

12 Addressing Modes, Directives and Data Types in ARM

CPE 221

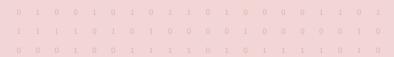
The University of Alabama in Huntsville

Rahul Bhadani March 25, 2025 Addressing Modes in ARMv7

Directivess in ARM Assembler

Data Types in ARMv7

Addressing Modes in ARMv7



Immediate Addressing Mode

The operand is a constant value embedded in the instruction.

Examples:

- 1. MOV RO, #0x3F Loads 0x3F into RO.
- 2. ADD R2, R3, #255 Adds 255 to R3 and stores result in R2.
- 3. CMP R4, #100 Compares R4 with value 100.

Advantages

- ► Fast access as no memory lookup required
- ▶ Instruction encoding is efficient for small constants



Register-Direct Addressing Mode

Operands are values stored in registers.

Examples:

- 1. ADD R1, R2, R3
 Adds R2 and R3, stores result in R1.
- MOV R5, R7 Copies value of R7 into R5.
- CMP RO, R1 Compares values in RO and R1.

Advantages

- Very fast execution (single cycle in most cases)
- ► No memory access overhead



Direct Addressing Mode

Memory address is specified directly (via label or absolute value).

Examples:

- LDR RO, =0x12345678
 Loads constant 0x12345678 into RO.
- LDR R1, data Loads value at label data into R1.

Implementation Note

- ARM typically uses PC-relative addressing for direct addressing.
- ▶ The assembler calculates the appropriate offset from the current PC value.



Register Indirect Addressing Mode

Address is stored in a register.

Examples:

- LDR RO, [R1] Loads value from address in R1 into RO.
- 2. STR R2, [R3] Stores R2 at address in R3.
- 3. LDRB R4, [R5] Loads byte from address in R5 into R4.

Usage

- Common for accessing array elements and data structures
- ▶ Efficient for pointer-based operations



Pre-increment Addressing Mode

Base register is updated **before** accessing memory.

Examples:

- 1. LDR RO, [R1, #4]!
 R1 += 4, then loads from new R1.
- 2. LDR R2, [R3, #-8]!
 R3 -= 8, then loads from R3.
- 3. STR R4, [R5, #16]!
 R5 += 16, then stores R4 at R5.



Post-increment Addressing Mode

Base register is updated after accessing memory.

Examples:

- 1. LDR RO, [R1], #4 Loads from R1, then R1 += 4.
- 2. STR R2, [R3], #8 Stores R2 at R3, then R3 += 8.
- 3. LDRH R4, [R5], #2
 Loads halfword from R5, then R5 += 2.



Directivess in ARM Assembler



Directives in ARM Assembler

Purpose: Control assembly process or define data. **Common Directives**:

- .text: Marks start of code section
- .data: Marks start of data section
- .global main: Declares main as global symbol
- .word 0x1234: Allocates 32-bit integer
- .asciz "Hello": Null-terminated string
- ▶ .align 4: Aligns data to 4-byte boundary



Memory Allocation Directives

Common Memory Allocation Directives

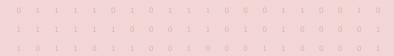
- space / .skip Reserve a block of memory
- align Align to a specified boundary
- balign Byte align to a specified boundary
- .p2align Align to a power of 2

Examples

```
    space 100 // Reserve 100 bytes of space
    align 4 // Align to a 4-byte (word) boundary
    buffer: space 1024 // Label a 1KB buffer
```



Data Types in ARMv7



Data Types in ARMv7

1. Byte (8-bit)

- ▶ .byte 0xAB Allocates 8-bit value 0xAB
- ▶ .byte 'A' Allocates ASCII character A

2. Halfword (16-bit)

- ▶ .hword 0x1234 Allocates 16-bit value
- ▶ .short 1000 Allocates 16-bit integer



Data Types in ARMv7 (Cont.)

3. Word (32-bit)

- .word OxDEADBEEF Allocates 32-bit value
- .int 42 Allocates 32-bit integer

4. Doubleword (64-bit)

- .quad 0x123456789ABCDEF0 Allocates 64-bit value
- .dword 3.14 Allocates 64-bit float

5. String

- .asciz "ARM" Null-terminated string
- ▶ .ascii "Test" Non-terminated string



Bit Manipulation and Special Types

Bit Fields

- ▶ Individual bits or bit fields within registers
- Manipulated using bit manipulation instructions
- ► Examples:

```
1. BFI RO, R1, #8, #4 // Bit field insert 4 bits at position 8
2. UBFX RO, R1, #4, #8 // Extract 8-bit unsigned field starting at bit 4
```

Packed BCD (Binary Coded Decimal)

- ▶ Decimal digits encoded as 4-bit values
- Used for certain arithmetic operations
- Examples:
 - 1. UADD8 R0, R1, R2 // Add 8-bit values independently (can be used for BCD)
 2. USUB8 R0, R1, R2 // Subtract 8-bit values independently



The End