ECE 120: Introduction to Computing

- Ensure that your exam booklet has 15 pages.
- **Please, write your NAME, NetID, and UIN clearly.**
- Do not tear the exam booklet apart. You can only detach the last two pages, (ASCII Table and scratch paper) if needed.
- This is a closed book/notes exam. You may use a calculator.
- You are allowed **one handwritten sheet** of notes (both sides). Write your name on the cheat sheet. The cheat sheet will be collected at the end of your exam.
- Absolutely no interaction between students is allowed.
- Clearly indicate any assumptions that you make.
- **The questions are not weighted equally**. Budget your time accordingly.
- Show your work and write legibly. Solutions in i**llegible handwriting will be graded as incorrect.**
- Write your **UIN (9-digit #) on each page in the provided space.**

NAME **Answer Key**

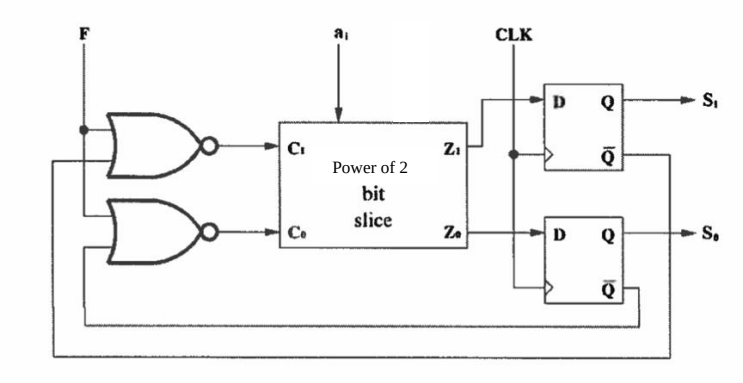
NetID

UIN

Problem 1	20 points	_____
Problem 2	20 points	_____
Problem 3	25 points	_____
Problem 4	14 points	_____
Problem 5	21 points	_____
Total	100 points	_____

Problem 1 (16+4 = 20 points): Serialization

Shown below is a serialized implementation of a power-of-2 checker that checks whether an unsigned integer $A=a_{n-1}a_{n-2}..a_1a_0$ is a power of 2. The bits are fed serially from LSB to MSB. Also, available is the signal F that is 1 when a_0 is being processed and 0 otherwise. One clock cycle after a_i is input, the output S_1S_0 has the meaning shown in the table below.



S_1S_0	Meaning
0 0	Have not seen any 1s so far
0 1	Seen only one '1' so far
1 0	Seen more than one '1' so far
1 1	Unused

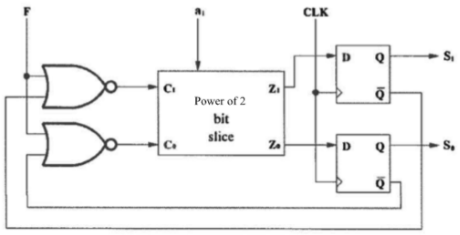
Interpret this circuit as a finite-state machine and complete the following state-transition table.

S_1	S_0	F	a_i	S_1+	S_0+
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Write down the S_1+ and S_0+ expressions below:

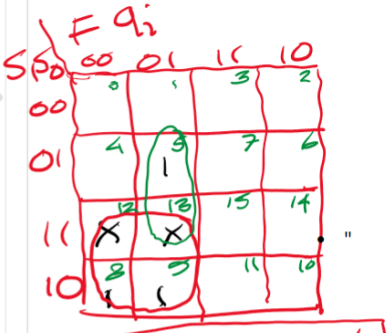
Problem 1 (16+4 = 20 points): Serialization

Shown below is a serialized implementation of a power-of-2 checker that checks whether an unsigned integer $A=a_{n-1}a_{n-2}..a_1a_0$ is a power of 2. The bits are fed serially from LSB to MSB. Also, available is the signal F that is 1 when a_0 is being processed and 0 otherwise. One clock cycle after a_i is input, the output S_1S_0 has the meaning shown in the table below.



S_1S_0	Meaning
0 0	Have not seen any 1s so far
0 1	Seen only one '1' so far
1 0	Seen more than one '1' so far
1 1	Unused

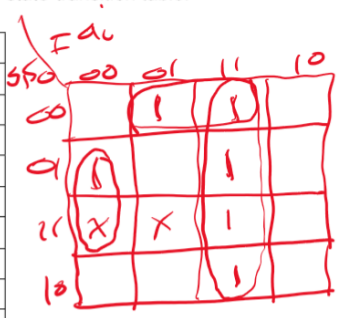
Interpret this circuit as a finite-state machine and complete the following state-transition table.



$S_1^+ = S_1F' + S_0Fq_i'$

SOP

S_1	S_0	F	a_i	S_1+	S_0+
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	x	x
1	1	0	1	x	x
1	1	1	0	0	0
1	1	1	1	0	1



$S_0^+ = Fq_i + \bar{S}_1\bar{S}_0q_i + S_0\bar{F}\bar{q}_i$

SOP

Write down the S_1+ and S_0+ expressions below:

POS expression should also be considered as correct.

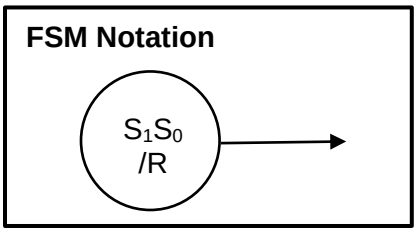
Problem 2 (20 points): Sequence Recognizer

In this question, you are going to work with a sequence different from the familiar 0-1 sequence. Your goal is to recognize the pattern "**ece**" from a random input sequence made up of "**e**","**c**" and "**b**" (Note: Only these three letters will show up).

An example for the FSM is shown below.

