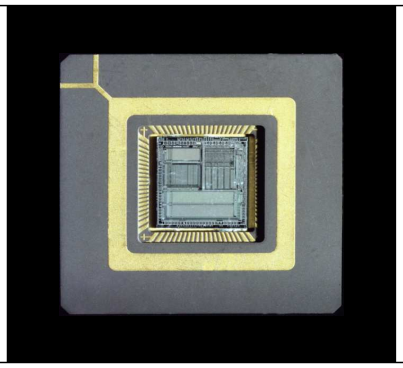
TUD Department of VLSI-Design, Diagnostic and Architecture

Department Seminar - Wed.12.Jun.2013

Transputer Architecture

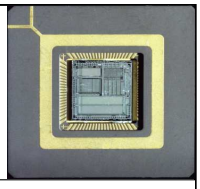


the Fascination of early, true Parallel Computing (1983)

INF 1096 - 2:50pm-4:20pm

Speaker: Dipl.-Ing. Uwe Mielke

the Transputer & me …



- I‘ve graduated 1984 at TU Ilmenau,
- my Diploma thesis was about „***the formal Petri-Net description and programming of a real time operating system kernel for embedded applications***“ @ Z80 (8bit CPU).

- Same time the Transputer appeared!
- **AMAZING** !! *Realtime in Silicon* !

Helmut Grubmüller, Helmut Heller, Klaus Schulten

Eine Cray für „jedermann“

Supercomputer aus München: Eine Cray für 100 000 DM

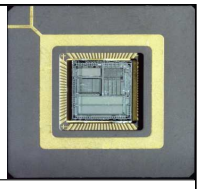
- No wonder that I liked to read such papers all over the years…
- Since 2006 I‘m collecting Transputer infos & artefacts for their revitalization ☺ … Your questions? → uwe.mielke@infineon.com

→ Target for next 60 Min.s: …give you an idea about the impressive capabilities of the Transputer Architecture…

“The Inmos Transputer *was more than a family of processor chips; it was a concept, a new way of looking at system design problems. In many ways that concept lives on in the hardware design houses of today, using macrocells and programmable logic. New Intellectual Property (IP) design houses now specialise in the market the transputer originally addressed, but in many cases the multi-threaded software written for that hardware is still designed and written using the techniques of the earlier sequential systems.”*

[Co99] **The Legacy of the transputer** – Ruth IVIMEY-COOK, Senior Engineer, ARM Ltd, 90 Fulbourn Road, Cherry Hinton, Cambridge – in: **Architectures, Languages and Techniques**, B. M. Cook(ed.) IOSPress, 1999

Agenda



Introduction

- INMOS & the IBM PC Era : some technical trends 198x, INMOS History
- Transputer Foundations : CSP & Occam, Persona
- The birth of the T414 : 1983

Transputer Architecture

- Hardware Details : CPU, Registers, Address Space, Links
- Instruction Set : Format, PFix & NFix, OpCodes
- Process Model : Queues, Events, Descheduling Points

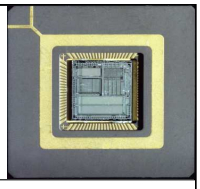
Occam in Silicon

- Process Example : Buffer Process
- Transputer Execution : Input & Output Communication

Outlook

- (missing topics) : T9000, IEEE-1355, Occam-Pi, ST20, XMOS

1. Introduction



parsytec *GigaCluster*

Supercomputer zum Anfassen *…*

… heute (seit 2004) im Heinz Nixdorf Computer Museumsforum in Paderborn.

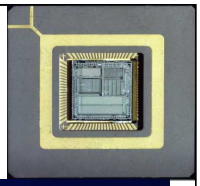
Das System hatte zwölf Jahre (1992-2004) im PC² (Paderborn Center for Parallel Computing) bis zuletzt treue Dienste geleistet.

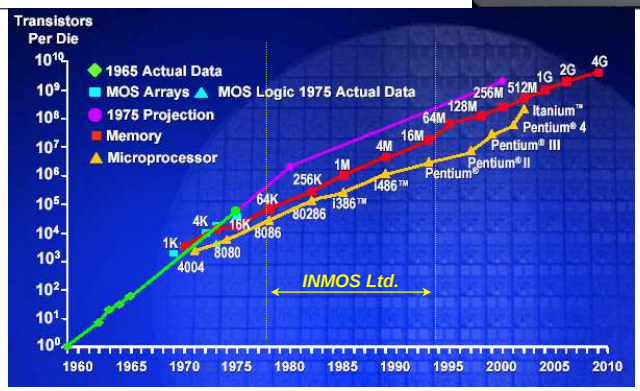
1992 *stand der Parsytec-GC auf* ***Platz 259*** *in der Liste der* ***Top500*** *Supercomputer.*

Die Rechenleistung der ***1024*** *Transputer à 30 MHz mit je 4,4 MFLOP/s, also insgesamt etwa 4,5 GFLOP/s, wird heutzutage von jedem bessern Laptop erreicht -- der GC benötigte dafür ein Gehäuse von 2,6 m Höhe und 2,53 m Breite.*

c't *Nov.2004*

1.1 the IBM PC Era
technical Trends ~ 198x





year	size(Mb)	cyc time
1980	0.0625	250 ns
1983	0.25	220 ns
1986	1	190 ns
1989	4	165 ns
1992	16	145 ns
1996	64	120 ns
2000	256	100 ns
2003	1024	60 ns

1981: IBM-PC, i8088, 4.77MHz, 512kB RAM, price: 5000US$

1984: PC-AT, i286, 6MHz, 640kB RAM, 20MB HDD, price: 4000US$

1985: C++ object oriented Language came up…

1987: IBM/PS2, i386, 16MHz, 4MB RAM, 40MB HDD, price: 3000US$

1989: i486 & i860 released same time…

1990: Windows 3.0 released (on top of MS-DOS 6.2)

INMOS COMPANY HISTORY

1978 founded as UK (Labour-)Government owned Memory Company, development of Memory Products (SRAM, DRAM) w/ great market success

1980 development of Occam Progr.Language based on C.A.Hoare's CSP Theory

1983 development of the 1st Occam based 32bit Transputer successfully finished

1984 T414 (15MHz) released to the market, Occam as assembly language

1985 over 150 1st class Patents about Semiconductor Manufacturing and Computer Engineering show strong INMOS → e.g. 100% patent exchange agreement w/ IBM

1985 1st privatization → Thorn EMI Industries Ltd. (by M.Thatcher Government for cash … no further investments nor subsidaries)

1986 US Memory Fab reliability Crisis → US Mgmt. fired, due to financial problems the Bristol development headcount has to be cut down by 50%

1987 Development of IEEE754 64bit FPU successfully finished (ESPRIT founded)

1988 T800 (20MHz) released to the market

1989 2nd privatization → ST Micro

1990 ESPRIT project to develop next generation transputer and router chips

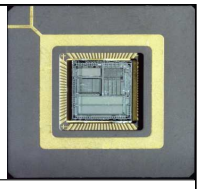
1993 shut down of T9000 (out of order execution) after 3 yrs development

1995 the ST20450 (40MHz) was released

1998 ST Micro announced the closure of Transputer production.

2009 ST20 (200+MHz) widely used in ST Micro set top box products (STi51xx)

1.2 Transputer Foundations
CSP & Occam



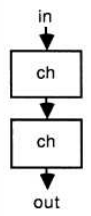
Communicating Sequential Processes (**CSP)** …

- was first described in a 1978 paper by C. A. R. Hoare. It evolved further in parallel with the development of Occam at INMOS.
- The full theoretical version of the CSP calculus was initially presented from in a 1984 article by Brookes, Hoare, and Roscoe, and later in Hoare's book *Communicating Sequential Processes*, which was published in 1985.

OCCAM as Programming Language …

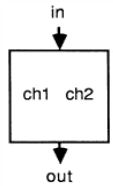
- was developed by David May at INMOS ~1980 together with the University of Oxford (C.A.R. „Tony“ Hoare) in terms of formal and provable correctness.

```
CHAN c:
PAR
  WHILE TRUE
    VAR ch:
    SEQ
      in ? ch
      c ! ch
  WHILE TRUE
    VAR ch:
    SEQ
      c ? ch
      out ! ch
```

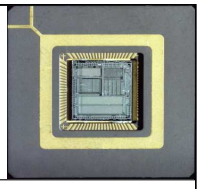


|| …

```
VAR ch1:
VAR ch2:
SEQ
 in ? ch1
 WHILE TRUE
   SEQ
     PAR
       in ? ch2
       out ! ch1
     PAR
       in ? ch1
       out ! ch2
```



1.2 Transputer Foundations
Occam



Statements:

- A **Process** is a piece of code having an *Input* and providing an *Output.*
- Processes communicate by Point-to-Point *Messages* (1…n Bytes) via *Channels.*
- A **Channel** is an *Address in Memory* on the same … or another Transputer.
- A Channel between 2 Transputers is formed by a *serial Link.* The Link will automatically drop („DMA“) the Message in the memory of the other Transputer.
- Communication will be *synchronized*, i.e. when sender AND receiver both are ready. The Process which is ready for Communication first … has to wait for its partner.
- The programmer has not to take care about how Messages are transfered !

- Process execution on Transputers is *Event-driven*, i.e. Processes which are waiting for an Event do not consume any processor time. **Events** can be caused by Communication, Timer-Setup or extenal Interrupt(s).
- **Occam** provides all necessary primitives for *Process Syncronization* (incl. Start, End, Alternative, …) and *Process Communication.*
- The programmer should focus on his Program Structure & Algorithms !

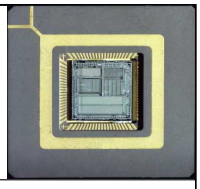
ieee-1985 the transputer – INMOS:

The architecture of the transputer is defined by reference to occam. Occam provides the model of concurrency and communication for all transputer systems. Defining the architecture at this level leaves open the option of using different pro*cessor designs* in different transputer products. This allows implementations which are optimized for different purposes. It also allows implementations to evolve with changes in technology, without compromising the standards established by the architecture.

A transputer contains memory, a processor and a number of standard point-to-point communication links which allow direct connection to other transputers.

In the transputer architecture, the exploitation of a high degree of concurrency is made possible through a decentralized model of *computation,* in which local computation takes place on local data, and concurrent processes communicate by passing messages on point to point channels.

1.2 Transputer Foundations
Occam



- **OCCAM** enables a system to be described as a collection of concurrent processes, which communicate with each other through channels.

- OCCAM programs are built from three primitive processes:
  - **`x := exp`** assign expression **`exp`** to variable **`x`**
  - **`ch1 ! exp`** output expression **`exp`** to channel **`ch1`**
  - **`ch2 ? x`** input from channel **`ch2`** to variable **`x`**

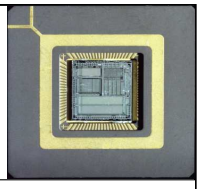
- The primitive processes are combined to form constructs:
  - **`SEQ`** uential execute processes one after another
  - **`PAR`** allel execute processes concurrently
  - **`ALT`** ernative execute only the first ready process

- **`IF`** and **`WHILE`** and **`CASE`** constructs are also provided.

- A construct is itself a process, and may be used as a component of another construct.

