

# TIS-100

TESSELLATED INTELLIGENCE SYSTEM

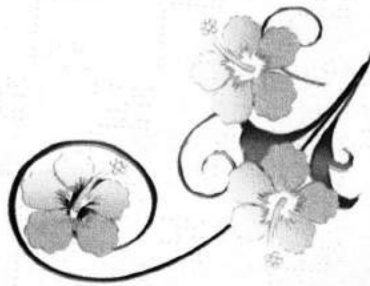
REFERENCE MANUAL

M--

We are all still in shock here from Uncle Randy's sudden passing. While we wait on word from the examiner about the cause we are coping as best we can. I've been occupying myself sorting through his things, especially his computers. Of course, I took one look in the garage and it all looks like a bunch of junk to me. I'll send some pictures when I get a chance.

For now, this is the machine that was set up on his workbench when he died. Maybe you'll be able to figure out what he was doing with it. He would've liked to know someone was going to finish his work.

Love,  
Aunt Doris.



# SYSTEM ARCHITECTURE OVERVIEW ORGANIZATION

The Tessellated Intelligence System is a massively parallel computer architecture comprised of non-uniformly interconnected heterogeneous nodes. The Tessellated Intelligence System is ideal for applications requiring complex data stream processing, such as automated financial trading, bulk data collection, and civilian behavioral analysis.

**Note:** Notes like this one will appear in this manual to indicate scenarios requiring special attention and to refer to other documents that contain more information on a topic.

Note: If you are looking for more information on a specific node or node type, the information is located in the appropriate node type manual. For example, if you are looking for information on the Node 1000, you would refer to the Node 1000 manual. The Tessellated Intelligence System is a massively parallel computer architecture comprised of non-uniformly interconnected heterogeneous nodes. The Tessellated Intelligence System is ideal for applications requiring complex data stream processing, such as automated financial trading, bulk data collection, and civilian behavioral analysis.

Note: If a node is used in a specific application, the information is located in the appropriate application manual. For example, if you are looking for information on the Node 1000 in a specific application, you would refer to the application manual. The Tessellated Intelligence System is a massively parallel computer architecture comprised of non-uniformly interconnected heterogeneous nodes. The Tessellated Intelligence System is ideal for applications requiring complex data stream processing, such as automated financial trading, bulk data collection, and civilian behavioral analysis.

Note: This document does not describe the hardware or software for a specific node or node type, as these values vary by node and hardware revision. Refer to the node-specific manual for a detailed description of performance characteristics of a particular device.

...information on a specific node or node type, the information is located in the appropriate node type manual. For example, if you are looking for information on the Node 1000, you would refer to the Node 1000 manual. The Tessellated Intelligence System is a massively parallel computer architecture comprised of non-uniformly interconnected heterogeneous nodes. The Tessellated Intelligence System is ideal for applications requiring complex data stream processing, such as automated financial trading, bulk data collection, and civilian behavioral analysis.

Note: If a node is used in a specific application, the information is located in the appropriate application manual. For example, if you are looking for information on the Node 1000 in a specific application, you would refer to the application manual. The Tessellated Intelligence System is a massively parallel computer architecture comprised of non-uniformly interconnected heterogeneous nodes. The Tessellated Intelligence System is ideal for applications requiring complex data stream processing, such as automated financial trading, bulk data collection, and civilian behavioral analysis.

Note: This document does not describe the hardware or software for a specific node or node type, as these values vary by node and hardware revision. Refer to the node-specific manual for a detailed description of performance characteristics of a particular device.

# SYSTEM ARCHITECTURE AND ORGANIZATION

---

The Tessellated Intelligence System consists of a large number of independent nodes connected on a local basis. (Refer to the model-specific manual to find the precise node population counts present on a particular device.) Node types can be broadly classified as processing or storage, with several variants within each category.

Generally, nodes are connected to up to four neighbors via ports. Ports enable lightweight message-passing communication between nodes. Communication over ports is coordinated by allowing either node to issue a read or write to a port and blocking until the request is filled by the corresponding node.

**Note:** If two nodes issue the same communication command (read or write) on the connection between them, the nodes will deadlock and a hardware fault will occur. Refer to the separate document "Tessellated Intelligence System Best Practices - Patterns of Node Communication" for details on how to use ports effectively and safely.

**Note:** If a node issues a communication command and it is never fulfilled by the corresponding node, the node will deadlock and a hardware fault will occur. (Exceptions to this rule exist; refer to the documentation of specific node types for details.) Refer to the separate document "Tessellated Intelligence System Best Practices - Patterns of Node Communication" for details on how to use ports effectively and safely.

**Note:** This document does not describe timing or throughput for node communication operations or instructions, as these values vary by model and hardware revision. Refer to the model-specific manual for a detailed description of performance characteristics of a particular device.

