

COS10004 – Computer System

Name: Phan Vĩ – Student Id: 104222099

LAB 9

9.1.1

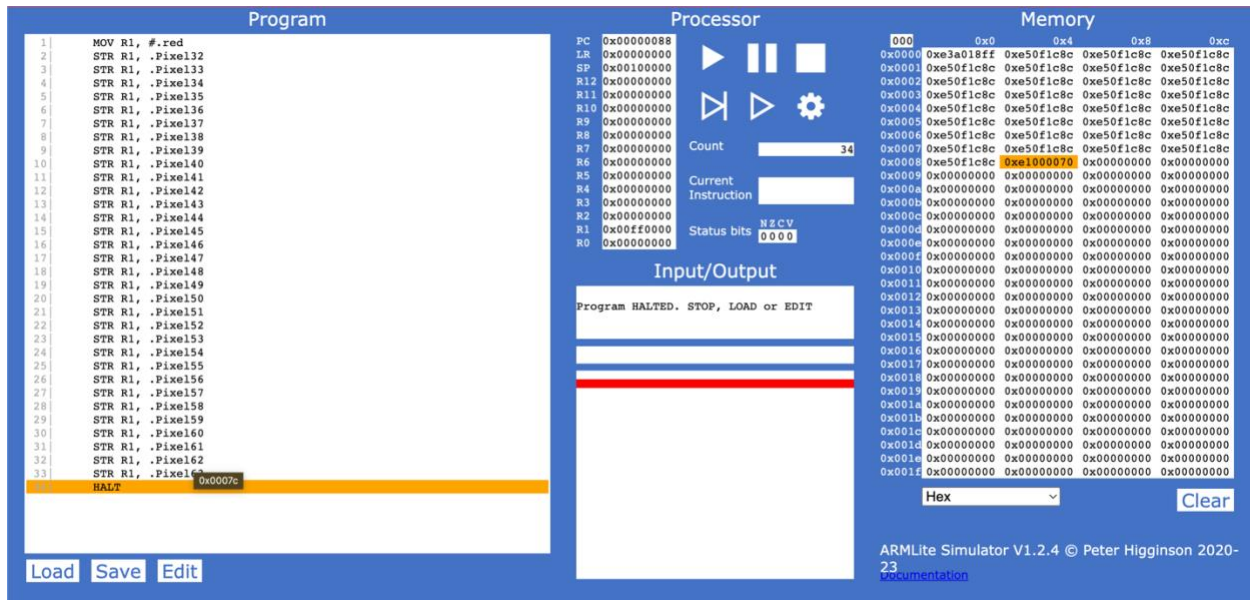
(a) Write a simple ARMLite assembly program that draws a single line of the same length across the second row (starting from the left-most column) in Low-res display mode.

```
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

```

The screenshot displays the ARMLite Simulator V1.2.4 interface, which is divided into three main sections: Program, Processor, and Memory.

Program Panel: This panel shows the assembly code being executed. The code consists of 25 lines, starting with `MOV R1, #.red` and ending with `HALT`. The `HALT` instruction is highlighted in orange, indicating it is the current instruction being executed.

