High-Level Information Functions for GCC (v0.3)

Bob Glamm

19 Mar 1996

1 Introduction

This document describes in detail programs and functions available to manipulate and use the High-Level Information (HLI) file generated by SUIF. This information can be used while performing optimizations on RTL in GCC. The goal of these functions is to provide the ability to perform high-level optimizations in GCC that aren't available at the RTL level.

2 Preprocessor

The preprocessor takes a raw HLI file output from SUIF and generates an indexed file for use with the GCC functions. The format of this postprocessed file is shown in Figure 1.

3 Core Functions

Phase I will consist of the implementation of basic HLI functions. These functions will

1) read the HLI file,

2) match up the HLI "items" (variables, instructions, etc.) with RTL "INSNs" (instructions),

3) return equivalence classes, loop carried data dependencies, and REF/MOD information for function calls or subregions between two RTL "INSNs."

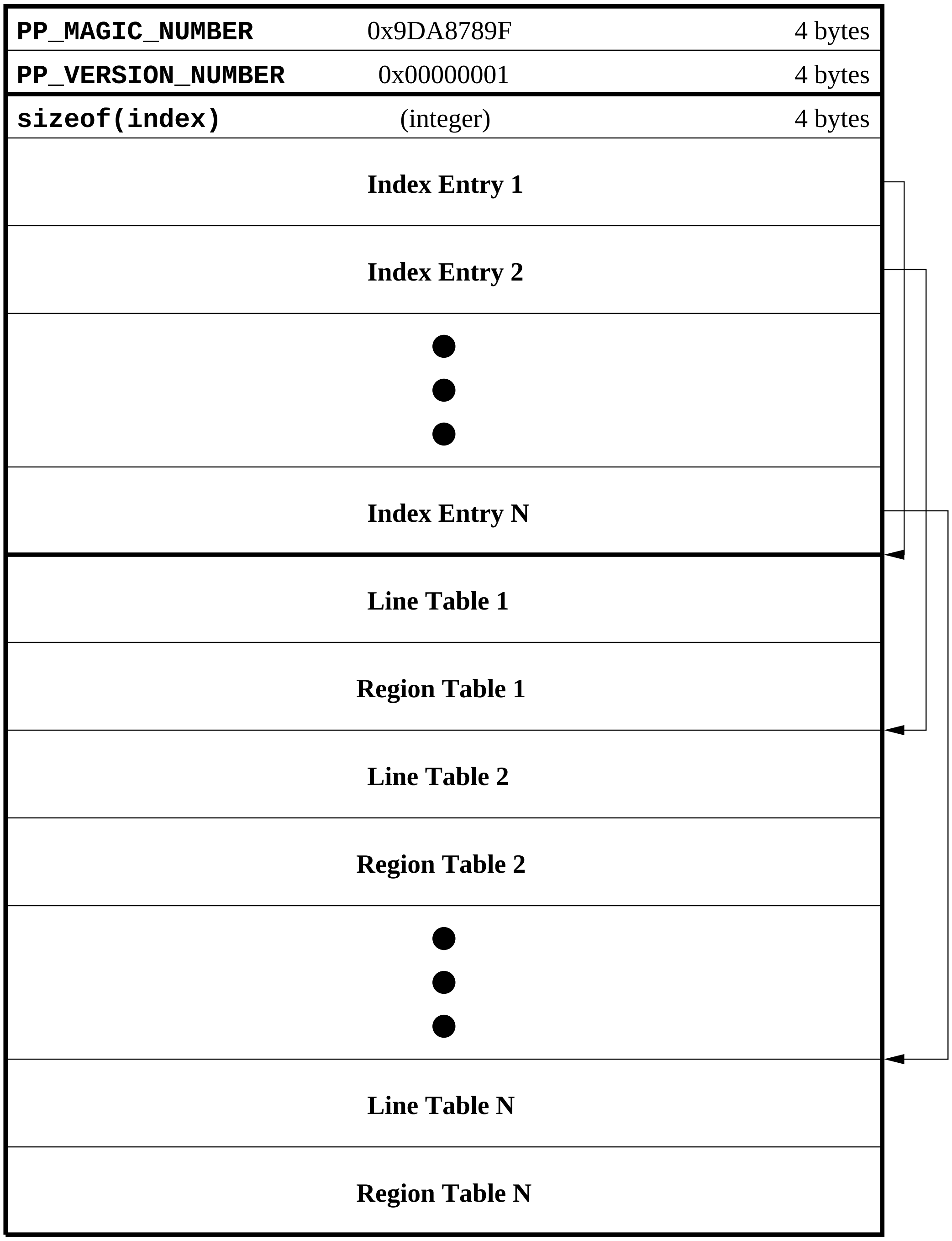

Figure 1: Postprocessed HLI file format

Length of function name	(integer)	4 bytes
Function Name	(string, 0-padded to word boundary)	N bytes
Offset to Line Table	(integer)	4 bytes
# of lines in Line Table	(integer)	4 bytes
sizeof(Line Table)	(integer)	4 bytes
# of regions in Region Table	(integer)	4 bytes
sizeof(Region Table)	(integer)	4 bytes

Figure 2: Index format (from Figure 1)

Here is a list of the functions used to access the high-level information.

- int read_hli(char **fn_name*);

This function will read the section of the HLI file corresponding to the GCC function currently being compiled. The HLI file will be read to fill in the HLI structures (see hli_rtl.h for more details). In addition, a mapping between RTL INSNs and HLI items will be constructed. This function will return 0 if successful and 1 on any error reading the file.

Note: this function will automatically be called for every function read in; it is not necessary for the user to call this directly.

- int get_item_id(rtx *insn*);

Given an RTL INSN, this function will return the integer ID of the corresponding item in the HLI data structure.

- int get_item_type(int *item_id*);

Given the ID of an HLI item, this function returns the type of that item (see hli_rtl.h).

- EQ_Type check_eq_acc(rtx *insn1*, rtx *insn2*);

This function returns the equivalence type between two instructions (see hli_rtl.h). This function will work across regions and subregions. Equivalence checks between different item ID numbers can be done via the RTL-to-item_id and item_id-to-RTL conversion functions listed above.

