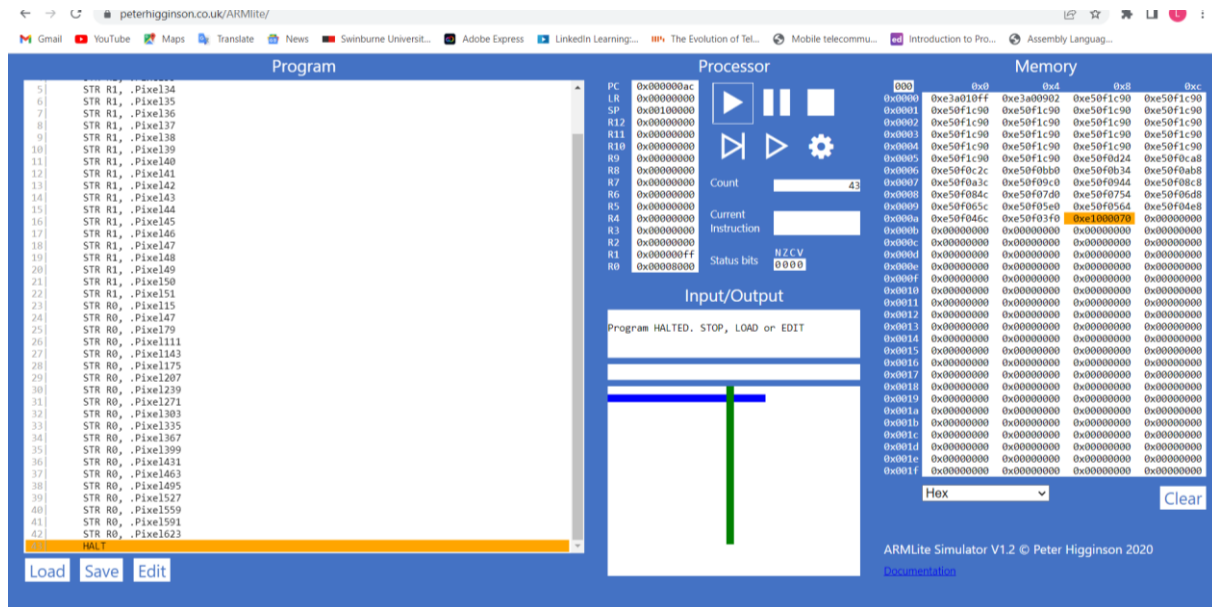


# Week 9 Lab - Part 9.1

## Indirect and Indexed Addressing

### Exercise 9.1.1

- (a) Write a simple ARM lite 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.
- (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

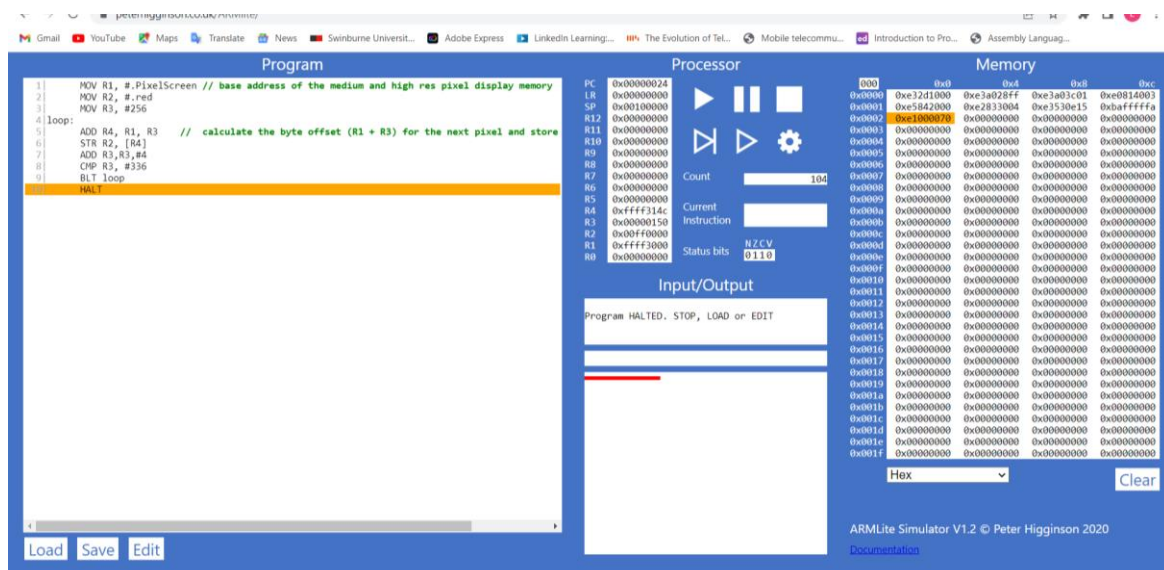


### Exercise 9.1.3

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

An indirect Addressing is an absolute address that contains another address. Therefore, `STR R2, [R4]` will store the memory address of R4 into R2.

