MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 127-1

# A Computer Simulation Facility for Packet Communication Architecture

by

Clement K. C. Leung
David P. Misunas
Andrij Neczwid
Jack B. Dennis

(A paper to be published in the Proceedings of the Third Annual Symposium on Computer Architecture to be held January 1976)

This paper was prepared with the support of the National Science Foundation under grant DCR75-04060.

August 1975

Revised November 1975

# A Computer Simulation Facility for Packet Communication Architecture *

by

Clement K.C. Leung
David P. Misunas
Andrij Neczwid
Jack B. Dennis

Project MAC
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract: Several proposals for computer data processing and memory systems that exploit the inherent parallelism in programs expressed in data flow form have been advanced recently. These systems have packet communication architecture -- each system consists of many units that interact only through the transmission of information packets through channels that link pairs of units.

A simulation facility for evaluating the programmability and potential performance of these proposed data processing and memory systems has been designed. The facility uses microprocessor modules to emulate the behavior of system units or groups of units. By conducting a simulation in accurate scale time a precise extrapolation of performance of a proposed system may be obtained.

The user of the facility will specify the system to be simulated in an architecture description language. A host computer translates the system description modules into microprocessor programs and controls the loading and monitors the operation of the microprocessors. Application of the facility is illustrated by consideration of a simple data flow processor.

## Introduction

Recently, a number of proposals for computer data processing and memory systems organized to exploit the parallelism inherent in programs expressed in data flow form have been developed. These include a series of machines of increasing capability described by Dennis and Misunas [2, 3], two machines capable of supporting high level language including procedures as data [5, 6, 8, 9, 10] and memory systems organized for highly parallel operation [1].

Each of these systems consists of many units connected by channels, and is organized so the units operate asynchronously and interact only through transmission of information packets over the channels. Each unit of these systems is designed so it never has to wait for a response to a packet it has transmitted to another unit, if other packets are waiting for its attention; this design principle permits a high level of concurrent processing. The units themselves may be constructed of simpler units and channels that cooperate in the same manner, yielding a hierarchical structure in which interactions occur only at well-defined interfaces. Systems structured to operate according to this discipline are called packet communication systems and are said to have packet communication architecture.

The application of packet communication architecture to computer system design is now sufficiently advanced that careful evaluation of the performance potential of proposed systems is required. Since analytic techniques of sufficient power are not known, evaluations must be carried out by simulation. The simulation of a conventional computer architecture is readily carried out by programming a conventional Von Neuman-type computer, and the result of such simulation may be easily interpreted to predict performance of a proposed machine. However, simulation of a highly asynchronous system is not so easily accomplished using a conventional sequential computer -- much effort (in programming and in simulation runs) would be spent in the implementation of psuedo parallel processes and the coordination of their interactions.

* The work reported here was supported by the National Science Foundation under grant DCR75-04060.

With the advent of low-cost LSI processors there is an attractive alternative to programmed simulation on a conventional computer: a system having packet communication architecture is divided into parts and each part is emulated by a microprocessor. We have designed an architectural simulation facility based on this idea. The facility consists of a number of microprocessor modules arranged so they may easily communicate through a network for the simulation of any packet communication system. The system to be simulated is specified in an architecture description language designed expressly for packet communication systems. A host computer translates architecture descriptions into program modules executed by the microprocessors. The host computer also provides means for debugging and for measuring performance of the simulated system.

Our explanation of the simulation facility is aided by discussing its application to modeling the operation of a simple data flow processor. We start with a brief description of the data flow processor and show how the structure of this processor might appear when expressed in our architecture description language. Next comes a detailed discussion of the hardware portion of the facility and how it supports the modeling of packet communication systems. We conclude with a brief discussion of the software support to be implemented on the host computer.

## An Example of Packet Communication Architecture

Throughout this paper we shall use a simple data flow processor as an example of a packet communication system. This data flow processor has been proposed for certain signal processing computations such as waveform generation and filtering in which a fixed constellation of operations is applied to a stream of data. The processor does not support data-dependent decisions, structured data, or procedures, though these features have been considered in generalized versions of this processor [3, 5, 6, 8].

