Lab 5 Part 2: Tiger Compiler without register allocation

Description

Write a complete runnable tiger compiler, your goal is to make your compiler generate working code that runs on x86-64 platform, you can skip register allocation in this lab.

Related files for this lab are:

- **src/tiger/frame/.*** Files related to function stack frame(chapter 6)
- **src/tiger/translate/.*** Files related to IR tree translation(chapter 7)
- **src/tiger/canon/.*** Files related to basic blocks & traces(chapter 8, this part has already been implemented)
- **src/tiger/codegen/.*** Files related to assembly code generation(chapter 9)
- **src/tiger/runtime/runtime.c** Tiger program runtime file(will be linked with code files generated by your compiler)
- **src/tiger/env/env.*** The EnvEntry classes which help compiler store the information of variables&functions
- **src/tiger/output/output.*** Files related to output assembly for frags.

To finish this lab, you will only need to finish the following modules: { x64 stack frame } { IR tree translation} { code generation } you can modify any file to finish your design

Notice: Before you start this lab, you should carefully read the chapter 6,7,8,9,12 of the textbook. And if you have any question about this lab, feel free to contact Jinze Si, who is the teaching assistant responsible for lab 5.

Important Notes

1. If you have not finished your translation in lab5-part1 yet, **just complement codes for IR tree translation in this lab.**

2. Before you start to implement IR tree translation, carefully read the chapter 6&7 of the textbook.

3. Before you start to implement code generation, carefully read the calling convention of x86-64 platform.

4. The file runtime.c is a C-language file containing several external functions useful to your Tiger program. These are generally reached by externalCall from code generated by your compiler.

5. Carefully read the chapter 12, which gives some useful interface declarations you may use in module *frame*.

6. You can only use one register(%rsp) as stack pointer, **%rbp is not allowed to be used as stack pointer.**

7. After you finish codegen, remember to check output.cc for the output of your assembly code, we have annotated some code for passing compilation.

8. You can read "main/test_codegen.cc" to figure out the order in which modules are called and the workflow of tiger compiler.

9.This lab is very difficult and hard to debug, **please leave yourself enough time(at least two week) to finish this lab.**

Introduction to Tiger Interpreter

Since you have not finished register allocation part yet, your assembly can't link using gcc yet. We have implement a brand new way to help you do judge if your answer is right, which we call it Tiger Interpreter. This interpreter take assembly file(.s) as input, interpre your assembly code even without RA. Your final assembly code should act should like follow:

```
movq t112, %rsp
addq %rcx, %rbx
leaq L1(t118), %rdi
```

This code doesn't make any sense, but it shows what assembly code you should generate in this lab, where t1xx represents temporary register not allocated yet. Note that you can use machine registers now, and you must use for some special insturctions(like imul/idiv, ret value).
We only support following instructions now:
movq, addq, subq, imulq, idivq, leaq, callq, cmpq, jmp, je, jne, jg, jge, jl, jle, retq, cqto
which is enough for you to finish this lab, if you have any better idea about instructions, you can start a issue in public repository to tell us your idea.
Besides, we have implemented a debugger inside this interpreter to help you figure out what is going wrong, details could be found here(updating). Tiger Interpreter's source code is in scripts/lab5_test, and is written in python.

Environment

You will use the same code framework that you had set up when you worked on previous lab. What you need to do now is to pull the latest update of the code framework if there are any. You may have to do some code merging jobs.

```
shell% git fetch upstream
shell% git checkout -b lab5-part2 upstream/lab5-part2
shell% git push -u origin
shell% git merge lab5-part1

You may have to do some code merging jobs here

After merging, you are supposed to push your update to your remote repo
shell% git add files
shell% git commit -m "[lab5-part2] merge lab5-part1"
shell% git push origin lab5-part2
```

If you haven't set it up before, you should follow the instructions here to set up your lab environment.

Grade Test

The lab environment contains a grading script named as **grade.sh**, you can use it to evaluate your code, and that's how we grade your code, too. If you pass all the tests, the script will print a successful hint, otherwise, it will output some error messages. You can execute the script with the following commands.

```
Remember grading your lab under docker or unix shell! Never run these commands under windows cmd.
shell% make gradelab5-2
shell% ...
shell% [^_^]: Pass #If you pass all the tests, you will see these messages.
shell% TOTAL SCORE: 100
```

Handin

The deadline of this lab is on Friday 12:00 AT NOON, Dec 8, 2023, and if you miss the deadline, points will be deducted based on the number of days you are late!

After you have passed the grade test, you need first commit your modification, then push it to your remote repository on gitlab. You can use the following commands to finish this step.

```
shell% git add files
shell% git commit -m "[lab5-part2] finish lab5-part2"
shell% git push origin lab5-part2
```

Go to Top // *Compilers* Home Page
Questions or comments regarding *Compilers* course? Send e-mail to the course Staffs or TAs.
Last updated: Wed Oct. 16 2019