COMP2130 Cheat Sheet

1 Week 1: Machine-Level Programming I: Basics

1.1 History of Intel Processors and Architectures

- **Intel x86 Processors:** Dominant in the PC market, featuring a backward-compatible, evolutionary design. They are Complex Instruction Set Computers (CISC).
- **Milestones:**
  - **8086 (1978):** 16-bit processor, basis for IBM PC/DOS.
  - **386 (1985):** First 32-bit (IA32) processor.
  - **Pentium 4E (2004):** First 64-bit (x86-64) processor.
  - **Core 2 (2006):** First multi-core processor.
  - **Core i7 (2008):** Four cores, integrated memory controller.
- **AMD:** Competitor to Intel, developed x86-64.

1.2 C, Assembly, and Machine Code

The compilation process transforms human-readable C code into machine-executable code.

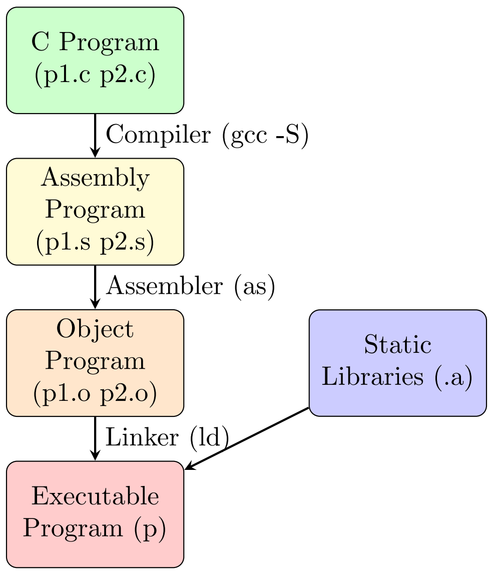

Figure 1: Compilation Process

1.3 Assembly Basics

1.3.1 Addressing Modes

Table 1: x86-64 Registers

Register	Use
%rax	Return value
%rdi, %rsi, %rdx, %rcx, %r8, %r9	First 6 function arguments
%rsp	Stack pointer

```
// Immediate: Constant integer data
movq $0x4, %rax //

// Register: One of 16 integer registers
movq %rax, %rdx //

// Memory: 8 consecutive bytes of memory at address
movq (%rax), %rdx //
```

1.4 Arithmetic & Logical Operations

Table 2: Common Arithmetic and Logical Instructions

Instruction	Description
`leaq Src, Dst`	Load Effective Address
`addq Src, Dst`	Add
`subq Src, Dst`	Subtract
`imulq Src, Dst`	Multiply
`salq Src, Dst`	Left Shift
`sarq Src, Dst`	Arithmetic Right Shift
`shrq Src, Dst`	Logical Right Shift
`xorq Src, Dst`	Exclusive OR
`andq Src, Dst`	And
`orq Src, Dst`	Or

2 Week 2: Machine-Level Programming II, III, & IV

2.1 Control Flow

- **Condition Codes:** Single-bit registers set by arithmetic/logical operations.
  - CF: Carry Flag
  - ZF: Zero Flag
  - SF: Sign Flag
  - OF: Overflow Flag
- **Instructions:**
  - `cmpq Src2, Src1`: Sets condition codes based on `Src1 - Src2`.
  - `testq Src2, Src1`: Sets condition codes based on `Src1 & Src2`.
  - `setX`: Sets a byte based on condition codes.
  - `jX`: Jumps based on condition codes.

2.2 Procedures

- **Stack Frame:** A region on the stack for a single procedure call.
- **Control Transfer:**

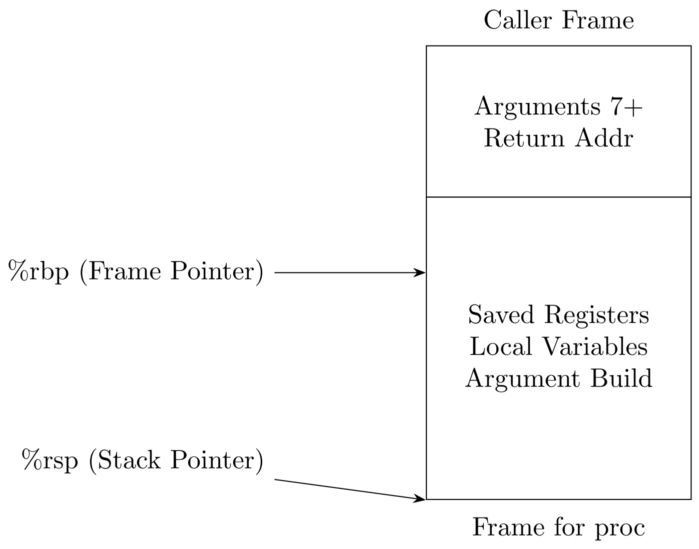

Figure 2: Stack Frame

- `call label`: Pushes return address, jumps to `label`.
- `ret`: Pops return address, jumps to it.

