Code Optimization

COMP360

"There is no programming language–no matter how structured–that will prevent programmers from making bad programs."

Larry Flon

Exam 1

- The first COMP360 exam will be on Monday, February 20, 2017
- You are allowed to have one 8½ by 11" page of notes

Stages of a Compiler

- Source preprocessing
- Lexical Analysis (scanning)
- Syntactic Analysis (parsing)
- Semantic Analysis
- Optimization
- Code Generation
- Link to libraries

Generating Good Code

- When a programmer writes a program in a high level language, they want the compiler to create a compact, fast, efficient executable
- Most compilers do a little optimization by default and will do much more if compile options request it
- Long ago programmers would write programs in assembler because they believed they could write better code than a compiler would generate
- This is no longer the case

Optimized Machine Language

- Load register with B
- Add C to register
- Store register in Temp1
- Load register with Temp1
- Add D to register
- Store register in Temp2
- Load register with Temp2
- Store register in A

Peephole Optimization

- Peephole optimization is a kind of optimization performed over a very small set of instructions in a segment of generated code
- This contrasts with global optimizations that consider the entire program

Common peephole optimization

- Constant folding – Evaluate constant subexpressions in advance
- Strength reduction – Replace slow operations with faster equivalents
- Null sequences – Delete useless operations
- Combine operations – Replace several operations with one equivalent
- Algebraic laws – Use algebraic laws to simplify or reorder instructions

from Wikipedia

Strength Reduction

- Strength reduction is where expensive operations are replaced with equivalent but less expensive operations
- There are multiple ways to divide by 8

```
divide       R3, 8
```

- can be simplified to

```
shiftRight   R3, 3      // if integer
multiply     R3, 0.125  // if double
```

Constant Simplification

- A statement may include arithmetic with constants that can be computed at compile time

```
volume = 4.0 / 3.0 * 3.14159 * r * r;
```

Can be simplified to

```
volume = 4.18879 * r * r;
```

Constant Propagation

- Constants can be carried to the next statements

```
int x = 14;
int y = 7 - x / 2;
return y * (28 / x + 2);
```

- Propagating x yields:

```
int x = 14;
int y = 7 - 14 / 2;
return y * (28 / 14 + 2);
```

- Further simplifying to

```
return 0;
```

from Wikipedia

Global Optimization

- The compiler can analyze the entire program or parts of the program to optimize
- Programs can be modeled as graphs that can be used to determine when data is changed or referenced

Loop Invariants

- Calculations inside a loop that do not change in the loop, can be moved before the loop.
- This can make the program much more efficient if the calculation moved out of the loop requires a lot of CPU time

```
for (int i=0; i<n; i++) {
   x = y+z;
   a[i] = 6*i + x*x;
}
```

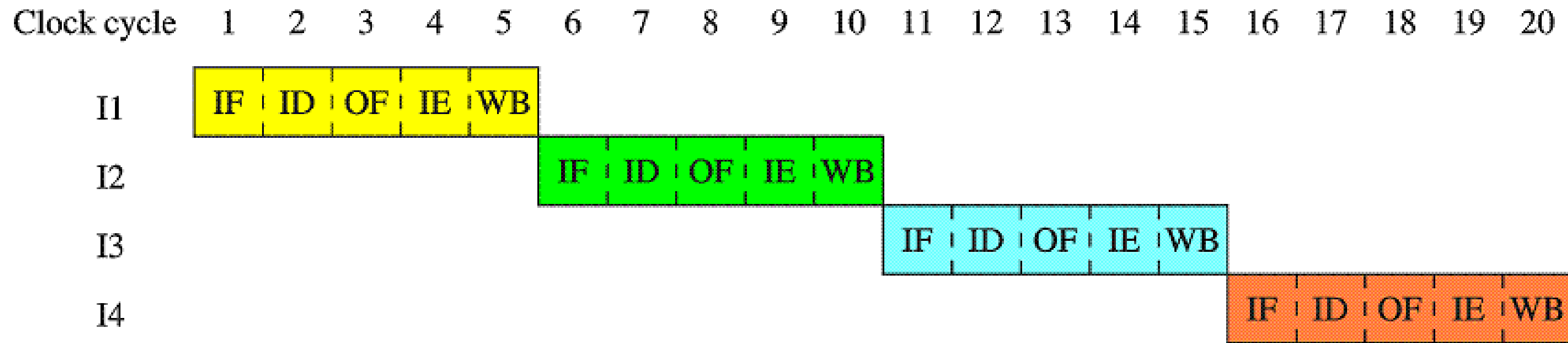
```
x = y+z;
temp = x*x;
for (int i=0; i<n; i++) {
   a[i] = 6*i + temp;
}
```

