EXPI

Home About Help Roadmap Documentation

LABEL TRANSLATION (LINKING*)

➤ Introduction
➤ Illustration

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.

```
decl
        int fact(int n);
enddecl

int fact(int n)
{
        decl
                int f;
        enddecl
        begin
                if(n<=1) then
                        f=1;
                else
                        f=n*fact(
                endif;
                return f;
        end
}

int main()
{
        decl
                int n,m,res;
        enddecl
        begin
                read(n);
                while( n >= 1 ) c
                        read(m);
                        res = fac
                        write(res
                        n = n-1;
                endwhile;
                return 0;
        end
}
```

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).

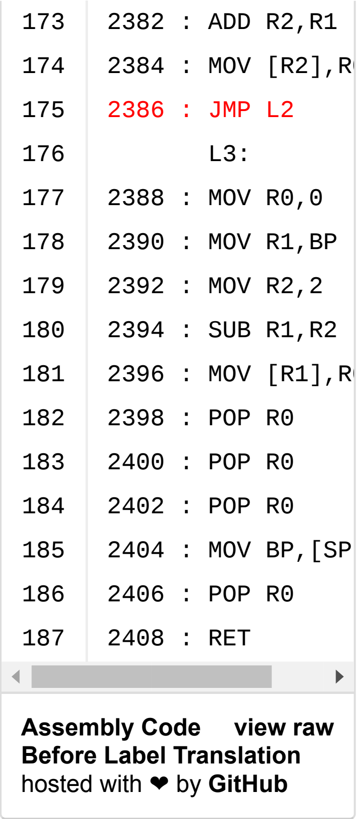
Figure 1

```
 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
```

```
2106 : SUB R1,R2
2108 : MOV R2,1
2110 : SUB R1,R2
2112 : MOV R0,[R1
2114 : PUSH R0
2116 : MOV R1,BP
2118 : MOV R2,2
2120 : SUB R1,R2
2122 : MOV R2,1
2124 : SUB R1,R2
2126 : MOV R0,[R1
2128 : MOV R1,1
2130 : SUB R0,R1
2132 : PUSH R0
2134 : PUSH R0
2136 : CALL F0
2138 : POP R0
2140 : POP R0
2142 : POP R0
2144 : MOV R1,3
2146 : MOV R2,SP
2148 : ADD R2,R1
2150 : MOV R1,[R2
2152 : MUL R0,R1
2154 : MOV R2,BP
2156 : MOV R1,1
2158 : ADD R2,R1
2160 : MOV [R2],R
       L1:
2162 : MOV R1,BP
2164 : MOV R0,1
2166 : ADD R1,R0
2168 : MOV R0,[R1
2170 : MOV R1,BP
2172 : MOV R2,2
2174 : SUB R1,R2
2176 : MOV [R1],R
2178 : POP R0
2180 : MOV BP,[SP
2182 : POP R0
2184 : RET
       MAIN:
2186 : PUSH BP
2188 : MOV BP,SP
2190 : PUSH R0
2192 : PUSH R0
2194 : PUSH R0
```

```
2196 : MOV R1,BP
2198 : MOV R0,1
2200 : ADD R1,R0
2202 : PUSH R0
2204 : PUSH R1
2206 : MOV R0,"Re
2208 : PUSH R0
2210 : MOV R0,-1
2212 : PUSH R0
2214 : PUSH R1
2216 : PUSH R0
2218 : PUSH R0
2220 : CALL 0
2222 : POP R0
2224 : POP R0
2226 : POP R0
2228 : POP R0
2230 : POP R0
2232 : POP R0
2234 : POP R0
       L2:
2236 : MOV R1,BP
2238 : MOV R0,1
2240 : ADD R1,R0
2242 : MOV R0,[R1
2244 : MOV R1,1
2246 : GE R0,R1
2248 : JZ R0,L3
2250 : MOV R1,BP
2252 : MOV R0,2
2254 : ADD R1,R0
2256 : PUSH R0
2258 : PUSH R1
2260 : MOV R0,"Re
2262 : PUSH R0
2264 : MOV R0,-1
2266 : PUSH R0
2268 : PUSH R1
2270 : PUSH R0
2272 : PUSH R0
2274 : CALL 0
2276 : POP R0
2278 : POP R0
2280 : POP R0
2282 : POP R0
2284 : POP R0
2286 : POP R0
```

