(e) [10 points] Fully describe the FSM with equations given that the states are encoded with **binary** encoding. Assign state encodings such that numerical values of states increase monotonically for states A through D while using the **minimum** possible number of bits to represent the states with binary encoding. Indicate the values you assign to each state *and* simplify all equations:

State assignments: A: 00, B: 01, C: 10, D: 11

NS[1] = $\overline{TS[1]}$ * ($\overline{TS[0]}$ * TB + TS[0] $\overline{TA}$) + TS[1] * ($\overline{TS[0]}$ + TS[0] * TB)

NS[0] = $\overline{TS[1]}$ * $\overline{TS[0]}$ * $\overline{TB}$ + TS[1]

O[1] = TS[1]

O[0] = TS[1] XOR TS[0]

(f) [10 points] Fully describe the FSM with equations given that the states are encoded with **output** encoding. Use the **minimum** possible number of bits to represent the states with output encoding. Indicate the values you assign to each state *and* simplify all equations:

State assignments: A: 10, B: 11, C: 01, D: 00

NS[1] = TS[1] * $\overline{TS[0]}$ * $\overline{TB}$ + TS[1] * TS[0] * TA + $\overline{TS[1]}$ * $\overline{TS[0]}$ * $\overline{TB}$

= $\overline{TS[0]}$ * $\overline{TB}$ + TS[1] * TS[0] * TA

NS[0] = TS[1] * $\overline{TS[0]}$ + TS[1] * TS[0] * $\overline{TA}$ + $\overline{TS[1]}$ * $\overline{TS[0]}$ * $\overline{TB}$

= TS[1] * ($\overline{TS[0]}$ + TS[0] * $\overline{TA}$) + $\overline{TS[1]}$ * $\overline{TS[0]}$ * $\overline{TB}$

O[1] = TS[1]

O[0] = TS[0]

(g) [10 points] Assume the following conditions:

- We can only implement our FSM with 2-input AND gates, 2-input OR gates, and D flip-flops.
- 2-input AND gates and 2-input OR gates occupy the *same* area.
- D flip-flops occupy 3x the area of 2-input AND gates.

Which state encoding do you choose to implement in order to **minimize the total area** of this FSM?

one-hot: 10 logics 4 FFs binary: 16 logics. 2 FFs output: 10 logics. 2 FFs
Output encoding has the least amount of circuitry elements.

5 ISA and Microarchitecture [45 points]

You are asked to complete the following program written in MIPS assembly with a sequence of MIPS instructions that perform **64-bit integer subtraction (A - B)**. The 64-bit integer to be subtracted *from (A)* is loaded into registers \$4 and \$5. Similarly, the 64-bit integer to subtract (B) is loaded into registers \$6 and \$7. Both numbers are in two's complement form. The upper 32-bit part of each number is stored in the corresponding even-numbered register.

```
Loop:  lw  $4, 0($1)
       lw  $5, 4($1)
       lw  $6, 8($1)
       lw  $7, 12($1)

       # 64-bit subtraction
       # goes here

       addi  $1, $1, 16
       j  Loop
```

(a) [15 points] Complete the above program to perform the 64-bit subtraction explained above **using at most 4 MIPS instructions**. (*Note: A summary of the MIPS ISA is provided at the end of this question.*)

A possible sequence of instructions is as follows:

```
subu $3, $5, $7 # Subtract the least significant part
sltu $2, $5, $7 # Check if borrowing is needed
add $2, $6, $2 # Add borrow
sub $2, $4, $2 # Subtract the most significant part
```

(b) [15 points] Assume that the program executes on a pipelined processor, which does *not* implement interlocking in hardware. The pipeline assumes that all instructions are independent and relies on the compiler to properly order instructions such that there is sufficient distance between dependent instructions. The compiler either moves other independent instructions between two dependent instructions, if it can find such instructions, or otherwise, inserts `nop`s. There is *no* internal register file forwarding (i.e., if an instruction writes into a register, another instruction *cannot* access the new value of the register until the next cycle). The pipeline does *not* implement any data forwarding. The datapath has the following *five pipeline stages*, similarly to the basic pipelined MIPS processor we discussed in lecture. Registers are accessed in the Decode stage. The execution stage contains *one ALU*.

(a) Fetch (one clock cycle)
(b) Decode (one clock cycle)
(c) Execute (one clock cycle)
(d) Memory (one clock cycle)
(e) Write-back (one clock cycle).

Reorder the existing instructions and insert as few as possible `nop` instructions to correctly execute the entire program that you completed in part (a) on the given pipelined processor. Show all the instructions necessary to correctly execute the **entire program**.

We reoder the `lw` instructions to first load the data that corresponds to the lower parts of the two numbers since we need the lower part first. We also reorder the completely independent `addi` instruction to hide part of the load latency. We insert sufficient number of `nop` instructions until the register is written before the dependent instruction reads the same register in the decode stage.

This is the resulting code:

```
Loop:  lw $5, 4($1)
       lw $7, 12($1)
       lw $4, 0($1)
       lw $6, 8($1)
       addi $1, $1, 16
       sltu $2, $5, $7
       subu $3, $5, $7
       nop
       nop
       add $2, $6, $2
       nop
       nop
       nop
       sub $2, $4, $2
       j Loop
```

(c) [5 points] What is the *Cycles Per Instruction (CPI)* of the program when executed on the pipelined processor provided in part (b)?

$CPI \approx 1.5$

Explanation.
Since the code is an infinite loop, the number of cycles to fill the pipeline becomes negligible after a large number of iterations. Thus, we can consider that the throughput is one instruction every cycle. We count the number of cycles for one loop iteration. It is 15 for 10 instructions. This way, $CPI \approx \frac{15}{10} = 1.5$.

(d) [10 points] Now, assume a processor with a *multi-cycle* datapath. In this multi-cycle datapath, each instruction type is executed in the following number of cycles: 4 cycles for R-type, 5 cycles for load, 4 cycles for store, and 3 cycles for jump. What is the CPI of the program in part (a) when executed on this multi-cycle datapath? Assuming the multi-cycle datapath runs at the same clock frequency as the pipelined datapath in part (b), how much speedup does pipelining provide?

CPI:

$CPI = 4.3$

Explanation.
For the multi-cycle datapath, we have to take into account the number of cycles for each instruction type: 4 cycles for R-type, 5 cycles for load, 4 cycles for store, and 3 cycles for jump.
Thus, $CPI = \frac{4\times5+5\times4+3\times1}{10} = 4.3$.

Speedup:

Pipelining provides 287% speedup.

Explanation.
We calculate the speedup as follows:
$Speedup = \frac{CPI_{multi-cycle}}{CPI_{pipelined}} = \frac{4.3}{1.5} = 2.87$.