CSC-640

Computer System Organization and Programming

Topic: x86_64 CPU Architecture

Sheng Kai Liao

American University 4400 Massachusetts Ave, NW

Spring 2021

Introduction

Since the computer been developing, CPU (Central Processing Unit) becomes more and more important as a computer element which mainly take respond for computing, program processes managing and I/O (Input and output) manipulating. We can describe it as the brain for humankind which control mostly all responds for real world. However, for each purpose and design logic, the CPU architectures could be different. Thus, for this paper, I will fully discuss x86_64 CPU Architecture's operation modes, registers, and ISA (Instructure set architecture).

The x86_64 is an extended mode form IA-32. We can also call it IA-32e mode, AMD64 or Intel 64. This expansion increases the main register bits to achieve more accurate calculation results, larger register space is used for more data, and more complex software designs, thereby improving the performance of the CPU. In addition, this extension removes some unnecessary functions from IA-32 and saves the actual unused space of physical addresses or virtual addresses for future expansion. Thus, since the design is based on IA-32, x86_64 can mostly operate the modes and features from IA-32 in 64-bits environment. Thanks to these improvements, the x86_64 architecture has been widely used in many CPU designs, such as Intel Core, Intel Xeon, Intel Pentium and Intel Atom.

System Register

Brief talking

To operate such big system, the x86_64 provided many registers for each task and working mode. RFLAGS register, extended from EFLAGS register in IA-32, manipulate the system task and mode switching, interrupt handling, instruction tracking and access right. The

control registers, such as CR0, CR2, CR3, CR4 and CR8, have a variety of flags and information address for estimate the operation level in system and controlling data and targeted address (Intel, 2016). Also, there have debugging registers, DR0-DR7, allow developer using in debugging programs. The GDTP, LDTP, and IDTR for storing GDT (global descriptor table), LDT (local descriptor table) and IDT (Interrupt Descriptor Table) with individual duties which I will do further description in this paper. The task register includes TSS (the task-state segment) for operating work. Several model-specific registers (MSRs), like group registers for operate-system or executive procedure, debug extensions, the performance-monitoring counters, the machine-check architecture, the memory type ranges (MTRRS).

Register operation

The system flags

To x86_64 or IA-32, the system sets up the operation mode, I/O privilege level, interrupt enable and Identification by referring system flags (intel, 2016). In x86_64, all flags are contained in RFLAGS register which has 64-bit usage space and 32-bit reserved space for extension.

The GDT and LDT

The GDT and LDT are two tables which includes information associated with the task. One GDT can define multiple LDTs. Both tables include segment descriptor which contains complete segment descriptor with the access right, the type and the usage information of segments (intel, 2016). By refer to these two tables, system will obtain the action access of register, the data which will be operated and the instructions. Furthermore, for each segment descriptor, it will be selected by an associated segment selector in system which contains the

number into the GDT and LDT, a global/local flag (using GDT or LDT) and the access right (Intel, 2016). As I mentioned, both tables are contained in specific register such as GDTR (global descriptor table register) and LDTR (local descriptor table register).

The GDTR and LDTA are 64-bit in IA-32e mode. Moreover, the GDT and LDT are 64-bit in 64-bit mode (sub-mode in IA-32e) but did not extend in compatibility mode (sub-mode in IA-32e) (Intel, 2016).

The IDT

The IDT (Interrupt Descriptor Table) takes responsibility to handle system external, software and exception interruptions. It contains the instruction data to interrupt and an exception controller (intel, 2016). The IDT is stored in IDTR (Interrupt Descriptor Table Register). In IA-32e mode, IDT and IDTR are turned into 64-bit address.

The TR

The TR (Task Register) mainly respond to the TSS in GDT. It required a segment selector to select the task segment from TSS, a mode address to identify the operation mode and a descriptor for current task (intel, 2016). The TR has 64-bit address in IA-32e mode.

The TSS and the task changing procedure

TSS (the task-state segment) is contained in GDT which include the stack pointers with different privilege level, pointer address for IDT and IO-accession address (intel, 2016). To change task, the current state will be stored into current TSS. Afterward, the system will load the new task data from new created TSS into GPR (general-purpose registers), the segment register,

LDTR, CR3, RFLAGS and EIP registers. Then, TR will load the segment selector and descriptor of new task from TSS and start running.

