INMOS Transputer and Occam Courses

Parallel C

Occam is a trade mark of INMOS Limited

SGS-THOMSON MICROELECTRONICS

INMOS is a member of the SGS-THOMSON Microelectronics group.
Parallel C on the Transputer
INMOS Training Course

This course refers to the INMOS IMS D711d software.

, inmos, IMS and occam are trade marks of INMOS Limited.

Copyright 1990 INMOS Limited.

Course plan

Monday
- Transputers and boards
- Multi-processor programming
- Compiling C

Tuesday
- Channels
- Multi-task programs
- Multi-transputer systems
- Priority and time

Wednesday
- Threads
- Flood configuring
- Other tools
- C implementation
Programming languages

Languages supported by INMOS on the transputer:

- C
- FORTRAN
- Occam
- Pascal

Host machines supported by INMOS:

- IBM PC and compatibles
- NEC PC
- VAX
- Sun 3 and 4

Transputers and boards

- Transputer architecture
- Parallel processing
- The transputer family
- INMOS boards
- C environment
Transputer architecture

- System services
- On-chip RAM
- Application specific interface

- Processor
- Link interface

Diagram showing the architecture with interconnected components.
Parallel processing

Connecting transputers

Clock frequencies within 200 ppm.
Clock phases may differ.
Transputer networks

Simple link connections for point-to-point communication.
Spare links can connect to peripherals via link adaptors.

Local resources

Signal Processing
Memory

Memory
Disk

Memory
Graphics

Memory

Input/Output

SGS-THOMSON MICROELECTRONICS
Connectivity

One link connects two transputers.

Two links allow pipelines.

Three links for tree structures.

Four links for square arrays.

The transputer family
32 bit transputers

Fixed point transputers
- IMS T400
- IMS T425
- Memory
- 2kbytes on-chip
- 4 fast links
- 4 kbytes on-chip

Floating point transputers
- IMS T800
- IMS T801
- IMS T805
- Memory
- Programmable emi
- 2 cycle emi
- Programmable emi
- Debug support

$20 each in quantity

SGS·THOMSON MICROELECTRONICS

16 bit transputers

Memory
- IMS T222
- IMS T225
- Memory
- 4kbytes on-chip
- 4 kbytes on-chip
- Faster links
- Debug support

Disk controlling transputer
- IMS M212
- Disk

SGS·THOMSON MICROELECTRONICS
### Transputer differences

<table>
<thead>
<tr>
<th></th>
<th>T212</th>
<th>T222</th>
<th>T225</th>
<th>T414</th>
<th>T400</th>
<th>T425</th>
<th>T800</th>
<th>T801</th>
<th>T805</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-chip memory (bytes)</td>
<td>2k</td>
<td>4k</td>
<td>4k</td>
<td>2k</td>
<td>2k</td>
<td>4k</td>
<td>4k</td>
<td>4k</td>
<td>4k</td>
</tr>
<tr>
<td>Floating point hardware</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Word length (bits)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Number of links</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Overlapped acknowledge</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Double-buffered link output</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Programmable DRAM controller</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>2 cycle external read/write</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Number of pins</td>
<td>68</td>
<td>68</td>
<td>68</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>100</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td>Errorln pin</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RefreshPending pin</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>EventWaiting pin</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ProcSpeedSelect pins</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DisableInRAM pin</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MemBAcc pin</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Separate address and data</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multiplexed address and data</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Transputer instruction set differences

<table>
<thead>
<tr>
<th></th>
<th>T212</th>
<th>T222</th>
<th>T225</th>
<th>T414</th>
<th>T400</th>
<th>T425</th>
<th>T800</th>
<th>T801</th>
<th>T805</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point unit (53)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fptesterror</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Floating point support (5)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>fmul</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2D block moves (4)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CRC operations(2)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bit operations (3)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Break point debugging (9)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fast negative prod</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>dup</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>pop</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>wsubdb</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>lddevid</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ldmemstartval</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Link chips

For software control of topology

For connection to buses and peripherals

INMOS boards
## INMOS board range

<table>
<thead>
<tr>
<th>Compute TRAMs</th>
<th>IMS B410</th>
<th>T801 160k sram</th>
<th>size 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMS B405</td>
<td>T800 8M dram</td>
<td>size 8</td>
<td></td>
</tr>
<tr>
<td>IMS B417</td>
<td>T800 64k sram, 4M dram</td>
<td>size 4</td>
<td></td>
</tr>
<tr>
<td>IMS B403</td>
<td>T414/T800 1M sram</td>
<td>size 4</td>
<td></td>
</tr>
<tr>
<td>IMS B404</td>
<td>T425/T800 32k sram, 2M dram</td>
<td>size 2</td>
<td></td>
</tr>
<tr>
<td>IMS B411</td>
<td>T425/T800 1M dram</td>
<td>size 1</td>
<td></td>
</tr>
<tr>
<td>IMS B401</td>
<td>T4xx/T80x 32k sram</td>
<td>size 1</td>
<td></td>
</tr>
<tr>
<td>IMS B416</td>
<td>T222 64k sram</td>
<td>size 1</td>
<td></td>
</tr>
<tr>
<td>IMS B402</td>
<td>T222 8k sram</td>
<td>size 1</td>
<td></td>
</tr>
<tr>
<td>Graphics TRAMs</td>
<td>IMS B419</td>
<td>Graphics G300</td>
<td>size 6</td>
</tr>
<tr>
<td>IMS B408</td>
<td>Graphics drawing and storage</td>
<td>size 8</td>
<td></td>
</tr>
<tr>
<td>IMS B409</td>
<td>Graphics display driver</td>
<td>size 8</td>
<td></td>
</tr>
<tr>
<td>i/o TRAMs</td>
<td>IMS B407</td>
<td>Ethernet</td>
<td>size 8</td>
</tr>
<tr>
<td>IMS B415</td>
<td>Link interface</td>
<td>size 1</td>
<td></td>
</tr>
<tr>
<td>IMS B422</td>
<td>SCSI interface</td>
<td>size 2</td>
<td></td>
</tr>
<tr>
<td>IMS B421</td>
<td>GPIB interface</td>
<td>size 4</td>
<td></td>
</tr>
<tr>
<td>Other TRAMs</td>
<td>IMS B418</td>
<td>ROM</td>
<td>size 2</td>
</tr>
<tr>
<td>IMS B420</td>
<td>Vector processing</td>
<td>size 4</td>
<td></td>
</tr>
<tr>
<td>Motherboards</td>
<td>IMS B008</td>
<td>IBM PC XT/AT</td>
<td>10 slots</td>
</tr>
<tr>
<td>IMS B015</td>
<td>NEC PC</td>
<td>4 slots</td>
<td></td>
</tr>
<tr>
<td>IMS B012</td>
<td>Double Eurocard</td>
<td>16 slots</td>
<td></td>
</tr>
<tr>
<td>IMS B014</td>
<td>VME slave</td>
<td>8 slots</td>
<td></td>
</tr>
</tbody>
</table>

## Compute only TRAMs

<table>
<thead>
<tr>
<th>IMS B402</th>
<th>16 bit 8k sram</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMS B416</td>
<td>16 bit 64k sram</td>
</tr>
<tr>
<td>IMS B401</td>
<td>32k sram</td>
</tr>
<tr>
<td>IMS B411</td>
<td>1M dram</td>
</tr>
<tr>
<td>IMS B404</td>
<td>32k sram 2M dram</td>
</tr>
<tr>
<td>IMS B410</td>
<td>T801 160k sram</td>
</tr>
<tr>
<td>IMS B403</td>
<td>1M dram</td>
</tr>
<tr>
<td>IMS B417</td>
<td>64k sram + 4M dram</td>
</tr>
<tr>
<td>IMS B405</td>
<td>8M dram</td>
</tr>
</tbody>
</table>
Special application TRAMs

IMS B408
Graphics drawing and storage

IMS B409
Graphics display driver

IMS B419
G300 Graphics

IMS B407
Ethernet

IMS B422
SCSI i/f

IMS B421
GPIB i/f

IMS B415
link i/f

IMS B420
Vector processor

IMS B418
ROM

Motherboards

IMS B012
Double ext eurocard
IMS T212
and 2xIMS C004
16 module slots

IMS B008
PC add-in board
IMS T212
and IMS C004
10 module slots
IMS B004 PC add-in board

2 Mbytes dynamic RAM with parity

IMST414

Link adaptor

IBM XT/AT bus connector

Edge connector

Toolset environment
The Toolset is designed for use with your favourite editor, 'make' utility and other software tools.

