# 高等计算机体系结构

## 实验一：分支预测

## 一. 实验目的：

- 设计一个基于 PC 的简单分支预测器（bimodal branch predictor）。
- 设计基于 PC 和全局分支寄存器（global branch history register，GBHR）的共享分支预测器（gshare branch predictor）。
- 利用所设计的分支预测器对 SPEC benchmark 地址流进行仿真，分析分支预测性能

## 二. 实验环境和工具：

- 系统要求：Ubuntu XX.XX（32bit，x86）
- 编程工具：gcc4.4
- 编程语言：C/C++/JAVA

## 三. 实验内容：

- 分别设计完成可模拟简单分支预测器和共享分支预测器的仿真器。
- 该仿真器使用具有标准格式的 PC 地址流文件作为输入（该地址流已由 SPEC 标准测试程序产生），并将最终分支历史表（branch history table，BHT）中所存储内容，即 2-bit 饱和计数器的计数值和分支预测分析结果以标准格式输出到结果文件中。

### 1. 简单分支预测器（bimodal branch predictor）

简单分支预测器的结构如图 1 所示，该预测器结构仅仅以分支指令的 PC 地址作为访问分支历史表 BHT 的索引。其中，该索引并不是采用分支指令的全部 PC 地址，而是仅使用 PC 地址的低 *m* 位作为索引。注意，因为对于 32-bit 字节编址处理器，PC 地址均是 4 字节对齐，因此 PC 地址的最末两位一定是“00”，故最终采用 PC 地址中的“*2 ~ m+1*”位作为访问 BHT 的索引。

**BHT 中的每一项：**

BHT 中的每一项都是一个 2-bit 饱和计数器。BHT 中的所有计数器在仿真开始时，应当被初始化为“2”，即“若跳转”（“weakly taken”。）

**关于分支冲突：**

由于仅仅使用一部分 PC 地址去索引 BHT，因此不同的分支可能会访问 BHT 中相同的项，称为“冲突”。在分支预测器中发生冲突不会产生问题，因为预测错误还会被纠正，所以并不会被检测或避免冲突（It just happen！）。

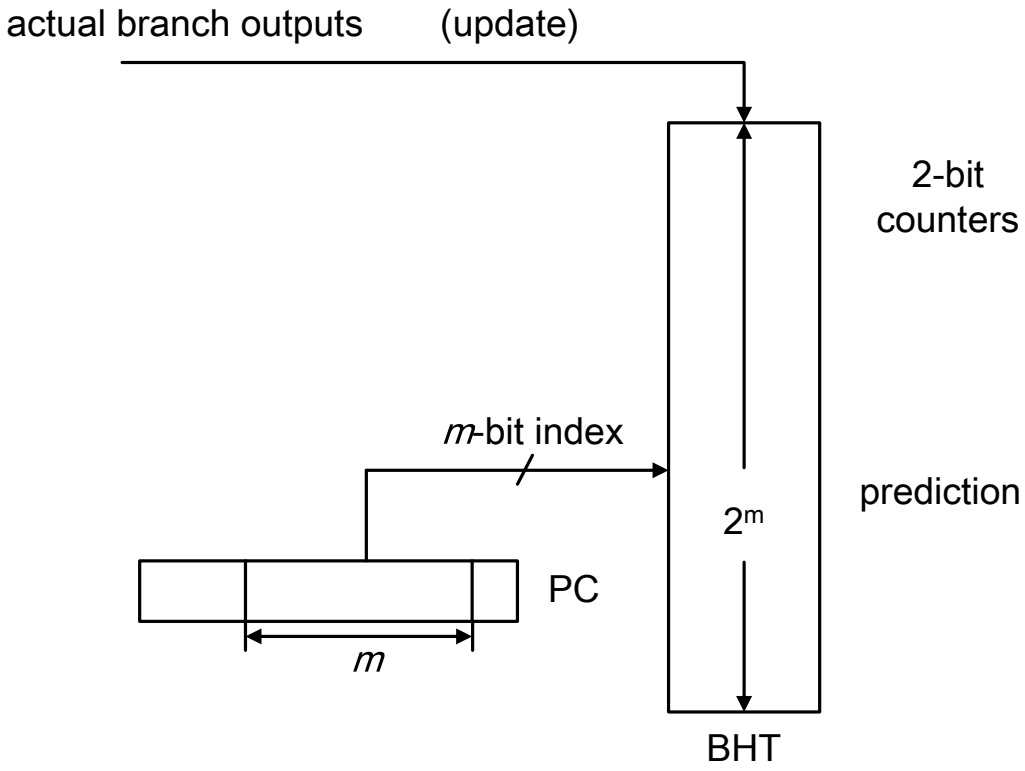


图 1 简单分支预测器（bimodal branch predictor）

**操作流程：**

当从 trace 文件中获得分支的 PC 地址后，需要完成以下 3 步完成预测：

(1). 根据“*m*”取值从 PC 地址中获得访问 BHT 的索引。

(2). 对分支方向进行判定。利用索引访问 BHT 中对应的 2-bit 计数器。如果计数器的计数值不小于 2，则预测分支发生跳转；否则预测分支不发生跳转，程序顺序执行。