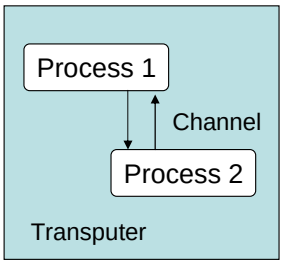
(see Links for free download in Appendix)

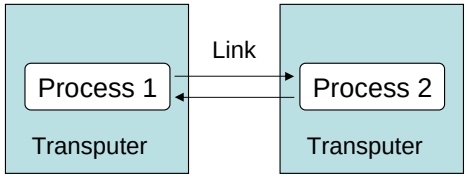
1.2 Transputer Foundations
CSP & Occam



Communication via Channels in Occam …

- can be between 2 processes on the same transputer or between 2 processes on different transputers,
- looks for the programmer all like the same (fully transparent),
- is synchronized, i.e. if sender and receiver both are ready the communication takes place.





1.2 Transputer Foundations Persona

William of Ockham (1287-1347):
"Entities should not be multiplied unnecessarily.„
→ keep it simple !

Iann Barron (born in June 1936)

Developed several Mini Computers, including the „Modulat-One". Visioneer and entrepreneur, initial founder of INMOS and CEO.

Tony (C.A.R.) Hoare (born 11.Jan.1934)

Quicksort algorithm originator. Since 1977 Professor of Computer Science at University of Oxford … today Fellow at Microsoft

David May (born 24.Feb.1951)

Joined 1978 INMOS microcomputer architecture team, since 1995 Prof. of Computer Science at Bristol Uni, 2006 Co-Founder of XMOS, CTO.

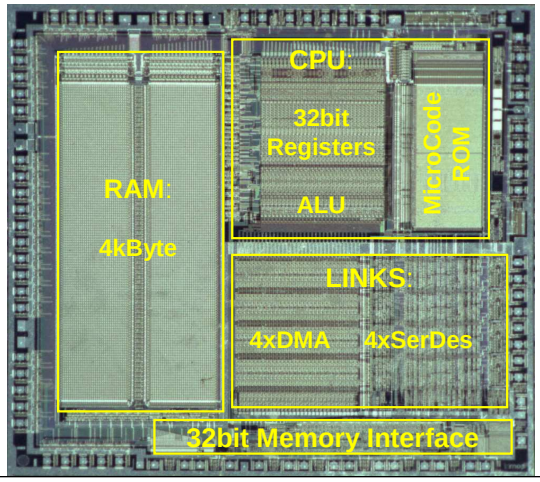
„…what they all wanted was a new simplicity in computers, in their structure and in the languages used to program them. In this context simplicity need not be the enemy of performance."

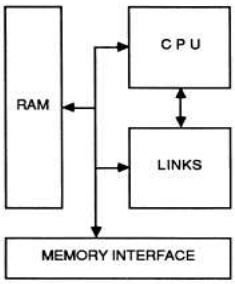
[LR85] M.McLean and T.Rowland „The Challenge of the Transputer",
Chapter 9 from „THE INMOS SAGA - A Triumph of National Enterprise?", © 1985

1.3 T414
Birthday 1983

1982 : the „Simple-42“ design completed
1983 : successfully 1st prototyping of T414A
1984 : redesign T414B (2 bugfixes)
1985 : volume production

Technology:	**1.5µm CMOS**
Clock (int.):	**15…20MHz**
Chip Size:	**8.5 x 8.3mm²**
Power Supply :	**+5V ±5%**
Packaging:	**CPGA 84**
Production:	**1985**
Price (1886):	**???**





IMS T414

Originally the plan was to make the transputer cost only a few dollars per unit. Inmos saw them being used for practically everything, from operating as the main CPU for a computer to acting as a channel controller for disk drives in the same machine. Spare cycles on any of these transputers could be used for other tasks, greatly increasing the overall performance of the machines.

Even a single transputer would have all the circuitry needed to work by itself, a feature more commonly associated with microcontrollers. The intention was to allow transputers to be connected together as easily as possible, without the requirement for a complex bus (or motherboard). Power and a simple clock signal had to be supplied, but little else: RAM, a RAM controller, bus support and even an RTOS were all built in.

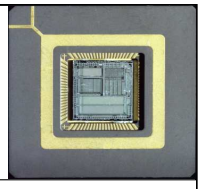
The **occam** language [xx] allows a system to be hierarchically decomposed into a collection of concurrent processes communicating via channels.

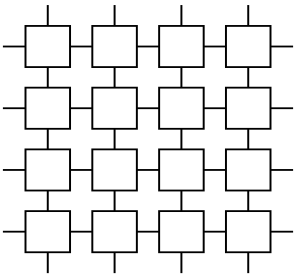
An occam **program** can be implemented by a single Transputer, or by a collection of Transputers each executing one or more occam processes.

… but the British designers were only to receive three batches of working silicon prototypes of the transputer during 1983.

Finally production start was 1985 in Bristol … compeeting with the start of intel 386 and Motorola 68000

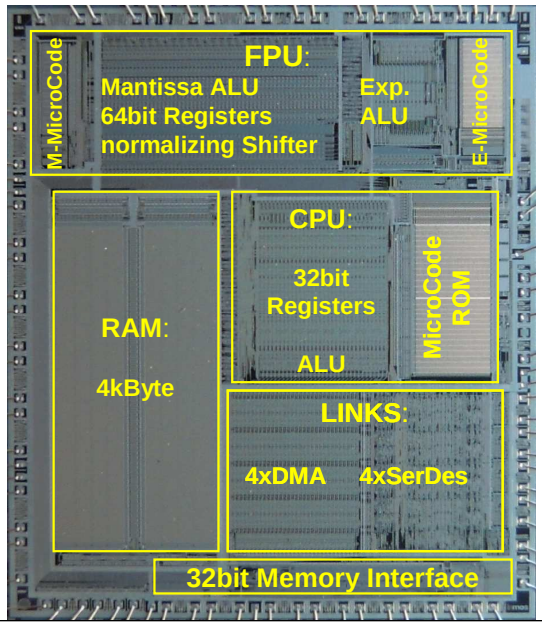
2. Transputer Architecture



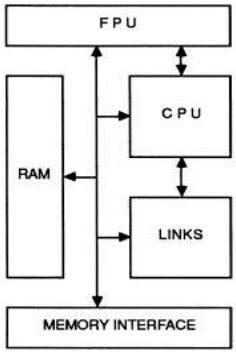


D.May: *„Occam and the Transputer are designed for each other. The mathematical formalism of Occam provides the concurrency- and communication-model for the Transputer‘s hardware“*

2.1 Hardware
T800, T805



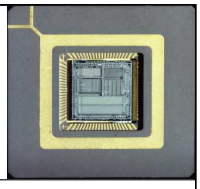
Technology:	**1.5µm CMOS**
Clock (int.):	**20MHz**
Chip Size:	**8.5 x 10.7mm²**
Power Supply :	**+5V ±5%**
Packaging:	**CPGA 84**
Production:	**1988**
Price (Nov.1988):	**1042,25 DM**



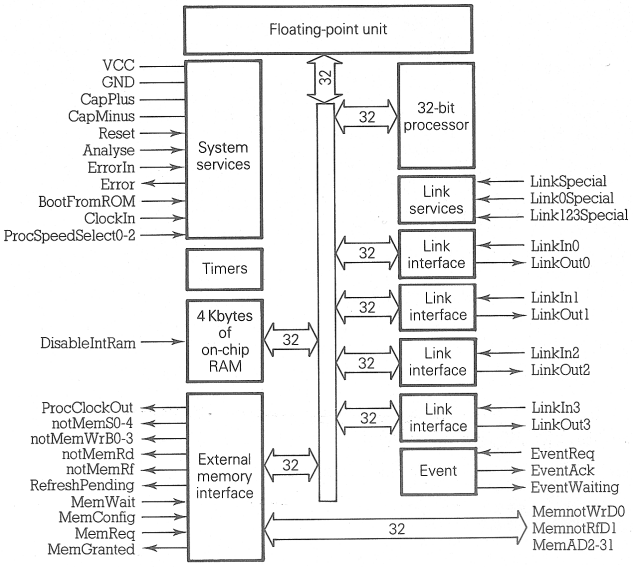
IMS T800

32 bit architecture
50 ns internal cycle time (20 MHz)
20 MIPS (peak) instruction rate
2.8 Mflops (peak) instruction rate
Pin compatible with IMS T4xx
Debugging support
64 bit on-chip floating point unit which conforms to IEEE 754
4 Kbytes on-chip static RAM
120 Mbytes/sec sustained data rate to internal memory
4 Gbytes directly addressable external memory
26.7 Mbytes/sec sustained data rate to external memory
950 ns response to interrupts
Four INMOS serial links 5/10/20 Mbits/sec
Bi-directional data rate of 2.4 Mbytes/sec per link
High performance graphics support with block move instructions
Boot from ROM or communication links
Single 5 MHz clock input
Single +5V 5% power supply
Packaging 84 pin PGA / 100 pin CQFP

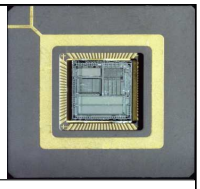
2.1 Hardware
T805 Block Diagram



- **32bit CPU + 64bit FPU**
- most instructions only 1 clock
- included: **Process Scheduler** w/ internal Communication Channels, Links and Timers.
- included: 4KByte **SRAM**, one clock cycle access time, register like quality.
- included: **Memory Interface** (programable) for easy to use RAS+CAS generation and direct connection of 8…16 dRAM Devices, full **4GByte** Address Space.
- **Event**-Handler for fast, deterministic Interrupt response time: 950ns@20MHz



2.1 Hardware Details
CPU: Registers



The **CPU** contains:

- sequential 32bit Integer **Processor**
- (micro-coded) **Scheduler** & **Timers**
- **Event Logic**

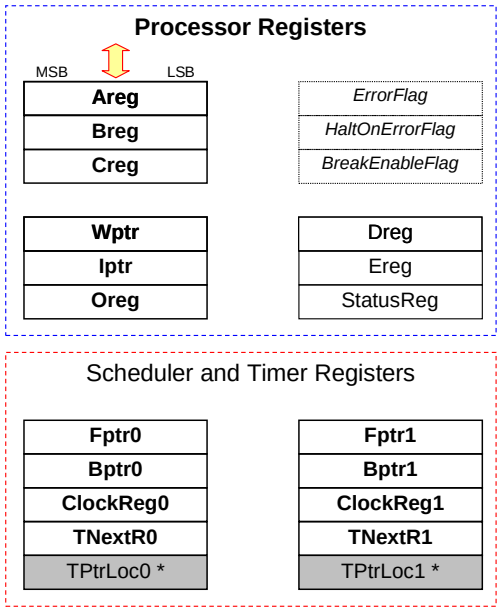
Reverse Polish Notation

Processor Registers:

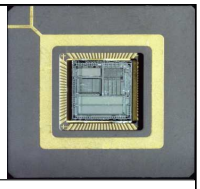
- Evaluation Stack (RPN) : **Areg**, **Breg**, **Creg**
- Workspace Pointer: **Wptr**
- Instruction Pointer: **Iptr**
- Operand Register: **Oreg**
- Flags: *Error, HaltOnError, BreakEnable*
- Internal Registers: Dreg, Ereg, StatusReg

Scheduler and Timer Registers

- Front- and Back-Pointers of high and low priority process queues: **FptrX**, **BprtX**
- Timer Counter (actual) and Timer Next Event Registers for high and low priority process queues: **ClockRegX, TNextX.**
- Timer Queue Pointers: TPtrLocX (* in Memory)

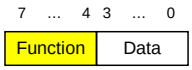


2.2 Instruction Set Format

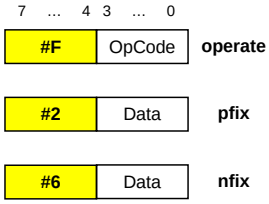


Instruction Format:

- 8 bit Op-Codes – very compact !
- Reason: due to statistics … 70% of all program code consist of load and store instructions with almost small operands.



