CSE 131 Midterm (Fa16)

Ranjit Jhala

October 28th, 2016

NAME ______________________________________

SID ______________________________________

The exam is **multiple choice**, for each question circle all valid choices.

- Each question is worth **5 points**
- You will receive fractional credit for each *correct* choice
- e.g. 1/2 of the points per correct choice, if *two* valid choices.
- You will **lose one point** for each *incorrect* choice.

<table>
<thead>
<tr>
<th>Problem</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Q2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Q3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Q4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Q5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Q6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Q7</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Q8</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Q9</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Problem A: Case Study

Suppose we added a `case-of` expression to `cobra`, with the following syntax

```haskell
case e of
  e1   : e1'
  e2   : e2'
  ...
  en   : en'
  default : e'
```

For example, the below should evaluate to

- 100 if `a` equals to `x`,
- 200 if `a` is not equal to `x` but equal to `y`,
- 1000 otherwise.

```haskell
case a of
  x    : 100
  y    : 200
  default : 1000
```

Q1: Representation

To represent `case-of` expressions, we can extend our `Expr` type:

```haskell
type Id = String

data Expr = ...
  | Number Integer
  | Var Id
  | Case Expr [(Expr, Expr)] Expr
```

What is the Haskell representation of the above example?

1. Case "a"
   ```haskell```
   ```haskell```
   ```haskell```
```haskell
(Name 1000)
2. Case (Var "a")
   [ (Var "x", Number 100)
     , (Var "y", Number 200)
   ]
   (Number 1000)

3. Case (Var "a")
   [ (Var "x", Number 100)
     , (Var "y", Number 200)
     , (Var "default", Number 1000)
   ]

4. Case "a"
   [ ("x", Number 100)
     , ("y", Number 200)
     , ("default", Number 1000)
   ]

5. Case (Var "a")
   [ (Number 100, Var "x")
     , (Number 200, Var "y")
   ]
   (Number 1000)

Q2: Immediate Expressions

Suppose you have generic case-of expression:

```plaintext
case e of
  e1: e1'
  ...
  en: en'
  default: e'
```

Which sub-expressions of the above must be immediate for the above to be in A-Normal Form. That is, which sub-expressions must be immediate so that we can generate assembly for case-of expressions?

<table>
<thead>
<tr>
<th>Imm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. e</td>
</tr>
<tr>
<td>2. e1</td>
</tr>
<tr>
<td>3. e1'</td>
</tr>
<tr>
<td>4. en</td>
</tr>
<tr>
<td>5. en'</td>
</tr>
<tr>
<td>6. e'</td>
</tr>
</tbody>
</table>
Compilation

Recall again, the example `case-of` expression from above

```plaintext
case a of
  x : 100
  y : 200
  default : 1000
```

Assuming that

- `a` is at `[ebp - 4]`
- `x` is at `[ebp - 8]`
- `y` is at `[ebp - 12]`
- `z` is at `[ebp - 16]`

Fill in the blanks so that the following assembly implements the `case-of`:

```assembly
mov eax, [ebp - 4]
label_1:
  cmp eax, [ebp - 8]
?1
  mov eax, 100
?2
label_2:
  cmp eax, [ebp - 12]
?3
  mov eax, 200
?4
label_3:
  mov eax, 1000
label_done:
```

**HINT:** The next questions are all sub-parts of the above. `nop` is the assembly for “do nothing, move to next instruction”. Just figure out what the right assembly should be, and then mark the right choices. In each case below there is exactly one right answer.

**Q3, 4: Instructions ?1 and ?2**

<table>
<thead>
<tr>
<th></th>
<th>?1</th>
<th>?2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>nop</code></td>
<td><code>nop</code></td>
</tr>
<tr>
<td>2</td>
<td><code>jmp label_2</code></td>
<td><code>jmp label_2</code></td>
</tr>
</tbody>
</table>
Q5, 6: Instruction ?3 and ?4

<table>
<thead>
<tr>
<th></th>
<th>?3</th>
<th></th>
<th>?4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>nop</td>
<td>1</td>
<td>nop</td>
</tr>
<tr>
<td>2</td>
<td>jmp label_3</td>
<td>2</td>
<td>jmp label_3</td>
</tr>
<tr>
<td>3</td>
<td>jmp label_done</td>
<td>3</td>
<td>jmp label_done</td>
</tr>
<tr>
<td>4</td>
<td>je label_3</td>
<td>4</td>
<td>je label_3</td>
</tr>
<tr>
<td>5</td>
<td>je label_done</td>
<td>5</td>
<td>je label_done</td>
</tr>
<tr>
<td>6</td>
<td>jne label_3</td>
<td>6</td>
<td>jne label_3</td>
</tr>
<tr>
<td>7</td>
<td>jne label_done</td>
<td>7</td>
<td>jne label_done</td>
</tr>
</tbody>
</table>
Problem B: Stack Allocation

Consider the expression

```
let a =
    let x = 1
    in
    let y = x + 1
    in
    let z = y + 2
    in
    z + 3
in
let b = a + 1
in
    b + 2
```

Q7: Stack Positions

At what positions on the stack are the binders (variables) of the above expression stored?

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Q8: How deep is the stack?

How many slots do we need to allocate on the stack to compile the above expression? (i.e. what should `countVars` return for the above expression?)

<table>
<thead>
<tr>
<th></th>
<th>Slots</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Problem C: Boolean Comparisons

Recall that in cobra we represent booleans as 32-bit values whose Most Significant Bit (MSB) is 1 for true and 0 for false i.e. the values have the HEX representation:

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>0x80000001</td>
</tr>
<tr>
<td>false</td>
<td>0x00000001</td>
</tr>
</tbody>
</table>

Suppose we want to compute the result of the comparison

\[ \text{arg1} < \text{arg2} \]

In lecture we saw how to do so using the assembly comparisons and jumps.

Q9: Fast comparisons by bit twiddling

Here’s a different and simpler approach, that relies on the observation:

the MSB of 32-bit value is 1 exactly when the value is negative.

\[
\begin{align*}
\text{mov eax, arg1} \\
\text{sub eax, arg2} \\
?1 \text{ eax, ?3} \\
?2 \text{ eax, ?4}
\end{align*}
\]

How should we fill in the values of ?1, ?2, ?3, ?4 so that we get a sequence of assembly such that at the end, the value in eax is true if \( \text{arg1} < \text{arg2} \) and false otherwise?

NOTE: Assume there are no overflows when doing the subtraction.

<table>
<thead>
<tr>
<th></th>
<th>?1</th>
<th>?2</th>
<th>?3</th>
<th>?4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>and</td>
<td>or</td>
<td>0x800000000</td>
<td>0x000000001</td>
</tr>
<tr>
<td>2.</td>
<td>or</td>
<td>and</td>
<td>0x800000000</td>
<td>0x000000001</td>
</tr>
<tr>
<td>3.</td>
<td>and</td>
<td>or</td>
<td>0x000000001</td>
<td>0x800000000</td>
</tr>
<tr>
<td>4.</td>
<td>or</td>
<td>and</td>
<td>0x000000001</td>
<td>0x800000000</td>
</tr>
<tr>
<td>5.</td>
<td>and</td>
<td>or</td>
<td>0xFFFFFFFF</td>
<td>0x000000001</td>
</tr>
</tbody>
</table>