18 Constructing a Compiler Level 2

Temporary Variable Re-use

In `cg_add()` and `cg_mul()`, insert

```
    if symbol[leftindex].startswith('.t'):
       tempcount -= 1
    if symbol[rightindex].startswith('.t'):
       tempcount -= 1
```

In `cg_assign()`, insert

```
    if symbol[rightindex].startswith('.t')
       tempcount -= 1
```

In `cg_print()` and `cg_neg()`, insert

```
    if symbol[index].startswith('.t'):
       tempcount -= 1
```

Constant Folding

```
x = 2 + 3
```

```
 1 ldr r0, =.i2          @ get address of .i2
 2 ldr r0, [r0]          @ load 2
 3 ldr r1, =.i3          @ get address of .i3
 4 ldr r1, [r1]          @ load 3
 5 add r0, r0, r1        @ add 2 and 3
 6 ldr r1, =.t0          @ get address of .t0
 7 str r0, [r1]          @ store the sum in .t0
 8 ldr r0,  =.t0         @ get address of .t0
 9 ldr r0, [r0]          @ get the sum in .t0
10 ldr r1, =x            @ get address of x
11 str r0, [r1]          @ store the sum in x
```

```
 1 ldr r0, =.i5          @ get address of .i5
 2 ldr r0, [r0]          @ load 5
 3 ldr r1, =x            @ get address of x
 4 str r0, [r1]          @ store 5 in x
```

```
    symbol[leftindex].startswith('.i')

    result = int(value[leftindex]) + int(value[rightindex])

    result = int(value[leftindex]) * int(value[rightindex])

    if result >= 0:
       return enter('.i'+ str(result), str(result), False)
    else:
       return enter('.i_'+ str(-result), str(result), False)
```

```
.i2:        .word    2
.i3:        .word    3
.i5:        .word    5
```

```
    ldr r0, =.i5        @ get address of .i5
```

```
1 def enter(s, v, w):
2 if s in symbol:                # s already in symbol?
3     return symbol.index(s)     # then return its index
4 # add s, v, w to symbol, value, and needword, respectively
5 index = len(symbol)            # get index of next slot
6 symbol.append(s)               # append s to next slot in symbol
7 value.append(v)                # append v to next slot in value
8 needword.append(w)             # append w to next slot in needword
9 return index
```

```
elif token.category == UNSIGNEDINT:
   if sign == 1:
      index  = enter('.i' + token.lexeme, token.lexeme, False)
   else:
      index  = enter('.i_' + token.lexeme, '-' + token.lexeme, False)
   advance()
   return index

outfile.write('          ldr r1, =' + symbol[index] + '\n')

needword[index] = True
```

Putting Constants in the .text Segment

```
ldr    r0, .i3          @ load value of .i3

ldr    r0, =.i3         @ get address of .i3
ldr    r0, [r0]         @ load value of .i3
```

Register Allocation

```
 1 def enter(s, v, w):
 2    global nextreg
 3    if s in symbol:     # check if s already in symbol
 4       return symbol.index(s)
 5    # if s is not a variable or reg not available, then don’t allocate
 6    if s.startswith('.i') or nextreg > 12:
 7       loc.append(None)
 8    else:               # allocate register
 9       loc.append('r' + str(nextreg))
10       nextreg += 1
11
12    # add s, v, and w to symbol, value, and needword, and return index
13    index = len(symbol) # get index of next slot
14    symbol.append(s)    # append s to next slot in symbol
15    value.append(v)     # append v to next slot in value
16    needword.append(w)  # append w to next slot in needword
17    return index        # return index of new entry
```

Figure 18.1

```
symbol = []      # list of variable names
value = []       # value of corresponding symbol
needword = []    # indicates if .word directive is needed
loc = []         # None or 'r' concatenated with reg number
```

