Lab4

Full in name: Nguyễn Ngọc Hải

Student ID: 23133021

Part 4.1 Memory in ARMLite

On the right side of the ARMLite simulator is a grid of memory addresses, with each block of 8 hex digits (all initialised to 0) representing a 32 bit word.

Click on any visible memory word and type in 101 (followed by the "Enter" key).

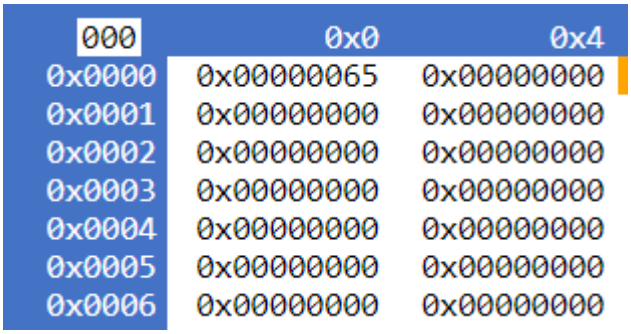
4.1.1 What value is displayed ? Why ?

Trả lời:

Khi nhập 101 ở hệ thập phân, giá trị được hiển thị sẽ là

0x00000065 ở hệ hex. Vì trong hệ thập phân, 101 tương ứng với

0x65 trong hệ hex.

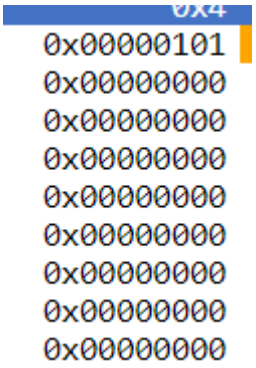


000	0x0	0x4
0x0000	0x00000065	0x00000000
0x0001	0x00000000	0x00000000
0x0002	0x00000000	0x00000000
0x0003	0x00000000	0x00000000
0x0004	0x00000000	0x00000000
0x0005	0x00000000	0x00000000
0x0006	0x00000000	0x00000000

Click on another memory word, enter 0x101

Trả lời:

- Khi nhập 0x101 (hệ hex), giá trị hiển thị sẽ là 0x00000101. Vì 0x101 là giá trị hex trực tiếp được lưu trữ trong ô nhớ ⇔ 257 trong hệ thập phân.



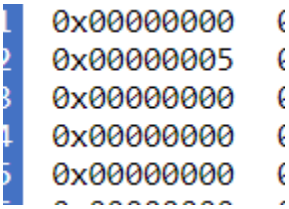
0x4
0x00000101
0x00000000
0x00000000
0x00000000
0x00000000
0x00000000
0x00000000
0x00000000
0x00000000

4.1.2 What value is displayed, and why?

On another memory word, enter 0b101

Trả lời:

Nhập 0b101, giá trị hiển thị là 0x5. Tiền tố 0b biểu thị nhị phân và 0b101 trong nhị phân tương đương với số 5 ở dạng thập phân, được hiển thị dưới dạng 0x5 trong hệ thập lục phân



4.1.3 What value is displayed, and why?

Trả lời:

Khi nhập "0b101", ARMLite nhận dạng "0b" là định dạng số nhị phân và lưu trữ giá trị 0b101 trong bộ nhớ, tương đương với 5 trong hệ thập phân => giá trị được lưu sẽ là

00000005 (hex).

If you now hover (don't click) the mouse over any of the memory words where you have entered a value you will get a pop-up 'tooltip'.

What does the tooltip tell you?

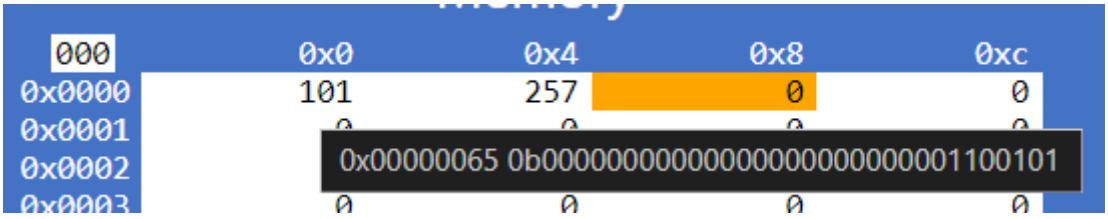
Trả lời:

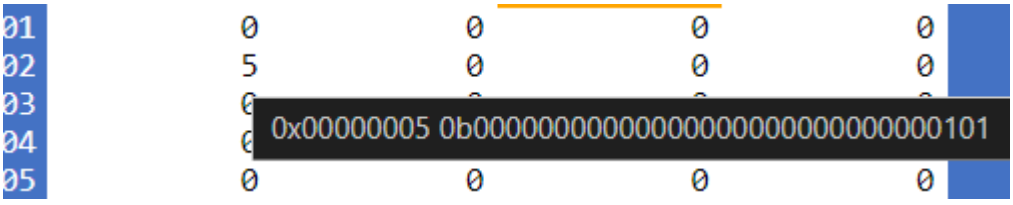
Tooltip hiển thị giá trị ở dạng thập phân và nhị phân có thể như sau:

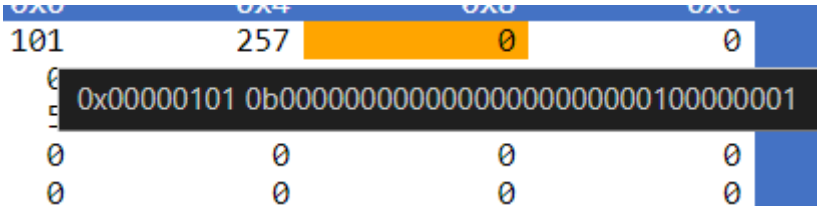
- **Giá trị trong thập phân**: 5
- **Giá trị trong nhị phân**: 0b101

The value in decimal and binary

Trả lời:







Below the grid of memory words is a drop down menu that looks like this:

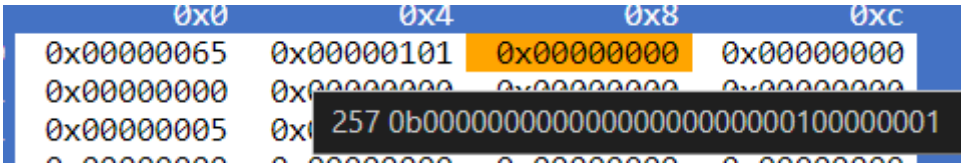
This drop-down selector allows you to change the base in which data is displayed. Changing the base does not change the underlying data value, only the base number system in which the value is displayed.

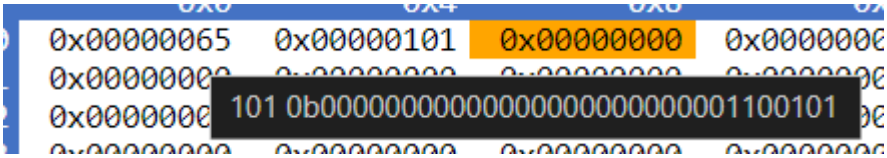
Change this to Decimal (unsigned) and note the change that has occurred to the three memory words you previously entered.

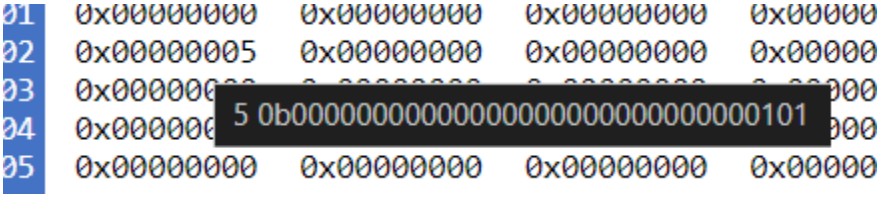
When you mouse over one of these words, what now appears in the tooltip?

The value in hex and binary

Trả lời:







4.1.4: Does changing the representation of the data in memory also change the representation of the row and column-headers (the white digits on a blue background)? Should it ?

Trả lời:

- Bộ nhớ và cách biểu diễn dữ liệu:

+Dữ liệu trong giả lập ARMLite có thể hiển thị dưới các hệ cơ số khác nhau như nhị phân, thập phân, và thập lục phân. Điều này giúp bạn dễ dàng quan sát và so sánh dữ liệu dưới nhiều dạng biểu diễn khác nhau.

+Tuy nhiên, việc thay đổi hệ cơ số chỉ thay đổi cách dữ liệu được hiển thị chứ không làm thay đổi giá trị thực tế được lưu trữ trong bộ nhớ.

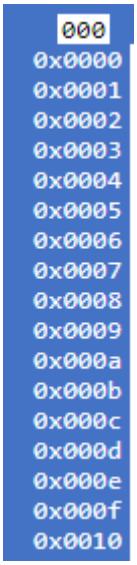
-Tiêu đề hàng và cột:

+Các tiêu đề hàng và cột (chữ trắng trên nền xanh dương) đại diện cho địa chỉ bộ nhớ chứ không phải dữ liệu. Những địa chỉ này được hiển thị dưới dạng thập lục phân và giúp xác định vị trí của mỗi ô nhớ.

+Vì các tiêu đề này là địa chỉ, không phải dữ liệu, nên chúng sẽ không thay đổi khi bạn thay đổi cách hiển thị dữ liệu trong ô bộ nhớ.

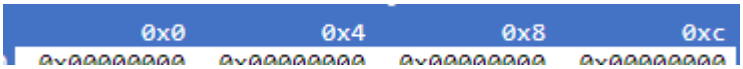
Part 4.2 Memory Addressing

Each word of memory has a unique ‘address’, expressed as a five-digit hex number. On the ARMLite Simulator, memory words are laid out in four columns, however this is only for visual convenience. Notice that each row starts with a four digit hex value (in white) that looks like this:



These values represent the first four digits of the address for all memoery words in that row.

Now look along the top of each column in the Memory grid and note these values:



These single digit hex values represent offsets from the row-header address. Therefore, the full address of any memory word is obtained by appending the column header digit to the 4 digit row-header. For example, the address of the top-left word on this screen is 0x00000, and the bottom-right is

0x001fc.

4.2.1 Notice these column header memory address offsets go up in multiples of 0x4. Why is this ?

Hint: remember how many bits are in each memory word !

ARMlite, in common with most modern processors uses 'byte addressing' for memory. When storing or retrieving a word you generally specify only the address of the first of the bytes making up each word - we'll come back to this when we start dealing with storing and loading values to and from memory.

