Lab/Tutorial 1
The XM23p ISA, XM23p Assembler, and XM23 Emulator
Design, Implementation and Testing Document

Prepared for: Dr. Larry Hughes

Abdulla Sadoun B00900541

Table of Contents

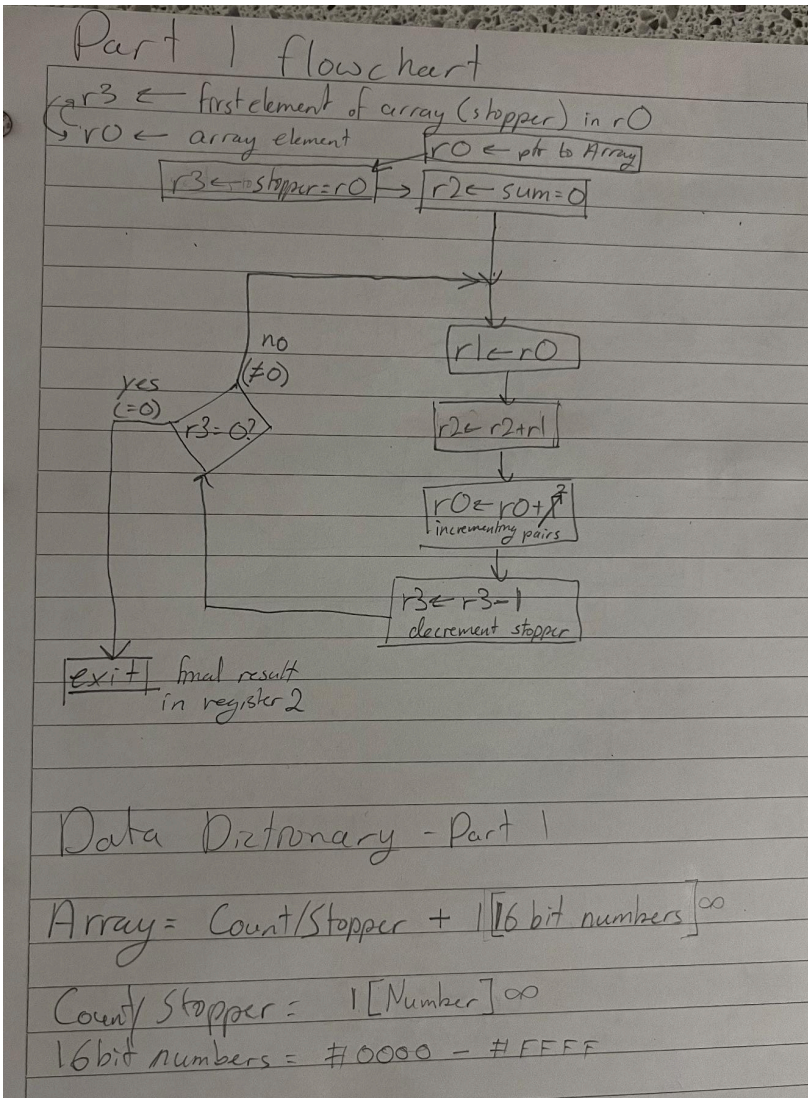
Problem Introduction

Statement of Purpose

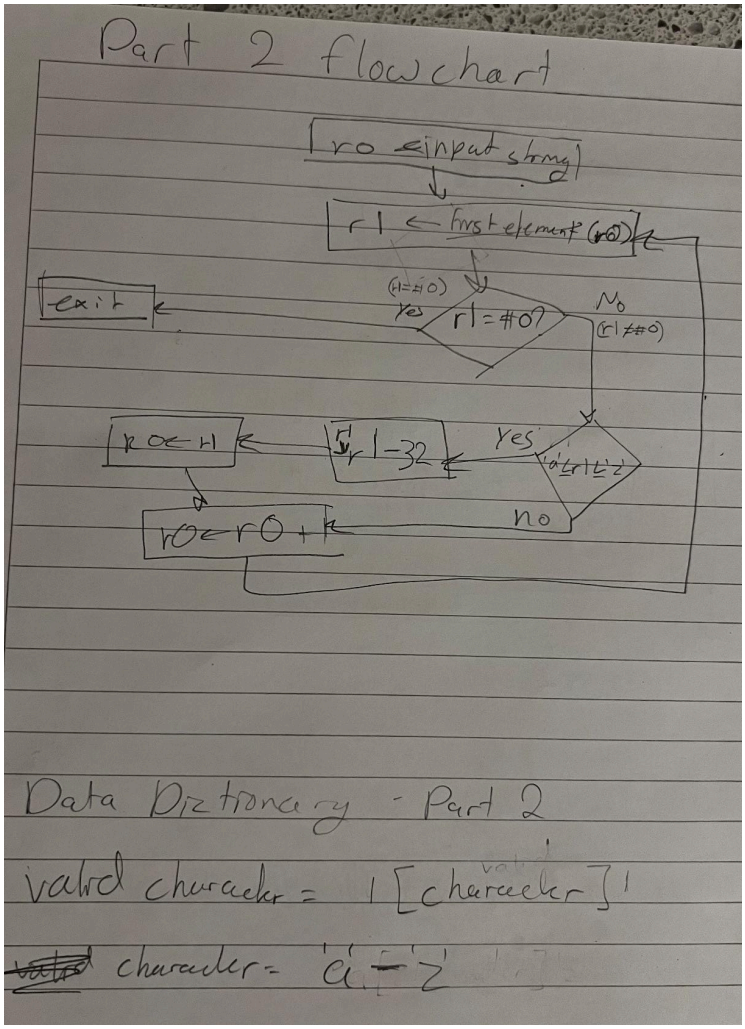
The purpose of this tutorial is to get familiar with using the XM23 ISA by writing .asm directives as well as