Intel 64 Instruction Encodings

CS520
Dept. of Computer Science
Univ. of New Hampshire

Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 2A: Instruction Set Reference, A-M

NOTE: The *Intel® 64 and IA-32 Architectures Software Developer's Manual* consists of seven volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-M*, Order Number 253666; *Instruction Set Reference N-Z*, Order Number 253667; *Instruction Set Reference*, Order Number 326018; *System Programming Guide, Part 1*, Order Number 253668; *System Programming Guide, Part 2*, Order Number 253669; *System Programming Guide, Part 3*, Order Number 326019. Refer to all seven volumes when evaluating your design needs.

Order Number: 253666-050US
February 2014

Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 2B: Instruction Set Reference, N-Z

NOTE: The *Intel® 64 and IA-32 Architectures Software Developer's Manual* consists of seven volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-M*, Order Number 253666; *Instruction Set Reference N-Z*, Order Number 253667; *Instruction Set Reference*, Order Number 326018; *System Programming Guide, Part 1*, Order Number 253668; *System Programming Guide, Part 2*, Order Number 253669; *System Programming Guide, Part 3*, Order Number 326019. Refer to all seven volumes when evaluating your design needs.

Order Number: 253667-050US
February 2014

CHAPTER 2
INSTRUCTION FORMAT

This chapter describes the instruction format for all Intel 64 and IA-32 processors. The instruction format for protected mode, real-address mode and virtual-8086 mode is described in Section 2.1. Increments provided for IA-32e mode and its sub-modes are described in Section 2.2.

2.1 INSTRUCTION FORMAT FOR PROTECTED MODE, REAL-ADDRESS MODE, AND VIRTUAL-8086 MODE

The Intel 64 and IA-32 architectures instruction encodings are subsets of the format shown in Figure 2-1. Instructions consist of optional instruction prefixes (in any order), primary opcode bytes (up to three bytes), an addressing-form specifier (if required) consisting of the ModR/M byte and sometimes the SIB (Scale-Index-Base) byte, a displacement (if required), and an immediate data field (if required).

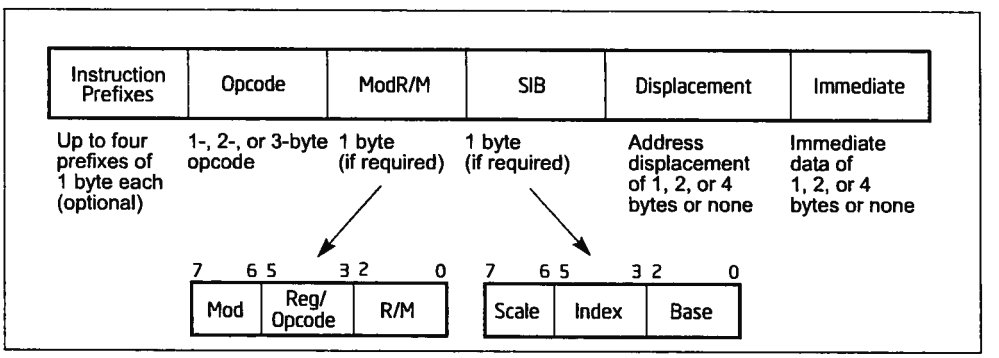


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

2.1.1 Instruction Prefixes

Instruction prefixes are divided into four groups, each with a set of allowable prefix codes. For each instruction, it is only useful to include up to one prefix code from each of the four groups (Groups 1, 2, 3, 4). Groups 1 through 4 may be placed in any order relative to each other.

- Group 1
  - Lock and repeat prefixes:
    - LOCK prefix is encoded using F0H
    - REPNE/REPNZ prefix is encoded using F2H. Repeat-Not-Zero prefix applies only to string and input/output instructions. (F2H is also used as a mandatory prefix for some instructions)
    - REP or REPE/REPZ is encoded using F3H. Repeat prefix applies only to string and input/output instructions.(F3H is also used as a mandatory prefix for some instructions)
- Group 2
  - Segment override prefixes:
    - 2EH—CS segment override (use with any branch instruction is reserved)
    - 36H—SS segment override prefix (use with any branch instruction is reserved)
    - 3EH—DS segment override prefix (use with any branch instruction is reserved)
    - 26H—ES segment override prefix (use with any branch instruction is reserved)

