Lab01 Report: Counting How Many 1's

0. Category

1. The purpose

In this assignment, it is required to write a program in LC-3 machine language that counts how many 1's are in the lower B bits of a given number A, and stores the output in memory.

The program should start at memory location x3000. The value of the A and B should be set manually in x3100 and x3101 respectively. A is a positive number ranging from 0x0001 to 0x7FFF. The output should be stored in memory location x3102.

2. Examples

Here are several examples:

Number A	Bits B	Output
13	3	x0002
167	6	x0004
32767	15	x000F

3. Pseudocode

It's relatively easier for me to write a pseudocode first to have a frame to work with, therefore I write the pseudocode first, which is shown after.

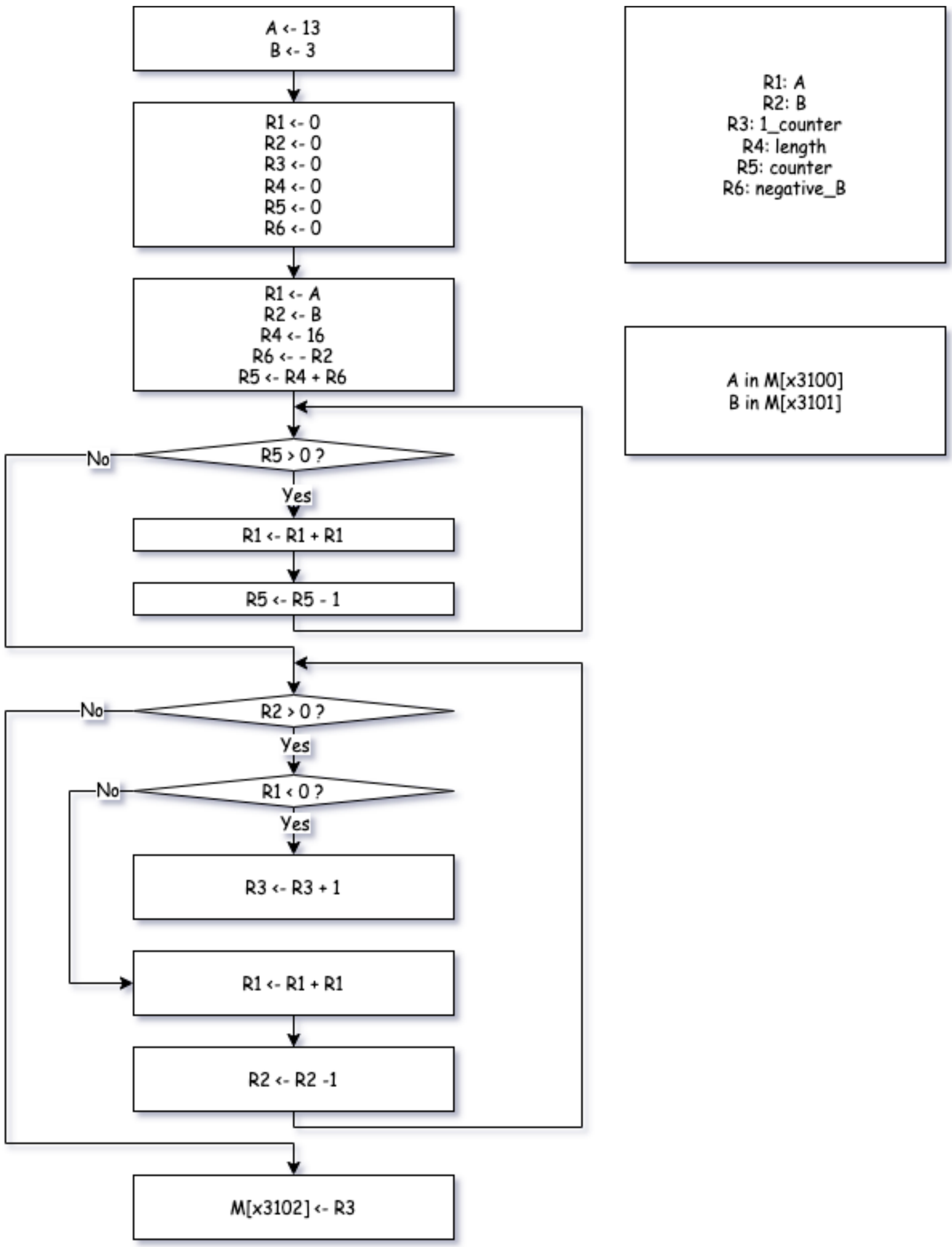
```
A = 0000 0000 0000 1101
B = 3

a = A
b = B
1_counter = 0
length = 16
counter = length - b

//First, left shift the number until there just B bits of effective data at the front.
while(counter > 0)
{
  //a.left_shift()
  a = a + a
  counter--
}

//Examine the fitst bit one by one. And then left shift it.
while(b > 0)
{
  if(a < 0)
  {
    1_counter++
  }
  //a.left_shift()
  a = a + a
  b--
}

store 1_counter
```

