ECE331 Project

REGISTERS AT THE START

```
(gdb) x/15gd $x0
0x75c7735ffb48: 7       16
0x75c7735ffb58: 23      40
0x75c7735ffb68: 11      39
0x75c7735ffb78: 37      10
0x75c7735ffb88: 2       18
0x75c7735ffb98: 44      83
0x75c7735ffba8: 87      5
0x75c7735ffbb8: 6
```

```
(gdb) x/15gd $x1
0x75c7735ffac8: 0       0
0x75c7735ffad8: 0       0
0x75c7735ffae8: 0       0
0x75c7735ffaf8: 0       0
0x75c7735ffb08: 0       0
0x75c7735ffb18: 0       0
0x75c7735ffb28: 0       0
0x75c7735ffb38: 0
```

```
(gdb) x/15gd $x2
0x75c7735ffa48: 0       0
0x75c7735ffa58: 0       0
0x75c7735ffa68: 0       0
0x75c7735ffa78: 0       0
0x75c7735ffa88: 0       0
0x75c7735ffa98: 0       0
0x75c7735ffaa8: 0       0
0x75c7735ffab8: 0
```

REGISTERS AT THE END

```
(gdb) x/16gd $x1
0x71bd3fdffac8: 7       23
0x71bd3fdffad8: 11      37
0x71bd3fdffae8: 2       83
0x71bd3fdffaf8: 5       11
0x71bd3fdffb08: 0       0
0x71bd3fdffb18: 0       0
0x71bd3fdffb28: 0       0
0x71bd3fdffb38: 0       0
```

```
(gdb) x/16gd $x2
0x71bd3fdffa48: 16      40
0x71bd3fdffa58: 39      10
0x71bd3fdffa68: 18      44
0x71bd3fdffa78: 87      6
0x71bd3fdffa88: 0       0
0x71bd3fdffa98: 0       0
0x71bd3fdffaa8: 0       0
0x71bd3fdffab8: 0       0
```

Register group: general

x0	0xb	11	x1	0x71bd3fdffac8	125057634400968
x2	0x71bd3fdffa48	125057634400840	x3	0x10	16
x4	0x71bd3fdffb48	125057634401096	x5	0x71bd3fdffac8	125057634400968
x6	0x71bd3fdffa48	125057634400840	x7	0x10	16
x8	0x10	16	x9	0x8	8
x10	0x8	8	x11	0xb	11
x12	0x6	6	x13	0x5	5
x14	0xa	10	x15	0x71bd4e8549fc	125057880115708
x16	0x71bd4c9efd68	125057848245608	x17	0x71bd4d2e76d0	125057857648336

