**Note:**  (Late: -15% Penalty).

1. (15 pts) Write a brief essay paragraph explaining one breakthrough in the history of computing and the significance of the breakthrough to advancing Computer Science.

Acceptable answers include explanations of vacuum tubes, transistors, integrated circuits, VLSI, binary arithmetic, quantum computing, and parallel computing.

Ans.

1. (15 pts) Using the CRC polynomial 1101, compute the CRC code for the information word, 1 0110 1101

Ans:

1 0110 1101 CRC 001

3rd order polynomial so add 3 digits to information word.

101101101 000

1101 XOR

01100

1101

0001110

1101

001110

1101

001100

1101

0001 CRC 001

1. (10 pts) Convert the following decimal fractions to binary with a maximum of six places to the right of the binary point:
   1. 26.78125 11010.110010
   2. 194.03125 11000010.000010
2. (15 pts) Define the following
   1. Combination Logic:

are made up from basic gates( AND, OR, NOT) or universal gates( NAND, NOR) gates that are “combined” or connected together to produce more complicated switching circuits.

* 1. Sequential Logic:

is a type of logic circuit whose output depends not only on the present value of its input signals but on the sequence of past inputs, the input history as well.

* 1. How are sequential circuits different than combinational circuits?

Sequential circuits are those which are dependent on clock cycles and depends on present as well as past inputs to generate any output. Combinational circuit in this output depends only upon present input.

|  |  |  |
| --- | --- | --- |
| Comparison | Combinational circuit | Sequential circuit |
| Basic | The output is discovered by the present state of the inputs. | Both the present input and past state output are used to identify the output. |
| Storage capability | Does not store data. | Can store a small amount of data. |
| Application | used in adders, encoders, multiplexer, etcetera. | Flip-flop and latches. |
| clock | Circuits do not rely on the clock. | Clock is utilized for performing triggering functions. |
| feedback | No requirement of the feedback. | Feedback is required. |

1. (10 pts) Add the following 8-bit two’s complement numbers (i.e. one sign bit and seven data bits) AND indicate “Overflow” if it occurs.

1110 1010

a.  1111 0101

+ 1101 0101

Answer: 1 1100 1010 No Overflow (Sign bit did not changed)

1111 1110

c.  0110 1011

+ 0101 0101

Answer: 0 1100 0000

Overflow: A Positive Number + a Positive number cannot be a Negative number

1. (15 pts) Given the boolean equation: x’yz + x(yz)' + x'(y+z) + (xyz)'
   1. Construct the truth table for the Boolean equation:

|  |  |  |  |
| --- | --- | --- | --- |
| x | y | z | F(xyz) |
| 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 | 0 |

* 1. Construct the K-map for this circuit.

xy

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| z | 00 | 01 | 11 | 10 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

* 1. Write a reduced but equivalent Boolean equation. Use either the K-map or apply the Boolean postulates/theorem’s to do your reduction.

(XYZ)`

1. (15 pts) Assume a 220 byte memory:
   1. What are the lowest and highest addresses if memory is byte-addressable?

0 through 220-1

* 1. What are the lowest and highest addresses if memory is word-addressable, assuming a 16-bit word?