Mod R/M byte

top 2 bits

- 00 (reg)
- 01 $displ_8$(reg)
- 10 $displ_{32}$(reg)
- 11 reg

middle 3 bits

register or extended opcode

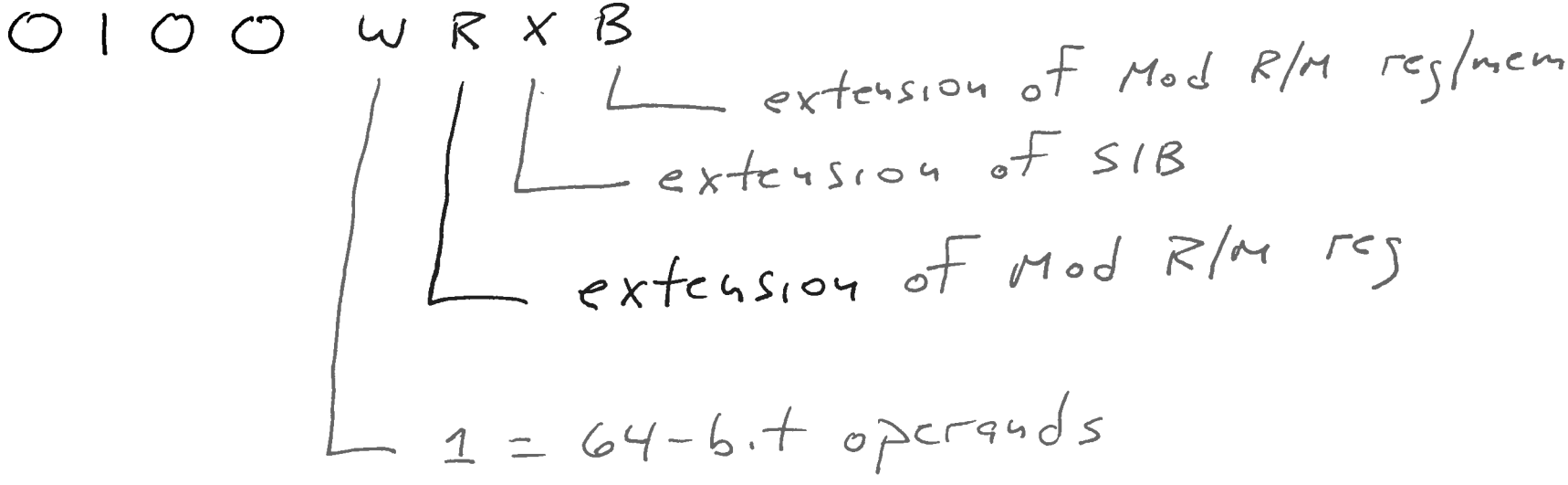
bottom 3 bits

register/memory operand

Mod R/M register encodings

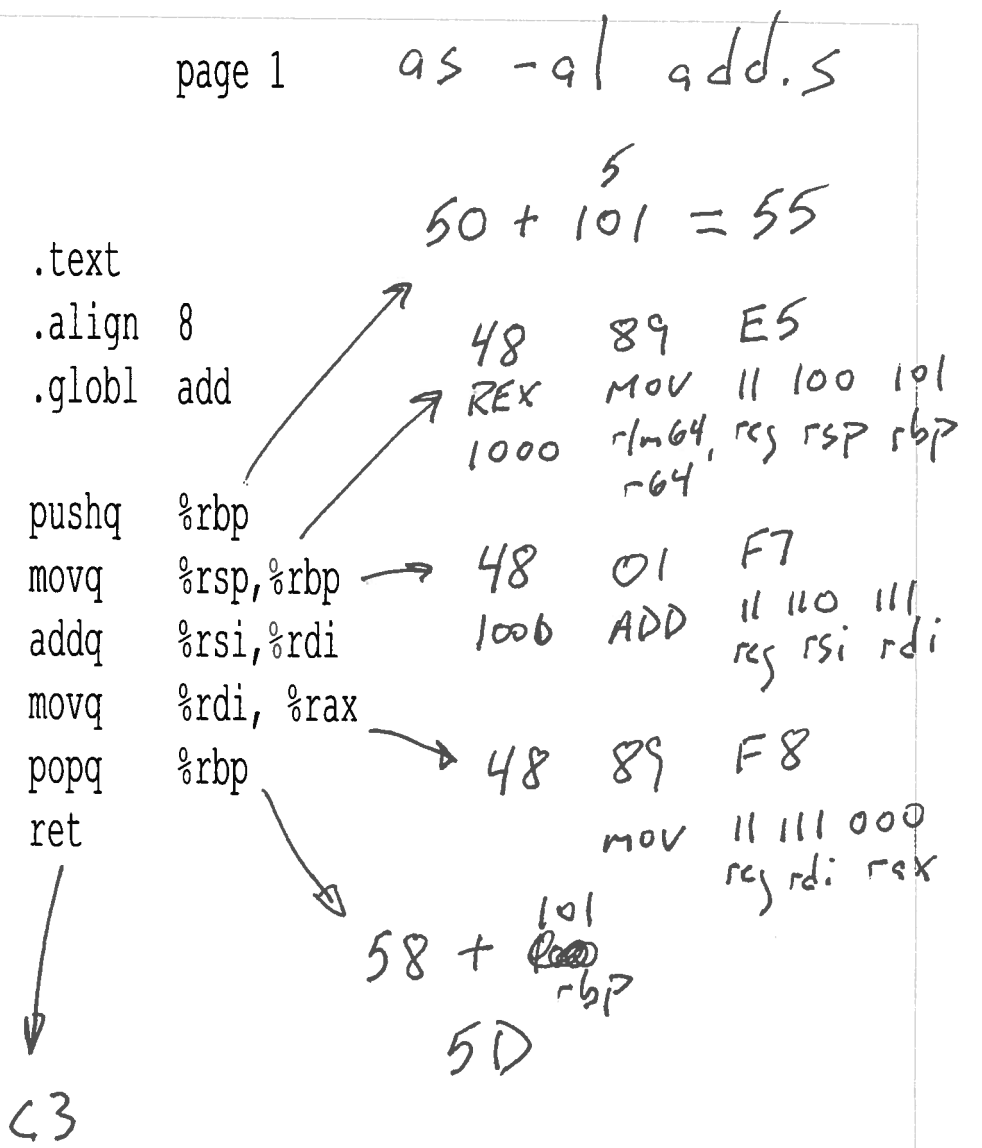
000	rax/r8	100	rsp/r12
001	rcx/r9	101	rbp/r13
010	rdx/r10	110	rsi/r14
011	rbx/r11	111	rdi/r15

