CS 201 Section 1 Homework 7 Suggested Solutions

1. In counting gates and gate delays we'll ignore inverters for simplicity. If you choose to count them, you'll naturally get somewhat higher numbers across the board.

(a) Ripple-carry adder.

From the logic equations for a full adder

$$\begin{aligned} S &= \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC \\ C_{out} &= BC + AC + AB \end{aligned}$$

we see that it circuit requires 9 gates and that the delay to obtain C_{out} is 2 gates.

Since a 16-bit ripple-carry adder simply chains 16 full adders together, the total gate count is $16 \times 9 = 144$ gates and the total delay to obtain C_{16} is $16 \times 2 = 32$ gates.

(b) Two-level carry lookahead adder.

First consider a 4-bit carry lookahead adder. We have the following logic equations for each i, $0 \leq i \leq 3$:

$$\begin{aligned} S_i &= \overline{A_i}\overline{B_i}C_i + \overline{A_i}B_i\overline{C_i} + A_i\overline{B_i}\overline{C_i} + A_iB_iC_i \\ G_i &= A_iB_i \\ P_i &= A_i + B_i \end{aligned}$$

Also, we have the following logic equations for the carry outs:

$$\begin{aligned} C_1 &= G_0 + P_0C_0 \\ C_2 &= G_1 + P_1G_0 + P_1P_0C_0 \\ C_3 &= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \\ C_4 &= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0 \end{aligned}$$

From these, we can conclude that the total gate count for the 4-bit adder is $4 \times (5+1+1) + 3+4+5+6 = 46$ gates. (The delay to calculate C_4 is irrelevant.)

Now consider the carry lookahead unit that puts together 4 carry lookahead adders. The carry-lookahead unit implements the following equations for each i, $0 \leq i \leq 3$:

$$\begin{aligned} P_n^4 &= P_{4n+3}P_{4n+2}P_{4n+1}P_{4n} \\ G_n^4 &= G_{4n+3} + P_{4n+3}G_{4n+2} + P_{4n+3}P_{4n+2}G_{4n+1} + P_{4n+3}P_{4n+2}P_{4n+1}G_4n \end{aligned}$$

and also the following equations for the carry outs:

$$C_1^4 = G_0^4 + P_0^4C_0$$

$$
\begin{aligned}
C_2^4 &= G_1^4 + P_1^4 G_0^4 + P_1^4 P_0^4 C_0 \\
C_3^4 &= G_2^4 + P_2^4 G_1^4 + P_2^4 P_1^4 G_0^4 + P_2^4 P_1^4 P_0^4 C_0 \\
C_4^4 &= G_3^4 + P_3^4 G_2^4 + P_3^4 P_2^4 G_1^4 + P_3^4 P_2^4 P_1^4 G_0^4 + P_3^4 P_2^4 P_1^4 P_0^4 C_0
\end{aligned}
$$

So the carry lookahead unit requires a total of $4 \times (1 + 5) + 3 + 4 + 5 + 6 = 42$ gates, and the grand total gate count for the whole circuit is $4 \times 46 + 42 = 226$ gates.

The carry lookaheard unit calculates C_4^4 from the P_i^4, G_i^4, and C_0 with 2 gate delays. The G_i^4 are calculated from the P_i and G_i with 2 gate delays. (The P_i^4 are calculated from the P_i with 1 gate delay, so they aren't on the critical path.) The P_i and G_i are calculated from A_i and B_i with 1 gate delay. So the total delay is just $2 + 2 + 1 = 5$ gates.

(c) Five-level carry select adder.

We can build a $2n$-bit carry select adder from 3 n-bit carry select adders plus the additional gates necessary to select the correct outputs from the two upper halves. These gates amount to a 2-input multiplexer for each of the n output lines and for the carry out. If we call the input signals D_0 and D_1 and the control signal H, the the output signal D is given by

$$D = \overline{H} D_0 + H D_1$$

So each multiplexer requires 3 gates and has a 2-gate delay.

Hence, we can write the following equations for $n \geq 1$:

$$
\begin{aligned}
Gatecount_{2n} &= 3 \times Gatecount_n + 3 \times (n + 1) \\
Delay_{2n} &= Delay_n + 2
\end{aligned}
$$

For $n = 1$, we will use an ordinary full adder:

$$
\begin{aligned}
Gatecount_1 &= 9 \\
Delay_1 &= 2
\end{aligned}
$$

Now all we have to do is compute the values of these quantities for each power of 2 up to 16:

n	$Gatecount_n$	$Delay_n$
1	9	2
2	33	4
4	108	6
8	339	8
16	1044	10

So the grand total is 1044 gates and 10 gate delays.

2. Here are the steps and corresponding contents of the 8-bit product register:

`Multiplicand = 0101`

initially	`00000110`
multiplier bit $0 = 0$: nop	`00000110`
shift product right	`00000011`
multiplier bit $1 = 1$: add	`01010011`
shift product right	`00101001`
multiplier bit $2 = 1$: add	`01111001`
shift product right	`00111100`
multiplier bit $3 = 0$: nop	`00111100`
shift product right	`00011110`
done	`00011110`

3. (a) We have $10_{10} = 1010_2 = 1.010_2 \times 2^3$. The sign bit is 0. For single precision, the biased exponent is $(127+3)_{10} = 130_{10} = 10000010_2$; for double precision it is $(1023+3)_{10} = 1026_{10} = 10000000010_2$. The part of the significand following the point is 010. So the single precision representation is:

```
0 10000010 01000000000000000000000
```

and the double precision representation is

```
0 10000000010 01000000000000000000
00000000000000000000000000000000
```

(b) We can determine the binary representation of $0.1_{10} = 1/10_{10} = 1/1010_2$ by doing binary long-division until we reach a repeating pattern in the quotient:

```
          0.000110011...
        --------------------
   1010 | 1.000000000...
          - 1010
          ------
            01100
           - 1010
            -----
               10000
             - 1010
                ----
                0110
```

I.e., $0.1_{10} = 0.0\overline{0011}_2 = 1.1\overline{0011}_2 \times 2^{-4}$ The sign bit is 0. For single precision, the biased exponent is $(127-4)_{10} = 123_{10} = 01111011_2$; for double precision, it is $(1023-4)_{10} = 1019_{10} = 01111111011_2$ The part of the significand following the point is 100110011... rounded to fit in the available number of bits. The single precision representation is

```
0 01111011 10011001100110011001101
```

and the double precision representation (assuming the "round to even" rule) is

```
0 01111111011 10011001100110011001
10011001100110011001100110011010
```

4. The idea is to simply add one to the exponent field. Since we don't have to worry about overflow, we can ignore the possibility that this addition will affect the sign bit. The code looks something like this:

```
ld [X], %l0
sethi %hi(0x00800000),%l1  ! 1 in LSB of exponent field
add %l0,%l1,%l0
st %l0,[X]
```

5. (4.3) Only one register may be gated out onto the B bus, so a single 4-bit field is sufficient to specify which one. The C bus may have many registers selected to be loaded simultaneously, so a full bit map is required.

6. (4.8) The statement is something like `i = (j - k - 6) + (j - k - 6)`, assuming the Java-to-IJVM compiler was fairly clever.