QUIZ-09-LVN-AND-LOOP-UNROLLING

Quiz question 1.

Here are two ways of unrolling a for loop; what are some of the advantages or disadvantages of each method?

```
for(...){
   a[i] = b[i] + c[i];
   i ++;
   a[i] = b[i] + c[i];
   i ++;
   a[i] = b[i] + c[i];
   i ++;
   a[i] = b[i] + c[i];
   i ++;
}
```

```
for(...){
   a[i] = b[i] + c[i];
   a[i + 1] = b[i + 1] + c[i + 1];
   a[i + 2] = b[i + 2] + c[i + 2];
   a[i + 3] = b[i + 3] + c[i + 3];
}
```

Quiz question 1.

Here are two ways of unrolling a for loop; what are some of the advantages or disadvantages of each method?

```
for(...){
   a[i] = b[i] + c[i];
   i ++;
   a[i] = b[i] + c[i];
   i ++;
   a[i] = b[i] + c[i];
   i ++;
   a[i] = b[i] + c[i];
   i ++;
}
```

May be harder to analyze for optmizations (indexing pattern analysis)
Multiple increements can be less efficient
May be harder to vectorize

Easy to understand

Quiz question 1.

Here are two ways of unrolling a for loop; what are some of the advantages or disadvantages of each method?

```
for(...){
   a[i] = b[i] + c[i];
   a[i + 1] = b[i + 1] + c[i + 1];
   a[i + 2] = b[i + 2] + c[i + 2];
   a[i + 3] = b[i + 3] + c[i + 3];
}
```

A: Easier to vectorize
A: Common expressions easily optimized
D: Harder to read
D: Greater care with index management

Quiz question 2.

Only loops without control flow in the loop body can be unrolled

[ ] True
[ ] False

Quiz question 2.

Only loops without control flow in the loop body can be unrolled

[ ] True
[v] False

Control flow most not be such that other considerations are not violated.

Loop unrolling conditions

- Several ways to unroll
    - More constraints: Simpler to unroll in code gen, more things to check
    - Less constraints: less things to check, harder to unroll in code gen

Base constraints (required for any unrolling):

Validate that we actually have an iteration variable

1. **find** candidate on lhs of assignment statement
2. **check** no assignments to candidate in body
3. **check** that it matches lhs of assignment_statement
4. **check** loop condition
    * check that candidate variable is on lhs
    * check that the rhs is a literal
      (i.e. a compile time constant value, e.g. $i<10*12$)

Loop unrolling conditions

- Simple unroll
  - Most constraints
  - Easiest code generation

For unroll factor F

Simple unroll constraints:

- Loop update increments by 1
- Find the concrete number of loop iterations (LI)
- F must divide LI evenly

Simple unroll code generation:

- create a new body = body + (update + body)*(F-1)
- perform codegen

Loop unrolling conditions

- general unroll

For unroll factor F

General unroll constraints:

- Loop update increments by 1
- Find the concrete number of loop iterations, LI

General unroll code generation:

- Create simple unrolled loop with new bound: (LI/F)*F
- Create cleanup (basic) loop with initialization: (LI/F)*F
- perform codegen

None of these numbers have to be concrete!

Quiz question 3.

Many compilers allow you to annotate loops with `pragma` operations to tell the compiler to unroll loops, and by how much. For example, in Clang, you can annotate a loop with `#pragma clang loop unroll_count(2)` to unroll a loop by a factor of 2.

Describe a case where you as a program may want to tell the compiler how many times to unroll a loop (or tell the compiler not to unroll a loop at all)

Quiz question 3.

