WPE Exam Question for Computer Architecture – Fall 2012

(a) An IEEE single-precision floating point number has 32 bits. For simplicity, consider a 5-bit floating point representation that maintains all the other characteristics of the IEEE floating point standard (denormalization, etc.). Our 5-bit floating point representation uses 1 sign bit (S), a 2-bit mantissa (M), and a 2-bit exponent (E) in excess-1 format (bias of 1), as depicted below.

S (1 bit)	E (2 bits)	M (2 bits)

Fill in the table below showing the decimal value and excess-1 notation for each of the possible 2-bit exponents. Then, using the table, find the decimal values of the numbers A=00001 and B=00100.

2s complement	Decimal	Excess-1
00	0	01
01	1	10
10	-2	11
11	-1	00

A = 00001 = ?
S = 0, E = 00 (note that 00 means denormalized), M = 01
A = (-1)^S * 0.M * 2^(-2^(n-1)+2) = 0.01 * 2^0 = 0.01 = 1.00 * 2^(-2)

B = 00100 = ?
S = 0, E = 01 (Excess-1) = 0 (Decimal), M = 00
B = (-1)^S * 1.M * 2^(E-2^(n-1)+1) = 1.00 * 2^(1-2^1+1) = 1.00 * 2^0

(b) Considering the 5-bit floating point representation and the values of A and B from part (a), find the value of the register RZ in each of the programs below, when the programs are executed on an in-order processor.

The instruction **`ADDFP RA RB RZ`** adds the contents of register RA to the contents of register RB using a 5-bit floating point adder and stores the result in register RZ. In each program, assume that the initial contents of RA and RB are the values of A and B from part (a), respectively.

Program 1:	**Program 2:**
`ADDFP RA RA RX`	`ADDFP RB RB RX`
`ADDFP RX RB RY`	`ADDFP RX RA RY`
`ADDFP RY RB RZ`	`ADDFP RY RA RZ`

Program 1:
RX = 1.00 * 2^(-2) + 1.00 * 2^(-2) = 10.00 * 2^(-2) = 1.00 * 2^(-1)
RY = 1.00 * 2^(-1) + 1.00 * 2^0 = 0.10 * 2^(0) + 1.00 * 2^0 = 1.10 * 2^0

RZ = 1.10 * 2^0 + 1.00 * 2^0 = 10.10 * 2^0 = 1.01 * 2^1

Program 2:
RX = 1.00 * 2^0 + 1.00 * 2^0 = 10.00 * 2^0 = 1.00 * 2^1
RY = 1.00 * 2^1 + 1.00 * 2^(-2) = 1.00 * 2^1 + 0.00 * 2^1 = 1.00 * 2^1
RZ = 1.00 * 2^1 + 1.00 * 2^(-2) = 1.00 * 2^1 + 0.00 * 2^1 = 1.00 * 2^1
(NOTE: floating point error due to truncation – FP addition is not associative)

(c) The performance of a processor pipeline depend on the pipeline depth of the processor. You are tasked with finding the optimal pipeline depth of a processor using the following model.

Let T_B represent the time that the processor is busy executing instructions, and T_{NB} represent the time that the processor is not busy executing instructions. Non-busy time is due to pipeline stalls. The total time to execute a program (T) equals the busy time plus the non-busy time.

$$T = T_B + T_{NB}$$

Busy time for a program execution on a scalar processor can be expressed as the number of instruction in a program (N_I) multiplied by the time to pass through each pipeline stage (t_S).

$$T_B = N_I t_S$$

The pipeline stage delay can be expressed in terms of the total logic delay of the processor (t_P), the latch delay (t_O), and the number of pipeline stages (p).

$$t_S = \frac{t_P}{p} + t_O$$

A superscalar processor can execute multiple instructions at the same time. Let us define the average degree of superscalar processing (S), such that the busy time on a superscalar processor is expressed as follows.

