CS 213, Fall 1998
Homework Assignment H2
Assigned: Sept. 10, Due: Wed., Sept. 16, 11:59PM

Chris Colohan (`colohan+213@cs.cmu.edu`) is the lead person for this assignment.

The purpose of this assignment is to learn Alpha assembly language and to become familiar with how C code is translated into assembly. You will do this by looking at a sequence of assembly language sequences and decompiling them to find the C source code that produced them. Reverse engineering this code will improve your understanding of both the C constructs and the assembly code.

Introduction

In this assignment you will be given the assembler dump for a function, and it is your goal to derive the C source code that produced it. To help you I will give you hints about which operators and constructs were used in the function. The further in the assignment you go, the fewer hints you will have to work from.

If we look at a very simple function then we expect the compiler to produce very little code. For example:

```
int increment(int arg)
{
    return arg + 1;
}
```

produces the code (compiled with `gcc -O`):

```
0x0:  addl    a0, 0x1, v0
0x4:  ret     zero, (ra), 1
```

From this simple example we can see that the argument `arg` is passed in register `a0`, and the return value is passed in register `v0`.

When you are trying to figure out what a given functions does, try creating a small example like this one to see what code the compiler emits. If you can create a series of small functions that produce part of the answer, you can then piece them together to create a solution.

Logistics

You may work in a group of up to 2 people in solving the problems for this assignment. The only "hand-in" will be electronic. Any clarifications and revisions to the assignment will be posted on Web page `assigns.html` in the class WWW directory.

All files for this assignment are in the directory:

`/afs/cs.cmu.edu/academic/class/15213-f98/H2`

First create a (protected) directory to work in, and copy our template code using the command `tar -xvf /afs/cs.cmu.edu/academic/class/15213-f98/H2/H2.tar`. This will create the files `Makefile`, `TEAM_MEMBERS`, `ques[1-10].c`, and `ques[1-10]_ref.S`. In this assignment you will only modify and hand in the files `ques[1-10].c`, as well as `TEAM_MEMBERS`. Please edit the `TEAM_MEMBERS` file now so you do not forget.

These problems are self-testing. To check the solution for question 1, run the command `make test1`. This will compile `ques1.c` into an object file and disassemble it into `ques1.S`. If the disassembly is identical to `ques1_ref.S` then your solution is correct. If it is not, then it is incorrect.

The C functions which created the reference files are only a few lines long (except for the switch statement, which is longer but repetitive), and use just the variables declared in `ques[1-10].c`. If you keep your program simple it is more likely to match the code given.

Evaluation

Each question is worth 3 points, and will be graded on two aspects: correctness and simplicity. If your code produces the same assembly code as the reference code, then it will get $2/2$ on correctness. Many of the programs can be expressed correctly using low level primitives but are still hard to read. If your solution contains a `while()` loop with an initialization and an increment, try turning it into a `for()` loop. If it contains a whole series of `if()` statements, see if you can express it as a `switch()` statement. Often a simple integer expression will be turned into a more complex one by the compiler as well — see if you can find the simpler one. If your function contains simple code, it will get $1/1$ for simplicity. If I tell you the function is a loop and you implement it as a series of `if()` statements, `goto` commands, and bit operations then you will get $0/1$ for simplicity.

The problems are all weighted equally. The earlier problems are easier than the later problems, so plan accordingly. Since there are 10 questions, the assignment will be graded out of 30.

Problems

Question 1: Integer Expression

This function implements an integer expression that uses `+`, `-` and `*`. You will notice that after the `ret` instruction is some no-ops, these are to align the next function in the program.

Question 2: Integer Division

Division by a constant is handled in a clever way by the compiler. The compiler uses tricks to handle both the positive and negative case, and emits tuned code to handle dividing by a power of two. More complex division is handled through a series of table lookups. This question contains a number of integer divisions by a power of two constant, as well as other operations.

Question 3: Bit Operations

When I first tried solving the `bang()` question on Homework 1, I did it using a tree of shifts and ors. This is a simplified version of my solution. It uses the operators `>>`, `|`, `&` and `^`.

Question 4: Conditionals

If your code contains `if` statements then the compiler will generate branches. This code contains a branch.

Question 5: Loops

Loops are just like conditionals, but the branch goes backwards. The compiler inserts `nop`s into the code to make the cache perform better.

Question 6: Small Switch Statement

If there are a small number of cases in a `switch()` statement then the compiler turns it into a series of if statements. Find the `switch()` that produced this code.

Question 7: Large Switch Statement

If there are a large number of cases in a `switch()` statement and they are consecutive then the compiler builds a jump table. In this question you can not see the entries in the table, but you can guess what they are from the code.

Question 8: Function Call

This function calls another function. This makes it necessary to set up the stack for the called function.

Question 9: Recursive Function Call

If a function calls itself then it can use this knowledge to skip some of the function prologue. Once you have deciphered this function it should be familiar to you.

Question 10: Grab Bag

This question combines a number of the operations and constructs listed above.

Hand In

Make sure you have edited the file `TEAM_MEMBERS` to identify who is in your programming team. If you run the command `make handin NAME=yourname`, where `yourname` is replaced with the Andrew Id of the first person on your team, a tarfile will be created and copied to the handin directory.

You only have write permission to this directory. If you make a mistake, e.g., by giving the file an incorrect name, send a copy of the correct file via email to the contact person for this assignment along with an explanation.

You may only hand in an assignment once. Make sure your code is in its final form before handing it in.