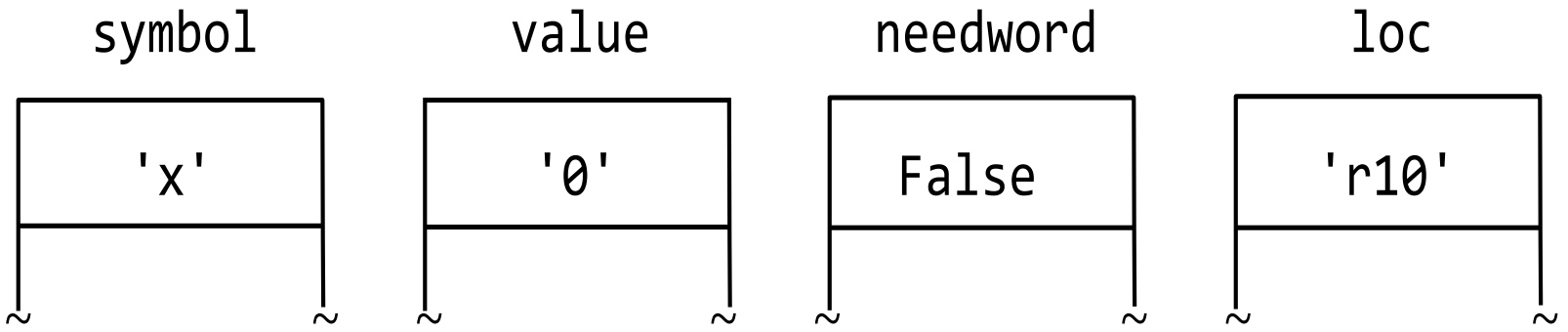

Figure 18.2

```
x = y
```

```
    mov     r10, r7
```

```
    ldr     r0, =x     @ get address of x
    ldr     r0, [r0]   @ get x
    ldr     r1, =y     @ get address of y
    str     r0, [r1]   @ store x in y
```

```
 1 def cg_assign(leftindex, rightindex):
 2    global tempcount
 3
 4    if symbol[rightindex].startswith('.t'):
 5       tempcount -= 1                    # free temporary variable
 6
 7    # determine what is on left and right sides
 8    if symbol[rightindex].startswith('.i'):
 9        constonright = True
10    else:
11        constonright = False
12    if loc[rightindex] == None:
13       regonright = False
14    else:
15       regonright = True
16       rightreg = loc[rightindex]
17    if loc[leftindex] == None:
18      regonleft = False
19    else:
20       regonleft = True
21    leftreg = loc[leftindex
```

```
23    # handle each case
24    if regonleft and regonright: # case 1: left and right are regs
25       outfile.write('          mov ' + leftreg + ', ' + rightreg + '\n')
26
27    elif regonleft and constonright: # case 2: left reg, right const
28       outfile.write('          ldr ' + leftreg + ', ' +
29                      symbol[rightindex] + '\n')
30       needword[rightindex] = True
31
32    elif regonleft and not constonright: # case 3: left reg, right var
33       outfile.write('          ldr r0, =' + symbol[rightindex] + '\n')
34       outfile.write('          ldr ' + leftreg + ', [r0]\n')
35       needword[rightindex] = True
36
37    elif regonright:            # case 4: left in mem, right reg
38       outfile.write('          ldr r0, =' + symbol[leftindex] + '\n')
39       outfile.write('          str ' + rightreg + ', [r0]\n')
40       needword[leftindex] = True
41
42    elif constonright:          # case 5: left in mem, right const in mem
43       outfile.write('          ldr r0, ' + symbol[rightindex] + '\n')
44       outfile.write('          ldr r1, =' + symbol[leftindex] + '\n')
45       outfile.write('          str r0, [r1]\n')
46       needword[rightindex] = True
47       needword[leftindex] = True
48
49    else:                       # case 6: left in mem, right var in mem
50       outfile.write('          ldr r0, =' + symbol[rightindex] + '\n')
51       outfile.write('          ldr r0, [r0]\n')
52       outfile.write('          ldr r1, =' + symbol[leftindex] + '\n')
53       outfile.write('          str r0, [r1]\n')
54       needword[rightindex] = True
55       needword[leftindex] = True
```

Figure 18.3

```
 1 def cg_neg(index):
 2    if symbol[index].startswith('.t'):
 3       tempcount -= 1
 4    if loc[index] == None:                        # not in a register?
 5       if symbol[index].startswith('.i'):         # constant?
 6          outfile.write('          ldr r0, ' + symbol[index] + '\n')
 7       else:                                      # variable
 8          outfile.write('          ldr r0, =' + symbol[index] + '\n')
 9          outfile.write('          ldr r0, [r0]\n')
10       needword[index] = True
11    else:                                         # in a register
12       outfile.write('          mov r0, ' + loc[index] + '\n')
13    outfile.write('          neg r0, r0\n')
14    tempindex = cg_gettemp()
15    if loc[tempindex] == None:                    # temp not in a register?
16       outfile.write('          ldr r1, =' + symbol[tempindex] + '\n')
17       outfile.write('          str r0, [r1]\n')
18       needword[tempindex] = True
19    else:                                         # temp in a register
20       outfile.write('          mov ' + loc[tempindex] + ', r0\n')
21    return tempindex
```

