

Lab 6: Fixed point Conversion

Victor Yuan

Microcontrollers and Embedded Systems

March 23, 2017

## Introduction:

The Goal of this lab was to write a program in assembly implementing conversion between a decimal fixed decimal point number to a binary-hexadecimal, binimal-heximal number, and converting a binary, binimal number to a decimal number.

## Procedure:

### Part1

Converting a non-integral number in base 10 to a binary fixed point number requires the number entered by the user to be first stored as a string to be manipulated. Once the number was stored as a string ldrb was used to access each number, character inside the whole string number. A compare was used to compare each number to the hex number 2e. 2e is the ascii representation for period(.) once 2e was seen in the string the program knew all the integral numbers had been accounted for. While each byte was loaded in a register if it was not 2e then it would either be 0x30 to 0x39, this program did not have error checking since it was assumed the user would input a correct number. 0x30 was then subtracted from the byte loaded inside the register to create the integral numbers 0 through 9. This was then multiplied by 10 and stored inside another register so as to create the numeric representation in decimal of the integer part. This integer was then called later in the program as a hex number by formatting the output as %x thus printing out the integer part.

The 4 decimal places entered after the radix point where converted into a integer representation in the same process as the number before however, changes to the format output was needed. The number was left shifted by 4, same thing as multiplying by 16 then divided by 10,000 for the to remove 4 decimal places. This would generate the first hexadecimal integer for the output. This was stored in variable hex1 to be output later. The integer would then be multiplied by 100 2 times to

simulate being multiplied by 10,000. The original value would subtract this and stored back in the same register to continue this process. Until all 4 hex digits were stored.

Finally the digits were called and formatted in the form desired.

## Part 2

For simplicity I wrote a separate program for part2 to clear up any registers. For binary to decimal conversion, I first used a loop to count how many 1 and 0s were in the string number before the radix point, 2e. Then a loop was constructed to multiply 1 or 0 with 2 to the power of the counter beforehand. After each loop the value was added to a separate register this allowed the value to be converted to a decimal integer and each loop the counter was decreased by 1.

For the 8 decimal places conversion I had to manipulate more numbers. I multiplied constants until 50,000,000 was achieved. I would then load single bytes from the string number, it would either be 1 or a 0 then multiply it with the register containing 50,000,000. Whether if it's a 1 or a 0 the result was added to a separate register and the register containing the 50,000,000 was right shifted by 1, divided by 2. This would then become 25,000,000 this process was looped 8 times to simulate  $2^{-1}$  to  $2^{-8}$ . The total number would look like the expected decimal once the output was formatted.

Code:

Part1:

---

```
.data
str1: .asciz "%s"
num: .asciz "%s%s%s%s%s%s"
p1: .word 0
p2: .word 0
prompt1: .asciz "enter a 4 decimal base 10 number\n"
ans: .asciz "The number you entered is %x."
ansp1: .asciz "%x"
ansp2: .asciz "%x\n"
hex1: .word 0
hex2: .word 0
hex3: .word 0
hex4: .word 0
.text
.global main
main:
    mov r7, #0
    ldr r0, =prompt1
    bl printf
    ldr r0, =str1
    ldr r1, =num
    bl scanf
    ldr r4, =num
    ldr r0, =p1
    mov r1, #10
loop:
    ldrb r5, [r4], #1
    cmp r5, #0x2e
    mulne r7, r7, r1
    subne r6, r5, #0x30 //now r6 holds first first digit
    addne r7, r7, r6
    bne loop
    str r7, [r0]
    ldr r0, =p2
    mov r7, #0 //resets register for the 2nd half of the number after decimal
    mov r2, #0 //4 decimal palces
loop2:
    ldrb r5, [r4], #1
    sub r6, r5, #0x30
    add r7, r7, r6
    add r2, #1
    cmp r2, #4
    mulne r7, r7, r1
    bne loop2
    str r7, [r0] //makes a copy of the number and store it at "p2:"
    mov r6, r7 //makes a copy of the number into r6
    lsl r7, #4
    mov r12, #105 //these 3 lines
    mul r7, r7, r12 //
    lsr r7, #20 //are = dividing by 10,000. chop off the digits at the end
    ldr r0, =hex1
    str r7, [r0]
    mov r12, #100
    mul r7, r7, r12
    mul r7, r7, r12 //multiply by 100 2 times is = mul by 10,000 other ways of doing this aswell
    lsl r6, #4
    sub r6, r6, r7
```

