LABEL TRANSLATION (LINKING*)

Introduction

Labels are generated while translating the high level program constructs such as if-then-else, while-do and function calls into target machine code as an intermediate step in code generation. Labels are needed because the target address of a JUMP instruction may not be known at the time of generating code. However these labels must be replaced with target addresses in the final target code. Thus the compiler designer needs to write a program to replace all the labels in the program with the correct memory addresses eventually. This conversion is called Label translation.

The input to the label translator program is the machine code with labels (generated by codeGen()). The output is the target machine code without the labels (replaced with corresponding addresses).

The main idea behind label translation is to execute the following two steps.

1. The input (machine code with labels) is parsed and a table in which the labels are mapped with their corresponding addresses is created.
2. Parse the input again and replace all the labels with their corresponding addresses by looking into the table created above.

*Note: In real systems, the compiler generates an object file which contains labels for functions as well as variables. The object file is processed by a separate software module called the linker which converts the object file into an excutable file. The advantage of this scheme is that a huge program could be divided into different stand alone source modules, each compiled seperately and finally linked together. A change in one module does not require recompilation of other modules and only the linking phase needs to

be run again. In this project, we keep the linking task as a part of the compilation itself as ExpL does not allow a program to be compiled into several object modules.

In the ExpL project, variable addresses are resolved before reaching the Label translation phase and only symbolic references to functions are left to be replaced with addresses in this phase.

Illustration

The above procedure is demonstrated with the help of a program snippet.

```
1  decl
2          int fact(int n);
3  enddecl
4
5  int fact(int n)
6  {
7          decl
8                  int f;
9          enddecl
10         begin
11                 if(n<=1) then
12                         f=1;
13                 else
14                         f=n*fact(
15                 endif;
16                 return f;
17         end
18 }
19
20 int main()
21 {
22         decl
23                 int n,m,res;
24         enddecl
25         begin
26                 read(n);
27                 while( n >= 1 ) d
28                         read(m);
29                         res = fac
30                         write(res
31                         n = n-1;
```

```
32                  endwhile;
33                  return 0;
34          end
35  }
```

Factorial.c hosted with ❤ by **GitHub** **view raw**

The code generated after the codeGen() phase is shown in the figure 1 (It consists of labels).

