ECE152B: BURP Demonstration Program

You must execute the following program for the final demonstration of your BURP. The program is given in assembly and you must convert it to your machine code. Please speak with a TA if you have any questions, find any bugs or want/need to make any small modifications.

This program reads the value x set by the user from the input port. It produces a square wave with period proportional to 4*x on the output port. It is carefully written so that the duty cycle is 50%. Hook any bit of the output port to the oscilloscope or analyzer to check this! (The TAs will do this when you demo.) If the user changes the value on the input port, the program will notice and produce a new square wave with period proportional to 4*x, where x is the updated value read from the input port. This program is given in both C and assembly.

```
while(1) {
    int x,y; /* nibbles */
    int i,count; /* bytes */
    x = in();
    y = x;
    count = x*4;
    out(y);
    while(1) {
        for (int i=count; i<256; i++)
            ;
        if (x!=in()) break;
        y = ~y;
        out(y);
    }
}
```

```
Start:  in RC          ; read input value and store in register C
        or RC,RC       ; just to make sure you can do this :)
        *** initialize registers
        cc             ; clear carry
        mvi RA,0       ; clear registers RD and RB
        mov RA,RD      ; most significant nibble of count, count.ms = 0
        and RB,RA      ; should set RB to 0 since anything & 0 is 0
        mov RC,RA      ; least significant nibble of count, count.ls = input

        *** count = input * 4
        add RC,RA      ; RC = RC + RA (count*2)
        add RD,RB      ; RD = RD + 0 + carry
        add RC,RA      ; RC = RC + RA (count*3)
        add RD,RB      ; RD = RD + 0 + carry
        add RC,RA      ; RC = RC + RA (count*4)
        add RD,RB      ; RD = RD + 0 + carry
```

```
        *** initial output is same as input value
        out RA
        *** save items on stack
        push RC       ; count.ls
        push RD       ; count.ms
        push RA       ; last output
        push RA       ; initial input
        *** main program loop
        *** main delay loop, increment count until overflow (carry) occurs
main:   mvi RA,loop1.ls     ; loop1.ls address stored in RA
loop1:  inc RC        ; RC = RC + 1
        add RD,RB     ; RD = 0 + carry
        nop           ; increase delay with nop
        jc break1     ; break from loop if carry is produced
        jmp loop1     ; most significant nibble of loop1 address is immediate field
        *** check if the user changed the input value
break1: pop RD        ; restore initial input value
        mov RD,RC     ; save initial input value in RC
        in RA         ; read current input value
        sc            ; prepare to do equality check
        sub RD,RA     ; RD will be zero iff RA==RD
        mvi RA,1      ; prepare to do zero check
        sc
        sub RD,RA     ; carry (overflow) will be produced if NOT zero
        jc break2     ; need to recompute count value
        *** compute logical inverse of last output and make it the new output
        pop RA        ; restore last output value
        sub RB,RA     ; RB = 0 - RA (1's complement) or RB = ~RA (Carry is clear,
pop should not alter carry)
        out RB        ; output the value
        mov RC,RA     ; saved initial input to RA
        pop RD        ; restore computed count value
        pop RC
        *** Restore the stack for next iteration
        push RC       ; ls count nibble
        push RD       ; ms count nibble
        push RB       ; last output value
        push RA       ; initial input value
        *** Set RB to zero
        mvi RA,0
        mov RA,RB
        mvi RA,main.ls ; ls nibble of main address stored in RA
        jmp main      ; ms nibble of main address is immediate field
        *** Restart if user has changed input value
break2: pop RA        ; flush the stack
        pop RA
        pop RA
        mvi RA,start.ls ; ls nibble of start address stored in RA
        jmp start
```