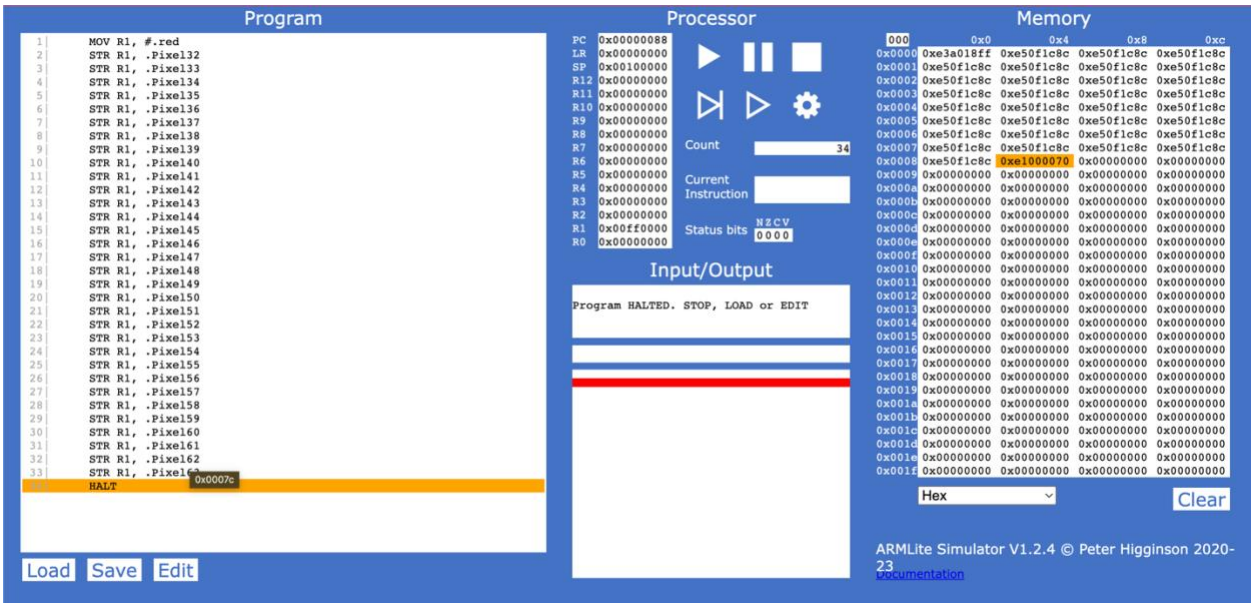
COS10004 – Computer System
Name: Phan Vũ – Student Id: 104222099
LAB 9

9.1.1

(a) Write a simple ARMlite assembly program that draws a single line of the same length across the second row (starting from the left-most column) in Low-res display mode.

```
MOV R1, #.red
STR R1, .Pixel32
STR R1, .Pixel33
STR R1, .Pixel34
STR R1, .Pixel35
STR R1, .Pixel36
STR R1, .Pixel37
STR R1, .Pixel38
STR R1, .Pixel39
STR R1, .Pixel40
STR R1, .Pixel41
STR R1, .Pixel42
STR R1, .Pixel43
STR R1, .Pixel44
STR R1, .Pixel45
STR R1, .Pixel46
STR R1, .Pixel47
STR R1, .Pixel48
STR R1, .Pixel49
STR R1, .Pixel50
STR R1, .Pixel51
STR R1, .Pixel52
STR R1, .Pixel53
STR R1, .Pixel54
STR R1, .Pixel55
STR R1, .Pixel56
STR R1, .Pixel57
```

