NSCSCC2021 初赛提交说明文档

西北工业大学 3 队

2021 年 8 月

目录

一、设计简介

实现了 57 条指令、6 个 CP0 寄存器、8 种例外 ,采用 AXI3 接口与外界交互。ICache 和 DCache 及 2 路组相联 8KB, Dcache 采用写回，按写分配的设计。

二、流水线设计

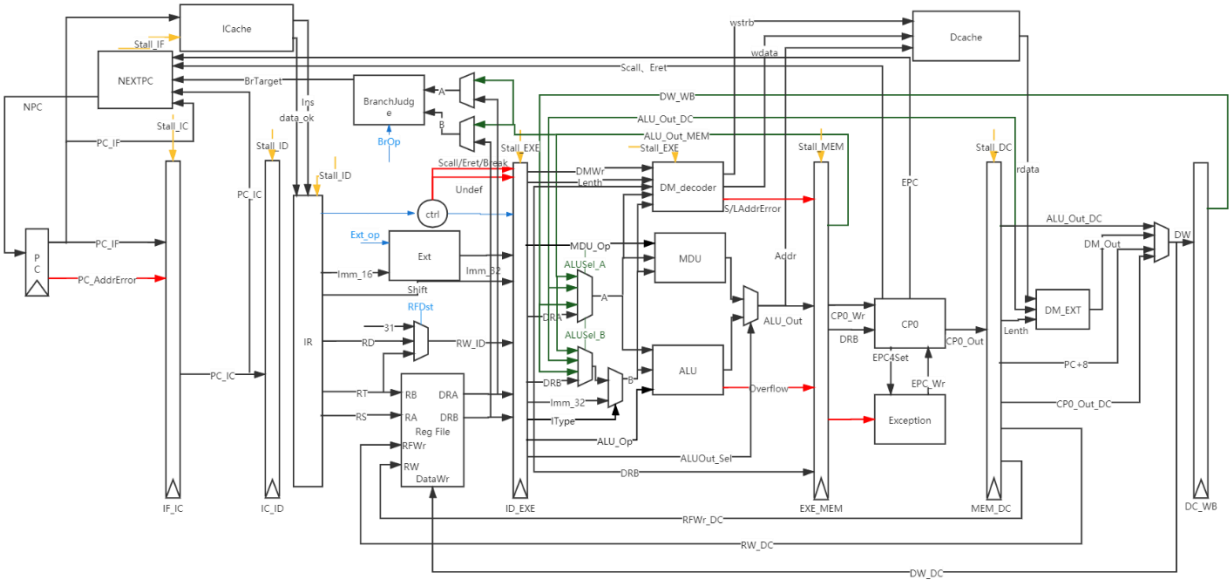

图 2.1

流水线分为 7 级，分别为 IF、ID、EXE、EXP、MEM1、MEM2、WB，另外还有一个伪流水级 pre_IF。各流水级功能如下：

0. pre_IF

使用 NPC 作为指令地址向 ICache 发起请求。

1. IF

取得 Icache 返回的指令。

2. ID

译码并访问寄存器，取出操作数，并根据 PC 进行分支预测。根据译码结果生成旁路信号。

3. EXE

获得 ALU 和 MDU 运算的两个操作数，进行 ALU 运算（除乘除法以外的运算均在 ALU 中处理）或 MDU 运算，根据译码产生的控制信号选择 ALU 运算结果与 MDU 运算结果。

