Lab 2: Introduction to ARM assembly language with Data Segment

Goals

1. Understanding of the fundamental concepts of Assembly Language
2. Able to write Simple Assembly program using ARM v7 ISA
3. Develop familiarity with a Computer System Simulator

Theory:

In ARM assembly, **LDR** (Load Register) and **STR** (Store Register) are fundamental instructions used to move data between memory and registers. Here's a detailed explanation of both:

1. LDR (Load Register)

The **LDR** instruction loads data from memory into a register. It's used to transfer data from a specific memory location (address) into a register.

Syntax:

LDR <destination register>, [<memory address>]

LDR R1, [R2]

Usage Variants:

- **Immediate addressing**: Load a constant value directly into a register.

  ```
  LDR R0, =0x1000  ; Load immediate value 0x1000 into R0
  ```

  This loads the address `0x1000` into `R0`. The `=` sign in ARM assembly is shorthand for loading an immediate value.

- **Register-based addressing**: Load data from an address specified by a register.

  ```
  LDR R1, [R2]  ; Load the value from memory at the address in R2 into R1
  ```

- **Offset addressing**: Load data from a memory address with an offset (displacement) from a base register.

  ```
  LDR R1, [R2, #4]  ; Load value from address (R2 + 4) into R1
  ```

  This loads the value from the memory address `R2 + 4` into `R1`.

---

2. STR (Store Register)

The **STR** instruction stores the value from a register into a specific memory location (address). It's used to transfer data from a register into memory.

Syntax:

```
STR <source register>, [<memory address>]
```

Example:

```
STR R1, [R2]
```

- This instruction stores the value in `R1` into the memory address stored in `R2`.

Usage Variants:

- **Register-based addressing**: Store the value of a register at the memory address specified by another register.

  ```
  STR R1, [R2]  ; Store the value of R1 into the memory location pointed to by R2
  ```

- **Offset addressing**: Store data in memory at an address with an offset from a base register.

  ```
  STR R1, [R2, #4]  ; Store R1's value into the address (R2 + 4)
  ```

Difference Between LDR and STR:

- **LDR** reads (loads) data from memory into a register.
- **STR** writes (stores) data from a register into memory.

Example Usage of Both:

```
LDR R0, =numbers      ; Load the address of the 'numbers' array into R0
LDR R1, [R0]          ; Load the first element of the array into R1
ADD R1, R1, #10       ; Add 10 to the value in R1
STR R1, [R0, #4]      ; Store the new value back into the second element of the array
```

In this example:

- **LDR R0, =numbers**: Loads the address of an array into `R0`.
- **LDR R1, [R0]**: Loads the value at the first element of the array into `R1`.
- **ADD R1, R1, #10**: Adds 10 to the value in `R1`.
- **STR R1, [R0, #4]**: Stores the new value back into the second element of the array (address `R0 + 4`).

Summary:

- **LDR** is for reading data from memory into a register.
- **STR** is for writing data from a register into memory.

These instructions allow ARM assembly programs to interact with data in memory, which is crucial for tasks such as processing arrays, accessing global variables, and manipulating data during program execution.

Data Segment:

The **data segment** in assembly programming is a section of memory where global variables, constants, and static data are stored. In ARM assembly, just like in most assembly languages, the program is divided into different segments, with the **data segment** being one of the key segments for storing static or constant data that will not change during program execution.

Key Features of the Data Segment:

- **Global Variables**: Variables declared in the data segment are typically available throughout the entire program.
- **Static Data**: Data defined in this segment remains in memory throughout the program's execution.
- **Constants**: You can define read-only constants in the data segment.
- **Uninitialized Data**: Variables that are declared but not initialized are stored in a separate section, typically the BSS (Block Started by Symbol) segment.

Segments in ARM Assembly:

1. **Text Segment (.text)**: Contains the executable instructions (code) of the program.
2. **Data Segment (.data)**: Contains initialized data (like global variables and constants).
3. **BSS Segment (.bss)**: Contains uninitialized variables or variables initialized to zero.

.data Segment Syntax:

The `.data` directive is used to define the data segment in ARM assembly.

Example of Defining the Data Segment:

```
.data
value1:  .word   5          ; Define a variable 'value1' initialized to 5
value2:  .word   10         ; Define another variable 'value2' initialized to 10
string1: .asciz  "Hello!"   ; Define a null-terminated string
```

- `value1` and `value2` are variables initialized to specific values (`5` and `10`).
- `string1` is a null-terminated string.

Common Directives Used in Data Segment:

- **`.word`**: Allocates a word (typically 4 bytes on 32-bit systems) for storing integers or addresses.
- **`.asciz`**: Allocates a null-terminated string.
- **`.byte`**: Allocates a single byte of data.

- **`.hword`**: Allocates a halfword (2 bytes).
- **`.space`**: Allocates a block of memory with uninitialized data.

Accessing Data in the Data Segment:

You can load values from the data segment into registers and store values back into the data segment using `LDR` and `STR` instructions.

Example: Using Data Segment in ARM Assembly:

```
.data
value1:  .word   5        @ Variable 'value1' with an initial value of 5
value2:  .word   10       @ Variable 'value2' with an initial value of 10

.text
.global _start

_start:
    LDR R0, =value1       @ Load address of 'value1' into R0
    LDR R1, [R0]          @ Load value from 'value1' into R1 (R1 = 5)

    LDR R0, =value2       @ Load address of 'value2' into R0
    LDR R2, [R0]          @ Load value from 'value2' into R2 (R2 = 10)

    ADD R3, R1, R2        @ Add values from R1 and R2 (5 + 10 = 15)

    STR R3, [R0]          @ Store result (15) in memory at address of 'value2'
```

- The program defines `value1` and `value2` in the data segment.
- It uses the `LDR` instruction to load the values of these variables into registers.
- It performs arithmetic and then uses `STR` to store the result back into memory.

Usage of Data Segment in Practical Scenarios:

- **Static Variables**: In embedded systems, constants like calibration data, configuration values, or read-only parameters are stored in the data segment.
- **Arrays and Buffers**: Arrays of values or buffers for input/output operations can be defined here and accessed throughout the program.
- **Strings**: Strings used for debugging or displaying messages can be placed in the data segment.

Lab Report

Lab 2: Introduction to ARM assembly language

Student Names: _____

Student ID: _____

Section: _____

Questions:

1. Do you know that ARM V7 does not have a divide instruction? What could be the possiblereason? (Hint. Divide instruction is not very common)
2. Write a pseudo algorithm that can perform divide instruction with other operations.
3. Explain LDR and STR and their Usecases.