How many operation units are adequate?

Wolfgang Matthes, 1991

A manuscript in a tentative stage.
The finished paper has been published in
Computer Architecture News, June 1991.

Abstract

A uniprocessor superscalar architecture is proposed which comprises four universal operation units arranged according to a tree-shaped dataflow graph, instruction issuing hardware, and operand selection means. The control principles are based on VLIW, microprogramming, and dataflow concepts. The proposal emerged mainly from investigations of inherent mathematical structures of application problems, especially from the analysis of dataflow graphs of elementary mathematical formulas (arithmetic of intervals, complex and rational numbers etc.). The particular operation unit itself is an ensemble of high-performance processing resources which may be compared to state-of-the-art processors (e. g. i860). It may require a silicon budget from one to five million transistors. The whole processor may require 10 to 50 million transistors, thus being a suitable implementation target for IC technologies of the 90's.

1. Introduction

To make use of the inherent parallelism in ordinary programs requires the availability of more than one processing resource to perform the desired operations. An obvious approach is to provide multiple operation units. This is known as the superscalar approach. Besides, multiple resources could be created by dividing an operation resource vertically into pipelined segments so that multiple operations can flow through in a step-by-step fashion (superpipelining approach). Both principles can be combined (superscalar/superpipelined machines; JO88).
In this paper a particular superscalar architecture will be proposed (with obvious possibilities to add superpipelining, too).
The main objective of the underlying research activities was to develop a performance-optimized uniprocessor architecture which should be

1) a versatile, powerful, and cost-effective ensemble of processing resources,

2) an advantageous implementation target for IC technologies of the 90's,

3) a suitable processing element for high-performance and massively parallel systems, according to the good old principle of engineering sciences, to optimize the components first before implementing them in cost-intensive technologies or assembling them together in large quantities.

2. Known Structures - an Overview

Superscalar machines can be built as ensembles of different operation units (integer add, integer multiply, floating point add, multiply etc.; the CDC 6600 [THO64] is a well-known example).
Evidently, an ensemble of universal operation units, each of which is capable of performing all the operations specified in the architecture, will provide more opportunity to exploit the available inherent parallelism. Hence we will concentrate on such structures.
In a rough taxonomy, known structures could be divided into two categories:

1) tree-like dataflow connection structures,

2) crossbar-like connection structures.

The Figures 1,2 show two structures of the first category. The first structure [WU83] was derived from the observation, that many operation sequences have the form

(a OP1 b) OP2 c,

with the SAXPY (linked triad) d(i)= (a * b(i)) + c(i) being a well-known example.
The second structure (proposed for a GaAs microprocessor; VLA88) had been based on the following empirical realization: Application programs can be divided into those with a low amount of calculations and those with an extraordinarily high amount. Programs belonging to the first category have in approximately 93% of all assignments no arithmetic operands or only one arithmetic operand. Calculation-intensive programs have in approximately 93% of all assignments up to 3 operands (see Figure 3).
Figure 4 shows the principial structure of a superscalar machine with crossbar-like connections. The obvious advantage is the unlimited universality which is not restricted by a particular scheme of data flow. On the contrary, crossbar networks are cost-intensive and may lead to a slower machine cycle. This scheme is typical of VLIW architectures [COL87, CO88], sometimes with the modification of separate register files, crossbars, and operation units for both integer/logical and floating point operations.

An important hardware viewpoint deserves consideration: the information paths in dataflow schemes are point-to-point connections which can be kept short. Tree-shaped structures have the additional advantage that the connections do not cross each other. Hence they are better suited for integration than totally universal connection schemes (crossbar or bus structures).

3. Foundations for Developing a New Architecture

The present superscalar architectures had been developed on the basis of comprehensive analytical work. The experiences had been gathered essentially by measurements of the frequency of usage of operations and operation sequences in comprehensive samples of application programs.

Such a measurement-oriented approach [MA86] may lead to considerably good machines, but it has two obvious drawbacks:

1) The rate of usable inherent parallelism is disappointingly low. In the literature, the recommended rate depends on the semantic level at which the investigations had been done. A lower level means less usable parallelism. If only the instruction level is considered, it has shown strong evidence, that it makes no sense to provide more than two operation units in parallel [JO89, SM89]. On the contrary, investigations at Fortran source code level had promised rates of usable parallelism from 16 to more than 128 [KU74].

2) Machine architectures derived from these data reflect current programming habits. Possible opportunities for further innovation may remain undiscovered.

Hence our approach is not to study programs, but to study the underlying mathematical structures or, in more general terms, the deep structures, the essence of important application problems (i. e. semantic levels above the programming languages).

Example:
In many numerical applications it is possible to execute both integer and floating point operations in parallel. This fact had been applied to some architectures (e. g. Trace, Transputer T 800, i860). But what is the essential cause behind this empirical observation? For what reason can integer units be kept busy in loops processing floating point data? – Evidently, the integer operations are necessary to do the address calculations for array element addressing. From this realization we can draw a significant advantage: We can sub-optimize the integer units. For example, we may restrict the number of universal integer units to one or two, and additionally we may provide some units specialized to address calculations (as many as needed to feed the floating point units). Thus we may exploit more of the inherent parallelism and keep cost comparatively lower.

To obtain initial empirical data for a new proposal we have simply browsed some collections of formulas.[1] Figures 5-8 show some frequently needed mathematical operations together with the corresponding dataflow graphs. We realize only two essential interconnection structures:

1) none, i. e. independently operating units,

2) tree-shaped structures.

A more elaborate bookkeeping of the resources needed (skipped here for sake of brevity) will show that four operation units may be used efficiently (i. e. they may be kept busy in nearly the whole time). Calculations whose dataflow graph comprises more than four nodes are to be executed in more than one processing step. Hence some bypass and local storage means have to be provided.

---

1 Of course, this simple method cannot substitute comprehensive investigations. But it is sufficient to demonstrate the feasibility of our approach.

4. The Proposed Structure

According to Figures 9,10 the proposed structure comprises four universal operation units and (not explicitely shown) a minimum/maximum and a delta detector. The operation units form a tree structure with four operand data paths from memory and one result path to memory. The structure is an extension of the structure according to Figure 2 by an additional unit whose 2nd operand input is attached to a stack-like accumulating memory or accumulator register, respectively. A stack-organized accumulating memory of sufficient capacity may be used as the runtime data stack (at least as a stack cache; DI87). Hence the problem how ordinary programs can exploit the tree structure efficiently may be reduced to the problem of tree height reduction.
The memory subsystem, the instruction issuing and control mechanisms, and address calculation means are implemented in additional hardware which will be explained below.
The structure according to Figures 9,10 is based on the assumption that a tree-like dataflow scheme will be more significant in a true universal processor than independent operation of the four units. Hence only the tree connections are provided in hardware to keep interconnection cost as low as possible. This approach requires some bypass provisions to feed the operation units 3,4 with control information, and, if operated independently, with memory data.
On the other hand, this cost/performance tradeoff will cause effectivity losses, if vectorized or unrolled code is to be executed. As a matter of routine, to allow for independent operation of four units would require at least 12 memory ports (4*2=8 to fetch the operands, 4 to store the results), and additional interconnection means would be necessary to implement the tree-shaped structure. The urge for cost reduction led to the extension of the proposed structure shown in Figure 11. Each of the operation units has a multipurpose memory (MPM) which can be used as an accumulator, a stack, a collection of vector registers, and a control storage. It has two independent ports for read and write accesses, respectively. Its capacity should be at least 8 kBytes, organized as 1024 buckets of 128 bits (if used as a vector register, it could hold two vectors of 1024 64-bit elements). The whole structure is connected to the memory subsystem via eight ports (four read-only and four read/write ports, the latter are used to provide the paths of the tree-shaped structure as well). This scheme allows to load and store the MPMs at maximum speed. Each of the operation units can execute even triadic operations (e. g. SAXPY) with two of the operands delivered via memory ports and one from the MPM. The results will be stored in the MPMs. They can be moved to the memory at maximum speed after the operations have been completed.

