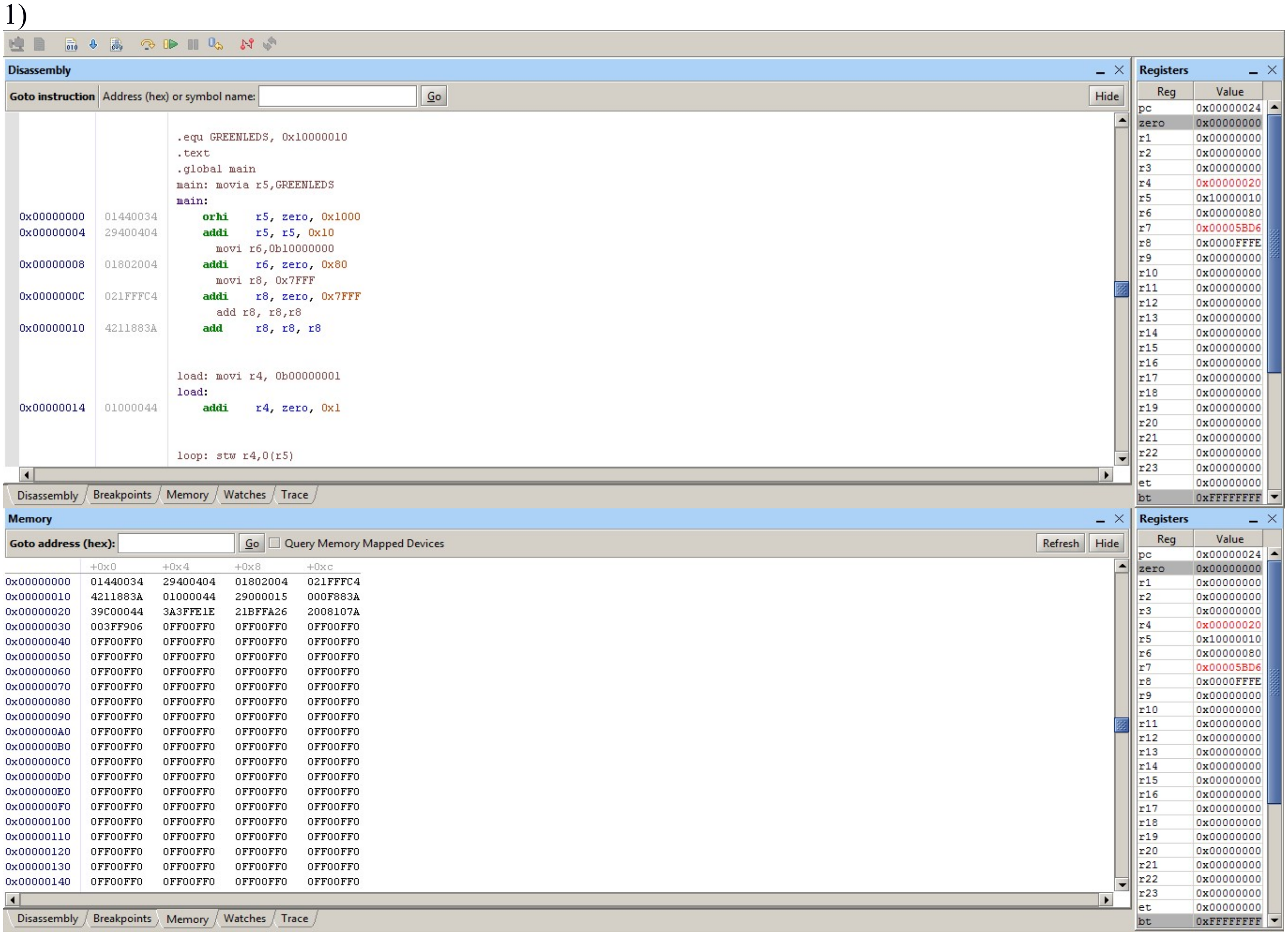
Old Dominion University
ECE 346
Salem Chemlal
Spring 2016
Lab Report 1
Bradley McKee (UIN: 00975338)

Introduction:
With this being the first lab of the semester, our main objective was to grasp some sort of understanding the basics of our DE0-NAN0 board. Since this is the first lab, I had to first install all of the programs and drivers according to the tutorial (lab 1). Next I followed through the set of instructions on how to set up my board with my pc. The main objective of the lab was to figure out and familiarize myself with the NAN0 board and Quartus 15.0.