$$T_B = \frac{N_I}{S} t_S$$

The non-busy time can be expressed as the number of hazards (N_H) multiplied by the average fraction (f_H) of total pipeline delay (t_{PIPE}) that the pipeline spends stalled when there is a hazard.

$$T_{NB} = N_H f_H T_{PIPE}$$

Write an expression for the total pipeline delay (T_{PIPE}) in terms of the total logic delay of the processor (t_P), the latch delay (t_O), and the number of pipeline stages (p). The total pipeline delay is the end-to-end pipeline delay including all logic stages and latches.

$$T_{PIPE} = ?$$

$$T_{PIPE} = t_P + t_O p$$

Based on the formulations above, write an expression for the performance of the processor (TPI), where performance is defined as the total execution time of a program (T) divided by the number of instructions in the program.

$$TPI = ?$$

$$TPI = \frac{T}{N_I} = \frac{T_B + T_{NB}}{N_I} = \frac{1}{N_I}\left[\frac{N_I}{S}\left(\frac{t_P}{p} + t_O\right) + N_H f_H (t_P + t_O p)\right]$$
$$= \frac{t_P}{Sp} + \frac{t_O}{S} + \frac{N_H f_H t_P}{N_I} + \frac{N_H f_H t_O p}{N_I}$$

As mentioned above, the performance of a processor pipeline depends on the pipeline depth. Find an expression for the performance-optimal pipeline depth (p_{opt}) of the superscalar processor described above. In other words, find a simplified expression for the optimal pipeline depth that minimizes TPI.

$$p_{opt} = ?$$

To find p_{opt} that minimizes TPI, take the derivative of TPI with respect to p, set equal to zero, and solve for p_{opt}.

$$\frac{dTPI}{dp} = \frac{d}{dp}\left[\frac{t_P}{Sp} + \frac{t_O}{S} + \frac{N_H f_H t_P}{N_I} + \frac{N_H f_H t_O p}{N_I}\right] = \frac{-t_P}{Sp^2} + \frac{N_H f_H t_O}{N_I}$$

$$\frac{-t_P}{S{p_{opt}}^2} + \frac{N_H f_H t_O}{N_I} = 0$$

$$\frac{t_P}{S{p_{opt}}^2} = \frac{N_H f_H t_O}{N_I}$$

$${p_{opt}}^2 = \frac{N_I t_P}{N_H f_H t_O S}$$

$$p_{opt} = \sqrt{\frac{N_I t_P}{N_H f_H t_O S}}$$

(d) Based on your expression for the optimal pipeline depth in part (c):

Does the optimal pipeline depth increase or decrease for a program with more hazards? Explain the intuition behind your answer based on your knowledge of pipelined processors.

Decrease – The longer the pipeline depth, the longer it takes to flush the pipeline on a hazard. Therefore, as the number of hazards increase, the optimal pipeline depth decreases.

Does the optimal pipeline depth increase or decrease as the latch delay decreases? Explain the intuition behind your answer.

Increase – When the latch delay decreases, the overhead of adding additional pipeline stages decreases. Therefore, the optimal pipeline depth increases.

Does the optimal pipeline depth increase or decrease as the average degree of superscalar processing increases? Explain the intuition behind your answer.

Decrease – Higher degree of superscalar processing reduces the busy time but does not reduce the non-busy time. Thus, the busy time becomes less relative to the non-busy time. This means that the effect of hazards becomes relatively more prominent.

Since hazard penalty increases with pipeline depth, the optimal pipeline depth decreases when higher degree of superscalar processing makes non-busy time a greater fraction of total time.

Does the optimal pipeline depth increase or decrease as the average fraction of the pipeline that stalls when there is a hazard decreases? Explain the intuition behind your answer.

Increase – When less of the pipeline stalls at a hazard, the cost of hazards decreases. This allows a higher optimal pipeline depth (which increases cost of hazards). The two factors offset, bringing the system back into equilibrium.