5. The Internal Structure of an Operation Unit

Each of the four operation units can process numerical and nonnumerical data, respectively.
The internal structure of a processing kernel is shown in Figure 12. In principle, some state-of-the-art high-performance processors may serve as a paradigm for processing kernel design (e. g. TMS 34082, Motorola DSP960002, i860, AMD 29000), and Figure 12 shows nothing but an ensemble of processing resources a high-performance machine should

have, according to today's knowledge. Compatibility to existing architectures was not our concern. Instead, we tried to put as many innovative ideas as possible in our proposal. Here are some of these concepts:

1) All data structures are packed in buckets (machine words) of 128 bits. In the hardware, a bucket can be divided in bags of 64, 32, 16, or 8 bits.

2) In the buckets, arbitrary bit fields can be selected.

3) The bit field is the basic type for nonnumerical data. Normally, such data structures are packed in bags (8-64 bits). In some cases, the bags of a bucket may be processed in parallel (scanning of character strings, graphics operations etc.).

4) For numerical data, there is only one basic type: the binary coded natural number. Arbitrary bit fields can be treated as natural numbers. They will be processed in multiples of 32 bits (with appropriate extension before processing, if necessary). All other numerical data types are extensions of this concept:
Integers are naturals extended by a sign bit (sign/magnitude representation in contrast to the usual two's complement representation).
Floating point numbers are composed of an integer mantissa and an integer exponent (fixed formats of 32, 64, and 96 bits).
BCD coded decimal numbers are not provided. Decimal numbers can be represented as rational numbers (fractions) of the form a/b.
The merits of this proposal have yet to be proved. But there are some obvious advantages:

a) For each type of operation, only one type of hardware resource is necessary.

b) Floating point operations could be controlled up to the elementary level (microcode level; DA89). High accuracy algorithms (e. g. an accurate scalar product; KU81) could be implemented efficiently. If desired, extremely long integers could be used instead of floating point numbers for intermediate variables within high-accuracy calculations.

c) BCD hardware can be avoided. Binary rational number arithmetic can use the tree structure efficiently (see Figure 7). This promises to be considerably faster than the usual nibble-by-nibble BCD arithmetic.

Of course, the machine should be compatible to wide-spread basic data structures (2's complement integers, bytes etc.). But conversion (e. g. 2's complement to sign/magnitude representation and vice versa) could be done on the fly and requires considerabely less hardware than independent resources for each data type.

To each of the basic operations, one dedicated hardware resource is assigned. Some resources could be operated in parallel, but this kind of parallelism has been restricted to keep cost down (e. g. in the numerical section, only multiply-add dataflow has been provided). Exponent calulations are performed in dedicated hardware. Special

circuitry has been provided for data conversion (unpacking of stored data into the internal representation and vice versa). This circuitry (in Figure 12: Argument selection/alignment) consists mainly of barrel shifters which can be exploited for multiple functions (bitfield extraction/insertion, floating point mantissa shifting etc.).

Since none of the resources and operations is completely new, estimations of expenditures can be based on known high-performance processors. For example, the i860 [INT90] comes very close to our proposal, including 8 kBytes on-chip memory, 64-and 128-bit data paths, multiply-add chaining in the FP unit, graphics operations, and integer multiplications done within the FP multiply hardware. The i860 requires slightly more than one million transistors. Thus we can estimate to implement an operation unit with a silicon budget between one and five million transistors, depending on particular cost/performance tradeoffs.

6. The Processor Structure

The overall processor structure, which contains the described ensemble of four operation units as a subsystem, is shown in Figure 13. The basic steps of the instruction processing are assigned to dedicated hardware resources:

- instruction issuing (control memory, Common Control),
- operand selection (Selector/Iterator Resources Ensemble, references/data memory),
- execution of operations (Processing Resources Ensemble, i. e. the structure of operation units described above).

For efficient operand selection, adequate hardware is provided to keep the operation units busy nearly the whole time. This hardware is responsible for machine word (bucket) addressing and for elementary address calculations .
Bitfield selection is done within the operation units. More complicated address calculations are executed by the operation units, too. The Selector/Iterator Resources Ensemble is provided to produce addresses according to access patterns [JE88] which are typical of many kinds of innermost loops. Hardware implementation of such access patterns allows for simple and efficient circuitry. An example is shown in Figure 14. This hardware structure is able to produce address values to access array structures from one to three dimensions. To formulate such an access pattern in a common programming language requires nested DO-loops, e. g.:

```
for AD3 = 1 to EC3 do
    for AD2 = 1 to EC2 do
        for AD1= 1 to EC1 do

.....calculations using variables Vi (AD1,AD2,AD3)....

    end;
       end;
          end;
```

The usual way to calculate the address of an array element Vi(AD1,AD2,AD3) is to apply the formula

ADDRESS(Vi) = ARRAY_BASE + AD1 + (AD2-1)*EC1 + (AD3-1)*EC1*EC2.

(EC1,2,3 represent the element count of first, second, and third dimension, respectively.) The iterator hardware avoids address calculations in the loop body. It is effective in parallel to the operation units. Multiply operations are only required for the set-up of the hardware (offset calculations) prior to loop execution.

The memory subsystem in Figure 13 is conceptually located outside of the processor. It must provide the necessary access paths as well as appropriate storage capacity (many Megabytes) and access bandwidth. In addition to this, it must provide the propagation of dataflow control information (see below). The control memory and the references/data memory are located inside of the processor. They may be used as instruction and data caches according to the well-known principles (e. g. set-associative access), but in our proposal the use of these memories should be controllable by software directly. The control memory contains the last recently used programs. It will be loadad via the four read-only ports. The preferred use of the references/data memory is to hold reference information of the last recently used programs (access descriptors), constants, and intermediate variables. Four references can be processed in parallel. This memory can exchange data with the memory subsystem and with the operation units. To avoid confusion, some details should be mentioned:

1. The Processing Resources Ensemble in Figure 13 corresponds to Figures 11,12, and the processor is designed with cost-effectivity in mind, thus multiple use of the data paths is necessary.

2. The references/data memory contains bidirectional bypasses for the read/write memory ports (5-8). The Processing Resources Ensemble has no access to this memory except for address calculations.

3. The memory subsystem contains dataflow-controlled bypasses from the ports 5-8 to the ports 1-4, thus data out of the reference/data memory can reach the operation units 1,2 via the corresponding ports.

4. If the Processing Resources Ensemble according to Figure 9 is to be chosen (low-cost alternative), then all internal memories are connected to the four read-only ports, and the write port from operation unit 4 is fed back to the references/data memory which in turn has write ports to the memory subsystem and may be used like a conventional data cache.

To give a rough estimation, the processor may be implemented within a silicon budget of 50 million transistors:

a) Memories: 2 Memories, each has 4 banks of 4 k buckets (total 16 k buckets = 256 kBytes = 2 Mbit). 2 Mbit*2 = 4 Mbit; 6 transistors/bit: 24 million transistors (+ address decoders etc.),
b) Common Control, Selector/Iterator Resources Ensemble, bypasses, glue and driver circuitry: 1-2 million transistors,
c) 4 Operation units: 4...20 (4*1...4*5) million transistors.