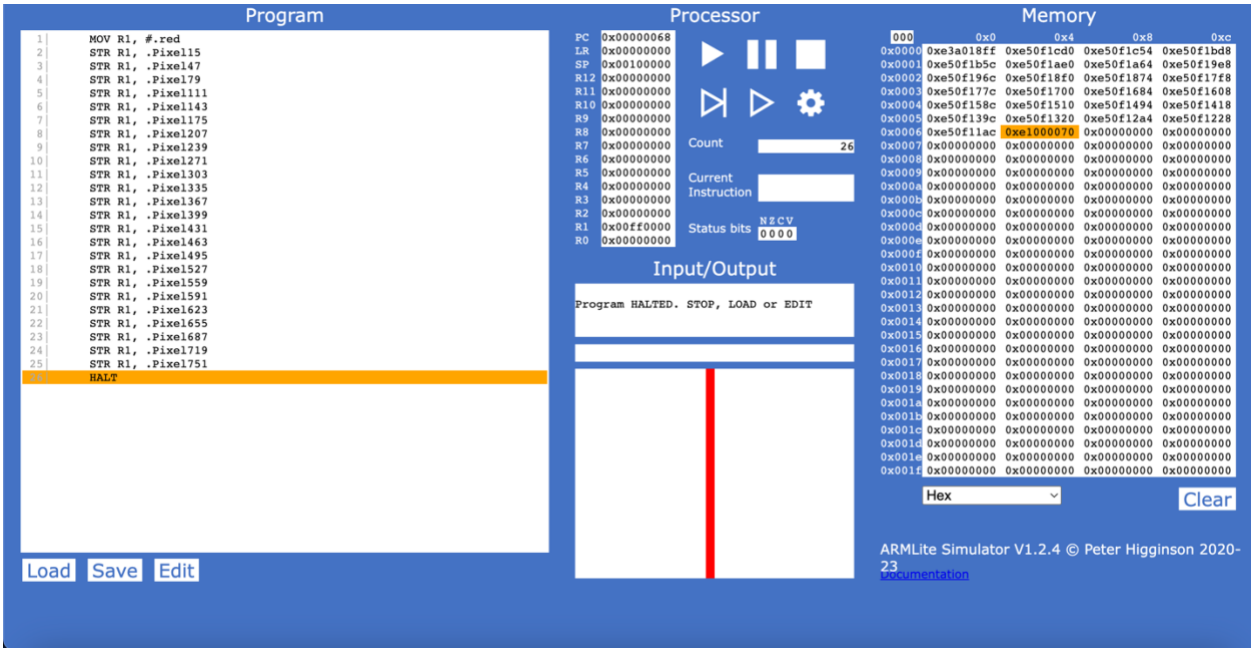
```
STR R1, .Pixel58
STR R1, .Pixel59
STR R1, .Pixel60
STR R1, .Pixel61
STR R1, .Pixel62
STR R1, .Pixel63
HALT
```



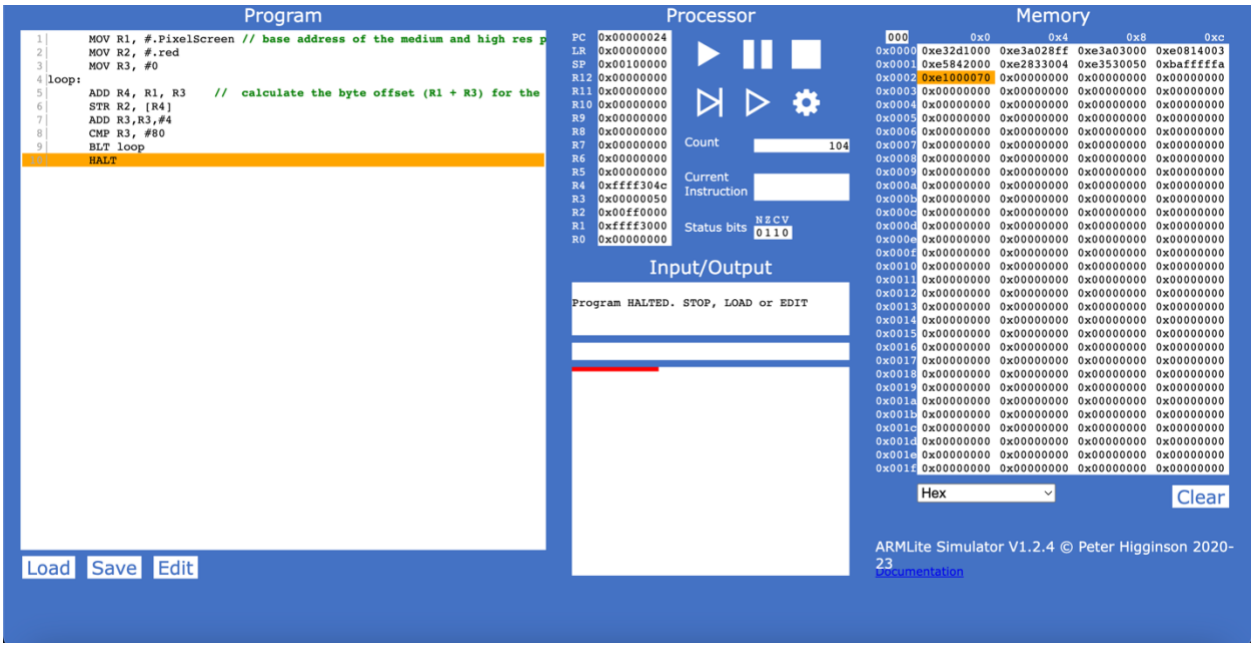
(b) Add to your assembly program code that draws a single line of the same length vertically, down the middle of the display in Low-res display mode

```
MOV R1, #.red
STR R1, .Pixel15
STR R1, .Pixel47
STR R1, .Pixel79
STR R1, .Pixel111
STR R1, .Pixel143
STR R1, .Pixel175
STR R1, .Pixel207
STR R1, .Pixel239
STR R1, .Pixel271
```

```
    STR R1, .Pixel303
    STR R1, .Pixel335
    STR R1, .Pixel367
    STR R1, .Pixel399
    STR R1, .Pixel431
    STR R1, .Pixel463
    STR R1, .Pixel495
    STR R1, .Pixel527
    STR R1, .Pixel559
    STR R1, .Pixel591
    STR R1, .Pixel623
    STR R1, .Pixel655
    STR R1, .Pixel687
    STR R1, .Pixel719
    STR R1, .Pixel751
    HALT
```



