4th Practical Exercise
Translation to MIPS

Lecture Compilers SS 2011

Jan Schäfer

Software Technology Group
TU Kaiserslautern

Grammar Extension

MiniJava.cup

```
ExpressionStatement ::= ...
    | Expression:l EQ Expression:r SEMI
      {: RESULT = Assignment(SourcePosition(lleft, lright),l,r); :}
    ;
```

MiniJava.katja

```
Statement = ...
          | Assignment ( SourcePosition sourcePos,
                         Expression left, Expression right )
```

MIPS-AST Changes

MIPS.katja

```
Instruction = ...
   | MUL     (Register reg0, Register reg1, Register reg2)
   | DIV     (Register reg0, Register reg1, Register reg2)
   | XOR     (Register reg0, Register reg1, Register reg2)
```

Unparser.java

1. Add new cases, remove MULT
2. load $\Rightarrow$ lw
3. store $\Rightarrow$ sw

Translation of Local Variables

```
static void m() {
   boolean b = true;
   int i = 5;
   if (b) {
      int j = 42;
   } else {
      int k = -1;
   }
}
```

Translation of Local Variables

```
static void m() {
   boolean b = true;
   int i = 5;
   if (b) {
      int j = 42;
   } else {
      int k = -1;
   }
}
```

Solution 1

```
# Space on the stack for
# 4 local variables
# sp + 12 : k
# sp + 8  : j
# sp + 4  : i
# sp + 0  : b
addi    $sp, $sp, -16
```

Translation of Local Variables

```
static void m() {
   boolean b = true;
   int i = 5;
   if (b) {
      int j = 42;
   } else {
      int k = -1;
   }
}
```

Solution 1

```
# Space on the stack for
# 4 local variables
# sp + 12 : k
# sp + 8  : j
# sp + 4  : i
# sp + 0  : b
addi    $sp, $sp, -16
```

Solution 2

```
# Space on the stack for
# 4 local variables
# sp + 8  : j/k
# sp + 4  : i
# sp + 0  : b
addi    $sp, $sp, -12
```

Translation of Recursive Procedures

Translation of Recursive Procedures

1. Code generation at the definition site
2. Code generation at the call site

Code Generation at the Definition Site

Code Generation at the Definition Site

1. Prolog
2. Body
3. Epilog

Prolog, Body, Epilog

Prolog

Prolog, Body, Epilog

Prolog

1. Label for jumping
   `some_procedure:`
2. Save return address ($ra) to stack when modified by subcalls
3. Save $fp to stack
4. Save $sp to stack
5. Save used saved temporaries ($s0-$s7) to stack

Body

Prolog, Body, Epilog

Prolog

1. Label for jumping
   `some_procedure:`
2. Save return address ($ra) to stack when modified by subcalls
3. Save $fp to stack
4. Save $sp to stack
5. Save used saved temporaries ($s0-$s7) to stack

Body

translate statements

Epilog

Epilog

1. Put return values into $v0, $v1
2. Restore saved registers from stack
3. `jr $ra`

Code Generation at the Call Site

Code Generation at the Call Site

1. save used temporary registers ($t0-$t9) to stack
2. save arguments to stack or to argument registers ($a0-$a3)
3. `jal proclabel`
4. restore $t0-$t9