http://www.realcomputerarchitecture.com

The ReAl Computer Architecture

ReAl = Resource Algebra

- ReAl principles
- ReAl operators
- ReAl machines
- Related work
- Call to action

The ReAl Computer Architecture

ReAl = Resource Algebra

The architectural principles of contemporary computers had been invented in the sixties, seventies and eighties. In those times hardware was scarce.

But now?

200 million transistors – and much more – should be used to try out something *ReAlly* new . . .

Let's start with a radically different hypothesis – all we need will be available in abundance:

- Hardware does not matter.
- Memory capacity does not matter.
- Hardware requirements for machine program generation do not matter.

Primary objectives of ReAl architecture design:

- To utilize the inherent parallelism in information processing operations to the highest possible degree – limited only by the very nature of the application problem and the available hardware.
- To provide interfaces between hardware and software that ensure machine-independence and interchangeability.

The ReAl Computer Architecture

ReAl = Resource Algebra

What constitutes a computer architecture?

- a set of data structures D,
- a set of operations O,
- a mapping of C onto D.

$$CA = \{D, O, MOD\}$$

This is essentially an algebraic structure. Each instance of a data structure needs some kind of hardware to be stored in, each operation needs a hardware device to be executed onto. Without hardware, even the most sophisticated software cannot be executed.

The basic hardware building blocks are called "resources."

Hence the name ReAl = Resource Algebra.

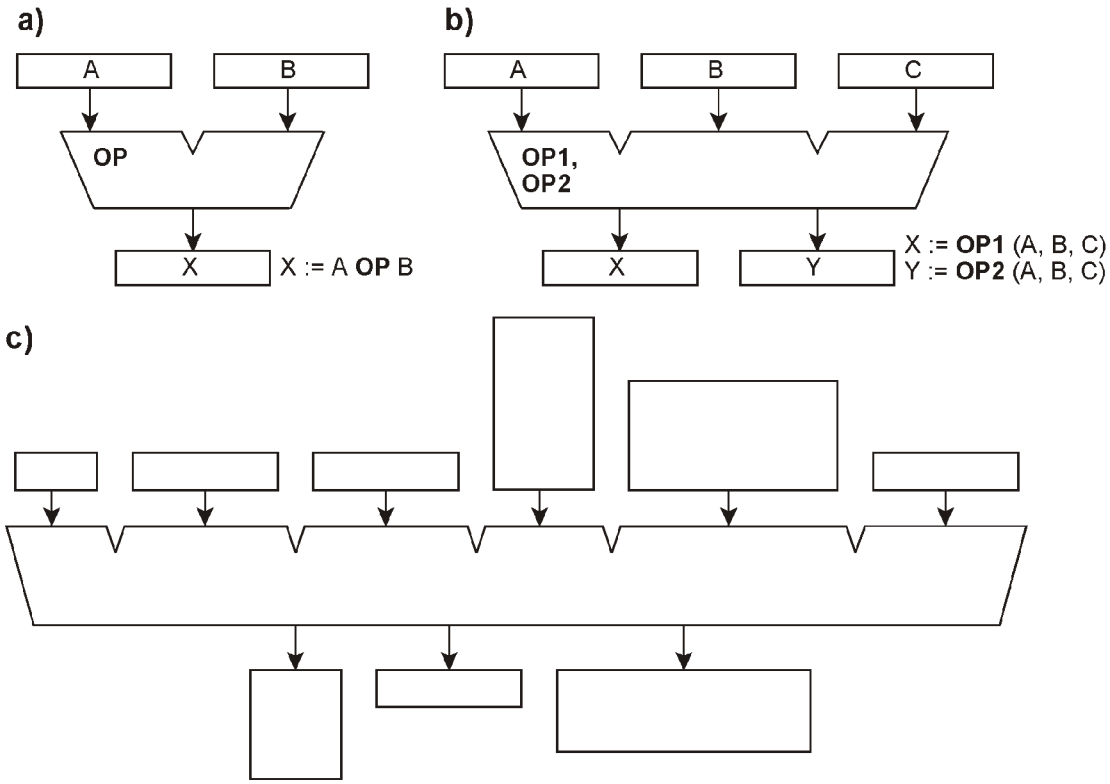
What is a resource?

Essentially a building block or functional unit, for example, an arithmetic/ logic unit (ALU), an address counter, or an addressable memory array. Typically, a resource will be described by its register transfer level (RTL) structure. Basic resources comprise input registers, combinational circuitry and output registers.

The ReAl Computer Architecture

ReAl = Resource Algebra

RTL diagrams of resources show operand storage means, combinational circuitry and result storage means. The storage means could be registers or memory arrays, respectively (for example, to accommodate vectors or character strings).



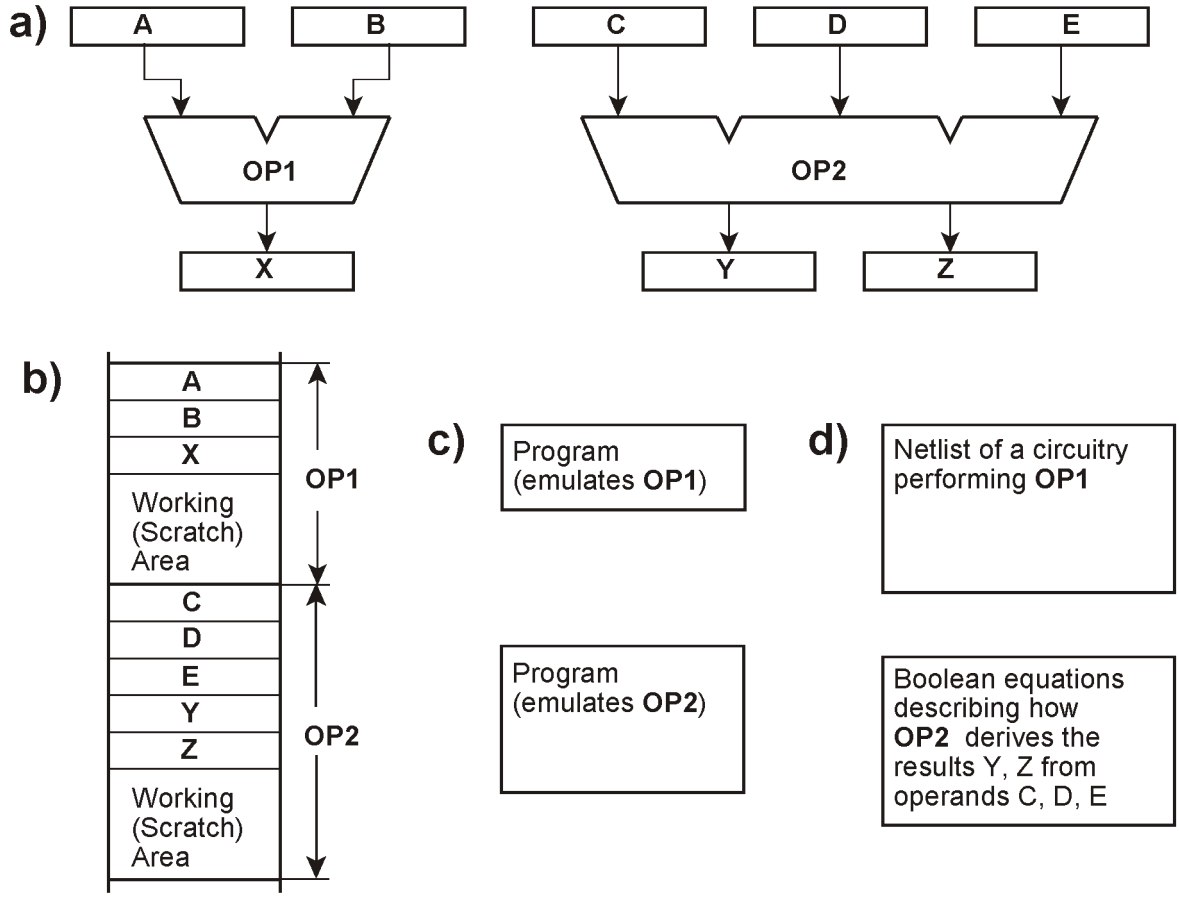
The ReAl Computer Architecture

ReAl = Resource Algebra

Resources can be implemented the hard or the soft way. Here are two examples.

a) Hardware implementation.
b) When implemented by software, an appropriate storage area accommodates the contents of the operand and result registers.

The functions of the combinational circuitry will be emulated by appropriate programs (c) or described by net lists, Boolean equations or the like (d).