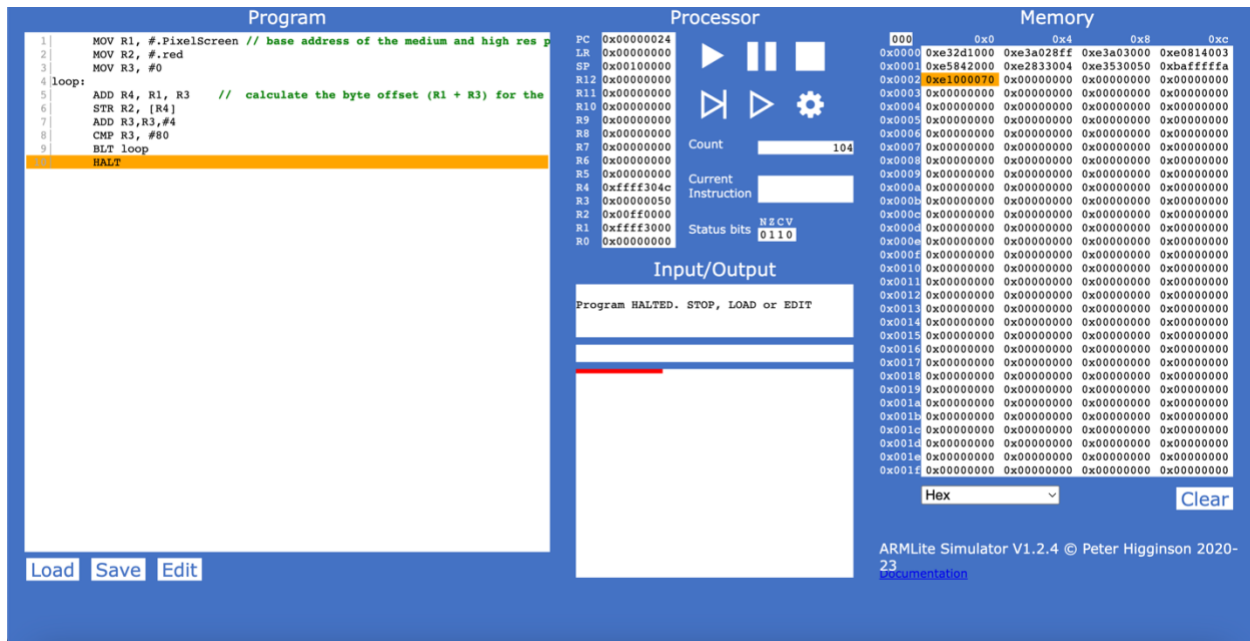
Processor Panel: This panel displays the state of the processor. It includes a register file with 16 registers (R0-R15) and their current values. The `PC` (Program Counter) is set to `0x00000068`. The `Count` is 26. The `Current Instruction` is `HALT`. The `Status bits` are `NECV 0000`.

Memory Panel: This panel shows the memory contents. It displays a hex dump of memory addresses from `0x0000` to `0x001f`. The memory is mostly zero, with some non-zero values at the beginning, such as `0xe3a018ff` at `0x0000`.

Input/Output Panel: This panel is currently empty, showing the text "Program HALTED. STOP, LOAD or EDIT".

Footer: The footer indicates the simulator version is V1.2.4, © Peter Higginson 2020, and provides a link to the documentation.

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 address 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

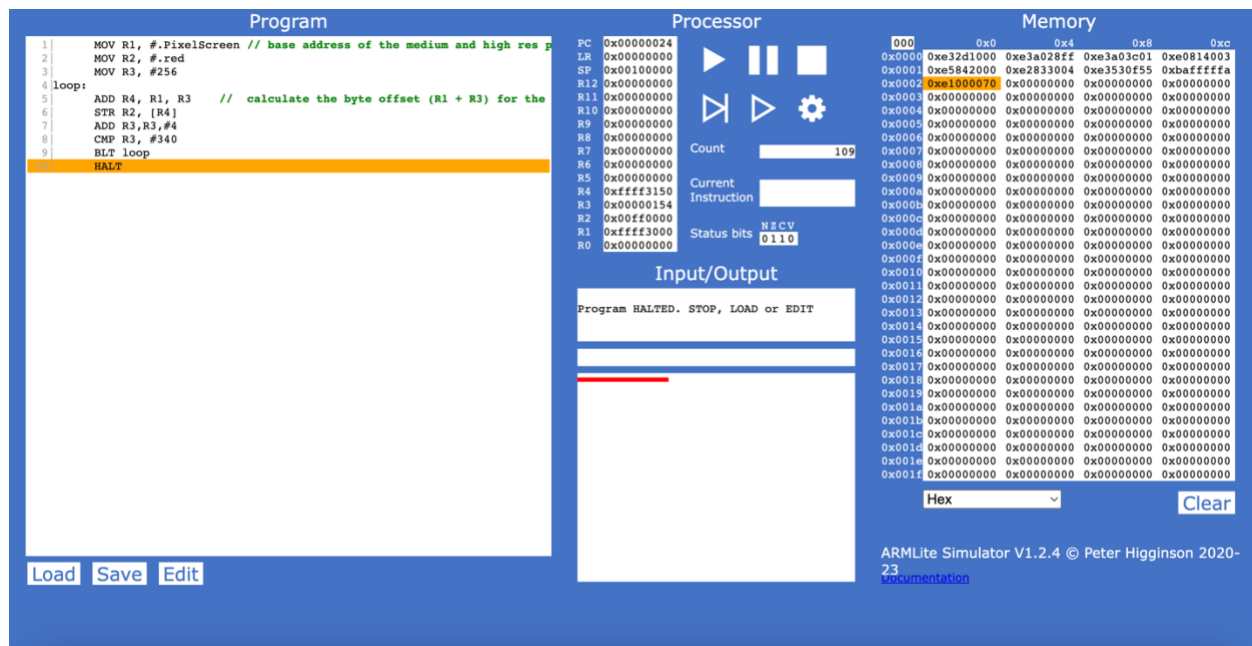
STR R2, [R4]

