Built In Self-Test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Result | Test | PC | Reason |
| Test 1: Shift Left (8: ASL) | B4 | 01 | 0x0C | Result is not D0 |
|  | 68 |  |  |  |
|  | D0 |  |  |  |
| Test 2: Rotate Left (9: ROL) | 68 | 02 | 0x1A | Carry is not set |
|  | D0 |  | 0x1F | Result is not 0x41 |
|  | A0 |  |  |  |
|  | 41 |  |  |  |
| Test 3: Logic SR (A: LSR) | 43 | 03 | 0x29 | Carry is not set |
|  | 21 |  | 0x2D | Result is not 0x21 |
| Test 4: Arithmetic SR (B: ASR) | C3 | 04 | 0x37 | Carry is not set |
|  | E1 |  | 0x3B | Result is not 0xE1 |
| Test 5: Add (1234 + 6DCD) = 0x8001 | 01 | 05 | 0x43 | Carry is not set |
|  | 80 |  | 0x49 | Carry is set |
|  |  |  | 0x4D | Result is not 0x80 |
| Test 6: Sub (A765 – 34AB) = 0x72BA | BA | 06 | 0x55 | Carry is set |
|  | 72 |  | 0x5B | Carry is not set |
|  |  |  | 0x5F | Result is not 0x72 |
| Test 7: Rotate Right (C: ROR) | 1B | 07 | 0x70 | Carry is not set |
|  | 0D |  | 0x74 | Result is not 0x61 |
|  | 86 |  |  |  |
|  | C3 |  |  |  |
|  | 61 |  |  |  |
| Test 8: SRAM Imediate Load Store  LMA / SMA | 33 | 08 |  |  |
|  | 22 |  |  |  |
|  | 00 |  | 0x86 | Result was not 00 |
| Test 9: SRAM Register Load/Store  LMR / SMR | 5,4,3,2,1 | 09 | 0x9A | Last Read was not 5 |
|  | 24,1 |  |  |  |
|  | 23,2 |  |  |  |
|  | 22,3 |  |  |  |
|  | 21,4 |  |  |  |
|  | 20,5 |  |  |  |
| Test 10: Stack PUSH/POP | 1E9 |  | 0x1E9 |  |
| END |  | 0A | 0x9D | PASS |