4. EXP

①Dcache 接收 CPU 的请求，以及读地址（发起读请求时）或者写地址和写数据（发起写请求时）。

②判断是否发生异常或中断。

③向 CP0 寄存器发起读写请求。

5. MEM1

① Dcache 从 ram 中获取数和 tag。

② Dcache 判断是否需要经过旁路将 MEM2 级的数据传至 MEM1 级。

6. MEM2

① 根据 tag 判断是否发生 cache miss 以及是否需要阻塞。

② 根据加载类型将 Dcache 数据扩展与移位。

③ 选择写回寄存器的数据。

7. WB

写回级的前半个周期将数据写回。

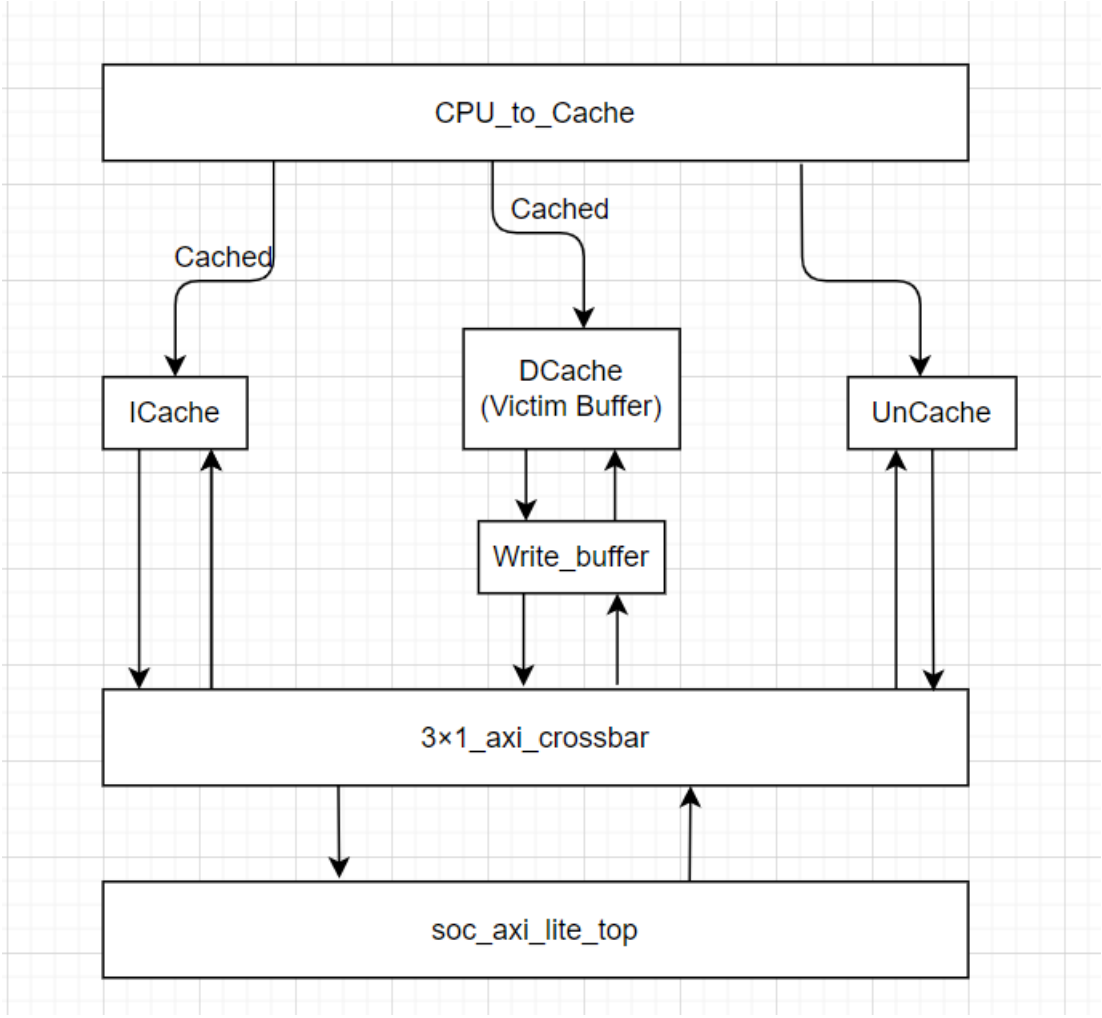

图 2.2 总体工程结构

三、 变量及模块

1.PC（程序计数器）

端口说明：

信号名	方向	描述
clk	I	时钟信号
rst	I	复位信号，将 PC 置为 32’h0x00003000 1：复位 0：无效
NPC[31:0]	I	下一条指令地址
PC[31:0]	O	当前指令地址
PCAddressError	O	是否发生 PC 地址未对齐异常
put_req	O	发起读指令请求