The units and channels that comprise the top-level description of the data flow processor are shown in Figure 1. Instructions of a data flow program to be executed by the data flow processor are stored in Instruction Cells (Figure 2). Each Instruction Cell holds an instruction of the program, contains registers for holding one or two operands of the instruction, and is

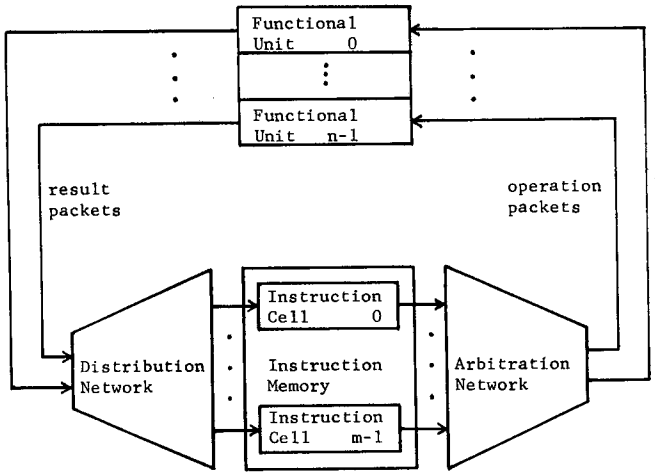


Figure 1. Structure of the elementary data flow processor.

designated by a unique cell identifier. An instruction specifies an operation to be performed on its operands and specifies each register (by a cell identifier and a register index 1 or 2) which is to receive a copy of the result. When all operands of an instruction are present in a Cell, the Cell is enabled and its content is transmitted as an operation packet to the Arbitration Network. Each operation packet is forwarded by the Arbitration Network to a Functional Unit capable of interpreting the operation packet. A Functional Unit performs the function specified by the instruction code of the operation packet it receives on the operands in the packet and, for each destination specified in the operation packet, generates a result packet consisting of a copy of the result and the cell identifier/register index of a destination cell register. The Distribution Network accepts result packets from the Functional Units, and delivers each result packet to the Cell addressed by the cell identifier in the packet. After the result packet is received by a Cell, the result in the packet is stored in the register addressed by the register index of the packet. If all of its operands are present, a Cell receiving a result packet is enabled and generates another operation packet to be processed. A more detailed description of the architecture and operation of the data flow processor is given in [2]. We note that depending on their construction, the Arbitration Network and the Distribution Network are capable of processing one or more packets simultaneously.

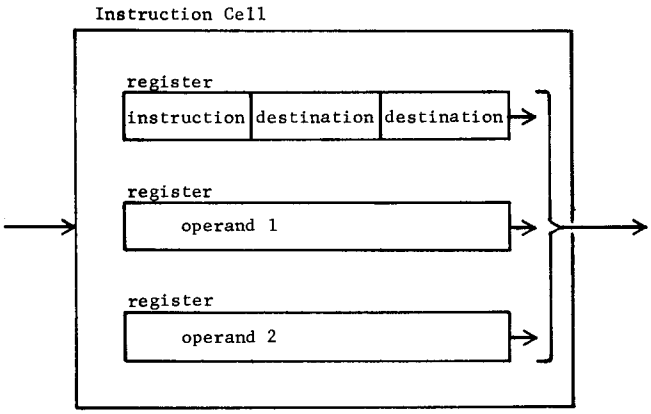


Figure 2. Structure of an Instruction Cell.

## Architecture Description Language

Our architecture description language is a design notation for packet communication systems. The basic unit of description is a module with a number of input ports and output ports. A description module is either a structural description or a behavioral description. A structural description of a module specifies the decomposition of the module into simpler modules and the channels connecting ports of these simpler modules. A behavioral description specifies the module's behavior in the form of a sequentially executed program that: (1) receives packets from a specified input port; (2) transmits packets over a specified output port; or (3) updates state variables of the module. In these respects our language is adapted from the notation used by Rumbaugh to formally describe his data flow multiprocessor [9].