- 4 bit Function Code = 16 instructions
- 4 bit Data Part … values #0…#F
- Function Code #F (operate) uses Data as function as well → +15 instructions
- 2 Functions Codes (**Pfix**, **NFix**) are used to extend Data Part w/ **Oreg …**
  - up to 32bit (for function #0…#E) as direct operand
  - up to 8…12bit (for function #F) as OpCode for further instructions



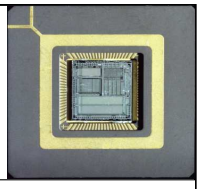
All transputers share the same basic instruction set. It contains a small number of instructions, all with the same format, chosen to give a compact representation of the operations most frequently occuring in programs. Each instruction consists of a single byte divided into two four bit parts.

The four most signicant bits are a function code, and the four least signicant bits are a data value. The sixteen functions include loads, stores, jumps and calls and enable the most common instructions to be represented in a single byte.

As this encoding permits only 4 bits of operand per instruction two of the function codes (prex and negative prex) are used to allow the data part of any instruction to be extended in length.

Another of the sixteen functions (operate) treats its data portion as an operation on values held in the processor registers. This allows up to 16 such operations to be encoded in a single byte instruction.

2.2 Instruction Set Overview



The **T414** has 100 instructions which can be grouped as follows [LM92]:

- 16 addressing and memory access instructions
- 6 branching and program control
- 41 arithmetic and logical
- 12 process scheduling and control
- 16 inter-process communication
- 9 miscellaneous

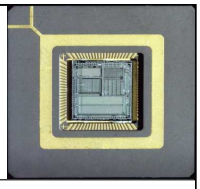
Only 4 Addressing Modi:

- immediate … constant is part of instruction (ldc := load constant)
- register-direct … register-to-register (e.g. within evaluation stack, …)
- register-indirect … address in register (either Wptr or Areg)
- register-relative … address and displacement in registers (Wptr and Areg)
- There are **two ways** of addressing memory, namely to specify the address as a fixed offset from the address in the workspace pointer (Wptr) **or** the A register.

The **T805** has 167 instructions, additionally are:

- 50 FPU instructions
- Special instructions … like 2D move for graphics applications
- Test & Analyze Support (j#0)

2.1 Hardware Details
CPU: Wptr, Iptr, Oreg



Registers are related to running Process
(process which is consuming CPU time)

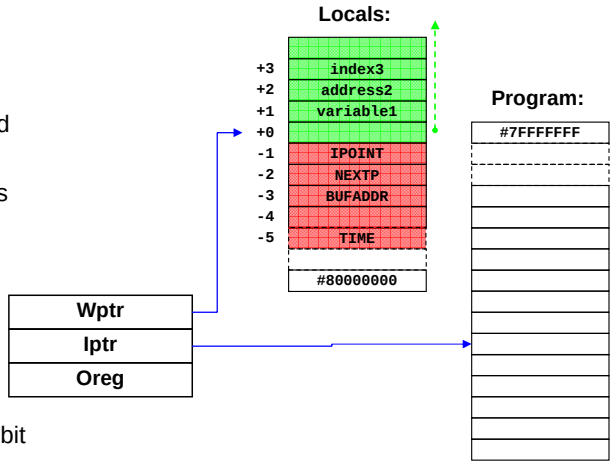
Instruction Pointer: **Iptr**

- points to next instruction to be executed

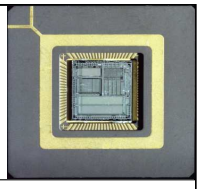
Workspace Pointer: **Wptr**

- points to Workspace of running process
- Wptr+0 … Wptr+x for *Program-Use*
  (very fast access to lower 16 words,
  4kB SRAM w/ Register Quality!)
- Wptr-1 …Wprt-5 for Process-Use

- Operand Register: **Oreg**
- used to extend the size of Operands (4bit …8…12…16…20…24…28…32bit)
- necessary to build more instruction codes by use of Prefixes



2.1 Hardware Details
CPU: Address Space



Address Space:

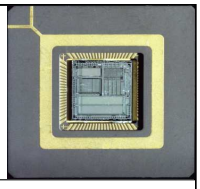
- highest: **MostPos** (most positive Integer)
- lowest: **MostNeg** (most negative Integer)
- totally **little Endian** Bit, Byte and Word Order
- single Byte **Write** is possible (Byte-Selector)
- **Read** always 32bit Word-wise (aligned)
- **internal RAM** at lowest Addresses

Reserved Locations:

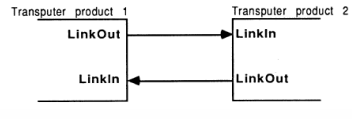
- Channel Control Words for **Link 0-3**
- Channel Control Word for **Event** channel
- Pointers to begin of high and low priority **Timer** queues: TPtrLocX
- **Interrupt Save Location** for (low Priority) processor status, in case of a high priority process is interrupting a low priority process.
- **Reserved for extended Functions** means: this area will be temporarily used by the processor during execution of 2D block move instructions, i.e. do not modify!

Machine Map	Byte address		Word offset	**Occam Map**
Reset Instr.	#7FFFFFFE			
	#7FFFFFF8			
	#7FFFFF6C			
	#00000000			
	#80001000	Start of ext.Memory	#0400	
	#80000070	**MemStart** (int.RAM)	#1C	
Reserved for extended functions	#8000006C			
	#80000048			
ERegIntSaveLoc	#80000044			
STATUSIntSaveLoc	#80000040			
CRegIntSaveLoc	#8000003C			
BRegIntSaveLoc	#80000038			
ARegIntSaveLoc	#80000034			
IptrIntSaveLoc	#80000030			
WdescIntSaveLoc	#8000002C			
TPtrLoc1	#80000028			
TPtrLoc0	#80000024			
Event	#80000020		#08	Event
Link 3 Input	#8000001C		#07	Link 3 Input
Link 2 Input	#80000018		#06	Link 2 Input
Link 1 Input	#80000014		#05	Link 1 Input
Link 0 Input	#80000010		#04	Link 0 Input
Link 3 Output	#8000000C		#03	Link 3 Output
Link 2 Output	#80000008		#02	Link 2 Output
Link 1 Output	#80000004		#01	Link 1 Output
Link 0 Output	#80000000	(Base of memory)	#00	Link 0 Output

2.1 Hardware Details
Links: Registers

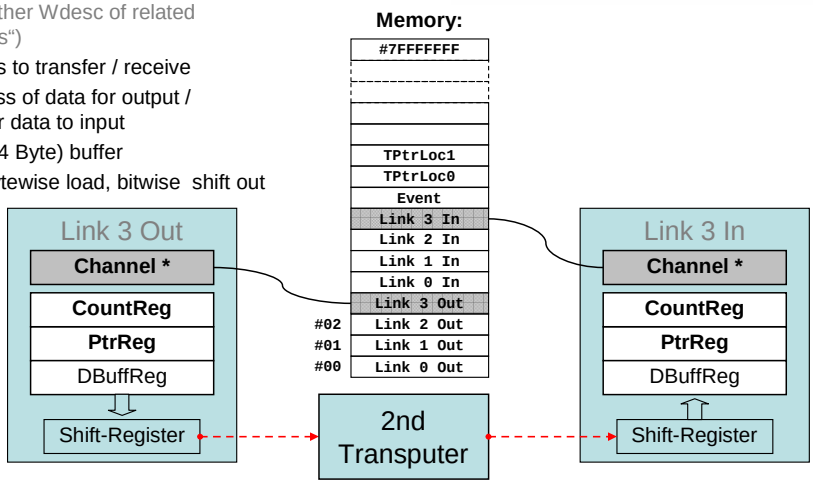


Transputers can be connected by their Links.

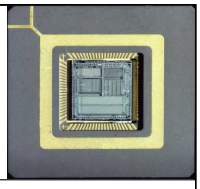


Each serial Link has an Input & an Output channel:

- **Channel**: channel control word → reserved location in memory (contains either Wdesc of related Process or „not.process“)
- **CountReg**: no. of bytes to transfer / receive
- **PtrReg**: Source Address of data for output / Destination Address for data to input
- DBuffReg: 32bit Data (4 Byte) buffer
- Shift-Register (8bit): bytewise load, bitwise shift out / in of data



2.1 Hardware Details
Links: Protocol



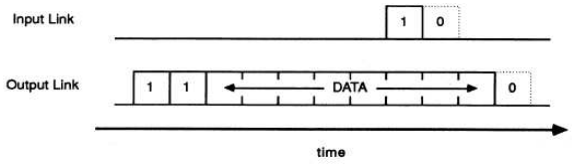
Each communication channel requires that all 4 input and output lines of the respective Links are connected.

Simple Link Protocol:

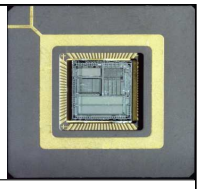
- 2 Start-Bits
- 8 Data-Bits
- 1 Stop-Bit

Each transfered Byte has to be confirmed by:

- 2 Acknowledge-Bits



2.3 Process Model
State Transitions (simplyfied)

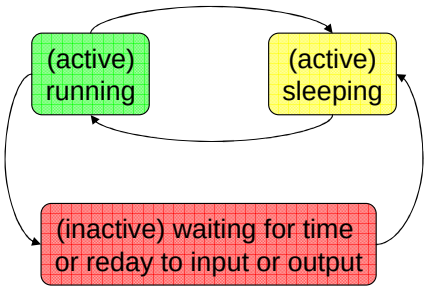


At any time, a concurrent process may be active

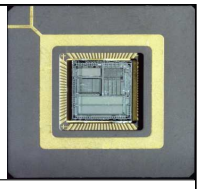
- being executed (running)
- on a list awaiting execution

inactive

- ready to input
- ready to output
- waiting until a specified time



2.3 Process Model
Wptr & Workspace Descriptor

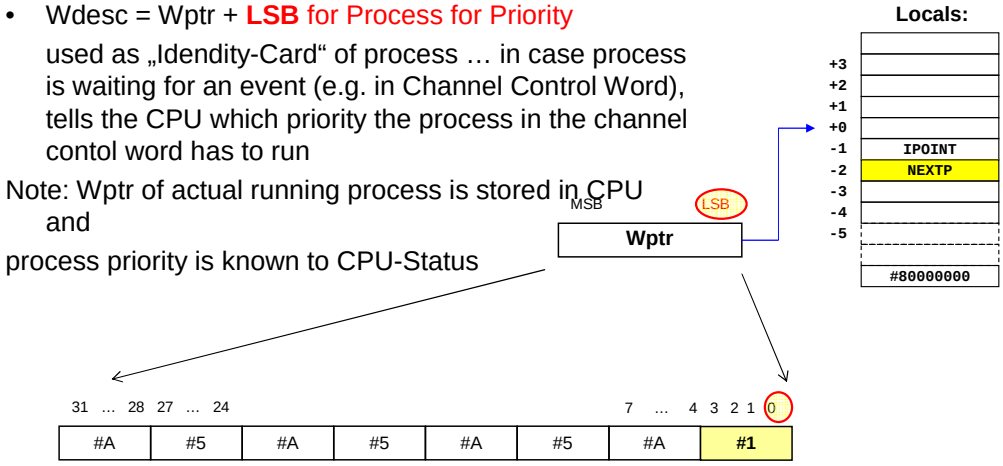


- **Wptr:** Workspace-Adress
- Wptr → lowest 2 bits always Zero !
- **Wdesc:** Workspace Descriptor
- Wdesc = Wptr + **LSB** for Process for Priority

