An Architecture of a Dataflow Single Chip Processor

Shuichi SAKAI, Yoshinori YAMAGUCHI, Kei HIRAKI,

Yuetsu KODAMA and Toshitsugu YUBA

Electrotechnical Laboratory

1-1-4 Umezono, Tsukuba, Ibaraki 305, JAPAN

ABSTRACT

A highly parallel (more than a thousand) dataflow machine **EM-4** *is now under development. The* **EM-4** *design principle is to construct a high performance computer using a compact architecture by overcoming several defects of dataflow machines. Constructing the* **EM-4**, *it is essential to fabricate a processing element (PE) on a single chip for reducing operation speed, system size, design complexity and cost. In the* **EM-4**, *the PE , called* **EMC-R**, *has been specially designed using a 50,000-gate gate array chip. This paper focuses on an architecture of the* **EMC-R**. *The distinctive features of it are: a strongly connected arc dataflow model; a direct matching scheme; a RISC-based design; a deadlock-free on-chip packet switch; and an integration of a packet-based circular pipeline and a register-based advanced control pipeline. These features are intensively examined, and the instruction set architecture and the configuration architecture which exploit them are described.*

1. Introduction

A dataflow architecture is supposed to be the most suitable architecture for highly parallel computers. The reasons are: it can naturally extract the maximum available concurrency in a computation; it is suitable for VLSI implementation since a large number of identical processing elements (PEs) and repetitive data networks can be used in its construction; and dataflow languages provide an elegant solution for writing concurrent programs. There are, however, many technical problems involved in realizing a practical dataflow computer. Feasibility studies in the practical use of dataflow computers are essential.

Several architectures based on the dataflow concept have been proposed[1,2,3,4,5,8,9], some of which have been implemented in experimental machines. Among them the **SIGMA-1** [5], which is a large-scale dataflow supercomputer for numerical computations, shows the possibility to surpass the conventional von Neumann computers. It consists of 128 PEs and has a processing performance of more than 100 MFLOPS. A **SIGMA-1** PE is implemented by several gate array chips and a large memory. To construct a highly parallel machine, direct extension by merely adding more **SIGMA-1** PEs is not practical because the architectural design is complicated and too much hardware is required to implement all the PEs and the network. To realize a highly parallel machine, one or more PEs should be implemented on a single chip and the network structure must be simplified. The total architecture including computation model must be reconsidered.

The **EM-4**[10], whose target structure is more than 1000 PEs, is also being developed at the Electrotechnical Laboratory on the basis of the **SIGMA-1** project. The design principles of the **EM-4** are as follows.

1. Simplify the total architecture of a dataflow machine, e.g. interconnection network with O(N) hardware, a RISC-based single-chip PE design, and a direct matching scheme.
2. Improve machine performance by integrating a packet-based circular pipeline and a register-based advanced control pipeline.
3. Afford versatile resource management facilities, which current dataflow machines do not have, by introducing a strongly connected arc dataflow model.

The **EM-4**'s single-chip processor, called the **EMC-R**, realizes these principles. This paper focuses on the architecture of the **EMC-R**.

Section 2 describes design features of the **EMC-R**. Defects of current dataflow architectures are listed and distinctive features of the **EMC-R** (or the **EM-4**) for conquering them are shown. In section 3, the instruction set architecture of the **EMC-R** which reflects the above principles is described. Features of an instruction set, instruction formats and a packet format are shown there. Section 4 describes the configuration architecture of the **EMC-R** which realizes all of the above.

2. Design Features of the EMC-R

2.1. Defects of Current Dataflow Architectures

In order to design an efficient parallel machine, we closely examined the defects of current dataflow architectures. They are summarized as follows.

D1. A circular pipeline [4] does not work well as a "pipeline" for less parallel execution.

This is because current dataflow execution models have no advanced control mechanism. In the case of highly parallel execution ($\geq N \times S$, where N is the number of PEs and S is the number of pipeline stages in each PE), all the stages of a circular pipeline can be filled with tokens. In other cases, it may occur that only one token is going round the pipeline cycle, and that PE throughput is less than one per a pipeline circulation time.

D2. Simple packet-based architecture cannot exploit registers or a register file efficiently.

If you always realize a token as a packet, and if you make each of the packets enter a PE whenever possible, it is nonsense to reserve tokens in registers for the future node operation. This is one of the main reasons why a fine pitch pipeline is difficult to implement in a dataflow machine.

D3. Time complexity and hardware complexity for matching are heavy if you adopt the colored token style.

Color matching needs special hardware like associative memory or hashing hardware. They both require complex control logics and matching takes considerable time.

