# ARM Assembly

01: mystery1

02: STMFD SP!, {R4-R8} #function prologue Store multiple instruction decrement before

03: LDRB R3, [R0] #Loads arg1 into register R3. arg1 is most likely a pointer.

04: CMP R3, #0x2D #Compare R3’s values to 45

05: BEQ loc\_B348 #If R3 is equal to 45 go to loc\_B348 on line 57.

06: CMP R3, #0x2B #Compare R3 to 43

07: MOV R6, #0 #Put 0 into R6

08: LDREQB R3, [R0,#1]! #Load R0+1 into R3 and then update R0 to R0+1 if R3 equals 43

09: loc\_B2AC

10: CMP R3, #0x30 #Compare R3 to 48

11: BNE loc B2C8 #If R3 is not equal to 48 go to loc B2C8 on line 18

12: ADD R3, R0, #1 # R3 = R0 + 1

13: loc\_B2B8

14: MOV R0, R3 #R0=R3

15: LDRB R2, [R3], #1 #R2=R3, R3=R3+1

16: CMP R2, #0x30

17: BEQ loc\_B2B8 #Loop back to line 13 while R2 == 48

18: loc\_B2C8

19: MOV R12, #0 #R12 = 0

20: MOV R4, #0 #R4 = 0

21: MOV R5, #0 #R5 = 0

22: MOV R8, #0xA #R8 = 10

23: B loc\_B2E4 #Branch to loc\_B2E4 on line 27

24: loc\_B2DC

25: ADDS R4,R2,R7 # R4 = R2+R7

26: ADC R5, R3, R7, ASR#31 #R5 = R3 +(R7>>31) with carry

27: loc\_B2E4

28: LDRB R7, [R0, R12] # R7 = R0+R12. R0 is bp and R12 is offset

29: ADD R12, R12, #1 #R12 = R12+1

30: UMULL R2, R3, R4, R8 # R4xR8. Least significant 32 bits R2 and most significant 32 bits R3.

31: SUBS R7, R7, #0x30 # R7 = R7 – 48

32: BMI loc\_B318 #branch to loc\_B318 on line 43 if R7 is negative

33: CMP R7, #9 #Compare R7 to 9

34: MLA R3, R8, R5, R3 #R3 = R8xR5+R3

35: BGT loc\_B318 # branch to loc\_B318 on line 43 if R7 greater than 9

36: CMP R12, #0xB #Compares R12 to 11

37: BNE loc\_B2DC #Branch to loc\_B2DC on line 24 if R12!=11

38: loc\_B30C

39: MOV R0, #0 #R0 = 0

40: loc\_B310

41: LDMFD SP!, {R4-R8} #function epilogue put from stack back into R4 and R8. IA mode

42: BX LR # return

43: loc\_B318

44: SUBS R2, R4, R6 # R2 = R4-R6

45: SBC R3, R5, R6, ASR#31 # R3 = R5-(R6>>31) with carry

46: CMP R2, #0x80000000 #Compare R2 with #0x80000000

47: SBCS R0, R3, #0 #R0 = R3 – 0

48: BGE loc\_B30c #Branch to loc\_B30C on line 38 if R2 greater than or equal 0x80000000

49: CMP R6,#0 #Compare R6 to 0

50: BEQ loc\_B33C #Branch to loc\_B33C on line 53 if 6 equals 0

51: RSBS R4, R4, #0 #R4 = 0-R4

52: RSC R5, R5, #0 #R5 = 0-R5

53: loc\_B33C

54: STR R4, [R1] # Store the value into R4 into R1(our second argument a pointer)

55: MOV R0, #1 # R0 = 1

56: B loc\_B310 #Branch to loc\_B310 on line 40.

57: loc\_B348

58: LDRB R3, [R0,#1]! #Load R0+1 into R3 and then update R0 to R0+1

59: MOV R6, #1 #Put 1 into R6

60: B loc\_B2Ac #Branch to loc\_B2Ac on line 09.

61: ; End of function mystery1

# Mode

This code is in ARM mode since all the instructions are 32 bits.

# Types

R0 type char \* when argument. When function returns R0 is a boolean

R1 type int \*

R2 type int

R3 type char (one byte) then later is a variable that is a char \*

int64\_t sum; //R5:R4

R6 type Boolean

R7 type int signed

R8 type int

R12 type int

# Function Prototype

mystery1(char \* arg1, int \* arg2)

# C Code

atoi(char \*arg1, int \* arg2)

{

char ch = arg1[0];

bool neg;

if(ch== ‘-‘)//check if the number is negative

{

ch = arg1[1];

arg1++;

neg = true;

}

else{ //if not negative

neg = false;

if(ch== ‘+’){ ch=arg1[1]; arg1++;} //if plus in front move over to next digit.

}

if(ch==’0’) //go through this until you reach a number that is not 0

{

char \* chp = arg1+1;

int cur\_dig;

do{

arg1 = chp;

cur\_dig = \*chp;

chp++;

} while(cur\_dig==’0’)

}

int i =0;

int num = 0;

//R5 is here in case R4 overflows from multiplication for 10 digit numbers greater than 231-1

int ten = 10;

LOOP:

int digit = arg[i];

i += 1;

int64\_t sum= num\*ten;

digit = digit – ‘0’;

if(digit>=0)

{

//R3= ten\*R3+R5 instruction there only incase of 10 digit numbers greater than 2^31-1

if(digit<= 9)

{

if(i!=11)

{

num = sum+digit; //add the digit to the end of the number

//the R5 = R3+(digit>>31) instruction is only there in case there is a ten digit value that is not 32 bit.

goto LOOP; //loop

}

else{ //max number of digits a 32 bit value can have is 10. If there are 11 or more fail.

goto fail;

}

}

}

sum = num – neg; //this is here in case the value is -2,147,483,648

if(sum >= #0x80000000) // if R2 is greater than the highest signed 32 bit value fail. 231-1 is the greatest signed 32 bit value.

{

fail: bool ret = false;

return ret;

}

if(!neg)

{

num = -num;

//R5 = -R5 is not really necessary ARM is doing this since we were working with 64 bits.

}

arg2 = &num;

bool ret = true;

return ret;

}

# Explanation

This function is atoi. It takes a string and accepts a pointer to an integer. The output is a Boolean where 0 is failing and 1 is passing. If there is a minus sign, the value is assumed negative. If it is blank or have a plus sign, that means the value will be positive. If there are any 0’s before the number, those are omitted. Then while there is a digit between 0 and 9, multiply the current number by ten and then add the digit to the running total. If there is more than 10 digits (greater than 32 bits), return 0 since we cannot have a number with that many digits. Also, if the value is not between [-231, 231-1], return 0 as well. Afterwards return 1 and have the second parameter of the function point to the integer. If the value was negative, set the negative sign on the integer.