In addition, our description language borrows much of its syntax, type structure and elementary control structure from PASCAL [11]. An information packet or a state variable is defined as a PASCAL record whose components are individually accessible. Packet type information is included in the specification for each channel connection, and for each input port and output port declation, permitting the support software for the simulation facility to enforce strong type checking throughout the hierarchical description of a system.

The overall architecture of the data flow processor is specified in the description language module Processor shown in Figure 3. Processor contains a list of submodules and a list of interconnections. The interface assumed for each submodule is given by the type of information packet which may be transmitted over its input and output ports. The relevant packet definitions

```
Processor: module (m: integer, n: integer);

structure:
Cell [1..m]: module
     distnet-in input port,
     arbnet-out output port;
Arbitration-Network: module (m, n)
     cell-in [1..m] input port,
     fcn-unit-out [1..n] output port;
Functional-Unit [1..n]: module
     arbnet-in input port
     distnet-out output port;
Distribution-Network: module (m, n)
     fcn-unit-in [1..n] input port,
     cell-out [1..m] output port;

Cell [1..m] . arbnet-out send operation-pkt
     to Arbitration-Network . cell-in [1..m];
Arbitration-Network . fcn-unit-out [1..n] send operation-pkt
     to Functional-Unit [1..n] . arbnet-in;
Functional-Unit [1..n] distnet-out send result-pkt
     to Distribution-Network.fcn-unit-in [1..n];
Distribution-Network . cell-out [1..m] send result-pkt
     to Cell [1..m] . distnet-in;

end Processor;
```

Figure 3. Top level description of the data flow processor.

```
address = record [cell-id: integer; register-id: integer];

operation-pkt = packet [opn: opcode;
                        destination: array [1..2] of address;
                        opd: array [1..2] of operand];

result-pkt    = packet [cell-id: integer;
                        register-id: integer;
                        opd: operand];
```

Figure 4. Packet definition.

for Processor are presented in Figure 4. The specification of the data types opcode and operand depends on the kind of computation to be implemented on the data flow processor and is not given in Figure 4. A complete specification of the data flow processor is obtained by supplying description modules for Cell, Arbitration-Network, Function-Unit and Distribution-Network. Each of these description modules must satisfy the interface requirements set forth in the definition of Processor and must implement the operation of the corresponding unit of the data flow processor as outlined above.

In illustration of the technique for specifying the behavior of a module, a specification of the operation of the module Cell is given in Figure 5. Cell communicates with the other submodules of Processor via its input port distnet-in and its output port arbnet-out. Packets of type result-pkt and operation-pkt are received and transmitted by Cell at distnet-in and arbnet-out respectively. The state variables of Cell provide storage for packets received and store state information for controlling the operation of Cell. The state variables are initialized and reset as necessary from one cycle of operation of Cell to the next. The when statement in Cell (Figure 5) is activated upon receipt, at distnet-in, of a result packet which delivers an operand to the instruction held in Cell. When all the required operands are available, an operation packet is formed and emitted at arbnet-out by the send statement (Figure 5). A when statement contains one or several blocks of statements, one block for processing the input packets arriving at each input port. The complete execution of a when statement embodies: (1) receiving and acknowledging an input packet from one of the input ports monitored by the when statement, and (2) executing the block of statements for processing input packets arriving at the input port.

The specifications of Processor and Cell illustrate the descriptive power of the architecture description language. Other submodules of Processor can be similarly defined. After presenting the hardware facilities in the next section, we will describe the implementation of the module Cell as a program executed on the processor modules.

```
Cell: module
     distnet-in input port receives result-pkt,
     arbnet-out output port sends operation-pkt;

behavior

/* State Variables */
respkt : record result-pkt;
operation: opcode;
dest1, dest2: address;
operand1, operand2: operand;
opd1-expected, opd2-expected: boolean;
opd1-received, opd2-received: boolean;

repeat begin
     opd1-received := if opd1-expected then false else true;
     opd2-received := if opd2-expected then false else true;
     while ¬ opd1-received ∨ ¬ opd2-received do
          when distnet-in receives respkt do
               case respkt · register-id of
               1: begin
                    if opd1-received then error;
                    opd1-received := true;
                    operand1 := respkt · value end;
               2: begin
                    if opd2-received then error;
                    opd2-received := true;
                    operand2 := respkt · value end;
               endcase;
     send [opn: operation;
          destination[1]: dest1; destination[2]: dest2;
          opd[1]: operand1; opd[2]: operand2]
     at arbnet-out;
end repeat;

end Cell;
```

Figure 5. Specification of the operation of an Instruction Cell.