Input: ***x*** = e c e c e e c e b e . .

Output: ***R*** = 0 0 1 0 1 0 0 1 0 0 . .



Note: Overlapping instances are counted as shown in the example. The output sequence is delayed by 1 clock cycle compared to the input sequence because the output **R** is a function of the flip-flop outputs (i.e. the state variables) in a Moore machine. That's why the output sequence becomes 1 in the cycle after "ece" has been recognized by the input.

1. **(3 points)** The first thing you need to do is to define the states for your FSM. We will assume that the FSM starts in state S_1S_0=00, and each state has the output R indicating if the pattern **"**ece**"** has been found (1 for found and 0 for not found). We will also use "x" (representing letters) to label the transitions between the states.

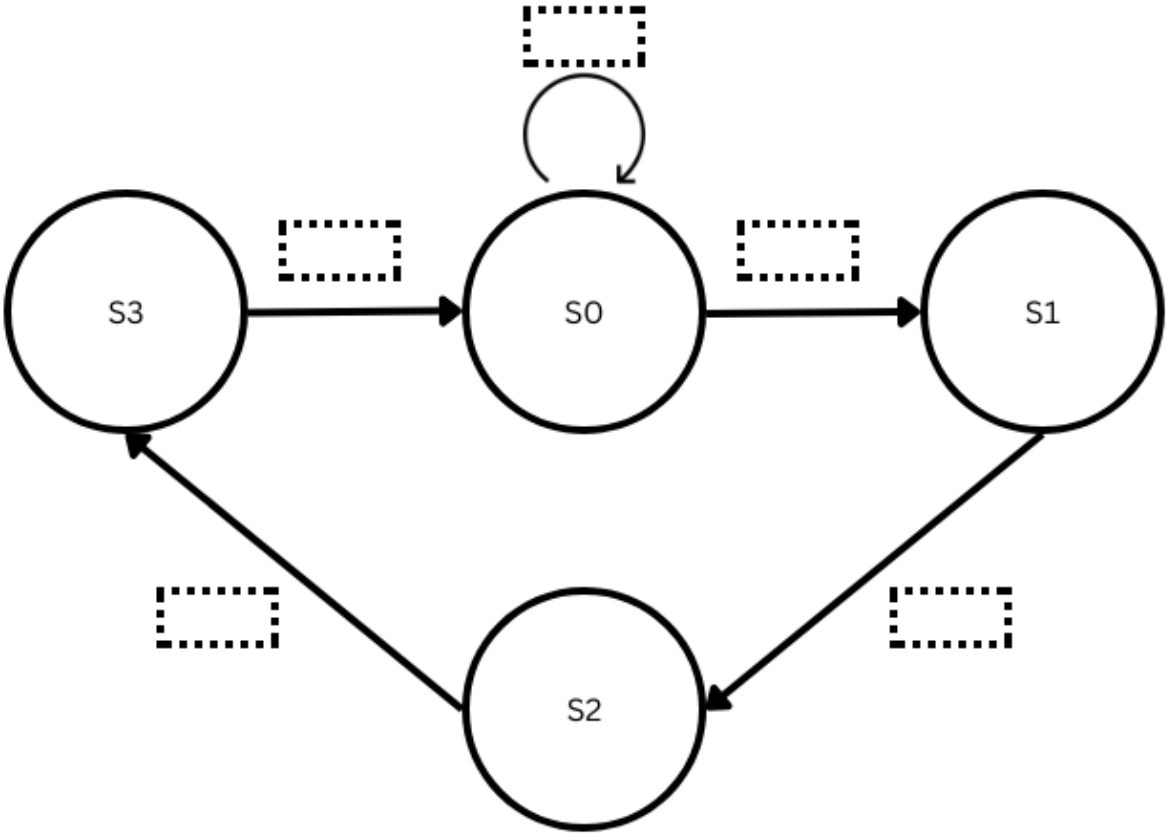
States (S_1S_0)	**Output (R)**	**State Meanings**
00	**0**	**Starting state**
01		
10		
11		

2. **(12 points)** Next, fill the state transition table.
(Note: S_1 and S_0 represent the current state and S_1^+ and S_0^+ represent the next state)

S_1^+	S_0^+	b	c	e	S_1	S_0
		1	**0**	**0**	**0**	**0**
		0	**1**	**0**	**0**	**0**
		0	**0**	**1**	**0**	**0**
		1	**0**	**0**	**0**	**1**
		0	**1**	**0**	**0**	**1**
		0	**0**	**1**	**0**	**1**
		1	**0**	**0**	**1**	**0**
		0	**1**	**0**	**1**	**0**
		0	**0**	**1**	**1**	**0**
		1	**0**	**0**	**1**	**1**
		0	**1**	**0**	**1**	**1**
		0	**0**	**1**	**1**	**1**

3. **(5 points)** Finally, it is time to draw your FSM design! While we have prepared a version for you, **it only shows some of the transitions and the labels for those transitions are missing.** Therefore, your task is to

   (1) Write the missing labels of the given transitions and, (2) complete the missing transitions. Also, for different values of "x" for the same transition, you should write them together on a single arrow like: $\xrightarrow{x_1, x_2}$, where x1 and x2 are different letters.



1. **(3 points)** The first thing you need to do is to define the states for your FSM. We will assume that the FSM starts in state S_1S_0=00, and each state has the output R indicating if the pattern "ece" has been found (1 for found and 0 for not found). We will also use "x" (representing letters) to label the transitions between the states.

States (S_1S_0)	Output (R)	State Meanings
00	0	Starting state
01	0	"seen e"
10	0	"seen ec"
11	1	"seen ece"

Other representation is also possible.

