Assignment serie 6

Jos Bonsink (10172920) & Mustafa Karaalioglu (10217665)

26 maart 2014

Assignment 18: Code Generation

Consider the following CiviC function definition.

```
int factorial ( int x )
{
        int res;
        if ( x <= 1) res = 1;
        else res = x * factorial ( x - 1);
        return res;
}
```

a) Manually generate CiviC-VM assembly code for the above function definition making use of labels to mark destinations of jump instructions.

b) Point out the relationship between assembly code and source code through line comments in the assembly code.

```
factorial:
        //Advance the top of the stack with 1 element
        esr 1
        //Push the value of variable x on the stack
        iload 0
        //Pus the integer value of 1 on the stack
        iloadc_1
        //Check whether x is smaller or equal to one
        ile
        //If false, branch to the calculation part of the code
        branch_f 1
        //If true, assign the value one to the variable res
        iloadc_1
        //Store the integer value of one into the second variable (res)
        istore 1
```

```
//Skip the code in the else statement, by performing a jump
jump 2

1:
        //Push the value of variable x on the stack
        iload 0
        //Initiate subroutine call and prepare function arguments
        isrg
        //Push the value of variable x on the stack
        iload 0
        //Push the integer value of one on the stack
        iloadc_1
        //Subtract the value x by one
        isub
        //Jump to subroutine, the beginning of the assembly code
        jsr 1 factorial
        //Multiply the variable x with the result of the function call
        imul
        //Store the result in variable res
        istore 1

2:
        //Push the value of variable res on the stack
        iload 1
        //Return the value of variable res
        ireturn
```

---

c) Add the number of bytes required for each line of CiviC-VM assembly code. Assume here jump instructions would take byte code offsets as arguments and not labels.

d) Compute the proper byte code offset for each jump instruction; consult the CiviC-VM manual for details on individual instructions

---

```
esr 1           // 1 + 1 Byte
iload 0         // 1 + 1 Byte
iloadc_1        // 1 Byte
ile             // 1 Byte
branch_f 6      // 1 + 2 Bytes
iloadc_1        // 1 Byte
istore 1        // 1 + 1 Byte
jump 14         // 1 + 2 Bytes
iload 0         // 1 + 1 Byte
isrg            // 1 Byte
iload 0         // 1 + 1 Byte
```

```
iloadc_1        // 1 Byte
isub            // 1 Bytes
jsr 1 -26       // 1 + 3 Bytes
imul            // 1 Byte
istore 1        // 1 + 1 Byte
iload 1         // 1 + 1 Byte
ireturn         // 1 Byte
```

Assigment 19: Compilation Schemes Revisited

Devise a compilation scheme that replaces each occurrence of a for-loop in the body of a CiviC function by semantically equivalent CiviC code that makes use of a while-loop instead. As a simplification consider only for-loops without a step specification and assume that CiviC would support arbitrary interleaving of variable declarations and statements in function bodies following the example of C99.

$$\mathcal{C}\left[\begin{array}{l}\text{for (int } i\text{=}lower\text{, } upper\text{) \{}\\ Body\\ \}\\ Rest\\ \end{array}\right] \tag{1}$$

$$\Rightarrow \begin{array}{l} i = lower;\\ \text{while}(i < upper)\{\\ \mathcal{C}\left[Body\right]\\ \text{i} = \text{i} + 1;\\ \}\\ \mathcal{C}\left[Rest\right]\end{array} \quad \Bigg| \, always \tag{2}$$