Microprocessor and Assembly Language Lab 03

Kazi Shadman Sakib, FH-97

February 10, 2022

1 Task i: Peripheral Clock Configuration

The RCC peripheral is used to control the internal peripherals, as well as the reset signals and clock distribution. Over the last half-decade, dramatically lowering current draw has been a goal for most microcontroller manufacturers. One of the techniques used to achieve this is to switch off on-chip peripherals by removing access to their master clocks. On the STM32 devices, these clocks are known as the hardware and peripheral clocks and are controlled by the RCC (Reset and Clock Control) group of registers. Since there are more than 32 on chip peripherals, there are actually two registers used to switch on a clock: RCC_AHB1 ENR and RCC_AHB2ENR for the Hardware clock, APB for the Peripheral clock. The clock is controlled by set/reset registers, so to turn a system on you set a bit in the ENR register, and to turn that same peripheral off you set the bit in the corresponding RCC_AHBxRSTR register.

1.1 Listing out all the clock register of Nucleo-STM32F446RE development board and their respective purpose:

RCC clock control register (RCC_CR): Purpose: Clock control register.

RCC PLL configuration register (RCC_PLLCFGR)): Purpose: PLL Configuration register.

RCC clock configuration register (RCC_CFGR): Purpose: Clock Configuration register.

RCC clock interrupt register (RCC_CIR): Purpose: Clock interrupt register.

RCC AHB1 peripheral reset register (RCC_AHB1RSTR): Purpose: AHB1 peripheral reset register.

RCC AHB2 peripheral reset register (RCC_AHB2RSTR): Purpose: AHB2 peripheral reset register.

RCC AHB3 peripheral reset register (RCC_AHB3RSTR): Purpose: AHB3 peripheral reset register.

RCC APB1 peripheral reset register (RCC_APB1RSTR): Purpose: APB1 peripheral reset register.

RCC APB2 peripheral reset register (RCC_APB2RSTR): Purpose: APB2 peripheral reset register.

RCC AHB1 peripheral clock enable register (RCC_AHB1ENR): Purpose: AHB1 peripheral enable register

RCC AHB2 peripheral clock enable register (RCC_AHB2ENR): Purpose: AHB2 peripheral enable register.

RCC AHB3 peripheral clock enable register (RCC_AHB3ENR): Purpose: AHB3 peripheral enable register.

RCC APB1 peripheral clock enable register (RCC_APB1ENR): Purpose: APB1 peripheral enable register

RCC APB2 peripheral clock enable register (RCC_APB2ENR): Purpose: APB2 peripheral enable register.

RCC AHB1 peripheral clock enable in low power mode register (RCC_AHB1LPENR): Purpose: AHB1 peripheral enable in low power register.

RCC AHB2 peripheral clock enable in low power mode register (RCC_AHB2LPENR): Purpose: AHB2 peripheral enable in low power register.

RCC AHB3 peripheral clock enable in low power mode register (RCC_AHB3LPENR): Purpose: AHB3 peripheral enable in low power register.

RCC APB1 peripheral clock enable in low power mode register (RCC_APB1LPENR): Purpose: APB1 peripheral enable in low power register.

RCC APB2 peripheral clock enabled in low power mode register (RCC_APB2LPENR): Purpose: APB2 peripheral enable in low power register.

RCC Backup domain control register (RCC_BDCR): Purpose: Backup Domain control register.

RCC clock control status register (RCC_CSR): Purpose: Clock control and status register.

RCC spread spectrum clock generation register (RCC_SSCGR): Purpose: Spread spectrum clock generation register.

RCC PLLI2S configuration register (RCC_PLLI2SCFGR): Purpose: PLLI2S configuration register.

RCC PLL configuration register (RCC_PLLSAICFGR): Purpose: PLLSAI configuration register.

RCC dedicated clock configuration register (RCC_DCKCFGR): Purpose: Dedicated clocks configuration register.

RCC clocks gated enable register (CKGATENR): Purpose: RCC clocks gated enable register.

RCC dedicated clocks configuration register 2 (DCKCFGR2): Purpose: RCC Dedicated Clocks Configuration Register 2.

1.2 Listing the memory address of the clock registers:

RCC clock control register (RCC_CR): Address offset: 0x00

RCC PLL configuration register (RCC_PLLCFGR)): Address offset: 0x04

RCC clock configuration register (RCC_CFGR): Address offset: 0x08

RCC clock interrupt register (RCC_CIR): Address offset: 0x0C

RCC AHB1 peripheral reset register (RCC_AHB1RSTR): Address offset: 0x10

RCC AHB2 peripheral reset register (RCC_AHB2RSTR): Address offset: 0x14

RCC AHB3 peripheral reset register (RCC_AHB3RSTR): Address offset: 0x18

RCC APB1 peripheral reset register (RCC_APB1RSTR): Address offset: 0x20

RCC APB2 peripheral reset register (RCC_APB2RSTR): Address offset: 0x24

RCC AHB1 peripheral clock enable register (RCC_AHB1ENR): Address offset: 0x30

RCC AHB2 peripheral clock enable register (RCC_AHB2ENR): Address offset: 0x34

RCC AHB3 peripheral clock enable register (RCC_AHB3ENR): Address offset: 0x38

RCC APB1 peripheral clock enable register (RCC_APB1ENR): Address offset: 0x40

RCC APB2 peripheral clock enable register (RCC_APB2ENR): Address offset: 0x44

RCC AHB1 peripheral clock enable in low power mode register (RCC_AHB1LPENR): Address offset: 0x50

RCC AHB2 peripheral clock enable in low power mode register (RCC_AHB2LPENR): Address offset: 0x54

RCC AHB3 peripheral clock enable in low power mode register (RCC_AHB3LPENR): Address offset: 0x58

RCC APB1 peripheral clock enable in low power mode register (RCC_APB1LPENR): Address offset: 0x60

RCC APB2 peripheral clock enabled in low power mode register (RCC_APB2LPENR): Address offset: 0x64

