CSCI-GA.1144-001
PAC II
Midterm Exam Spring 2014
(120 minutes)

Last name: First name:

Notes:
• If you perceive any ambiguity in any of the questions, state your assumptions clearly
• Questions vary in difficulty; it is strongly recommended that you do not spend too much time on any one question.
• This exam is open book/notes but no electronic devices.

1. [6 points] Circle the correct answer among the choices given. If you circle more than one answer, you will lose the grade of the corresponding question.

(A) As the technology advances, which of the following types of algorithms can become feasible to be used?
   a. Undecidable   b. Unsolvable   c. Intractable   d. All of them

(B) The following MIPS instruction
   \texttt{bne r1,r2,loop} (that is: branch to loop if r1 \neq r2) is:
   a. encoded as \texttt{R} format instruction   b. encoded as \texttt{I} format instruction
   (c. encoded as \texttt{J} format instruction   d. none of the above

(C) The executable is generated by:
   a. compiler   b. assembler   c. linker   d. loader   e. the programmer

(D) The ALU is a:
   a. sequential circuit   b. combinational circuit
   c. we can build it either ways   e. both sequential and combinational

(E) The size of MIR (Micro Instruction Register) of the control unit depends on:
   a. IR   b. PC   c. the data path   d. The memory system

(F) When a MIPS \textit{jump} to a backward instruction is executed:
   a. A value is added to the PC.
   b. A value is subtracted from the PC
   (c) A value is loaded into the PC
   d. The PC is only incremented by 4
   e. None of the above
2. [2 points] Can you build a device that, logically, behaves like an OR gate from only AND and NOT gates? If so, do so (just for the case where AND and OR gates have only two inputs). If not, explain why not.

Yes, we can do that because:

\[ a \text{ OR } b = \text{NOT}( \text{NOT}(a) \text{ AND } \text{NOT}(b) ) \] [An application of De Morgan’s law]

The following table summarizes it.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>(\text{NOT}(a))</th>
<th>(\text{NOT}(b))</th>
<th>(\text{NOT}(a) \text{ AND } \text{NOT}(b))</th>
<th>(\text{NOT}(\text{previous column}) = a \text{ OR } b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

3. For the following logic circuit:

![Logic Circuit Diagram]

a. [4 points] Draw the truth table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>c</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

b. [1 point] Is this function sequential or combinational? Why?

This is a combinational circuit because there are not storage elements (flip flops) and therefore, the output depends only on the input and not on a previous state.
4. For the datapath shown below (assume the register file can submit the contents of two registers at the same time):

   a. [1 point] Can we get rid of AOR? Justify

No, we cannot. Because if we do, the output from ALU will go directly to memory and MDR which may cause wrong results or may get lost.

   b. [2 points] Write 5 microinstructions that can be executed in parallel

Possible solution:
R1-> Rrt  R2->Rrs  AOR-> MDR  SE(imm)->offset  PC+4-> PC

   c. [4 points] Suppose we invented the following instruction: stw r1, offset(r2)
   
   This instruction does the following:
   - Copy the value in register r1 to memory location 
   \{ SE(imm) + register r2 \} (that is: M[SE(imm)+ r2] = r1)
   - Decrement the register r2 by the value of r1 (that is, r2 = r2 – r1)

   Write the microinstructions needed to execute the above instruction (no need to write fetch phase). Optimize as much as you can.

   SE(imm) \rightarrow offset  r2 \rightarrow Rrs  r1 \rightarrow Rrt
   offset + Rrs \rightarrow AOR
   Rrt \rightarrow M[AOR]
   Rrs - Rrt \rightarrow AOR
   AOR \rightarrow MDR
   MDR \rightarrow r2