_SUBI, MVM_SP, MVM_SP, n*sizeof(Word));
*619     }
 620
*621     /* ブロック中の各 <Statement> を処理する。*/
 622     while (token != mc_rcb)
 623         Statement();
 624
*625     /* スタックポインタを戻して、局所変数のための領域を解放する。 */
*626     if (n > 0)
*627         PutInst(MVM_ADDI, MVM_SP, MVM_SP, n*sizeof(Word));
 628
*629     /* このブロックのためのスコープを廃棄する。 */
 630     GetToken();
 631     ExitScope();
 632 }
 633
*634 /* Prog(): プログラム全体 <Prog> を構文解析し、対応する命令を生成する。 */
 635
 636 static void Prog(void)
 637 {
*638     Address l1, l2;
*639     Object *obj;
 640
*641     /* コード生成部の初期化を行う */
*642     InitInst();
*643     CurrentVRegTop = 0;
 644
*645     /* 大域スコープの生成 */
 646     CurrentScope = (Scope *)0;
 647     NewScope((Offset)0);
 648     GlobalScope = CurrentScope;
 649
*650     /* 初期化コードを生成する */
*651     PutInst(MVM_ADD, MVM_FP, MVM_SP, MVM_R0); /* フレームポインタの初期化 */
*652     l1 = CurrentPC;
*653     PutInst(MVM_LDI, MVM_GP, 0, 0); /* グローバルポインタの初期化 */
*654     obj = UseObject("main");
*655     l2 = CurrentPC;
*656     PutInst(MVM_CALL, obj->reg, 0, obj->offset); /* main の呼び出し */
*657     obj->offset = l2;
*658     PutInst(MVM_EXIT, MVM_R1, 0, 0); /* プログラムの終了命令 */
 659
*660     /* 大域変数の宣言 <VarDef>、関数定義 <FunDef> の処理 */
 661     while (token == mc_int || token == mc_ident) {
 662         if (token == mc_int)
 663             VarDef();
 664         else
 665             FunDef();
 666     }
 667
*668     /* グローバルポインタの初期値の決定 */
*669     FixInst(l1, CurrentPC);
 670
*671     /* 大域スコープの廃棄 */
 672     ExitScope();
 673 }
 674
 675 void Parse(void)
 676 {
```

```
 677     GetToken();
 678     Prog();
 679     if (token != mc_eof)
 680         Error("変数宣言や関数定義として認識できません");
 681 }
 682
 683 int main (int argc, char *argv[])
 684 {
 685     if (argc < 2) {
 686         fprintf(stderr, "ソースファイルが指定されていません\n");
 687         exit(1);
 688     }
 689     if (OpenFile(argv[1]) != 0) {
 690         fprintf(stderr, "ソースファイルを読み込むことができません\n");
 691         exit(1);
 692     }
 693     Parse();
 694     CloseFile();
*695     if (WriteObjectFile("mcc.out") != 0) {
*696         fprintf(stderr, "オブジェクトファイルを作ることができません\n");
*697         exit(1);
*698     }
 699     return 0;
 700 }
 701
```

12.4 情報処理実習室の Linux 環境での実行

瀬田学舎の情報処理実習室の Linux 環境で、この Minimum C コンパイラを実際に使ってみることができます。次のようにして、必要なファイルを自分でコピーし、コンパイルして下さい。

```
t150000@s01cd0542-160:~$ mkdir MCC
t150000@s01cd0542-160:~$ cd MCC
t150000@s01cd0542-160:~/MCC$ cp /roes/sample/nakano/MCC/* .
t150000@s01cd0542-160:~/MCC$ make
```

`~/MCC` に `mcc` と `mvm` という 2 つの実行可能プログラムができあがります。`mcc` はこの講義で紹介した `scan.h`、`scan.c`、`gen.h`、`gen.c`、`parse.c` をコンパイル、リンクして作った Minimum C コンパイラです。

```
t150000@s01cd0542-160:~/MCC$ ./mcc gcd.mc
```

のように実行すると、引数として与えた `gcd.mc` という (Minimum C で書かれた) ソースファイルをコンパイルして MVM の機械語を生成し、`mcc.out` という名前のオブジェクトファイルに格納します。一方、`mvm` は MVM の機械語で書かれたオブジェクトプログラムを実行するプログラムです。

```
t150000@s01cd0542-160:~/MCC$ ./mvm mcc.out
```

のように実行します。`./mvm -d mcc.out` のように `-d` オプションを付けて実行すると `mcc.out` 中の機械語命令をニーモニックで表示してくれます。また `./mvm -t mcc.out` のように `-t` オプションを付けて実行すると MVM のレジスタやスタックの内容を表示しながら、1 ステップずつ `mcc.out` を実行させることができます。

記号処理・第 12 回・終わり