Function Calls

The breakpoint set in the delay function shows there is only a single instruction "BL" executed when calling delay:

```
        0x7a: 0x2002          MOVS     R0, #2
        0x7c: 0x6020          STR      R0, [R4]
    delay();
        0x7e: 0xf7ff 0xffdf   BL       delay          ...
    *(GPIO_PORTF_AHB_DATA_BITS_R + 0x02) &= 0x00;
        0x82: 0x6820          LDR      R0, [R4]
        0x84: 0x2000          MOVS     R0, #0
        0x86: 0x6020          STR      R0, [R4]
    delay();
        0x88: 0xf7ff 0xffda   BL       delay          ...
    *(GPIO_PORTF_AHB_DATA_BITS_R + 0x08) = G_LED;
        0x8c: 0x4c0b          LDR.N    R4, ??DataTable1_6...
        0x8e: 0x2008          MOVS     R0, #8
        0x90: 0x6020          STR      R0, [R4]
    delay();
        0x92: 0xf7ff 0xffd5   BL       delay          ...
```

4 byte instruction

BL is a branch instruction and thus alters the program counter However, it also saves the address of the next instruction in the link register (LR)

This allows the code to remember which address to return to after the code ends. However the address saved is not an even address rather it is the address of the next instruction + 1 (odd).

```
APSR    0x2000'0000
IPSR    0x0000'0000
EPSR    0x0100'0000
PC      0x0000'0040
SP      0x2000'0ff0
LR      0x0000'0083
PRIMASK 0x0000'0000
```

```
        0x7c: 0x6020          STR      R0, [R4]
    delay();
        0x7e: 0xf7ff 0xffdf   BL       delay
    *(GPIO_PORTF_AHB_DATA_BITS_R + 0x02) &= 0x00;
        0x82: 0x6820          LDR      R0, [R4]
        0x84: 0x2000          MOVS     R0, #0
        0x86: 0x6020          STR      R0, [R4]
```

+1

Here LR's address is 0x0000 0083 instead of 0x0000 0082

The stack The stack is a part of RAM that can grow or shrink. The stack has a top and bottom address and in ARM the top address is contained in the SP register

```
void delay(void)
{
                delay:
        0x40: 0xb081          SUB      SP, SP, #0x4
  int volatile count = 0;
        0x42: 0x2000          MOVS     R0, #0
        0x44: 0x9000          STR      R0, [SP]
        0x46: 0xe002          B.N      ??delay_0
```

stack space is incremented as 4 is subtracted from it's base address of 0x2000 0fec

```
EPSR    0x0100'0000
PC      0x0000'0042
SP      0x2000'0fec
LR      0x0000'008f
```

Top of stack address

```
0x2000'0fd8   04894270
0x2000'0fdc   0387eb04
0x2000'0fe0   1111ea42
0x2000'0fe4   2c08f853
0x2000'0fe8   00000000
0x2000'0fec   00000018
0x2000'0ff0   00000000
0x2000'0ff4   0000010b
0x2000'0ff8   fef5eda5
0x2000'0ffc   fef5eda5
0x2000'1000   f0480801
```

The stack. The stack grows towards lower addresses and shortens toward higher addresses (for ARM only). Once stack is grown (by minusing 4 from address) to 0x2000fe8, the counter variable is then saved at top of stack (0x2000 0fec)

Setting breakpoint at end of delay() function, we see that the stack address is added to 4 to shrink back stack size once function is finished

```
}
        0x56: 0xb001          ADD         SP, SP, #0x4
        0x58: 0x4770          BX          LR
int main()
{
```

The "Bx" instruction stands for branch and exchange and sets the program counter to the specified address of the link register (LR). Here the address just after the delay func was executed is 0x8e (this was saved in LR as 0x0000008e+1). At the end of the function during Bx instruction the value saved in LR to be sent to PC is 0x0000008f.

```
delay();
    0x8a: 0xf7ff 0xffd9   BL          delay
*(GPIO_PORTF_AHB_DATA_BITS_R + 0x02) &= 0x00;
    0x8e: 0x6820          LDR         R0, [R4]
    0x90: 0x2000          MOVS        R0, #0
    0x92: 0x6020          STR         R0, [R4]
delay();
```

This address is saved in LR just before function call

```
PC          0x0000'0056
SP          0x2000'0fec
LR          0x0000'008f
PRIMASK     0x0000'0000
```

```
        0x54: 0xdbf8          BLT.N       ??delay_1
}
        0x56: 0xb001          ADD         SP, SP, #0x4
        0x58: 0x4770          BX          LR
```

If the least significant bit of the LR register is 1 then the processor switches to thumb instruction set, if 0 then the switch is to the thumb2 instruction set. This bit is not used in addressing.

The Bx instruction set then takes us back to the address just after the address where the delay function was called.

```
PC          0x0000'0056
SP          0x2000'0fec
LR          0x0000'008f
PRIMASK     0x0000'0000
```

```
        0x54: 0xdbf8          BLT.N       ??delay_1
}
        0x56: 0xb001          ADD         SP, SP, #0x4
        0x58: 0x4770          BX          LR
```

↓ set least significant bit of LR to 0 to get to proper address

```
delay();
    0x8a: 0xf7ff 0xffd9   BL          delay
*(GPIO_PORTF_AHB_DATA_BITS_R + 0x02) &= 0x00;
    0x8e: 0x6820          LDR         R0, [R4]
    0x90: 0x2000          MOVS        R0, #0
    0x92: 0x6020          STR         R0, [R4]
delay();
```

Return addressing for functions and Link register :

Values loaded to the PC are always odd. This is arranged by the hardware in the CPU and by the compiler. For example, the instruction BL (for calling functions) loads the LR register with the address of the next instruction after BL (to return from the function). This value in LR will be odd, because the CPU will set the bit-0 in LR to 1, so that when the LR will be loaded back to PC (to return from the function), the PC will be also odd.