Nothing these:

- The left shift operation is done by double the binary number;
- The approach to count the number of 1's is by examining whether the shifted number is negative or positive. If it's negative, then increment the 1-counter.

4. Flowchart & Principle

The flowchart is shown below.



5. First Step: Initialization

The first step is and always will be to initialize all the variables, which means providing starting values (aka initial values) for R0 to R6.

1. Notice that in fact R0 is not in the flowchart - it is because

when writing the machine language, I find it much more convenient to store the x3100 memory address in R0.
2. R1 will get the number A stored in memory x3100.
3. R2 will get the number B stored in memory x3101.
4. R3 will keep track of the number of 1's.
5. R4 will save the decimal number 16, which is the total number of bits in a memory address.
6. R5 will keep track how many times more the binary numebr stored in R1 should be shift left before counting the number of 1's, which is A minus B.
7. R6 is used to stored decimal number - B, which could make subtract operation more friendly to do.

Second Step: Shift A Left Before Counting

The approach to control the iteration is using a counter. Say A is 167 and B is 6, the bits we need to pay attention to are only bits[5:0]. Therefore, before actually counting the 1's, I left shift binary number A 161—that is A minus B times.

In each iteration of the loop, the contents of R2 is compared to 0. If it is zero or negative, the loop will be exited. If it is positive, then left shift the number and decrement the counter in R3. After that, repeat it.

Third Step: Count the 1's

If the Loop in the second step is exited, the most significant part of the program will begin.

The way to control the loop is very similar to the second step, which is utilizing a counter. In each iteration, whether the contents in R2, which is how many effective bits left in the binary number, is positive or not will be examined.

If the contents in R2 is negative or zero, which means all the bits are already examined, the loop will be exited.

Or else, it comes to examine the first bit in the binary number. And we can do that by judge whether the number is negative or not. If it's negative, it means the first bit is 1 and R3 should be incremented, and vice versa. Then left shift contents in R1 and decrement R2.

Fourth Step: Store the Result

Store the contents in R3.

6. Program & Principle

```
0011 0000 0000 0000; The program starts at x3000
0101 000 000 1 00000; AND R0, R0, #0    X3000
0101 001 001 1 00000; AND R1, R1, #0
0101 010 010 1 00000; AND R2, R2, #0
0101 011 011 1 00000; AND R3, R3, #0
0101 100 100 1 00000; AND R4, R4, #0
0101 101 101 1 00000; AND R5, R5, #0
0101 110 110 1 00000; AND R6, R6, #0    X3006

1110 000 0 1111 1000; LEA R0, #X3100    X3007
0110 001 000 0 00000; LDR R1, R0, #0
0110 010 000 0 00001; LDR R2, R0, #1
0001 100 100 1 01111; ADD R4, R4, #15
0001 100 100 1 00001; ADD R4, R4, #1
1001 110 010 111111; NOT R6, R2
0001 110 110 1 00001; ADD R6, R6, #1
0001 101 100 0 00 110; ADD R5, R4, R6   X300E

0000 110 0 0000 0011; BRnz #3           X300F
0001 001 001 0 00 001; ADD R1, R1, R1
0001 101 101 1 11111; ADD R5, R5, #-1
0000 111 1 1111 1100; BRnzp #-4         X3012

0001 010 010 1 00000; ADD R2, R2, #0    X3013

0000 110 0 0000 0110; BRnz #6           x3014
0001 001 001 1 00000; ADD R1, R1, #0
0000 011 0 0000 0001; BRzp #1
0001 011 011 1 00001; ADD R3, R3, #1
0001 001 001 0 00 001; ADD R1, R1, R1
0001 010 010 1 11111; ADD R2, R2, #-1
0000 111 1 1111 1001; BRnzp #-7         x301A

0111 011 000 0 00010; STR R3, R0, #2
1111 0000 0010 0101; HALT
```