Figure 1

```
  1  2048 : 0
  1  2049 : 2056
  1  2050 : 0
  1  2051 : 0
  1  2052 : 0
  1  2053 : 0
  1  2054 : 0
  1  2055 : 0
  5  2056 : MOV SP,409
  6  2058 : MOV BP,409
  7  2060 : PUSH R0
  8  2062 : CALL MAIN
  9  2064 : INT 10
 10         F0:
 11  2066 : PUSH BP
 12  2068 : MOV BP,SP
 13  2070 : PUSH R0
 14  2072 : MOV R1,BP
 15  2074 : MOV R2,2
 16  2076 : SUB R1,R2
 17  2078 : MOV R2,1
 18  2080 : SUB R1,R2
 19  2082 : MOV R0,[R1
 20  2084 : MOV R1,1
 21  2086 : LE R0,R1
 22  2088 : JZ R0,L0
 23  2090 : MOV R0,1
 24  2092 : MOV R2,BP
 25  2094 : MOV R1,1
 26  2096 : ADD R2,R1
 27  2098 : MOV [R2],R
 28  2100 : JMP L1
 29         L0:
 30  2102 : MOV R1,BP
 31  2104 : MOV R2,2
```

```
32  2106 : SUB R1,R2
33  2108 : MOV R2,1
34  2110 : SUB R1,R2
35  2112 : MOV R0,[R1
36  2114 : PUSH R0
37  2116 : MOV R1,BP
38  2118 : MOV R2,2
39  2120 : SUB R1,R2
40  2122 : MOV R2,1
41  2124 : SUB R1,R2
42  2126 : MOV R0,[R1
43  2128 : MOV R1,1
44  2130 : SUB R0,R1
45  2132 : PUSH R0
46  2134 : PUSH R0
47  2136 : CALL F0
48  2138 : POP R0
49  2140 : POP R0
50  2142 : POP R0
51  2144 : MOV R1,3
52  2146 : MOV R2,SP
53  2148 : ADD R2,R1
54  2150 : MOV R1,[R2
55  2152 : MUL R0,R1
56  2154 : MOV R2,BP
57  2156 : MOV R1,1
58  2158 : ADD R2,R1
59  2160 : MOV [R2],R
60         L1:
61  2162 : MOV R1,BP
62  2164 : MOV R0,1
63  2166 : ADD R1,R0
64  2168 : MOV R0,[R1
65  2170 : MOV R1,BP
66  2172 : MOV R2,2
67  2174 : SUB R1,R2
68  2176 : MOV [R1],R
69  2178 : POP R0
70  2180 : MOV BP,[SP
71  2182 : POP R0
72  2184 : RET
73         MAIN:
74  2186 : PUSH BP
75  2188 : MOV BP,SP
76  2190 : PUSH R0
77  2192 : PUSH R0
78  2194 : PUSH R0
```

```
 79   2196 : MOV R1,BP
 80   2198 : MOV R0,1
 81   2200 : ADD R1,R0
 82   2202 : PUSH R0
 83   2204 : PUSH R1
 84   2206 : MOV R0,"Re
 85   2208 : PUSH R0
 86   2210 : MOV R0,-1
 87   2212 : PUSH R0
 88   2214 : PUSH R1
 89   2216 : PUSH R0
 90   2218 : PUSH R0
 91   2220 : CALL 0
 92   2222 : POP R0
 93   2224 : POP R0
 94   2226 : POP R0
 95   2228 : POP R0
 96   2230 : POP R0
 97   2232 : POP R0
 98   2234 : POP R0
 99          L2:
100   2236 : MOV R1,BP
101   2238 : MOV R0,1
102   2240 : ADD R1,R0
103   2242 : MOV R0,[R1
104   2244 : MOV R1,1
105   2246 : GE R0,R1
106   2248 : JZ R0,L3
107   2250 : MOV R1,BP
108   2252 : MOV R0,2
109   2254 : ADD R1,R0
110   2256 : PUSH R0
111   2258 : PUSH R1
112   2260 : MOV R0,"Re
113   2262 : PUSH R0
114   2264 : MOV R0,-1
115   2266 : PUSH R0
116   2268 : PUSH R1
117   2270 : PUSH R0
118   2272 : PUSH R0
119   2274 : CALL 0
120   2276 : POP R0
121   2278 : POP R0
122   2280 : POP R0
123   2282 : POP R0
124   2284 : POP R0
125   2286 : POP R0
```

```
126   2288 : POP R0
127   2290 : MOV R1,BP
128   2292 : MOV R0,2
129   2294 : ADD R1,R0
130   2296 : MOV R0,[R1
131   2298 : PUSH R0
132   2300 : PUSH R0
133   2302 : CALL F0
134   2304 : POP R0
135   2306 : POP R0
136   2308 : MOV R0,2
137   2310 : MOV R1,SP
138   2312 : ADD R1,R0
139   2314 : MOV R0,[R1
140   2316 : MOV R2,BP
141   2318 : MOV R1,3
142   2320 : ADD R2,R1
143   2322 : MOV [R2],R
144   2324 : MOV R1,BP
145   2326 : MOV R0,3
146   2328 : ADD R1,R0
147   2330 : MOV R0,[R1
148   2332 : MOV [2042]
149   2334 : PUSH R0
150   2336 : MOV R0,"Wr
151   2338 : PUSH R0
152   2340 : MOV R0,-2
153   2342 : PUSH R0
154   2344 : MOV R0,204
155   2346 : PUSH R0
156   2348 : PUSH R0
157   2350 : PUSH R0
158   2352 : CALL 0
159   2354 : POP R0
160   2356 : POP R0
161   2358 : POP R0
162   2360 : POP R0
163   2362 : POP R0
164   2364 : POP R0
165   2366 : MOV R1,BP
166   2368 : MOV R0,1
167   2370 : ADD R1,R0
168   2372 : MOV R0,[R1
169   2374 : MOV R1,1
170   2376 : SUB R0,R1
171   2378 : MOV R2,BP
172   2380 : MOV R1,1
```

```
173   2382 : ADD R2,R1
174   2384 : MOV [R2],R
175   2386 : JMP L2
176          L3:
177   2388 : MOV R0,0
178   2390 : MOV R1,BP
179   2392 : MOV R2,2
180   2394 : SUB R1,R2
181   2396 : MOV [R1],R
182   2398 : POP R0
183   2400 : POP R0
184   2402 : POP R0
185   2404 : MOV BP,[SP
186   2406 : POP R0
187   2408 : RET
```

