Architecture of the BIPS03 32-bit RISC Embeded Processor and Its SystemC Implement

Xiao Jian-Xiong

【Abstract】The design of a high-speed 32-bit RISC processor (BIPS03) which is compatible with the MIPS32 Instruction Architecture in the EDA method of VLSI is presented. This paper first gives the background and considerations of designing the BIPS series of microprocessor. It then introduces the architecture of the BIPS03 processor, including compatible MIPS32-ISA Instruction decoder, pipeline structure, hazard controlling & branch predicting strategy and low-power data path, and Interrupt & Exception Controller. Finally, A SystemC RTL-Level description conclusion is given.

【Key words】Processor；architecture；instruction pipeline；system-on-chip；VLSI

一、序言

CPU 是现代计算机的核心，是信息产业的基本部件。在通信、计算机、自动化、家电等领域起着重要作用。近年来，随着 VLSI 技术的发展，集成电路的设计已经进入片上系统(System-on-chip，SoC) 时代。SoC 在便携式系统、智能卡、信息家电等领域都得到了广泛的应用。作为核心部件之一，高性能的微处理器对 SoC 的实现有着举足轻重的作用。然而，由于各种原因，我国信息产业中涉及 CPU 的许多核心技术与产品仍然全部依赖外国，不仅经济上受制于人，而且国家安全也面临威胁。因此，为使国家经济、国防安全得到保证，必须发展具有自主版权的 CPU 技术，并以此带动存储器、ASIC、Soc 片上系统以及其它集成电路技术的发展，促成中国 CPU 群体性崛起，以实现我国 IT 产业的跨越式发展。本项目主要研究高性能通用微处理器的设计技术。主要目标是完成一个与主流微处理器兼容的 MIPS 高性能通用 CPU 芯片的设计及相关的系统开发，并掌握高性能通用 CPU 芯片的基本体系结构、逻辑和物理设计技术，为下一阶段研制更高性能的通用 CPU 芯片打下基础。

1.1 处理器结构设计的技术路线

坚持高起点，从高性能通用处理器入手跨越式发展的技术路线。使其性能接近当今主流市场水平，能产生经济效应。在国际上，追求处理器的高性能已经到达一定的极限，很难再有大的进展，因此研究热点已经转向追求低功耗和移动计算中的处理器设计等。我们由于是从头开始，没有历史包袱，可以直接采用先进的设计，一步到位，采取跨越式发展的战略。

1.2 处理器结构设计的基本考虑

坚持兼容性设计，把兼容当作通用处理器的生命。由于现代计算机中软件开发费用已经大大超过硬件开发费用，因此兼容性设计是本系列处理器设计的重要目标。本系列 CPU 全部指令系统都保持与 MIPS 指令系统兼容。

1.3 BIPS03 简介

微处理器一般可分为复杂指令集计算机(CISC)结构和精简指令集计算机(RISC)结构。学术界和工业界已有研究表明，RISC 结构更为合理，有利于流水线和超标量技术的实现。故我们所设计的微处理器 BIPS03 属于典型的 RISC 结构。它的主要性能指标如下:

1. 5 级超流水线结构,支持冒险检测和旁发、以及分支预测能力；
2. 哈佛 Cache 存储器结构：指令 I-Cache 和数据 D-Cache；
3. 面向寄存器的操作和装入存储型存储器访问，32 位固定长度指令, 指令集与 MIPS32 核心指令集兼容；

4. 32 位数据访问、处理能力；

5. 精确异常和快速中断处理能力；

6. 成熟的定向旁路技术，确保数据相关及分支代价最小。

1.4 处理器资源

1.4.1 通用寄存器

BIPS03 提供了 32 个 32 位宽 GPRs(General Purpose Register)通用寄存器，用于整数计算及其他指令。其中根据 MIPS 体系结构标准，0 号寄存器值恒等于零。