RCC Backup domain control register (RCC_BDCR): Address offset: 0x70

RCC clock control status register (RCC_CSR): Address offset: 0x74

RCC spread spectrum clock generation register (RCC_SSCGR): Address offset: 0x80

RCC PLLI2S configuration register (RCC_PLLI2SCFGR): Address offset: 0x84

RCC PLL configuration register (RCC_PLLSAICFGR): Address offset: 0x88

RCC dedicated clock configuration register (RCC_DCKCFGR): Address offset: 0x8C

RCC clocks gated enable register (CKGATENR): Address offset: 0x90

RCC dedicated clocks configuration register 2 (DCKCFGR2): Address offset: 0x94

2 Task ii: General Purpose Input/ Output Registers

Each General-Purpose I/O port has four 32-bit configuration registers (GPIOx _MODER, GPIOx_OTYPER, GPIOx_OSPEEDR and GPIOx_PUPDR), two 32-bit data registers (GPIOx_IDR and GPIOx_ODR), a 32-bit set/reset register (GPIOx_BSRR), a 32-bit locking register (GPIOx_LCKR) and two 32-bit alternate function selection register (GPIOx_AFRH and GPIOx_AFRL).

2.1 Listing out all the GPIO register of Nucleo-STM32F446RE development board and their respective purpose:

Mode Register (GPIOx_MODER): The GPIOx_MODER register is a 32-bit memory-mapped control register to configure upto 16 I/Os. The GPIOx_MODER register is used to select the I/O direction (input, output, AF, analog). The configuration length of each port mode register is 2 bits.

Bits 2y:2y+1:

These configuration bits are written by software to configure the I/O direction mode:

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

Very commonly used modes are input mode, general purpose output mode and alternate function mode. ADC or DAC uses Analog mode. Also, whenever a microcontroller undergoes reset all the pins of different GPIO ports will be by default in the input state.

Output Type Register (GPIOx_OTYPER): The GPIOx_OTYPER register is a 32-bit memory-mapped control register to configure upto 16 I/Os. The GPIOx_OTYPER register is used to select the output type-

a) Push-Pull mode: In push-pull mode "0" in the Output data register activates the N-MOS whereas a "1" in the Output data register activates the P-MOS. The data present on the I/O pin is sampled into the input data register over every AHB1 clock cycle. Read access to the input data register gets the I/O state. Read access to the output data register gets the last written value. To set GPIO output type register in push-pull mode, in GPIO_OTYPER register bit is set to "0".

b) Open Drain mode: In Open-drain configuration, PMOS doesn't exist, and two output possibilities are high or floating. A "0" in the Output data register activates the N-MOS and the I/O pin driven to the ground. Whereas "1" in the Output data register leaves the port in Hi-Z (the P-MOS is never activated), so I/O state is not defined. To solve this issue either activate an internal pull-up resistor or give an external pull-up resistor. So, once a pull-up resistor is activated, the I/O pin gets its state to VDD. To set GPIO output type register in open-drain mode, in GPIO_OTYPER register bit is set to "1".

Bits 31:16 are Reserved. Only Bits 15:0 are used.

The configuration bits are written by software to configure the output type of the I/O port.

0: Output Push-Pull (reset state)

1: Output Open-Drain

Speed Register (GPIOx_xOSPEEDR): The GPIOx_OSPEEDR register is a 32-bit memory-mapped control register to configure upto 16 I/Os. The GPIOx_OSPEED register is used to select the speed (the I/O speed pins are directly connected to the corresponding GPIOx_OSPEEDR register bits whatever the I/O direction). It is only applicable when the GPIO pin is in output mode. By using the

GPIO output speed register, one can configure the GPIO transitions from high to low and low to high, which means the slew rate of a pin can be controlled by GPIO output speed register.

Bits 2y:2y+1:

These bits are written by software to configure the I/O output speed.

00: Low speed

01: Medium speed

10: Fast speed

11: High speed

Pull-UP / Pull-Down Register (GPIOx_PUPDR): The GPIOx_PUPDR register is a 32-bit memory-mapped control register to configure upto 16 I/Os. The GPIOx_PUPDR register is used to select the Pull-Up / Pull-Down whatever the I/O direction. GPIO Pull-Up and Pull-Down registers are used to control the internal or external Pull-Up and Pull-Down registers. Bits 2y:2y+1:

These bits are written by software to configure the I/O pull-up or pull-down.

00: No pull-up, pull-down

01: Pull-Up

10: Pull-Down

11: Reserved

GPIO port Bit Set and Reset Register (GPIOx_BSRR): The Bit Set and Reset register (GPIOx_BSRR) is a 32-bit register which allows the application to set and reset each individual bit in the output data register (GPIOx_ODR). The purpose of the GPIOx_BSRR register is to allow atomic read/modify accesses to any of the GPIO registers. In this way, there is no risk of an IRQ occurring between the read and the modify access. The GPIOx_BSRR register provides a way of performing atomic bitwise handling. It is possible to modify one or more bits in a single atomic AHB1 write access. This register consists of two write-only bit masks, each 16-bit wide. The high half- 16 bits to 31 is the reset mask. Writing a 1 to any of these bits clear bit N-16 in the GPIO_ODR. The low half - bits 0 to 15 is the set mask. Writing a 1 to any of these bits sets the corresponding bit in the GPIO_ODR.

Bits 31:16 BRy: Port x reset bit y

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits return the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx bit

Bits 15:0 BSy: Post x set bit y

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits return the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit

Alternate Function (AF): Two registers are provided to select one out of the sixteen Alternate Function inputs/outputs available for each I/O. With these registers, one can connect an alternate function to some other pin as required by an application. This means that a number of possible peripheral functions are multiplexed on each GPIO using these two alternate registers-

a) GPIO Alternate Function Low Register (GPIOx_AFRL): GPIOx_AFRL contains the multiplexer settings for port bits 0 to 7.

Bits 31:0 Alternate function selection for port x bit y

These bits are written by software to configure alternate functions I/Os.

AFRLy selection:

0000: AF0 1000: AF8

0001: AF1 1001: AF9

0010: AF2 1010: AF10

0011: AF3 1011: AF11

0100: AF4 1100: AF12

0101: AF5 1101: AF13

0110: AF6 1110: AF14

0111: AF7 1111: AF15

b) GPIO Alternate Function High Register (GPIOx_AFRH): GPIOx_AFRH contains the multiplexer settings for port bits 8 to 15.