Step 1: Initialization

1. The instructions start at x3000 to x3006 clear all the registers needed by ANDing them with x0000.
2. The instruction in x3007 loads R0 with the address x3100;
3. The instruction in x3008 loads R1 with the contents in memory x3100, which is A;
4. The instruction in x3009 loads R2 with the contents in memory x3101, which is B;
5. The instructions in x300A and x300B set R4 to decimal 16;
6. The instructions in x300C and x300D give R6 the decimal number -B;
7. The instruction in x300E sets R5 to decimal number 16 - B;

Step 2: Loop Before the Counting Begin

- The instructions in x300E and x3011 guarantee that in this loop, the N, Z, P is constantly associated with the contents in R5;
- The instruction in x300F examine whether the contents in R5 is (negative or zero) or not, and if it's true, the conditional branch instruction takes the PC to the address x3013, making it out of this loop;
- The instruction in x3010 is the left shift operation;
- The instruction in x3011 decrement the contents in R5 by 1, meaning a loop has been executed;
- The instruction in x3012 is actually an unconditional branch, taking the PC back to x300F under all circumstances.

Step 3: Loop of Counting 1's

- The instruction in x3013 associate NZP with R2;
- The instruction in x3014 examine whether the content in R2 is (negative or zero) or not. If it is true, this conditional branch instruction will take the PC to address x301B to exit the loop. Or else, just increment PC;
- The instruction in x3015 associate NPZ with R1;
- The instruction in x3016 examine whether the content in R1 is (postive or zero) or not. If it is true, this conditional branch instruction will take the PC to address x3018 to jump over the instruction in x3017. Or else, just increment PC;
- The instruction in x3017 increment R3, which contains the number of 1's that is already discovered;
- The instruction in x3018 left-shift the contents in R1 by

doubling it;
- The instruction in x3019 decement R2. That is, one time less the number in R1 needs to be shifted left;
- The instruction in x301A is an unconditional branch instruction, which takes the PC back to x3014.

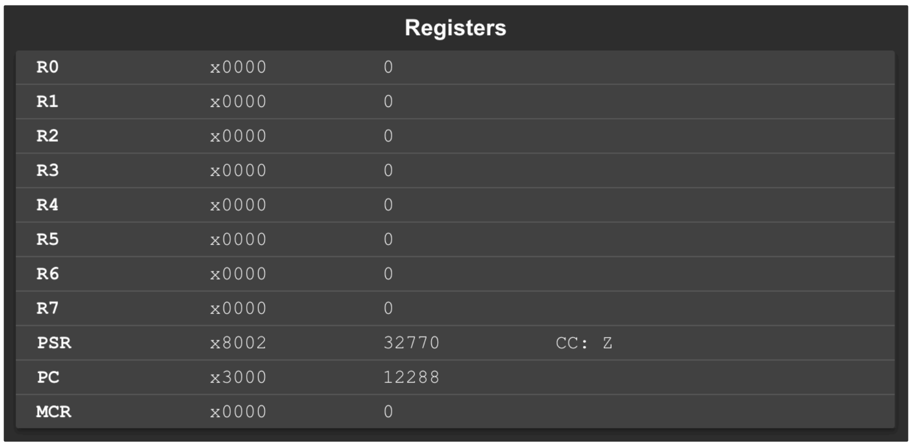
Step 4: Finish the Program

- The instruction in x301B stores the contents in R3, which is the total number of 1's to the memory addressed x3102;
- The instruction in x301C calls the system service to HALT the program.

7. Run the Program

1. Firstly, save the file as a bin file;
2. Then convert the bin file to a obj file;
3. Enter the Simulator, and it is easy to find that the instructions are already written into the memory started at x3000;
4. Manually enter A and B respectively into memory x3100 and x3101;
5. Set a break point at the HALT trap;
6. Run the program.