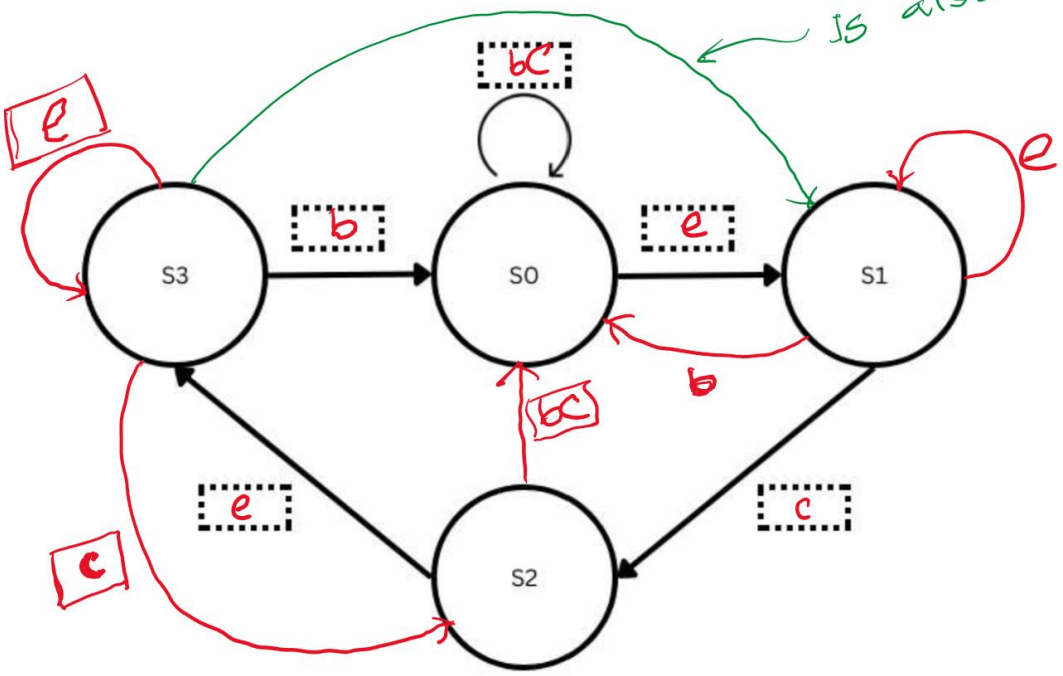
2. **(12 points)** Next, fill the state transition table.
(Note: S_1 and S_0 represent the current state and S_1^+ and S_0^+ represent the next state)

2. **(12 points)** Next, fill the state transition table.
(Note: S_1 and S_0 represent the current state and S_1^+ and S_0^+ represent the next state)

S_1^+	S_0^+	b	c	e	S_1	S_0
0	0	1	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	1
1	0	0	1	0	0	1
0	1	0	0	1	0	1
0	0	1	0	0	1	0
0	0	0	1	0	1	0
1	1	0	0	1	1	0
0	0	1	0	0	1	1
1	0	0	1	0	1	1
0	1	0	0	1	1	1

3. **(5 points)** Finally, it is time to draw your FSM design! While we have prepared a version for you, **it only shows some of the transitions and the labels for those transitions are missing.** Therefore, your task is to

   (1) Write the missing labels of the given transitions and, (2) complete the missing transitions. Also, for different values of "x" for the same transition, you should write them together on a single arrow like: $\xrightarrow{x_1,\ x_2}$, where x1 and x2 are different letters.



Problem 3 (25 points) FSM Design

Problem statement: Given your knowledge of ECE120, you have been hired as a hardware engineer in a very lucrative company. Your first assignment is to design an FSM to implement the following C code fragment that calculates the sum of a set of 10 integers (stored in an array defined as values[]).

```
int values[10];
int idx ;
int sum = 0;
for (idx = 0 ; 10 > idx; idx = idx + 1)
{
   if (values[idx] > 0) {
      sum = sum + values[idx];
   }
}
```

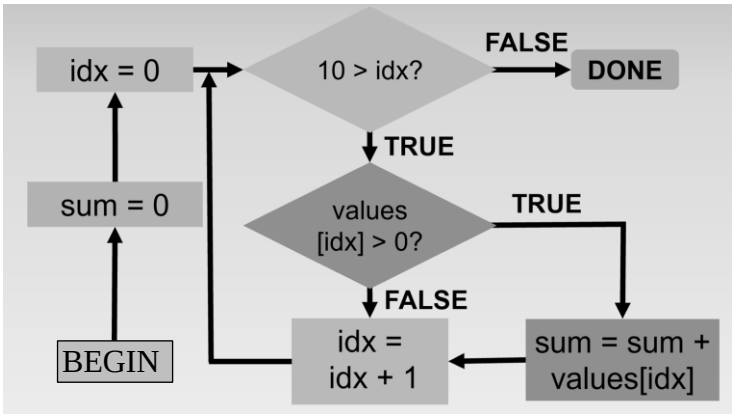


Figure 1 flow chart of the given C code

Given the available components, your project teammates have already selected a datapath (as shown in Fig. 2). A brief description of each element in the datapath is given on the last page as an appendix. Note that the datapath uses a 32-bit serial adder to add two integer values, during the calculation of "sum = sum + values[idx]". **<u>Your goal is to design the FSM controller that generates the necessary control signals for the datapath to produce the expected functionality.</u>**