# NODE TYPE T20 - RESERVED

**Note:** This node type identifier is restricted to specific models of the Tessellated Intelligence System and will not be described in this document. Documentation for node type T20 is distributed only with systems containing this node type. **Unauthorized requests for copies of documentation describing this node are reported to the state security bureau, as required by law. ???**

The first section of the manual is devoted to a description of the system. The second section describes the program specific instructions and data operations to perform. Operations are performed sequentially, one after another, in the order in which they are listed. After the completion of the program, the user is notified by a message on the terminal. The terminal also displays the current value of the variables. Node type T20 is a program that is used to perform the following operations:

1. To read the contents of a file and store it in a system table. The program Node type T20 is a member of a group of programs that are used to perform the following operations. No additional memory is required. The user is notified by a message on the terminal. The user should see the value of the variable in Node type T20 storage table.

An operator can change values between operations. The operations of the program Node type T20 are performed in the following order: 1. Read the contents of a file and store it in a system table. 2. Store the contents of a file in a system table. 3. Read the contents of a file and store it in a system table. 4. Store the contents of a file in a system table.

## 1-1. ACC

Type: INT

Description: ACC is the primary storage register for the system. ACC is used to store the results of operations. The user is notified by a message on the terminal. The user should see the value of the variable in Node type T20 storage table.

## 1-2. BAK

Type: INT (1 to 255)

Description: BAK is a register, storage for values of ACC. It is used to store the results of operations. The user is notified by a message on the terminal. The user should see the value of the variable in Node type T20 storage table.

## 1-3. BIL

Type: INT (0 to 255)

Description: BIL is a register, storage for values of ACC. It is used to store the results of operations. The user is notified by a message on the terminal. The user should see the value of the variable in Node type T20 storage table.

## 1-4. LEFT, RIGHT, UP, DOWN

Type: INT

Description: The four independent registers LEFT, RIGHT, UP, and DOWN are stored in the first part of the system table. The user is notified by a message on the terminal. The user should see the value of the variable in Node type T20 storage table.

# NODE TYPE T21 - BASIC EXECUTION NODE

---

## 1. Architecture

The Basic Execution Node is responsible for coordinating the behavior of the Tessellated Intelligence System. Processing can occur within the Basic Execution Node, or can be delegated to specialized processing and storage nodes.

The Basic Execution Node executes a program specified in the Basic Execution Node Instruction Set. A Basic Execution Node program specifies computational and communication operations to perform. Operations are performed sequentially, beginning with the first instruction in the program. After executing the last instruction of the program, execution automatically continues to the first instruction. This behavior supports the common usage of Basic Execution Nodes, in which programs are written to operate in a continuous loop.

In addition to the communication ports common to all Tessellated Intelligence System nodes, the Basic Execution Node contains a number of registers that are used in the execution of its program. No additional memory is available on the Basic Execution Node; if additional storage is required, the node should coordinate with another Basic Execution Node or a storage node.

All registers store integer values between -999 and 999 (inclusive). The representation of register values is implementation-defined, and knowledge of the representation is not required to program the Basic Execution Node.

### 1-1. ACC

Type: Internal

**Description:** ACC is the primary storage register for a Basic Execution Node. ACC is used as the implicit source or destination operand of many instructions, including arithmetic and conditional instructions.

### 1-2. BAK

Type: Internal (non-addressable)

**Description:** BAK is temporary storage for values in ACC. It is only accessible through the SAV and SWP instructions, and cannot be read or written directly.

### 1-3. NIL

Type: Internal (special)

**Description:** Reading NIL produces the value zero. Writing to NIL has no effect. NIL can be used as a destination operand to execute an instruction for its side effects only, discarding the result.

### 1-4. LEFT, RIGHT, UP, DOWN

Type: Port

**Description:** The four communication registers UP, DOWN, LEFT, and RIGHT correspond to the four ports that all Basic Execution Nodes use to communicate with topologically adjacent nodes. Some ports will be disconnected on certain nodes within the hardware, and will block indefinitely if a READ or WRITE command is issued. Refer to the interconnection diagram for the node to determine which ports are available for use.

## 1-5. ANY

Type: Port (pseudo-port)

Description: When ANY is used as the source of an instruction, the instruction will read the first value that becomes available on any port. When ANY is used as the destination of an instruction, the result of the instruction will be sent to the first node that reads from any port on this node.

## 1-6. LAST

Type: Port (pseudo-port)

Description: LAST refers to the port last read or written using the ANY pseudo-port. It otherwise behaves identically to explicitly specifying a port. Reading from or writing to LAST before it has been set by a successful read or write using the ANY pseudo-port will result in implementation-defined behavior. Refer to the separate document "Tessellated Intelligence System Best Practices - Patterns of Node Communication" for sample code demonstrating the use of the LAST pseudo-port.