Assembly Code Before Label Translation view raw
hosted with ❤ by **GitHub**

Figure 2

```
  1   2048 : 0
  1   2049 : 2056
  1   2050 : 0
  1   2051 : 0
  1   2052 : 0
  1   2053 : 0
  1   2054 : 0
  1   2055 : 0
  5   2056 : MOV SP,409
  6   2058 : MOV BP,409
  7   2060 : PUSH R0
  8   2062 : CALL 2186
  9   2064 : INT 10
 10   2066 : PUSH BP
 11   2068 : MOV BP,SP
 12   2070 : PUSH R0
 13   2072 : MOV R1,BP
 14   2074 : MOV R2,2
 15   2076 : SUB R1,R2
 16   2078 : MOV R2,1
 17   2080 : SUB R1,R2
 18   2082 : MOV R0,[R1
 19   2084 : MOV R1,1
 20   2086 : LE R0,R1
 21   2088 : JZ R0,2102
 22   2090 : MOV R0,1
```

```
23  2092 : MOV R2,BP
24  2094 : MOV R1,1
25  2096 : ADD R2,R1
26  2098 : MOV [R2],R
27  2100 : JMP 2162
28  2102 : MOV R1,BP
29  2104 : MOV R2,2
30  2106 : SUB R1,R2
31  2108 : MOV R2,1
32  2110 : SUB R1,R2
33  2112 : MOV R0,[R1
34  2114 : PUSH R0
35  2116 : MOV R1,BP
36  2118 : MOV R2,2
37  2120 : SUB R1,R2
38  2122 : MOV R2,1
39  2124 : SUB R1,R2
40  2126 : MOV R0,[R1
41  2128 : MOV R1,1
42  2130 : SUB R0,R1
43  2132 : PUSH R0
44  2134 : PUSH R0
45  2136 : CALL 2066
46  2138 : POP R0
47  2140 : POP R0
48  2142 : POP R0
49  2144 : MOV R1,3
50  2146 : MOV R2,SP
51  2148 : ADD R2,R1
52  2150 : MOV R1,[R2
53  2152 : MUL R0,R1
54  2154 : MOV R2,BP
55  2156 : MOV R1,1
56  2158 : ADD R2,R1
57  2160 : MOV [R2],R
58  2162 : MOV R1,BP
59  2164 : MOV R0,1
60  2166 : ADD R1,R0
61  2168 : MOV R0,[R1
62  2170 : MOV R1,BP
63  2172 : MOV R2,2
64  2174 : SUB R1,R2
65  2176 : MOV [R1],R
66  2178 : POP R0
67  2180 : MOV BP,[SP
68  2182 : POP R0
69  2184 : RET
```

```
 70   2186 : PUSH BP
 71   2188 : MOV BP,SP
 72   2190 : PUSH R0
 73   2192 : PUSH R0
 74   2194 : PUSH R0
 75   2196 : MOV R1,BP
 76   2198 : MOV R0,1
 77   2200 : ADD R1,R0
 78   2202 : PUSH R0
 79   2204 : PUSH R1
 80   2206 : MOV R0,"Re
 81   2208 : PUSH R0
 82   2210 : MOV R0,-1
 83   2212 : PUSH R0
 84   2214 : PUSH R1
 85   2216 : PUSH R0
 86   2218 : PUSH R0
 87   2220 : CALL 0
 88   2222 : POP R0
 89   2224 : POP R0
 90   2226 : POP R0
 91   2228 : POP R0
 92   2230 : POP R0
 93   2232 : POP R0
 94   2234 : POP R0
 95   2236 : MOV R1,BP
 96   2238 : MOV R0,1
 97   2240 : ADD R1,R0
 98   2242 : MOV R0,[R1
 99   2244 : MOV R1,1
100   2246 : GE R0,R1
101   2248 : JZ R0,2388
102   2250 : MOV R1,BP
103   2252 : MOV R0,2
104   2254 : ADD R1,R0
105   2256 : PUSH R0
106   2258 : PUSH R1
107   2260 : MOV R0,"Re
108   2262 : PUSH R0
109   2264 : MOV R0,-1
110   2266 : PUSH R0
111   2268 : PUSH R1
112   2270 : PUSH R0
113   2272 : PUSH R0
114   2274 : CALL 0
115   2276 : POP R0
116   2278 : POP R0
```

```
117  2280 : POP R0
118  2282 : POP R0
119  2284 : POP R0
120  2286 : POP R0
121  2288 : POP R0
122  2290 : MOV R1,BP
123  2292 : MOV R0,2
124  2294 : ADD R1,R0
125  2296 : MOV R0,[R1
126  2298 : PUSH R0
127  2300 : PUSH R0
128  2302 : CALL 2066
129  2304 : POP R0
130  2306 : POP R0
131  2308 : MOV R0,2
132  2310 : MOV R1,SP
133  2312 : ADD R1,R0
134  2314 : MOV R0,[R1
135  2316 : MOV R2,BP
136  2318 : MOV R1,3
137  2320 : ADD R2,R1
138  2322 : MOV [R2],R
139  2324 : MOV R1,BP
140  2326 : MOV R0,3
141  2328 : ADD R1,R0
142  2330 : MOV R0,[R1
143  2332 : MOV [2042]
144  2334 : PUSH R0
145  2336 : MOV R0,"Wr
146  2338 : PUSH R0
147  2340 : MOV R0,-2
148  2342 : PUSH R0
149  2344 : MOV R0,204
150  2346 : PUSH R0
151  2348 : PUSH R0
152  2350 : PUSH R0
153  2352 : CALL 0
154  2354 : POP R0
155  2356 : POP R0
156  2358 : POP R0
157  2360 : POP R0
158  2362 : POP R0
159  2364 : POP R0
160  2366 : MOV R1,BP
161  2368 : MOV R0,1
162  2370 : ADD R1,R0
163  2372 : MOV R0,[R1
```

```
164   2374 : MOV R1,1
165   2376 : SUB R0,R1
166   2378 : MOV R2,BP
167   2380 : MOV R1,1
168   2382 : ADD R2,R1
169   2384 : MOV [R2],R
170   2386 : JMP 2236
171   2388 : MOV R0,0
172   2390 : MOV R1,BP
173   2392 : MOV R2,2
174   2394 : SUB R1,R2
175   2396 : MOV [R1],R
176   2398 : POP R0
177   2400 : POP R0
178   2402 : POP R0
179   2404 : MOV BP,[SP
180   2406 : POP R0
181   2408 : RET
```