The ReAl Computer Architecture

ReAl = Resource Algebra

Inherent Parallelism

All resources belonging to some kind of ReAl machine can be thought to belong to a set or pool.

The size of this pool is transfinite. The number of resources is restricted only by the maximum value which can be represented by the largest number format of the implementation (for example, more than 4 billion in case of a 32-bit-machine).

Being able to request an arbitrary number of resources, the parallelism inherent in an application program can be exploited up to the utmost degree.

The ReAl Computer Architecture

ReAl = Resource Algebra

Efficiency of implementation (... of the application problem)

The universal computer is essentially nothing more than a makeshift solution. It does not solve the application problem proper; it can only execute comparatively simple instructions.

The true optimum solution would be a dedicated hardware whose machine cycles are spent exclusively to compute the desired final results – neither clock cycles nor memory bandwidth is wasted fetching instructions, loading and storing intermediate results, entering subroutines and so on.

Such a dedicated machine is essentially an application-specific dataflow machine. It has to be designed. Even if sophisticated design tools are used (like silicon compilers), there is a division between development time and run time.

ReAl machines should be true universal machines which can be morphed into application-specific machines dynamically during runtime.

The ReAl Computer Architecture

ReAl = Resource Algebra

Our basic paradigm

When we want to do something, we will fetch an appropriate piece of hardware out of a magazine (like a hammer to drive in a nail or a wrench to fasten a nut) and use it to perform the information processing task to be executed.

When we want to add two numbers together, we take an adder, when we want to compare two values, we take a comparator and so on. A piece of hardware which has done its duty will be returned to the magazine. We will take as many tools as we need, e.g., 50 hammers if 50 nails are to be driven in, or 50 adders if 50 pairs of numbers are to be added together.

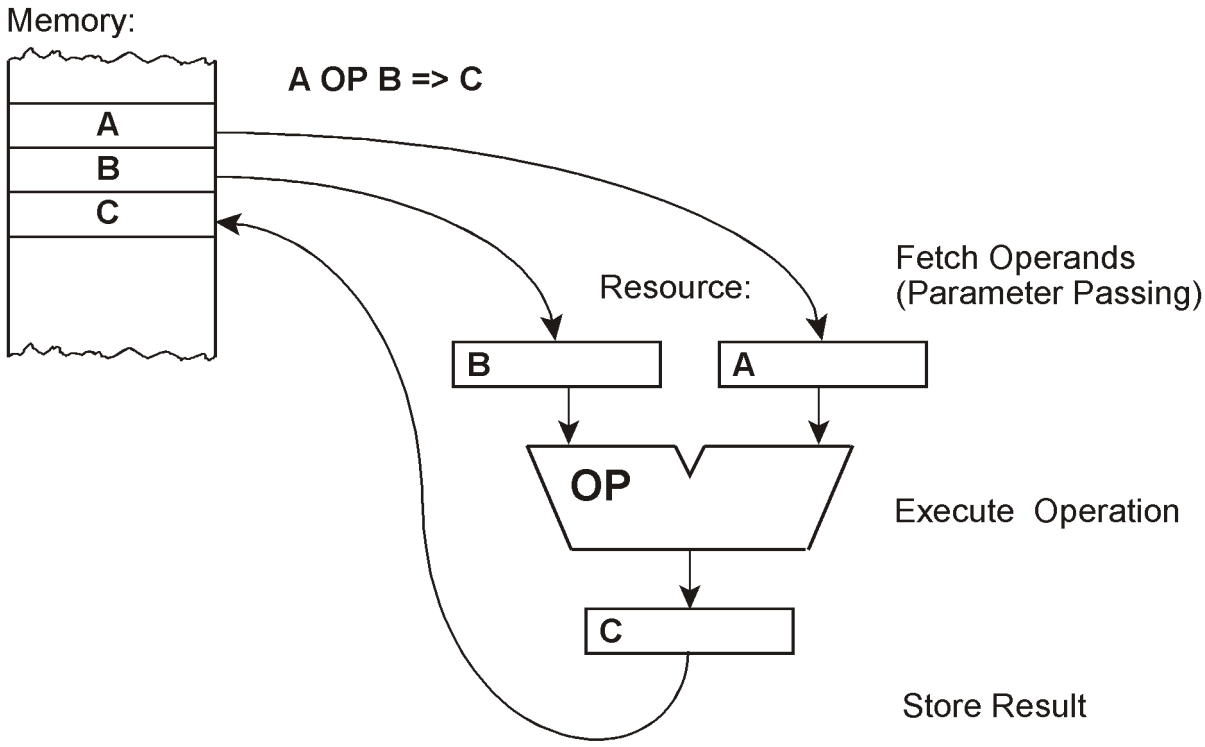
Basic processing steps:

- Appropriate resources will be selected out of the resource pool.
- The resources will be fed with parameters.
- Then the processing operations will be initiated.
- Results will be stored in memory or written to I/O devices; intermediate results will be forwarded to other resources.
- Further steps of parameter passing and assignment of results will be executed until the processing task has been completed.
- Resources which are not longer needed will be returned to the resource pool.

The ReAl Computer Architecture

ReAl = Resource Algebra

How a single resource is used – the basic computational model.



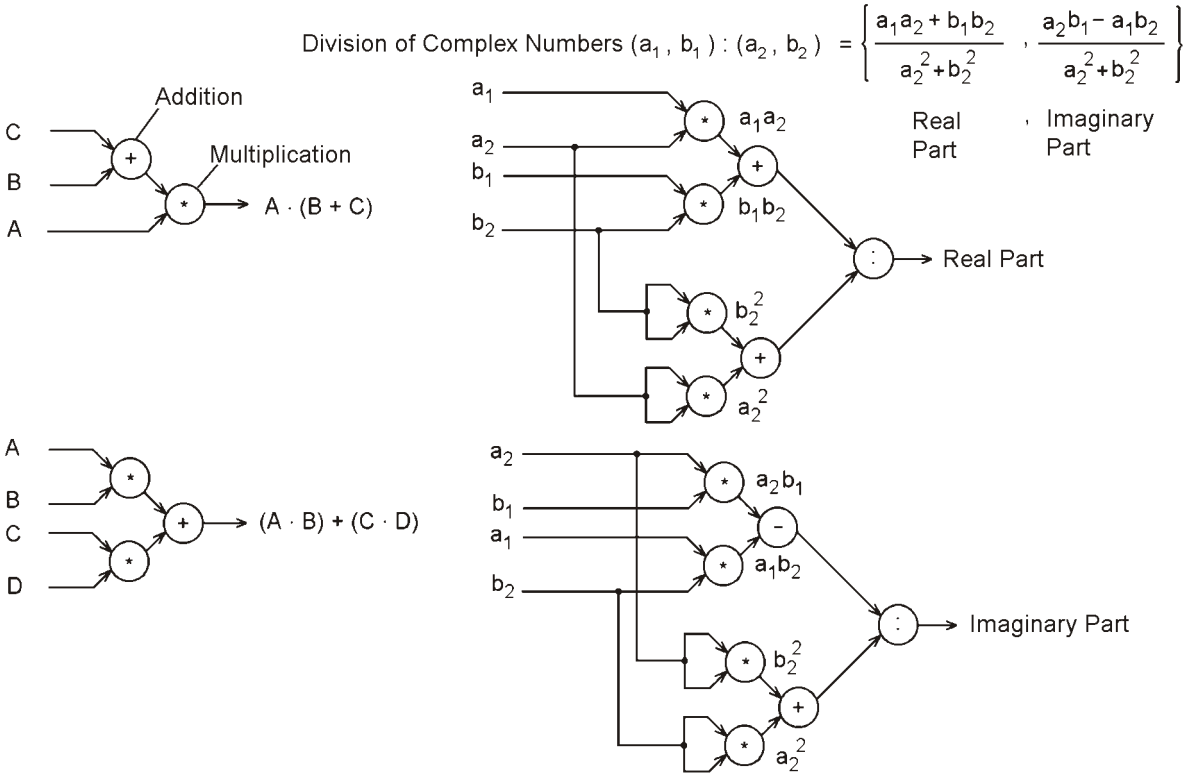
The ReAl Computer Architecture

ReAl = Resource Algebra

Concatenation

The ReAl computational model provides for connecting resources according to the data flow diagrams of the respective processing operations and for disconnecting these connections. Such connections will be referred to as concatenations. Once a concatenation has been established, the steps of parameter passing, initiation of operations and assignment of results will be performed automatically; there is no need to control each single processing step by separate instructions.