assembling them and executing them on the XM23 emulator. We will then view how it changes the memory and registers in the cpu within the emulator.

Design:

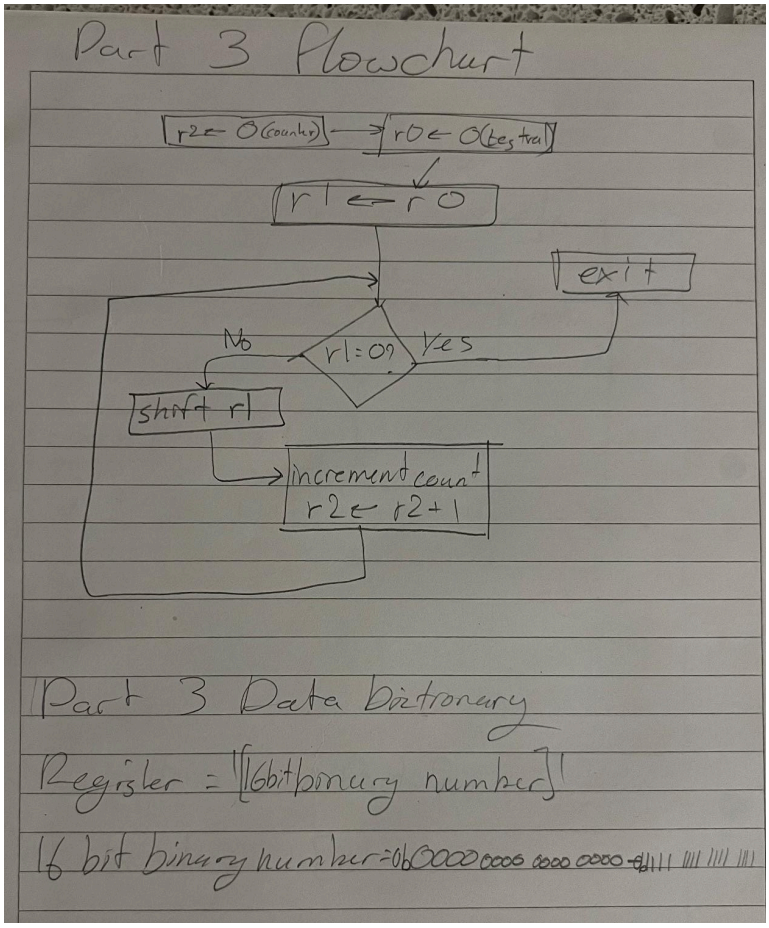
Part 1:



Part 2:


Part 2 flowchart
r0 ← input string
r1 ← first element (r0)
(r1≠0)
yes
r1 = #0?
No
(r1≠0)
exit
r0 ← r1
r1 - 32
yes
no
r0 ← r0 + 1
Data Dictionary - Part 2
valid character = ![character]!
character = 'a' - 'z'