# //////////////////////////////////

# // Test 1: Shift Left (8: ASL) //

# //////////////////////////////////

# 0x2d, 0x01, // 0x00: OUT #01, I // Print 01

# 0x0d, 0x20, // 0x02: OUT #0x20, A // Print ' '

# 0x0d, 0x41, // 0x04: OUT #0x41, A // Print 'A'

# 0x0d, 0x53, // 0x06: OUT #0x53, A // Print 'S'

# 0x0d, 0x4c, // 0x08: OUT #0x4C, A // Print 'L'

# 0x0d, 0x3a, // 0x0a: OUT #0x3A, A // Print ':'

# 

# 0x83, 0x5a, // 0x0c: LDR R2, #0x5A

# 0x82, 0x28, // 0x0e: ASL R2, R2

# 0x28, 0x20, // 0x10: OUT R2, I // (0xB4)

# 0x0d, 0x20, // 0x12: OUT #0x20, A // Print ' '

# 0x82, 0x28, // 0x14: ASL R2, R2

# 0x28, 0x20, // 0x16: OUT R2, I // (0x68) + Carry

# 0x0d, 0x20, // 0x18: OUT #0x20, A // Print ' '

# 0x82, 0x28, // 0x1a: ASL R2, R2 // (Unlike ROL, ASL does not take the carry in back)

# 0x28, 0x20, // 0x1c: OUT R2, I // (0xD0)

# 0xc3, 0xd0, // 0x1e: LDR R3, #0xD0

# 0xc2, 0xb7, // 0x20: SUB R3, R3, R2 (carry is set)

# 0x67, 0x01, // 0x22: BEQ +2 // RELATIVE JUMP

# 0x01, 0x00, // 0x24: BRK // Test 1 failed

# 0xc6, 0x16, // 0x26: ADD R3, #0x01 // (reset carry)

# 0x0d, 0x10, // 0x28: OUT #0x10, A // Newline

# //////////////////////////////////

# // Test 2: Rotate Left (9: ROL) //

# //////////////////////////////////

# 0x2d, 0x02, // 0x2a: OUT #02, I // Print 02

# 0x0d, 0x20, // 0x2c: OUT #0x20, A // Print ' '

# 0x0d, 0x52, // 0x2e: OUT #0x52, A // Print 'R'

# 0x0d, 0x4f, // 0x30: OUT #0x4F, A // Print 'O'

# 0x0d, 0x4c, // 0x32: OUT #0x4C, A // Print 'L'

# 0x0d, 0x3a, // 0x34: OUT #0x3A, A // Print ':'

# 0x83, 0x34, // 0x36: LDR R2, #0x34

# 0x82, 0x29, // 0x38: ROL R2, R2 // 34 -> 68

# 0x28, 0x20, // 0x3a: OUT R2, I // (0x68)

# 0x0d, 0x20, // 0x3c: OUT #0x20, A // Print ' '

# 0x82, 0x29, // 0x3e: ROL R2, R2 // 68 -> D0

# 0x28, 0x20, // 0x40: OUT R2, I // (0xD0)

# 0x0d, 0x20, // 0x42: OUT #0x20, A // Print ' '

# 0x82, 0x29, // 0x44: ROL R2, R2 // D0 -> A0 + C

# 0x28, 0x20, // 0x46: OUT R2, I // (0xA0)

# 0x0d, 0x20, // 0x48: OUT #0x20, A // Print ' '

# 0x82, 0x29, // 0x4a: ROL R2, R2 // C+A0 -> 41 + C

# 0x28, 0x20, // 0x4c: OUT R2, I // (0x41)

# 0x47, 0x01, // 0x4e: BCS +2 // RELATIVE JUMP

# 0x01, 0x00, // 0x50: BRK // Test 2 failed: Carry was not set

# 0xc3, 0x41, // 0x52: LDR R3, #0x41

# 0xc2, 0xb7, // 0x54: SUB R3, R3, R2

# 0x67, 0x01, // 0x56: BEQ +02 // RELATIVE JUMP

# 0x01, 0x00, // 0x58: BRK // Test 2 failed: Result is not 0x41

# 0x0d, 0x10, // 0x5a: OUT #0x10, A // Newline

# 

# ///////////////////////////////////////////

# // Test 3: Logic Shift Right (A: LSR) //

