# 计算机组成CPU设计文档-P5

[计算机组成CPU设计文档-P5 1](#_Toc531216646)

[整体结构与概览 2](#_Toc531216647)

[1、 CPU基本参数与指标 2](#_Toc531216648)

[2、 CPU模块结构 2](#_Toc531216649)

[一、 模块规格（数据通路） 4](#_Toc531216650)

[1、 数据通路 4](#_Toc531216651)

[2、 IFU（取指令单元） 6](#_Toc531216652)

[3、 GRF（通用寄存器组） 6](#_Toc531216653)

[4、 ALU（逻辑运算单元） 7](#_Toc531216654)

[5、 DM（数据存储器） 8](#_Toc531216655)

[6、 Ext（位数扩展器） 9](#_Toc531216656)

[7、 NPC（分支跳转指令地址计算器） 9](#_Toc531216657)

[8、 CMP（分支条件判断） 10](#_Toc531216658)

[9、 PipeReg（流水线寄存器） 10](#_Toc531216659)

[二、 模块规格（控制电路） 11](#_Toc531216660)

[三、 CPU功能测试 17](#_Toc531216661)

[1、 功能测试原则 17](#_Toc531216662)

[2、 测试策略 17](#_Toc531216663)

[3、 测试实例 18](#_Toc531216664)

[四、本章思考题 24](#_Toc531216665)

[五、有关CPU扩展的说明 27](#_Toc531216666)

[六、版本信息 29](#_Toc531216667)

## 整体结构与概览

## CPU基本参数与指标

处理器类型：流水线CPU

处理器字长：32位

处理器支持指令集：{addu, subu, jr, sll(nop), ori, lui, beq, lw, sw, jal, jr, j}

顶层封装模块端口：

表格 1 顶层封装模块端口

|  |  |  |
| --- | --- | --- |
| 端口名 | 类型 | 描述 |
| reset | In | 接受同步复位信号，对PC、GRF、Mem和PipeReg复位。 |
| clk | In | 接受时钟驱动信号，驱动PC、GRF/W、IM/W、PipeReg |

## CPU模块结构

### 数据通路

1、IFU（取指令单元）：包括PC和存放指令的ROM，用于输出当前指令码。

2、GRF（通用寄存器组）：内含32个寄存器，支持对寄存器值的读写。

3、ALU（算术逻辑单元）：运算执行部件，对32位数执行多种运算。

4、DM（数据存储器）：存储数据部件，支持读写。

5、EXT（位扩展器）：将16位数扩展为32位数，支持有/无符号扩展。

6、NPC（外部跳转分支计算器）：支持跳转计算和分支计算与判断，若为分支指令，会根据CMP结果选择正确的pc值。

7、CMP（分支比较器）：根据指令条件设置，返回判断结果。

8、PipeReg（流水线寄存器层）：实现CPU流水并行效果，上升沿时接受前一层运行完且需要传递至下一级的数据，其他时刻释放本层功能所需的数据。一共分为FD，DE，EM，MW四层。

9、FuncMux（功能多选器）：对同一端口多个数据源进行筛选，目前有AluSrc、WaSel、WdSel三个。

10、TMux（转发多选器）：用于转发解决冲突时所使用的多选器，目前有GRF\_RD1, GRF\_RD2, DE\_RD1, DE\_RD2, EM\_RD2。

### 控制信号

1. 主控器：识别指令并生成CPU各部分的控制信号，使用逻辑阵列实现。
2. 冲突控制器：
   1. GID：通用指令译码器：根据所给指令和流水线段，返回Tuse、Tnew、写地址、数据管道等参数。
   2. STALL：暂停控制器，输出是否暂停的指令。
   3. TRANSMIT：转发控制器，输出5个转发多选器的控制信号。

## 模块规格（数据通路）

## 数据通路

表格 2 数据通路端口合成-无转发



表格 3 数据通路端口合成-转发



表格 4 转发多选器



## IFU（取指令单元）

### 端口

表格 5 IFU端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| Clock | In | 控制信号，接受时钟信号 |
| Reset | In | 控制信号，接受Pc同步复位信号 |
| Branch\_Jump | In | 控制信号，是否接受NPC输出作为PC新值 |
| Pc\_Update[31:0] | In | 数据通路，分支/跳转指令中接受PC更新值 |
| PC4[31:0] | Out | 数据通路，输出PC+4（32位**Byte编址**） |
| Instr[31:0] | Out | 数据通路，输出32位指令二进制码 |
| PC[31:0] | Out | 调试信号，输出PC（32位**Byte编址**） |