Some example data flow diagrams.



The ReAl Computer Architecture

ReAl = Resource Algebra

Operators – the ReAl instruction set

The processing steps are controlled by stored instructions. The abstract (machine-independent) instructions are called operators. There are at least eight basic types:

1. Select resources: s-operator.
2. Establish concatenations between resources: c-operator.
3. Feed resources with operands (parameter passing): p-operator.
4. Initiate the information processing operations: y-operator (yield).
5. Move data between resources: l-operator (link).
6. Assign results: a-operator.
7. Disconnect concatenations: d-operator.
8. Return resources to the resource pool: r-operator.

Operators describe the basic steps of information processing. Machine-independent ReAl programs are sequences of operators. To be stored and executed, operators have to be encoded. Basically, there are three types of ReAl codes: textcodes, bytecodes, and fixed-format machine codes.

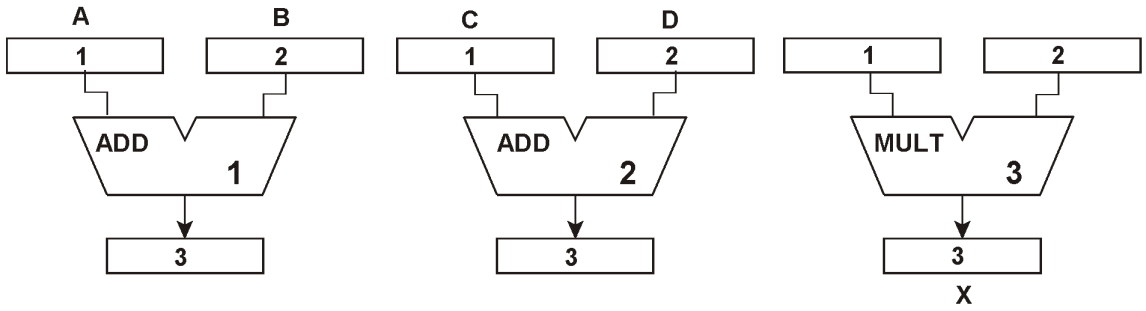
The ReAl Computer Architecture

ReAl = Resource Algebra

A basic example

The application problem: X := (A + B) • (C + D).

This task requires three resources; two adders (ADD) and one multiplier (MULT). The diagram shows the ordinal numbers of the resources and their parameters.



The ReAl Computer Architecture

ReAl = Resource Algebra

The program listing

Left: Program written in a step-by-step notation.

Right: Some code formats allow for longer argument lists, so more activities can be initiated by one operator.

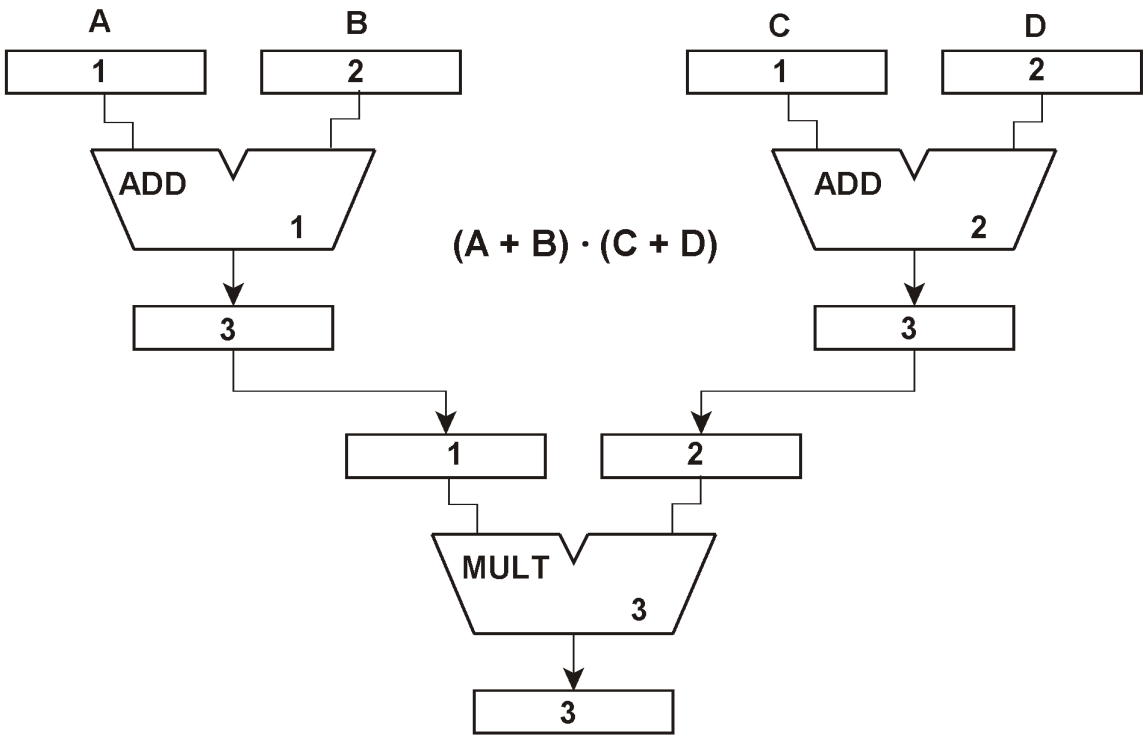
s (ADD s (ADD) s (MULT) p (A => 1.1) p (B => 1.2) p (C => 2.1) p (D => 2.2) y (1) y (2) l (1.3 => 3.1) l (2.3 => 3.2) r (1) r (2) y (3) a (3. 3 => X) r (3)	s (ADD, ADD, MULT) p (A => 1.1, B => 1.2, C => 2.1, D => 2.2) y (1, 2) l (1.3 => 3.1, 2.3 => 3.2) r (1, 2) y (3) a (3.3 => X) r (3)

The ReAl Computer Architecture

ReAl = Resource Algebra

The basic example solved by concatenating the resources

The resources are concatenated according to the dataflow graph of the application problem.




```
s  (ADD, ADD, MULT)
c (1.3 => 3.1, 2.3 => 3.2)
p (A => 1.1, B => 1.2, C => 2.1, D => 2.2)
a (3.3 => X)
r (1, 2, 3)
```

The ReAl Computer Architecture

ReAl = Resource Algebra

ReAl machines

... comprise processing resources, platform resources, and storage means.

A processing resource is a functional unit – more than a logic block of an FPGA and less than a complete processor. An arithmetic/logic unit (ALU) with some addressing, control, and storage means may serve as a typical example.

Resources in ReAl machines are less complex than the operation units of the contemporary high-performance processors. Above all, provisions for internal pipelining will not be necessary. Instead, effects of operation overlapping will show up as a consequence of concatenation and of employing multiple resources simultaneously.

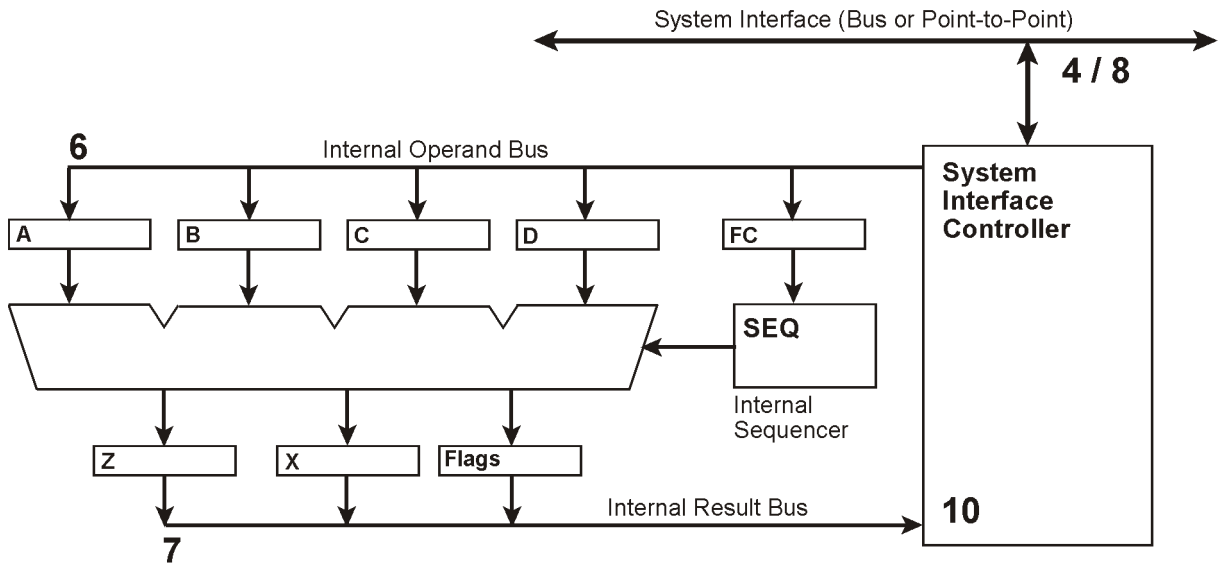
Platform resources are provided to fetch the instructions from memory. The platform comprises the resources that are required in order to initiate and maintain operation of the system. Basic platforms contain an instruction counter and provisions for branching and for calling of subroutines.