Bits 31:0 Alternate function selection for port x bit y

These bits are written by software to configure alternate functions I/Os. AFRHy selection:

0000: AF0 1000: AF8

0001: AF1 1001: AF9

0010: AF2 1010: AF10

0011: AF3 1011: AF11

0100: AF4 1100: AF12

0101: AF5 1101: AF13

0110: AF6 1110: AF14

0111: AF7 1111: AF15

The application can thus select any one of the possible functions for each I/O. The AF selection signal being common to the alternate function input and alternate function output, a single channel is selected for the alternate function input/output of one I/O.

Some more functions and roles of related Registers with GPIO Port and Pins:

Input Data Register (GPIOx_IDR): The input data register (GPIOx_IDR) is a 16-bit memory-mapped data register. The input data register (GPIOx_IDR) captures the data present on the I/O pin at every AHB1 clock cycle. The data input through the I/O are stored into the input data register (GPIOx_IDR), a read-only register. Here only Bits 15:0 are used and 31:16 bits are reserved.

Output Data Register (GPIOx_ODR): The output data register (GPIOx_ODR) is a 16-bit memory-mapped data register. When the pin is configured as output, the value written to the output data register (GPIOx_ODR) is output on the I/O pin. It is possible to use the output driver in push-pull mode or open-drain mode (only the N-MOS is activated when 0 is output). GPIOx_ODR stores the data to be output, it is read/write accessible. Here only 16-bits are used in this register. So, 16 to 31 are reserved, and the 0th-bit position will control the I/O pin 0.

Configuration Lock Register (GPIOx_LCKR): It is possible to freeze the GPIO control registers by applying a specific write sequence to the GPIOx_LCKR register. The frozen registers are GPIOx_MODER, GPIOx_OTYPER,GPIOx_OSPEEDR,GPIOx_PUPDR,GPIOx_AFRL, GPIOx_AFRH.

Analog Switch Control Register (GPIOx_ASCR): GPIOx_ASCR register helps to control analog interconnection between an I/O pin and ADC input.

2.2 Listing the memory address of the GPIO registers:

Mode Register (GPIOxMODER): Address offset: 0x00

Output Type Register (GPIOxOTYPER): Address offset: 0x04

Speed Register (GPIOxxOSPEEDR): Address offset: 0x08

Pull-UP / Pull-Down Register (GPIOxPUPDR): Address offset: 0x0C

GPIO port Bit Set and Reset Register (GPIOxBSRR): Address offset: 0x18

GPIO Alternate Function Low Register (GPIOxAFRL): Address offset: 0x20

GPIO Alternate Function High Register (GPIOxAFRH): Address offset: 0x24

Input Data Register (GPIOxIDR): Address offset: 0x10

Output Data Register (GPIOxODR): Address offset: 0x14

Configuration Lock Register (GPIOxLCKR): Address offset: 0x1C

3 Task iii

3.1 Detailed Code Explanation

Allocated two halfword of memory as 0x0032 and 0x0124 in V1 and V2 using the DSD instruction. Then in the main2 identifier Loaded the variable V1 to the register r1 and Loaded the variable V2 to the register r2 using the LDR instruction. AND Operation on r1 and r2 and storing result it in r3 using AND instruction. OR operation on r1 and r2 and storing result it in r4 using ORR instruction. Using the MVN instruction the NOR operation was done on r4 register and saved the result in r5 register. Using the MVN instruction the NAND operation was done on r3 and result was saved in r6 register. XOR operation was done on registers r1 and r2 and then result was stored in r7 register using EOR instruction. And lastly MVN instruction was used to do the XNOR operation on r7 and saved it to r8 register.

3.2 Detailed description of the instruction used to design the program.

LDR: ARM uses a load model for memory access which means that only load (LDR) instruction can access memory and load a value from the memory to the register.

AND:The AND instruction is used for supporting logical expressions by performing bitwise AND operation. The bitwise AND operation returns 1, if the matching bits from both the operands are 1, otherwise it returns 0.

ORR:The OR instruction is used for supporting logical expression by performing bitwise OR operation. The bitwise OR operator returns 1, if the matching bits from either or both operands are one. It returns 0, if both the bits are zero.

MVN:Operation The MVN instruction takes the value of Operand2, performs a bitwise logical NOT operation on the value, and places the result into Rd.

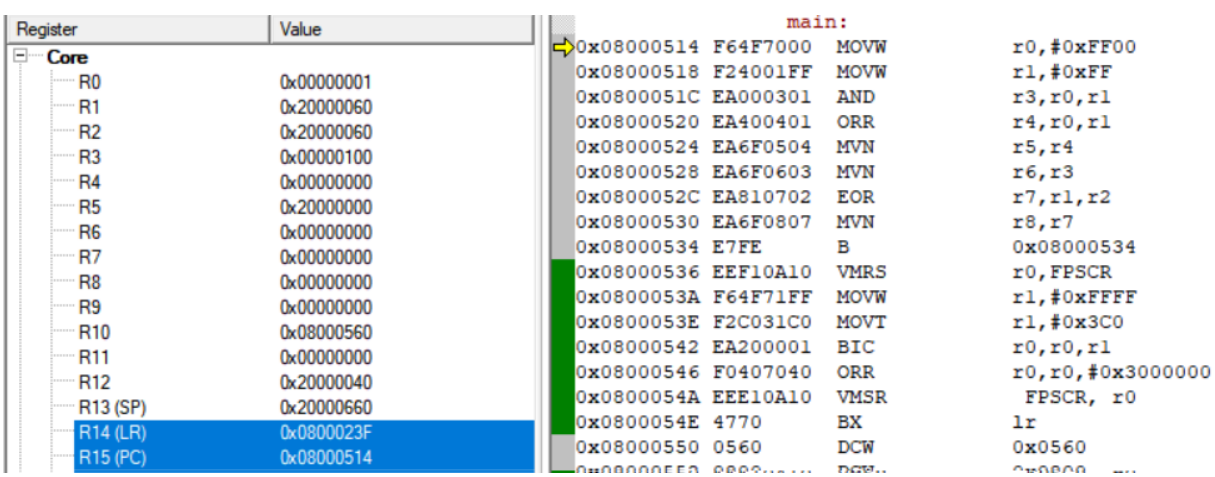
EOR:The EOR instruction performs bitwise Exclusive OR operations on the values in Rn and Operand2.