- (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.



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

The screenshot shows the ARMLite Simulator V1.2 interface. The Program window contains the following assembly code:

```

1 MOV R1, #PixelScreen // base address of the medium and high res pixel display memory
2 MOV R2, #blue
3 MOV R3, #128
4 loop:
5   ADD R4, R1, R3 // calculate the byte offset (R1 + R3) for the next pixel and store
6   STR R2, [R4]
7   ADD R3, R3, #256
8   CMP R3, #4096
9   BLT loop
10  HALT

```

The Processor window shows the current instruction as `STR R2, [R4]` and the status bits as `NZCV 0110`. The Memory window shows the memory contents, with the address `0xe3a020ff` highlighted. The Input/Output window shows the program status as `Program HALTED. STOP, LOAD or EDIT`.

## Week 9 Lab - Part 9.2

### Indexed Addressing

In ARM lite, rewrite the code above so that it uses *indexed addressing* to draw the line in Medium-res display mode.

The screenshot shows the ARMLite Simulator V1.2 interface. The Program window contains the following assembly code:

```

1 MOV R1, #PixelScreen
2 MOV R2, #red
3 MOV R3, #0
4 loop:
5   STR R2, [R1 + R3]
6   ADD R3, R3, #4
7   CMP R3, #80
8   BLT loop
9   HALT

```

The Processor window shows the current instruction as `STR R2, [R1 + R3]` and the status bits as `NZCV 0110`. The Memory window shows the memory contents, with the address `0xe3a020ff` highlighted. The Input/Output window shows the program status as `Program HALTED. STOP, LOAD or EDIT`.

# Week 9 Lab - Part 9.3

## Arrays

### Exercise 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 word address divisible by 256

### Exercise 9.3.1 (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)

### Exercise 9.3.1 (c)

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

The screenshot shows the ARMLite Simulator V1.2 interface. The 'Program' pane on the left contains the following assembly code:

```
1: .ALIGN 256
2: arrayLength: 10
3: arrayData: 9
4:
5: 8
6: 7
7: 6
8: 5
9: 4
10: 3
11: 2
12: 1
13: 0
14: MOV R1, #0x00010
15: LDR R0, [R1]
```

The 'Processor' pane in the center shows the current instruction as `LDR R0, [R1]` and the status bits as `NZCV 0000`. The 'Memory' pane on the right shows a memory dump starting at address 0x00000000, with the value at address 0x00000004 highlighted as `0x00000004`.

### Exercise 9.3.3

Using the original array definition, modify your code so that it adds up all the values in the array. Your program should use indexed addressing to access each value and write the result to R0.

← → ↻ peterhigginson.co.uk/ARMLite/ Gmail YouTube Maps Translate News Swinburne Universit... Adobe Express LinkedIn Learning... The Evolution of Tel... Mobile telecommu... Introduction to Pro... Assembly Lang...

### Program

```
1 MOV R4, #arrayData
2 MOV R1, #0
3 MOV R0, #0
4 arrayLoop:
5   LDR R3, [R4, R1]
6   ADD R0, R0, R3
7   ADD R1, R1, #4
8   CMP R1, #arrayLength
9   BLT arrayLoop
10 HALT
11 .ALIGN 256
12 arrayLength: 10
13 arrayData: 9
14 8
15 7
16 6
17 5
18 4
19 3
20 2
21 1
22 0
```

Load Save Edit

### Processor

PC	0x00000024
LR	0x00000000
SP	0x00100000
R12	0x00000000
R11	0x00000000
R10	0x00000000
R9	0x00000000
R8	0x00000000
R7	0x00000000
R6	0x00000000
R5	0x00000000
R4	0x00000104
R3	0x00000000
R2	0x00000000
R1	0x00000100
R0	0x0000002d

Count 324

Current Instruction

Status bits NZCV 0110

### Input/Output

Program HALTED. STOP, LOAD or EDIT

### Memory

000	0x0	0x4	0x8	0xc
0x0000	0xe3a04f41	0xe3a01000	0xe3a00000	0xe7943001
0x0001	0xe0800003	0xe2811004	0xe3510c01	0xbaffffff
0x0002	0xe1000070	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 © Peter Higginson 2020  
[Documentation](#)