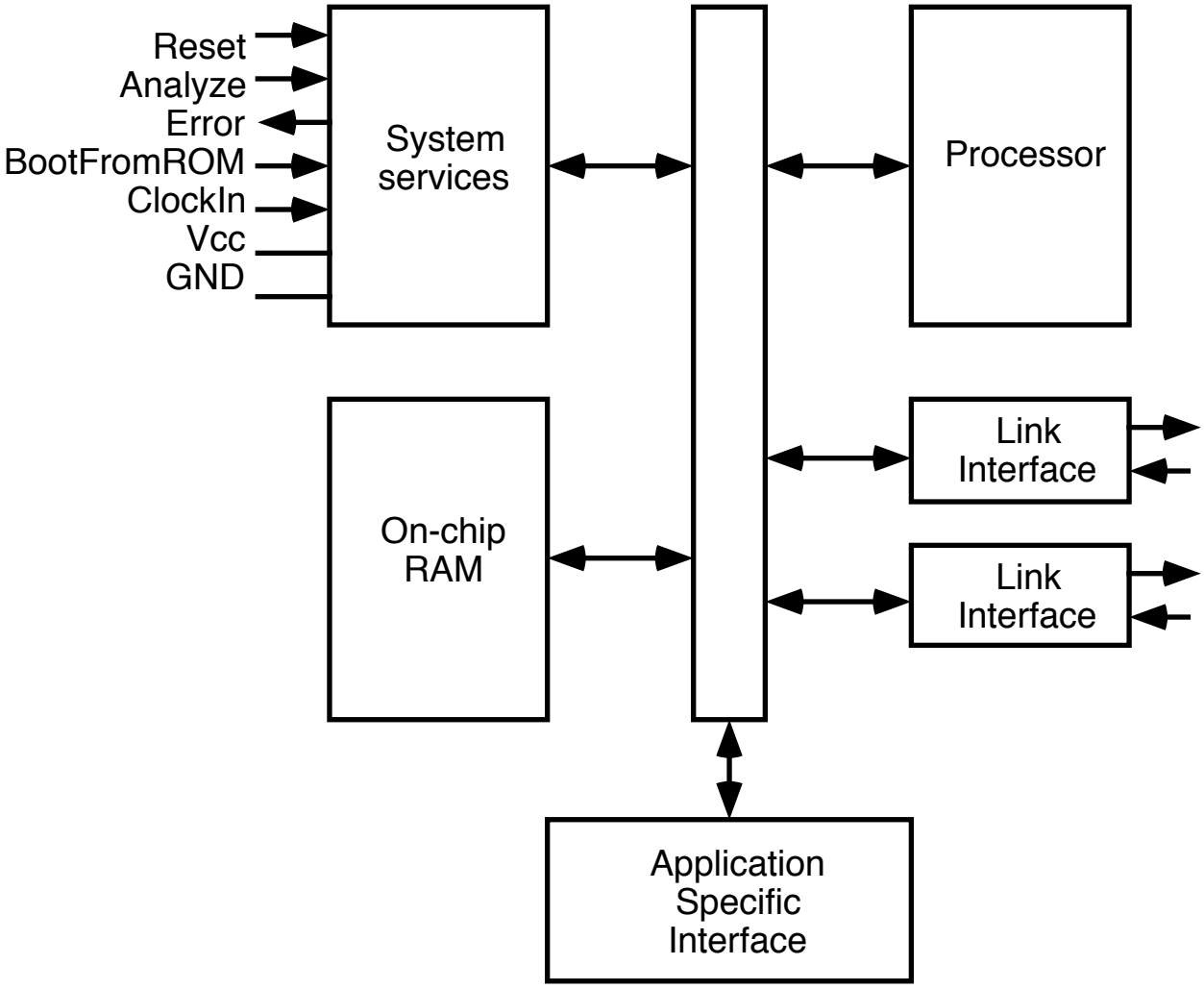
Topics in computer architecture

Transputer architecture

P.J. Drongowski
SandSoftwareSound.net

Transputer architecture

- Processor (integer CPU, optional floating point unit)
- On-chip RAM
- Link interfaces
- System services
- Application specific interface
- occam programming language