DCD:DCD is used to "reserve a 32 bit word"

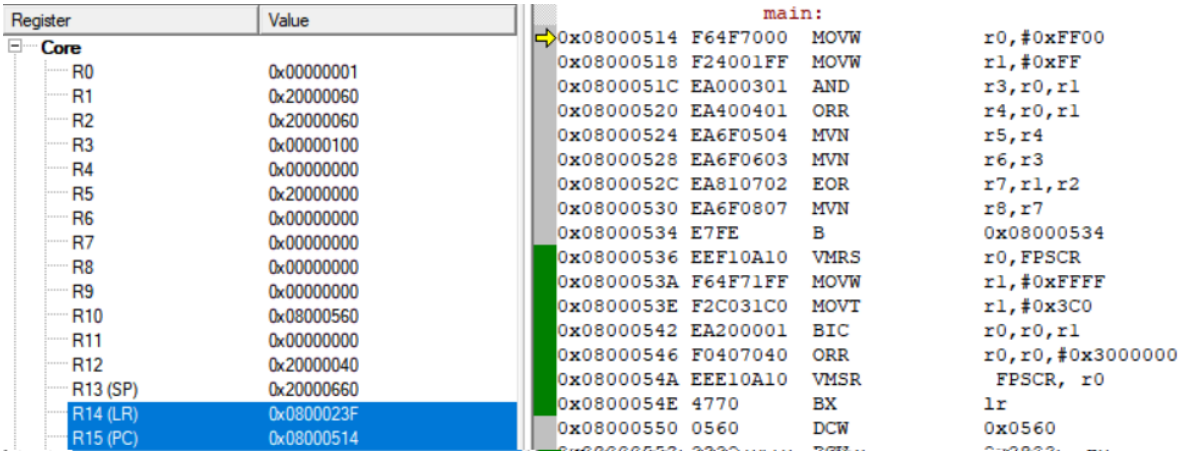
3.3 Screenshot of Debugger

3.3.1 After the Code has been Loaded

For 16 Bit



For 32 Bit

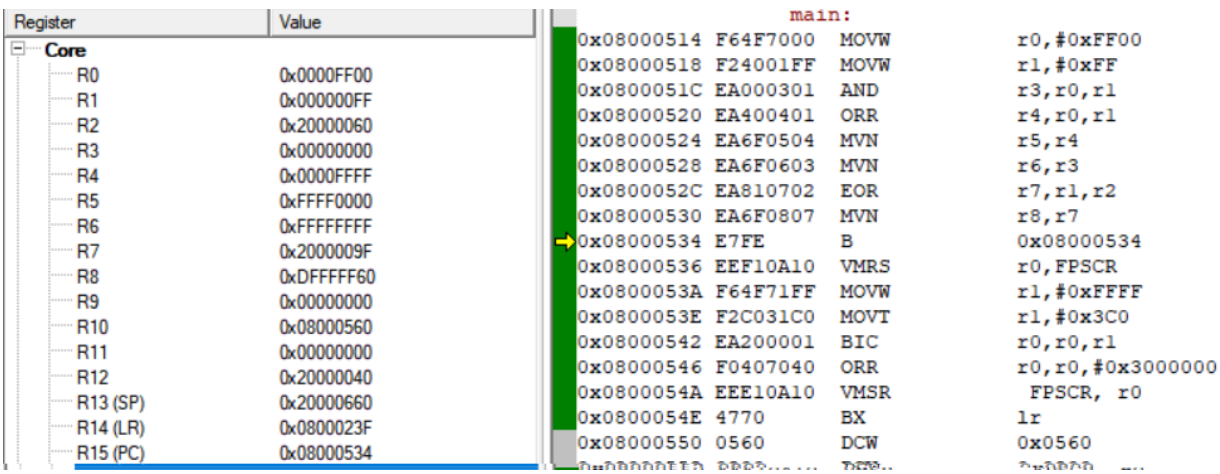


Register	Value
Core	
R0	0x00000001
R1	0x20000060
R2	0x20000060
R3	0x00000100
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000560
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000660
R14 (LR)	0x0800023F
R15 (PC)	0x08000514

```
                    main:
0x08000514 F64F7000  MOVW    r0,#0xFF00
0x08000518 F24001FF  MOVW    r1,#0xFF
0x0800051C EA000301  AND     r3,r0,r1
0x08000520 EA400401  ORR     r4,r0,r1
0x08000524 EA6F0504  MVN     r5,r4
0x08000528 EA6F0603  MVN     r6,r3
0x0800052C EA810702  EOR     r7,r1,r2
0x08000530 EA6F0807  MVN     r8,r7
0x08000534 E7FE      B       0x08000534
0x08000536 EEF10A10  VMRS    r0,FPSCR
0x0800053A F64F71FF  MOVW    r1,#0xFFFF
0x0800053E F2C031C0  MOVT    r1,#0x3C0
0x08000542 EA200001  BIC     r0,r0,r1
0x08000546 F0407040  ORR     r0,r0,#0x3000000
0x0800054A EEE10A10  VMSR     FPSCR, r0
0x0800054E 4770      BX      lr
0x08000550 0560      DCW     0x0560
```

