Milestone : Register allocator

submit by 25th January 2010

for CM2 students
for JMC2 students taking course 221 Compilers
Second Year Computing Laboratory
Department of Computing,Imperial College

Purpose

This is a *milestone* for the compiler lab. Completing a tested register allocator before starting the rest of the compiler will make it more likely that the complete compiler will be correct.

Requirement

The MIPS architecture has a large number of general purpose registers and most instructions operate only on registers or constants. To make the best use of the registers, variable values should be held in registers when possible, but loaded from memory and stored to memory when required.

Because there are no user function calls, in the pico language more of the mips registers can be used for general purposes. Also the location of the program's variables in main memory are fixed at compile time (rather than only being an offset from the frame pointer). The parser assigns a unique name for each variable, which can be used as a label in the data area.

If the compiler cannot correctly translate a program it should not crash or write out incorrect assembler code, but print an clean error message and stop.

Register to use

Which mips registers can't be used

- `$at` the assembler temporary register is used when the assembler translates one instruction into two basic instructions.
- `$k0,$k1` are reserved for the kernel and interrupt handling.
- The `$zero` register always contains 0, and moving another value into it has no effect.

Can be used carefully

Before a system call is made the `$v0` register must be loaded with the number of the system call.

If the system call requires an argument that must be loaded into `$a0`.

If the system call returns a value when it finishes, the value is put in `$v0`. No other user registers are changed by the system call.

At other times `$a0` and `$v0` can be used as general purpose registers.

If the compiled program ends by using the `exit` system call the `$ra` (return address register), `$sp` and `$fp` can be used.

Register use

Registers will be used for temporary use during evaluating expressions, and can be released when the value has been used.

A register can be used to mirror the value of a variable, and reduce the number of loads from memory and stores to memory. Such a register can be in one of two states:

clean	where the register will have the same value as the variable when the program runs.
dirty	where the register has been changed since it was last stored in the memory.

A dirty register should not be released without its value being stored in the associated variable.

If the allocator is called incorrectly (for example the same register is freed twice) it should print an error and stop the compiler.

Problems with mirrored registers

Given a loop like

```
  x = 1;
  while ( x < 10) {
  ..
  /* statements using lots of registers */
  x=x+1;
}
```

the code might look like

```
    li $a1,1      # use $
    L0:
    bge $a1,10,L1
#  short of registers store $a1 in x
    sw $a1,x
    ...
#  use $a1 for something  else
    li  $a1,2
    lw  $a2,x     # load x into a free register
    add $a2,$a2,1
    b L0   # branch back to the test
L1:  # end of while loop
```

The compiler reads the source from the top of the loop, but when the program runs it can also go back from the end of the loop.

Methods

A possible set of methods in `Registers` for the compiler.

Information about a register could be held in the `Register` or in the `Registers` object.

Defining methods for the register lets you change the implementation of register allocator.

Register getFreeReg()	Returns a free register, or null if there are no free registers.
void free(Register)	The register can be reused.
boolean isFree(Register)	Is a particular register is free?

int freeRegs()	how many registers are free?
setVar(Register reg,String variable)	The register is not free, and will contain the value of the variable.
unSetVar(Register reg)	The register will not contain a variables value.
setDirty(Register reg)	The register has been changed and has not been stored in the variable in main memory. The code generator might not call the this method each time it generates code to change the register.
setClean(Register reg)	The register value is the same as variable in memory.
boolean isDirty(Register reg)	Returns true if code has been generated so the register value is not the same as the memory location for the variable.
setTemp(Register reg)	The register is used for a temporary value and not for mirroring a variable.
notTemp(Register reg)	The register does not a temporary register.

Testing

Some of the methods will be used in the compiler, others might just be used in the `Registers` class. It is not an error for a compiler to run out of registers providing the report is clear, it is an error to return invalid registers or allocate the same register twice. Think about what how your compiler will use registers and the suggested methods below, and what errors you need to

Write a JUnit test framework for your `Registers` class, using the eclipse junit support or the command line program

Submissions

Submit the source of your register allocator classes and your test framework, in the file `Registers.zip`.