1.4.2 专用寄存器

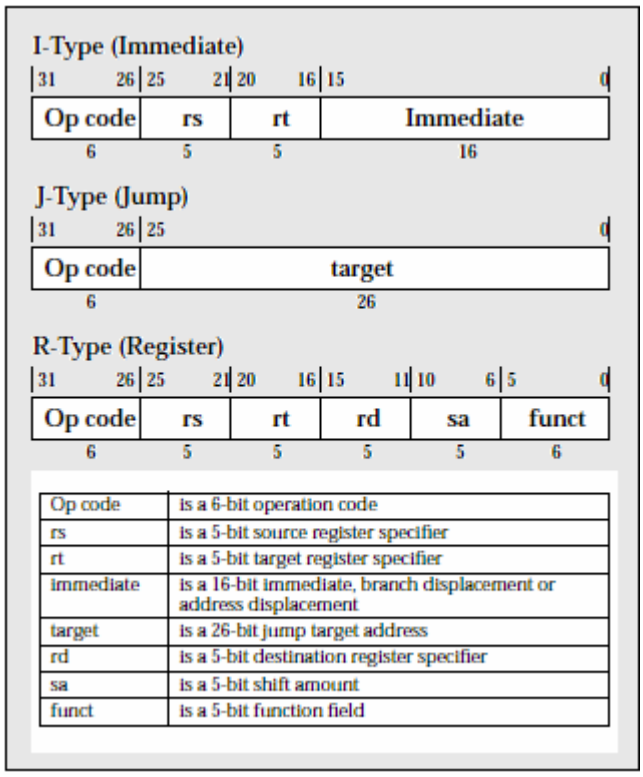
BIPS03 处理器中包括了若干个重要的寄存器，用于内存管理及中断异常的处理，其中最重要的是$30 STATUS 协处理器状态寄存器、$31 CAUSE 中断异常原因寄存器、EPC 中断异常地址寄存器。

一、 指令系统

所有 BIPS03 指令均在二进制级兼容 MIPS32 指令集，并且均为 gcc 编译器使用频率较高的 MIPS 的核心指令，全部是 32 位单字对齐。

2.1 指令类型

MIPS32 ISA 通过有效地指令编码，使其各段有效正交，从而简化了 CPU 解码过程，进而提高了运行速度。指令分三种类型：I 类型、J 类型和 R 类型，如图 1 所示。



Op code	is a 6-bit operation code
rs	is a 5-bit source register specifier
rt	is a 5-bit target register specifier
immediate	is a 16-bit immediate, branch displacement or address displacement
target	is a 26-bit jump target address
rd	is a 5-bit destination register specifier
sa	is a 5-bit shift amount
funct	is a 5-bit function field

图 1：MIPS 指令类型

2.2 指令系统

a) 存取指令(Load & Store)

用于在存储器和通用寄存器中移动数据，全部属于 I 类型指令。唯一支持的储存期寻址方式是基址寄存器加上 16 位符号扩展的直接偏移量。

助记符	指令功能
LW	取出字
SW	储存字

b) 计算指令(Computational)

用于进行算数运算、逻辑运算、移位操作。均属于 R 类型指令，其操作数均需为寄存器和 16 位直接数。

c) 跳转与分支指令(Jump & Branch)

用于改变程序控制流程，属于 R 类型或者 I 类型指令。由于分支或跳转后紧接的指令受分支跳转执行结果影响，故所有跳转分支指令均会导致流水线产生 1 次气泡。

助记符	指令功能
BEQ	如果相等则分支到
BNE	如果不等则分支到
J	跳转到

d) 特殊指令(Special)

e) 异常指令(Exception)

f) 协处理器指令(Coprocess)

