```
                .section .vectors, "ax"
        B _start // reset vector
        B SERVICE_UND // undefined instruction vector
        B SERVICE_SVC // software inteRESET_LOCATIONSupt vector
        B SERVICE_ABT_INST // aborted prefetch vector
        B SERVICE_ABT_DATA // aborted data vector
        .WORD 0 // unused vector
        B SERVICE_IRQ // IRQ interrupt vector
        B SERVICE_FIQ // FIQ interrupt vector
.text
.global _start
_start:
        /* Set up stack pointers for IRQ AND SVC processor modes */
        MOV R1, #0b11010010 // interrupts masked, MODE = IRQ
        MSR CPSR_c, R1 // change to IRQ mode
        LDR SP, =0xFFFFFFFF - 3 // set IRQ stack to A9 onchip memory

        /* Change to SVC (supervisor) mode with interrupts disaBLed */
        MOV R1, #0b11010011 // interrupts masked, MODE = SVC
        MSR CPSR, R1 // change to supervisor mode
        LDR SP, =0x3FFFFFFF - 3 // set SVC stack to top of DDR3 memory

        BL CONFIG_GIC // configure the ARM GIC
        BL CONFIG_KEYS // configure the pushbutton KEYs

        BL CONFIG_PRIVATE_TIMER // configure the PRIVATE timer

        // enaBLe IRQ interrupts in the processor
        MOV R0, #0b01010011 // IRQ unmasked, MODE = SVC
        MSR CPSR_c, R0

// idle until the game_status=2
IDLE:
        LDR R0,=GAME_STATUS
        LDR R0,[R0]
        CMP R0,#2
        BLEQ main
        B IDLE

main:
        PUSH {R0-R12,LR}
        LDR R2,=BALANCE
        LDR R0,=BET_AMOUNT
        //BALANCE = BALANCE - BET AMOUNT
        LDR R3,[R0] //BET AMOUNT
        LDR R1,[R2] //BALANCE
        CMP R1,#0
        BLE GAME_OVER
        SUB R0,R1,R3
        CMP R0,#0
        MOVLE R0,R1
        LDRLE R1,=BET_AMOUNT
        STRLE R0,[R1]
        MOVLE R0,#0
        STR R0,[R2]

IS_MINE:
        LDR R4,=MINE_LOCATIONS_BIT
        LDR R5,[R4] //MINE_LOCATION
        LDR R6,=CURRENT_LED_STATUS
        LDR R7,[R6] //CURRENT_LED_STATUS
```

```
		LDR R12,=WIN  //WIN = Ob1111111111
		MOV R11,#0
		EOR R11,R5,R12
		AND R11,R11,R12
		CMP R11,R7 //CHECKS FOR WIN CONDITION
		BEQ WIN_SEQUENCE
		LDR R0,=GAME_STATUS
		LDR R1, [R0]
		CMP R1,#2
		BNE CHECK_OUT
		LDR R0,=0xFF200000
		LDR R2,=0xFF200040
		LDR R4,=MINE_LOCATIONS_BIT
		LDR R6,=CURRENT_LED_STATUS
		LDR R1,[R0] //LED
		LDR R3,[R2] //SWICH
		LDR R5,[R4] //MINE_LOCATION
		LDR R7,[R6] //CURRENT_LED_STATUS
		ORR R8,R7,R3
		CMP R7,R8 //CHECK IF ANY NEW SWICH TURNED
		STRNE R8,[R6] //STORE NEW CURRENT LED STATUS
		STRNE R8,[R0] //SHOW CLEARED LOCATIONS ON THE LEDS
		ANDNE R8,R8,R5
		BEQ IS_MINE
		CMP R8,#0 //CHECK IF NEW INPUT IS MINE
		BNE LOST_SEQUENCE