Training course editors:

<table>
<thead>
<tr>
<th>Editor</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folding MicroEmacs</td>
<td>fold &lt;filename&gt;</td>
</tr>
<tr>
<td>MicroEmacs</td>
<td>emacs &lt;filename&gt;</td>
</tr>
<tr>
<td>PCwrite</td>
<td>ed &lt;filename&gt;</td>
</tr>
<tr>
<td>Sidekick</td>
<td>Ctrl + Alt</td>
</tr>
<tr>
<td>Edlin</td>
<td>edlin &lt;filename&gt;</td>
</tr>
</tbody>
</table>
Folding MicroEmacs macros

Cursor movement:-
- Ctrl← Word left
- Ctrl→ Word right
- Home Top of file
- End Bottom of file
- Page Up Page Up
- Page Dn Page Down

Editing:-
- Del Delete ch forwd
- Alt F7 Delete line
- F3 Cut line
- F4 Copy line
- Sh F4 Paste line
- Sh F3 Clear paste buffer
- F9 Search and Replace

Folds:-
- F5 Enter
- F6 Exit
- Sh F5 Open
- Sh F6 Close
- F7 Mark
- F8 Create fold
- Sh F7 Remove fold

Files:-
- Alt F5 New file
- Alt F6 Toggle buffers
- Alt F2 Save
- F2 DOS shell
- Sh F2 Finish
- Sh F9 Run checker
- Sh F10 Locate error

PCwrite

Cursor movement:-
- Home Start of line
- End End of line
- Ctrl← Word left
- Ctrl→ Word right
- Alt + Top of file
- Alt - Bottom of file
- PgUp Scroll up
- Pg Dn Scroll down

Editing:-
- Del Delete ch forwd
- Ctrl bksp Delete word left
- Ctrl Del Delete word right
- Scroll lock Toggle insert/overwrite
- F3 Mark / copy
- F4 Mark / delete
- F6 Mark / move
- F5 Unmark
- F8 Change case

Files:-
- F1 F2 Save and exit
- F1 F3 Save
- F1 F9 F2 Exit without save

Help:-
- Esc F keys
- Shift F1 Display F keys
- F1 F1 General
Editor practical

Create a directory with DOS command `md`.

Enter directory with DOS command `cd`.

Either install your own editor

or familiarise yourself with one of the three editors provided.
Multi-processor programming

The transputer model
Channels
Parallel programming
Table building exercise

The transputer model
Communicating sequential processes

Parallel tasks run independently.
One way point to point channels.

Development system

Host
Multi-transputer system

Independent processors connected by two-way links.
Concurrent processing on each processor.

Mixed languages

C -> C -> FORTRAN
occam -> Pascal
Channels

Point to point
One way
Unbuffered
Synchronised
Control of channels

A channel allows two processes to communicate.

The C programmer has complete control of input and output.

When two tasks are connected by a channel one must output, the other must input.

Both tasks must be ready.
The first task ready always waits.

Synchronisation on channels

[Diagram showing the process of synchronisation on channels, with Task 0 waiting, Input request, Data copied, Task 1 starts, Output request]
Links as channels

Task 0 starts
Input request to link
Task 0 waits
Data copied by link
Link restarts task 0

Task 1 starts
Output request to link
Link restarts task 1
Parallel programming

Sequential program

```c
wash_dishes ()
{
    for (;;) {
        clean_dish ();
        rinse_dish ();
        dry_dish ();
        store_dish ();
    }
}
```
Automatic dishwasher

Cooperating processes

Resources are local to each processor.
Parallel tasks in C

Each task is a complete C program.

Communicating in C

Parallel tasks communicate via channels.
Channels should be given names.

in  soapy_dishes  wet_dishes  dry_dishes  out

clean  rinse  dry  store
The rinse process

```c
main (...) 
... parameters 
{
  int dish;
  for (;;) 
  { 
    input(soapy_dishes,dish);
    rinse_dish (dish);
    output(soapy_dishes,dish);
  } 
}
```

The dry process

```c
main (...) 
... parameters 
{
  int dish;
  for (;;) 
  { 
    input(wet_dishes,dish);
    rinse_dish (dish);
    output(dry_dishes,dish);
  } 
}```
Table building exercise

```c
build_table ()
{
    for (;;) {
        struct wood table_top, set_of_legs, table;
        set_of_legs = make_set_of_legs ();
        table_top = make_table_top ();
        table = assemble_table(set_of_legs, table_top);
    }
}
```

How do you code a production line with three people?
Table building - possible solution page 1

```c
main (...)
... parameters
{
    for (;;) {
        struct wood set_of_legs;
        set_of_legs = make_set_of_legs ();
        output (legs, set_of_legs);
    }
}

main (...)
... parameters
{
    for (;;) {
        struct wood table_top;
        table_top = make_table_top ();
        output (tops, table_top);
    }
}
```

Table building - possible solution page 2

```c
main (...)
... parameters
{
    for (;;) {
        struct wood table_top, set_of_legs, table;
        input(legs, set_of_legs);
        input(tops, table_top);
        table = assemble_table (set_of_legs, table_top);
        output (out, table);
    }
}
```
Compiling C

Overview
Compilers
Linker
Bootstrap tool
Make
Server
Hello World practical

Overview
Tools

- t4c: C compiler for IMS T414
- t8c: C compiler for IMS T800
- ilink: Linker
- iboot: Bootstrap tool
- config: Configurer
- fconfig: Flood configurer

Compilation steps

1. C source -> C compiler (t4c, t8c) -> bin
2. library build -> lib
3. compiled library -> ilibr
4. configuration source
5. linker (ilink) -> c?x -> b4
6. bootstrap tool -> iboot
7. config, fconfig
Compilers

C compilers

C source → C compiler → Compiled C

C source

C compiler

Compiled C

t4c

t8c

bin

t8c myc
Compiler switches

<table>
<thead>
<tr>
<th>Switch</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Check only</td>
</tr>
<tr>
<td>FB filename</td>
<td>Output file - default <em>bin</em></td>
</tr>
<tr>
<td>FL filename</td>
<td>Put listing in file</td>
</tr>
<tr>
<td>S</td>
<td>Single length floating point arithmetic</td>
</tr>
<tr>
<td>T4 / T8 / T8A</td>
<td>Target transputer type</td>
</tr>
<tr>
<td>V</td>
<td>Display progress messages</td>
</tr>
</tbody>
</table>

Indirection

Using PC-DOS hosts, to redirect screen output to a file:

```
t4c myprog > error.dat
```
Linker

Linking

- Compiler
- Source code "myprog.c"
- Library "mylib.lib"
- Pre-linked code
- Linker indirect command file
- Linker: ilink
- Linked code
- Module map
- Symbol table

Example command:
```
ilink mainent.c8x myprog.c8x crt1.lib /o myprog.c8x
```

Output file option is generally necessary.
Mainent

Mainent acts as interface between a single task program and the iserver.
On PC system, mainent.c4x and mainent.c8x are in directory \tc2v0\libs.
Must be linked first.

Run time libraries

The run time libraries contain all the standard libraries.
Two libraries:

crtl.lib for i/o tasks
sacrtl.lib for standalone tasks

On PCs, both libraries in directory \tc2v0\libs.
Indirect linker file

Lists the files to be linked together and options.

Extension l?x.

File: myprog.l8x

| mainent.c8x |
| myprog.t8x |
| crtl.lib |
| /o myprog.c8x |

ilink /f myprog.l8x

is equivalent to:

ilink mainent.c8x myprog.t8x crtl.lib /o myprog.c8x

Linker options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F filename</td>
<td>Indirect linker file</td>
</tr>
<tr>
<td>O filename</td>
<td>Output file - default *.cxx</td>
</tr>
<tr>
<td>I</td>
<td>Information</td>
</tr>
<tr>
<td>U</td>
<td>Unresolved references allowed</td>
</tr>
<tr>
<td>E</td>
<td>Extends linker capacity by making 2 passes</td>
</tr>
<tr>
<td>Q (symbol, ..)</td>
<td>Quick libraries - code placed low in memory</td>
</tr>
</tbody>
</table>
Bootstrap tool

Single transputer target

The bootstrap tool adds the loading information
### Bootstrap options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>O filename</code></td>
<td>Output file - default *.bxx</td>
</tr>
<tr>
<td><code>I</code></td>
<td>Information</td>
</tr>
<tr>
<td><code>S stacksize</code></td>
<td>Stack size in words</td>
</tr>
<tr>
<td><code>C</code></td>
<td>Task image for configuration</td>
</tr>
</tbody>
</table>

### Make
Overview

Make is a standard system program management tool.
A single command compiles, links and configures as needed.
When source is edited, make will recompile etc only the
minimum of files.
Not supplied with the toolset -
public domain, available freely.
Uses a dependency makefile.
Borland, UNIX and GNU makes supported.

Using make

Makefile → C source "myprog.c" → Configuration source → Make → Bootable file
Library build → Compiled library

make -f myprog
Building makefiles

output.fil: input1.fil input2.fil
  toolname param1 param2

For each file:

- Line 1 gives file name and lists files on which it depends;
- Line 2 gives command line to generate file.

◊ indicates tab character.

Complete makefile

myprog.b8x: myprog.c8x
  iboot myprog.c8x

myprog.c8x: myprog.18x myprog.bin
  ilink /f myprog.18x

myprog.bin: myprog.c
  t8c myprog

The first file to be created is the last in the make file.
Other makes use the opposite order.
Server purpose

Server boots target, loads program
and provides access to host services.
Running a program

To run a program called program.b8x:-

```
iserver /se /sb program.b8x
```

To continue with a program without rebooting:-

```
iserver /se /ss data
```

Server options

<table>
<thead>
<tr>
<th>SB filename</th>
<th>SE</th>
<th>SI</th>
<th>SL address</th>
<th>SR</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot file onto root transputer.</td>
<td>Transputer error flag polled.</td>
<td>Information.</td>
<td>Link address or device name.</td>
<td>Reset root transputer.</td>
<td>Rerun program without rebooting.</td>
</tr>
</tbody>
</table>
Hello World practical

Write code to display a short message on the screen.
Create a directory for your work with the command md.
Enter the directory with the command cd.
Create a subdirectory for this practical.
Build a makefile and an indirect linker file.
Compile your program using make.
Create a batch file to boot the transputer and run a program.
Channels

Channels
Using channels
Change case practical
Integer buffer

```c
#include <chan.h>

int_buffer (in, out)
CHAN *in, *out;
{
    int n;
    for (;;) {
        chan_in_word (&n, in);
        chan_out_word (n, out);
    }
}
```
### Array buffer

```c
#include <chan.h>
#define length 80

array_buffer (in, out)
CHAN *in, *out;
{
    char string[length];
    for (;;)
    {
        chan_in_message (length, string, in);
        chan_out_message (length, string, out);
    }
}
```

### Character buffer

```c
#include <chan.h>

char_buffer (in, out)
CHAN *in, *out;
{
    char letter;
    for (;;)
    {
        chan_in_byte (&letter, in);
        chan_out_byte (letter, out);
    }
```
### Channel i/o functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Parameter type</th>
</tr>
</thead>
<tbody>
<tr>
<td>chan_in_byte</td>
<td>buffer channel</td>
<td>pointer CHAN pointer</td>
</tr>
<tr>
<td>chan_in_word</td>
<td>buffer channel</td>
<td>pointer CHAN pointer</td>
</tr>
<tr>
<td>chan_in_message</td>
<td>no of bytes</td>
<td>integer pointer CHAN pointer</td>
</tr>
<tr>
<td></td>
<td>buffer channel</td>
<td></td>
</tr>
<tr>
<td>chan_out_byte</td>
<td>data channel</td>
<td>byte, eg char CHAN pointer</td>
</tr>
<tr>
<td>chan_out_word</td>
<td>data channel</td>
<td>word, eg int CHAN pointer</td>
</tr>
<tr>
<td>chan_out_message</td>
<td>no of bytes</td>
<td>integer pointer CHAN pointer</td>
</tr>
<tr>
<td></td>
<td>buffer channel</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters of main

Fixed parameters give access to channels.

```c
#include <chan.h>

main (argc, argv, envp, in_ports, inlen, out_ports, outlen)
int argc; /* no of tokens in argv */
char *argv[]; /* token strings */
char *envp[]; /* NULL */
CHAN *in_ports[], *out_ports[]; /* channel pointer arrays */
int inlen, outlen; /* sizes of channel pointer arrays */
{
  int value, i;
  chan_out_word (value, out_ports[i]);
}
```
Multi-task program

I/O process

Standalone process

Reserved channels

<table>
<thead>
<tr>
<th></th>
<th>IO process</th>
<th>Standalone process</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_ports[0]</td>
<td>unused</td>
<td>unused</td>
</tr>
<tr>
<td>out_ports[0]</td>
<td>debug</td>
<td>debug</td>
</tr>
<tr>
<td>in_ports[1]</td>
<td>standard input</td>
<td>standard output</td>
</tr>
<tr>
<td>out_ports[1]</td>
<td>standard output</td>
<td>standard output</td>
</tr>
</tbody>
</table>
Using channels

Communication security

Unmatched communications cause deadlock.
Only occam checks communications.
In C, the programmer is responsible.
Use good programming practice to avoid errors:
   draw data flow diagrams;
   put input and output in library subprograms;
   use meaningful names.
Variable size arrays

```c
in_array (message, len, in)
    char message[];
    int len;
    CHAN *out;
    
    chan_in_word(len, in);
    chan_in_message(len, message, in);

out_array (message, len, out)
    char message[];
    int len;
    CHAN *out;
    
    chan_out_word(len, out);
    chan_out_message(len, message, out);
```

Structures

