Computer Architecture
Machine Representation of Programs: Fundamentals

Cain Susko

Queen's University
School of Computing

February 13, 2022

Intro

Before, we were talking about data representation in Computers it

- Integers
- Characters
- Floats

This section of the course pretains to how and what computers do with this data

Machine code is a useful thing to know as while, we normally work with computers at a high level, it is useful to know machine code because:

- it can make you a better programmer
- you can understand the real-time behaviour or programs
- understand the vunerabilities of a computer system

Intel x86 The Intel x86 Processor dominates the computer market currently. The x86 language was introduced in the 1970's. More features were added as time went on. The code used is relatively complex and because of this the language consumes a considerable amount of power.

Definitions

Architecture Also known as ISA: instruction set architecture. This is the parts of the processor design that one needs to understand to write machine code. For example: instruction set specifications, registers...

Microarchitecture Implementation of the Architecture. Like cache sizes and core frequency

Code Forms

- Machine Code - byte level programs that processor executes
- Assembly Code - text level representation of machine code

There are different examples for instructions and architecture ie. ARM, IA32, x86...

Assembly and Machine code

Within a computer a CPU contains the Program Counter, Registers, and Condition Codes while the memory holds the code, data and stack. Addresses are send from the CPU to the memory and Data and Instructions are returned to the CPU.

- Program Counter (PC) has the address of the next instruction and is called RIP in x86-64
- Register File contains heavily used program data
- Condition Codes store status information about most recent arithmetic or logical operation. Also used for conditional branching
- Memory is a byte addressed which contains code and user data as well as a stck to support procedures (see cisc 223 W5 - state automaton)

In C, the code is turned into machine executable code like so:

C program $\rightarrow^{compiler}$ Asm Program $\rightarrow^{assembler}$ Object Program $\rightarrow^{linker}$ Executable Program.

Assembly Data Types There are many types of integer data within 1-8 bytes: Note, there are no aggregate types like a list or array struc-

C declaration	Intel data type	code suffix	size (bytes)
`char`	Byte	`b`	1
`short`	Word	`w`	2
`int`	Double word	`l`	4
`long int`	Quad word	`q`	8
`long long int`	Quad word	`q`	8
`char *`	Quad word	`q`	8
`float`	Single precision	`s`	4
`double`	Double precision	`d`	8

ture.Everything is just continuous bytes in memory.

Operations

- Transfer Data between Memory and Register - load data into register and save resister data into memory
- Perform arithmetic on register data
- Transfer control - unconditional jumps to or from procedures and conditional branches.

The Assembler translates Asm Code into Object code, which is in binary. This binary is nearly executable accept the linker must do its namesake and resolves references between files like math (ie. -lm math). Some libraries are dynamically linked - this coccurs when the program begins execution.

Example

This is a machine instruction example from C to Assembly to Machine code.

`*dest = t` store the value t designated as $dest$

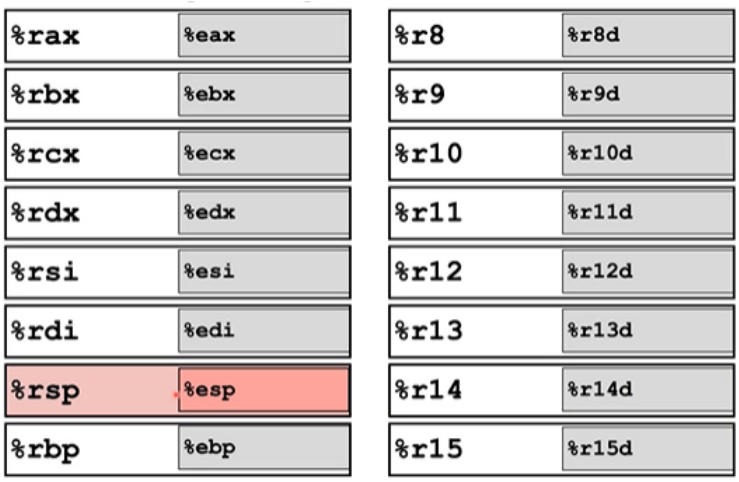
`movq %rax, (%rbx)` Move 8-byte value (aka Quad Word) to memory.
$\%rax = t$, $\%rbx = dest$, $M[\%rbx] = *dest$
Where rax, rbx are in the register and $M[rbx]$ is in memory

`0x40059e:48 89 03` Three byte instruction stored at address $0x40059e$

Looking at the full code from `objdump -d sumprogram` we can see the object code of a program:

```
Dump of assembler code for function sumstore:
 0x0000000000400595 <+0>: push   %rbx
 0x0000000000400596 <+1>: mov    %rdx,%rbx
 0x0000000000400599 <+4>: callq  0x400590 <plus:
 0x000000000040059e <+9>: mov    %rax,(%rbx)
 0x00000000004005a1 <+12>:pop    %rbx
 0x00000000004005a2 <+13>:retq
```

The full list of the registers ($\%xy$) that can be used above are as follows:



Moving Data

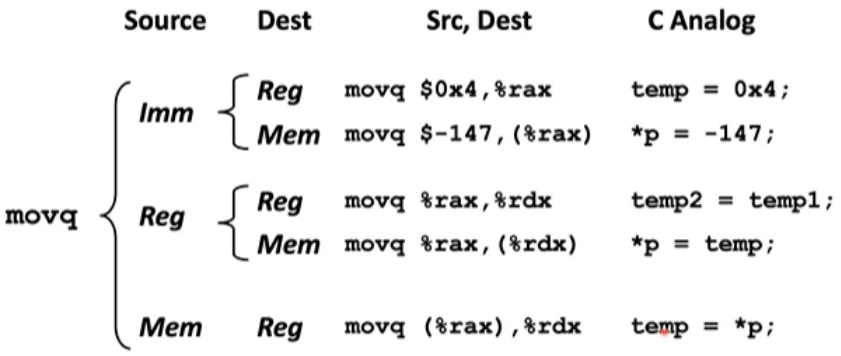
In Assembly Code, use `movq` *`source destination`*
The operand typed for the `movq` command are:

Immediate Constant integer data like `$0x400, $-53`

Register One of 16 integer registers like `%rax, %r13`. However note that `%rsp` is reserved for special use.

Memory 8 consecutive bytes of memory at address given by register. for example, `%rax`. There are various 'address modes' within Memory

Clearly there are many combinations of operands when using `movq`. They can be visualized like so:



Simple Addressing Modes

There are 2 types of Addressing modes when using movq.

Normal (R) Mem[Reg[R]]
Register R specifies a memory address.
`movq (%rcx), %rax`

Displacement D(R) Mem[Reg[R]+D]
Register R specifies start of memory region. The constant displacement D specifies offset.
`movq 8(%rbp), %rdx`

Note: Registers start with a %.

```
void swap
   (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
   movq    (%rdi), %rax
   movq    (%rsi), %rdx
   movq    %rdx, (%rdi)
   movq    %rax, (%rsi)
   ret
```