D4. Packet flow traffic is too heavy.

With a simple packet architecture, packet flow traffic is too heavy and a high-bandwidth low-delay interconnection network must be implemented. However, network performance is limited by current device technology.

D5. Current dataflow concepts cannot provide flexible and efficient resource management mechanisms.

If you realize mutual exclusion for resource management in a dataflow machine, you should provide some serialization mechanism (e.g. waiting queue). Its implementation is difficult, however, because the access to a concerned section can be violated by anyone else.

D6. It takes much time to eliminate garbage tokens.

If a program uses switch operations for conditional computations, there may occur a lot of garbage tokens. At the end of the execution, they must be collected to allow the working space to be reused. Time overhead for such operations is usually considerably large.

To overcome these defects, several novel facilities and mechanisms are introduced in the **EMC-R**, which are described in the following subsections.

2.2. Strongly Connected Arc Model

Although the basic **EM-4** architecture is based on the dataflow model, a new model called a *strongly connected arc model* [7] has been introduced to compensate for the pure dataflow architecture. The strongly connected arc model can solve all the problems described in the previous subsection.

In this model, dataflow graph arcs are divided into two categories: the normal arcs and the strongly connected arcs. A dataflow subgraph whose nodes are connected by strongly connected arcs is called a strongly connected block. The control strategy of the modified dataflow model in which this model is introduced is that the operation nodes are executed exclusively in a strongly connected block. Figure 1 shows an example of strongly connected blocks. In this figure, there are two strongly connected blocks, A and B. When node 5 or node 6 is fired, block A or block B is executed exclusively. This has the effect of giving the nodes in a strongly connected block the highest priorities if nodes are executed depending on priorities. A strongly connected block acts as a macro node which includes several instructions and is executed as if it were a single basic node. In the **EM-4**, each strongly connected bock is executed by a single PE.

There are several advantages in the strongly connected arc model, which are shown below.

A1. This model makes it easy to introduce an advanced control pipeline to dataflow architecture, because the exclusion of the outer block instructions causes more deterministic execution of codes. This solves the problem of **D1**.

A2. Instruction execution cycles can be reduced by introducing a strongly connected register file which is used for storing tokens in a strongly connected block. This is possible as the registered data in the concerned block are not violated by any other data. This overcomes the defect **D2**.

A3. In an intra-block execution, matching can be realized much more easily and efficiently because it is not necessary to match function identifiers (i.e. colors) if the block is included in the function. This solves the inefficiency problem of **D3**.

These **A1**, **A2** and **A3** enable the **EMC-R** to have a fine pitch execution pipeline.

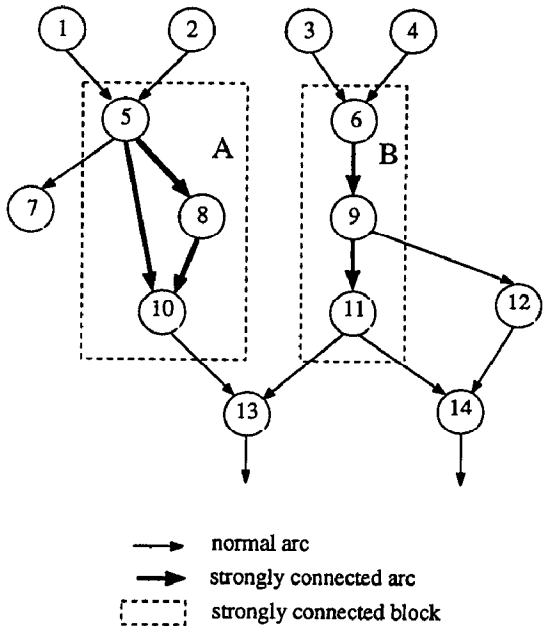


Figure 1 An Example of a Strongly Connected Block.

A4. There are no packet transfers in a strongly connected block. This reduces the defect **D4**.

A5. It can provide flexible resource management facilities by constructing an indivisible instruction sequence (e.g. test and set). This overcomes the defect **D5**.

A6. It can simplify the problem of remaining garbage tokens. This is because only the flag resets of a strongly connected register file are needed for eliminating the garbage tokens. The operation can be performed simultaneously with the result data transfer, i.e. with no overhead. This reduces the defect **D6**.

2.3. Direct Matching Scheme

To remove the defect **D3**, a fast and simple data matching scheme is needed. A strongly connected model solves this problem within a block but matching overhead on a normal dataflow node is not solved. We have designed a simple new data matching scheme, called a *direct matching scheme*. This scheme is implemented using ordinary memories. Since the logic for realizing the scheme is fairly simple, it can be easily implemented by wired logic on the **EMC-R**.