- DD_Type check_lcdd(rtx *insn1*, rtx *insn2*, int **distance*);

This function returns the loop-carried data dependency between two RTL INSNs (see hli_rtl.h). Again, this function works across loop regions. The data dependency returned is the dependency of the innermost loop that encloses both RTL INSNs. The dependency distance is returned in the third parameter of the function. If there is no loop carried data dependency, DT_None is returned and distance is set to 0. If the INSNs are not in the same loop, DT_Error will be returned.

- boolean check_call_ref(rtx *insn*, rtx *call_insn*);
- boolean check_call_mod(rtx *insn*, rtx *call_insn*);

This function returns true if the given RTL INSN belongs to an equivalence class that may be referenced/modified in the function call (second argument). A -1 is returned if the second argument is not a function call INSN.

- boolean check_subreg_ref(rtx *insn*, int *subregion_id*);
- boolean check_subreg_mod(rtx *insn*, int *subregion_id*);

This function returns true if the given RTL INSN belongs to an equivalence class that may be referenced/modified in the subregion (the subregion is the second argument, identified by item ID number). A -1 is returned if the second argument is not a valid subregion ID number.

These functions extract all of the information available from the HLI file.

4 Optimization Functions

(Note: this section will be more completely specified after I've looked through the GCC optimization functions).

The functions presented below are intended for keeping consistency between HLI items and RTL INSNs as GCC performs optimizations on the RTL.

These are for user use. However, it may be possible to include these automatically within GCC's own add_insn() and delete_insn() subroutines. Optimization passes that used those two subroutines exclusively could then ignore these functions.

- boolean add_item_id(int *item_id*, rtx *insn*);
- boolean move_item_id(int *item_id*, rtx *source_insn*, rtx *dest_insn*);
- boolean delete_item_id(int *item_id*, rtx *insn*);

These functions will add/move/delete an item based on ID number and the RTL INSN it is associated with.

- boolean redo_mapping(*void*);

This function will go through and reconstruct the mapping between the HLI items and RTL INSNs. Note: this function is for consistency checking and debugging only.

5 Timeline

The core set of functions in Section 3 should be completed by the end of May, 1996. This should allow GCC to read in the HLI items and match them with the RTL INSNs and successfully compile a program unit with optimizations turned off. During this summer work on adding the optimization functions for GCC's optimization passes will be done.