功能定义：

序号	功能名称	功能描述
1	复位	当复位信号有效时，PC 被设置为 32’h0xbfc0_0000
2	更新	当 PCWr 有效且时钟上升沿到来时，PC 更新为 NPC 的输入值
3	异常	传出 PC 地址异常

2. RF（寄存器堆）

端口说明：

信号名	方向	描述
clk	I	时钟信号
RFWr	I	写使能信号 1：可向 RF 中写入数据 0：不能向 RF 中写入数据
RA[4:0]	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD1
RB[4:0]	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD2
RW[4:0]	I	5 位地址输入信号，指定 32 个寄存器中的一个，作为写入的目标寄存器
DW[31:0]	I	32 位数据输入信号
RDA[31:0]	O	输出 RA 指定的寄存器中的 32 位数据
RDB[31:0]	O	输出 RB 指定的寄存器中的 32 位数据

功能定义：

序号	功能名称	功能描述
1	读数据	读出 RA,RB 地址对应寄存器中所存储的数据到 RDA,RDB

2	写数据	当 RFWr 有效且时钟上升沿来临时，将 RFWr 写入 A3 所对应的寄存器中

3. EXT（位扩展单元）

端口说明：

信号名	方向	描述
Imm16[15:0]	I	16 位输入数据
ExtOp[1:0]	I	位扩展方式选择信号 0：符号扩展 1：零扩展 2：加载至高位
Imm32[31:0]	O	扩展后的 32 位输出数据

功能定义：

序号	功能名称	功能描述
1	符号扩展	将 16 位输入数据进行符号扩展，输出 32 位数据
2	零扩展	将 16 位输入数据进行零扩展，输出 32 位数据
3	加载至高位	将 16 位输入数据加载至高 16 位，并将低 16 位置 0

4. BranchJudge（分支判断单元）

端口说明：

信号名	方向	描述
Operand_A[31:0]	I	跳转指令第一个操作数，来自寄存器或旁路
Operand_B[31:0]	I	跳转指令第二个操作数，来自寄存器或旁路
BrOp[2:0]	I	Branch 比较类型： 000: blt 001: bge 100: beq 101: bne 110: ble 111: bgt 011: 不是 branch 指令
branch	O	是否发生分支跳转

功能定义：

序号	功能名称	功能描述
1	判断分支	计算是否发生分支跳转

6. NEXTPC（指令地址计算单元）

端口说明：

信号名	方向	描述
Stall	I	当前是否阻塞 1：阻塞 0：非阻塞
PC_ID[31:0]	I	当前 ID 阶段地址
mis_predict	I	分支预测判断是否正确
EPC[31:0]	I	当指令为 Rret 时，返回指令的 PC
Jump	I	是否为 J、JR、JAL、JALR 指令
JR	I	是否为 JR、JALR 指令
Operand_A[31:0]	I	JR 指令寄存器中的 PC 值
Exc[1:0]	I	是否发生异常，触发异常处理程序
NPC[31:0]	O	计算出的下一条指令的地址

功能定义：

序号	功能名称	功能描述
1	分支跳转	计算分支地址 branch_target
2	J 跳转	通过位拼接操作计算 J 跳转地址
3	JR 跳转	选择寄存器读出的地址
4	阻塞	Stall 生效时 NPC=PC，不改变
5	异常处理	Exc 生效时跳转到异常处理子程序地址
6	中断返回	Eret 生效时回到 EPC 地址

7. ALU（算术逻辑单元）

端口说明：

信号名	方向	描述
Operand_A[31:0]	I	操作数 1
Operand_B[31:0]	I	操作数 2
Shift[4:0]	I	移位数
ALUOp [3:0]	I	ALU 功能选择信号 0000:add 0001:addu 0010:sub 0011:subu 0100:and 0101:or 0110:xor 0111:nor 1010:slt 1011:sltu 1000:sll 1001:srl 1100:sra