When a function is invoked, an instance of storage is allocated to a group of PEs. This instance is called an *operand segment*. It is used for waiting and matching of operands. An operand segment has memory words whose number is equal to or larger than that of two operand instructions in the concerned function. The compiled codes for the function are stored in another area, which is called a *template segment*. The address of the instruction in the template segment has a one-to-one simple correspondence with that of the matching. Binding of an operand segment and a template segment is also performed at the function invocation time.

The matching is executed by checking the stored data in an operand segment and by storing new data if its partner is absent. The matching location for each dataflow node is uniquely given as an absolute memory address in an operand segment. Thus matching can be carried out without using the associative memory or a hashing mechanism.

2.4. Processor Connected Omega Network

The **EM-4** uses a processor connected omega network as its interconnection network. Figure 2 illustrates an example of its topology. The advantages of this network are: the average distance from any PE to any other PE is order log(N), while N is the total number of PEs; the number of connection links from a PE is a small constant even if there are many PEs; and total number of switching elements is O(N) which is smaller than that of a multi-stage network (O(Nlog(N))).

The precise routing algorithm of the network will be reported in another paper.

The **EMC-R** contains an element of the processor connected omega network. One reason for this is to reduce the packet transfer time between a switch and a PE. The other reasons are the low hardware cost and the design simplicity. This element and processing function can work independently and concurrently.

3. Instruction Set Architecture

3.1. RISC Architecture

We adopt a RISC architecture for the **EMC-R** for simplicity and execution efficiency. Current dataflow architectures are not suited to RISC because defects **D2, D3,** and **D4** in 2.1 obstruct its implementation. The **EMC-R** is suited to RISC for its features in 2.2 and 2.3. It can exploit a fine pitch pipeline, each stage of which is simple. It has a register file whose member is the strongly connected register described in 2.2.

The **EMC-R** is a dataflow RISC chip due to the following features: it has only 26 instructions; there are only four kinds of instruction formats; it has only two memory addressing modes; it has a register file; it uses no microprograms; its packet size is fixed; there are only a few packet types; and it has simple synchronization mechanisms (direct matching and a register-based sequencing).

Among them, the latter three should be regarded as the features of a *RISC PE for a parallel computer.*

In the following subsections, the features of **EMC-R** instruction set architecture are described.

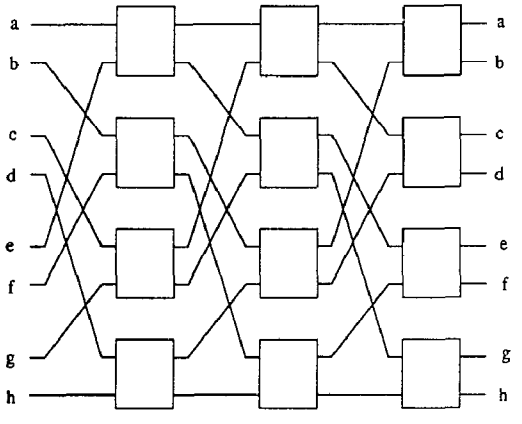


Figure 2 Processor Connected Omega Network.

3.2. Instruction Set

Table 1 shows an instruction set of the **EMC-R**. It has twenty-six kinds of instructions, each of which is executed in a single clock cycle (except continuous-memory-word-access instructions).

Details of an operation are afforded by the **AUX** field. For instance, the **SHF** instruction left or right shifts according to the contents of the **AUX** field.

In the **EMC-R**, a complex instruction can be performed by a strongly connected bock which contains simpler instructions. This is called a *macro instruction*. For instance, an integer division operation, a function call operation and complex structure operations are provided as macro instructions.

The following instructions are the characteristic instructions of the **EMC-R**.

(1) Branch Instructions

In the **EMC-R**, an action of a data switch operation in a strongly connected block is to fire one of the adequate nodes of its destination, without flowing any data. For this reason, the word **BRANCH** is used instead of **SWITCH** in the **EMC-R** for representing a data switch. There are six **BRANCH** instructions as shown in the Table 1. In order to simplify sequencing, they are implemented in a delayed branch style.

The **EMC-R** can provide a normal dataflow switch by strongly connecting the **BRANCH** instruction and the **MKPKT** instructions described below. In a good program, however, almost all of the switches are realized in a strongly connected block, because a packet flow overhead and a garbage token collection overhead are removed with this method (see 2.2).