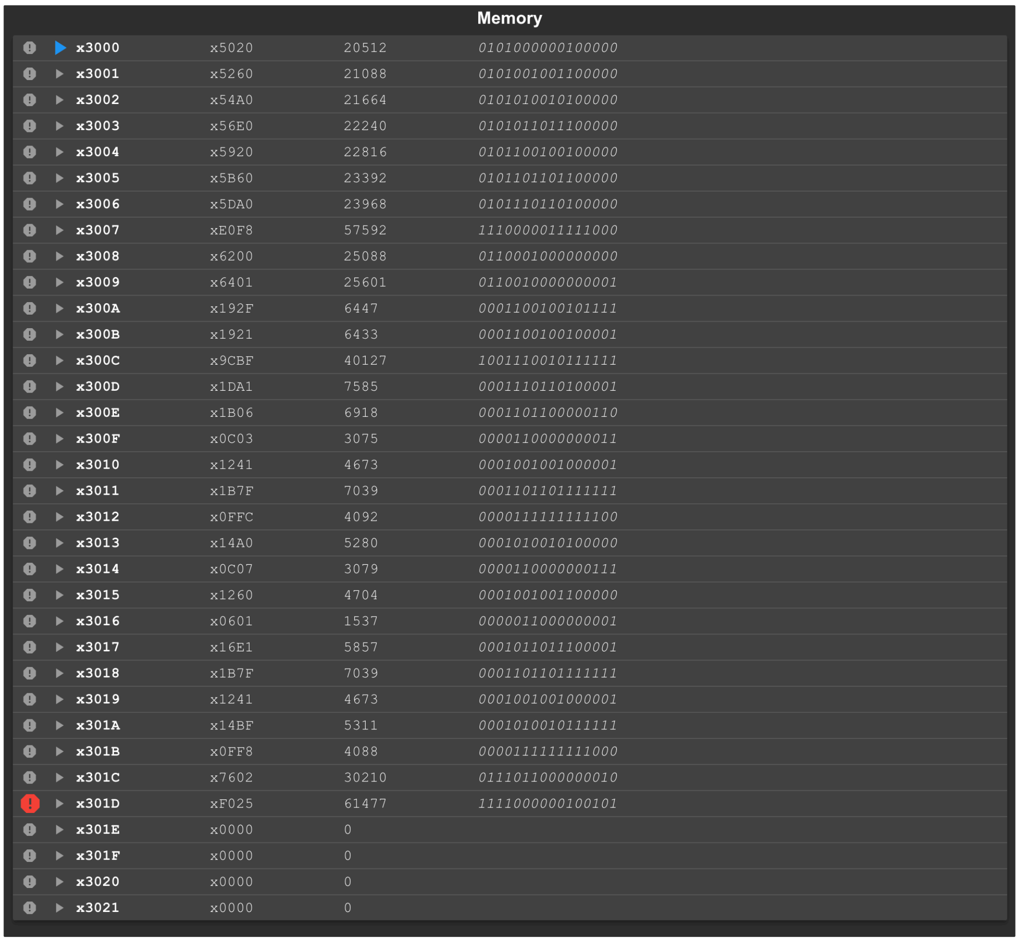
Before running the program, the registers are like this:



Registers

R0	x0000	0	
R1	x0000	0	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8002	32770	CC: Z
PC	x3000	12288	
MCR	x0000	0	

The console is just empty; And the Memory panel is like this:

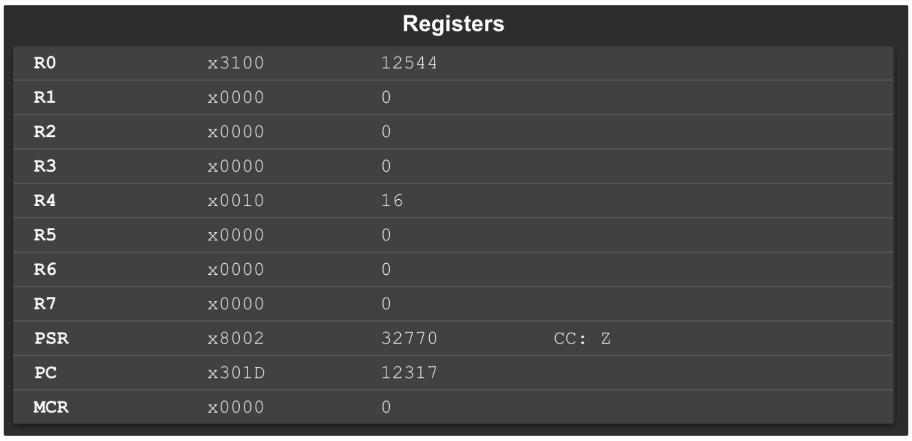


Memory

x3000	x5020	20512	0101000000100000
x3001	x5260	21088	0101001001100000
x3002	x54A0	21664	0101010010100000
x3003	x56E0	22240	0101011011100000
x3004	x5920	22816	0101100100100000
x3005	x5B60	23392	0101101101100000
x3006	x5DA0	23968	0101110110100000
x3007	xE0F8	57592	1110000011111000
x3008	x6200	25088	0110001000000000
x3009	x6401	25601	0110010000000001
x300A	x192F	6447	0001100100101111
x300B	x1921	6433	0001100100100001
x300C	x9CBF	40127	1001110010111111
x300D	x1DA1	7585	0001110110100001
x300E	x1B06	6918	0001101100000110
x300F	x0C03	3075	0000110000000011
x3010	x1241	4673	0001001001000001
x3011	x1B7F	7039	0001101101111111
x3012	x0FFC	4092	0000111111111100
x3013	x14A0	5280	0001010010100000
x3014	x0C07	3079	0000110000000111
x3015	x1260	4704	0001001001100000
x3016	x0601	1537	0000011000000001
x3017	x16E1	5857	0001011011100001
x3018	x1B7F	7039	0001101101111111
x3019	x1241	4673	0001001001000001
x301A	x14BF	5311	0001010010111111
x301B	x0FF8	4088	0000111111111000
x301C	x7602	30210	0111011000000010
x301D	xF025	61477	1111000000100101
x301E	x0000	0	
x301F	x0000	0	
x3020	x0000	0	
x3021	x0000	0	

8. Result

After the program is executed, the register panel is shown below:



Registers

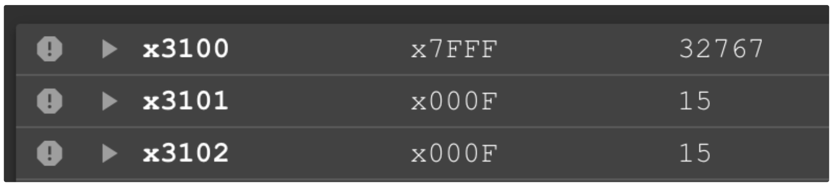
R0	x3100	12544	
R1	x0000	0	
R2	x0000	0	
R3	x0000	0	
R4	x0010	16	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8002	32770	CC: Z
PC	x301D	12317	
MCR	x0000	0	

The PC is at x301D.

Say A is 13 and B is 3. Now if I go to the memory x3102, I can find that the number stored is 2, which is exactly what it supposed to be.

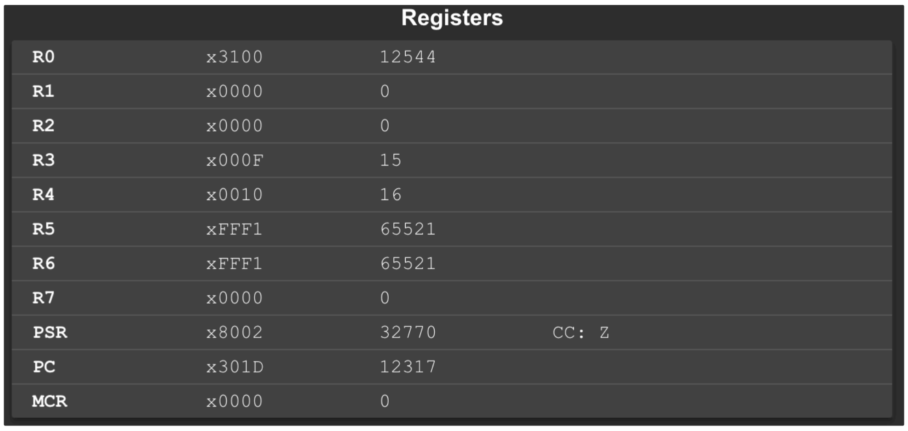
Take another instance: A = 32767 and B = 15.

The rightful result is shown below:



x3100	x7FFF	32767
x3101	x000F	15
x3102	x000F	15

And the registers are:



Registers

R0	x3100	12544	
R1	x0000	0	
R2	x0000	0	
R3	x000F	15	
R4	x0010	16	
R5	xFFF1	65521	
R6	xFFF1	65521	
R7	x0000	0	
PSR	x8002	32770	CC: Z
PC	x301D	12317	
MCR	x0000	0	

The End