用于对协处理进行操作，协处理器专用寄存器的存取指令属于 R 类型。

助记符	指令功能
BREAK	引发一个断点中断
SYSCALL	引发一个系统调用中断

2.3 实现指令

BIPS03 实现了 25 条 gcc 编译器中使用频率较高的 MIPS 的核心指令，具体如图 2 所示。

Author: XIAO,jianxiong eg_xjx@stu.ust.hk

ID	缩写	类型	功能	格式	行为	31 26	25 21	20 16	15 11	10 6	5 0	gcc编译器使用频率
						6	5	5	5	5	6	
1	add	R	加法	add rd,rs,rt	rd <- rs+rt	000000	rs	rt	rd	00000	100000	0%
2	addi	I	立即数加法	addi rt,rs,imm	rd <- rs+imm	001000	rs	rt	imm			0%
3	addu	R	无符号数加法	addu rd,rs,rt	rd <- rs+rt	000000	rs	rt	rd	00000	100001	9%
4	addiu	I	无符号数立即数加法	addiu rt,rs,imm	rd <- rs+imm	001001	rs	rt	imm			17%
5	sub	R	减法	sub rd,rs,rt	rd <- rs-rt	000000	rs	rt	rd	00000	100010	0%
6	subu	R	无符号数减法	subu rd,rs,rt	rd <- rs-rt	000000	rs	rt	rd	00000	100011	0%
7	and	R	与	and rd,rs,rt	rd <- rs and rt	000000	rs	rt	rd	00000	100100	1%
8	andi	I	立即数与	andi rt,rs,imm	rt <- rs and imm	001100	rs	rt	imm			2%
9	or	R	或	or rd,rs,rt	rd <- rs or rt	000000	rs	rt	rd	00000	100101	0%
10	ori	I	立即数或	ori rt,rs,imm	rd <- rs or imm	001101	rs	rt	imm			0%
11	sll	R	逻辑左移	sll rd,rt,sa	rd <- rt << sa	000000	00000	rt	rd	sa	000000	5%
12	srl	R	逻辑右移	srl rd,rt,sa	rd <- rt >> sa	000000	00000	rt	rd	sa	000010	0%
13	lw	I	读入字	lw rt,imm(rs)	rt <- MEM[rs+imm]	100011	rs	rt	imm			21%
14	sw	I	存储字	lw rt,imm(rs)	MEM[rs+imm] <- rt	101011	rs	rt	imm			12%
15	beq	I	相等时跳转	beq rs,rt,imm	if(rs==rt) then imm	000100	rs	rt	imm			9%
16	bne	I	不等时跳转	bne rs,rt,imm	if(rs!=rt) then imm	000101	rs	rt	imm			8%
17	j	J	跳转	j target	PC <- PC[31,28]||target||00	000010	target					2%
18	slt	R	小于则置1	slt rd,rs,rt	rd <- (rs<rt)	000000	rs	rt	rd	00000	101010	2%
19	slti	I	小于立即数则置1	slti rt,rs,imm	rt <- (rs<imm)	001010	rs	rt	imm			1%
20	sltu	R	小于无符号数则置1	sltu rd,rs,rt	rd <- (rs<rt)	000000	rs	rt	rd	00000	101011	1%
21	sltui	I	小于无符号立即数则置1	sltiu rt,rs,imm	rt <- (rs<imm)	001011	rs	rt	imm			1%
22	break		中断		EPC <- PC+4 , PC <- INT_VECTOR	000000	code				001101	0%
23	syscall		中断		EPC <- PC+4 , PC <- INT_VECTOR	000000	code				001100	0%
24	eret		中断返回		PC <- EPC	010000	10000	00000	00000	00000	011000	0%
25	nop		空			000000	00000	00000	00000	00000	000000	0%

图2： BIPS03处理器指令集(最后一列数据来自MIPS32相同指令在gcc编译器的统计数据)

