Answers to Self-Test Questions

Tutorial 10: Storing Values in Buffers

……To Go Back to QUESTIONS Click Here

1.	A *buffer* is an area of data memory. It is commonly used to store data for transmission or to receive data from an external device.
2.	A buffer can be implemented in C using an array e.g. `int buffer[20]; /* An array of 20 values */`
3.	When using the instruction `DMOV` with a repeat, we start from the high end and work towards the low end. If we were to work the other way round, then the whole buffer would be incorrectly overwritten with the first value.
4.	A straight buffer useful in DSP because it maps well to the *z* transform. It is widely used for implementing Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters. The architecture of the TMS320F24x has features built in to support straight buffers.
5.	The instruction `DMOV` is not often used on its own because it can be combined together with other instructions e.g. `LTD`

6.	The following three instructions
	`APAC`
	`LT *`
	`DMOV *`
	can be combined together to form the instruction `LTD *`

7.	If we were to use the instruction `DMOV` on the highest element of the buffer, then the value would be moved to a location past the end of the buffer. This is likely to corrupt another value stored in data memory.
8.	When using a buffer, indirect addressing allows us to increment or decrement a pointer as part of the instruction. This allows us to use a loop; something not possible using direct addressing.
9.	We would place a dummy value at the end of an array to allow us to move the highest value off the end of the array. This gives us simpler code, at the expense of an extra data memory location.

10.	The correct order is `label, *, ARx`
	`a) BANZ label, *-, AR3 ; Correct.`
	`b) BANZ label, AR3, *- ; AR3 should be last.`
	`c) BANZ *- , label, AR3 ; *- should be second.`
	`d) BANZ *-, AR3, label ; *- should be second.`
	`e) BANZ AR3, label, *- ; Label should be first.`
	`f) BANZ AR3, *-, label ; AR3 should be last.`

11.	The instruction `BANZ` means Branch on Auxiliary register Non-Zero.

12.	When we multiply two signed 16-bit values, the 32-bit product consists of 2 sign bits and 30 data bits. We only need one sign bit, so a shift to the left loses one of the sign bits and gives us increased resolution. The usage of this shift is under control of `SPM`.
13.	The instruction `SPM` means Set P Mode. It determines what shift should be applied to the contents of the P register before accumulation.
14.	In order to add a value of greater than FFFFh to the accumulator, we need to maintain the sign bit. If we are using a positive value, then we want sign-extension mode turned off i.e. `CLRC SXM`. If we are using a negative value, we want sign-extension mode turned on i.e. `SETC SXM`.