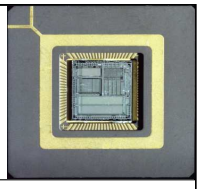
  used as „Idendity-Card“ of process … in case process is waiting for an event (e.g. in Channel Control Word), tells the CPU which priority the process in the channel contol word has to run

Note: Wptr of actual running process is stored in CPU and

process priority is known to CPU-Status

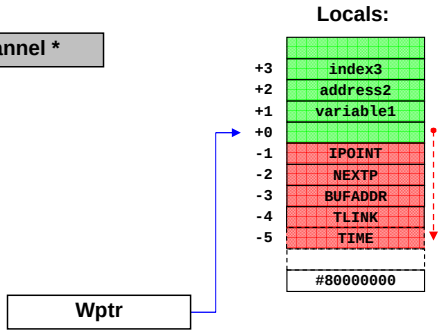


2.3 Process Model
Wptr & Process Status



- Process Status (Wdesc) is needed for pre-emptive Multitasking:
  Wptr (+Prio) is Id-card of process ! → **Channel ***

  In case a Process becomes **descheduled** … the Locations below Wptr are used as follows:
- **-1 IPOINT**: points to next instruction of a descheduled Process, i.e. form here the process can be continued
- **-2 NEXTP**: points to Wptr of next Process, if in lo/hi Prio Process Queue (active-waiting)
- **-3 BUFADDR**: used during channel communication, points to data to be transferred
- **-4 TLINK:** points to Wptr of next Process, if in lo/hi Prio Timer Queue (-or- … TALT Flag)
- **-5 TIME:** time value the process is waiting for, if in lo/hi Prio Timer Queue



The least significant bit instead is used to store the process priority, which is 0 for a high priority and 1 for a low priority. This combination of the workspace address and the priority bit is referred to as the process descriptor.

A few words of memory just below the workspace pointer are used by various parts of the scheduling hardware as follows (relative to address pointed to by Wptr) :

-1 holds the IPtr of a descheduled process

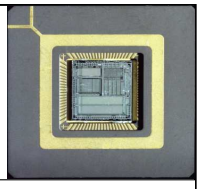
-2 maintain a list of active but descheduled processes.

-3 Used during channel communication to hold the address of the data to be transferred.

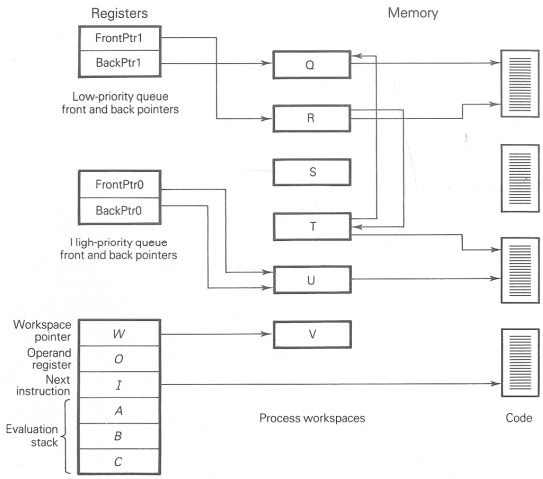
-4 flag used during timer ALTs to indicate a valid time to wait for.

-5 used during time ALTs to hold a time to wait for.

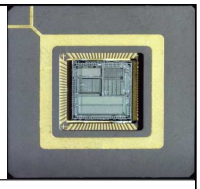
2.3 Process Model
Process Queues



- **2 Process Queues**: one for high priority and one for low priority processes
- Queues are organized as linked List's, Fptr is pointing to top of queue and Bptr to bottom of queue, i.e.:
- **Fptr** contains Wdesc of next process to become scheduled
- **Bptr** contains Wdesc of last process which has been descheduled
- The linked list is organized via Wptr-2 of each process in queue



2.3 Process Model
Timer Queues



- **2 Timer Queues**: one for high priority and one for low priority processes, organized as linked List's, **TPtrLoc** is containing the Wdesc of the process, which is next to be waked up

High priority Timer:

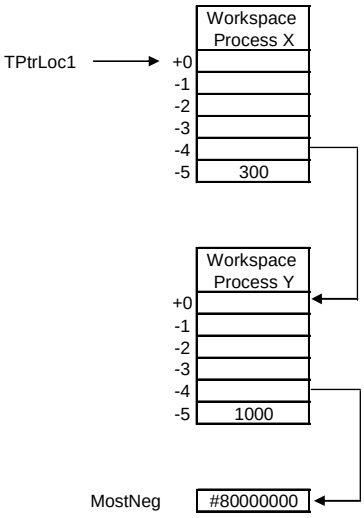
- one increment (tick) every 1 µSec

Low priority Timer:

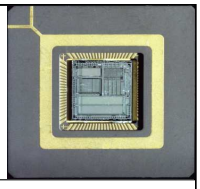
- one increment (tick) every 64 µSec
- If a low prio process exceeds his general time slot of 1 Millisecond it will be descheduled during next timeslot

Timer Registers Definitions:

- ClkReg +1 < Future < ClkReg + MostPos
- ClkReg > Past > ClkReg + MostNeg
- can be RESET or read … but not written



2.3 Process Model
Descheduling Points



- in general all instructions run as „*Atomic Operation*“, i.e. only at dedicated instructions (j, lend, in, out, outb, outw, altwt, taltw, tin), so called **Descheduling Points,** the scheduler can put a low prio process to sleep, e.g. if the process has exceeded his 1ms time slot.
- The (Occam-) Compiler has to avoid endless atomic operations, i.e. if there are no loops at all … then from time to time there may be a NOP-like descheduling operation (j0) included
- Note: in case of Descheduling the registers and process Status will not be saved … only Iptr! Above Descheduling instructions ensure, that the evaluation stack is empty, all process owned variables and results have be saved in workspace already. Therefore process switching time is incredible fast.
- A high prio process (e.g. ext. Event) allways can interrupt any running low prio process. A reserved SRAM area will be used to store all registers & the processor status. Interrupt response time is 19-58 clocks (due to the current running instructions has to be completed first!), i.e. 0.95-2.9µs @20MHz.

A process (low or high priority) will be descheduled when one of the following conditions occur:

1) The process executes an instruction in order to communicate with another process.

2) The process executes the TIN (=Timer Input) instruction which causes it to wait until a specified time. In the case of interprocess communication the process will then be put on the list of inactive processes for that priority. Here the back-of-the-list pointer is used. One of the differences between low- and high-priority processes is that low-priority processes must share the CPU (pre-emptive multitasking). So, when the process is a low-priority process, there is another condition under which the process will be descheduled.

3) The low-priority process has used up all its time-slice.

Low priority processes are subject to round-robin scheduling with a time-slice period of about 1 ms in a T800. But there is a limitation: descheduling due to the expiration of a time-slice can *only* happen after the execution of certain instructions. These instructions are:

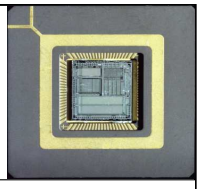
--- an unconditional jump (J; jump)

--- a special instruction which is very often used in loops (LEND; Loop End)

--- several others (e.g. 2D block move, …, sqrt)

As a result, a particular low-priority process which cleverly avoids these instructions can dominate the other low-priority processes. On the other hand, the scheduler does not consume any CPU time for processes which are descheduled.

2.3 Process Model
Events & Descheduling Points



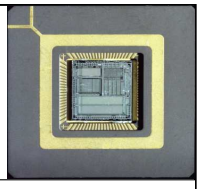
For the Transputer everything of the following is an Event:

- **Timer** Counter has reached a preset value
- **Input** communication request
- **Output** communication request
- **external Event** requires Interrupt

Channels are telling the system which process is related to which event.

→ So events can be handled completely by Hardware & Microcode, i.e. they are full transparent to the user.

2.4 System Services -in Arbeit-
Reset, Analyze, Boot



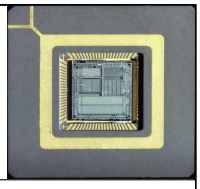
- No dedicated in-circuit Emulator required / avaliable at that time
- No MENTOR FastScan avail (intro 199x)
- The **Analyze**-Pin was used for Software Debugging, therefore exist …

2 Kinds of Reset:

1.) **Reset** w/o Analyze = normal PwrUp … internal Status is „virgin“

2.) **Reset** w/ Analyze = Debug-Mode … internal Status is preserved, communication is still completing, Processor halted awaiting Boot over Link

2.4 System Services -in Arbeit-
Boot over Link

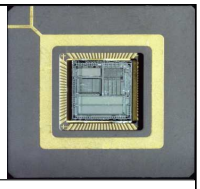


- Microcoded „**Boot over Link**“ Procedure:
  1st Byte = #0 → **poke** Operation: read next 8 Byte as address + data to write
  1st Byte = #1 → **peek** Operation: read next 4 Byte as address, output data
  1st Byte > #2 → **boot** Operation: 1st Byte = number bytes (<256) to receive
  ...write these Bytes @MemStart into internal memory and
  …**Start** this as program (e.g. Bootloader for larger Programs)
- i.e. consequently this can be used:
  … either for **Booting a whole big big system over a Worm** …
  …or Software debug after Analyze+Reset to read/modify processor status

- Example: ispy protocol of a 4 Transputer System incl. Memory & Linkspeed

```
Using 150 ispy 3.23 | mtest 3.22
   # Part rate Link# [  Link0  Link1  Link2  Link3 ] RAM,cycle
   0 T800d-25 288k 0 [   HOST    ...    ...    1:0 ] 4K,1 1024K,3;
   1 T425c-20 1.6M 0 [    0:3    2:0    3:0    ... ] 4K,1 4092K,3.
   2 T400c-20 1.7M 0 [    1:1    ...    ...    ... ] 2K,1 1022K,3.
   3 T400c-20 1.8M 0 [    1:2    ...    ...    ... ] 2K,1 4094K,3.
```

3. Occam

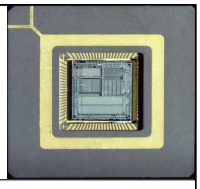


„Barron, Hoare and May went 1980 to a hotel for a week-long brainstorming session and returned with the specification for the new language.“

„Occam was just as revolutionary as any other aspect of the transputer. It was intended not just as a programming language but also as a means of describing the structure of a computing system.“

