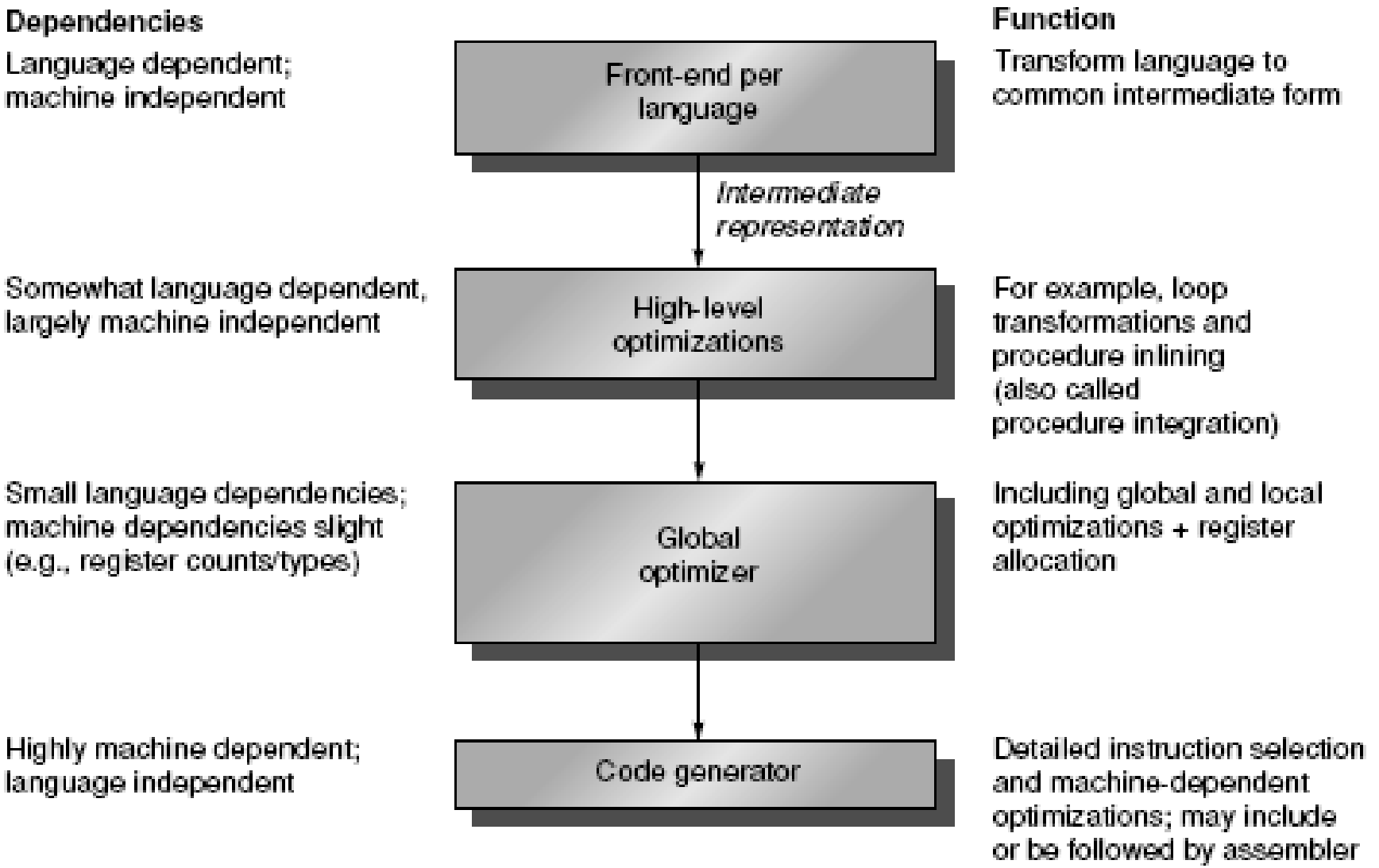
Instruction Set Principles
Appendix A

- Classifying ISA
- Design alternatives available to the instruction set architect
- Role of Compiler
- The RISC-V ISA

The Role of Compilers

- **Goals of a compiler**
  - **Correctness**
  - **Speed of compiled code**
    - **Fast compilation**
    - **Debugging support**
    - **Interoperability among languages**
- ***Passes reduce complexity but enforces ordering of some transformations over others***
  - ***Phase ordering problem* is encountered**
    - **Procedure inlining**
- **Global common sub-expression elimination**
  - **For optimized performance, register allocation should be done**
- **Classification of Optimizations**
  - ***ISA has a greater impact on optimizations***

Passes of compilers



Slide credit : Prof. Dr. Hasina Khatoon

Major types of compiler optimizations

Optimization name	Explanation	Percentage of the total number of optimizing transforms
High-level	**At or near the source level; processor-independent**	
Procedure integration	Replace procedure call by procedure body	N.M.
Local	**Within straight-line code**	
Common subexpression elimination	Replace two instances of the same computation by single copy	18%
Constant propagation	Replace all instances of a variable that is assigned a constant with the constant	22%
Stack height reduction	Rearrange expression tree to minimize resources needed for expression evaluation	N.M.
Global	**Across a branch**	
Global common subexpression elimination	Same as local, but this version crosses branches	13%
Copy propagation	Replace all instances of a variable A that has been assigned X (i.e., $A = X$) with X	11%
Code motion	Remove code from a loop that computes same value each iteration of the loop	16%
Induction variable elimination	Simplify/eliminate array-addressing calculations within loops	2%
Processor-dependent	**Depends on processor knowledge**	
Strength reduction	Many examples, such as replace multiply by a constant with adds and shifts	N.M.
Pipeline scheduling	Reorder instructions to improve pipeline perfomance	N.M.
Branch offset optimization	Choose the shortest branch displacement that reaches target	N.M.

Slide credit : Prof. Dr. Hasina Khatoon

The Role of Compilers

- **Register Allocation:**
  - **How many registers are sufficient?**
    - **Allocation uses *graph coloring technique***
      - **Works well only when the number of registers is greater than 16**
- **Optimized code gives better and accurate analysis of frequency of occurrence**

- **How can the Architect help the Compiler Writer?**
- **Use basic principle**

  Make the frequent case fast and make the rare case correct
- **Instruction set properties that make it easier for a compiler to generate efficient and correct code**

Slide credit : Prof. Dr. Hasina Khatoon

The Role of Compilers

- **Desired Instruction Set properties that help the compiler writer:**

1. ***Provide Regularity***

   Primary components of an instruction set should be *orthogonal*

   Helps simplify code generation

2. ***Provide primitives, not solutions***

   Special features that "match" a particular semantics are rarely used

3. ***Simplify tradeoffs among alternatives***

   Cache and pipelining have made the problem more complex

4. ***Provide instructions that bind the quantities known at compile time as constants***

Slide credit : Prof. Dr. Hasina Khatoon

The Role of Compilers

- **Compiler Support (or Lack of) for Multimedia Instructions**
- **SIMD instructions are rarely used**
  - **SIMD instructions tend to be solutions**
    - **These are not primitive operations**
  - **Vector architecture have vector registers and their own compilers to support SIMD operations**
  - ***Gather/Scatter* operations add to the overhead**
- **SIMD instructions may be found in hand-coded libraries rather than in compiled code**

Slide credit : Prof. Dr. Hasina Khatoon