Instruction	Name	Parameters	Function	Notes
			Accessing Memory	
ldr	load	<ra>, [<rb>]	load value from address in <rb> into <ra>	**offset** (in range [-256, 255]): ldr ra, *[rb, #offset]*
str	store	<ra>, [<rb>]	store value in <ra> to address in <rb>	**post index** (changes value to rb after load/store): ldr ra, *[rb], #30*
ldp	load pair	<ra1>, <ra2>, [<rb>]	load values from address in <rb> into <ra1> and <ra2>	**pre-index** (changes value to rb before load/store): ldr r0, *[r3, #30]!*
stp	store pair	<ra1>, <ra2>, [<rb>]	store values in <ra1> and <ra2> to address in <rb>	
			Loading Immediates	
mov	move	<ra>, #<immediate>, LSL #<shift>	load <immediate> into <ra>, optionally shifted left by <shift>	**immediate** must be 16 bits, **shift** must be multiple of 16
movk	move/keep	<ra>, #<immediate>, LSL #<shift>	same function as mov, without replacing any other bits	assembler can convert mov x12, #(1 << 21) into mov x12, 0x20, LSL #16
			Loading Addresses from Labels	
<label>:	label	<label>:	assembly code can be labeled using <label>:	
adr	load address	<ra>, <label>	load the address of the first instruction after the label to <ra>	used if label is within the same linker section
ldr	load address	<ra>, <label>	load the address of the first instruction after the label to <ra>	used if label is in different linker section
			Moving Between Registers	
mov	move	<ra1>, <ra2>	copy contents from register <ra2> to <ra1>	same instruction as loadign immediates
			Read and Write Special Registers	
msr	write	<special_register>, <ra>	write to a special register from another register	
mrs	read	<ra>, <special_register>	read from a special register into another register	
			Arithmetic and Logical Instructions	
add	add	<dest> <a> <b>	add <a> and <b>, store result in <dest>	**<dest>** must be a register, can be same as <a> or <b>
sub	subtract	<dest> <a> <b>	subtract <b> from <a>, store result in <dest>	**<a>** must be register
and	bitwise and	<dest> <a> <b>	bitwise and <a> and <b>, store result in <dest>	**<b>** may be register or immediate
orr	bitwise or	<dest> <a> <b>	bitwise or <a> and <b>, store result in <dest>	
			Branching	
b	jump	<label>	will unconditionally jump to address of <label>	
bl	store then jump	<label>	stores next address in link register and jumps to address of <label>	ret instruction jumps to address in link register
br	jump (register)	<ra>	same as b, but jumps to address in register <ra>	
blr	store then jump (register)	<ra>	same as bl, but jumps to address in register <ra>	ret instruction jumps to address in link register
			Conditional Branching	
cmp	compare	<ra1>, <ra2/immediate>	compares values in <ra1> with <ra2> or <immediate> and sets flags for future conditional branching instructions	
bne	branch not equal	<label>	branches to <label> if condition flags show not equal	if branch isn't taken execution continues forward
beq	branch if equal	<label>	branches to <label> if condition flags show equal	
blt	branch if less than	<label>	branches to <label> if condition flags show less than	
ble	branch less than or equal	<label>	branches to <label> if condition flags show less than or equal	
bgt	branch if greater than	<label>	branches to <label> if condition flags show greater than	
bge	branch greater than or equal	<label>	branches to <label> if condition flags show greater than or equal	
cbz	compare, branch on zero	<ra>, <label>	compares value in <ra> to zero, branches to <label> if equal	if branch isn't taken execution continues forward
cbnz	compare, branch if not zero	<ra>, <label>	compares value in <ra> to zero, branches to <label> if not equal	does not set condition flags