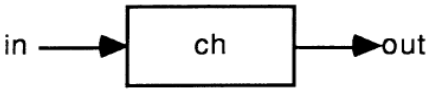
[LR85] M.McLean and T.Rowland „The Challenge of the Transputer“,
Chapter 9 from „THE INMOS SAGA - A Triumph of National Enterprise?“, © 1985

3. Occam as Assembly Language
Input & Output Example



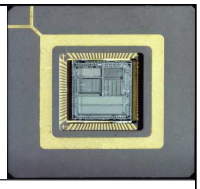
- A very simple example of an occam program is the buffer process:.

```
WHILE TRUE
   VAR ch:
   SEQ
      in ? ch
      out ! ch
```



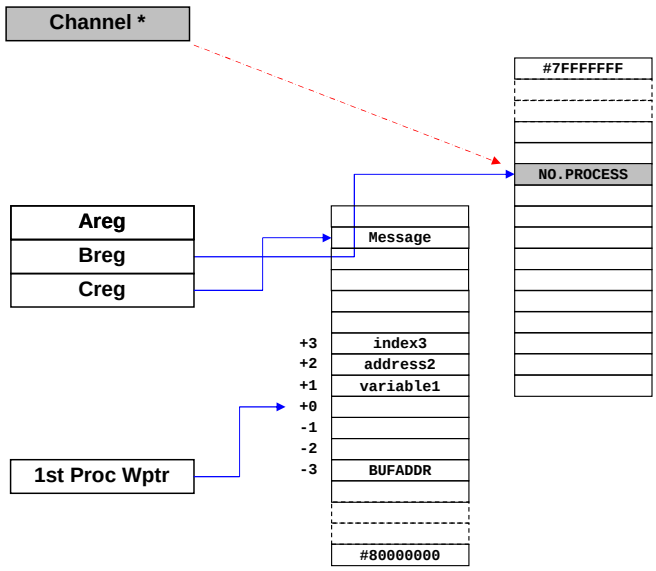
- ***Note**: No Brackets! Indentation is used to indicate the program structure!*
- The buffer consists of an endless loop, first setting the variable **`ch`** to a value from the channel **`in`**, and then outputting the value of **`ch`** to the channel **`out`**. The variable **`ch`** is declared by **`VAR ch`**.
- The direct correspondence between the program text and the pictorial representation is a useful starting point in the design of an efficiently implementable concurrent algorithm.

3. Occam as Assembly Language internal Channel Comm 1/4



- A Channel is a word in memory = Channel Control Word
- The channel can be marked as unused (empty) by a descriptor „no.process“ = #80000000

→ a channel is a **semaphore**

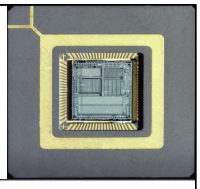


1. Begin of a Communication

- For input or output operation the CPU registers are:
- **Areg**: message length in Byte
- **Breg**: Channel Address
- **Creg**: Pointer to Databuffer

- Example: Lets consider the 1st Process is ready for **Output**

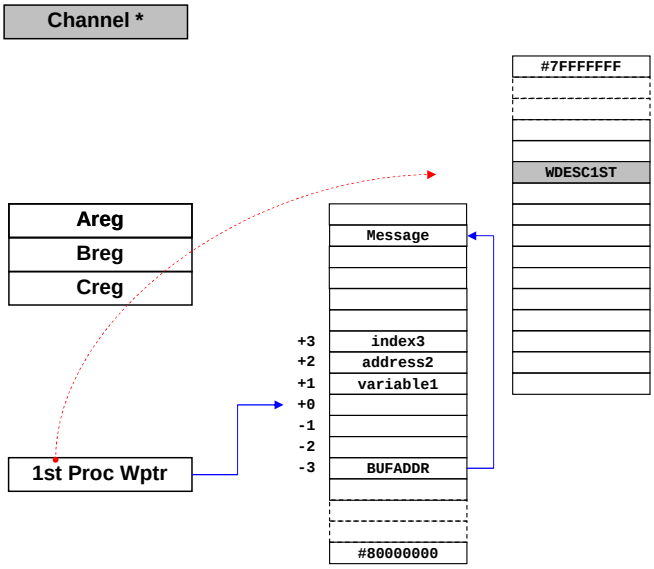
3. Occam as Assembly Language internal Channel Comm 2/4



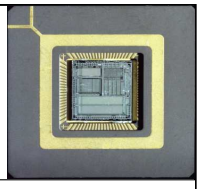
- If the Channel is empty, i.e. „no.process“ = #80000000, then the 1st (Output) Process „knows“ he has to wait for the 2nd (Input) Process to become ready for communication

2. Initialization of Communication

- 1st (Output) Process CPU registers will be written to:
- **Areg**: message length in Byte … will got lost ☹ (i.e. 2nd (Input) Process will determine no. of Bytes later … if not matching … then its programmers fault)
- **Breg**: Channel Address, → 1st (Output) Process will write his Wdesc into the Channel
- **Creg**: Pointer to Databuffer → will be written to own Wptr-3
- The 1st (Output) Process will now be descheduled … w/o queing into waiting list!



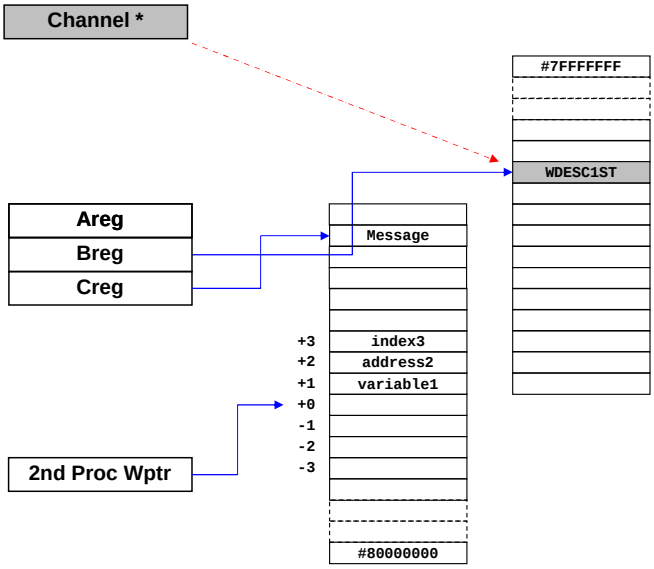
3. Occam as Assembly Language internal Channel Comm 3/4



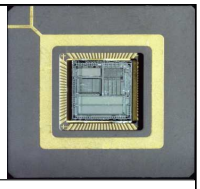
- If the 2nd (Input) Process becomes ready for communication … he will read the channel and detect his partner process is already waiting for communication

3. Execution of Communication

- 2nd (Input) Process is reading Wdesc = Wptr of 1st (Output) Process to find its data pointer @ Wptr-3 to read message…
- CPU registers of 2nd process:
- **Areg**: message length in Byte … will determine no. of Bytes now ☺ for data transfer
- **Breg**: Channel Address,
- **Creg**: Pointer to own Databuffer → here data will be written to now!



3. Occam as Assembly Language internal Channel Comm 4/4



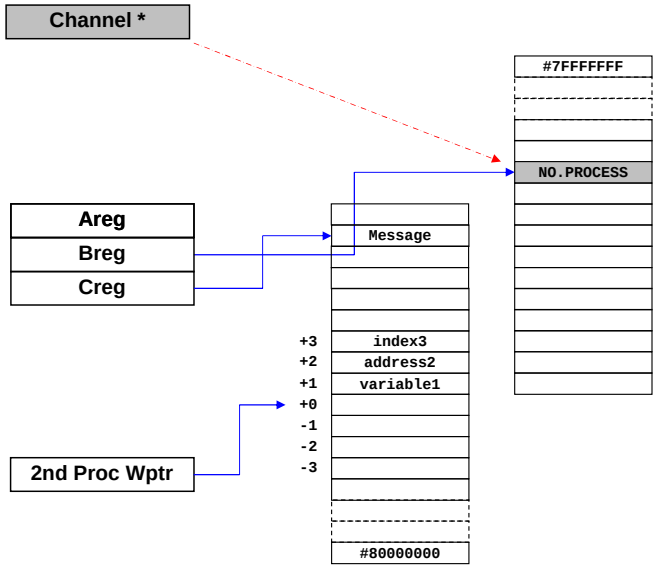
- The 2nd (Input) Process will transfer data from 1st (Output) workspace into his workspace

4. Finish of Communication

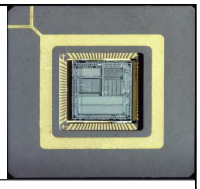
- CPU registers of 2nd process:
- **Areg**: message length in Byte … will determine no. of Bytes now ☺ for data transfer
- **Breg**: Channel Address,
- **Creg**: Pointer to own Databuffer → here data will be written to !

After all data have been copied:

- The Wdesc of 1st (Output) process will be added to the list of waiting processes (→Bptr)
- channel will be set back to empty → no.process = #80000000
- The 2nd (Input) process has finished communication and can continue w/ next instruction



3. Occam as Assembly Language internal Channel Comm …

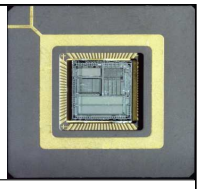


- Example was about Output Process arrives first.
- What will happen if **Input** Process arrives **first** ?

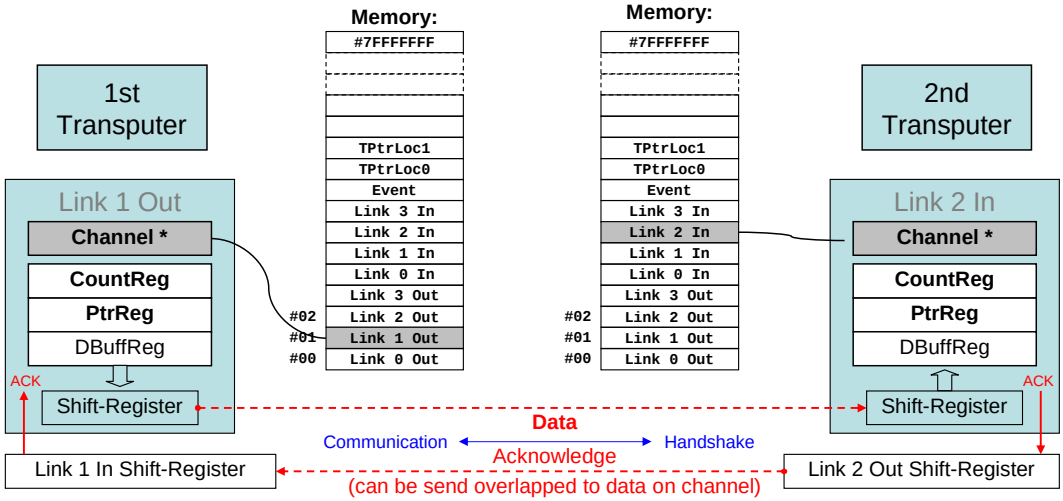
???

- Answer: the same procedure!
- in this case the 2nd (Output) Process has to do the copy job …
- i.e. always the „last“ process of both communication partners determines the number of bytes to be transfered.

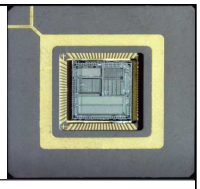
3. Occam as Assembly Language external Channel Comm



- (Link-)Channel is in reserved Memory area … process which arrives 1st has to wait
- Protocol: each received Byte will be acknowledged, but only if receiving process is ready!
- Sender can always send one (1st) Byte … but w/o acknowledge after … he has to wait!