		BLEQ INCREASE_BET_AMOUNT
		B IS_MINE

//RESET GAME VARIABLES AND UPDATE NEW BALANCE
CHECK_OUT:
		MOV R1,#0
		LDR R2,=NUMBER_OF_MINES
		STR R1,[R2]
		LDR R0,=0xFF200000
		STR R1,[R0]
		LDR R0,=CURRENT_LED_STATUS
		STR R1,[R0]
		LDR R2,=MINE_LOCATIONS_BIT
		STR R1,[R2]
		LDR R2,=GAME_STATUS
		LDR R2,=0xFF200000
		STR R1,[R2]
		STR R1,[R2]
		LDR R2,=PREVIOUS_BET_AMOUNT
		STR R1,[R2]
		LDR R2,=BALANCE
		LDR R0,=BET_AMOUNT
		//UPDATE BALANCE
		LDR R3,[R0] //BET AMOUNT
		LDR R1,[R2] //BALANCE
		ADD R1,R1,R3
		STR R1,[R2]
		MOV R1,#50
		STR R1,[R0]
		MOV R1,#10//reset locations
RESET_LOCATIONS:
		MOV R0,#0
		LDR R2,=LOCATIONS
		STR R0,[R2,R1,LSL #2]
		SUBS R1,#1
```

```
        BGE RESET_LOCATIONS
//GOES BACK TO IDLE TO GET NEW DATA FOR THE NEXT ROUND
        POP {R0-R12,PC}

//WHEN PLAYER HAS NO BALANCE TO PLAY, OUTPUT -LOSE- ON SEVENT SEGMENT
GAME_OVER:
        LDR R0,=OSE
        LDR R1,=L
        LDR R2,=0xFF200020
        LDR R3,=0xFF200030
        STR R0,[R2]
        STR R1,[R3]
        B GAME_OVER

//If a player clears a safe space increase the amount of the bet
INCREASE_BET_AMOUNT:
        PUSH {R0-R12,LR}
        LDR R0,=BET_AMOUNT
        LDR R1,=NUMBER_OF_MINES
        LDR R2,[R0]//BET_AMOUNT
        LDR R3,[R1]//NUMBER_OF_MINES
        LDR R4,=MULTIPLIER
        LDR R5,[R4,R3,LSL #2] //GET MULTIPLIER ACCORDING TO MINE NUMBER
        MUL R2,R2,R5
        MOV R6,#0 //DIVIDER COUNTER
DIVIDER_LOOP:
        CMP R2,#10
        SUBGE R2,R2,#10
        ADDGE R6,R6,#1
        BGE DIVIDER_LOOP
        STR R6,[R0]
        BL SEVEN_SEGMENT_SLOW_INCREASE
        LDR R0,=PREVIOUS_BET_AMOUNT
        STR R6,[R0] //TO START COUNTING FROM LAST BET AMOUNT
        POP {R0-R12,PC}

//PLAYER LOST A ROUND
LOST_SEQUENCE:
        LDR R0,=BET_AMOUNT
        MOV R1, #0
        STR R1,[R0]
        PUSH {R0-R12}
        LDR R3, =BET_AMOUNT
        LDR R0, [R3] //BET AMOUNT
        BL SEVEN_SEGMENT_DISPLAYER
        POP {R0-R12}
//FLIPPING THE MINES SO THAT USER CAN SEE
//WHERE ARE THE MINES PLACED, AFTER LOSING THE ROUND
LOST_SEQUENCE2:
//BOTH WIN AND LOST SEQUENCES USES THE SAME FLIPPING METHOD
WIN_SEQUENCE:
        LDR R0,=0xFF200000
        LDR R2,=MINE_LOCATIONS_BIT
        LDR R4,=CURRENT_LED_STATUS
        LDR R1,[R0] //LED
        LDR R3,[R2] //MINE_LOCATION
        LDR R5,[R4] //CURRENT_LED_STATUS
        ORR R6,R3,R5
        EOR R7,R6,R3
        STR R7,[R0]
DELAY:
```

```
        LDR R7, =2000000 // delay counter
DELAY_LOOP:
        SUBS R7, R7, #1
        BNE DELAY_LOOP
        STR R6,[R0]
DELAY1:
        LDR R7, =2000000 // delay counter
DELAY_LOOP1:
        SUBS R7, R7, #1