9.1.2



9.1.3

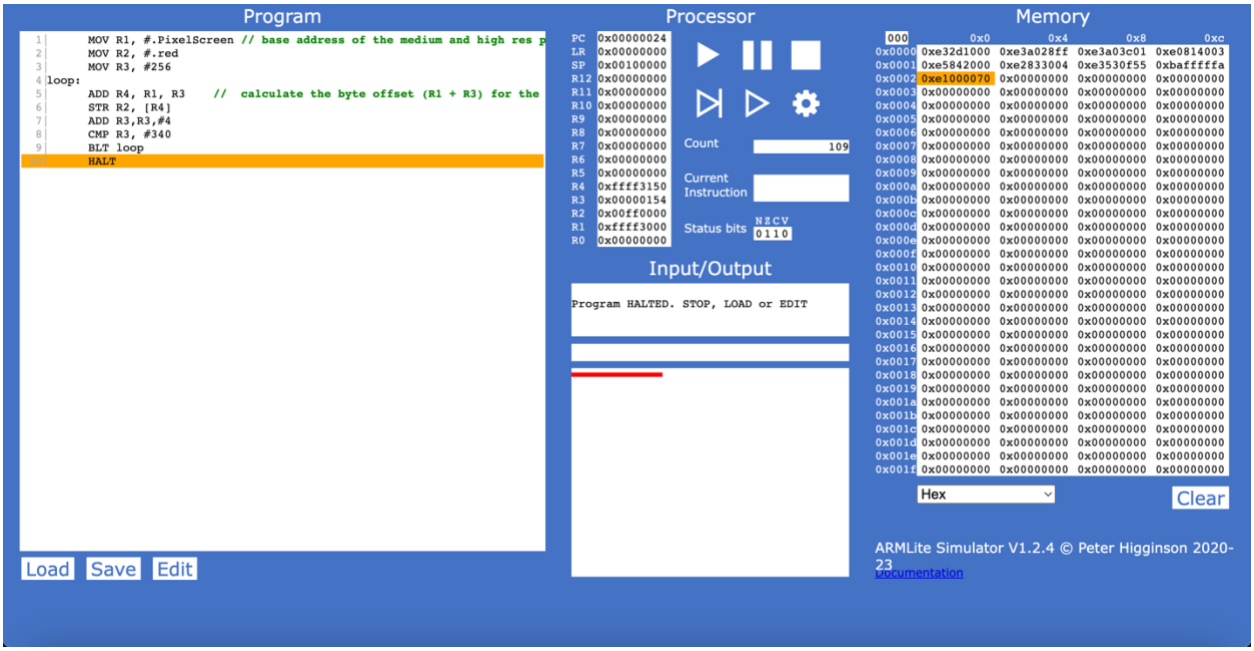
(a) Explain what specifically makes this code an example of indirect addressing? How is it using indirect addressing to draw each pixel?

- The given code is indirect addressing as it can store the value of R2 in [R4], which has a memory address and its value. The code adds 4 bytes every time to the pixel until it reaches 80 and moves data to the memory address 4 bytes, 32 bits. And R1 is the base address; if we add value in R3, it will add to the R1, which stores the adrress and can form the new pointer to the next pixel.

(b) Once you're confident to understand the code, modify the program so that it draws a line of the same length along the second row of the Mid-res display.

```
    MOV R1, #.PixelScreen // base address of the medium and high res pixel display memory
    MOV R2, #.red
    MOV R3, #256
loop:
```

