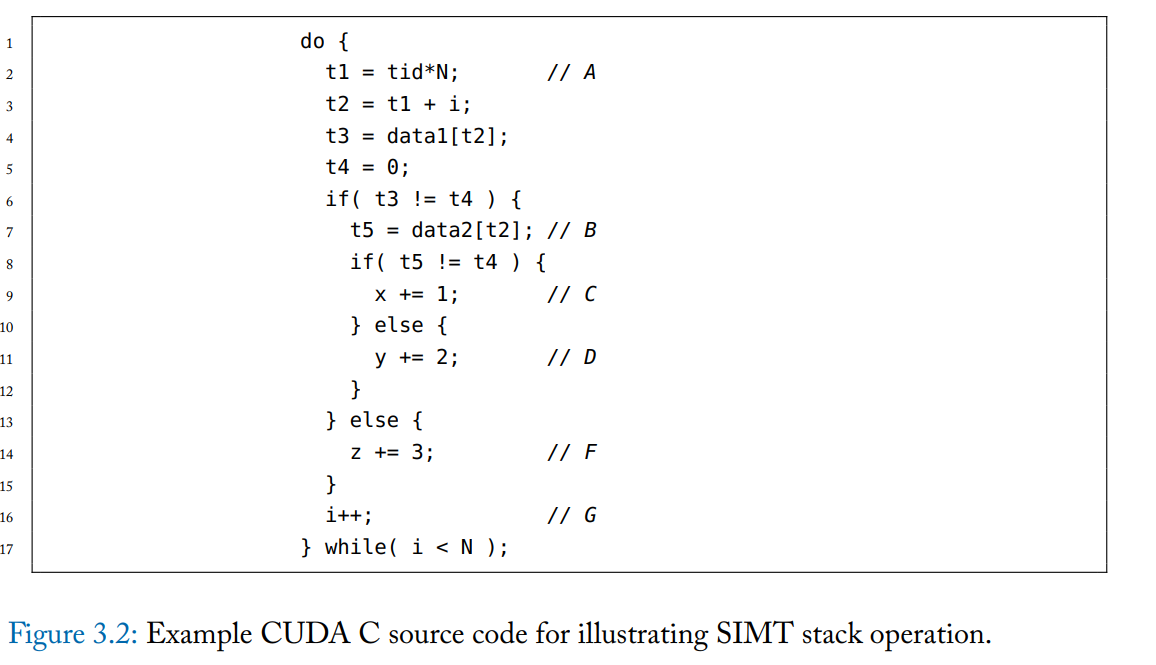
GPGPU的一大特点在于计算核心的SIMT执行模式，这一模式意味着，尽管在单个SIMT核内部，有且仅有一个活跃PC，但由于每个线程计算的数据不同，实际上每个线程都是完全独立的进程，在当前主流GPGPU架构中，线程间分歧的管理是通过栈结构对掩码（Mask）进行管理，这个用来管理掩码的栈称为SIMT Stack.

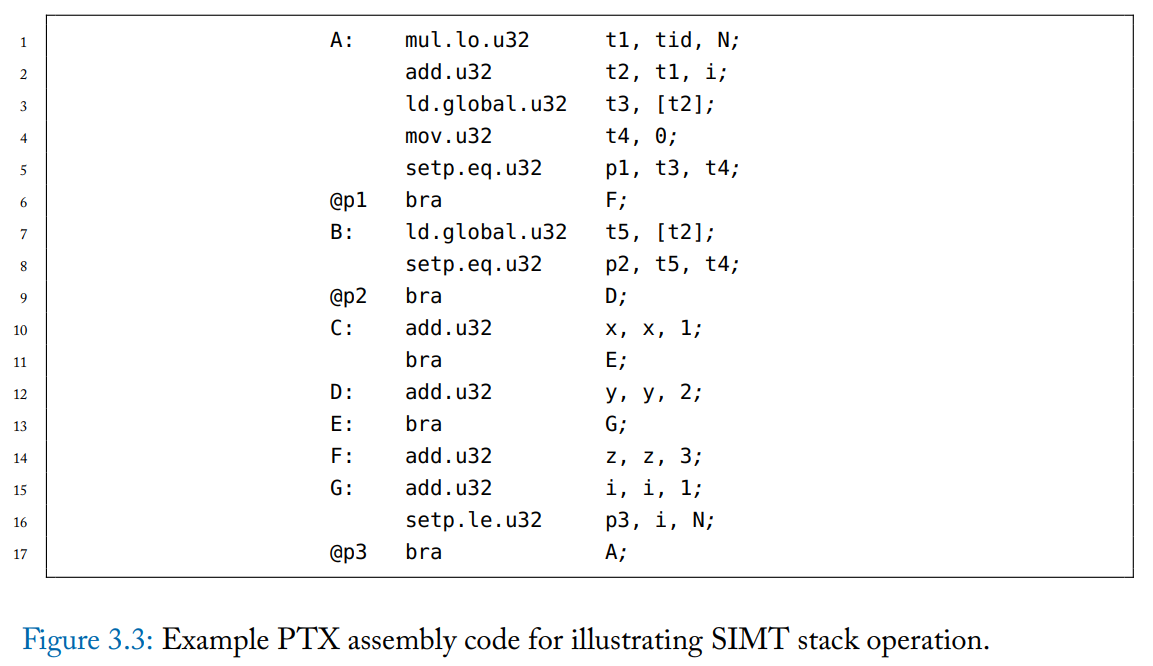
为了确保程序运行时各个线程之间的独立性，同时最大限度维护SIMD的执行效率，SIMT stack的主要功能如下：维护分支嵌套控制流，保障程序运行的正确性；在实际没有分支分歧发生时，跳过不必要程序段的执行。