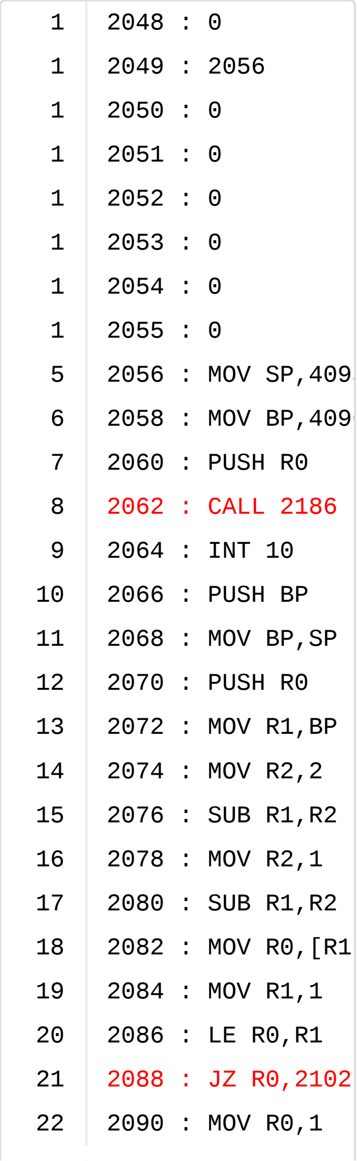
```
2288 : POP R0
2290 : MOV R1,BP
2292 : MOV R0,2
2294 : ADD R1,R0
2296 : MOV R0,[R1
2298 : PUSH R0
2300 : PUSH R0
2302 : CALL F0
2304 : POP R0
2306 : POP R0
2308 : MOV R0,2
2310 : MOV R1,SP
2312 : ADD R1,R0
2314 : MOV R0,[R1
2316 : MOV R2,BP
2318 : MOV R1,3
2320 : ADD R2,R1
2322 : MOV [R2],R
2324 : MOV R1,BP
2326 : MOV R0,3
2328 : ADD R1,R0
2330 : MOV R0,[R1
2332 : MOV [2042]
2334 : PUSH R0
2336 : MOV R0,"Wr
2338 : PUSH R0
2340 : MOV R0,-2
2342 : PUSH R0
2344 : MOV R0,204
2346 : PUSH R0
2348 : PUSH R0
2350 : PUSH R0
2352 : CALL 0
2354 : POP R0
2356 : POP R0
2358 : POP R0
2360 : POP R0
2362 : POP R0
2364 : POP R0
2366 : MOV R1,BP
2368 : MOV R0,1
2370 : ADD R1,R0
2372 : MOV R0,[R1
2374 : MOV R1,1
2376 : SUB R0,R1
2378 : MOV R2,BP
2380 : MOV R1,1
```



```
2382 : ADD R2,R1
2384 : MOV [R2],R
2386 : JMP L2
       L3:
2388 : MOV R0,0
2390 : MOV R1,BP
2392 : MOV R2,2
2394 : SUB R1,R2
2396 : MOV [R1],R
2398 : POP R0
2400 : POP R0
2402 : POP R0
2404 : MOV BP,[SP
2406 : POP R0
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
```

```
2092 : MOV R2,BP
2094 : MOV R1,1
2096 : ADD R2,R1
2098 : MOV [R2],R
2100 : JMP 2162
2102 : MOV R1,BP
2104 : MOV R2,2
2106 : SUB R1,R2
2108 : MOV R2,1
2110 : SUB R1,R2
2112 : MOV R0,[R1
2114 : PUSH R0
2116 : MOV R1,BP
2118 : MOV R2,2
2120 : SUB R1,R2
2122 : MOV R2,1
2124 : SUB R1,R2
2126 : MOV R0,[R1
2128 : MOV R1,1
2130 : SUB R0,R1
2132 : PUSH R0
2134 : PUSH R0
2136 : CALL 2066
2138 : POP R0
2140 : POP R0
2142 : POP R0
2144 : MOV R1,3
2146 : MOV R2,SP
2148 : ADD R2,R1
2150 : MOV R1,[R2
2152 : MUL R0,R1
2154 : MOV R2,BP
2156 : MOV R1,1
2158 : ADD R2,R1
2160 : MOV [R2],R
2162 : MOV R1,BP
2164 : MOV R0,1
2166 : ADD R1,R0
2168 : MOV R0,[R1
2170 : MOV R1,BP
2172 : MOV R2,2
2174 : SUB R1,R2
2176 : MOV [R1],R
2178 : POP R0
2180 : MOV BP,[SP
2182 : POP R0
2184 : RET
```

```
2186 : PUSH BP
2188 : MOV BP,SP
2190 : PUSH R0
2192 : PUSH R0
2194 : PUSH R0
2196 : MOV R1,BP
2198 : MOV R0,1
