School of
Information Systems

SMU
SINGAPORE MANAGEMENT
UNIVERSITY

Computer Hardware and Embedded Systems

Cheat Sheet

Data Representation

- In C variables can be of different types & sizes
  - Integer Types on **32-bit** **(64-bit)** architectures

C Type (Signed)	C Type (Unsigned)	Bytes	Bits	X86 Name
char	unsigned char	**1**	**8**	byte
short	unsigned short	**2**	**16**	word
int / int32	unsigned int /unit_32	**4**	**32**	double word
long	unsigned long	**4 (8)**	**32 (64)**	double (quad) word
long long / int64_t	unsigned long long / uint64_t	**8**	**64**	quad word
char *	-	**4 (8)**	**32 (64)**	double (quad) word
int *	-	**4 (8)**	**32 (64)**	double (quad) word

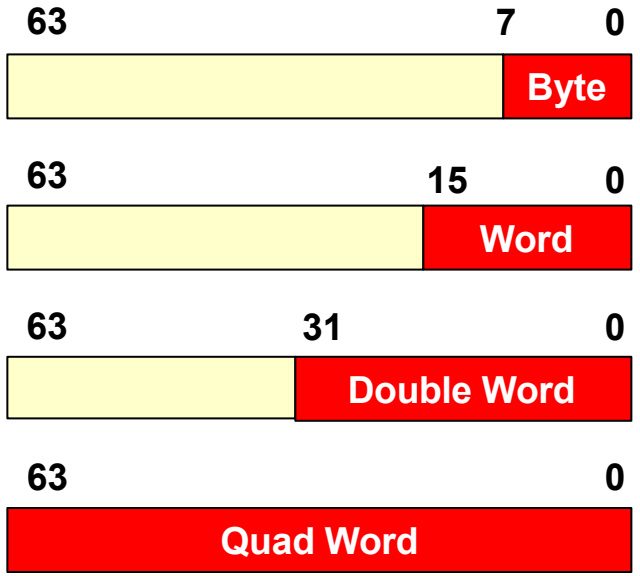
C Operator Precedence

Precedence	Operator	Description
1	++ -- + - ! ~	Prefix increment and decrement Unary plus and minus Logical NOT and bitwise NOT
2	* / %	Multiplication, division, and remainder
3	+ -	Addition and subtraction
4	<< >>	Bitwise left shift and right shift
5	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively
6	== !=	For relational = and ≠ respectively
7	&	Bitwise AND
8	^	Bitwise XOR
9	|	Bitwise OR
10	&&	Logical AND
11	||	Logical OR
12	=	Simple assignment

