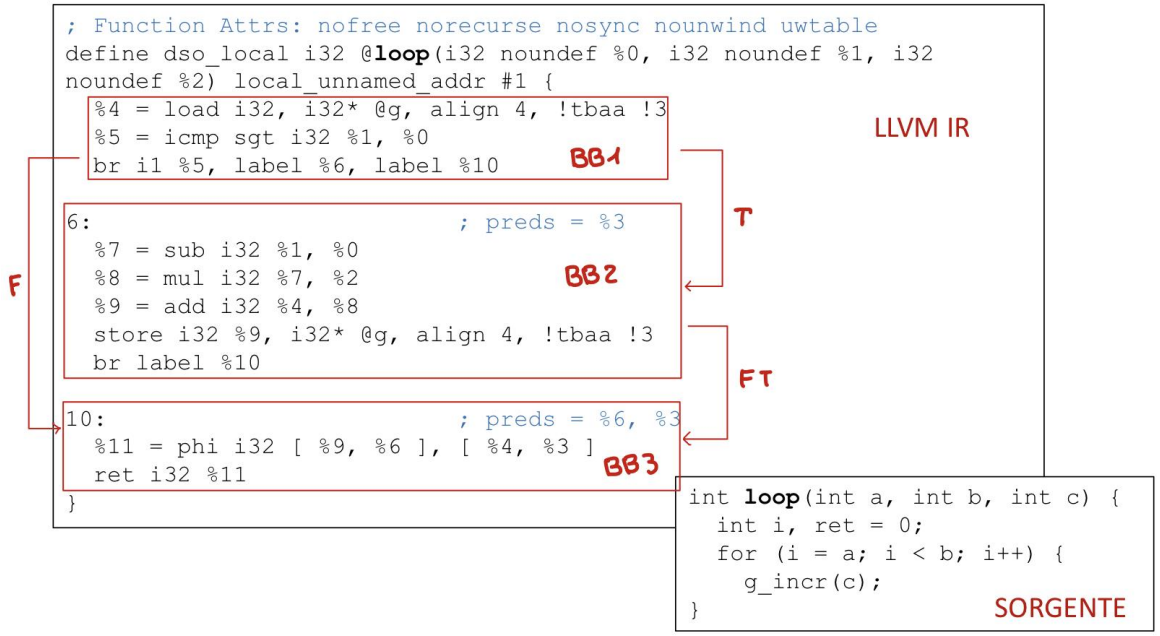
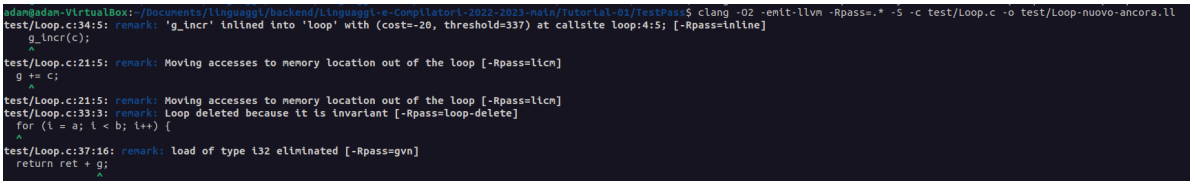
Esercizio 1



L'opzione -O0 di clang non ottimizza il codice, a differenza di -O2.
Con l'opzione -O2 viene utilizzato SGT, nel controllo tra A e B, quindi la condizione del for diventa la seguente: a>b.
Con l'opzione -O0 viene usato SLT, nel controllo tra A e B, quindi la condizione del for diventa la seguente: a<b.

Il flag -Rpass=.* di Clang, crea dei report sul terminale per tener traccia dell'ottimizzazione del codice



Esercizio 2

Codice realizzato:

```
#include <llvm/Passes/PassBuilder.h>
#include <llvm/Passes/PassPlugin.h>
#include <llvm/Support/raw_ostream.h>
#include <string>

using namespace llvm;

namespace {

class TestPass final : public PassInfoMixin<TestPass> {
public:
  PreservedAnalyses run([[maybe_unused]] Module &M, ModuleAnalysisManager &) {
    outs() << "Passo di test per il corso di Linguaggi e Compilatori"
           << "\n";

      // TODO: Completare il metodo come indicato per il LAB1.
      for (auto iter = M.begin(); iter != M.end(); ++iter){
        Function &F = *iter;
        auto nBasicBlocks = 0;
        std::string variable_arg = "";

        outs() << "Nome funzione: "<< F.getName() <<"\n";

        if(F.isVarArg()){
          variable_arg = "+*";
        }
        outs() << "Numero parametri: "<< F.arg_size() << variable_arg <<"\n";
        for (auto &basic_block : F) {
          nBasicBlocks++;
        }

        const CallGraphNode * llvm::CallGraph::operator[](  const Function *  F     )
        outs() << "Numero basic block : " << nBasicBlocks << "\n";
        outs() << "Numero istruzioni : " <<F.getInstructionCount() << "\n";
        outs() << "--------------------" << "\n";
       }

    return PreservedAnalyses::all();
    }
  };
} // class TestPass
```

```
 // anonymous namespace

extern "C" PassPluginLibraryInfo llvmGetPassPluginInfo() {
  return {
      .APIVersion = LLVM_PLUGIN_API_VERSION,
      .PluginName = "TestPass",
      .PluginVersion = LLVM_VERSION_STRING,
      .RegisterPassBuilderCallbacks =
          [](PassBuilder &PB) {
            PB.registerPipelineParsingCallback(
                [](StringRef Name, ModulePassManager &MPM,
                   ArrayRef<PassBuilder::PipelineElement>) -> bool {
                  if (Name == "test-pass") {
                    MPM.addPass(TestPass());
                    return true;
                  }
                  return false;
                });
          }
  };
}
```

OUTPUT

```
Passo di test per il corso di Linguaggi e Compilatori
Nome funzione: g_incr
Numero parametri: 1
Numero basic block : 1
Numero istruzioni : 4
--------------------
Nome funzione: loop
Numero parametri: 3
Numero basic block : 3
Numero istruzioni : 10
--------------------
```

Abbiamo riscontrato difficoltà con l'implementazione del punto 3.