3.3.2 After the Code has been Executed

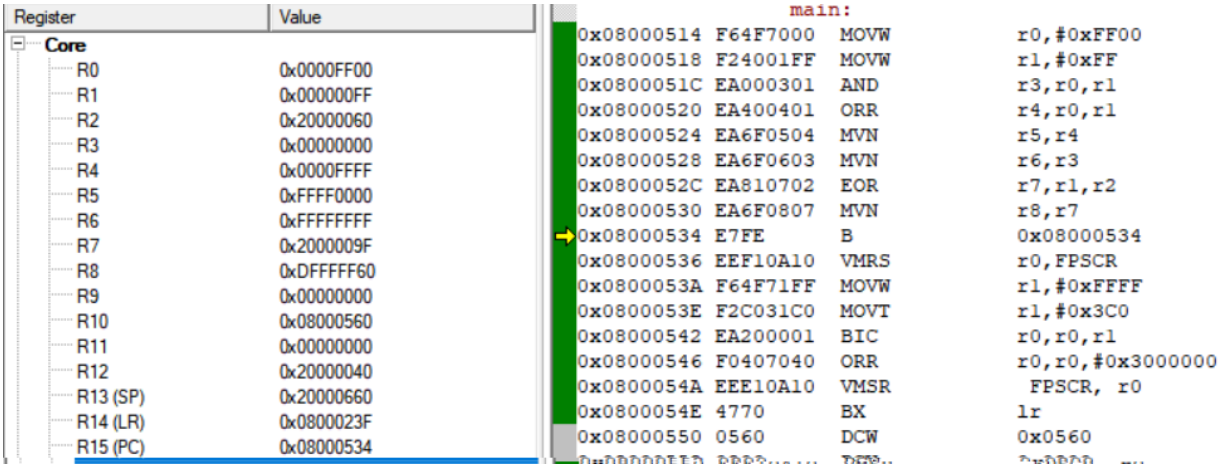
For 16 Bit



Register	Value
Core	
R0	0x0000FF00
R1	0x000000FF
R2	0x20000060
R3	0x00000000
R4	0x0000FFFF
R5	0xFFFF0000
R6	0xFFFFFFFF
R7	0x2000009F
R8	0xDFFFFF60
R9	0x00000000
R10	0x08000560
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000660
R14 (LR)	0x0800023F
R15 (PC)	0x08000534

```
                    main:
0x08000514 F64F7000  MOVW    r0,#0xFF00
0x08000518 F24001FF  MOVW    r1,#0xFF
0x0800051C EA000301  AND     r3,r0,r1
0x08000520 EA400401  ORR     r4,r0,r1
0x08000524 EA6F0504  MVN     r5,r4
0x08000528 EA6F0603  MVN     r6,r3
0x0800052C EA810702  EOR     r7,r1,r2
0x08000530 EA6F0807  MVN     r8,r7
0x08000534 E7FE      B       0x08000534
0x08000536 EEF10A10  VMRS    r0,FPSCR
0x0800053A F64F71FF  MOVW    r1,#0xFFFF
0x0800053E F2C031C0  MOVT    r1,#0x3C0
0x08000542 EA200001  BIC     r0,r0,r1
0x08000546 F0407040  ORR     r0,r0,#0x3000000
0x0800054A EEE10A10  VMSR     FPSCR, r0
0x0800054E 4770      BX      lr
0x08000550 0560      DCW     0x0560
```

For 32 Bit



Register	Value
Core	
R0	0x0000FF00
R1	0x000000FF
R2	0x20000060
R3	0x00000000
R4	0x0000FFFF
R5	0xFFFF0000
R6	0xFFFFFFFF
R7	0x2000009F
R8	0xDFFFFF60
R9	0x00000000
R10	0x08000560
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000660
R14 (LR)	0x0800023F
R15 (PC)	0x08000534

```
                    main:
0x08000514 F64F7000  MOVW    r0,#0xFF00
0x08000518 F24001FF  MOVW    r1,#0xFF
0x0800051C EA000301  AND     r3,r0,r1
0x08000520 EA400401  ORR     r4,r0,r1
0x08000524 EA6F0504  MVN     r5,r4
0x08000528 EA6F0603  MVN     r6,r3
0x0800052C EA810702  EOR     r7,r1,r2
0x08000530 EA6F0807  MVN     r8,r7
0x08000534 E7FE      B       0x08000534
0x08000536 EEF10A10  VMRS    r0,FPSCR
0x0800053A F64F71FF  MOVW    r1,#0xFFFF
0x0800053E F2C031C0  MOVT    r1,#0x3C0
0x08000542 EA200001  BIC     r0,r0,r1
0x08000546 F0407040  ORR     r0,r0,#0x3000000
0x0800054A EEE10A10  VMSR     FPSCR, r0
0x0800054E 4770      BX      lr
0x08000550 0560      DCW     0x0560
```

4 Task iv

4.1 Detailed Code Explanation

Allocated two halfword of memory as 0x0032 and 0x0124 in V1 and V2 variables. Used the LDR instruction to load the variable V1 in r1 register. LSR instruction was done to use logical shift right operation on r1 by 2 places bits and stored the result in r2. ASR instruction was done to provide the signed value of the contents of a register divided by a power of two and stored it in r3 register. Lastly, used the LSL instruction to logical shift left by 2 places.

4.2 Detailed description of the instruction used to design the program.

DCD:DCD is used to "reserve a 32 bit word"

LDR: ARM uses a load model for memory access which means that only load (LDR) instruction can access memory and load a value from the memory to the register.

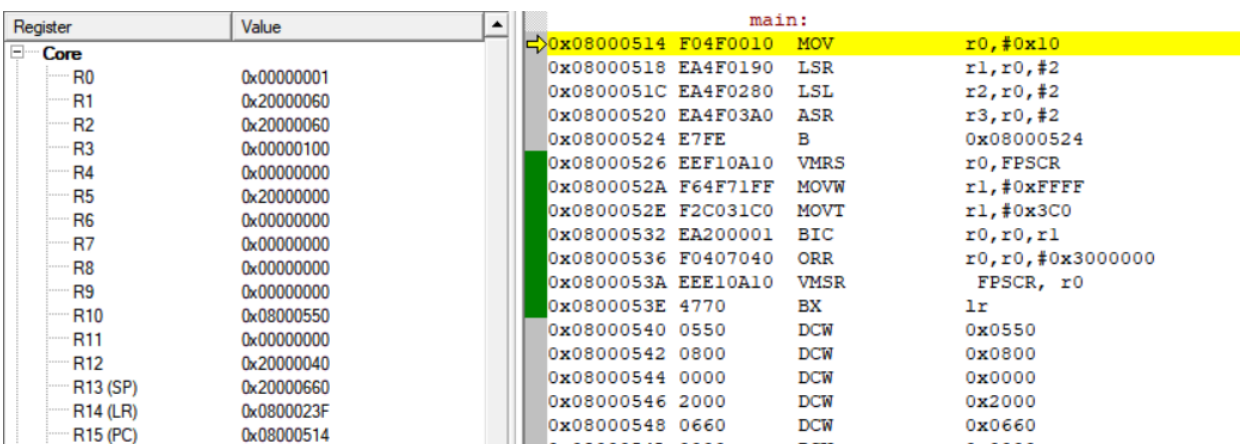
LSR: LSR is a logical shift right by 0 to 32 places.

