CHECKSUM (BINARY)

```
SIZE = 8
def binary_to_decimal(bin_str):
   return int(bin_str, 2)
def decimal_to_binary(num): return
   format(num, '08b')
def calculate_checksum(segments):
   total = sum(binary_to_decimal(seg) for seg in segments)
   checksum = (~total) C 0xFF
   return decimal_to_binary(checksum)

def main():
   input_bits = input("Enter binary data: ").strip() while
   len(input_bits) % SIZE != 0:
      input_bits = '0' + input_bits
   segments = [input_bits[i:i+SIZE] for i in range(0, len(input_bits), SIZE)]
   checksum = calculate_checksum(segments)
   print("Checksum:", checksum)
   received_checksum = input("Enter received checksum: ").strip()
   segments.append(received_checksum)
   result = calculate_checksum(segments)
   print("Final checksum after complement:", result) if
   result == "00000000":
      print("Data is correct (No Error)")
   else:
      print("Error detected in data")

if _name____== "__main__":
   main()
```

CHECKSUM (HEXADECIMAL)

```
def compute_hex_checksum():
   n = int(input("Enter number of hexadecimal inputs: "))
   hex_values = []

   for i in range(n):
      h = input(f"Enter hex string {i+1} (e.g., 1A3F): ") hex_values.append(h)

   total = 0
   for h in hex_values:
      total += int(h, 16)

   while total > 0xFFFF:
      carry = total >> 16
      total = (total C 0xFFFF) + carry

   checksum = total ^ 0xFFFF

   print("\n--- Output ---")
   print(f"Sum (before 1's complement):
{hex(total).upper()[2:].zfill(4)}")
   print(f"Checksum: {hex(checksum).upper()[2:].zfill(4)}")

compute_hex_checksum()
```

CRC

```
def xor(a, b):
    return ''.join('0' if i == j else '1' for i, j in zip(a[1:], b[1:]))

def mod2div(dividend, divisor): pick
    = len(divisor)
    tmp = dividend[:pick] while
    pick < len(dividend):
        tmp = xor(divisor, tmp) + dividend[pick] if tmp[0] == '1' else
xor('0'*pick, tmp) + dividend[pick]
        pick += 1
    return xor(divisor, tmp) if tmp[0] == '1' else xor('0'*pick, tmp)

def encode(data, key):
    remainder = mod2div(data + '0'*(len(key)-1), key)
    return data + remainder, remainder

def main():
    data = input("Enter data bits: ")
    key = input("Enter generator polynomial bits: ") codeword,
    crc = encode(data, key)

    print(f"\nCodeword sent: {codeword} (CRC: {crc})") received
    = input("Enter received codeword: ")
    print("Error detected." if '1' in mod2div(received, key) else "No
error detected.")

if name____== "__main__":
    main()
```

GO-BACK-N ARQ

```
def go_back_n(frames, window_size): i =
   0
   while i < len(frames):
      print(f"\nSending frames from {i} to {min(i + window_size - 1,
len(frames)-1)}:")
      for j in range(i, min(i + window_size, len(frames))):
         print(f"Sent Frame {j}")

      error = input("Did all frames get ACKed? (yes/no):
").strip().lower()
      if error == "yes":
         i += window_size
      else:
         error_frame = int(input("Enter the frame number where error
happened: "))
         print(f"Going back to Frame {error_frame}") i =
         error_frame

def main():
   n = int(input("Enter number of frames: ")) frames =
   list(range(n))
   window_size = int(input("Enter window size: "))
   go_back_n(frames, window_size)

if _name____== "__main__":
   main()
```

ISP

```
def ip_to_int(ip):
   parts = list(map(int, ip.split('.')))
   return parts[0]*256**3 + parts[1]*256**2 + parts[2]*256 + parts[3]
def int_to_ip(num):
   return
f"{(num>>24)C255}.{(num>>16)C255}.{(num>>8)C255}.{numC255}"
def next_power_of_2(x): p
   = 1
   while p < x:
      p *= 2
   return p

def main():
   base_ip = input("Enter base IP (e.g. 192.168.1.0): ")
   total_customers = int(input("Enter number of customers: "))
   customer_hosts = []
   for i in range(total_customers):
      h = int(input(f"Hosts needed for customer {i+1}: "))
      customer_hosts.append((i+1, h + 2))
   customer_hosts.sort(key=lambda x: x[1], reverse=True) start
   = ip_to_int(base_ip)
   print("\nIP Allocation:")
   for cid, hosts in customer_hosts:
      block = next_power_of_2(hosts)
      print(f"Customer {cid} --> {int_to_ip(start)} - {int_to_ip(start +
block - 1)} (Hosts: {hosts - 2})")
      start += block

if _name____== "__main__":
   main()
```

SUBNETTING

```
def ip_to_bin(ip):
  return ''.join(f"{int(part):08b}" for part in ip.split('.')) def
bin_to_ip(bin_str):
  return '.'.join(str(int(bin_str[i:i+8], 2)) for i in range(0, 32, 8))

def main():
  ip = input("Enter base IP (e.g. 192.168.1.0): ") mask =
  int(input("Enter subnet mask (e.g. 24): "))
  subnets = int(input("Enter number of subnets to create: "))

  bits_needed = 0
  while (1 << bits_needed) < subnets:
    bits_needed += 1

  new_mask = mask + bits_needed
  subnet_size = 2 ** (32 - new_mask)
  base_ip_bin = ip_to_bin(ip)
  base_network = base_ip_bin[:mask] + '0' * (32 - mask) print(f"\nNew
  subnet mask: /{new_mask}")
  print(f"Each subnet has {subnet_size} IPs\n")

  for i in range(subnets):
    subnet_bin = base_network[:mask] + f"{i:0{bits_needed}b}" + '0'
* (32 - mask - bits_needed)
    start_ip = bin_to_ip(subnet_bin)
    end_ip = bin_to_ip(bin(subnet_size - 1 + int(subnet_bin,
2))[2:].zfill(32))
    print(f"Subnet {i+1}: {start_ip} - {end_ip}")

if name____== "__main__":
  main()
```

HAMMING CODE

```
def encode(data):
    d = data[::-1]
    r = 0
    while (1 << r) < len(d) + r + 1: r += 1
    n, msg = len(d) + r, ['0'] * (len(d) + r)

    j = 0
    for i in range(1, n + 1):
        if i & (i - 1): msg[i - 1] = d[j]; j += 1

    for i in range(r):
        p = 0
        for j in range(1, n + 1):
            if j & (1 << i): p ^= int(msg[j - 1])
        msg[(1 << i) - 1] = str(p)

    return ''.join(msg[::-1])

def detect(received):
    rcv = received[::-1]
    r, e = 0, 0
    while (1 << r) < len(rcv) + 1: r += 1

    for i in range(r):
        p = 0
        for j in range(1, len(rcv) + 1):
            if j & (1 << i): p ^= int(rcv[j - 1])
        if p: e += (1 << i)

    print("Error at bit:", e) if e else print("No
    error detected.")
```

```
def main():
    data = input("Enter data bits: ")
    enc = encode(data)
    print("Encoded Hamming Code:", enc)
    detect(input("Enter received Hamming
     code: "))

if __name__ == "__main__":
    main()
```