数字逻辑与处理器基础实验

流水线MIPS处理器设计实验报告

2020010860

无08 王知衡

目录

一、实验名称与内容 ………………………………………………… 2

二、设计方案 ………………………………………………………… 2

（一）实现的指令集 ………………………………………………………… 2

（二）数据通路 ……………………………………………………………… 2

（三）模块介绍 ……………………………………………………………… 2

（四）工作流程 ……………………………………………………………… 4

（五）冒险与转发 …………………………………………………………… 4

（六）外设 …………………………………………………………………… 7

三、关键代码及文件清单 …………………………………………… 7

四、仿真结果及分析 …………………………………………………16

五、综合情况 …………………………………………………………17

六、硬件调试情况 ……………………………………………………18

七、实验小结 …………………………………………………………18

**一、实验名称与内容**

实验名称：流水线MIPS处理器设计

实验内容：将春季学期实验四设计的单周期（多周期） MIPS 处理器改进为流水线结构，并利用此处理器完成字符串匹配。

二、设**计方案**

**（一）实现的指令集**

本次实验实现的指令集包含

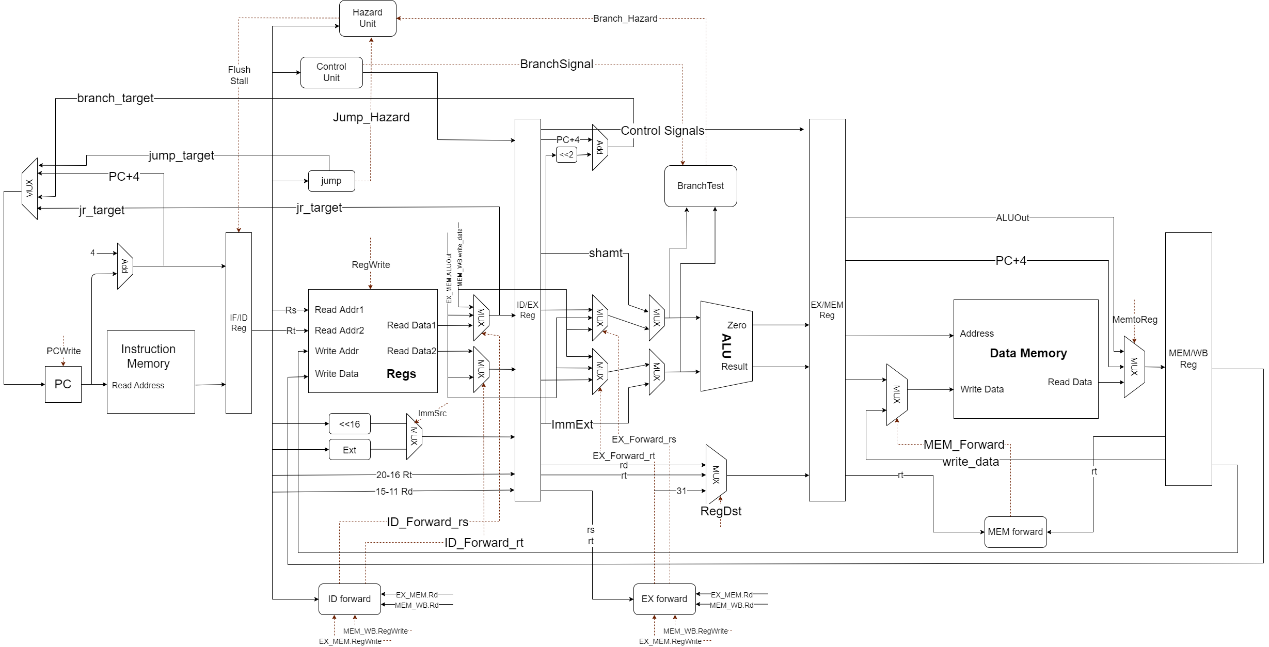
1、R型指令：add、addu、sub、subu、and、or、xor、nor、slt、sltu、sll、srl、sra、jr、jalr；

