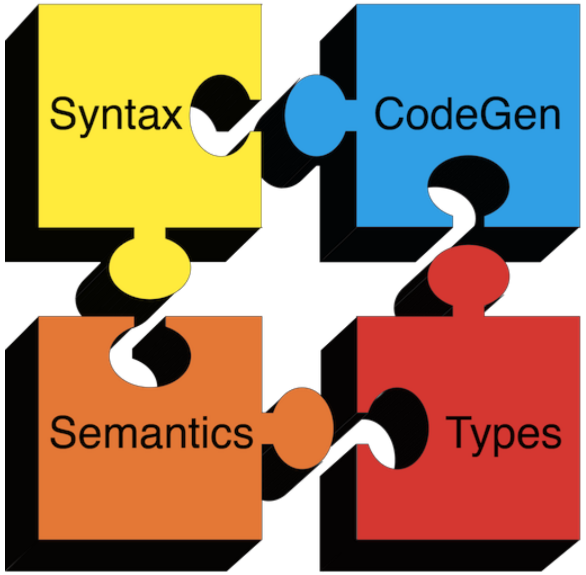

Compilers

Introduction to Code Generation

Code Generation

- We focus on generating code for a stack machine with accumulator

- We want to run the resulting code on a real machine
  - e.g., the MIPS processor (or simulator)

- We simulate stack machine instructions using MIPS instructions and registers

Code Generation

- The accumulator is kept in MIPS register $a0
- The stack is kept in memory
  - The stack grows towards lower addresses
  - Standard convention on MIPS
- The address of the next location on the stack is kept in MIPS register $sp stack pointer
  - The top of the stack is at address $sp + 4

Code Generation

- MIPS architecture
  - Prototypical Reduced Instruction Set Computer (RISC)
  - Most operations use registers for operands & results
  - Use load & store instructions to use values in memory
  - 32 general purpose registers (32 bits each)
    - We use $sp, $a0 and $t1 (a temporary register)
- Read the SPIM documentation for details

  ↳ simulator

Code Generation

- $\underline{\text{lw}}\ \text{reg}_1\ \underline{\overset{\#}{\text{offset}}}(\underline{\text{reg}_2})$
  - Load 32-bit word from address reg_2 + offset into reg_1
- add reg_1 $\underline{\text{reg}_2}$ $\underline{\text{reg}_3}$
  - $\text{reg}_1 \leftarrow \text{reg}_2 + \text{reg}_3$
- $\underline{\text{sw}}\ \underline{\text{reg}_1}\ \underline{\text{offset}}(\underline{\text{reg}_2})$
  - Store 32-bit word in reg_1 at address reg_2 + offset
- $\text{add}\underline{\text{iu}}\ \text{reg}_1\ \underline{\text{reg}_2}\ \underline{\overset{\#}{\text{imm}}}$
  - $\text{reg}_1 \leftarrow \text{reg}_2 + \text{imm}$
  - "u" means overflow is not checked
- $\text{li}\ \underline{\text{reg}}\ \underline{\overset{\#}{\text{imm}}}$
  - $\text{reg} \leftarrow \text{imm}$

Code Generation

The 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
```