In more advanced ReAl machines, the platform will be used merely for initialization, administration of the resources and the like. Decisions, conditional execution, loop control and so on will be delegated essentially to the processing resources.

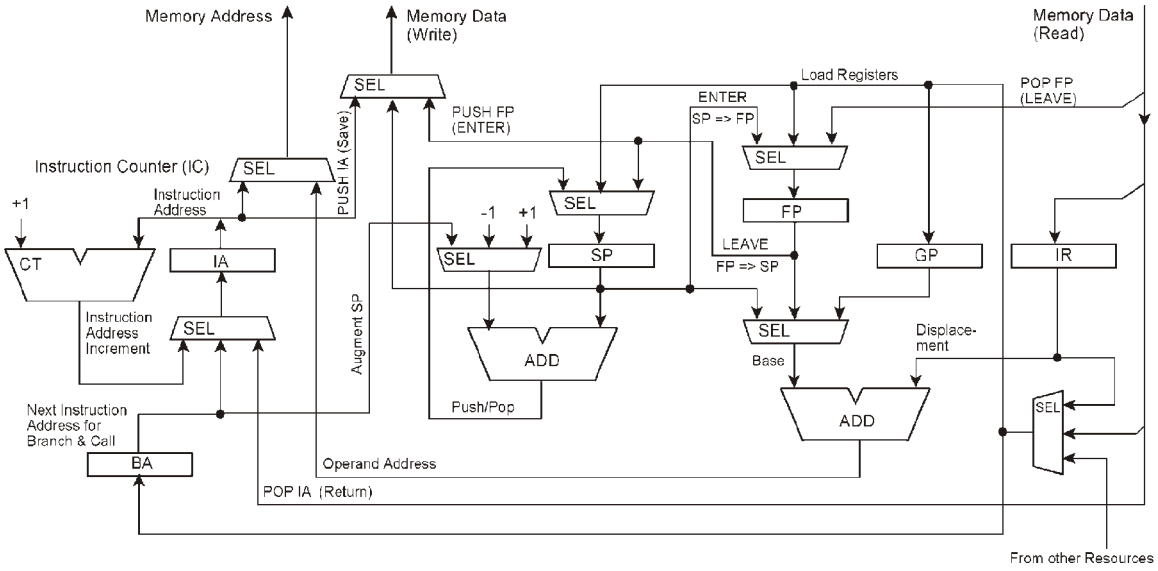
The ReAl Computer Architecture

ReAl = Resource Algebra

A typical – somewhat more advanced – universal processing resource.



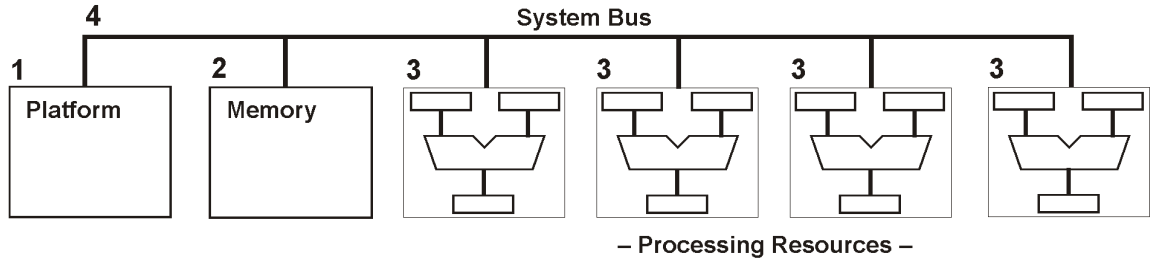
This platform supports instruction addressing, conditional branching and C/Unix-like function calls.



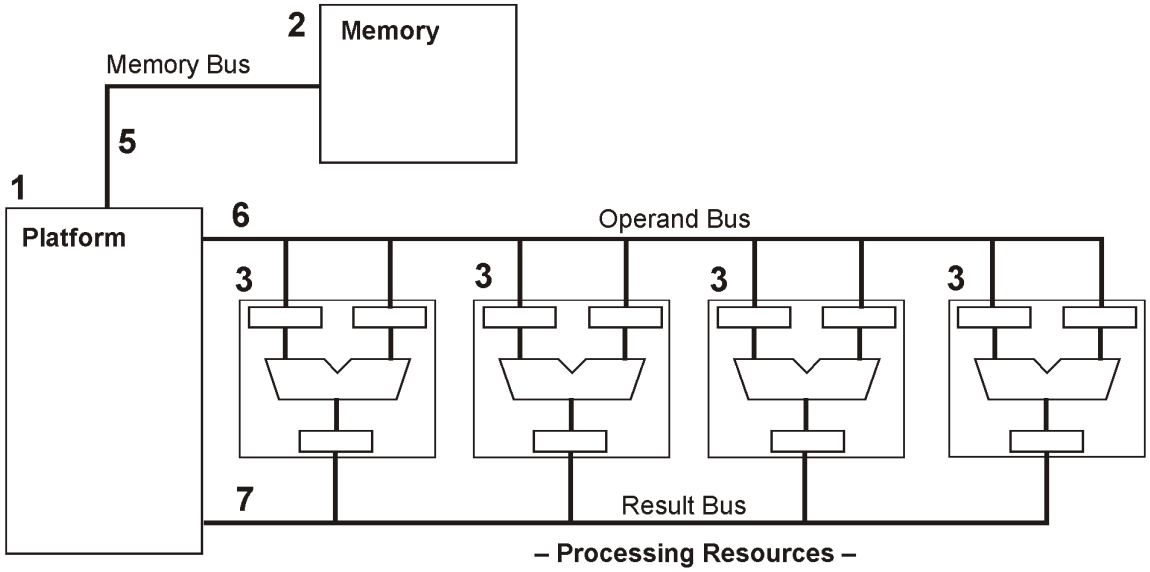
The ReAl Computer Architecture

ReAl = Resource Algebra

A very basic ReAl machine.



