Screenshots

1. While Loop

Here, Loop starts from 0 till it is equal to 5.

Instructions for While Loop:

MOVE R1 0

MOVE R2 1

MOVE R3 5

MOVE R7 1

MOVE R8 0

JMP Label2

Label1 Alu_op add R7 R8 R8

Alu_op add R2 R1 R1

Label2 CMPQ R1 R3

JLE Label1

Alu_op add R2 R1 R1

Inside loop, adding 1 every time to register R8. So, once coming out of the loop, R8 value will be 6 i.e. 6 times iteration done.

```
***** JUMP Operation *****


***** Jump if Less than or equal Function *****
Out of the Loop


00011000010001000001000010000000


***** ALU Operation *****


***** Addition Function *****


 ----- General Purpose Registers -----
 R[0] = 00000000000000000000000000000000 (0)
 R[1] = 00000000000000000000000000000111 (7)
 R[2] = 00000000000000000000000000000001 (1)
 R[3] = 00000000000000000000000000000101 (5)
 R[4] = 00000000000000000000000000000000 (0)
 R[5] = 00000000000000000000000000000000 (0)
 R[6] = 00000000000000000000000000000000 (0)
 R[7] = 00000000000000000000000000000001 (1)
 R[8] = 00000000000000000000000000000110 (6)
 R[9] = 00000000000000000000000000000000 (0)
 R[10] = 00000000000000000000000000000000 (0)
 R[11] = 00000000000000000000000000000000 (0)
 R[12] = 00000000000000000000000000000000 (0)
 R[13] = 00000000000000000000000000000110 (6)
 R[14] = 11111111111111111111111111111111 (-1)
 R[15] = 00000000000000000000000000001000 (8)
```

2. Do while Loop

Instructions for do while loop:

MOVE R4 0

MOVE R5 1

MOVE R6 5

MOVE R7 2

MOVE R8 1

Label3 Alu_op add R5 R4 R4

Alu_op mul R7 R8 R8

CMPQ R4 R6

JLE Label3

Alu_op add R5 R4 R4

Here, Loop starts from 0 till it is equal to 5. Inside loop, we are performing multiplication of register R8 with 2 and so final result of R8 is 64 means it iterates inside loop for 6 times.

```
***** Jump if Less than or equal Function *****
Out of the Loop


00011000010010100100001000000000


***** ALU Operation *****


***** Addition Function *****


 ----- General Purpose Registers -----
 R[0] = 00000000000000000000000000000000 (0)
 R[1] = 00000000000000000000000000000111 (7)
 R[2] = 00000000000000000000000000000000 (0)
 R[3] = 00000000000000000000000000000000 (0)
 R[4] = 00000000000000000000000000000111 (7)
 R[5] = 00000000000000000000000000000001 (1)
 R[6] = 00000000000000000000000000000101 (5)
 R[7] = 00000000000000000000000000000010 (2)
 R[8] = 00000000000000000000000001000000 (64)
 R[9] = 00000000000000000000000000000000 (0)
 R[10] = 00000000000000000000000000000000 (0)
 R[11] = 00000000000000000000000000000000 (0)
 R[12] = 00000000000000000000000000000000 (0)
 R[13] = 00000000000000000000000000000110 (6)
 R[14] = 11111111111111111111111111111111 (-1)
 R[15] = 00000000000000000000000000001000 (8)
```

3. For Loop

Instructions for For Loop:

MOVE R1 0

MOVE R2 1

MOVE R3 5

MOVE R7 2

MOVE R8 1

JMP Label4

Label5 Alu_op mul R7 R8 R8

Alu_op add R2 R1 R1

Label4 CMPQ R1 R3

JLE Label5

Alu_op add R2 R1 R1

Similarly, here as well, loop starts from 0 and iterates till equal to 5. Inside loop, we are performing multiplication of register R8 with 2 and so final result of R8 is 64 means it iterates inside loop for 6 times.

```
***** Jump if Less than or equal Function *****
Out of the Loop


00011000010001000001000010000000


***** ALU Operation *****


***** Addition Function *****


 ----- General Purpose Registers -----
 R[0] = 00000000000000000000000000000000 (0)
 R[1] = 00000000000000000000000000000111 (7)
 R[2] = 00000000000000000000000000000001 (1)
 R[3] = 00000000000000000000000000000101 (5)
 R[4] = 00000000000000000000000000000000 (0)
 R[5] = 00000000000000000000000000000000 (0)
 R[6] = 00000000000000000000000000000000 (0)
 R[7] = 00000000000000000000000000000010 (2)
 R[8] = 00000000000000000000000001000000 (64)
 R[9] = 00000000000000000000000000000000 (0)
 R[10] = 00000000000000000000000000000000 (0)
 R[11] = 00000000000000000000000000000000 (0)
 R[12] = 00000000000000000000000000000000 (0)
 R[13] = 00000000000000000000000000000110 (6)
 R[14] = 11111111111111111111111111111111 (-1)
 R[15] = 00000000000000000000000000001000 (8)
  ----- Special Purpose Registers -----
 pc = 0x60b660 (556)     sp = 0x60acf8 (255)     fp = 0
 flag = 0        mar = 0x60b658
 mdr = 00011000010001000001000010000000 (407113856)
```

4. LEA – Load Effective Address

Instruction usage:

- LEA DestReg1 offset SrcReg1

  Here, Effective Address will be loaded into a Register in a binary format.

1. Offset is 2

```
***** Load Effective Address Operation *****
Address 0x60c510 loaded into Register in binary format


 ----- General Purpose Registers -----
 R[0] = 00000000000000000000000000000000 (0)
 R[1] = 00000000000000000000000000000111 (7)
 R[2] = 00000000000000000000000000000000 (0)
 R[3] = 00000000000000000000000000000000 (0)
 R[4] = 00000000000000000000000000000000 (0)
 R[5] = 00000000000000000000000000000000 (0)
 R[6] = 00000000000000000000000000000000 (0)
 R[7] = 00000000000000000000000000000010 (2)
 R[8] = 00000000000000000000000000000000 (0)
 R[9] = 00000000000000000000000000000000 (0)
 R[10] = 00000000000000000000000000000000 (0)
 R[11] = 00000000000000000000000000000000 (0)
 R[12] = 00000000000000000000000000000000 (0)
 R[13] = 00000000000000000000000000000110 (6)
 R[14] = 11111111111111111111111111111111 (-1)
 R[15] = 00000000000000000000000000001000 (8)
  ----- Special Purpose Registers -----
 pc = 0x60b540 (520)     sp = 0x60acf8 (255)     fp = 0
 flag = 0         mar = 0x60c510
 mdr = 00000000000000000000000000000111 (7)

 ----- Focused Instruction Memory Dump -----
 Memory Address           Instruction Data
 0x60b500        00001000010000000010000000000000
 0x60b520        01101001110001000010000000000000
 0x60b540        00000000000000000000000000000000
 0x60b560        00000000000000000000000000000000
 0x60b580        00000000000000000000000000000000
 0x60b5a0        00000000000000000000000000000000
 0x60b5c0        00000000000000000000000000000000
 0x60b5e0        00000000000000000000000000000000
 0x60b600        00000000000000000000000000000000
 0x60b620        00000000000000000000000000000000

 ----- Focused Data Memory Dump -----
 Memory Address           Data
 0x60c500        00000000000000000000000000000111 (7)
```

