1) Problem 11.2 on the book.

**Solution**

![Diagram of READ, ADDRESS, DATA, COMPLETE signals]

2) Given four registers (R1, R2, R3, and R4), and an additional signal A, you are required to design the interconnection between them so that the resulting datapath can support the operations described in the following cases:

   a) transfer the value of any register or A into any register, and being able to do several transfers at the same time.

   b) transfer the value of any register or A into any register, but only one transfer at a time.

   c) divide the registers into two groups: group 1 contains R1 and R2, and group 2 contains R3 and R4; you should be able to transfer one register of one group or A into one or both registers of the other group, and being able to do two inter-group transfers at the same time.

**Solution**

   a) Point-to-point connection:

   ![Diagram of point-to-point connection]

   b) Bus:

   ![Diagram of bus connection]

   c) Two buses:

   ![Diagram of two bus connection]
3) Given the datapath shown in section 9, page 1-20 of the lecture notes, describe in register transfer notation the execution of the following instructions:
   a) \( rt = 55h \) (Hint: the constant 55h is located in the memory right after the opcode)
   b) \( rt = \text{mem}(10h) \) (Hint: the constant 10h is located in the memory right after the opcode)

Solution

a) Instruction Fetch:
   \[ \text{mabus} \leftarrow \text{PC} \ (\text{PCmaEN}) \]
   \[ \text{memread} \ (\text{mr}) \]
   \[ \text{IR} \leftarrow \text{memory} \ (\text{IRld}) \]
   \[ \text{op} \leftarrow \text{add} \ (\text{srcA}, \text{srcB1}, \text{op}) \]
   \[ \text{PC} \leftarrow \text{ALUout} \ (\text{PCld}) \]

Instruction Decode:
   IR to controller

Instruction Execution:
   \[ \text{mabus} \leftarrow \text{PC} \ (\text{PCmaEN}) \]
   \[ \text{memread} \ (\text{mr}) \]
   \[ \text{MBR} \leftarrow \text{memory} \ (\text{MBRld}) \]
   \[ \text{op} \leftarrow \text{add} \ (\text{srcA}, \text{srcB1}, \text{op}) \]
   \[ \text{PC} \leftarrow \text{ALUout} \ (\text{PCld}) \]
   \[ \text{rt} \leftarrow \text{MBR} \ (\text{regWrite, wrDataSel, wrRegSel}) \]

b) Instruction Fetch:
   (as in the previous part)

Instruction Decode:
   (as in the previous part)

Instruction Execution:
   Load the constant:
   \[ \text{mabus} \leftarrow \text{PC} \ (\text{PCmaEN}) \]
   \[ \text{memread} \ (\text{mr}) \]
   \[ \text{MBR} \leftarrow \text{memory} \ (\text{MBRld}) \]
   \[ \text{op} \leftarrow \text{add} \ (\text{srcA}, \text{srcB1}, \text{op}) \]
   \[ \text{PC} \leftarrow \text{ALUout} \ (\text{PCld}) \]

Send the constant to the address bus and read the memory contents:
   This operation is not possible with the given datapath, because there is no connection between the MBR output and the memory address bus; the only possibility is to pass the value from the MBR to the register bank, and from there thru the ALU to the memory address bus. Since the register \( rt \) will be overwritten with the new value, we can use it to store temporarily the constant.
   \[ \text{rt} \leftarrow \text{MBR}(\text{regWrite, wrDataSel, wrRegSel}) \]
   \[ \text{op} \leftarrow \text{pass} \ (\text{srcA}, \text{op}) \]
   \[ \text{mabus} \leftarrow \text{ALUout} \ (\text{ALUmaEN}) \]
   \[ \text{memread} \ (\text{mr}) \]
   \[ \text{MBR} \leftarrow \text{memory} \ (\text{MBRld}) \]
   \[ \text{rt} \leftarrow \text{MBR} \ (\text{regWrite, wrDataSel, wrRegSel}) \]
4) The figure shows the datapath of a simple Digital Signal Processor. The MAC unit performs the multiply-accumulate operation \((A+B\times C)\). The registers have a load control signal (PCld, IRld, Ald, Bld, Cld, MARld) and those with two inputs have a select signal (PCsel, Asel, MARsel and MBRsel). Additionally, MBR can enable a buffer to let the data flow from the memory to the registers by using the signal MBRen.

a) Describe in register transfer notation how would you implement the instruction \(A=n\), where \(n\) is a constant value. (Include the opcode fetch).

b) Which of the following instructions you can't implement with this datapath?
   i) \(B=\text{mem}(n)\)
   ii) \(A=A+B\times C\)
   iii) \(B=C\)
   iv) jump \(n\)
   v) \(A=B+C\)
   vi) jump \(A\)

c) Give a sequence of valid instructions to perform \(Y=X^2\), where \(X\) and \(Y\) are variables located in \(\text{mem}(10h)\) and \(\text{mem}(11h)\) respectively.

