X86 64 Register and Instruction Quick Start

This page contains very basic information on the x86_64 architecture: the register layout and naming and the some basic instructions.

Contents

Registers

General-Purpose Registers

The 64-bit versions of the 'original' x86 registers are named:

- rax - register a extended
- rbx - register b extended
- rcx - register c extended
- rdx - register d extended
- rbp - register base pointer (start of stack)
- rsp - register stack pointer (current location in stack, growing downwards)
- rsi - register source index (source for data copies)
- rdi - register destination index (destination for data copies)

The registers added for 64-bit mode are named:

- r8 - register 8
- r9 - register 9
- r10 - register 10
- r11 - register 11
- r12 - register 12
- r13 - register 13
- r14 - register 14
- r15 - register 15

These may be accessed as:

- 64-bit registers using the 'r' prefix: rax, r15
- 32-bit registers using the 'e' prefix (original registers: e_x) or 'd' suffix (added registers: r__d): eax, r15d
- 16-bit registers using no prefix (original registers: _x) or a 'w' suffix (added registers: r__w): ax, r15w
- 8-bit registers using 'h' ("high byte" of 16 bits) suffix (original registers - bits 8-15: _h): ah, bh
- 8-bit registers using 'l' ("low byte" of 16 bits) suffix (original registers - bits 0-7: _l) or 'b' suffix (added registers: r__b): al, bl, r15b

Usage during syscall/function call:

- First six arguments are in rdi, rsi, rdx, rcx, r8d, r9d; remaining arguments are on the stack.
- For syscalls, the syscall number is in rax.
- Return value is in rax.
- The called routine is expected to preserve rsp,rbp, rbx, r12, r13, r14, and r15 but may trample any other registers.

Floating-Point and SIMD Registers

x86_64 also defines a set of large registers for floating-point and single-instruction/multiple-data (SIMD) operations. For details, refer to the Intel or AMD documentation.

Instructions

Starter Kit

These instructions are sufficient to complete the SPO600 Assembler Lab (GAS syntax):

```
add %r10,%r11    // add r10 and r11, put result in r11
add $5,%r10      // add 5 to r10, put result in r10
call label       // call a subroutine / function / procedure
cmp %r10,%r11    // compare register r10 with register r11.  The comparison sets flags in the process
cmp $99,%r11     // compare the number 99 with register r11.  The comparison sets flags in the proces
div %r10         // divide rax by the given register (r10), places quotient into rax and remainder in
inc %r10         // increment r10
jmp label        // jump to label
je  label        // jump to label if equal
jne label        // jump to label if not equal
jl  label        // jump to label if less
jg  label        // jump to label if greater
mov %r10,%r11    // move data from r10 to r11
mov $99,%r10     // put the immediate value 99 into r10
mov %r10,(%r11)  // move data from r10 to address pointed to by r11
mov (%r10),%r11  // move data from address pointed to by r10 to r10
mul %r10         // multiplies rax by r10, places result in rax and overflow in rdx
push %r10        // push r10 onto the stack
pop %r10         // pop r10 off the stack
```

```
ret              // routine from subroutine (counterpart to call)
syscall          // invoke a syscall (in 32-bit mode, use "int $0x80" instead)
```

Note the syntax:

- Register names are prefixed by %
- Immediate values are prefixed by $
- Indirect memory access is indicated by (parenthesis).
- Hexadecimal values are indicated by a 0x prefix.
- Character values are indicated by quotation marks. Escapes (such as '\n') are permitted.
- Data sources are given as the first argument (mov %r10,%r11 moves FROM r10 INTO r11).

For the MOV instruction:

- You can append a suffix indicating the amount of data to be moved -- e.g., q for quadword (64 bits), d for doubleword (32 bits), w for word (16 bits), or b for byte (8 bits).

Resources

- CPU Instruction Set and Software Developer Manuals
  - AMD: https://developer.amd.com/resources/developer-guides-manuals/ (see the AMD64 Architecture section, particularly the *AMD64 Architecture Programmer’s Manual Volume 3: General Purpose and System Instructions*)
  - Intel: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
- Web sites
  - http://ref.x86asm.net/
  - http://sandpile.org/
- GAS Manual - Using as, The GNU Assembler: https://sourceware.org/binutils/docs/as/

Retrieved from "https://wiki.cdot.senecacollege.ca/w/index.php?title=X86_64_Register_and_Instruction_Quick_Start&oldid=146014"

- This page was last edited on 2 March 2020, at 14:05.