Trả lời:

- Trong ARMlilte mỗi từ trong bộ nhớ có kích thước 32 bit (4 byte), cho nên khi được sử dụng, mỗi từ trong bộ nhớ chiếm 4 byte đồng nghĩa với việc mỗi địa chỉ của mỗi từ sẽ cách nhau 4 byte trong bộ nhớ và trong hệ Hex, 4 được biểu diễn là 0x4. Cho nên độ lệch của mỗi cột phải là bội số của 4 hay 0x0, 0x4, 0x8, 0xc.

- Việc này đảm bảo việc lưu trữ các từ trong hệ 32 bit cũng như việc truy cập được thực hiện một cách chính xác.

- Ví dụ: địa chỉ của từ đầu tiên sẽ là 0x00000, từ thứ hai sẽ là 0x00004 (tăng lên 4 byte), tương tự với từ thứ ba và thứ bốn sẽ là 0x00008 và 0x0000c

Part 4.3: Editting and Submitting Assembly Code

On the left side of the ARMlite simulator is the Program window. This is where you can load and/or edit assembly code to be executed by the simulator.

Click the Edit button below the Program window, and then copy and Paste the following ARMLite assembly program into the window:

```
 MOV R1, #.PixelScreen
 MOV R2, #screen2
 MOV R6, #0
 MOV R9, #.black
 MOV R10, #.white
 MOV R3, #0
loopWhite: STR R10, [R2+R3]
 ADD R3, R3, #4
 CMP R3, #12288
 BLT loopWhite
 MOV R3, #260
randLoop: LDR R0, .Random
 AND R0, R0, #1
 CMP R0, #0
 BNE .+2
 STR R9, [R2+R3]
 BL nextCell
 CMP R3, #12032
 BLT randLoop
copyScreen2to1: MOV R3, #0
copyLoop: LDR R0, [R2+R3]
 STR R0, [R1+R3]
 ADD R3, R3, #4
 CMP R3, #12288
 BLT copyLoop
 ADD R6, R6, #1
 MOV R3, #260
nextGenLoop: MOV R5, #0
 SUB R4, R3, #256
 BL countIfLive
 SUB R4, R3, #252
 BL countIfLive
 ADD R4, R3, #4
 BL countIfLive
 ADD R4, R3, #260
 BL countIfLive
 ADD R4, R3, #256
```

```
 BL countIfLive
 ADD R4, R3, #252
BL countIfLive
 SUB R4, R3, #4
 BL countIfLive
 SUB R4, R3, #260
 BL countIfLive
 CMP R5, #4
 BLT .+3
 STR R10, [R2+R3]
 B continue
 CMP R5, #3
 BLT .+3
 STR R9, [R2+R3]
 B continue
 CMP R5, #2
 BLT .+2
 B continue
 STR R10, [R2+R3]
continue: BL nextCell
 MOV R0, #12032
 CMP R3, R0
 BLT nextGenLoop
 B copyScreen2to1
countIfLive: LDR R0, [R1+R4]
 CMP R0, R10 //White
 BEQ .+2
 ADD R5, R5, #1
 RET
nextCell:
 ADD R3, R3, #4
 AND R0, R3, #255
 CMP R0, #0
 BEQ .-3
 CMP R0, #252
 BEQ .-5
 RET
```

HALT

.ALIGN 1024

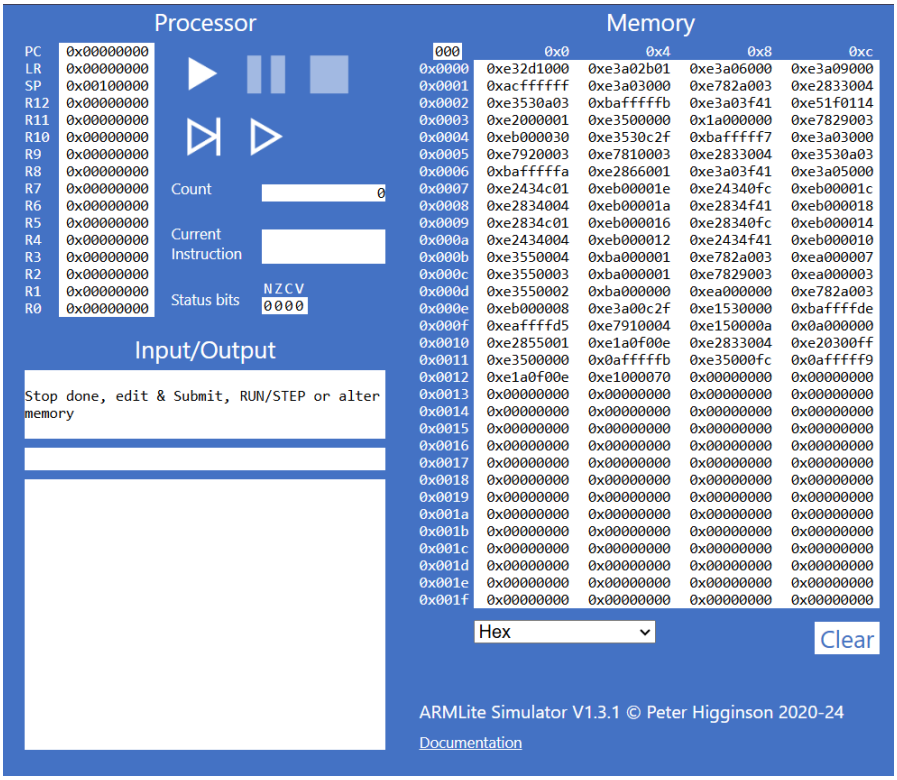
screen2: 0

Once copied, click the Submit button. This invokes the assembler and all going well, should not give any errors. If it does then you may need to check you correctly copied all the code above (nothing more, nothing less).

4.3.1 Take a screen shot of the simulator in full and add it to your submission document

Notice that the memory window has changed ? You should see lots of values in at least the first 13 rows of memory words.

Trả lời:



4.3.2 Based on what we've learnt about assemblers and Von Neuman architectures, explain what you think just happened.

You will also see that ARMlite has now added 'line numbers' to your program. These do not form part of the source code, but are there to help you navigate and discuss your code.

Hover the mouse over one of the lines of the source code (after the code has been submitted).

You will see a pop-up tooltip showing a 5 digit hex value.

Trả lời:

Khi mã assembly được biên dịch thành công và nạp vào trình giả lập ARMlite, những thay đổi trong cửa sổ bộ nhớ cho thấy rằng chương trình đã thực hiện các bước sau:

1. **Nạp dữ liệu vào bộ nhớ**: Các giá trị trong 13 hàng đầu tiên của bộ nhớ đại diện cho dữ liệu đã được lưu trữ từ các lệnh trong mã assembly. Điều này bao gồm các giá trị được khởi tạo cho màn hình và các màu sắc (đen và trắng) trong mã. Các lệnh như STR đã ghi các giá trị vào các địa chỉ bộ nhớ cụ thể.
2. **Kiến trúc Von Neumann**: Kiến trúc Von Neumann cho phép xử lý đồng thời dữ liệu và lệnh từ cùng một bộ nhớ. Điều này có nghĩa là khi mã assembly được thực thi,

nó không chỉ thao tác với dữ liệu mà còn có thể truy cập lệnh tiếp theo để thực hiện. Điều này giúp tối ưu hóa quá trình xử lý, vì dữ liệu và chương trình có thể chia sẻ cùng một không gian bộ nhớ.

3. **Biên dịch và thực thi**: Quá trình biên dịch của trình biên dịch đã chuyển đổi mã assembly thành các mã máy, cho phép CPU hiểu và thực thi chúng. Sau khi nạp vào bộ nhớ, các lệnh này đã được trình giả lập xử lý, tạo ra kết quả mà có thể quan sát được trong cửa sổ bộ nhớ.
4. **Thay đổi bộ nhớ**: Những thay đổi trong cửa sổ bộ nhớ cho thấy rằng mã đã được thực thi, và giá trị tại các địa chỉ bộ nhớ đã được cập nhật theo các lệnh trong mã assembly, điều này cho thấy rằng chương trình đã hoạt động đúng cách.

4.3.3 Based on what we have learnt about memory addressing in ARMlite, and your response to 4.3.2, what do you think this value represents ?

If you're not sure what's going on here, ask your tutor for assistance.

It's time to try a few other things.

Hit **Edit** and try inserting:

- A couple of blank lines
- Additional spaces before an instruction, or just after a comma (but not between other characters)
- A comment on a line of its own, starting with // such as //My first program
- A comment after an instruction but on the same line

Submit the code again.

What has happened to:

- The blank lines
  - Its be removed
- Additional spaces
  - Its be removed
- The comments
  - Cmt showed in green but note affect the outcome and the memory
- The line numbers
  - Cmt showed in green but note affect the outcome and the memory
  -
- The total number of instructions that end up as words in memory? (Why?)
  - 44 because each instruction occupies exactly one word

Click **Edit** again and remove the comma from the first line of code. What happens when you **Submit** now?

OK - enough mucking about. Restore the program to its original condition, either by going back to Edit, or just copying it again, and click **Submit**. Time to run this thing!

