CS480

Data Flow Analysis

Optimization

chapter 9 & 10

Today is SINGLE THREADED

April 29, 2013

Data Flow Analysis

- How does the data flow through the code?
  - with respect to control flow

Definition of variables (assign/write)

Use of variables (read)

Code Analysis

- Basic Blocks
  - in order sequence of statements
  - one entry point
  - one exit point
  - If you execute the first instruction, you execute all instructions in the BB
  - Basic Block X dominates BB Y if for every execution through the code that runs Y runs X first
- Flow Graphs

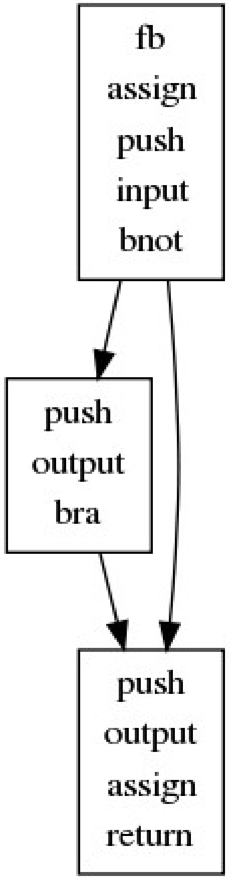
Examples - foo

```
int x;

input(&x);

if( x )
{
  output(x);
}

output(2);
```



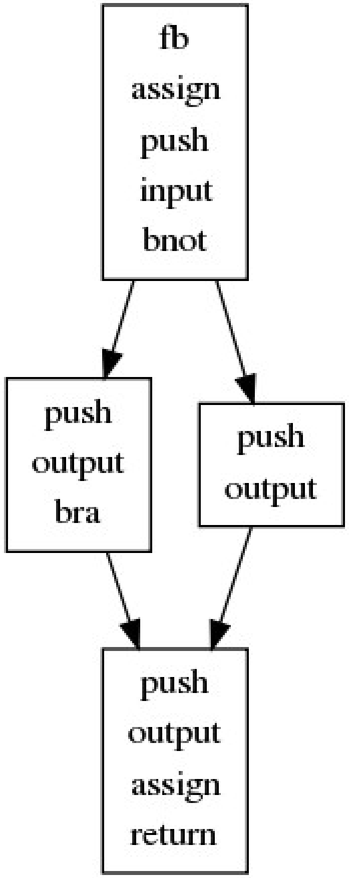
Examples - bar

```
int x;

input(&x);

if( x )
{
  output(x);
}
else
{
  output(0);
}

output(2);
```



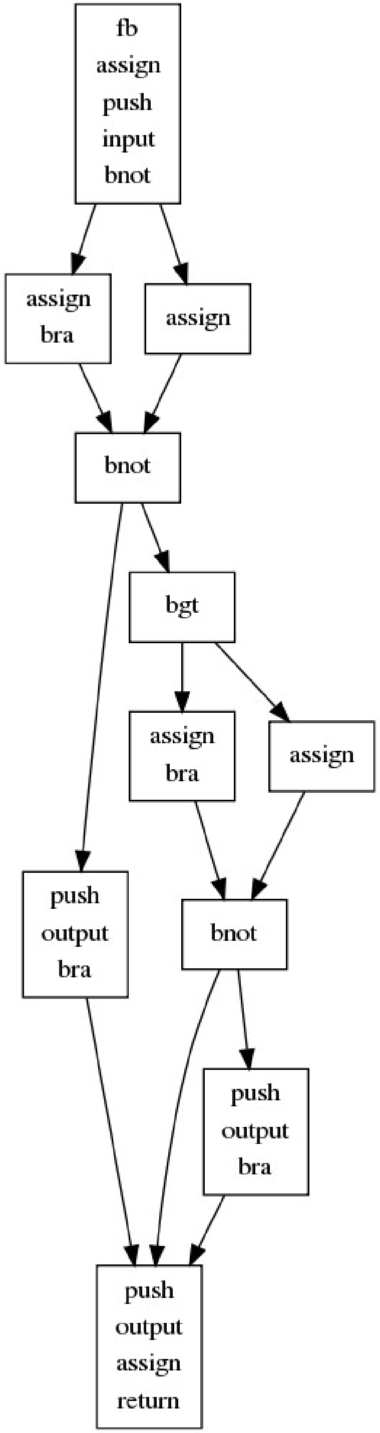
Examples - oof

```
int x;

input(&x);

if( x < 100 )
{
  output(x);
}
else if ( x > 202 )
{
  output(0);
}

output(2);
```



Data Flow Analysis

- Reaching definitions
  - – live variables
  - – definition-use chain

Basic Block B:
x = 1 // point d1
y = x // point d2
x = 2 // point d3

Which points are in gen[B]?