# ///////////////////////////////////////////

# 0x2d, 0x03, // 0x5c: OUT #03, I // Print 03

# 0x0d, 0x20, // 0x5e: OUT #0x20, A // Print ' '

# 0x0d, 0x4c, // 0x60: OUT #0x4C, A // Print 'L'

# 0x0d, 0x53, // 0x62: OUT #0x53, A // Print 'S'

# 0x0d, 0x52, // 0x64: OUT #0x52, A // Print 'R'

# 0x0d, 0x3a, // 0x66: OUT #0x3A, A // Print ':'

# 

# 0x83, 0x87, // 0x68: LDR R2, #0x87

# 0x82, 0x2a, // 0x6a: LSR R2, R2 // 87 -> 43 + Carry

# 0x28, 0x20, // 0x6c: OUT R2, I // (0x43)

# 0x0d, 0x20, // 0x6e: OUT #0x20, A // Print ' '

# 0x82, 0x2a, // 0x70: LSR R2, R2 // 43 -> 21 + Carry

# 0x28, 0x20, // 0x72: OUT R2, I // (0x21)

# 0x0d, 0x20, // 0x74: OUT #0x20, A // Print ' '

# 0x47, 0x01, // 0x76: BCS +02 // Relative

# 0x01, 0x00, // 0x78: BRK // Test 3 failed: Carry was not set

# 0xc3, 0x21, // 0x7a: LDR R3, #0x21

# 0xc2, 0xb7, // 0x7c: SUB R3, R3, R2

# 0x67, 0x01, // 0x7e: BEQ +02 // relative

# 0x01, 0x00, // 0x80: BRK // Test 3 failed: Result is not 0x21

# 0x0d, 0x10, // 0x82: OUT #0x10, A // Newline

# //////////////////////////////////////////////

# // Test 4: Arithmetic Shift Right (B: ASR) //

# //////////////////////////////////////////////

# 0x2d, 0x04, // 0x84: OUT #04, I // Print 04

# 0x0d, 0x20, // 0x86: OUT #0x20, A // Print ' '

# 0x0d, 0x41, // 0x88: OUT #0x41, A // Print 'A'

# 0x0d, 0x53, // 0x8a: OUT #0x53, A // Print 'S'

# 0x0d, 0x52, // 0x8c: OUT #0x52, A // Print 'R'

# 0x0d, 0x3a, // 0x8e: OUT #0x3A, A // Print ':'

# 0x83, 0x87, // 0x90: LDR R2, #0x87

# 0x82, 0x2b, // 0x92: ASR R2, R2 // 87 -> C3 + Carry

# 0x28, 0x20, // 0x94: OUT R2, I // (0xC3)

# 0x0d, 0x20, // 0x96: OUT #0x20, A // Print ' '

# 0x82, 0x2b, // 0x98: ASR R2, R2 // C3 -> E1 + Carry

# 0x28, 0x20, // 0x9a: OUT R2, I // (0xE1)

# 0x47, 0x01, // 0x9c: BCS +02 // Relative

# 0x01, 0x00, // 0x9e: BRK // Test 4 failed: Carry was not set

# 0xc3, 0xe1, // 0xa0: LDR R3, #0xE1

# 0xc2, 0xb7, // 0xa2: SUB R3, R3, R2

# 0x67, 0x01, // 0xa4: BEQ +02 // Relative

# 0x01, 0x00, // 0xa6: BRK // Test 4 failed: Result is not 0xE1

# 0x0d, 0x10, // 0xa8: OUT #0x10, A // Newline

# //////////////////////////////////////////////

# // Test 5: Add 0x1234 + 0x6DCD = 0x8001 //

# //////////////////////////////////////////////

# 

# 0x2d, 0x05, // 0xaa: OUT #05, I // Print 05

# 0x0d, 0x20, // 0xac: OUT #0x20, A // Print ' '

# 0x0d, 0x41, // 0xae: OUT #0x41, A // Print 'A'

# 0x0d, 0x44, // 0xb0: OUT #0x44, A // Print 'D'

# 0x0d, 0x44, // 0xb2: OUT #0x44, A // Print 'D'

# 0x0d, 0x3a, // 0xb4: OUT #0x3A, A // Print ':'

# 0x83, 0x34, // 0xb6: LDR R2, #0x34