## 2-2. Labels

Type: Label

Description: Labels are used to identify targets for jump instructions. When an instruction jumps, the nodes following the label will be executed.

Example:

```
LOOP:
  MOV R, ACC
  JUMP LOOP
```

## 2-3. NOP

Type: Instruction

Description: NOP is a pseudo-instruction that does nothing.

NOP is implemented as a NOP instruction.

## 2-4. MOV

Type: Instruction

Description: MOV is used to move data between registers.

Example:

```
MOV R, ACC
MOV R, ACC
MOV R, ACC
```

## 2. Instruction Set

<SRC> and <DST> instruction parameters may specify a port or internal register. Any use of a port will block until the corresponding node connected to that port completes the communication by reading or writing a value. Additionally, a <SRC> parameter may be a literal integer value between -999 and 999 (inclusive).

BAK cannot be specified as a <SRC> or <DST> operand. The value of BAK is only accessible through special instructions SAV and SWP.

<LABEL> parameters are arbitrary textual names used to specify jump targets within the program.

### 2-1. Comments

**Syntax:** # COMMENT TEXT

**Description:** All text including and after the comment symbol (#) is ignored.

**Note:** Text placed after two comment symbols (##) will be used as the title of the program in which it occurs, and is displayed in the debugger to make browsing programs easier.

### 2-2. Labels

**Syntax:** <LABEL>:

**Description:** Labels are used to identify targets for jump instructions. When used as a jump target, the instruction following the label will be executed next.

**Examples:**

LOOP:	This label is on a line by itself.
L: MOV 8, ACC	This label is on a line with another instruction

### 2-3. NOP

**Syntax:** NOP

**Equivalent syntax:** ADD NIL

**Description:** NOP is a pseudo-instruction that has no effect on the node's internal state or communication ports. NOP is automatically converted to the instruction ADD NIL.

### 2-4. MOV

**Syntax:** MOV <SRC>, <DST>

**Description:** <SRC> is read and the resulting value is written to <DST>.

**Examples:**

MOV 8, ACC	The literal value 8 is written to the ACC register.
MOV LEFT, RIGHT	A value is read from the LEFT port, and then written to RIGHT.
MOV UP, NIL	A value is read from the UP port and then discarded.

## **2-5. SWP**

Syntax: SWP

Description: The values of ACC and BAK are exchanged.

## **2-6. SAV**

Syntax: SAV

Description: The value of ACC is written to BAK.

## **2-7. ADD**

Syntax: ADD <SRC>

Description: The value of <SRC> is added to the value of ACC and the result is stored to ACC.

Examples:

ADD 16            The literal value 16 is added to the value in the ACC register.  
ADD LEFT        A value is read from the LEFT port, and then added to ACC.

## **2-8. SUB**

Syntax: SUB <SRC>

Description: The value of <SRC> is subtracted from the value of ACC and the result is stored to ACC.

Examples:

SUB 16            The literal value 16 is subtracted from the value in the ACC register.  
SUB LEFT        A value is read from the LEFT port, and then subtracted from ACC.

## **2-9. NEG**

Syntax: NEG

Description: The value of ACC is arithmetically negated. A value of zero remains the same.

## **2-10. JMP**

Syntax: JMP <LABEL>

Description: Transfer execution unconditionally. The instruction after the label <LABEL> will be executed next.

## **2-11. JEZ**

Syntax: JEZ <LABEL>

Description: Transfer execution conditionally. The instruction after the label <LABEL> will be executed next if the value of ACC is zero.



## 2-12. JNZ

Syntax: JNZ <LABEL>

Description: Transfer execution conditionally. The instruction after the label <LABEL> will be executed next if the value of ACC is not zero.

## 2-13. JGZ

Syntax: JGZ <LABEL>

Description: Transfer execution conditionally. The instruction after the label <LABEL> will be executed next if the value of ACC is positive (greater than zero).

## 2-14. JLZ

Syntax: JLZ <LABEL>

Description: Transfer execution conditionally. The instruction after the label <LABEL> will be executed next if the value of ACC is negative (less than zero).

## 2-15. JRO

Syntax: JRO <SRC>

Description: Transfer execution unconditionally. The instruction at the offset specified by <SRC> relative to the current instruction will be executed next.

Examples:

JRO 0	This instruction will be executed next, effectively halting execution.
JRO -1	The previous instruction will be executed next.
JRO 2	The next instruction will be skipped, executing the instruction following it.
JRO ACC	The next instruction to execute will be determined by the value in ACC.

### 3. Example Programs

The following sample program reads a sequence of values from the LEFT port, doubling each value read and writing that to the RIGHT port. Because of the automatic looping behavior of the Basic Execution Node, it continues to the first instruction after executing the last instruction.

