C Code	Description	ARM Assembly Equivalent				
Variable Assignment						
int x = 5;	Assign constant to variable	MOV R0, #5 (if x is in R0)				
int x = a + b;	Assign sum to variable	ADD R0, R1, R2 (where a is in R1, b is in R2, and x is in R0)				
Stack Operations						
N/A	Save registers to the stack	PUSH {R0, R1, R2, LR} (pushes R0, R1, R2, and link register LR to stack)				
N/A	Restore registers from the stack	POP {R0, R1, R2, PC} (pops values from stack into R0, R1, R2, and program counter PC to return)				
Arithmetic Operations						
x = a + b;	Addition	ADD R0, R1, R2				
x = a - b;	Subtraction	SUB R0, R1, R2				
x = a * b;	Multiplication	MUL R0, R1, R2				
x = a / b;	Division (integer)	SDIV R0, R1, R2				
x = a % b;	Modulus	SDIV R0, R1, R2 followed by MLS R0, R0, R2, R1				
x = -a;	Negation	NEG R0, R1 (where a is in R1 and result x is in R0)				
x = a >> b;	Arithmetic right shift (preserves sign)	ASR R0, R1, R2 (shift a in R1 right by b bits, storing result in R0)				
x = a << b;	Logical left shift	LSL R0, R1, R2				
Logical Operations						
x = a & b;	Bitwise AND	AND R0, R1, R2				
`x = a	b;`	Bitwise OR				
x = a ^ b;	Bitwise XOR	EOR R0, R1, R2				
x = ~a;	Bitwise NOT	MVN R0, R1				
Comparison Operations						
if (a == b)	Equality check	CMP R1, R2 \n BEQ label				
if (a != b)	Inequality check	CMP R1, R2 \n BNE label				
if (a < b)	Less than	CMP R1, R2 \n BLT label				
if (a <= b)	Less than or equal to	CMP R1, R2 \n BLE label				
if (a > b)	Greater than	CMP R1, R2 \n BGT label				
if (a >= b)	Greater than or equal to	CMP R1, R2 \n BGE label				
Branching (Unconditional and Conditional)						
goto label;	Unconditional branch	B label				
if (a == b)	Conditional branch (equal)	CMP R1, R2 \n BEQ label				
if (a != b)	Conditional branch (not equal)	CMP R1, R2 \n BNE label				
if (a < b)	Conditional branch (less than)	CMP R1, R2 \n BLT label				
Control Structures						
if (condition) { ... }	If statement	CMP ... \n BEQ/BNE/BLT/BGT ... \n ...				
if (condition) { ... } else { ... }	If-Else statement	CMP ... \n BEQ true_label \n ... \n B end_label \n true_label: ... \n end_label:				
while (condition) { ... }	While loop	loop_start: CMP ... \n BEQ end_loop \n ... \n B loop_start \n end_loop:				
for (i = 0; i < n; i++) { ... }	For loop	MOV R0, #0 \n loop_start: CMP R0, R1 \n BGE end_loop \n ... \n ADD R0, R0, #1 \n B loop_start \n end_loop:				
do { ... } while (condition);	Do-While loop	loop_start: ... \n CMP ... \n BNE loop_start				
switch (x) { case 1: ...; break; case 2: .	Switch statement	CMP R0, #1 \n BEQ case_1 \n CMP R0, #2 \n BEQ case_2 \n ... \n case_1: ... \n B end_switch \n case_2: ... \n end_switch:				
Function Calls and Returns						
int func(int a, int b) { ... }	Function declaration	func: ... \n MOV PC, LR				
x = func(a, b);	Function call	MOV R0, a \n MOV R1, b \n BL func				
return x;	Return from function	MOV R0, x \n MOV PC, LR				

Array Operations						
int arr[4];	Array declaration (static)	Allocate space manually, e.g., .word 0, 0, 0, 0				
x = arr[2];	Array access	LDR R0, [R1, #8] (assuming arr base address in R1 and each element is 4 bytes)				
arr[2] = x;	Array assignment	STR R0, [R1, #8]				
Pointer Operations						
int *p = &x;	Pointer assignment	LDR R0, =x				
x = *p;	Dereferencing pointer	LDR R0, [R1] (if p is in R1)				
*p = x;	Setting value through pointer	STR R0, [R1]				
Structs						
struct { int a; int b; } s;	Struct declaration	Allocate memory for fields and use offsets				
x = s.a;	Access struct member	LDR R0, [R1] (assuming s base address in R1)				
s.b = x;	Modify struct member	STR R0, [R1, #4]				
Advanced Control Flow						
break;	Break from loop	Use B to jump to the end of the loop				
continue;	Continue to next iteration	Use B to jump to the start of the loop				
Logical Expressions						
x = (a && b);	Logical AND	CMP R1, #0 \n BEQ false \n CMP R2, #0 \n MOVEQ R0, #0 \n MOVNE R0, #1 \n false:				
`x = (a		b);`				
Increment/Decrement						
x++;	Increment	ADD R0, R0, #1				
x--;	Decrement	SUB R0, R0, #1				