Transputer family

Feature	T212	T222	T414	T425	T800	T801
Bus width	16	16	32	32	32	32
Cycle time (ns)	50	50	50	33	33	33
Peak MIPS	10	20	20	30	30	30
Peak MFLOPS					4.3	4.3
RAM (Kbytes)	2	4	2	4	4	4
Serial links	4	4	4	4	4	4
Rate (Mbyte/s)	1.6	2.4	1.6	1.6	2.4	2.4
Interrupt (ns)	950	950	950	630	630	630
Pins	68	68	84	84	84	100

- Common features
  - CMOS circuit technology
  - ISA (occam programming language)
- Support devices
  - M212 Disk processor
    - 16-bit CPU, 2Kb RAM, 4Kb ROM, 2 serial links
    - 5 MHz clock, 68-pin package
    - ST506/ST412, SA400/450 compatible interface
    - Hardware ECC / CRC
    - Commands: PollDrives, Seek, FormatTrack, etc.
  - C011 and C012 Link adapters
    - Peripheral and bus interface (C011)
    - Bus interface (C012)
    - One serial link
    - Programmable parallel interface
  - C004 Programmable link switch
    - 32 way crossbar switch
    - Configuration (serial) link
    - 32 in/out serial links (20 Mbits/sec maximum)

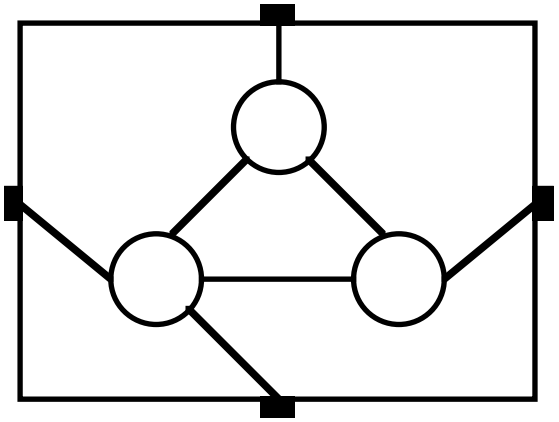
Goals

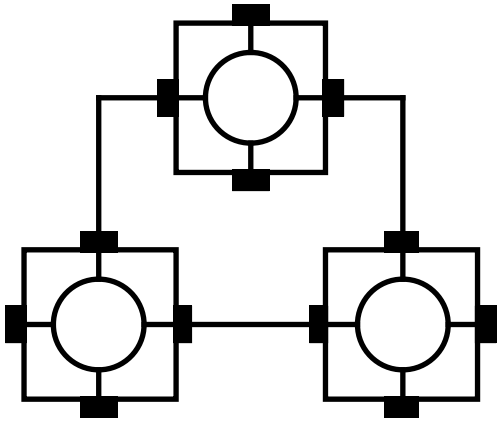
- High performance through concurrent network of systems
  - Organize many Pc-M elements into network
  - Network topology can be arbitrary (e.g., linear, plex)
  - Element-to-element communication is direct
  - Concurrent programs must execute correctly
  - Direct support for HL parallel programming language
  - Compatible with VLSI technology

- VLSI technology
  - Mfg large numbers of identical devices inexpensively
  - Customize configuration for concurrent application
  - Off-chip communication is slow
    - Put processor and memory on same chip
    - Allow for co-resident processes
  - Serial communication
    - Off-chip communication is large percentage of area
    - Package size
      - ◊ Determined more by pin count
      - ◊ Functionality is less important
    - PCB connections need large amount of board area
  - Direct point to point communication
    - Higher speed (no intermediaries)
    - Shared bus needs arbitration and special drivers
  - Microprogrammed, sequential implementation
    - 32 instructions for simple sequential programs
    - Long arithmetic
    - Process scheduling
  - Architecture was formally specified
    - This state - next state semantics
    - Is independent of data word width!
    - Allows microcode reusability across family

