Programmer som Data - Assignment 7

Bastjan Rosgaard Sejberg, Søren Kastrup, Weihao Chen Nyholm-Andersen

November 2023

8.1

I

```
24       // LDARGS
19 1 5   // CALL m a               (void main (int n))
25       // STOP                   (end of program)

15 1     // INCSP m; increase
13       // GETBP
0 1      // CSTI i                 (var i)
1        // ADD
0 0      // CSTI i                 (i=0)
12       // STI                    (store value of i)
15 -1    // INCSP m; decrease
16 43    // GOTO a;                (jump to label; #L3)
13       // GETBP
0 1      // CSTI i
1        // ADD
11       // LDI                    (load the value of i)
22       // PRINTI                 (print i)
15 -1    // INCSP m; decrease
13       // GETBP
0 1      // CSTI i
1        // ADD
11       // LDI                    (i = i + 1)
22       // PRINTI
15 -1    // INCSP m; decrease
13       // GETBP
0 1      // CSTI i;
1        // ADD
13       // GETBP
0 1      // CSTI
1        // ADD
11       // LDI
0 1      // CSTI
1        // ADD
12       // STI
15 -1    // INCSP; decrease
15 0     // INCSP; increase

13       // GETBP;                 (Label L3)
0 1      // CSTI
1        // ADD
11       // LDI
13       // GETBP
0 0      // CSTI
1        // ADD
11       // LDI
7        // LT                     (while (1 < n))
18 18    // IFNZERO a              (conditional jump if the result of comparison is non-zero)
15 -1    // INCSP; decrease
21 0     // RET m                  (returning from the function)
```

Listing 1: ex3_byte

```
// INITIALIZATION
24       // LDARGS
19 1 5   // CALL m a               (void main (int n))
25       // STOP

// MAIN
15 1     // INCSP; increase
13       // GETBP
0 1      // CSTI                   (int r;)
1        // ADD
13       // GETBP
0 0      // CSTI
1        // ADD
11       // LDI
12       // STI                    (r = n)
15 -1    // INCSP; decrease
15 1     // INCSP; increase
13       // GETBP
0 0      // CSTI                   (int r; (within inner block))
1        // ADD
11       // LDI
13       // GETBP
0 2      // CST
1        // ADD
19 2 57  // CALL m a               (square(n, &r))
15 -1    // INCSP; decrease
13       // GETBP
0 2      // CSTI
1        // ADD
11       // LDI
22       // PRINTI
15 -1    // INCSP; decrease
15 -1    // INCSP; decrease
13       // GETBP
0 1      // CSTI
1        // ADD
11       // LDI
22       // PRINTI                 (print r)

// END MAIN
15 -1    // INCSP; decrease
15 -1    // INCSP; decrease
21 0     // RET m

// START SQUARE
13       // GETBP
0 1      // CSTI
1        // ADD
11       // LDI
13       // GETBP
0 0      // CSTI
1        // ADD
11       // LDI                    (*rp)
13       // GETBP
0 0      // CSTI                   (deref *rp )
1        // ADD
11       // LDI
3        // MUL                    (i * i)
12       // STI                    (store result in *rp)

// END SQUARE
15 -1    // INCSP; decrease
15 0     // INCSP; increase
21 1      // RET m
```