Further cost/performance tradeoffs may reduce the transistor budget to the 10 millions range (dynamic memory cells, less memory, off-chip memory, operation units with less or performance-reduced resources).

8. Control Principles

A combination of VLIW, microprogramming, and dataflow control principles is employed. The instruction formats are based on 128-bit buckets. Basically, the following types of control information are provided:

1. Resource Control Words (RCWs). RCWs are similar to horizontal microinstructions. An active RCW controls the information flow of the corresponding unit in the current machine cycle. RCWs can be executed out of the control memory or (in the operation units) out of the MPM.

2. Incarnation Control Words (ICWs). ICWs are used to control resources according to dataflow principles. Examples of such information structures are shown in Figure 15. An ICW is composed at least of a Resources Selection Bucket (RSB) and an Argument Selector Bucket (ASB). Thus resources control and argument selection are isolated from each other. A RSB contains 8 code fields. A code field can select a particular resource in a particular unit and a argument selector field out of the ASB. There is no operation code. Instead, the resources are identified by ordinal numbers, and the selected arguments will be delivered to the selected resource. E. g. to multiply two numbers in a particular operation unit requires feeding the arguments to the multiplicand and multiplicator registers in the desired unit. The appropriate control information accompanies the data. This control information is packed into 32-bit Resource Selection Words (RSWs) by the processor's Common Control circuitry. The RSWs must be propagated via the memory subsystem to appear together with the data buckets at the operation units. Proper synchronization is achieved by an order number in each RSW, which is generated by Common Control. A particular resource will be active only if all corresponding arguments have the same order number. The result will be forwarded with the same or with an advanced order number, according to the Advance Control bit in the RSW. Obviously, it poses no principial difficulties to introduce superpipelining by inserting appropriate pipeline stages into the processing resources.

9. Conclusion

An overview of a proposal for a high-performance uniprocessor architecture has been given. Of course, many details and intricacies had to be skipped, and a lot of research work remains to be done. Obviously, the following problems deserve special interest:

- refinements of the dataflow control principles, especially with respect to branch and start-up latencies,
- evaluation of each innovation against well-introduced principles,
- migration paths from or even compatibility to systems representing de-facto standards,
- compiler-related issues.

10. References

CO89 R. Cohn et al., "Architecture and Compiler Tradeoffs for a Long Instruction Word Microprocessor," SIGARCH Computer Architecture News Vol. 17, No. 2, pp.2-14, April 1989.

COL87 R.P. Colwell et al., "A VLIW Arcitecture for a Trace Scheduling Computer," Operating Systems Review Vol. 21, No. 4, pp. 180-192, October 1987.

DA89 W.J. Dally, "Micro-Optimization of Floating-Point Operations," SIGARCH Computer Architecture News Vol. 17, No. 2, pp. 283-289, April 1989.

DI87 D.R. Ditzel, "Design Tradeoffs to Support the C Programming Language in the CRISP Microprocessor," Operating Systems Review Vol. 21, No. 4, pp. 158-163, October 1987.

IN88 INMOS Ltd. Transputer Reference Manual. 1988.

INT90 Intel Corporation. i860 64-bit Microprocessor Programmers Reference Manual. 1990.

JE88 Y. Jegou, "Access Patterns: A Useful Concept in Vector Programming," Supercomputing. 1st International Conference Athens, Greece, June 1988 Proceedings, pp. 377-391 (Springer 1988, LNCS Vol. 297).

JO88 N.P. Jouppi, "Superscalar vs. Superpipelined Machines," SIGARCH Computer Architecture News Vol. 16, No. 5, pp. 71-80, November 1988.

JO89 N.P. Jouppi, D.W. Wall, "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines," SIGARCH Computer Architecture News Vol. 17, No. 2, pp. 272-282, April 1989.

KU74 D.J. Kuck et al., "Measurements of Parallelism in Ordinary FORTRAN Programs," Computer Vol. 1, No. 1, pp. 37-46, January 1974.

KUL81 U.W. Kulisch, W.L. Miranker. Computer Arithmetic in Theory and Practice. Academic Press, 1981.

MA86 M.J. Mahon et al., "Hewlett-Packard Precision Architecture: The Processor," Hewlett-Packard Journal, August 1986, pp. 4-22.

SM89 M.D. Smith et al., "Limits on Multiple Instruction Issue," SIGARCH Computer Architecture News Vol. 17, No. 2, pp. 290-302, April 1989.

SO89 G.S. Sohi, S. Vajapeyam, "Tradeoffs in Instruction Format Design for Horizontal Architectures," SIGARCH Computer Architecture News Vol.17, No. 2, pp. 15-25, April 1989.

THO64 J.E. Thornton, "Parallel Operation in the Control Data 6600," AFIPS Proc. FJCC, Part 2, Vol. 26, pp. 33-40, 1964.

VLA88 H. Vlakos, V. Milutinovic, "GaAs Microprocessors and Digital Systems. An Overview of R&D Efforts," IEEE Micro Vol. 8, No. 1, pp. 28-56, February 1988.

WU87 Wm. A. Wulf, "The WM Computer Architecture," SIGARCH Computer Architecture News Vol. 15, No. 4, pp. 70-84, September 1987.

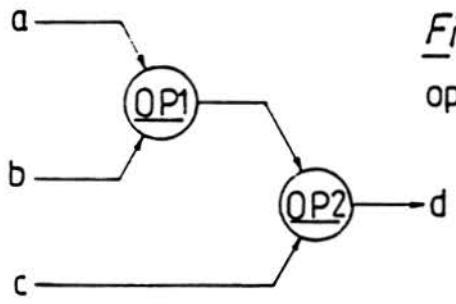


Figure 1: Structure of two operation units (WU87).

d := (a OP1 b) OP2 c

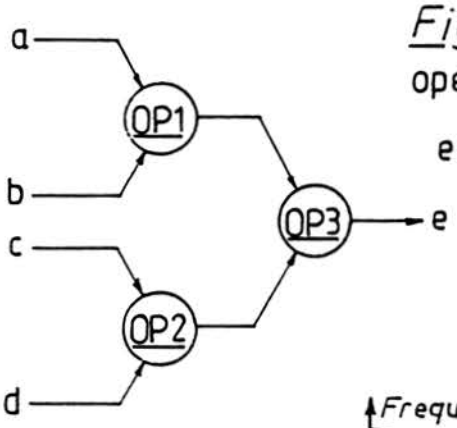


Figure 2: Structure of three operation units (VLA88).

e := (a OP1 b) OP3 (c OP2 d)

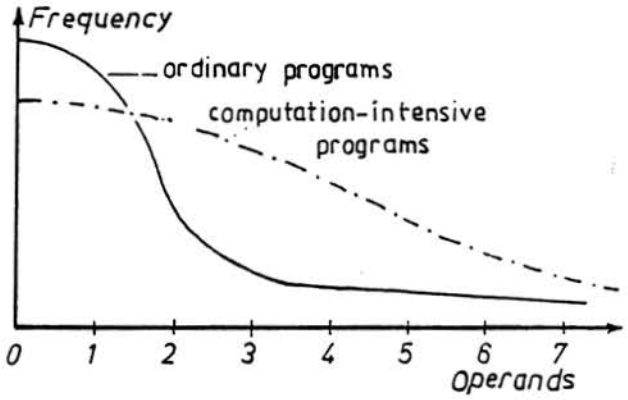


Figure 3: Frequency of the number of operands in arithmetic assignments (VLA88).