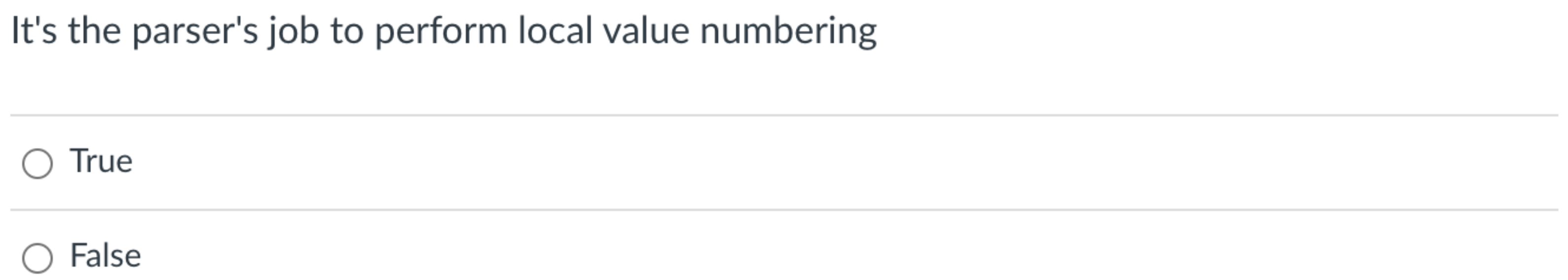
Many compilers allow you to annotate loops with `pragma` operations to tell the compiler to unroll loops, and by how much. For example, in Clang, you can annotate a loop with `#pragma clang loop unroll_count(2)` to unroll a loop by a factor of 2.

Describe a case where you as a program may want to tell the compiler how many times to unroll a loop (or tell the compiler not to unroll a loop at all)

Some Considerations may be:

* Sensitivity to Code Size (e.g. embedded applications, so you may want to restrict loop unrolling optimization)
* Pipelining considerations due to loop body interdependencies
* Possible tuning to hardware (e.g. SIMD, superscalar processors, etc.)
* Possible effects on cache locality

Quiz question 4.



It's the parser's job to perform local value numbering

○ True

○ False

Discussion (question 4)

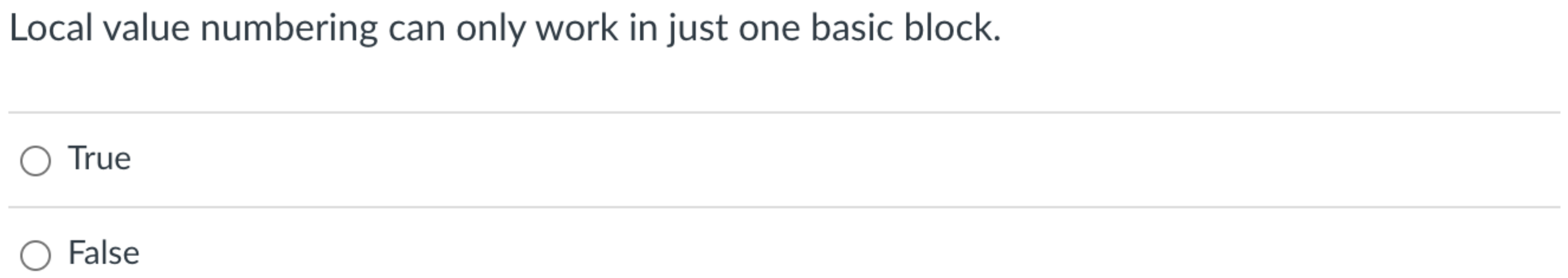
- Local value numbering operates over 3 address code
- The parser produces 3 address code
- It is very rare that the parser may use LVN, since LVN is typically performed at a later stage, but it is possible to want to do it in some cases.

https://chatgpt.com/share/68421d3d-1b80-8002-a840-fa23ba96e66e

```
→  a2 = b0 + c1;
   b4 = a2 - d3;
   c5 = b4 + c1;
   d6 = a2 - d3;
```

```
H = {
        “b0 + c1” : “a2”,
}
```

Quiz question 5.



Local value numbering can only work in just one basic block.

○ True

○ False

Discussion (question 5)

- Reminder about basic blocks

Discussion

- Programs can be split into **Basic Blocks:**
  - A sequence of 3 address instructions such that:
  - There is a single entry, single exit

- *Important property*: an instruction in a basic block can assume that all preceding instructions will execute

How might they appear in a high-level language?