```c
struct record {int n; float x; char ch};

struct record in_record(in)
    CHAN *in;
    
    {struct record r;
        chan_in_message (sizeof(r), &r, in);
        return(r);
    }

out_record (r, out)
    struct record r;
    CHAN *out;
    
    {chan_out_message (sizeof(r), &r, out);
    }
```
Unions

union quantity {int no; double weight;};

out_quantity (q, out)
    union quantity q;
    CHAN *out;
    { chan_out_message (sizeof(q), &q, out); }

in_quantity (q, in)
    union quantity q;
    CHAN *in;
    { chan_in_message (sizeof(q), &q, in); }

Change case practical

Write a case changing program consisting of two tasks.

The terminal handler alternately reads the keyboard and the results channel. It passes on all characters from the keyboard to the case changer. When it receives any character from the results channel it displays it on the screen. When it receives a '0/0' character it terminates the server and itself and passes on the terminate message to the case changer.
The case changer process changes upper case letters to lower case and lower case to upper and sends the result back down the results channel. Any other characters are returned unchanged. When it receives a terminate message it terminates itself.
Compile all the code.
Multi-task programs

Configuration
Task attributes
The configurer
Configuration practical

Configuration
Tasks

Separate C programs.
Communication by channels.
Joined by the configurer.
Separate data space, code and libraries.
All data initialised.

Configuration purpose

Describe hardware.
Describe tasks.
Connect tasks.
Map software onto hardware.
Describe hardware

![Diagram of hardware: host connected to root link 0]

- processor host
- processor root

wire link root[0] host[0]
!host connected to root link 0

Hidden software

![Diagram of software: iserver connected to filter, which is connected to io, which is connected to app]

Filter is a standard task provided with Parallel C.
Describe tasks

- task io  ins=3 outs=3
- task app  ins=2 outs=2 data=10k
- task filter  ins=2 outs=2 data=10k
- task iserver  ins=1 outs=1

Connecting tasks

Numbers refer to channel subscripts in C source.

- connect ?  filter[0]  iserver[0]
- connect ?  iserver[0]  filter[0]
Mapping

place iserver host
place io root
place app root
place filter root

Complete configuration

processor host
processor root
wire link root[0] host[0]
task io ins=3 outs=3
task app ins=2 outs=2 data=10k
task filter ins=2 outs=2 data=10k
task iserver ins=1 outs=1
place iserver host
place io root
place app root
place filter root
connect ? filter[0] iserver[0]
connect ? iserver[0] filter[0]
Task attributes

Channels

Number of input and output channels defined by INS and OUTS respectively. Includes hidden channels. Must be given for all tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Ins</th>
<th>Outs</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>io</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>app</td>
<td>2</td>
<td>2</td>
<td>10k</td>
</tr>
<tr>
<td>filter</td>
<td>2</td>
<td>2</td>
<td>10k</td>
</tr>
<tr>
<td>iserver</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Task file

Default filename for linked code is

```
task name with extension .b4
```

Use file attribute to override default.

```
task io ins=3 outs=3 file=termhand.b8x
```

```
task app ins=2 outs=2 data=10k
```

Priority

Transputer has two levels of priority:

NOTURGENT and URGENT.

Default is NOTURGENT.

```
task io ins=3 outs=3 urgent
```

```
task app ins=2 outs=2 data=10k
```
Memory allocation

Either total data space is declared in bytes
or stack and heap (static) separately.
Scaling factors k and M may be used.
Minimum stack 1k, minimum heap 5k.
At most one task per processor may have free memory.

```
task io  ins=3  outs=3
task appl ins=3 outs=3 data=10k
task app2 ins=2 outs=2
    stack=1k  heap=10k
```

On-chip memory

For maximum performance
place critical data / code
in fastest memory.

- 3 cycle read / write
  (2 cycle on IMS T801)
  33 Mbytes/sec
- 4 kbyte
  single cycle read / write
  100 Mbytes/sec
Using on-chip memory

Stack, code or heap may be placed on chip using OPT attribute.
The linker places code in the order of linking, starting on fastest memory.

```
task io ins=3 outs=3
task appl ins=2 outs=2 data=10k
task app2 ins=2 outs=2 stack=1k -
    heap=10k opt=stack opt=code
```
Linking and booting

Each user task must be linked with taskharn.

```
ilink taskharn.t8x task1.bin sacrtl.bin /o task1.c8x
```

Add bootstrap to each task with /c option and output file *.b4.

```
iboot task1.c8x /o task1.b4 /c
```

Configuring

Configuration file "myprog.cfg"

Bootable tasks "io.b4", "app.b4"

Any number of tasks on each processor.

```
t8c io
ilink taskharn.t8x io.bin crtl.lib /o io.c8x
iboot io.c8x /o io.b4 /c
t8c app
ilink taskharn.t8x app.bin sacrtl.lib /o app.c8x
iboot app.c8x /o app.b4 /c
config myprog.cfg myprog.btl
```
Makefile

myprog.btl: io.b4 app.b4 myprog.cfg
Oconfig myprog.cfg myprog.btl

io.b4: io.c8x
Oiboot io.c8x /c /o io.b4

io.c8x: io.18x io.bin
Oilink /f io.18x

io.bin: io.c
Ot8c io

app.b4: app.c8x
Oiboot app.c8x /c /o app.b4

app.c8x: app.18x app.bin
Oilink /f app.18x

app.bin: app.c
Ot8c app

Indirect linker files

File: io.l8x
- taskharn.t8x
- io.bin
- crtl.lib
  /o io.c8x

File: app.l8x
- taskharn.t8x
- app.bin
- sacrtl.lib
  /o app.c8x
Change case practical - part 2

Write a configuration file for your change case code.
Write a makefile and indirect linker files.
Compile the code and run it.
Multi-transputer systems

Board connections
Course system
Board wiring practical

Board connections
**Links**

Link cables are multicoloured pairs of twisted pairs.

End plugs have pin 4 blanked off.

**Reset pins**

Reset pin to initialise.

Analyse pin to debug

Error pin to detect error.

Connection to these three pins allows control of the transputer.
Reset ports

Master board

Slave boards

Up/Down propagates reset, analyse and error by hardware.

SubSys is a latch accessed by software.

INMOS board system control

Reset cables are grey or black three core.

Up end plugs have pin 5 blanked off.

Down and Subsys end plugs have pins 4 and 5 blanked off.
Training course system

IBM XT / AT

File server Editor

IMS B008 PC motherboard

IMS B404 T800 module 2Mbyte

Compilers Linker Configurer Utilities

IBMT bus

Host

Root

Target

IMS B004 PC add-in board

IMS T414 2 Mbytes

Application
IMS B008 PC motherboard

Master

Slaves

Host link

Link switch

Pipetail

10 Edge Links

ConfigDown

PC motherboard break-out board pinout

DN Down

SS Subsystem

L11 ConfigDown

L10 PipeTail

L9 Edge Link 9 (= Patch link 1)

L8 Edge Link 8 (= Patch link 0)

L7 Edge Link 7

L6 Edge Link 6

L5 Edge Link 5

L4 Edge Link 4

L3 Edge Link 3

L2 Edge Link 2

L1 Edge Link 1

L0 Edge Link 0

UP Up
IMS B404 2Mbyte transputer module

IMS B004 PC add-in board
Run your change case code on two transputers.

Connect the PC motherboard to the PC add-in board as shown:

Use setc004 to set the link switch. Check the wiring with check. Copy and change your configuration file to two transputers. Change your makefile to compile the case changer for an IMS T414. Make and run your code.
Priority and time

Priority
Priority

The transputer has two levels of priority:

0 high priority
  - no timeslicing.

1 low priority (default)
  - possible timeslice after 1 to 2 msec.
  - possible interrupt by high priority.

Priority can affect performance.

Bit zero of the workspace pointer shows priority.

Defining priority

Priority of a task is fixed at configuration time.

Low priority is default.

High priority is URGENT.

task app ins=2 outs=2
 task io ins=4 outs=4 data=10k urgent
Uses of priority

Use high priority for:

- handling link communications;
- handling interrupts;
- timing fast processes.

Timers
Transputer time

Two registers, clock0 and clock1.
Incremented regularly ± 200ppm.
Used for:
  - benchmarking;
  - delays;
  - timeouts;
  - timed events.

Clock rates

<table>
<thead>
<tr>
<th>priority</th>
<th>low</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>time between ticks</td>
<td>64 microsecs</td>
<td>1 microsec</td>
</tr>
<tr>
<td>ticks per second</td>
<td>15625</td>
<td>1000000</td>
</tr>
<tr>
<td>approx time between resets - 16 bit</td>
<td>4 secs</td>
<td>1/16 sec</td>
</tr>
<tr>
<td>approx time between resets - 32 bit</td>
<td>76 hours</td>
<td>1 hour</td>
</tr>
</tbody>
</table>
Timing a process

Timer\_now gives current time in ticks.