**Solution**

a) Fetch 1: \(\text{PC}\rightarrow\text{MAR} (\text{MARsel, MARld})\)
   \(\text{PC}+1\rightarrow\text{PC} (\text{PCsel, PCld})\)

   Fetch 2: \(\text{mem}\rightarrow\text{IR} (\text{MBRen, IRld, memread})\)

   Decode: IR goes to controller

   Execute 1: \(\text{PC}\rightarrow\text{MAR} (\text{MARsel, MARld})\)
   \(\text{PC}+1\rightarrow\text{PC} (\text{PCsel, PCld})\)

   Execute 2: \(\text{mem}\rightarrow\text{A} (\text{MBRen, Ald, memread})\)

b) i) yes
   ii) yes
   iii) no
   iv) yes
   v) no
   vi) no

c) \(A=0\)
   \(B=\text{mem}(10h)\)
   \(C=\text{mem}(10h)\)
   \(A=A+B\times C\)
   \(\text{mem}(11h)=A\)

5) Given the state diagram of the figure, design an implementation of the FSM using the jump counter method. Your design should be complete, including state assignment, minimized Boolean expressions for Clear, Load and Count, jump state logic (if you use a ROM, give the contents) and schematic.
Solution

State assignment:

<table>
<thead>
<tr>
<th>Q3..Q0 state</th>
<th>Q3..Q0 state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>A</td>
</tr>
<tr>
<td>0001</td>
<td>B</td>
</tr>
<tr>
<td>0010</td>
<td>C</td>
</tr>
<tr>
<td>0011</td>
<td>D</td>
</tr>
<tr>
<td>0100</td>
<td>E</td>
</tr>
<tr>
<td>0101</td>
<td>H</td>
</tr>
<tr>
<td>0110</td>
<td>K</td>
</tr>
<tr>
<td>0111</td>
<td>M</td>
</tr>
</tbody>
</table>

\[
\text{CNT} = A + B \cdot \text{WAIT} + C \cdot \text{WAIT}' + E + H + K + M + F + G + J \cdot \text{WAIT}'
\]

\[
\text{LD} = D + I
\]

\[
\text{CLR} = N + L
\]
Using K-maps:

\[
\begin{align*}
\text{CNT} & = \text{WAIT'.Q3'.Q0'}+\text{WAIT'.Q2.Q0'}+\text{WAIT.Q3'.Q1'+Q3'.Q2+Q3.Q2'.Q0} \\
\text{LD} & = Q3'.Q2'.Q1.Q0'+Q3.Q1.Q0' \\
\text{CLR} & = Q3.Q2'.Q1'.Q0'+Q3.Q2.Q0
\end{align*}
\]

For the jump ROM, we need the inputs OP1 and OP0, and we need to differentiate D (0011) from I (1010), so we select Q3 also as an input. Therefore the contents has to be:

<table>
<thead>
<tr>
<th>address (Q3,OP1,OP0)</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0100</td>
</tr>
<tr>
<td>001</td>
<td>1001</td>
</tr>
<tr>
<td>010</td>
<td>1011</td>
</tr>
<tr>
<td>011</td>
<td>1011</td>
</tr>
<tr>
<td>100</td>
<td>0111</td>
</tr>
<tr>
<td>101</td>
<td>0111</td>
</tr>
<tr>
<td>110</td>
<td>0111</td>
</tr>
<tr>
<td>111</td>
<td>0111</td>
</tr>
</tbody>
</table>

Finally, the schematic is:

6) Using the datapath of Problem 4, identify all the microoperations. Write the opcode fetch implementation using a horizontal microcode format.

