CSC 221: Computer Organization, Spring 2009

Practice Exam 2 Solutions

The exam will be open-book, so that you don't have to memorize the ASCII table or the details of the Pep/8 architecture.

1. With a two-address architecture, most machine instructions take two addresses as operands. An instruction such as

   ```
   add     X, Y
   ```

   says to add the value stored at address `Y` to the value at address `X`, leaving the result in `X`. That is, it is roughly equivalent to the C++ statement `X += Y;`. How many memory reads are required to fetch and execute this instruction on a two-address architecture (where `X` and `Y` are direct-mode operands)?

   There will be one group of reads to fetch the instruction and its operands (it is not specified how many bytes this will involve, nor is it clear how many bytes may be fetched in one read, but let's assume that this counts as a single read operation). Then the value of `X` will need to be read, followed by the value of `Y`, for a total of **three** reads.

   How many memory *writes* are required?

   The only write will be to store the modified value of `X` back into memory.

   Give an equivalent sequence of instructions for the Pep/8 architecture, and tell how many memory reads and writes are required for it.

   Since the Pep/8 is a one-address architecture, we need to use the accumulator to do the addition:

   ```
   LDA     X,d
   ADDA    Y,d
   STA     X,d
   ```

   This requires fetching three separate instructions from memory (for a total of nine bytes; each instruction causes a one-byte read of the instruction specifier, followed by a two-byte read of the operand), plus one read each for `X` and `Y`(two bytes each). This makes for **five** reads (or **eight**, if you count the operand fetches separately), for a total of 13 bytes. As with the two-address code, there is only one write needed, of the two-byte result for `X`.

2. Convert the following C++ program to Pep/8 assembly language:

   ```
   #include <iostream>
   using namespace std;

   int n;

   int f(int x)
   {
       if ((x & 1) == 1) {
           return (3 * x) + 1;
       } else {
   ```

```
        return x / 2;
    }
}

int main()
{
    cin >> n;
    while (n > 1) {
        n = f(n);
        cout << n << endl;
    }
}
```

```
        BR      main
n:      .BLOCK  2
x:      .EQUATE 2
retVal: .EQUATE 4
f:      LDA     x,s
        ANDA    1,i
        CPA     1,i
        BRNE    L1
        LDA     x,s
        ADDA    x,s
        ADDA    x,s
        ADDA    1,i
        BR      L2
L1:     LDA     x,s
        ASRA
L2:     STA     retVal,s
        RET0
main:   DECI    n,d
L3:     LDA     n,d
        CPA     1,i
        BRLE    L4
        STA     -4,s
        SUBSP   4,i
        CALL    f
        ADDSP   4,i
        LDA     -2,s
        STA     n,d
        DECO    n,d
        CHARO   '\n',i
        BR      L3
L4:     STOP
        .END
```

3. Convert the following C++ program to Pep/8 Assembly Language:

```
#include <iostream>
using namespace std;

int a, b;

int main() {
  cin >> a;
  cin >> b;
  b += a;
  a = b - a;
  cout << a;
  cout << b;
}
```

```
        BR      main
a:      .BLOCK  2
b:      .BLOCK  2
```

```
main:   DECI    a,d
        DECI    b,d
        LDA     b,d
        ADDA    a,d
        STA     b,d
        LDA     b,d     ; redundant
        SUBA    a,d
        STA     a,d
        DECO    a,d
        DECO    b,d
        STOP
        .END
```

4. Convert the following Pep/8 program to an equivalent program in C++:

```
newLine: .EQUATE 0x000A
         BR      main
x:       .WORD   1
y:       .WORD   2
z:       .WORD   3
c:       .BYTE   4
main:    DECI    y,d
         LDA     y,d
         ASLA
         STA     x,d
         ASLA
         ASLA
         ADDA    x,d
         ADDA    z,d
         STA     x,d
         DECO    x,d
         CHARO   newLine,i
         DECO    y,d
         LDA     z,d
         ORA     0x0030,i
         STBYTEA c,d
         CHARO   c,d
         STOP
         .END
```

```
#include <iostream>
using namespace std;

int x = 1;
int y = 2;
int z = 3;
char c = 4;

int main() {
  cin >> y;
  x = y * 2;
  x = y * 8 + x + z; // effect is x = y * 10 + z;
  cout << x << endl;
  cout << y;
  c = z | '0';
  cout << c; // effect is cout << z; if z is a single (decimal) digit
}
```

What is the output of the above program if the user enters `42`?

The output will be

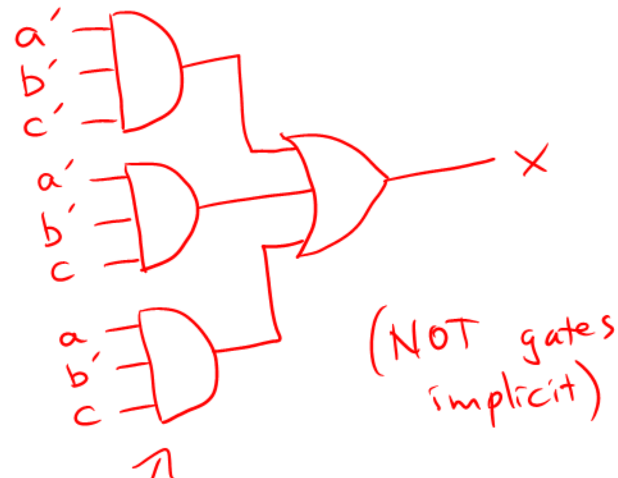
```
423
423
```

Note that printing `y * 10 + z` is always the same as printing `y` followed by `z`, if `z` is a single digit (provided the arithmetic doesn't overflow).