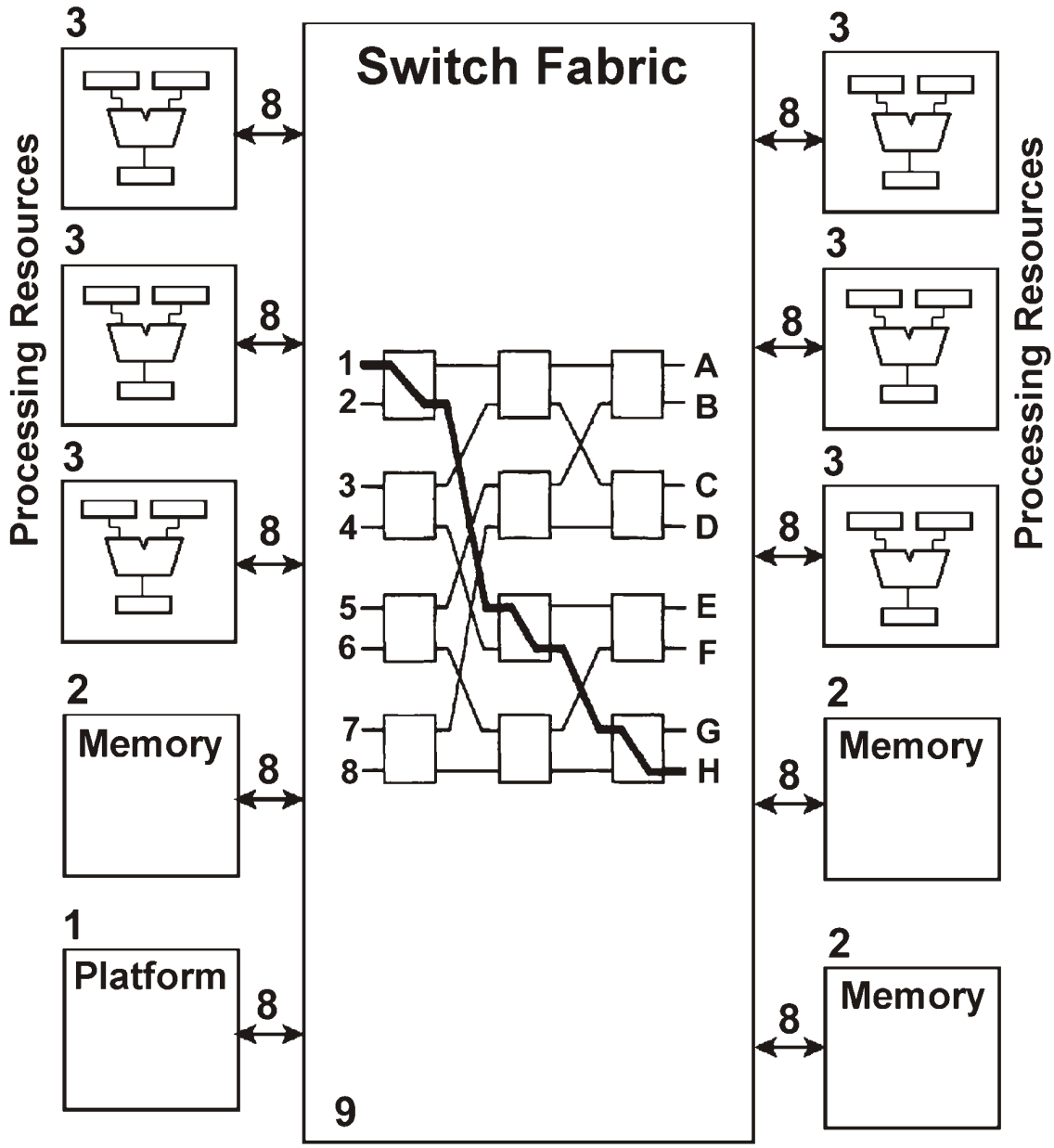
A ReAl machine comprising multiple bus systems.



The ReAl Computer Architecture

ReAl = Resource Algebra

A ReAl machine based on switched interconnections.

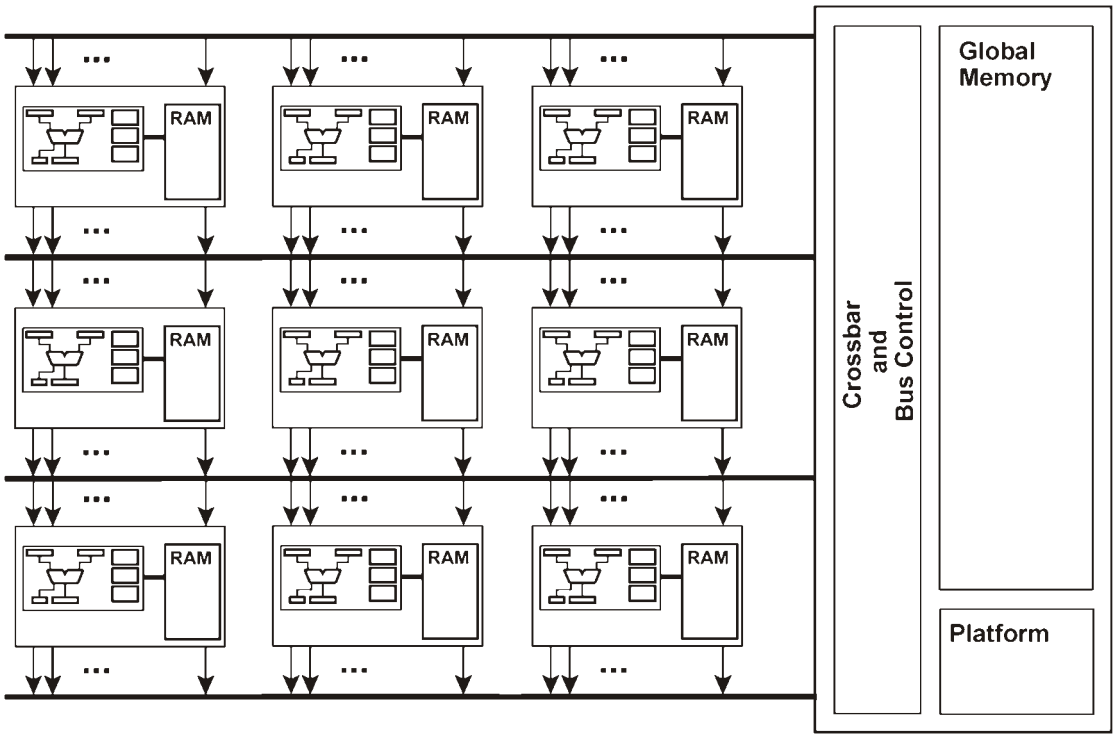


The ReAl Computer Architecture

ReAl = Resource Algebra

Resource cells in ReAl FPGA circuits.

Each cell is hard wired. It comprises an arithmetic/logic unit, addressing means and some memory capacity (for example, enough to hold the variables of a typical C-like function or a few floating point vectors).



The ReAl Computer Architecture

ReAl = Resource Algebra

Resources should be able to work autonomously. Instruction fetch cycles as well as load and store cycles have to be avoided whenever possible.

In an ideal machine, only data related to the solution of the application problem would be fetched and stored.

In a ReAl machine, instructions will set up configurations of resources which corresponds to parts of the dataflow graph of the application problem and leave the execution of operations and the transport of intermediate data to the processing resources. Each resource knows what it has to do (set up by s-operators), which parameters are to be processed and where the results are to be delivered (set up by p- and c-operators).

Some potential candidates for partial dataflow graphs:

- Basic blocks (linear sequences of instructions between jumps or subroutine calls).
- Innermost loops.
- Conditional statements.
- Subroutines (for example, C-like functions).

The ReAl Computer Architecture

ReAl = Resource Algebra

Resource configurations

The interaction of resources concatenated arbitrarily seems to require complex and expensive interconnection networks. Furthermore, one may suspect that the communications overhead may cost more machine cycles than the conventional instruction fetches etc. we try to avoid. We assume that these problems can be circumvented by restricting the hardware topology to a few essential configurations and by coalescing processing resources and memory.

Essential resource topologies

We assume that only two configurations need to be supported in hardware:

1. Independent resources operating in parallel.
2. Inverted binary trees.

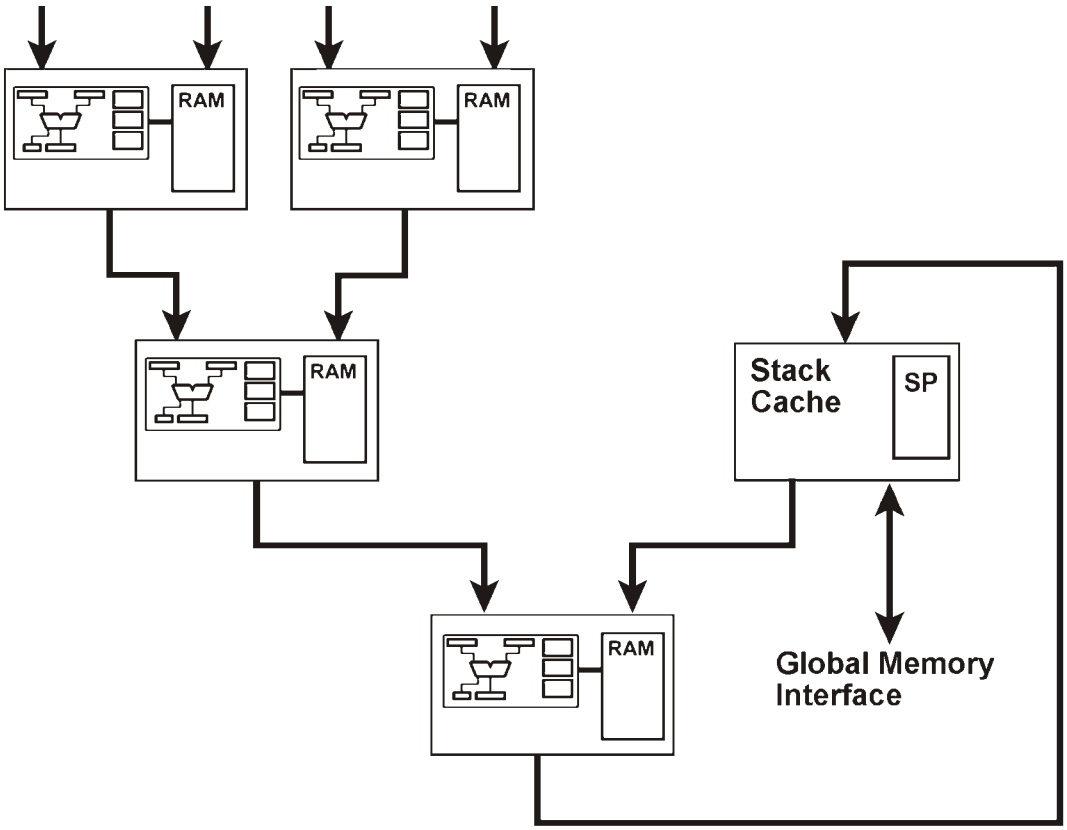
All other topologies could be emulated (virtual connections).