## Organization of the Simulation Facilities

The simulation facility shown in Figure 6 is composed of a host computer, a number of microcomputer modules each consisting of a microprocessor and a number of memory modules, a control bus for host-microcomputer communication, and a Routing Network for transmitting packets between microcomputer modules. The host computer loads simulation programs into microcomputer modules, monitors and controls the progress of a simulation, and collects statistical data for performance evaluation. The control bus transmits commands, addressing information and data from the host to the microcomputer modules, and transmits acknowledge signals and memory word contents from the microcomputer modules to the host. Under control of the host, microcomputer modules execute programs which simulate the operation of units of a simulated system. In addition to communicating with the host via the control bus, each microcomputer module is connected by an input port and an output port to the Routing Network, through which the module sends or receives packets from other modules.

The Routing Network provides a buffered path between every pair of microcomputer modules, permitting the transmission of packets without regard for whether the destination processor is ready to receive them. A packet transmitted to the Routing Network from a microcomputer consists of a destination address for the packet and the packet content. The destination address is used by the Routing Network to direct the packet to the input port of the appropriate microcomputer module. The Routing Network performs arbitration and distribution functions in a manner similar to that described for the Arbitration and Distribution Networks in [2].

Before we describe the structure of the commands issued by the host and the various schemes by which the

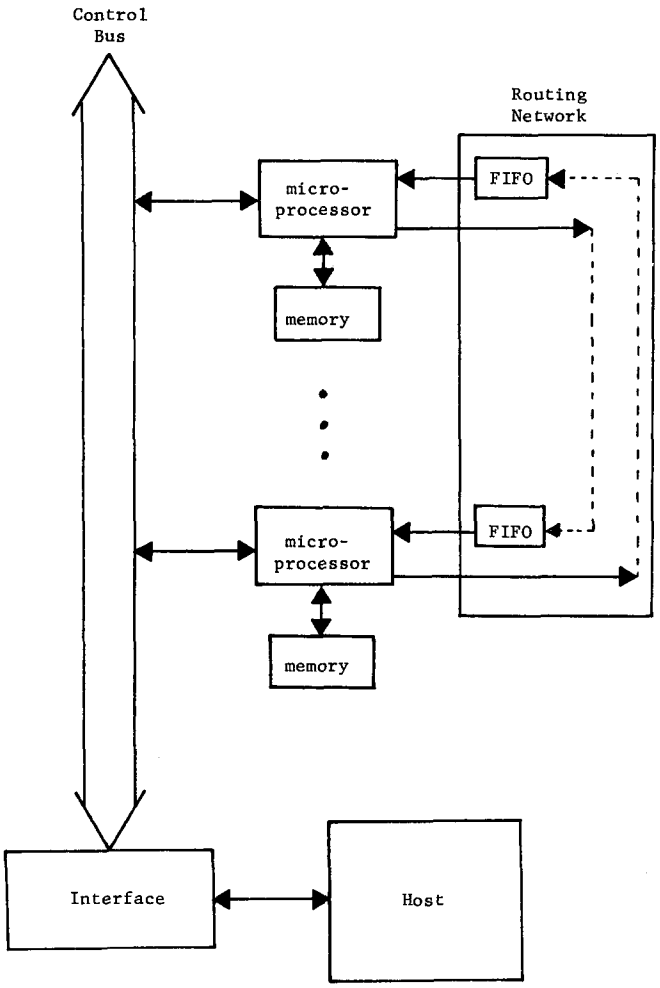


Figure 6. Organization of the simulation facility.

host controls a simulation, let us examine the mechanisms available for controlling a microcomputer module in more detail. The simulation program module contained in each microcomputer module is organized so program execution starts from a home state and returns to this home state after each transaction, that is, after the complete processing of a packet. Each microcomputer module also has a wait state in which no instructions are executed and control of the internal busses of the microcomputer module is relinquished to the host.