```c
#include <timer.h>

benchmark()
{
    int start, end;
    start = timer\_now();
    marathon\_process();
    end = timer\_now();
}
```

Time arithmetic

Use modulo operators for time.
Comparing times

Use timer_after to compare times.

```c
#define FALSE 0
#define RUN_TIME 1000
#include <timer.h>

timed_out_loop()
{
    int now, stop_time;
    now = timer_now();
    stop_time = timer_plus(now, RUN_TIME);
    while (timer_after(stop_time, now) != FALSE)
    {
        main_routine();
        now = timer_now();
    }
}
```

Modulo arithmetic

Use t-code diff, sum and times.

```c
int timer_minus (x, y)
int x, y;
{
    int z;
    asm {ldl x; ldl y; diff; stl z;}
    return (z);
}

int timer_plus (x, y)
int x, y;
{
    int z;
    asm {ldl x; ldl y; sum; stl z;}
    return (z);
}
```
Timer delay

Use timer_delay.

Current task or thread is descheduled and uses no processor time.

Delay is in ticks.

```c
#include <timer.h>
reset_subsystem()
{
    analyse_signal (0);
    reset_signal (1);
    timer_delay (1);
    reset_signal (0);
    timer_delay (1);
}
```

Regular pulses

```
#include <timer.h>
ticker (delay, out)
    int delay;
    CHAN *out;
{
    int time;
    time = timer_now();
    for (;;)
    {
        time = timer_plus (time, delay);
        timer_wait (time);
        chan_out_word (0, out);
    }
```

Timer functions

```c
#include <timer.h>

timer_now () read current timer value.
timer_wait (wake_time) deschedule until wake_time.
timer_delay (delay) deschedule for delay ticks.
timer_after (t1, t2) non-zero if t1 after t2, zero otherwise.
```

Channel input with timeout

```c
#include <chan.h>

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Parameter type</th>
</tr>
</thead>
<tbody>
<tr>
<td>chan_in_byte_t</td>
<td>buffer channel timeout</td>
<td>pointer CHAN pointer int</td>
</tr>
<tr>
<td>chan_in_word_t</td>
<td>buffer channel timeout</td>
<td>pointer CHAN pointer int</td>
</tr>
<tr>
<td>chan_in_message_t</td>
<td>no of bytes buffer channel timeout</td>
<td>int pointer CHAN pointer int</td>
</tr>
</tbody>
</table>

Similarly output.

Timeout is in timer ticks.
Host time

#include <time.h>

<table>
<thead>
<tr>
<th>Params</th>
<th>Result Type</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>none</td>
<td>int</td>
</tr>
<tr>
<td>time</td>
<td>none</td>
<td>int</td>
</tr>
</tbody>
</table>

Provided only in i/o run-time library.

Clock practical

Write a process to display a clock that starts from zero and runs forever.
Write two parallel processes as shown.

Ticker sends a signal every second.
The screen handler stores the time, incremented every second, and displays it.
Threads

Overview
Thread functions
Semaphores
Multiplexor

Thread overview

Parallel code within a task.
Used for parallel output and input in a single task.
Must run on the same transputer.
Urgent or not urgent priority.
Shared static, extern and heap data.
Each thread has own stack for auto variables.
Use semaphores for allocation of shared resources.
Extensive library support.
Thread functions

Starting a simple thread

```c
#include <thread.h>

thread_create (fun, ws_size,
       n, arg1, arg2, ..., argn);
 void (*function)();
 int ws_size;
 int n, arg1, arg2, ..., argn;
```

The thread will execute the function `fun` with:
- arguments `arg1`, `arg2`, ..., `argn`;
- workspace of size `ws_size` bytes from the heap;
- current priority.
Starting a general thread

```c
#include <thread.h>

thread_start (fun, worksp, ws_size, 
    priority, 
    n, arg1, arg2, ..., argn);
void (*function)();
char *worksp;
int ws_size;
int n, arg1, arg2, ..., argn;
```

The thread will execute the function `fun` with:

- arguments `arg1`, `arg2`, ..., `argn`;
- workspace `worksp` of size `ws_size` bytes;
- priority `THREAD_URGENT` or `THREAD_NOTURG`.

Terminating a thread

Every thread created remains as a parallel process until:

- the function it executes returns or
- the function executes `thread_stop`.

```c
#include <thread.h>

thread_stop ();
```
## Other thread functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread_priority</td>
<td>Returns priority.</td>
</tr>
<tr>
<td>thread_deschedule</td>
<td>Deschedule briefly.</td>
</tr>
<tr>
<td>thread_restart</td>
<td>Restart after resetting channel.</td>
</tr>
</tbody>
</table>

## Semaphores
**Purpose**

To share resources, e.g., variables or channels, between threads of the same priority.

Semaphore holds number of resources free.

**Stack of waiting threads**

- Thread with access
- Thread with access

**Semaphore**

- No resources free

---

**Setting up a semaphore**

```c
#include <sema.h>
SEMA *resource;
sema_init (resource, n);
```

Initialise semaphore to number of resources.

Type SEMA declared in sema.h.

**Empty stack of waiting threads**

- Free resource
- Free resource

**Semaphore**

- 2 resources free
Requesting resources

sema_wait (resource);
sema_wait_n (resource, n);

If resources available (ie semaphore > 0)
grabs resource (ie semaphore = semaphore - n)
else waits until resource available.

Releasing resource

sema_signal (resource);
sema_signal_n (resource, n);

Frees resource (ie semaphore = semaphore + n).
Gives it to a thread if any waiting.
Sharing run-time libraries

Only one thread may perform a library operation at once.
Use pre-defined semaphore par_sema to allocate.

