Chapter 4 example problems:

1. Assume you have a 1Gx32 memory built out of 32Mx16 chips. Answer the following questions. Assume word-level addressing.
   a. How many chips will it take to build this memory?
   b. How many chips are needed to make up one bank?
   c. How many banks will there be?
   d. How many bits will it take to make up an address (e.g., the number of inputs to the memory decoder)?
   e. Of the bits for part d, if you have low-order interleave, which of the bits go to the bank select and which of the bits make up the address on the chip?

Answers:
   a. $1G \times 32 / 32M \times 16 = 32 \times 2 = 64$ chips
   b. $32$ bit word size / $16$ bits per chip = $2$ chips per bank
   c. $64$ chips / $2$ chips per bank = $32$ banks
   d. $1G$ of words, $1G = 2^{30}$, so we need $\log 2^{30} = 30$ bits per address
   e. With low-order interleave, the high order bits (leftmost) are the address on the chip and the low order bits (rightmost) are the bank select. There are $32$ banks which require $5$ bits, so we have the leftmost $25$ bits are the address to the chip and the rightmost $5$ bits are the bank.

For 2-3, provide the RTN for the new instruction.

2. New instruction: Double X. This instruction fetches the datum at location X, doubles it (adds it together), and saves it back to X. Provide the RTN.

Answer:

Fetch:

<table>
<thead>
<tr>
<th>MAR $\leftarrow$ PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBR $\leftarrow$ M[MAR]</td>
</tr>
<tr>
<td>IR $\leftarrow$ MBR // authors combine these to IR $\leftarrow$ M[MAR]</td>
</tr>
<tr>
<td>PC $\leftarrow$ PC + 1</td>
</tr>
</tbody>
</table>

Decode:

<table>
<thead>
<tr>
<th>MAR $\leftarrow$ IR[11-0] // move memory location X to MAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decode IR[15-12]</td>
</tr>
</tbody>
</table>

Get op:

<table>
<thead>
<tr>
<th>MBR $\leftarrow$ M[MAR] // X loaded into MBR</th>
</tr>
</thead>
</table>

Execute:

<table>
<thead>
<tr>
<th>AC $\leftarrow$ MBR // needed to have X in the adder</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC $\leftarrow$ AC + MBR // AC $\leftarrow$ X + X</td>
</tr>
<tr>
<td>MBR $\leftarrow$ AC // moves 2 * X to MBR to write to memory</td>
</tr>
<tr>
<td>M[MAR] $\leftarrow$ MBR // addr X is still in MAR, write 2*X to mem</td>
</tr>
</tbody>
</table>
3. New instruction: JZ X. This instruction jumps to memory location X if the AC is 0. This allows us to have an easier instruction than the skipcond. It requires doing PC ← X if AC = 0. Provide the RTN.

Answer:

Fetch: MAR ← PC
MBR ← M[MAR]
IR ← MBR // authors combine these to IR ← M[MAR]
PC ← PC + 1

Decode: Decode IR[15-12]
Get op: not needed
Execute: If AC = 0 then PC ← IR[11-0]

NOTE: we could also include MAR ← IR[11-0] in the decode stage and then change execute to be If AC = 0 then PC ← MAR

4. Imagine that we want to add more registers to MARIE, we will call them R1, R2, R3, R4. What impact will this have on using MARIE?

The primary impact is that we now need to address which register we want to use in any instruction. For instance, Load X would now become Load R1, X. Since there are 4 registers, this adds 2 bits to our instruction. We either need to have 18 bit instructions, or we would have to reduce our addresses by 2 bits. If we go with the former, it means that the IR, data bus, and every memory location will now require 18 bits instead of 16 (thus, we change the computer to an 18 bit word size). With these increases in data sizes, our data will become 18 bits requiring that we increase the AC to 18 bits and make the ALU into an 18-bit ALU. If we go with the latter and decrease the size of the operand address, then we only have 10 bits for an address limiting us to 1024 bytes instead of 4096. With this decrease, we should also decrease the size of the address bus to 10 bits and make the MAR and PC 10 bits.

The second impact is that we have to add the register to instructions that previously had no registers: clear, input, output, skipcond. Before, we always used the value in the AC, but now we have to specify which of the 4 registers we want to clear, output, input into, or test for skipcond.