Unoptimized version

```
 1 @ Mon Feb 12 21:03:17 2018                    YOUR NAME HERE
 2 @ Compiler    = c1.py
 3 @ Input file  = c1.in
 4 @ Output file = c1.s
 5 @---------------------------------------- Assembler code
 6           .global main
 7           .text
 8 main:
 9           push {lr}
10
11 @ print(-59 + 20*3)
12           ldr r0, =.i20
13           ldr r0, [r0]
14           ldr r1, =.i3
15           ldr r1, [r1]
16           mul r0, r1, r0
17           ldr r1, =.t0
18           str r0, [r1]
19           ldr r0, =.i_59
20           ldr r0, [r0]
21           ldr r1, =.t0
22           ldr r1, [r1]
23           add r0, r0, r1
24           ldr r1, =.t1
25           str r0, [r1]
26           ldr r0, =.fmt0
27           ldr r1, =.t1
28           ldr r1, [r1]
29           bl printf
30
31 @ a = 2
32           ldr r0,  =.i2
33           ldr r0, [r0]
34           ldr r1, =a
35           str r0, [r1]
36
37 @ bb_1 = -(a) + 12
38           ldr r0, =a
39           ldr r0, [r0]
40           neg r0, r0
41           ldr r1, =.t2
42           str r0, [r1]
43           ldr r0, =.t2
44           ldr r0, [r0]
45           ldr r1, =.i12
46           ldr r1, [r1]
```

```
47          add r0, r0, r1
48          ldr r1, =.t3
49          str r0, [r1]
50          ldr r0,  =.t3
51          ldr r0, [r0]
52          ldr r1, =bb_1
53          str r0, [r1]
54
55 @ print(a*bb_1 + a*3*(-1 + -1 + -1))
56          ldr r0, =a
57          ldr r0, [r0]
58          ldr r1, =bb_1
59          ldr r1, [r1]
60          mul r0, r1, r0
61          ldr r1, =.t4
62          str r0, [r1]
63          ldr r0, =a
64          ldr r0, [r0]
65          ldr r1, =.i3
66          ldr r1, [r1]
67          mul r0, r1, r0
68          ldr r1, =.t5
69          str r0, [r1]
70          ldr r0, =.i_1
71          ldr r0, [r0]
72          ldr r1, =.i_1
73          ldr r1, [r1]
74          add r0, r0, r1
75          ldr r1, =.t6
76          str r0, [r1]
77          ldr r0, =.t6
78          ldr r0, [r0]
79          ldr r1, =.i_1
80          ldr r1, [r1]
81          add r0, r0, r1
82          ldr r1, =.t7
83          str r0, [r1]
84          ldr r0, =.t5
85          ldr r0, [r0]
86          ldr r1, =.t7
87          ldr r1, [r1]
88          mul r0, r1, r0
89          ldr r1, =.t8
90          str r0, [r1]
91          ldr r0, =.t4
92          ldr r0, [r0]
93          ldr r1, =.t8
94      ldr r1, [r1
```

```
 95          add r0, r0, r1
 96          ldr r1, =.t9
 97          str r0, [r1]
 98          ldr r0, =.fmt0
 99          ldr r1, =.t9
100          ldr r1, [r1]
101          bl printf
102
103          mov r0, #0
104          pop {pc}
105
106          .data
107 .fmt0:    .asciz "%d\n"
108 .i_59:    .word -59
109 .i20:     .word 20
110 .i3:      .word 3
111 .t0:      .word 0
112 .t1:      .word 0
113 a:        .word 0
114 .i2:      .word 2
115 bb_1:     .word 0
116 .t2:      .word 0
117 .i12:     .word 12
118 .t3:      .word 0
119 .t4:      .word 0
120 .t5:      .word 0
121 .i_1:     .word -1
122 .t6:      .word 0
123 .t7:      .word 0
124 .t8:      .word 0
125 .t9:      .word 0
```

Figure 18.5

Optimized version

```
@ Mon Feb 12 21:13:16 2018                  YOUR NAME HERE
@ Compiler    = c2.py
@ Input file  = c2.in
@ Output file = c2.s
@---------------------------------------- Assembler code
          .global main
          .text
main:
          push {lr}

@ print(-59 + 20*3)
          ldr r0, =.fmt0
          ldr r1, .i1             # -59 + 20*3 folded into 1
          bl printf

@ a = 2
          ldr r4, .i2             # a in r4

@ bb_1 = -(a) + 12
          mov r0, r4
          neg r0, r0
          mov r6, r0              # .t0 in r6
          ldr r1, .i12
          add r6, r6, r1
          mov r5, r6              # bb_1 in r5

@ print(a*bb_1 + a*3*(-1 + -1 + -1))
          mul r6, r4, r5
          ldr r1, .i3
          mul r7, r4, r1          # .t1 in r7
          ldr r1, .i_3            # 3*(-1 + -1 + -1) folded into -3
          mul r7, r7, r1
          add r6, r6, r7
          ldr r0, =.fmt0
          mov r1, r6
          bl printf

          mov r0, #0
          pop {pc}
.fmt0:    .asciz "%d\n"           # constants in .text segment
.i3:      .word 3
.i1:      .word 1
.i2:      .word 2
.i12:     .word 12
.i_3:     .word -3
```