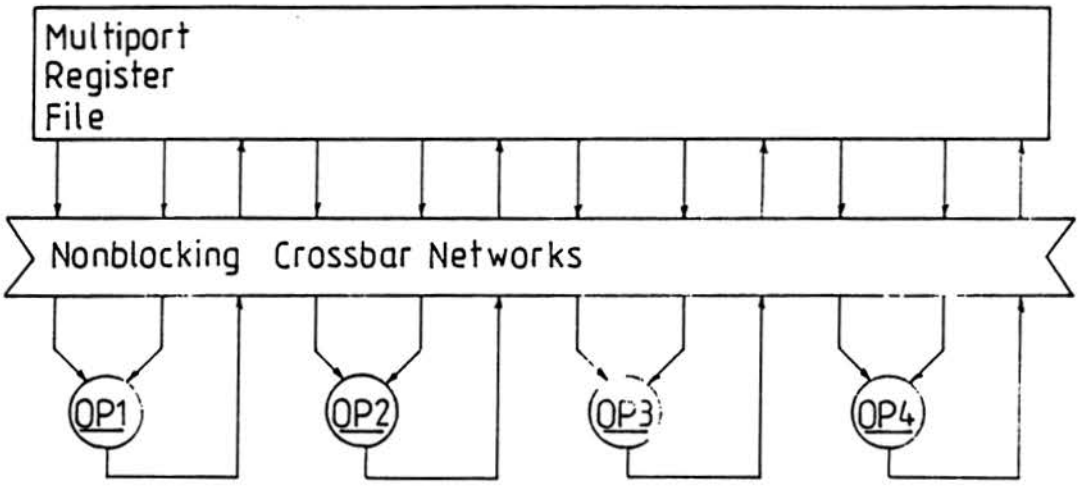
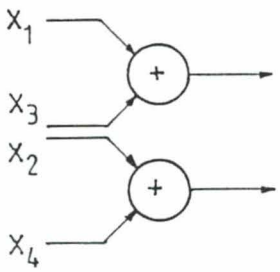


Figure 4: Crossbar connection structure in a superscalar (VLIW) machine.

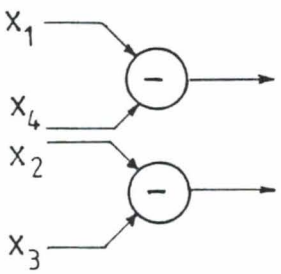
Addition

$$[x_1, x_2] + [x_3, x_4] = [x_1 + x_3, x_2 + x_4]$$



Subtraction

$$[x_1, x_2] - [x_3, x_4] = [x_1 - x_4, x_2 - x_3]$$



Multiplication

$$[x_1, x_2] * [x_3, x_4] = [\min(P), \max(P)] \; ;$$

with $P = (x_1 * x_3, x_1 * x_4, x_2 * x_3, x_2 * x_4)$

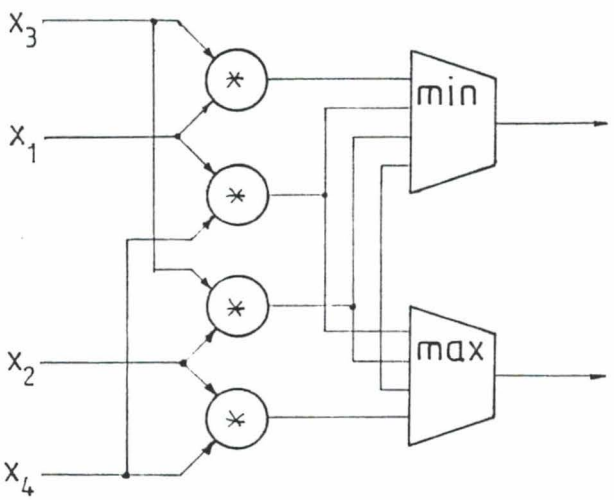
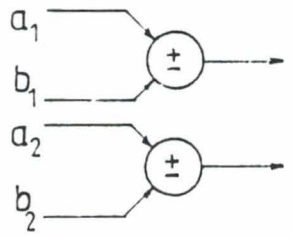


Figure 5: Examples of elementary interval arithmetic operations.

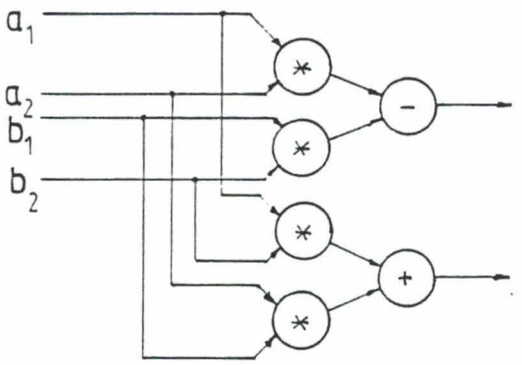
Addition/subtraction

$(a_1, b_1) \pm (a_2, b_2) = (a_1 \pm a_2, b_1 \pm b_2)$



Multiplication

$(a_1, b_1) * (a_2, b_2) = (a_1 a_2 - b_1 b_2 , a_1 b_2 + a_2 b_1)$



Division

$$(a_1, b_1) / (a_2, b_2) = \left[\frac{a_1 a_2 + b_1 b_2}{a_2^2 + b_2^2} , \frac{a_2 b_1 - a_1 b_2}{a_2^2 + b_2^2} \right]$$

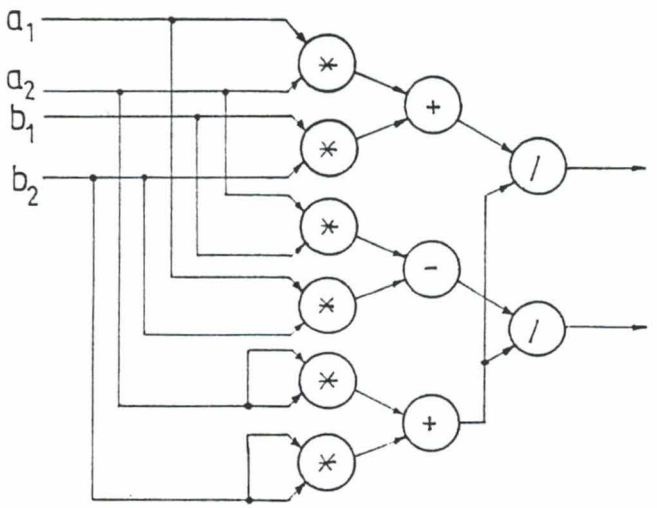
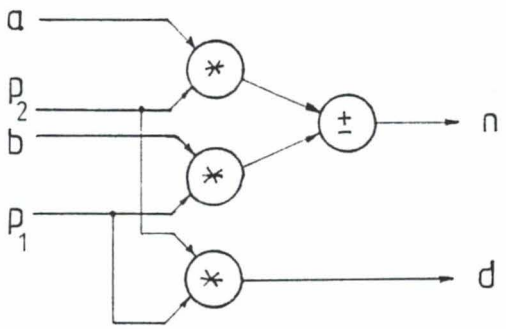


Figure 6: Examples of elementary complex number arithmetic operations.

