Cheatography

x86 cheatsheet Cheat Sheet

by notaperv via cheatography.com/11848/cs/1378/

Common register usage

Register	Special usage	Called function preserves contents
rax	1st function return value.	No
rbx	Optional base pointer.	Yes
rcx	Pass 4*th* argument to function.	No
rdx	Pass 3*rd* argument to function; 2*nd* function return value.	No
rsp	Stack pointer.	Yes
rbp	Optional frame pointer.	Yes
rdi	Pass 1*st* argument to function.	No
rsi	Pass 2*nd* argument to function.	No
r8	Pass 5*th* argument to function.	No
r9	Pass 6*th* argument to function.	No
r10	Pass function's static chain pointer.	No
r11		No
r12		Yes
r13		Yes
r14		Yes
r15		Yes

Registers

bits 63-0	bits 31-0	bits 15-0	bits 15-8	bits 7-0
rax	eax	ax	ah	al
rbx	ebx	bx	bh	bl
rcx	ecx	cx	ch	cl
rdx	edx	dx	dh	dl
rsi	esi	si		sil
rdi	edi	di		dil
rbp	ebp	bp		bpl
rsp	esp	sp		spl
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

Register argument order

Argument	Register
first	rdi
second	rsi
third	rdx
fourth	rcx
fifth	r8
sixth	r9

Common Jumps

instruction	meaning	immediately after a `cmp` ...
ja	jump above	jump if destination is above source in sequence
jae	jump above or equal	jump if destination is above or in same place as source in sequence
jb	jump below	jump if destination is below source in sequence
jbe	jump below or equal	jump if destination is below or in same place as source in sequence

Table 10.2: Conditional jump instructions for unsigned values.

More Common Jumps

instruction	meaning	immediately after a `cmp` ...
jg	jump greater	jump if destination is greater than source
jge	jump greater or equal	jump if destination is greater than or equal to source
jl	jump less	jump if destination is less than source
jle	jump less or equal	jump if destination is less than or equal to source

Table 10.3: Conditional jump instructions for signed values.

Flow Control

program flow control:

opcode	location	action	see page:
call	*label*	call function	173
iret		return from kernel function	388
ja	*label*	jump above (unsigned)	239
jae	*label*	jump above/equal (unsigned)	239
jb	*label*	jump below (unsigned)	239
jbe	*label*	jump below/equal (unsigned)	239
je	*label*	jump equal	239
jg	*label*	jump greater than (signed)	240
jge	*label*	jump greater than/equal (signed)	240
jl	*label*	jump less than (signed)	240
jle	*label*	jump less than/equal (signed)	240
jmp	*label*	jump	241
jne	*label*	jump not equal	239
jno	*label*	jump no overflow	239
j*cc*	*label*	jump on condition codes	239
leave		undo stack frame	192
ret		return from function	192
syscall		call kernel function	201
sysret		return from kernel function	390

cc = condition codes

Jump List

instruction	action	condition codes
ja	jump if above	$(CF = 0) \cdot (ZF = 0)$
jae	jump if above or equal	$CF = 0$
jb	jump if below	$CF = 1$
jbe	jump if below or equal	$(CF = 1) + (ZF = 1)$
jc	jump if carry	$CF = 1$
jcxz	jump if cx register zero	
jecxz	jump if ecx register zero	
jrcxz	jump if rcx register zero	
je	jump if equal	$ZF = 1$
jg	jump if greater	$(ZF = 0) \cdot (SF = OF)$
jge	jump if greater or equal	$SF = OF$
jl	jump if less	$SF \neq OF$
jle	jump if less or equal	$(ZF = 1) + (SF \neq OF)$
jna	jump if not above	$(CF = 1) + (ZF = 1)$
jnae	jump if not above or equal	$CF = 1$
jnb	jump if not below	$CF = 0$
jnbe	jump if not below or equal	$(CF = 0) \cdot (ZF = 0)$
jnc	jump if not carry	$CF = 0$
jne	jump if not equal	$ZF = 0$
jng	jump if not greater	$(ZF = 1) + (SF \neq OF)$
jnge	jump if not greater or equal	$SF \neq OF$
jnl	jump if not less	$SF = OF$
jnle	jump if not less or equal	$(ZF = 0) \cdot (SF = OF)$
jno	jump if not overflow	$OF = 0$
jnp	jump if not parity or equal	$PF = 0$
jns	jump if not sign	$SF = 0$
jnz	jump if not zero	$ZF = 0$
jo	jump if overflow	$OF = 1$
jp	jump if parity	$PF = 1$
jpe	jump if parity even	$PF = 1$
jpo	jump if parity odd	$PF = 0$
js	jump if sign	$SF = 1$
jz	jump if zero	$ZF = 1$

