1. Verilog instruction to know : 8

mv, mvt, add, sub, ld, st, and, bf cond

Instruction	To	T_1	T_2	T_3	T_4	T_5
mv	Sel=PC, ADDRin PCinc	wait	IRin	Sel=rY or IR, rXin, Done		
mvt	Sel=PC, ADDRin PCinc	wait	IRin	Sel=IR, rXin, Done		
add	Sel=PC, ADDRin PCinc	wait	IRin	Sel=rX, Ain	Sel=rY or IR, Gin	Sel=G, rXin, Done
sub	Sel=PC, ADDRin PCinc	wait	IRin	Sel=rX, Ain	Sel=rY or IR, AddSub, Gin	Sel=G, rXin, Done
ld	Sel=PC, ADDRin PCinc	wait	IRin	Sel=rY, ADDRin	wait Sel=rX,	Sel=DIN rXin, Done
st	Sel=PC, ADDRin PCinc	wait	IRin	Sel=rY, ADDRin	DOUTin, W	Done

Instruction Fetch

2. ARM processor & Assembly Programming

- Discussed word & byte addressing in ARM

  word: LDR, STR byte: LDRB, STRB

- Addressing modes [Rn], [Rn, #offset] (more on aid sheet)

- Arithmetic, logic, shifting

- Conditional branches

  - without "S", the flags are not affected → e.g: ADD
  - with "S", the result affect condition code flags (E,V,N,C) → e.g: ADDS

- Using immediate constants in instruction

```
MOV  R0,#0      R0 ← 0
```

- Another way to get 32 bit constants is a trick done by assembler, for you:

```
LDR  R1,=0xFF200020
```

Assembler turns this into 2 things: (at address .word 0xFF200020)

```
LDR  R1,[PC,#offset]
MOV  R1,#0xFFFFFFFF
```

3. Write a program that calls a subroutine to compute for N, the $\sum_{i=1}^{N} i$

in C:

```
int FINDSUM(int N){
  int Sum = 0;
  while (N!=0){
      Sum = Sum+N;
      N = N-1;
  }
  return (Sum);
}
```

in assembly:

```
          .global _start
_start:   MOV  R4,#N
          LDR  R0,[R4]      // R0 ← N
          BL   FINDSUM      // LR ← PC   PC ← FINDSUM
          STR  R0,[R4,#4]   // Put answer into SUM
END:      B    END
N:        .word 10
SUM:      .word 0
FINDSUM:  MOV  R1,#0        // R1 ← 0 (SUM)
WHILE:    CMP  R0,#0
          BEQ  ENDLOOP
          ADD  R1,R0        // SUM = SUM+N  (R1 = SUM)
          SUB  R0,#1        // N = N-1   (R0 = N here)
          B    WHILE
ENDLOOP:  MOV  R0,R1        // Put answer (sum = R1) into R0
          MOV  PC,LR        // PC ← LR  return from subroutine
```

same but in Enhanced/Lab2 processor:

```
_start:   mv   w,#N
          ld   r0,[r4]
          mv   r5,r1
          b    &FINDSUM
          add  r4,#1
          st   r0,[r4]
End:      b    END
N:        .word 10
SUM:      .word 0
FINDSUM:  mv   r1,#0
WHILE:    add  r0,#0    (like CMP)
          beq  &ENDLOOP  (C while loop)
          add  r1,r0     (inner while loop)
          sub  r0,#1
          b    &WHILE
ENDLOOP:  mv   r0,r1
          add  r5,#1     (add 1 to link register)
          mv   r1,r5     (PC ← LR return)
```