How many basic blocks?

```
…
if (expr) {
    …
}
else {
   …
}
…
```

Single Basic Block

```
Label_x:
op1;
op2;
op3;
br label_z;
```

Two Basic Blocks

```
Label_x:
op1;
op2;
op3;

Label_y:
op4;
op5;
```

Discussion

```
Label_0:
x = a + b;
y = a + b;
```

optimized to →

```
Label_0:
x = a + b;
y = x;
```

```
Label_0:
x = a + b;

Label_1:
y = a + b;
```

CANNOT always optimized to →

```
Label_0:
x = a + b;

Label_1:
y = x;
```

```
br Label_1;

Label_0:
x = a + b;

Label_1:
y = a + b;
```

Quiz question 6.

After performing local value numbering (LVN) on the following program, how many operations can you save?

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

Quiz question 6.

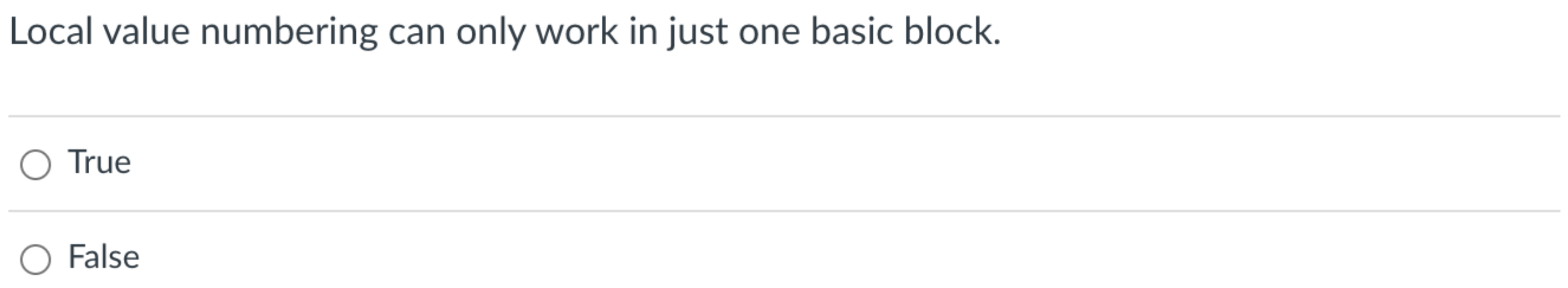
After performing local value numbering (LVN) on the following program, how many operations can you save?

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

Answer is: 2

```
a = b + c
d =  e * f
b = a
c = c + b         #  commutative of b + c but b has changed
g = d             # commutative of 3 * f, neither e nor f have changed
```

Quiz question 7.



Local value numbering can only work in just one basic block.

○ True

○ False

TRUE: Local Vaule Numbering is by definition an “local” to a basic block. There are similar techniques that go by different names that go beyond this. Such GVN (Global Value Numbering), SVN (Sparse Value Numbering), PRE (Partial Redundancy Elimination), and others. These optimization take advantage of Data Flow Analysis, and Single Static Assignment (SSA) form. You may look further in the text book or check with some AI platform for more information on this subject. e.g.

https://chatgpt.com/share/68421f7b-3d28-8002-b7bc-52961056ca2c

Quiz question 8.

Briefly describe why local value numbering is easiest applied to a single basic block. Think about the structure of an if/else statement. Knowing that structure could you do some form of local value numbering? Briefly describe how you might do it.

Quiz question 8.

Briefly describe why local value numbering is easiest applied to a single basic block. Think about the structure of an if/else statement. Knowing that structure could you do some form of local value numbering? Briefly describe how you might do it.

https://chatgpt.com/share/6842291f-9a24-8002-9126-8f3bcfe4e082

Quiz question 9

10. What is a minimal number of virtual registers needed
    for the intermediate nodes of the following expression?

```
int a, x, y;
a = ((x + 1) * y - 1) / 2.0;
```

[ ] 1
[ ] 2
[ ] 3
[ ] 4
[ ] 5
[ ] 6

Quiz question 9

10. What is a minimal number of virtual registers needed
    for the intermediate nodes of the following expression?

```
int a, x, y;
a = ((x + 1) * y - 1) / 2.0;

   v1 = (x + 1)
   v2 = v1 * y
   v3 = v2 – 1
   v4 = int2float(v3)
   v5 = v4 / 2.0
   v6 = float2int(v4)
   a = v6
```