Addition/subtraction

$$\frac{a}{p_1} \pm \frac{b}{p_2} = \frac{a p_2 \pm b p_1}{p_1 p_2}$$

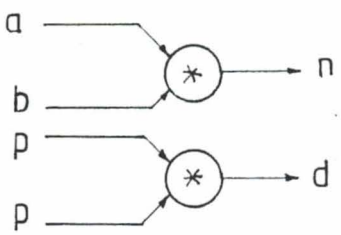


n: result numerator

d: result denominator

Multiplication

$$\frac{a}{p_1} * \frac{b}{p_2} = \frac{a b}{p_1 p_2}$$



Result rounding (according to a specified precision p)

$$\frac{r}{p} \stackrel{!}{=} \frac{n}{d} \; ; \quad r = \frac{n p}{d}$$

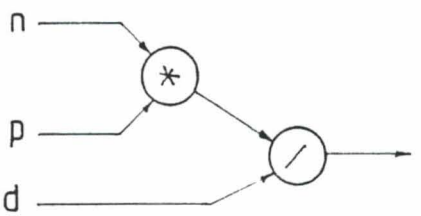


Figure 7: Examples of elementary rational number arithmetic operations.

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

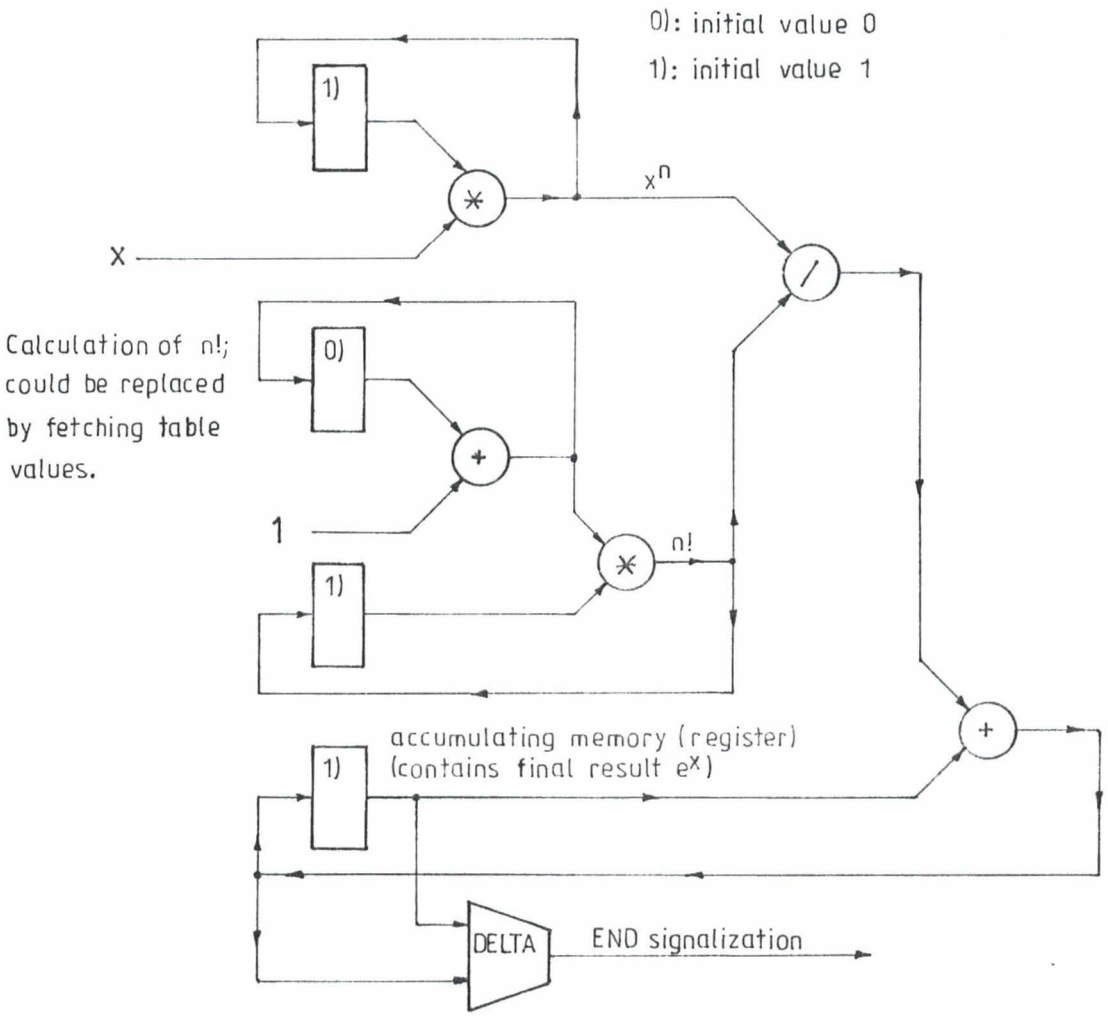


Figure 8: Series expansion example.

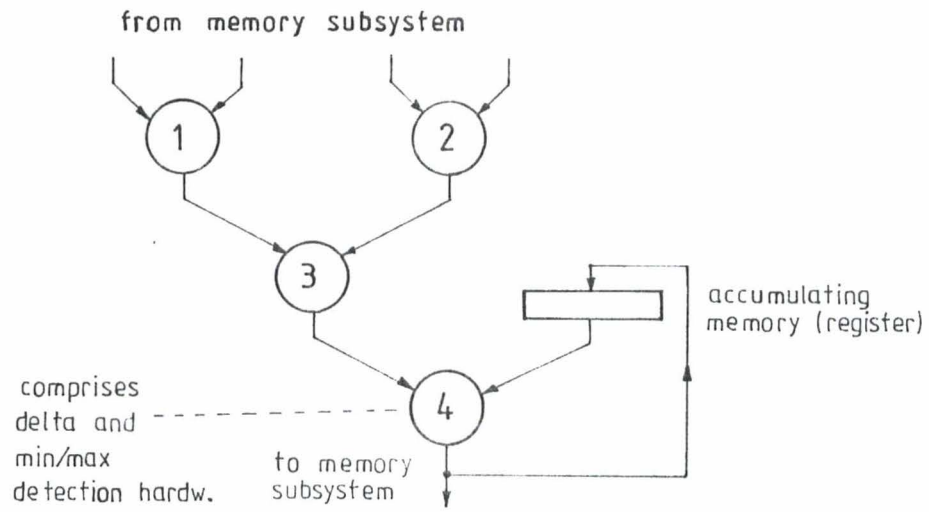
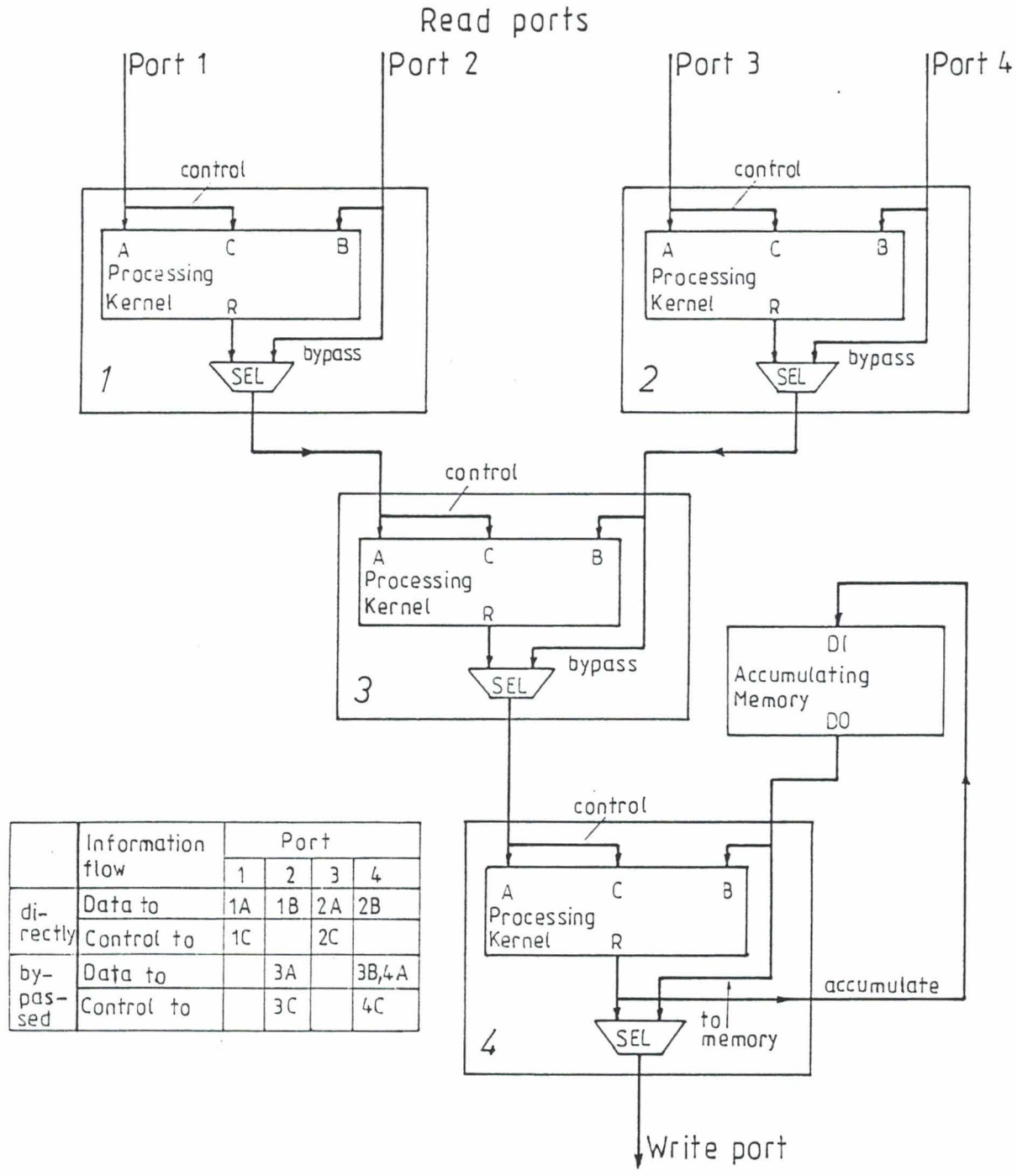