MOV LEFT, ACC	Read a value from the LEFT port into the ACC register.
ADD ACC	Add the value in ACC to itself, doubling it.
MOV ACC, RIGHT	Write the value in the ACC register to the RIGHT port.

The following sample program reads a sequence of values from the UP port, writing positive values to the RIGHT port and negative values to the LEFT port. Zero values are discarded.

START:

MOV UP, ACC	Read a value from the UP port into the ACC register.
JGZ POSITIVE	If the value in ACC is greater than zero, jump to "POSITIVE".
JLZ NEGATIVE	If the value in ACC is less than zero, jump to "NEGATIVE".
JMP START	The value was neither positive nor negative, so jump to "START".

POSITIVE:

MOV ACC, RIGHT	Write the value in the ACC register to the RIGHT port.
JMP START	Jump to "START".

NEGATIVE:

MOV ACC, LEFT	Write the value in the ACC register to the LEFT port.
JMP START	Jump to "START".

# NODE NODE TYPE T30 - STACK MEMORY NODE NODE

---

## 1. Architecture

The Stack Memory Node enables read/write access to a large number of values according to a simple stack-based communication protocol. (Refer to the model-specific manual to find the capacity of the Stack Memory Nodes on a particular device.)

## 2. Communication Protocol

All interaction with the Stack Memory Node is performed through ports. Writing to the Stack Memory Node adds the value to the top of the stack. If the stack is full, the write will block until space becomes available. Reading from the Stack Memory Node removes the top value from the stack and produces that value. If the stack is empty, the read will block until a value is available.

Stack Memory Nodes are typically connected to multiple other nodes, and can be used by any connected node. Simultaneous reads and writes to a Stack Memory Node resolve in an undefined order, but each individual communication will behave according to the described communication protocol. For more information on using storage nodes from multiple nodes effectively and predictably, refer to the separate document "Tessellated Intelligence System Best Practices - Patterns of Node Communication".

# NODE TYPE T31 - RANDOM ACCESS MEMORY NODE

**Note:** The Random Access Memory Node is not yet available in standard Tessellated Intelligence System devices. Emulators and prototype hardware are available to interested users. The specification and behavior is not yet finalized and therefore is omitted from this document.

- Control of the node
- Control of the node
- Control of the node
- Control of the node
- Control of the node
- Control of the node
- Control of the node
- Control of the node
- Control of the node
- Control of the node

## 2. Breakpoints

The breakpoints are placed in the code to allow the user to stop the execution of the program at a specific point. The program will be paused before the breakpoint is reached, allowing the user to inspect the state of the program through the debugger.

MOV TEST, ACC  
LAW ACC  
MOV ACC, TEST

- TODO
- FIGURE OUT WHO SOLD T3100 TO SWAP MEET DEALER
- ~~REVIEW SIGNAL MULTIPLEXER~~
- LOOK FOR BOOK OF MICRO-OPTIMIZATION TIPS
- RENEW LICENSE RATE TABS

# EMBEDDED INTERACTIVE DEBUGGER

---

## 1. Keyboard Shortcuts

The interactive debugger contains the following keyboard shortcuts:

**Control-Z:** Undo last change

**Control-Y:** Redo last change

**Control-X:** Cut selected text to clipboard

**Control-C:** Copy selected text to clipboard

**Control-V:** Paste clipboard text

**Control-Arrow:** Navigate to the adjacent execution node

## 2. Breakpoints

To set a breakpoint, place an exclamation mark (!) at the beginning of a line. When a breakpoint is set, the program will be paused before that line is executed, allowing you to easily debug code that would be too tedious to step through one instruction at a time.

```
MOV LEFT, ACC
```

```
!ADD ACC
```

```
MOV ACC, DOWN
```

The program will be paused before this instruction is executed.

# VISUALIZATION MODULE

---

## 1. Visualization Module Usage

The TIS-100 contains a visualization module that allows programs to programmatically create and display images. The module contents can be modified by sending command sequences, which consist of the starting X coordinate, the starting Y coordinate, one or more color values, and a terminating negative value (often -1). The coordinate system starts at (0, 0), which is located in the top-left of the display area.

The visualization module supports the following colors:

- 0: Black
- 1: Dark grey
- 2: Bright grey
- 3: White
- 4: Red

## 2. Visualization Module Resolution

The standard TIS-100 visualization module is 30 characters wide and 18 characters tall.

The "image console sandbox" contains a larger visualization module that is 36 characters wide and 22 characters tall.

## 2. Example Command Sequences

- 0,0,3,-1 Draw a single white pixel in the top-left corner of the module's display.
- 0,0,4,4,4,4,4,-1 Draw a horizontal red line in the top-left corner of the module's display.