        BNE DELAY_LOOP1
        LDR R0,=GAME_STATUS
        LDR R1, [R0]
        CMP R1,#2
        BNE CHECK_OUT
        B LOST_SEQUENCE2

.equ KEY_BASE, 0xFF200050
.equ LED_BASE, 0xFF200000
.equ SWITCH_BASE, 0xFF200040
.equ WIN, 0x3FF
.equ OSE, 0x3F6D7940
.equ L, 0x4038
.equ ADDR_7SEG1, 0xFF200020
CURRENT_LED_STATUS: .WORD 0
NUMBER_OF_MINES: .WORD 0
GAME_STATUS: .WORD 0
MINE_LOCATIONS_BIT: .WORD 0
LOCATIONS: .WORD 0,0,0,0,0,0,0,0,0,0
MULTIPLIER: .WORD 1,11,13,15,19,22,30,45
BET_AMOUNT: .WORD 50
PREVIOUS_BET_AMOUNT:.WORD 0
BALANCE: .WORD 300
HEXTABLE:.WORD 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F
DIVISORS:.WORD 0x3E8,0x64,0xA,0x1

/*****************************************************************
* Pushbutton - Interrupt Service Routine
*
* This routine checks which KEY has been pressed. It writes to HEX0
*****************************************************************/
KEY_ISR:
        LDR R0, =KEY_BASE // base address of pushbutton KEY port
        LDR R1, [R0, #0xC] // read edge capture register
        MOV R2, #0xF
        STR R2, [R0, #0xC] // clear the interrupt

//GET THE NUMBER OF MINES FOR THE PLAY
CHECK_KEY0:
        MOV R3, #0x1
        ANDS R3, R3, R1 // CHECK FOR KEY0
        BEQ CHECK_KEY1
        LDR R0,=GAME_STATUS
        LDR R1, [R0]
        CMP R1,#0
        BNE SHOW_MINE_NUM
        //INCREASE MINE NUMBER IF GAME_STATUS=0
        LDR R0, =NUMBER_OF_MINES
        LDR R1, [R0]
        CMP R1,#7 //MINE RANGE IS BETWEEN 1-7
        MOVEQ R1,#0
        ADD R1,R1,#1
```

```
        STR R1, [R0]
SHOW_MINE_NUM:
        PUSH {R0-R12,LR}
        LDR R3, =NUMBER_OF_MINES
        LDR R0, [R3]//NUMBER_OF_MINES
        BL SEVEN_SEGMENT_DISPLAYER
        POP {R0-R12,LR}
        B END_KEY_ISR

//DECREASE THE BET AMOUNT IF GAME_STATUS=1
CHECK_KEY1:
        MOV R3, #0x2
        ANDS R3, R3, R1 // CHECK FOR KEY1
        BEQ CHECK_KEY2
        LDR R0,=GAME_STATUS
        LDR R1, [R0]
        CMP R1,#1
        BNE SHOW_BET
        LDR R0, =BET_AMOUNT
        LDR R1, [R0]
        CMP R1,#10 //MINIMUM BET AMOUNT
        MOVEQ R1,#20
        SUB R1,R1,#10
        STR R1, [R0]
SHOW_BET:
        PUSH {R0-R12,LR}
        LDR R3, =BET_AMOUNT
        LDR R0, [R3]//BET_AMOUNT
        BL SEVEN_SEGMENT_DISPLAYER
        POP {R0-R12,LR}
        B END_KEY_ISR

//INCREASE THE BET AMOUNT IF GAME_STATUS=1
CHECK_KEY2:
        MOV R3, #0x4
        ANDS R3, R3, R1 // CHECK FOR KEY2
        BEQ CHECK_KEY3
        LDR R0,=GAME_STATUS
        LDR R1, [R0]
        CMP R1,#1
        BNE SHOW_BALANCE
        LDR R0, =BET_AMOUNT
        LDR R1, [R0]
        CMP R1,#100
        MOVEQ R1,#90
