

CS 61C Reference Card

Version 1.8.0

	Instruction	Name	Description	Type	Opcode	Funct3	Funct7
Arithmetic	add rd rs1 rs2	ADD	rd = rs1 + rs2	R	011 0011	000	000 0000
	sub rd rs1 rs2	SUBtract	rd = rs1 - rs2	R	011 0011	000	010 0000
	and rd rs1 rs2	bitwise AND	rd = rs1 & rs2	R	011 0011	111	000 0000
	or rd rs1 rs2	bitwise OR	rd = rs1 rs2	R	011 0011	110	000 0000
	xor rd rs1 rs2	bitwise XOR	rd = rs1 ^ rs2	R	011 0011	100	000 0000
	sll rd rs1 rs2	Shift Left Logical	rd = rs1 << rs2	R	011 0011	001	000 0000
	srl rd rs1 rs2	Shift Right Logical	rd = rs1 >> rs2 (Zero-extend)	R	011 0011	101	000 0000
	sra rd rs1 rs2	Shift Right Arithmetic	rd = rs1 >> rs2 (Sign-extend)	R	011 0011	101	010 0000
	slt rd rs1 rs2	Set Less Than (signed)	rd = (rs1 < rs2) ? 1 : 0	R	011 0011	010	000 0000
	sltu rd rs1 rs2	Set Less Than (Unsigned)		R	011 0011	011	000 0000
	addi rd rs1 imm	ADD Immediate	rd = rs1 + imm	I	001 0011	000	
	andi rd rs1 imm	bitwise AND Immediate	rd = rs1 & imm	I	001 0011	111	
	ori rd rs1 imm	bitwise OR Immediate	rd = rs1 imm	I	001 0011	110	
	xori rd rs1 imm	bitwise XOR Immediate	rd = rs1 ^ imm	I	001 0011	100	
	slli rd rs1 imm	Shift Left Logical Immediate	rd = rs1 << imm	I*	001 0011	001	000 0000
	srl rd rs1 imm	Shift Right Logical Immediate	rd = rs1 >> imm (Zero-extend)	I*	001 0011	101	000 0000
	srai rd rs1 imm	Shift Right Arithmetic Immediate	rd = rs1 >> imm (Sign-extend)	I*	001 0011	101	010 0000
Memory	lb rd imm(rs1)	Load Byte	rd = 1 byte of memory at address rs1 + imm, sign-extended	I	000 0011	000	
	lbu rd imm(rs1)	Load Byte (Unsigned)	rd = 1 byte of memory at address rs1 + imm, zero-extended	I	000 0011	100	
	lh rd imm(rs1)	Load Half-word	rd = 2 bytes of memory starting at address rs1 + imm, sign-extended	I	000 0011	001	
	lhu rd imm(rs1)	Load Half-word (Unsigned)	rd = 2 bytes of memory starting at address rs1 + imm, zero-extended	I	000 0011	101	
	lw rd imm(rs1)	Load Word	rd = 4 bytes of memory starting at address rs1 + imm	I	000 0011	010	
	sb rs2 imm(rs1)	Store Byte	Stores least-significant byte of rs2 at the address rs1 + imm in memory	S	010 0011	000	
	sh rs2 imm(rs1)	Store Half-word	Stores the 2 least-significant bytes of rs2 starting at the address rs1 + imm in memory	S	010 0011	001	
	sw rs2 imm(rs1)	Store Word	Stores rs2 starting at the address rs1 + imm in memory	S	010 0011	010	

	Instruction	Name	Description	Type	Opcode	Funct3
Control	beq <i>rs1 rs2 label</i>	Branch if Equal	if (<i>rs1</i> == <i>rs2</i>) PC = PC + <i>offset</i>	B	110 0011	000
	bge <i>rs1 rs2 label</i>	Branch if Greater or Equal (signed)	if (<i>rs1</i> >= <i>rs2</i>) PC = PC + <i>offset</i>	B	110 0011	101
	bgeu <i>rs1 rs2 label</i>	Branch if Greater or Equal (Unsigned)	PC = PC + <i>offset</i>	B	110 0011	111
	blt <i>rs1 rs2 label</i>	Branch if Less Than (signed)	if (<i>rs1</i> < <i>rs2</i>) PC = PC + <i>offset</i>	B	110 0011	100
	bltu <i>rs1 rs2 label</i>	Branch if Less Than (Unsigned)	PC = PC + <i>offset</i>	B	110 0011	110
	bne <i>rs1 rs2 label</i>	Branch if Not Equal	if (<i>rs1</i> != <i>rs2</i>) PC = PC + <i>offset</i>	B	110 0011	001
	jal <i>rd label</i>	Jump And Link	<i>rd</i> = PC + 4 PC = PC + <i>offset</i>	J	110 1111	
	jalr <i>rd rs1 imm</i>	Jump And Link Register	<i>rd</i> = PC + 4 PC = <i>rs1</i> + <i>imm</i>	I	110 0111	000
Other	auipc <i>rd immu</i>	Add Upper Immediate to PC	<i>imm</i> = <i>immu</i> << 12 <i>rd</i> = PC + <i>imm</i>	U	001 0111	
	lui <i>rd immu</i>	Load Upper Immediate	<i>imm</i> = <i>immu</i> << 12 <i>rd</i> = <i>imm</i>	U	011 0111	
	ebreak	Environment BREAK	Asks the debugger to do something (<i>imm</i> = 0)	I	111 0011	000
	ecall	Environment CALL	Asks the OS to do something (<i>imm</i> = 1)	I	111 0011	000
Ext	mul <i>rd rs1 rs2</i>	MULTiply (part of mul ISA extension)	<i>rd</i> = <i>rs1</i> * <i>rs2</i>	(omitted)		