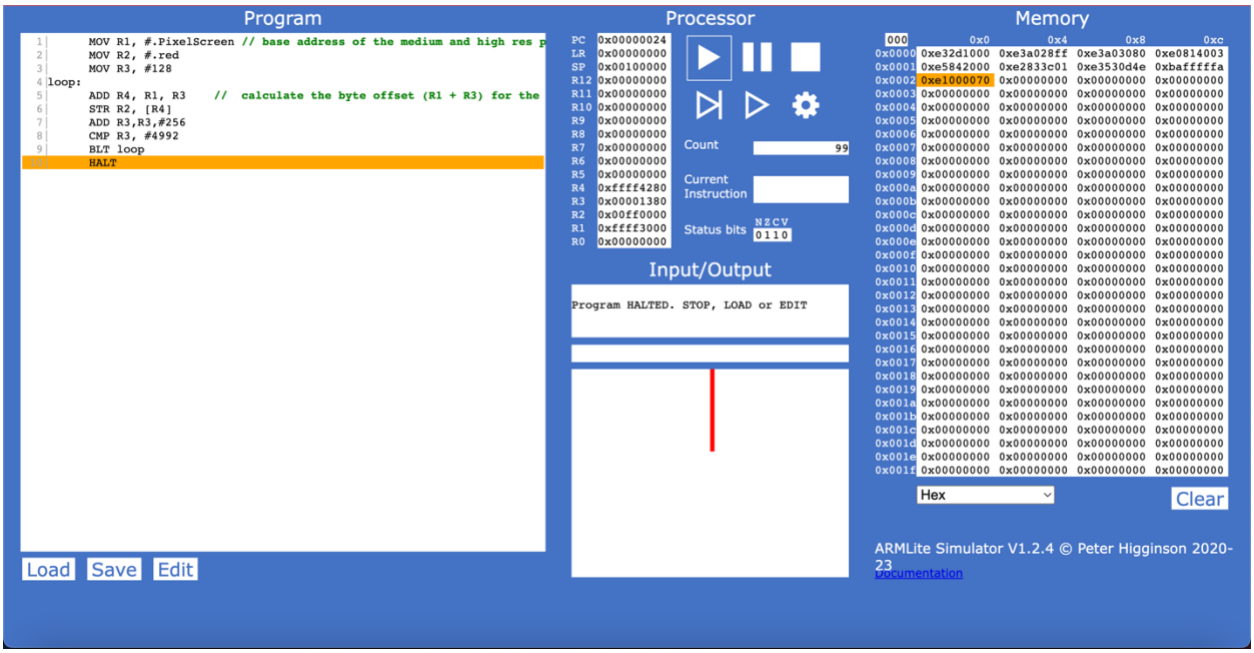
ADD R4, R1, R3 // calculate the byte offset (R1 + R3) for the next pixel and store new address in R4
STR R2, [R4]
ADD R3,R3,#4
CMP R3, #340
BLT loop
HALT



(c) Further modify your program so that it also draws a line of the same length vertically down the middle of the display.

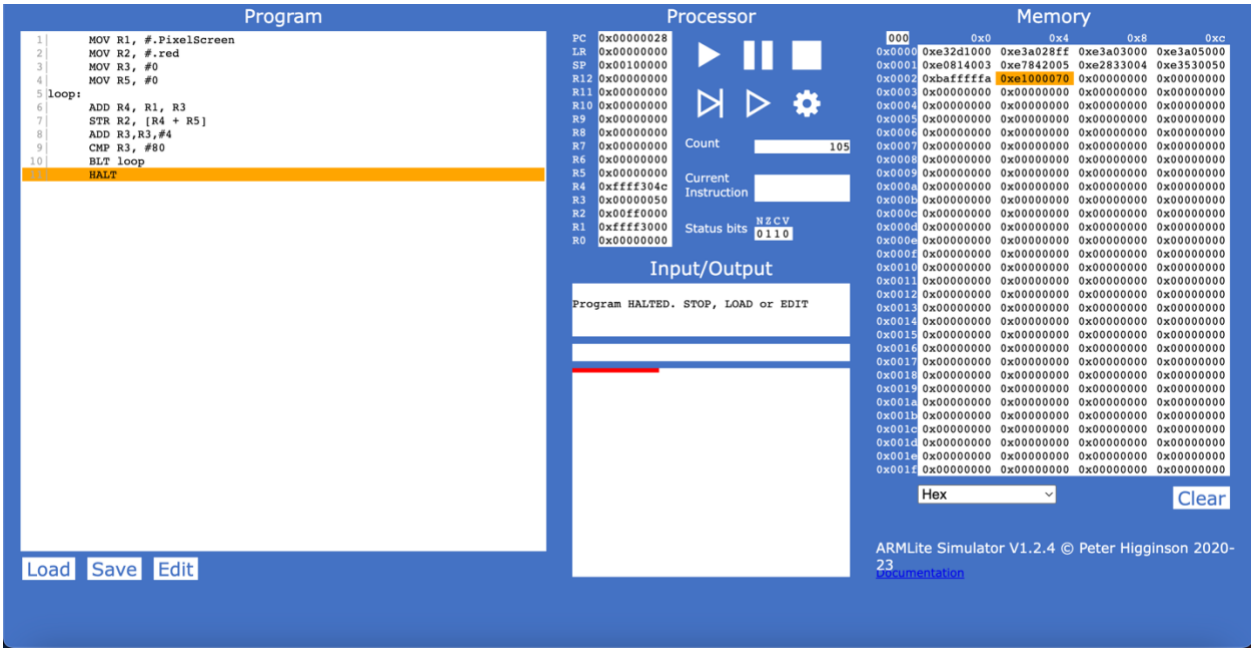
MOV R1, #.PixelScreen // base address of the medium and high res pixel display memory
MOV R2, #.red
MOV R3, #128
loop:
ADD R4, R1, R3 // calculate the byte offset (R1 + R3) for the next pixel and store new address in R4
STR R2, [R4]
ADD R3,R3,#256
CMP R3, #4992

