前言

最近在Steam上购买了TIS-100编程游戏，该游戏是一款使用汇编语言解决问题的高智商游戏。并附有一份14页的说明文档 。我对该说明文档进行了完整翻译。 该文档英文原版pdf可以在这里下载：http://xuchen.wang/down/TIS-100ReferenceManual.pdf

综述

栅格智能系统（Tessellated Intelligence System，以下简称TIS）是由不规则互相连接且不均匀分布的节点组成的大规模并行计算机架构。TIS是为需求复杂数据流处理的应用而设计的。常见适用应用有：自动化金融交易、批量数据收集和大众行为分析。

> - 注：本手册用以这种形式出现的注释来指出需要特别留心的地方，并且提供更多关于主题的相关文档以备读者查询。

系统架构和组成

TIS由连接在同一总线上的大量独立节点构成。（在每种特定设备上的精确节点分布统计请参阅模型详情手册。）节点类型可大体分为处理节点和存储节点两大类，并在每大类中可精确细分。

通常，节点通过端口与最多4个相邻节点建立连接。端口允许节点间的轻量级信息传递。在端口上的通信通过协调处于连接上的两个节点中任一节点发起读命令或写命令，并在另一方发出回应前阻塞通讯实现。

> - 注：如果两个连接中的节点同时向对方发起了相同的通信指令（读或写），这两个节点会陷入死锁状态并引发一个硬件错误。请参考一遍独立的文档《栅格智能系统：节点通信模式》来获取如何高效安全的使用端口的细节。

> - 注：如果一个节点向另一个节点发起了一个通讯指令，却永远不会得到对应节点的应答，该节点会陷入死锁状态并并引发一个硬件错误。（但本规则也有例外，详情请参考具体节点类型的说明文档。）请参考一遍独立的文档《栅格智能系统最佳实践：节点通信模式》来获取如何高效安全的使用端口的细节。

> - 注：这篇文档没有详细阐明节点通信和相关指令的时序和吞吐量，它们会根据模型和硬件版本不同而变化。在特定设备表现的详细数据请参考具体模型的说明文档。

节点类型T20 - 预留类型

> - 这种节点类型标识符被用来限定TIS的特定模型，不在本文档叙述范围内。T20类型的节点文档仅会在支援这种节点类型的系统中进行描述。根据联邦法律，未尽授权的描述此种节点的说明文档拷贝将被反馈至国家安全局。

节点类型T21 - 基本执行节点

架构

基本执行节点负责整合TIS的行为特性。处理过程可以在基本执行节点内进行，也可以被传递至专用处理和存储节点中。 基本执行节点执行以基本执行节点指令集编写的程序。一个基本执行节点程序指明需要计算和通讯的操作。程序执行过程会从第一条指令开始逐条执行。**当执行完程序的最后一条指令后，会自动返回第一条指令继续执行。**这种行为特性使得基本执行节点在需要编写连续循环的程序时最为适用。 与TIS所有节点相同，基本执行节点拥有通讯端口。除此之外，基本执行节点包含了用于程序执行过程的一系列寄存器。基本执行节点没有额外的存储器。如果需要额外的存储单元，这个节点需要协调另一基本执行节点或者存储节点用以存储。 所有寄存器存储-999～999（包含边界）之间的整数。寄存器值的表示是基于定义实现的，编写基于基本执行节点的应用程序并不需要理解这种表示法。

ACC

- 类型：内部
- 描述：ACC是基本执行节点的基本存储寄存器。ACC被用作隐式源或许多指令的目标操作数，如算术指令和条件指令。

BAK

- 类型：内部（不可寻址）
- 描述：BAK是ACC中值的临时存储器。它仅可通过SAV或SWP访问，**不能直接读写**。

NIL

- 类型：内部（特殊）
- 描述：从NIL中读会产生0值。向NIL中写不会产生任何效果。如果你想舍弃一条指令的结果，可以使用NIL作为目标操作数。

LEFT,RIGHT,UP,DOWN

- 类型：端口
- 描述：四个通信寄存器UP,DOWN,LEFT和RIGHT协调了所有基本执行节点用来与其拓扑相邻节点通信的4个端口。硬件内一些端口会断开连接并当接收到读写指令时会无限期的阻塞。请参考节点互联图表去查看哪些端口可用。

