1. **Cache memories (35 points total)**

It turns out that object-oriented software makes heaviest use of the memory it has most recently allocated (this memory contains "new objects"). This suggests that one might improve its performance by guaranteeing that recently allocated memory stays in the cache. We can do this by assuring that no other blocks of main memory map to the same lines as new objects.

Suppose we decide to allocate new objects at memory addresses between $m \cdot 2^k$ and $(m+1) \cdot 2^k - 1$. We must assure that no other memory addresses in our process map to the same cache lines as addresses between $m \cdot 2^k$ and $(m+1) \cdot 2^k - 1$.

(a) **(4 points)** This strategy results in "holes" (forbidden addresses) in the memory that our process may use. Are they holes in virtual memory or physical memory? Explain. Assume that the cache is physically addressed. That is, assume that it is the physical address that determines which cache line a block maps to.

(b) **(6 points)** Suppose our system has a 256 KB direct-mapped cache and we allocate new objects between addresses 0 and 64K−1. To avoid conflicts for these addresses, what memory addresses must our process *not* use?

(c) **(6 points)** If main memory is 32 MB, what fraction of memory is unavailable to our process as a result of this strategy? For full credit, please be precise. You may give an approximation for partial credit.

(d) **(9 points)** Would your answer to part (b) change —

(i) if the cache’s line size changed, but the cache size and associativity stayed the same?

(ii) if the cache size changed, but the line size and associativity stayed the same?

(iii) if the associativity of the cache changed, but the line size and cache size stayed the same?

For full credit, explain your answers! *Note:* You should be able to do this part even if you get the wrong answer to part (b).

(e) and (f) **(5 points each)** Redo parts (b) and (c), supposing that instead of a 256 KB direct-mapped cache, we have a 512K two-way set-associative cache.
2. Two Topics in Computer Architecture (35 points total)

If you make an assumption about any of these problems, please use precision in stating the details of that assumption.

A. The figure below shows two interconnection permutations from a paper in *IEEE Trans. on Computers*. Larger versions of these permutations extend the figures in a consistent way. For this problem about permutations of the types shown, let $N$ represent the number of inputs, with the constraint that $N$ is a power of 2. There are obviously $N$ outputs also.

![Figure 3](image)

(a) (4 points) The shuffle can be described as $\sigma(A) = \begin{cases} 2A & \text{for } A < N/2 \\ 2A \mod(N) + 1 & \text{for } A \geq N/2 \end{cases}$

Provide a similar description for permutation $\tau$ shown in part (b) of the figure. That is, provide an equation such that if you are given $A$ as a decimal number, you can directly compute $\tau(A)$.

(b) (5 points) If the binary expansion of address $A$ is $a_{k-1}, a_{k-2}, ..., a_1, a_0$, then permutation $\sigma$ can be described as $\sigma(a_{k-1}, a_{k-2}, ..., a_1, a_0) = a_{k-2}, ..., a_1, a_0, a_{k-1}$. Provide a similar description for permutation $\tau$ shown in part (b) of the figure. That is, provide an equation such that if you are given $A$ as a binary address, you can directly compute $\tau(A)$ as a binary address. The equation could involve logical or arithmetic operations on the address bits.

(c) (4 points) It is well-known that $\log_2(N)$ applications of the shuffle permutation will return all addresses to their original values. Prove or disprove the proposition: A fixed number of applications of permutation $\tau$ will return all addresses to their original values. State your conclusion.

(d) (5 points) Suppose a multi-stage interconnection network is made from exchange switches having either straight or exchange settings, and uses the $\tau$ permutation to interconnect the exchange switches in a manner similar to an omega network. Can this network be used to support a "full-information function", as described in the Stone text? Justify your answer.
B. The Stone text presents a general model of a computational workload when there is both serial and parallel work. The model, in chapter 3, expresses efficiency in two ways in equations (3.1) and (3.2).

(a) (4 points) For this part, assume that a certain application is such that 20% of the workload must be spent in serial code and the rest is spent in parallel code on a parallel processing system with 16 processors. That is, \( \alpha = 0.20 \) in Stone's model as expressed in the two equations. What is the efficiency?

(b) (4 points) Continuing with the 16 processor system and the same workload, if the parallel code produces 100 results and takes 100 units of time, how many units of time are required for the serial code?

(c) (4 points) Now assume the parallel system has only eight processors but the amount of work remains the same as above. What is the efficiency?

(d) (5 points) Finally, assume that the computing system is a linear pipeline with 16 stages, the pipeline clock period is one unit of time, and the amount of work remains the same as above. Let the unit of time be the same as in part (b). Assume that serial work uses one stage at a time of the pipeline and takes the same overall time as the serial work in (b). The pipeline is also used for the parallel part, but serial and parallel work cannot be overlapped in the pipeline. What is the efficiency of the pipeline for the total process?
3. (30 Points total) Instruction Sets, CPU Design, and Assembler Programming

For all questions, justify your answer.

Recursion is a powerful technique in computing. Many computer architectures provide support for recursion, using a stack. The CPU and instruction set of [Mano, *Computer System Architecture*, Chapter 5, figures 5-4, 5-6, and table 5-6] is summarized in the attachment to this exam. In this question, you are to modify this CPU and instruction set to support recursion, and then, using your modified CPU, write a recursive program.

1. (6 points) Two new instructions need to be added to the instruction set: **PUSH**, and **POP**. **PUSH** stores the Accumulator Register (AC) contents on the top of the stack. **POP** loads contents at the top of the stack into the Accumulator Register.

What additional hardware does the CPU need to support these instructions? Be specific about the function, inputs, and outputs of each new piece of hardware (no logic design is required, however).

2. (6 points) Write the register-transfer language (RTL) description of the execute phase of the **PUSH** and **POP** instructions.

3. (6 points) What other instructions, if any, need to be modified or added for this CPU to support recursion? Write the RTL description of the execute phase of each.

4. (12 points) Using your modified CPU and instruction set, write a program that computes the sum of the numbers from 1 to N, using recursion. The parameter value N is input by a user, and the value of the resulting sum is output to the user. In the high-level language C, the recursive subroutine would look something like:

   ```c
   int summation(int N)
   {
       if (N == 0)
           return(0);
       else
           return(N + summation(N-1));
   }
   ```

Your program must pass all parameters and return values through the stack; do not use any of the registers for this purpose, even if you find the resulting code does unnecessary load and store operations. You are given the main routine below; write the subroutine, SUM.

```
ORG 100
LP1,  
  SKI  
  BUN  LP1
  INP  
  CLRSP 
  PUSH 
  BSA  SUM
  POP  
LP2,  
  SKO  
  BUN  LP2
  OUT  
  HLT
```