COMP1203 - Lab 3 Prep

Dominik Tarnowski

1. What do characters `@` and `#` mean in ARM assembly language?

`@`

The character is used to determine a single line comment. Anything following the **@** character will be ignored by the compiler and treated as a comment.

`#`

The hash character is used to determine immediate operands. This means that instead of getting a value from a register, the compiler will use the number after the # instead.

For example: Executing `mov r0, #11` will move the immediate decimal value of 11 to the register r0.

However, since the arm v8 compiler, `#` was made optional for immediate value.

2. What does `add r0, r0, #5` do?

The **add operator** adds the 2nd and 3rd values and stores them in the register specified in the first argument. In this case, `r0 = r0 + 5` (r0 becomes the sum of itself and 5), as `#5` is treated as an immediate value.

3.

r0	r1	result
4	X	9 (0x9)
10	X	15 (0xf)
2	X	7 (0x7)
35	X	40 (0x28)

4. What is the difference if you use “adds r0,r0,r1”?

The `ADD` operand does **not** update flags. The `ADDS` operand on the other hand does update the **N,C,V,Z** flags based on the result.

Simply, the extra `S` character means that the **ASPR** (Application Processor Status Register) will be updated depending on result. * **N** - true if result is negative. * **C** - carry flag. Indicates data lost because it didn’t fit into 32 bits. * **V** - overflow flag. Indicates if an overflow occurred. * **Z** - true if result equals 0.

5. What does “subs r0,r0,r1” do?

- Subtracts a value in register `r1` from register `r0` and puts the result in `r0` register
- also updates **aspr** registers

6. What does “muls r0,r0,r1” do?

- Takes the value from register `r1` as well as from `r0`, multiplies them together, and the result is placed back in register `r0`
- also updates **aspr** registers

7. What do the “mov”, “cmp” and “ble” instructions do?

```
mov r0, #0
cmp r1, #1
ble .L5
mov r0, #42
.L5:
```

MOV

The `MOV` instruction can be used in two ways: * `MOV Rd, #v` - places immediate value `v` in `Rd` * or `MOV Rd, Rm` - places value in register `Rm` in `Rd`

In this case, mov is used to set the value in register 0 to `0`.

CMP

This instruction is used to compare value. For example:

```
CMP r0, r1
CMP r0, #v
```

Would compare the value in register 0 with the value of register 1 or immediate value `v`. It compares them by subtracting the second operand from the first and setting the condition flags. Works in the same way as `SUBS`, but it does not store the result in a register.

In this case, we are taking away `1` from the value of `r1` which happens to be `0`.

`BLE`

The leading `B` indicates a **branch**. LE stands for less than or equal to. So, given a comparison branch, `BLE` can be used to check if the result is less than or equal to another value.

In this case, BLE is used to branch to `.L5` label if `value in r0 <= 1`.

8. What does the new function `x` do?

The new function is used to compare a number in `r1` to immediate value `1`. If `r1` is bigger than `1`, the value of `r0` is set to 42, otherwise it remains 0.

```
def x(r1):
    if r1 <= 1:
        return 0
    return 42
```

9. What do the `PUSH` and `POP` instructions do?

```
mov r0, sp
push {lr}
bl .L5
pop {lr}
sub r0,r0,r1
bx lr
.L5:
mov r1,sp
```

The `PUSH` and `POP` commands are used to store/load registers on the stack, based on the `SP` (Stack Pointer). The most common usage of `PUSH` and `POP`, same as in the case above, is to save values of multiple registers on the stack and be able to pop them back to their original values later, same as local variables / scopes work in higher level languages.

Generally, you want to keep the stack aligned on a 64 bit boundary so generally an even number of registers is pushed/popped.

10. What are the curly braces used for?

The curly braces are used to indicate the list of registers to be stored/loaded from the stack. Though, keep it in mind that the registers have to be in the same order, both when being stored and loaded.

11. What do the instructions `BL (label)` and `BX lr` do?

BL (label)

This instruction (unconditionally) causes a branch to `label`. However, if a condition is added at the end, the branch becomes conditional. This also copies the address of the next instruction into `LR` (link register).

This is usually used to call an assembly function, allowing it to come back to the next instruction later.

BX lr

This instruction causes a brand to the address in the `LR` register. It also exchanges the instruction set based on the value of `LR`: * if the least significant bit of `LR` equals to 1, it will treat the code as **Thumb**, otherwise, it will be treated as **ARM**.

This is usually used to return to the line after where a function was called.