Features

- System services
  - Clock distribution
  - Start-up (initialize processor, links AS-interface)
  - Bootstrapping
    - Via link interface
      - ◊ Bootstrap from first link to receive a message
      - ◊ First message byte is size of program to follow
      - ◊ Other links will be ignored until explicitly read
    - Via ROM
      - ◊ Transfer control to top two bytes in memory
      - ◊ Contains a backward jump into ROM
  - Errors
    - Asserts error pin; sets error flag
    - transputer may stop on error
    - Example conditions
      - ◊ Arithmetic overflow
      - ◊ Divide by zero
      - ◊ Array bounds violation
- occam programming
  - Building block is the *process*
  - System is designed as interconnected processes
  - Processes communicate (point-to-point) via *channels*
  - Process internals are hidden
  - Visible behavior is specified by msg's sent and received
  - Communication between processes is synchronized
  - Process can be composed of other processes
  - Processes must be assigned to physical processors





Communications

- Advantages of point-to-point communications
  - No contention for communication mechanism
  - No capacitive load penalty as transputers are added
  - Bandwidth does not saturate as system size increases

- Memory is local to each transputer
  - Total bandwidth is proportional to number of transputers
  - Bandwidth is shared in large global memory machines
  - Memory and communication interfaces are separate

- Link communication
  - A *link* has two one-direction signal lines
  - All data is transmitted serially
  - Two wires carry two occam channels (bi-directional)
  - Each line carries data and control information
  - Link protocol (synchronized occam communication)
    - Each message is a sequence of single bytes
    - Only a single byte buffer is required
    - A byte is transmitted serially with
      - ◊ A start bit followed by a one bit
      - ◊ Eight data bits
      - ◊ One stop bit
    - Acknowledge is a start bit followed by a zero bit
      - ◊ Indicates byte was received
      - ◊ Indicates readiness for next byte
      - ◊ Sender is rescheduled after ack of last msg byte
    - Data bytes and ack's are multiplexed on signal line
    - Continuous transmission can be sustained
  - Serial communication makes PCB layout easier
  - All transputers support standard 10 Mbit/sec link freq
  - Link communication is not sensitive to clock phase

Sequential processing

- Six integer CPU registers
  - Workspace pointer to local variable area (Wptr)
  - Instruction pointer (Iptr)
  - Operand register to form instruction operands (Oreg)
  - A, B and C registers which form and evaluation stack
    - Balance code size and implementation complexity
    - Used for expression evaluation
    - Instructions are zero address and very short
    - Hardware has *no* overflow detection

- Instruction set design
  - Simple and efficient compilation of HLL programs
  - Relatively small number of instructions
  - All instructions have the same format
  - Compact representation of frequently used operations
  - Instruction set independent of processor word length
  - Use same microcode across family
  - Memory is word accessed; fetch several instr's at once

- Instruction format
  - Single byte divided into two 4-bit parts
  - Four most significant bits are a function code
  - Four least significant bits are a data (operand) value

- Rationale
  - Frequent operations
    - Loading small literal values
    - Load from or store to small number of variables
  - Access to variables
    - Access 16 locals relative to workspace pointer (Wptr)
    - Non-locals, others accessed relative to A registers

Direct functions

- Sixteen most frequently occurring instructions
- All direct functions are one byte

`load constant`	**`ldc`**
`add constant`	**`adc`**
`load local`	**`ldl`**
`load local pointer`	**`ldlp`**
`store local`	**`stl`**
`load non-local`	**`ldnl`**
`loal non-local pointer`	**`ldnlp`**
`store non-local`	**`stnl`**
`prefix`	**`pfix`**
`negative prefix`	**`nfix`**
`jump`	**`j`**
`conditional jump`	**`cj`**
`call`	**`call`**
`equals constant`	**`eqc`**
`adjust workspace`	**`ajw`**
`operate`	**`opr`**