Table 10.1: Conditional jump instructions.

Arithmatic functions

arithmetic/logic:

opcode	source	destination	action	see page:
adds	$imm/%reg	%reg/mem	add	214
adds	*mem*	%reg	add	214
ands	$imm/%reg	%reg/mem	bit-wise and	290
ands	*mem*	%reg	bit-wise and	290
cmps	$imm/%reg	%reg/mem	compare	237
cmps	*mem*	%reg	compare	237
decs	%reg/mem		decrement	249
divs	%reg/mem		unsigned divide	315
idivs	%reg/mem		signed divide	317
imuls	%reg/mem		signed multiply	310
incs	%reg/mem		increment	248
leaw	*mem*	%reg	load effective address	191
muls	%reg/mem		unsigned multiply	309
negs	%reg/mem		negate	322
ors	$imm/%reg	%reg/mem	bit-wise inclusive or	290
ors	*mem*	%reg	bit-wise inclusive or	290
sals	$imm/%cl	%reg/mem	shift arithmetic left	302
sars	$imm/%cl	%reg/mem	shift arithmetic right	301
shls	$imm/%cl	%reg/mem	shift left	302
shrs	$imm/%cl	%reg/mem	shift right	301
subs	$imm/%reg	%reg/mem	subtract	215
subs	*mem*	%reg	subtract	215
tests	$imm/%reg	%reg/mem	test bits	238
tests	*mem*	%reg	test bits	238
xors	$imm/%reg	%reg/mem	bit-wise exclusive or	290
xors	*mem*	%reg	bit-wise exclusive or	290

s = b, w, l, q; *w* = l, q

x87 Floating Point

x87 floating point:

opcode	source	destination	action	see page:
fadds	*memfloat*		add	373
faddp			add/pop	373
fchs			change sign	373
fcoms	*memfloat*		compare	373
fcomp			compare/pop	373
fcos			cosine	373
fdivs	*memfloat*		divide	373
fdivp			divide/pop	373
filds	*memint*		load integer	373
fists		*memint*	store integer	373
fld*s*	*memint*		load floating point	373
fmuls	*memfloat*		multiply	373
fmulp			multiply/pop	373
fsin			sine	373
fsqrt			square root	373
fst*s*		*memint*	floating point store	373
fsubs	*memfloat*		subtract	373
fsubp			subtract/pop	373

s = b, w, l, q; *w* = l, q

SSE floating point

SSE floating point conversion:

opcode	source	destination	action	see page:
cvtsd2si	%xmmreg/mem	%reg	scalar double to signed integer	368
cvtsd2ss	%xmmreg	%xmmreg/%reg	scalar double to single float	368
cvtsi2sd	%reg	%xmmreg/mem	signed integer to scalar double	368
cvtsi2sdq	%reg	%xmmreg/mem	signed integer to scalar double	368
cvtsi2ss	%reg	%xmmreg/mem	signed integer to scalar single	368
cvtsi2ssq	%reg	%xmmreg/mem	signed integer to scalar single	368
cvtss2sd	%xmmreg	%xmmreg/mem	scalar single to scalar double	368
cvtss2si	%xmmreg/mem	%reg	scalar single to signed integer	368
cvtss2siq	%xmmreg/mem	%reg	scalar single to signed integer	368