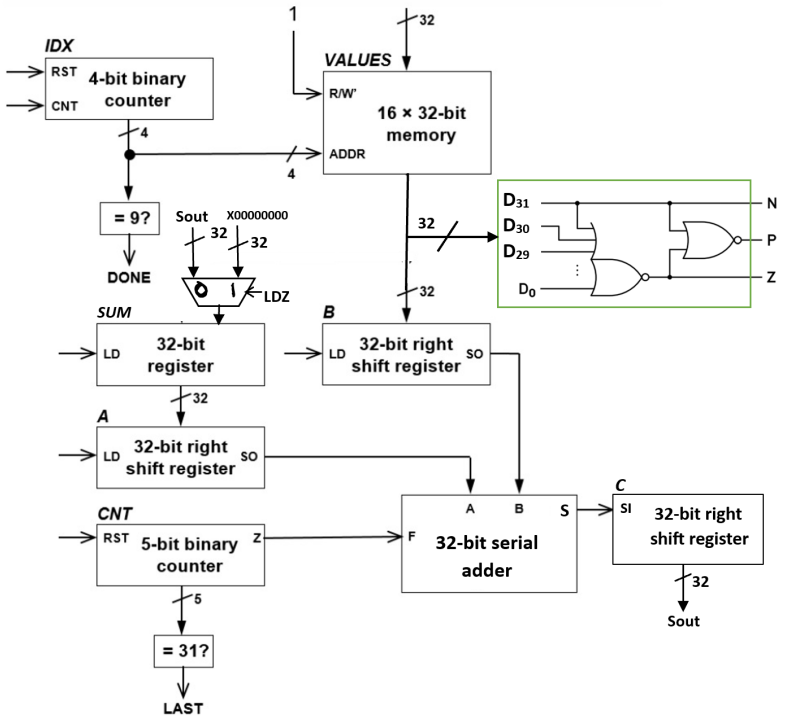


Figure 2 Datapath

Given the datapath and the flow chart, the first step of the FSM controller design process is to convert the flow chart of the program into its state diagram that can be realizable by the chosen datapath. Your teammates have gone back and forth between the components and the FSM, grouping the flowchart boxes as shown in Fig. 3(a) yielding the state diagram of the desired FSM controller as shown in 3(b).

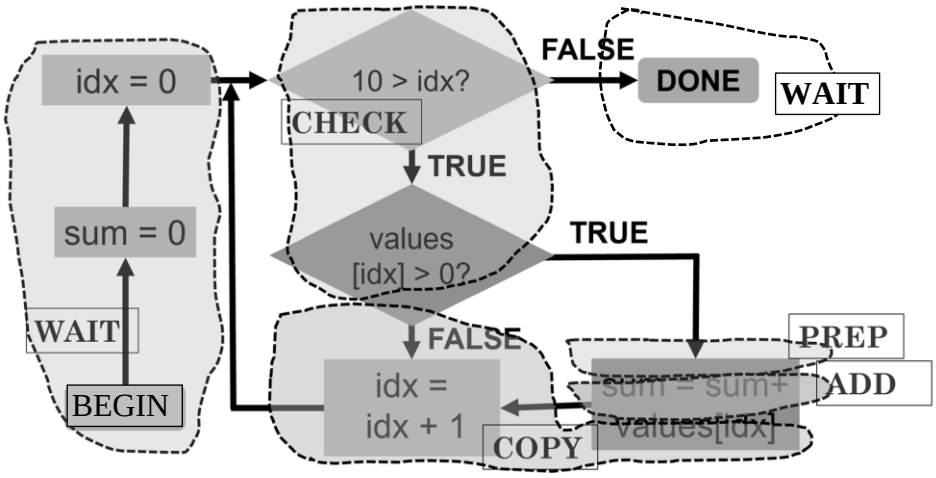


Figure 3(a)

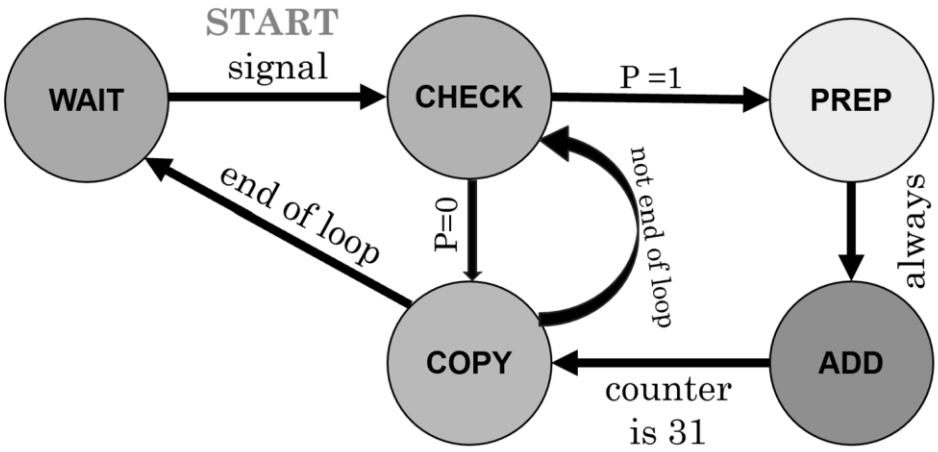


Figure 3 (b) Desired State Diagram

To help you get started with the design process, you have been given with a partially completed next state transition table (**Fig 4**) along with the RTL of each state, and the partially completed table with the necessary control signals that the state machine needs to generate in every state **(Fig 5)**. An external device generates START signal (not shown here) to let the FSM know that the memory has been filled in with 10 integer values

Question: [9+8 = 17 points] Complete the tables and write expressions (i.e. next states and output expressions) to design the FSM-controller.

