

**Micro-code for Booths Algorithm Stack Multiply on the MIC-1**

We use H to hold the Multiplicand since it is the only register that can be subtracted. The TOS register will act as the accumulator since it will hold the MSW of the result at the end, which will be placed at the top of stack anyway. The MDR register will act as the multiplier since it is already loaded with the second stack entry when it is popped, and holds the LSW of the result at the end ready to be pushed. We will use the OPC register as a counter on the multiply operations. Initializing the counter is the most awkward part of the algorithm.

Mult:	OPC=1, >>8	OPC = 256
	OPC=OPC,SRA1	OPC=128
	OPC=OPC,SRA1	OPC=64
	OPC=OPC,SRA1	OPC=32
	MAR=SP-1, Read	Read 2 <sup>nd</sup> stack word (X)
	H=TOS	MDR now contains X, H contains Y
	TOS =0	Clear Accum. Just handled a 0 by Def.
M0x:	ALU=MDR, SRA1, if(notLSB) gotoM00	Just handled a zero. Test for (0,0) or (1,0)
M10:	TOS=TOS-H, SRA1	1,0-Subtract Y, start long shift
	MDR=MDR, RROT	Complete shift. LSB has next recode bit
	OPC=OPC-1, if Z goto End0	Decr. Cntr. Go to "just handled a 1"
M1x:	ALU=MDR,SRA1, if (notLSB) gotoM01	Just handled a one. Test for (0,1) or (1,1)
M11:	TOS=TOS,SRA1, goto (M10+1)	1,1 case - Start long shift
M01:	TOS=TOS+H, SRA1	0,1 case- Add Y, Start long shift
	MDR=MDR, RROT	Complete shift. LSB has next recode bit
	OPC=OPC-1, goto M0x, if Z goto End1	Count. Go to "just handled a 0"
M00:	TOS=TOS, SRA1, goto (M01+1)	0,0 case. Start long shift
End0:	Write, goto (End1+1)	(Note: MAR still holds SP-1)
End1:	Write.	LSW stored in 2 <sup>nd</sup> stack entry.
	MAR=SP	
	MDR=TOS, Write, goto Main1	MSW stored at top-of-stack

Note there must be two entries to the End operations, since there are two different micro-instructions making the breakout test. This is a consequence of the Mic-1 scheme whereby conditional jumps must be exactly 128 from the specified next\_address.

The sequence contains 21 total micro-instructions. Of these, 10 are initialization and storing results. On any one of the 32 iterations, 4 micro-instructions are executed. Hence the CPI for this ISA Mult instruction would be  $10+32*4 = 138$  cycles. Is it worth having this ISA level instruction? Would it be better to have a software macro for multiply, in RISC type manner? With a Micro-program already present, and sufficient space in the micro-store, it costs little to include it. With a few other minor HW changes, we may even be able to use a modified Booths Algorithm to cut the CPI nearly in half. How well would the algorithm pipeline on a machine like the MIC-3?