3. Occam as Assembly Language
further Constructs …



Further available Occam Constructs in Microcode are:

- PAR
- ALT

→ These Constructs are more complicated, due to additional necessary counters for all inclued processes. Furthermore constructs with Timer contribution have to be considered different.

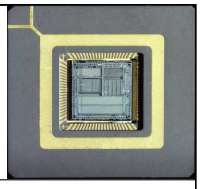
→ Therefore … pls see literature for detailed descriptions.

End of Presentation.

4. Outlook

„An interesting observation was made that the programmers with a background in hardware design fared better with the design of these highly parallel systems than did those with a traditional computer science background.“

4. Outlook
Discussion … missing Features



The Transputer is excellent for embedded (trusted) applications

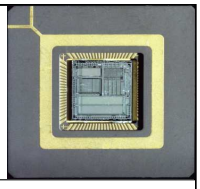
General purpose use is **handicaped** by …

- No MMU (memory protection between different applications on same chip impossible - but chip to chip 100% true)
- No more (finer grain) than two Priority Levels
- Virtual Channels (only in Software) to allow processor-independent process placement & move (as well to speed up serial communication)

Some of this lacks have been overcome by the T9000 + C104 design.

- Unfortunately the T9000 ooO-design-issues could not be solved in time
- The complicated T9000 chip never became productive ☹ with ist 10MHz
- Nevertheless a couple of MIMD machines (64 x T9000) have been built (CERN, University of Kent) and are still running … ☺

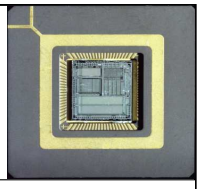
4. Outlook
open topics…



Open Topics … which could not be covered in this presentation:

- Transputer Chip Family, Peripherals, C004: 32 Channel Link Switch
- Transputer modular Industry Standards: Boards & TRAMs
- Transputer Development Systems
- further Programing Languages
- Operating Systems
- Transputer Main Applications+Markets (AddOn Boards, embedded, MIMD)
- 2nd Generation Transputers + Routers:
  - T9000, C104, the IEEE-1355 Spacewire Standard, IEEE-1394
  - ST20450 (1995), ST20 embedded CPU (200+MHz) up today
- 3rd Generation Occam / CSP Languages
  - Occam-Pi, Handel-C … HDLs for FPGA synthesis
  - KrOC – Kent retargetable Occam Compiler
- Transputer Emulator
- Today‘s Transputers: www.xmos.com

Literature, Sources, Links



General+History

- Paper: [Co99] R.Cook „The Legacy of the Transputer“ – http://www.wotug.org/papers/IvimeyCook_W22.pdf
- Book: [LR85] M.McLean and T.Rowland „The Challenge of the Transputer“, Chapter 9 from „THE INMOS SAGA - A Triumph of National Enterprise?“, free download: http://www.transputer.net/fbooks/saga/saga.pdf
- Interview: Iann Barron, „Inmos and the Transputer“, Part1 & 2: http://www.cs.man.ac.uk/CCS/res/res32.htm#c & http://www.cs.man.ac.uk/CCS/res/res33.htm#c
- [GHS88] H.Grubmüller, H.Heller, K.Schulten, „Superrechner – eine Cray für Jedermann“, mc.88.11.048-064, http://www.mpibpc.mpg.de/276339/paper_mc_1988.pdf
- [Me06] M.Helzle, „Transputer - das verkannte Genie“, COMPUTERPRAXIS 21.Jul.2006, http://www.embedd.it/downloads/Transputer%20-%20das%20verkannte%20Genie%20Juli%202006.pdf
- [Wa03] Paul Walker „the Origins of SpaceWire“, 2003, http://www.4links.co.uk/bibliography/Origins-of-SpaceWire-4Links-ESA-SpW-Conference-2003.pdf

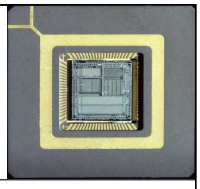
Documentation

- Wikipedia: http://en.wikipedia.org/wiki/Transputer
- Documentation: www.tranputer.net (Website of Michael Brüstle) → INMOS Datasheets & Technical Notes
- About Parallel: http://www.classiccmp.org/transputer/ (Ram Meenakshisundaram's Transputer Home Page) → Boards, Hardware, Software
- Intro: [St85] C.W.Strevens (INMOS), „the transputer“ IEEE 1995
- Intro: [Mo97] T.Modi, „Parallel Processing using Transputers“, http://teknirvana.com/documents/Transputers.pdf
- Review: [LM92] J.deLeeuw, A.deMes,"Transputers - design and use as a building block" http://www.science.uva.nl/~mes/psdocs/transputers.ps.gz%E2%80%8E
- Book: [RS91] H.Reinecke, J.Schreiner, „Transputer-Leitfaden – Eine Einführung und umfassende Beschreibung“, C.Hanser München 1991, ISBN-3-446-16063-9
- Book: [Eb93] Heinz Ebert, „Transputer und Occam, Das Handbuch für Systementwickler“ Heise 1993, ISBN-3-88229-0005
- Book: John Roberts, „Transputer Assembly Programming” 1992 transbook, ISBN-10 0-442-00872-4, free download: http://www.transputer.net/iset/pdf/transbook.pdf
- Homepage of Transputer-Architekt David May: http://www.cs.bris.ac.uk/~dave/index.html
- Book: Networks, Routers and Transputers, http://wotug.ukc.ac.uk/docs/nrat/book.psz.tar free download (Postscript-Format)
- The Transterpreter Project: http://www.transterpreter.org/Transputer
- Transputer-Emulator: https://sites.google.com/site/transputeremulator/

CSP+Occam

- Book: [Ho85] C.A.R.Hoare „Communicating Sequential Processes“, 21jun2004, ISBN-01-31-53289-8, free download: http://www.usingcsp.com
- Booklet: [Hy95] D.C.Hyde, „Introduction to the Programming Language Occam”, free download: http://www.eg.bucknell.edu/~cs366/occam.pdf
- Book: [PM86] D.Pountain, D.May, „A Tutorial Introduction to Occam Programming“, MacGraw-Hill NewYork 1986, ISBN-0-632-01847-X
- Book: [PR87] D.Pointain, R.Rudolph, „Occam - das Handbuch – Anleitung zum Programmieren paralleler Rechnersysteme“, Heise 1987, ISBN-3-88229-001-3
- Software: KrOC – the Kent retatgetable Occam Compiler, http://www.cs.kent.ac.uk/projects/ofa/kroc/
- WoTUG-Archive: http://www.wotug.org/parallel/ - World Transputer User Group, Proceedings & Papers

Literature, Sources, Links



some unsorted Papers … Outlook

- Paper: [RWW91] H.Roebbers, P.Welch, K.Wijbrans, „A generalized FFT algorithm on transputers“, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.2284
- Paper: Towards concurrency - occampi on LEGO Mindstorm http://www.cs.kent.ac.uk/pubs/2004/2004/content.pdf
- Paper: Roger Heeley, “The Application of the T9000 Transputer at CERN“ (1995), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.7796
- Interview: [Pa00] Ian Page, „Software to Silicon with HandelC“, https://www.doc.ic.ac.uk/~wl/teachlocal/arch2/ianpint.pdf
- Announcement: [Gu09] Guildford (University of Surrey), “Formal Verification of an Occam-to-FPGA Compiler and its Generated Logic Circuits”, http://www.surrey.ac.uk/computing/news/events/2009/formal_verification_of_an_occamtofpga_compiler_and_its_generated_logic_circuits.htm
- The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software http://www.gotw.ca/publications/concurrency-ddj.htm
- The Landscape of Parallel Computing Research: A View From Berkeley http://view.eecs.berkeley.edu/wiki/Main_Page

INMOS-Patents _→ www.patentgenius.com

- US-Pat-4730308 Interface between a computer bus and a serial packet link - 08Mar1988
- US-Pat-4758948 Microcomputer - INMOS 19Jul1988
- US-Pat-4783734 Computer with variable length process communication - INMOS 08Nov1988
- US-Pat-4811277 Communication interface - INMOS 07Mar1989
- US-Pat-4794526 Microcomputer with priority scheduling - INMOS 27Dec1988
- US-Pat-4811277 Communication interface - INMOS 07Mar1989
- US-Pat-4819151 Microcomputer - INMOS 04Apr1989
- US-Pat-4885740 Digital signal switch - INMOS 05Dec1989
- US-Pat-4967326 Microcomputer building block_30Oct1990
- US-Pat-4989133 System for executing time dependent processes - INMOS 29Jan1991
- US-Pat-5031092 Microcomputer with RAM in separate isolation well - INMOS 09Jul1991

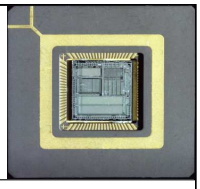
Origin of Pictures

- Original Chip Picture of T414 – from http://www.chilton-computing.org.uk
- Original Chip Picture of T805 – from www.tranputer.net → Pictures (thanks to Michael Brüstle)

Misc

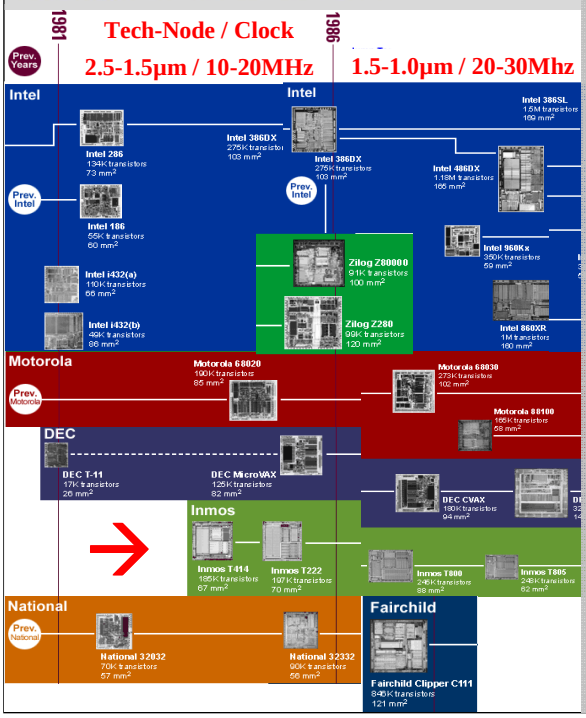
- Ockham's Razor: http://www.seanparnell.com/Hyperion%20Cantos/Web%20Pages/Occam%27s%20Razor.htm
- INMOS History & pictures: http://www.inmos.com/
- home of real men's hardware: http://www.geekdot.com/
- Transputers can be fun: http://www.michaelp.org/transputer
- Ispy & Mtest: http://www.wizzy.com/wizzy/transputer.html

Appendix



Exhibits

Microprocessors: 1981 to 1990



COMPUTER HISTORY MUSEUM

http://www.computerhistory.org/microprocessors/

Dominant Processors

Personal-Computer:

- *16bit: Intel 8086+8087, 286+287, 386SX*
- *32bit: 386DX+387*
- *32bit: Motorola 68020+68881*

Embedded Market:

- *8bit: i8048, Z80, M680x*
- *16bit: 68000*

Minicomputers & Workstations:

- *32bit: Micro-Vax*

Super-Computing:

- *32/64bit: CRAY-1 (1983) Vector Processor*

Exhibits

Microprocessors: 1990 to 1996

Dominant Processors

Personal-Computer:

- *32bit: 386DX+387, 486SX, 486DX*
- *32bit: Motorola 68030+68040*

Embedded Market:

- *8bit: i8049, Z180, M680x*
- *16bit: 68000*
- *32bit: (i960), **T805***

Minicomputers & Workstations:

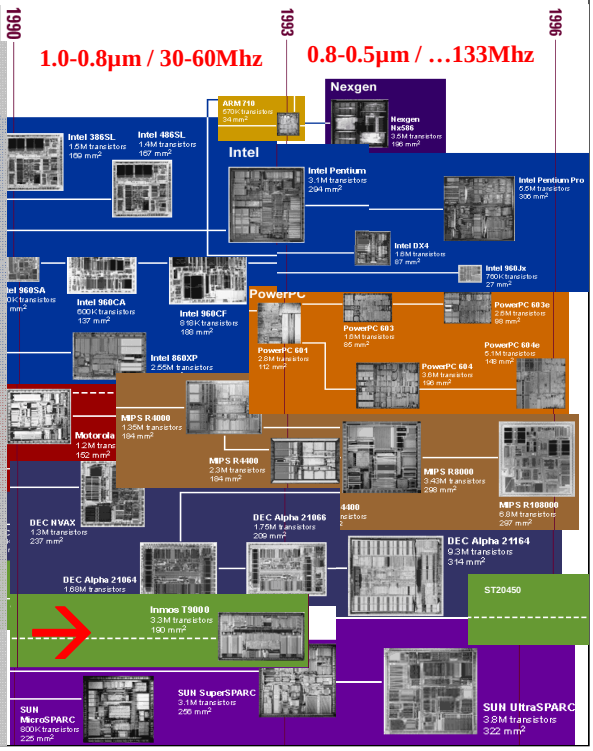
- *32bit: i860, SUN-Sparc, MIPS*
- *64bit: DEC-Alpha*

Super-Computing:

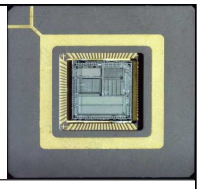
- *32/64bit: CRAY X-MP (1989) Vector Processor*

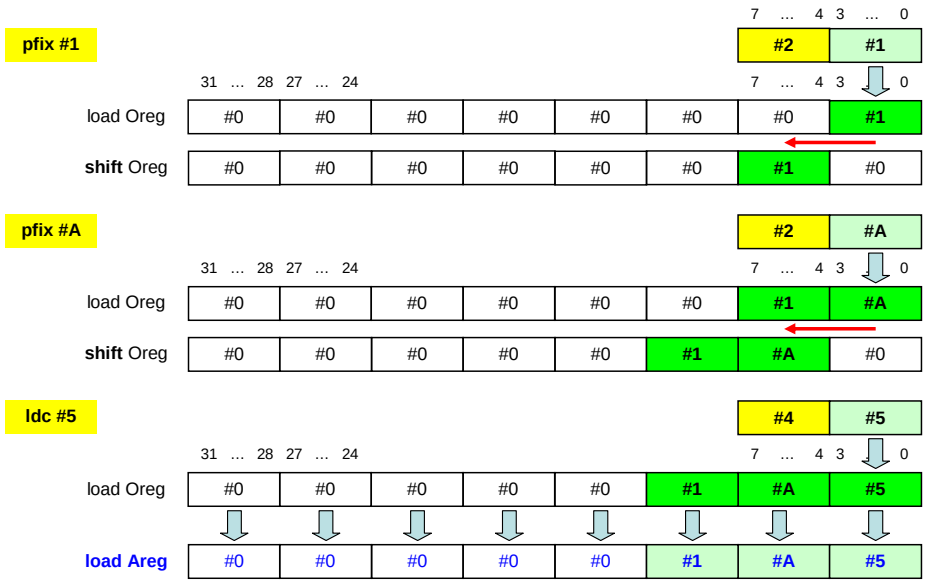
COMPUTER HISTORY MUSEUM

http://www.computerhistory.org/microprocessors/



2.2 Instruction Set
Opr Register during pfix # (1Clk)





How to build a 32bit Constant or Address

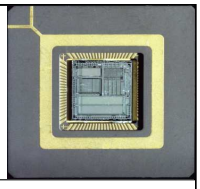
The prefixx instruction loads its four data bits into the O register, and then shifts the O register up four places. The negative prex instruction is similar, except that it complements the operand register before shifting it up.

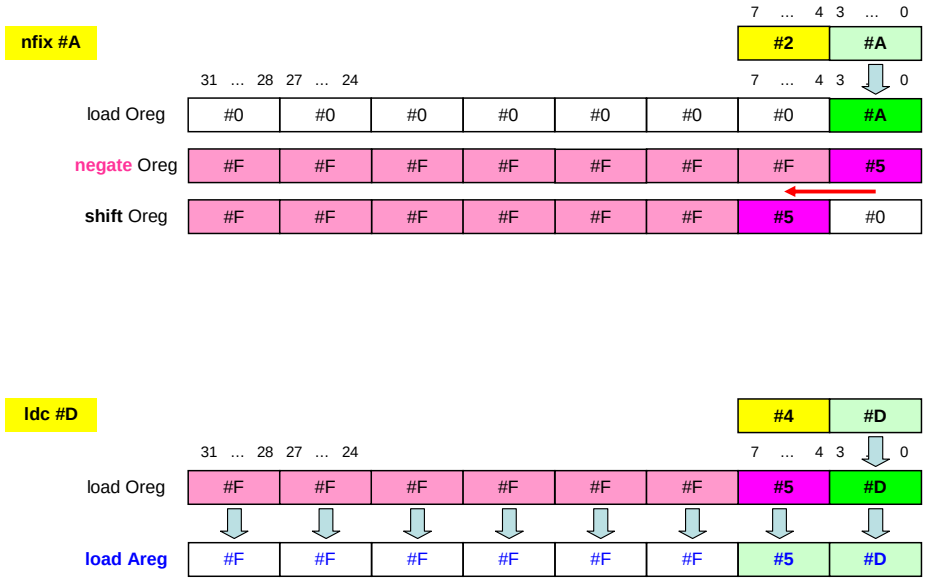
Consequently operands can be extended to any length up to the length of the operand register by a sequence of prex instructions.

The prex functions can be used to extend the operand of an operate instruction just like any other. The instruction representation therefore provides for an indenite number of operations.

The encoding of operations is chosen so that the most common operations, such as add and greater than, are represented without a prex instruction.

2.2 Instruction Set
Opr Register during nfix # (1Clk)





How to build a 32bit Constant or Address

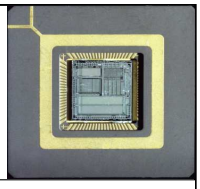
The prefixx instruction loads its four data bits into the O register, and then shifts the O register up four places. The negative prex instruction is similar, except that it complements the operand register before shifting it up.

Consequently operands can be extended to any length up to the length of the operand register by a sequence of prex instructions.

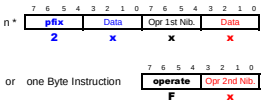
The prex functions can be used to extend the operand of an operate instruction just like any other. The instruction representation therefore provides for an indenite number of operations.

The encoding of operations is chosen so that the most common operations, such as add and greater than, are represented without a prex instruction.

2.2 Instruction Set
1st OpCode Table



one Byte Operation Codes: **31**

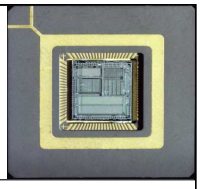


1st Nibble	2nd Nibble 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	**j**	address relative to Iptr														
1	**ldlp**	offset relative to Wptr														
2	**pfix**	operand														
3	**ldnl**	offset relative to Areg														
4	**ldc**	operand														
5	**ldnlp**	offset relative to Areg														
6	**nfix**	operand														
7	**ldl**	offset relative to Wptr														
8	**adc**	operand														
9	**call**	address relative to Iptr														
A	**cj**	address relative to Iptr														
B	**ajw**	operand														
C	**eqc**	operand														
D	**stl**	offset relative to Wptr														
E	**stnl**	offset relative to Areg														
F	**opr**	**rev**	**lb**	**bsub**	**endp**	**diff**	**add**	**gcall**	**in**	**prod**	**gt**	**wsub**	**out**	**sub**	**startp**	**out byte**	**out word**

1st Nibble	Operation	Cycles	FctCode
0	**jump**	**3**	**0**
1	**load local pointer**	**1**	**1**
2	**prefix**	**1**	**2**
3	**load non-local**	**2**	**3**
4	**load constant**	**1**	**4**
5	**load non-local pointer**	**1**	**5**
6	**negative prefix**	**1**	**6**
7	**load local**	**2**	**7**
8	**add constant**	**1**	**8**
9	**call**	**7**	**9**
A	**cond. jump (not taken)** **cond. jump (taken)**	**2** **4**	**A**
B	**adjust workspace**	**1**	**B**
C	**equals constant**	**2**	**C**
D	**store local**	**1**	**D**
E	**store non-local**	**2**	**E**
F	**operate**		**F**

2nd Nibble	Operation	Cycles	OpCode
0	**reverse**	**1**	**00**
1	**load byte**	**5**	**01**
2	**byte subscript**	**1**	**02**
3	**end process**	**13**	**03**
4	**difference**	**1**	**04**
5	**addition**	**1**	**05**
6	**general call**	**4**	**06**
7	**input message**	**2w+19**	**07**
8	**product**	**b+4**	**08**
9	**greater than**	**2**	**09**
A	**word subscript**	**2**	**0A**
B	**output message**	**2w+19**	**0B**
C	**subtraction**	**1**	**0C**
D	**start process**	**12**	**0D**
E	**output byte**	**23**	**0E**
F	**output word**	**23**	**0F**

2.2 Instruction Set
5th OpCode Table … example



two Byte Operation Codes: **16**

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
pfix				4				operate				Opr 2nd Nib.			
2				4				F				x			

1st Nibble	2nd Nibble															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1st N.	F	opr	shr	shl	mint	alt	altwt	altend	and	enbt	enbc	enbs	move	or	csngl	ccnt1	talt	ldiff

1st Nibble	Operation	clk
F	operate	

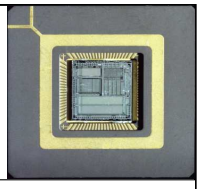
2nd Nibble	Operation	Cycles	OpCode
0	shift right	1	40
1	shift left	1	41
2	minimum integer	1	42
3	alt start	2	43
4	alt wait (channel not ready)	17*	44
5	alt end	4	45
6	and	1	46
7	enalbe timer	8	47
8	enable channel (ready)	7*	48
9	enable skip	3	49
A	move message	2w+8	4A
B	or	1	4B
C	check single	3	4C
D	check counter from 1	3	4D
E	timer alt start	4	4E
F	long diff	3	4F

OpCode Tables:

- one table for one Byte OpCodes
- 11 tables for 2 Byte OpCodes (#1x…#Bx)
- one table for 3 Byte OpCodes (#17x)

