ENEE 446, FALL 2005
Solutions for the Mid Term Examination

1. We assume that all the functions below are equal to 0 for all \( t < 0 \)

(a) \( f(t) = \frac{2t}{t + 5} = 2 - \frac{10}{t + 5} \). Clearly, \( f(0) = 0 \) and the second term in the preceding expression is positive and decreases monotonically with increasing \( n \). Thus, \( f(t) \) is concave, positive, non-decreasing and can be the work function for a physical system. Throughput, \( \rho(t) = f'(t) = \frac{10}{(t + 5)^2} \).

(b) \( f(t) = \sqrt{t}e^{-t} \). \( f(0) = 0 \), but clearly this function decreases in the interval \([0, 1]\) and thus can not be the work function of a physical system.

(c) \( f(t) = e - e^{1-2t} \). \( f(0) = 0 \) and \( f'(t) = 2e^{1-2t} \) decreases monotonically. Since, \( f(t) \) is strictly non-negative this implies that \( f(t) \) is concave, non-negative and increasing for all \( t \geq 0 \) and hence can be the work function of a physical system.

2. We assume that the processor takes 3 clock cycles to execute one instruction. Now, because the execution is done at the same rate at which the instruction is fetched from the buffer, the total processing time for an instruction in the buffer is \( t_b = 3 + 3 = 6 \) cycles.

Fetching from memory takes an additional 3 cycles, therefore total processing time for an instruction in the memory is \( t_m = 3 + 6 = 9 \) cycles.

Memory speedup, \( (p) = 2 \).

(a) Equal likelihood of finding instruction in memory and buffer. Therefore, from Amdahl’s Law:

\[
S = \frac{T(1)}{T(p)} = \frac{0.5 * 6 + 0.5 * 9}{0.5 * 6 + 0.5 * (3 + 0.5 * 6)} = \frac{15}{12} = 1.25
\]

(b) Buffer is 3 times more likely to contain the instruction:

\[
S = \frac{T(1)}{T(p)} = \frac{0.75 * 6 + 0.25 * 9}{0.75 * 6 + 0.25 * (3 + 0.5 * 6)} = \frac{27}{24} = 1.125
\]

Note: There can be 2 other interpretations of this problem due to ambiguity in the problem statement.
1) The instruction is fetched into the buffer from the memory, so only the extra time it takes to fetch from the memory is speeded up, i.e., \( t_m = 3 + 3 + 3/2 = 7.5 \) after speedup. The answers in case are 1.11 for part (a) and 1.059 for part (b).

2) Fetching the instruction from the buffer does not take any additional time as the execution occurs at the same rate as the fetch. Thus, in this case \( t_b = 3 + 0 = 3, \ t_m = 3 + 3 = 6 \) before speedup and \( t_b = 3, \ t_m = 3 + 3/2 = 4.5 \) after speedup. The answers are 1.2 for part (a) and 1.11 for part (b).

Irrespective of the interpretation, note that the speedup in part (b) is always lesser than the speedup in part (a) because the proportion of instructions fetched from the memory, which is sped up, is reduced.

3. Efficiency is \( E(p) = (p^3 - 1) \frac{e^{1-p}}{6} \).

(a) Maximum efficiency implies \( E'(p) = 0 \), i.e.,

\[
E'(p) = 0 \\
\Rightarrow (3p^2 - (p^3 - 1)) \frac{e^{1-p}}{6} = 0 \\
\Rightarrow p^3 - 3p^2 - 1 = 0
\]

Solving for \( p \) in the above equation we get \( p^* \approx 3.11 \). Calculating the second derivative:

\[
E''(p^*) = (6p - 3p^2 - 3p^2 + p^3 - 1) \frac{e^{1-p}}{6} \Bigg|_{p=p^*} \\
= (p^3 - 6p^2 + 6p^* - 1) \frac{e^{1-p^*}}{6} = -0.207 < 0
\]

Thus, \( E(p) \) is maximized at \( p^* = 3.11 \). Therefore, best number of processors for maximum efficiency = \([3.11]\) = 3.

(b) Speedup \( S(p) = E(p) \ast p \). Therefore, speedup at max efficiency is \( S(3) = 3 \ast E(3) = 1.7594 \).

(c) \( r(p) = p^{1/3} \).

Utilization, \( U(p) = r(p) \ast E(p) = p^{1/3} (p^3 - 1) \frac{e^{1-p}}{6} \). Maximizing the utilization we get:

\[
U'(p) = \left( \frac{10}{3} p^{7/3} - \frac{1}{3} p^{-2/3} - p^{10/3} + p^{1/3} \right) \frac{e^{1-p}}{6} = 0 \\
\Rightarrow \frac{10}{3} p^3 - \frac{1}{3} - p^3 + p = 0 \\
\Rightarrow p = 0.2718, 3.4109
\]
Similarly

\[ U''(p) = \frac{e^{1-p}}{54p^{5/3}} (9p^5 - 60p^4 + 70p^3 - 9p^2 + 6p + 2) \]

\[
U''(p) \bigg|_{p=3.4109} = -0.2732 \\
U''(p) \bigg|_{p=0.2718} = 1.3645
\]

Therefore, \( U(p) \) is maximized at \( p = 3.4109 \). Comparing the utilization at the nearest integer values, \( U(3) = 0.846 \) and \( U(4) = 0.8298 \) we get max utilization 0.846 at \( p = 3 \) processors.