二、 流水线数据通路

3.1 取指段单元

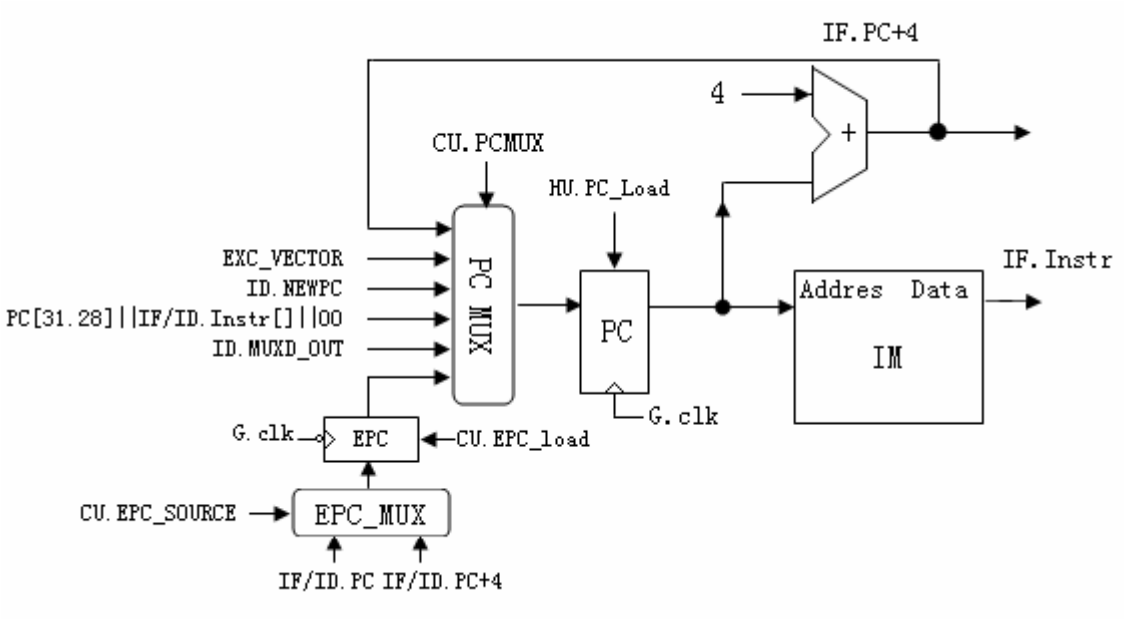


图 3：取指段单元

3.2 解码段单元

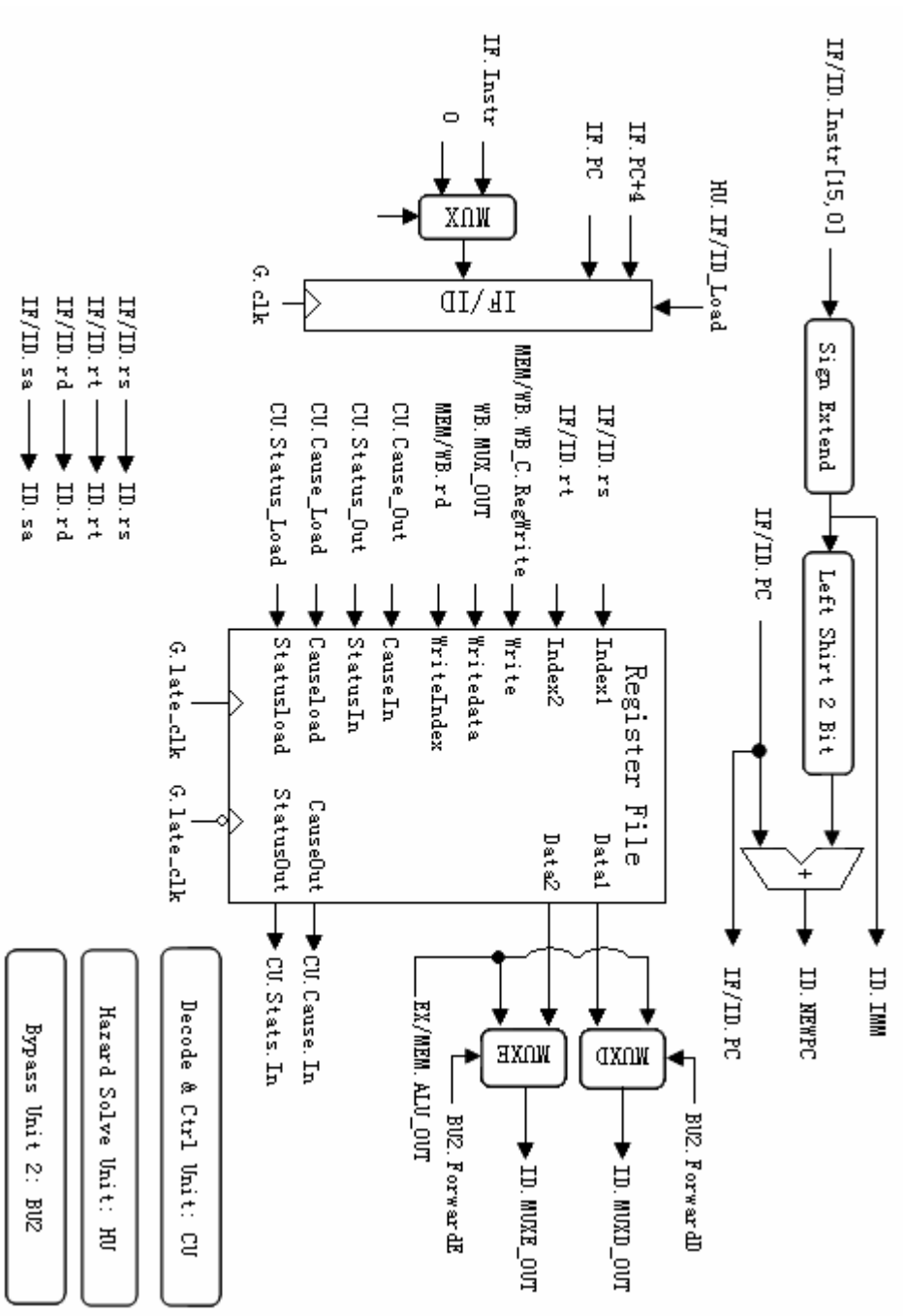


图 4：解码段单元

3.3 执行段单元

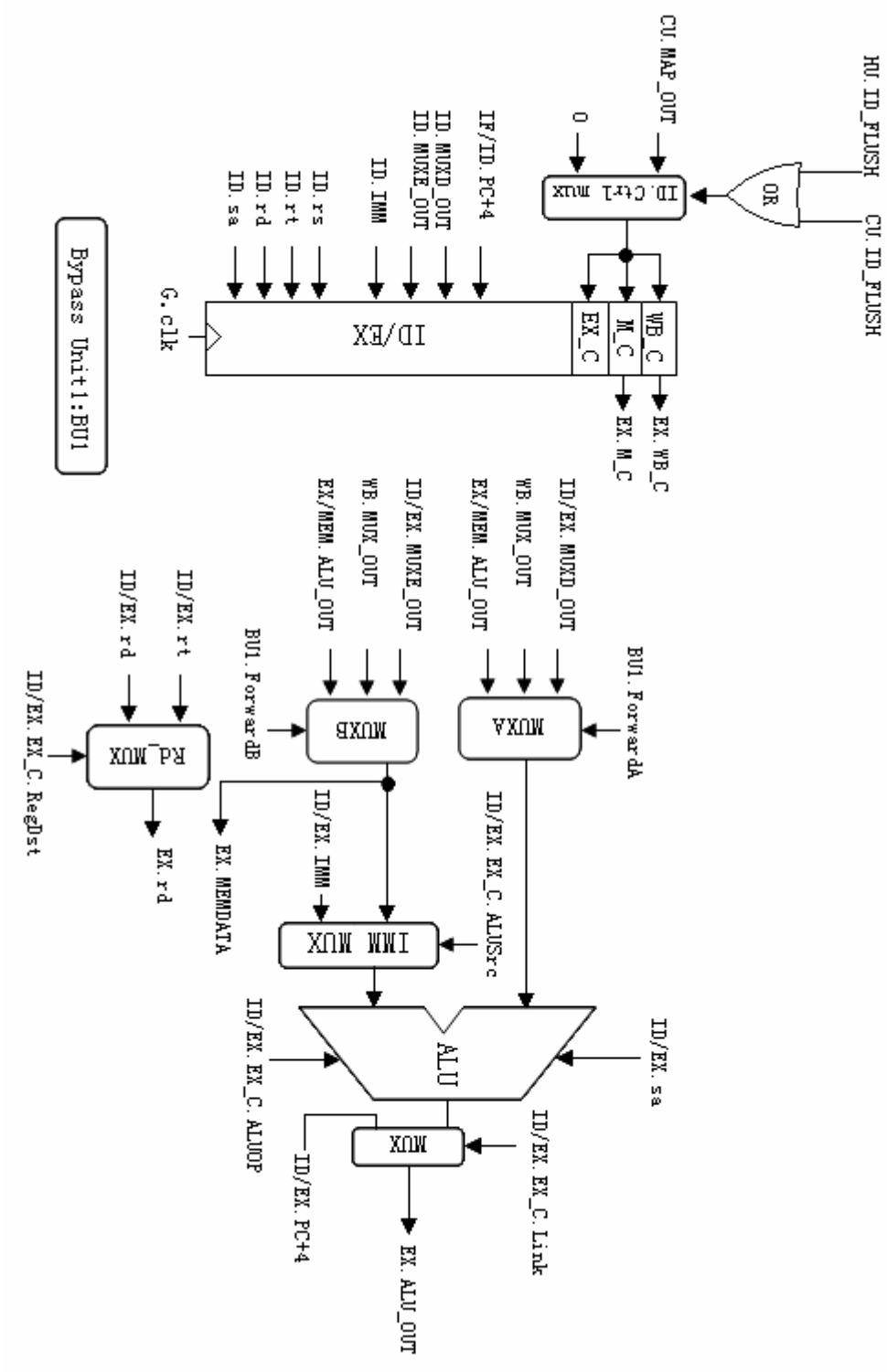


图 5：执行段单元

3.4 存储段单元

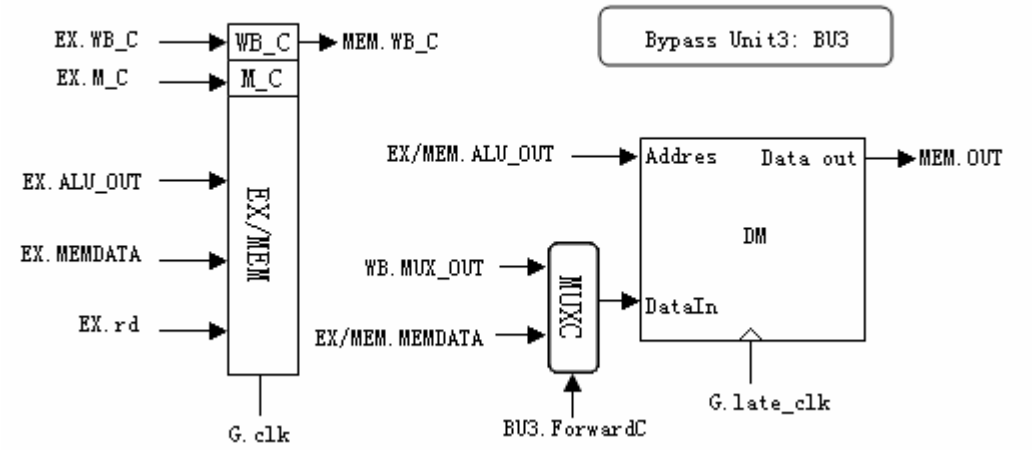


图 6：存储段单元

3.5 回写段单元

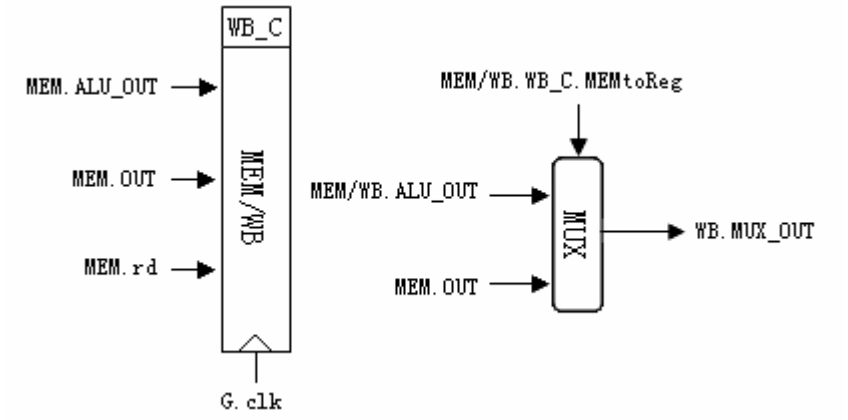


图 7：回写段单元

四、冒险处理

4.1 数据相关

由于流水线结构会导致程序相关和流水线相关，故采用冒险处理策略来减少流水线停顿。

在 ISA 的抽象层次上，程序是汇编语言指令的序列。一条典型的指令可以定义为一个函数 $i:T \leftarrow S1\,op\,S2$，其中指令 i 的定义域为 $D(i)=\{S1,S2\}$,值域为 $R(i)=\{T\}$,从定义域到值域的映射由操作 op 定义。给定两条指令 i 和 j，指令 j 跟在指令 i 之后，那么如果下列三个条件有一个成立，就认为指令 i 和 j 之间存在数据相关，或者指令 j 依赖于指令 i 并记为 $i\delta j$。

