**Wei Fei (2538810)**

**EECS 665 Project 3 Report**

**Elluru, Bharath Chandra**

**12/01/15**

**Project 3: Code Generation for x86**

**Objective**

In this project, we need to make a code generation for X86, which can generate working code for the test example named “test.t”. Our object is to be able to correctly generate code for all of integer operations, such as add, subtract, multiply, divide, return, etc. In order to reach the object, we are supposed to edit the file “cgen.y”. When we compile the program, the system will generate x86 assembly code in the file “test.t.s”, and then generate test executable file. If we use “./test” without having any error, the output should have a set of calculation results from “test.t”.

**Implementation**

**(How does my program works)**

We only need to implement all functions of integer operations in the file “cgen.y”:

- ID ISHL ID for integer arithmetic shift left

- ID ISHR ID for integer arithmetic shift right

- ID IADD ID for integer add

- ID ISUB ID for integer subtract

- ID IMUL ID for integer multiply

- ID IDIV ID for integer division

- ID IMOD ID for integer modules

- IARG ID, ICALL ID INT for integer call and integer argument

- IRET ID for integer return

For more details of implementation, the operation add, subtract and multiply are similar, we can use the simple register “eax” to be an accumulator to implement these functions. However, for the operation of divide and mod, they are more complicated. We need to use another register named “ebx”, since we should divide the contents of “edx” by two parts. On the one hand, we should place the quotient in “eax”. On the other hand, we should also put the remainder in “edx”.

The coding strategy is like this:

***| ID IDIV ID { printf( " movl " );***

***function\_printtemp(function,$3);***

***printf( ", %%eax\n" );***

***printf(" cdq\n" );***

***printf( " movl " );***

***function\_printtemp(function,$1);***

***printf( ", %%ebx\n" );***

***printf( " idiv " );***

***printf( " %%ebx\n" ); }***

***| ID IMOD ID { printf( " movl " );***

***function\_printtemp(function,$3);***

***printf( ", %%eax\n" );***

***printf( " cdq\n" );***

***printf( " movl " );***

***function\_printtemp(function,$1);***

***printf( ", %%ebx\n" );***

***printf( " idiv " );***

***printf( " %%ebx\n" );***

***printf( " movl %%edx, %%eax\n" ); }***

For the operation of arithmetic shift operations, we need to use “sall” for shifting left and “sarl” for shifting right. The sign bit is shifted out to the carry flag., and a zero bit is placed in the least significant bit. Other bits are simply shifted to the left. The number of bits to shift can be constrained in the %cl register.

For example, here are partial code for shift left and shift right operations:

for the function “ID ISHL ID”

***printf( ", %%cl\n" );***

***printf( " sall " );***

***printf( "%%cl");***

for the function “ID ISHR ID”

***printf( ", %%cl\n" );***

***printf( " sarl " );***

***printf( "%%cl");***

**Discussion**

* **Troubles**

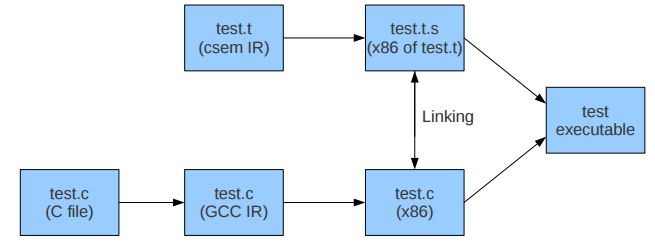
**(The issues I had in writing and testing)**

1. Since the “cgen,y” file has already given us the implementation of integer add operation, we can imitate those code to implement other similar functions. When I test the operation of subtract and multiply, the results are correct. However, when I test the operation of division and modules, the output shows me segmentation fault. I use GDB to find the potential error and find some useful information on the resource link of this lab. I found that we cannot use the register “eax” to handle the division and modules directly since these two operations may have remainder, which are different with other three arithmetic operations. We need to use another register named “ebx”, since we should divide the contents of “edx” by two parts, which place the quotient in “eax” and the remainder in “edx”. After I rewrite the code for the function “IDIV” and “IMOD”, I got the correct results.

2. When I was starting to write the code, I need to know how to test my program. I see two commands in make file, which are make test and make runtest. I tried both of them. The command “make test” gave me the process of compiling, but the command “make runtest” gave a series of intermediate code. I did not understand those output at that moment. After I loop up some relevant information about x86 assembly code, I figure out my confusions.

**Conclusion**

After we adding the codes in the required functions in the file “cgen.y”, the program has been executed and the output is correct. After completing this project, I understand how x86 assembly code generation process work. When we compile the program, the system will generate x86 assembly code in the file “test.t.s”, and then generate test executable file. The code generation framework is here:



We can use different accumulator to do the code generation, such as “eax” or “ebx”. Eventually, we got the correct output and results.

**Sample Outputs**