        ADD R1,R1,#10
        STR R1, [R0]
SHOW_BALANCE:
        PUSH {R0-R12,LR}
        LDR R0,=GAME_STATUS
        LDR R1, [R0]
        CMP R1,#1 //IF GAME STATUS 1 LOAD BET_AMOUNT, ELSE LOAD BALANCE
        LDRNE R3,=BALANCE
        LDREQ R3, =BET_AMOUNT
        LDR R0, [R3] //BALANCE or BET_AMOUNT
        BL SEVEN_SEGMENT_DISPLAYER
        POP {R0-R12,LR}
        B END_KEY_ISR

//GAME STATUS CHANGER
CHECK_KEY3:
```

```
        MOV R3, #0x8
        ANDS R3, R3, R1 // CHECK FOR KEY3
        BEQ END_KEY_ISR
        LDR R0, =NUMBER_OF_MINES
        LDR R1, [R0] //NUMBER_OF_MINES
//IF A PLAYER STARTS THE GAME WITHOUT SETTING THE MINE NUMBER
// AUTOMATICALLY SET MINE NUMBER TO ONE
        CMP R1,#0
        MOVEQ R1,#1
        STREQ R1,[R0]
        LDR R0,=GAME_STATUS
        LDR R1, [R0]
        ADD R1,R1,#1
        STR R1,[R0]
        CMP R1, #1
//ONLY WHEN PROCEEDING TO THE GAME_STATUS 1
//GET THE RANDOM MINE LOCATIONS
        PUSH {R0-R12,LR}
        BLEQ GET_MINE_LOCATIONS_BIT
        POP {R0-R12,LR}
        CMP R1, #3
        MOVEQ R1,#0
        STR R1,[R0]

END_KEY_ISR:
//SHOW THE GAME STATUS IN THE SEVEN SEGMENT
        LDR R1,=GAME_STATUS
        LDR R1,[R1]
        LDR R3, =HEXTABLE
        LDR R12, [R3, R1, LSL #2]
        LDR R2, =0xFF200030
        LSL R12,#8 //TO SHOW IT IN THE LEFT SIDE OF THE SEVEN SEGMENT
        STR R12,[R2]
        BX LR

//FILL THE LOCATIONS ARRAY WITH RANDOM NUMBERS
//GET NECESSARY NUMBER OF ELEMENTS FROM THE ARRAY
//STORE THEM IN MINE_LOCATIONS_BIT
GET_MINE_LOCATIONS_BIT:
LOOP://RANDOMIZE THE ARRAY
        LDR R2, =0xFFFEC600
        LDR R3, [R2,#4] //PRIVATE TIMER CURRENT VALUE
DELAY_MINE:
        LDR R7, =2000// delay counter
DELAY_LOOP_MINE:
        SUBS R7, R7, #1
        BNE DELAY_LOOP_MINE
//MASK LAST 8 BIT OF THE NUMBER AND GET A NUMBER BETWEEN 1 AND 10 IN R3
LOOP2:
        AND R3,R3, #0xFF
        CMP R3,#0
        BEQ LOOP
        CMP R3,#10
//SUBTRACT THE MASK NUMBER UNTIL WE HAVE THE LAST DIGIT
        SUBGT R3,R3,#10
        BGT LOOP2

// CHECK THE WHOLE ARRAY IF THERE IS A DOUBLE
MOV R0,#10 //COUNTER FOR IS_REPEATED IN R0
LDR R1,=LOCATIONS
IS_REPEATED:
```

```
		LDR R2,[R1],#4
		CMP R2,R3
		BEQ LOOP
		SUB R0,R0,#1
		CMP R0,#0
		BNE IS_REPEATED

// STORE THE NUMBER IF THERE ARE NO REPETITION
MOV R0,#9 //COUNTER FOR STORE_NUM IN R0
LDR R1,=LOCATIONS
STORE_NUM:
		LDR R2,[R1],#4
		CMP R2,#0
		STREQ R3,[R1,#-4]
		BEQ LOOP
		SUB R0,R0,#1
		CMP R0,#0
		BNE STORE_NUM

//WHEN ARRAY IS FILLED WITH RANDOM NUMBERS
//CONVERT THE NECESSARY NUMBERS TO BITWISE LOCATIONS
//EXAMPLE: 7= 0001000000
LDR R0,=NUMBER_OF_MINES