Assembly Code after Label Translation **view raw**
hosted with ❤ by **GitHub**

In the machine code generated after codegen section, labels occur in two different ways.

1. Label declarations in which labels are followed by semicolon (:). Examples : F0: is a label for the instruction at address 2066, L0: for 2102, L1: for 2162 and MAIN: for 2186 as in figure1

.

2. Instructions which contain labels in them are JMP, JZ, JNZ and CALL. Examples : JMP L1 at address 2100, JZ R0,L3 at address 2248, CALL MAIN at address 2062 etc., as in figure1.

Now, we will understand the label translation procedure with the help of an example label F0 from the figure1.
First F0: need to be recognized and the memory address of the label occurance need to be stored in a table called Label-Address Table. In this case, the address corresponding to F0: is 2066. We need to parse the entire machine code to identify all labels and store all (label, address) pairs in the Label-Address table. In this pass, we remove the label F0: as well as other labels from the program. The next task is to replace labels occuring in instructions with the addresses of the labels from the label-address table. Refering to the figure1, there are two instructions that use the label F0: CALL F0 occuring at address 2136 and CALL F0 occuring at address 2302. We will

replace the labels in these instructions with the address of F0 (2066) by looking up the label-address table. Thus, after translation, both these instructions will translate to CALL 2066 as shown in figure2.

The label translater program need to parse the entire machine code two times. In the first parse we identify all the label declarations and the memory addresses in the label-address table. The table constructed after the first parse of the above code is shown below.

LABEL-ADDRESS Table

LABEL	ADDRESS
F0	2066
L0	2102
L1	2162
MAIN	2186
L2	2236
L3	2388

In the second parse, we remove the label declarations and replace the labels in instructions like JUMP, CALL etc. with the corresponding memory address from the label-address table.

The target code after the label translation is shown in the figure 2.

Implementing Label Translation

The above two pass translation process can be implemented using a single Lex program.The following functions provided by Lex can be useful while implementing the parser :

1. yyless(k) : Returns all but the first n characters of the current token back to the input stream, where they will be rescanned when the scanner looks for the next match. yytext and yyleng are adjusted appropriately (e.g., yyleng will now be equal to n).
For example, on the input "foobar" the following will write out "foobarbar":

```
%%
foobar ECHO; yyless(3);
[a-z]+ ECHO;
```

An argument of 0 to yyless will cause the entire current input string to be scanned again. Unless you've changed how the scanner will subsequently process its input, this will result in an endless loop. Note that yyless is a macro and can only be used in the flex input file, not from other source files.
2. yytext+k : ignores the first k characters in yytext.