(2) MKPKT

In the **EMC-R**, all of the instructions, except those of memory access and branch instructions, can make a packet for sending their results. In addition, it has a **MKPKT** instruction dedicated to packet

Table 1 Instruction Set

CATEGORY	INSTRUCTION	ACTION
Arithmetic and Logic	ADD	integer add
	SUB	integer subtract
	MUL	integer multiply
	DIV0	preparation of division
	DIV1	element of division
	DIV2	correction of division 0
	DIVR	remainder of division 0
	DIVQ	quotient of division 0
	SHF	shift
	AND	bitwise AND
	OR	bitwise OR
	EOR	bitwise exclusive OR
	NOT	bitwise NOT
	ALUTST	ALU test
Branch	BEQ	branch by equality
	BGT	branch by greaterness
	BGE	branch by greaterness or equality
	BTYPE	branch by data type
	BTYPE2	branch by 2 data types
	BOVF	branch by overflow
Memory or Register Read or Write	L	load from memory
	S	store to memory
	LS	load and store from/to memory
	LDR	load from register
Others	GET	send packet for remote operation
	MKPKT	make packet by two operands

generation. The **MKPKT** sends a packet whose address part is its first operand and whose data part is its second operand. Continuous **MKPKTs** perform the efficient distribution of the same data. Moreover, **MKPKT** supplies two special operations: an inter-function data transfer and a global data transfer such as a structure data transfer.

(3) GET

GET is an instruction for remote operations. It sends a return address to the address represented as its operand. For instance, CAR, CDR and sending a return address of a function are performed by the **GET**.

3.3. Instruction Format

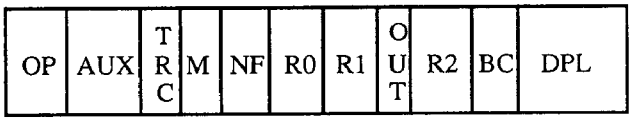
The **EMC-R** has four types of instruction formats. Two of them are shown in Figure 3 (the other two are the immediate attached ones). Both of them are stored in a single memory word where each memory word is 38 bits long.

The **OP** field holds an operation code. **AUX** is a secondary field of the **OP**. If the concerned strongly connected block ends with the next instruction execution, then the **M** field (mode field) contains zero. **R0** and **R1** are the strongly connected registers used for the *next* instruction, if this instruction is strongly connected to other instructions. In this way, register-based advanced control is implemented in the **EMC-R**.

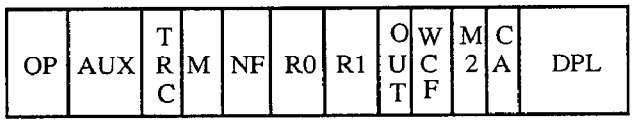
OUT is a tag field indicating whether the instruction generates a packet or not. If OUT is zero (Figure 3 (a)), then no packets are generated and the result is stored in **R2**. In this case, **BC** is a branch condition field which describes a branching style and **DPL** contains the displacement of a branching address. The latter two fields are used only in branch instructions.

If the **OUT** field contains one (Figure 3 (b)), then a packet is generated. The address part of the output packet is made up of the **WCF**, **M2**, **CA**, and **DPL** fields, and the operand segment identifier of the concerned function. The data part of the packet is the operation result.

A typical instruction execution is illustrated in more detail in 4.1 and 4.2.



(a) Without Immediate, without Packet Output (OUT = 0)



(b) Without Immediate, with Packet Output (OUT = 1)

Each figure is the field size.

Figure 3 Two Typical Types of Instruction Format.

3.4. Packet Format

A typical packet format is illustrated in Figure 4. Each packet consists of an address part and a data part, both of which have 39 bits.

(1) Address Part

HST field indicates whether this packet is bound for a normal destination or a host. **PT** is a packet type field. **WCF** is a waiting condition flag field, which indicates a type of matching. **M** is also a flag which indicates a type of dataflow arc. If it is zero, then the packet will fire a normal node; otherwise it will fire a strongly connected block. **(GA, CA, MA)** are the destination address fields. **GA** is a destination PE group address, **CA** is its column (i.e. member) address, and **MA** is a memory address. If this is a normal data packet, then **MA** is the matching address.

(2) Data Part

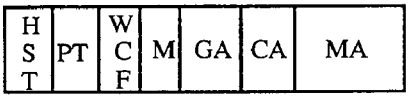
C is a cancel bit field indicating that the packet is a nonsense packet. **DT** and **D** are the data type and the data, respectively, which this packet carries.

3.5. Special Packets

A normal data packet has a NORM packet type, but packets for function control, structure access, remote memory access, etc. have special types. These special packets have no operand segment number, i.e. they are colorless. A special packet is generated by a **MKPKT** instruction or by a **GET** instruction. It is executed by a special strongly connected block named **SP Monitor**. A system manager can set the **SP Monitor** in any way desired during the system initialization, so the effect of a special packet can be properly determined by the manager. Thus the special packet execution in the **EMC-R** is completely flexible.