(e) The peak performance of a processor is often dictated by how fast the processor can bring data onto the chip and process the data. Consider a many-core processor with 256 scalar, in-order cores that operate at 1.4 GHz. Each core can perform up to two single-precision floating point operations per clock period (a floating point multiply-add operation). The peak off-chip memory (DRAM) bandwidth of the processor is 128 GB/s.

What is peak single-precision floating point throughput of the processor in Floating Point Operations Per Second (FLOPS)?

Peak throughput = 256 cores * 1.4G cycles/second * 2 FLOP/cycle = 716.8 GFLOPS

The following matrix multiplication kernel is used to calculate P=M*N on the processor for 16x16 square matrices M and N. Each core runs the same kernel code for a different row and column of the output matrix. In other words, each core computes one element of the product matrix (P). Given the performance constraints listed above, what is the peak performance in FLOPS attainable by the kernel? Assume that only loads from off-chip memory (DRAM) count toward off-chip memory bandwidth usage (don't count the storing of the results). M and N are stored in off-chip memory (DRAM).

```
void MatrixMulKernel(float* M, float* N, float* P, int Row,
int Col)
{
  // Row is the row index of the product (P) element
  // computed by this core
  // Col is the column index of the product (P) element
  // computed by this core
  float Pvalue = 0;
  // each thread performs a scalar product to compute
  // one element of the product
  for(int k = 0; k < 16; ++k)
    Pvalue += M[Row*16+k] * N[k*16+Col];
  P[Row*16+Col] = Pvalue;
```

```
}
```

Peak performance for the kernel = ?

The kernel loads 2*16 floats from DRAM and performs 2*16 floating point operations. Each float is 32 bits = 4 bytes, so the kernel requires 2*16*4 B/s of off-chip bandwidth to perform 2*16 FLOPS.

$$\frac{2 * 16\ FLOPS}{2 * 16 * 4\ B/s} = \frac{X\ FLOPS}{128\ GB/s}$$

X = 32 GFLOPS
(NOTE: This level of performance saturates the off-chip memory bandwidth but is well below the peak floating point throughput of the processor.)

The many-core processor contains a fast on-chip memory. You decide to re-write the kernel as shown below to reduce the off-chip memory bandwidth usage in hopes of improving performance. What is the peak performance in FLOPS attainable by the new kernel? Again, assume that only loads from off-chip memory (DRAM) count toward off-chip memory bandwidth usage (don't count the storing of the results). M and N are stored in off-chip memory (DRAM), but M_on and N_on reside in on-chip memory. (The qualifier __onchip__ declares memory in the on-chip memory structure.) In the new kernel, the cores cooperate to load the input data from off-chip memory into on-chip memory, then perform the matrix multiplication.

```
void MatrixMulKernel(float* M, float* N, float* P, int Row,
int Col)
{
  // declare memory in the on-chip storage
  __onchip__ float M_on[16][16];
  __onchip__ float N_on[16][16];
  float Pvalue = 0;
  // Collaborative loading of M and N into on-chip memory
  M_on[Row][Col] = M[Row*16 + Col];
  N_on[Row][Col] = N[Row*16 + Col];
  // wait for all cores to finish loading
  // into the on-chip memory
  Synchronize_cores();
  // compute the product element for this core
  for(int k = 0; k < 16; ++k)
    Pvalue += M_on[Row*16+k] * N_on[k*16+Col];
  P[Row*16+Col] = Pvalue;
}
```

Peak performance for the new kernel = ?

The kernel loads 2 floats from DRAM into the on-chip memory and performs 2*16 floating point operations on data in the on-chip memory. Each float is 32 bits = 4 bytes, so the kernel requires 2*4 B/s of off-chip bandwidth to perform 2*16 FLOPS.

$$\frac{2 * 16\ FLOPS}{2 * 4\ B/s} = \frac{X\ FLOPS}{128\ GB/s}$$

X = 512 GFLOPS

(NOTE: This level of performance saturates the off-chip memory bandwidth. By reducing the off-chip bandwidth requirement by 16x, the performance increases by 16x.)