X64 Assembly Reference

COMP2100 & COMP6100 Updated 8 September 2021

Registers

64-bit register	Low 32 bits	Low 16 bits	Low 8 bits
%rax	%eax	%ax	%al
%rcx	%ecx	%cx	%cl
%rdx	%edx	%dx	%dl
%rbx	%ebx	%bx	%bl
%rsi	%esi	%si	%sil
%rdi	%edi	%di	%dil
%rsp	%esp	%sp	%spl
%rbp	%ebp	%bp	%bpl
%r8	%r8d	%r8w	%r8b
%r9	%r9d	%r9w	%r9b
%r10	%r10d	%r10w	%r10b
%r11	%r11d	%r11w	%r11b
%r12	%r12d	%r12w	%r12b
%r13	%r13d	%r13w	%r13b
%r14	%r14d	%r14w	%r14b
%r15	%r15d	%r15w	%r15b

Instruction Operands

Type	Form	Operand value
Immediate	$*Imm*	*Imm*
Register	*r*	R[*r*]
Memory	$Imm(r_b,r_i,s)$	M[Imm+R[r_b]+R[r_i]*s]

Instructions

Each of the following 64-bit (quadword) instructions has variants for byte operations (ending in b), 16-bit word operations (ending in w) and 32-bit longword operations (ending in l). Disassembly may rely on registers named as operands S or D to indicate the length of data.

Instruction	Effect	Description	Sets Condition Codes?
movq S,D	D ← S	Move	N
xchgq S,D	D ↔ S	Exchange S with D	N
incq D	D ← D + 1	Increment	Y
decq D	D ← D – 1	Decrement	Y
negq D	D ← -D	Negate	Y
notq D	D ← ~D	Complement	Y
addq S,D	D ← D + S	Add	Y
subq S,D	D ← D – S	Subtract	Y
xorq S,D	D ← D ^ S	Exclusive-or	Y
andq S,D	D ← D & S	And	Y
orq S,D	D ← D | S	Or	Y
salq k,D	D ← D << k	Shift left	Y
shlq k,D	D ← D << k	Shift left (same as salq)	Y
sarq k,D	D ← D $>>_A$ k	Arithmetic shift right	Y
shrq k,D	D ← D $>>_L$ k	Logical shift right	Y
cmpq S2,S1	CC ← compare(S1 - S2)	Compare S1 with S2	Y
testq S2,S1	CC ← test(S1 & S2)	Test bits	Y

Miscellaneous instructions

Instruction		**Effect**	**Description**
`leaq`	`S,D`	`D ← &S`	Load effective address. Does not set condition codes.

`pushq`	`S`	`%rsp ← %rsp - 8`
		`M[%rsp] ← S`
			Push onto stack

`popq`	`D`	`D ← M[%rsp]`
		`%rsp ← %rsp + 8`
			Pop from stack

`movabs`	`S,D`	`D ← S`	Move 64-bit immediate into register, or move data from a 64-bit address
`movsbw`	`S,D`	`D ← signextend(S)`	Move sign-extended byte to word
`movsbl`	`S,D`	`D ← signextend(S)`	Move sign-extended byte to longword
`movswl`	`S,D`	`D ← signextend(S)`	Move sign-extended word to longword
`movsbq`	`S,D`	`D ← signextend(S)`	Move sign-extended byte to quadword
`movswq`	`S,D`	`D ← signextend(S)`	Move sign-extended word to quadword
`movslq`	`S,D`	`D ← signextend(S)`	Move sign-extended longword to quadword
`movzbw`	`S,D`	`D ← zeroextend(S)`	Move zero-extended byte to word
`movzbl`	`S,D`	`D ← zeroextend(S)`	Move zero-extended byte to longword
`movzwl`	`S,D`	`D ← zeroextend(S)`	Move zero-extended word to longword
`movzbq`	`S,D`	`D ← zeroextend(S)`	Move zero-extended byte to quadword
`movzwq`	`S,D`	`D ← zeroextend(S)`	Move zero-extended word to quadword
`movzlq`	`S,D`	`D ← zeroextend(S)`	Move zero-extended longword to quadword
`nop`		`No change`	No operation
`outb`	`S,D`	`IO[D] ← S`	Output byte
`inb`	`S,D`	`D ← IO[S]`	Input byte