isPrimeAssembly.s

```
[achin@AidanEOS Project]$ qemu-aarch64 -L /usr/aarch64-linux-gnu/ -g 1234 ./isPrime
Input Array elements were: 7 16 23 40 11 39 37 10 2 18 44 83 87 5 6 11
Prime Array elements are: 7 23 11 37 2 83 5 11 0 0 0 0 0 0 0 0
Composite Array elements are: 16 40 39 10 18 44 87 6 0 0 0 0 0 0 0 0
```

```
.globl isPrimeAssembly

isPrimeAssembly:
    MOV     x4, x0              ; Save original array base
address in x4
    MOV     x5, x1              ; Save prime array base
address in x5
    MOV     x6, x2              ; Save composite array base
address in x6
    MOV     x7, x3              ; Save length in x7
    MOV     x15, x30            ; Store link register

    MOV     x8, #0              ; Initialize counter
    MOV     x9, #0              ; Initialize prime count
    MOV     x10, #0             ; Initialize non-prime
count

processArray:
    CMP     x8, x7              ; Compare counter with
length
    BGE     endLoop             ; if counter >= length, end
loop

    LDR     x0, [x4, x8, LSL #3] ; Load
original_array[counter] into x0
    BL      isPrime             ; Call isPrime
    CBZ     x0, nonPrime        ; If result is 0 (non-
prime), go to nonPrime

Prime:
    LDR     x0, [x4, x8, LSL #3] ; Load
original_array[counter] into x0
    STR     x0, [x5, x9, LSL #3] ; Store in
prime_array[prime count]
    ADDI    x9, x9, #1           ; Increment prime count
    B       incrementCount       ; Go to next iteration
```

```
nonPrime:
    LDR     x0, [x4, x8, LSL #3] ; Load
original_array[counter] into x0
    STR     x0, [x6, x10, LSL #3]; Store in
composite_array[non-prime count]
    ADDI    x10, x10, #1         ; Increment non-prime
count
    B       incrementCount       ; Go to next iteration

incrementCount:
    ADDI    x8, x8, #1           ; Increment counter
    B       processArray         ; Repeat loop

endLoop:
    MOV     x30, x15             ; Return the link address
    BR      X30                  ; Return to caller by
branching to X30

isPrime:
    MOV     x11, x0              ; Store input number in
x11
    MOV     x12, #2              ; Initialize divisor with
2

conditional:
    LSR     x13, x11, #1         ; Set upper limit to input
/ 2
    CMP     x12, x13             ; Check if divisor > input
/ 2
    BGT     return1              ; If divisor > input / 2,
number is prime
    B       divide

divide:
    UDIV    x14, x11, x12        ; x14 = input / divisor
    MUL     x14, x14, x12        ; x14 = divisor * (input /
divisor)
    CMP     x11, x14             ; Check if input ==
```

```
(divisor * (input /

; divisor))
    BEQ     return0             ; If equal, not prime

increment:
    ADDI    x12, x12, #1        ; Increment divisor
    B       conditional

return0:
    MOVZ    x0, #0              ; Not prime
    B       exit

return1:
    MOVZ    x0, #1              ; Prime
    B       exit

exit:
    RET                         ; Return to caller
```

Description of Code

begins by initializing variables
process array takes input array and runs each element through the isPrime function, which branches to prime handler if prime and nonprime handler if nonprime
prime and nonprime input the element into the correct array
increment count keeps count of how many times the function has run
endloop sends back to parent function
isprime marks beginning of isprime function, initializes the variables used
conditional checks if divisor is > input / 2 (a quick square root approx, since you only need to test one half for primeness) if greater return 0 and branch to return 1 , otherwise branch to divide instead
divide checks for remainder with a divide and multiply to implement a modulo function, if remainder is 0, branch to return 0
increment increments divisor to move through and branches back to beginning of conditional

return 0 writes into the register and returns
return 1 writes into the register and returns
exit marks end of function isPrime

.globl isPrimeAssembly

isPrimeAssembly:

MOV x4, x0

MOV x5, x1

MOV x6, x2

MOV x7, x3

MOV x8, #0

MOV x9, #0

MOV x10, #0

MOV x15, x30

BL processArray

processArray:

CMP x8, x7

BGE endLoop

LDR x0, [x4, x8, LSL #3]

BL isPrime

CBZ x0, nonPrime

Prime:

LDR x0, [x4, x8, LSL #3]

STR x0, [x5, x9, LSL #3]

ADD x9, x9, #1

```
B incrementCount

nonPrime:

LDR x0, [x4, x8, LSL #3]

STR x0, [x6, x10, LSL #3]

ADD x10, x10, #1

incrementCount:

ADD x8, x8, #1

B processArray

endLoop:

MOV x30, x15

BR x30

isPrime:

MOV x11, x0

MOV x12, #2

conditional:

LSR x13, x11, #1

CMP x12, x13

BGT return1

divide:

UDIV x14, x11, x12

MUL x14, x14, x12

CMP x11, x14

BEQ return0
```

```
increment:

ADD x12, x12, #1

B conditional

return0:

MOVZ x0, #0

B exit

return1:

MOVZ x0, #1

B exit

exit:

RET
```