		1101:sllv 1110:srlv 1111:srav
ALUOut[31:0]	O	ALU 的计算结果
Overflow	O	是否溢出

功能定义：

序号	功能名称	功能描述
1	加运算	Out = A + B
2	减运算	Out = A - B
3	与运算	Out = A & B
4	或运算	Out = A | B
5	异或运算	Out = A ^ B
6	或非运算	Out = ~(A | B)
7	逻辑左移	Out = B << A[4:0]
8	逻辑右移	Out = B >> A[4:0]
9	算术右移	Out = $signed(B) >>> A[4:0]
10	小于置 1（有符号）	Out = ($signed(A) < $signed(B)) ? 1 : 0
11	小于置 1（无符号）	Out = (A < B) ? 1 : 0
12	异常	带符号加减运算时，传出溢出异常

8.MDU（乘除单元）

端口说明：

信号名	方向	描述
clk	I	时钟信号
Flush	I	清除信号
MDU_en	I	使能信号
MDUOp[1:0]	I	MDU 操作码： 00:mult 01:multu 10:div 11:divu
Waiting	I	是否有乘除法等待握手
operand_x[31:0]	I	操作数 1
operand_y[31:0]	I	操作数 2

MFT_Sel[1:0]	I	寄存器操作码 00:MFHI 10:MFLO 01:MTHI 11:MTLO
finish	O	乘除法完成
HandShake	O	信号与数据握手成功
MDUOut[31:0]	O	HI\LO 寄存器输出
Flush_busy	O	清除 MDU 的 busy 信号

功能定义：

序号	功能名称	功能描述
1	清除	当清除信号有效时，发出信息清除 busy 状态
2	有符号乘	{HI, LO} = $signed(A) * $signed(B)
3	无符号乘	{HI, LO} = A * B
4	有符号除	LO = $signed(A) / $signed(B) HI = $signed(A) % $signed(B)
5	无符号除	LO = A / B HI = A % B
6	写 LO 寄存器	LO = A
7	写 HI 寄存器	HI = A
8	读 LO 寄存器	Out = LO
9	读 HI 寄存器	Out = HI

9.DM_decoder（Dcache 写信号生成单元）

功能定义：

序号	功能名称	功能描述
1	生成四位写使能信号	通过地址后两位与存储指令类型，生成对应的四位写使能
2	数据处理	通过地址后两位与存储指令类型，将待写入的数据移位

10.DM_EXT（data 扩展单元）

功能定义：

序号	功能名称	功能描述
1	将所需内容进行扩展	通过地址后两位与加载指令类型，将对应内容扩展

11.Forward_for_ALU（ALU 旁路选择）

功能定义：

序号	功能名称	功能描述
1	生成旁路选择信号	当前四条指令写 ALU 待使用寄存器时 生成对应旁路选择信号

12. STALL（阻塞）

功能定义：

序号	功能名称	功能描述
1	数据冒险阻塞	出现 Load-Use 或 Write-Branch 冒险时阻塞一个时钟周期
2	乘除法单元阻塞	出现 MF 指令时阻塞直到乘除法单元运算结束
3	Cache 数据阻塞	Cache 未返回数据时阻塞等待
4	DCache 握手阻塞	DCache 读写冲突时阻塞等待写完成

13.控制器设计

功能定义：

序号	功能名称	功能描述
1	Ctrl	译码

14. EXCEPTION（异常）

功能定义：

序号	功能名称	功能描述
1	判断异常类型	根据异常信号判断异常类型，并生成 ExcCode 错误码以及相应 CP0 内部写信号
2	判断延迟槽	判断当前指令是否处于延迟槽
3	判断是否处理异常	当前不是中断状态，且发生异常时处理
4	清空流水线	产生 Flush 信号清空当前 IF、ID、EXE 级内容
5	选择错误地址	在发生地址错异常时，选择地址是 PC 还是 MEM
6	计算 EPC	根据延迟槽信号确定返回的 PC 值

15.CP0（系统控制寄存器）

功能定义：

序号	功能名称	功能描述
1	复位	将计数器清空//最好添加 CP0 其余寄存器的复位
2	计数	每两个时钟周期 Count 计数一次，与 compare 比较，相等产生计数器中断
3	写 CP0	根据寄存器编号写对应 CP0 寄存器
4	读 CP0	根据寄存器编号读对应 CP0 寄存器
5	记录错误地址	在发生地址错异常时，BadVAddr 记录发生错误的地址
6	记录 EPC	EPC 记录发生错误的指令 PC
7	记录错误原因	Cause 寄存器记录错误码与延迟槽信息
8	记录状态	Status 寄存器记录当前状态，屏蔽中断、是否正在处理异常