LDR R1,[R0] //NUMBER OF MINES
LDR R0,=LOCATIONS
MOV R2,#1 //TEMPORARY VARIABLE TO SHIFT
MOV r4,#0 //LAST VERSION OF MINE_LOCATIONS_BIT
SHOW_NUM:
		LDR R3,[R0],#4
		SUB R3,R3,#1
		LSL R3,R2,R3
		ORR R4,R4,R3
		SUBS R1,R1,#1
		BGT SHOW_NUM
		LDR R0,=MINE_LOCATIONS_BIT
		STR R4,[R0]
//COMMENT OUT FOR NORMAL GAME
////////////////
		LDR R0,=0xff200000 //LEDs ADDRESS
		STR R4,[R0]
////////////////
		BX LR


SEVEN_SEGMENT_DISPLAYER:
		LDR R5, =DIVISORS
		LDR R6,[R5]			//DIVISOR IN R6
		MOV R2,#0  //NEED TO RESETED FOR THE PROGRAM
		MOV R10,#0 //NEED TO RESETED FOR THE PROGRAM
COUNTER_FOR_DECIMAL:
		MOV R1,#0 //COUNTER FOR EVERY DECIMAL BIT
//COUNTER TO CHECK WE RUN 4 TIMES TOTAL, FOR EVERY DECIMAL BIT ONE TIME
		ADD R2,#1
		CMP R2,#4
		BGT FIN
DIVISOR:
		CMP R0,R6
//AFTER CHECKING OUR NUMBER IS SMALLER THAN OUR DIVISOR
//WE CHANGE TO SMALLER ONE
		BLT SMALLER_DIVISOR
		SUB R0,R0,R6
```

```
        ADD R1,R1,#1
        B DIVISOR
SMALLER_DIVISOR:
        LDR R6,[R5,#4]! //GET THE NEXT DIVISOR FROM LIST OF DIVISORS
        LDR R3, =HEXTABLE
        LDR R12, [R3, R1, LSL #2]
        ORR R10,R10,R12
        CMP R2,#4
        LSLLT R10,#8        //TO STORE THEM IN ONE REGISTER
        B COUNTER_FOR_DECIMAL           //TO UPDATE VARIABLES
FIN:
        LDR R2, =ADDR_7SEG1
        STR R10,[R2]
        BX LR

//FOR VISUAL PURPOSES INCREASE THE BET AMOUNT SLOWLY
SEVEN_SEGMENT_SLOW_INCREASE:
        PUSH {R0-R12,LR}
        LDR R0, =PREVIOUS_BET_AMOUNT
        LDR R3,[R0]
SLOW_START:
        LDR R1, =HEXTABLE
        LDR R2, =ADDR_7SEG1
        MOV R0,R3
        LDR R5, =DIVISORS
        LDR R6,[R5] //DIVISOR
        PUSH {R1-r4}
        MOV R2,#0
        MOV R10,#0
        BL SLOW_COUNTER_FOR_DECIMAL
        POP {R1-r4}
        STR R10,[R2]

SLOWDO_DELAY:
        LDR R7, =1000000 // DELAY COUNTER
SLOW_SUB_LOOP:
        SUBS R7, R7, #1
        BNE SLOW_SUB_LOOP
        LDR R9,=BET_AMOUNT
        LDR R9,[R9]
        CMP R3,R9
        ADD R3,#1
        BEQ SLOW_END
        B SLOW_START
SLOW_COUNTER_FOR_DECIMAL:
//COUNTER FOR EVERY DECIMAL BIT
        MOV R1,#0
//COUNTER TO CHECK WE RUN 4 TIMES TOTAL, FOR EVERY DECİMAL BIT ONE TIME
        ADD R2,#1
        CMP R2,#4
        BGT SLOW_FIN
SLOW_DIVISOR:
        CMP R0,R6
//AFTER CHECKING OUR NUMBER IS SMALLER THAN OUR DIVISOR
//WE CHANGE TO SMALLER ONE
        BLT SLOW_SMALLER_DIVISOR
        SUB R0,R0,R6
        ADD R1,R1,#1
        B SLOW_DIVISOR

SLOW_SMALLER_DIVISOR:
```

```
        LDR R6,[R5,#4]! //GET THE NEXT DIVISOR
        LDR R3, =HEXTABLE
        LDR R12, [R3, R1, LSL #2]
        ORR R10,R10,R12
        CMP R2,#4
