# S3 - Basics of Architecture, Machine Code

**Date:** 2013-05-04
**tags:** hwswint

Translation Impacts Performance

The time required to execute a program depends on:

- The program (as written in C, for instance)
- The compiler: what set of assebler it translates the C program into
- The Instruction Set Architecture (ISA): what set of instructions it makes available to the compiler
- The hardware implementation: how much time it takes to execute an instruction

ISA (Instruction Set): The parts of processor one needs to understand to write assembly code. (What is directly visible to the software)

The ISA defines:

- The system's state (e.g. registers, memory, program counter)
- The instructions that CPU can execute
- The effect that each of these instructions will have on the system state

General ISA Design Decisions:

- **Instructions**
    - What instructions are available? What do they do?
    - How are they encoded?
- **Registers**
    - How many registers are there?
    - How wide are they?
- **Memory**
    - How do you specify a memory location?

## x86

x86 is one type of ISA. Processors that implement the x86 ISA completely dominate the server, desktop and laptop markets.

Evolutionary design

- Backwards compatible up until Intel 8086, introduced in 1978
- Added more features as time goes on

x86 can be defined as Complex Instruction Set Computer (CISC)

- Many different instructions with many different formats but only small subset is encountered with Linux programs
- (as opposed to Reduced Instructions Set Computers (RISC) which use simpler instructions)

Intel and AMD both develop x86 and x86-64 processors with different hardware implementations underneath. But since the ISA (which is an abstraction) remains the same i.e. x86, same programs can run on both of them.

## Assembly Programmer's View

Programmer-Visible State

- **PC: Program Counter**
    - Address of the next instructions
    - Called "EIP" (IA32) or "RIP" (x86-64)
- **Register file**
    - Heavily used program data
- **Condition codes**
    - Store status information about most recent arithmetic operation
    - Used for conditional branching
- **Memory**
    - Byte addressable memory
    - Code, user data, (some) OS data
    - Includes stack used to support procedures

## Assembly

Three basic kinds of Instructions

- Perform arithmetic function on register or memory data
- **Transfer data between memory and register**
    - Load data from memory into register'
    - Store register data into memory
- **Transfer control**
    - Unconditional jumps to/from procedures
    - Conditional branches

Data Types

- **"Integer" data of 1, 2, 4 (IA32), or 8 (just in x86-64) bytes. Can be:**
    - Data values
    - Addresses (untyped pointers)
- Floating point data of 4, 8, or 10 bytes
- No aggregate types such as arrays or structs, just contiguosly allocated bytes in memory

Turning C into Object Code

- Code in files p1.c p2.c
- **Compile using command `gcc -O1 p1.c p2.c -o p`**

    - Use basic optimizations (-01)
    - Put result binary in file p

Pathway

- text `C program (p1.c p2.c)` -- Compiler (`gcc -S`) --
- --> text `Asm program (p1.s p2.s)` -- Assembler (gcc or as) --
- --> binary `Object program (p1.o p2.o)` -- Linker (gcc or ld) --
- --> binary `Executable program (p)` <--- Static libraries (.a)

Assembler

- Translates `.s` into `.o`
- Binary encoding of each instruction
- Near complete image of executable code
- Missing links between code in different files

Linker

- Resolves references between object files and (re)locates theie data
- Combines with static run-time libraries, e.g. code for `malloc`, `printf`
- Some libraries are dynamically linked. Linking occurs when program begins execution

## Disassemblers

`objdump -d <file>`

- Useful tool for examining object code
- Analyzes bit pattern series of instructions (delineates instructions)
- Produces near-exact rendition of assembly code
- Can be run either on the executable or .o files (object code).

Alternatively, gdb debugger can also be used for disassembly.

## Registers

What is Register?

- A location in the CPU that stores a small amount of data which can be accessed very quickly (once every clock cycle)
- Registers are at the heart of assembly programming. They are precious commodity in all architectures but especially x86

IA32 has 8 different registers of size 32 bits each. Two of them are special purpose (stack pointer and base pointer).

x86-64 has 16 registers each of size 64 bits (also backward compatible as 8, 16, 32 bits). They happen to be the superset of the previous architectures so they are backward compatible. Only one is for special purpose.