BLT loop
HALT



9.2.1

MOV R1, #.PixelScreen
MOV R2, #.red
MOV R3, #0
MOV R5, #0
loop:
ADD R4, R1, R3
STR R2, [R4 + R5]
ADD R3,R3,#4
CMP R3, #80
BLT loop
HALT

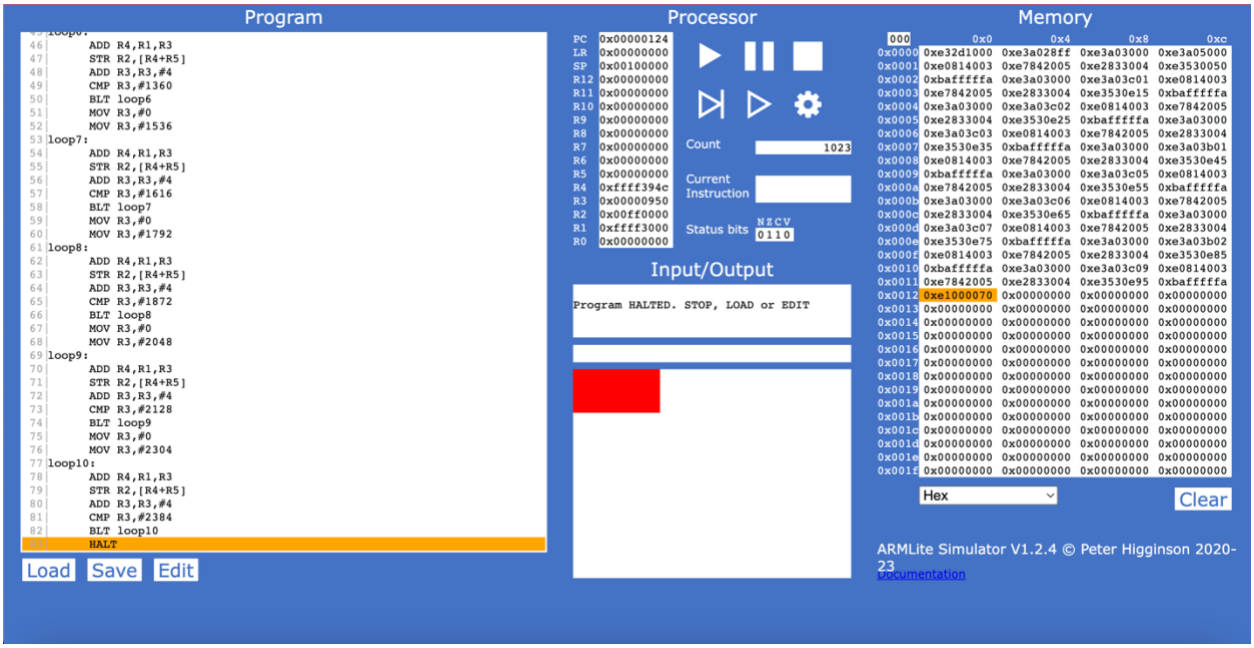


9.2.2

```
MOV R1, #.PixelScreen
      MOV R2, #.red
      MOV R3, #0
      MOV R5, #0
loop1:
      ADD R4, R1, R3
      STR R2,[R4+R5]
      ADD R3,R3,#4
      CMP R3, #80
      BLT loop1
      MOV R3, #0
      MOV R3, #256
loop2:
      ADD R4,R1,R3
      STR R2,[R4+R5]
      ADD R3,R3,#4
      CMP R3, #336
      BLT loop2
      MOV R3,#0
```

```
    MOV R3,#512
loop3:
    ADD R4,R1,R3
    STR R2,[R4+R5]
    ADD R3,R3,#4
    CMP R3, #592
    BLT loop3
    MOV R3,#0
    MOV R3,#768
loop4:
    ADD R4,R1,R3
    STR R2,[R4+R5]
    ADD R3,R3,#4
    CMP R3,#848
    BLT loop4
    MOV R3,#0
    MOV R3,#1024
loop5:
    ADD R4,R1,R3
    STR R2, [R4+R5]
    ADD R3,R3,#4
    CMP R3,#1104
    BLT loop5
    MOV R3,#0
    MOV R3,#1280
loop6:
    ADD R4,R1,R3
    STR R2,[R4+R5]
    ADD R3,R3,#4
    CMP R3,#1360
    BLT loop6
    MOV R3,#0
    MOV R3,#1536
loop7:
    ADD R4,R1,R3
    STR R2,[R4+R5]
```

```
        ADD R3,R3,#4
        CMP R3,#1616
        BLT loop7
        MOV R3,#0
        MOV R3,#1792
loop8:
        ADD R4,R1,R3
        STR R2,[R4+R5]
        ADD R3,R3,#4
        CMP R3,#1872
        BLT loop8
        MOV R3,#0
        MOV R3,#2048
loop9:
        ADD R4,R1,R3
        STR R2,[R4+R5]
        ADD R3,R3,#4
        CMP R3,#2128
        BLT loop9
        MOV R3,#0
        MOV R3,#2304
loop10:
        ADD R4,R1,R3
        STR R2,[R4+R5]
        ADD R3,R3,#4
        CMP R3,#2384
        BLT loop10
        HALT
```



