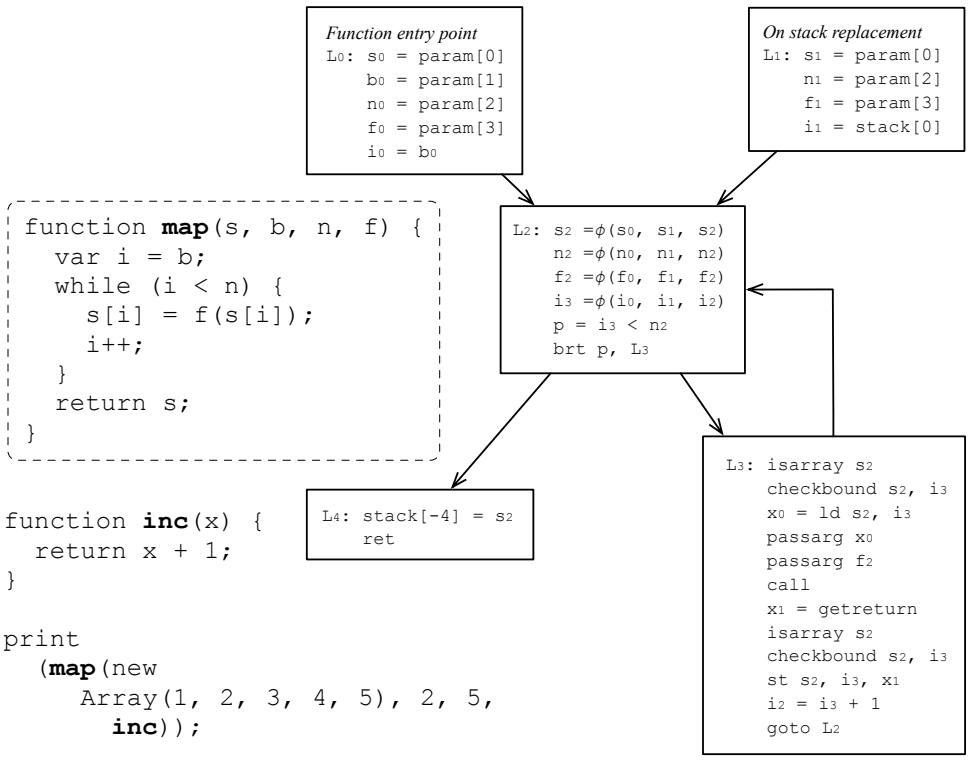
DCC888 – Just-in-Time Compilation

Name: _____ ID: _____

1. Consider the function `map`, whose CFG, as produced by a just-in-time compiler, can be seen on the right. An example of use of `map` is also given. The rest of this question refers to this implementation and call of `map`:



   (a) What would be the values of s_0, n_0 and f_0 in the call `map(new array(1, 2, 3, 4, 5), 2, 5, inc)`?

   (b) How would the CFG of `map` look like, after we perform constant propagation, given the specialization of `map(new array(1, 2, 3, 4, 5), 2, 5, inc)`? Draw it in the space at the end of the page. If you think the space is too limited, remember what Fermat wrote: "*J'ai découvert une démonstration véritablement merveilleuse de cette proposition que cette marge est trop étroite pour contenir.*" If you are still concerned, then use another page.

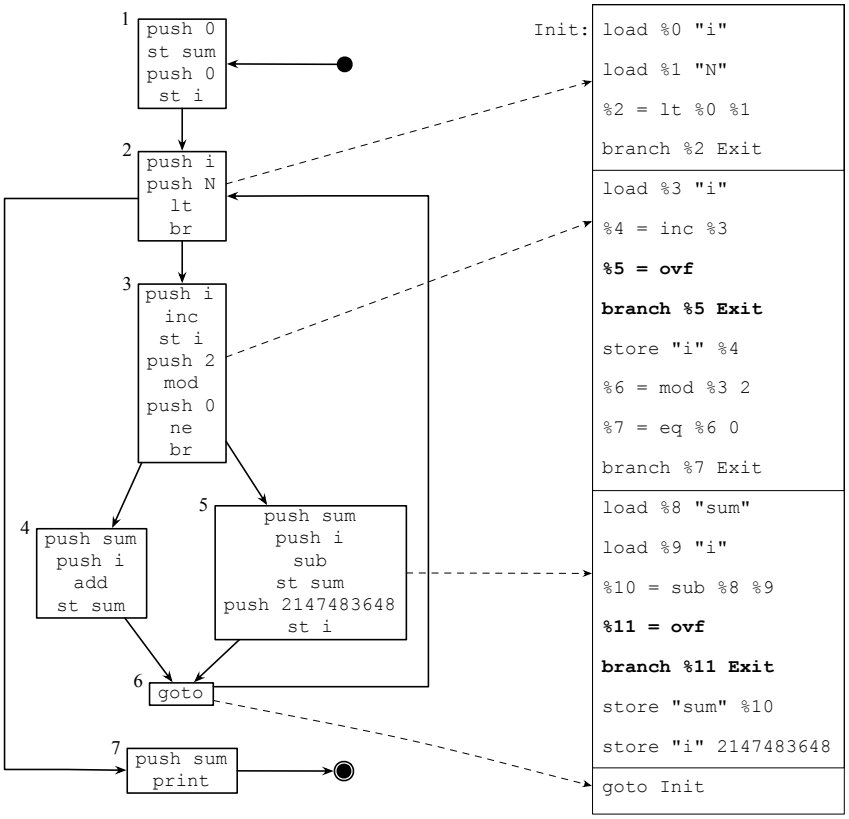
(c) How would be the new CFG after the application of loop inversion. Let's start from the program after constant propagation. Is it necessary to keep the safe-guard code that checks for loops with zero trip counts?