Otherwise use interlocked functions:

```c
#include <par.h>
par_printf
par_fprintf
par_malloc
par_free
```

Multiplexor
The task

One task for each input channel.
Output channel and buffer are shared resources.

multiplexor

```
#include <chan.h>
#include <thread.h>
#include <sema.h>
char buffer[1024];
SEMA buffer_free;

multiplexor (in, no_ins, out)
   CHAN *in[], *out;
   int no_ins;
{
   extern void receive();
   int i;
   sema_init (&buffer_free, 1);
   for (i=0; i<no_ins; i++)
      thread_create (receive, 50*size_of(int),
                     2, in[i], out);
}
```
Semaphore

```c
void receive(in, out)
CHAN *in, *out;
{
    int msglen;
    for (;;) {
        chan_in_word (&msglen, in);
        sema_wait (&buffer_free);
        chan_in_message (msglen, &buffer[0], in);
        chan_out_word (msglen, out);
        chan_out_message (msglen, &buffer[0], out);
        sema_signal (&buffer_free);
    }
}
```
Flood configuring

Processor farms
Configuring

Processor farm

Master ← Worker ← Worker ← Worker ← Worker

Worker

Worker
Advantages and disadvantages

Advantages
Uses all available processors.
Automatically balances load.
Can be used on mixed networks (IMS T4 and T8).

Disadvantages
20kbytes of hidden code per node.
Master may be bottleneck.
Constrains task layout.
Workers must only compute.
Task frouter

frouter channels:

<table>
<thead>
<tr>
<th>channel</th>
<th>direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2</td>
<td>down</td>
</tr>
<tr>
<td>3</td>
<td>up</td>
</tr>
<tr>
<td>4</td>
<td>to master</td>
</tr>
<tr>
<td>5</td>
<td>to worker</td>
</tr>
</tbody>
</table>

frouter is a standard task provided with Parallel C.

Communications

Communications between master and worker use the library functions `net_send` and `net_receive`.

Master communicates with next ready worker.
Messages may be complete tasks or results.
Workers cannot communicate with each other.
Reading messages

Messages are chopped into packets.

```c
read_message (buffer, length)
char buffer[];
int length;
{
  int complete;
  length = 0;
  complete = 0;
  while (complete != 1)
  {
    length = length +
      net_receive (&buffer[length], complete);
  }
}
```

Sending messages

Maximum packet size is NET_MAX_PACKET_LENGTH bytes (1024).

```c
#define PACKET_SIZE 256
send_message (message, length)
char message[];
int length;
{
  int complete, start, to_send;
  to_send = length;
  start = 0;
  while (to_send > PACKET_SIZE)
  {
    start = start +
      net_send (PACKET_SIZE, &message[start], 0);
    to_send = length - start
  }
  start = start +
      net_send (to_send, &message[start], 1);
}
```
Worker

Worker is usually sequential.

```c
#include <net.h>
main ()
{
    char data[MAX_DATA], results[MAX_RESULTS];
    int data_len, results_len;
    for (;;)
    {
        read_message (data, data_len);
        do_task (data, data_len, results, results_len);
        send_message (results, results_len);
    }
}
```

Master

Master uses threads to send and receive in parallel.

```c
#include <thread.h>
#include <sema.h>
main ()
{
    SEMA io_ready;
    sema_init (&io_ready, 1);
    thread_create (send_data, 10000,
                   3, io_ready, fs, ts);
    display_results (io_ready, fs, ts);
}
Configuring

Configuration file

Flood configuration needs only:

description of master task;
description of worker task.

```
task master file=mymaster
task worker file=myworker data=10k
```

The names master and worker are understood by the configurer.
Mixed transputer networks

task t4master file=master4
task t4worker file=worker4 data=10k
task t8master file=master8
task t8worker file=worker8 data=10k

The names t4master, t8master, t4worker and t8worker
are understood by the configurer.

Configuring

Configuration
file
"myprog.cfg"

Bootable
tasks
"master.b4", "worker.b4"

Configurer

fconfig

btl

```
t8c master
ilink taskharn.t8x master.bin crtl.lib /o io.c8x
iboot io.c8x /o io.b4 /c
t8c app
ilink taskharn.t8x app.bin sacrtl.lib /o app.c8x
iboot app.c8x /o app.b4 /c
config myprog.cfg myprog.btl
```
Makefile

myprog.btl: master.b4 worker.b4 myprog.cfg
    fconfig myprog.cfg myprog.btl

master.b4: master.c8x
    iboot master.c8x /c /o master.b4

master.c8x: master.18x master.bin
    ilink /f master.18x

master.bin: master.c
    t8c master

worker.b4: worker.c8x
    iboot worker.c8x /c /o worker.b4

worker.c8x: worker.18x worker.bin
    ilink /f worker.18x

worker.bin: worker.c
    t8c worker
Other tools

Binary decoder
Libraries
Decoder use

```
bin
   decoder
   decode
   To screen

c
```

To interpret compiled binary files, i.e., .bin files.
Disassembles and matches with source.
Gives assorted other information.
Outputs to screen.

```
delete myprog > myprog.dcd
```

Decoder output

```
Transputer DECODE (V1.2) of hello.bin
ID T4 "occam 2 V2.1" "CC_transputer V2.0.1"
SC 0
TOTALCODE 144 0
STATIC 1
REF #0, "printf"
   20 0008F
   00000000 00058
   4324F4C 4C454807 00000001 0007C
   1 main ()

...............

2 { printf ("Hello World");
   646C726F 57206F6C 65480C 00000
   00 0000C
   46 64 00007 ldc -74
   00 00007 1dp
   70 0004B ld1 0
   20 20 20 20 20 20 20 0004C call printf
   B1 00052 ajw 1
   F0 22 00053 ret

...............

4 }
```
Libraries

Using libraries

Main source "main.c" → Compiler → bin

Compiled and built library "mylib.lib" → Linker → CXX

#include <mylib.lib>

t8c main
ilink mainharn.c8x main.bin mylib.lib crtl.lib
Creating a library

Compile all the code in modules - i.e. the smallest unit that can be loaded.
For multiple targets and modes, compile for each.
Create a library build file.
Run the librarian, ilibr, to join the modules.

```
iliibr /f mylib.lbb
```

Library build files

Lists the files used to build a library.
Used by the librarian and make.

```
<table>
<thead>
<tr>
<th>File: mylib.lbb</th>
</tr>
</thead>
<tbody>
<tr>
<td>myproc1.c4x</td>
</tr>
<tr>
<td>myproc1.c8x</td>
</tr>
<tr>
<td>myproc2.c4x</td>
</tr>
</tbody>
</table>
```

```
iliibr /f mylib.lbb
```
Makefiles

Libraries may have separate makefiles

or be included in makefiles of user programs.

```
mylib.lib: mylib.libb myprocl.c4x \\
myprocl.c8x myproc2.c4x
libmr /f mylib.lib
```

Include user libraries in dependency lists.

```
main.bin: main.c mylib.lib
	t8c main
```
C language implementation

- Standards and extensions
- Data type implementation
- Run time libraries

Standards and extensions
Standards

The 3L compiler complies with:
Kernighan & Ritchie - 1978
ANSI standard X3J11.

Sequential extensions

Assignment to whole struct and union variables.
Functions with structures as arguments or results.
No restrictions on common struct member names.
31 significant characters in identifiers.
$ allowed anywhere in identifiers.
Escape sequences of unlimited length in strings, eg \ddddddd.
Library for access to DOS host system.
Multiple type-specifiers in type-names.
Other features

Register type implemented as integers.
enum and void are keywords. entry is not.
void data types allowed, but not enum.
All bit fields implemented as unsigned int.
>> gives a logical shift, rather than arithmetic.
Loose type checking of . and -> operators.
White space is not allowed within compound operators.
sizeof is not allowed in array size expressions.
#line is accepted but ignored.
=op and int x 3 are not allowed.

Assembling T-code

```c
int a=123, b;
asm { ld1 a; ld1 b; add; stl b;}
```

Assembler works out prefixes and operations.
Symbolic labels and identifiers.
Macros

Compiler switch d defines a macro.
Default value is 1.