3.6. Program Examples

Figure 5 illustrates two FIBONACCI programs. Figure 5 (a) is a pure dataflow program and Figure 5 (b) is a program with strongly connected blocks. In the latter one, type checking and data switching



(a) Address Part

(b) Data Part

Each figure is the field size.

Figure 4 Typical Packet Format.

```
FIB:    (COPY   <EQ1 imml> <SW1 r>)
EQ1:    (EQ '1 <SW1 l>)
SW1:    (SWITCH (T <CNST imml>)
          (F <EQ2 imml> <SW2 r>))
EQ2:    (EQ '2 <SW2 l>)
SW2:    (SWITCH (T <CNST imml>)
          (F <SUB1 imml> <SUB2 imml>))
CNST:   (COPY '1 <RETV r>)
SUB1:   (SUB '1 <CALL0 s>)
SUB2:   (SUB '2 <CALL1 s>)
```

```
CALL0:  {*GCALL 'FIB <ADD l>}
CALL1:  {*GCALL 'FIB <ADD r>}
ADD:    (ADD <RETV r>)
RETV:   {*PASSVAL <KILL s>}
KILL:   {*KILL}
```

(a) A Normal Dataflow Program

```
FIB:    [BEQ '1 (LEAF) (R0:NL)]
        [SUB '1 R1 (R0:NL) (NL:NL $)]
        [BEQ '2 (LEAF) (R0:NL)]
        [SUB '2 R2 (SEG:R1) (NL:NL $)]
        [MKPKT <CALL0 s> (SEG:R2 $)]
        [MKPKT <CALL1 s> (NL:NL)]
LEAF:   [MKPKT '1 <RETV r> (NL:NL)]
```

```
CALL0:  {*GCALL 'FIB <ADD l>}
CALL1:  {*GCALL 'FIB <ADD r>}
ADD:    (ADD <RETV r>)
RETV:   {*PASSVAL <KILL s>}
KILL:   {*KILL}
```

(b) A Strongly Connected Dataflow Program

Figure 5 FIBONACCI Program.

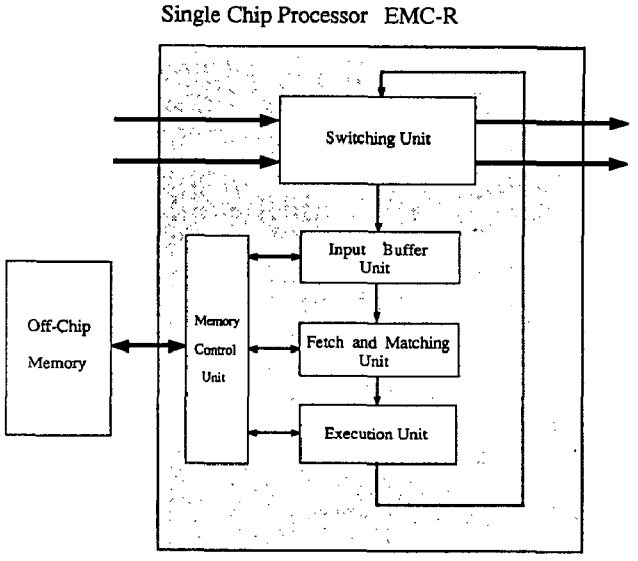


Figure 6 Block Diagram of the EMC-R.

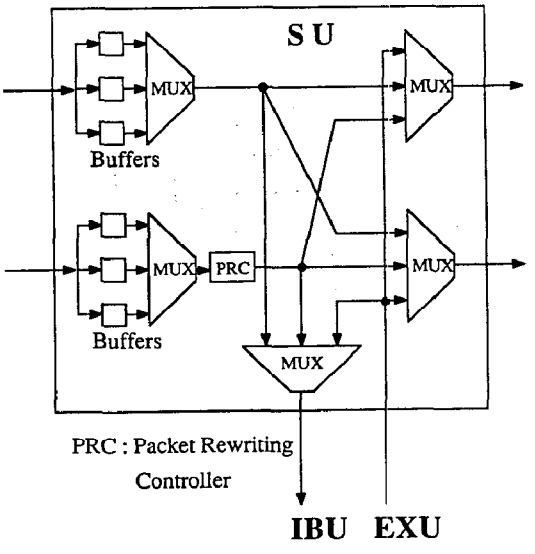


Figure 7 Switching Unit Organization.

are strongly connected to a single block illustrated as a rectangle in Figure 5 (b). It takes twenty-three clocks to execute the rectangular part if you select the former program (in the case of occurring recursive calls). If you select the latter program, it only takes nine clocks. This means that the strongly connected method can execute this subprogram about two and half times as fast as the pure dataflow method.