RAW 先写后读 $R(i)\cap D(j)\neq\varnothing$ $i\delta_d j$ 真相关

WAR 先读后写 $R(j)\cap D(i)\neq\varnothing$ $i\delta_a j$ 反相关

WAW 先写后写 $R(i)\cap R(j)\neq\varnothing$ $i\delta_o j$ 输出相关

由于流水线相关是由潜在地违背程序相关性而引起的，所以采用确定流水线中可能发生的流水线相关的系统方法。首先，将数据相关分为：

1. 存储器数据相关
   a) 输出相关
   b) 反相关
   c) 真数据相关
2. 寄存器数据相关
   a) 输出相关
   b) 反相关
   c) 真数据相关

存储器数据相关是指两条指令访问（读或写）存放在存储器中的同一变量。对于load/store 型体系结构，存储器数据相关职能在 load/store 指令之间发生。由于 BIPS04 采用了分离的 cache 并且只有在 MEM 能访问 D-cache，因此，所有 load/store 型指令对存储器的访问都必须且只能在 MEM 段进行，因为只有这一段能访问数据存储器，因此，对于 BIPS04 流水线，不存在由存储器数据相关引起的流水线相关。

而且，由于 BIPS03 的流水结构，反相关 $i\delta_a j$ 和输出相关 $i\delta_o j$ 出现的必要条件均不存在，因此 BIPS03 流水线中唯一能引发流水线相关的寄存器数据相关是真数据相关。真数据相关会导致 RAW 流水线相关，因为后续指令达到读寄存器的流水段的时间要限于前面的指令完成寄存器写的时间。

为了减少数据相关产生的速度开销，BIPS03 采用了定向路径旁路技术。给定两条指令 i 和 j，指令 j 跟在指令 i 之后，如果指令 i 是 ALU 指令，那么指令 j 所需要的结果将在 ALU 段产生，并且当指令结束 ALU 段时，结果就可用。如果 ALU 段的输出能够通过一条物理的定向路径（forwarding path）到达 ALU 段的输入端，那么指令 j 就可以在指令 i 离开 ALU 段的时候进入到 ALU 段。这是，指令 j 不需要在 DI 段读取寄存器堆并且访问与指令 i 相关的操作数，而是通过访问 ALU 段的输出端来获得操作数。应用技术，BIPS03 设计了 Bypass 单元进行控制。

4.2 控制相关

除了数据相关外，两条指令之间还存在控制控制相关。给定指令 i 和 j， j 跟在 i 的后面，如果指令 j 是否执行依赖于指令 i 的结果，则称指令 j 控制依赖于指令 i，记为 $i\delta_G j$。控制相关是程序的控制流程结构引起的，条件分支将给指令的顺序带来不确定因素，条件分支之后的指令与分支指令存在控制相关。