四、Crossbar

CPU 与总线的交互调用 AXI Crossbar IP 来处理 CPU 的对外总线请求。此 CPU 中使用的 3x1 Crossbar 处理来自 uncache ，icache，dcache 的 axi 总线信号，进行仲裁后发送到外部总线。

Crossbar 的具体参数如下：

- number of slave interfaces 3
- number of master interfaces 1
- ID width 2
- Protocol AXI3
- addr/data width 32
- base addr 0x0

五、ICache

ICache 的容量为 8KB，采用二路组相联的结构，其中每一路容量为 4KB。每一路共有 128 个块，每个块的大小为 32 字节，用 8 个 4 字节的 Bank 进行存储，Bank 均采用伪双端口的结构，让写过程与读过程独立。使用 LRU 算法标记其中一路的最近使用情况，由此生成所需替换的路号，完成对 Cache 块的替换。ICache 在一拍内完成从 ram 中取数、Tag 及 V 位的比较以及根据比较的命中情况参与对输出数据的选择。

CPU 对 ICache 发起请求，当请求命中时，可以在下一拍返回数据，如果 CPU 的请求连续命中，可以连续返回相应的数据，如果某次请求发生了 Miss，则通过 AXI 总线接口模块向外发起对缺失 Cache 行的访问，之后根据 LRU 中所标记路号的情况完成对 Cache 块的替换。

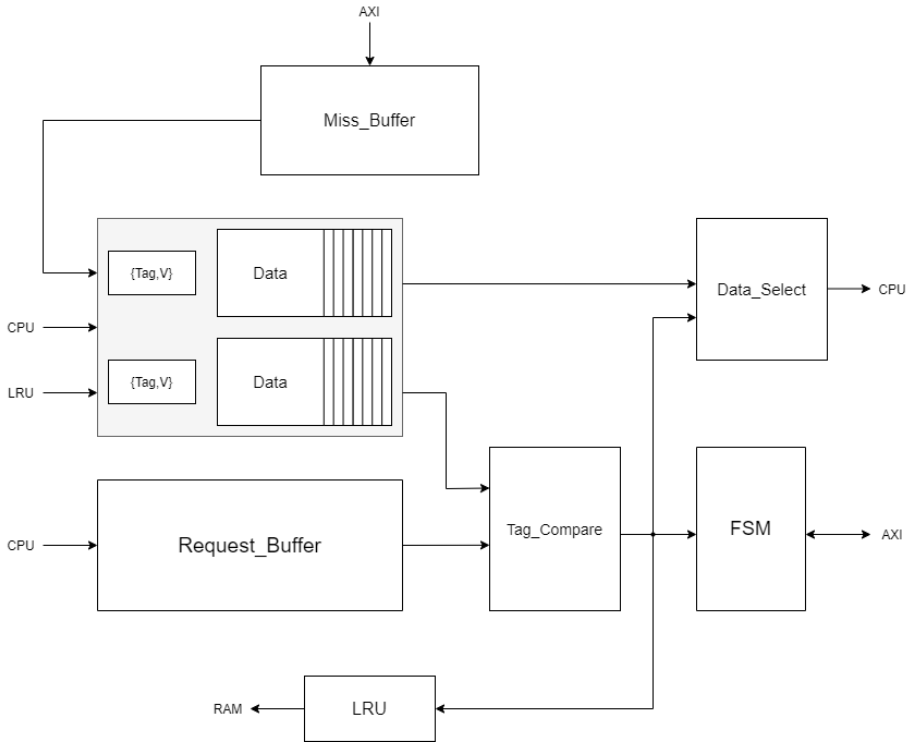

图 5.1 ICache 结构图

六、DCache

DCache 的容量为 8KB，采用二路组相联的结构，其中每一路容量为 4KB。每一路共有 256 个块，每个块的大小为 16 字节，用 4 个 4 字节的 Bank 进行存储，Bank 均采用伪双端口的结构，让写过程与读过程独立。使用 LRU 算法标记其中一路的最近使用情况，由此生成所需替换的路号，完成对 Cache 块的替换。DCache 在第一拍完成对 Tag 及 V 位的比较，生成的命中信息传递到第二拍参与对输出数据的选择。