(d) Starting from the previous CFG, what would we get after we perform a round of dead-code-elimination in our program? Can we eliminate any basic block entirely?

(e) The variable i is bound by n_2, which, by the way, should be a constant by now. This variable is initialized with b_0, which is also a constant during a single call of `map`. Variable i is only incremented in the program. Given all these facts, can we eliminate any array bounds checks? Draw the resulting CFG.

(f) As a last optimization, we can inline `inc` into `map`. Draw the resulting CFG, assuming that we start after array bounds check elimination.

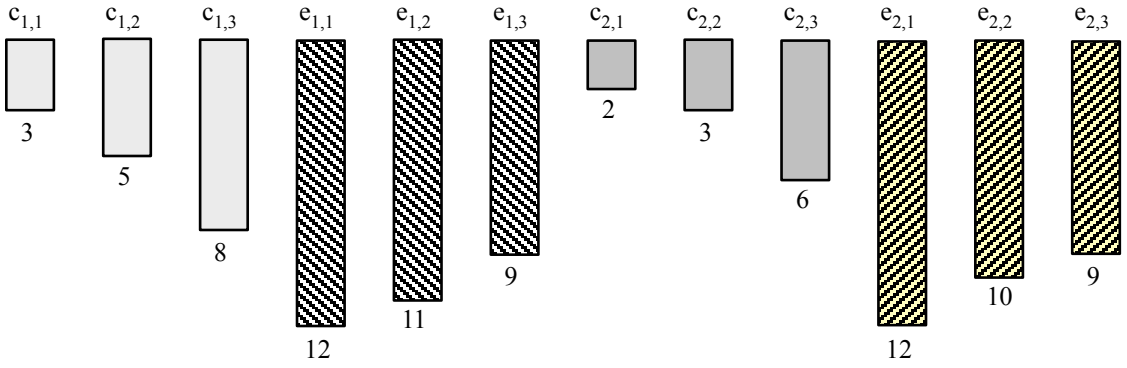
2. Consider the program below, and a trace that results from its partial compilation. This trace contains two overflow checks, which are necessary to preserve the semantics of the JavaScript program. On the fly, numbers can be treated as integers. But if any arithmetic operation on these numbers results in an integer overflow, they must be converted back into doubles. Given the result of a range analysis fed with runtime information, which overflow checks can be eliminated, assuming that `N` = 10?



3. Determining the order when functions should be JIT compiled is a very tough problem. To be precise, this problem is NP-Complete[1]. We can define the Optimal Compilation Scheduling Problem (OCSP9) as follows:

> **Inputs:** a sequence of function invocations, where each element, say m_f represents an invocation of a function. A function can appear once, or multiple times in the invocation sequence. For each function, there are multiple possible compilation levels. Let $c_{i,j}$ represent the length of the time needed to compile function m_i at level j, and $e_{i,j}$ represent the length of the corresponding execution time of the produced code. Assuming all $c_{i,j}$ and $e_{i,j}$ are known and independent of compilation orders, and $\forall i$ and $j_1 < j_2$, we have that $c_{i,j_i} \leq c_{i,j_2}$, and $e_{i,j_i} \geq e_{i,j_2}$. A function cannot run unless it has been compiled at least once (no matter at which level) in an execution of the invocation sequence. A function can be compiled once or multiple times in a run. The code produced by the latest compilation is used for execution.
> **Goal:** to find a sequence of compilation events that minimizes the make-span of the program execution. Here, make-span is the time spanning from the start of the first compilation event to the end of the program execution.