9.3.1

(a) The above code defines an array of 10 32 bit integers. What is the purpose of the .Align 256 instruction ?

- Ensure the next instruction is aligned with a divisible word address by 256. Usually, the number must be a multiple of 4 for word addressing, while any number can be used for in-byte addressing.

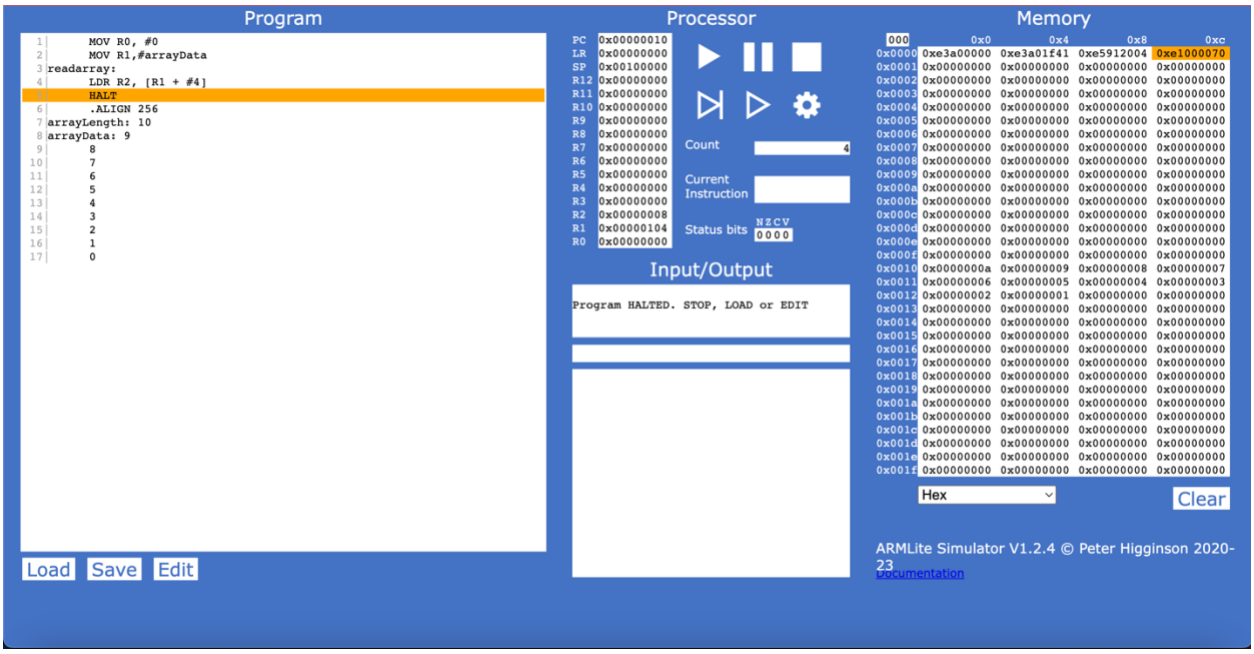
(b) Add a line of code to the above to read the 5th value of the array to register R0 (i.e,. it should use indirect addressing to access the 5th cell in the array)

```
   MOV R1,#arrayData
   LDR R2, [R1 + #20]
   HALT
   .ALIGN 256
arrayLength: 10
arrayData: 9
      8
      7
```

```
      6
      5
      4
      3
      2
      1
      0
```

(c) Now modify your code so that the index to read from in the array is provided in R1.

