

CSE 351 Reference Sheet (Midterm)

Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

IEEE 754 FLOATING-POINT STANDARD

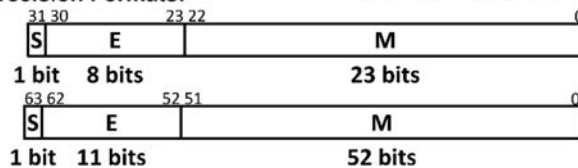
Value: $\pm 1 \times \text{Mantissa} \times 2^{\text{Exponent}}$

Bit fields: $(-1)^S \times 1.M \times 2^{(E-\text{bias})}$

where Single Precision Bias = 127,
Double Precision Bias = 1023.

IEEE Single Precision and

Double Precision Formats:



IEEE 754 Symbols

Exponent	Fraction	Object
0	0	± 0
0	$\neq 0$	\pm Denorm
1 to MAX - 1	anything	\pm Fl. Pt. Num.
MAX	0	$\pm \infty$
MAX	$\neq 0$	NaN

S.P. MAX = 255, D.P. MAX = 2047

Assembly Instructions

<code>mov a, b</code>	Copy from a to b.
<code>movs a, b</code>	Copy from a to b with sign extension. Needs <i>two</i> width specifiers.
<code>movz a, b</code>	Copy from a to b with zero extension. Needs <i>two</i> width specifiers.
<code>lea a, b</code>	Compute address and store in b. <i>Note:</i> the scaling parameter of memory operands can only be 1, 2, 4, or 8.
<code>push src</code>	Push <code>src</code> onto the stack and decrement stack pointer.
<code>pop dst</code>	Pop from the stack into <code>dst</code> and increment stack pointer.
<code>call <func></code>	Push return address onto stack and jump to a procedure.
<code>ret</code>	Pop return address and jump there.
<code>add a, b</code>	Add from a to b and store in b (and sets flags).
<code>sub a, b</code>	Subtract a from b (compute $b-a$) and store in b (and sets flags).
<code>imul a, b</code>	Multiply a and b and store in b (and sets flags).
<code>and a, b</code>	Bitwise AND of a and b, store in b (and sets flags).
<code>sar a, b</code>	Shift value of b <i>right (arithmetic)</i> by a bits, store in b (and sets flags).
<code>shr a, b</code>	Shift value of b <i>right (logical)</i> by a bits, store in b (and sets flags).
<code>shl a, b</code>	Shift value of b <i>left</i> by a bits, store in b (and sets flags).
<code>cmp a, b</code>	Compare b with a (compute $b-a$ and set condition codes based on result).
<code>test a, b</code>	Bitwise AND of a and b and set condition codes based on result.
<code>jmp <label></code>	Unconditional jump to address.
<code>j* <label></code>	Conditional jump based on condition codes (<i>more on next page</i>).
<code>set* a</code>	Set byte based on condition codes.

Conditionals

Instruction		(op) s, d	test a, b	cmp a, b
je	“Equal”	d (op) s == 0	b & a == 0	b == a
jne	“Not equal”	d (op) s != 0	b & a != 0	b != a
js	“Sign” (negative)	d (op) s < 0	b & a < 0	b-a < 0
jns	(non-negative)	d (op) s >= 0	b & a >= 0	b-a >= 0
jg	“Greater”	d (op) s > 0	b & a > 0	b > a
jge	“Greater or equal”	d (op) s >= 0	b & a >= 0	b >= a
jl	“Less”	d (op) s < 0	b & a < 0	b < a
jle	“Less or equal”	d (op) s <= 0	b & a <= 0	b <= a
ja	“Above” (unsigned >)	d (op) s > 0U	b & a < 0U	b > a
jb	“Below” (unsigned >)	d (op) s < 0U	b & a > 0U	b < a

Registers

Name	Convention	Name of “virtual” register		
		Lowest 4 bytes	Lowest 2 bytes	Lowest byte
%rax	Return value – Caller saved	%eax	%ax	%al
%rbx	Callee saved	%ebx	%bx	%bl
%rcx	Argument #4 – Caller saved	%ecx	%cx	%cl
%rdx	Argument #3 – Caller saved	%edx	%dx	%dl
%rsi	Argument #2 – Caller saved	%esi	%si	%sil
%rdi	Argument #1 – Caller saved	%edi	%di	%dil
%rsp	Stack Pointer	%esp	%sp	%spl
%rbp	Callee saved	%ebp	%bp	%bpl
%r8	Argument #5 – Caller saved	%r8d	%r8w	%r8b
%r9	Argument #6 – Caller saved	%r9d	%r9w	%r9b
%r10	Caller saved	%r10d	%r10w	%r10b
%r11	Caller saved	%r11d	%r11w	%r11b
%r12	Callee saved	%r12d	%r12w	%r12b
%r13	Callee saved	%r13d	%r13w	%r13b
%r14	Callee saved	%r14d	%r14w	%r14b
%r15	Callee saved	%r15d	%r15w	%r15b

Sizes

C type	x86-64 suffix	Size (bytes)
char	b	1
short	w	2
int	l	4
long	q	8