Lowering C to two-operand normal form

COMP 40

October 24, 2011

Recorder:
Other group members:

Normal forms resembling machine code

Computational machine instructions resemble C assignments, except they have a very limited form:

- At most one operator appears on the right-hand side.
- If there is a binary operator, the variable assigned to is the same as the right-hand argument.
- If an access to memory is involved, typically there is no operator.

Some examples:

```
y = x + y;
y = x - y;
y = m[x+12];  // memory access: load
m[rsp-4] = x; // memory access: store
```

```
y = 17; // load 'immediate'
y = Array2_map_row_major;
y = (double) x;
```

Translation into this form is simple:

- For a complex expression like $a \times (b + c)$, simplify by first storing $(b + c)$ in a variable.
- For a "three-address" expression like `z = x + y;`, translate to two instructions:

```
z = y;
z = x + z;
```

Translation problem

Floating-point parameters are passed in registers `%xmm0` through `%xmm7`, and a floating-point result is returned in register `%xmm0`. Translate this procedure into normal form:

```
float luminance(float red, float green, float blue) {
  return 0.299 * red + 0.587 * green + 0.114 * blue;
}
```

(bonus problem on the back)

Bonus translation problem

Integer parameters are passed in registers `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, and `%r9`. Translate this procedure into normal form:

```
/* squared difference of scaled integers; denominators may differ */
double sqdiff(int n1, int d1, int n2, int d2) {
  double diff = (double)n1/(double)d1 - (double)n2/(double)d2;
  return diff * diff;
}
```