2、I型指令：lui、addi、addiu、andi、ori、xori、slti、sltiu、lw、sw、beq、bne、blez、bgtz、bltz；

3、J型指令：j、jal。

**（二）数据通路**

数据通路图如下：



**（三）模块介绍**

本程序共包括以下几个模块：

·CPU：顶层模块，负责完成整个数据通路的数据流动，如根据控制信号等确定PC的输入值、根据转发控制信号进行数据转发等，同时与外设（leds、an、bcd）进行交互；

·PC：负责PC的更新，受PCWrite控制（PCWrite=1时方可更新）；

·InstMem：根据PC值从指令存储器中读取指令，其中预先存好了用汇编转化过来的机器码（可实现预期功能，即字符串查找与数码管显示）。

·IF\_ID\_Reg：IF与ID级间寄存器，受IF\_Flush和stall控制清空和阻塞；

·ControlUnit：根据指令产生相应的控制信号以及标记该指令是否为分支和跳转指令，方便后续操作。

·ImmExtendUnit：立即数扩展模块，可选择实现有符号扩展、无符号扩展、左移16位；

·RegisterFile：寄存器堆；

·HazardUnit：判断冒险单元，可判断是否存在load-use冒险、分支指令导致的控制冒险、跳转指令导致的控制冒险，并产生相应的控制信号，其他冒险由转发单元判断；

·ID\_Forward\_Ctrl：到ID阶段的转发单元，判断是否需要由EX\_MEM或MEM\_WB将数据转发到ID阶段，并产生相应控制信号。

·ID\_EX\_Reg：ID与EX级间寄存器，受ID\_Flus控制清空；

·EX\_Forward\_Ctrl：到EX阶段的转发单元，判断是否需要由EX\_MEM或MEM\_WB将数据转发到EX阶段，并产生相应控制信号。

·ALU：算术逻辑单元，对从寄存器读出的数值（转发的数值）和立即数进行计算。

·BranchTest：在EX阶段判断分支指令是否跳转。

·EX\_MEM\_Reg：EX与MEM级间寄存器。

·MEM\_Forward\_Ctrl：到MEM阶段的转发单元，判断是否需要由MEM\_WB将数据转发到EX阶段，并产生相应控制信号。

·DataMem：数据存储器，可写入或读取数据（预先存入字符串），被分为数据RAM和外设地址空间。

·MEM\_WB\_Reg：MEM与WB级间寄存器。

**（四）工作流程**

IF 阶段负责程序计数器的更新以及取指令，以PC寄存器的输出为指令地址得到指令，并储存在 IF/ID 寄存器中。

ID 阶段负责寄存器堆的读取和立即数扩展，并进行分支指令与跳转指令的判断。本阶段将寄存器堆中读取的寄存器数据以及立即数扩展结果存入到ID/EX 寄存器中。

EX 阶段进行 ALU 运算，以及将要写入的寄存器地址 WriteAddr 的选择。还会判断分支指令是否跳转。

MEM 阶段进行数据存储器的读取或写入。

EX 阶段选择写入寄存器堆的数据，并将数据写入寄存器堆。

**（五）冒险与转发**

（1）数据冒险

数据冒险是由不同指令的操作数具有相互依赖关系而造成的。在本处理器程序中采用数据转发和阻塞流水线的方法解决数据冒险。

下面阐述五条转发路径的设计思路以及其解决的问题：

ⅰ.MEM\_WB到ID的转发

MEM\_WB寄存器中存储了WB将要写入寄存器的数据和地址，而ID阶段将要进行的是读取寄存器中的数据，理论上若不进行转发，读写寄存器将同时进行。将MEM\_WB的数据转发到ID阶段，可以实现先写后读，如此一来，如果读取的位置和写入的位置相同，将读取到新写入的数据。

判断依据为：

·WB阶段将进行写入寄存器操作且目标寄存器ID不为0；

·写入地址与读取地址相同。

ⅱ.EX\_MEM到ID的转发

jr指令在ID阶段就要获取寄存器的值，但如果上一条指令写入寄存器与jr指令读取寄存器相同的话，就需要将其要写入的数据转发到jr指令所在的ID阶段。而jr指令的上一条指令此时正在EX阶段，因此需要在上一条指令经过ALU计算后立刻将计算结果转发到jr指令的读取结果，这就是EX\_MEM到ID的转发。

