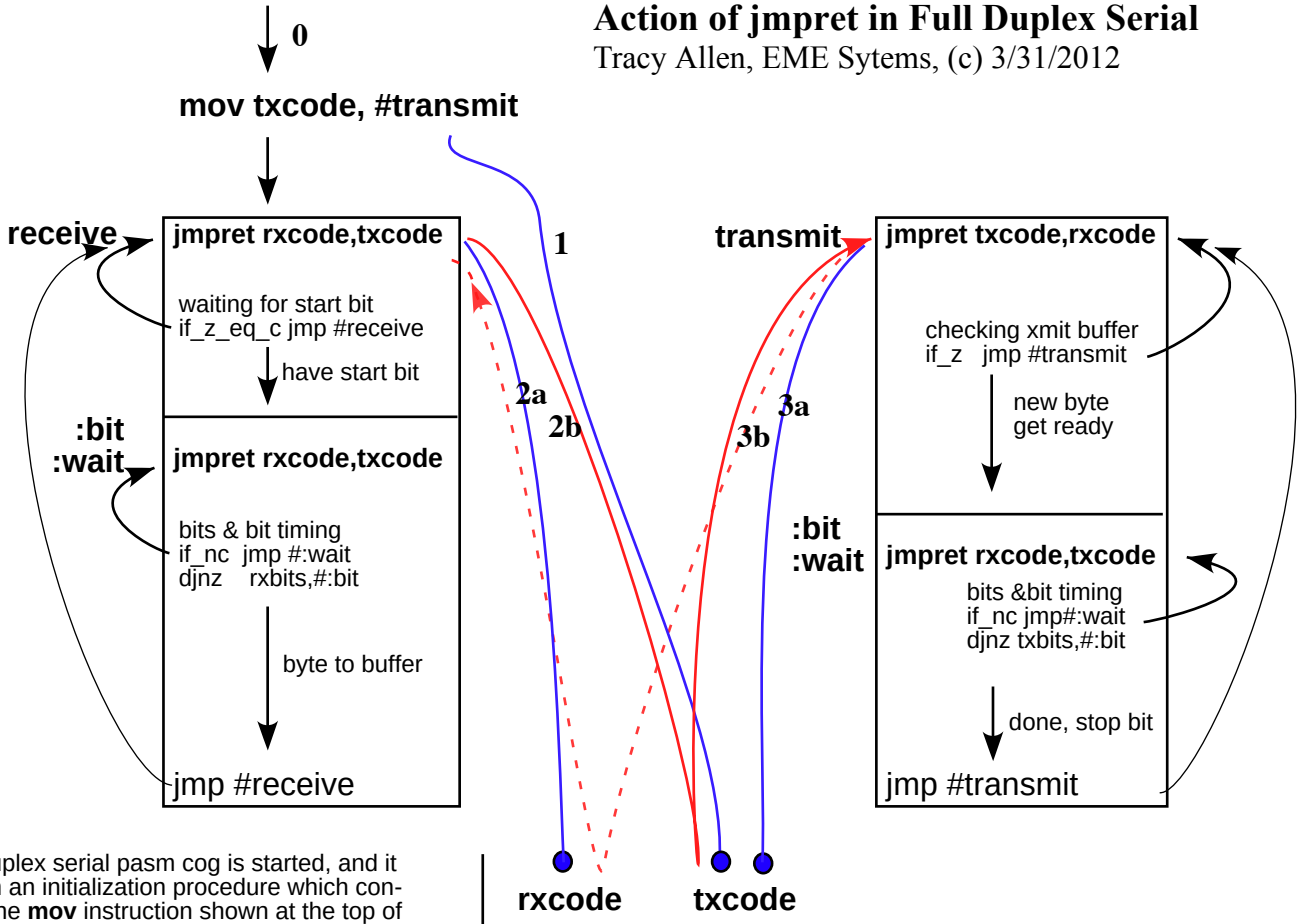
Action of jmpret in Full Duplex Serial

Tracy Allen, EME Sytems, (c) 3/31/2012



0) The full duplex serial pasm cog is started, and it goes through an initialization procedure which concludes with the **mov** instruction shown at the top of the diagram. The jmpret mechanism depends on two cog registers, rxcode and txcode that will be used to store addresses for indirect jumps.

1) the **mov txcode, #transmit** instruction plugs the address of the transmit routine into the register txcode.
2a) The **jmpret rxcode,txcode** plugs the address of receive+1 into the source field of register rxcode, and, more or less at the same time...
2b) jumps indirectly to transmit, using the value found in the source field of register txcode (just installed there by the previous **mov**.)
3a)At transmit now, its **jmpret txcode,rxcode** plugs the address of transmit+1 into the source field of register txcode, and then...
3b) jumps indirectly using the value stored in the source field of rxcode, which takes it to receive+1.

Now the ping-pong is primed. Execution bounces between the two co-routines with granularity of microseconds, waiting for a start bit or for a byte to transmit. Each time around each loop, they hit their respective **jmpret,** there again load the $+1 address into the source field of rxcode or txcode, and jump via those pointers to transmit+1 or receive+1.

The chart shows what might be stored in rxcode and txcode as time progresses. Same mechanism. Suppose a start bit is detected. The receive half drops on through and when it hits the second **jmpret**, it will store :wait+1 in rxcode. And if while it is receiving, it happens that a byte becomes available for transmission, the transmit routine will be storing its own :wait+1 address in txcode.

When finished with a complete byte received or sent, each routine executes a direct **jmp #receive** or **jmp #transmit** and returns to the top of its loop, waiting for again for the start bit or byte to send.

rxcode	txcode	
xxxxxxxx	#transmit	priming
#receive+1	#transmit	
#receive+1	#transmit+1	
#receive+1	#transmit+1	idling
#receive+1	#transmit+1	
#receive+1	#transmit+1	
...	...	
#:wait+1	#transmit+1	
#:wait+1	#transmit+1	start bit detected, bits received
#:wait+1	#transmit+1	
#:wait+1	#transmit+1	
#:wait+1	#transmit+1	
...	...	
#:wait+1	#:waitt+1	
#:wait+1	#:waitt+1	
#:wait+1	#:waitt+1	byte to transmit, still receiving
#:wait+1	#:waitt+1	
#:wait+1	#:waitt+1	
#:wait+1	#:waitt+1	
...	...	
#receive+1	#:wait+1	
#receive+1	#:wait+1	
#receive+1	#:wait+1	
#receive+1	#:wait+1	byte received, still transmitting
#receive+1	#:wait+1	
...	...	
#receive+1	#transmit+1	
#receive+1	#transmit+1	
#receive+1	#transmit+1	idling
#receive+1	#transmit+1	
...	...	