Figure 9: The proposed structure of four operation units (at first glance).



	Information flow	Port			
		1	2	3	4
directly	Data to	1A	1B	2A	2B
	Control to	1C		2C	
by-passed	Data to		3A		3B,4A
	Control to		3C		4C

Figure 10: The proposed structure: detailed view.

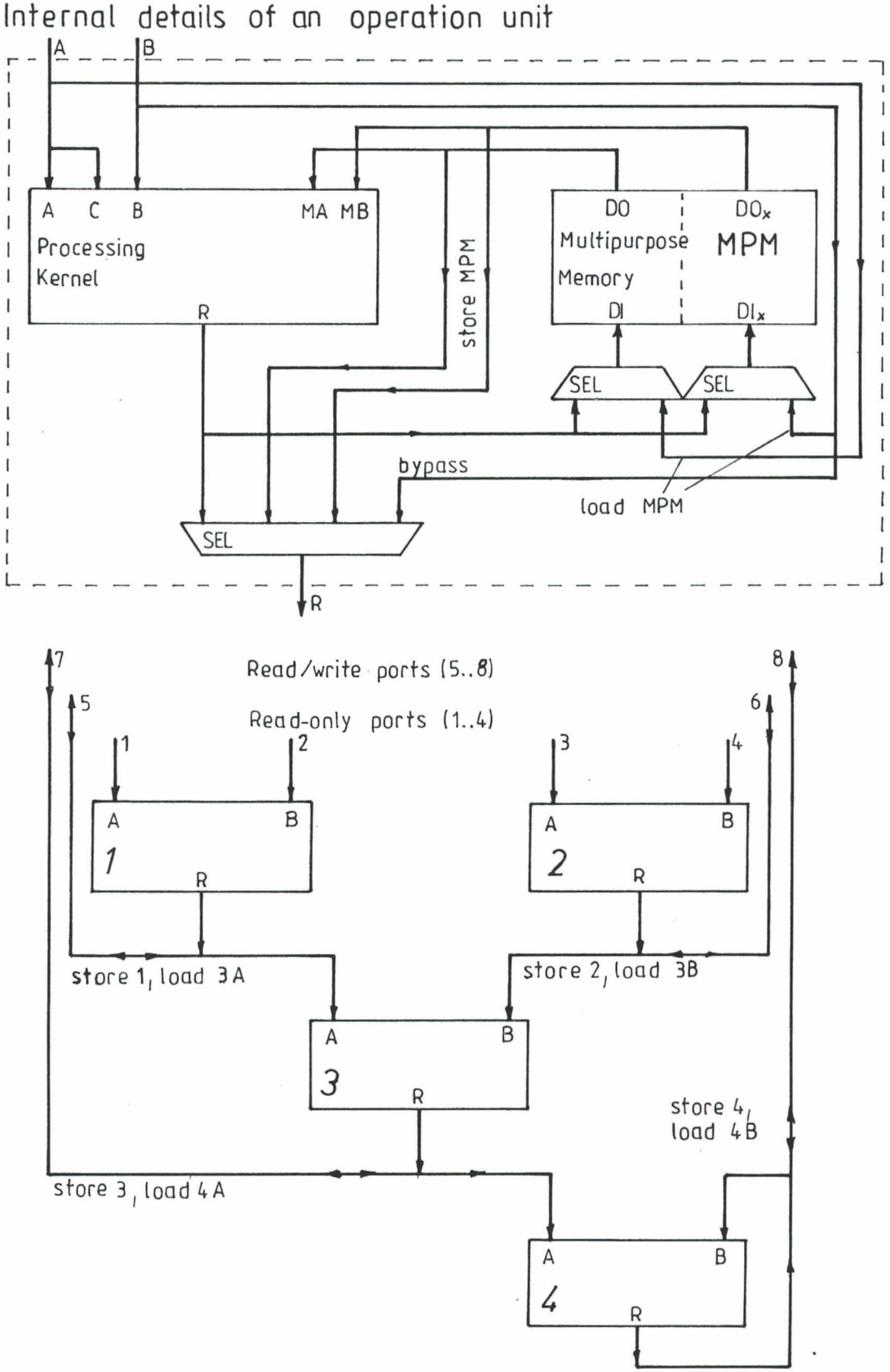


Figure 11: The proposed structure extended.