Next state transitions expressions:
Example:
$S_1^+ =$
$S_0.START + S_4.DONE'$

State ($S_4S_3S_2S_1S_0$)	actions (simultaneous)	condition	next state
WAIT (00001)	IDX ← 0 SUM ← X00000000	START START'	**CHECK** ……
CHECK (00010)	No need to generate any control signal for Datapath. FSM just checks the P signal.	…… ……	**PREP** **COPY**
PREP (00100)	A ← SUM B ← VALUES[IDX] CNT ← 0	(always)	……
ADD (01000)	Run serial Adder. No need to generate any control signal for Datapath. FSM just checks the LAST signal	LAST ……	**COPY** ……
COPY (10000)	P: SUM ← Sout IDX ← IDX+1 FSM just checks the DONE signal	…… DONE'	**WAIT** **CHECK**

Figure 4

Output expressions:
Example, **IDX.RST = S_0**

state	$S_4S_3S_2S_1S_0$	IDX. RST	IDX. CNT	SUM. LD	LDZ	A. LD	B. LD	CNT.RST
WAIT	**00001**	1	0	1	1	0	0	0
CHECK	**00010**	0						
PREP	**00100**	0						
ADD	**01000**	0						
COPY	**10000**	0	1	P	0			

Figure 5

[8 points] Now draw the FSM controller using the expressions you developed and complete the datapath in **Figure 6**. Specifically,
(a) **show the necessary circuits for the next state transitions and connect them to the corresponding FFs,**
(b) **label the control signals generated by your FSM-controller**, and
(c) **connect the control signals to the corresponding inputs on the datapath**.

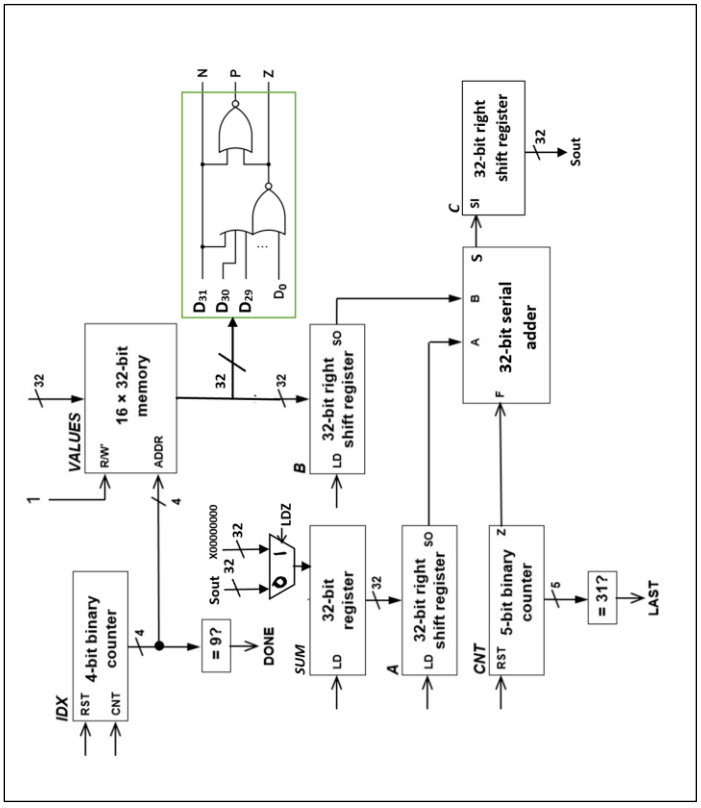


Figure 6 Datapath

Transition logic	DFFs	Output logic

Question: [9+8 = 17 points] Complete the tables and write expressions (i.e. next states and output expressions) to design the FSM-controller.

Next state transitions expressions:
Example:
$S_1^+ =$
$S_0.START + S_4.DONE'$

$S_0^+ = S_0 START' + S_4 DONE$

$S_2^+ = S_1 P$

$S_3^+ = S_2 + S_3 LAST'$

$S_4^+ = S_3 LAST + S_1 \cdot P'$

State ($S_4S_3S_2S_1S_0$)	actions (simultaneous)	condition	next state
WAIT (00001)	IDX ← 0 SUM ← X00000000	START START'	CHECK wait
CHECK (00010)	No need to generate any control signal for Datapath. FSM just checks the P signal.	P P'	PREP COPY
PREP (00100)	A ← SUM B ← VALUES[IDX] CNT ← 0	(always)	ADD
ADD (01000)	Run serial Adder. No need to generate any control signal for Datapath. FSM just checks the LAST signal	LAST LAST'	COPY ADD
COPY (10000)	P: SUM ← Sout IDX ← IDX+1 FSM just checks the DONE signal	DONE DONE'	WAIT CHECK

Output expressions:
Example, **IDX.RST = S_0**

$IDX.CNT = S_4$

$SUM.LD = S_0 + P$

$LDZ = S_0$

$A.LD$
$B.LD$
$CNT.RST$ $\} = S_2$

state	$S_4S_3S_2S_1S_0$	IDX. RST	IDX. CNT	SUM. LD	LDZ	A. LD	B. LD	CNT.RST
WAIT	00001	1	0	1	1	0	0	0
CHECK	00010	0	0	0	0	0	0	0
PREP	00100	0	0	0	0	1	1	1
ADD	01000	0	0	0	0	0	0	0
COPY	10000	0	1	P	0	0	0	0