CPU 对 DCache 发起请求，当 lw 指令发起的请求命中时，在两拍后返回数据，如果 lw 指令发起的请求连续命中，可连续返回请求的数据。当 sw 指令发起的请求命中时，在两拍后写入 ram 内部。如果 CPU 发起的请求 Miss，则通过 AXI 总线接口模块向外发起对缺失 Cache 行的访问，之后根据 LRU 中所标记路号的情况以及需要写入的数据（仅对于 sw 指令）完成对整个 Cache 块的替换。

如果遇到 sw 指令将要写入的块的索引与此时来自 CPU 指令的索引相同的情形，为了防止伪双端口 ram 产生访问冲突，我们将 DCache 阻塞一个周期。如果 sw 指令在第二拍需要写回 ram 的位置与此时第一拍中从 ram 中所取出数据的位置相同，则产生一个从第二拍到第一拍的旁路。

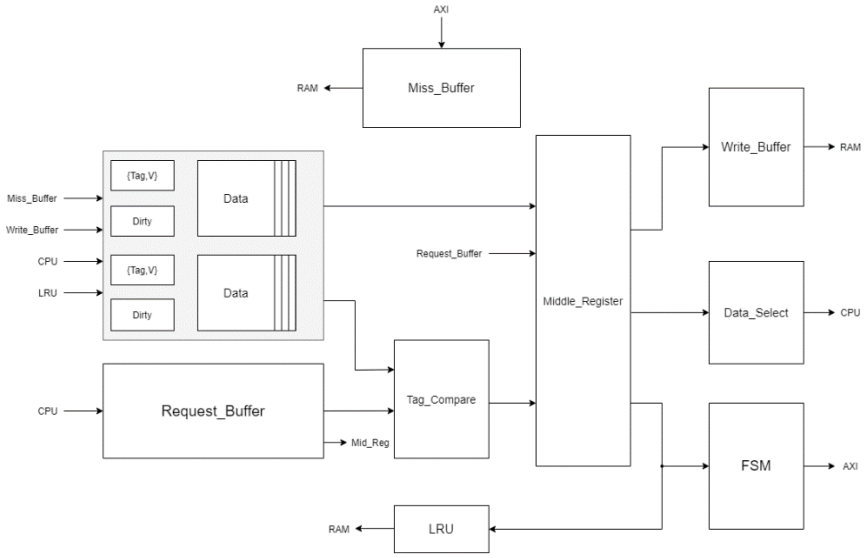

图 6.1 DCache 结构图

为了更高效地利用 DCache，我们设计了 Victim Buffer 用来存储需要写入主存的数据。Victim Buffer 可以在需要写回主存时存储好写回的目标地址与数据，并独立发起写请求，此时不影响 DCache 内部运转。如果写回过程没有结束，且之后的某一条指令 Miss 后也需要写回时，将 DCache 阻塞至写回过程结束。如果写回过程没有结束，且之后的某一条指令 Miss 后需要读取内存中同一段地址时，将 DCache 阻塞至写回过程结束。如果 Miss 后数据写入 ram 中的位置与第一拍中从 ram 中所取出数据的位置相同，则产生一个从第二拍到第一拍的旁路，并且此时无论第一拍的信息是否命中，都根据写回 ram 时替换的路号将第一拍中该路的命中信息置为 1。