from Wikipedia

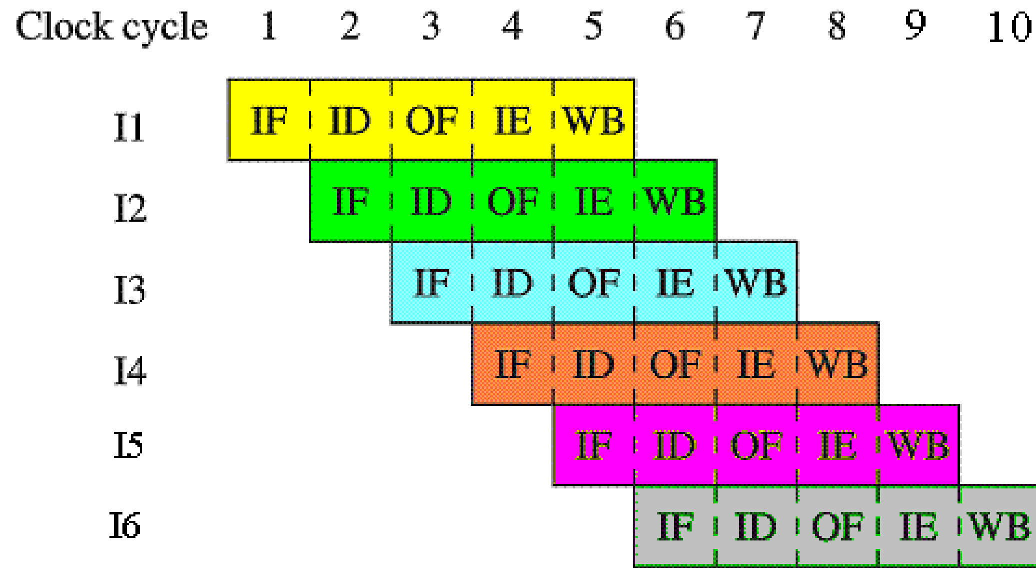
Assembly Line

- Pipelining is like an assembly line. Each stage of the execution cycle performs its function and passes the instruction to the next cycle
- Each stage can be working on a different instruction at the same time
- Several instructions can be in the process of execution simultaneously
- There is no reason to keep the ALU idle when you are fetching or storing an operand

Pipelining



Serial execution



Pipelined execution

Hazards

- A hazard is a situation that reduces the processors ability to pipeline instructions
- **Resource** – When different instructions want to use the same CPU resource
- **Data** – When the data used in an instruction is modified by the previous instruction
- **Control** – When a jump is taken or anything changes the sequential flow

Avoiding Data Hazards

- The compiler can generate code that avoids data hazards
- Interleaving different parts of an expression or the program can space access to data values to avoid data hazards
- Using multiple registers instead of always the same register helps reduce data hazards