Part 3:



Implementation:

Part 1:

```
;
; Sum an array of 16-bit numbers
; ECED 3403
; 15 May 24
;
        CODE
        org     #1000
;
;
Main    movlz   Array,R0        ; r0=Address of the array
        ld      R0,R3           ; load stopper into r3
```

```
        add     #2,R0           ; move r0 to the next element (first element to be summed)
increment by 2 as bytes are in pairs
        mov     R1,R2           ; setting r2 as sum register and making it 0
;
loop    ld      R0,R1           ; load the array's element into r1
        add     R1,R2           ; Add the element to the sum
        add     #2,R0           ; Increment R0 to point to the next element in the array
;
; check if stopper is 0 to stop summing
;
        sub     #1,R3           ; stopper - 1
        bz      Done            ; end loop if stopper is 0
;
        bra     loop            ; continue adding
;
; adding complete, result are in r2
;
Done
;
; Finished - busy wait
;
BWait   bra     BWait
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Data space
;
        DATA
        org     #40
;
; the array of integers used:
;
Array   word    #5              ; (5=stopper in r3)
        word    #1000
        word    #2000
        word    #3000
    word        #4000
    word        #5000
;
; no store for result they remained in register
;
        end     Main
```

Part 2:

```
;
; LowerToUpper ASCII converter
; ECED 3403
; 15 May 24
;
NUL equ #0
;
        CODE
        org     #1000
;
Main    movlz   InStr,R0        ; r0<-input string address
;
loop    ld.b    R0,R1           ; r1<-array element
        cmp.b   NUL,R1                  ; NUL char. check
        bz      Done    ; skip loop if NUL found
;
; load constants into registers for comparison
movlz   'a',R2  ; used to check within range
movlz   'z',R3  ; ""
movlz   $32,R4; 32 which offsets lower/upper in ascii
;
; check if lower (must be between a and z)
cmp.b   R2,R1   ; compare element with a
blt     NextChar        ; skip if less that a
cmp.b   R3,R1   ; compare element with z
bge     NextChar        ; skip if greater than z
;
; convert current element to upper
        sub     R4,R1   ; subtract offset to get upper
        st.b    R1,R0           ; save new char.
;
NextChar
        add     #1,R0           ; increment bit for next element check
        bra     loop            ; loop through directives
;
Done
;
; Finished - busy wait
;
BWait   bra     BWait
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;
; Data space
;
        DATA
        org     #40
;
; input string
;
InStr   ascii   "hello world!"  byte    NUL
;
;       org     #80
;OutStr ascii   "################################################"
;
;
        end     Main
```

Part 3:

```
;
; set bits in a register counter
; ECED 3403
; 15 May 24
;
NUL equ #0
;
        CODE
        org     #1000
;
; Initialize the register whose bits we want to count
; Assume the register R1 contains the value for which we need to count set bits
;
Main    movlz   $0,R0
        ld      R0,R1     ; testing register r1
        movlz   $0,R2           ; set r2 as 0 (counter)
   movlz $1,R4
   movlz $0,R0
;
CountBits
        movlz   #16,R3          ; r3 = 16 which are the amount of bits in r1
;
CountLoop
        cmp     R0,R1           ; check if r1 is zero
        bz      Done            ; if r1 is zero no bits are set
```

```
    add  R4,R2              ; Increment the count of set bits
    sra R1          ; shift r1 by 1
    sub R4,R3       ; another check to stop looping at 16bits
    cmp R0,R3       ; ""
    bz Done
    bra CountLoop   ; if program got here then it should repeat loop
Done
;
BWait
            bra     BWait                       ;
;
;.....................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Data space
;
            DATA
            org     #40
;
            end     Main
```