The evaluation of nested expressions (including function calls) can be mapped well onto inverted tree structures. This is also true for operations which compute a single result from multiple data (like SAXPY or SAD). Within such a tree, the data paths between the resources are simple (and short) point-to-point connections.

The ReAl Computer Architecture

ReAl = Resource Algebra

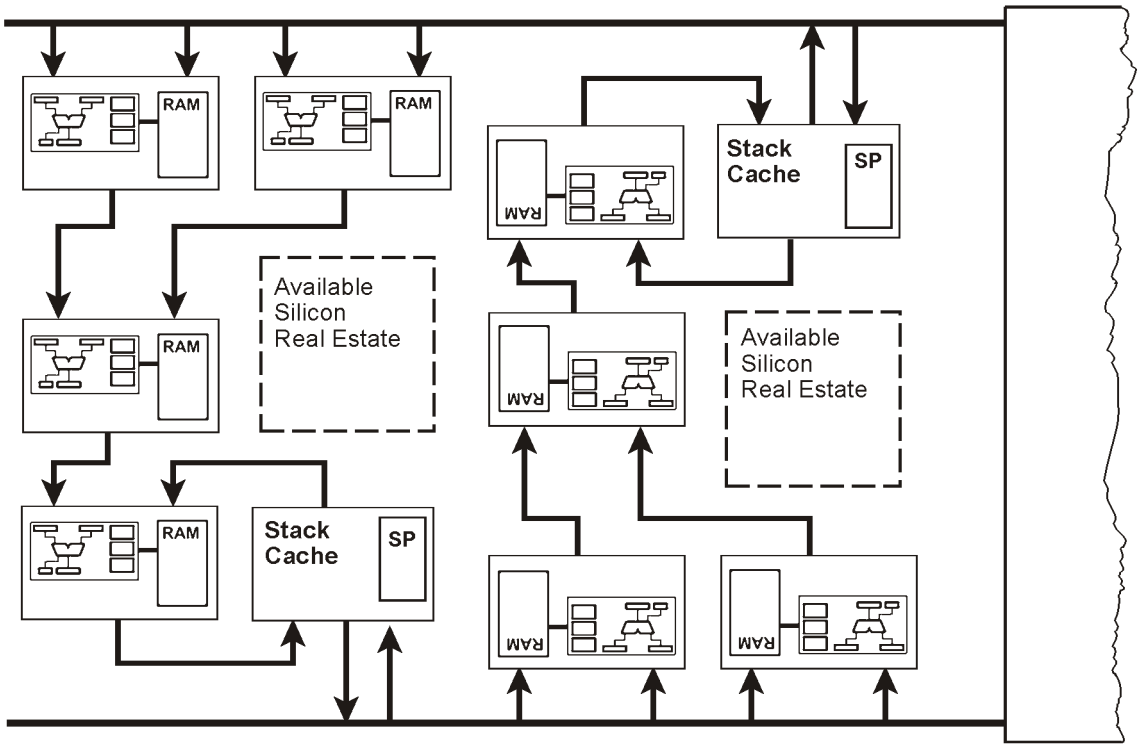
Resources concatenated to an inverted binary tree.
To support emulation of a function call stack, an additional stack cache has been provided.



The ReAl Computer Architecture

ReAl = Resource Algebra

Inverted trees with associated stack caches fit well between the bus structures of a ReAl FPGA.



The ReAl Computer Architecture

ReAl = Resource Algebra

ReAl and programming languages

Fundamentally, ReAl programs can be likened to manufacturing or machining instructions* – ReAl programming means just to plan ahead. Which manufacturing steps are to be executed? Which tools and machines are necessary? Which part has to be supplied to which machine in the course of time? No engineer would begin designing cars, ships and so on writing down instructions of this kind. Analogously, a programmer will not use a ReAl text code for jotting down his programming ideas.

Instead, ReAl programs will be generated automatically from source programs written in higher-level languages. Machine-independent ReAl codes can be seen as intermediate languages, similar to the well-known Java byte code. However, the goal is not code compactness but to describe precisely the inherent parallelism and essential intricacies of program operation. In this respect, ReAl may be better compared to Postscript than to Java.

*: Something like "To manufacture this gearbox, we will need three lathes, five milling machines and so on. Part No. 33 will be machined on lathe No. 2 and then finished on grinding machine No. 6."

The ReAl Computer Architecture

ReAl = Resource Algebra

Java, JVM	ReAl
• Code compactness (bytecode) • Developed for small programs (applets) • Executable on thin machines • Programs to be downloaded via internet • JVM is a conventional stack machine, hence its operations are inherently sequential • JVM bytecode describes one operation at on time, hence inherent parallelism is to be detected during runtime	• To make best possible use of hardware • Developed for large and computing-intensive programs (graphics, equation solving, simulation, data bases, neural networks, AI) • There will always be enough hardware. Memory capacity and code size are irrelevant • Executable on machines which can be built with future IC technology (dozens or even hundreds of operation units on one integrated circuit) • ReAl code describes completely the inherent parallelism of program operation • Creation of virtual special processors which correspond to the dataflow graph of the application problem • Inherent parallelism will be detected not during runtime but in statu nascendi (in other words, by examination of the programming intentions) • A sufficiently standardized ReAl instruction set is a unified machine language, which can describe hardware as well as software

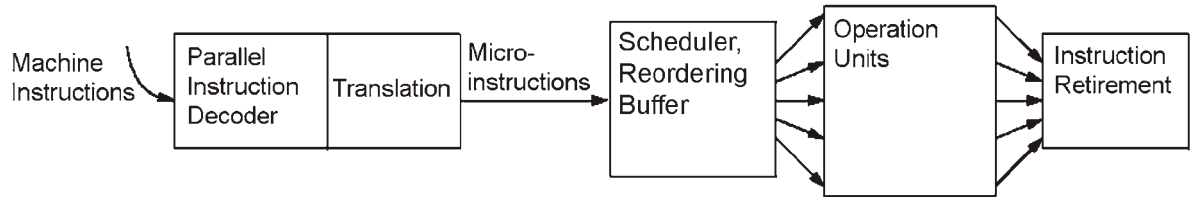
The ReAl Computer Architecture

ReAl = Resource Algebra

Superscalar and ReAl Architectures

The multiple operation units in superscalar machines are controlled by appropriately formatted instructions (explicit instruction level parallelism) or by a speculation mechanism. This mechanism tries to emulate some kind of dataflow machine, executing instructions according to the availability of the data to be processed.

Simplified block diagram of a typical superscalar processor.

