Lecture #26

Code Generation

Code Generation

- Code that can be executed on a real machine
  - The MIPS processor
- We will simulate a stack machine model using MIPS instructions and registers
- The accumulator is kept in MIPS register $a0

- The stack is kept in memory
  - The stack grows towards lower addresses in MIPS
- Address of the next location on stack is kept in register $sp
  - Top of the stack is at address $sp + 4

- MIPS uses RISC processor model
- 32 general purpose registers (32 bits each)
- We use $sp, $a0 and $t1 (a temporary register)

Code Generation

- lw reg1 offset(reg2)
  - Load 32-bit word from address reg_2 + offset into reg_1
- add reg_1 reg_2 reg_3
  - $reg_1 \leftarrow reg_2 + reg_3$
- sw reg1 offset(reg_2)
  - Store 32-bit word in reg_1 at address reg_2 + offset
- addiu reg_1 reg_2 imm
  - $reg_1 \leftarrow reg_2 + imm$
  - "u" means overflow is not checked
- li reg imm
  - $reg \leftarrow imm$

Code Generation

- Stack-machine code for **7 + 5** in MIPS

acc ← 7	li	$a0	7	
push acc	sw	$a0	0($sp)	
	addiu	$sp	$sp -4	
acc ← 5	li	$a0	5	
acc ← acc + top_of_stack	lw	$t1	4($sp)	
	add	$a0	$a0	$t1
pop	addiu	$sp	$sp	4

Code Generation

- A language with integers and integer operations

$P \rightarrow D; P \mid D$

- A program consists of a list of declarations

$D \rightarrow \text{def id(ARGS)} = E;$

$\text{ARGS} \rightarrow \text{id, ARGS} \mid \text{id}$

- A declaration is a function definition.
- The function takes a list of identifiers as arguments.
- The function body is an expression.

$E \rightarrow \text{int} \mid \text{id} \mid \text{if } E_1 = E_2 \text{ then } E_3 \text{ else } E_4$

$\mid E_1 + E_2 \mid E_1 - E_2 \mid \text{id}(E_1, \ldots, E_n)$

- Expressions are integers, identifiers, if-then-else with a predicate which allows the equality test, sums and differences of expressions and function calls.

- The first function definition in the list is the entry point, that is the *main* routine.

Code Generation

- This language may be used to define the fibonacci function:

```
def fib(x) = if x = 1 then 0 else
        if x = 2 then 1 else
            fib(x - 1) + fib(x - 2)
```

- To generate code for this language, we generate MIPS code for each expression **e** that:
    - Computes the value of **e** in **$a0**
    - Preserves **$sp** and the contents of the stack

- We define a code generation function **cgen(e)** whose result is the code generated for **e**

Code Generation

- **cgen(e)** is going to work by cases.
- The code to evaluate a constant simply copies it into the accumulator:
  - `cgen(i) = li $a0 i`
  - *This preserves the stack, as required*

- $\texttt{cgen(}e_1 + e_2\texttt{) =}$

```
cgen(e1)
sw $a0 0($sp)
addiu $sp $sp - 4
cgen(e2)
lw $t1 4($sp)
add $a0 $t1 $a0
addiu $sp $sp 4
```

- The code for **+** is a template with “holes” for code for evaluating e_1 and e_2
- Stack machine code generation is recursive
- Code generation for expressions can be done as a recursive-descent of the AST

Code Generation

- MIPS instruction: `sub` $\texttt{reg}_1$ $\texttt{reg}_2$ $\texttt{reg}_3$
  - Implements $reg_1 \leftarrow reg_2 - reg_3$

- `cgen(`e_1 `-` e_2`) =`

```
cgen(e1)
sw $a0 0($sp)
addiu $sp $sp - 4
cgen(e2)
lw $t1 4($sp)
sub $a0 $t1 $a0
addiu $sp $sp 4
```

Code Generation

- Write MIPS assembly code for the given expressions following the 1-register stack machine model:
    - 1 + (2 - 3)
    - (5 - 4) + 3