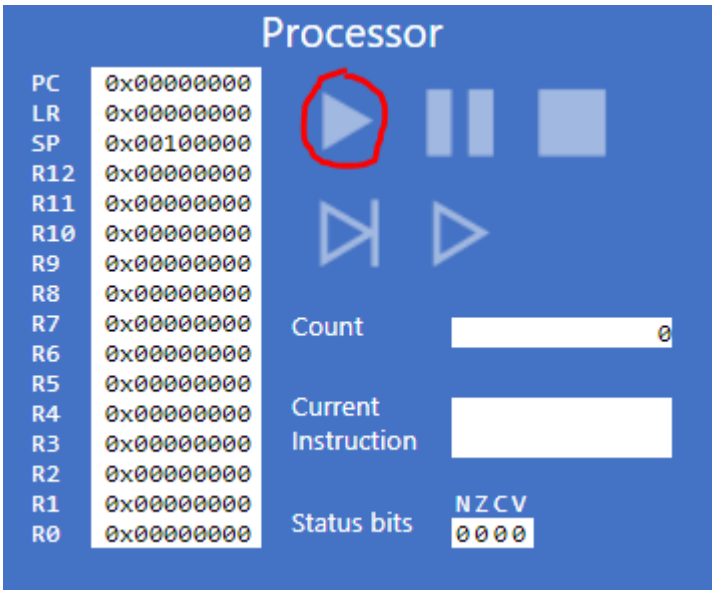
Trả lời:

Giá trị hex 5 chữ số mà hiện lên khi di chuột qua một dòng mã trong trình giả lập ARMlite đại diện cho địa chỉ bộ nhớ nơi mà dòng mã đó được nạp trong bộ nhớ. Dưới đây là phân tích về ý nghĩa của giá trị này:

1. **Địa chỉ bộ nhớ**: Trong hệ thống máy tính, mỗi phần dữ liệu và lệnh đều được lưu trữ tại một địa chỉ cụ thể trong bộ nhớ. Giá trị hex này xác định vị trí trong bộ nhớ nơi mà mã máy tương ứng với dòng mã assembly được lưu trữ.
2. **Bắt đầu từ địa chỉ cơ bản**: Địa chỉ bộ nhớ trong ARMlite thường bắt đầu từ một địa chỉ cố định, gọi là địa chỉ cơ bản. Mỗi lệnh trong mã assembly được lưu trữ tại các địa chỉ này và chiếm một lượng bộ nhớ nhất định, thường là 4 byte (1 word) cho mỗi lệnh trong kiến trúc ARM. Giá trị hex hiển thị khi di chuột qua mã cho thấy vị trí cụ thể của lệnh đó trong bộ nhớ.
3. **Giúp Điều Hướng**: Các số dòng và giá trị địa chỉ giúp lập trình viên dễ dàng định vị và thảo luận về các phần cụ thể của mã, từ đó phân tích logic chương trình và phát hiện lỗi hiệu quả hơn.
4. **Liên Kết Giữa Mã Nguồn và Mã Máy**: Giá trị này là cầu nối giữa mã nguồn (assembly) và mã máy (machine code). Khi mã assembly được biên dịch, nó sẽ được chuyển đổi thành mã máy tại các địa chỉ bộ nhớ cụ thể, và giá trị hex giúp xác định vị trí đó.

Part 4.4 Executing and Debugging Assembly Code

To execute the assembled source code, we need to click the Run button, circled below:



You'll see a spinning gearwheel appear near the run controls to indicate that the processor is active.

You will also observe a lot of activity in the 'graphics screen' (the lowest of the three panes under

Input/Output). After a short while (a few seconds to a couple of minutes) the display will stabilise.

The program you have loaded and just run is a simulation of a colony of simple organisms, being born, reproducing and, eventually dying. (Individual cells never move, but the patterns of cells being born and dying give the impression of movement, and many interesting dynamic patterns emerge). The code is a variant of a very famous program called Life (see Conway's Game of LifeLinks to an external site. for more information).

To stop the program, you can click on the square Stop button in the above image. To run the program again, simply click Run .

You will notice the behaviour of the program is different each time you run it - this is because the starting pattern of cells is randomised.

Run the program once more.

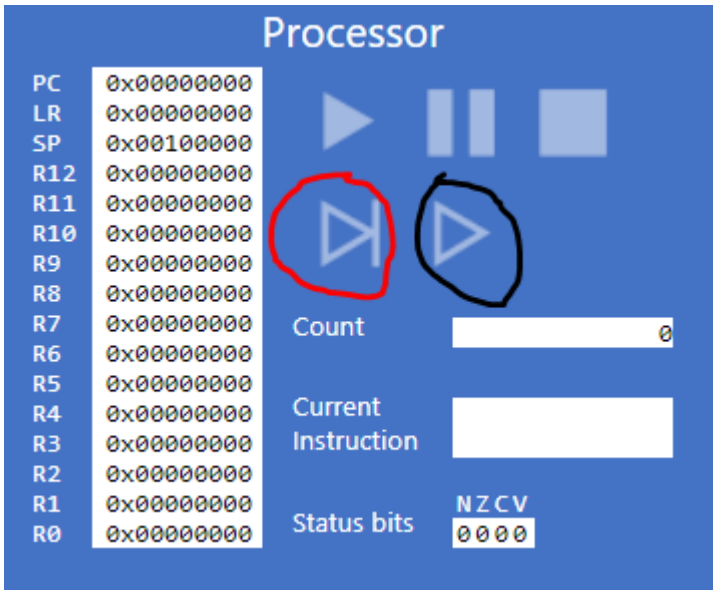
While the program is executing, click the Pause button (between the Run and Stop buttons).

As well as freezing the graphics screen, you will also see orange highlighting appear in both the Program and Memory windows.

4.4.1 What do you think the highlighting in both windows signifies ?

You can continue execution by pressing **Play** again. Do this and then click **Pause** again.

Now click the button circled in red below.



Trả lời:

- Nút được khoanh đỏ là nút “Step” của trình giả lập ARMlite. Nút này có chức năng cho phép bạn thực thi từng lệnh một trong chương trình Assembly. Có thể nói, nút “Step” rất có ích cho việc sửa lỗi(debug) vì nó kiểm tra từng bước thực hiện cảu chương trình, giúp phát hiện và sửa lỗi.

- Nút được khoanh đen là nút “Slow” của trình giả lập ARMlite. Nút này giúp quan sát rõ hơn từng bước thực thi của chương trình. Khi nhấn nút "Slow", trình giả lập sẽ thực thi các lệnh với tốc độ chậm hơn bình thường, cho phép bạn theo dõi sự thay đổi của các thanh ghi, bộ nhớ và các thành phần khác một cách dễ dàng hơn.

4.4.2 What do you think happens when you click the button circled in red ?

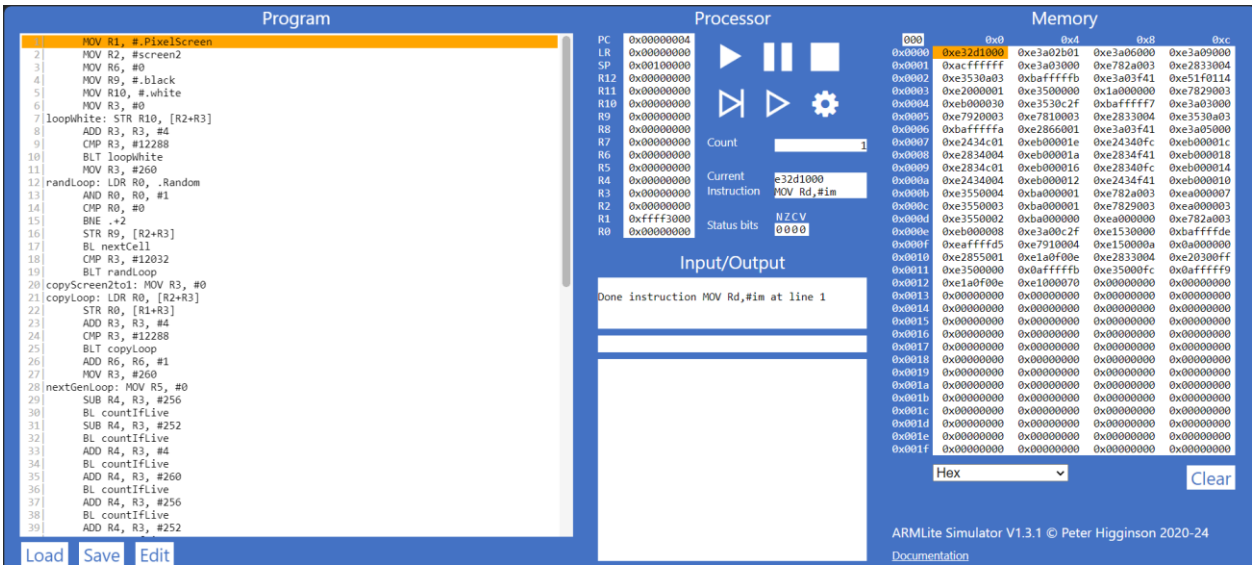
Now click the button circled in black and notice what happens.

You will hopefully notice that the program resumes execution, but at a substantially slower pace than before. You will also notice the orange highlighting in the Program window stepping through lines of code.

Now click the same button again and see what happens. You will hopefully see that when you click the button a few times in succession, the execution speed increases.

Trả lời:

Khi nhấn nút khoanh màu đỏ, màn hình sẽ hiển thị:



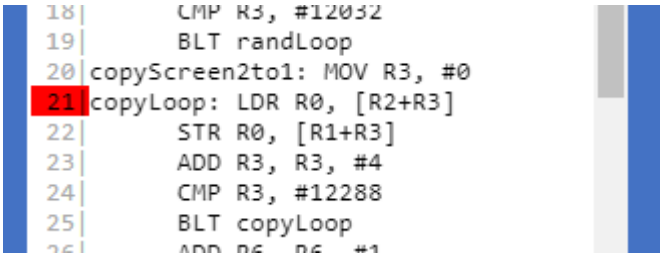
-Khi nhấn nút, bộ xử lý sẽ thực thi một lệnh duy nhất trong mã nguồn, đồng thời trình giả lập sẽ dừng lại khi thực hiện lệnh đó. Vị trí con trỏ trong cửa sổ sẽ di chuyển đến lệnh tiếp theo cùng với đó là các thanh ghi và bộ nhớ sẽ được cập nhật kết quả.

-Khi nhấn nút đen thì chương trình sẽ chạy vòng lặp không dừng ở loopwhite.

These two buttons allow you to slow things down to literally, in the case of the red circled button, single steps of code execution. This is invaluable for debugging code, particularly when you want to check whether the outcome of a given instruction has produced what you expect (be that a value in memory, in a register, or a graphical element on the display etc).

FInally, while paused, click line number 21 of the source code in the Program Window.

This will paint a red background behind the line number like this:



This is called ‘setting a break point’ and will cause processing to be paused when the breakpoint is reached.
Having set the breakpoint, continue running until the pause is observed (almost immediately!).

Trả lời:

- Khi chạy chương trình thì chương trình sẽ thực thi cho đến khi gặp breakpoint. Tại breakpoint, chương trình sẽ tạm dừng, đồng thời sẽ thấy thông báo "Breakpoint detect at PC = 0x00050" trong cửa sổ Input/Output. Tại đây, chúng ta có thể sử dụng các công cụ gỡ lỗi như nút “Step” để kiểm tra chương trình.

4.4.3 Has the processor paused just before, or just after executing the line with the breakpoint ?

From the breakpoint you will find that you can single-step, or continue running slowly or at full speed.

While paused you can remove a breakpoint by clicking on the line again.