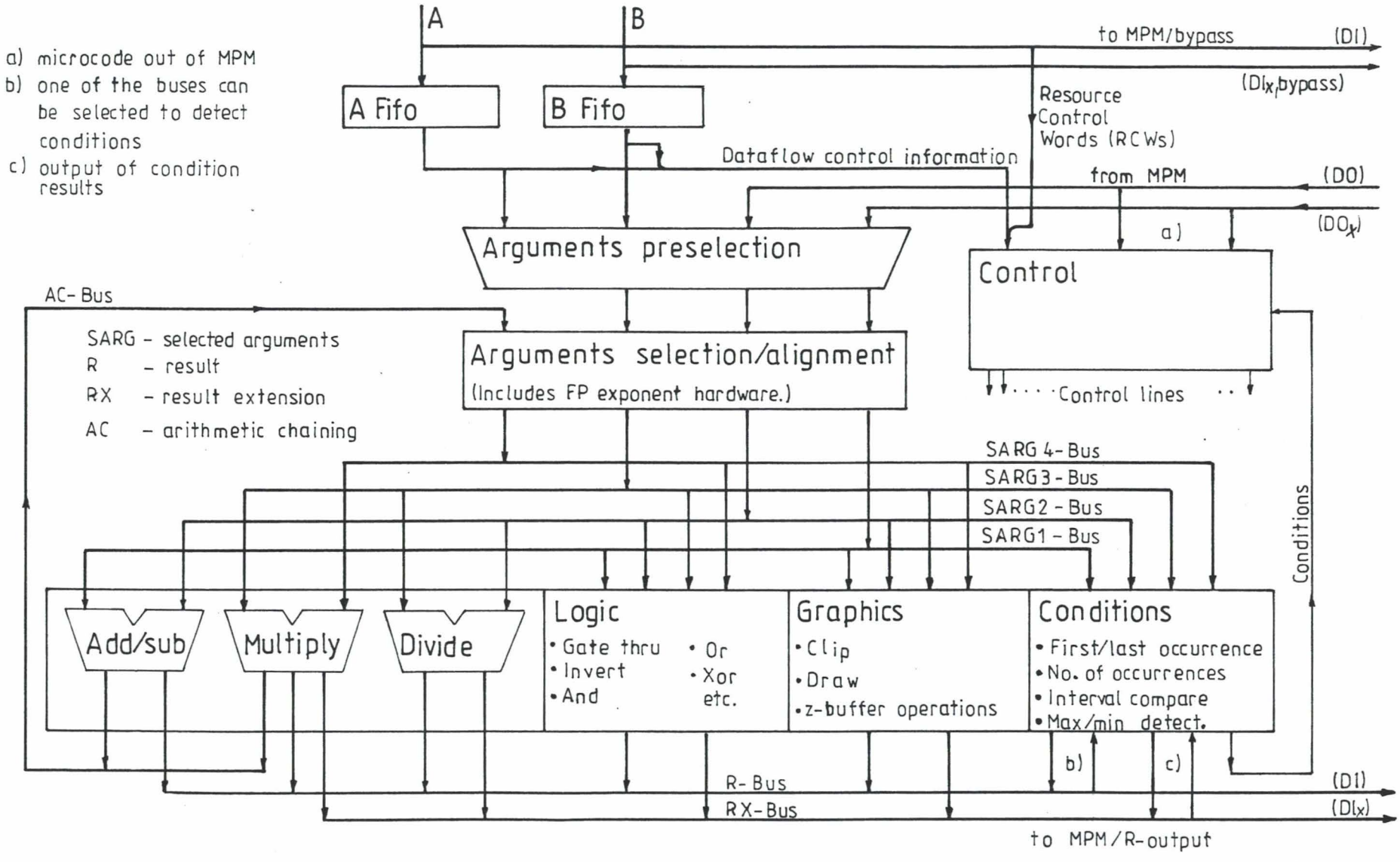


Figure 12: Internal structure of a processing kernel.

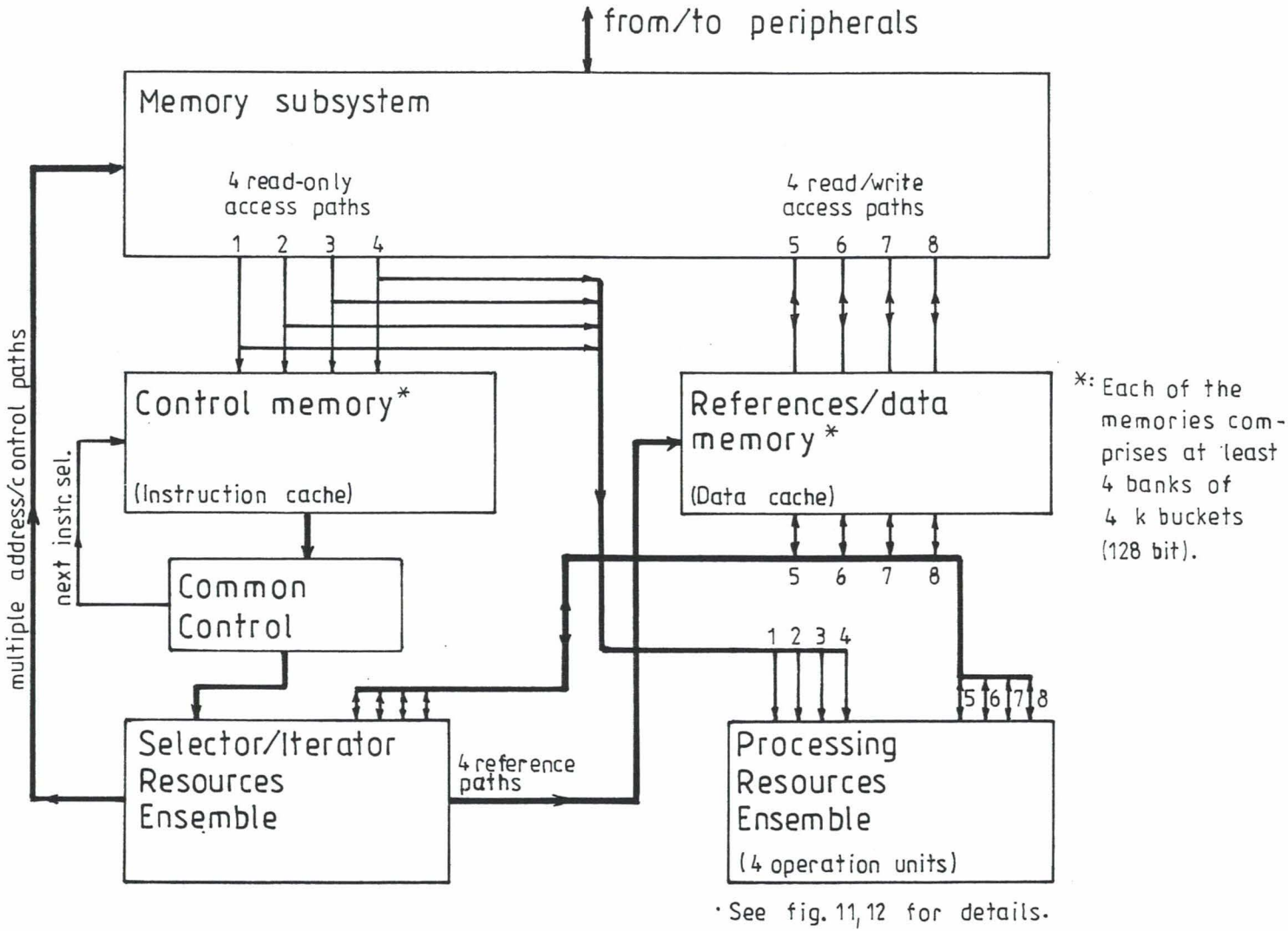


Figure 13: Processor structure.