- **Register Saving Conventions:**
  - **Caller-saved:** %rax, %rdi-%r9, %r10, %r11
  - **Callee-saved:** %rbx, %r12-%r14, %rbp, %rsp

2.3 Data

- **Arrays:** Contiguous memory blocks.
  - 1D: `T A[L]`
  - 2D: `T A[R][C]` (row-major order)
- **Structs:** Memory block holding all fields, with padding for alignment.

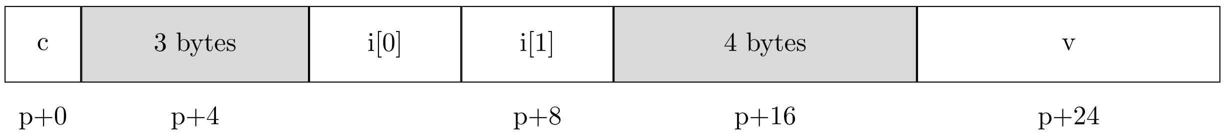

Figure 3: Struct Alignment

3 Week 3: Linking, Advanced Topics, and Optimization

3.1 Linking

3.2 Advanced Topics

- **Memory Layout:** Stack, Heap, Data, Text.
- **Buffer Overflow:** Writing data beyond a buffer's boundaries, a major security vulnerability.
- **Unions:** Data structures that store different data types in the same memory location.