Trả lời:

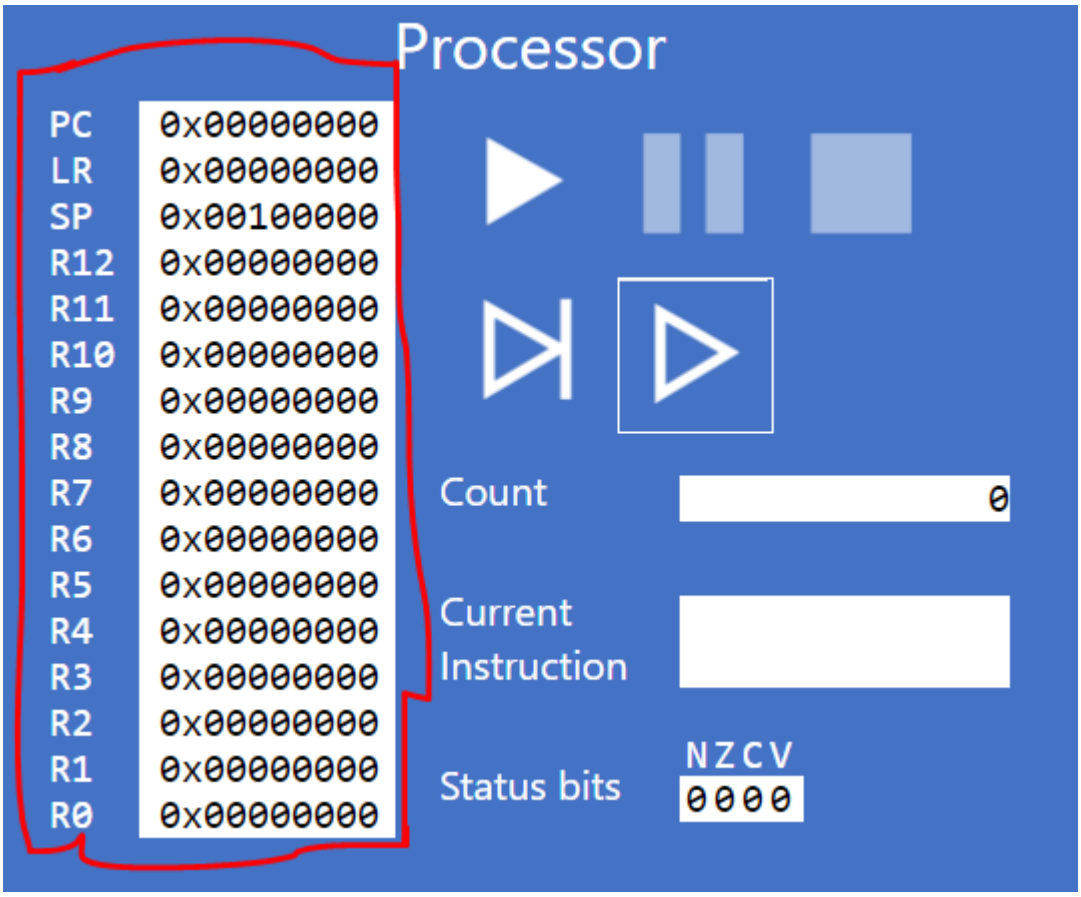
- Khi chạy chương trình thì chương trình sẽ thực thi cho đến khi gặp breakpoint. Tại breakpoint, chương trình sẽ tạm dừng, đồng thời sẽ thấy thông báo "Breakpoint detect at PC = 0x00050" trong cửa sổ Input/Output. Tại đây, chúng ta có thể sử dụng các công cụ gỡ lỗi như nút “Step” để kiểm tra chương trình.

Part 4.5 Registers and Basic Operations

Registers are fundamental to how data is stored and manipulated within a CPU. In the early part of this unit we learnt how registers are generally built from banks Flip Flops, each storing a single bit. Here we see how registers serve the needs of the programs we seek to write and execute.

The ARMlite simulator provides 16 registers, including 13 so called *general purpose registers* which can be used within the programs we write.

The general purpose registers are labelled, R0-R12 as shown in the image below.



As with main memory storage in ARMlite, registers each hold 32 bits, with the value in each register represented by an 8 digit hexidecimal value.

In lectures we introduced the MOV instruction for storing values in registers. For example:

MOV R1, #15

takes the decimal value 15 and stores it in register R1.

MOV R2, R1

takes the value stored in R1 and copies it to register R2.

We also introduced some basic arithmetic operations like ADD (for adding) and SUB (for subtracting).

We are not going to write a game in assembly language (yet!), but rather, we are going to play a game that involves assembly programming.

Task: You are given 6 input values and a target value. Your task is to write a simple assembly program that implement a mathematical equation involving only these 6 input values, and the three instructions MOV, ADD and SUB, such that the result is as close as possible to the target value.

Here is an example to get you started. Your initial input numbers are 100, 25,8,4,3,1 and your target is 84. This is pretty straight forward from a mathematical perspective: 1+8+100-25 = 84

A possible implementation of this equation in ARM assembly code would be:

MOV R0,#1

ADD R1,R0,#8

ADD R2,R1,#100

SUB R3,R2,#25

HALT

Clear the Program window in ARMlite and type in the lines of code above. When done, Submit the code ready for execution (and if any errors occur, check your syntax matches the above).

Now, using the Step button (rather than the Run button), execute the first instruction, MOV R0, #1

Verify R0 contains the value 0x00000001

The second instruction is ADD R1,R0,#8.

4.5.1 Before executing this instruction, describe in words what you think this instruction is going to do, and what values you expect to see in R0 and R1 when it is complete ?

Trả lời:

MOV R0, #1

- Hàm MOV di chuyển giá trị 1 vào thanh ghi R0, khi lệnh này thực hiện, giá trị trong R0 sẽ là 1.

ADD R1, R0, #8

- Hàm ADD cộng giá trị trong thanh ghi R0 với hằng số 8 và lưu kết quả vào thanh ghi R1, với R0 có giá trị 1, lệnh này sẽ tính toán 1 + 8 = 9. Vậy giá trị trong R1 sẽ là 9.

ADD R2, R1, #100

- Lệnh này cộng giá trị trong thanh ghi R1 với hằng số 100 và lưu kết quả vào thanh ghi R2, với R1 có giá trị 9, lệnh này sẽ tính toán 9 + 100 = 109. Giá trị trong R2 sẽ là 109.

SUB R3, R2, #25

- Hàm SUB trừ hằng số 25 từ giá trị trong thanh ghi R2 và lưu kết quả vào thanh ghi R3, với R2 có giá trị 109, lệnh này sẽ tính toán 109 - 25 = 84. Vậy giá trị trong R3 sẽ là 84.

HALT

Sau khi lệnh HALT được thực hiện, chương trình sẽ ngừng lại.

Vậy ta có các giá trị trong các thanh ghi

R0 = 1

R1 = 9

R2 = 109

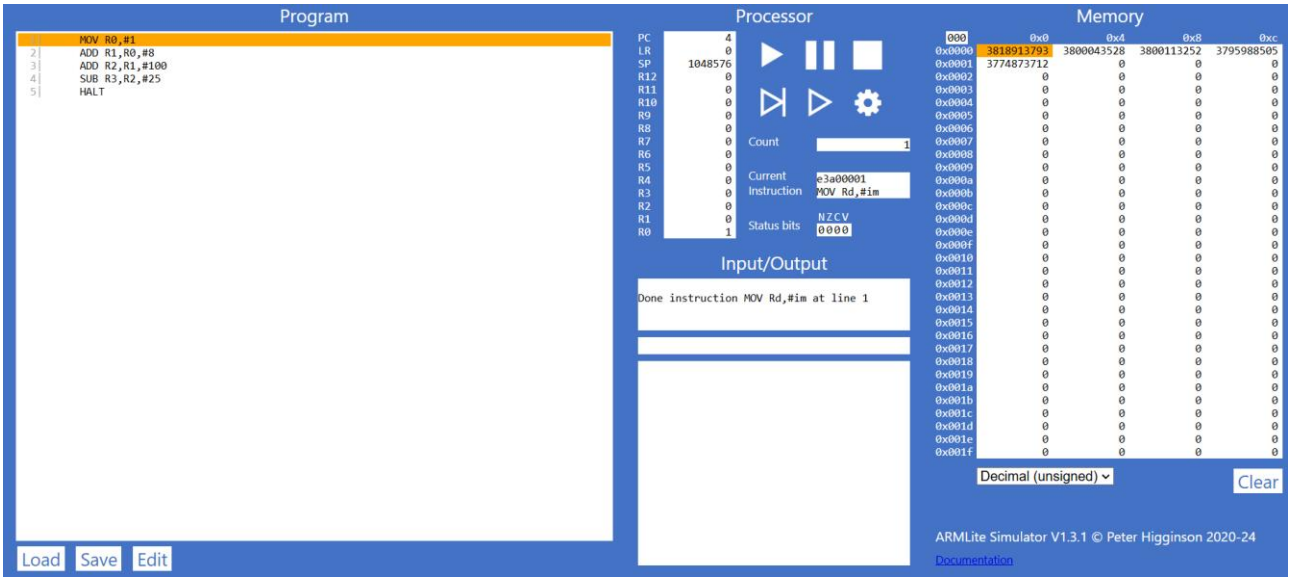
R3 = 84

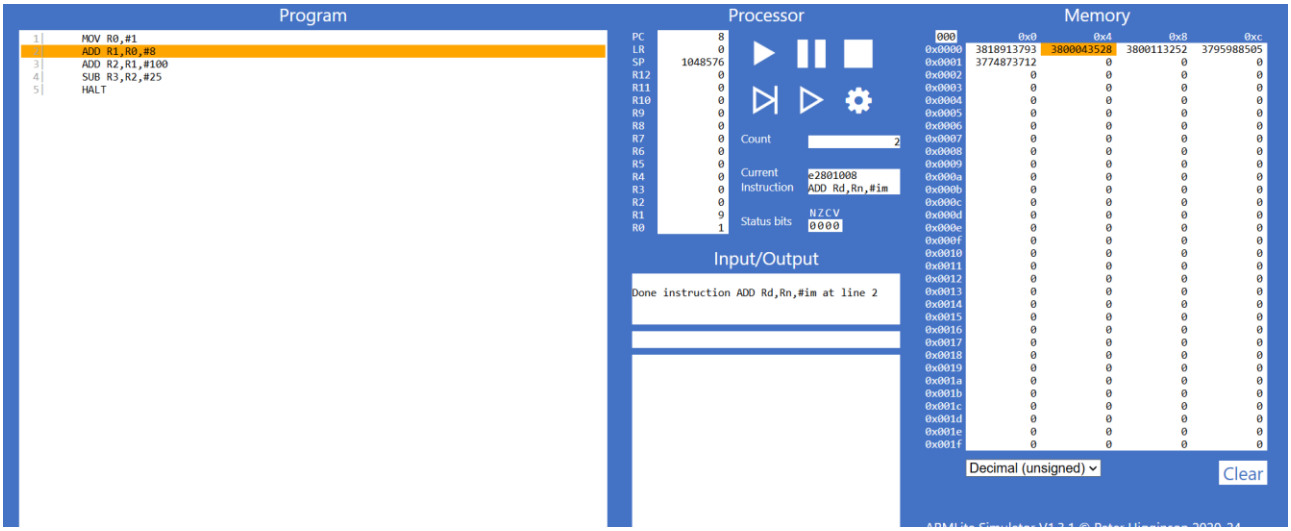
Now execute the instruction and verify whether the output matches your expectation.