2. Offset is 0

```
***** Load Effective Address Operation *****
Address 0x60c500 loaded into Register in binary format


 ----- General Purpose Registers -----
 R[0] = 00000000000000000000000000000000 (0)
 R[1] = 00000000000000000000000000000111 (7)
 R[2] = 00000000000000000000000000000000 (0)
 R[3] = 00000000000000000000000000000000 (0)
 R[4] = 00000000000000000000000000000000 (0)
 R[5] = 00000000000000000000000000000000 (0)
 R[6] = 00000000000000000000000000000000 (0)
 R[7] = 00000000000000000000000000000000 (0)
 R[8] = 00000000000000000000000000000000 (0)
 R[9] = 00000000000000000000000000000000 (0)
 R[10] = 00000000000000000000000000000000 (0)
 R[11] = 00000000000000000000000000000000 (0)
 R[12] = 00000000000000000000000000000000 (0)
 R[13] = 00000000000000000000000000000110 (6)
 R[14] = 11111111111111111111111111111111 (-1)
 R[15] = 00000000000000000000000000001000 (8)
  ----- Special Purpose Registers -----
 pc = 0x60b540 (520)     sp = 0x60acf8 (255)     fp = 0
 flag = 0         mar = 0x60c500
 mdr  = 00000000000000000000000000000111 (7)
```

5. Set if Equal

Here, if R11==R12 then it will set R1.

```
***** SET Operation *****


***** Set if Equal Function *****


 ----- General Purpose Registers -----
 R[0] = 00000000000000000000000000000000 (0)
 R[1] = 00000000000000000000000000000001 (1)
 R[2] = 00000000000000000000000000000000 (0)
 R[3] = 00000000000000000000000000000000 (0)
 R[4] = 00000000000000000000000000000000 (0)
 R[5] = 00000000000000000000000000000000 (0)
 R[6] = 00000000000000000000000000000000 (0)
 R[7] = 00000000000000000000000000000000 (0)
 R[8] = 00000000000000000000000000000000 (0)
 R[9] = 00000000000000000000000000000000 (0)
 R[10] = 00000000000000000000000000000000 (0)
 R[11] = 00000000000000000000000000000001 (1)
 R[12] = 00000000000000000000000000000001 (1)
 R[13] = 00000000000000000000000000000110 (6)
 R[14] = 11111111111111111111111111111111 (-1)
 R[15] = 00000000000000000000000000001000 (8)
```

6. Set if Not Equal

Here, if R11!=R12, then set R1. So, both R11 and R12 are not equal and so R1 is set to 1.

```
***** Set if Not Equal Function *****


 ----- General Purpose Registers -----
 R[0] = 00000000000000000000000000000000 (0)
 R[1] = 00000000000000000000000000000001 (1)
 R[2] = 00000000000000000000000000000000 (0)
 R[3] = 00000000000000000000000000000000 (0)
 R[4] = 00000000000000000000000000000000 (0)
 R[5] = 00000000000000000000000000000000 (0)
 R[6] = 00000000000000000000000000000000 (0)
 R[7] = 00000000000000000000000000000000 (0)
 R[8] = 00000000000000000000000000000000 (0)
 R[9] = 00000000000000000000000000000000 (0)
 R[10] = 00000000000000000000000000000000 (0)
 R[11] = 00000000000000000000000000000001 (1)
 R[12] = 00000000000000000000000000000010 (2)
 R[13] = 00000000000000000000000000000110 (6)
 R[14] = 11111111111111111111111111111111 (-1)
 R[15] = 00000000000000000000000000001000 (8)
  ----- Special Purpose Registers -----
```

Likewise, we have implemented other set instructions such as SETS, SETNS, SETG, SETGE, SETL and SETLE.

7. TEST condition:

Here, if R1 & R2 == 0 then set zero flag. So, both AND of R1 and R2 will be zero so zero flag is set to 1.

```
***** TEST Operation *****
Zero flag is 1


 ----- General Purpose Registers -----
 R[0] = 00000000000000000000000000000000 (0)
 R[1] = 00000000000000000000000000000001 (1)
 R[2] = 00000000000000000000000000000010 (2)
 R[3] = 00000000000000000000000000000000 (0)
```