Background:
The DE0-NAN0 board is a Field Programmable Gate Array (FPGA). For our board, the processor is a Nios II Processor, which gives us a register structure to follow, as well as an instruction set. An instruction set is how you manipulate the binary code for registers.

Preliminary:
The preliminary work for this lab was to basically follow the instructions until the end and then modify the code accordingly. I had to do a lot of personal research to try and figure out the delay until a good example was done in class. I messed around a lot with the LEDS to figure out the functionality and how it works.

Results:
1)



```
                                .equ GREENLEDS, 0x10000010
                                .text
                                .global main
                                main: movia r5,GREENLEDS
                                main:
0x00000000     01440034             orhi     r5, zero, 0x1000
0x00000004     29400404             addi     r5, r5, 0x10
                                      movi r6,0b10000000
0x00000008     01802004             addi     r6, zero, 0x80
                                      movi r8, 0x7FFF
0x0000000C     021FFFC4             addi     r8, zero, 0x7FFF
                                      add r8, r8,r8
0x00000010     4211883A             add      r8, r8, r8


                                load: movi r4, 0b00000001
                                load:
0x00000014     01000044             addi     r4, zero, 0x1


                                loop: stw r4,0(r5)
```

```
              +0x0      +0x4      +0x8      +0xc
0x00000000    01440034  29400404  01802004  021FFFC4
0x00000010    4211883A  01000044  29000015  000F883A
0x00000020    39C00044  3A3FFE1E  21BFFA26  2008107A
0x00000030    003FF906  0FF00FF0  0FF00FF0  0FF00FF0
0x00000040    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x00000050    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x00000060    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x00000070    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x00000080    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x00000090    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x000000A0    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x000000B0    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x000000C0    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x000000D0    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x000000E0    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x000000F0    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x00000100    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x00000110    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x00000120    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x00000130    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
0x00000140    0FF00FF0  0FF00FF0  0FF00FF0  0FF00FF0
```

Registers

| Reg | Value |
|---|---|
| pc | 0x00000024 |
| zero | 0x00000000 |
| r1 | 0x00000000 |
| r2 | 0x00000000 |
| r3 | 0x00000000 |
| r4 | 0x00000020 |
| r5 | 0x10000010 |
| r6 | 0x00000080 |
| r7 | 0x00005BD6 |
| r8 | 0x0000FFFE |
| r9 | 0x00000000 |
| r10 | 0x00000000 |
| r11 | 0x00000000 |
| r12 | 0x00000000 |
| r13 | 0x00000000 |
| r14 | 0x00000000 |
| r15 | 0x00000000 |
| r16 | 0x00000000 |
| r17 | 0x00000000 |
| r18 | 0x00000000 |
| r19 | 0x00000000 |
| r20 | 0x00000000 |
| r21 | 0x00000000 |
| r22 | 0x00000000 |
| r23 | 0x00000000 |
| et | 0x00000000 |
| bt | 0xFFFFFFFF |

Registers

| Reg | Value |
|---|---|
| pc | 0x00000024 |
| zero | 0x00000000 |
| r1 | 0x00000000 |
| r2 | 0x00000000 |
| r3 | 0x00000000 |
| r4 | 0x00000020 |
| r5 | 0x10000010 |
| r6 | 0x00000080 |
| r7 | 0x00005BD6 |
| r8 | 0x0000FFFE |
| r9 | 0x00000000 |
| r10 | 0x00000000 |
| r11 | 0x00000000 |
| r12 | 0x00000000 |
| r13 | 0x00000000 |
| r14 | 0x00000000 |
| r15 | 0x00000000 |
| r16 | 0x00000000 |
| r17 | 0x00000000 |
| r18 | 0x00000000 |
| r19 | 0x00000000 |
| r20 | 0x00000000 |
| r21 | 0x00000000 |
| r22 | 0x00000000 |
| r23 | 0x00000000 |
| et | 0x00000000 |
| bt | 0xFFFFFFFF |

2)

movia – Move Immediate Address (loads a 32-bit value that goes to a label.   Hi/Lo order.  This breaks up into orhi/ori)

movi – Move Immediate (sign entends the immediate 16 bits to 32 bit and loatds it into the new register.)

add – adds contents of two registers

stw – Store Word instruction.  addresses it by an offset value and store it's contents into the new register

addi – add immediate (adds content of a register then sign-extends 16 bits.  Same for signed/unsigned values.

bne – Branch if not equal (A !== B) (Boolean function)

roli – rotate left immediate.  Rotates the bits of a register from right to left by the # of bits given (IMMED5) then stores it into the new register

br – samething as a jump instruction, but executes unconditionally to the address it has..  Addresses the label as well.