Remark that, in the above comparison, a **SWITCH** instruction and a **COPY** instruction were supposed to exist in the **EMC-R**. And remark that all the normal nodes of the **EMC-R** are executed in the same clock cycles with the pure dataflow machine **SIGMA-1**.

4. Configuration Architecture

4.1. EMC-R Architecture

Figure 6 shows a block diagram of the **EMC-R** which realizes all of the features described in the previous sections. The **EMC-R** consists of a Switching Unit (**SU**), an Input Buffer Unit (**IBU**), a Fetch and Matching Unit (**FMU**), an Execution Unit (**EXU**), a Memory Control Unit (MCU), and Maintenance Circuits.

(1) Switching Unit

The Switching Unit (**SU**) is a three-by-three packet switch which is an element of a processor connected omega network. It switches data independently of and concurrently with the other units. Each input port of the network has structured buffers. Organization of the **SU** is illustrated in Figure 7.

When a processor connected omega network is used in a buffered manner, store-and-forward deadlock prevention facilities must be supplied. In the **EMC-R**, a three bank buffer is provided in each input port. Firstly, a packet is buffered in the least level bank. When the packet arrives at the zeroth stage of the network, it is pushed into the one-upper-level bank. Because of the topological property of the processor connected omega network, any packet never covers three rounds of this network, so the logical structure of a packet transfer cannot make loops. This three bank strategy thus removes all store-and-forward deadlocks from the network.

Another feature of the SU is the function level dynamic load balancing facility. In the **EM-4**, special packets which monitor the load of PEs travel through the network. The SU rewrites these packets within the time period required by a normal packet transfer, i.e. with no overhead, and performs the load balancing. This rewriting is made by the **PRC** in Figure 7.

(2) Input Buffer Unit

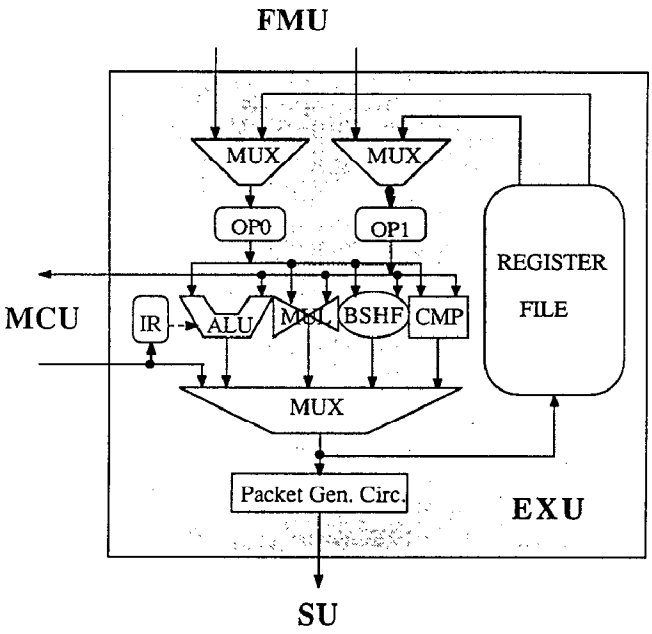
The Input Buffer Unit (**IBU**) is a buffer for packets waiting for execution. A 32-word FIFO type buffer is implemented using a dual port RAM on chip. If this buffer is full, a part of the off-chip memory is used as a secondary buffer.

(3) Fetch and Matching Unit

The Fetch and Matching Unit (**FMU**) is used for matching tokens and fetching instructions. It performs a direct matching for a packet and a sequencing for a strongly connected block. It controls the pipelines of the processor, especially integrating two types of pipelines (see 4.2). The **FMU** contains an instruction address register, a packet data register, a register for matching data, several multiplexers, and control circuits.

(4) Execution Unit

The Execution Unit (**EXU**) is an instruction executor. Figure 8 illustrates its organization. The **EXU** contains an instruction register, two operand registers, a register file, an ALU, a barrel shifter, a multiplier, a versatile comparator, packet generation circuits, and control circuits.



IR : Instruction Register OPi : Operand Register i

Figure 8 Execution Unit Organization.

In an execution cycle, the contents of operand registers are sent to the ALU, etc. Then the operation is carried out according to the **OP** and **AUX** fields of the instruction register. The result is sent in a packet or stored in a register file. All of these actions are made in a single clock, and, in the same clock, fetch and decode of the next instruction and data load from the **FMU** or a register file are performed (see 4.2).