ASR:ASR provides the signed value of the contents of a register divided by a power of two. It copies the sign bit into vacated bit positions on the left. Thumb instructions must not use PC or SP.

LSL: LSL is a logical shift left by 0 to 31 places.

4.3 Screenshot of Debugger

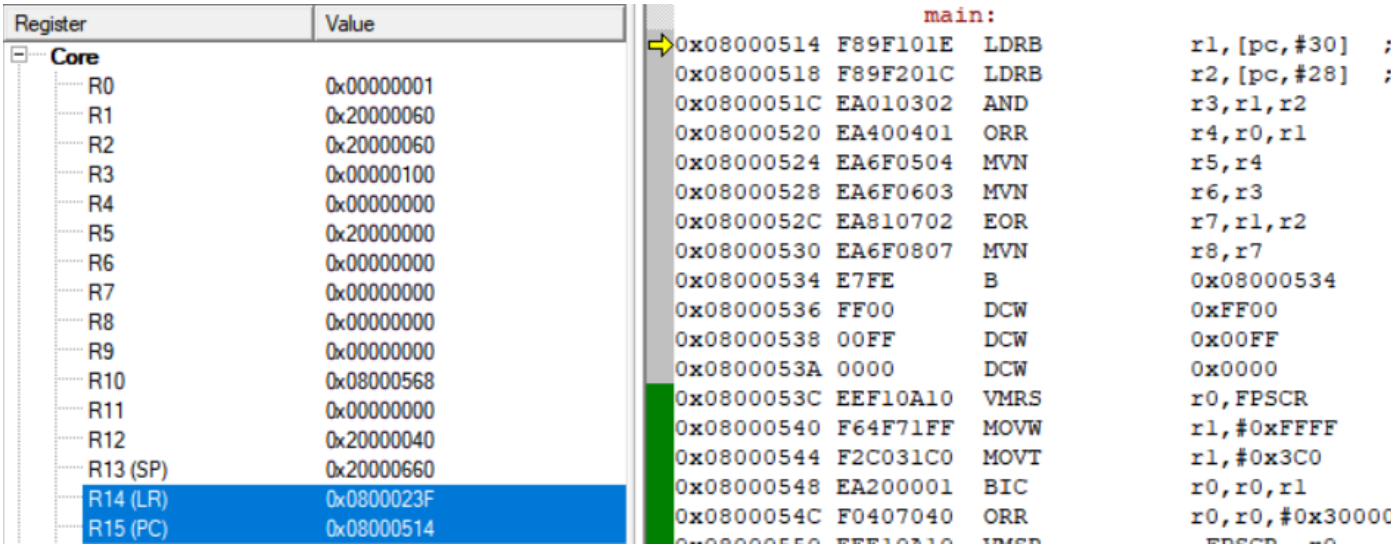
4.3.1 After the Code has been Loaded



Register	Value
Core	
R0	0x00000001
R1	0x20000060
R2	0x20000060
R3	0x00000100
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000550
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000660
R14 (LR)	0x0800023F
R15 (PC)	0x08000514

```
                    main:
0x08000514 F04F0010  MOV     r0,#0x10
0x08000518 EA4F0190  LSR     r1,r0,#2
0x0800051C EA4F0280  LSL     r2,r0,#2
0x08000520 EA4F03A0  ASR     r3,r0,#2
0x08000524 E7FE      B       0x08000524
0x08000526 EEF10A10  VMRS    r0,FPSCR
0x0800052A F64F71FF  MOVW    r1,#0xFFFF
0x0800052E F2C031C0  MOVT    r1,#0x3C0
0x08000532 EA200001  BIC     r0,r0,r1
0x08000536 F0407040  ORR     r0,r0,#0x3000000
0x0800053A EEE10A10  VMSR     FPSCR, r0
0x0800053E 4770      BX      lr
0x08000540 0550      DCW     0x0550
0x08000542 0800      DCW     0x0800
0x08000544 0000      DCW     0x0000
0x08000546 2000      DCW     0x2000
0x08000548 0660      DCW     0x0660
```

4.3.2 After the Code has been Executed



Register	Value
Core	
R0	0x00000001
R1	0x20000060
R2	0x20000060
R3	0x00000100
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000568
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000660
R14 (LR)	0x0800023F
R15 (PC)	0x08000514

```
                    main:
0x08000514 F89F101E  LDRB        r1,[pc,#30]
0x08000518 F89F201C  LDRB        r2,[pc,#28]
0x0800051C EA010302  AND         r3,r1,r2
0x08000520 EA400401  ORR         r4,r0,r1
0x08000524 EA6F0504  MVN         r5,r4
0x08000528 EA6F0603  MVN         r6,r3
0x0800052C EA810702  EOR         r7,r1,r2
0x08000530 EA6F0807  MVN         r8,r7
0x08000534 E7FE      B           0x08000534
0x08000536 FF00      DCW         0xFF00
0x08000538 00FF      DCW         0x00FF
0x0800053A 0000      DCW         0x0000
0x0800053C EEF10A10  VMRS        r0,FPSCR
0x08000540 F64F71FF  MOVW        r1,#0xFFFF
0x08000544 F2C031C0  MOVT        r1,#0x3C0
0x08000548 EA200001  BIC         r0,r0,r1
0x0800054C F0407040  ORR         r0,r0,#0x30000
```

5 Task v

5.1 Detailed Code Explanation

First, we move the value 0xBBBABABABA to register r1 and 0xEFEFEFEFEF to register r2.