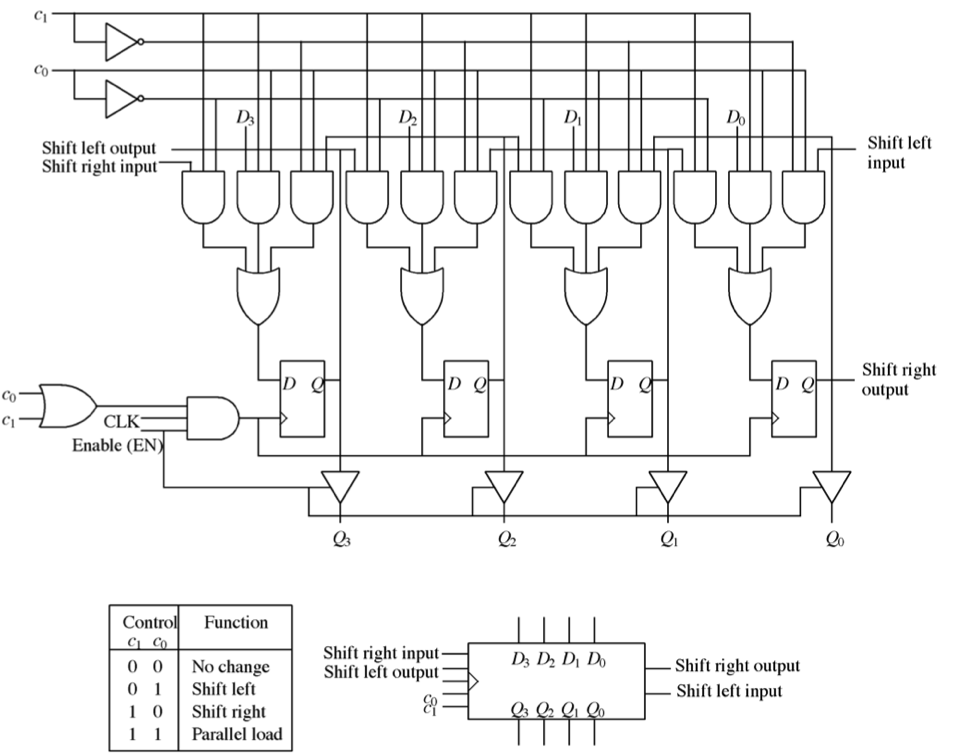
 0 through 219-1

* 1. What are the lowest and highest addresses if memory is word-addressable, assuming a 32-bit word?

0 through 218-1

1. (5 pts) Given the left-right shift register shown below answer the following questions. Place an ‘x’ for don’t care conditions on the non-relevant outputs.

Note: Each Control input ‘c0 and c1’ as listed in the function table is shown twice in the diagram.



Enable

1. Fill in the state output of Q0-to-Q3 on clock Clk4 given the input ‘Enable=1’ AND the following Input serial load of the shift-register in clocks Clk0-to-Clk3

Clk0-to-Clk3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Clk0 | Clk1 | Clk2 | Clk3 | C1 | C0 |
| Shift Right Input | 0 | 1 | 0 | 0 | 0 | 1 |
| Shift Left Input | 1 | 0 | 1 | 1 |

Clk4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Q3 | Q2 | Q1 | Q0 | C1 | C0 | Enable |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |

1st four clocks control bits set to ‘shift right’ so use the shift right input bit sequence to input values into the latches. Clk4 control bits are set to ‘no change’ and enable provides latch outputs to Q0 through Q3

1. (15 pts) Using the MarieSim emulator - Write working assembly code that executes the logical programming construct:

Turing machines and Finite State Machines will continue until told to ‘halt’. Please halt your code upon completion of the assigned task. Include comments to your code!

Initialize A=5; B=9;

UNTIL (A >=B) ! Until A is greater than or equal to B

DO

C = A + B ! Add A to B

STORE C ! store in C

A=A-1 ! Subtract 1 from A

B=B-2 ! Subtract 2 from B

DONE

Answer:

org 100

Loop, LOAD A /load A into AC

ADD B /add B to contents of AC

STORE C /store result in C

LOAD A /load A into AC

SUBT ONE /subtact value stored in ONE

STORE A /store result in A

LOAD B /load B into AC

SUBT TWO /subtract value in TWO from AC

STORE B /store result in B

LOAD B /load B into AC

SUBT A /subtract A from AC

Skipcond 000 /skip next instruction if AC is negative

Jump Loop /jump to Loop

HALT /end program

A, DEC 5

B, DEC 9

C, HEX 0

ONE, DEC 1

TWO, DEC 2

CTR, HEX 0

1. (15 pts) Using the MarieSim emulator - Write a working assembly code that executes the logical programming construct:

Turing machines and Finite State Machines will continue until told to ‘halt’. Please halt your code upon completion of the assigned task. Include comments to your code!

IF (A = B)

THEN

C = A + B

ELSE

C = A – B

Answer:

If, Load A /Load the first value

Subt B /Subtract the value of B; store the result in AC

Skipcond 400 /If AC=0, skip the next instruction

Jump Else /Jump the Else part if AC is not equal to 0

Then, Load A /Load A to do summation

Add B /Calculate the sum

Store C /Store the result in C

Jump Endif /Skip over the else part of the program

Else, Load A /Start the subtraction by loading A

Subt B /Subtract B from A

Store C /Store the result in C

Endif, Halt /Terminate the program

A, Dec 12 /The A value

B, Dec 10 /The B value

C, Dec 000 /Initializing C

1. (15 pts) Write working assembly code that executes the logical programming construct:

Include comments to your code!