Functions that change (pseudo code changes):

movia – orhi/addi

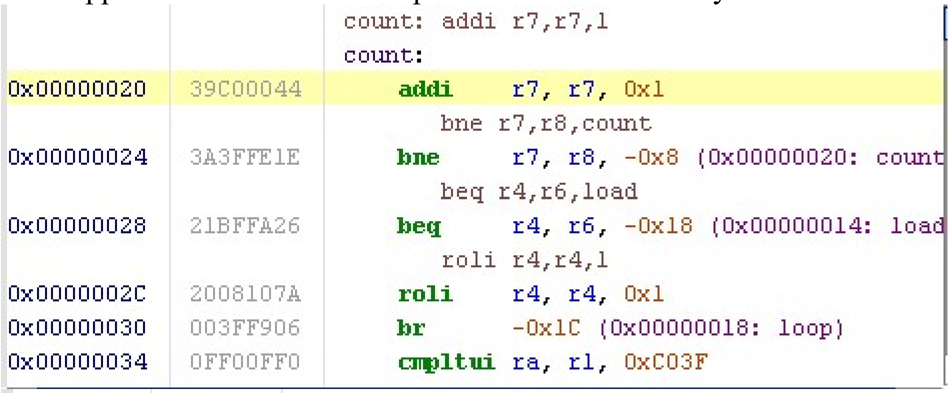
movi – addi

add r7,zero,zero = mov r7,r0

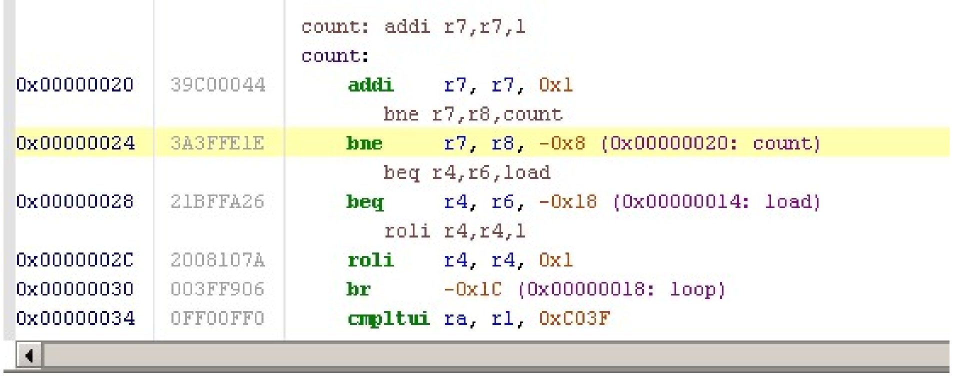
bne changed to show the address of the labels when broken down.

3)