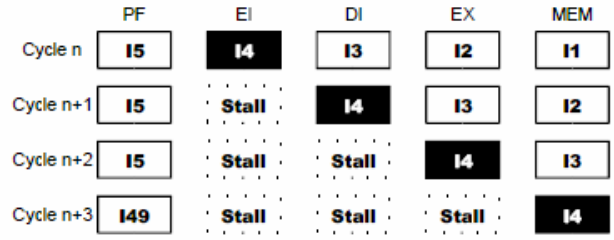


图 7: 分支预测原因

由于流水线产生分支跳转指令真实转移地址时间晚于下一流水级指令运行，故会导致无法在转移地址产生前执行下一条指令，国外已有研究统计表明，如果单纯在流水线中插入气泡 Stall，即产生控制冒险（control hazard）即分支冒险(branch hazard)，在很大程度上降低处理器性能 IPC,因此我们决定在 BIPS03 处理器中采用分支预测算法。为简化设计，采用静态分支预测算法（如假定分支不发生）。

4.3 BU1 定向路径旁路单元

BU1 处理 WB 段、MEM 段到 EX 段的的旁路控制，为主旁路单元。

其逻辑伪码如下：

```
If (EXMEM_WB_C_RegWrite && (EXMEM_rd!=0) && (EXMEM_rd == IDEX_rs))
    BU1_ForwardA =  2;
else if (MEMWB_WB_C_RegWrite && (MEMWB_rd!=0) && (MEMWB_rd==IDEX_rs))
    BU1_ForwardA =  1;
else
    BU1_ForwardA =  0;

If (EXMEM_WB_C_RegWrite && (EXMEM_rd!=0) && (EXMEM_rd == IDEX_rt))
    BU1_ForwardB =  2;
else if (MEMWB_WB_C_RegWrite && (MEMWB_rd!=0) && (MEMWB_rd==IDEX_rt))
    BU1_ForwardB =  1;
else
    BU1_ForwardB =  0;
```

4.4 BU2 定向路径旁路单元

BU2 位于 ID 段，处理跳转分支指令在 MEM 段、EX 段到 ID 段的旁路控制，而 WB 段到 ID 段的旁路控制由寄存器时钟错位解决，对于非跳转分支指令，直接利用 BU1 在 EX 段

跳转解决。

其逻辑伪码如下：

```
if (CU_O_B_or_J){
    if (EXMEM_WB_C_RegWrite && !EXMEM_WB_C_MEMtoREG
      && (EXMEM_rd!=0) && (EXMEM_rd == IFID_rs))
            BU2_ForwardD=1;
    else
            BU2_ForwardD=0;
    if (EXMEM_WB_C_RegWrite && !EXMEM_WB_C_MEMtoREG
      && (EXMEM_rd!=0) && (EXMEM_rd == IFID_rt))
            BU2_ForwardE=1;
    else
            BU2_ForwardE=0;
}else{
    BU2_ForwardD=0;
    BU2_ForwardE=0;
}
```

4.5 BU3 定向路径旁路单元

BU3 位于 MEM 段，处理 WB 段到 MEM 段的旁路控制：

其逻辑伪码如下：

```
if(EXMEM_M_C_MEMWrite && (EXMEM_rt==MEMWB_rt))
        BU3_ForwardC=0;
else
        BU3_ForwardC=1;
```

4.6 HU 冒险控制阻塞单元

HU 处理没法通过旁路技术解决的冲突，由于将跳转在 ID 段完成，所以需要区分是否是跳转来进行阻塞：

其逻辑伪码如下：

```
if (CU_O_B_or_J)  {
    if((IDEX_WB_C_RegWrite && (EX_rd!=0) &&
        ((EX_rd == IFID_rs)||(EX_rd == IFID_rt)))||(
        EXMEM_WB_C_RegWrite && EXMEM_WB_C_MEMtoREG
        && (EXMEM_rd!=0) && ((EXMEM_rd == IFID_rt) || (EXMEM_rd == IFID_rs))))
    {
        HU_PC_Load     = 0;
        HU_IFID_Load = 0;
        HU_ID_FLUSH    = 1;
    }else{
        HU_PC_Load     = 1;
        HU_IFID_Load = 1;
```

```
            HU_ID_FLUSH    = 0;
        }
}else{
        if    ( IDEX_M_C_MEMRead && (IDEX_rt == IFID_rs
            || (IDEX_rt == IFID_rt && !(CU_M_C_MEMWrite))))
        {
            HU_PC_Load     = 0;
            HU_IFID_Load = 0;
            HU_ID_FLUSH    = 1;
        }else{
            HU_PC_Load     = 1;
            HU_IFID_Load = 1;
            HU_ID_FLUSH    = 0;
        }
}
```