REX prefix


0100 W R X B
extension of Mod R/M reg/mem
extension of SIB
extension of Mod R/M reg
1 = 64-bit operands

as -al add.s

```
 1                            .text
 2                            .align 8
 3                            .globl add
 4                    add:
 5 0000 55                    pushq  %rbp
 6 0001 4889E5                movq   %rsp,%rbp
 7 0004 4801F7                addq   %rsi,%rdi
 8 0007 4889F8                movq   %rdi, %rax
 9 000a 5D                    popq   %rbp
10 000b C3                    ret
11
```



PUSH—Push Word, Doubleword or Quadword Onto the Stack

Opcode*	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
FF /6	PUSH *r/m16*	Valid	Valid	Push *r/m16.*
FF /6	PUSH *r/m32*	N.E.	Valid	Push *r/m32.*
FF /6	PUSH *r/m64*	Valid	N.E.	Push *r/m64.* Default operand size 64-bits.
50+*rw*	PUSH *r16*	Valid	Valid	Push *r16.*
50+*rd*	PUSH *r32*	N.E.	Valid	Push *r32.*
50+*rd*	PUSH *r64*	Valid	N.E.	Push *r64.* Default operand size 64-bits.
6A	PUSH *imm8*	Valid	Valid	Push sign-extended *imm8. Stack pointer is incremented by the size of stack pointer.*
68	PUSH *imm16*	Valid	Valid	Push sign-extended *imm16. Stack pointer is incremented by the size of stack pointer.*
68	PUSH *imm32*	Valid	Valid	Push sign-extended *imm32. Stack pointer is incremented by the size of stack pointer.*
0E	PUSH CS	Invalid	Valid	Push CS.
16	PUSH SS	Invalid	Valid	Push SS.
1E	PUSH DS	Invalid	Valid	Push DS.
06	PUSH ES	Invalid	Valid	Push ES.
0F A0	PUSH FS	Valid	Valid	Push FS and decrement stack pointer by 16 bits.
0F A0	PUSH FS	N.E.	Valid	Push FS and decrement stack pointer by 32 bits.
0F A0	PUSH FS	Valid	N.E.	Push FS. Default operand size 64-bits. (66H override causes 16-bit operation).
0F A8	PUSH GS	Valid	Valid	Push GS and decrement stack pointer by 16 bits.
0F A8	PUSH GS	N.E.	Valid	Push GS and decrement stack pointer by 32 bits.

