EE 7725 **Homework 3** **Due: 11 November 1996**

The procedure below is part of a parallel program designed to exercise an interconnection network. It is executed by all processors; the procedure composes a simple message, computes the destination processor, pauses at the barrier, then sends the message. Because of the barrier all processors will send their messages at the same time. Assume that only the `send_msg` call generates network traffic.

```
/*
 Predefined variables and constants:

 n              The number of stages in the network.
 m              The size of the cells.
 this_proc_num  The processor number (which equals the network input number).

*/

taskmain()
{
 int desto, source_dig[n], desto_dig[n], i;
 char msg[80];

 /* Copy a silly message into character array msg. */
 sprintf(msg,"Happy Fall Holiday from processor %2d\n", proc_number);

 /* Convert processor number into an n-digit, radix-m number with digits
    placed in array source_dig. */
 split_to_digits(source_dig,this_proc_num,n,m);

 /* Create a radix-m representation of the destination. (% is the modulo operator.) */
 for(i=0; i<n; i++) desto_dig[i] = ( i + source_dig[i] ) % m;

 /* Convert the destination to an integer. */
 desto = combine_to_integer(desto_dig,n,m);

 /* Wait for everyone to finish code above. (Assume the barrier generates no traffic.) */
 barrier();

 /* Send the message. */
 send_message(desto, msg);
}
```

Problem 1: Show that there will be no contention in a system using an omega network running the code above. *Hint: find the path through the network for a request (say (s, d)).*

Problem 2: Show that there will be contention in a system using a baseline network running the code above.

Problem 3: Modify the program so that it runs on a baseline network without contention without an identity connection assignment. (That is, processor i should not send its message to itself.)

Solution on next page.

Problem 1: Consider request (a, α). By inspection of the program, $\alpha_{(y)} = a_{(y)} + y \bmod m$. Consider another request (b, β); because $a_{(y)} < m$ and $b_{(y)} < m$ it can easily be shown that $a_{(y)} = b_{(y)} \iff \alpha_{(y)} = \beta_{(y)}$. (If only $a_{(y)} = b_{(y)} \Rightarrow \alpha_{(y)} = \beta_{(y)}$ was shown no conclusion could be drawn if we knew that $\alpha_{(y)} = \beta_{(y)}$.)

Consider the links at cell outputs in stage x used by requests (a, α) and (b, β) such that $a \neq b$:

$$a_{(n-x-2)}a_{(n-x-3)} \cdots a_{(1)}a_{(0)}\alpha_{(n-1)}\alpha_{(n-2)} \cdots \alpha_{(n-x)}\alpha_{(n-x-1)}$$

and

$$b_{(n-x-2)}b_{(n-x-3)} \cdots b_{(1)}b_{(0)}\beta_{(n-1)}\beta_{(n-2)} \cdots \beta_{(n-x)}\beta_{(n-x-1)}$$

Let j be a digit position in which a and b differ. If $j < n - x - 1$ then the links numbers are different; if $j \geq n - x - 1$ then $\alpha_{(j)} \neq \beta_{(j)}$ and again the link numbers are different.

Since distinct requests (those with different input numbers) cannot share a link, there will be no contention in the network.

Problem 2: Consider the requests offered to inputs zero and one in a $(n > 1)$-stage baseline network. These will enter the same cell and the most-significant digits of their input numbers will be the same. In a baseline network the output used in the first-stage cell is the most-significant digit of the destination, in this case computed from the most-significant digit of the input (source). Therefore the two requests will be routed over the same link, indicating contention.

Problem 3: There are many ways the program could be modified; keeping in the spirit of the program, digit i of the destination could be found from digit $n - 1 - i$ (rather than i) of the source.

```
taskmain() {
 int desto, source_dig[n], desto_dig[n], i;   char msg[80];

 /* Copy a silly message into character array msg. */
 sprintf(msg,"Happy Fall Holiday from processor %2d\n", proc_number);

 /* Convert processor number into an n-digit, radix-m number with digits
    placed in array source_dig. */
 split_to_digits(source_dig,this_proc_num,n,m);

 /* Create a radix-m representation of the destination. (% is the modulo operator.) */
 for(i=0; i<n; i++) desto_dig[i] = ( i + source_dig[n-i-1] ) % m;

 /* Convert the destination to an integer. */
 desto = combine_to_integer(desto_dig,n,m);

 /* Wait for everyone to finish code above. (Assume the barrier generates no traffic.) */
 barrier();

 /* Send the message. */
 send_message(desto, msg);
}
```