Two special registers in each microcomputer module, the run count and the wait flag, are set by the host and utilized to control the progress of a simulation. The run count of a microcomputer module is set by the host to the desired number of cycles of operation of the simulated unit for the current simulation. Each time a cycle of operation is completed, the run count is decremented. If the decremented run count is zero, the microcomputer module enters a wait state and signals to the host that it has entered that state. A negative run count enables a microcomputer module to process transactions until halted by the host. The wait flag is set by the host when it is desired that the designated microcomputer module(s) enter the wait state. The flag is checked by a microcomputer module when the module is in the home state. Hence, microcomputer modules placed in the wait state through setting of the wait flag have no partially completed transactions, and all state variables of the modules are in a consistent state.

The host performs its control functions by issuing commands to the microcomputer modules via the control bus. Commands issued by the host are either addressed or universal. A universal command is obeyed by all microcomputer modules, and such commands are used by the host to start, stop, and temporarily suspend the execution of a simulation. An addressed command is executed only by a designated microcomputer module. Each command transmitted over the bus consists of a selection code and a command name. The selection code specifies which of the microcomputer modules is to respond to the associated command. Each microcomputer module examines the selection code of each command to determine whether the module should respond to it.

The host can issue one of nine commands to a microcomputer module. The possible commands are Read, Write, Hold, Release, Halt, Enable, Clear, Start and Reset. The Read and Write commands provide the capability to examine or alter the contents of a memory module associated with a microcomputer module. The other commands are used in the selection of a microcomputer module for execution of a Read or Write command, or for controlling the progress of a simulation.

Often, it is desired that several, but not all, of the microcomputer modules respond to a Write command simultaneously, for example, when loading a simulation program into a number of microcomputers which are to simulate identical units. This function is accomplished by individually issuing Enable commands to the desired processors. Commands issued subsequently are executed by all enabled processors until a Clear command is received from the host. Note that the Clear command can be either addressed or universal.

The Start, Hold, Release, Halt and Reset commands are used to implement the various schemes by which the host controls a simulation. All microcomputer modules of the system are initially in the wait state. A simulation is initiated by a universal Start command which places all microcomputer modules in their home states. A simple scheme to halt a simulation is to issue a universal Hold command which halts program execution in all microcomputer modules immediately. The host is then free to read or write into the memory modules by issuing Read and Write commands. Program execution at each microcomputer module can be restarted at the point of interruption by issuing a universal Release command.

All microcomputer modules can be put into their wait states simultaneously and immediately by issuing a Reset command. However, when the microcomputer modules are to be stopped for the purpose of debugging and evaluation, all modules should be in consistent states. This is accomplished through the use of a universal Halt command. Execution of the Halt command sets the wait flag of each microcomputer module by generating a universal Hold command followed by a universal Write into the wait flags, and then a universal Release. Each microcomputer module, upon reaching its home state, then discovers that its wait flag is set, enters its wait state, and signals the host. When the host has received an acknowledge signal designating that each microcomputer module has entered its wait state, it can examine and alter the memory contents of any microcomputer module, and it can examine the status of each microcomputer input port to see if there are any packets present.

Once a simulation has been halted and the status of the facility has been determined, one or several microcomputer modules can be enabled for a specified

number of transactions by properly setting their run counts, setting the wait flags of the other microcomputer modules and then issuing a universal Start command. Receipt of the Start command causes each microcomputer module to exit its wait state and reenter its home state. The microcomputer modules whose run counts were set will accept packets at their input ports. All others will immediately reenter their wait states.