The Control Register

The control registers attribute the setting of operating mode, and they are manipulating by MOV CRn instructions. The control registers are shown below:

- CR0: Stores the operating mode setting and state flags for the processor.
- CR1: Reserved.
- CR2: Stores the address for hardware error (page-fault).
- CR3: Contains the physical address about the paging-structure hierarchy and flags for PCD (Page-level cache disable) and PWT (Page-level write-through) which both control caching paging structure from the processor's internal data caches.
- CR4: Includes flags to enable architecture extensions and represent the operation system or support particular processor capabilities.
- CR8: Offers r/w access for TPR (task priority register) to control the priority class in operating system.

Operation Modes

- Protected mode: The native set of operation mode. You have high flexibility in this mode to modify the system, such as architectural features, performance, and backward compatibility.
- Real-address mode: It delivers Intel 8086 processor developing environment.

• System management mode (SMM): In this mode, it allows processors to use a separate address space for storing data. After that, the SMM-specific code will be executed transparently, and keep the result into its state before to the SMI.

• Virtual-8086 mode: The system can execute 8086 software in a virtual environment by using this mode.

• IA-32e mode: This is the mode for 64-bits environment for IA-32 architecture. There have two sub-modes: compatibility mode and 64-bit mode. Compatibility mode majorly run the unchanged protected-mode features. The 64-bit mode offer 64-bit linear addressing for dealing the physical address which size over 64 GBs.

ISA

The register syntax:

syntax	Function	Saved after operating
%rax	Temporary register; will return value	N
%rbx	Callee-saved	Y
%rcx	Pass 4th argument to functions	N
%rdx	Pass 3rd argument to functions	N
%rsp	Stack pointer	Y
%rbp	Callee-saved; base pointer	Y
%rsi	Pass 2nd argument to functions	N
%rdi	Pass 1st argument to functions	N
%r8	Pass 5th argument to functions	N
%r9	Pass 6th argument to functions	N
%r10-r11	Temporary register	N
%r12-15	Callee-saved register	Y

Table 1. Register syntax table

Instruction Set:

Instruction	Description
mov src, dest	Coping and returning a value from source register to destination register
cmove %src, %dest	Coping and returning a value from source register to destination register if the value from source register is equal to the value from destination register
cmovne %src, %dest	Coping and returning a value from source register to destination register if the value from source register is inequal to the value from destination register
cmovg %src, %dest	Coping and returning a value from source register to destination register if the value from source register is greater than the value from destination register
cmovl %src, %dest	Coping and returning a value from source register to destination register if the value from source register is less than the value from destination register
cmovge %src, %dest	Coping and returning a value from source register to destination register if the value from source register is equal to or greater than the value from destination register
cmovle %src, %dest	Coping and returning a value from source register to destination register if the value from source register is equal to or less than the value from destination register

Table 2. Instruction Set table

Major ALC instruction set:

Instruction	Description
add src,dest	Coping and returning the result of addition with the value of source register and destination register to destination register
sub src, dest	Coping and returning the result of subtraction with the value of source register and destination register to destination register
imul src,dest	Coping and returning the result of multiplication with the value of source register and destination register to destination register
idiv src, dest	Coping and returning the result of division with the value of source register and destination register to destination register
shr reg	Shifting the value from expected register to right
shl reg	Shifting the value from expected register to left
ror src, dest	Rotate dest to the left or right by src bits
cmp src, dest	Set flags for whether dest is less, equal or greater than src

Table 3. Major ALC instruction set table

Stack management:

Instruction	Description

enter $x, $0	Sets up a procedure's stack frame by first pushing the current value of %rbp on to the stack, storing the current value of %rsp in %rbp, and finally decreasing %rsp to make room for x byte
leave	Removing local variable from stack
push src	Decreasing %rsp first and Placing src at the new memory location pointed to %rsp
pop dest	Returning the value from location pointer to dest and increase %rsp

Table 4. Stack management table

Control flow:

Instruction	Description
call target	Jumping to target and placing return value to stack
ret	Popping value from stack and jumping to its address
jmp target	Jumping to target as a memory location
je target (equal), jne target (inequal)	Jumping to target if the condition happened

Table 5. Control flow table

Reference list

Intel. (sep,2016). Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1. https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html

MIT. X86-64 Architecture Guide. http://6.s081.scripts.mit.edu/sp18/x86-64-architecture-guide.html