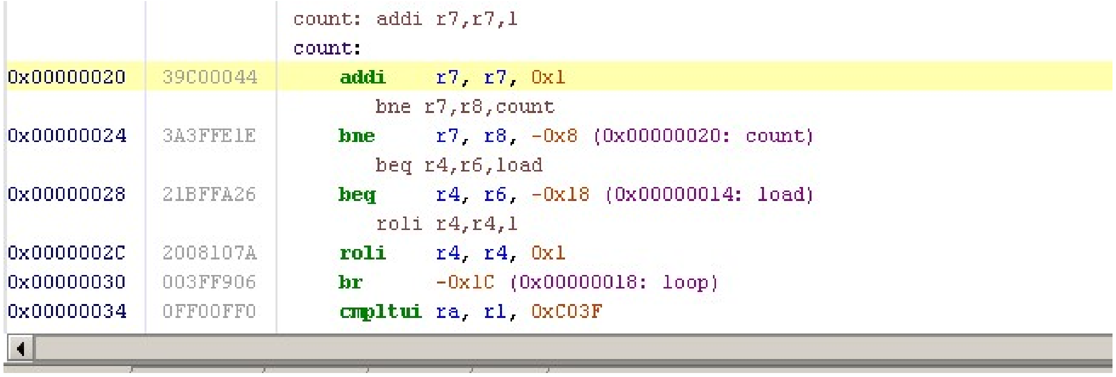
The program gets stuck between two instructions because it is acting as a loop and has a secondary step between each instruction.  Once it hits the bne (Branch Not ) it will continue to loop until r7 = r8. Once this happens it then exits the loop then loads the memory/ turn on the next LED.



```
                                 count: addi r7,r7,1
                                 count:
0x00000020   39C00044               addi    r7, r7, 0x1
                                       bne r7,r8,count
0x00000024   3A3FFE1E               bne     r7, r8, -0x8 (0x00000020: count
                                       beq r4,r6,load
0x00000028   21BFFA26               beq     r4, r6, -0x18 (0x00000014: load
                                       roli r4,r4,1
0x0000002C   2008107A               roli    r4, r4, 0x1
0x00000030   003FF906               br      -0x1C (0x00000018: loop)
0x00000034   0FF00FF0               cmpltui ra, r1, 0xC03F
```



```
                               count: addi r7,r7,1
                               count:
0x00000020   39C00044             addi    r7, r7, 0x1
                                     bne r7,r8,count
0x00000024   3A3FFE1E             bne     r7, r8, -0x8 (0x00000020: count)
                                     beq r4,r6,load
0x00000028   21BFFA26             beq     r4, r6, -0x18 (0x00000014: load)
                                     roli r4,r4,1
0x0000002C   2008107A             roli    r4, r4, 0x1
0x00000030   003FF906             br      -0x1C (0x00000018: loop)
0x00000034   0FF00FF0             cmpltui ra, r1, 0xC03F
```



```
                            count: addi r7,r7,1
                            count:
0x00000020   39C00044          addi    r7, r7, 0x1
                                  bne r7,r8,count
0x00000024   3A3FFE1E          bne     r7, r8, -0x8 (0x00000020: count)
                                  beq r4,r6,load
0x00000028   21BFFA26          beq     r4, r6, -0x18 (0x00000014: load)
                                  roli r4,r4,1
0x0000002C   2008107A          roli    r4, r4, 0x1
0x00000030   003FF906          br      -0x1C (0x00000018: loop)
0x00000034   0FF00FF0          cmpltui ra, r1, 0xC03F
```

4)

From my understanding the branches and jump instructions are the reasons why the LEDs cycle (LED0-LED7) The branches will make sure the values of r4 and r6 are the same before loading/cycling to the next LED. (0x0000FFFF). It affects the PC because it had to automatically allocate the space for the labels (count, load, loop). The address of PC changes according to the last used label from my observation and understanding.

For example:

0x0000020 and 0x0000024 (addi/bne for-loop)

5)

Main: stores and creates the location of LEDS. I'm pretty sure this is where we would add our delay for problem 6). R4 holds the addresses for the LEDS which gives the addresses shown below. R7 is a counter that will go until it equals R8 (which from my understanding R8 is a constant 0x7FFF)

