CSE 331L: Microprocessor Interfacing and Embedded Systems Lab

Summer 2025

Arm Assembly (Part-1)

Class # 02

Recap

- Microprocessor
- Memory (Primary → Register)
- Register Type & Size
- Instructions Parts & Type
- CPULator Basics
- MOV
- ADD
- SUB

Instructions

- An instruction is a **low-level operation** or command that the processor executes. It tells the ARM processor what specific action to perform, such as arithmetic operations, data movement, branching, and more
- **MOV R0, R5** @ moves the value of R5 to R1

MOV

- Move (register) copies a value from a register to the destination register. It can optionally update the condition flags based on the value.

```
MOV{S}<c> <Rd>, <Rm>
```

Reference

ADD immediate

This instruction adds an immediate value to a register value, and writes the result to the destination register. It can optionally update the condition flags based on the result.

```
ADD{S}<c> <Rd>, <Rn>, #<const>
```

Reference

ADD

- This instruction adds a register value and an optionally-shifted **register value**, and writes the result to the destination register. It can optionally update the condition flags based on the result.

```
ADD{S}<c> <Rd>, <Rn>, <Rm>{, <shift>}
```

Reference

SUB immediate

This instruction subtracts an **immediate value** from a **register value**, and writes the result to the destination register. It can optionally update the condition flags based on the result.

```
SUB{S}<c> <Rd>, <Rn>, #<const>
```

Reference

SUB

- This instruction adds a register value and an optionally-shifted **register value**, and writes the result to the destination register. It can optionally update the condition flags based on the result.

```
SUB{S}<c>.W <Rd>, <Rn>, <Rm>{, <shift>}
```

Reference

MUL

- Multiply multiplies two register values. The least significant 32 bits of the result are written to the destination register. These 32 bits do not depend on whether the source register values are considered to be signed values or unsigned values.

```
MUL<c> <Rdm>, <Rn>, <Rdm>
```

Reference

Memory Segments

8086 vs Arm

Declaring Variables

- Must be declared in data segment

  var_name : .<data_type> <value>

```
.global _start

.data
a: .word 7
b: .word 10

.text
_start

ldr r0, =a
ldr r1, [r0]

mov r2, #9
ldr r3, =b
str r2, [r3]
```

LDR (Load Register)

- Load Register (register) calculates an address from a base register value and an offset register value, **loads a word from memory, and writes it to a register**. The offset register value can optionally be shifted.

```
LDR<c> <Rt>,[<Rn>,+/-<Rm>{, <shift>}]{!}
```

Reference

LDR (Load Register)

- Two steps:
  - First, load the address of a variable

ldr <register1> , **= <variable_name>**

  - then, use that address to get the values in a general purpose register

ldr <register2> , **[** <register1> **]**

  - Now, we get: **register2 = variable_name**

STR

- Store Register (register) calculates an address from a base register value and an offset register value, stores a word from a register to memory. The offset register value can optionally be shifted

```
STR<c> <Rt>,[<Rn>,+/-<Rm>{, <shift>}]{!}
```

STR

- Two steps:
  - First, load the address of a variable

    ldr <register1> , **= <variable_name>**

  - then, use that address to get the values in a general purpose register

    ldr <register2> , **[** <register1> **]**

  - Now, we get: **variable_name = register2**

How to find address manually!!