# 0x43, 0xcd, // 0xb8: LDR R1, #0xCD

# 0xc2, 0x66, // 0xba: ADD R3, R2, R1 // 34 + CD = 0x01 + Carry

# 0x28, 0x30, // 0xbc: OUT R3, I // (0x01)

# 0x0d, 0x20, // 0xbe: OUT #0x20, A // Print ' '

# 

# 0x47, 0x01, // 0xc0: BCS +02 // Relative

# 0x01, 0x00, // 0xc2: BRK // Test 5 failed: Carry was not set

# 0x83, 0x12, // 0xc4: LDR R2, #0x12

# 0x43, 0x6d, // 0xc6: LDR R1, #0x6D

# 0xc2, 0x6e, // 0xc8: ADC R3, R2, R1 // 0x80 = 0x12 + 0x6D + Carry

# 0x28, 0x30, // 0xca: OUT R3, I // (0x80)

# 0xc7, 0x01, // 0xcc: BCC +02 // Relative

# 0x01, 0x00, // 0xce: BRK // Test 5 failed: Carry was set

# 0x83, 0x80, // 0xd0: LDR R2, #0x80

# 0xc2, 0xb7, // 0xd2: SUB R3, R3, R2

# 0x67, 0x01, // 0xd4: BEQ +02 // Relative

# 0x01, 0x00, // 0xd6: BRK // Test 5 failed: Result is not 0x80

# 0x0d, 0x10, // 0xd8: OUT #0x10, A // Newline

# //////////////////////////////////////////////

# // Test 6: Sub (0xA765 – 0x34AB) = 0x72BA //

# //////////////////////////////////////////////

# 0x2d, 0x06, // 0xda: OUT #06, I // Print 06

# 0x0d, 0x20, // 0xdc: OUT #0x20, A // Print ' '

# 0x0d, 0x53, // 0xde: OUT #0x53, A // Print 'S'

# 0x0d, 0x55, // 0xe0: OUT #0x55, A // Print 'U'

# 0x0d, 0x42, // 0xe2: OUT #0x42, A // Print 'B'

# 0x0d, 0x3a, // 0xe4: OUT #0x3A, A // Print ':'

# 0x83, 0x65, // 0xe6: LDR R2, #0x65

# 0x43, 0xab, // 0xe8: LDR R1, #0xAB

# 0xc2, 0x67, // 0xea: SUB R3, R2, R1 // 65 - AB = 0xBA + Carry

# 0x28, 0x30, // 0xec: OUT R3, I // (0xBA) (No carry, means borrow)

# 0x0d, 0x20, // 0xee: OUT #0x20, A // Print ' '

# 0xc7, 0x01, // 0xf0: BCC +02 // Relative

# 0x01, 0x00, // 0xf2: BRK // Test 6 failed: Carry was set

# 0x83, 0xa7, // 0xf4: LDR R2, #0xA7

# 0x43, 0x34, // 0xf6: LDR R1, #0x34

# 0xc2, 0x6f, // 0xf8: SBC R3, R2, R1 // R2 - R1 + Cin = A7 - 34 + Cin = 72 + Cout

# 0x28, 0x30, // 0xfa: OUT R3, I // (0x72)

# 0x47, 0x01, // 0xfc: BCS +02 // Relative

# 0x01, 0x00, // 0xfe: BRK // Test 6 failed: Carry was not set

# 0x83, 0x72, // 0x100: LDR R2, #0x72

# 0xc2, 0xb7, // 0x102: SUB R3, R3, R2

# 0x67, 0x01, // 0x104: BEQ +02 // Relative

# 0x01, 0x00, // 0x106: BRK // Test 6 failed: Result is not 0x72

# 0x0d, 0x10, // 0x108: OUT #0x10, A // Newline

# ////////////////////////////////////

# // Test 7: Rotate Right (C: ROR) //

# ////////////////////////////////////

# 0x0d, 0x07, // 0x10a: OUT #07, A // Print 7

# 0x0d, 0x20, // 0x10c: OUT #0x20, A // Print ' '

# 0x0d, 0x52, // 0x10e: OUT #0x52, A // Print 'R'

# 0x0d, 0x4f, // 0x110: OUT #0x4F, A // Print 'O'

# 0x0d, 0x52, // 0x112: OUT #0x52, A // Print 