**1 SIMT stack功能概述**

SIMT Stack的原型架构是依靠硬件结构实现的线程分歧管理器，它不依赖于专用的分支处理指令，因此可以作为典型架构介绍SIMT Stack的原型结构和线程分歧管理流程。

对于如下所示的cuda程序块，依据线程间分支发生和结束的位置，可以将程序划分为7块，其中A-G字母代表的是对应程序起始位置的PC值。B、F程序块为A程序块末尾处if-else分支语句（称为分支1）对应的不同分支，G程序块为分支1的结束点，也即执行B、执行F的不同分支均需要执行G程序块；类似地，C、D程序块为B程序块中间位置if-else分支语句（称为分支2）对应的不同分支，E程序块为分支2的结束点，也即执行C、执行D的不同分支均需要执行E程序块。





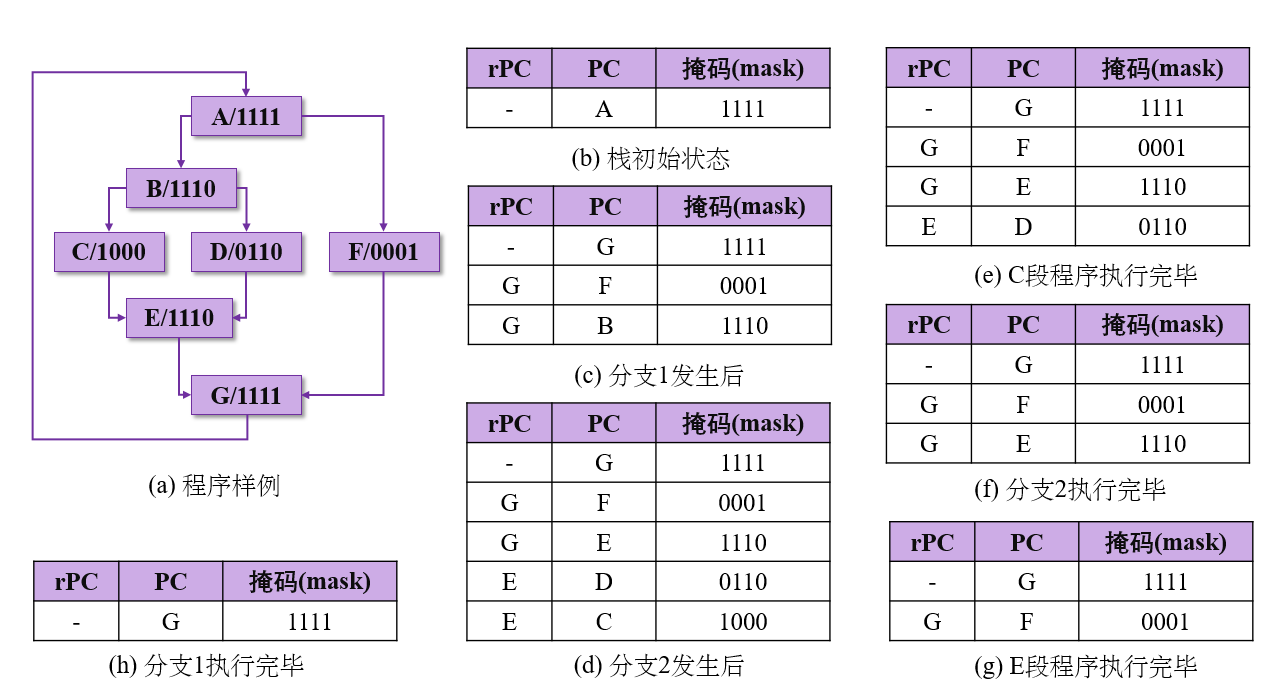
原型SIMT stack中每一行需要管理对应某一程序块的以下信息：

下一条指令地址（PC）；

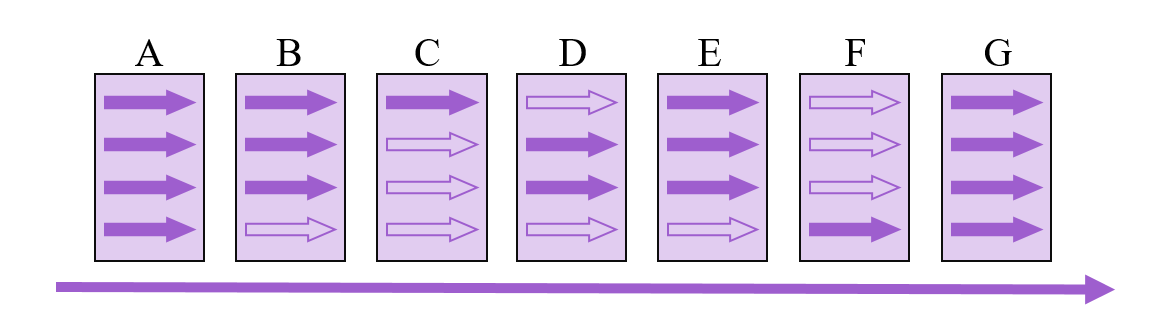
分支结束位置汇合点指令地址（rPC）；

需要执行该程序块的线程掩码（mask）。

以一个四线程warp为例，下图为执行到程序关键位置时，SIMT stack中存储的内容，其中，栈表的最后一行为对应时刻的栈顶(top of stack, TOS)信息。



初始状态时，栈内掩码为全1，代表所有线程均为活跃状态；当程序执行到A程序段末尾时，遇到分支1，出现B、F对应的两个分支，此时，A对应的栈项PC值改为分支1的汇合点PC值（即G），同时，按照特定顺序将B、F对应的PC、rPC、掩码压入栈内，上图实例中，此时TOS为B程序段对应信息，也即采取分支时先执行if再执行else的分支策略。当执行到B段程序末尾时，遇到分支2，出现C、D对应的两个分支，与上述过程类似地，B对应的栈项PC值改为分支2的汇合点PC值（即E），同时将C、D对应的信息压入栈内，此时程序将带着掩码1000继续执行C段程序。