4. The multiplication can be done without extending the sign by using the algorithm given below:

(a) Algorithm to multiply \( x \) by \( y \) without sign extension is given in C pseudo-code.

Algorithm

(2’s complement multiplication without sign extension of multiplier)

```
{ switch(sign(x))
    {case 0:  switch(sign(y))
        {case 0:  p = xy; break;
        case 1:  p = xy - x*2^n; break;
            //multiply and correct the product
        } break;
    case 1:  switch(sign(y))
        {case 0:  p = (2^2n - |x|) y; break;
        case 1:  p = (2^2n - |x|) (2^n - |y|) + x*2^n; break;
            //multiply and correct the product
        } break;
    }
    // Set the flags
    S = sign(p); if(p = 0) Z = 1 else Z = 0;
}
```

(b) \( x = -5, y = 13 \). \(-5 = (11011)_2 \) and \( 13 = (01101)_2 \), therefore, choose \( n = 5 \). The steps are shown below:

**Step 1**: \( \text{sign}(x) = 1, \text{sign}(y) = 0 \)

**Step 2**:

\[ p = (2^{2\times5} - 5) \times 13 \mod 2^{10} \]
\[ = 1019 \times 13 \mod 2^{10} \]
\[ = 13247 \mod 1024 = 959 = (1110111111)_2 = -(0001000001)_2 = -65 \]

**Step 3**: \( S = \text{sign}((1110111111)_2) = 1; p \neq 0 \Rightarrow Z = 0 \)

Note: many of you have given an algorithm similar to: \( y = y_{n-1}y_{n-2} \cdots y_1y_0 \) in binary
Multiply \((x, y)\)
{
prod = 0; //Prod is 2n bits long
for (i=0; i<=n-1; i++)
    prod = prod + x*y[i]*2^(i);
return prod;
}

This works for the case \(x = -5\) and \(y = 13\) but it will not work in the general case. For example, let \(n = 3, x = -1 = 111_2\) and \(y = -2 = 110_2\). Using the above algorithm gives \(prod = -1 \times 2 - 1 \times 4 = -6\) which is obviously not correct.

5. Initialize \(r = 135\) and \(i = 4\). The two procedures are shown below:

(a) Restoring Division:
\[
\begin{align*}
    r &= 135 - 2^3.14 = 135 - 112 = 23 > 0 \Rightarrow q_3 = 1 \\
    r &= 23 - 2^2.14 = 23 - 56 = -33 < 0 \Rightarrow q_2 = 0 \\
    &\text{restore : } r = -33 + 2^2.14 = 23 \\
    r &= 23 - 2^1.14 = 23 - 28 = -5 < 0 \Rightarrow q_1 = 0 \\
    &\text{restore : } r = -5 + 2^1.14 = 23 \\
    r &= 23 - 2^0.14 = 23 - 14 = 9 > 0 \Rightarrow q_0 = 1
\end{align*}
\]
Thus, quotient \(q = 1001_2 = 9\) and remainder, \(r = 9\).

Non-Restoring Division:
\[
\begin{align*}
    r &= 135 - 2^3.14 = 135 - 112 = 23 > 0 \Rightarrow q_3 = 1 \\
    r &= 23 - 2^2.14 = 23 - 56 = -33 < 0 \Rightarrow q_2 = 0 \\
    r &= -33 + 2^1.14 = -33 + 28 = -5 < 0 \Rightarrow q_1 = 0 \\
    r &= -5 + 2^0.14 = -5 + 14 = 9 > 0 \Rightarrow q_0 = 1
\end{align*}
\]
Thus, quotient \(q = 1001_2 = 9\) and remainder, \(r = 9\).

(b) In one iteration of the while loop for non-restoring division, 1 addition/subtraction and 1 multiplication operation = 2 total operations are done irrespective of whether a bit is predicted correctly or wrongly. On the other hand for restoring division, 2 addition/subtraction and multiplication operations = 4 total operations are done in a single loop for mis-predicted bits while the count for correctly predicted bits remains the same.

It is twice as likely to mis-predict a bit as it is to predict it right \(\implies\)
Total steps (restoring division) = \(2 \times 1/3 \times n + 4 \times 2/3 \times n = 10n/3\)
Total steps (non-restoring division) = \(2n\)
Where \(n = \lceil \log_2 (y/x) \rceil\) is the number quotient bits.