EE357 – Fall 2013 Final project (45 pts.)
Nov. 8th ~ Dec.6rd, 2013
<u>RISC emulator</u>

1. Write a Coldfire assembly code emulating a RISC ISA as given below; use an assembly project on the Codewarrior IDE.

2. Code in a pseudo high level language

```
unsigned char count = 0;
for(i=11; i<0; i--){
    if(array[index] == 0xff){
            count++;
    }
    index++;
}
diplayLED(count);
```

3. Assembly code

```
        ADDI R1,R0,#11 // R1(= i) = 11; R0 is hardwired to 0.
        ADD R2,R0,R0 // R2(= count) = 0;
        ADDI R3,R0,#array // R3(= index) = (the address of ‘array’);
        ADDI R4,R0,#0xff // R4 = 0xff
L2:     LOAD R5,R3,#0 // R5 = Mem[R3=index];
        BNE R4,R5,L1 // Branch to L1 if R4 != R5;
        ADDI R2,R2,#1 // R2 = R2 + 1; count++;
L1:     SUBI R1,R1,#1 // R1 = R1 – 1; i--;
        ADDI R3,R3,#4 // Points to the next item.
        BNE R0,R1,L2 // Branch to L2 if R1 != 0;
        DISP.B R2 // Display contents at the byte position in R2 on the LEDs’;
```

4. Instruction format
All instructions are 32 bits in width with the first 6 bits for the opcode.
R0 contains 0 all the time.
#Imm values are represented by 20-bit 2’s complement.
4.1 ADD: ADD rd,rs,rt ~ rd = rs + rt

000001	rs (3 bits)	rt (3 bits)	rd (3 bits)	Not used

4.2 ADDI: ADDI rt,rs,#Imm. (Immediate value)

000010	rs (3 bits)	rt (3 bits)	#Imm.

4.3 LOAD: LOAD rt,rs,#Imm. ~ rt = Mem[rs + (#Imm + starting address of the memory)]

000011	rs (3 bits)	rt (3 bits)	#Imm.

4.4 BNE: BNE rt,rs,#Imm. ~ pc = pc + #Imm. if rs != rt

000100	rs(3 bits)	rt(3 bits)	#Imm.

4.5 SUBI: SUBI rt,rs,#Imm. ~ rt = rs – #Imm.

001000	rs(3 bits)	rt(3 bits)	#Imm.

4.6 DISP.B rs ~ display the right most byte in rs

100000	rs (3 bits)	Not used.

5. Memory map (x – don't care)
code in binary:

```
0000 1000 0001 0000 0000 0000 0000 1011
0000 0100 0000 010x xxxx xxxx xxxx xxxx
0000 1000 0011 0000 0000 0000 0000 0000
0000 1000 0100 0000 0000 0000 1111 1111

0000 1101 1101 0000 0000 0000 0000 0000
0001 0010 1100 0000 0000 0000 0000 1000
0000 1001 0010 0000 0000 0000 0000 0001
0010 0000 1001 0000 0000 0000 0000 0001

0000 1001 1011 0000 0000 0000 0000 0100
0001 0000 1000 1111 1111 1111 1110 1100
1000 0001 0xxx xxxx xxxx xxxx xxxx xxxx
```

```
.data
risc_code:      .long 0x0810000b, 0x04040000, 0x08300000, 0x084000ff
                .long 0x0dd00000,0x12c00008, 0x09200001, 0x20900001
                .long 0x09b00004,0x108fffec,0x81000000
registers:      .long 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0
memory:         .long 0x0,0xff,0x1,0x2,0xff,0xff,0x0,0xf,0xff,0xa,0xff
```

6. Code skeleton

```
movea.l #risc_code,a0 //a0 for the PC (Program Counter)
movea.l #registers,a1 // a1 for the registers, R0 ~ R7
movea.l #memory,a2 // a2 for the (data) memory

fetch:

Fetch instruction using the PC into a register;
Decode the instruction using the opcode;
if( opcode == ADD){
execute the ADD instruction;
}
else if(opcode == ADDI){
execute the ADDI instruction;
}
.
.
.
else{
…
}

Increment PC by 4; // The instruction size is fixed as 4 bytes.

BRA fetch; // Branch always to fetch.
```