ANY

- 类型：端口（伪端口）
- 描述：当一条指令从ANY端口发出时，指令会首先读取在任一端口上可用的第一个值。当一条指令发往ANY端口时，指令结果会被发送至从该节点任一端口试图读取的第一个节点。

LAST

- 类型：端口（伪端口）
- 描述：LAST表示上一个使用ANY伪端口读写的实际端口。它相对应的明确了一个端口。在成功从ANY端口读或向ANY端口写使得LAST端口被成功设置之前，从LAST端口读或向LAST端口写会导致一个未定义的行为。请参考一遍独立的文档《栅格智能系统最佳实践：节点通信模式》来获取示范LAST端口用法的样例代码。

指令集

<SRC>和<DST>指令参数可以具体指定为一个端口或内部寄存器。在一个端口连接到相关节点完成读写通信操作前，任何对此端口的使用会被阻断。另外，<SRC>参数可能是-999～999（包含边界）之间的字面值整数。

BAK不能作为一个<SRC>或<DST>操作数。BAK的值仅可以通过特殊指令SAV和SWP访问。

用于跳转指定目标的<LABEL>参数可以由任意字符构成。

注释

- 语法：# 注释文本
- 描述：在一行中，注释符(#)后出现所有文本都会被忽略。

> - 注：如果有在双注释符(##)后出现的文本，它将被当作程序的标题，并会显示在调试器中便于浏览程序。

标签

- 语法：<LABEL>:
- 描述：标签被用来辨识跳转指令的目标地。当一个标签被用来作为跳转目标时，标签后的指令将会接着被执行。
- 示例：

```
LOOP:           # 这个标签独自占据一行
L: MOV 8, ACC   # 这个标签和一条指令共用一行
```

NOP

- 语法：NOP
- 等价语法：ADD NIL
- 描述：NOP是一条伪指令。使用NOP对节点的内部状态和通讯端口都没有任何作用。NOP会被自动转换成指令ADD NIL。

MOV

- 语法：MOV <SRC>,<DST>
- 描述：<SRC>将会被读取，返回值会被写入<DST>。
- 示例：

```
MOV 8, ACC       # 常值8被写入ACC寄存器
MOV LEFT, RIGHT  # LEFT中的值被读出然后写入RIGHT
MOV UP, NIL      # UP中的值被读出然后被舍弃
```

SWP

- 语法：SWP
- 描述：交换ACC和BAK中的值

SAV

- 语法：SAV
- 描述：将ACC中值写入BAK

ADD

- 语法：ADD <SRC>
- 描述：<SRC>中的值与ACC相加，结果重写写入ACC
- 示例：

```
ADD 16          # 常值16与ACC相加，结果重写写入ACC
ADD LEFT        # LEFT中的值被读取并与ACC相加，结果重写写入ACC
```

SUB

- 语法：SUB <SRC>
- 描述：ACC减去<SRC>中的值，结果重写写入ACC
- 示例：

```
SUB 16          # ACC减去常值16，结果重写写入ACC
SUB LEFT        # ACC减去LEFT中的值，结果重写写入ACC
```

NEG

- 语法：NEG
- 描述：对ACC取算数相反数并存入ACC，0值保持不变。

JMP

- 语法：JMP <LABEL>
- 描述：无条件跳转指令。标签<LABEL>后的指令会紧接着被执行。

JEZ

- 语法：JEZ <LABEL>
- 描述：条件跳转指令。如果ACC为0，标签<LABEL>后的指令会紧接着被执行。

JNZ

- 语法：JNZ <LABEL>
- 描述：条件跳转指令。如果ACC不为0，标签<LABEL>后的指令会紧接着被执行。

JGZ

- 语法：JGZ <LABEL>
- 描述：条件跳转指令。如果ACC大于0，标签<LABEL>后的指令会紧接着被执行。

JLZ

- 语法：JLZ <LABEL>
- 描述：条件跳转指令。如果ACC小于0，标签<LABEL>后的指令会紧接着被执行。

JRO

- 语法：JRO <SRC>
- 描述：无条件跳转指令。会跳转到<SRC>指定的偏移量处执行。

- 示例：

```
JRO 0           # 重复执行本行语句，可以用来使得程序中断执行
JRO -1          # 跳转到上一行语句执行
JRO 2           # 下一行语句将被跳过，直接执行下下行语句。
JRO ACC         # 跳转到偏移量为ACC的行执行。
```