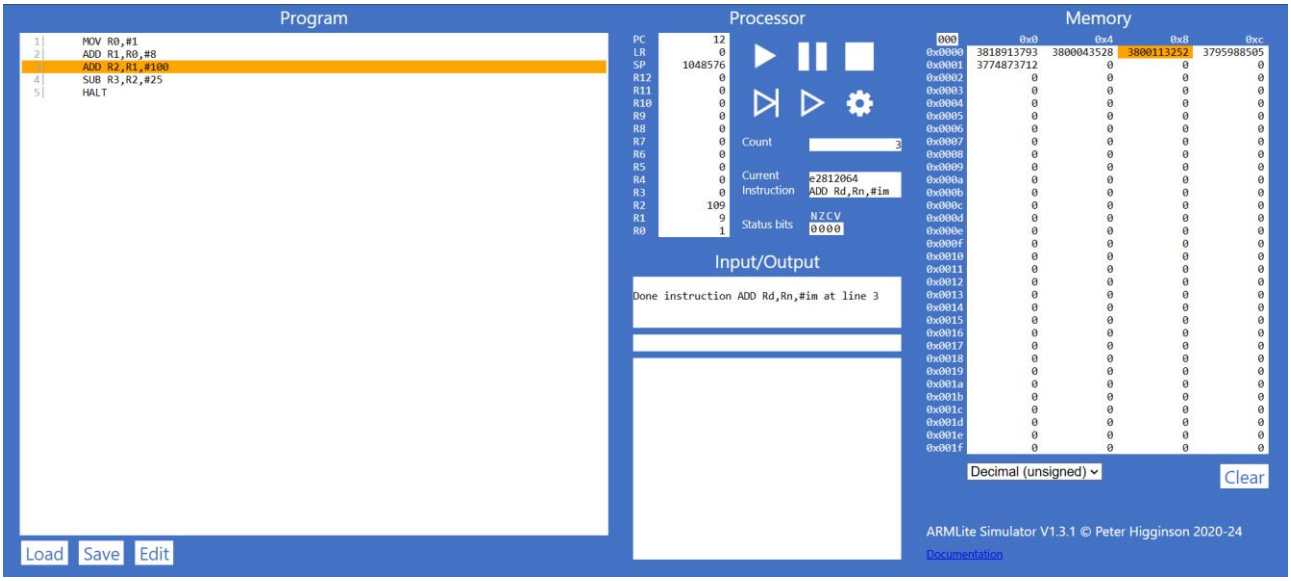
Do the same for the remaining instructions.

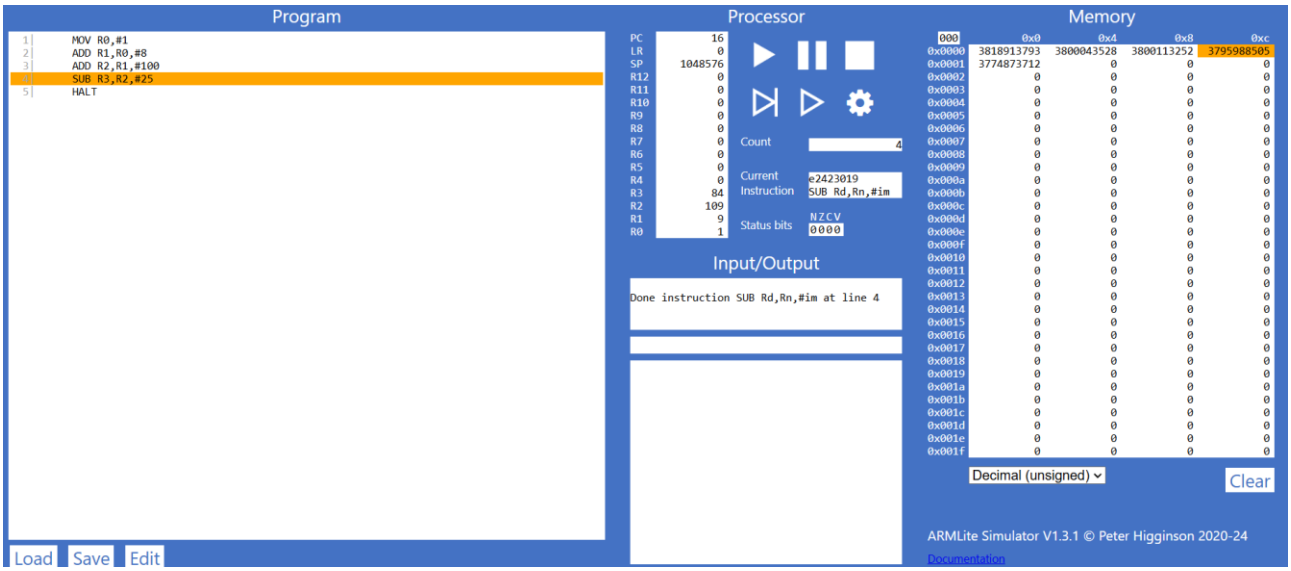
4.5.2 When the program is complete, take a screen shot of the register table showing the values.

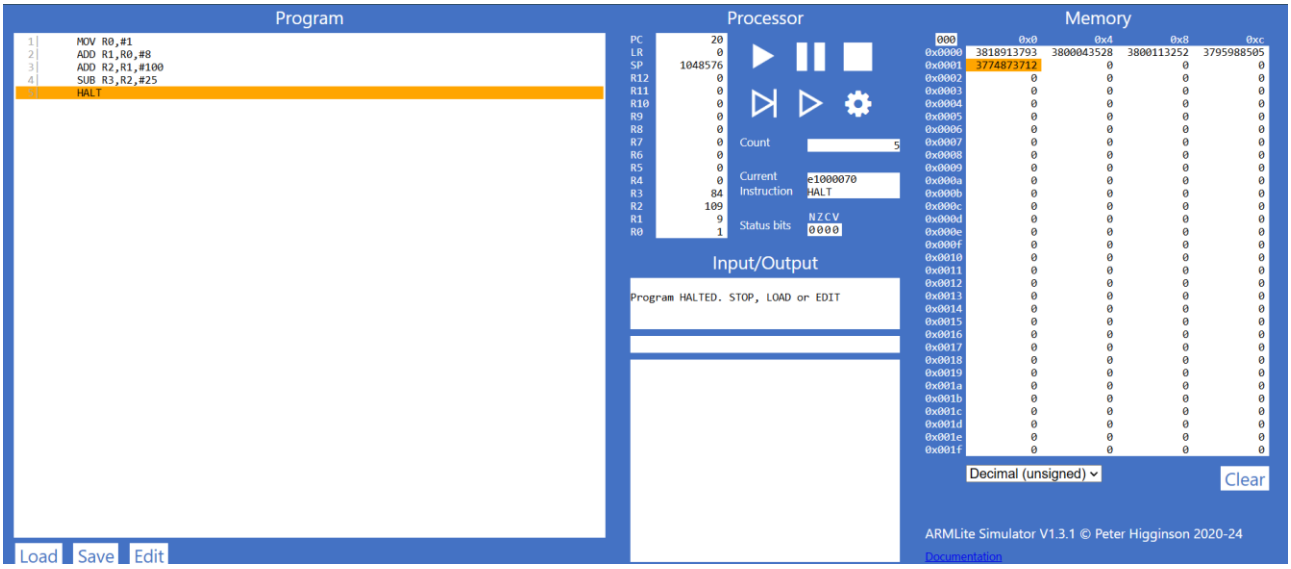
Trả lời:











*You will have noticed the **HALT** instruction, which does the obvious task of halting the program. This instruction is important for telling the program counter (which automatically steps through the program instructions one-by-one to cease doing so. Without this instruction, program execution would simply continue to sequentially read and execute 32 bit words of memory beyond the last address of the loaded program code. In practise, this will most likely result in the program simply not working, however on a real microprocessor, can be potentially dangerous due to the possibility of executable code residing in memory from previously loaded code - this can result in unpredictable program behaviour.*

4.5.3 Task: ***Your 6 initial numbers are now 300, 21, 5, 64, 92, 18. Write an Assembly Program that uses these values to compute a final value of 294 (you need only use MOV, ADD and SUB). Place your final result in register R4 (don't forget the HALT instruction)***

When the program is complete, take a screen shot of the code and the register table.