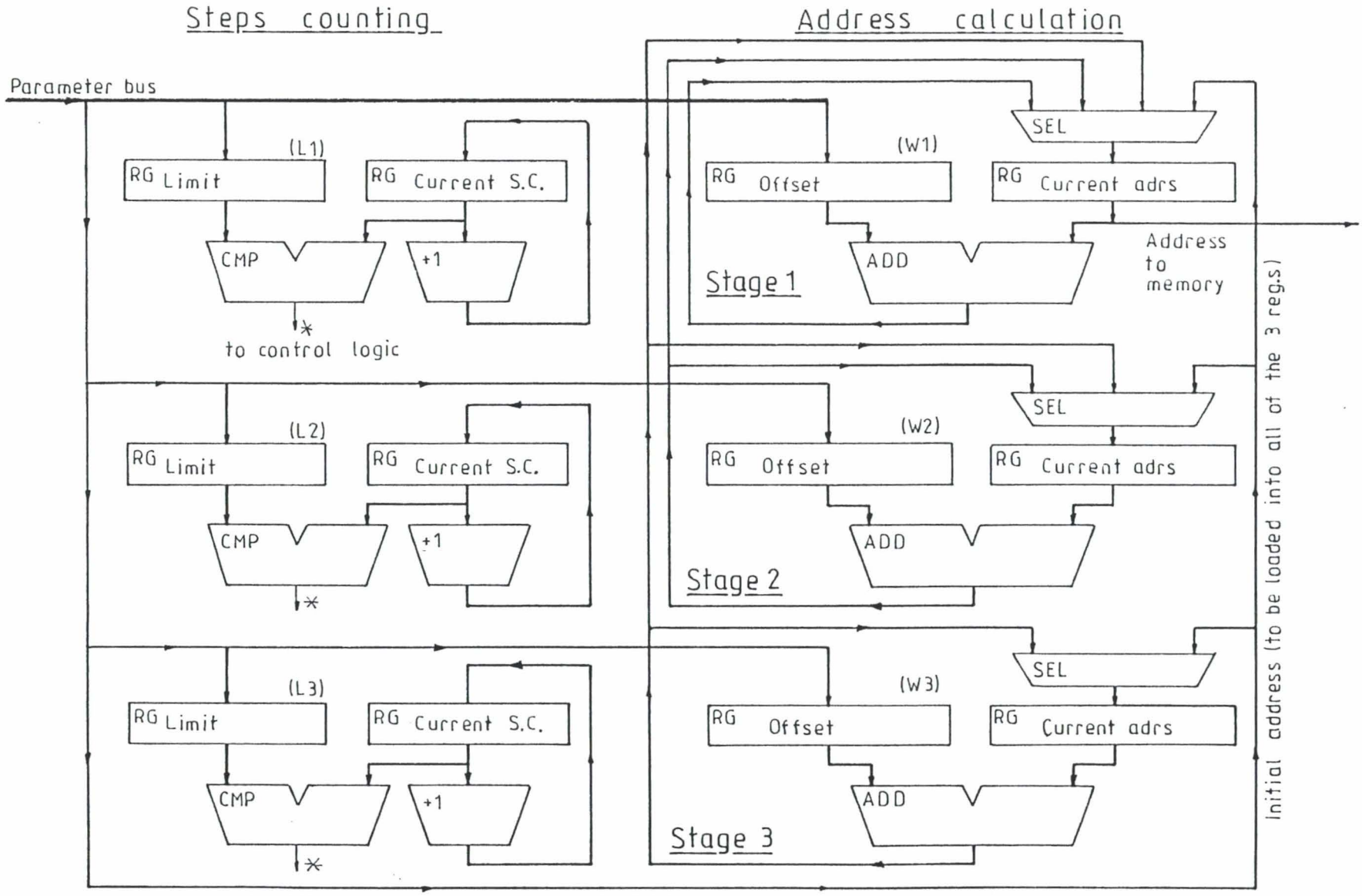


Figure 14: Iterator hardware for three nested DO-loops. (Stage 1 corresponds to innermost loop.)

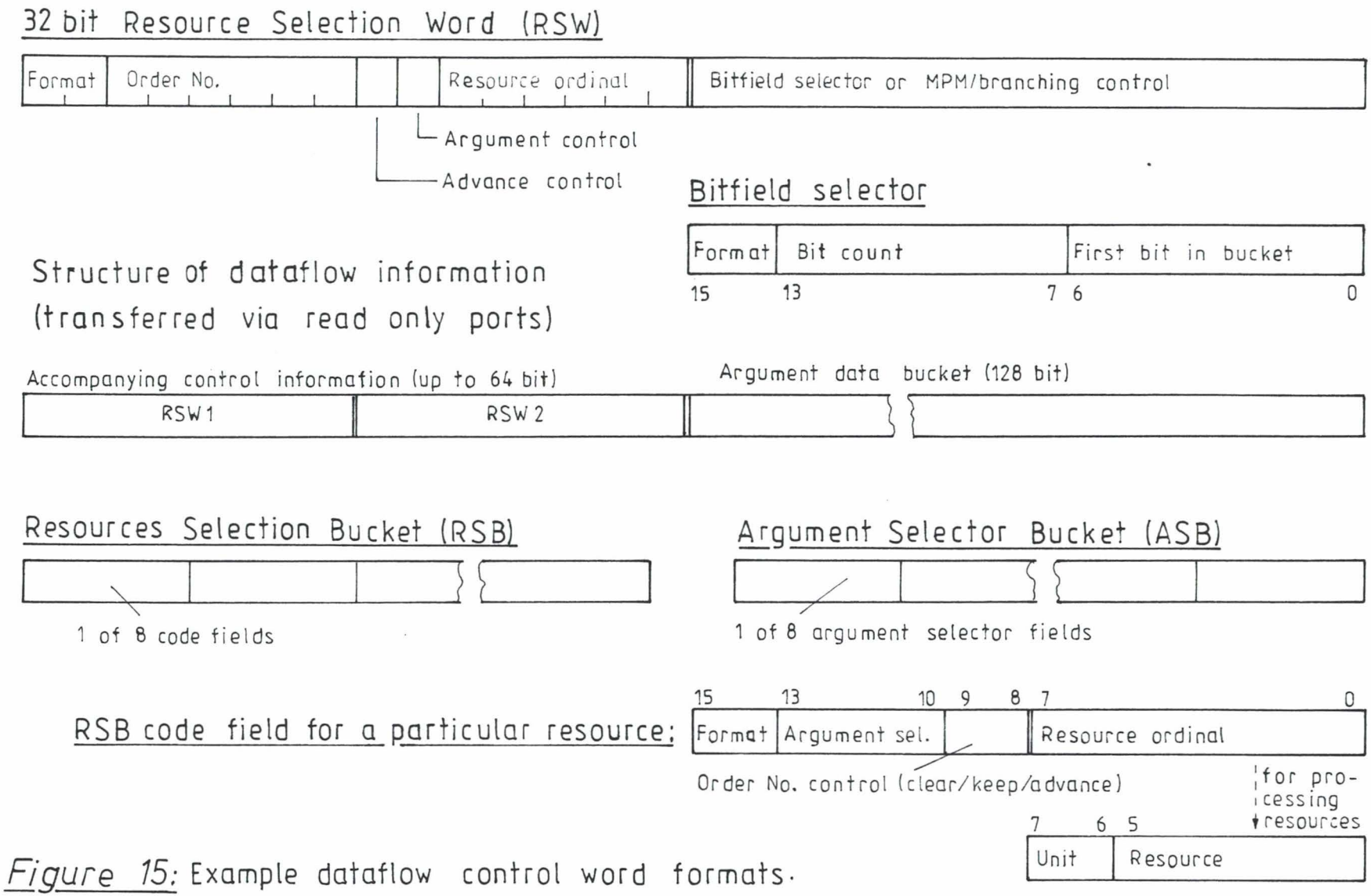


Figure 15: Example dataflow control word formats.