

Massachusetts Institute of Technology  
Department of Electrical Engineering and Computer Science

6.111 - Introductory Digital Systems Laboratory

**Problem Set 5**

**Issued:** March 10, 1999

**Due:** March 19, 1999

Another long night at the lab. Tired and frustrated, you've been working on your final project nonstop since about noon. You've been on the help queue for four hours now, but when the TA comes over, somehow you can't understand what he's telling you to do. He's waving his arms a little and poking at your kit, but the words he's saying don't make any sense. Exasperated, you mumble a thanks, and he wanders off to help someone else; your project still works about as well as it ever has, that is, not at all, and you've only got three days to get the whole thing working. As a sigh rises from the depths of your weary soul, the world slowly begins to slip away. Your eyes lose their focus and your mind floats free...

Mmmm, pizza.

If only you had a big, fresh, tasty pizza. Then things wouldn't seem so hopeless. Food trucks? No, its been dark for hours. Bertucci's? No, already closed. Even the late-night delivery places have packed it up for the night. Where will you find a pizza at this hour of the morning?

As you slowly come back to yourself, it occurs to you that you've been staring for the past ten minutes or so at the big, oddly-shaped, heavy-looking device way over in the corner of the lab. Funny, you think, you've never noticed that before. You walk over to have a closer look.

The unit appear to have a section which is refrigerated, a complicated transportation mechanism, and a section which looks like it is capable of high temperatures. You open the refrigerated section to find all of the ingredients for pizza: dough, sauce, cheese and a bunch of toppings. You surmise that the transportation mechanism is intended to move ingredients from the refrigerated section into the oven section where the pizza can be cooked.

Intrigued, you examine the front of the unit and find several jumpers. The first is labeled **Load Dough, Sauce, and Cheese**, the second is labeled **Load Pepperoni**, the third is labeled **Load Mushrooms**, and the fourth is labeled **Cook**. One LED is labeled **Cooking**. There are other jumpers, but you have no idea what they are for. Since you don't find any controls for the temperature, you assume the oven section maintains a constant temperature appropriate for baking pizzas until they are done.

The only thing missing from this machine is a control mechanism, the Pizza Control Unit (PCU). You list the inputs and outputs needed (Tables 1 and 2), figuring that you can complete the Pizza Machine and have the best 6.111 project ever! (Or at least the most popular...)

Table 1: Inputs to PCU

<u>Input</u>	<u>Description</u>
GOSYNC	The synchronized go signal, indicating that the user desires pizza.
PEP	Indicates that the user wants pepperoni on the pizza.
MUSH	Indicates that the user wants mushrooms on the pizza.

Table 2: Outputs from the PCU

<u>Output</u>	<u>Description</u>
LDPEP	Load pepperoni on a pizza already in the oven.
LDMUSH	Load mushrooms on a pizza already in the oven.
COOK	Start the Pizza Machine cooking.
LDPIZZA	Load an unbaked crust, sauce, and cheese into the oven.

You envision the Pizza Machine working as follows:

- The user sets switches indicating whether he wants pepperoni, mushrooms, both, or neither
- The user pushed the Go button (which causes GOSYNC to be asserted).
- The PCU loads an unbaked crust, adds whatever ingredients are necessary, and cooks the pizza.
- The user knows not to hit the Go button or change the topping switches while the pizza is cooking.
- When the **Cooking** light goes out, the user know to remove the pizza.

### Problem 1:

**Draw a flowchart showing the steps the PCU needs to follow.**

You decide that the PCU should have two instructions in its instruction set: assertions and conditional branches. This is illustrated in Table 3.

Table 3: PCU Instruction Set

Instruction	I7	I6	I5	I4	I3	I2	I1	I0
If <i>Cond</i> CJUMP <i>Address</i>	0	C2	C1	C0	A3	A2	A1	A0
ASSERT <i>Signals</i>	1	S6	S5	S4	S3	S2	S1	S0

### Problem 2:

**Think about how many conditions must be checked (don't forget, you always need a condition which is always true), and assign each to a setting of C2, C1 and C0. Likewise, choose positions in the instruction word for each of the output signals. Turn in your assignments for the input and output signals.**

As you begin to think about your design, you remember that you fried several of the chips that came with your kit, but you never bothered to get them replaced. Now the instrument desk is closed, so you'll have to work with what you've got.