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:

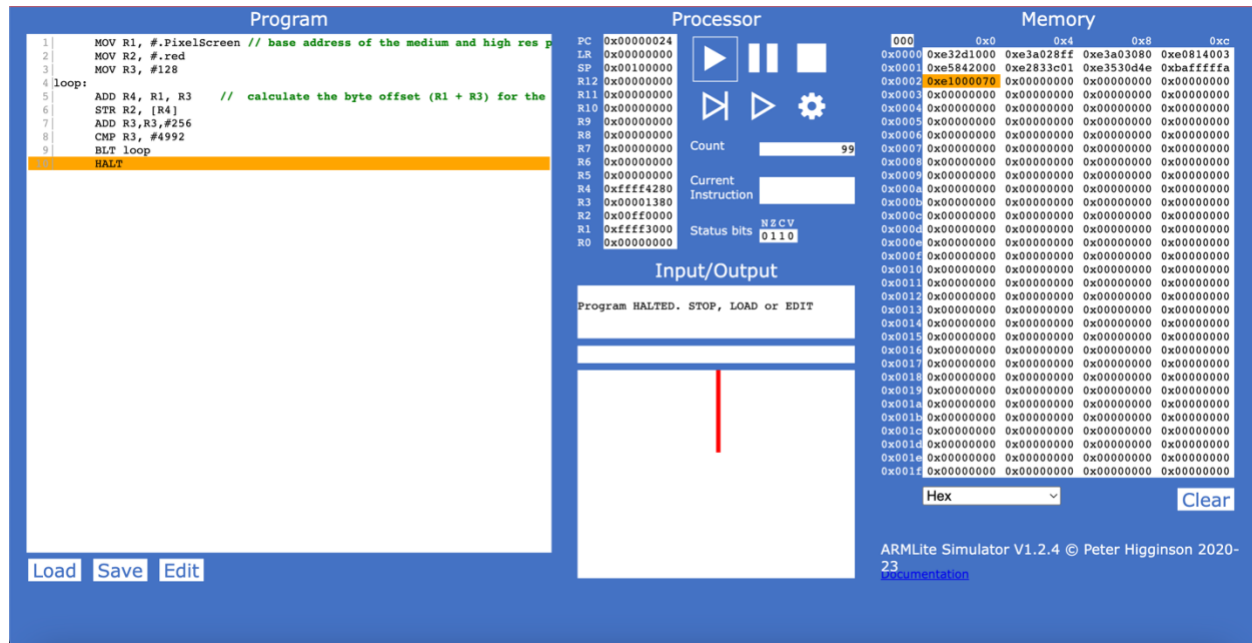
ADD R4, R1, R3 // calculate the byte offset (R1 + R3) for the next pixel and store new address in R4

STR R2, [R4]