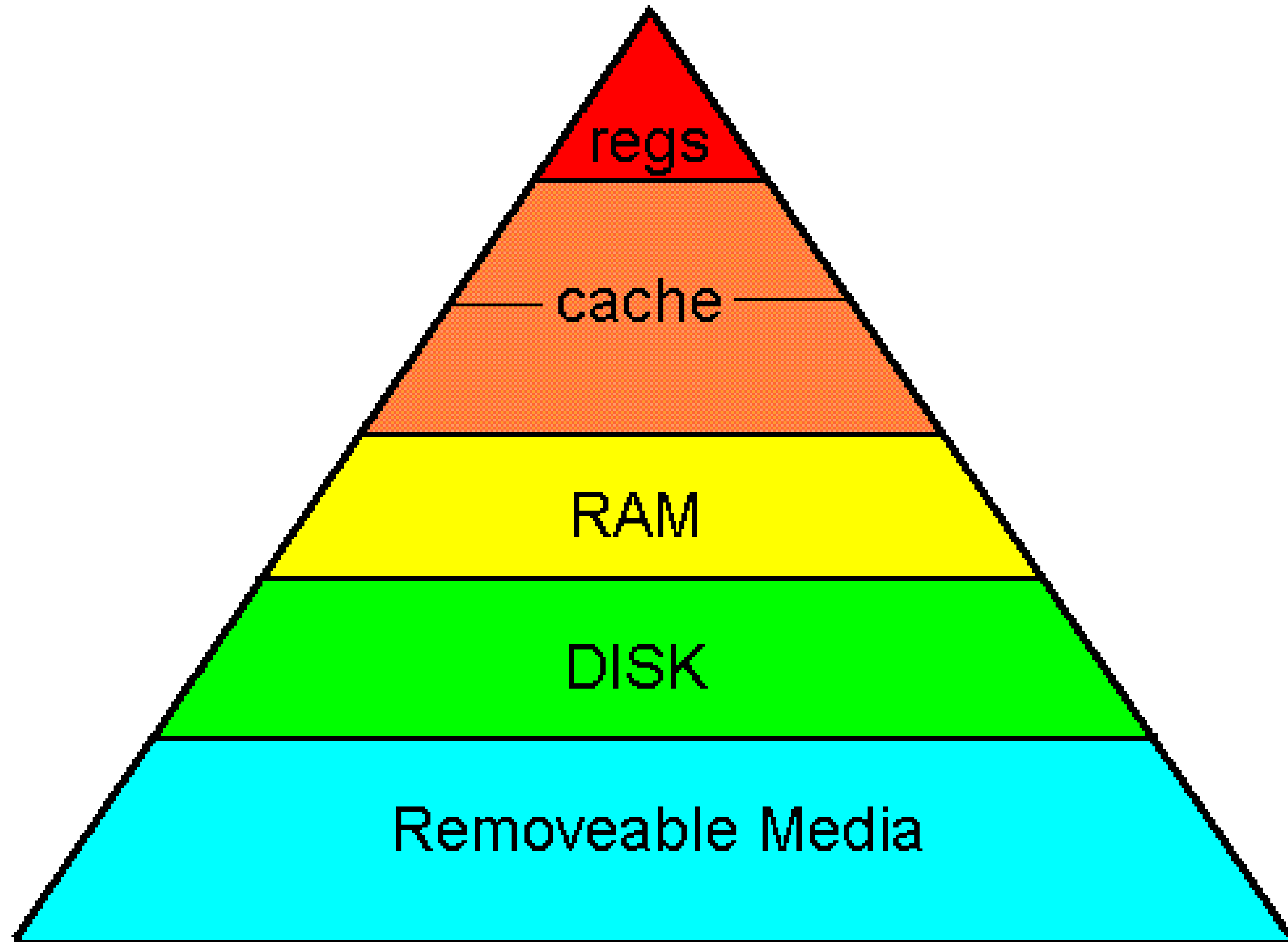
Pipeline Efficient Code

Inefficient	Pipeline Efficient (4 stage pipeline)
Load R1, W	Load R1, W
Add R1, 7	Load R2, X
Store R1, W	Load R3, Y
Load R1, X	Load R4, Z
Add R1, 8	Add R1, 7
Store R1, X	Add R2, 8
Load R1, Y	Add R3, 9
Add R1, 9	Add R4, 10
Store R1, Y	Store R1, W
Load R1, Z	Store R2, X
Add R1, 10	Store R3, Y
Store R1, X	Store R4, Z

Register Allocation

- Keeping values in the registers instead of memory makes the program run much faster
- Many computers have 16 or 32 general purpose registers allowing several values to be kept in a register
- Finding the optimal register allocation is equivalent to the NP-Complete graph coloring problem

Memory Hierarchy



Exam 1

- The first COMP360 exam will be on Friday, February 17, 2017
- You are allowed to have one 8½ by 11" page of notes