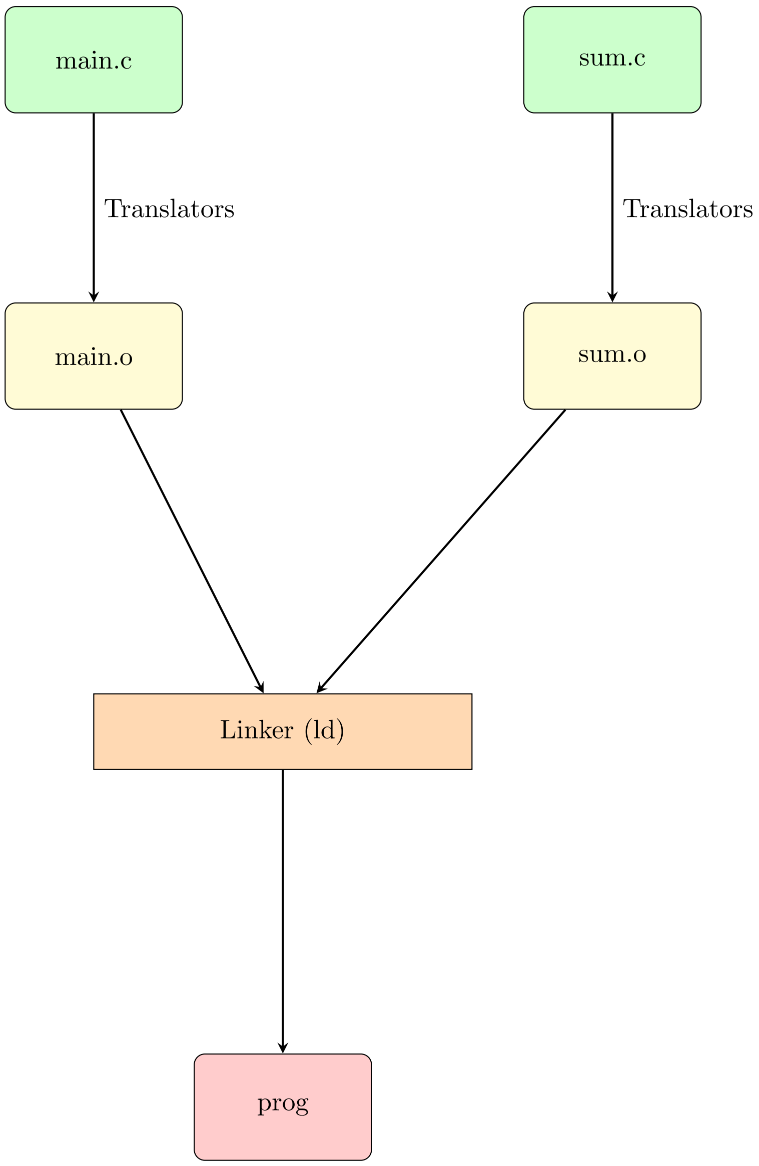

Figure 4: Linking Process

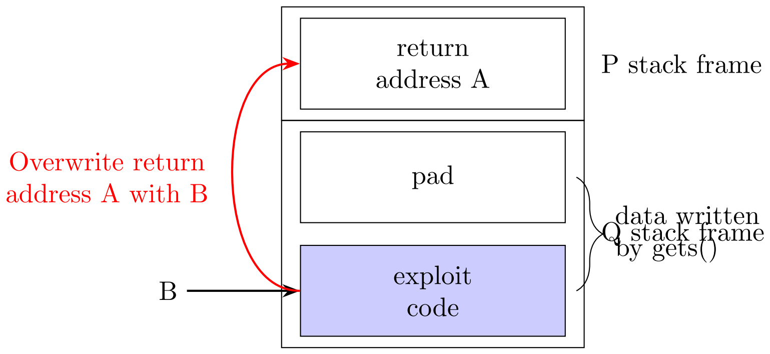

Figure 5: Buffer Overflow Attack

3.3 Optimization

- **Constant Folding:** Compile-time evaluation of constant expressions.
- **Dead Code Elimination:** Removing unreachable code.

- **Common Subexpression Elimination:** Reusing results of computations.
- **Code Motion:** Moving loop-invariant code out of loops.
- **Inlining:** Replacing function calls with the function's body.

4 Week 4: ECF and Memory Hierarchy

4.1 Exceptional Control Flow (ECF)

- **Exceptions:** Transfer of control to the OS.
- **Processes:** Instances of running programs.
- **Signals:** Messages notifying a process of an event.

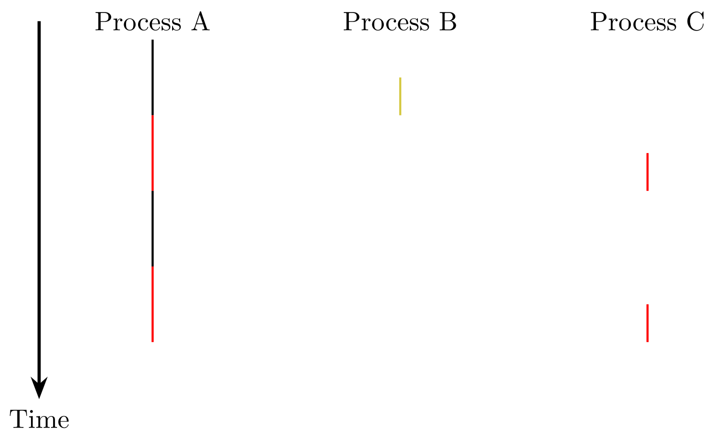

Figure 6: Concurrent Process Execution

4.2 The Memory Hierarchy

- **Locality:**
  - **Temporal:** Recently accessed items are likely to be accessed again.
  - **Spatial:** Items with nearby addresses are likely to be accessed together.
- **Caching:** Using a smaller, faster memory to store a subset of data from a larger, slower memory.

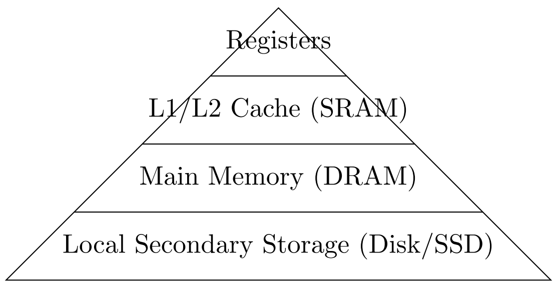

Figure 7: Memory Hierarchy

5 Week 5: Virtual Memory

5.1 Virtual Memory (VM)

- **Address Spaces:** Virtual (program's view) and Physical (hardware's view).
- **Page Table:** Maps virtual pages to physical pages.
- **Page Fault:** An exception that occurs when a program accesses a page that is not in physical memory.
- **Address Translation:** The process of converting a virtual address to a physical address.

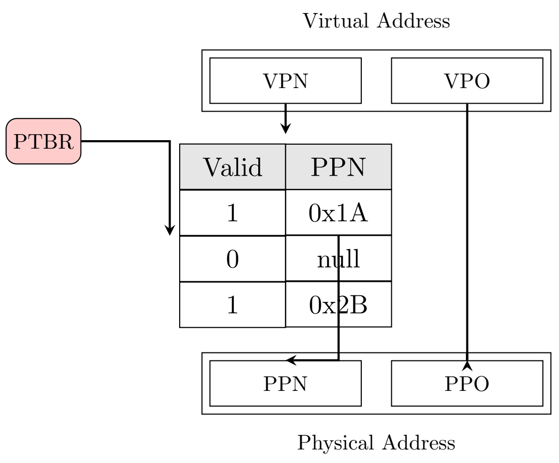

Figure 8: Address Translation