

**ECE540 – Optimizing Compilers**  
**Assignment 2**  
**Dataflow Analysis**

## Objective

The purpose of this assignment is to build a generic dataflow problem solver (an engine) that can solve any dataflow problem when given appropriate parameters. This engine will use the code you wrote for the first assignment and may be useful for the project that you will be doing next.

## Procedure

You must first write a dataflow solver capable of solving any/all-path forward/backward dataflow problems. Your dataflow solver should operate on the control flow graph generated by your code from the first assignment. Beyond this, you are free to use any of the algorithms for dataflow analysis available although we recommend you use the iterative algorithm described in class and in the textbook.

For each procedure in the input source file, then, you will use this solver to find both the reaching definitions and the available expressions in the procedure. Writing two dataflow solvers, one for each problem, is unacceptable and will be strongly penalized.

## Output Format

For each procedure you must output:

1. A list of numbered definitions (numbering from 1)

```
definitions <procedure name> <number of definitions>
def <definition number> <variable defined> <block number containing def>
def <definition number> <variable defined> <block number containing def>
def <definition number> <variable defined> <block number containing def>
...
```

The definition numbers should be assigned in program order in each basic block. You should begin the definition numbering with 1 for each procedure, not 0. If definition I precedes definition J in a basic block, then  $I < J$ . You can visit the basic blocks in any order.

2. The reaching definitions KILL sets for each basic block

```
kill_rd_sets <procedure name> <number of basic blocks in procedure>
kill_rd <first block number> <bit vector for first block's KILL set>
```

```
kill_rd <second block number> <bit vector for second block's KILL set>
...
```

The ordering of blocks does not matter.

Each bit vector consists of a string of 0's and 1's. There should be a character for every variable in the procedure, i.e. every single string should have the same length. Do not put spaces or any other character between the 0's or 1's, and do not put brackets around the bit vector.

For example (assuming 15 definitions in the procedure):

```
kill_rd_set foo 3
kill_rd 0 000000000000000
kill_rd 1 110000001001000
kill_rd 2 000111000010001
```

3. The reaching definitions GEN sets for each basic block

```
gen_rd_sets <procedure name> <number of basic blocks in procedure>
gen_rd <first block number> <bit vector for first block's GEN set>
gen_rd <second block number> <bit vector for second block's GEN set>
...
```

The ordering of blocks does not matter. The bit vectors should be output as for the KILL set.

4. The reaching definitions at the END of each basic block

```
reaching_defs <procedure name> <number of basic blocks in procedure>
rd_out <first block number> <bit vector for first block's reaching definitions OUT set>
rd_out <second block number> <bit vector for second block's reaching definitions OUT set>
...
```

The ordering of blocks does not matter

5. Available Expressions: A list of numbered expressions (numbering from 1). You will describe the expressions by the earliest instruction that computes that expression. Earliest, in this context, refers to the instruction with the lowest instruction number.

```
expressions <procedure name> <number of expressions>
expr <first expr number (1)> <earliest instruction defining first expression>
expr <first expr number (2)> <earliest instruction defining second expression>
...
```

For example, if expression 5 is  $r1+r2$  and instructions 15 and 27 are:

```
15:    add (s.32)  t6 = r1, r2
27:    add (s.32)  t8 = r1, r2
```

Then the output line should read:

```
expr 5 15
```

because instruction 15 is the earliest appearance of the expression  $r1+r2$ .

#### 6. The EVAL sets for each basic block

```
eval_ae_sets <procedure name> <number of basic blocks in procedure>
eval_ae <first block number> <bit vector for first block's EVAL set>
eval_ae <second block number> <bit vector for second block's EVAL set>
...
```

#### 7. The available expressions KILL sets for each basic block

```
kill_ae_sets <procedure name> <number of basic blocks in procedure>
kill_ae <first block number> <bit vector for first block's KILL set>
kill_ae <second block number> <bit vector for second block's KILL set>
...
```

#### 8. The available expressions at the END of each basic block

```
available_exprs <procedure name> <number of basic blocks in procedure>
avail_exprs_out <first block number> <bit vector for first block's AEout set>
avail_exprs_out <second block number> <bit vector for second block's AEout set>
...
```

## Report

Submit a one-page report describing your implementation. This code and report are both due Friday, March 7th, 2004 by 11:59:59 pm. You may submit your report electronically along with the code, using the submit command.

In addition to the report, submit the source and executable of your implementation using the following commands:

```
tar cvf - Makefile *.c *.h | gzip -c > hw2.tar.gz
submitce540 2 hw2.tar.gz
```

After you do this, you should copy `hw2.tar.gz` to a temporary directory and uncompress it to ensure it was created correctly:

```
mkdir tmp
cp hw2.tar.gz tmp
```

```
gunzip hw2.tar.gz  
tar xvf hw2.tar
```

Try editing at least one of the files to ensure they contain code.

If there are any files required to compile your code that don't end in .c or .h, add them to the command line as well. Also add your report to the command line. The report may be in pdf, ps or Microsoft Word format.