Jump instructions

Instruction		**Synonyms**	**Description**
`call`	`label`		Procedure call
`call`	`*operand`		Indirect procedure call
`ret`			Return from call
`jmp`	`label`		Direct jump
`jmp`	`*operand`		Indirect jump
`je`	`label`	`jz`	Equal/zero
`jne`	`label`	`jnz`	Not equal/not zero
`js`	`label`		Negative
`jns`	`label`		Non-negative
`jg`	`label`	`jnle`	Greater (signed >)
`jge`	`label`	`jnl`	Greater or equal (signed >=)
`jl`	`label`	`jnge`	Less (signed <)
`jle`	`label`	`jng`	Less or equal (signed <=)
`ja`	`label`	`jnbe`	Above (unsigned >)
`jae`	`label`	`jnb`	Above or equal (unsigned >=)
`jb`	`label`	`jnae`	Below (unsigned <)
`jbe`	`label`	`jna`	Below or equal (unsigned <=)

Set instructions. There is a set instruction corresponding to each conditional jump. Only a few examples are shown here. The set instructions use a single byte destination register D which is set to 1 if the condition is true, otherwise to 0. The remainder of the 64-bit register is unchanged.

Instruction	Synonym	Effect	Description
sete D	setz	D ← ZF	Equal / zero
sets D		D ← SF	Negative
setl D	setnge	D ← SF ^ OF	Less (signed <)
setb D	setnae	D ← CF	Below (unsigned <)

Conditional move - examples. There is a conditional move corresponding to each conditional jump. In x64, conditional move instructions are available for 16-bit, 32-bit and 64-bit registers. The register names indicate the size of the object being moved.

Instruction	Synonym	Effect	Description
cmove S,D	cmovz	if (ZF) D ← S	Move if equal / zero
cmovs S,D		if (SF) D ← S	Move if negative
cmovl S,D		if (SF ^ OF) D ← S	Move if less than (signed <)
cmovb S,D		if (CF) D ← S	Move if below (unsigned <)

Integer multiply. imul is signed multiplication whereas mul is unsigned. There are a variety of multiply instructions for special purposes. The two shown here are signed two-operand multiplication and a three-operand version for multiplying a source register by an immediate value and storing the result in another register.

imulq S,D	D ← D * S	Signed multiply	Partially
imulq Imm,S,D	D ← I * S	Signed immediate multiply	Partially

Linux procedure call conventions

Parameters: `%rdi, %rsi, %rdx, %rcx, %r8, %r9`. Additional parameters are passed on the stack.
Return value: `%rax`
Caller-save registers: Parameters and `%rax, %r10, %r11`
Callee-save registers: `%rbx, %rbp, %r12, %r13, %r14, %r15`
Special: `%rsp`

ASCII Reference

Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NUL	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BEL	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	0x2F	47	/	0x4F	79	O	0x6F	111	o
10	16	DLE	0x30	48	0	0x50	80	P	0x70	112	p
11	17	DC1	0x31	49	1	0x51	81	Q	0x71	113	q
12	18	DC2	0x32	50	2	0x52	82	R	0x72	114	r
13	19	DC3	0x33	51	3	0x53	83	S	0x73	115	s
14	20	DC4	0x34	52	4	0x54	84	T	0x74	116	t
15	21	NAK	0x35	53	5	0x55	85	U	0x75	117	u
16	22	SYN	0x36	54	6	0x56	86	V	0x76	118	v
17	23	ETB	0x37	55	7	0x57	87	W	0x77	119	w
18	24	CAN	0x38	56	8	0x58	88	X	0x78	120	x
19	25	EM	0x39	57	9	0x59	89	Y	0x79	121	y
1A	26	SUB	0x3A	58	:	0x5A	90	Z	0x7A	122	z
1B	27	ESC	0x3B	59	;	0x5B	91	[	0x7B	123	{
1C	28	FS	0x3C	60	<	0x5C	92	\	0x7C	124	|
1D	29	GS	0x3D	61	=	0x5D	93	]	0x7D	125	}
1E	30	RS	0x3E	62	>	0x5E	94	^	0x7E	126	~
1F	31	US	0x3F	63	?	0x5F	95	_	0x7F	127	Delete