'R'

# 0x0d, 0x3a, // 0x114: OUT #0x3A, A // Print ':'

# 0x83, 0x36, // 0x116: LDR R2, #0x36

# 0x86, 0x06, // 0x118: ADD R2, #0 // Reset Carry

# 0x82, 0x2c, // 0x11a: ROR R2, R2 // 36 -> 1B

# 0x28, 0x20, // 0x11c: OUT R2, I // (0x1B)

# 0x0d, 0x20, // 0x11e: OUT #0x20, A // Print ' '

# 0x82, 0x2c, // 0x120: ROR R2, R2 // 1B -> 0D + Carry

# 0x28, 0x20, // 0x122: OUT R2, I // (0x0D)

# 0x0d, 0x20, // 0x124: OUT #0x20, A // Print ' '

# 0x82, 0x2c, // 0x126: ROR R2, R2 // 0D -> 86 + Carry

# 0x28, 0x20, // 0x128: OUT R2, I // (0x86)

# 0x0d, 0x20, // 0x12a: OUT #0x20, A // Print ' '

# 0x82, 0x2c, // 0x12c: ROR R2, R2 // 86 -> C3

# 0x28, 0x20, // 0x12e: OUT R2, I // (0xC3)

# 0x0d, 0x20, // 0x130: OUT #0x20, A // Print ' '

# 0x82, 0x2c, // 0x132: ROR R2, R2 // C3 -> 61 + Carry

# 0x28, 0x20, // 0x134: OUT R2, I // (0x61)

# 0x47, 0x01, // 0x136: BCS +02 // RELATIVE JUMP

# 0x01, 0x00, // 0x138: BRK // Test 7 failed: Carry was not set

# 0xc3, 0x61, // 0x13a: LDR R3, #0x61

# 0xc2, 0xb7, // 0x13c: SUB R3, R3, R2

# 0x67, 0x01, // 0x13e: BEQ +02 // RELATIVE JUMP

# 0x01, 0x00, // 0x140: BRK // Test 7 failed: Result is not 0x61

# 0x0d, 0x10, // 0x142: OUT #0x10, A // Newline

# /////////////////////////////////////////////////////

# // Test 8: SRAM Immediate Load/Store LMA / SMA //

# /////////////////////////////////////////////////////

# 0x2d, 0x08, // 0x144: OUT #08, I // Print 08

# 0x0d, 0x20, // 0x146: OUT #0x20, A // Print ' '

# 0x0d, 0x78, // 0x148: OUT #0x78, A // Print 'x'

# 0x0d, 0x4d, // 0x14a: OUT #0x4D, A // Print 'M'

# 0x0d, 0x41, // 0x14c: OUT #0x41, A // Print 'A'

# 0x0d, 0x3a, // 0x14e: OUT #0x3A, A // Print ':'

# 0x43, 0x11, // 0x150: LDR R1, #0x11

# 0x4a, 0x10, // 0x152: SMA R1, #0x10 // Store 11 in address 10

# 0x83, 0x22, // 0x154: LDR R2, #0x22

# 0x8a, 0x20, // 0x156: SMA R2, #0x20 // Store 22 in address 20

# 0xc3, 0x33, // 0x158: LDR R3, #0x33

# 0xca, 0x30, // 0x15a: SMA R3, #0x30 // Store 33 in address 30

# 

# 0xc9, 0x20, // 0x15c: LMA R3, #0x20 // Load value at address 20 into R3 (R3 = 22)

# 0x89, 0x10, // 0x15e: LMA R2, #0x10 // Load value at address 10 into R2 (R2 = 11)

# 0x49, 0x30, // 0x160: LMA R1, #0x30 // Load value at address 30 into R1 (R1 = 33)

# 0x28, 0x10, // 0x162: OUT R1, I // (0x33)

# 0x0d, 0x20, // 0x164: OUT #0x20, A // Print ' '

# 0x42, 0x97, // 0x166: SUB R1, R1, R2

# 0x28, 0x10, // 0x168: OUT R1, I // (0x22)

# 0x0d, 0x20, // 0x16a: OUT #0x20, A // Print ' '

# 0x42, 0xd7, // 0x16c: SUB R1, R1, R3

# 0x28, 0x10, // 0x16e: OUT R1, I // (0x00)

# 0x67, 0x01, // 