        LSLLT R10,#8        //WE SHIFT IT ONLY WHEN R2<4
        B SLOW_COUNTER_FOR_DECIMAL    //TO UPDATE VARIABLES

SLOW_FIN:
        BX LR

SLOW_END:
        POP {R0-R12,PC}



/* Define the exception service routines */
/*--- Undefined instructions --------------------------------------*/
SERVICE_UND:
        B SERVICE_UND
/*--- Software interrupts ----------------------------------------*/
SERVICE_SVC:
        B SERVICE_SVC
/*--- Aborted data reads -----------------------------------------*/
SERVICE_ABT_DATA:
        B SERVICE_ABT_DATA
/*--- Aborted instruction fetch ----------------------------------*/
SERVICE_ABT_INST:
        B SERVICE_ABT_INST

/*--- IRQ ---------------------------------------------------------*/
SERVICE_IRQ:
        PUSH {R0-R7, LR}
        /* Read the ICCIAR from the CPU Interface */
        LDR R4, =0xFFFEC100
        LDR R5, [R4, #0x0C] // read from ICCIAR
PRIVATE_TIMER_CHECK:
        CMP R5, #29 // check for FPGA timer interrupt
        BNE FPGA_IRQ1_HANDLER
        BL PRIVATE_TIMER_ISR
        B EXIT_IRQ
FPGA_IRQ1_HANDLER:
        CMP R5, #73
UNEXPECTED:
        BNE UNEXPECTED // if not recognized, stop here
        BL KEY_ISR
EXIT_IRQ:
        /* Write to the End of Interrupt Register (ICCEOIR) */
        STR R5, [R4, #0x10] // write to ICCEOIR
        POP {R0-R7, LR}
        SUBS PC, LR, #4
/*--- FIQ ---------------------------------------------------------*/
        SERVICE_FIQ:
        B SERVICE_FIQ

/* ^^^^ END of Define the exception service routines ^^^^ */


PRIVATE_TIMER_ISR:
        LDR R1, =0xFFFEC600 // interval timer base address
        MOV R0, #1
```

```
        STR R0, [R1,#12] // clear the interrupt
        BX LR

//CONFIGURATIONS

/* Configure the Generic Interrupt Controller (GIC)
        */
CONFIG_GIC:
        PUSH {LR}
/* To configure the FPGA KEYS interrupt (ID 73):
* 1. set the target to cpu0 in the ICDIPTRn register
* 2. enaBLe the interrupt in the ICDISERn register */

/* CONFIG_INTERRUPT (int_ID (R0), CPU_target (R1)); */
        MOV R0, #73 // KEY port (Interrupt ID = 73)
        MOV R1, #1 // this field is a bit-mask; bit 0 targets cpu0
        BL CONFIG_INTERRUPT
        MOV R0, #29 // Private timer port (Interrupt ID = 29)
        MOV R1, #1
        BL CONFIG_INTERRUPT

        /* configure the GIC CPU Interface */
        LDR R0, =0xFFFEC100 // base address of CPU Interface

        /* Set Interrupt Priority Mask Register (ICCPMR) */
        LDR R1, =0xFFFF // enaBLe interrupts of all priorities levels
        STR R1, [R0, #0x04]

        /* Set the enaBLe bit in the CPU Interface Control Register (ICCICR).