```
t8c /dDEBUG /dhelp=3 /dJOE=Jim cats
```

is equivalent to:

```
#define DEBUG 1
#define help 3
#define JOE Jim
```

at the top of cats.c.

Data type implementation
### Simple C types

Sizes of variables and constants in bytes.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>32 bit transputer</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>character</td>
<td>1</td>
</tr>
<tr>
<td>register</td>
<td>32 bit integer</td>
<td>4</td>
</tr>
<tr>
<td>int</td>
<td>32 bit integer</td>
<td>4</td>
</tr>
<tr>
<td>unsigned int</td>
<td>16 bit unsigned integer</td>
<td>4</td>
</tr>
<tr>
<td>short</td>
<td>32 bit integer</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>32 bit integer</td>
<td>4</td>
</tr>
<tr>
<td>float</td>
<td>32 bit floating point</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>64 bit floating point</td>
<td>8</td>
</tr>
<tr>
<td>pointer</td>
<td>address</td>
<td>4</td>
</tr>
</tbody>
</table>

All types except char are word aligned.

### Other C types

struct and union types are rounded up to whole words.

Each sequence in a struct is byte aligned. Sequences of more than one byte are word aligned.

```c
struct s {char c; int f:7, g:7; char d}
```

```c
struct s {char c; int f:7; char d; int g:7}
```

enum is not implemented.
Run time libraries

Library files

<table>
<thead>
<tr>
<th>Directory</th>
<th>IO process library</th>
<th>Stand-alone library</th>
</tr>
</thead>
<tbody>
<tr>
<td>\tc2v0\libs</td>
<td>crtl.lib</td>
<td>sacrt1.lib</td>
</tr>
</tbody>
</table>
### Library contents

<table>
<thead>
<tr>
<th>IO library</th>
<th>Stand-alone library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input and output routines</td>
<td>Channel i/o routines</td>
</tr>
<tr>
<td>Channel i/o routines</td>
<td></td>
</tr>
<tr>
<td>Parallel processing</td>
<td>Parallel processing</td>
</tr>
<tr>
<td>Transputer routines</td>
<td>Transputer routines</td>
</tr>
<tr>
<td>Mathematical routines</td>
<td>Mathematical routines</td>
</tr>
<tr>
<td>String handling routines</td>
<td>String handling routines</td>
</tr>
<tr>
<td>Date and time routines</td>
<td></td>
</tr>
</tbody>
</table>

### Standard run-time libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Header</th>
<th>Stand-alone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard i/o</td>
<td>#include &lt;stdio.h&gt;</td>
<td></td>
</tr>
<tr>
<td>Stream i/o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary i/o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text i/o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mathematical</td>
<td>#include &lt;math.h&gt;</td>
<td>✓</td>
</tr>
<tr>
<td>String handling</td>
<td>#include &lt;string.h&gt;</td>
<td>✓</td>
</tr>
<tr>
<td>Character classification</td>
<td>#include &lt;ctype.h&gt;</td>
<td>✓</td>
</tr>
<tr>
<td>Conversion</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Memory allocation</td>
<td>#include &lt;assert.h&gt;</td>
<td>✓</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>#include &lt;setjmp.h&gt;</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>#include &lt;ascll.h&gt;</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>#include &lt;errno.h&gt;</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Non-standard run-time libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Header</th>
<th>Stand-alone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel i/o</td>
<td>#include &lt;chan.h&gt;</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>#include &lt;chanio.h&gt;</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>#include &lt;boot.h&gt;</td>
<td>✔</td>
</tr>
<tr>
<td>Booting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timers</td>
<td>#include &lt;timer.h&gt;</td>
<td>✔</td>
</tr>
<tr>
<td>Threads</td>
<td>#include &lt;thread.h&gt;</td>
<td>✔</td>
</tr>
<tr>
<td>Semaphores</td>
<td>#include &lt;sema.h&gt;</td>
<td>✔</td>
</tr>
<tr>
<td>Parallel processing</td>
<td>#include &lt;par.h&gt;</td>
<td>✔</td>
</tr>
<tr>
<td>Processor farm</td>
<td>#include &lt;net.h&gt;</td>
<td>✔</td>
</tr>
<tr>
<td>Date and time</td>
<td>#include &lt;time.h&gt;</td>
<td></td>
</tr>
<tr>
<td>DOS</td>
<td>#include &lt;dos.h&gt;</td>
<td></td>
</tr>
</tbody>
</table>

### Links

- [Link 1](#)
- [Link 2](#)
- [Link 3](#)
Link input and output

Links may be used to talk to external devices

eg a linkswitch;

a link adaptor;

an unbooted transputer.

---

Driving a link adaptor

Use channels LinkInput and LinkOutput

(defined in <chan.h>) to talk down links.

```c
#include <chan.h>

send_to_device (data)
int data;
{
    chan_out_word (data, Link2Output);
}

read_from_device (result)
int result;
{
    chan_in_word (&result, Link2Input);
}
```
Bind statement

Channels may be bound to i/o links with the BIND statement in the configuration.

```
task link_adaptor ins=3 outs=3 data=10k
bind input  link_adaptor[2] value=&80000010
bind output link_adaptor[2] value=&80000000
```

Link addresses

<table>
<thead>
<tr>
<th>Link</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link0Output</td>
<td>&amp;80000000</td>
</tr>
<tr>
<td>Link1Output</td>
<td>&amp;80000004</td>
</tr>
<tr>
<td>Link2Output</td>
<td>&amp;80000008</td>
</tr>
<tr>
<td>Link3Output</td>
<td>&amp;8000000C</td>
</tr>
<tr>
<td>Link0Input</td>
<td>&amp;80000010</td>
</tr>
<tr>
<td>Link1Input</td>
<td>&amp;80000014</td>
</tr>
<tr>
<td>Link2Input</td>
<td>&amp;80000018</td>
</tr>
<tr>
<td>Link3Input</td>
<td>&amp;8000001C</td>
</tr>
</tbody>
</table>
Events

Event handling

Event is like a link but only synchronises.
Generally used for interrupts.
Should be handled by a separate task.
Programming events

Event is like an input only link.
No data is received.
Use high priority for interrupts.
EventReq is defined in <chan.h>.

```c
event_handler ()
{
    for (; ;)
    {
        int trash;
        chan_in_word (trash, EventReq);
        deal_with_event();
    }
}
```

Binding events

Channels may be bound to the event hardware
with the BIND statement in the configuration.