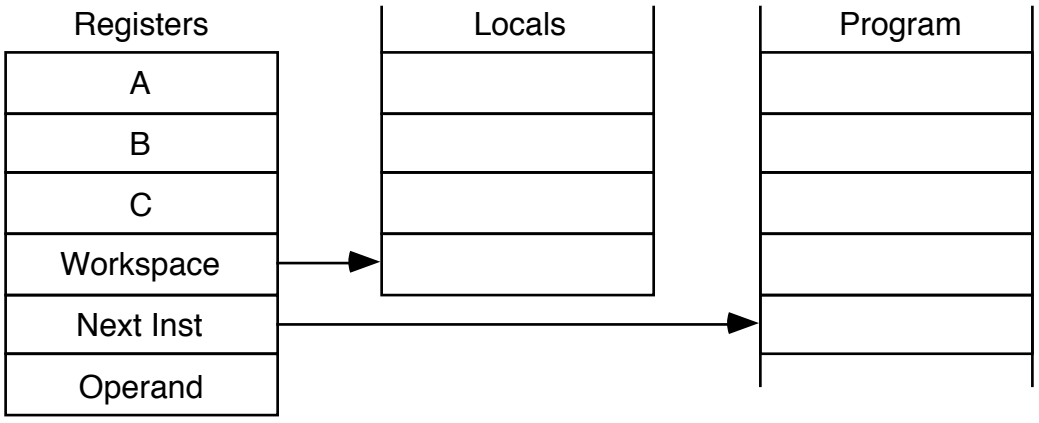
- Prefix functions
  - All instructions load 4-data bits into 4-LSB's of Oreg
  - Oreg is then used as the instructions operand
  - All instructions except prefix end by clearing Oreg
  - `pfix` loads its 4-data bits, then shifts Oreg 4 places
  - `nfix` is similar, but first complements Oreg
  - Operands can be represented up to length of Oreg
  - Range -256 to 255 represented with one `pfix`
  - Advantages
    - Decoded and executed like all other instructions
    - Uniformly allow instruction operands of any size
    - Operands represented independent of word length

Indirect functions

- `operate` instruction
- Causes operand to be intepreted as operation
- Sixteen such instructions encoded in one byte

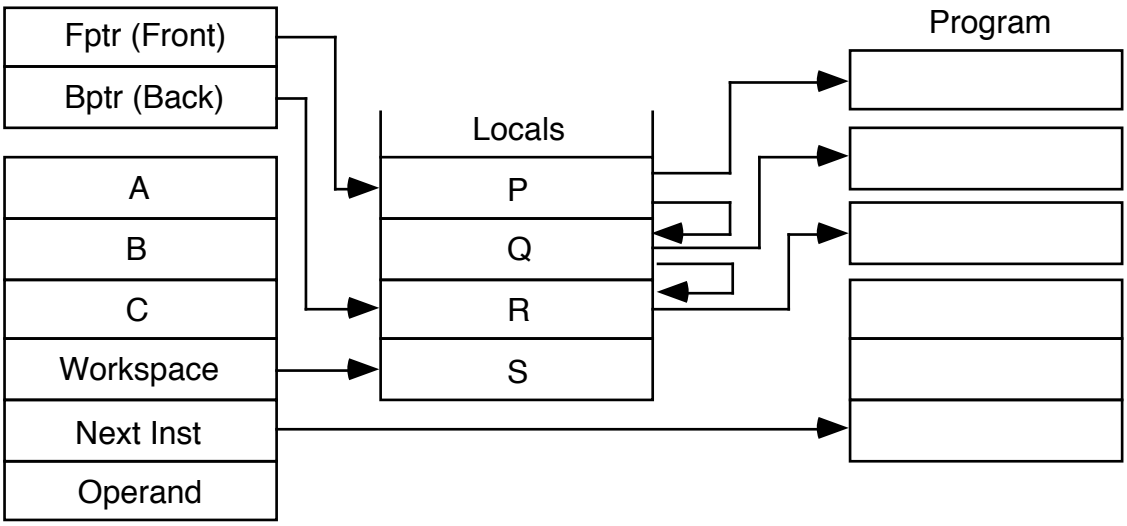
`add`	**`add`**
`subtract`	**`sub`**
`greater than`	**`gt`**
`difference`	**`diff`**
`product`	**`prod`**
`reverse (swap)`	**`rev`**
`byte subscript`	**`bsub`**
`word subscript`	**`wsub`**
`load byte`	**`lb`**
`input message`	**`in`**
`output message`	**`out`**
`output word`	**`outword`**
`output byte`	**`outbyte`**
`general call`	**`gcall`**
`start process`	**`startp`**
`end process`	**`endp`**