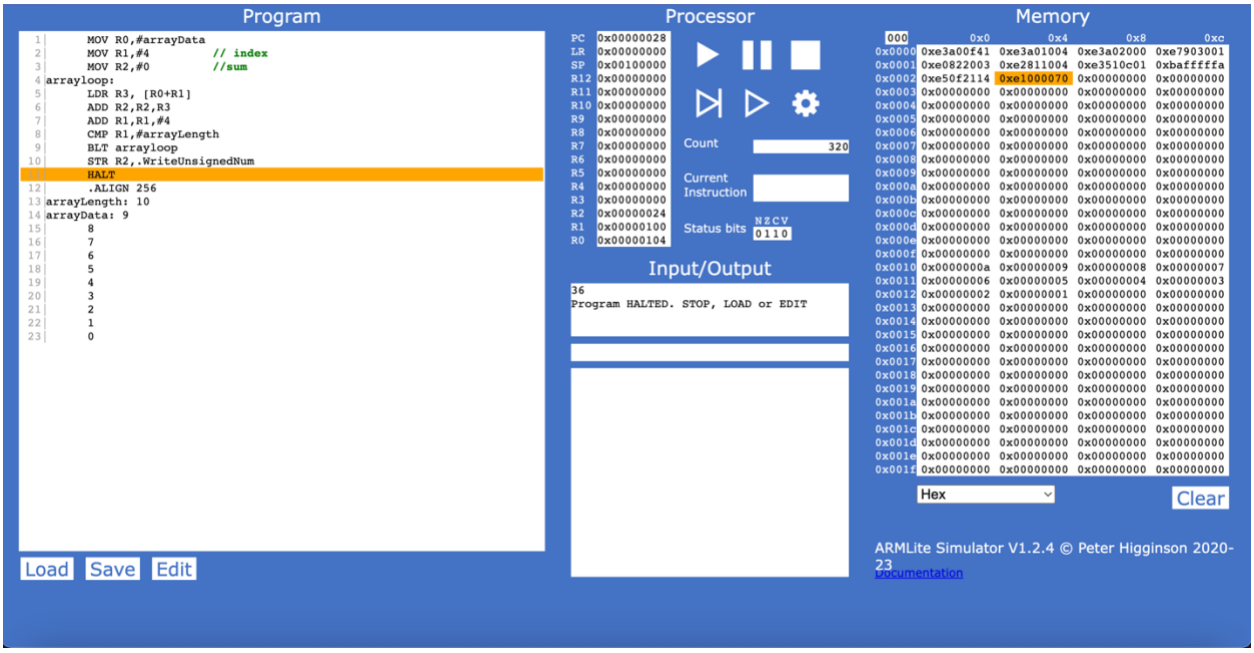
```
   MOV R0, #0
   MOV R1,#arrayData
readarray:
   LDR R2, [R1 + #4]
   HALT
   .ALIGN 256
arrayLength: 10
arrayData: 9
      8
      7
      6
      5
      4
      3
      2
      1
      0
```



9.3.2

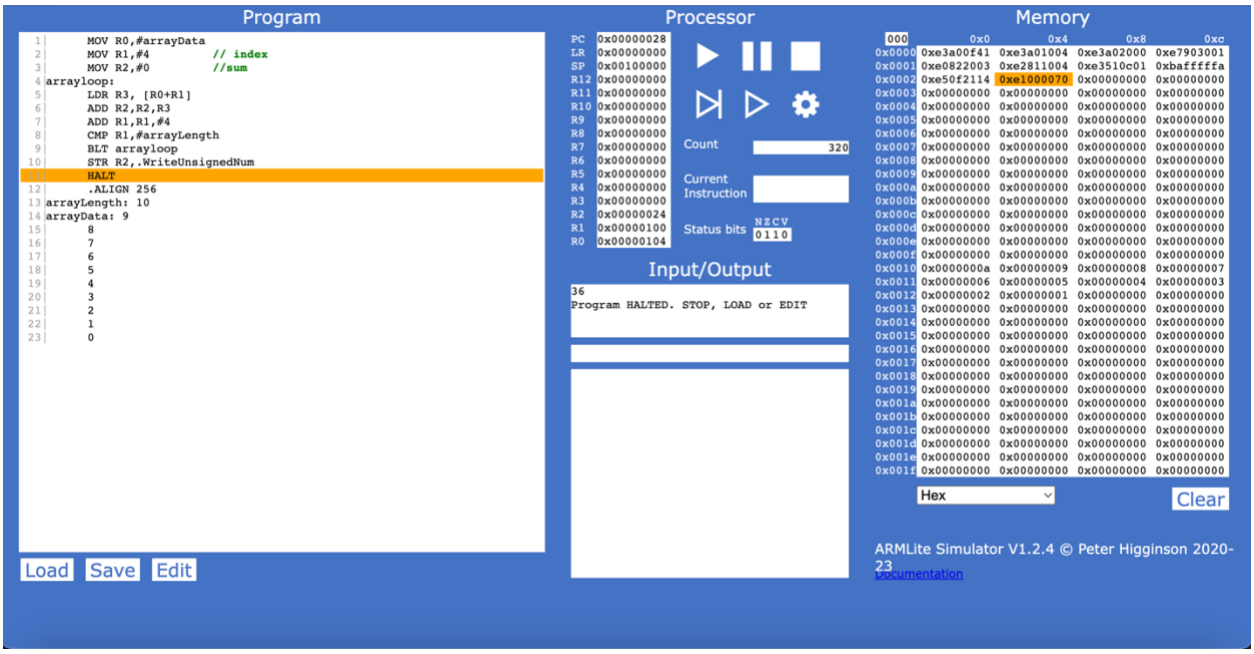
```
MOV R0,#arrayData
    MOV R1,#4       // index
    MOV R2,#0       //sum
arrayloop:
    LDR R3, [R0+R1]
    ADD R2,R2,R3
    ADD R1,R1,#4
    CMP R1,#arrayLength
    BLT arrayloop
    STR R2,.WriteUnsignedNum
    HALT
    .ALIGN 256
arrayLength: 10
arrayData: 9
    8
    7
    6
    5
    4
```