First part of code where the numbers before and after the decimal were determined and placed into p1 and p2.

```

//at this point hex1, the first number after the heximal(radix, decimal for hex numbers) is stored repeat this 3 more time
//for total of 3 decimal? heximal? places
    mov r7, r6
    lsl r7, #4
    mov r12, #105
    mul r7, r7, r12
    lsr r7, #20
    ldr r0, =hex2
    str r7, [r0]
    mov r12, #100
    mul r7, r7, r12
    mul r7, r7, r12
    lsl r6, #4
    sub r6, r6, r7
//heximal place 2 completed
    mov r7, r6
    lsl r7, #4
    mov r12, #105
    mul r7, r7, r12
    lsr r7, #20
    ldr r0, =hex3
    str r7, [r0]
    mov r12, #100
    mul r7, r7, r12
    mul r7, r7, r12
    lsl r6, #4
    sub r6, r6, r7
//hex3 complete
    mov r7, r6
    lsl r7, #4
    mov r12, #105
    mul r7, r7, r12
    lsr r7, #20
    ldr r0, =hex4
    str r7, [r0]
    mov r12, #100
    mul r7, r7, r12
    mul r7, r7, r12
    lsl r6, #4
    sub r6, r6, r7

```

R7 and R6 held the same value r6 was the backup version for when I needed to access it again.

R7 was multiplied by 16 divided by 10,000 stored in hex1, 2, 3, 4, then multiplied by 10,000 and subtracted from r6 so the process can repeat for all 4 hex numbers.

```

//hex4 complete now to write the print out statement
    ldr r10, =p1
    ldr r11, =p2
    ldr r10, [r10]
    ldr r11, [r11]
    ldr r1, =p1
    ldr r1, [r1]
    ldr r0, =ans
    bl printf
    ldr r1, =hex1
    ldr r1, [r1]
    ldr r0, =ansp1
    bl printf
    ldr r1, =hex2
    ldr r1, [r1]
    ldr r0, =ansp1
    bl printf
    ldr r1, =hex3
    ldr r1, [r1]
    ldr r0, =ansp1
    bl printf
    ldr r1, =hex4
    ldr r1, [r1]
    ldr r0, =ansp2 //switched to ansp2 for the "/n for formating issues"
    bl printf

```

Print out commands.

```

.data
str1: .asciz "%s"
num: .asciz "%s%s%s%s%s%s"
p1: .word 0
p2: .word 0
prompt1: .asciz "Enter a 4 binimal base 2 number(Only Binary!)\n"
ans: .asciz "The number you entered is %d."
ans2: .asciz "%d\n"

.text
.global main

main:
    ldr r0, =prompt1
    bl printf
    ldr r0, =str1
    ldr r1, =num
    bl scanf
    ldr r4, =num
    mov r1, #0

count: //counts amt of lsl needed
    ldrb r5, [r4], #1
    cmp r5, #0x2e
    addne r1, #1
    bne count
    ldr r0, =p1

    sub r4, r4, r1 //resets r4
    sub r4, #1
    sub r1, #1

loop:
    ldrb r5, [r4], #1 //all values in r5 should be 0 or 1

    cmp r5, #0x2e //compare byte to ,
    streq r7, [r0]
    beq afterDec //if the byte is not equal to . then cont untill . is seen

    sub r6, r5, #0x30 //conversion to 0 or 1
    lsl r6, r1 //if r6 is 0 then left shifting it by 2^(r1) wont matter
    add r7, r7, r6
    sub r1, r1, #1 //decrease the shift counter by 1
    b loop