An active microcomputer module will signal the host computer after completing the specified number of transactions. The acknowledge signals from the microcomputer modules are ANDed and ORed to produce a Universal Acknowledge and an Addressed Acknowledge, indicating that the appropriate processors have responded to a universal or addressed command.

The various control schemes and communication protocols presented provide a minimal capability for controlling and examining system operation during a simulation. The fact that the host can readily access the individual memory modules allows one to easily extend the control, analysis and debugging capabilities in software. Each microcomputer module can store any desired status information in its memory for the host to retrieve, even to the point of retaining all packets processed by the module.

An example of a software evaluation facility is the evaluation of performance of individual sections of a simulated processor through analysis of event counts. An event count is a count maintained by an individual microcomputer of the number of transactions which have taken place since initiation of a timing interval. The use of event counts allows the study of the relative efficiency of sections of the simulated processor and provides data necessary for determining such parameters as cache size and structure of the memory/processor interconnection networks.

## Simulation of a Packet Communication System on the Hardware Facility

A packet communication system is simulated on the hardware facility through simulation of one or more units of the system on each microcomputer module. The constructs used in the simulation programs are implemented on a microcomputer module in a straightforward manner. The implementation of packet transmission and processing, the identification of microcomputer states during program execution and the coordination between packet processing and microcomputer state transitions are further illustrated in this section using the module Cell (Figure 5) as an example.

In general, a unit simulated on a microcomputer module may have several input ports. A separate input buffer is allocated in memory for each input port of the simulated unit. Every packet transmitted through the Routing Network specifies a target port, which is an input port of a simulated unit. A microcomputer module, upon receipt of a packet, uses this target port designation to deposit the packet in one of its input buffers.

The program module Cell has one input port distnet-in. If Cell is the only unit simulated on a microcomputer module, every packet arriving at the input port of the microcomputer is automatically deposited in the buffer associated with distnet-in. Any output packet of the module Cell is transmitted through the output port arbnet-out.

Each microcomputer module is in a wait state after the simulation programs have been loaded. A Start command transfers the microcomputer module from the wait state to the home state, and initiates execution of the simulation program. Unless temporarily halted by a Hold command, the execution of a simulation program on a microcomputer module proceeds until a when statement is reached, at which point the microcomputer reenters its home state. Upon reentering its home state, the module examines its wait flag and enters the wait state if the wait flag has been set by the host. If the wait flag is not set, the microcomputer module queries its wait flag and the status of the input ports monitored by the when statement in turn using a round-robin algorithm, until the wait flag is set or an input packet becomes available. If the wait flag is set, the microcomputer enters its wait state. If an input packet becomes available first, the when statement is executed.

When the program module simulating Cell is executed on a microcomputer, the microcomputer enters its home state each time the when statement (Figure 5) that receives result packets at distnet-in is reached. The run count of a microcomputer module is decremented at the end of each cycle of operation of the simulated unit, and the microcomputer module enters its wait state if the updated run count becomes zero. In the case of Cell, the run count is decremented and examined each time the body of the outermost repeat statement is executed.

## Software Support

The structure of the controlling software system for the simulation facility is presented in Figure 7. Operation of sections of the simulated system is specified by modules in the architecture description language in the manner described earlier. These modules are translated into relocatable microprocessor object