#	Name	Description	#	Name	Desc
x0	zero	Constant 0	x16	a6	<i>Args</i>
x1	ra	<i>Return Address</i>	x17	a7	
x2	sp	Stack Pointer	x18	s2	
x3	gp	Global Pointer	x19	s3	<i>Saved Registers</i>
x4	tp	Thread Pointer	x20	s4	
x5	t0	<i>Temporary Registers</i>	x21	s5	
x6	t1		x22	s6	
x7	t2		x23	s7	
x8	s0	Saved Registers	x24	s8	
x9	s1		x25	s9	
x10	a0	<i>Function Arguments or Return Values</i>	x26	s10	
x11	a1		x27	s11	
x12	a2	<i>Function Arguments</i>	x28	t3	<i>Temporaries</i>
x13	a3		x29	t4	
x14	a4		x30	t5	
x15	a5		x31	t6	
<i>Caller saved registers</i>					
<i>Callee saved registers (except x0, gp, tp)</i>					

Pseudoinstruction	Name	Description	Translation
beqz <i>rs1 label</i>	Branch if Equals Zero	if (<i>rs1</i> == 0) PC = PC + <i>offset</i>	beq <i>rs1</i> x0 <i>label</i>
bnez <i>rs1 label</i>	Branch if Not Equals Zero	if (<i>rs1</i> != 0) PC = PC + <i>offset</i>	bne <i>rs1</i> x0 <i>label</i>
j <i>label</i>	Jump	PC = PC + <i>offset</i>	jal x0 <i>label</i>
jr <i>rs1</i>	Jump Register	PC = <i>rs1</i>	jalr x0 <i>rs1</i> 0
la <i>rd label</i>	Load absolute Address	<i>rd</i> = & <i>label</i>	auipc , addi
li <i>rd imm</i>	Load Immediate	<i>rd</i> = <i>imm</i>	lui (if needed), addi
mv <i>rd rs1</i>	MoVe	<i>rd</i> = <i>rs1</i>	addi <i>rd</i> <i>rs1</i> 0
neg <i>rd rs1</i>	NEGate	<i>rd</i> = - <i>rs1</i>	sub <i>rd</i> x0 <i>rs1</i>
nop	No OPeration	do nothing	addi x0 x0 0
not <i>rd rs1</i>	bitwise NOT	<i>rd</i> = ~ <i>rs1</i>	xori <i>rd</i> <i>rs1</i> -1
ret	RETurn	PC = <i>ra</i>	jalr x0 <i>x1</i> 0
<div> <div>312524201915141211760</div> <div> <div>R</div> <div>funct7</div> <div>rs2</div> <div>rs1</div> <div>funct3</div> <div>rd</div> <div>opcode</div> </div> </div>			
<div> <div>I</div> <div>imm[11:0]</div> <div>rs1</div> <div>funct3</div> <div>rd</div> <div>opcode</div> </div>			
<div> <div>I*</div> <div>funct7</div> <div>imm[4:0]</div> <div>rs1</div> <div>funct3</div> <div>rd</div> <div>opcode</div> </div>			
<div> <div>S</div> <div>imm[11:5]</div> <div>rs2</div> <div>rs1</div> <div>funct3</div> <div>imm[4:0]</div> <div>opcode</div> </div>			
<div> <div>B</div> <div>imm[12 10:5]</div> <div>rs2</div> <div>rs1</div> <div>funct3</div> <div>imm[4:1 11]</div> <div>opcode</div> </div>			
<div> <div>U</div> <div>imm[31:12]</div> <div>rd</div> <div>opcode</div> </div>			
<div> <div>J</div> <div>imm[20 10:1 11 19:12]</div> <div>rd</div> <div>opcode</div> </div>			

Immediates are sign-extended to 32 bits, except in **I*** type instructions and **s1tiu**.