Write a working assembly code that uses **stack parameter passing linkage** to call a subroutine labeled “**sub\_1**” to subtract two numbers and return the results to the main routine. t = (r – s)

Your main routine must:

* 1. load your two parameters (r,s) from memory and push them onto your stack.
  2. call sub\_1
  3. pop the result from your stack and store it into memory label ‘t: ‘.

Your sub-routine must:

1. pop your two parameters from your stack
2. subtract the parameters as shown in the problems formula
3. push the result onto the stack
4. return to your ‘main’ routine

Please initialize ‘t: 0’ to start your code. t = (r – s)

Use care not to let your memory stack step on your code.

Answer:

Load R

Store Temp /Stores R in a memory location from which Push subroutine will retrieve it

JnS Push /Jumps to the Push subroutine and stores return address

Load S

Store Temp

JnS Push

JnS sub\_1 / Jumps to the sub\_1 subroutine and stores return address

JnS Pop /Jumps to Pop subroutine which will retrieve the value of T

Store T

Halt

sub\_1, Hex 0 /Return address will be stored here

JnS Pop /Pops the value of S

Store Temp1

JnS Pop /Pops the value of R

Subt Temp1 /R - S

Store Temp /T

JnS Push /Pushes T into the stack

JumpI sub\_1 /Returns from sub\_1

Push, Hex 0

Load Pointer

Add One /Increments Pointer by 1

Store Pointer

Load Temp /Loads value to be pushed into the stack

StoreI Pointer /Stores previously loaded value using the address indicated by Pointer

JumpI Push /Returns from the Push subroutine

Pop, Hex 0

LoadI Pointer /Loads a value from the address indicated by Pointer

Store Temp

Load Pointer

Subt One /Decrements the Pointer

Store Pointer

Load Temp

JumpI Pop /Returns from Pop subroutine

R, Dec 12 /Values for R and S are defined for testing purposes only

S, Dec 8

One, Dec 1 /Used to increment or decrement by 1

Pointer, Hex 10B /Initial address for the Pointer

Temp, Dec 0 /Defines temporary storage for use of subroutines

T, Dec 0

Temp1, Dec 0 /One of the subroutines calls another, so I need 2nd temporary storage

org 100

LOAD R /load R from memory into AC

StoreI LOC1 /store value in AC (R) to address LOC1

LOAD S /load S from memory into AC

StoreI LOC2 /store value in AC (S) to address LOC2

JnS Subr /Jump to subroutine

LoadI LOC1 /loads value from LOC1 into AC

STORE T /stores value from AC into T

HALT

LOC1, HEX EE0

LOC2, HEX EE1

R, DEC 20

S, DEC 4

T, HEX 0

tempS, DEC 0

tempR, DEC 0

tempT, HEX 0

Subr, HEX 0

LoadI LOC2 /load value from LOC2 to AC

STORE tempS /store value in AC to tempS

LoadI LOC1 /load value from LOC1 to AC

STORE tempR /store value in AC to tempR

LOAD tempR /load value in tempR into AC

SUBT tempS /subtract tempS from value in AC

STORE tempT /store result into tempT

StoreI LOC1 /store value in AC at LOC1

JumpI Subr /return to main routine

1. (15 pts) Write working assembly code that executes the array initialization C code shown below:

Include comments to your code!

*#include <stdio.h>*

*int main () {*

*int n[ 10 ]; /\* n is an array of 10 integers \*/*

*int i;*

*/\* initialize elements of array n to 0 \*/*

*for ( i = 0; i < 10; i++ ) {*

*n[ i ] = i + 100; /\* set element at location i to i + 100 \*/*

*}*

*return 0;*

*}*

Loop, Load Size

Subt Counter

Skipcond 800 /Skip the next command if AC is positive

Halt

Load Counter

Add Hundred /Adds 100 to the value of AC

StoreI Arrayptr /Stores the value of AC in the address indicated by Arrayptr

Load Arrayptr

Add One /Increment Arrayptr by 1

Store Arrayptr

Load Counter

Add One /Increment Counter by 1

Store Counter

Jump Loop /Returns to the beginning of the loop

One, Dec 1

Hundred, Dec 100

Counter, Dec 0 /Index of the current array element

Size, Dec 10 /Size of the array

Arrayptr, Hex 10E /Address for the next array element to be stored