Lab 1 – Part 2: Debugging and Basic Assembly

Instructor:
Dr. Carl Latino
carl.latino@okstate.edu

Graduate Teaching Assistant:
Francisco E. Fernandes Jr.
feferna@okstate.edu

School of Electrical and Computer Engineering
Oklahoma State University
Fall 2019

Today's Goals

- **Addition using Assembly programming.**
- **Bitwise Operations and Masking.**
- **Learn how to debug your program.**
- **Write a small assembly program.**

Basic Assembly program – **main.s**

```
.syntax unified
.global main
.include "stm32l476xx_constants.s"
main:
        // Configure clock speed
        // Configure peripherals (GPIO)
        // Your program logic goes here!
stop:       B          stop // dead loop & program hangs here
```

Load Constant Values into Registers

- You can use `R0` to `R12` to hold your "variables".
- **`MOV Rd, #<immed_8>`**
  - **Loads a 8-bit immediate value (constant) to the register.**
  - **Example:**
    - `MOV R0, #0xFF`
    - R0 is now equal to 255 in decimal.
- **`LDR Rd, =<immed_8>`** or **`LDR Rd, =#<immed_8>`**
  - **Loads a 8-bit immediate value (constant) to the register.**
  - **Example:**
    - `LDR R0, =0xFF` or `LDR R0, =#0xFF`
    - R0 is now equal to 255 in decimal.
- **`LDR Rd, =<immed_32>`** or **`LDR Rd, =#<immed_32>`**
  - **Pseudo-instruction. Loads a 32-bit immediate value (constant) to the register.**

Simple Addition in Assembly

- **ADD {Rd,} Rn, Op2**
  - **Does NOT update NZCV flags.**
- **ADDS {Rd,} Rn, Op2**
  - **Updates NZCV flags.**

```
.syntax unified

.global main


main:
     MOV R0, #10          // R0 = 10 (decimal)
     MOV R1, #1           // R1 = 1 (decimal)

     ADD R0, R0, R1       // R0 = R0 + R1 = 11

stop:     B          stop // dead loop & program hangs here
```

Bitwise Operations in Assembly

- `R0 = 0xA2;   R1 = 0x34;`

AND

R0	10100010
R1	00110100
AND R2, R0, R1	00100000

OR

R0	10100010
R1	00110100
ORR R2, R0, R1	10110110

EXCLUSIVE OR

R0	10100010
R1	00110100
EOR R2, R0, R1	10010110

NOT

R0	10100010
MVN R2, R0	01011101

SHIFT RIGHT

R0	10100010
R0>>2	00101000
LSR R0, #2	

SHIFT LEFT

R0	10100010
R0<<2	10001000
LSL R0, #2	

Masking

- With computers, sometimes bits are used to mask bits. That is, they are utilized to turn bits **ON** or **OFF**.
- Typically, **OR** is used to turn items **ON** (or **set**) a bit and **AND** is utilized to turn items **OFF** (or **clear**) a bit.
- https://en.wikipedia.org/wiki/Mask_(computing)

Masking

- Masking example:
  - Suppose **A** holds an **unknown** binary number.
  - You want to turn **ON** all bits in **A**, but you don't want change the unknown value in **bit 3.**
  - This operation can be performed by using a bitwise **OR** operation with a **MASK** variable equal to **`11110111`**.

A → R0	????????
MASK → R1	11110111
ORR R0, R0, R1	
A → R0 = 1111?111	

Bit 3 does not change and it is still unknown.

- Masking example:
  - Now, using the final **A** value from the previous slide, suppose you want to turn **OFF** bit 3 in **A**.
  - This operation can be performed by using a bitwise **NOT** operation, followed by a bitwise **AND** operation, with a **MASK** variable equal to **`00001000`**.

`A → R0`	`1111?111`
`MASK → R1`	`00001000`

```
MVN R1, R1
AND R0, R0, R1
```

`A → R0 = 11110111`

Now, bit 3 is equal to **zero**.

Masking – Checking a bit

Uses bitwise **AND**.

a → R0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
MASK→ R1	0	0	1	0	0	0	0	0
AND R2, R0, R1	0	0	a_5	0	0	0	0	0

Masking – Setting a bit

Uses bitwise **OR**.

a → R0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
MASK→ R1	0	0	1	0	0	0	0	0
ORR R0, R0, R1	a_7	a_6	1	a_4	a_3	a_2	a_1	a_0

Masking – Clearing a bit

Uses bitwise **NOT**, followed by bitwise **AND**.

a → R0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
MASK→ R1	0	0	1	0	0	0	0	0
BIC R0, R1	a_7	a_6	0	a_4	a_3	a_2	a_1	a_0

The **BIC** instruction incorporates the **NOT** and **AND** in a single instruction.

Masking – Toggling a bit

Uses bitwise **EXCLUSIVE-OR (XOR)**.

a → R0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
MASK→ R1	0	0	**1**	0	0	0	0	0
EOR R0, R0, R1	a_7	a_6	NOT (a_5)	a_4	a_3	a_2	a_1	a_0

Lab Assignment

- Go to Canvas and answer all **FIVE** questions in the following assignment: **Lab 2 – Week 2**.
  - **Canvas will automatically grade your work! You don't need to show your work to the T.A.!**
  - The T.A. will help you with any problem you may face while answering the Canvas quiz.
  - All questions should be answered with the help of the **debugging environment** in the STM32CubeIDE.
    - Don't forget to use **Tutorial 4 – Debugging** to help you!
  - Create a project from scratch, a **main.s** file from scratch, and use the concepts you learned today.
  - **We are not interfacing any hardware with the development kit today. So, you don't need to use any include file.**

Next Class

- **Lab 2 – Pre-lab Quiz is due next class! Pre-lab Quiz is available on Canvas!**

- **Lab 2 – Interfacing the joystick with the LEDs:**
  - **Lab lecture:** Introduction to General Purpose Input and Output (GPIO).