The final impact is that things become a bit easier to implement. We can now have multiple data in the CPU at any one time rather than having to always move things into and out of memory.
For 5-6, provide MARIE code for the given C code. You may either assume a value is stored in a variable of the same name (e.g., 1 is stored in one, 10 is stored in 10) or you may use #1, #10 to denote the datum.

5.

```c
scanf("%d %d", &x, &y);
if(x > y) z = x;
else z = y;
printf("%d", z);
```

Answer:

```
Input
Store X
Input
Store Y
Load X
Subt Y
Skipcond 10
Jump Else
Load X
Store Z
Jump Next
Else:
    Load Y
    Store Z
Next:
Output
```

6. Write MARIE code to compute z = x / y. Assume x and y are already loaded into memory. Also assume that it’s ok to alter X and Y, we do not need them again.

Answer:

```
Clear
Store Z       // initialize Z to 0
Load X
Subt Y
Top: Skipcond 10  // while X – Y > 0 continue
      Jump Next
    Store X     // replace X with X – Y
    Load Z     // another successful subtraction, increment Z
    Add #1
    Store Z
    Load X     // repeat X – Y condition for top of loop comparison
    Subt Y
    Jump Top
Next: ...     // Z has had 1 added to it for each successful subtraction
```
7. Let’s consider an older CPU with few pins. The 8086 for instance would reuse pins depending on the clock cycle. Our fictitious CPU would work like this. During the first cycle, pins would be used for addressing. During the second cycle, pins would be used for commands. During the third and fourth cycles, pins would be used for data transfer and interrupts. The machine uses a special device called an interrupt controller so all interrupt signals by devices are sent to the interrupt controller and the interrupt controller sends a single bit interrupt to the CPU so we only need 1 pin for the interrupt signal. Assume a 64K addressable memory, 16 bit data divided into 2 8-bit transfers, 18 total control commands and the 1 interrupt signal.

   a. How many pins would this CPU need?
   b. If this CPU permitted all information to be transmitted in the same cycle, how does your answer change?

Answer:

   a. We need to determine the maximum number of pins. For addressing in the first cycle, there will be 64K sized addresses which requires log 64K = 16 bits so 16 pins. In the second cycle, we have 18 commands, so 18 pins. For the third cycle, we have 8 bits of data so 8 pins. For the fourth cycle, we have 8 bits of data plus a possible interrupt, so 9 bits. We would need 18 pins on the computer.

   b. If all information was transmitted in 1 cycle, we would have to combine all of these. We would need 16 pins for the address portion of the bus, 16 pins for the data portion of the bus and 19 pins for the control portion of the bus for a total of 51 pins.

8. Hand-compile the MARIE multiplication code (page 253) into hexadecimal and binary.

Answer:

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Instruction</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>If</td>
<td>Load X</td>
<td>110C</td>
<td>0001000100001100</td>
</tr>
<tr>
<td>101</td>
<td></td>
<td>Subt Y</td>
<td>410D</td>
<td>0100000100001101</td>
</tr>
<tr>
<td>102</td>
<td></td>
<td>Skipcond 400</td>
<td>8400</td>
<td>1000010000000000</td>
</tr>
<tr>
<td>103</td>
<td></td>
<td>Jump Else</td>
<td>9108</td>
<td>1001000100001000</td>
</tr>
<tr>
<td>104</td>
<td>Then</td>
<td>Load X</td>
<td>110C</td>
<td>0001000100001100</td>
</tr>
<tr>
<td>105</td>
<td></td>
<td>Add X</td>
<td>310C</td>
<td>0011000100001100</td>
</tr>
<tr>
<td>106</td>
<td></td>
<td>Store X</td>
<td>210C</td>
<td>0010000100001100</td>
</tr>
<tr>
<td>107</td>
<td></td>
<td>Jump Endif</td>
<td>910B</td>
<td>1001000100001011</td>
</tr>
<tr>
<td>108</td>
<td>Else</td>
<td>Load Y</td>
<td>110D</td>
<td>0001000100001101</td>
</tr>
<tr>
<td>109</td>
<td></td>
<td>Subt X</td>
<td>410C</td>
<td>0100000100001100</td>
</tr>
<tr>
<td>10A</td>
<td></td>
<td>Store Y</td>
<td>210D</td>
<td>0010000100001101</td>
</tr>
<tr>
<td>10B</td>
<td>Endif</td>
<td>Halt</td>
<td>7000</td>
<td>0111000000000000</td>
</tr>
</tbody>
</table>