Intermediate Representations (CH 10)

Compilation process:
- scanner - tokens
- parser - AST
- semantic - AST with attributes
- optimization
- code generation

Often optimization and code generation use some intermediate representation other than the AST.

```
a <-- b * c + b * d
```

- Abstract syntax trees
- Triples
- Quadruples

1) (*,b,c)  1) (*, b, c, t1)
2) (*,b,d)  2) (*, b, d, t2)
3) (+,(1),(2))  3) (+, t1, t2, t3)
4) (<--,a,(3))  4) (<--, a, t3, --)

(arbitrary number of temps)
Compilers:

GCC:
- Front End: Language -> AST
- Next: AST -> RTL (Register Transfer Language, A tuple space)
- Opt: RTL -> RTL
- CG: RTL -> Machine language
- CG and front end changeable

- TeX - uses .dvi (device independent)
- Ada - uses DIANA (Descriptive Intermediate Attributed Notation for Ada)
- LLVM - AST -> LLVM IR (a low-level RISC-like virtual instruction set)

- Often called "Middle-end", provides for machine independent transformations and optimizations
Java:
- result is java byte code
- JVM

hc: also could be considered an IR
- stack based may not be best IR

GNU bc: uses a byte code for execution

Portable Pascal: from 1975 era, byte code, p-machine

Bytecode translation
- translate bytecode to real CPU
- just in time compiler (JIT)
  - JVM byte code can be considered a middle-end IR
- Book spends quite a bit of time on the JVM
- Portable Pascal: distribute byte code
  - write bytecode interpreter or a translator, have a working Pascal system
Target ("Real") Code Generation -- Ch 13

AST or IR -> real machine instructions (typically an assembly language)
Issues:
- instruction selection and address selection
- register allocation
- code scheduling

Registers -- "temporary storage"
- General purpose (allocatable registers)
- Special purpose (address, other)
- Reserved (e.g. stack pointer ...)
- Volatile (used for very short sequences)

void get_temp()
void free_temp()

volatile
- a <-- b (quick temporary use)
- built-in registers (multi-word moves)
- Special instructions, sobgtr
 Registers

- Register spills
  - no more available
  - use memory for spills (local AR)
  - reduce these

- live registers & temps
- dead registers & temps

Basic Block -- a block of code
- with no jumps out (except at end)
- with no jumps in (except at beginning)

- May have computations reordered!
  - (result must be the same)
Code Generation and Local Optimization

- Tuple -> machine language
- 3 major tasks
  - instruction selection
  - address-mode selection
  - register allocation

Example Basic Block:

\[
\begin{align*}
  a & \leftarrow b \times c + d \times e; \\
  d & \leftarrow c + (d - b); \\
  f & \leftarrow e + a + c; \\
  a & \leftarrow d + e;
\end{align*}
\]

- live/dead analysis
  - a from first line never needs to be stored to memory
Register Allocation

- Most modern CPUs have registers, not stack based
- Intermediate results end up in registers
- Register needs for expressions:
  - \((a-b) + ((c+d) + (e \times f))\)
  - left to right evaluation -- needs 4
  - out of order -- needs 3
- Sethi-Ullman numbering algorithm can detect minimum needed
- Commutative operators: \(a+1\) vs \(1+a\) ... can be exploited for fewer registers on some machines
- Associative operators: \(a+b+c\) as \((a+b)+c\) vs \(a+(b+c)\) may not really be associative
- "On the fly register allocation: getReg, freeReg"
- Register Allocation by graph coloring (read book)
- Priority-Based register allocation (read book)
- inter-procedural register allocation (read book)
Code Scheduling

- Many modern architectures are a pipelined architecture
  - several instructions are being executed at the same time
    - Can you use the result of a previous instruction in the next one?
    - Stalled pipeline -- pipeline has to wait which stops all instructions
    - schedule code to reduce pipeline stalls
    - code scheduling is typically a basic block issue
  - a jump instruction won’t transfer control immediately
    - n instructions after jump will still be executed
    - typical to put "no-ops" as the instructions
    - straight jump -- duplicate starting instructions
    - conditional jump -- common expression in both choices?
    - move final instruction or two to after jump?
  - Book talks about various approaches to code scheduling