3
2
1
0



9.3.3

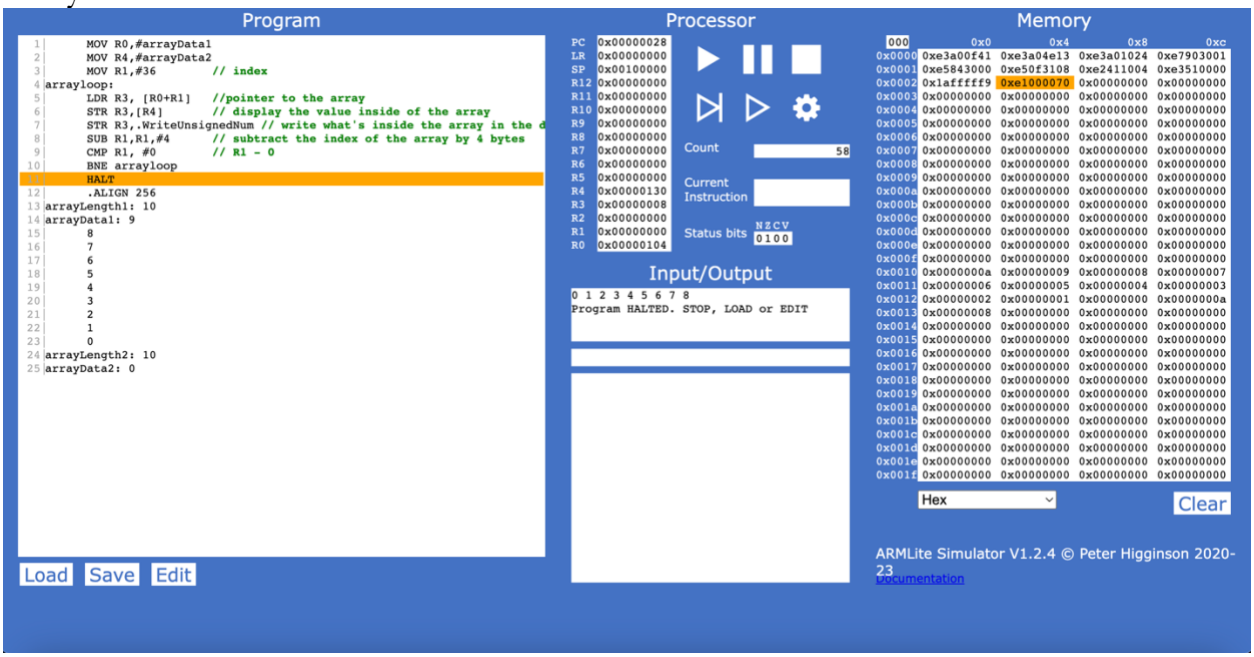
We basically used the same thing from ***9.3.2***



9.4.1

```
MOV R0,#arrayData1
    MOV R4,#arrayData2
    MOV R1,#36        // index
arrayloop:
    LDR R3, [R0+R1]   //pointer to the array
    STR R3,[R4]       // display the value inside of the array
    STR R3,.WriteUnsignedNum // write what's inside the array in the
display
    SUB R1,R1,#4      // subtract the index of the array by 4 bytes
    CMP R1, #0        // R1 - 0
    BNE arrayloop
    HALT
    .ALIGN 256
arrayLength1: 10
arrayData1: 9
    8
    7
    6
    5
```

4
3
2
1
0
arrayLength2: 10
arrayData2: 0



9.4.2

MOV R0,#arrayData1
MOV R1,#36 // index
arrayloop:
LDR R3, [R0+R1] //pointer to the array
STR R3,.WriteUnsignedNum // write what's inside the array in the display
SUB R1,R1,#4 // subtract the index of the array by 4 bytes
CMP R1, #0 // R1 - 0
BNE arrayloop
HALT
.ALIGN 256
arrayLength1: 10

arrayData1: 9

8

7

6

5

4

3

2

1

0