Testing - Part 1

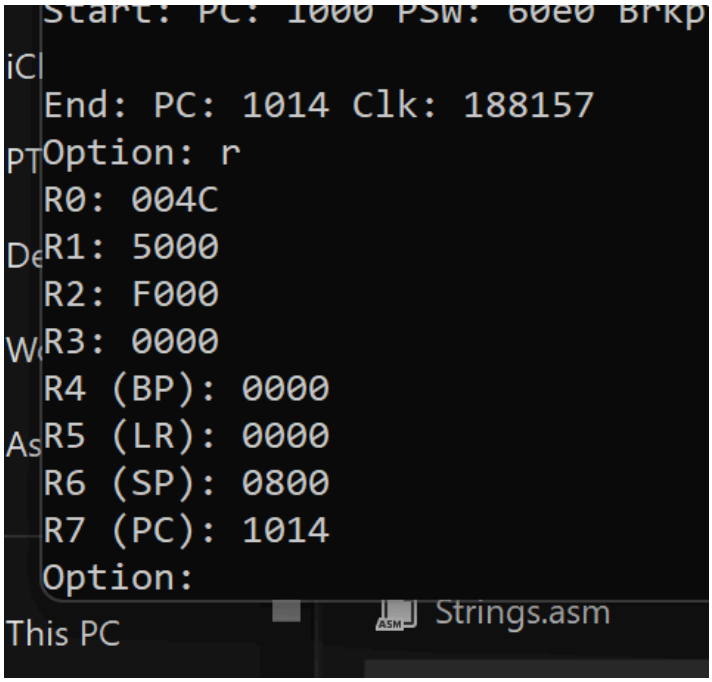
Test 1: Valid input test.

Purpose/Objective: The purpose of this test is to check if the software accepts and correctly processes valid inputs.

Test Configuration: I have altered the data section to make the array have the following elements:

```
39      DATA
40      org #40
41  ;
42  ; the array of integers used:
43  ;
44  Array   word    #5      ; (5=stopper in r3)
45      word    #1000
46      word    #2000
47      word    #3000
48      word    #4000
49      word    #5000
50  ;
51  ; no store for result they remained in register
52  ;
53      end Main
```

Expected Results: The Program shouldn't have a problem executing and the correct result would be displayed in r2 where it should be.
Actual Results: The actual result was as expected and the software did not have any problems executing.



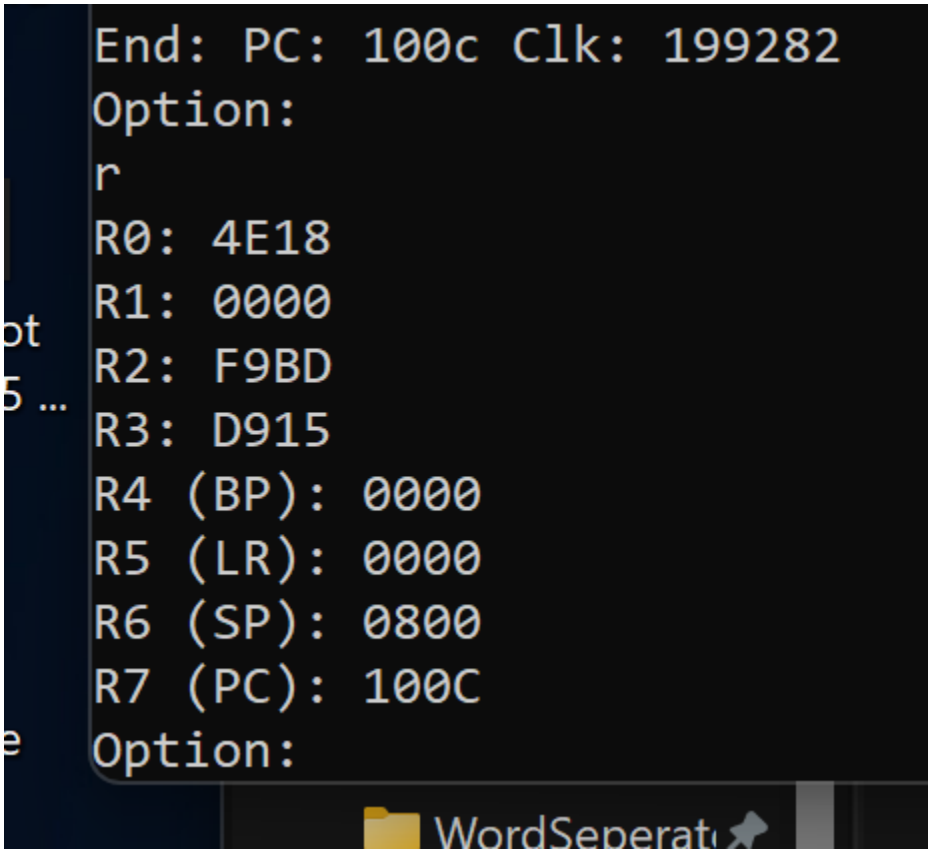
Pass/Fail: Pass

Test 2: 0 stopper test

Purpose/Objective: The purpose of this test is to see how the program would react if the stopper or the limit for the summer is equal to 0
Test Configuration: I have kept the same array as the one used in the previous test but I have changed the first element to be 0.

```
40      org #40
41  ;
42  ; the array of integers used:
43  ;
44  Array   word    #0      ; (5=stopper in r3)
45      word    #1000
46      word    #2000
47      word    #3000
48      word    #4000
49      word    #5000
50  ;
51  ; no store for result they remained in register
52  ;
53      end Main
```

