EE213 – Digital Systems Design

Homework 1
GNOME 2 and address decoding

NOTE: Just as on tests, I expect you to show all your work on homework problems. A correct answer with no work might not be given full credit.

1. Ch1, question 1
2. Ch1, question 4. List at least 4 features that may vary between family members. (Appendix A might be helpful).
3. For the following circuit:

   a) What is the address space of RAM1 and ROM1?

   b) List at least 3 locations that we could address REG1.

4. Say you are trying to hook up RAM (each chip with a standard $\overline{CS}$) to a microcontroller with 16 address lines (A0 through A15).

   a) Draw a circuit which uses just 2 74LS138s (and no other parts) to create an address space for 16 4kx8 RAM chips. (Hint: There is a good reason for having multiple chip selects on the 74LS138).

   b) Extending this model, how would you hook up 74LS138s to create separate address spaces for 8 4kx8 and 16 2kx8 RAM chips (you need only label the pins of the 74LS138s, showing what they would be connected to).

   c) For part b), find the address space for each chip.

5. **a)** Write an ASM program for the GNOME which performs the same function as the code:

```
temp = 0
for i = 0 to 5
    temp = temp + i
next i
end
```

Your ASM program should also count <u>up</u>.

b) Give the machine code for the first three and last three instructions (Assuming the first instruction is at code address 0).

c) Prove your code works by showing a table of values in appropriate registers and memory locations as the program executes (like we did in class for program 3*5).

d) For 2 extra homework points, write an ASM program which uses "for i=0 to 10" instead of "for i=0 to 5". (You do not have to do this part. It must be correct if you want the extra credit).

6. Draw a timing diagram over a single instruction cycle showing each of the control signals in the GNOME, when the GNOME executes the instruction: sta Rn, which is located at memory address 42H. The control signals that should appear on your plot are: write, OE, WR, load_IR_all, load_IR_bottom, rd_data, load_PC, jump_pc, load_ACC, alu_op (assume alu_op = 0 means do nothing), and clock. Please mark each state (fetch, decode, execute).

7. The 8051 has a command **AJMP** *addr*, where *addr* is an absolute address within the current "page". For instance, if PC = $PC_6PC_5 \cdots PC_0$ and addr= $A_3A_2A_1A_0$, then AJMP addr would change the PC to PC = $PC_6PC_5PC_4A_3A_2A_1A_0$. The page is set by $PC_6PC_5PC_4$ and the address within the page by $A_3A_2A_1A_0$. (Note that for the 8051, PC and addr both contain several more bits, but you get the basic idea). We would like to create a similar command for the GNOME.

a) Modify the original GNOME diagram with hardware which allows the bottom 4 bits of the PC to be loaded into the ACC.

b) Add hardware which allows the ACC to be loaded into the bottom 4 bits of the PC.

c) Create opcodes for the two instructions above (let's call them "lda_pc" and "ajmp_acc") that cannot be confused with any of the existing instruction opcodes (for instance, you cannot use the opcode 1000101 because the GNOME would interpret this as a jmp 5).