当SIMT stack对比得知当前执行的PC值与TOS的rPC相同时，判断C段程序已经执行完毕、到达汇合点，弹出TOS，跳转至D程序段，带掩码0110执行D程序段；当C、D对应的两个分支均执行完毕后，此时TOS为E段程序对应信息，将以掩码1110继续执行E段程序。程序实际执行过程如下图所示，其中实箭头表示该线程活跃，虚箭头表示该线程在对应周期并不执行指令。



总结可知SIMT stack的原型结构有以下特点：

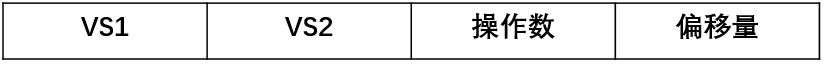
1. 遇到分支时，更改TOS的PC项为分支结束时的PC值，并且依照次序将对应if、else的程序段相关信息压入栈内，也即在分支发生时，SIMT stack需要获得掩码信息、if-else程序段的起始PC值信息、分支结束点的PC值信息；
2. 程序总是调度SIMT stack的TOS对应程序段执行，也即发生分支时，程序执行顺序有一定确定性，在上例中为A-B-C-D-E-F-G；
3. 判断分支程序段是否到达汇合点的依据为当前正在执行的PC值是否与TOS的rPC项相等。

由于指令空间的限制，实际操作中，很难支持上述原型SIMT stack的操作要求，所以在以NVIDIA架构为代表的GPGPU架构中，一般采取自定义指令的方式实现对SIMT stack的管理。

**2 自定义扩展指令集**



与分支管理相关的自定义扩展指令集有上表所示7条，其中1-6条为分支指令。以vbeq指令为例，需要完成的功能为：取源操作数vs2与vs1，valu模块对这两个向量寄存器中的元素一一进行比较，对于第i个元素，若vs2[i]=vs1[i]，则计算结果out[i]为1，最终valu的输出结果out为分支指令对应的else路径掩码，同时译码模块将向分支管理模块发送分支发生标记以及else路径PC起始值PC branch。