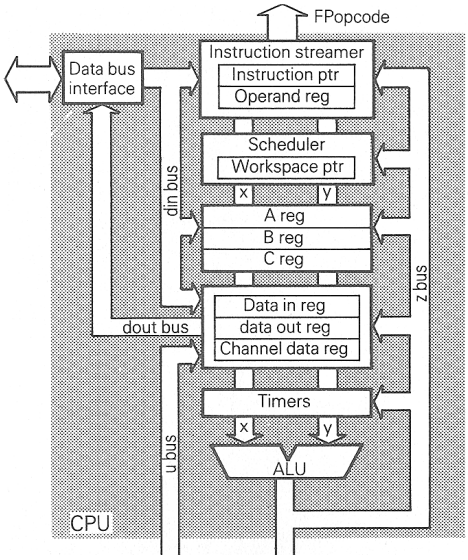
All togeter = 151 direct instructions
+16 indirect instructions (FPU only)

2.1 Hardware Details
CPU: Data-Paths

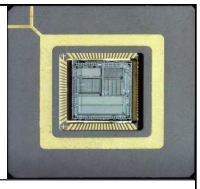


- 4 Phase Clock → 1 Clock Execution
- Data Path controlled by horizontal Microcode (~80 bit wide)
- X- and Y-Bus for operand transfer
- Z-Bus for result transfer and data exchange with Link Channel DMA's or reading the actual Timer value
- U-Bus to control (arbiter) the Z-Bus, i.e. either processor or DMA's can be master!
- RISC instructions, e.g. almost all ALU operations
- CISC instructions, e.g. all Scheduler Operations, 2D Blok Move, …

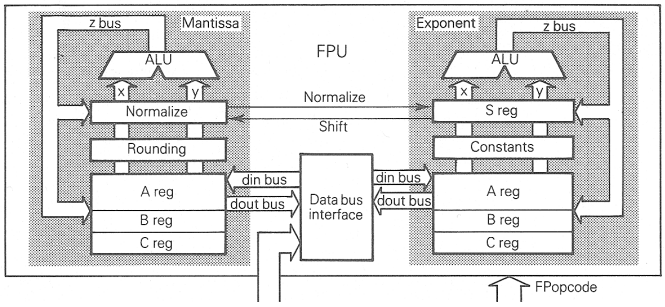
For more HW details … see Patent List in Appendix.



2.1 Hardware Details
FPU



Processor		Whetstones/second single length
Intel 80286/80287	8 MHz	300K
IMS T414-20	20 MHz	663K
NS 32332-32081	15 MHz	728K
MC 68020/68881	16/12 MHz SUN 3	755K
VAX 11/780 FPA	UNIX 4.3 BSD	1083K
IMS T800-20	20 MHz	4000K
IMS T800-30	30 MHz	6000K

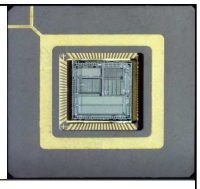


full IEEE-754 compatible single and double precision (64bit) FPU w/ 50 instructions

- All Arithmetic operations have been formally verified and proven
- Full parallel FP operation to integer CPU (e.g. address calculations)
- Note: no big hardware multiplier! But Silicon area vs speed optimized
- For 64bit: **fmul** 27 clocks, **fdiv** 43 clocks → ca. 1.5 MFLOPs @ 20MHz

For more internal details about FPU pls see [72-TCH-047-00]
"The role of occam in the design of the IMS T800", INMOS technical Notes, Sep88.

2.1 Hardware Details
Links: Data-Paths

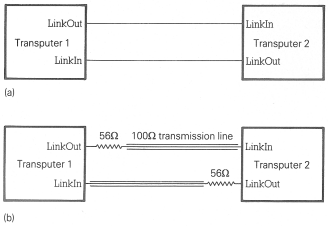


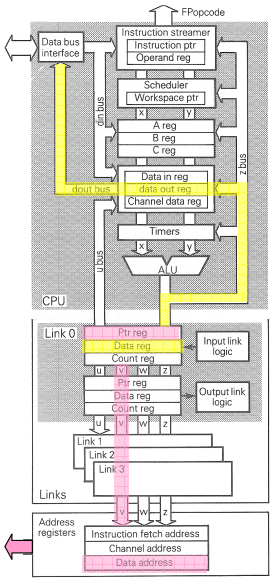
Each Link has separate input logic and output logic, combined with own DMA.
Therewith Link operation can be fully overlapped w/ CPU operation.

- **U-Bus**: Data-Bus and Address-Bus Arbitration (Link DMA vs CPU)
- **V-Bus, W-Bus**: provides Source (input) or Destination (output) Address from PtrReg via DataAddrReg (CPU) to Address-Bus
- **Z-Bus**: connects Link DBufReg via ChannelDataReg (CPU) to Data-Bus

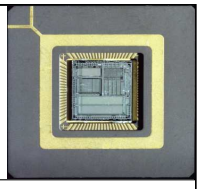
(a) Link Transfer Rate nominal is 20Mbps (1,2MByte/s) for short distance direct Transputer to Transputer connection.
In case of more than 30cm distance Fast-TTL buffering is recommended.

(b) For long distance connection (>20m up 1km) matching RS422 is used.
In case of larger distances the use of fiber optics is recommended.



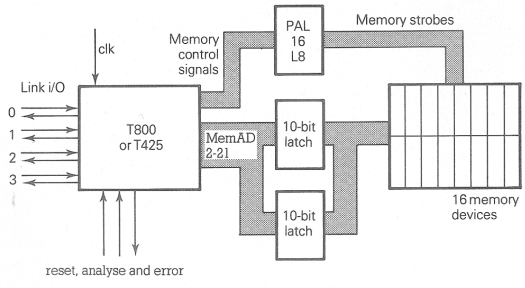


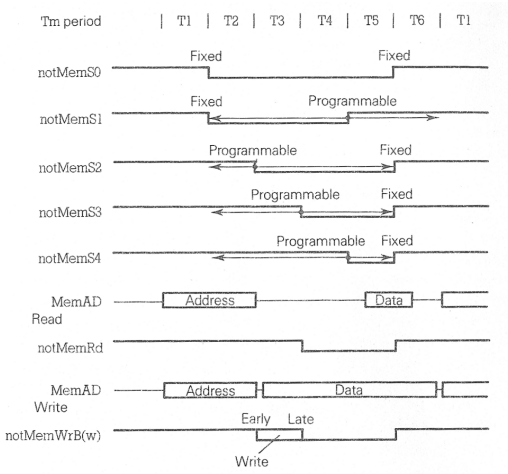
2.4 System Services
onChip RAM + Mem-IF



- full programmable memory timing from 3 to 6 T cycles (each 50ns) for dRAM access times from 50…150ns
- direct RAS / CAS signals
- Refresh control register in CPU

- for small outline TRAM design
- only few additional circuits needed:

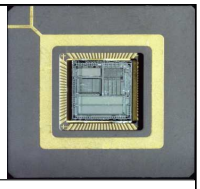




4. Outlook
Transputer Target Applications

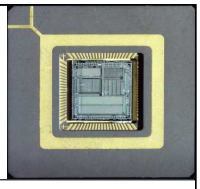
- Scientific and mathematical applications
- High speed multi processor systems
- High performance graphics processing
- Supercomputers
- Workstations and workstation clusters
- Digital signal processing
- Accelerator processors
- Distributed databases
- System simulation
- Telecommunications
- Robotics
- Fault tolerant systems
- Image processing
- Pattern recognition
- Artificial intelligence

4. Outlook
other Programming Languages



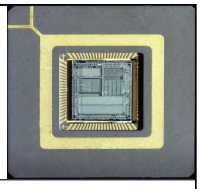
- Ada
- C
- C++
- Fortran
- Forth
- Java

4. Outlook
Transputer OS



- **CHORUS (**UNIX) System V
- **Helios** (UNIX), distributed OS, µKernel based („Nucleus“) → see next Page
- **Idris** (UNIX), POSIX compatible, User-IF running on one CPU only, distributed Communication Kernels for Message Passing
- **Trollius** (UNIX), node based Kernel (same on each CPU), Lib. for Message Passing
- TINIX
- **Virtuoso** (UNIX), µKernel based (Nano-Kernel: Processes & Channels), available for different Hardware Platforms: T8/T9, TMS320C30, MIPS, 68030, … x86

4. Outlook
OS: Helios



ParHelion GmbH:

- **Helios** (UNIX), distributed OS, µKernel based („Nucleus“), Client-Server Model, Message Passing, all resources are named Objects, e.g. Task Moving possible (secure autentication),

  Nucleus consists of 4 components:

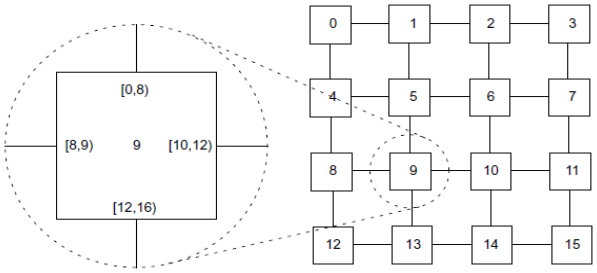
  – Kernel (Message Passing, Memory Mgmt),
  – System Lib (Sys Calls),
  – Loader (Code & Data Mgmt),
  – Processor Mngr (Task & I/O Mgmt)
- Memory requirements for µKernel ~ 1MB RAM, 4MB TRAM recommended.

4. Outlook
Transputer Networks

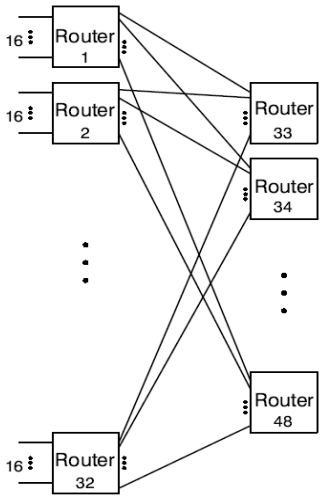
IMSC004

(1988)

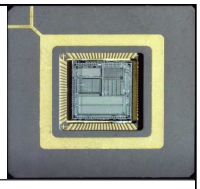
CrossBar LinkSwitch
for 32x32 Channels



Transputer Grid **…** **Router Network**

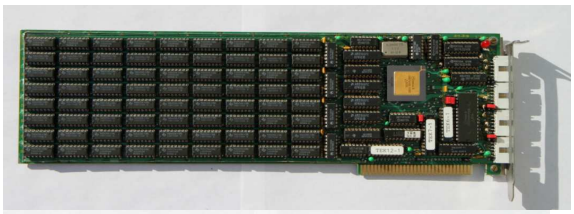


4.Outlook
Standard Boards

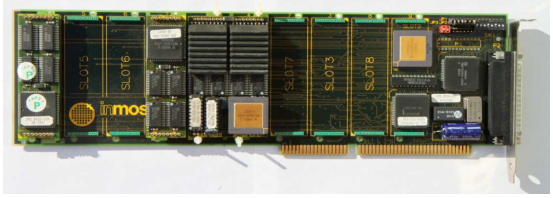


IMB-PC Development & Accellerator Boards (ISA):

- B004 (1985) T414-15, 2MB RAM



- B008 (1987): up to 10x TRAM,



VME Development Boards:

- B011 VME Master (1st Gen.)
- B016 VME Master (2nd Gen.)
- B014: up to 8x TRAM slave board