Trả lời:

Chương trình
MOV R0, #300

SUB R0, R0, #21

ADD R0, R0, #5

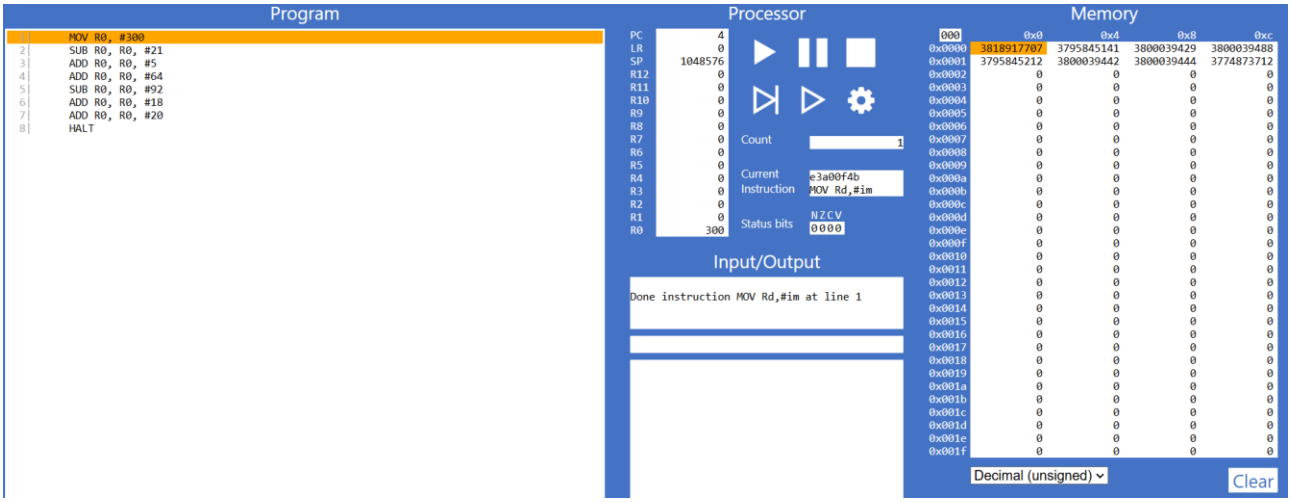
ADD R0, R0, #64

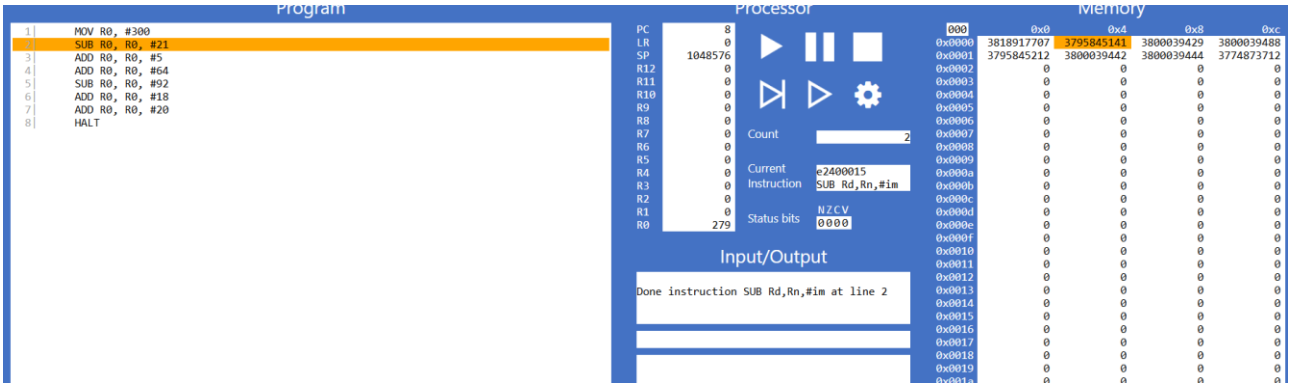
SUB R0, R0, #92

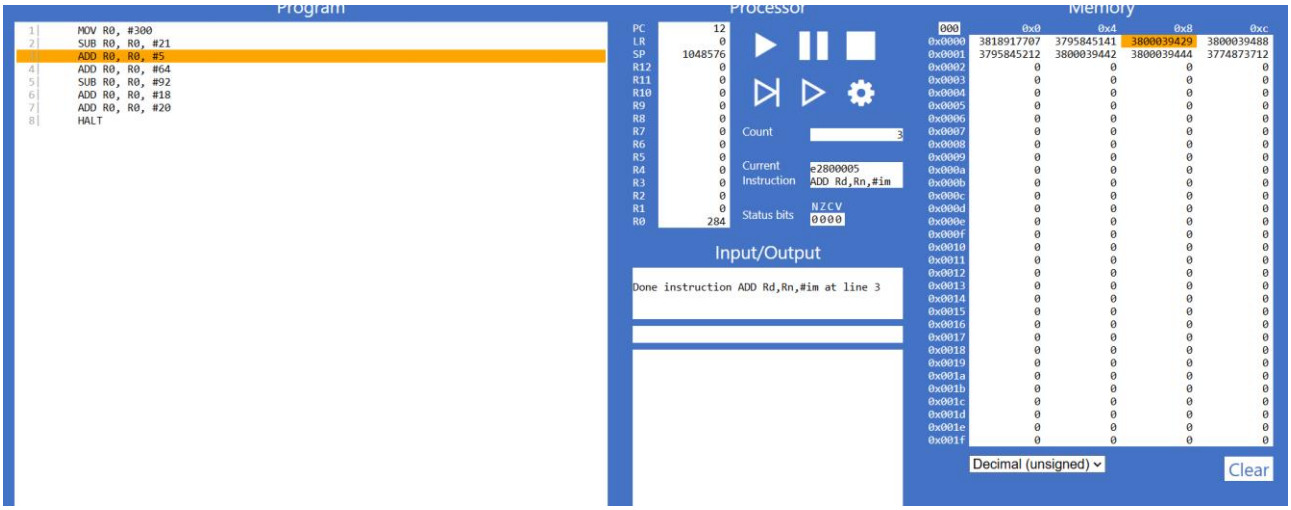
ADD R0, R0, #18

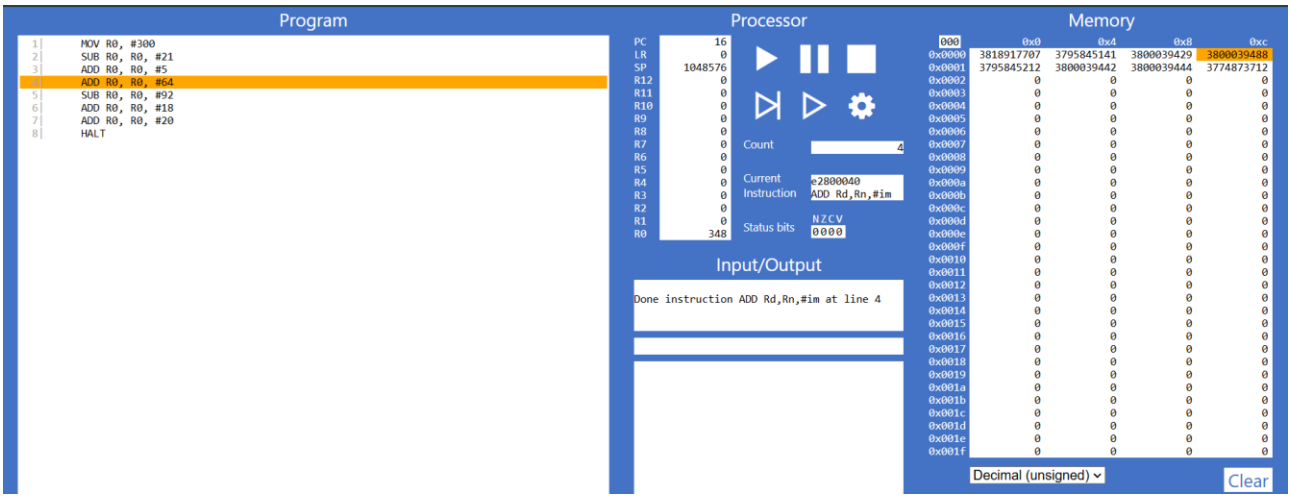
ADD R0, R0, #20

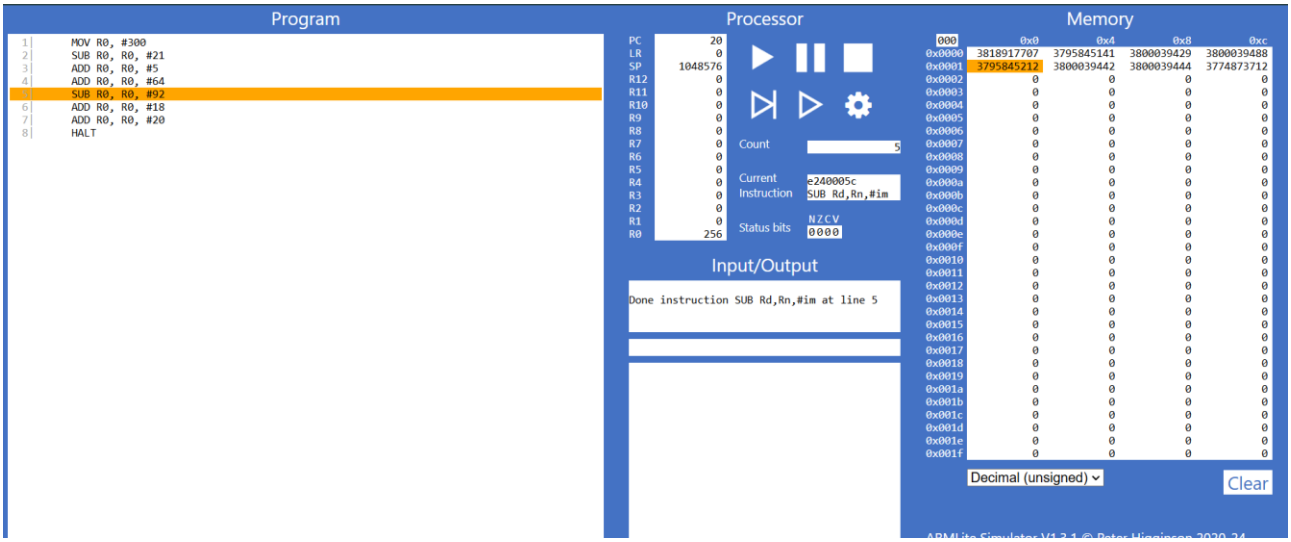
HALT

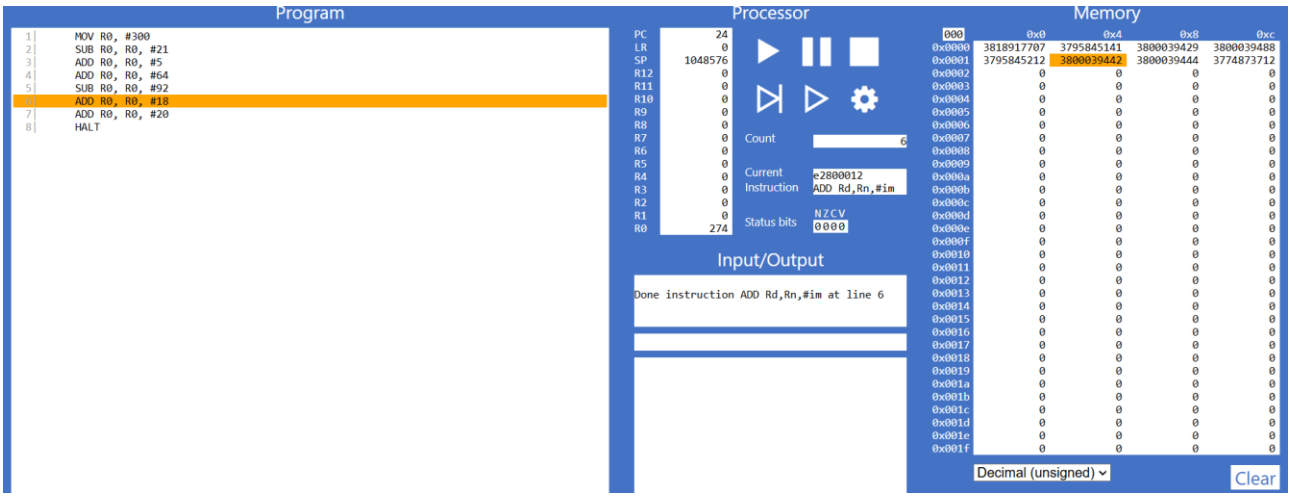


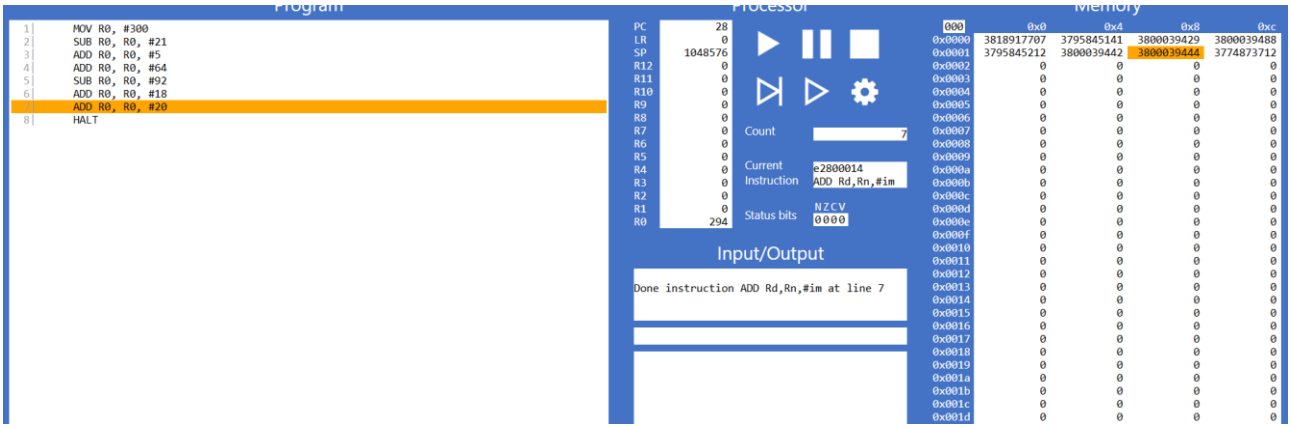


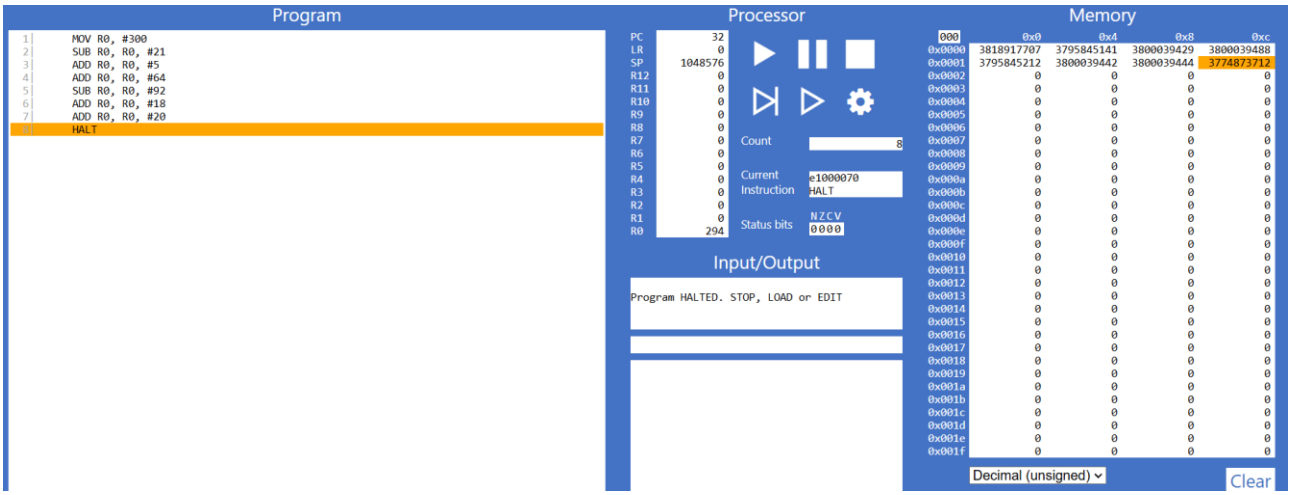

Program
1 MOV R0, #300
2 SUB R0, R0, #21
3 ADD R0, R0, #5
4 ADD R0, R0, #64
5 SUB R0, R0, #92
6 ADD R0, R0, #18
7 ADD R0, R0, #20
8 HALT
Processor
PC 12
LR 0
SP 1048576
R12 0
R11 0
R10 0
R9 0
R8 0
R7 0
R6 0
R5 0
R4 0
R3 0
R2 0
R1 0
R0 284
Count 3
Current Instruction e2800005 ADD Rd,Rn,#im
Status bits NZCV 0000
Input/Output
Done instruction ADD Rd,Rn,#im at line 3
Memory
000 0x0 0x4 0x8 0xc
0x0000 3818917707 3795845141 3800039429 3800039488
0x0001 3795845212 3800039442 3800039444 3774873712
Decimal (unsigned)
Clear


Program
1 MOV R0, #300
2 SUB R0, R0, #21
3 ADD R0, R0, #5
4 ADD R0, R0, #64
5 SUB R0, R0, #92
6 ADD R0, R0, #18
7 ADD R0, R0, #20
8 HALT
Processor
PC 16
LR 0
SP 1048576
R0 348
Count 4
Current Instruction e2800040 ADD Rd,Rn,#im
Status bits NZCV 0000
Input/Output
Done instruction ADD Rd,Rn,#im at line 4
Memory
000 0x0 0x4 0x8 0xc
0x0000 3818917707 3795845141 3800039429 3800039488
0x0001 3795845212 3800039442 3800039444 3774873712
Decimal (unsigned)
Clear


Program
1 MOV R0, #300
2 SUB R0, R0, #21
3 ADD R0, R0, #5
4 ADD R0, R0, #64
5 SUB R0, R0, #92
6 ADD R0, R0, #18
7 ADD R0, R0, #20
8 HALT
Processor
PC 20
LR 0
SP 1048576
R0 256
Count 5
Current Instruction e240005c SUB Rd,Rn,#im
Status bits NZCV 0000
Input/Output
Done instruction SUB Rd,Rn,#im at line 5
Memory
000 0x0 0x4 0x8 0xc
0x0000 3818917707 3795845141 3800039429 3800039488
0x0001 3795845212 3800039442 3800039444 3774873712
Decimal (unsigned)
Clear


Program
1 MOV R0, #300
2 SUB R0, R0, #21
3 ADD R0, R0, #5
4 ADD R0, R0, #64
5 SUB R0, R0, #92
6 ADD R0, R0, #18
7 ADD R0, R0, #20
8 HALT
Processor
PC 24
LR 0
SP 1048576
R0 274
Count 6
Current Instruction e2800012 ADD Rd,Rn,#im
Status bits NZCV 0000
Input/Output
Done instruction ADD Rd,Rn,#im at line 6