could be X

Figure 5

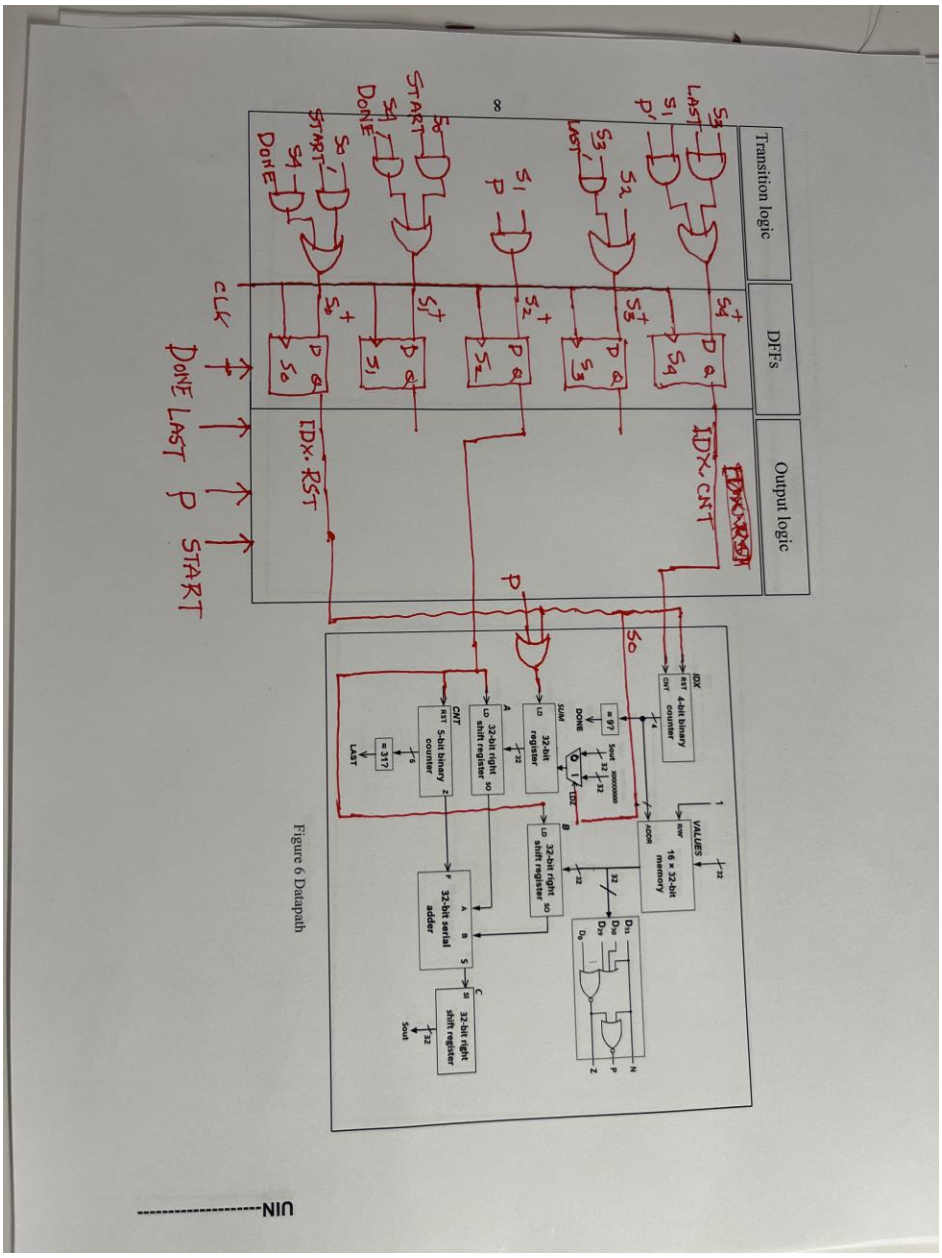


Figure 6 Datapath

Problem 4 (14 points): Memory

1. **(8 points)** The figure below shows the block diagram of a 64K x 8-bit RAM chip.

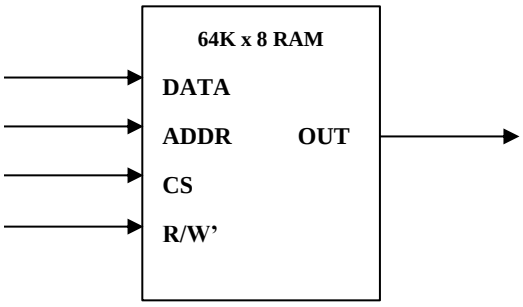
(Note: 1k = 1024.) Provide the inputs to the RAM chip in order to store the ASCII character uppercase 'W' in location 0xECEB. Answer in binary with the correct number of bits.

DATA _____

ADDR _____

CS _____

R/ W' _____



2. **(6 Points)** You are provided with the following 6 RAM chips:

   - Two 64x8 RAM
   - Four 32x4 RAM

You are also supplied with decoders and MUXes of your choice. You are asked to build larger memories out of these components making sure that all address and data bits are used in all RAM chips. Hint: Draw a picture of the memory chip arrangement for each of the configurations in parts (a) and (b).

(a) What is the largest memory addressability that you can achieve? And the corresponding address space? How many bits do you need for the address?

We can build a RAM unit with:

Addressability of _____ bits

Address space of _____ memory locations (addresses)

Address of _____ bits

(b) What is the largest memory address space that you can achieve? And the corresponding addressability? How many bits do you need for the address?

We can build a RAM unit with:

Addressability of _____ bits

Address space of _____ memory locations (addresses)

Address of _____ bits

Problem 4 (14 points): Memory

1. **(8 points)** The figure below shows the block diagram of a 64K x 8-bit RAM chip.

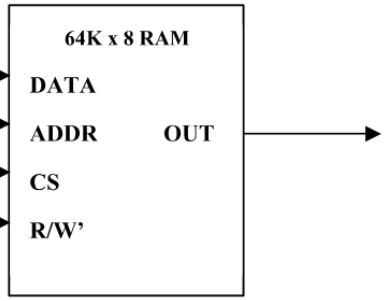
(Note: 1k = 1024.) Provide the inputs to the RAM chip in order to store the ASCII character uppercase 'W' in location 0xECEB. Answer in binary with the correct number of bits.

DATA 0101 0111

ADDR 1110 1100 1110 1011

CS 1

R/ W' 0



(a) What is the largest memory addressability that you can achieve? And the corresponding address space? How many bits do you need for the address?

We can build a RAM unit with:

Addressability of 24 bits

Address space of 64 memory locations (addresses)

Address of 6 bits

(b) What is the largest memory address space that you can achieve? And the corresponding addressability? How many bits do you need for the address?

