CSA (March 16 to 22)

Topics

Practical 3 : Write a program that will prompt for the input of a binary value. Find out following compliments.
a. One's complement
b. Two's complement

```
import sys
def Complement(binarySequence):
    convertedSequence = [0] * len(binarySequence)
    carryBit = 1
    for i in range(0, len(binarySequence)):
        if binarySequence[i] == '0':
            convertedSequence[i] = 1
        else:
            convertedSequence[i] = 0
    print(''.join(str(x) for x in convertedSequence))
    if convertedSequence[-1] == 0:
            convertedSequence[-1] = 1
            print(''.join(str(x) for x in convertedSequence))
    else:
        for bit in range(0, len(binarySequence)):
            if carryBit == 0:
                break
            index = len(binarySequence) - bit - 1
            if convertedSequence[index] == 1:
                convertedSequence[index] = 0
                carryBit = 1
            else:
                convertedSequence[index] = 1
                carryBit = 0
        print(''.join(str(x) for x in convertedSequence))
while True:
    binarySequence=input("Enter binary number : ")
    i=0
    for x in binarySequence:
        if int(x)<2:
            i=i+1
            continue
        else:
            print("Not a valid binary number !!!\n")
```

```
            break
    if i==len(binarySequence):
            Complement(binarySequence)
    while True:
        ans=input("Would you like to continue : (Y/N) ")
        if ans=="Y":
            break
        elif ans=="N":
            sys.exit()
        else:
            print("\nNot Valid Choice Try again !!!")
```

Practical 4:
Write a program to print the values of a 5 bit binary up-down counter. User should be able to specify the up or down nature of the counter.

```
import sys
counter=[0]*5
def incr(counter):
    carry = False
    bits = list(counter[::-1])
    for i, bit in enumerate(bits):
        carry = (bit != "0")
        if bit == "0":
            bits[i] = "1"
            break
        else:
            bits[i] = "0"
    return ''.join(str(x) for x in bits[::-1])
def decr(counter):
    bits = list(counter[::-1])
    for i, bit in enumerate(bits):
        if bit == "1":
            bits[i] = "0"
            break
        else:
            bits[i] = "1"
    return ''.join(str(x) for x in bits[::-1])
C=input("DO you have some initial input for counter : ")
if C=='y':
    print("Enter 5 bit value : ")
    for i in range(0,5):
```

```
        counter[i]=input()
    counter=list(counter)
    print ("Value in counter :"+".join(str(x) for x in counter))
else:
    print (".join(str(x) for x in counter))
while True:
 print("Want to up or down the counter ")
 i=int(input("For UP enter 1 & For DOWN enter 2 : "))
 counter=".join(str(x) for x in counter)
 if i==1:
    if counter=='11111':
        counter=[0]*5
        counter=".join(str(x) for x in counter)
        print(counter)
    else:
        counter=incr(counter)
        print(counter)
 elif i==2:
    if counter=='00000':
        counter=[1]*5
        counter=".join(str(x) for x in counter)
        print(counter)
    else:
        counter=decr(counter)
        print (counter)
 else:
    print("Please enter correct choice!!")
    break
 while True:
     ans=input("would you like to continue_")
     if ans=='y':
         break
     elif ans=='n':
         sys.exit();
     else:
         print ("invalid choice")
```

CSA 5.4 Timing and Control
https://youtu.be/lXGsXnxRzE4

CSA 5.5 Instructions Cycle
https://youtu.be/SFsnysyVhzA

Find the below hands written notes of Topic 5.4 and 5.5

Types of Instructions

1) Arithmetic, Logic and shift
2) Memory ⇄ Registers
3) Program control [If-else]
4) Input and output

- ADD
- AC(CMA) – complement
- Ac(INC) – Increment

By using these 3 Instructions, we can do binary Addition and Substraction.

CIR - CIL => for shifts

5-4 Timing and Control

Register Master clock

The clock pulses do not change the state of a register unless the register is enabled by a control signal.

The control signals are generated in control unit and provide control inputs for the multiplexers in the common bus, control inputs in ~~Common~~ bus in processor registers, and microoperations for the accumulator.

Types of Control organisations
① ↳ Hardware organisation
② ↳ Microprogrammed " .

In ① Control Logic is implemented with Gates, flip-flops, Decoders and other digital circuits.

Advantage:- produce fast mode of operations.

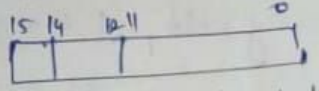
In ② Control Information is stored in a control Memory. The control memory is programmed to initiate the required sequence of microoperations.