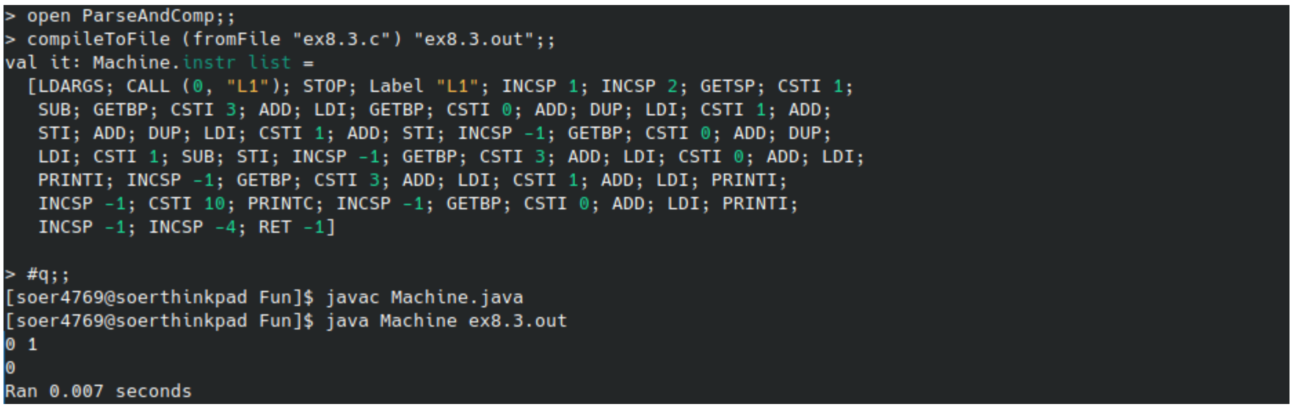
Listing 2: ex5_byte

II

```
/*
    The brackets indicate contents of
    registers.
    For example, in step 1 (CALL 1 5), on
    the left side,
    it shows the loaded args, which is 4 (
    as the provided argument).
    Around step 56, we have [ 4 -999 4 4
    ]{56: INCSP -1}, meaning
    i = 4 and n = 4, thus ends the loop.
*/

// Starting program and calling main
[ ]{0: LDARGS}
[ 4 ]{1: CALL 1 5}

// Setting up variables and entering loop
[ 4 -999 4 ]{5: INCSP 1}
[ 4 -999 4 0 ]{7: GETBP}
[ 4 -999 4 0 2 ]{8: CSTI 1}
[ 4 -999 4 0 2 1 ]{10: ADD}
[ 4 -999 4 0 3 ]{11: CSTI 0}
[ 4 -999 4 0 3 0 ]{13: STI}
[ 4 -999 4 0 0 ]{14: INCSP -1}
[ 4 -999 4 0 ]{16: GOTO 43}

// Start of the loop
[ 4 -999 4 0 ]{18: GETBP}
[ 4 -999 4 0 2 ]{19: CSTI 1}
[ 4 -999 4 0 2 1 ]{21: ADD}
[ 4 -999 4 0 3 ]{22: LDI}
[ 4 -999 4 0 0 ]{23: PRINTI}
0 [ 4 -999 4 0 0 ]{24: INCSP -1}
[ 4 -999 4 0 ]{26: GETBP}
[ 4 -999 4 0 2 ]{27: CSTI 1}
[ 4 -999 4 0 2 1 ]{29: ADD}
[ 4 -999 4 0 3 ]{30: GETBP}
[ 4 -999 4 0 3 2 ]{31: CSTI 1}
[ 4 -999 4 0 3 2 1 ]{33: ADD}
[ 4 -999 4 0 3 3 ]{34: LDI}
[ 4 -999 4 0 3 0 ]{35: CSTI 1}
[ 4 -999 4 0 3 0 1 ]{37: ADD}
[ 4 -999 4 0 3 1 ]{38: STI}
[ 4 -999 4 1 1 ]{39: INCSP -1}
[ 4 -999 4 1 ]{41: INCSP 0}

// Loop header
[ 4 -999 4 1 ]{43: GETBP}
[ 4 -999 4 1 2 ]{44: CSTI 1}
[ 4 -999 4 1 2 1 ]{46: ADD}
[ 4 -999 4 1 3 ]{47: LDI}
[ 4 -999 4 1 1 ]{48: GETBP}
[ 4 -999 4 1 1 2 ]{49: CSTI 0}
[ 4 -999 4 1 1 2 0 ]{51: ADD}
[ 4 -999 4 1 1 2 ]{52: LDI}
[ 4 -999 4 1 1 4 ]{53: LT}
[ 4 -999 4 1 1 ]{54: IFNZRO 18}
...
[ 4 -999 4 4 0 ]{54: IFNZRO 18}

// Ending the program
[ 4 -999 4 4 ]{56: INCSP -1}
[ 4 -999 4 ]{58: RET 0}
[ 4 ]{4: STOP}
```

Listing 3: ex3_byte

8.3



```
> open ParseAndComp;;
> compileToFile (fromFile "ex8.3.c") "ex8.3.out";;
val it: Machine.instr list =
  [LDARGS; CALL (0, "L1"); STOP; Label "L1"; INCSP 1; INCSP 2; GETSP; CSTI 1;
   SUB; GETBP; CSTI 3; ADD; LDI; GETBP; CSTI 0; ADD; DUP; LDI; CSTI 1; ADD;
   STI; ADD; DUP; LDI; CSTI 1; ADD; STI; INCSP -1; GETBP; CSTI 0; ADD; DUP;
   LDI; CSTI 1; SUB; STI; INCSP -1; GETBP; CSTI 3; ADD; LDI; CSTI 0; ADD; LDI;
   PRINTI; INCSP -1; GETBP; CSTI 3; ADD; LDI; CSTI 1; ADD; LDI; PRINTI;
   INCSP -1; CSTI 10; PRINTC; INCSP -1; GETBP; CSTI 0; ADD; LDI; PRINTI;
   INCSP -1; INCSP -4; RET -1]

> #q;;
[soer4769@soerthinkpad Fun]$ javac Machine.java
[soer4769@soerthinkpad Fun]$ java Machine ex8.3.out
0 1
0
Ran 0.007 seconds
```

