Addition of register variables to an existing table.

1. Introduction

This is a short description of the newest feature in the table driven code generator for the Amsterdam Compiler Kit. It describes how to add register variables to an existing table. This assumes a distribution of October 1983 or later. It is not clear whether one should read this when starting with a table for a new machine, or waiting till the table is well debugged already.

2. Modifications to the table itself.

2.1. Register section

Just before the properties of the register one of the following can be added:

- regvar
- regvar (pointer)
- regvar (loop)
- regvar (float)

All register variables of one type must be of the same size, and they may have no subregisters.

2.2. Codesection

- Two pseudo functions are added to the list allowed inside expressions:
  1) inreg (expr) has as a parameter the offset of a local, and returns 0,1 or 2:
     2: if the variable is in a register.
     1: if the variable could be in a register but isn't.
     0: if the variable cannot be in a register.
  2) regvar (expr) returns the register associated with the variable. Undefined if it is not in a register. So regvar (expr) is defined if and only if inreg (expr) == 2.
- It is now possible to remove() a register expression, this is of course needed for a store into a register local.
- The return out of a procedure may now involve register restores, so the special word 'return' in the table will invoke a user defined function.

3. Modifications to mach.c

If register variables are used in a table, the program *cgg* will define the word REGVARS during compilation of the sources. So the following functions described here should be bracketed by #ifdef REGVARS and #endif.

- regscore(off,size,typ,freq,totyp) long off;
  This function should assign a score to a register variable, the score should preferably be the estimated number of bytes gained when it is put in a register. Off and size are the offset and size of the variable, typ is the type, that is reg_any, reg_pointer, reg_loop or reg_float. Freq is the number of times it occurs statically, and totyp is the type of the register it is planned to go into.
  Keep in mind that the gain should be net, that is the cost for register save/restore sequences and the cost of initialisation in the case of parameters should already be included.
- i_regsave()
  This function is called at the start of a procedure, just before register saves are done. It can be used to initialise some variables if needed.

- f_regsave()
  This function is called at end of the register save sequence. It can be used to do the real saving if multiple register move instructions are available.
- regsave(regstr,off,size) char *regstr; long off;
  Should either do the real saving or set up a table to have it done by f_regsave. Note that initialisation of parameters should also be done, or planned here.
- regreturn()
  Should restore saved registers and return. The function result is already in the function return area by now.

4. Examples

Here are some examples out of the PDP 11 table

```
lol inreg($1)==2| |       | regvar($1)          | |

lil inreg($1)==2| |       | {regdef2, regvar($1)}        | |

stl inreg($1)==2| xsource2 |
                  remove(regvar($1))
                  move(%[1],regvar($1))      |    | |

inl inreg($1)==2| |   remove(regvar($1))
                  "inc %(regvar($1)%)"
                  setcc(regvar($1))      |    | |
```

5. Afterthoughts.

At the time of this writing the tables for the PDP 11 and the M68000 and the VAX are converted, in all cases the two byte wordsize versions. No big problems have occurred, but experience has shown that it is necessary to check the table carefully for all patterns with locals in them. Code may be generated that uses the memoryslot the local is not in.