Compiler Assignment

Phase 1

Team Members:

Anvaya B Narappa (an001)

Arnav Rahalkar (araha008)

Step 2:

i) source (.c) to binary (executable)

Binary code that can be executed directly on the CPU

```
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$ clang test.c -o testExecutable
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$ ll
total 28
drwxrwxr-x 2 vagrant vagrant  4096 Nov  6 01:57 ./
drwxrwxr-x 5 vagrant vagrant  4096 Oct 11  2022 ../
-rw-rw-r-- 1 vagrant vagrant   331 Nov  5  2020 test.c
-rwxrwxr-x 1 vagrant vagrant 15632 Nov  6 01:57 testExecutable*
```

CMD: clang test.c -o testExecutable

Input File: test.c

Output File: testExecutable

ii) source (.c) to object file (.o)

Intermediary files generated in the compilation process

```
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$ clang -c test.c -o testObject.o
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$ ll
total 32
drwxrwxr-x 2 vagrant vagrant  4096 Nov  6 01:57 ./
drwxrwxr-x 5 vagrant vagrant  4096 Oct 11  2022 ../
-rw-rw-r-- 1 vagrant vagrant   331 Nov  5  2020 test.c
-rwxrwxr-x 1 vagrant vagrant 15632 Nov  6 01:57 testExecutable*
-rw-rw-r-- 1 vagrant vagrant  1280 Nov  6 01:57 testObject.o
```

CMD: clang -c test.c -o testObject.o

Input File: test.c

Output File: testObject.o

iii) source (.c) to machine assembly (.s)

Human-readable form of machine instructions that can be executed on the CPU.

```
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$ clang -S test.c -o testAssembly.s
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$ ll
total 36
drwxrwxr-x 2 vagrant vagrant  4096 Nov  6 01:57 ./
drwxrwxr-x 5 vagrant vagrant  4096 Oct 11  2022 ../
-rw-rw-r-- 1 vagrant vagrant   331 Nov  5  2020 test.c
-rw-rw-r-- 1 vagrant vagrant  2099 Nov  6 01:57 testAssembly.s
-rwxrwxr-x 1 vagrant vagrant 15632 Nov  6 01:57 testExecutable*
-rw-rw-r-- 1 vagrant vagrant  1280 Nov  6 01:57 testObject.o
```

CMD: clang -S test.c -o testAssembly.s

Input File: test.c

Output File: testAssembly.s

iv) source (.c) to LLVM bitcode (.bc); source (.c) to LLVM IR (.ll)

LLVM bitcode files enable platform-independent optimizations.

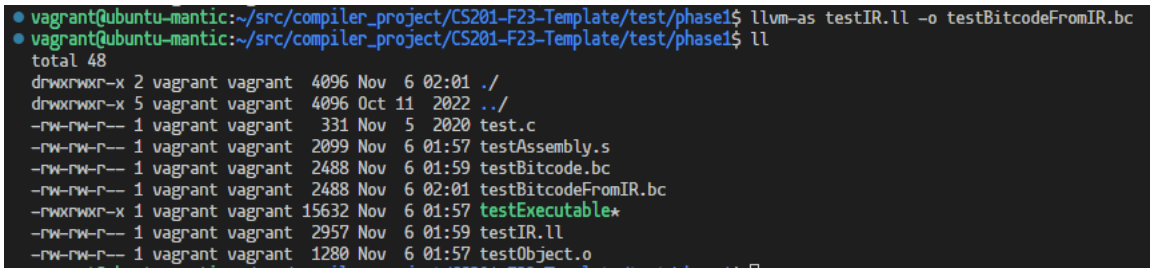
```
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$ clang -emit-llvm -c test.c -o testBitcode.bc
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$ ll
total 40
drwxrwxr-x 2 vagrant vagrant  4096 Nov  6 01:59 ./
drwxrwxr-x 5 vagrant vagrant  4096 Oct 11  2022 ../
-rw-rw-r-- 1 vagrant vagrant   331 Nov  5  2020 test.c
-rw-rw-r-- 1 vagrant vagrant  2099 Nov  6 01:57 testAssembly.s
-rw-rw-r-- 1 vagrant vagrant  2488 Nov  6 01:59 testBitcode.bc
-rwxrwxr-x 1 vagrant vagrant 15632 Nov  6 01:57 testExecutable*
-rw-rw-r-- 1 vagrant vagrant  1280 Nov  6 01:57 testObject.o
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$ clang -emit-llvm -S test.c -o testIR.ll
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/test/phase1$ ll
total 44
drwxrwxr-x 2 vagrant vagrant  4096 Nov  6 01:59 ./
drwxrwxr-x 5 vagrant vagrant  4096 Oct 11  2022 ../
-rw-rw-r-- 1 vagrant vagrant   331 Nov  5  2020 test.c
-rw-rw-r-- 1 vagrant vagrant  2099 Nov  6 01:57 testAssembly.s
-rw-rw-r-- 1 vagrant vagrant  2488 Nov  6 01:59 testBitcode.bc
-rwxrwxr-x 1 vagrant vagrant 15632 Nov  6 01:57 testExecutable*
-rw-rw-r-- 1 vagrant vagrant  2957 Nov  6 01:59 testIR.ll
-rw-rw-r-- 1 vagrant vagrant  1280 Nov  6 01:57 testObject.o
```

