Transputer

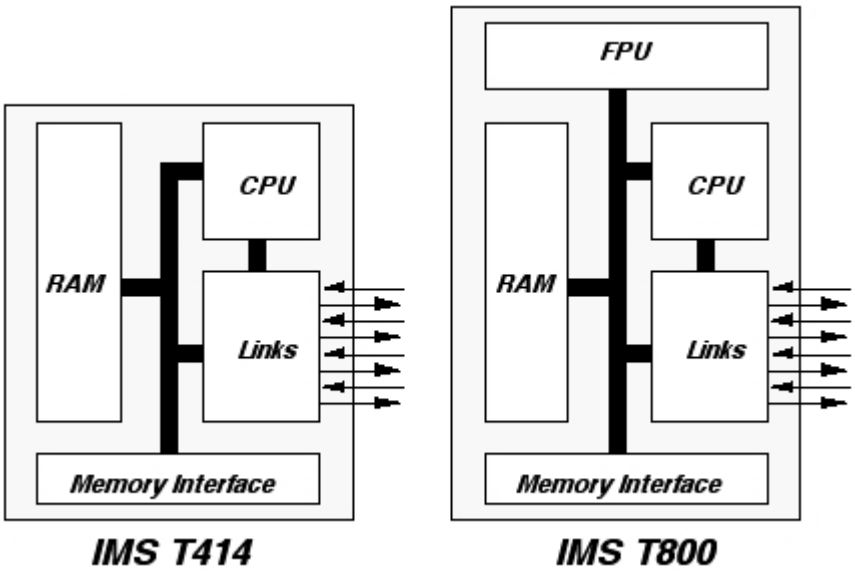
A transputer is a large number of simple processors which are connected together like a mesh. Each processor (often only a 1 bit processor) has 4 others around it in the directions of north, east, south and west. The computer works essentially like a RISC chip (reduced instruction set computing). What this means is that in computing terms it is faster to add one 64 times that to count to 64 in 1s. The first requires less memory as a running count is not needed and the total is reached in a few clock pulses whereas the second case like most PCs takes at least 64 pulses and a bit of memory. INMOS introduced the first transputer in 1985.

The transputer architecture is very flexible as several thousand of the processors can be linked together in any physical configuration to give a customised computer for many different situations which can be easily changed for the next situation. However as the processors are in a mesh they don't have any direct input/output with the outside world, only each other and so a special programming language is required. A common one is called OCCAM. The disadvantages of the special language are dwarfed by the advantages of very fast speed and flexibility. (The processors can only add 1 number or pass on information. That's how simple they are.)

They are used today in very fast PCs which contain RISC processors and in mainframe computers. The PC versions easily outperform the fastest Pentiums.

Transputer Architecture

The basic transputer (the original T414, for example) consisted of a conventional, sequential, RISC processor, a communication subsystem (implemented as four high-speed, inter-processor links), 4 k of on-chip RAM and an on-chip memory interface.

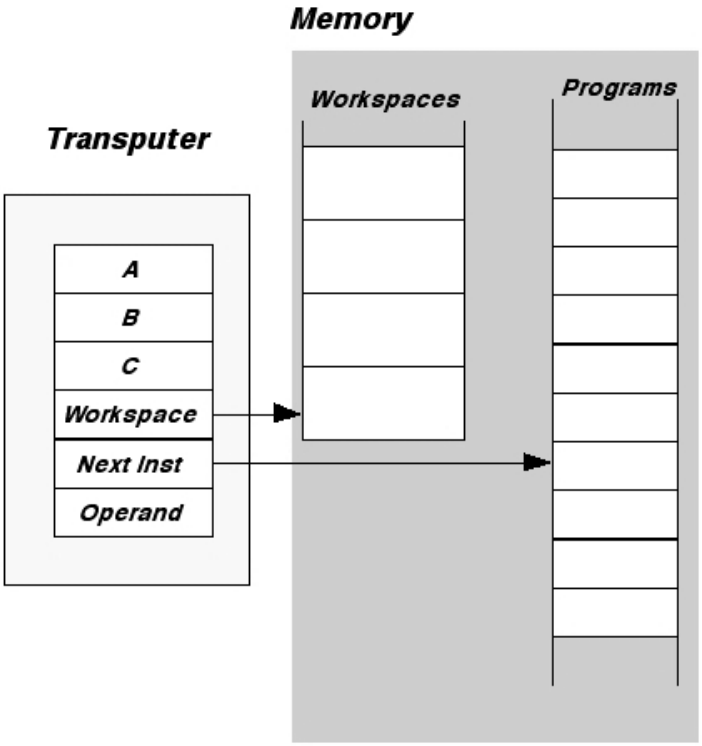


Functional Diagram

The processor had only three general-purpose registers, A, B and C. These registers are treated as a stack by the transputer instruction set. Loading a value into A pushes the previous contents of A into B, and the previous contents of B into C. Similarly, storing a value from A pops B into A and C into B. The instruction set works with this register stack implicitly.