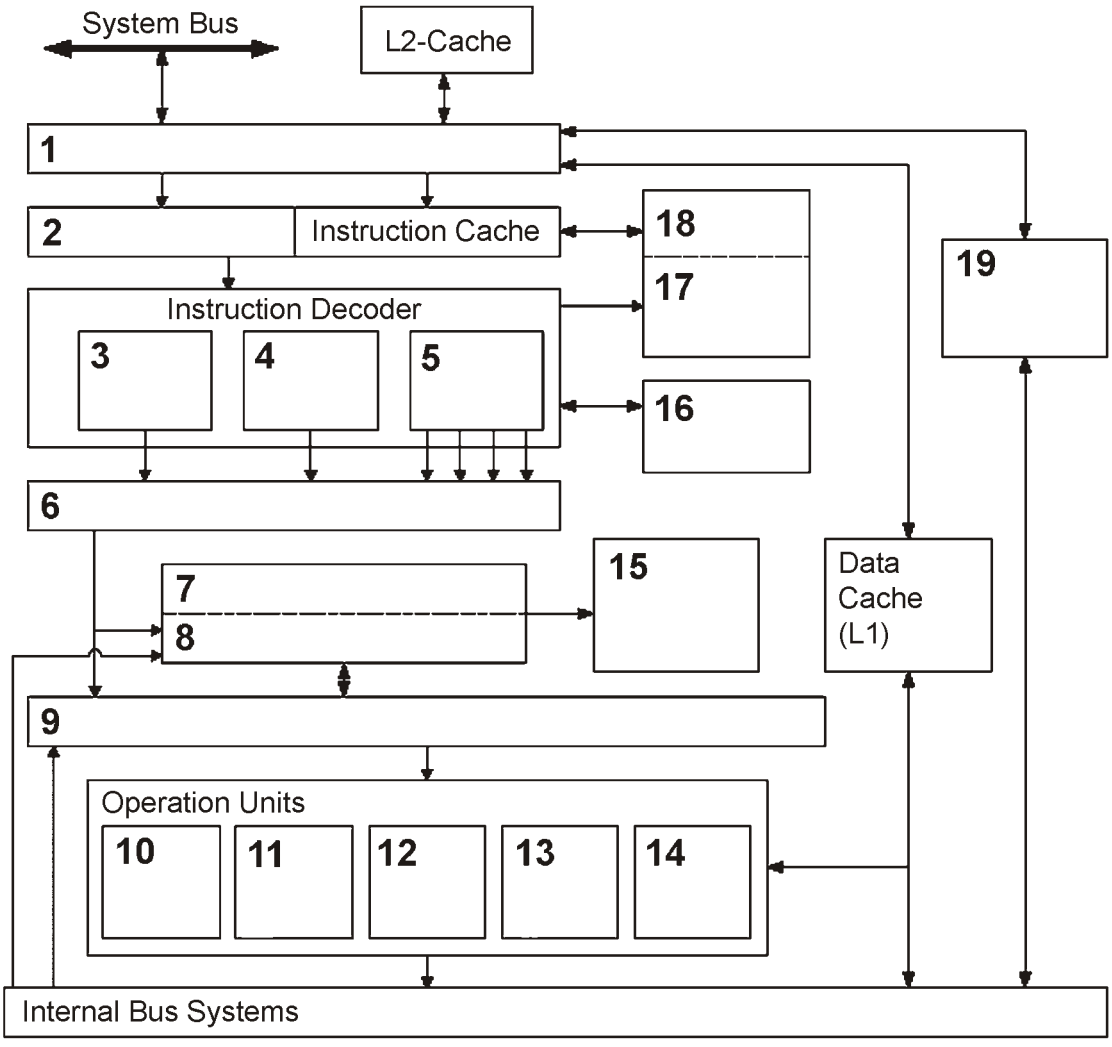


This type of parallel processing is essentially a trial and error approach. Inherent parallelism can be detected only within short instruction sequences. Since in case of a conflict the execution of the instruction must be repeated, the processing performance will drop. Moreover, because of the controlling and monitoring overhead, typically only elementary instructions will be supported this way. Instructions with complex functions are often executed serially. The complexity of the control circuitry is comparatively high.

The ReAl Computer Architecture

ReAl = Resource Algebra

A superscalar processor in more detail (source: Intel).



1 - system bus controller; 2 - instruction fetch unit; 3, 4 - instruction decoder for simple instructions; 5 - instruction decoder for complex instructions; 6 - register allocation unit; 7 - instruction retirement; 8 - microinstructions reordering buffer; 9 - microinstructions scheduler; 10, 11 - floating point operation units; 12, 13 - integer operation units; 14 - memory access controller; 15 - architecture registers; 16 - conventional microprogram control (controls everything that is too complex to be executed in parallel; 17- branch target buffer; 18 - architecture instruction counter; 19 - memory access buffer.

The ReAl Computer Architecture

ReAl = Resource Algebra

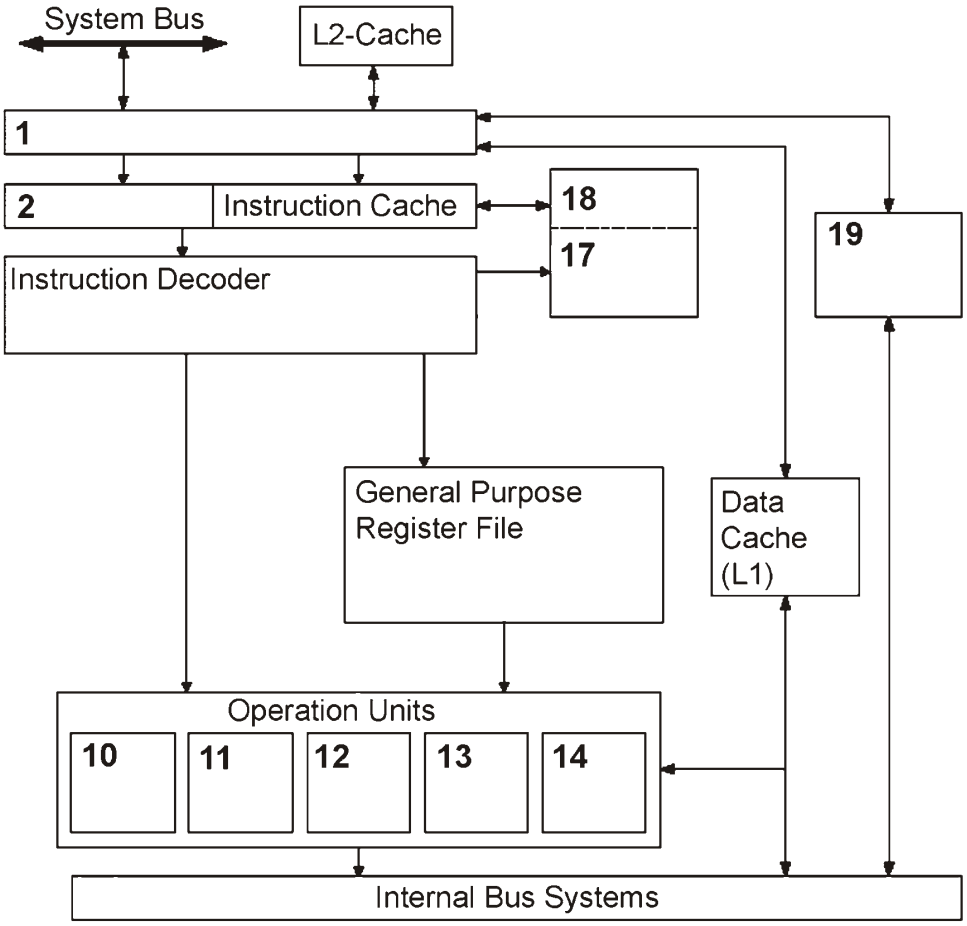
What a current superscalar machine does implicitly and speculatively at a comparatively small scale (for example, with 4 to 16 operation units), a ReAl machine could do explicitly and in a deterministic way at a scale only limited by semiconductor technology.

Conventional superscalar machines	ReAl machines
• More than one operation unit (e.g., 2 to 16) • Complex pipelined circuitry • Provides partial data flow operation by speculation • Complex hardware to detect inherent parallelism between instructions • Complex hardware to detect conflicts and hazards during execution • Rigid processor structures (the application problem must match sufficiently well, or there will be inefficiencies) • Conventional instruction set. Downwardly compatible instruction set architectures can be supported (cf. the processors of the personal computers)	• More than one processing resource • Each resource is a comparatively less complex, non-pipelined circuitry • Provides partial data flow operation based on a deterministic description (c-operators), • Inherent parallelism detected during compile time; no dedicated circuitry required • The ensemble of resources could be morphed to (virtual) application-specific machines • New instruction set architecture; describing parallelism and dataflow operation in detail

The ReAl Computer Architecture

ReAl = Resource Algebra

The superscalar processor shown above could be turned into a ReAl machine. The operation units, the cache memories, the buffers as well as the bus interfaces remain. The instruction decoder is significantly more straightforward. The general-purpose register file could be extended significantly (for example, up to 256 registers). The operation units could be directly connected with the general-purpose registers. Since the complex control circuitry is not needed, the set of operations could be expanded or additional operation units could be provided.