Expected Results: The value of r2 should remain at 0 as the loop should not run.
Actual Results: The program has displayed an incorrect value in r2.



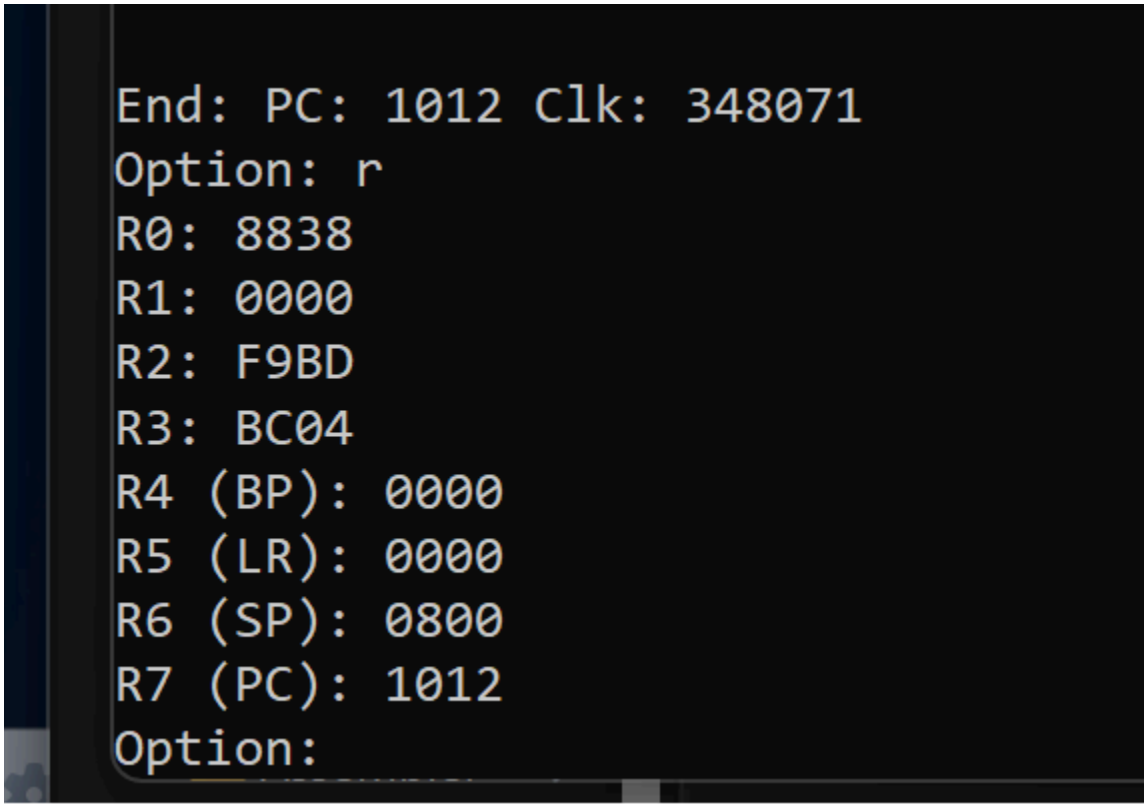
Pass/Fail: Fail

Test 3: Negative stopper test

Purpose/Objective: The purpose of this test is to see how the program would react if the first element of the array or the stopper for summing is altered.
Test Configuration: I have changed the first element of the array to be -1.

```
42  ; the array of integers used:
43  ;
44  Array   word    $-1     ; (5=stopper in r3)
45      word    #1000
46      word    #2000
47      word    #3000
48      word    #4000
49      word    #5000
50  ;
51  ; no store for result they remained in register
52  ;
53      end Main
```

Expected Results: I have not accounted for this case when documenting the program so I think it would not loop and a similar value to the one obtained in the zero test will be obtained.
Actual Results: The program has failed to display the correct answer as expected and did in fact show the same value for r2 as it did in the previous test like expected.