### 功能描述

* IFU主要由PC和存放指令的ROM组成，用于取出指令和PC更新。
* ROM规格为1024\*32bits，**字编址（访问时需地址转换）**。
* PC为32位二进制，起始地址为0x00003000，**字节编址**，支持向字编址转换（除4）。

### 注意事项

* ROM和RAM部件的地址端口为字编址地址。为保证兼容性，该设计在顶层设计时用字节编址，而在次层具体部件设计时会进行字编址转换。

## GRF（通用寄存器组）

### 端口

表格 6 GRF端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| Clock | In | 控制信号，接受时钟信号 |
| Reset | In | 控制信号，接受同步复位信号 |
| RegWrite | In | 控制信号，接受寄存器写使能信号 |
| ReadAddr1/R[4:0] | In | 数据通路，读，接受Rs寄存器地址 |
| ReadAddr2/R[4:0] | In | 数据通路，读，接受Rt寄存器地址 |
| WriteAddr/W[4:0] | In | 数据通路，写，接受被写入寄存器地址 |
| WriteData/W[31:0] | In | 数据通路，写，接受被写入数据 |
| RegData1/R[31:0] | Out | 数据通路，读，输出Rs寄存器值 |
| RegData2/R[31:0] | Out | 数据通路，读，输出Rt寄存器值 |
| WPC[31:0] | None | 调试信号，用于寄存器写时display |

### 功能描述

* GRF中共有32个寄存器，对应MARS中的32个通用寄存器。（注意：不包括hi，lo，pc寄存器。）
* 读：GRF读功能时作为组合逻辑电路，根据输入地址信号，输出数据。
* 写：GRF写功能作为时序逻辑电路，相应的Addr, Data, RegWrite应在时钟上升沿前做好准备。

### 备注

* 此版本GRF读写功能的地址和数据端口独立，可实现同步读写操作。
* 0号寄存器恒为0值，不可被改写。

## ALU（逻辑运算单元）

### 端口

表格 7 ALU端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| AluOp[3:0] | In | 控制信号，接受算术逻辑信号 |
| A[31:0] | In | 数据通路，接受算术逻辑操作数A |
| B[31:0] | In | 数据通路，接受算术逻辑操作数B |
| C[4:0] | In | 数据通路，接受算术逻辑操作数C（常于移位） |
| Result[31:0] | Out | 数据通路，输出结果 |

### 功能描述

* ALU受AluController的控制信号控制，输出不同的算术逻辑结果：

表格 8 ALU功能表

|  |  |  |
| --- | --- | --- |
| AluOp | 功能 | 适用指令 |
| 000/0 | 32位 +，不带溢出检测 | addu, lw, sw |
| 001/1 | 32位 - ，不带溢出检测 | subu |
| 010/2 | 32位 左移运算B<<C | sll(nop) |
| 011/3 | 32位 | | ori |
| 100/4 | 32位B<<16 | lui |
| 101/5 | 32位 B+4（PC+8） | Jal |

## DM（数据存储器）

### 端口

表格 9 数据存储器端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| Clock | In | 控制信号，接受时钟信号 |
| Reset | In | 控制信号，接受异步复位信号 |
| MemWrite | In | 控制信号，接受写使能信号 |
| MemRead | In | 控制信号，接受读使能信号 |
| MemAddr[31:0] | In | 数据通路，接受**读/写**操作地址, **byte编址** |
| WriteData[31:0] | In | 数据通路，写，写入数据 |
| ReadData[31:0] | Out | 数据通路，读，输出数据 |
| WPC[31:0] | None | 调试信号，用于主存写时display |

### 描述

* 数据存储器使用RAM实现，容量为1024\*32bits，RAM**字节编址**。
* 读/写共用一个地址端口，同一时钟周期只能进行读/写的其中之一。
* 起始地址：0x00000000。

## Ext（位数扩展器）

表格 10 16-32位扩展器端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| ExtOp | In | 控制信号，控制扩展方式（0-zero，1-sign） |
| In[15:0] | In | 数据通路，接收待扩展的16位数字。 |
| Out[31:0] | Out | 数据通路，输出扩展后的32位数字。 |

## NPC（分支跳转指令地址计算器）

表格 11 NPC端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| nPc\_Sel[2:0] | In | 控制信号，控制地址计算方式  （0-branch，1-j/jal，2-jr） |
| Cmp | In | 数据通路，接受CMP的分支决策信号。(beq) |
| Im32[31:0] | In | 数据通路，接受EXT的**符号扩展**立即数。(beq) |
| Im26[25:0] | In | 数据通路，接受指令中的26位立即数。(j/jal) |
| Pc4[31:0] | In | 数据通路，接受当前指令Pc+4。（j/jal/beq） |
| RegPc[31:0] | In | 数据通路，接受从寄存器中读取的跳转值。（jr） |
| Pc\_Update[31:0] | Out | 数据通路，输出PC分支跳转计算的地址。 |