*Refer to the files **Absyn.fs**, **CPar.fsy**, **Comp.fs** and **ex8.3.c** in the folder **Exercise_8.3**.*

8.4

A

```
// Setup
24           // LDARGS;          (args)
19 0 5       // CALL 0 5;        (call main)
25           // STOP;

// Main
15 1         // INCSP 1;   #L1
13           // GETBP;
0 0          // CSTI 0;          (i)
1            // ADD;
0 20000000   // CSTI 20000000;   (add to i)
12           // STI;
15 -1        // INCSP -1;
16 35        // GOTO 35;         (goto #L3)

// Loop i != 0
13           // GETBP;     #L2
0 0          // CSTI 0;          (get i)
1            // ADD;
13           // GETBP;
0 0          // CSTI 0;          (get i)
1            // ADD;
11           // LDI;
0 1          // CSTI 1;          (val 1)
2            // SUB;             (i = i-1)
12           // STI;
15 -1        // INCSP -1;
15 0         // INCSP 0;
13           // GETBP;     #L3
0 0          // CSTI 0;          (get i)
1            // ADD;
11           // LDI;
18 18        // IFNZRO 18;       (goto #L2,
                                  if i != 0)
// End
15 -1        // INCSP -1;
21 -1        // RET -1;
```

Listing 4: ex8.c

```
// Setup
0 20000000  // CSTI 20000000; (i)
16 7        // GOTO 7;        (goto #L2)

// Loop i != 0
0 1         // CSTI 1;   #L1  (val 1)
2           // SUB;
9           // DUP;           (i -= 1)
18 4        // IFNZRO 4; #L2  (goto #L1,
                               if i != 0)
// End
25          // STOP;
```

Listing 5: prog1

By comparing the symbolic bytecode above, it can be seen that the program **ex8.c** is much slower because it needs to get the value of the variable stored on the stack thrice (one for updating the value, one for getting the existing value to subtract one, and one for checking if it is non-zero), when **prog1** only needs to get the value stored once to do all of these steps at once - which for both programs is done in every round of their loops until a variable reaches zero.

Furthermore the program **ex8.c** first declares the variable on the stack which automatically sets it to zero and then afterwards initializes it to the desired value, where **prog1** declares and initializes it to its desired value all at once saving valuable time.

B

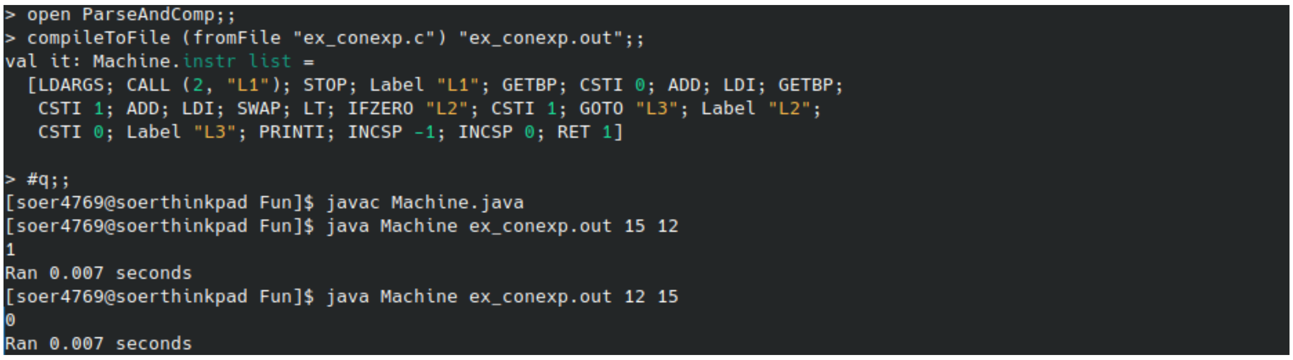
```
// Setup
24            // LDARGS;          (args: j)
19 1 5        // CALL 1 5;        (call main)
25            // STOP;

// Main
15 1          // INCSP 1;    #L1
13            // GETBP;
0 1           // CSTI 1;           (i)
1             // ADD;
0 1889        // CSTI 1889;        (add to i)
12            // STI;
15 -1         // INCSP -1;
16 95         // GOTO 95;          (goto #L3)

// Loop body top (set i = i+1)
13            // GETBP;      #L2
0 1           // CSTI 1;           (get i)
1             // ADD;
13            // GETBP;
0 1           // CSTI 1;           (get i)
1             // ADD;
11            // LDI;
0 1           // CSTI 1;           (val 1)
1             // ADD;              (i = i+1)
12            // STI;
15 -1         // INCSP -1;

// Loop if check
// (i % 4 == 0
// && (i % 100 != 0 || i % 400 == 0))
13            // GETBP;
0 1           // CSTI 1;          (get i)
1             // ADD;
11            // LDI;
0 4           // CSTI 4;          (val 4)
5             // MOD;             (i % 4)
0 0           // CSTI 0;          (val 0)
6             // EQ;              (i%4==0)
17 77         // IFZERO 77;       (goto #L7
13            // GETBP;            if false)
0 1           // CSTI 1;          (get i)
1             // ADD;
11            // LDI;
0 100         // CTSI 100;        (val 100)
5             // MOD;             (i % 100)
0 0           // CSTI 0;          (val 0)
6             // EQ;              (i%100==0)
8             // NOT;             (i%100!=0)
18 73         // IFNZRO 73;       (goto #L9
13            // GETBP;            if true)
0 1           // CSTI 1;          (get i)
1             // ADD;
11            // LDI;
0 400         // CSTI 400;        (val 400)
5             // MOD;             (i % 400)
0 0           // CSTI 0;          (val 0)
6             // EQ;              (i%400==0)
16 75         // GOTO 75;         (goto #L8)
0 1           // CSTI 1;    #L9
16 79         // GOTO 79;   #L8   (goto #L6)
0 0           // CSTI 0;    #L7
17 91         // IFZERO 91; #L6   (goto #L4)

// Loop if body (print i)
13            // GETBP;
0 1           // CSTI 1;           (get i)
1             // ADD;
11            // LDI;
22            // PRINTI;           (print i)
15 -1         // INCSP -1;
16 93         // GOTO 93;          (goto #L5)
15 0          // INCSP 0;   #L4
15 0          // INCSP 0;   #L5

// Loop check (i < j)
13            // GETBP;      #L3
0 1           // CSTI 1;           (get i)
1             // ADD;
11            // LDI;
13            // GETBP;
0 0           // CSTI 0;           (get j)
1             // ADD;
11            // LDI;
7             // LT;               (i < j)
18 18         // IFNZRO 18;        (goto #L2
                                    if true)
// End
15 -1         // INCSP -1;
21 0          // RET 0;
```