Pass/Fail: Fail

Testing - Part 2

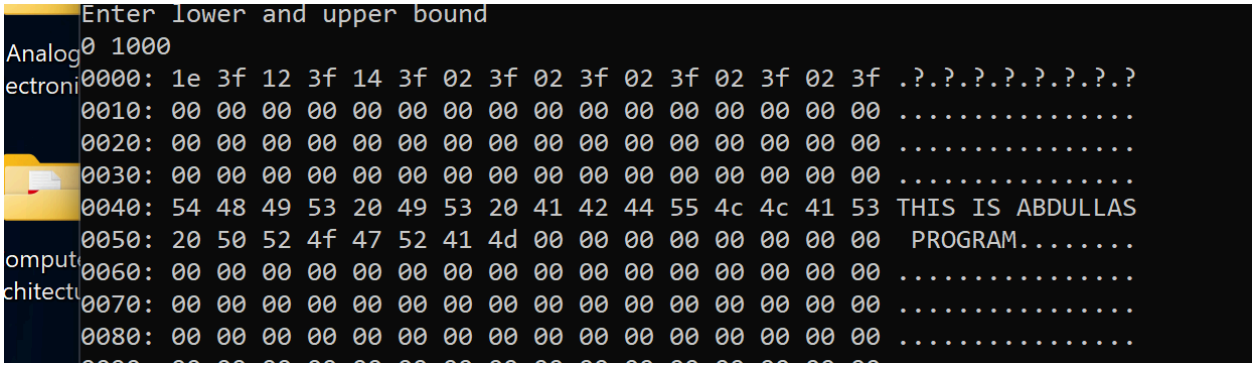
Test 1: Valid input test.

Purpose/Objective: The purpose of this test is to see whether the program behaves like it should when a valid input string is used.

Test Configuration: I have entered a simple string in lowercase letters to be changed into upper case letters.

```
43  ;
44  ; Data space
45  ;
46      DATA
47      org #40
48  ;
49  ; input string
50  ;
51  InStr   ascii   "this is abdullas program"  byte    NUL
52  ;
53  ;   org #80
54  ;OutStr ascii   "##########################################"
55  ;
56  ;
57      end Main
58
```

Expected Results: The program should have no problem converting the input to uppercase output.
Actual Results: The program successfully converted the lowercase input string to uppercase characters.



Pass/Fail: Pass

Test 2: String Contains Numbers

Purpose/Objective: The purpose of this test to see how the program treats numbers.

Test Configuration: I have added a string that contains a number in the program.

```
46      DATA
47      org #40
48  ;
49  ; input string
50  ;
51  InStr   ascii   "this is part 2"
52  byte    NUL
53  ;
54  ;   org #80
55  ;OutStr ascii   "##########################################"
56  ;
57  ;
58      end Main
59
```

Expected Results: The program should successfully only alter the lower case character as it checks if they are within the a-z range before converting them.
Actual Results: The program successfully converted as expected

```
Enter lower and upper bound
0 100
0000: 1e 3f 12 3f 14 3f 02 3f 02 3f 02 3f 02 3f .?.?.?.?.?.?.?.?
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0040: 54 48 49 53 20 49 53 20 50 41 52 54 20 32 00 00 THIS IS PART 2..
0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Pass/Fail: Pass

Test 3: Special Characters String (!@#$)

Purpose/Objective: The purpose of this test is to see how the program would treat special characters in the string.
Test Configuration: I have added a new string that has special character and I will run the program and see what would happen to them after the conversion.
Expected Results: The program should ignore them as I have added a check to make sure it only converts a character if it is lowercase in the a-z range.

Actual Results: The program has successfully only converted the lowercase characters as expected.