MOV—Move

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
88 /r	MOV *r/m8,r8*	Valid	Valid	Move *r8* to *r/m8*.
REX + 88 /r	MOV *r/m8****,*r8****	Valid	N.E.	Move *r8* to *r/m8*.
89 /r	MOV *r/m16,r16*	Valid	Valid	Move *r16* to *r/m16*.
89 /r	MOV *r/m32,r32*	Valid	Valid	Move *r32* to *r/m32*.
REX.W + 89 /r	MOV *r/m64,r64*	Valid	N.E.	Move *r64* to *r/m64*.
8A /r	MOV *r8,r/m8*	Valid	Valid	Move *r/m8* to *r8*.
REX + 8A /r	MOV *r8****,*r/m8****	Valid	N.E.	Move *r/m8* to *r8*.
8B /r	MOV *r16,r/m16*	Valid	Valid	Move *r/m16* to *r16*.
8B /r	MOV *r32,r/m32*	Valid	Valid	Move *r/m32* to *r32*.
REX.W + 8B /r	MOV *r64,r/m64*	Valid	N.E.	Move *r/m64* to *r64*.
8C /r	MOV *r/m16,Sreg***	Valid	Valid	Move segment register to *r/m16*.
REX.W + 8C /r	MOV *r/m64,Sreg***	Valid	Valid	Move zero extended 16-bit segment register to *r/m64*.
8E /r	MOV *Sreg,r/m16***	Valid	Valid	Move *r/m16* to segment register.
REX.W + 8E /r	MOV *Sreg,r/m64***	Valid	Valid	Move *lower 16 bits of r/m64* to segment register.
A0	MOV AL,*moffs8**	Valid	Valid	Move byte at (*seg:offset*) to AL.
REX.W + A0	MOV AL,*moffs8**	Valid	N.E.	Move byte at (*offset*) to AL.
A1	MOV AX,*moffs16**	Valid	Valid	Move word at (*seg:offset*) to AX.
A1	MOV EAX,*moffs32**	Valid	Valid	Move doubleword at (*seg:offset*) to EAX.
REX.W + A1	MOV RAX,*moffs64**	Valid	N.E.	Move quadword at (*offset*) to RAX.
A2	MOV *moffs8*,AL	Valid	Valid	Move AL to (*seg:offset*).
REX.W + A2	MOV *moffs8****,AL	Valid	N.E.	Move AL to (*offset*).
A3	MOV *moffs16**,AX	Valid	Valid	Move AX to (*seg:offset*).

ADD—Add

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
04 *ib*	ADD AL, *imm8*	Valid	Valid	Add *imm8* to AL.
05 *iw*	ADD AX, *imm16*	Valid	Valid	Add *imm16* to AX.
05 *id*	ADD EAX, *imm32*	Valid	Valid	Add *imm32* to EAX.
REX.W + 05 *id*	ADD RAX, *imm32*	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to RAX.
80 /0 *ib*	ADD *r/m8, imm8*	Valid	Valid	Add *imm8* to *r/m8.*
REX + 80 /0 *ib*	ADD *r/m8**, *imm8*	Valid	N.E.	Add *sign*-extended *imm8* to *r/m64.*
81 /0 *iw*	ADD *r/m16, imm16*	Valid	Valid	Add *imm16* to *r/m16.*
81 /0 *id*	ADD *r/m32, imm32*	Valid	Valid	Add *imm32* to *r/m32.*
REX.W + 81 /0 *id*	ADD *r/m64, imm32*	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to *r/m64.*
83 /0 *ib*	ADD *r/m16, imm8*	Valid	Valid	Add *sign*-extended *imm8* to *r/m16.*
83 /0 *ib*	ADD *r/m32, imm8*	Valid	Valid	Add *sign*-extended *imm8* to *r/m32.*
REX.W + 83 /0 *ib*	ADD *r/m64, imm8*	Valid	N.E.	Add *sign*-extended *imm8* to *r/m64.*
00 /r	ADD *r/m8, r8*	Valid	Valid	Add *r8* to *r/m8.*
REX + 00 /r	ADD *r/m8**, *r8**	Valid	N.E.	Add *r8* to *r/m8.*
01 /r	ADD *r/m16, r16*	Valid	Valid	Add *r16* to *r/m16.*
01 /r	ADD *r/m32, r32*	Valid	Valid	Add r32 to *r/m32.*
REX.W + 01 /r	ADD *r/m64, r64*	Valid	N.E.	Add r64 to *r/m64.*
02 /r	ADD *r8, r/m8*	Valid	Valid	Add *r/m8* to *r8.*
REX + 02 /r	ADD *r8**, *r/m8**	Valid	N.E.	Add *r/m8* to *r8.*
03 /r	ADD *r16, r/m16*	Valid	Valid	Add *r/m16* to *r16.*
03 /r	ADD *r32, r/m32*	Valid	Valid	Add *r/m32* to *r32.*
REX.W + 03 /r	ADD *r64, r/m64*	Valid	N.E.	Add *r/m64* to *r64.*

NOTES:

* In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

7c

movq $1, %rcx

48	C7	C1	01 00 00 00
REX 1000	MOV r/m64, imm32	11 000 001 reg rcx	

```
          R64     R/M64
movq      %r9,  16(%rdi)
```

4C 89 8F 10 00 00 00

REX 1100 | MOV | 10 | 001 | 111

- REX 1100: ↑ R, ↑ R/M
- MOV: R/M64, R64
- 10: $disp_{32}(\%r_s)$
- 001: %r9 (↑ R)
- 111: %rdi (↑ R/M)