Listing 6: ex13.c

The symbolic bytecode for **ex13.c** tells how the structure for loops and conditional statement interact in this kind of programming environment, which is kinda similar to how it is done in *c* but different enough that it needs an explanation. Basically the flow of the code starts out how one would normally expect, by the setup process retrieving the program's argument numeric variable *j* and calling the main method wherein it creates an interval variable *i* with a value of *1889*. Next it uses a *goto* statement to jump to a later section that acts as the header for a while loop like structure, which checks if the value of *j* or larger than *i*, if that is the case it jumps yet again to the section below the *goto* statement from earlier, if not the program terminates.

From here on out the mid section of the bytecode will run continuously until the statement turns false, with the top section of the loop getting the value of the variable *i* twice as to be able to increase its value by one until it becomes larger than *j* so that the program can terminate, as the shorthand version *––i* is not used in this program. Lastly the loop contains a conditional if-statement that if true prints the current value of *i* to the screen, if not the loops continues around this section to the loop check. The conditional statement is made out of two section and three checks, the first jumps to label *#L7→#L6→#L4* if the variable *i* % 4 is not true, the second jumps *#L9→#L8→#L6* and prints *i* if *i* % 100 is true as this is an or-statement, so if it instead it is false the third statement computes *i* % 400 before jumping *#L8→#L6→#L4* where #L6 checks if this value is false else it prints *i*.

8.5



```
> open ParseAndComp;;
> compileToFile (fromFile "ex_conexp.c") "ex_conexp.out";;
val it: Machine.instr list =
  [LDARGS; CALL (2, "L1"); STOP; Label "L1"; GETBP; CSTI 0; ADD; LDI; GETBP;
   CSTI 1; ADD; LDI; SWAP; LT; IFZERO "L2"; CSTI 1; GOTO "L3"; Label "L2";
   CSTI 0; Label "L3"; PRINTI; INCSP -1; INCSP 0; RET 1]

> #q;;
[soer4769@soerthinkpad Fun]$ javac Machine.java
[soer4769@soerthinkpad Fun]$ java Machine ex_conexp.out 15 12
1
Ran 0.007 seconds
[soer4769@soerthinkpad Fun]$ java Machine ex_conexp.out 12 15
0
Ran 0.007 seconds
```

*Refer to the files **Absyn.fs**, **CLex.fsy**, **CPar.fsy**, **Comp.fs** and **ex_conexp.c** in the folder **Exercise_8.5**.*

8.6