CMD: clang -emit-llvm -S test.c -o testIR.ll

Input File: test.c

Output File: testIR.ll

v) LLVM IR (.ll) to LLVM bitcode (.bc)

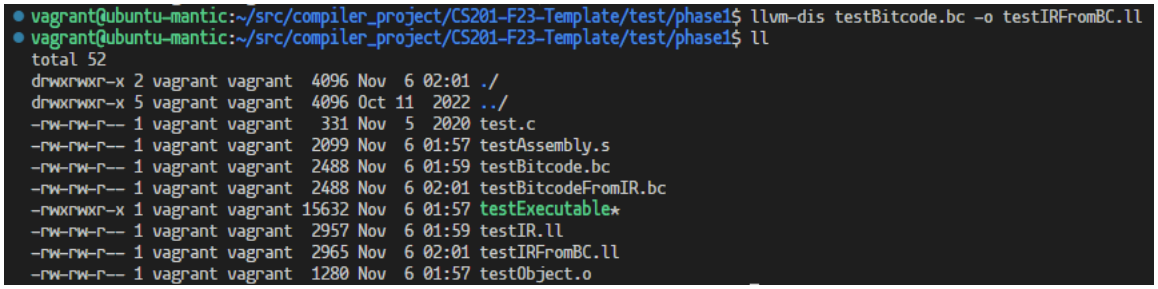


CMD: llvm-as testIR.ll -o testBitcodeFromIR.bc

Input File: testIR.ll

Output File: testBitcodeFromIR.bc

vi) LLVM bitcode (.bc) to LLVM IR (.ll)

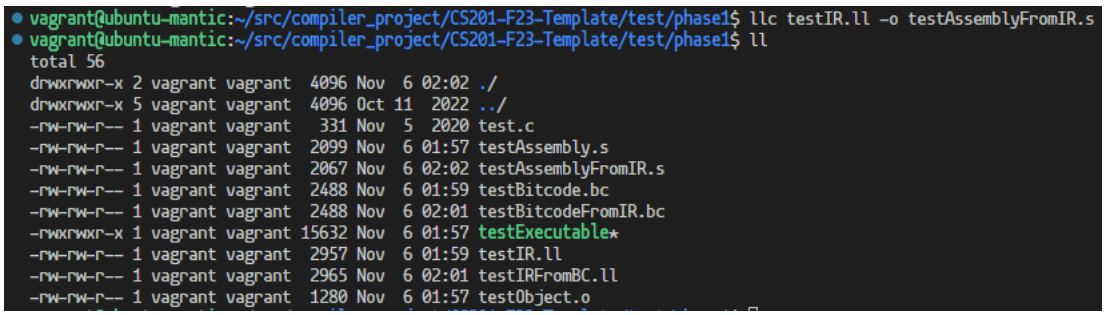


CMD: llvm-dis testBitcodeFromIR.bc -o testIRFromBC.ll

Input File: testBitcodeFromIR.bc

Output File: testIRFromBC.ll

vii) LLVM IR (.ll) to machine assembly (.s)