ADD R3, R3, #256

CMP R3, #4992

BLT loop
HALT

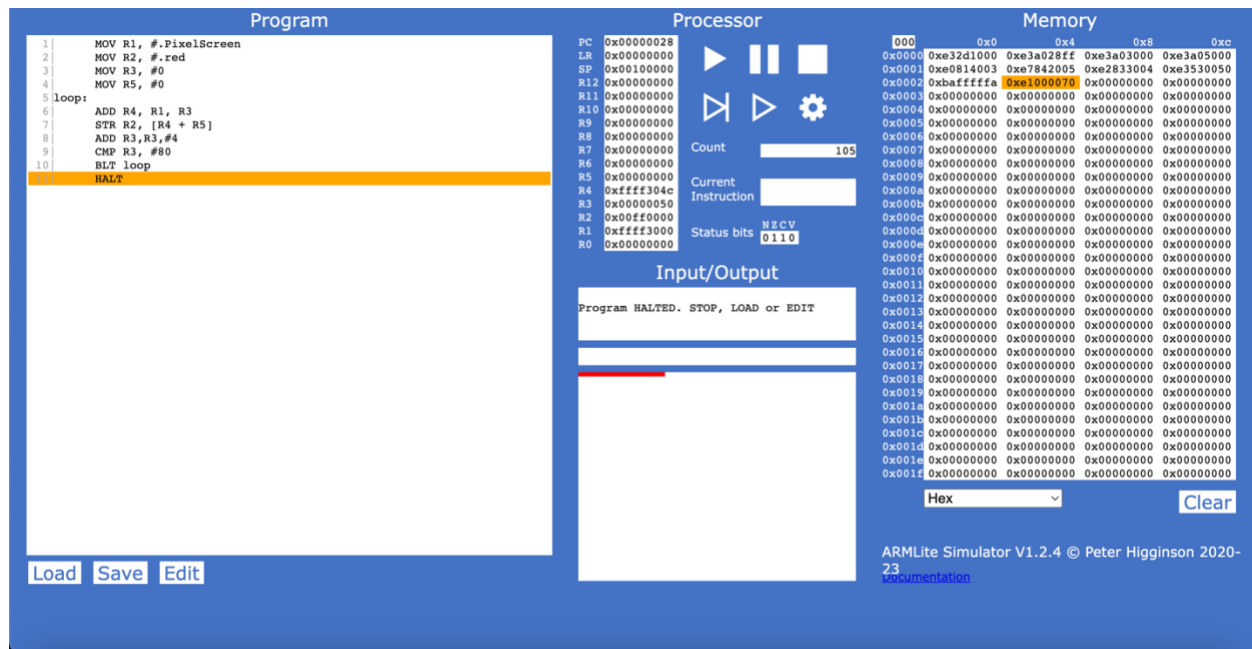


9.2.1

```

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
```

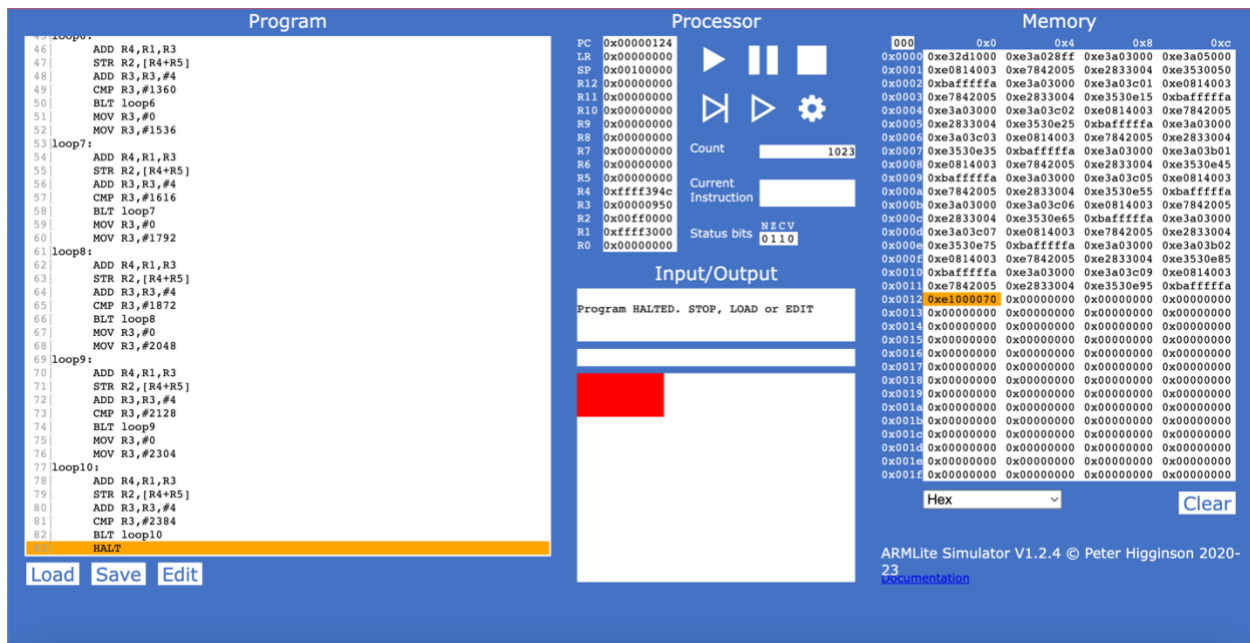
```
loop1:
ADD R4, R1, R3
STR R2, [R4+R5]
ADD R3, R3, #4
CMP R3, #80
BLT loop1
MOV R3, #0
MOV R3, #256
```

```
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.

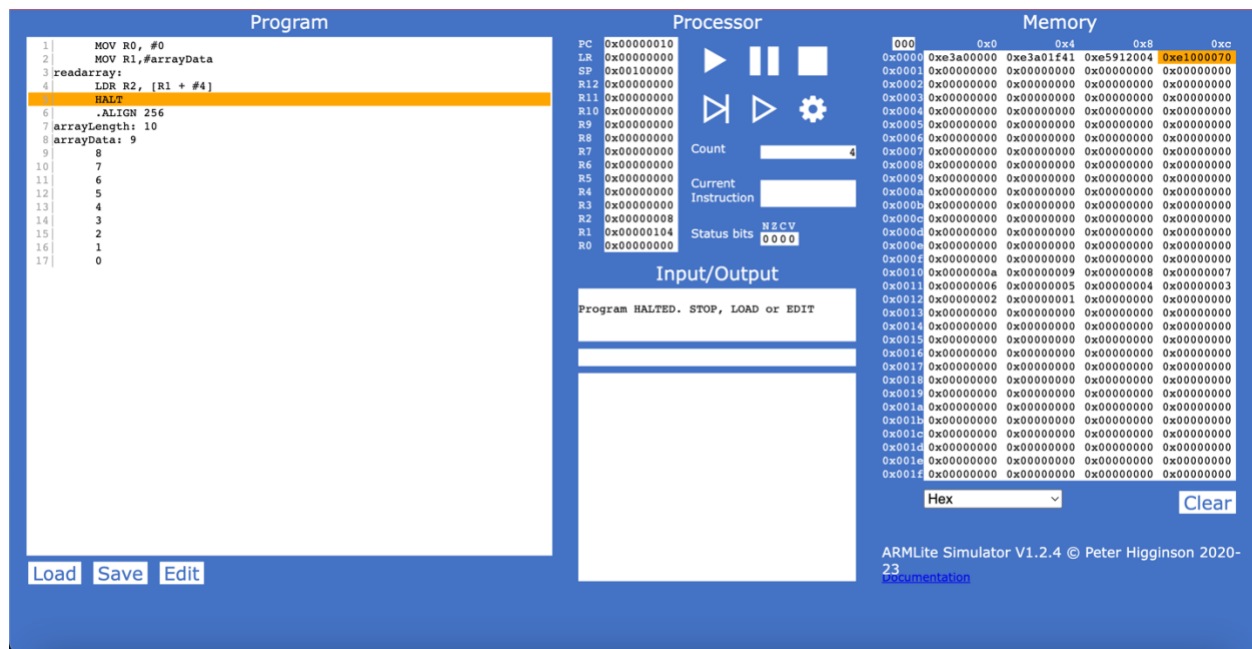
(b) Add a line of code to the above to read the 5th value of the array to register R0 (i.e., it should use indirect addressing to access the 5th cell in the array)

```
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

The screenshot displays the ARMLite Simulator V1.2.4 interface, which is divided into several sections:

- Program:** A list of assembly instructions with line numbers. Line 11 is highlighted in orange.