0x170: BEQ +02 // RELATIVE JUMP

# 0x01, 0x00, // 0x172: BRK // Test 8 failed: Result is not 0x00

# 0x0d, 0x10, // 0x174: OUT #0x10, A // Newline

# /////////////////////////////////////////////////////

# // Test 9: SRAM Register Load/Store LMR / SMR //

# /////////////////////////////////////////////////////

# 0x2d, 0x09, // 0x176: OUT #09, I // Print 09

# 0x0d, 0x20, // 0x178: OUT #0x20, A // Print ' '

# 0x0d, 0x78, // 0x17a: OUT #0x78, A // Print 'x'

# 0x0d, 0x4d, // 0x17c: OUT #0x4D, A // Print 'M'

# 0x0d, 0x52, // 0x17e: OUT #0x52, A // Print 'R'

# 0x0d, 0x3a, // 0x180: OUT #0x3A, A // Print ':'

# 0x43, 0x05, // 0x182: LDR R1, #0x05

# 0x83, 0x20, // 0x184: LDR R2, #0x20

# //:store

# 0x4c, 0x20, // 0x186: SMR R1, R2 // Store 5 in address 20

# 0x08, 0x10, // 0x188: OUT R1, A // (20:0x05 , 21:0x04, 22:0x03, 23:0x02, 24:0x01)

# 0x0d, 0x20, // 0x18a: OUT #0x20, A // Print ' '

# 0x86, 0x16, // 0x18c: INC R2

# 0x46, 0x17, // 0x18e: DEC R1

# 0xe7, 0xfa, // 0x190: BNE -0x05 // back to :Store

# 0x0d, 0x10, // 0x192: OUT #0x10, A // C9: Newline

# 0x43, 0x05, // 0x194: LDR R1, #0x05

# 0x83, 0x24, // 0x196: LDR R2, #0x24

# //:load

# 0x28, 0x20, // 0x198: OUT R2, I // (0x24, 0x23, 0x22, 0x21, 0x20)

# 0x0d, 0x3a, // 0x19a: OUT #0x3A, A // Print ':'

# 0xcb, 0x20, // 0x19c: LMR R3, R2 // Load address[R2] to R3

# 0x08, 0x30, // 0x19e: OUT R3, A // (1, 2, 3, 4, 5)

# 0x0d, 0x20, // 0x1a0: OUT #0x20, A // Print ' '

# 0x86, 0x17, // 0x1a2: DEC R2

# 0x46, 0x17, // 0x1a4: DEC R1

# 0xe7, 0xf8, // 0x1a6: BNE -0x07 // back to Load

# 0xc6, 0x57, // 0x1a8: SUB R3, #05 // R3 = R3 - #5

# 0x67, 0x01, // 0x1aa: BEQ +02 // RELATIVE JUMP

# 0x01, 0x00, // 0x1ac: BRK // Test 9 failed: R3 != #5

# 0x0d, 0x10, // 0x1ae: OUT #0x10, A // Newline

# 

# /////////////////////////////////////////////////////

# // Test 10: PUSH/POP //

# /////////////////////////////////////////////////////

# 0x2d, 0x0a, // 0x1b0: OUT #10, I // Print 010

# 0x0d, 0x20, // 0x1b2: OUT #0x20, A // Print ' '

# 0x0d, 0x53, // 0x1b4: OUT #0x53, A // Print 'S'

# 0x0d, 0x54, // 0x1b6: OUT #0x54, A // Print 'T'

# 0x0d, 0x43, // 0x1b8: OUT #0x43, A // Print 'C'

# 0x0d, 0x3a, // 0x1ba: OUT #0x3A, A // Print ':'

# 0x43, 0xff, // 0x1bc: LDR R1, #0xFF // Stack max capacity

# 0x0e, 0x10, // 0x1be: PUSH R1

# 0x46, 0x17, // 0x1c0: DEC R1

# 0xe7, 0xfd, // 0x1c2: BNE -2

# 0x46, 0x16, // 0x1c4: INC R1

# 0x8f, 0x00, // 0x1c6: POP R2

# 

# 0x0d, 0x10, // 0x1c8: OUT #0x10, A // Newline

# 0x0d, 0x52, // 0x1ca: OUT #0x52, A // Print 'R'

# 0x0d, 0x01, // 