2200 : ADD R1,R0
2202 : PUSH R0
2204 : PUSH R1
2206 : MOV R0,"Re
2208 : PUSH R0
2210 : MOV R0,-1
2212 : PUSH R0
2214 : PUSH R1
2216 : PUSH R0
2218 : PUSH R0
2220 : CALL 0
2222 : POP R0
2224 : POP R0
2226 : POP R0
2228 : POP R0
2230 : POP R0
2232 : POP R0
2234 : POP R0
2236 : MOV R1,BP
2238 : MOV R0,1
2240 : ADD R1,R0
2242 : MOV R0,[R1
2244 : MOV R1,1
2246 : GE R0,R1
2248 : JZ R0,2388
2250 : MOV R1,BP
2252 : MOV R0,2
2254 : ADD R1,R0
2256 : PUSH R0
2258 : PUSH R1
2260 : MOV R0,"Re
2262 : PUSH R0
2264 : MOV R0,-1
2266 : PUSH R0
2268 : PUSH R1
2270 : PUSH R0
2272 : PUSH R0
2274 : CALL 0
2276 : POP R0
2278 : POP R0
```

```
2280 : POP R0
2282 : POP R0
2284 : POP R0
2286 : POP R0
2288 : POP R0
2290 : MOV R1,BP
2292 : MOV R0,2
2294 : ADD R1,R0
2296 : MOV R0,[R1
2298 : PUSH R0
2300 : PUSH R0
2302 : CALL 2066
2304 : POP R0
2306 : POP R0
2308 : MOV R0,2
2310 : MOV R1,SP
2312 : ADD R1,R0
2314 : MOV R0,[R1
2316 : MOV R2,BP
2318 : MOV R1,3
2320 : ADD R2,R1
2322 : MOV [R2],R
2324 : MOV R1,BP
2326 : MOV R0,3
2328 : ADD R1,R0
2330 : MOV R0,[R1
2332 : MOV [2042]
2334 : PUSH R0
2336 : MOV R0,"Wr
2338 : PUSH R0
2340 : MOV R0,-2
2342 : PUSH R0
2344 : MOV R0,204
2346 : PUSH R0
2348 : PUSH R0
2350 : PUSH R0
2352 : CALL 0
2354 : POP R0
2356 : POP R0
2358 : POP R0
2360 : POP R0
2362 : POP R0
2364 : POP R0
2366 : MOV R1,BP
2368 : MOV R0,1
2370 : ADD R1,R0
2372 : MOV R0,[R1
```

```
2374 : MOV R1,1
2376 : SUB R0,R1
2378 : MOV R2,BP
2380 : MOV R1,1
2382 : ADD R2,R1
2384 : MOV [R2],R
2386 : JMP 2236
2388 : MOV R0,0
2390 : MOV R1,BP
2392 : MOV R2,2
2394 : SUB R1,R2
2396 : MOV [R1],R
2398 : POP R0
2400 : POP R0
2402 : POP R0
2404 : MOV BP,[SP
2406 : POP R0
2408 : RET
```

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

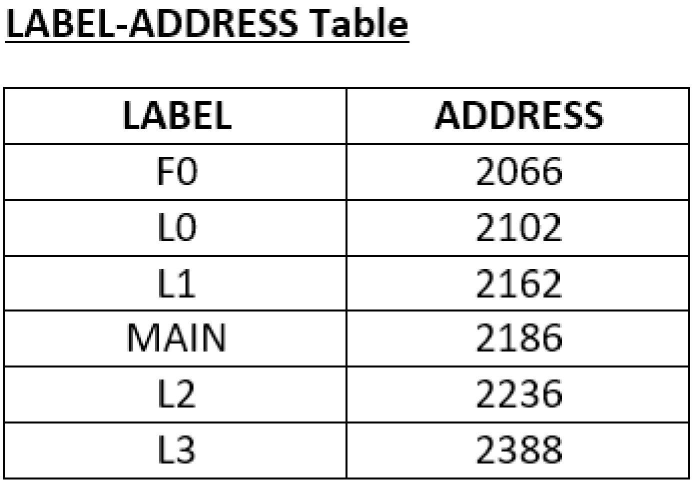
In the machine code generated after codegen section, labels occur in two different ways.

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.

Contributed By : Thallam Sai Sree Datta, N Ruthvik