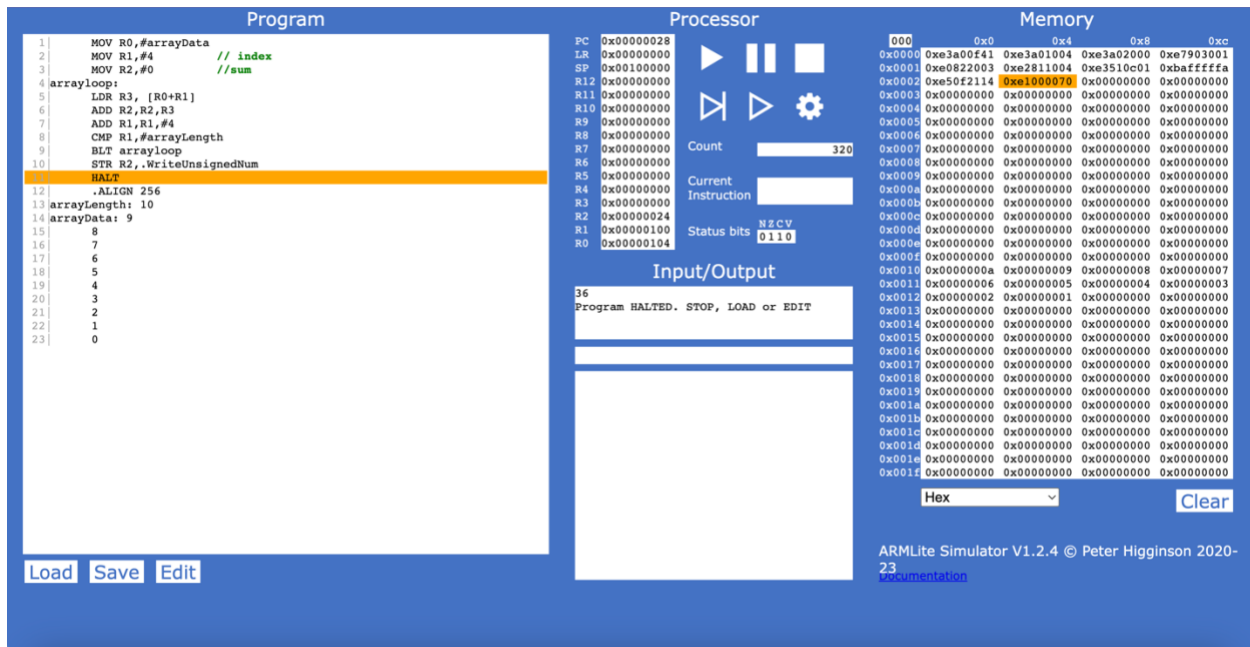

```

1  MOV R0,#arrayData
2  MOV R1,#4          // index
3  MOV R2,#0          //sum
4  arrayloop:
5      LDR R3, [R0+R1]
6      ADD R2,R2,R3
7      ADD R1,R1,#4
8      CMP R1,arrayLength
9      BLT arrayloop
10     STR R2,.WriteUnsignedNum
11     HALT
12     .ALIGN 256
13     arrayLength: 10
14     arrayData: 9
15         8
16         7
17         6
18         5
19         4
20         3
21         2
22         1
23         0
      
```
- Processor:** Shows the current state of the processor registers and control flags.
 - PC: 0x00000028
 - LR: 0x00000000
 - SP: 0x00100000
 - R12: 0x00000000
 - R11: 0x00000000
 - R10: 0x00000000
 - R9: 0x00000000
 - R8: 0x00000000
 - R7: 0x00000000
 - R6: 0x00000000
 - R5: 0x00000000
 - R4: 0x00000000
 - R3: 0x00000000
 - R2: 0x00000024
 - R1: 0x00000100
 - R0: 0x00000104
- Memory:** A table showing memory addresses and their corresponding values. The address 0xe1000070 is highlighted in orange.

000	0x0	0x1	0x8	0xc0
0x0000	0xe3a00f41	0xe3a01004	0xe3a02000	0xe7903001
0x0001	0xe0822003	0xe2811004	0xe3510c01	0xbaffffff
0x0002	0xe50f2114	0xe1000070	0x00000000	0x00000000
0x0003	0x00000000	0x00000000	0x00000000	0x00000000
0x0004	0x00000000	0x00000000	0x00000000	0x00000000
0x0005	0x00000000	0x00000000	0x00000000	0x00000000
0x0006	0x00000000	0x00000000	0x00000000	0x00000000
0x0007	0x00000000	0x00000000	0x00000000	0x00000000
0x0008	0x00000000	0x00000000	0x00000000	0x00000000
0x0009	0x00000000	0x00000000	0x00000000	0x00000000
0x000a	0x00000000	0x00000000	0x00000000	0x00000000
0x000b	0x00000000	0x00000000	0x00000000	0x00000000
0x000c	0x00000000	0x00000000	0x00000000	0x00000000
0x000d	0x00000000	0x00000000	0x00000000	0x00000000
0x000e	0x00000000	0x00000000	0x00000000	0x00000000
0x000f	0x00000000	0x00000000	0x00000000	0x00000000
0x0010	0x0000000a	0x00000009	0x00000008	0x00000007
0x0011	0x00000006	0x00000005	0x00000004	0x00000003
0x0012	0x00000002	0x00000001	0x00000000	0x00000000
0x0013	0x00000000	0x00000000	0x00000000	0x00000000
0x0014	0x00000000	0x00000000	0x00000000	0x00000000
0x0015	0x00000000	0x00000000	0x00000000	0x00000000
0x0016	0x00000000	0x00000000	0x00000000	0x00000000
0x0017	0x00000000	0x00000000	0x00000000	0x00000000
0x0018	0x00000000	0x00000000	0x00000000	0x00000000
0x0019	0x00000000	0x00000000	0x00000000	0x00000000
0x001a	0x00000000	0x00000000	0x00000000	0x00000000
0x001b	0x00000000	0x00000000	0x00000000	0x00000000
0x001c	0x00000000	0x00000000	0x00000000	0x00000000
0x001d	0x00000000	0x00000000	0x00000000	0x00000000
0x001e	0x00000000	0x00000000	0x00000000	0x00000000
0x001f	0x00000000	0x00000000	0x00000000	0x00000000
- Input/Output:** A section for monitoring and controlling the processor's input and output. It shows the current instruction (HALT) and status bits (NZCV: 0110).