(3). 基于分支的实际输出更新分支预测器。当分支发生跳转时，BHT 中所对应的计数器递增“1”；若不发生跳转，BHT 中所对应的计数器则减“1”。注意，计数器将在“0”和“3”时进入饱和状态，即当计数值为“3”时，计数值不能再增加，而当计数值为“0”时，计数值不能再减少。

## 2. 共享分支预测器（gshare branch predictor）

相比 bimodal 分支预测器，共享分支预测器设置了一个 *n* 位全局分支历史寄

存器（global branch history register，GBHR）。访问 BHT 的索引由分支指令的 PC 地址和 GBHR 共同决定，其结构如图 2 所示。全局分支历史寄存其在仿真开始时被初始化为全 0（00...0）。

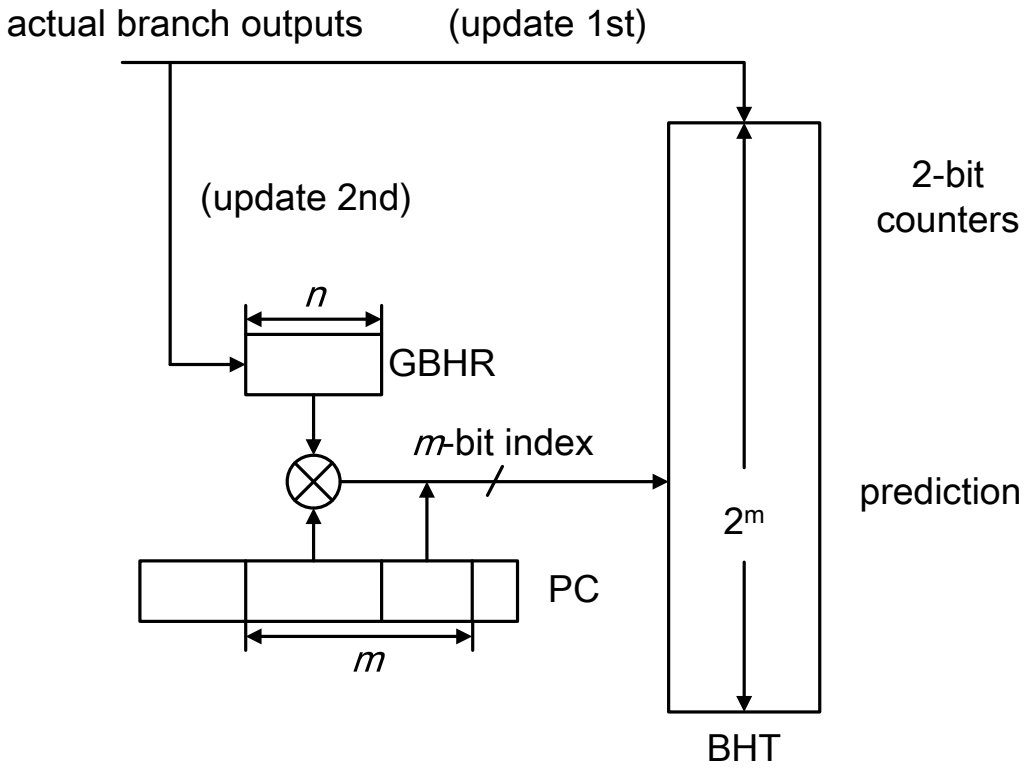


图 2 共享分支预测器（gshare branch predictor）

**操作流程：**

当从 trace 文件中获得分支的 PC 地址后，需要完成以下 4 步完成预测：

(1). 根据分支的 PC 地址和 GBHR 中的内容共同决定访问 BHT 的索引。将分支 PC 地址中作为索引的 *m* 位中的高 *n* 位与 *n* 位 GBHR 中的内容进行 XOR 异或运算，将最终生成的 *m* 位索引送入 BHT 中查找对应的 2-bit 饱和计数器。

(2). 对分支方向进行判定。利用索引访问 BHT 中对应的 2-bit 计数器。如果计数器的计数值不小于 2，则预测分支发生跳转；否则预测分支不发生跳转，程序顺序执行。

(3). 基于分支的实际输出更新分支预测器。当分支发生跳转时，BHT 中所对应的计数器递增“1”；若不发生跳转，BHT 中所对应的计数器则减“1”。注意，计数器将在“0”和“3”时进入饱和状态，即当计数值为“3”时，计数值不能再增加，而当计数值为“0”时，计数值不能再减少。