Performing an add instruction adds the top two operands on the stack. In other words, an add instruction always adds B to A, leaving the result in A. There is no mechanism provided to ensure that a stack overflow does not occur. This is left to the compiler or assembly language programmer.

A similar programming model exists for the floating point unit found in the T8-series of transputers. In addition to the three general-purpose registers, the transputer also has a workspace pointer, an instruction pointer and an operand register. The workspace pointer points to a region of memory where the parameters of the currently executing task are stored. The instruction pointer references the next instruction to be fetched and executed by the processor. The operand register is used in the formation of instruction operands (the transputer's instruction set is somewhat unusual in this respect). The programmer's model is



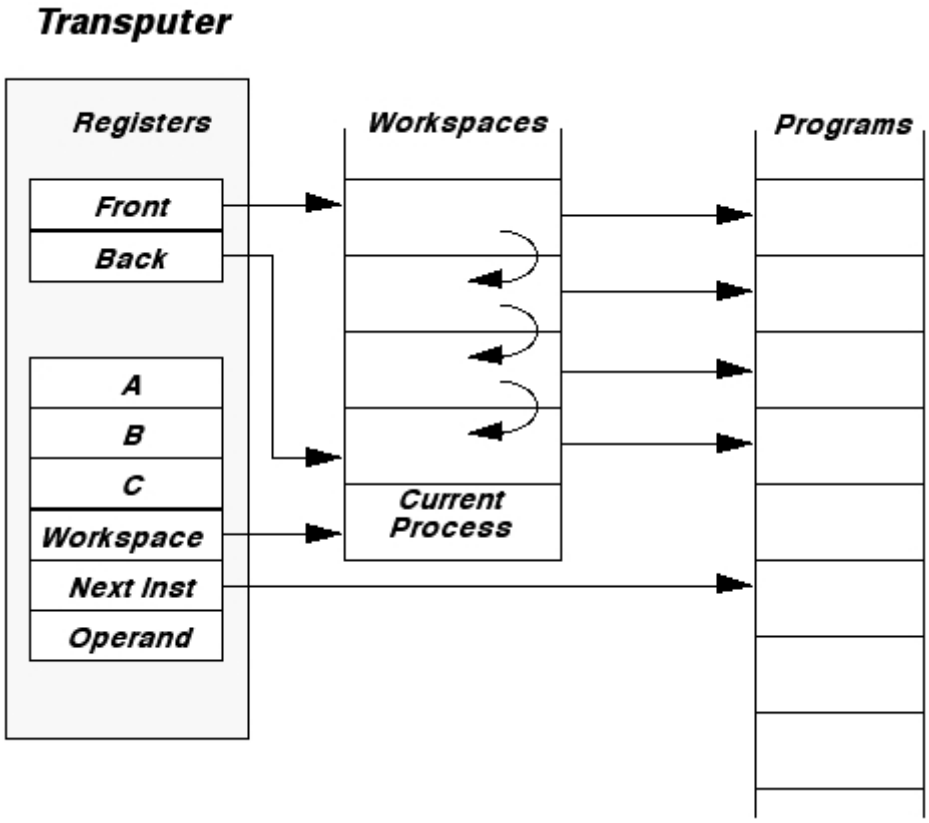
shown below.

Transputer Programmer's Model

It can be seen from the above that the transputer programming model is a simple one. The transputer exploited the high-speed, internal RAM to overcome the limitations of a small register set. The small register set and RISC instructions meant that the transputer had simple and fast data-paths and control logic.

The instruction set of the transputer is an unusual one. A design decision was taken early in the transputer's development that the transputer should be programmed in a high-level language. One look at the instruction set for these machines is sufficient to convince even the most die-hard of assembly-language programmers to follow the INMOS suggestion. The instruction set contains a relatively small number of instructions. Each instruction is one byte long. The upper four bits of the opcode specify the function to be performed, the lower four bits contain the data. Two of the function codes allow the operand of an instruction to be extended to any length up to the size of the operand register.

The transputer supported two priority levels for executing tasks. High-priority tasks took precendence over low-priority ones. A low-priority task will only be executed if there is no high-priority task in the schedule. At any one time, a process may be either active (executing or awaiting execution) or inactive (waiting for I/O, waiting until a specified time or waiting for a semaphore). The architecture of the transputer is such that inactive processes do not use any processor time. The transputer had two registers which point to a linked list of workspaces. This list of workspaces constitute the process table. The task switching time is very small and hence the transputer implementation of multi-tasking is a very efficient one. The transputer instruction set directly supports process creation and termination.



Programmer's Model of the Transputer Process Table