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

The screenshot shows the ARMLite Simulator V1.2.4 interface. The **Program** panel on the left displays assembly code for a loop that writes data to an array. The **Processor** panel in the center shows the current state of the processor, including the PC, registers, and status bits. The **Memory** panel on the right shows the memory dump, with the address 0x10000070 highlighted. The **Input/Output** panel at the bottom shows the program has halted.

Program

```

1  MOV R0,#arrayData1
2  MOV R4,#arrayData2 // index
3  MOV R1,#36
4  arrayloop:
5      LDR R3, [R0+R1] //pointer to the array
6      STR R3,[R4] // display the value inside of the array
7      STR R3,.WriteUnsignedNum // write what's inside the array in the d
8      SUB R1,R1,#4 // subtract the index of the array by 4 bytes
9      CMP R1, #0 // R1 - 0
10     BNE arrayloop
11     HALT
12     .ALIGN 256
13     arrayLength1: 10
14     arrayData1: 9
15     8
16     7
17     6
18     5
19     4
20     3
21     2
22     1
23     0
24     arrayLength2: 10
25     arrayData2: 0

```

Processor

PC: 0x00000028
LR: 0x00000000
SP: 0x00100000
R12: 0x00000000
R11: 0x00000000
R10: 0x00000000
R9: 0x00000000
R8: 0x00000000
R7: 0x00000000
R6: 0x00000000
R5: 0x00000000
R4: 0x00000130
R3: 0x00000008
R2: 0x00000000
R1: 0x00000000
R0: 0x00000104

Count: 58
Current Instruction:
Status bits: NSCV 0100

Input/Output

0 1 2 3 4 5 6 7 8
Program HALTED. STOP, LOAD or EDIT

Memory

000	0x0	0x4	0x8	0xc
0x0000	0xe3a00f41	0xe3a04e13	0xe3a01024	0xe7903001
0x0001	0xe5843000	0xe50f3108	0xe2411004	0xe3510000
0x0002	0x1afffff9	0xe1000070	0x00000000	0x00000000
0x0003	0x00000000	0x00000000	0x00000000	0x00000000
0x0004	0x00000000	0x00000000	0x00000000	0x00000000
0x0005	0x00000000	0x00000000	0x00000000	0x00000000
0x0006	0x00000000	0x00000000	0x00000000	0x00000000
0x0007	0x00000000	0x00000000	0x00000000	0x00000000
0x0008	0x00000000	0x00000000	0x00000000	0x00000000
0x0009	0x00000000	0x00000000	0x00000000	0x00000000
0x000a	0x00000000	0x00000000	0x00000000	0x00000000
0x000b	0x00000000	0x00000000	0x00000000	0x00000000
0x000c	0x00000000	0x00000000	0x00000000	0x00000000
0x000d	0x00000000	0x00000000	0x00000000	0x00000000
0x000e	0x00000000	0x00000000	0x00000000	0x00000000
0x000f	0x00000000	0x00000000	0x00000000	0x00000000
0x0010	0x00000000	0x00000000	0x00000000	0x00000000
0x0011	0x00000000	0x00000000	0x00000000	0x00000000
0x0012	0x00000000	0x00000000	0x00000000	0x00000000
0x0013	0x00000000	0x00000000	0x00000000	0x00000000
0x0014	0x00000000	0x00000000	0x00000000	0x00000000
0x0015	0x00000000	0x00000000	0x00000000	0x00000000
0x0016	0x00000000	0x00000000	0x00000000	0x00000000
0x0017	0x00000000	0x00000000	0x00000000	0x00000000
0x0018	0x00000000	0x00000000	0x00000000	0x00000000
0x0019	0x00000000	0x00000000	0x00000000	0x00000000
0x001a	0x00000000	0x00000000	0x00000000	0x00000000
0x001b	0x00000000	0x00000000	0x00000000	0x00000000
0x001c	0x00000000	0x00000000	0x00000000	0x00000000
0x001d	0x00000000	0x00000000	0x00000000	0x00000000
0x001e	0x00000000	0x00000000	0x00000000	0x00000000
0x001f	0x00000000	0x00000000	0x00000000	0x00000000