```

Count for determine the variable for  $2^{r1}$ , used to calculate integral values before the decimal

```

afterDec:

    ldr r0, =p2

    mov r1, #100
    mov r2, #50
    mul r2, r2, r1
    mul r2, r2, r1
    mul r2, r2, r1 //do this 3 times for 50,000,000
    mov r1, #0
    mov r7, #0 //r7 still holds the number
    //at the start of the loop r2 holds 50,000,000
    //goal to have it become 00.390,625 at the end
    //to do this divide 50m by 2, 7 times lsr, 1 each time
loop2:
    ldrb r5, [r4], #1
    sub r6, r5, #0x30
    mul r6, r6, r2 //r6 is either 0 or 1 r2 is 50m divided by 2 every time it loops
    add r7, r7, r6
    lsr r2, #1
    add r1, #1
    cmp r1, #8
    blt loop2

    str r7, [r0]

    ldr r7, =p1
    ldr r7, [r7]
    mov r6, r7 //r6,r7 both holds binary _ _ _ _ number

    ldr r1, =p1
    ldr r1, [r1]
    ldr r0, =ans
    bl printf

    ldr r1, =p2
    ldr r1, [r1]
    ldr r0, =ans2
    bl printf

```

Loop 2 used for determining value used in numbers after decimal

Results:

Part1:

```
lab7
enter a 4 decimal base 10 number:
1234,1234
The number you entered is 4d2,1f97
█
```

```
lab7
enter a 4 decimal base 10 number:
1337,3210
The number you entered is 539,522d
█
```

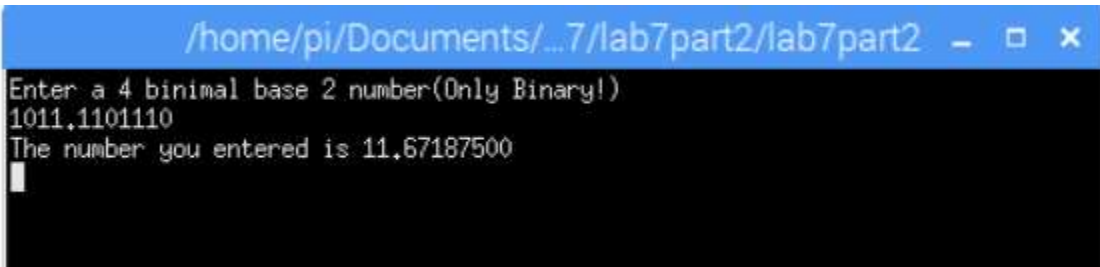
```
lab7
enter a 4 decimal base 10 number:
16,9375
The number you entered is 10,f000
█
```

Part2:

```
/home/pi/Documents/_7/lab7part2/lab7part2 - □ ×
Enter a 4 binimal base 2 number(Only Binary!)
1111,11111111
The number you entered is 15,99609375
█
```

```
/home/pi/Documents/_7/lab7part2/lab7part2 - □ ×
Enter a 4 binimal base 2 number(Only Binary!)
10101010,00110011
The number you entered is 170,95312500
█
```



A terminal window with a blue title bar containing the path "/home/pi/Documents/...7/lab7part2/lab7part2" and standard window controls. The terminal has a black background with white text. It displays a prompt "Enter a 4 binimal base 2 number(Only Binary!)", the user input "1011.1101110", and the program output "The number you entered is 11.67187500". A white cursor is visible on the line following the output.

```
/home/pi/Documents/...7/lab7part2/lab7part2 - □ ×  
Enter a 4 binimal base 2 number(Only Binary!)  
1011.1101110  
The number you entered is 11.67187500  
█
```

The program was successful.

Conclusion:

There definitely a more efficient way of achieving the same result. A number stored in a memory location can be forced to display as a hexadecimal number by using %x. also an ascii value can be converted to numerical number by subtracted by 0x30.