判断依据为：

·转发目标读取的不是0寄存器；

·转发源与转发目标的寄存器编号一致；

·ID阶段为jr或jalr指令，EX阶段指令需写入寄存器（RegWrite=1）。

ⅲ.MEM\_WB到EX的转发

对于前前条执行访存或者其他需要写入寄存器的指令，可以采取从MEM\_WB\_Reg转发到 EX 阶段的策略，将同一时刻正在 WB 阶段的写入数据用于 EX 的计算功能中。

判断依据为：

·转发源与转发目标的寄存器编号一致；

·MEM/WB阶段必须写入寄存器（寄存器堆的写使能为1且目标寄存器ID不是0）。

ⅳ.EX\_MEM到EX的转发

对于前一条、需要写入寄存器的指令，可以采取从EX\_MEM\_Reg转发到 EX 阶段的策略，将同一时刻正在 MEM 阶段的写入数据用于 EX 的计算功能中。

判断依据为：

·转发源与转发目标的寄存器编号一致；

·EX/MEM阶段必须写入寄存器（寄存器堆的写使能为1且目标寄存器ID不是0）。

值得注意的是，EX/MEM寄存器的转发优先级比MEM/WB寄存器的转发优先级高。

ⅴ.MEM\_WB到MEM的转发

还需要考虑对memory操作的情况，即相邻的lw和sw将内存中一个位置的数写到另一个位置。此时，需要进行MEM/WB到EX/MEM的转发，将lw即将写入寄存器的数据直接转发给sw。

判断依据为：

·转发目标读取的不是0寄存器；

·转发源与转发目标的寄存器编号一致；

·EX/MEM阶段的指令要写memory，而MEM/WB阶段的指令读memory。

除此之外，还有一种特殊的数据冒险——load-use冒险。通过lw从memory中读取一个数后立刻要在ALU中参与计算，会导致这种冒险。load-use冒险无法只用数据转发的方法解决。我们需要在检测到这种冒险时阻塞流水线一个周期，再进行转发，同时还要清空ID\_EX寄存器。load-use冒险出现的条件为：

·ID/EX阶段的指令要读取memory；

·IF/ID阶段的指令要读取的寄存器和ID/EX阶段的指令要写入的寄存器一致。

（2）控制冒险

ⅰ.跳转指令的控制冒险

j指令的跳转目标地址在ID阶段被计算出来，但j指令的下一条指令的IF阶段依赖于这一结果，因而造成了控制冒险。对于j、jal、jr、jalr四条指令，在 ID 阶段控制信号产生模块生成 jump\_hazard 信号，传入冒险处理模块。根据冒险模块产生的控制信号，将本来不该执行的指令从流水线中清除（flush）出去。

ⅱ.分支指令的控制冒险

对于分支指令，处理器默认不跳转，即IF和ID不暂停工作。在分支指令的 ID 阶段，控制信号产生模块根据分支指令的类型，产生BranchSignal并存储到流水线寄存器中。在 EX阶段， BranchTest模块接收转发后的rs\_data和rt\_data作为输入，并进行相关判断。当判断为真时，确定存在控制冒险，Branch\_Hazard 信号拉高，冒险模块随即产生 IF\_Flush 和 ID\_Flush 指令，将对应的 IF\_ID\_Reg 和 ID\_EX\_Reg 中的关键信号改为 0，即清理掉错误的指令。而当判断为假时，处理器继续运行，可以减少一定的 CPI。简言之，此 CPU 对分支的预测总为假。

