Problem 1. (20 points) A simple SAS multiprocessor system is as shown here. The system data bus is 32 bits wide and runs at 100 MHz. Each processor requires 40 ns to arbitrate for the bus and 40 ns to access memory for reading or writing.

(a) A program running on this system has five parallel portions, each of which executes for 800 ns and requires eight 32-bit data words from memory. The memory access time is not included in the execution time. After completing the calculations on the data, each processor stores its result as two 32-bit data words. All calculations in the parallel phase are independent and do not rely on results from the other processors. After all processors finish, one of the processors executes the serial portion of the code, which adds up the results stored by the five processors in memory. The final result is stored as two 32-bit data words in memory. Execution of the serial phase takes 500 ns. This does not include memory access time.

How much time is needed for a processor to read eight 32-bit words from the memory?

(b) What is the total time to complete executing the parallel part of the program and to store the results into memory? Assume each processor acquires the bus in round-robin fashion starting from CPU 0 and they all finish the calculation in the order they started.

(c) What is the total time to complete executing the program and store the final result in memory?

(d) What would be the total time required if all the work were done by a single processor? Assume that the time to execute the parallel and serial portion of the program remain the same and that the data required for calculations is read at once as a single chunk of forty 32-bit words from memory. Also assume that the result of the first five calculations are stored in internal registers of the processor, and only the result from the serial portion of the value is written back to the memory as the result. This value consists of two 32-bit words. (In other words you will read the memory once and write to the memory once during the whole process.)
Problem 2. (20 points) For the interconnection topology shown below, each link is a 128-bit wide bus operating at 100MHz. There is a communication base cost of 10ns between links.

(a) (i) What is the time to send 512 bytes to an adjacent processor?
    (ii) What is the time to send 512 bytes to a processor three links (hops) away? (assuming no intermediate delays)
    (iii) What is the time to send 512 bytes to a processor on the opposite corner of a 3×3×3 generalized hypercube?
(b) For the shared-memory architecture shown below, there is a communication base cost of 10ns, and the communication network is a 128-bit wide bus operating at 100MHz. What is the transfer time to send 512 bytes to any other processor? (Ignore propagation time differences and assume an idle bus.)

(c) Compare the transfer times for the two architectures when there are three transmissions to be made: (i) Assume in the message-passing model that a processor is communicating with three of its direct neighbors, and (ii) in the shared-memory model all three transmissions go over the common bus. Each transmission transfers 512 bytes.

Problem 3. (10 points) You are running a job for a customer who would like to have it completed as fast as possible, and is willing to pay you $5 to get it done in an hour, $10 for half an hour and $15 if you can get it done in 15 minutes. The job takes one hour to run on a single processor, and 80% of it is parallelizable. If it costs you $1 to use a processor for 15 minutes, how long will you take to run the job to make the maximum profit?
Problem 4. (30 points) Suppose that it is necessary to multiply two $256 \times 256$ arrays $A$ and $B$ to obtain another array $C$ on an array processor with 64 parallel arithmetic units.

(a) How would you arrange the arrays in memory? Draw a diagram like that on page 9 of lecture 3.

(b) Assume that each of the matrices $A$, $B$ and $C$ have been divided into sixteen partitions as shown below. Each partition is called a sector, which contains an array of $64 \times 64$ elements. Just as an individual element of matrix $C$ is the sum of products of elements from $A$ and $B$, so a sector of $C$ is the matrix sum of matrix products of sectors from $A$ and $B$. For example, in the diagram below, sector $(0,0)$ of $C$ is the matrix sum of the matrix products

$$A(0,0) \times B(0,0) + A(0,1) \times B(1,0) + A(0,2) \times B(2,0) + A(0,3) \times B(3,0)$$

Assume that you have the following procedures

Matrix_Sector_Zero($X$) — sets all elements of a sector $X$ to zero.

Matrix_Sector_Mult($X$, $Y$, $Z$) — multiplies sectors $X$ and $Y$ and stores the result in sector $Z$.

Matrix_Sector_Add($X$, $Y$, $Z$) — adds sectors $X$ and $Y$ and stores the result in sector $Z$.

Write pseudo-code to multiply matrices $A$ and $B$ and store the result in matrix $C$, using the procedure calls given above.

(c) Write the code for the three procedures mentioned in part (b) using pseudo-code.

(d) Now that you have established the actions to be performed, let us see if you can write this in terms of a single procedure. The next step is to take into account the way that the arrays are actually stored in memory. In any of the 64 memories, the elements of sector $(i, j)$ begin at offset $64i + 256j$ from the beginning of the array. You can use this formula for the beginning address of a sector to rewrite your pseudo-code. Let us define new $1024 \times 64$ "alias" arrays $A$, $B$ and $C$ (boldface names), which occupy the same memory locations as $A$, $B$ and $C$. Re-write your pseudo-code from section (b) so that all operations are performed on individual elements (and not sectors) of the alias arrays. Comment your code to indicate which portions will be done in parallel. Show whether this can be reduced to more elegant code, e.g., by merging loops, rearranging order of summation etc.

Question 5. (20 points) Given below is one possible solution to (d). Translate it into the assembly code that was discussed in lecture 3.

Assume that you also have the following instructions.
ADNX k, i, j  
(Adds the contents of index registers i and j, leaving the result in index register k.)

SHLNX j, i, 3  
(Shifts index register i left by three bits, effectively multiplying its contents by 8, and leaves this quantity in index register j)

**procedure** Mat_Mult (matrix A, B, C) {
    for (k = 0; k < 768; k += 256) {
        for (i = 0; i < 255; i++) {
            p = i + k;
            C[p, j] = 0, (0 ≤ j ≤ N–1);
            for (j = 0; j < 192; j += 64) {
                q = i + 4j;
                r = j + k;
                for (j = 0; j < 63; j++) {
                    s := r + j;
                    C[p, k] := C[p, k] + A[q, j] × B [s, k], (0 ≤ k ≤ N–1);
                }
            }
        }
    }
}