**Solution**

The microoperations are:

- PClD (active high)
- PCsel ('0'=PC+1, '1'=IR)
- IRld (active high)
- Ald (active high)
- Asel ('0'=A+B*C, '1'=MBR)
- Bld (active high)
- ClD (active high)
- MARld (active high)
- MARsel ('0'=PC, '1'=IR)
- MBRld (active high)
- MBRsel ('0'=A, '1'=memory)
- MBRen (active high)
- memread (active high)
- memwrite (active high)

The opcode fetch (from problem 4 solution) has two states and can be coded as:
7) A processor has 3 general purpose registers (A,B,C) and is able to perform the following operations: load/store value from/to memory to/from any register, add A+B and store result in any register, subtract A-B and store result in any register, shift right/left any register, jump to any memory location unconditionally and conditionally on zero or sign of the result.

a) Draw a datapath for this processor (you can use an ALU and a Shifter).

b) Select a coding for the instructions, grouping the bits in a convenient way.

c) List the microoperations implied by the datapath.

d) Write a sequence of microoperations to perform a conditional jump on the sign of the result from the ALU. The jump instruction mnemonic is JP N,label and means: if the sign is negative, jump to the address 'label'.

**Solution**

a)

![Datapath Diagram](image)

b) We use 2 bits to code the type of instruction (data transfer, ALU, Shifter or jump), 2 more bits to distinguish which instruction it is, 2 bits to select a register (if applicable) and finally n bits to select the address or second register (if applicable).

<table>
<thead>
<tr>
<th>instruction</th>
<th>type (2)</th>
<th>subtype (2)</th>
<th>register (2)</th>
<th>address (n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld r,(mem)</td>
<td>00</td>
<td>00</td>
<td>rr</td>
<td>aaa...a</td>
</tr>
<tr>
<td>ld (mem),r</td>
<td>00</td>
<td>01</td>
<td>rr</td>
<td>aaa...a</td>
</tr>
<tr>
<td>ld r1,r2</td>
<td>00</td>
<td>10</td>
<td>r1r1</td>
<td>r2r2x...x</td>
</tr>
<tr>
<td>r=A+B</td>
<td>01</td>
<td>00</td>
<td>rr</td>
<td>xxx...x</td>
</tr>
<tr>
<td>r=A-B</td>
<td>01</td>
<td>01</td>
<td>rr</td>
<td>xxx...x</td>
</tr>
<tr>
<td>sr r</td>
<td>10</td>
<td>00</td>
<td>rr</td>
<td>xxx...x</td>
</tr>
<tr>
<td>sl r</td>
<td>10</td>
<td>01</td>
<td>rr</td>
<td>xxx...x</td>
</tr>
<tr>
<td>jp address</td>
<td>11</td>
<td>00</td>
<td>xx</td>
<td>aaa...a</td>
</tr>
<tr>
<td>jp z,address</td>
<td>11</td>
<td>01</td>
<td>xx</td>
<td>aaa...a</td>
</tr>
<tr>
<td>jp n,address</td>
<td>11</td>
<td>10</td>
<td>xx</td>
<td>aaa...a</td>
</tr>
</tbody>
</table>
c) A\rightarrow IBus (Aen) 
IBus\rightarrow A (Ald) 
B\rightarrow IBus (Ben) 
IBus\rightarrow B (Bld) 
C\rightarrow IBus (Cen) 
IBus\rightarrow C (Cld) 
Shifter\rightarrow IBus (Sen) 
IBus\rightarrow Shifter (Sld, ShiftR/L)
A+B\rightarrow IBus (Asel=A, ALUen, ALUop=\text{sum}) 
A-B\rightarrow IBus (Asel=A, ALUen, ALUop=\text{sub}) 
PC+1\rightarrow PC (Asel=PC, ALUop=inc, PCsel=ALU, PCld)
mem\rightarrow IBus (MBRrd) 
IBus\rightarrow mem (MBRwr) 
IBus\rightarrow IR (IRld) 
IR\rightarrow PC (PCsel=IR, PCld) 
PC\rightarrow memadd (MARsel=PC, MARld) 
IR\rightarrow memadd (MARsel=IR, MARld)

d) Instruction fetch:
1) PC\rightarrow memadd 
   PC+1\rightarrow PC 
2) mem\rightarrow IBus
   IBus\rightarrow IR
Instruction decode:
3) IR goes to the controller 
   controller looks at ALUsign 
   if ALUsign=0, go back to 1) 
   if ALUsign=1, go to 4)
Instruction execution:
4) IR\rightarrow PC