8. Binary Search

Array → 10 11 12 13 14 15 16

Key → 11 to be searched

```
 0x60e300       00000000000000000000000000000010 (2)
 0x60e320       00000000000000000000000000000000 (0)
 0x60e340       00000000000000000000000000001011 (11)
 0x60e360       00000000000000000000000011111001 (249)
 0x60e380       00000000000000000000001011011100 (732)
 0x60e3a0       00000000000000000000000000000000 (0)
 0x60e3c0       00000000000000000000000000000000 (0)
 0x60e3e0       00000000000000000000000000000000 (0)
 0x60e400       00000000000000000000000000000000 (0)
 0x60e420       00000000000000000000000000001010 (10)
 0x60e440       00000000000000000000000000001011 (11)
 0x60e460       00000000000000000000000000001100 (12)
 0x60e480       00000000000000000000000000001101 (13)
 0x60e4a0       00000000000000000000000000001110 (14)
 0x60e4c0       00000000000000000000000000001111 (15)
 0x60e4e0       00000000000000000000000000010000 (16)

--------------------------------------------------------------------------------
Incase of binary search, the index value found in the array matching the key (11) is 1
```

Key → 13

```
 0x60e340       00000000000000000000000000001101 (13)
 0x60e360       00000000000000000000000011111001 (249)
 0x60e380       00000000000000000000001011011100 (732)
 0x60e3a0       00000000000000000000000000000000 (0)
 0x60e3c0       00000000000000000000000000000000 (0)
 0x60e3e0       00000000000000000000000000000000 (0)
 0x60e400       00000000000000000000000000000000 (0)
 0x60e420       00000000000000000000000000001010 (10)
 0x60e440       00000000000000000000000000001011 (11)
 0x60e460       00000000000000000000000000001100 (12)
 0x60e480       00000000000000000000000000001101 (13)
 0x60e4a0       00000000000000000000000000001110 (14)
 0x60e4c0       00000000000000000000000000001111 (15)
 0x60e4e0       00000000000000000000000000010000 (16)

--------------------------------------------------------------------------------
Incase of binary search, the index value found in the array matching the key (13) is 3
```

Key → 55

```
 0x60e3a0       00000000000000000000000000000000 (0)
 0x60e3c0       00000000000000000000000000000000 (0)
 0x60e3e0       00000000000000000000000000000000 (0)
 0x60e400       00000000000000000000000000000000 (0)
 0x60e420       00000000000000000000000000001010 (10)
 0x60e440       00000000000000000000000000001011 (11)
 0x60e460       00000000000000000000000000001100 (12)
 0x60e480       00000000000000000000000000001101 (13)
 0x60e4a0       00000000000000000000000000001110 (14)
 0x60e4c0       00000000000000000000000000001111 (15)
 0x60e4e0       00000000000000000000000000010000 (16)

--------------------------------------------------------------------------------
Incase of binary search, the index value found in the array matching the key (55) is -1
```

Key → 12 (odd array)

Array → 10 11 12 13 14 15

```
0x60e380        00000000000000000000001011001100 (716)
0x60e3a0        00000000000000000000000000000000 (0)
0x60e3c0        00000000000000000000000000000000 (0)
0x60e3e0        00000000000000000000000000000000 (0)
0x60e400        00000000000000000000000000000000 (0)
0x60e420        00000000000000000000000000001010 (10)
0x60e440        00000000000000000000000000001011 (11)
0x60e460        00000000000000000000000000001100 (12)
0x60e480        00000000000000000000000000001101 (13)
0x60e4a0        00000000000000000000000000001110 (14)
0x60e4c0        00000000000000000000000000001111 (15)
0x60e4e0        00000000000000000000000000000000 (0)

-----------------------------------------------------------------------
Incase of binary search, the index value found in the array matching the key (12) is 2
```

9. Compare:

Instruction: CMPQ R2 R1

```
--------------------------------------------------
Flags: zf - 0, sf - 1, cf - 0, of -0
Incase of binary search, the index value found in t
floodlight@floodlight:~/Downloads$
```

Her R2 is less than R1 so signed flag is set.