## CMP（分支条件判断）

表格 12 CMP端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| CmpOp | In | 控制信号，指定branch指令比较策略（0-equal） |
| A[31:0] | In | 数据通路，比较数A |
| B[31:0] | In | 数据通路，比较数B |
| Cmp | Out | 比较结果（1-成立，0-不成立） |

## PipeReg（流水线寄存器）

表格 13 PipeReg端口表

|  |  |  |
| --- | --- | --- |
| 流水线级别 | 端口 | 功能描述 |
| FD段 | F\_IR → FD\_IR | D段指令 |
| F\_Pc4 → FD\_Pc4 | D段PC+4 |
| F\_Pc → FD\_Pc | D段PC（display） |
| DE段 | D\_IR → DE\_IR | E段指令 |
| D\_Pc4 → DE\_Pc4 | E段Pc+4 |
| D\_RD1 → DE\_RD1 | E段GRF[RS] |
| D\_RD2 → DE\_RD2 | E段GRF[RT] |
| D\_EXT → DE\_EXT | E段扩展立即数 |
| D\_Pc → DE\_Pc | E段PC（display） |
| EM段 | E\_IR → EM\_IR | M段指令 |
| E\_ALU → EM\_ALU | M段ALU计算结果 |
| E\_RD2 → EM\_RD2 | M段GRF[RT] |
| E\_Pc → EM\_Pc | M段PC（display） |
| MW段 | M\_IR → MW\_IR | W段指令 |
| M\_ALU → MW\_ALU | W段ALU计算结果 |
| M\_MD → MW\_MD | W段Memory读取结果 |
| M\_Pc → MW\_Pc | W段Pc（display） |

## 模块规格（控制电路）

与单周期CPU不同的是，流水线CPU在运行中存在结构冒险、数据冒险和控制冒险三种冒险问题。结构冒险利用GRF和指令数据DM分离的方式已经解决，控制冒险使用分支跳转提前+延迟槽的方式实现，而数据冒险需要使用暂停和转发逻辑实现。

综上，控制电路分为主控单元MainController和冒险控制单元HazardController。

### Controller 主控单元

* 1. 端口

表格 14 主控单元端口功能表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能（所在通路，作用部件，描述） |
| Op[5:0] | In | 数据通路，指令的Instr[31:26] |
| Func[5:0] | In | 数据通路，指令的Instr[5:0] |
| D\_Branch\_Jump | Out | 控制信号，IF，IF接受外部Pc更新信号（1-允许，0-不允许） |
| D\_ExtOp | Out | 控制信号，16-32位扩展类型（1-符号扩展，0-无符号扩展） |
| D\_nPc\_Sel[2:0] | Out | 控制信号，PC分支跳转类型  0：Branch  1：Jump/Jal  2：Jr |
| D\_CmpOp | Out | 控制信号，Branch指令比较策略（0-EQUAL） |
| E\_AluSrc[1:0] | Out | 控制信号，MUX，ALU-B端口选择器信号（1-扩展器，0-RD2） |
| E\_AluOp[3:0] | Out | 控制信号，ALU，ALU驱动信号（具体请参加ALU功能表） |
| M\_MemRead | Out | 控制信号，Mem，Mem读使能信号（1-允许，0-不允许:高阻） |
| M\_MemWrite | Out | 控制信号，Mem，Mem写使能信号（1-允许，0-不允许） |
| W\_WaSel[1:0] | Out | 控制信号，MUX，GRF写地址选择  0：Rt  1：Rs  2：31 |
| W\_WdSel[1:0] | Out | 控制信号，MUX，GRF写数据选择  0：Alu  1：Memory |
| W\_RegWrite | Out | 控制信号，GRF，GRF写使能信号 （1–允许，0-不允许） |