We can build a RAM unit with:

Addressability of 8 bits

Address space of 192 memory locations (addresses)

Address of 8 bits

Problem 5 (21 points): LC-3 instructions

The following LC-3 program fragment, represented as three hexadecimal numbers, is stored in memory at the indicated locations and the following values are stored in registers:

address	instruction
xB4FF	X993F
xB500	X6503
xB501	X7903
xB502	XECEB

register	value
R0	xAAAA
R1	xBBBB
R2	xCCCC
R3	X2222

register	value
R4	X4B00
R5	X4B01
R6	X1111
R7	X4B03

1. **(6 points)** Re-write three instructions in binary representation and provide their corresponding RTL. (*Note: formats of the entire LC-3 instruction set are provided at the end of the exam booklet.*)

address	instruction	binary instruction	RTL (be specific to this instruction)
xB4FF	X993F		
xB500	X6503		
xB501	X7903		

2. **(10 points)** Assuming PC is initially set to xB4FF, trace the execution of the given program segment for **two** instruction cycles, filling in the table below. Write down the values stored in the PC, IR, MAR, MDR, N, Z, and P registers **at the end of the instruction cycle**. Values for PC, IR, MAR, and MDR should be written in hexadecimal. Values for N, Z, and P should be written in binary.

PC	IR	MAR	MDR	N	Z	P

3. **(1 point)** What hexadecimal value will be stored in R2 after the three instruction cycles?

   Answer: _____

4. **(1 point)** What hexadecimal value will be stored at address xB502 after the three instruction cycle??

   Answer: _____

5. **(3 point)** Add an instruction at address xB503 such that value xDDDD will be stored at R5 after the instruction cycle. You will need the information that xDDDD is not located anywhere in memory. Write your answer in hexadecimal.

   Answer: _____

The following LC-3 program fragment, represented as three hexadecimal numbers, is stored in memory at the indicated locations and the following values are stored in registers:

address	instruction
xB4FF	X993F
xB500	X6503
xB501	X7903
xB502	XECEB

register	value
R0	xAAAA
R1	xBBBB
R2	xCCCC
R3	X2222

register	value
R4	X4B00
R5	X4B01
R6	X1111
R7	X4B03

0100 1011 0000 0000
1011 0100 1111 1111
B 4 F F

1. **(6 points)** Re-write three instructions in binary representation and provide their corresponding RTL. (*Note: formats of the entire LC-3 instruction set are provided at the end of the exam booklet.*)

address	instruction	binary instruction	RTL (be specific to this instruction)
xB4FF	X993F	1001 1001 0011 1111	R4 ← NOT(R4)
xB500	X6503	0110 0101 0000 0011	R2 ← M[R4+SEXT[3]]
xB501	X7903	0111 1001 0000 0011	M[R4+SEXT[3]] ← R4

2. **(10 points)** Assuming PC is initially set to xB4FF, trace the execution of the given program segment for **two** instruction cycles, filling in the table below. Write down the values stored in the PC, IR, MAR, MDR, N, Z, and P registers **at the end of the instruction cycle**. Values for PC, IR, MAR, and MDR should be written in hexadecimal. Values for N, Z, and P should be written in binary.

PC	IR	MAR	MDR	N	Z	P
x B500	x 993F	x B4FF	x 993F	1	0	0
x B501	x 6503	x B502	x ECEB	1	0	0

3. **(1 point)** What hexadecimal value will be stored in R2 after the three instruction cycles?

Answer: xECEB

4. **(1 point)** What hexadecimal value will be stored at address xB502 after the three instruction cycle??

Answer: xB4FF

5. **(3 point)** Add an instruction at address xB503 such that value xDDDD will be stored at R5 after the instruction cycle. You will need the information that xDDDD is not located anywhere in memory. Write your answer in hexadecimal.

Answer: x0001 101 001 0 00 011

x 1 A 43 or x1AC1

ADD R5, R1, R3
ADD R5, R3, R1

Appendix:

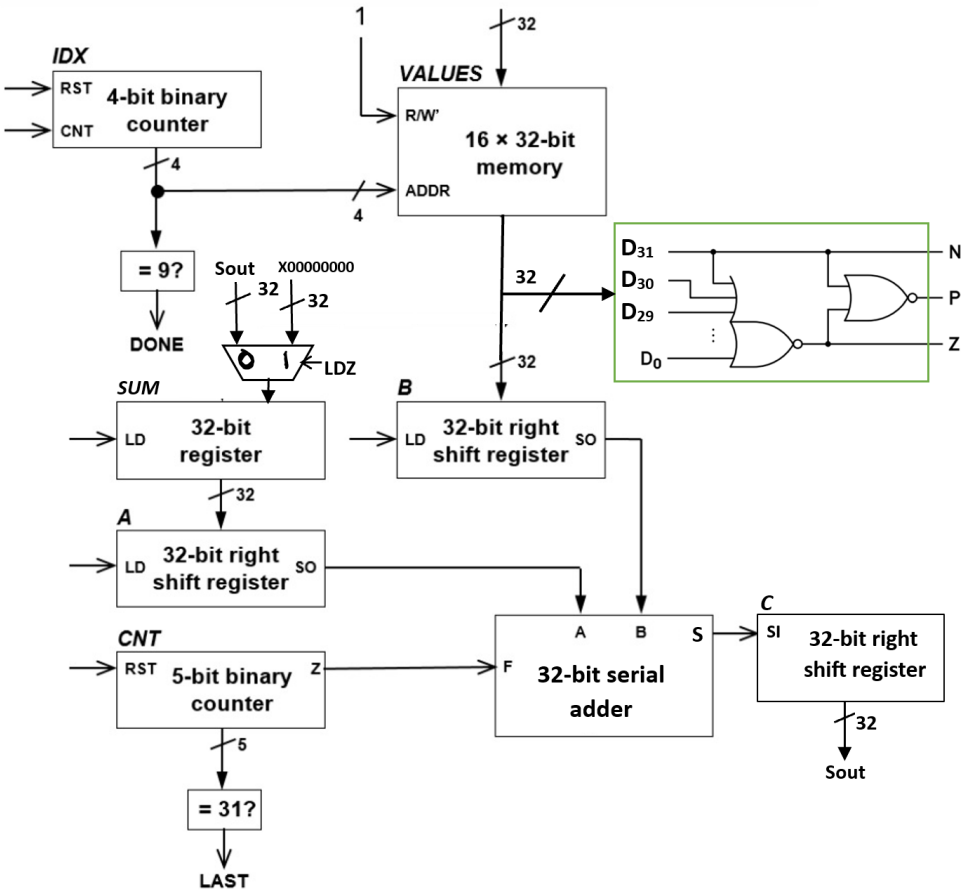


Figure 7 Datapath

VALUES – it is a 16x32 memory to hold the ten integer values of the array, values[]. We assume that the memory was filled in with the 10 integer values by some external device before the FSM start processing the data. The external device generates START signal (not shown here) to let the FSM know that the memory has been filled in with 10 integer values. The R/W' is set to 1 to enable memory read mode during the operation of the FSM. values[idx] refers to the memory element at the current memory location/address.

IDX – is a 4-bit counter. The counter output is used as the address line for the memory to read ten integer values in order. The **DONE** signal generated by the counter is used to indicate whether the FSM has processed ten integer values of the array.

Control signals: IDX.RST is used to reset the counter to zero. IDX.CNT is used to increment the counter. When IDX.CNT is 1, the counter is incremented by 1 with the rising edge of the clock.

CNT – is a 5-bit counter. It is used to count whether the serial adder has completed processing 32 pairs of bits of shift registers A and B. The **LAST** signal becomes 1 when the serial adder has received the last pair of bits of A and B. The counter is initialized to zero at the beginning of every serial addition process (i.e. when the shift registers A and B are loaded with two integer values to be added by the 32-bit serial adder). It generates a signal, Z=1, when the counter is reset to zero, to indicate that the first pair of bits of A and B have arrived at the serial adder. Otherwise, Z will be set to 0.

Control signals: CNT.RST =1 is used to reset the counter to zero. Otherwise, the counter is incremented by 1 with every rising of the clock.

NPZ – is used to indicate whether the value at the current memory address/location, i.e. values[idx] is negative, positive or zero.
SUM – is a 32-bit register that will hold the summation of the positive integers of the array. The SUM is connected to a multiplexer so that it can be initialized to zeros, or it can be loaded with the Sout (i.e. sum+values[idx]) during the execution of FSM. Note: SUM will hold the result only for one cycle after the COPY state.

Control signal: LD signal along with the multiplexer selection input LDZ are used to initialize the register or to load the current sum into the register.
A, B, and C – are 32-bit right shift registers. A and B are connected to the serial inputs of the 32-bit serial adder and C is connected to the output of the serial adder. Control signals LD.A and LD.B are used to load A with SUM, and B with values[idx] respectively. Serial adder output S is connected to the SI (serial Input) of C, which stores the results of serial addition. Note: **Sout** will have the result of 32-bit serial addition in the next cycle when **LAST** signal becomes 1 (i.e. after 32 pairs of bits of A and B have been processed in the serial adder).

Table of ASCII Characters

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
(nul)	0	00	(sp)	32	20	@	64	40	`	96	60
(soh)	1	01	!	33	21	A	65	41	a	97	61
(stx)	2	02	"	34	22	B	66	42	b	98	62
(etx)	3	03	#	35	23	C	67	43	c	99	63
(eot)	4	04	$	36	24	D	68	44	d	100	64
(enq)	5	05	%	37	25	E	69	45	e	101	65
(ack)	6	06	&	38	26	F	70	46	f	102	66
(bel)	7	07	'	39	27	G	71	47	g	103	67
(bs)	8	08	(	40	28	H	72	48	h	104	68
(ht)	9	09	)	41	29	I	73	49	i	105	69
(lf)	10	0a	*	42	2a	J	74	4a	j	106	6a
(vt)	11	0b	+	43	2b	K	75	4b	k	107	6b
(ff)	12	0c	,	44	2c	L	76	4c	l	108	6c
(cr)	13	0d	-	45	2d	M	77	4d	m	109	6d
(so)	14	0e	.	46	2e	N	78	4e	n	110	6e
(si)	15	0f	/	47	2f	O	79	4f	o	111	6f
(dle)	16	10	0	48	30	P	80	50	p	112	70
(dc1)	17	11	1	49	31	Q	81	51	q	113	71
(dc2)	18	12	2	50	32	R	82	52	r	114	72
(dc3)	19	13	3	51	33	S	83	53	s	115	73
(dc4)	20	14	4	52	34	T	84	54	t	116	74
(nak)	21	15	5	53	35	U	85	55	u	117	75
(syn)	22	16	6	54	36	V	86	56	v	118	76
(etb)	23	17	7	55	37	W	87	57	w	119	77
(can)	24	18	8	56	38	X	88	58	x	120	78
(em)	25	19	9	57	39	Y	89	59	y	121	79
(sub)	26	1a	:	58	3a	Z	90	5a	z	122	7a
(esc)	27	1b	;	59	3b	[	91	5b	{	123	7b
(fs)	28	1c	<	60	3c	\	92	5c	|	124	7c
(gs)	29	1d	=	61	3d	]	93	5d	}	125	7d
(rs)	30	1e	>	62	3e	^	94	5e	~	126	7e
(us)	31	1f	?	63	3f	_	95	5f	(del)	127	7f