The ReAl Computer Architecture

ReAl = Resource Algebra

A rough estimate:

Conventional high-performance processors consist of about 10 to 50 million transistors. An integrated circuit with 200 million transistors can accomodate four superscalar processor cores, each comprising approximately 50 million transistors. However, the performance capability of this arrangement can become effective only when at least four programs are to be executed at the same time; the individual program cannot be accelerated in itself. The operation units of one of the processor cores correspond roughly to eight 64-bit arithmetic/logic units (the differences between integer and floating point units etc. being neglected here). These 4 cores • 8 operation units correspond to 32 resources. The instruction fetch and execution control hardware is to be replaced by ReAl platform circuitry. Cache memories, control circuits, bus systems etc. are maintained (same size, but modified structure). Some more resources could be located on the silicon area otherwise occupied by auxiliary and control circuitry (pipelining, detection of hazards and the like). Therefore, one can reasonably expect a processor IC containing approximately 48 to 64 high-performance processing resources. According to the requirements of the applications to be executed, this ensemble of resources could be morphed into graphic engines, database engines etc. under control of ReAl operators.

The ReAl Computer Architecture

ReAl = Resource Algebra

Optimization of high-performance processors vs. ReAl

Some activities to develop optimized high-performance processors correspond to important goals of the ReAl approach.

Recommendations to improve computational throughput in conventional processors*	**Within ReAl machines, these recommendations will be more than fulfilled . . .**
• Reduce the amount of load/store cycles (more than 30% of the instructions executed in a RISC architecture are load and store instructions)	• If resources can be set up according to the data flow, no load/store cycles will be needed at all. Even the instruction fetch cycles are avoided, as the control codes have been loaded into the resources.
• Streamline repetitive operations (perform time-critical operations on multiple data simultaneously)	• This will pose no problem if enough processing resources are available
• Maximize utilization of pipeline resources	• Since the processing resources are not part of a rigid hardware pipeline but can be interconnected freely, some utilization problems and hazards are avoided which are typical of pipelined hardware
• Minimize branch latency	• Can be achieved by appropriate platform design. Even multiway branches can be supported

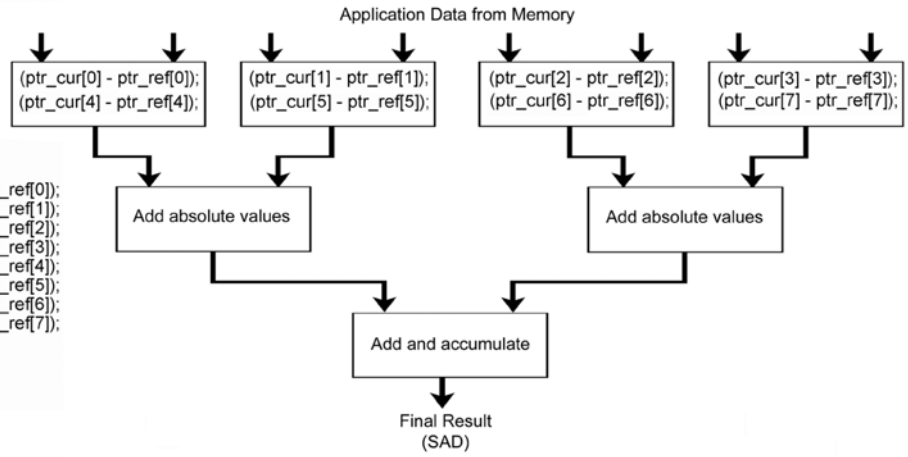
*: according to Atmel.

The ReAl Computer Architecture

ReAl = Resource Algebra

An algorithm to calculate the sum of absolute differences (SAD). Left: The original C-code (source: xvid codec). Right: an appropriate ReAl configuration of processing resources.

```
uint32_t
sad8_c(const uint8_t * const cur,
       const uint8_t * const ref,
       const uint32_t stride)
{
   uint32_t sad = 0;
   uint32_t j;
   uint8_t const *ptr_cur = cur;
   uint8_t const *ptr_ref = ref;

   for (j = 0; j < 8; j++) {

      sad += abs(ptr_cur[0] - ptr_ref[0]);
      sad += abs(ptr_cur[1] - ptr_ref[1]);
      sad += abs(ptr_cur[2] - ptr_ref[2]);
      sad += abs(ptr_cur[3] - ptr_ref[3]);
      sad += abs(ptr_cur[4] - ptr_ref[4]);
      sad += abs(ptr_cur[5] - ptr_ref[5]);
      sad += abs(ptr_cur[6] - ptr_ref[6]);
      sad += abs(ptr_cur[7] - ptr_ref[7]);

      ptr_cur += stride;
      ptr_ref += stride;

   }

   return sad;
}
```



This example has been used to illustrate optimization problems of conventional processor architectures. Obviously, the SAD value could be calculated within a loop. But, to gain some speed, this (innermost) loop has been unrolled. 16 operand values are to be fetched and 32 operations have to be executed. Each operation requires at least one instruction, which is to be fetched, too. These operations can easily be mapped onto an inverted tree of concatenated processing resources. Once the resource configuration has been set up, there is no need to fetch more instructions, as all control codes reside within the resources. Therefore, the memory access paths and the total memory bandwith will be available for moving the application data.

The ReAl Computer Architecture

ReAl = Resource Algebra

Multiprocessor systems vs. ReAl

Obviously, multiprocessor systems are advantageous if the application problem matches the system structure. However, there are some basic drawbacks which apply to all systems built of multiple independent processors:

- Each processor needs its own instruction fetch mechanism, instruction cache, instruction sequencer etc.
- The synchronization between processors is difficult, requiring special hardware means (like test-and-set instructions and cache coherency provisions) and causing overhead during runtime.
- If the particular processor is too small, and if the amount of non-parallelizable code cannot be neglected, then Amdahl's Law will be effective.
- Some processors will be unused if the processor arrangement does not match the structure or the size of the application problem (e.g., 16 processors but only 7 threads to be executed in parallel).

The ReAl Computer Architecture

ReAl = Resource Algebra

To a large extent, the ReAl approach has been stimulated by the desire to circumvent these drawbacks. The key points can be summarized as follows:

- To break down the complete processor into its functional units (in other words: to provide less complex processing resources, but more of them).
- To provide for sufficiently efficient, optimized interconnections.
- To develop principles of operation and instruction set architectures which can cope with such hardware configurations.

Dedicated hardware vs. ReAl

Dedicated hardware systems are comparatively expensive. The development task is complex. More recent approaches try to combine hardware and software development more closely. But such systems on silicon are still highly specialized systems. Their structure can be changed only during development time. This is also true for highly flexible processor structures.

The ReAl Computer Architecture

ReAl = Resource Algebra

In contrast, ReAl systems don't know a rigid division between general-purpose processors and specialized circuitry. On a ReAl programmable integrated circuit, application-specific machines can be created on the fly by exploiting all of the resources according to the particular needs of the application problem.

Conventional systems on silicon	ReAl systems on silicon
• Hardware structure to be determined during development time • Hard IP cores cannot be modified • If a hardware unit is to be laid out or modified depending on the application problem, it must be implemented the soft way (with programmable cells) • The general-purpose processor can only be modified, but not changed in its basic instruction set architecture	• Hardware structure can be changed during run time • Typical ReAl resources are hard IP cores of intermediate complexity (considerably more than a macrocell, but much smaller than a general-purpose processor) • If necessary, general-purpose processing hardware can be created on the fly according to the requirements of the application

The ReAl Computer Architecture

ReAl = Resource Algebra

Call to Action:

- Implement emulators on industry standard computing platforms (PCs, microcontrollers).
- Implement appropriate hardware solutions (based on state-of-the-art FPGAs).
- Develop new FPGA architectures with embedded medium grain resource cells.
- Modify state-of-the-art superscalar processors into ReAl hardware platforms (caches, operation units und bus systems remain, instruction decoder, register sets, instruction sequencing and microprograms are to be modified).
- Develop true supercomputer architectures (neither vector processors nor processor farms).
- Write appropriate compilers.
- Write top-notch applications which make use of the ReAl paradigm.

The ReAl Computer Architecture

ReAl = Resource Algebra

The next steps to be taken:

ReAl development will be a time-consuming process.

The first implementations will be emulators.

Toy-like implementations are worthless.

The first experimental ReAl architecture implementation should be a reasonable compiler target.

Hence the complexity will be comparable to the architectures of contemporary high-performance processors (for example, Intel IA-32 and IA-64).

Numerous details have to be taken into consideration. (Contemporary processors have 200 instructions and more.)

A set of basic resources has to be defined in detail.

The theory of operation has to be worked out and described in detail.

All the primary evaluation work has to be done manually, as we cannot afford a number of iteration cycles (design a architecture – write a compiler – evaluate the work based on real-world applications – improve the architecture and so on).

The set of reference manuals will comprise more than 1000 pages.

Algorithms for converting conventional programs into ReAl operator sequences have to be developed.