Selected ASCII values

HEX	DEC	CHAR	HEX	DEC	CHAR	HEX	DEC	CHAR	HEX	DEC	CHAR	HEX	DEC	CHAR	HEX	DEC	CHAR
0x20	32	SPACE	0x30	48	0	0x40	64	@	0x50	80	P	0x60	96	`	0x70	112	p
0x21	33	!	0x31	49	1	0x41	65	A	0x51	81	Q	0x61	97	a	0x71	113	q
0x22	34	"	0x32	50	2	0x42	66	B	0x52	82	R	0x62	98	b	0x72	114	r
0x23	35	#	0x33	51	3	0x43	67	C	0x53	83	S	0x63	99	c	0x73	115	s
0x24	36	\$	0x34	52	4	0x44	68	D	0x54	84	T	0x64	100	d	0x74	116	t
0x25	37	%	0x35	53	5	0x45	69	E	0x55	85	U	0x65	101	e	0x75	117	u
0x26	38	&	0x36	54	6	0x46	70	F	0x56	86	V	0x66	102	f	0x76	118	v
0x27	39	'	0x37	55	7	0x47	71	G	0x57	87	W	0x67	103	g	0x77	119	w
0x28	40	(0x38	56	8	0x48	72	H	0x58	88	X	0x68	104	h	0x78	120	x
0x29	41)	0x39	57	9	0x49	73	I	0x59	89	Y	0x69	105	i	0x79	121	y
0x2A	42	*	0x3A	58	:	0x4A	74	J	0x5A	90	Z	0x6A	106	j	0x7A	122	z
0x2B	43	+	0x3B	59	;	0x4B	75	K	0x5B	91	[0x6B	107	k	0x7B	123	{
0x2C	44	,	0x3C	60	<	0x4C	76	L	0x5C	92	\	0x6C	108	l	0x7C	124	
0x2D	45	-	0x3D	61	=	0x4D	77	M	0x5D	93]	0x6D	109	m	0x7D	125	}
0x2E	46	.	0x3E	62	>	0x4E	78	N	0x5E	94	^	0x6E	110	n	0x7E	126	~
0x2F	47	/	0x3F	63	?	0x4F	79	O	0x5F	95	_	0x6F	111	o	0x00	0	NULL

C Format String Specifiers

Specifier	Output
d or i	Signed decimal integer
u	Unsigned decimal integer
o	Unsigned octal
x	Unsigned hexadecimal integer, lowercase
X	Unsigned hexadecimal integer, uppercase
f	Decimal floating point, lowercase
F	Decimal floating point, uppercase
e	Scientific notation (significand/exponent), lowercase
E	Scientific notation (significand/exponent), uppercase
g	Use the shortest representation: %e or %f
G	Use the shortest representation: %E or %F
a	Hexadecimal floating point, lowercase
A	Hexadecimal floating point, uppercase
c	Character
s	String of characters
p	Pointer address

IEEE 754 Floating Point Standard

	Sign	Exponent	Significand
Single Precision	1 bit	8 bits (bias = -127)	23 bits
Double Precision	1 bit	11 bits (bias = -1023)	52 bits
Quad Precision	1 bit	15 bits (bias = -16383)	112 bits

Standard exponent bias: $-(2^{E-1}-1)$ where E is the number of exponent bits

SI Prefixes

Size	Prefix	Symbol	Size	Prefix	Symbol	Size	Prefix	Symbol
10^{-3}	milli-	m	10^3	kilo-	k	2^{10}	kibi-	Ki
10^{-6}	micro-	μ	10^6	mega-	M	2^{20}	mebi-	Mi
10^{-9}	nano-	n	10^9	giga-	G	2^{30}	gibi-	Gi
10^{-12}	pico-	p	10^{12}	tera-	T	2^{40}	tebi-	Ti
10^{-15}	femto-	f	10^{15}	peta-	P	2^{50}	pebi-	Pi
10^{-18}	atto-	a	10^{18}	exa-	E	2^{60}	exbi-	Ei
10^{-21}	zepto-	z	10^{21}	zetta-	Z	2^{70}	zebi-	Zi
10^{-24}	yocto-	y	10^{24}	yotta-	Y	2^{80}	yobi-	Yi

Laws of Boolean Algebra

$$\begin{array}{lll}
 x \cdot \bar{x} = 0 & x + \bar{x} = 1 & (xy)z = x(yz) \\
 x \cdot 0 = 0 & x + 1 = 1 & (x + y) + z = x + (y + z) \\
 x \cdot 1 = x & x + 0 = x & x(y + z) = xy + xz \\
 x \cdot x = x & x + x = x & x + yz = (x + y)(x + z) \\
 x \cdot y = y \cdot x & x + y = y + x & \overline{x \cdot y} = \bar{x} + \bar{y} \\
 xy + x = x & (x + y)x = x & \overline{(x + y)} = \bar{x} \cdot \bar{y}
 \end{array}$$

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✓		✓		✓		✓		✓		✓		✓		✓		✓		✓		
	p2		✓	✓			✓	✓			✓	✓			✓	✓			✓	✓		
	p4				✓	✓	✓	✓					✓	✓	✓	✓					✓	
	p8								✓	✓	✓	✓	✓	✓	✓	✓						
	p16																✓	✓	✓	✓	✓	



