### 12.9.1) CALL #{\}A Hardware Stack (403)

CALL #{\}A Call to A by pushing {C, Z, 10'b0, PC[19:0]} onto stack. If R = 1 then PC += A, else PC = A. "\" forces R = 0. (403)

C = Carry Flag = C31

Z = Zero Flag = Z30

10’b0 =0000000000 = B28B27B26B25B24 B23B22B21B20 (Verilog Notation)

PC = PC19PC18PC17PC16PC15PC14PC13PC12PC11PC10PC9PC8PC7PC6PC5PC4PC3PC2PC1PC0 (Call Address)

Call\_Stack = L7L6L5L4L3L2L0 (CALL\_SHIFT\_REGISTER)

L7  = B31B30B29B28B27B26B25B24 B23B22B21B20B19B18B17B16 B15B14B13B12B11B10B9B8B7B6B5B4B3B2B1B0

R = Jump Type Flag (R = 1 then relative address jump R = 0 then absolute address jump)

PC+ = Next PC value = If (R = 1 PC+ = PC + A) or If (R =0 PC+ = A)

The CALL instruction uses a an 8 level hardware stack that is stored in Cog CPU. The hardware stack has no usable stack pointer and is implemented as a big (8\*32=256-bit) shift register CALL\_SHIFT\_REGISTER. All eight longs are shifted/written by PUSH/CALL and seven longs are shifted by POP/RET. So basically the data itself moves up and down in the queue, but the "pointer" being used is fixed at the top.

PUSH should proceed POP. There should be pairs of this instruction otherwise the return address will be lost. You can only PUSH and POP register data from CALL\_SHIFT\_REGISTER along with CALL or RET instructions. PUSH and CALL write a new 32 bit data value to this stack and POP or RET remove the value.

The CALL\_SHIFT\_REGISTER is not a wrap around shift register. You can only PUSH and POP register data from it and also do CALL or RET using it. PUSH and CALL write a new 32 bit data value to this stack and POP or RET remove the value. That's all you can do with it. If you overflow or underflow it will wrap around. It is not addressable.

You can only PUSH and POP register data from it and also do CALL or RET using it. PUSH and CALL write a new 32 bit data value to this stack and POP or RET remove the value

If you push an extra value it will just drop the bottom as it pushes down. Same goes for popping. As you pop one off, they all move up the fifo, and the **bottom entry is copied up**, so when you pop them all off, the stack will contain all 8 identical values, equal to the final one popped. The hardware is a bit different in that it has a pointer to the top of stack into the 8 internal registers, but that is irrelevant to it's perceived operation.

R is the Jump Type Flag and is store in the Cog CPU. #\LABEL is an absolute jump, #LABEL is a relative jump. So R = 1 is set if you don't write a backslach “\” . IF CALL #\5 is Executed then R = 1 and PC = PC +5 the next instruction executed is at PC + 5 (relative to CALL address PC). If CALL #Label is executed R = 0 and PC = Label address (absolute address).

#### 12.9.1.1\_Example\_WRD\_CALL #\Sub\_Absolute Jump Address

Use CALL #\Subroutine (Absoloute Address)

1) Call subroutine then display message "Now in Subroutine"

2) PUSH value Num01 to stack then POP stack value POPNum01

3) Repeat for Num02-Num07

4) Display “Returned from Subroutine” and “POPNum01-POPNum07”

#### 12.9.1.2\_Example\_WRD\_CALL #\Sub\_Absolute Jump Address

Use CaLL #\Subroutine (Absolute Address)

1) Call subroutine then display message "Now in Subroutine"

2) PUSH value Num01 – Num07 to stack t

2) Then POP stack value POPNum01-Num07

3) Display “Returned from Subroutine” and “POPNum01-POPNum07”

#### 12.9.1.3\_Example\_WRD\_CALL #Sub\_Relative Jump Address

Use CALL #SUB #SUB = (JMP\_ADDRESS - CALL\_ADDRESS) = Offset (Relative Address))

1) Call subroutine then display message "Now in Subroutine"

2) PUSH value Num01 – Num07 to stack t

2) Then POP stack value POPNum01-Num07

3) Display “Returned from Subroutine” and “POPNum01-POPNum07”

### 12.9.2) CALL D Hardware Stack (318)

D is the destination address that CALL will Jump too for PC instruction. It uses the same Hardware stack as CALL #{\}A uses.

If “LABEL” is a DATA Long the address can be loaded with MOV D\_LABEL , #LABEL and the Call would be CALL D\_LABEL. (where D\_LABEL is an address value containing the next instruction for PC)

#### 12.9.2.1\_Example\_WRD\_CALL D Jump to D for PC

Example

Call Subroutine Using CALL D

1) Call subroutine then display message "Now in Subroutine"

2) PUSH value Num01 to stack then POP stack value POPNum01

3) Repeat for Num02-Num07

4) Display “Returned from Subroutine” and “POPNum01-POPNum07”