4.7 CU 解码控制异常处理单元

CU 负责指令解码，以及所有 CU 控制信号的生成、分支处理、中断处理、异常处理、协处理器处理。其接口信号如下：

```
in   :      <32 >          CU_Cause_I   ;
in   :      <32>           CU_Status_I   ;
in   :      <32>           CU_Instr_I;
in   :      <32>           ID_MUXD_OUT;
in   :      <32>           ID_MUXE_OUT;
in   :      <8>            INT_REQ_I;
out :       <bool >        CU_EPC_SOURCE;
out :       < 3>           CU_PCMUX_O    ;
out :       < bool >       CU_EPC_LOAD_O    ;
out :       <32>           CU_Cause_O    ;
out :       <32>           CU_Status_O    ;
out :       <bool >        CU_Cause_Load_O    ;
out :       <bool >        CU_Status_Load_O    ;
out :       <bool >        CU_IF_FLUSH_O    ;
out :       <bool >        CU_ID_FLUSH_O    ;
out :       <wb_c >        CU_WB_C    ;
out :       <m_c >         CU_M_C    ;
out :       <ex_c >        CU_EX_C    ;
out :       <bool >        CU_M_C_MEMWrite_O    ;
out :       <bool >        CU_O_B_or_J;
```

五、设计实现

设计采用 SystemC 语言做 RTL 级描述。SystemC 是一种完成电子系统从软件到硬件的全部建模过程的语言，在系统设计方面有明显优势。它是于 1999 年 9 月，由微电子业内一

流的EDA公司、IP提供商、半导体制造商及系统和嵌入式软件设计公司在加利福尼亚州Saint Jose 举行的“嵌入式系统会议”上联合创建的。其 SystemC 2.01 作为一个稳定的版本已经提交 IEEE 进行标准化。目前，SystemC 在国内也有了广泛的应用，已有国内外一些大公司利用 SystemC 开发项目成功的案例，深圳清华大学研究院 EDA 与网络应用重点实验室使用 SystemC 开发的数字音频芯片亦流片成功。因此，使用 SystemC 这种规则化的软硬皆可以实现的成熟语言，比自己独立开发一个与之不兼容的语言或库更加可行，故采用 SystemC 进行设计具有很大的优势。

本设计采用自顶向下的设计 top-down 方法,采用 Mentor Graphics ModelSim SE PLUS 6.0 进行仿真验证、Visual C++6.0 进行编译，设计正确。由于时间及设备限制（需要 Solaris8 工作站及 SystemC Compiler 软件、版图设计软件），尚未进行 FPGA 验证、物理设计、流片联调，这些工作将在实验条件允许的情况下尽快开展。

参考文献

[1] John L. Hennessy, David A. Patterson. Computer Architecture: A Quantitative Approach. (Third Edition) Morgan Kaufmann Publishers, 2001

[2] John L. Hennessy, David A. Patterson. Computer Organization & Design: The Hardware/Software Interface (Second Edtion), Morgan Kaufmann Publishers ,2001

[3] Kessler R. The Alpha 21264 Microprocessor. IEEE Micro, 1999,19(2): 24～36

[4] Kenneth Yeager. The MIPS R10000 Superscalar Microprocessor. IEEE Micro, 1996,16(2): 28～41

[5] Ashok Kumar. The HP PA-8000 RISC CPU. IEEE Micro, 1997,17(2): 27～32

[6] IEEE Computer Society. 1076 IEEE Standard VHDL Language Reference Manual, 1993

[7] M. Morris Mano , Charles R. Kime. Logic and Computer Design Fundamentals,2E Updated. Prentice Hall, 2002

[8] MIPS32™ Architecture For Programmers Volume II: The MIPS32™ Instruction Set. MIPS Technologies Inc, 2003

[9] IEEE 1754, IEEE standard for a 32-bit microprocessor architecture, IEEE, 1994

[10] IEEE standard for boot (initialization configuration) firmware instruction set architecture (ISA) supplement for IEEE 1754, IEEE, 1994.

[11] The SPARC Architecture Manual (Version 8), SPARC International, inc, 1992.

[12] John Paul Shen and Mikko H. Lipasti. Modern Processor Design : Fundamentals of Superscalar Processor. McGraw-Hill, 2003

[13] 大型 RISC 处理器设计，[德]Ulrich Golze 著，田泽等译，北京航空海天大学出版社，2005 年