Mid-term exam of Computer Architecture

Solve the following problems and submit a report through the google classroom by Jan. 24, 2025 (Firm deadline).

PROBLEM 1

Your task is to compare the memory efficiency of four different styles of instruction set architectures. The architecture styles are

1) Accumulator- All operations occur between a single register and a memory location

2) Memory-memory- All three operands of each instruction are in memory

3) Stack- All operations occur on top of the stack. Only push and pop access memory, and all other instructions remove their operands from the stack and replace them with the result. The implementation uses a stack for the top two entries; accesses that use the other stack positions are memory references.

4) Load-store- All operations occur in registers, and register-to-register instructions have three operands per instruction. There are 16 general-purpose registers, and register specifiers are 4 bits long.

To measure memory efficiency, make the following assumptions about all four instruction sets:

・The opcode is always 1 byte (8 bits)
・All memory addresses are 2 bytes (16 bits)
・All data operands are 4 bytes (32 bits)
・All instructions are an integral number of bytes in length.

There are no other optimizations to reduce memory traffic, and the variables A, B, and C are initially in memory.

Invent your own assembly language mnemonics and write the best equivalent assembly language code for the high-level-language fragment given. Write the four code sequences for

D=(A-B)*(A+B)/C+(A-B)/C

Calculate the instruction bytes fetched and the memory-data bytes transferred. Which architecture is most efficient as measured by code size? Which architecture is most efficient as measured by total memory bandwidth required (code + data)?

PROBLEM 2

The following matrix transformation is called matrix transposition.

$$\begin{bmatrix} A11 & A12 & A13 & A14 \\ A21 & A22 & A23 & A24 \\ A31 & A32 & A33 & A34 \\ A41 & A42 & A43 & A44 \end{bmatrix} \Rightarrow \begin{bmatrix} A11 & A21 & A31 & A41 \\ A12 & A22 & A32 & A42 \\ A13 & A23 & A33 & A43 \\ A14 & A24 & A34 & A44 \end{bmatrix}$$

For example, the code to transpose a 3x3 matrix, written in C, would look like this

```
for (i = 0; i<3; i++) {
        for (j = 0; j < 3; j++) {
                output[j][i] = input[i][j]
        }
}
```

Assume that 256x256 double-precision 2D arrays *input* and *output* are stored sequentially in memory in row-major order (rows are stored sequentially in memory in the order of the right-most array index). If the transposition is performed using a processor with a full-associative cache of 16 KB, whose block size is 64 bytes, answer the following questions.

(1) Find the number of cache misses that occur in the transposition process.

(2) If you consider blocking arrays to minimize the number of cache misses, find the appropriate block size.
In addition, find the number of misses at that time and how much the miss rate improves compared to (1).

(3) Using the block size parameter B, write an improved code using the C language that performs a transposition for each block of size BxB.
(4) In the case of single-precision data in the matrix transportation, how much the miss rate will be improved by introducing the BxB blocking.

(5) Furthermore, run the created code on a personal computer, determine the relationship between the size of B and the execution time, and from this estimate the cache size of the personal computer used. In order to confirm the effect of blocking, it is necessary to examine the

specifications of the CPU used in the PC, define an array of a size that is not held on the cache, and examine how the program time changes by adjusting B. If changing B does not result in a blocking effect, consider the reasons why.

The program execution time measurement can be obtained, for example, in the C language, using the clock function as follows.

```
#include< stdio.h>
#include< time.h>

int main(void)
{
  clock_t start, end;
  int i;.

  start = clock();
  printf( "Start time:%d¥n", start );

  /* Write here the program kernel for which you want to find
the execution time */

  end = clock();
  printf( "End time:%d¥n", end );
  printf( "Processing time:%d[ms]³", end - start );

  return 0;
   }
```