- Prefix instructions can extended operand
- Allows variable length instructions



Processes

- Transputer provides a microcoded scheduler
- Compiler allocates space to concurrent processes
- Process has two major states
  - Active
    - Executing
    - Waiting (on a ready-to-execute list) to execute
  - Inactive
    - Ready to `input`
    - Ready to `output`
    - Waiting until a specified time
- Ready-to-execute list
  - Linked list of process workspaces
  - One list each for high and low priority processes
  - Two hardware registers
    - `Fptr` points to first process (Front)
    - `Bptr` points to last process (Back)
- Context switching
  - Process runs until input/output wait or timer wait
  - Its Iptr is saved in its workspace
  - The next process is taken from ready-to-execute list
- Remarks
  - Not necessary to save eval stack on rescheduling
  - Descheduling points are carefully defined in hardware
  - Expression evaluation proceeds to completion
  - Low priority processes are timesliced (~ 10,000 cycles)
  - Switch time is short (1μs) since little state needs



Process priority

- Two priorities provided: high and low priority
- Low priority process
  - Priority level 1
  - Executed when no high priority processes are active
  - Periodically time-sliced to distribute processor time
- Time-slicing
  - One or two "periods" in duration
  - Period lasts 5120 cycles
  - Period is 1 ms at clock frequency of 5 MHz
  - High priority process may steal cycles from time-slice
- High priority process
  - Priority level 0
  - Expected to run for a short time
  - Process runs until:
    - ◊ Communication wait
    - ◊ Timer wait
    - ◊ Processing completes
- Latency
  - Low priority process
    - ◊ Assume N low priority processes
    - ◊ Latency: Time from becoming active to execution
    - ◊ $2 \times N - 2$ time-slice periods
  - High priority process
    - ◊ Waiting on external channel
    - ◊ Assume no high priority other process is active
    - ◊ Interrupt latency is 19 cycles (ave), 78 cycles (max)

occam synchronization

- Sequential construct
  - Processes are executed one after another
  - Straight-line, sequential code

```
SEQ                SEQ
  P1                 c1 ? x
  P2                 x := x + 1
  P3                 c2 ! x
  ...
```

- Parallel construct
  - Component processes are executed together
  - Construct terminates after all components terminate
  - Parallel processes communicate via channels

```
PAR                PAR
  P1                 c1 ? x
  P2                 c2 ! y
  P3
  ...
```

- Alternative construct
  - Wait on disjunction of input events
  - Input process acts as a "guard"

```
ALT                ALT
  input1             count ? signal
    P1                 counter := counter + 1
  input2             total ? signal
    P2                 SEQ
  input3                 out ! counter
    P3                   counter := 0
    ...
```

Channel communication

- Communication
  - Achieved by means of *channels*
  - occam is point-to-point, synchronized, unbuffered
  - Channel does not need process queue, buffer, etc.
  - Two instructions: `input message`, `output message`
  - Use address of channel to determine internal/external
  - Same instruction sequence for hard/soft channels
  - Communicate when both processes are ready
  - First ready to communicate process must wait
  - Communication sequence:
    - ◊ Push pointer to message, channel address, count
    - ◊ Execute input or output instruction
- Internal communication
  - Processes are co-resident on transputer
  - Channel is implemented by one memory word
  - Word holds either identity of process or *empty*
  - Sequence:
    - ◊ Identity of first ready process is put in channel word
    - ◊ Process is physically descheduled
    - ◊ Run processes from active queue
    - ◊ When second process is ready to communicate
      - ◊ Message is copied
      - ◊ First waiting process is added to active list
      - ◊ Channel is reset to initial state
- External communication
  - Prcoesses communicate via link interface
  - Sequence:
    - ◊ Processor delegates link interface
    - ◊ Deschedule the communicating process
    - ◊ Process becomes active after acknowledgement
  - Allows concurrent computation and communication
  - Each link interface has three registers:
    - ◊ A prointer to a process workspace
    - ◊ A pointer to a message
    - ◊ A count of bytes in the message