While vs. repeat loops

Repeat loops are easier to optimize: body statements dominate loop exit.

Example:

While loop

```
read(i);
while (i < 10) {
   t = a[0];
   a[i] = a[i] – t;
   i = i + 1;
}
write(t);
```

Repeat loop

```
read(i);
do {
   t = a[0];
   a[i] = a[i] – t;
   i = i + 1;
} while (i < 10)
write(t);
```

Assignment `t = a[0]` (which is **not** loop invariant):

- while loop satisfies (1) and (2) but not (3).
- repeat loop also satisfies (3).

Induction variables

Induction variable:

A variable whose value is only changed in the loop by adding (or subtracting) a constant value to it.

Example:

```
     size = 2
     i = 0
     t0 = size * 5
     t1 = t0 - 1
5:   if (i <= t1) goto 6 else goto 11
6:   t2 = i * 4
     t3 = M[a+t2]
     sum = sum + t3
     i = i + 1
     goto 5
11:  write(sum)
```

Basic induction variable:

A variable `i` whose only definition in loop is of the form

```
i = i + c
```

where `c` is constant or loop-invariant.

Derived induction variable:

A variable `j` whose only (one) definition in loop is of the form

`j = i + c` *or* `j = i * c`

where `i` is an induction variable and `c` is constant or loop-invariant.

Strength reduction:

Changes to induction variables can be made by additions and subtractions:

replace `j = i * c` by `j = j + k`

E.g., because `t2 = i * 4` is an invariant (always true):

Before

```
     size = 2
     i = 0
     t0 = size * 5
     t1 = t0 - 1
5:   if (i <= t1) goto 6
       else goto 11
6:   t2 = i * 4
     t3 = M[a+t2]
     sum = sum + t3
     i = i + 1
     goto 5
11:  write(sum)
```

After

```
     size = 2
     i = 0
     t2 = i * 4
     t0 = size * 5
     t1 = t0 - 1
6:   if (i <= t1) goto 7
       else goto 12
7:   t3 = M[a+t2]
     sum = sum + t3
     i = i + 1
     t2 = t2 + 4
     goto 6
12:  write(sum)
```

Eliminating induction variables:

All but one induction variables can be eliminated.

E.g., replace uses of `i` by uses of `t2`, where possible:

Before

```
     size = 2
     i = 0
     t2 = i * 4
     t0 = size * 5
     t1 = t0 - 1
6:   if (i <= t1) goto 7
       else goto 12
7:   t3 = M[a+t2]
     sum = sum + t3
     i = i + 1
     t2 = t2 + 4
     goto 6
12:  write(sum)
```

After

```
     size = 2
     i = 0
     t2 = i * 4
     t0 = size * 5
     t1 = t0 - 1
     t4 = t1 * 4
7:   if (t2 <= t4) goto 8
       else goto 13
8:   t3 = M[a+t2]
     sum = sum + t3
     i = i + 1
     t2 = t2 + 4
     goto 7
13:  write(sum)
```

Useless variables:

A variable v is *useless* in a loop if it is used only in definitions of v and is dead at all loop exits.

Example:

1. Delete `i = i + 1` from loop because `i` is useless variable
2. Copy propagation: replace `t2 = i * 4` by `t2 = 0`
3. Dead code elimination: delete `i = 0`

Before

```
     size = 2
     i = 0
     t2 = i * 4
     t0 = size * 5
     t1 = t0 - 1
     t4 = t1 * 4
7:   if (t2 <= t4) goto 8
       else goto 13
8:   t3 = M[a+t2]
     sum = sum + t3
     i = i + 1
     t2 = t2 + 4
     goto 7
13:  write(sum)
```

After

```
     size = 2
     t2 = 0
     t0 = size * 5
     t1 = t0 - 1
     t4 = t1 * 4
6:   if (t2 <= t4) goto 7
       else goto 11
7:   t3 = M[a+t2]
     sum = sum + t3
     t2 = t2 + 4
     goto 6
11:  write(sum)
```

Loop unrolling

Overhead in loops:

- Testing loop exit condition
- Branching to beginning of loop
- Incrementing loop counter

Overhead is repeated for each iteration of loop.

To unroll loop: put many copies of loop body in sequence. E.g.:

```
for (i=0; i<n; i++) S;        for (i=0; i<n-3; i+=4) {
                                S; S; S; S;
                              }
                              for (; i<n; i++) S;
```

Advantages:

- avoids some of the overhead

- improves instruction scheduling

Problem:

- greater code size

General method:

- unroll loop (2 copies)

```
         size = 2
         t2 = 0
         t0 = size * 5
         t1 = t0 - 1
         t4 = t1 * 4
     6:  if (t2 <= t4) goto 7 else goto 16
     7:  t3 = M[a+t2]
         sum = sum + t3
         t2 = t2 + 4
         goto 11
     11: if (t2 <= t4) goto 12 else goto 16
     12: t3 = M[a+t2]
         sum = sum + t3
         t2 = t2 + 4
         goto 6
     16: write(sum)
```

- use knowledge of induction variables

```
        size = 2
        t2 = 0
        t0 = size * 5
        t1 = t0 - 1
        t4 = t1 * 4
    6:  if (t2 <= t4-4) goto 7 else goto 13
    7:  t3 = M[a+t2]
        sum = sum + t3
        t3 = M[a+4+t2]
        sum = sum + t3
        t2 = t2 + 8
        goto 6
    13: if (t2 <= t4) goto 14 else goto 18
    14: t3 = M[a+t2]
        sum = sum + t3
        t2 = t2 + 4
        goto 14
    18: write(sum)
```