### Problem 3:

**Design the PCU using only a 74LS163, an EPROM, a 74LS151, a 20V8 PAL (for your assert logic), and some wire. Signal need only be asserted for one clock period (you already have working system clock), and the GOSYNC signal is already synchronized. Turn in a properly drawn schematic diagram (all inputs and outputs labeled, but no pin numbers) and a listing of the VHDL file you would use to program the 20V8. Be careful not to allow any glitches on the signals you assert.**

You now need to create a file which is used by the microcode assembler to interpret the microcode that you will write. This file, `pizza.sp`, is started for you below.

```
/* pizza.sp, the spec file for pizza.as*/
/* define opcode and address fields */
op<7:0>
-add more code here-

/* define opcodes */
CJUMP nop;
-add more code here-

/* define condition codes */
-add more code here-

/*define assertions*/
-add more code here-
```

**Problem 4:**

Complete the `pizza.sp` file, add appropriate comments, and turn in a hard copy.

You must now write the microcode. The file `pizza.as` is started for you below:

```

/* pizza.as */
#SPEC_FILE = pizza.sp; /*where to find the spec file*/
#LIST_FILE = pizza.lst; /*where to send listing*/
#SET_ADDRESS = 0; /*address to start assembling*/
#LOAD_ADDRESS = 0; /*where to load code in the PROM*/

START:
-insert code here-

```

**Problem 5:**

Complete the `pizza.as` file, add appropriate comments, and turn in a hard copy.

You wire everything, program the chips, turn it on, and order a pepperoni pizza. The Pizza Machine works perfectly; at least when the Cooking light turns off, the pizza you take out of the oven is just about the best you've ever eaten.

Satisfied for the moment, you begin playing with the unlabelled jumps on the Pizza Machine. After a couple of minutes of hacking around, you decide the jumpers should be labeled **Load Sausage**, **Load Olives**, and **Load Onions**. You decide to modify your PCU to accept choices of sausage, olives, and onions, as well as the pepperoni and mushrooms from before. The inputs and outputs of the new PCU are listed in Tables 4 and 5.

Table 4: Inputs to the New PCU

Input	Description
GOSYNC	The synchronized go signal, indicating that the user desires a pizza.
PEP	Indicates that the user wants pepperoni on the pizza.
MUSH	Indicates that the user wants mushrooms on the pizza.
SAUS	Indicates that the user wants onions on the pizza.
OLIVE	Indicates that the user wants olives on the pizza.

Table 5: Outputs from the New PCU

Output	Description
LDPEP	Load pepperoni on a pizza already in the oven.
LDMUSH	Load mushrooms on a pizza already in the oven.
COOK	Start the Pizza Machine cooking.
LDPIZZA	Load a crust, sauce, and cheese into the oven.
LDSAUS	Load sausage on a pizza already in the oven
LDONION	Load onions on a pizza already in the oven..
LDOLIVE	Load olives on a pizza already in the oven.

It seems to you that implementing the new PCU will require only a few small addition to `pizza.sp`. Furthermore, the microcode to handle the additional toppings will be quite easy to add to `pizza.as`. However, as you begin to write this microcode, it dawns on you that the single 74LS163 that you're using

as a sequencer can only address a maximum of 16 lines of code; adding the code to deal with the new toppings increases the length of your microprogram to more than sixteen instructions.

Before you fall into despair though, you have a brilliant idea! You decide to change the instruction set slightly. Your new instruction set will allow *conditional* assertions, as well as conditional branches. Of course, you also need to provide some way of unconditionally asserting and branching. The instruction set is as shown in Table 6.

Table 6: New PCU Instruction Set

Instruction	I7	I6	I5	I4	I3	I2	I1	I0
If <i>Cond</i> CJUMP <i>Address</i>	0	C2	C1	C0	A3	A2	A1	A0
CASSERT <i>Signals</i>	1	C2	C1	C0	S3	S2	S1	S0

You predict that the new CASSERT instruction will reduce the length of your code by quite a bit, so it fits within your limit of 16 instructions. However, making the assertions conditional reduces the number of signals which can be asserted to only four; you will have to be clever with the assert logic in your 20v8.

**Problem 6:**

**Design a new PCU using only a 74LS163, an EPROM, a 74LS151, a 20v8 PAL diagram (all inputs and outputs labeled, but no pin number) and a VHDL file you would use to program the PAL. Once again, be careful not to allow any glitches on the signals you assert.**

**Problem 7:**

**Create and turn in a new pizza.sp file appropriate for the PCU you designed above. Make sure you account for all new inputs, outputs and instructions.**

**Problem 8:**

**Write the microcode for your new PCU. Create and turn in a new pizza.as file. Use conditional assertions whenever it makes your code shorter.**

By this point, the other students in the lab have really started to notice that you've been busy with the Pizza Machine. The smell of baking pepperoni, sausage, and onion pizza fills the lab, and a crowd begins to gather. As the sun comes up over Cambridge, you and your classmates breakfast on some of the best pizza to be had anywhere.