示例程序

以下示例程序从LEFT端口读取数据，将其翻倍后写入RIGHT端口。由于基本执行节点的自动循环特性，该程序会完成最后的操作跳转至第一条指令继续执行。

```
MOV LEFT, ACC     # 读取LEFT端口中的数据保存至ACC寄存器
ADD ACC           # 将ACC值加至自身
MOV ACC, RIGHT    # 将ACC值写入RIGHT端口
```

以下示例程序从UP端口读取一个值序列，将其中的正值写入RIGHT端口，负值写入LEFT端口，0值丢弃。

```
START:
    MOV UP, ACC     # 读取UP端口中的数据保存至ACC寄存器
    JGZ POSITIVE    # 如果ACC值>0，跳转至POSITIVE处继续执行
    JLZ NEGATIVE    # 如果ACC值<0，跳转至NEGATIVE处继续执行
    JMP START       # 此时如果还没跳转，说明ACC值为0，直接跳转至START读取下一个数据
POSITIVE:
    MOV ACC, RIGHT  # 将ACC值写入RIGHT端口
    JMP START       # 跳转至START读取下一个数据
NEGATIVE:
    MOV ACC, LEFT   # 将ACC值写入LEFT端口
    JMP START       # 跳转至START读取下一个数据
```

节点类型T30 - 栈存储节点

架构

栈存储节点基于一个简单的栈通信协议，支持大数的读写操作。（请参考具体模型手册来获取特定设备上的栈存储节点的容量。）

通信协议

栈存储节点上的所有交互通过端口完成。向栈存储节点写入数据会将数据保存至栈顶。如果一个栈已满，写入操作会被阻断直到空间可用。从栈存储节点读取数据会移除栈顶数据并将其作为返回值返回。如果一个栈为空，读取操作会被阻断直到一个值可用。

栈存储节点通常会连接很多其他节点，并可被其他节点所使用。同时对同一节点进行读和写操作会以不确定的顺序完成，但任一通讯都会按照描述的通信协议进行。要获取更多关于高效并可预测地使用多个存储节点，请参考一遍独立的文档《栅格智能系统最佳实践：节点通信模式》。

节点类型T31 - 随机存取存储器节点

> - 注：随机存取存储器节点在标准TIS设备中尚不可用。感兴趣的用户可以使用仿真器和硬件原型。具体特性和行为表现尚未定型，所以本文档并不讨论。

内置交互调试器

键盘快捷键说明

内置交互调试器提供了一下键盘快捷键： Ctrl-Z：撤销 Ctrl-Y：恢复 Ctrl-X：剪切 Ctrl-C：复制 Ctrl-V：粘贴 Ctrl-方向键：定位至相邻执行节点 F1：查看指令快捷帮助 F2：查看防篡改证书状态 F5：运行当前程序 F6：单步执行或暂停当前程序

断点

要设置一个断点，只需在行首放置一个感叹号(!)。如果在某行设置了一个断点，程序会在执行这行之前暂停，这是你可以免于逐行单步调试而轻松的调试代码。

```
MOV LEFT, ACC
!ADD ACC           # 程序会在执行此行前暂停
MOV ACC, DOWN
```

可视化模块

可视化模块用途

TIS-100包含了一个可视化模块，使得程序以编程方式生成并显示图像。这个模块内容可以通过发送命令序列修改，包含了一个起始的X坐标、起始的Y坐标、一个或多个色彩值和一个表示结束的负值（通常为-1）。坐标系起始于(0,0)，显示在可视区域的左上方。 可视化模块支持以下颜色： 0：黑色 1：深灰色 2：浅灰色 3：白色 4：红色

可视化模块分辨率

标准TIS-100可视化模块为宽30字符，高18字符。 “图像控制台沙盒”包含了一个更大的可视化模块，宽36字符，高22字符。

指令序列实例

```
0,0,3,-1          # 在模块显示区左上角画一个白色像素点
0,0,4,4,4,4,4,-1  # 在模块显示区左上角画一条红色水平线
```