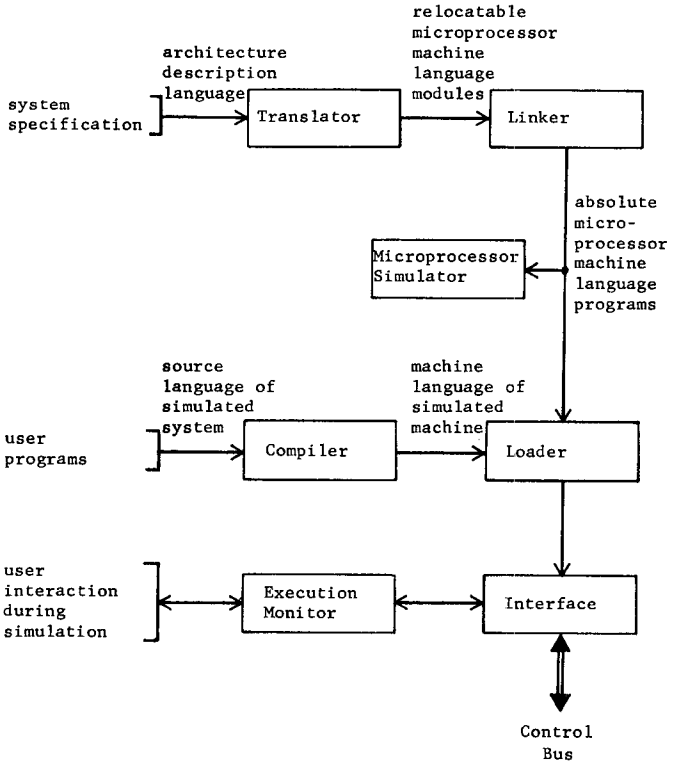


Figure 7. Structure of the simulation control system.

code and are stored in the file system of the host computer; the necessary programs from the file are linked together to form a non-relocatable microprocessor program. Either the individual procedures or a complete simulation program can be tested by use of a microprocessor simulator residing in the host computer. Once the simulation programs have been validated by use of the microprocessor simulator, the programs are loaded into the microprocessors, and the facility is ready to execute a program of the simulated machine.

A user program to be executed on the simulated architecture is compiled into the machine language of the simulated machine and sent to the microprocessor system for execution. The debugging and evaluation capabilities of the system are used to control execution of the program and evaluate feasibility of the proposed system architecture.

## Conclusion

The architecture simulation facility appears to be a powerful tool for the evaluation of packet communication systems. Its capabilities permit the testing and evaluation of a broad range of architectural concepts. The facility is currently under construction using the Motorola M6800 microprocessor and a DEC PDP-11 host computer. Portions of the software system are being developed on a PDP-10 computer to allow use of the language CLU [4, 7]. The system is intended to be used primarily for an investigation of the design and capabilities of data-flow processors, and we expect it to be invaluable for this application.

## Acknowledgements

The authors wish to thank Bob Jacobsen and Dave Isaman for many helpful comments and suggestions.

## References

1. Dennis, J. B. Packet communication architecture. Proceedings of the 1975 Sagamore Computer Conference on Parallel Processing, IEEE, New York, 1975.
2. Dennis, J. B., and D. P. Misunas. A computer architecture for highly parallel signal processing. Proceedings of the ACM 1974 National Conference, ACM, New York, November 1974, 402-409.
3. Dennis, J. B., and D. P. Misunas. A preliminary architecture for a basic data-flow processor. Proceedings of the Second Annual Symposium on Computer Architecture, IEEE, New York, 1975, 126-132.
4. Liskov, B. H., and S. N. Zilles. Programming with abstract data types. Proceedings of ACM SIGPLAN Conference on Very High Level Languages, SIGPLAN Notices 9, 4 (April 1974), 50-59.
5. Misunas, D. P. A Computer Architecture for Data-Flow Computation. S.M. Thesis, Department of Electrical Engineering and Computer Science, M.I.T., Cambridge, Mass., June 1975.
6. Misunas, D. P. Structure processing in a data-flow computer. Proceedings of the 1975 Sagamore Computer Conference on Parallel Processing, IEEE, New York, 1975.
7. Project MAC Progress Report XI, July 1973-1974. Project MAC, M.I.T., Cambridge, Mass., 35-50.
8. Project MAC Progress Report XI, July 1973-1974. Project MAC, M.I.T., Cambridge, Mass., 84-90.
9. Rumbaugh, J. E. A Parallel Asynchronous Computer Architecture for Data Flow Programs. Report TR-150, Project MAC, M.I.T., Cambridge, Mass., May 1975.
10. Rumbaugh, J. E. A data flow multiprocessor. Proceedings of the 1975 Sagamore Computer Conference on Parallel Processing, IEEE, New York 1975.
11. Wirth, N. The programming language PASCAL. Acta Informatica 1 (1971), 35-63.