TDT4205 Compiler Construction (2019)

README

Jørgen Bele Reinfjell

April 15, 2019

1 VSL

2 Structure

2.1 Binaries/programs

```
compiler      (ex5)  The vsl-to-assembly compiler.
print_symtab  (ex4)  Prints the generated symbol table of a vsl program.

vsl2py               The vsl-to-python3 transpiler (does not support 'asm').
                     For this to work (since this was created before symbol
                     tables were added) one has to define a main function in
                     the source code.

vsl_recreate         Converts a vsl program to high-level ir ast and then
                     recursively prints the ast, sort of like a formatter.
                     Used to check/test the high-level optimizations using
                     self_check.sh.

vsl_simplify  (ex3)  Simplifies the high-level ir ast and then prints a
                     representation of this new ast. This program
                     matches the example files in vsl_programs/*.tree.correct
                     except for function calls, where the 'reference' implementation
                     emits (null), and mine emits "func_call".
```

2.2 Source code organization

`nodetypes.c`

```
/nodetypes.c    Contains definitions and commonly used lookup-tables for a node.

node.c/node.h   Functions which aid the creation and usage of ast nodes.

tree.c/tree.h   Functions which apply to the parse tree (before symbol table binding).
                --> Converts the parse tree to a more easily used tree (ast), and
                    evaluates constant expressions.

instr.h         (inline) Helper routines to make emits of instructions look better.
ir.c/ir.h       Contains the symbol table creation. Binds symbols to ast nodes.

generator.c
/generator.h    Functions which generate the x86_64 assembly code from the ast
                Has to be used after symbol table creation.

vec.h           A vector/dynamic-array implementation which is used to define
                custom vector types and functions. (Kind of like c++ templates).
                This is used throughout the codebase where arrays of unknown max-size,
                for example stacks, are used. (Where explicit reallocation seems
                unnecessary).

utils.c         xmalloc(), xcalloc(), xrealloc() and debug().

vslmode.el      Syntax highlighting mode for vsl (for EMACS).d
```

2.3 Additions to the scanner, parser and compiler (outside the assignment)

I added a ’asm’ ’statement’ which inlines the rest of the line into the produced assembly. This makes it possible to do calls to external routines without modifying how the language handles unknown functions (which would require the use of function prototypes or assumptions). See `hashtable.vsl` and `libvsl.vsl` for examples.

Thus the following vsl program:

```
# #+BEGIN_SRC python
def f()
begin
    var x
    asm movq $10, -8(%rpb)
    return x # returns 10
end
```

will compile to the following assembly code:

```
_f:
    pushq %rbp
    movq %rsp, %rbp
    subq $16, %rsp
    movq $0, -8(%rbp)
    movq $10, -8(%rbp) # <--- This is the inline asm
    movq -8(%rbp), %rax
    movq %rbp, %rsp
    popq %rbp
     ret
```