        * This allows interrupts to be forwarded to the CPU(s) */
        MOV R1, #1
        STR R1, [R0]

        /* Set the enaBLe bit in the Distributor Control Register (ICDDCR).
        * This enaBLes forwarding of interrupts to the CPU Interface(s) */
        LDR R0, =0xFFFED000
        STR R1, [R0]
        POP {PC}

/* Configure the pushbutton KEYS to generate interrupts */
CONFIG_KEYS:
        // write to the pushbutton port interrupt mask register
        LDR R0, =0xFF200050 // pushbutton key base address
        MOV R1, #0xF // set interrupt mask bits
        STR R1, [R0, #0x8] // interrupt mask register is (base + 8)
        BX LR

/* Configure the Private timer to generate interrupts */
CONFIG_PRIVATE_TIMER:
        LDR R0, =0xFFFEC600 // Interval timer base address
        LDR R1, =5000000 // 1/(200 MHz) ×(5000000) = 100 msec
        STR R1, [R0] // write to timer load register
        MOV R2, #0b111 // set bits: mode = 1 (auto), enaBLe = 1
        STR R2, [R0, #0x8] // write to timer control register
        BX LR

/*
* Configure registers in the GIC for an individual Interrupt ID
* We configure only the Interrupt Set EnaBLe Registers (ICDISERn) AND
* Interrupt Processor Target Registers (ICDIPTRn). The default (reset)
* values are used for other registers in the GIC
```

```
* Arguments: R0 = Interrupt ID, N
* R1 = CPU target
*/
CONFIG_INTERRUPT:
        PUSH {R4-R5, LR}
/* Configure Interrupt Set-EnaBLe Registers (ICDISERn).
* reg_offset = (integer_div(N / 32) * 4
* value = 1 << (N mod 32) */
        LSR R4, R0, #3 // calculate reg_offset
        BIC R4, R4, #3 // R4 = reg_offset
        LDR R2, =0xFFFED100
        ADD R4, R2, R4 // R4 = address of ICDISER
        AND R2, R0, #0x1F // N mod 32
        MOV R5, #1 // enaBLe
        LSL R2, R5, R2 // R2 = value

/* Using the register address in R4 AND the value in R2 set the
* correct bit in the GIC register */
        LDR R3, [R4] // read current register value
        ORR R3, R3, R2 // set the enaBLe bit
        STR R3, [R4] // store the new register value

/* Configure Interrupt Processor Targets Register (ICDIPTRn)
* reg_offset = integer_div(N / 4) * 4
* index = N mod 4 */
        BIC R4, R0, #3 // R4 = reg_offset
        LDR R2, =0xFFFED800
        ADD R4, R2, R4 // R4 = WORD address of ICDIPTR
        AND R2, R0, #0x3 // N mod 4
        ADD R4, R2, R4 // R4 = byte address in ICDIPTR

/* Using register address in R4 AND the value in R2 write to
* (only) the appropriate byte */
        STRB R1, [R4]
        POP {R4-R5, PC}
```