**1.2.2 Validation of Operation via Microprogrammed Control**

//1. Write a subroutine that get an 8-bit number and computes it 2’s complement.

/\*/\*//\* 1. subroutine \*//\*/\*/

//initially r0 is assigned 1111 1111

//r3 's content is 1

//then r1 contains the our number which is 8-bit number which will be computed it 2’s complement

//lets say r1=00110011 we will compute its 2's complement

xor r2,r1,r0 //take the complement of the number in r1

add r2,r3,r2 //r2 contains the twos complement of the number which is stored in r1

/\*Machine Code of subroutine 1\*/

//instructions for the subroutine 1

DATAmem[0] = 16'b0011\_0010\_0010\_0000;//xor r2=r1^r0 d=2, n=1, m=0

DATAmem[4] = 16'b0000\_0010\_0110\_1000;//add rd 2 rn 3 rm 2

DATAmem[8] = 16'b1111\_1111\_1111\_1111; //end instruction

//2. Write a subroutine that computes the sum of an array of numbers and stores it in a memory location. The length of the array is constant and can be taken as 5.

/\*/\*//\* 2. subroutine \*//\*/\*/

//to compute summation

//initially r1's content is 0

//r3 keeps the array[0] memory location r3=112

//take number from address and add it

//r4 is equal to 4 initially

ldr r2,[r3] //ldr loads last 8 bits of addressed mem. Data to reg

add r1,r1,r2 //add the stored number

add r3,r3,r4 //increase the memory pointer location to reach next array location

ldr r2,[r3]

add r1,r1,r2

add r3,r3,r4

ldr r2,[r3]

add r1,r1,r2

add r3,r3,r4

ldr r2,[r3]

add r1,r1,r2

add r3,r3,r4

ldr r2,[r3]

add r1,r1,r2

/\*Machine Code of subroutine 2\*/

//instructions for the subroutine 2

DATAmem[0] = 16'b1000\_0010\_0110\_0000; //ldr r2,[r3]

DATAmem[4] = 16'b0000\_0001\_0010\_1000; //add rd 1 rn 1 rm 2 then result is " 3" 3=0+3

DATAmem[8] = 16'b0000\_0011\_0111\_0000; //add rd 3 rn 4 rm 3 then result r3+4

DATAmem[12] = 16'b1000\_0010\_0110\_0100; //ldr r2,[r3]

DATAmem[16] = 16'b0000\_0001\_0010\_1000; //add rd 1 rn 1 rm 2 then result is " 8" 8=5+3

DATAmem[20] = 16'b0000\_0011\_0111\_0000; //add rd 3 rn 4 rm 3 then result r3+4

DATAmem[24] = 16'b1000\_0010\_0110\_1000; //ldr r2,[r3]

DATAmem[28] = 16'b0000\_0001\_0010\_1000; //add rd 1 rn 1 rm 2 then result is " 15" 15=7+8

DATAmem[32] = 16'b0000\_0011\_0111\_0000; //add rd 3 rn 4 rm 3 then result r3+4

DATAmem[36] = 16'b1000\_0010\_0110\_1100; //ldr r2,[r3]

DATAmem[40] = 16'b0000\_0001\_0010\_1000; //add rd 1 rn 1 rm 2 then result is " 24" 24=9+15

DATAmem[44] = 16'b0000\_0011\_0111\_0000; //add rd 3 rn 4 rm 3 then result r3+4

DATAmem[48] = 16'b1000\_0010\_0111\_0000; //ldr r2,[r3]

DATAmem[52] = 16'b0000\_0001\_0010\_1000; //add rd 1 rn 1 rm 2 then result is " 35" 35=24+11

DATAmem[56] = 16'b1111\_1111\_1111\_1111;

//memory

DATAmem[112] = 16'b0000\_0000\_0000\_0011;//3

DATAmem[116] = 16'b0000\_0000\_0000\_0101;//5

DATAmem[120] = 16'b0000\_0000\_0000\_0111;//7

DATAmem[124] = 16'b0000\_0000\_0000\_1001;//9

DATAmem[128] = 16'b0000\_0000\_0000\_1011;//11

//3. Write a subroutine that determines the evenness/oddity of a 8-bit number, n7n6n5n4n3n2n1n0. If odd, the expected output is n4n3n2n10n7n6n5, otherwise, 0n4n3n2n1000.

/\*/\*//\* 3. subroutine \*//\*/\*/

//then r1 contains the our number which is 8-bit number which will be evaluated

//r0's content initialized as 0000 0001

and r2,r1,r0 //if r2 is zero then even

beq even\_case

//odd case manipulations

// n7 n6 n5 n4 n3 n2 n1 n0-->n4 n3 n2 n1 0 n7 n6 n5

// 0 n7 n6 n5 n4 n3 n2 n1 get rid of n0-->lsr

lsr r1,r1//then rotate left

rol r1,r1 // n7 n6 n5 n4 n3 n2 n1 0

rol r1,r1 // n6 n5 n4 n3 n2 n1 0 n7

rol r1,r1 // n5 n4 n3 n2 n1 0 n7 n6

rol r1,r1 // n4 n3 n2 n1 0 n7 n6 n5

b jump

even\_case

//even case manipulations

// n7 n6 n5 n4 n3 n2 n1 n0-->0 n4 n3 n2 n1 0 0 0

lsr r1,r1 // 0 n7 n6 n5 n4 n3 n2 n1

lsl r1,r1 // n7 n6 n5 n4 n3 n2 n1 0

lsl r1,r1 // n6 n5 n4 n3 n2 n1 0 0

lsl r1,r1 // n5 n4 n3 n2 n1 0 0 0

lsl r1,r1 // n4 n3 n2 n1 0 0 0 0

lsr r1,r1 // 0 n4 n3 n2 n1 0 0 0

jump

END

/\*Machine Code of subroutine 3\*/

//instructions for the subroutine 3

DATAmem[0] = 16'b0010\_0010\_0010\_0000;//and r2=r1&r0

DATAmem[4] = 16'b1101\_1000\_0001\_1100; //beq even\_case then branch 40

//beq will be added

DATAmem[8] = 16'b0110\_0001\_0010\_0000;//lsr r1= lsr r1

DATAmem[12] = 16'b0100\_0001\_0010\_0000;//rol r1=rol r1

DATAmem[16] = 16'b0100\_0001\_0010\_0000;//rol r1=rol r1

DATAmem[20] = 16'b0100\_0001\_0010\_0000;//rol r1=rol r1

DATAmem[24] = 16'b0100\_0001\_0010\_0000;//rol r1=rol r1

DATAmem[28] = 16'b1100\_0000\_0001\_1100; //b jump pc+8+imm8(28)=64

//even\_case

DATAmem[40] = 16'b0110\_0001\_0010\_0000;//lsr r1= lsr r1

DATAmem[44] = 16'b0101\_0001\_0010\_0000;//lsl r1= lsl r1

DATAmem[48] = 16'b0101\_0001\_0010\_0000;//lsl r1= lsl r1

DATAmem[52] = 16'b0101\_0001\_0010\_0000;//lsl r1= lsl r1

DATAmem[56] = 16'b0101\_0001\_0010\_0000;//lsl r1= lsl r1

DATAmem[60] = 16'b0110\_0001\_0010\_0000;//lsr r1= lsr r1

//jump

DATAmem[64] = 16'b1111\_1111\_1111\_1111;