Hardware Control for basic Computer is presented here.

Control unit consists of
- 2 Decoders
- 1 Sequence Counter
- number of control logic Gates

Diagram from book (Fig 5.6)

An instruction read from memory is placed in IR (Instruction Register)



Bits 12-14 are decoded into 3x8 decoder.

Bits 0-11 are applied to control logic gates

The sequence counter (SC) can be incremented or cleared synchronously. We have timing signal from T_0 to T_5, generated by a decoder of 4x16.

$D_3 T_4 :$ SC ← 0

Sequence counter is providing time T_0, T_1, T_2, T_3, T_4 and at T_4, SC is cleared to 0 if Decoder output D_3 is Active.

$T_0 :$ AR ← PC

Specify transfer the content of PC into AR if timing signal T_0 is active.

5-5 Instruction cycles

Prog (in Memory unit) → seq' of in instructions.

Each instruction consist of cycle

Each cycle consist of subcycle at Phases.

In the basic computer each instruction cycle consists of the following phases:

1) Fetch the instruction from Memory
2) Decode the Instruction
3) Read the effective address from memory if the instruction has an indirect address
4) Execute the instruction

Fetch → Decode → Decision → Execute

Fetch and decode

Initially SC ← 0

Fetch:
- $T_0 :$ AR ← PC
- $T_1 :$ IR ← M[AR], PC ← PC + 1

Decode:
- $T_2 :$ $D_0, \ldots, D_7$ ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)

I 000 →
0 111 → reg'
1 111 → I/O

Decision:
- $D_7' I T_3 :$ AR ← M[AR]
- $D_7' I' T_3 :$ Nothing
- $D_7 I' T_3 :$ Execute a reg'-ref' instruction
- $D_7 I T_3 :$ I/O Instruction

Register Reference Instructions

$D_7 = 1, \quad I = 0$

Each control fn' needs the Boolean relation $D_7 I' T_3$, which we designate by symbol r.

By assigning the symbol B_i to bit i of IR, all the control fn' simply denoted by $r B_i$

. CLA has hexa code 7800

Binary equivalent

0111 1000 0000 0000

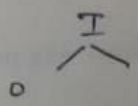
I opcode | IR(0-11)

$D_7 I' T_3 B_{11} = r B_{11}$

Memory-Reference Instructions

	I=0	I=1
AND	0	8
ADD	1	9
LDA	2	A
STA	3	B
BUN	4	C
BSA	5	D
ISZ	6	E

The decoded output D_i for i = 0, 1, 2, 3, 4, 5 and 6 belongs to seven operations.



T_2 T_3

The effective address of the instruction is in AR was placed there during timing signal T_2 when I=0

OR

T_3 when I=1

Symbolic Representation

AND D_0 $AC \leftarrow AC \wedge M[AR]$

But the actual execution of instruction in the bus system will require a seq of microoperations. This is because data stored in memory cannot be processed directly

AND to AC

D_0T_4 : $DR \leftarrow M[AR]$

D_0T_5 : $AC \leftarrow AC \wedge DR$, $SC \rightarrow 0$

ADD to AC

D_1T_4 : $DR \leftarrow M[AR]$

D_1T_5 : $AC \leftarrow AC + DR$, $E \leftarrow Cout$, $SC \leftarrow$

LDA : Load to AC

D_2T_4 : $DR \leftarrow M[AR]$

D_2T_5 : $AC \leftarrow DR$, $SC \leftarrow 0$

STA : Store to AC

D_3T_4 : $M[AR] \leftarrow AC$; $SC \leftarrow 0$

BUN : Branch Unconditionally (Jump)

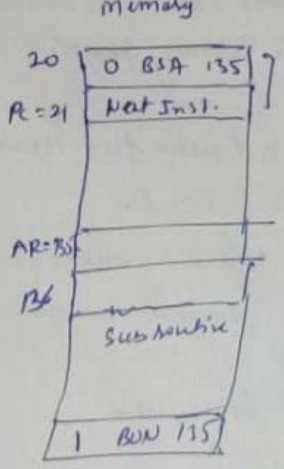
D_4T_4 : $PC \leftarrow AR$, $SC \leftarrow 0$

as PC holds the address of next instruction.

BSA : Branch and Save Return Addr

(Subroutine or procedure or function)

$M[AR] \leftarrow PC$, $PC \leftarrow PC+1$



(a) Memory, PC and AR at time T_4