5. Consider the boolean formula $(a + b') \cdot (b' + c') \cdot (a' + c)$.

   a. Construct a truth table for this formula.

a b c	x
0 0 0	1
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	0

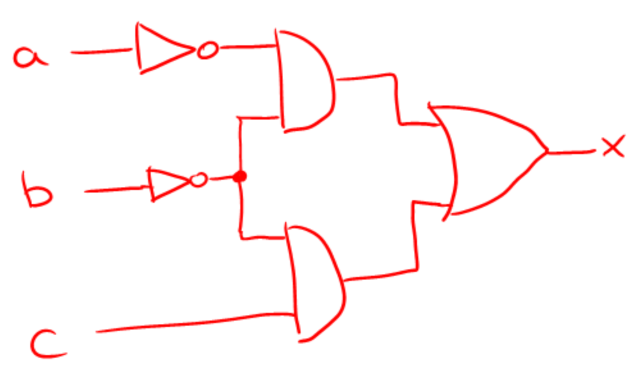


   b. Draw a circuit using AND, OR, and NOT gates with inputs *a*, *b*, and *c*, whose output is the value of this formula.

   ~~Instead of trying to draw a circuit here,~~ the boolean formula for the obvious two-level AND-OR circuit from the truth table is *a'b'c'+a'b'c+ab'c*.

   c. Draw an equivalent circuit using as few gates as possible.

   Here is the formula, based on the Karnaugh map minimization: *a'b'+b'c*.



6. Convert the following Pep/8 program to an equivalent program in C++:

```
        BR      main
n:      .BLOCK  2
fact:   .WORD   1

i:      .EQUATE 0
p:      .EQUATE 2
mul:    SUBSP   4, i
        LDA     0, i
        STA     p, s
        STA     i, s
L3:     CPA     n, d
        BREQ    L4
        LDA     p, s
        ADDA    fact, d
        STA     p, s
        LDA     i, s
        ADDA    1, i
        STA     i, s
        BR      L3
L4:     LDA     p, s
        STA     fact, d
        RET4
```

```
main:    LDA     7, i
         STA     n, d
L1:      CPA     0, i
         BREQ    L2
         CALL    mul
         LDA     n, d
         SUBA    1, i
         STA     n, d
         BR      L1
L2:      DECO    fact, d
         CHARO   '\n', i
         STOP
         .END
```

```
#include <iostream>
using namespace std;

int n;
int fact = 1;

void mul()
{
    int i, p;
    p = 0;
    i = 0;
    while (i != n) {
        p = p + fact;
        i = i + 1;
    }
    fact = p;
}

int main()
{
    n = 7;
    while (n != 0) {
        mul();
        n = n - 1;
    }
    cout << fact << endl;
    return 0;
}
```

7. Modify the above program so that the subroutine `mul` doesn't use the global variables `n` and `fact`; instead, it should take the values of `n` and `fact` as parameters, and produce the new value of `fact` as a return value. Show both the modifications necessary to `mul` and to `main`.

   Here is the modified program:

```
         BR      main
n:       .BLOCK  2
fact:    .WORD   1

i:       .EQUATE 0
p:       .EQUATE 2
n2:      .EQUATE 6
fact2:   .EQUATE 8
retVal:  .EQUATE 10
mul:     SUBSP   4, i
         LDA     0, i
         STA     p, s
         STA     i, s
L3:      CPA     n2, s
         BREQ    L4
```

```
        LDA     p, s
        ADDA    fact2, s
        STA     p, s
        LDA     i, s
        ADDA    1, i
        STA     i, s
        BR      L3
L4:     LDA     p, s
        STA     retVal, s
        RET4

main:   LDA     7, i
        STA     n, d
L1:     CPA     0, i
        BREQ    L2
        STA     -6, s    ; push n as first argument
        LDA     fact, d
        STA     -4, s    ; push fact as second argument
        SUBSP   6, i     ; allocate 2 arguments and a return value
        CALL    mul
        ADDSP   6, i
        LDA     -2, s    ; get the return value
        STA     fact, d
        LDA     n, d
        SUBA    1, i
        STA     n, d
        BR      L1
L2:     DECO    fact, d
        CHARO   '\n', i
        STOP
        .END
```

8. Design a combinational network that implements a *two-bit comparator*. This is a component that takes two pairs of input signals, a_1a_0 and b_1b_0, and produces one output line, labeled GT, which is 1 exactly when the binary number a_1a_0 is greater than the binary number b_1b_0. For example, if the inputs are 10 and 01, then the output should be 1; if the inputs are 10 and 10, or 01 and 10, then the output should be 0. Try to use as few logic gates as possible.

   See next page

   Here's the best formula I get: $a_1b_1' + a_1b_1a_0b_0' + a_1'b_1'a_0b_0'$.

9. A fancier version of the two-bit comparator would have three outputs, say GT, EQ, and LT, which reflect whether the first input (a_1a_0) is respectively greater than, equal to, or less than the second (b_1b_0). Show how to use one or more copies of this component (NOTE: *you do not need to design this component*, just draw a box with the appropriate input and output lines), plus a few logic gates, to construct a *four-bit comparator*; that is, a component which takes two groups of four input signals, $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$, and produces a 1 on exactly one of the three outputs GT, EQ, and LT depending on whether $a>b$, $a=b$, or $a<b$ (where a is the value given by the unsigned binary number a_1a_0, and b is given by b_1b_0).

   Use one comparator to compare the high-order bits, and another to compare the low-order bits. The GT output is the OR of the high-order GT with the AND of the high-order EQ and the low-order GT (that is, a is greater than b if either the first two bits are greater, or the first two bits are the same and the second two bits are greater). The LT output is similar, and the EQ output is simply the AND of both of the two-bit EQ outputs.

Maintained by Brian Howard (bhoward@depauw.edu). Last updated Friday, April 17, 2009