(4). 基于分支的实际输出更新全局分支历史寄存器 GBHR。首先，将 GBHR 中的内容右移 1 位，然后将分支的实际输出存放在 GBHR 的最高有效位。

### 3. 仿真器的输入

仿真器所读入的 trace 文件的格式如下所示：

<hex branch PC> t|n

<hex branch PC> t|n

…

<hex branch PC>是分支指令的地址，用于索引 BHT。

“t”表示分支实际发生了跳转（注意：不是预测的跳转！）。“n”表示分支实际并没有发生跳转。

例如：

00a3b5fc t

00a3b604 t

00a3b60c n

…

### 4. 仿真器输出的性能指标

仿真器最终输出如下性能指标：

a. 分支的数目

b. 分支预测错误的数目

c. 分支预测错误率（分支预测错误的数目/分支的数目）

### 5. 仿真结果的验证

利用所提供的验证文件与仿真器的输出文件进行自动化（非手工）比对，对所设计仿真器功能的正确性进行验证。所需验证的内容包括：

- 分支预测器的配置
- BHT 中最终的存储内容（计数值）
- 各种性能指标

仿真器的输入结果必须在内容和格式上都与验证文件一致。授课教师将通过“diff”命令评判仿真器功能的正确性。因此，在提交仿真器之前，学生需要通过如下两个步骤来对仿真器的功能进行验证。

- 首先，将仿真器的输出结果重定向到一个临时文件，可通过**“>”**操作符实现。
- 然后，通过运行**“diff”**命令，将保存输出结果的临时文件与验证文件进行比较，若输出“nothing”则表示输出结果正确。例如：

  $ diff –iw my_output validation_run

其中，-iw 表示 diff 命令将忽略大小写和空格。

## 6. 仿真器编译与运行的要求

学生需要将仿真器的源代码提交到课程云盘，授课老师对仿真器进行编译和运行，并给出成绩。因此，所提交的源代码必须满足**如下的严格要求**，否则仿真器可能无法正常运行，导致评分时被扣除相应的分数（评分标准参见第五部分）。

- 仿真器的编译和运行环境必须是 Ubuntu XX.XX（32bit，x86），以保证授课教师能够正确编译和运行所提交的仿真器。
- 除了提交源代码文件，还必须提交 **Makefile** 文件以完成仿真器的自动编译。通过此 Makefile 文件编译生成的仿真器**可执行文件名为“sim_bimodal”**和**“sim_gshare”**，同时 Makefile 文件还需要提供“make clean”命令，以便于删除目标（.o 文件）文件和可执行文件。由于学生缺乏 Linux 下的编程经验，为了减轻设计难度，本实验工程可复用实验一中的 makefile 文件，并通过适当修改以满足需求。
- 仿真器运行时，能够在终端命令行：

Bimodal predictor：./sim bimodal <M2> <tracefile>。其中 M2 参数用于设置分支 PC 地址中用于索引的位宽。

Gshare predictor: ./sim gshare <M1> <N> <tracefile>。其中，M2 参数用于设置分支 PC 地址中用于索引的位宽；N 用于设置 GBHR 的位宽。

## 7. 实验工程的提交：

学生需要将设计完成的仿真器提交到课程云盘之上，以 ZIP 压缩包的形式提交，文件名为（学号_姓名_ project2.zip）。该压缩包中应该包含如下文件：

- 源代码
- Makefile 文件

在 Linux 下创建 ZIP 压缩包的命令如下，假设源文件为.c 和.h 文件：

$ zip 学号_姓名_ project 1 *.c *.h Makefile

## 8. 评分标准：

评分要求：

- 所有性能指标正确（参见第 4 节）
- BHT 中最终的存储内容正确

满分 100 分，分值分布如下：

| | | |
|---|---|---|
| 仿真器能够编译通过并运行 | | 20 |
| *bimodal* predictor | val_bimodal_1.txt | 10 |
| | val_bimodal_2.txt | 10 |
| | val_bimodal_3.txt | 10 |
| | val_bimodal_4.txt | 10 |
| *gshare* predictor | val_gshare_1.txt | 10 |
| | val_gshare_2.txt | 10 |
| | val_gshare_3.txt | 10 |
| | val_gshare_4.txt | 10 |
| Total | | 100 |

分数扣除：

- **根据云盘时间戳，每晚提交 1 小时扣除“1”分。**
- **仿真器无法编译通过将判为“0”分。**

- 上表中某相应项仿真发生错误或无法完成，扣除相应分数。
- 使用比对工具对源代码检测，若发现抄袭现象，双方都计为“0”分。