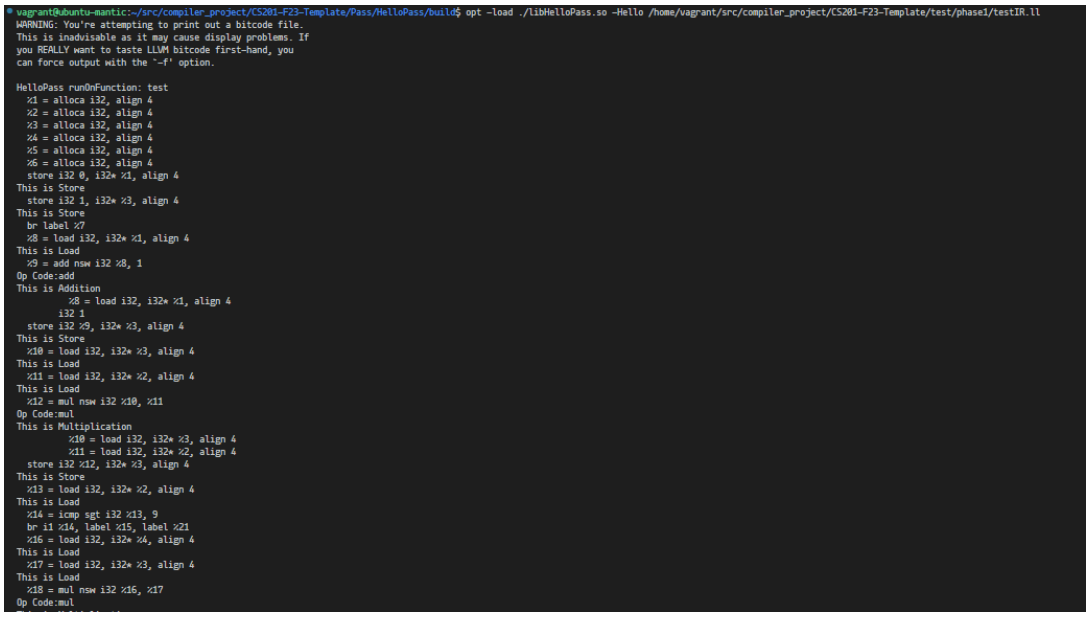
CMD: llc testIR.ll -o testAssemblyFromIR.s

Input File: testIR.ll

Output File: testAssemblyFromIR.s

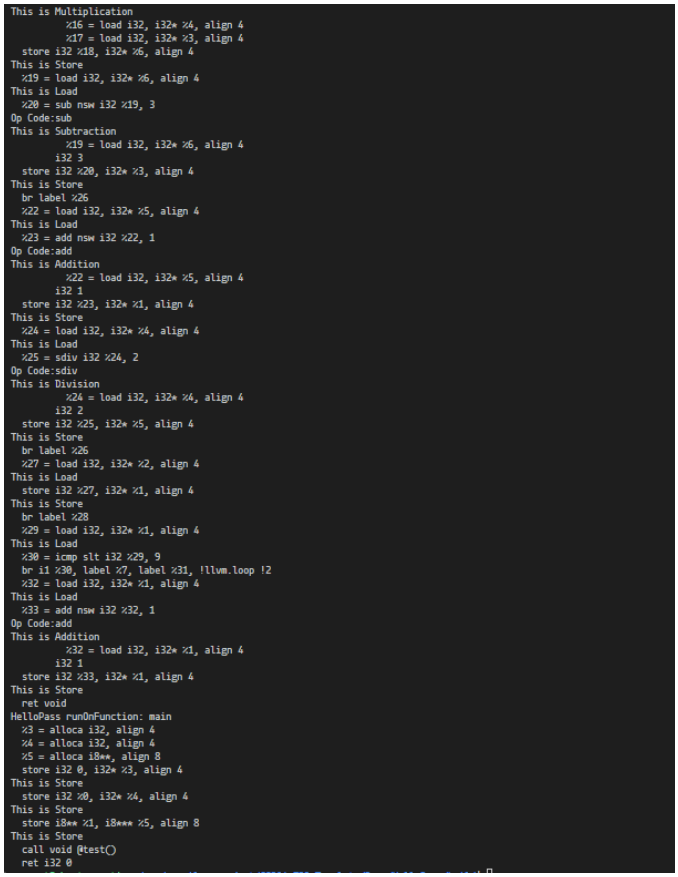
Step 3:

Running the opt on HelloPass without any changes.



```
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/Pass/HelloPass/build$ opt -load ./libHelloPass.so -Hello /home/vagrant/src/compiler_project/CS201-F23-Template/test/phase1/testIR.ll
WARNING: You're attempting to print out a bitcode file.
This is inadvisable as it may cause display problems. If
you REALLY want to taste LLVM bitcode first-hand, you
can force output with the `-f' option.

HelloPass runOnFunction: test
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store i32 0, i32* %1, align 4
This is Store
  store i32 1, i32* %3, align 4
This is Store
  br label %7
  %8 = load i32, i32* %1, align 4
This is Load
  %9 = add nsw i32 %8, 1
Op Code:add
This is Addition
        %8 = load i32, i32* %1, align 4
       i32 1
  store i32 %9, i32* %3, align 4
This is Store
  %10 = load i32, i32* %3, align 4
This is Load
  %11 = load i32, i32* %2, align 4
This is Load
  %12 = mul nsw i32 %10, %11
Op Code:mul
This is Multiplication
         %10 = load i32, i32* %3, align 4
         %11 = load i32, i32* %2, align 4
  store i32 %12, i32* %3, align 4
This is Store
  %13 = load i32, i32* %2, align 4
This is Load
  %14 = icmp sgt i32 %13, 9
  br i1 %14, label %15, label %21
  %16 = load i32, i32* %4, align 4
This is Load
  %17 = load i32, i32* %3, align 4
This is Load
  %18 = mul nsw i32 %16, %17
Op Code:mul
```



```
This is Multiplication
         %16 = load i32, i32* %4, align 4
         %17 = load i32, i32* %3, align 4
  store i32 %18, i32* %6, align 4
This is Store
  %19 = load i32, i32* %6, align 4
This is Load
  %20 = sub nsw i32 %19, 3
Op Code:sub
This is Subtraction
         %19 = load i32, i32* %6, align 4
       i32 3
  store i32 %20, i32* %3, align 4
This is Store
  br label %26
  %22 = load i32, i32* %5, align 4
This is Load
  %23 = add nsw i32 %22, 1
Op Code:add
This is Addition
         %22 = load i32, i32* %5, align 4
       i32 1
  store i32 %23, i32* %1, align 4
This is Store
  %24 = load i32, i32* %4, align 4
This is Load
  %25 = sdiv i32 %24, 2
Op Code:sdiv
This is Division
         %24 = load i32, i32* %4, align 4
       i32 2
  store i32 %25, i32* %5, align 4
This is Store
  br label %26
  %27 = load i32, i32* %2, align 4
This is Load
  store i32 %27, i32* %1, align 4
This is Store
  br label %28
  %29 = load i32, i32* %1, align 4
This is Load
  %30 = icmp slt i32 %29, 9
  br i1 %30, label %7, label %31, !llvm.loop !2
  %32 = load i32, i32* %1, align 4
This is Load
  %33 = add nsw i32 %32, 1
Op Code:add
This is Addition
         %32 = load i32, i32* %1, align 4
       i32 1
  store i32 %33, i32* %1, align 4
This is Store
  ret void
HelloPass runOnFunction: main
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  %5 = alloca i8**, align 8
  store i32 0, i32* %3, align 4
This is Store
  store i32 %0, i32* %4, align 4
This is Store
  store i8** %1, i8*** %5, align 8
This is Store
  call void @test()
  ret i32 0
```