(5) Memory Control Unit and Off-Chip Memory

The Memory Control Unit (**MCU**) arbitrates memory access requests from the **IBU**, the **FMU**, and the **EXU**, and sends data between the off-chip memory and the **EMC-R**. The **MCU** consists of a data multiplexer, an address multiplexer, and an arbitration controller.

The off-chip memory size can be up to 5 megabytes. It is used for a secondary packet buffer, a matching store, an instruction store, an area for the SP monitor (see 3.5), a structure store, and a working area.

(6) Maintenance Circuits

The Maintenance Circuits make initialization of the chip and memory words, handle many kinds of errors and provide the dynamic monitor which reports a system performance and the system status such as processor load, an active function number and structure area used. We will report the concept and the construction method of the Maintenance Circuits in another paper.

4.2. Pipeline Organization

Figure 9 illustrates the pipeline organization of the **EMC-R**. Basically, the pipeline has four stages, some of which have two substages. Each stage has a rectangle which represents a single clock action, or a bypass line which means no clock action. A small rectangle represents a half clock action.

Each packet from a network is buffered in the **IBU** if necessary (the far left side of the figure). Then the concerned template segment number (see 2.3) is fetched from the off-chip memory in the first stage (**TNF**). The number is stored at the top of the operand segment at the function invocation time. The first stage is bypassed when the packet is not a normal data packet.

The second stage is the matching stage. In the case of matching with the immediate, an immediate fetch (**IMF**) occurs. In the case of matching with data in the matching store, data at the concerned address is read in the former half clock (**RD**). If a partner exists, the flag at the address is eliminated in the latter half clock (**EL**); else a new packet data is written in the latter half clock (**WR**). This read-modify-write action is completed in a single clock cycle. The second stage is omitted if a new token fires a single operand instruction.

The third stage is an instruction fetch (**IF**) and decode (**DC**) stage. Each operation is performed in a half clock.

The fourth stage is an execution stage. Transfer of the result packet can be overlapped with the execution. If the next instruction is strongly connected with the current instruction, instruction fetch and data load of the next instruction are overlapped with the execution. The third stage and the fourth stage are repeated in the overlapped manner until there are no executable instructions in the concerned strongly connected block. If the instruction is a normal mode instruction or the last instruction of the strongly connected block, its execution can be overlapped with the instruction fetch and decode of a next packet processing. Thus, an integration of two types of pipelines, a packet-based circular pipeline (illustrated as thin lines in Figure 9) and a register-based advanced control pipeline (illustrated as thick lines in Figure 9), is realized.

During each stage, the **TNF**, the **IMF**, the **RD**, the **WR**, the **EL** and the **IF** are performed by the **FMU** with the assistance of the **MCU**. The **DC** and the **EX** are performed by the **EXU**.

The register-based advanced control pipeline is a fast and simple pipeline exploited by strongly connected blocks. Its throughput is at most six times as high as that of the packet-based circular pipeline.

Each PE has a peak processing performance of more than 12 MIPS.

4.3. Implementation

The **EMC-R** is a real processor chip, so its implementation is limited by current chip fabrication technology. Our chip contains 50,000 CMOS gates and 256 signal lines.

Table 2 shows the gate usage and pin usage of each **EMC-R** unit. The Switching Unit is complex as it has three bank buffers and their multiplexer at each port. The Execution Unit is also complex because it has many large modules such as a register file, a multiplier, a barrel shifter, and a packet generator. The Fetch and Matching Unit requires a little hardware, because the direct matching reduces the hardware cost of it.

As for pins, almost all of them are used for data buses of the network and the off-chip memory.

The **EMC-R** will be fabricated by June 1989. Then the **EM-4** prototype which has 80 PEs will be constructed. It will consist of 16 processor boards, each of which will have five PEs and a mother board which the network will be implemented on. The **EM-4** prototype hardware will be operational in 1990. Peak performance of this prototype is expected to be more than 1 GIPS. Construction of an efficient real machine of 1,000 PEs is the next program, which is the goal of the **EM-4** project.