分支指令包含的信息项

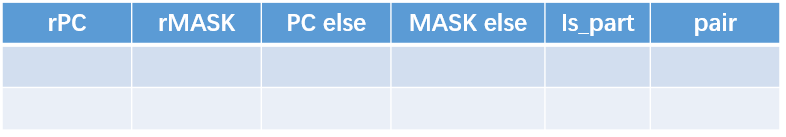
上表中指令7为汇合指令，需要完成的主要功能为：标记分支语段执行完毕，弹出SIMT stack栈项，并根据弹出信息判断PC next的取值：PC+4、PC branch、PC join。



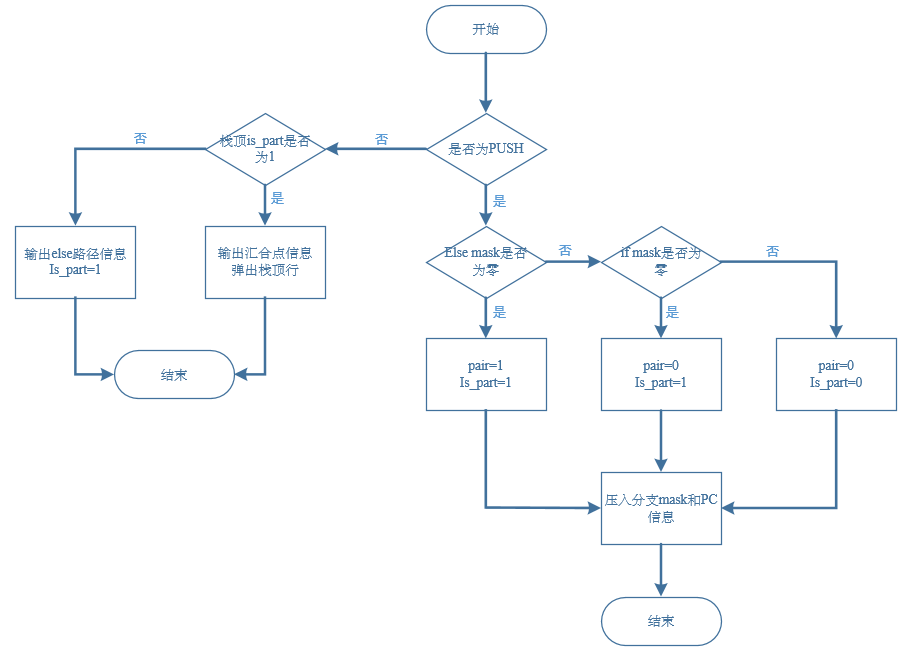
分支同步指令包含的信息项

**3 SIMT stack硬件结构**

每一个warp都单独维护一个信息管理栈，该信息管理栈的栈项如下图所示：



栈中每一行管理一次分支对应的信息，对于压栈、出栈和内部寄存器的管理流程图如下：



在上述实现中，采取了双数据存储结构（即在同一行内，同时存储else路径信息和汇合点信息），因此启用is\_part项用来标记和选择最终的输出有效信息信息，is\_part为0时，输出为栈顶else路径信息，is\_part为1时，输出为栈顶汇合点信息。

在确保程序运行正确性之外，SIMT stack的另一大功能是识别分支是否发生，并且跳过不必要代码段的执行，也即在分支时只执行if代码段或只执行else代码段时，跳过当前分支对应的掩码为全0的段落。

在SIMT stack的设计中，通过指令集扩展增加功能和硬件支持的方式实现该功能。

指令集扩展增加的功能为：

分支同步指令在标记分支代码结束之外，增加了代码空间用来指向分支同步位置PC地址，从而在分支else路径掩码为0的情况下可以跳过else部分代码的执行，if路径执行完毕后直接跳转到分支同步位置执行。

硬件支持的设计如下：

为支持跳过if路径代码的功能，若输入的if路径掩码为0，则将输出线程掩码设置为else路径掩码，同时发送从issue模块得到的PC branch信息，并示意取值模块发生PC地址跳转，需冲刷流水线。

为支持跳过else路径代码的功能，对栈中每一行增加pair项，用于记录对应分支的else路径掩码是否为0，在SIMT stack给出分支线程执行所需要的掩码和取指令控制信号时，根据弹出栈项的is\_part项和pair项共同决定掩码设置和取值控制信号设置。

若当前is\_part项为0，则表示跳转目标为else路径程序起始位置；若当前is\_part项为1，则表示执行的目标为同步位置程序段，若此时栈顶pair为0表示当前分支为实际出现分歧的分支，当前状态为if、else路径对应程序均执行完毕，输出为线程同步位置的对应掩码，无需流水线冲刷，若此时栈顶pair项为1表示当前分支并未实际出现分歧，当前状态为if路径程序执行完毕，需跳转至同步位置继续执行程序，输出为线程同步位置掩码，由于出现PC地址跳转故需要流水线冲刷。

下图演示的为与压栈、出栈过程中与分支判断有关的操作。