Memory
000 0x0 0x4 0x8 0xc
0x0000 3818917707 3795845141 3800039429 3800039488
0x0001 3795845212 3800039442 3800039444 3774873712
Decimal (unsigned)
Clear





Hướng dẫn	Giá trị thập phân của thanh ghi đích sau khi thực hiện hướng dẫn
MOV	300
SUB	279
ADD	284
ADD	348
SUB	256
ADD	274
ADD	294
HALT	exit

In this week's lectures we also introduced a small set of so-called Bit-wise instructions, designed to manipulate bits within a register in specific (and highly useful ways).

Recall the following:

Instruction	Example	Description
AND	AND R2, R1, #4	Performs a bit-wise logical AND on the two input values, storing the result in the equivalent bit of the destination register.
ORR	ORR R1, R3, R5	As above but using a bit-wise logical OR
EOR	EOR R1, R1, #15	As above but using a bit-wise logical ‘Exclusive OR’

LSL	LSL R1, R1, #3	'Logical Shift Left'. Shifts each bit of the input value to the left, by the number of places specified in the third operand, losing the leftmost bits, and adding zeros on the right.
LSR	LSR R1, R1, R2	'Logical Shift Right'. Shifts each bit of the input value to the right, by the number of places specified in the third operand, losing the right-most bits, and adding zeros on the left.

4.5.4 Task: ***Write your own simple program, that starts with a MOV (as in the previous example) followed by five instructions, using each of the five new instructions listed above, once only, but in any order you like – plus a HALT at the end, and with whatever immediate values you like.***

Trả lời:

Chương trình

```
MOV R0, #12
AND R1, R0, #15
ORR R2, R0, #20
EOR R3, R0, #5
LSL R4, R0, #6
LSR R5, R0, #4
HALT
```

Note: Keep all your immediate values less than 100 (decimal). Also, when using LSL, don't shift more than, say #8 places. Using very large numbers, or shifting too many places to the left, runs the risk that you will start seeing negative results, which will be confusing at this stage. (We'll be covering negative numbers in the final part of this chapter.)

You may use a different destination register for each instruction, or you may choose to use only R0, for both source and destination registers in each case - both options will work.