(a) Consider that we have a program with two functions, f_1 and f_2, and that we have three different compilation modes. The times to compile and execute each function, in each mode, is given by the figure below:



These bars imply, for instance, that we take 3 time units to compile function f_1 in mode 1, or that we take 5 time units to compile function f_1 in mode 2. Similarly, the figure says that function f_1 takes 12 time units to execute if compiled in mode 1, and 11 time units to execute if compiled in mode 2. What are the best compilation schedules for each sequence below?

i. $f_1; f_2$

[1] All the notions involved in this exercise have been borrowed from the paper *Finding the Limit: Examining the Potential and Complexity of Compilation Scheduling for JIT-Based Runtime Systems.*

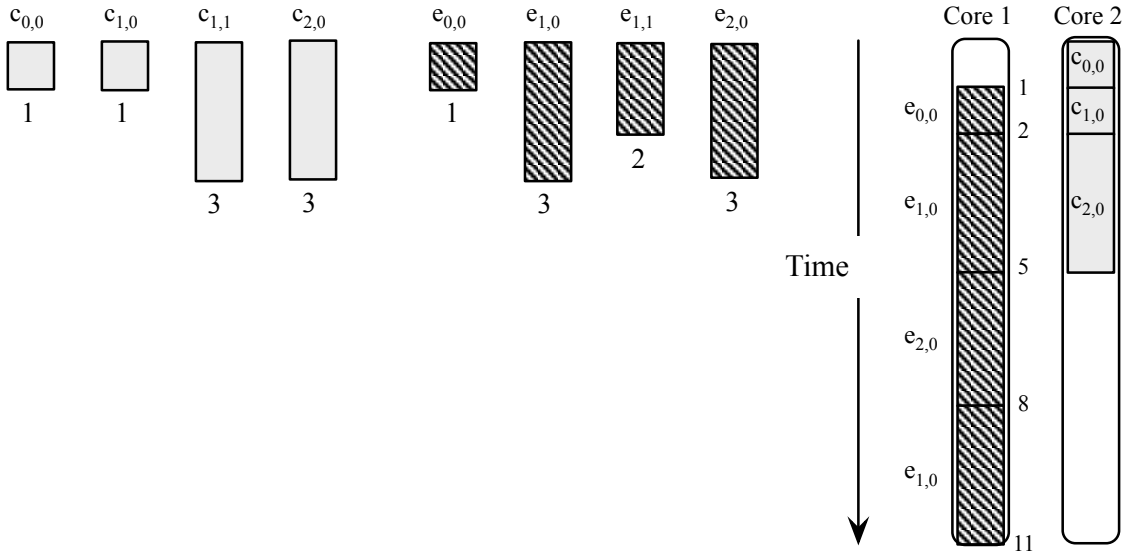
ii. $f_1; f_1; f_2$

iii. $f_1; f_2; f_2$

iv. $f_1; f_1; f_2; f_2$

(b) If we have only one core, then we must compile and execute all the functions in the same processor. In this case, there is a simple way to determine the best compilation schedule, given that we are allowed to see the entire sequence of functions that will be invoked. Which algorithm can we use to find the best scheduling?

(c) On the other hand, if we have two or more cores, then we might, for instance, compile a function while we execute another function. In this case, the problem of finding an optimal compilation scheduling is NP-complete. Imagine, for instance, that we have two cores. One of these cores can only be used to compile functions, and the other core can only be used to execute functions. Let's assume that we need to execute three functions: f_0, f_1 and f_2, in the sequence: $f_0; f_1; f_2; f_1$. We have the following constraints for the scheduling problem:

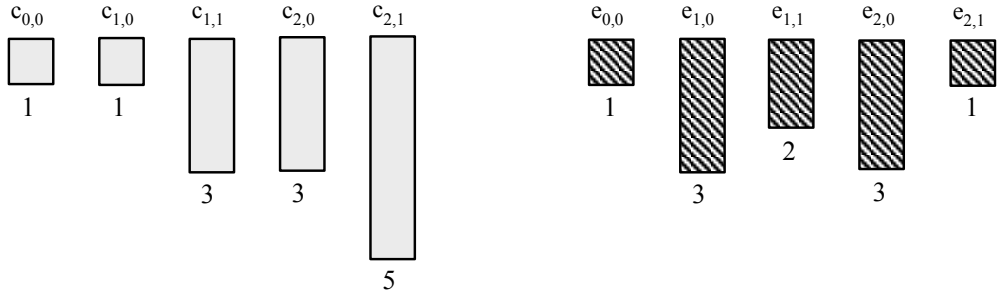


The figure, on the right, shows one scheduling, in which we compile each function in mode 0. Level 0 is the fastest compilation mode, on the other hand, it yields the slowest code. Overall, we took 11 time units to finish the execution of this example.

i. How would be the scheduling if we compiled f_1 using level 1, and compiled all the other functions, e.g., 0 and 2, using level 0? Would we get a faster, or a slower execution?

ii. Can you find an execution scheduling that takes only 10 time units? That is not a very easy task, but it might be fun: this problem is very puzzle-like!

iii. Consider, now, that $c_{2,1} = 5$, and $e_{2,1} = 1$, as we have in the scheme below:



Can you find an scheduling for the sequence $f_0; f_1; f_2; f_1; f_2$ that finishes execution in only 12 time units?