ECE 468
Problem Set 1: Compiler basics **[Solutions]**

1. Consider a machine with no branch predictors (*i.e.*, the hardware does not use past behavior to tell whether a branch will be taken or not taken). Instead, the machine's ISA allows a compiler to set a flag on each branch indicating if the branch is usually taken or usually not taken. Which kind of compiler would you expect to perform better for this machine: a static compiler (like gcc)? Or a just-in-time compiler? Explain.

   Answer:

   In this question, I was looking more for a justification for your answer, rather than one specific answer. One answer is that since branch prediction is a fundamentally dynamic task, and is often input-dependent (*i.e.*, whether a branch is taken or not depends on the particular input that you give to a program), a just-in-time compiler will do better, as it will be able to take into account the actual behavior before setting the taken/not-taken flag for a branch. Conversely, you could argue that a static compiler like gcc might fare better because any time you have to offload more work onto the compiler (such as branch prediction) it is better to give the compiler enough time to perform sophisticated analyses and look at more code.

2. Using the production rules of MICRO, give a derivation of the program: `BEGIN id := id - id; READ(id); END`. Is this the only possible derivation?

   Answer:

   There was some confusion regarding what, exactly, this question was asking for. I was not looking for a particular piece of code that would lead to the given string of tokens. Rather, I was looking for a series of rule applications that would allow you to rewrite `program SCANEOF` into the given program (as we did in slides 11–20 of the Lecture 2).

   (a) `program SCANEOF` [I will drop SCANEOF from the following steps, for brevity]

   (b) `BEGIN statement_list END` [expand statement_list]

   (c) `BEGIN statement; {statement;} END` [add one more statement]

   (d) `BEGIN statement; statement; END` [expand 1st statement]

   (e) `BEGIN ID := expression; statement; END` [expand expression]

   (f) `BEGIN ID := primary {add_op primary}; statement; END` [continue with add_op primary]

   (g) `BEGIN ID := primary add_op primary; statement; END` [expand 1st primary]

(h) `BEGIN ID := ID add_op primary; statement; END` [expand add_op]

(i) `BEGIN ID := ID - primary; statement; END` [expand primary]

(j) `BEGIN ID := ID - ID; statement; END` [expand statement]

(k) `BEGIN ID := ID - ID; READ(id_list); END` [expand id_list]

(l) `BEGIN ID := ID - ID; READ(ID ,id); END` [drop remaining ids]

(m) `BEGIN ID := ID - ID; READ(ID); END` [done!]

Applying these particular rules is the only way to derive this program. However, you could apply certain rules in different orders (for example, when given the choice between non-terminals to expand, as in step d, you could expand the rightmost non-terminal instead of the leftmost).

3. Describe a scenario where writing code directly in assembly might be preferable to using a compiler. Describe a scenario when the opposite is true.

Answer:

If you have a tight loop that is run many times, or one that can take advantage of complex hardware features (like SSE), you may be able to achieve better performance by taking the time to write *that section of code* in assembly, as compilers have a hard time identifying places where SSE can be useful. In almost all other situations (including many we gave in class: portability, long code, etc.), you will be better off using a compiler.

4. Give a regular expression that matches valid email addresses. A valid email address, for the purposes of this problem, consists of a string of alphanumeric characters, followed by an "@" sign, followed by a domain name that is a string of alphanumeric characters followed by ".com" or ".edu" Subdomains are allowed. For example, the following are valid email addresses:

milind@purdue.edu, bob@bob.foo.com

and the following are not:

bob@bob.net, @bar@foo.com

Answer:

Note that there are several (equivalent) ways to write this regular expression. Here is one.

Let α = [a-zA-Z0-9]

$\alpha^+@(\alpha^+ .)^+(\text{com}|\text{edu})$