* 1. 信号真值表

表格 15主控单元信号真值表

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | addu | subu | sll | ori | lui | lw | sw | beq | j | jal | jr |
| Branch\_Jump | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| RegWrite | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| ExtOp | x(0) | x(0) | x(0) | 0 | x(0) | 1 | 1 | 1 | x(0) | x(0) | x(0) |
| nPc\_Sel | x(0) | x(0) | x(0) | x(0) | x(0) | x(0) | x(0) | 0 | 1 | 1 | 2 |
| AluOp | Add  (0) | Sub  (1) | Sll  (2) | Or  (3) | s16  (4) | Add  （0） | Add  （0） | x  (0) | x  (0) | B+4  (5) | x  (0) |
| MemRead | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| CmpOp | x(0) | x(0) | x(0) | x(0) | x(0) | x(0) | x(0) | 0 | x(0) | x(0) | x(0) |
| AluSrc | 0 | 0 | 0 | 1 | 1 | 1 | 1 | x(0) | x(0) | 2 | x(0) |
| WaSel | 1 | 1 | 1 | 0 | 0 | 0 | x(0) | x(0) | x(0) | 2 | x(0) |
| WdSel | 0 | 0 | 0 | 0 | 0 | 1 | x(0) | x(0) | x(0) | 0 | x(0) |

\*：lui指令与ori指令：前者完全不顾及GRF读操作，且对扩展方式不在意。

### Hazard冲突控制单元

* 1. 概览

冲突控制是目前流水线CPU与单周期CPU差异最大的地方，在实现上也具有一定难度。本CPU的冲突控制单元主要解决的是数据冒险问题，（结构冒险和控制冒险已通过功能部件和数据通路构造），通过“流水线工程化”[[1]](#footnote-1)方法，主要比较Tnew、Tuse等值，即可实现对应的转发和暂停策略。

* 1. General Instruction Decoder （通用指令译码器）

流水线工程化方法实质是对各级指令所对应的Tnew、Tuse、操作寄存器等参数进行比对，形成解决策略。由于每条指令的参数不相同，为模块化功能，设计了“通用指令译码器”进行一系列参数的计算。

表格 16 GID端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| IR[31:0] | In | 数据通路，对应流水段指令 |
| Pipe[2:0] | In | 数据通路，流水段编号（F-1，D-2，E-3，M-4） |
| Tuse\_Rs[2:0] | Out | 数据通路，当前流水段Rs的Tuse  no\_more\_use = 7 |
| Tuse\_Rt[2:0] | Out | 数据通路，当前流水段Rt的Tuse  no\_more\_use = 7 |
| RegWriteNonZero | Out | 数据通路，当前指令是否**向非零寄存器写值** |
| A3 | Out | 数据通路，当前指令写入寄存器地址  Rt：lw、cali  Rd：calr  Jal：$31 |
| Tnew[2:0] | Out | 数据通路，当前指令产生写入结果所需时间  no\_more\_new = 0 |
| Dport[2:0] | Out | 数据通路，当前指令写入结果所在寄存器“管口”  1：EM\_ALU  2：MW\_ALU  3：MW\_MD |

表格 17 指令分类与读写功能统计表格

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 指令分类： |  |  |  | 读 | 写 |
| calr | addu | subu | sll | √ | √ |
| cali | lui | ori |  | √ | √ |
| ld | lw |  |  | √ | √ |
| st | sw |  |  | √ |  |
| btype | beq |  |  | √ |  |
| jal\_type | jal |  |  |  | √ |
| j\_type | j |  |  |  |  |
| jr\_type | jr |  |  | √ |  |

表格 18 F段Tuse

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tuse | | | | | | | | | |
| IF/ID | | | | | | | | | |
| calr/rs/1 | calr/rt/1 | cali/rs/1 | cali/rt/1 | ld/rs/1 | st/rs/1 | st/rt/2 | btype/rs/0 | btype/rt/0 | jr/rs/0 |

表格 19 各级Tnew（仅针对产生写的指令）

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tnew | | | | | | | | | | | |
| ID/EX | | | | EX/MEM | | | | MEM/WB | | | |
| calr/rd1 | cali/rt1 | ld/rt2 | jal/$311 | calr/rd0 | cali/rt0 | ld/rt1 | jal/$310 | calr/rd0 | cali/rt0 | ld/rt0 | jal/$310 |

* 1. STALL暂停控制模块

表格 20 STALL暂停控制模块端口

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| FD\_IR[31:0] | In | 数据通路，D段指令 |
| DE\_IR[31:0] | In | 数据通路，E段指令 |
| EM\_IR[31:0] | In | 数据通路，M段指令 |
| MW\_IR[31:0] | In | 数据通路，W段指令 |
| Stall | Out | 数据通路，暂停信号 |