Enter your program into ARMlite, submit the code and when its ready to run, step through the program, completing the table below (make a copy of it in your submission document)

Instruction	Decimal value of the destination register after executing this instruction	Binary value of the destination register after executing this instruction
MOV	12	0b00000000000000000000000000001100
AND	12	0b00000000000000000000000000001100
ORR	28	0b00000000000000000000000000011100
EOR	9	0b00000000000000000000000000001001
LSL	768	0b00000000000000000000001100000000
LSR	0	0b00000000000000000000000000000000

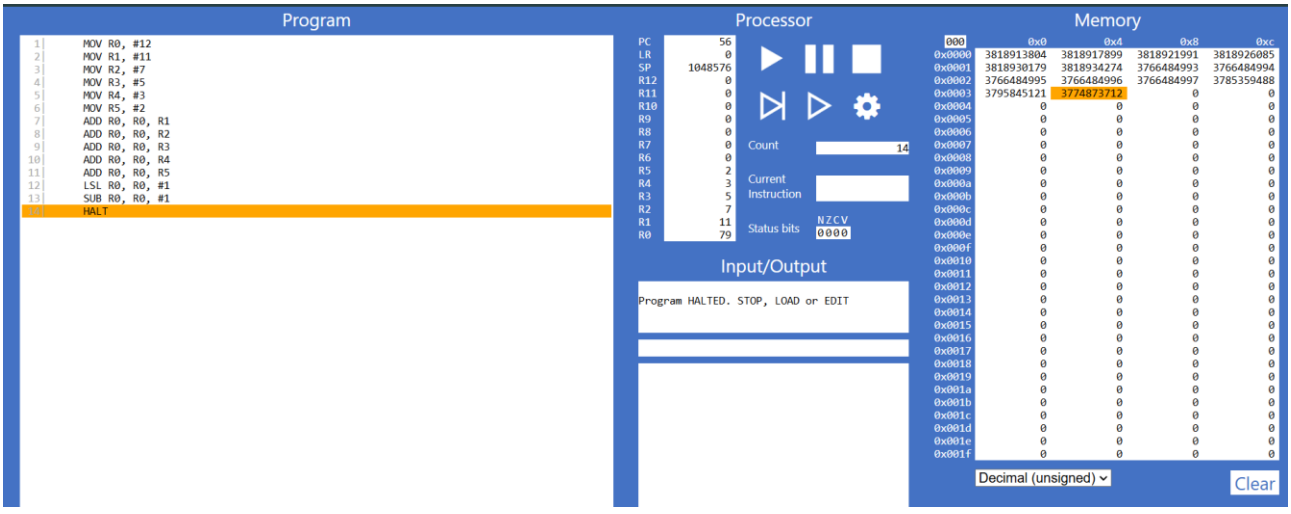
Task 4.5.5 Lets play the game we played in 4.5.3, but this time you can use any of the instructions listed in this lab so far (ie,. MOV, AND, OR, and any of the bit-wise operators).

Your six initial numbers are: 12, 11, 7, 5, 3, 2 and your target number is: 79

Trả lời:

Chương trình

```
MOV R0, #12
MOV R1, #11
MOV R2, #7
MOV R3, #5
MOV R4, #3
MOV R5, #2
ADD R0, R0, R1
ADD R0, R0, R2
ADD R0, R0, R3
ADD R0, R0, R4
ADD R0, R0, R5
LSL R0, R0, #1
SUB R0, R0, #1
HALT
```



When the program is complete, take a screen shot of the code and the register table and paste into your submission document.

Task 4.5.6: Let's play again !

Your six initial numbers are: 99, 77, 33, 31, 14, 12 and your target number is: 32

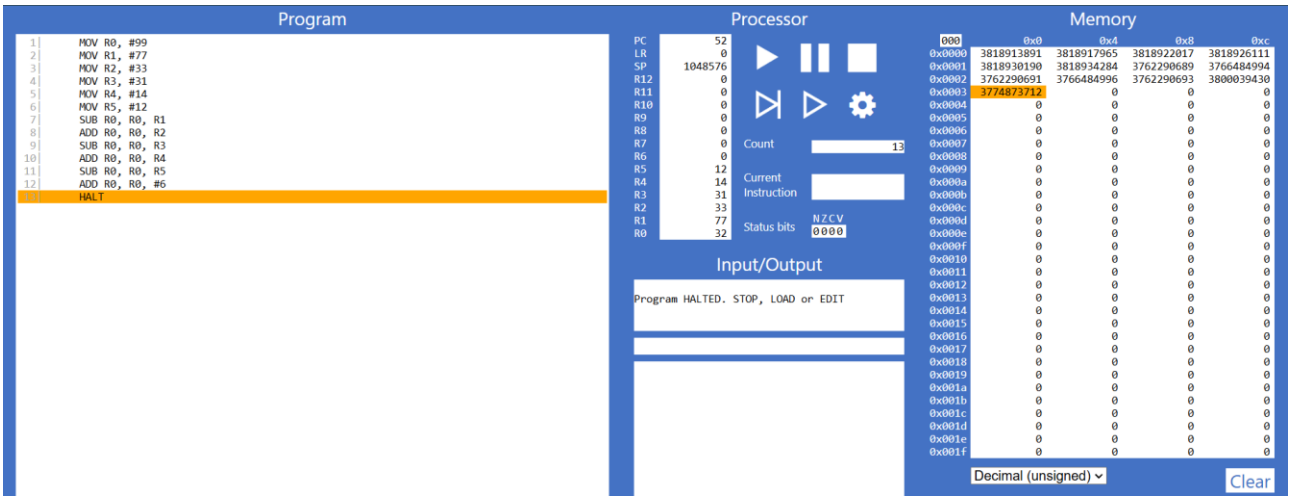
When the program is complete, take a screen shot of the code and the register table and paste into your submission document.

Trả lời:

Chương trình

```
1|      MOV R0, #99
```

```
 2|      MOV R1, #77
 3|      MOV R2, #33
 4|      MOV R3, #31
 5|      MOV R4, #14
 6|      MOV R5, #12
 7|      SUB R0, R0, R1
 8|      ADD R0, R0, R2
 9|      SUB R0, R0, R3
10|      ADD R0, R0, R4
11|      SUB R0, R0, R5
12|      ADD R0, R0, #6
13|      HALT
```



Part 4.6 Signed Integers

Copy and Paste the following code into the ARMlite code editor and submit the code.

MOV R0, #9999

LSL R1, R0, #18

HALT

Before executing, switch ARMlite to display data in memory in Decimal (signed) using the drop down box below the memory grid.

Now run the program and note the result in register R1.

4.6.1 - Why is the result shown in R1 a negative decimal number, and with no obvious relationship to 9999 ?

Hint: Mouse over the values in R0 and R1 and take a look at the binary strings.

Switch ARMlite to display in Binary format using the dropdown box under the memory grid.

You can't edit register values directly, but you can edit memory words. Click on the top-left memory word (address 0x00000) and type in the following values, which will be interpreted as decimal and translated into the 32-bit two's complement format, which you can then copy back into your answers.

Trả lời:

- MOV R0, #9999: gán giá trị 9999 vào R0

- LSL R1, R0,#18: dịch bit R0 sang trái 18 lần, tương đương 9999 x 2^18 = 2,621,177,856

- Khi giá trị tính toán vượt quá giới hạn của số dương (2,147,483,647), bit cao nhất (bit thứ 31) sẽ trở thành 1. Theo quy tắc bù 2 thì bit cao nhất là một thì biểu thị số này là số âm.

4.6.3 - What is the binary representation of each of these signed decimal numbers: 1, -1, 2, -2

What pattern do you notice ? Make a note of these in your submission document before reading on.

The ARMlite simulator is using 2's Compliment to represent signed integer values. Recall from lectures how 2's Compliment works to get the negative version of a number:

1. invert (or 'flip') each of the bits
2. Add 1

Trả lời:

1: 0000 0000 0000 0000 0000 0000 0000 0001

-1: 1111 1111 1111 1111 1111 1111 1111 1111

2: 0000 0000 0000 0000 0000 000 0000 0010

-2: 1111 1111 1111 1111 1111 1111 1111 1110

- Khi đưa một số về số âm nó sẽ flip tát cả các bit (từ 1 thành 0 và 0 thành 1) sau đó cộng thêm 1 bit.

Ví dụ: là số 4 có bit là 0000 0000 0000 0000 0000 0000 0000 0100

Khi đưa về -4 thì flip: 1111 1111 1111 1111 1111 1111 1111 1011 + 1

Kết quả trả về: : 1111 1111 1111 1111 1111 1111 1111 1100

4.6.4 - Write an ARM Assembly program that converts a positive decimal integer into its negative version. Start by moving the input value into R0, and leaving the result in R1.

Hint: the ARM instruction MVN, which works like MOV but flips each bit in the destination register, may be useful for this !

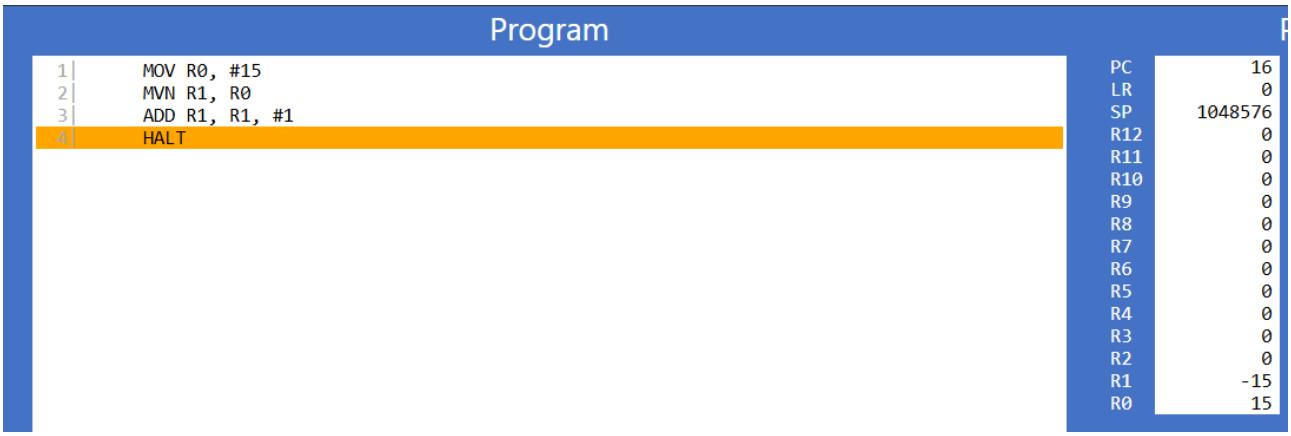
Take a screen shot showing your program and the registers after succesful execution, and paste into your submission document.

Using the program you wrote above, enter the negative version of the number you previously tested as the input (ie use the output of the previous test as the input).

What do you notice ?

All things working as they should be, you should see that the exact same 2's Compliment conversion works in both directions (ie positive to negative, and negative to positive).

Trả lời:



MOV input là số dương hoặc âm bất kì. Dùng MVN để flip các bit và dùng ADD để cộng thêm 1. Với chương trình trên nó có thể biến số âm thành dương và ngược lại.