Hex Clear

ARMLite Simulator V1.2.4 © Peter Higginson 2020-23
[Documentation](#)

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

1| MOV R0,#arrayData1
2| MOV R1,#36 // index
3| arrayloop:
4| LDR R3,[R0+R1] //pointer to the array
5| STR R3,WriteUnsignedNum // write what's inside the array in the d
6| SUB R1,R1,#4 // subtract the index of the array by 4 bytes
7| CMP R1,#0 // R1 - 0
8| BNE arrayloop
9| HALT
10| .ALIGN 256
11| arrayLength1: 10
12| arrayData1: 9
13| 8
14| 7
15| 6
16| 5
17| 4
18| 3
19| 2
20| 1
21| 0

Load Save Edit

Processor

PC 0x00000020
LR 0x00000000
SP 0x00100000
R12 0x00000000
R11 0x00000000
R10 0x00000000
R9 0x00000000
R8 0x00000000
R7 0x00000000
R6 0x00000000
R5 0x00000000
R4 0x00000000
R3 0x00000008
R2 0x00000000
R1 0x00000000
R0 0x00000104

▶ ⏏ ⏏
⏏ ▶ ⚙
Count 48
Current Instruction
Status bits NECV 0100

Input/Output
0 1 2 3 4 5 6 7 8
Program HALTED. STOP, LOAD or EDIT

Memory

000	0x0	0x4	0x8	0xc
0x0000	0xe3a00f41	0xe3a01024	0xe7903001	0xe50f3100
0x0001	0xe2411004	0xe3510000	0x1affffff	0xe1000070
0x0002	0x00000000	0x00000000	0x00000000	0x00000000
0x0003	0x00000000	0x00000000	0x00000000	0x00000000
0x0004	0x00000000	0x00000000	0x00000000	0x00000000
0x0005	0x00000000	0x00000000	0x00000000	0x00000000
0x0006	0x00000000	0x00000000	0x00000000	0x00000000
0x0007	0x00000000	0x00000000	0x00000000	0x00000000
0x0008	0x00000000	0x00000000	0x00000000	0x00000000
0x0009	0x00000000	0x00000000	0x00000000	0x00000000
0x000a	0x00000000	0x00000000	0x00000000	0x00000000
0x000b	0x00000000	0x00000000	0x00000000	0x00000000
0x000c	0x00000000	0x00000000	0x00000000	0x00000000
0x000d	0x00000000	0x00000000	0x00000000	0x00000000
0x000e	0x00000000	0x00000000	0x00000000	0x00000000
0x000f	0x00000000	0x00000000	0x00000000	0x00000000
0x0010	0x0000000a	0x00000009	0x00000008	0x00000007
0x0011	0x00000006	0x00000005	0x00000004	0x00000003
0x0012	0x00000002	0x00000001	0x00000000	0x00000000
0x0013	0x00000000	0x00000000	0x00000000	0x00000000
0x0014	0x00000000	0x00000000	0x00000000	0x00000000
0x0015	0x00000000	0x00000000	0x00000000	0x00000000
0x0016	0x00000000	0x00000000	0x00000000	0x00000000
0x0017	0x00000000	0x00000000	0x00000000	0x00000000
0x0018	0x00000000	0x00000000	0x00000000	0x00000000
0x0019	0x00000000	0x00000000	0x00000000	0x00000000
0x001a	0x00000000	0x00000000	0x00000000	0x00000000
0x001b	0x00000000	0x00000000	0x00000000	0x00000000
0x001c	0x00000000	0x00000000	0x00000000	0x00000000
0x001d	0x00000000	0x00000000	0x00000000	0x00000000
0x001e	0x00000000	0x00000000	0x00000000	0x00000000
0x001f	0x00000000	0x00000000	0x00000000	0x00000000

Hex Clear

ARMLite Simulator V1.2.4 © Peter Higginson 2020-23
[Documentation](#)