暂停判断逻辑基于GID返回的参数结果，判断式中不含有具体的指令类型，因此具有可延伸性。具体判断逻辑式如下（以Rs与第X段判断为例）：

* 1. TRANSMIT转发控制模块

表格 21 TRANSMIT端口

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| FD\_IR[31:0] | In | 数据通路，D段指令 |
| DE\_IR[31:0] | In | 数据通路，E段指令 |
| EM\_IR[31:0] | In | 数据通路，M段指令 |
| MW\_IR[31:0] | In | 数据通路，W段指令 |
| TMux\_GRF\_RD1\_Sel[2:0] | Out | 数据通路，TMux\_GRF\_RD2转发器选择信号 |
| TMux\_GRF\_RD2\_Sel[2:0] | Out | 数据通路，TMux\_GRF\_RD2转发器选择信号 |
| TMux\_EM\_RD1\_Sel[2:0] | Out | 数据通路，TMux\_GRF\_RD2转发器选择信号 |
| TMux\_EM\_RD2\_Sel[2:0] | Out | 数据通路，TMux\_GRF\_RD2转发器选择信号 |
| TMux\_MW\_RD2\_Sel[2:0] | Out | 数据通路，TMux\_GRF\_RD2转发器选择信号 |

转发采用了改进的“暴力转发，随时转发”策略，其意思为：被转发位置的指令无论是否需要用到某个寄存器的值，一旦此寄存器的新值在后续流水段中已经产生，则会进行转发。当多个后级流水线转发时，级数较低的优先级更高。

以TMux\_GRF\_RD1接受X段PORT“寄存器管口”更新值为例：

* 1. TMux（转发多选器）

转发多选器TMux规格是同样的，其每个端口所对应供给者是固定的，因此存在空缺端口的情况。

表格 22 TMUX端口表

|  |  |  |
| --- | --- | --- |
| 端口名称 | 类型 | 功能描述 |
| TMux\_Sel[2:0] | In | 控制信号，转发器选择信号 |
| Ori[31:0] | In | 数据通路，原始通路 |
| EM\_ALU[31:0] | In | 数据通路，EM的ALU供给者 |
| MW\_ALU[31:0] | In | 数据通路，MW寄存器的ALU供给者 |
| MW\_MD | In | 数据通路，MW寄存器的MD供给者 |
| Forward[31:0] | Out | 数据通路，通路正确结果 |

## CPU功能测试

## 功能测试原则

* 所测试的指令不能够超出CPU支持的范围。（谨防同指令标识的拓展指令。）
* 测试的首要目标：全面覆盖性（例如：冲突覆盖、单指令界限等）。
* 测试步骤：控制信号（主控） -> 数据通路 -> 控制信号（暂停）->冒险覆盖性测试（暂停+转发）

## 测试策略

1. 控制信号检查（主控）：独立检查每条指令的控制信号是否符合设计。
2. 数据通路检查：
   1. Addu, subu: 正正（溢出/不溢），负负（溢出/不溢），正负，负正。
   2. Lui，ori：负数的构造。
   3. Sw，lw：非零数边界存储，正负偏移存储。
   4. Beq：成立与不成立跳转，下跳与上跳。
   5. Sll检查：Nop功能。
   6. Jal检查：正负绝对跳转，$31赋值细节检查。（pc4）
   7. Jr检查：$31, $0, $8检查
3. 控制信号（冲突）检查
   1. GID译码正确性：使用testbench模块测试GID对每条指令的返回结果。
   2. STALL正确性：使用testbench模块测试STALL模块所有STOP组合。
4. 冒险覆盖性测试：
   1. 暂停：枚举所有STOP情况和部分NOSTOP情况，观察Stall信号。
   2. 转发：覆盖性测试，自动化评测。

## 测试实例

### 控制信号正确性检查

使用下列代码检查各指令运行时，控制信号值是否符合期望，控制信号期望输出于第三节以表格给出，在此不再赘述。

addu $t1, $t2, $t3

subu $t1, $t2, $t3

lw $t1, 12($t2)

sw $t1, 12($t2)

ori $t2, $t2, 0xffff

lui $t1, 0xf

sll $t1, $t2, 2

back:

beq $t1, $t2,back

jal next

j next

nop

next:

jr $ra

label:

代码 1 控制信号正确性检查代码

### 单指令正确性检查

单指令正确性检查首先是lui和ori，因为这两条指令才能构造非零数。而后，利用构造完成的各种数值，对指令addu、subu、lw和sw进行测试。

测试代码与期望结果如下：

# 单指令运算正确性检查:addu, subu, lw, sw, ori, lui # 检查内容与期望结果

# Lui, Ori

lui $s0, 0x7ff1 # 功能检查：$16 = 0x7ff10000

# nop \* 5

ori $s1, $zero, 0xf001 # 功能检查：$17 = 0x0000f001

# nop \* 5

lui $s2, 0xffff

# nop \* 5

ori $s3, $s2, 0xfffc # 负数构造：$19 = -4

# nop \* 5

# addu, subu

addu $t0, $s0, $s1 # 不溢出相加：$8 = 0x7ff1f001

# nop \* 5

addu $t1, $s0, $s0 # 溢出相加：$9 = 0xffe20000

# nop \* 5

addu $t2, $s3, $s3 # 负负相加 $10 = -8

# nop \* 5

addu $t3, $s1, $t2 # 正负相加 $11 = 61433

# nop \* 5

subu $t4, $s1, $t3 # 正正相减 $12 = 8

# nop \* 5

subu $t5, $t3, $s1 # 正正相减 $13 = -8

# nop \* 5

subu $t6, $s3, $t5 # 负负相减 $14 = 4

# nop \* 5

lui $s4, 0x8000

# nop \* 5

subu $t7, $s4, $t4 # 负正相减(溢出) $15 = 0x7ffffff8

# nop \* 5

# sw, lw

ori $t8, 0

# nop \* 5

sw $t0, 0($t8) # sw下边界测试：mem[0] = 0x7ff1f001

# nop \* 5

sw $t1, 16($t8) # sw 偏移量测试：mem[4] = 0xffe20000

# nop \* 5

ori $t9, 4092

# nop \* 5

sw $t4, 0($t9) # sw上边界测试：mem[1023] = 8

# nop \* 5

sw $t5, -16($t9) # sw 负偏移测试：mem[1019] = -8

# nop \* 5

ori $t8, 13

# nop \* 5

lw $s4, 3($t8) # 和为4倍偏测试：$20 = 0xffe20000

# nop \* 5

lw $s5, 0($t9) # 上边界测试 $21 = 8

# nop \* 5

lw $s6, 4($t9) # 无效界读入测试

# nop \* 5

代码 2 单指令真确性检查代码1

而后针对beq、jal、jr指令进行检测

# 单指令正确性检查：ori, lui, beq, jal, jr # 检查期望结果和pc跳转情况

# beq # 检测目的与预期结果

ori $t0, 1

# nop\*5

ori $t1, 2

# nop\*5

ori $t2, 3

# nop\*5

ori $t4, 1

# nop\*5

addu $s0, $s1, $s2 # 非跳转语句选择无效检测：pc = pc + 4 (not pc + 4 + 0x8021)

# nop\*5

beq $t0, $t1, label2 # 不跳转检测：pc = pc+4

# nop\*5

nop

label1:

lui $s0, 0xf000

# nop\*5

beq $t0, $t4, label4 # 正跳转检测：pc = pc+4+8

# nop\*5

label2:

lui $s1, 0x0f00

# nop\*5

label3:

lui $s2, 0x00f0

# nop\*5

jal function\_begin

# nop\*5

ori, $t5, 0x3048 # 非ra跳转，至return标签

# nop\*5

jr $t5

# nop\*5

label4:

lui $s3, 0x000f

# nop\*5

addu $t0, $t0, $t4

# nop\*5

beq $t0, $t1, label3 # 逆跳转检测：pc = pc+4-16

# nop\*5

jal function\_end

# nop\*5

return:

ori $s6, $zero, 0x00f0

# nop\*5

jal function\_end

# nop\*5

function\_begin:

ori $s4, $zero, 0xf000

# nop\*5

jr $ra

# nop\*5

ori $s5, $zero, 0x0f00

# nop\*5

function\_end:

# 顺序：label1 - > label4 -> label 3-> function\_begin -> return -> function\_end

# 最终期望输出：$16 = 0xf0000000, $17 = 0, $18 = 0x00f00000, $19 = 0x000f0000, $20 = 0x0000f000, $21 = 0, $22 = 000000f0, $31 = 00003050, $8 = 3

代码 3 单指令正确性检查代码2

### 冲突控制信号（暂停）检查

此时并未进行CPU检查，而是针对STALL模块进行testbench检查。

lw $1, 0($2) # calr - ld

subu $2, $1, $2

lw $1, 0($3) # cali - ld

ori $2, $0, 1

lw $1, 0($2) # ld - ld (rs)

lw $2, 0($1)

lw $1, 0($2) # st - ld (rs)

sw $2, 0($1)

lw $1, 0($2) # no stop! st-ld(rt)

sw $1, 0($0)

lw $1, 0($0) # btype - ld

beq $2,$1,label

lw $1, 0($0) # jr\_type - ld

jr $1

addu $1,$2,$3 # btype -calr

beq $1,$2,label

ori $1,$0,100 # btype - cali

beq $2,$1,label

addu $1,$2,$3 # jr\_type - calr

jr $1

lui $1,0xf # jr\_type - cali

jr $1

lw $1 0($1) # beq - ld

nop

beq $1,$2,label

lw $1 0($1) # jr - ld

nop

jr $1

label:

代码 4 冲突控制信号（暂停）测试代码

### 转发覆盖性检查

由于转发情况甚多，使用testbench检查过于复杂，因此采用自动化评测的方式测试，测试代码构建表格如下，测试代码过大，请参见附件。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 位置 | 前序指令 | 冲突寄存器 | 实例代码 |
| 1 | D-M | nop, addu | Rs | addu $1, $2, $3  nop  beq $1,$2, label |
| 2 | D-M | nop, sll | Rt | sll $1, $2, 5  nop  beq $2, $1, label |
| 3 | D-M | nop, ori | Rs | ori $1, $1, 5  nop  jr $1 |
| 4 | D-M | nop, jal | Rt | jal label  nop  beq $ra $1 label |
| 5 | D-M | nop, jal | Rs | jal label  nop  jr $ra |
| 6 | D-W | nop, nop, lw | Rt | lw $1, 0($0)  nop  nop  beq $1 $2 label |
| 7 | D-W | nop, nop, subu | Rs | subu $1, $2, $3  nop  nop  jr $1 |
| 8 | D-W | nop, nop, lui | Rt | lui $1, 100  nop  nop  beq $2, $1, label |
| 9 | D-W | nop, nop, jal | Rs | jal label  nop  nop  jr $ra |
| 10 | E-M | subu | Rs | subu $1, $2, $3  addu $2, $1, $2 |
| 11 | E-M | lui | Rs | lui $1, 100  addu $2, $1, $1 |
| 12 | E-M | sll | Rt | sll $1,$1,5  ori $1, $1, 1 |
| 13 | E-M | jal | Rs | jal label  lw $1, -4($31) |
| 14 | E-M | jal | Rs | jal label  sw $1, 4($31) |
| 15 | E-M | ori | Rs | ori $1, 100  lw $2, 100($1) |
| 16 | E-W | nop, lw | Rs | lw $1, 0($0)  nop  sw $2, 0($1) |
| 17 | E-W | nop. lw | Rt | lw $1, 0($0)  nop  subu $2,$1,$1 |
| 18 | E-W | nop, lw | RS | lw $1, 80($0)  nop  ori $1,$1,0xf000 |
| 19 | E-W | nop, lw | Rs | lw $1, 200($0)  nop  lw $2, 0($1) |
| 20 | E-W | nop, ori | Rt | ori $1 $2 4  nop  addu $1 $0 $1 |
| 21 | E-W | nop, jal | Rs | jal label  nop  sw $1 0($31) |
| 22 | E-W | nop, subu | Rs | subu $2,$1,$1  nop  lw $3,0($2) |
| 23 | M-W | lw | Rt | lw $1 0($0)  sw $1 4($0) |
| 24 | M-W | lui | Rt | lui $1 100  sw $1 0($0) |
| 25 | M-W | jal | Rt | jal label  sw $31 100($0) |
| 26 | M-W | subu | Rt | subu $1,$0,$1  sw $1, 100($0) |
| 27 | 无转发 |  |  | lw $0, 0($0)  nop  addu $1,$0,$1 |
| 28 | 无转发 |  |  | lui $0 100  sw $0 0($1) |

## 四、本章思考题

* 在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？请有条理的罗列出来。(**非常重要**)

答：实验中遇到了两类数据冒险，一类可以通过暂停来解决，而另一类则需通过转发来解决。下面针对两种情况分类作答。

a.通过“流水线工程化方法”，我们容易得到所有需要暂停的情况，且暂停情况相较于转发情况是相当有限的，本CPU的暂停策略如下：

表格 23 STALL情况分析表格



由以上表格，我罗列了伪代码表格，并在本章第四节附有实测代码：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 位置 | 前序指令 | 冲突寄存器 | 实例代码 |
| 1 | F-D | lw | Rs | lw $1, 0($2)  subu $2, $1, $2 |
| 2 | F-D | lw | Rt | lw $1, 0($3)  ori $2, $0, 1 |
| 3 | F-D | lw | Rs | lw $1, 0($2)  lw $2, 0($1) |
| 4 | F-D | lw | Rt | lw $1, 0($2)  sw $2, 0($1) |
| 5 | F-D | lw | None | lw $1, 0($2)  sw $1, 0($0) |
| 6 | F-D | lw | Rs | lw $1, 0($0)  beq $1,$2,label |
| 7 | F-D | lw | Rt | lw $1, 0($0)  beq $2,$1,label |
| 8 | F-D | lw | Rs | lw $1, 0($0)  jr $1 |
| 9 | F-D | calr | Rs | addu $1,$2,$3  beq $1,$2,label |
| 10 | F-D | calr | Rt | ori $1,$0,100  beq $1,$2,label |
| 11 | F-D | cali | Rs | ori $1,$0,100  beq $1,$2,label |
| 12 | F-D | cali | Rt | ori $1,$0,100  beq $2,$1,label |
| 13 | F-D | calr | Rs | addu $1,$2,$3  jr $1 |
| 14 | F-D | cali | Rs | lui $1,0xf  jr $1 |
| 15 | F-E | nop, lw | Rs | lw $1 0($1)  nop  beq $1,$2,label |
| 16 | F-E | nop, lw | Rs | lw $1 0($1)  nop  beq $2,$1,label |
| 17 | F-E | nop, lw | Rs | lw $1 0($1)  nop  jr $1 |