```
Enter lower and upper bound
0 100
0000: 1e 3f 12 3f 14 3f 02 3f 02 3f 02 3f 02 3f .?.?.?.?.?.?.?.?
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
0040: 54 48 45 53 45 20 41 52 45 20 4d 59 20 43 48 41 THESE ARE MY CHA
0050: 52 41 43 54 45 52 53 20 21 40 23 24 00 00 00 00 RACTERS !@#$....
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Pass/Fail: Pass

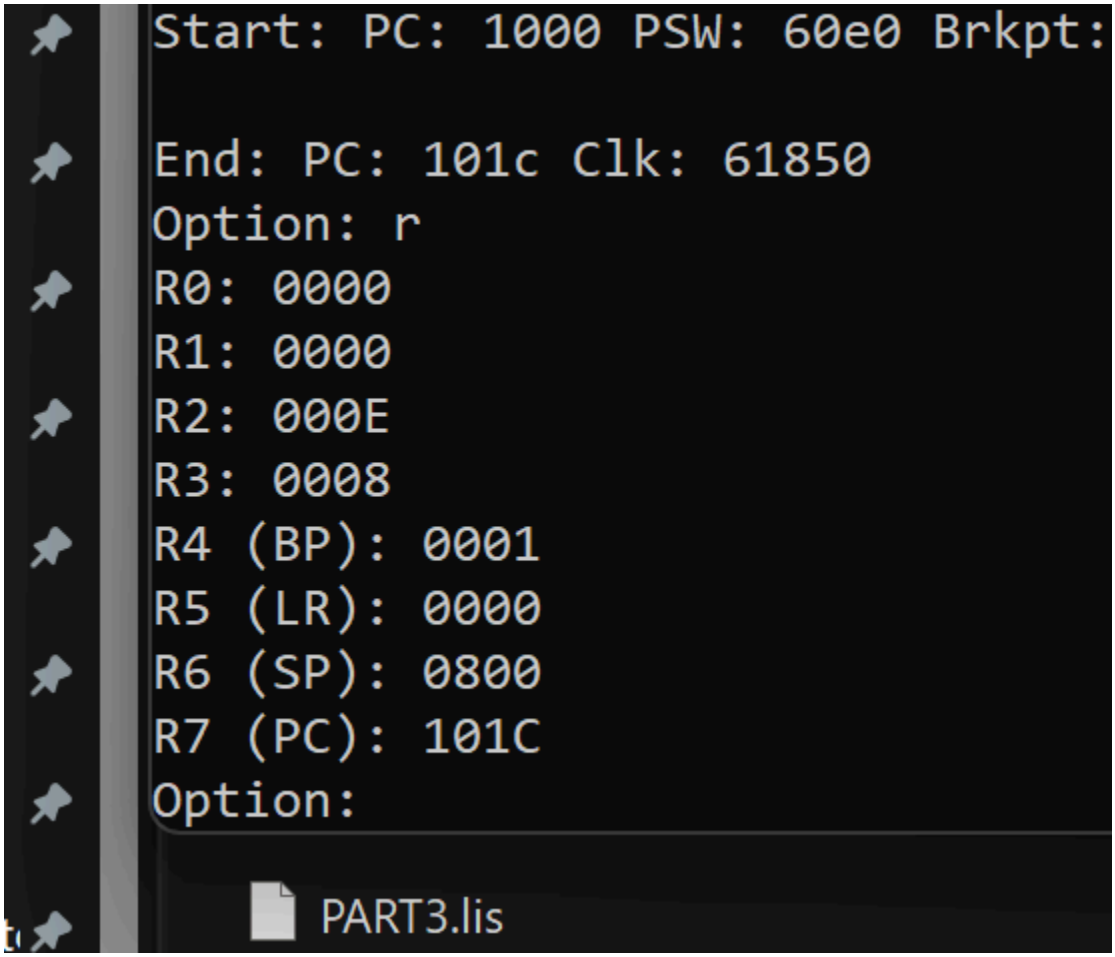
Testing - Part 3

Test 1: Correct output check

Purpose/Objective: The purpose of this test is to see if the program would correct the correct number of set bits in a register as it should
Test Configuration: I have set register 1 to be 0 meaning it would have no set bits (0000000000000000)

```
9       org #1000
10  ;
11  ; Initialize the register whose bits we want to count
12  ; Assume the register R1 contains the value for which we need to count set
13  ;
14  Main    movlz   $0,R0
15      ld  R0,R1       ; testing register r1
16      movlz   $0,R2       ; set r2 as 0 (counter)
17      movlz $1,R4
18      movlz $0,R0
19  ;
20  CountBits
21      movlz   #16,R3      ; r3 = 16 which are the amount of bits in r1
```

Expected Results: The program should have 0 in r2 as there are no set bits
Actual Results: The program has failed to properly execute.



Pass/Fail: Fail