Register r2 contains the addition result of values stored in register r0 and r1 and updates a flag register if there is an overflow using the ADDS command. If overflow occurs, we branch to sum_overflow

Register r4 contains the subtraction result of values stored in register r0 and r1 and updates a flag register if there is an overflow using the SUBS command. If overflow occurs, we branch to sub_overflow.

Register r6 contains the multiplication result of values stored in r0 and r1 and sets the value of register r7 to 1 if overflow occurs and branch to label mult_overflow.

5.2 Detailed description of the instruction used to design the program.

DCD:DCD is used to "reserve a 32 bit word"

LDR: ARM uses a load model for memory access which means that only load (LDR) instruction can access memory and load a value from the memory to the register.

MOVT:. In the Thumb instruction set MOVT, instruction moves 16-bit immediate value to top halfword (bits 16 to 31) and the bottom halfword remains unaltered.

ADDS: Addition operation o the registers

SUBS: Subtraction operation o the registers

MUL:This instruction is for multiplying binary data. The MUL (Multiply) instruction handles unsigned data

ADC:The ADC (Add with Carry) instruction adds the values in Rn and Operand2 , together with the carry flag. You can use ADC to synthesize multiword arithmetic.

BCC:BCC (short for "Branch if Carry is Clear") is the mnemonic for a machine language instruction

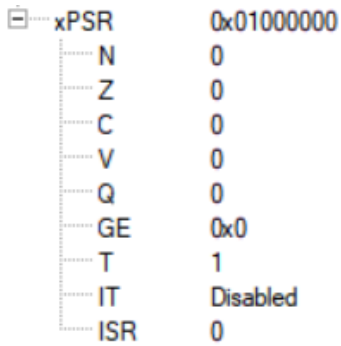
which branches, or "jumps", to the address specified if, and only if the carry flag is clear.

B:The B instruction causes a branch to label.

UMULL:The UMULL instruction interprets the values from Rn and Rm as unsigned integers. It multiplies these integers and places the least significant 32 bits of the result in Rd

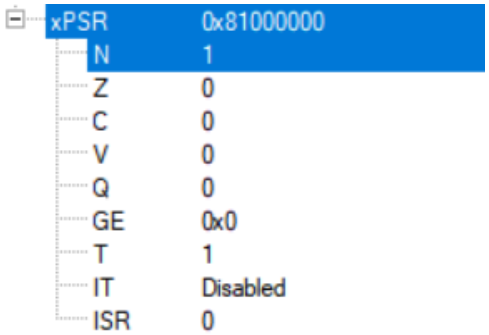
5.3 Status of the status registers after the operation

Status Initial:



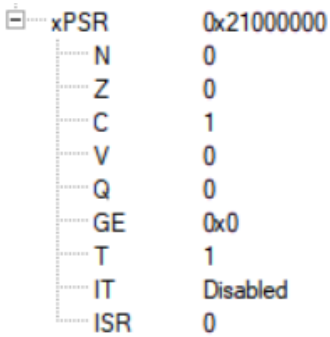
xPSR	0x01000000
N	0
Z	0
C	0
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0

Status Mid:



xPSR	0x81000000
N	1
Z	0
C	0
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0

Status Final:

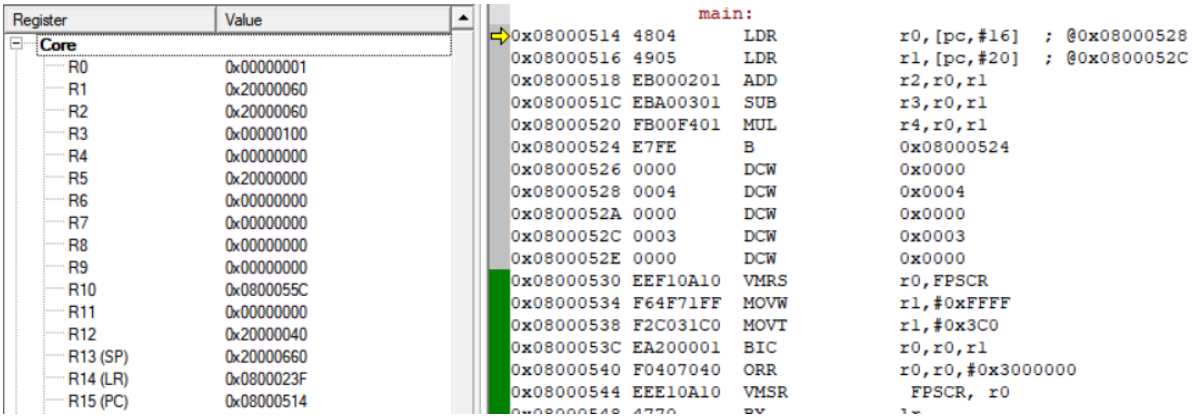


xPSR	0x21000000
N	0
Z	0
C	1
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0