After Modification to HelloPass to print the predecessors and successors

```
vagrant@ubuntu-mantic:~/src/compiler_project/CS201-F23-Template/Pass/HelloPass/build$ opt -load ./libHelloPass.so -Hello /home/vagrant/src/compiler_project/CS201-F23-Template/test/phase1/testIR.ll
WARNING: You're attempting to print out a bitcode file.
This is inadvisable as it may cause display problems. If
you REALLY want to taste LLVM bitcode first-hand, you
can force output with the `-f' option.

HelloPass runOnFunction: test
Number of Predecessors: 0
Number of Successors: 1
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store i32 0, i32* %1, align 4
This is Store
  store i32 1, i32* %3, align 4
This is Store
  br label %7
Number of Predecessors: 2
Number of Successors: 2
  %8 = load i32, i32* %1, align 4
This is Load
  %9 = add nsw i32 %8, 1
Op Code:add
This is Addition
         %8 = load i32, i32* %1, align 4
       i32 1
  store i32 %9, i32* %3, align 4
This is Store
  %10 = load i32, i32* %3, align 4
This is Load
  %11 = load i32, i32* %2, align 4
This is Load
  %12 = mul nsw i32 %10, %11
Op Code:mul
This is Multiplication
         %10 = load i32, i32* %3, align 4
         %11 = load i32, i32* %2, align 4
  store i32 %12, i32* %3, align 4
This is Store
  %13 = load i32, i32* %2, align 4
This is Load
  %14 = icmp sgt i32 %13, 9
  br i1 %14, label %15, label %21
Number of Predecessors: 1
Number of Successors: 1
  %16 = load i32, i32* %4, align 4
This is Load
  %17 = load i32, i32* %3, align 4
This is Load
  %18 = mul nsw i32 %16, %17
Op Code:mul
This is Multiplication
         %16 = load i32, i32* %4, align 4
         %17 = load i32, i32* %3, align 4
  store i32 %18, i32* %6, align 4
This is Store
  %19 = load i32, i32* %6, align 4
This is Load
  %20 = sub nsw i32 %19, 3
Op Code:sub
This is Subtraction
         %19 = load i32, i32* %6, align 4
       i32 3
  store i32 %20, i32* %3, align 4
```

```
This is Store
  br label %26
Number of Predecessors: 1
Number of Successors: 1
  %22 = load i32, i32* %5, align 4
This is Load
  %23 = add nsw i32 %22, 1
Op Code:add
This is Addition
         %22 = load i32, i32* %5, align 4
        i32 1
  store i32 %23, i32* %1, align 4
This is Store
  %24 = load i32, i32* %4, align 4
This is Load
  %25 = sdiv i32 %24, 2
Op Code:sdiv
This is Division
         %24 = load i32, i32* %4, align 4
        i32 2
  store i32 %25, i32* %5, align 4
This is Store
  br label %26
Number of Predecessors: 2
Number of Successors: 1
  %27 = load i32, i32* %2, align 4
This is Load
  store i32 %27, i32* %1, align 4
This is Store
  br label %28
Number of Predecessors: 1
Number of Successors: 2
  %29 = load i32, i32* %1, align 4
This is Load
  %30 = icmp slt i32 %29, 9
  br i1 %30, label %7, label %31, !llvm.loop !2
Number of Predecessors: 1
Number of Successors: 0
  %32 = load i32, i32* %1, align 4
This is Load
  %33 = add nsw i32 %32, 1
Op Code:add
This is Addition
         %32 = load i32, i32* %1, align 4
        i32 1
  store i32 %33, i32* %1, align 4
This is Store
  ret void
HelloPass runOnFunction: main
Number of Predecessors: 0
Number of Successors: 0
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  %5 = alloca i8**, align 8
  store i32 0, i32* %3, align 4
This is Store
  store i32 %0, i32* %4, align 4
This is Store
  store i8** %1, i8*** %5, align 8
This is Store
  call void @test()
  ret i32 0
```