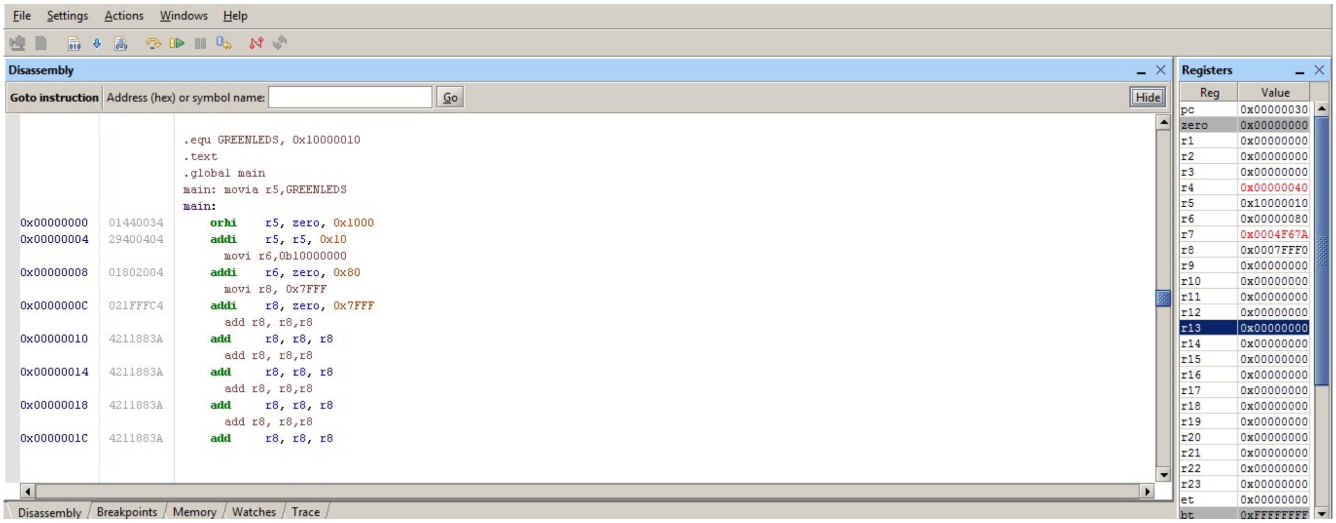
R4 = LEDS

| | |
|---|---|
| LED0 | 0x00000001 |
| LED1 | 0x00000002 |
| LED2 | 0x00000004 |
| LED3 | 0x00000008 |
| LED4 | 0x00000010 |
| LED5 | 0x00000020 |
| LED6 | 0x00000040 |
| LED7 | 0x00000080 |

The LEDs flash when there is a bit loaded into it. The label load, loads a bit into the register r4. The label loop: makes the program go into an infinite loop until it is interrupted by the user. The label count: counts up a temporary variable r7 until it equals r8, then it load a bit into the next LED.

6)

I wanted to add a delay like we learned in class recently, it's a helpful tip especially since we know that our DE0 board runs @ 50 Mhz. Sadly, I didn't have a chance to copy it down from the board to use it for my code. I took note that r8 is what set the speed of the LEDs, I then repeated to add r8, r8, r8. The result of add r8, r8, r8 is a delay because the processor has to waste more processes therefore slowing down the LED so it is much easier to observe.



```
                                  .equ GREENLEDS, 0x10000010
                                  .text
                                  .global main
                                  main: movia r5,GREENLEDS
                                  main:
0x00000000    01440034                orhi    r5, zero, 0x1000
0x00000004    29400404                addi    r5, r5, 0x10
                                        movi r6,0b10000000
0x00000008    01802004                addi    r6, zero, 0x80
                                        movi r8, 0x7FFF
0x0000000C    021FFFC4                addi    r8, zero, 0x7FFF
                                        add r8, r8,r8
0x00000010    4211883A                add     r8, r8, r8
                                        add r8, r8,r8
0x00000014    4211883A                add     r8, r8, r8
                                        add r8, r8,r8
0x00000018    4211883A                add     r8, r8, r8
                                        add r8, r8,r8
0x0000001C    4211883A                add     r8, r8, r8
```

Registers

| Reg | Value |
|---|---|
| pc | 0x00000030 |
| zero | 0x00000000 |
| r1 | 0x00000000 |
| r2 | 0x00000000 |
| r3 | 0x00000000 |
| r4 | 0x00000040 |
| r5 | 0x10000010 |
| r6 | 0x00000080 |
| r7 | 0x0004F67A |
| r8 | 0x0007FFF0 |
| r9 | 0x00000000 |
| r10 | 0x00000000 |
| r11 | 0x00000000 |
| r12 | 0x00000000 |
| r13 | 0x00000000 |
| r14 | 0x00000000 |
| r15 | 0x00000000 |
| r16 | 0x00000000 |
| r17 | 0x00000000 |
| r18 | 0x00000000 |
| r19 | 0x00000000 |
| r20 | 0x00000000 |
| r21 | 0x00000000 |
| r22 | 0x00000000 |
| r23 | 0x00000000 |
| et | 0x00000000 |
| bt | 0xFFFFFFFF |

7) My program was incredibly close to functioning as asked. My LEDs lit up in the order of LED0 ,LED1, LED2, LED3 ,LED7. Sadly the flashing does stop because of the MSB rotation not really working like it's suppose to. I tried to figure it out, but I couldn't successfully accomplish the non-stop flashing of the LEDs. The assembly program will be attached.