x86-64 Data Sizes

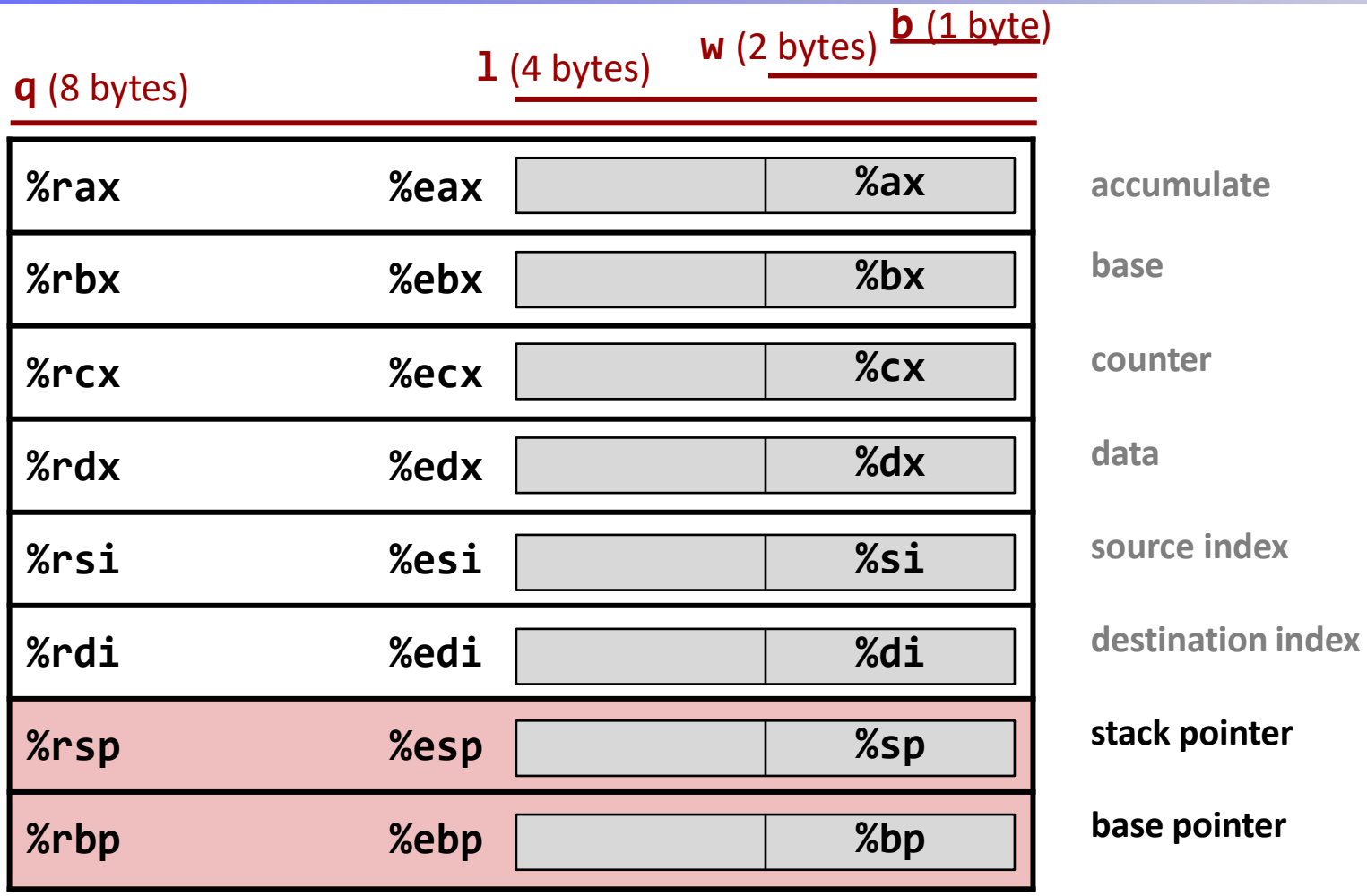
Integer

- 4 Sizes Defined
  - Byte (B)
    - 8-bits
  - Word (W)
    - 16-bits = 2 bytes
  - Double word (L)
    - 32-bits = 4 bytes
  - Quad word (Q)
    - 64-bits = 8 bytes



In x86-64, instructions generally specify what size data to access from memory and then operate upon.

Intel x86 Register Set



In addition: **%al**, **%bl**, **%cl**, **%dl**, **%sil**, **%dil**, **%spl**, **%bpl** for LSB

In addition: **%r8** to **%r15** (**%r8d** / **%r8w** / **%r8b** for lower 4 / 2 / 1 bytes)

School of Information Systems

SMU Singapore Management University

Common x86-64 Addressing Modes

Name	Form	Example	Description
Immediate	`$imm`	`movq $-500,%rax`	R[rax] = imm.
Register	r_a	`movq %rdx,%rax`	R[rax] = R[r_a]
Direct	`imm`	`movq 2000,%rax`	R[rax] = M[imm]
Indirect	(r_a)	`movq (%rdx),%rax`	R[rax] = M[R[r_a]]
Base + Displacement	imm(r_b)	`movq 40(%rdx),%rax`	R[rax] = M[R[r_b]+imm]
Base + Scaled Index	(r_b,r_i,s†)	`movq (%rdx,%rcx,4),%rax`	R[rax] = M[R[r_b]+R[r_i]*s]
Base + Scaled Index + Displacement	imm(r_b,r_i,s†)	`movq 80(%rdx,%rcx,2),%rax`	R[rax] = M[imm + R[r_b]+R[r_i]*s]

†Known as the scale factor and can be **{1,2,4, or 8}**

Imm = Constant, R[*x*] = Content of register *x*, M[addr] = Content of memory @ addr.

Purple values = effective address (EA) = Actual address used to get the operand

School of Information Systems

SMU Singapore Management University

Arithmetic and Logic Instructions

C operator	Assembly	Notes
+	add[b,w,l,q] src1,src2/dst	src2/dst += src1
-	sub[b,w,l,q] src1,src2/dst	src2/dst -= src1
&	and[b,w,l,q] src1,src2/dst	src2/dst &= src1
|	or[b,w,l,q] src1,src2/dst	src2/dst |= src1
^	xor[b,w,l,q] src1,src2/dst	src2/dst ^= src1
~	not[b,w,l,q] src/dst	src/dst = ~src/dst
-	neg[b,w,l,q] src/dst	src/dst = (~src/dst) + 1
++	inc[b,w,l,q] src/dst	src/dst += 1
--	dec[b,w,l,q] src/dst	src/dst -= 1
* (signed)	imul[b,w,l,q] src1,src2/dst	src2/dst *= src1
<< (signed)	sal cnt, src/dst	src/dst = src/dst << cnt
<< (unsigned)	shl cnt, src/dst	src/dst = src/dst << cnt
>> (signed)	sar cnt, src/dst	src/dst = src/dst >> cnt
>> (unsigned)	shr cnt, src/dst	src/dst = src/dst >> cnt
==, <, >, <=, >=, != (x ? y : z)	cmp[b,w,l,q] src1, src2 test[b,w,l,q] src1, src2	cmp performs: src2 – src1 test performs: src1 & src2

Conditional Jump Instructions

- Figure 3.15 from CS:APP, 3e

Instruction	Synonym	Jump Condition	Description
`jmp label`			
`jmp *(Operand)`			
`je label`	`jz`	ZF	Equal / zero
`jne label`	`jnz`	~ZF	Not equal / not zero
`js label`		SF	Negative
`jns label`		~SF	Non-negative
`jg label`	`jnle`	~(SF ^ OF) & ~ZF	Greater (signed >)
`jge label`	`jnl`	~(SF ^ OF)	Greater or Equal (signed >=)
`jl label`	`jnge`	(SF ^ OF)	Less (signed <)
`jle label`	`jng`	(SF ^ OF) | ZF	Less of equal (signed <=)
`ja label`	`jnbe`	~CF & ~ZF	Above (unsigned >)
`jae label`	`jnb`	~CF	Above or equal (unsigned >=)
`jb label`	`jnae`	CF	Below (unsigned <)
`jbe label`	`jna`	CF | ZF	Below or equal (unsigned <=)

Reminder: For all jump instructions other than jmp (which is unconditional), some previous instruction (cmp, test, etc.) is needed to set the condition codes to be examined by the jmp