# 463 Concepts

**Nathan Warner**

Northern Illinois University

Computer Science
Northern Illinois University
United States

## Contents

- **Pipeline stalling**: Stalling means that the pipeline is no longer full because one instruction took more cycles than expected
- **Pipeline flushing**: Flushing the pipeline means that no new instructions can enter the pipeline until the given instruction is finished.
- **Pipeline hazards**: These calculations takes a number of things for granted. First, we have to assume that the architecture supports fetching instructions and data in parallel. Second, we assume that the pipeline can be kept filled at all times. This is not always the case. Pipeline hazards arise that cause pipeline conflicts and stalls

  An instruction pipeline may stall or be flushed for any of the following reasons

  1. **Resource conflicts**: When an instruction needs a resource, such as a register, that a previous instruction still in the pipeline is still using
  2. **Data dependencies**: When an instruction needs the result of a previous instruction still in the pipeline
  3. **Conditional branching**: When a branch statement such as "if a then go to b else go to c" occurs in the pipeline, and b or c occur soon after this statement. Neither b nor c can enter the pipeline until the result of the branch is determined, because we do not know whether they will ever be executed at all.

  When any of these pipeline hazards occurs, the later instruction cannot enter or continue along the pipeline until the conflict is resolved

- **Object files**: Assemblers create an *object program file* from mnemonic source code in two passes.

  An object file is a partially compiled file produced by a compiler before linking. It contains machine code, but the code is not yet fully ready to run because external references (functions, variables in other files, libraries, etc.) are still unresolved.

- **The linker**: The linker is a tool that takes one or more object files (the partially-compiled outputs from the assembler/compiler) and combines them into a single final program, which is either an executable file, a library, or a module

  When you compile a program with multiple files, each object file contains

  - machine code
  - missing addresses for external symbols (functions/variables in other files)
  - a symbol table

  The linker's job is to:

  - Match up all the undefined symbols with their actual definitions.

  - Fill in all missing addresses in the machine code.
  - Lay out all the code and data in memory in a single, unified executable.

- **Output of assemblers (relocatable binary code)**: The output of most assemblers is a stream of **relocatable binary code**. In relocatable code, operand addresses are relative to where the operating system chooses to load the program.

  When relocatable code is loaded for execution, special registers provide the base address. Addresses specified within the program are interpreted as offsets from the base address.

  **Note:** Absolute (nonrelocatable) code is most suitable for device and operating system control programming

- **Absolute code**: Absolute code is machine code that uses fixed, hard-coded memory addresses. Because the addresses are built directly into the code, the program:
  - must be placed at a specific location in memory
  - cannot be moved without breaking the program
  - has no relocation, no flexibility

- **Binding**: The process of assigning physical addresses to program variables is called **binding**.

  Binding can occur at compile time, load time, or run time. Compile time binding gives us absolute code.

- **Load time binding**: Load time binding assigns physical addresses as the program is loaded into memory.

  With load time binding the program can be loaded into different places but it cannot be moved once it is loaded. That is not common today and is included here only for the sake of completeness.

- **Run time binding**: Run time binding requires a base register or equivalent to carry out the address mapping. That is the normal way of loading programs today

  Runtime binding means the program does NOT know the address of a symbol until the moment that symbol is actually used while the program is running.