Table 2 Hardware Complexity

UNIT	Gates	Pins
Switching Unit	10,112	176
Input Buffer Unit	9,238	-
Fetch and Matching Unit	3,504	-
Execution Unit	19,692	-
Memory Control Unit	1,518	67
Maintenance Circuits	1,589	12
Total	45,653	255

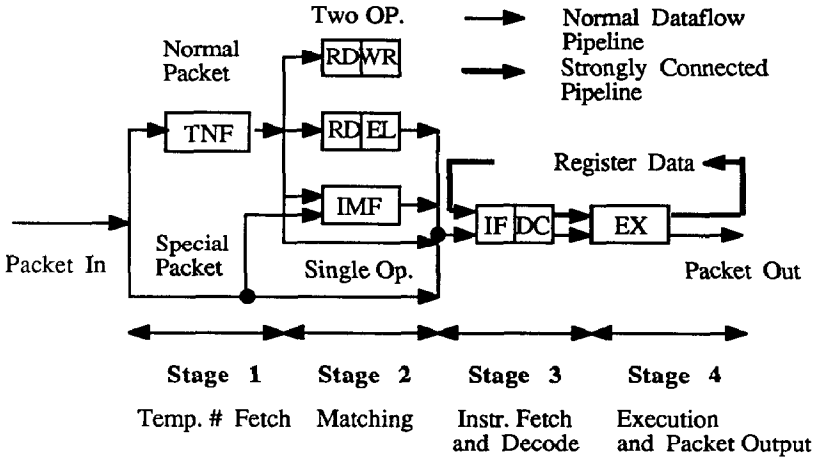


Figure 9 Pipeline Organization of the EMC-R.

5. Conclusion

This paper describes the architecture of the EMC-R, a single-chip dataflow processor which is a PE of the EM-4. The distinctive features of the EMC-R are:

(1) a strongly connected arc dataflow model;
(2) a direct matching scheme;
(3) a RISC-based design;
(4) a deadlock-free on-chip packet switch; and
(5) an integration of a packet-based circular pipeline and a register-based advanced control pipeline.

These features were examined, and the instruction set architecture and configuration architecture which exploit them were described.

The schedule of the hardware implementation is written in 4.3. As for softwares, a high level language and an optimizing compiler are currently under development. The latter has a node labeling scheduler for the intra-function load balancing [6] and a block constructor for automatically making strongly connected blocks. The optimization schemes and algorithms will be reported in another paper.

Future problems are as follows.

(1) Close consideration and expansion of a strongly connected arc dataflow model.
(2) Consideration of a chip design using much highly-integrated VLSI.

Acknowledgement

We wish to thank Dr. Hiroshi Kashiwagi, Deputy Director-General of the Electrotechnical Laboratory, Dr. Akio Tojo, Director of the Computer Science Division and Mr. Toshio Shimada, Chief of the Computer Architecture Section for supporting this research, and the staff of the Computer Architecture Section for the fruitful discussions.

References

[1] Amamiya,M., Takesue,M., Hasegawa,R. and Mikami,H.: Implementation and Evaluation of a List-Processing-Oriented Data Flow Machine, Proc. the 13th Annu. Symp. on Computer Architecture, pp.10-19 (June 1986).

[2] Arvind, Dertouzos,M.L. and Iannucci,R.A.: A Multiprocessor Emulation Facility, MIT-LCS Technical Report 302 (Sep. 1983).

[3] Dennis,J.B., Lim,W.Y.P. and Ackerman,W.B.: The MIT Dataflow Engineering Model, Proc. IFIP Congress 83, 553-560 (1983).

[4] Gurd,J., Kirkham,C.C. and Watson,I.: The Manchester Prototype Dataflow Computer, Commun. ACM, 21, 1, pp.34-52 (1985).

[5] Hiraki,K., Sekiguti,S. and Shimada,T.: System Architecture of a Dataflow Supercomputer, TENCON87, Seoul (1987).

[6] Otsuka,Y., Sakai,S. and Yuba,T.: Static Load Allocation in Dataflow Machines, Proc. of Technical Group on Computer Architecture, IECE Japan, CAS86-136, in Japanese (1986).

[7] Sakai,S., Yamaguchi,Y., Hiraki,K. and Yuba,T.: Introduction of a Strongly-Connected-Arc Model in a Data Driven Single Chip Processor EMC-R, Proc. Dataflow Workshop 1987, IECE Japan, pp.231-238, in Japanese (1987).

[8] Shimada,T., Hiraki,K., Nishida,K. and Sekiguchi,S.: Evaluation of a Prototype Data Flow Processor of the SIGMA-1 for Scientific Computations, Inf. Process. Lett.,16, 3, pp.139-143 (1983).

[9] Yamaguchi,Y., Toda,K. and Yuba,T.: A Performance Evaluation of a Lisp-Based Data-Driven Machine(EM-3), Proc. 10th Annual Symposium on Computer Architecture, pp.163-169 (1983).

[10] Yamaguchi,Y., Sakai,S., An Architectural Design of a Highly Parallel Dataflow Machine, to appear in IFIP'89 (1989).