- Equation:
  - for basic block *S*
  - out[S] = gen[S] $\cup$ (in[S] - kill[S])

d is a point that defines a variable x
d is in gen[S] if d reaches the end of S

More Examples

```
int x , y;

input(&x);

if( x < 100 )
{
  y = 1;
}
else
{
  y = 2;
}
// does y have a value?
output(y);
```

```
int x;

input(&x);

if( x < 100 )
{
  return 1;
}
else
{
  return 2;
}

// do we need a return?
```

Examples, More

```
int x , y;

input(&x);
// can we optimize away y?
if( x < 100 )
{
  x--;
  y = 1;
}
else
{
  y = 2;
  x++
}
return x;
```

```
int x , y;

input(&x);
// can we optimize away y?
if( x < 100 )
{
  x--;
  y = 1;
}
else
{
  y = 2;
  x++
}
y = 2;
return y + x;
```

BB Transformations p 531

- Common sub expression elimination

- dead-code elimination

- renaming of temporary variables

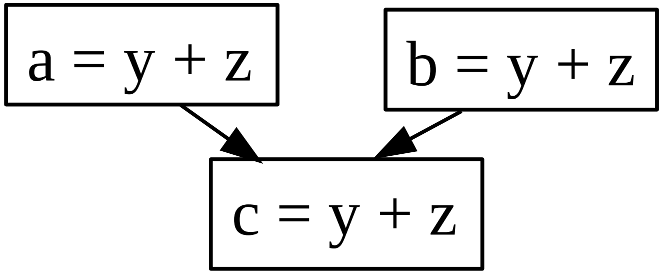
- reorder adjacent statements (to avoid stalls)

Common Subexpression Elimination

```
a = b + c
b = a - d
c = b + c
d = a - d
```

```
a = b + c
b = b + c - d
c = b + c
d = b + c - d
```

```
a = b + c
b = a - d
c = b + c
d = b
```



a = y + z

b = y + z

c = y + z

Copy Propagation

```
b = t3
a = c + d
z = b
```

Code Reordering

a = b + c
e = f + g
d = b + c

a = b + c
d = b + c
e = f + g

Why do these reorderings?

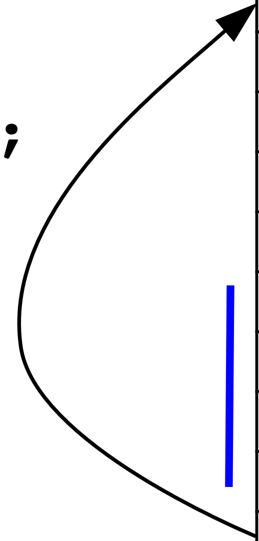
```
while( i <= limit - 2)
```

```
...
t = limit-2
while(i <=  t)
```

limit-2 must be *loop invariant*

Loop Unrolling

```
for(  x = 1;
      x < 100 ;
      x ++)
{
  output(x);
}
```



lvl		opcode	am	op1	am	op2	am	op3
2	assign	26	2	5	0	0	3	1
2	blt	10	4	1	2	3	0	56
2	assign	26	0	0	0	0	3	2
2	bra	19	0	0	0	0	0	57
2	assign	26	0	1	0	0	3	2
2	bnot	18	4	2	0	0	0	63
2	push	20	4	1	0	0	0	0
2	output	25	0	1	0	0	0	0
2	assign	26	4	1	0	0	3	3
2	increment	7	0	0	0	0	3	1
2	bra	19	0	0	0	0	0	53

Machines/Assembly Languages

```
x = 1 + y;
```

- Register Machines

- Stack Machines

Memory
Heap
Stack
Register
RAM
Cache

- Our Interpreter

Code Generation Considerations

- Instruction Selection
  - correctness
  - speed
- Register Allocation
- Instruction Ordering