0x1cc: OUT #01, A // Print '1'

# 0x0d, 0x3a, // 0x1ce: OUT #0x3A, A // Print ':'

# 0x0d, 0x20, // 0x1d0: OUT #0x20, A // Print ' '

# 0x28, 0x10, // 0x1d2: OUT R1, I

# 0x0d, 0x20, // 0x1d4: OUT #0x20, A // Print ' '

# 0x0d, 0x20, // 0x1d6: OUT #0x20, A // Print ' '

# 0x0d, 0x20, // 0x1d8: OUT #0x20, A // Print ' '

# 0x0d, 0x52, // 0x1da: OUT #0x52, A // Print 'R'

# 0x0d, 0x02, // 0x1dc: OUT #02, A // Print '2'

# 0x0d, 0x3a, // 0x1de: OUT #0x3A, A // Print ':'

# 0x0d, 0x20, // 0x1e0: OUT #0x20, A // Print ' '

# 0x28, 0x20, // 0x1e2: OUT R2, I

# 

# 0x82, 0x67, // 0x1e4: SUB R2, R2, R1

# 0x67, 0x01, // 0x1e6: BEQ +2

# 0x01, 0x00, // 0x1e8: BRK // Fail R1 != R2

# 0x46, 0x16, // 0x1ea: INC R1

# 0xe7, 0xec, // 0x1ec: BNE -19

# // End of BIST

# 0x0d, 0x50, // 0x1ee: OUT #0x50, A // Print 'P'

# 0x0d, 0x41, // 0x1f0: OUT #0x41, A // Print 'A'

# 0x0d, 0x53, // 0x1f2: OUT #0x53, A // Print 'S'

# 0x0d, 0x53, // 0x1f4: OUT #0x53, A // Print 'S'

# 0x01, 0x00, // 0x1f6: BRK // (end of BIST)

# 

# // ROBERTSON'S MULTIPLICATION SIGNED version 3 (with rotate right)

byte robertsonv3[] = {

0x03, 0x00, // 0 LDR R0, #0

0x43, 0x80, // 1 LDR R1, #80 (-128)

0x83, 0x01, // 2 LDR R2, #1 (1) --> Expect 0x0080 //--> Expect 0xF71D (-2275)

0xc3, 0x07, // 3 LDR R3, #7 (0x07)

0x46, 0x0a, // 4 LSR R1 (Shift right using immediate opcode R1[0] -> Carry)

0xc7, 0x01, // 5 BNC + 1 (relative jump over 1 instruction)

0x02, 0x26, // 6 ADD R0, R0, R2

0x46, 0x08, // 7 LSL R1 (resume R1 position)

0x06, 0x0b, // 8 ASR R0 (Shift Right: R0 -> Carry -> R1)

0x46, 0x0c, // 9 ROR R1

0x28, 0x00, // A OUT R0 H

0x18, 0x10, // B OUT R1 L

0xc6, 0x17, // C SUB R3, #1

0xe7, 0xf6, // D BNZ 4 (relative jump back 9 instructions by substracting 10 from PC)

0x46, 0x0a, // E LSR R1 (Shift right using immediate opcode R1[0] -> Carry)

0xc7, 0x01, // F BNC + 1 (relative jump over 1 instruction)

0x02, 0x87, // 10 SUB R0, R0, R2 R0 = R0 - R2

0x46, 0x08, // 11 LSL R1 (resume R1 position)

0x06, 0x0b, // 12 ASR R0 (Shift Right: R0 -> Carry -> R1)

0x46, 0x0c, // 13 ROR R1

0x28, 0x00, // 14 OUT R0 H

0x18, 0x10, // 15 OUT R1 L

0x01, 0x00 // 16 BRK

};

# // Hierarchical Loops

byte TwoLoops[] = {

0x03, 0x01, // 0: LDR R0, 0x01

0x43, 0x00, // 1: LDR R1, 0x00

0x83, 0x00, // 2: LDR R2, 0x00

0xC3, 0x00, // 3: LDR R3, 0x00

0xC2, 0xC6, // 4: ADD R3, R3, R0

0x18, 0x30, // 5: OUT Low, R3

0xE5, 0x04, // 6: Branch Not Zero, 4

0x82, 0x86, // 7: ADD R2, R2, R0

0x28, 0x20, // 8: OUT HIGH R2

0xE5, 0x04, // 9: Branch Not Zero, 4

0x01, 0x00 // 10: Break

};