```
task event_handler ins=3 outs=3 data=10k
bind input event_handler[2] value=&80000020
```
Hello World - possible solution page 1

File: hello.c

```c
main ()
{
    printf ("Hello World");
}
```

File: hello.b8x -- makefile

```make
hello.b8x: hello.c8x
    objboot hello.c8x

hello.c8x: hello.18x hello.bin
    objlink /f hello.18x

hello.bin: hello.c
    t4c hello
```

File: hello.l8x -- linker file

```make
mainent.c8x
hello.bin
crtl.lib
/o hello.c8x
```

Hello World - possible solution page 2

File: run.bat

```bash
iserver /se /sb %1.b8x
```

To compile and run hello:

```bash
make -f hello
run hello
```
Change case • possible solution page 1

File: casech.c

```c
#include <chan.h>
#include <ctype.h>
#define keys in
#define results out
#define data (char) 0
#define terminate (char) 1

main (argc, argv, envp, in, inlen, out, outlen)
    char *argv[], *envp[];
    int argc, inlen, outlen;
   CHAN *in[], *out[];
{
    char tag, ch;
    int going = 1;
    while (going == 1)
    {
        chan_in_byte (&tag, keys);
        if (Tag == terminate)
            going = 0;
        else
            chan_in_byte (&ch, keys);
        chan_out_byte (chcase(ch), results);
    }
}
```

```c
char chcase (ch)
    char ch;
    {
        if (isupper(ch))
            return (tolower(ch));
        else if (islower(ch))
            return (toupper(ch));
        else
            return (ch);
    }
```

Change case • possible solution page 2

File: termhand.c

```c
#include <chan.h>
#include <stdio.h>
#define keys in
#define results out
#define data (char) 0
#define terminate (char) 1
#define terminatech '
'
main (argc, argv, envp, in, inlen, out, outlen)
    char *argv[], *envp[];
    int argc, inlen, outlen;
   CHAN *in[], *out[];
{
    char ch;
    int going = 1;
    while (going == 1)
    {
        ch = getchar();
        if (ch == terminatech)
            { 
                going = 0;
                chan_out_byte (terminate, keys);
            }
        else
            { 
                chan_out_byte (data, keys)
                chan_out_byte (ch, keys);
                chan_in_byte (&ch, results);
                putchar(ch);
            }
    }
} 
```
### File: chcase.cfg

```plaintext
processor host
processor root
wire link root[0] host[0]
task termhand ins=3 outs=3
task casech ins=2 outs=2 data=10k
task filter ins=2 outs=2 data=10k
task iserver ins=1 outs=1
place iserver host
place termhand root
place casech root
place filter root
connect ? filter[0] iserver[0]
connect ? iserver[0] filter[0]
```

### File: chcase

```plaintext
chcase.btl:\termhand.b4 casech.b4 chcase.cfg
\config chcase.cfg chcase.btl

termhand.b4:\termhand.c8x
\iboot termhand.c8x /c /o termhand.b4

termhand.c8x:\termhand.18x termhand.bin
\ilink /f termhand.18x

termhand.bin:\termhand.c
\t8c termhand

casech.b4:\casech.c8x
\iboot casech.c8x /c /o casech.b4

casech.c8x:\casech.18x casech.bin
\ilink /f casech.18x

casech.bin:\casech.c
\t8c casech
```
Change case 2 - possible solution page 3

<table>
<thead>
<tr>
<th>File: termhand.l8x</th>
<th>File: casech.l8x</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskharn.t8x</td>
<td>taskharn.t8x</td>
</tr>
<tr>
<td>termhand.bin</td>
<td>casech.bin</td>
</tr>
<tr>
<td>ctrl.lib</td>
<td>sacrtl.lib</td>
</tr>
<tr>
<td>/o termhand.b4</td>
<td>/o casech.b4</td>
</tr>
</tbody>
</table>
Change case 3 - possible solution

File: chcase.cfg

<table>
<thead>
<tr>
<th>processor host</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor root</td>
</tr>
<tr>
<td>processor target</td>
</tr>
<tr>
<td>wire link1 host[0] root[0]</td>
</tr>
<tr>
<td>wire link2 root[3] target[0]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>task</th>
<th>ins</th>
<th>outs</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>termhand</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>casech</td>
<td>2</td>
<td>2</td>
<td>10k</td>
</tr>
<tr>
<td>filter</td>
<td>2</td>
<td>2</td>
<td>10k</td>
</tr>
<tr>
<td>iserver</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>place</th>
<th>host</th>
</tr>
</thead>
<tbody>
<tr>
<td>iserver</td>
<td></td>
</tr>
<tr>
<td>termhand</td>
<td></td>
</tr>
<tr>
<td>filter</td>
<td></td>
</tr>
<tr>
<td>casech</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>connect</th>
<th>filter[0]</th>
<th>iserver[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>iserver[0]</td>
<td>filter[0]</td>
</tr>
</tbody>
</table>
Change case - 3 possible solution page 3

chcase.btl : termhand.b4 casech.b4 chcase.cfg
config chcase.cfg chcase.btl
termhand.b4 : termhand.c8x
iboot termhand.c8x
termhand.c8x : termhand.18x termhand.bin
ilink /f termhand.18x
termhand.bin : termhand.c
t8c termhand
casech.b4 : casech.c4x
iboot casech.c4x
casech.c4x : casech.14x casech.bin
ilink /f casech.14x
casech.bin : casech.c
t4c casech
File: ticker.c

```c
#define tick out[1]
#define delay 15625
#include <timer.h>
#include <chan.h>

main (argc, argv, envp, in, inlen, out, outlen)
    char *argv[], *envp[];
    int argc, inlen, outlen;
    CHAN *in[], *out[];
{
    int time;
    time = timer_now();
    for (; ;)
    {
        time = timer_plus (time, delay);
        timer_wait (time);
        chan_out_word (0, tick);
    }
}
```

File: ticker.l8x

taskharn.t8x
ticker.bin
sacrt1.lib
/o ticker.c8x

File: scrnhand.c

```c
#include <chan.h>
#include <stdio.h>
#define tick in[2]

main (argc, argv, envp, in, inlen, out, outlen)
    char *argv[], *envp[];
    int argc, inlen, outlen;
    CHAN *in[], *out[];
{
    int signal, secs;
    for (secs=0; secs>=0; secs++)
    {
        printf ("%d\n", secs);
        chan_in_word (&signal, tick);
    }
}
```

File: scrnhand.l8x

taskharn.t8x
scrnhand.bin
crt1.lib
/o scrnhand.c8x
**File: clock.cfg**

<table>
<thead>
<tr>
<th>processor host</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor root</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>wire link root[0] host[0]</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>task scrnhand</th>
<th>ins=3 outs=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>task ticker</td>
<td>ins=2 outs=2 data=10k</td>
</tr>
<tr>
<td>task filter</td>
<td>ins=2 outs=2 data=10k</td>
</tr>
<tr>
<td>task iserver</td>
<td>ins=1 outs=1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>place iserver</th>
<th>host</th>
</tr>
</thead>
<tbody>
<tr>
<td>place scrnhand</td>
<td>root</td>
</tr>
<tr>
<td>place ticker</td>
<td>root</td>
</tr>
<tr>
<td>place filter</td>
<td>root</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>connect ?</th>
<th>filter[0] iserver[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect ?</td>
<td>iserver[0] filter[0]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>connect fs</th>
<th>filter[1] scrnhand[1]</th>
</tr>
</thead>
</table>


---

**File: clock - makefile**

```
clock.btl: scrnhand.b4 ticker.b4 clock.cfg
    config clock.cfg clock.btl

scrnhand.b4: scrnhand.c8x
    iboot scrnhand.c8x /c /o scrnhand.b4

scrnhand.c8x: scrnhand.18x scrnhand.bin
    t8c scrnhand

scrnhand.bin: scrnhand.c
    t8c scrnhand

ticker.b4: ticker.c8x
    iboot ticker.c8x /c /o ticker.b4

ticker.c8x: ticker.18x ticker.bin
    t8c ticker.18x

ticker.bin: ticker.c
    t8c ticker
```