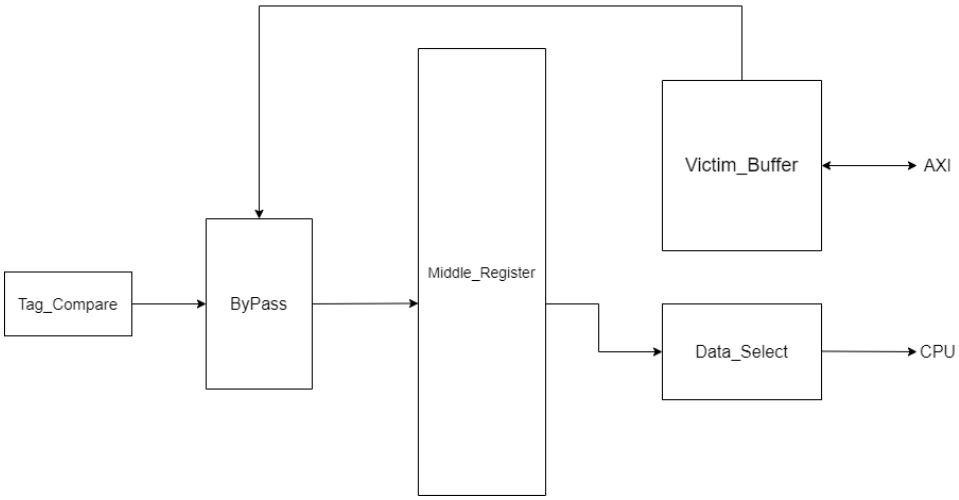

图 6.2 Victim Buffer 结构图

七、分支预测

采用 2 位饱和计数器，根据一条分支前两次的跳转结果来预测本次跳转。会有如下四种状态：

（1） Strongly taken：饱和状态，本次预测发生跳转。编码 11。

（2） Weakly taken：不饱和状态，本次预测发生跳转。编码 10。

（3） Weakly not taken：不饱和状态，本次预测不发生跳转。编码 01。

（4） Strongly not taken：饱和状态，本次预测不发生跳转。编码 00。

状态机如图 7.1 所示。

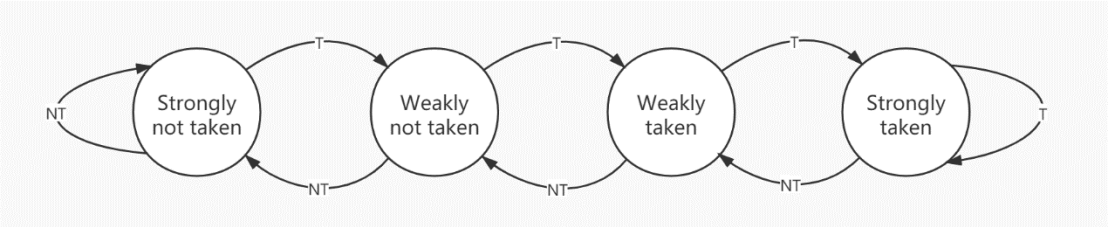

图 7.1 基于 2 位饱和计数器的分支预测状态机

状态机处于饱和状态时，只有两次判断错误才会改变预测结果，对于总是向同一方向跳转或不跳转的指令，预测结果较为准确。但若分支指令的方向频繁变化，就难以达到饱和状态，预测结果不够准确。

故在两位饱和计数器的基础上增加局部历史作为预测标准。采用 2 位 BHR（Branch History Register）记录本条指令过去两次的分支历史。每一种分支历史均对应一个 2 位饱和计数器。每当一条分支跳转指令确定方向后，更新对应历史状态的饱和计数器状态机，同时更新历史状态。

将 2 位历史状态与 4 个饱和计数器集成为一行 10 位的表 BHT（Branch History Table），利用跳转指令 PC 的 2 至 10 位索引。为避免别名导致的跳转目标地址的预测错误，仅在分支预测模块中完成对于是否跳转的预测，跳转的目标地址在进行分支预测的同时，根据当前 PC 与指令并行计算得到。

八、参考文献

计算机组成与设计: 硬件/软件接口

MIPS Architecture For Programmers Volume I-A Introduction to the MIPS32 Architecture

MIPS Architecture For Programmers Volume II-A - The MIPS32 Instruction Set

MIPS Architecture For Programmers Volume III - The MIPS32 and microMIPS32 Privileged Resource Architecture

See_MIPS_Run_2nd_Edition. Morgan_Kaufmann

超标量处理器设计. 姚永斌