<u>Study Questions (Part 7)</u>
<u>Sunday March 31, 2019</u>

<u>Covering:</u>

- Stack, stack frame, call by value, call by reference

<u>Question number 1 was originally part of assignment 5, but it was decided to shift it to the study questions, due to the shortage of time.</u>

1. A linked list is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a data element and a *link* to the next node in the sequence Linked lists allow for efficient insertion or removal of elements from any position in the sequence.

   Assume that we have a linked list, whose first node is pointed at by register `r0`. Each node in this list composed of a single 32-bit data element and a 32-bit address (*link*) pointing to the next node in the list. The last node in the list points to the null address 0.

   Draw ***<u>a detailed flowchart</u>*** and write an ARM assembly ***<u>program</u>*** to search a linked list for the first node in the list whose data is equal to the data in register `r1`. On success, set register `r2` to `0xFFFFFFFF`, and register `r3` should contain the address of the desired node. On failure, set register `r2` to `0xF0F0F0F0`.

   Your code should be highly optimized, i.e., use as little number of instructions as possible.

   Hint, for testing purpose, you can represent a linked list as follow:

```
List   DCD 0x12341111, Item5
Item2 DCD 0x12342222, Item3
Item3 DCD 0x12343333, Item4
Item4 DCD 0x12344444, Item6
Item5 DCD 0x12345555, Item2
Item6 DCD 0x12346666, Item7
Item7 DCD 0x12347777,  0x00    ;terminator
```

   You should change the number of nodes and the value of the data elements in the list to test various cases, including empty linked list, one element linked list, and long linked list. You should also test your program by searching for values in various locations in the list, as well as values not in the list.

2. Using only 3 ARM instructions, show how to store the current value of the frame pointer in the stack, create a 16-byte stack frame **<u>and</u>** make the *frame pointer* to point to the base of this frame, **<u>while</u>** the *stack pointer* to point to the top of the stack. A*ssume that an **FD** stack is in use, an appropriate stack space is already allocated to the stack, and the stack pointer is appropriately initialized*.

3. Using only 2 ARM instructions show how to collapse the above created 16-byte stack frame and restore the original values of the *frame pointer* and the *stack pointer*. Assume that an ***FD*** stack is in use.

4. Using only 3 ARM instructions, show how to store the current value of the frame pointer in the stack, create a 16-byte stack frame **<u>and</u>** make the *frame pointer* to point to the base of this frame, **<u>while</u>** the *stack pointer* to point to the top of the stack. A*ssume that an **FA** stack is in use, an appropriate stack space is already allocated to the stack, and the stack pointer is appropriately initialized*.

5. Using only 2 ARM instructions show how to collapse the above created 16-byte stack frame and restore the original values of the *frame pointer* and the *stack pointer*. Assume that an ***FA*** stack is in use.

6. Using only 3 ARM instructions, show how to store the current value of the frame pointer in the stack, create a 16-byte stack frame **and** make the *frame pointer* to point to the base of this frame, **while** the *stack pointer* to point to the top of the stack. A*ssume that an* ***ED*** *stack is in use, an appropriate stack space is already allocated to the stack, and the stack pointer is appropriately initialized*.

7. Using only 2 ARM instructions show how to collapse the above created 16-byte stack frame and restore the original values of the *frame pointer* and the *stack pointer*. Assume that an ***ED*** stack is in use.

8. Using only 3 ARM instructions, show how to store the current value of the frame pointer in the stack, create a 16-byte stack frame **and** make the *frame pointer* to point to the base of this frame, **while** the *stack pointer* to point to the top of the stack. A*ssume that an* ***EA*** *stack is in use, an appropriate stack space is already allocated to the stack, and the stack pointer is appropriately initialized*.

9. Using only 2 ARM instructions show how to collapse the above created 16-byte stack frame and restore the original values of the *frame pointer* and the *stack pointer*. Assume that an ***EA*** stack is in use.

10. Draw a sketch to show the relative locations inside the stack of a stack frame that contains 3 pushed registers (r2, the FP, and the LR) and 8 bytes of local variables, as well as the relative locations of the 12 bytes of passed parameters. Make the *frame pointer* to point to the base of this frame. Assume an FD stack is used.

11. Draw a sketch to show the relative locations inside the stack of a stack frame that contains 3 pushed registers (r2, the FP, and the LR) and 8 bytes of local variables, as well as the relative locations of the 12 bytes of passed parameters. Make the *frame pointer* to point to the base of this frame. Assume an ED stack is used.

12. Draw a sketch to show the relative locations inside the stack of a stack frame that contains 3 pushed registers (r2, the FP, and the LR) and 8 bytes of local variables, as well as the relative locations of the 12 bytes of passed parameters. Make the *frame pointer* to point to the base of this frame. Assume an EA stack is used.

13. Draw a sketch to show the relative locations inside the stack of a stack frame that contains 3 pushed registers (r2, the FP, and the LR) and 8 bytes of local variables, as well as the relative locations of the 12 bytes of passed parameters. Make the *frame pointer* to point to the base of this frame. Assume an FA stack is used.

14. Write ARM code to implement the following C operation

```
int increment(int  *var)
{
   *var = *var+10;
   return *var;
}

int main()
{
     int num1=20;
     int num2;
     num2 = increment(&num1);
}
```

15. Write ARM code to implement the following C operation
```
void increment(int  *var)
{
   *var = *var+10;
   return;
}

int main()
{
     int num1=20;
     int num2;
     increment(&num1);
     num2= num1;
     return;
}
```

16. Write ARM code to implement the following C operation
```
void increment(int  var)
{
   var = var+10;
   return;
}

int main()
{
     int num1=20;
     int num2;
     increment(num1);
     num2= num1;
     return;
}
```

17. Write ARM code to implement the following C operation
```
int increment(int  var)
{
   var = var+10;
   return var;
}

int main()
{
     int num1=20;
     int num2;
     num2 = increment(num1);
}
```