THUMB INSTRUCTIONS IN ARM

The Thumb instruction set is a subset of the ARM instruction set designed to provide improved code density and reduced memory usage while maintaining performance. Thumb instructions are generally 16 bits in length, compared to the standard 32-bit ARM instructions. This makes them particularly useful for applications with limited memory resources, such as embedded systems.

Key Features of Thumb Instructions

1. **16-bit Instruction Length: Thumb instructions are encoded in 16 bits, allowing for more compact code.**
2. **Subset of ARM Instructions: Thumb instructions are a subset of the full ARM instruction set, offering a balance between code density and functionality.**
3. **Mode Switching: The processor can switch between ARM and Thumb instruction sets using the `BX` instruction.**

Common Thumb Instructions

Here are some commonly used Thumb instructions along with their ARM equivalents:

1. **Data Processing Instructions:**
   - `MOV Rd, Rs` **: Move the contents of Rs to Rd.**

- `ADD Rd, Rs, #imm` **: Add an immediate value to the contents of Rs and store the result in Rd.**
- `SUB Rd, Rs, #imm` **: Subtract an immediate value from the contents of Rs and store the result in Rd.**
- `CMP Rd, Rs` **: Compare the contents of Rd and Rs.**

2. **Load/Store Instructions:**
   - `LDR Rd, [Rn, #offset]` **: Load a word from memory into Rd.**
   - `STR Rd, [Rn, #offset]` **: Store a word from Rd into memory.**
   - `LDRB Rd, [Rn, #offset]` **: Load a byte from memory into Rd.**
   - `STRB Rd, [Rn, #offset]` **: Store a byte from Rd into memory.**
3. **Branch Instructions:**
   - `B label` **: Unconditional branch to the specified label.**
   - `BEQ label` **: Branch to the specified label if the Zero flag is set.**
   - `BNE label` **: Branch to the specified label if the Zero flag is clear.**
   - `BL label` **: Branch to the specified label and link (used for subroutine calls).**

Example Thumb Code

Here is an example of a simple Thumb assembly program:

```
.syntax unified
```

```
    .thumb


    .global _start


_start:

    MOV R0, #5          ; Load immediate value 5 into R0

    MOV R1, #10         ; Load immediate value 10 into R1

    ADD R2, R0, R1   ; Add R0 and R1, store result in R2

    CMP R2, #15         ; Compare R2 with immediate value 15

    BEQ equal_label  ; Branch to equal_label if R2 == 15

    B end_label  ; Branch to end_label unconditionally


equal_label:

    MOV R3, #1          ; Load immediate value 1 into R3


end_label:
    B end_label  ; Infinite loop to end the program
```

<u>Switching Between ARM and Thumb States</u>

The ARM processor can switch between ARM and Thumb states using the `BX` instruction. The `BX` instruction branches to an address and updates the Thumb state based on the least significant bit (LSB) of the address:

- **If the LSB is `0`, the processor switches to ARM state.**
- **If the LSB is `1`, the processor switches to Thumb state.**

```
    .global switch_to_thumb


switch_to_thumb:

    LDR R0, =thumb_code + 1 ; Load the address of thumb_code, with LSB set to 1

    BX R0                   ; Switch to Thumb state and branch to thumb_code


thumb_code:

    MOV R0, #5              ; Example Thumb instruction
```

```
B .             ; Infinite loop

.global switch_to_thumb
```

Advantages of Thumb Instructions

1. **Improved Code Density: Thumb instructions are smaller, which can reduce the size of the compiled code.**
2. **Efficient Use of Memory: Smaller instruction sizes lead to better utilization of limited memory resources, making Thumb ideal for embedded systems.**
3. **Performance: Despite being more compact, Thumb instructions often achieve similar performance to ARM instructions due to improved cache usage and reduced memory access times.**

Considerations

- **Instruction Set Limitations: Thumb instructions are a subset of ARM instructions, so not all ARM instructions are available in Thumb mode. This can sometimes require additional instructions to achieve the same functionality.**
- **Mode Switching Overhead: Switching between ARM and Thumb modes has a small performance overhead, so it should be minimized in performance-critical sections of code.**

Overall, Thumb instructions provide a valuable tool for optimizing code size and memory usage, especially in resource-constrained environments like embedded systems.