表格 24 Stall功能测试

b.转发相较于暂停，若只停留在指令覆盖性层面，则复杂性大幅度增长。因此对于转发正确性的测量，应该去到更抽象的维度，用更抽象的模型去代表一类指令，在实际测试时用少量的指令代表一类指令即可。因此这个测试问题量级就从一维变成了二维乘法了，测试中，首先是确保“抽象”过程的完全正确，其次随机选择少量指令代表性地检查。

1.在本CPU中抽象方法是GID（通用指令译码器），其测试方法是用testbench测试每一条指令，观察其输出值（建立的模型）是否正确。期望输出请参见第三节中的指令分类、Tnew、Tuse表格。

2.基于抽象模型建立正确的基础上，分析转发者和需求者：

1. 需求者：

a) D:beq, jr

b) E:calr, cali, lw, sw

c) M:sw

2. 供给者

a) EM:calr, cali, jal

b) MW: lw+(calr, cali, jal)

\*：括号（）中的内容代表其不实新晋的供给者，但是依然具有供给功能。

同时这样就构建出了D-EM, D-MW, E-EM, E-MW, M-MW五类转发情况，对于每类转发情况，从中需求者和供给者中选出有代表性的指令测试即可。（实际操作时是对新晋供给者指令进行全覆盖测试，同时包含Rs和Rt，不是新晋的供给者则抽样测试。）具体测试伪代码和代码请参见第三节代码和附件。

## 五、有关CPU扩展的说明

本CPU支持一定功能的扩展，需要在Verilog上进行改进。由于涉及到流水线的分层结构和冒险管理问题，因此较单周期CPU需要考虑的内容更多，因此在新增指令和调试时请务必参考以下的步骤。

1. 分析新增指令的需求，必要难以理解时及时借助MARS测试，将指令拆分为数据通路+控制逻辑。
2. 若时间充裕，以下所有的内容都应该**手稿分析后再实践**。
3. 数据通路：
   1. 根据指令需求绘制出数据通路图（注意分层和转发器位置）
   2. **先微观，再宏观：**若仅需细调部分部件功能，则实时调整；若需要新增通路，则应该以“尽量少增转发器和转发点”为原则构建，同时结合图与先前的数据通路表格，确定新增数据通路结构。
   3. 在完成数据通路搭建后，时间充裕情况下应进行检测。
4. 主控制指令：与单周期类似，可以**先按照单周期分析**，而后考虑流水线寄存器。若新增或修改指令，除此还需要考虑指令所在流水段，进行正确的添加。
5. ！冲突控制：分为暂停和转发
   1. 冲突控制一定要**借助通路图和工程化方法的指标**分析。
   2. 暂停：完善指令类型，明确指令的Tnew，加入到GID中。
   3. ！转发：
      1. 若无新通路产生，需求解Tuse，更改GID，并细致分析。
      2. 若有新的承载数据的通路，则TMux、GID、TRANSMIT可能需要都需要修改。
      3. 若有新的读取GRF的通路（不建议！），则需新增TMux，修改GID端口、修改TRANSMIT。
   4. 遇到问题：**看暂停表和CPU供给关键点图**。
6. 核查：沿着手绘的数据通路+两类控制单元复查。
7. 练手指令：movz, branch+link, jalr .

## 六、版本信息

1. Version0：构建了不支持转发和暂停的流水线数据通路。
2. Version1：增加了Hazard文件，GID、STALL、TRANSMIT和TMUX新增，支持暂停和转发。（无法通过中强测试）。
3. Version2：修复了bugs。Beq指令未成功执行时，外部返回的pc更新值应该是pc+8；GID模块中lw指令没有抽象成ld指令，增加了拓展性。

1. L15-流水线工程化方法-2018-V1，高小鹏，北京航空航天大学《计算机组成课程设计》. [↑](#footnote-ref-1)