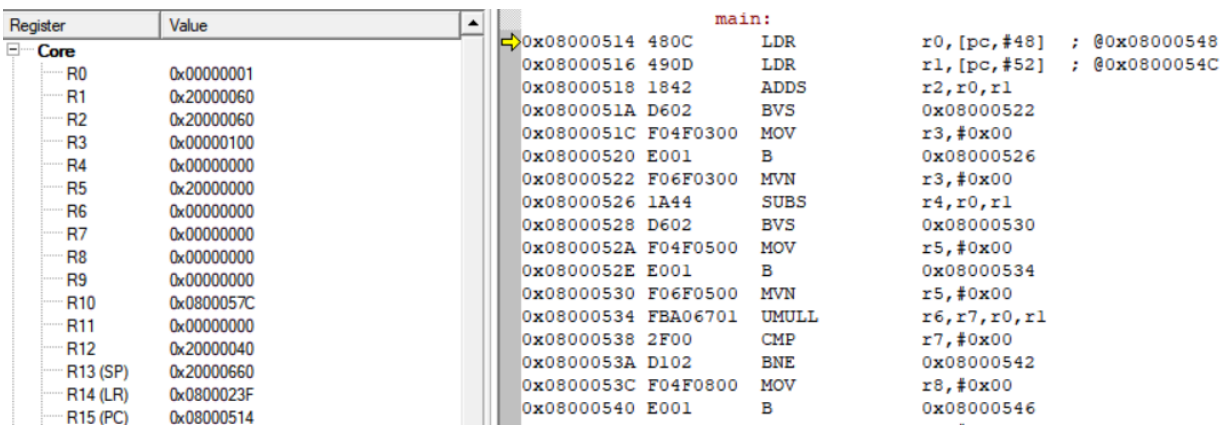
5.4 Screenshot of Debugger

5.4.1 After the Code has been Loaded

For Arithmetics with Restriction

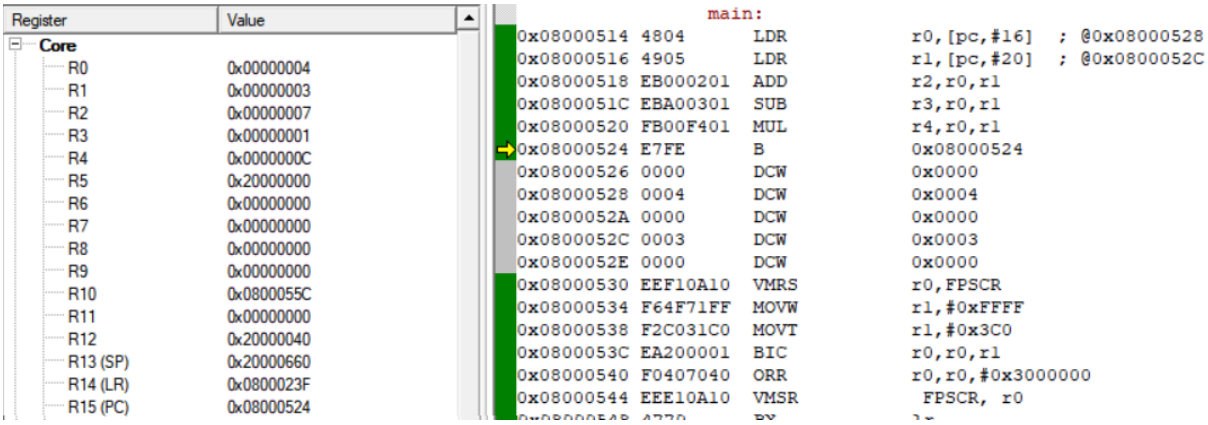


For Arithmetics with Overflow handle

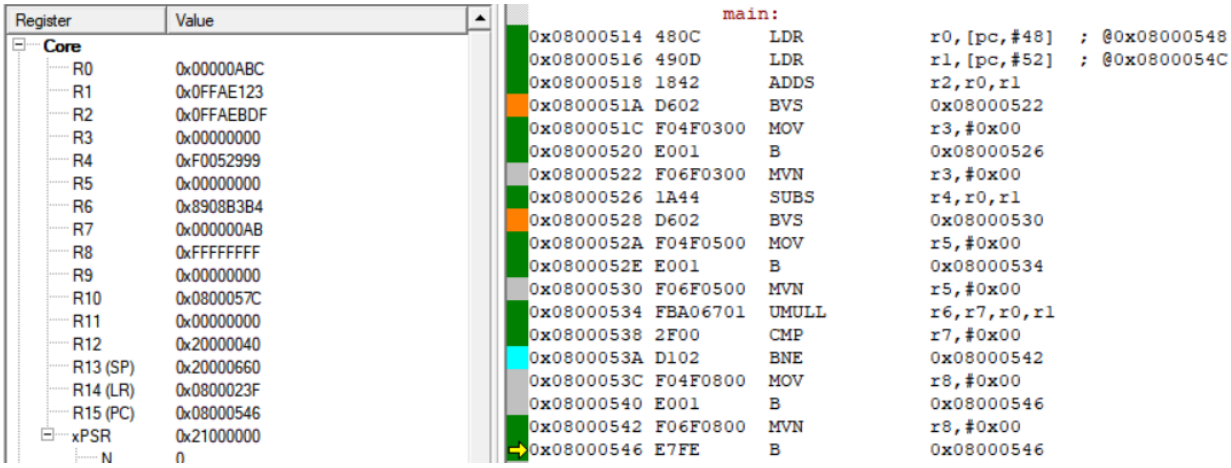


5.4.2 After the Code has been Executed

For Arithmetics with Restriction



For Arithmetics with Overflow handle



Register	Value
Core	
R0	0x00000ABC
R1	0x0FFAE123
R2	0x0FFAEBDF
R3	0x00000000
R4	0xF0052999
R5	0x00000000
R6	0x8908B3B4
R7	0x000000AB
R8	0xFFFFFFFF
R9	0x00000000
R10	0x0800057C
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000660
R14 (LR)	0x0800023F
R15 (PC)	0x08000546
xPSR	0x21000000
N	0

```
                    main:
0x08000514 480C      LDR        r0,[pc,#48]  ; @0x08000548
0x08000516 490D      LDR        r1,[pc,#52]  ; @0x0800054C
0x08000518 1842      ADDS       r2,r0,r1
0x0800051A D602      BVS        0x08000522
0x0800051C F04F0300  MOV        r3,#0x00
0x08000520 E001      B          0x08000526
0x08000522 F06F0300  MVN        r3,#0x00
0x08000526 1A44      SUBS       r4,r0,r1
0x08000528 D602      BVS        0x08000530
0x0800052A F04F0500  MOV        r5,#0x00
0x0800052E E001      B          0x08000534
0x08000530 F06F0500  MVN        r5,#0x00
0x08000534 FBA06701  UMULL      r6,r7,r0,r1
0x08000538 2F00      CMP        r7,#0x00
0x0800053A D102      BNE        0x08000542
0x0800053C F04F0800  MOV        r8,#0x00
0x08000540 E001      B          0x08000546
0x08000542 F06F0800  MVN        r8,#0x00
0x08000546 E7FE      B          0x08000546
```