**（六）外设**

实验结果需要在BCD数码管上显示，由于本实验采用纯软件编写，因此需要将计算出来的数据通过汇编指令译码，并写入相应的外设地址空间中。

实验中要用到的七段数码管地址为0x40000010，从0bit到11bit分别对应CA、CB、CC、……、CG、DP、AN0、AN1、AN2、AN3。

**三、关键代码及文件清单**

1、关键代码

汇编指令：

li $a0 41 # $a0=len\_str

li $a1 0 # $a1=str\_address

li $a2 4 # $a2=len\_pattern

li $a3 1024 # $a3=pattern\_address

brute\_force:

sub $t0 $a0 $a2 #$t0=len\_str - len\_pattern

li $t1 0 #$t1=i

li $t2 0 #$t2=j

li $v0 0 #$v0=cnt

bf\_i:

slt $t3 $t0 $t1 #i>len\_str-len\_pattern ->$t3=1

bgtz $t3 bf\_i\_exit #if i<len\_str-len\_pattern break

bf\_j:

slt $t5 $t2 $a2

beqz $t5 bf\_j\_exit #if j>=len\_pattern break

add $t6 $a3 $t2 #$t6=j+pattern\_address

add $t6 $t6 $t2 #$t6=2\*j+pattern\_address

add $t6 $t6 $t2 #$t6=3\*j+pattern\_address

add $t6 $t6 $t2 #$t6=4\*j+pattern\_address

lw $t6 0($t6) #$t6=pattern[j]

add $t7 $a1 $t1 #$t7=i+str\_address

add $t7 $t7 $t1 #$t7=2\*i+str\_address

add $t7 $t7 $t1 #$t7=3\*i+str\_address

add $t7 $t7 $t1 #$t7=4\*i+str\_address

add $t7 $t7 $t2 #$t7=4\*i+j+str\_address

add $t7 $t7 $t2 #$t7=4\*i+2\*j+str\_address

add $t7 $t7 $t2 #$t7=4\*i+3\*j+str\_address

add $t7 $t7 $t2 #$t7=4\*i+4\*j+str\_address

lw $t7 0($t7) #$t7=str[i+j]

bne $t6 $t7 bf\_j\_exit #if != break

addi $t2 $t2 1 #j++

j bf\_j

bf\_j\_exit:

beq $t2 $a2 cnt #if j == len\_pattern

li $t2 0 #j=0

addi $t1 $t1 1 #i++

j bf\_i

cnt:

addi $v0 $v0 1 #cnt++

li $t2 0 #j=0

addi $t1 $t1 1 #i++

j bf\_i

bf\_i\_exit:

li $t4 0 #t4=pos

li $a0 0

lui $a0 1

addi $a0 $a0 0x86a0 # a0=1e5

add $t1 $v0 $zero # t1: gewei

sll $t1 $t1 28

srl $t1 $t1 28

add $t2 $v0 $zero # t2: shiwei

sll $t2 $t2 24

srl $t2 $t2 28

add $t6 $v0 $zero # t6: baiwei

sll $t6 $t6 20

srl $t6 $t6 28

add $t7 $v0 $zero # t7: qianwei

sll $t7 $t7 16

srl $t7 $t7 28

dis\_pos:

slti $t3 $t4 4 # t3 = (t4<4)

beqz $t3 bf\_i\_exit

beq $t4 $zero gewei

subi $t3 $t4 1

beq $t3 $zero shiwei

subi $t3 $t4 2

beq $t3 $zero baiwei

subi $t3 $t4 3

beq $t3 $zero qianwei

li $t9 0

pre:

sll $t5 $t5 8 # confirm an

beq $t9 $zero j0

subi $t3 $t9 1

beq $t3 $zero j1

subi $t3 $t9 2

beq $t3 $zero j2

subi $t3 $t9 3

beq $t3 $zero j3

subi $t3 $t9 4

beq $t3 $zero j4

subi $t3 $t9 5

beq $t3 $zero j5

subi $t3 $t9 6

beq $t3 $zero j6

subi $t3 $t9 7

beq $t3 $zero j7

subi $t3 $t9 8

beq $t3 $zero j8

subi $t3 $t9 9

beq $t3 $zero j9

subi $t3 $t9 10

beq $t3 $zero j10

subi $t3 $t9 11

beq $t3 $zero j11

subi $t3 $t9 12

beq $t3 $zero j12

subi $t3 $t9 13

beq $t3 $zero j13

subi $t3 $t9 14

beq $t3 $zero j14

subi $t3 $t9 15

beq $t3 $zero j15

start:

sw $t5 0x40000010($zero)

li $t8 0 #t8=count

dis\_count:

slt $t3 $t8 $a0 # t3 = (t8<1e5)

beqz $t3 dis\_count\_exit

addiu $t8 $t8 1

j dis\_count

dis\_count\_exit:

addiu $t4 $t4 1

j dis\_pos

j0:

addi $t5 $t5 192

j start

j1:

addi $t5 $t5 249

j start

j2:

addi $t5 $t5 164

j start

j3:

addi $t5 $t5 176

j start

j4:

addi $t5 $t5 153

j start

j5:

addi $t5 $t5 146

j start

j6:

addi $t5 $t5 130

j start

j7:

addi $t5 $t5 248

j start

j8:

addi $t5 $t5 128

j start

j9:

addi $t5 $t5 144

j start

j10:

addi $t5 $t5 136

j start

j11:

addi $t5 $t5 131

j start

j12:

addi $t5 $t5 198

j start

j13:

addi $t5 $t5 161

j start

j14:

addi $t5 $t5 134

j start

j15:

addi $t5 $t5 142

j start

gewei:

add $t9 $zero $t1

li $t5 14 # 1110

j pre

shiwei:

add $t9 $zero $t2

li $t5 13

j pre

baiwei:

add $t9 $zero $t6

li $t5 11

j pre

qianwei:

add $t9 $zero $t7

li $t5 7

j pre

CPU.v PC更新

assign PC\_Plus4=PC+4;

assign PC\_next=(Branch\_Hazard) ? branch\_target :

(PCSrc==2'b00) ? PC\_Plus4 :

(PCSrc==2'b01) ? jump\_target :

(PCSrc==2'b10) ? jr\_target : 32'b0;

CPU.v ID阶段转发

assign rs\_data\_forward\_id =

(ID\_Forwarding\_rs==2'b10) ? MEM\_WB\_Reg.write\_data :

(ID\_Forwarding\_rs==2'b01) ? EX\_MEM\_Reg.ALUOut : ReadData1;

assign rt\_data\_forward\_id = (ID\_Forwarding\_rt) ? MEM\_WB\_Reg.write\_data : ReadData2;

CPU.v EX阶段转发

assign rs\_data\_forward\_ex =

(EX\_Forwarding\_rs==2'b10) ? MEM\_WB\_Reg.write\_data :

(EX\_Forwarding\_rs==2'b01) ? EX\_MEM\_Reg.ALUOut : ID\_EX\_Reg.ReadData1;

assign rt\_data\_forward\_ex = (EX\_Forwarding\_rt==2'b10) ? MEM\_WB\_Reg.write\_data :

(EX\_Forwarding\_rt==2'b01) ? EX\_MEM\_Reg.ALUOut : ID\_EX\_Reg.ReadData2;

CPU.v ALU输入控制

assign ALU\_op1 =

(ID\_EX\_Reg.ALUSrc1)?{27'b0,ID\_EX\_Reg.shamt}:rs\_data\_forward\_ex;

assign ALU\_op2 = (ID\_EX\_Reg.ALUSrc2)?ID\_EX\_Reg.Imm\_Ext:rt\_data\_forward\_ex;

CPU.v 写回数据选择

assign wb\_wr\_data =

(EX\_MEM\_Reg.MemtoReg==2'b00)?EX\_MEM\_Reg.ALUOut:

(EX\_MEM\_Reg.MemtoReg==2'b01)?mem\_read\_data:EX\_MEM\_Reg.PC\_Plus4;

2、程序清单

source：

·ALU.v

·BranchTest.v

·Control.v

·CPU.v

·DataMem.v

·EX\_Forwarding.v

·EX\_MEM.v

·Hazard.v

·ID\_EX.v

·ID\_Forwarding.v

·IF\_ID.v

·ImmExtend.v

·InstMem.v

·MEM\_Forwarding.v

·MEM\_WB.v

·PC.v

·RegisterFile.v

simulation:

·test\_cpu.v

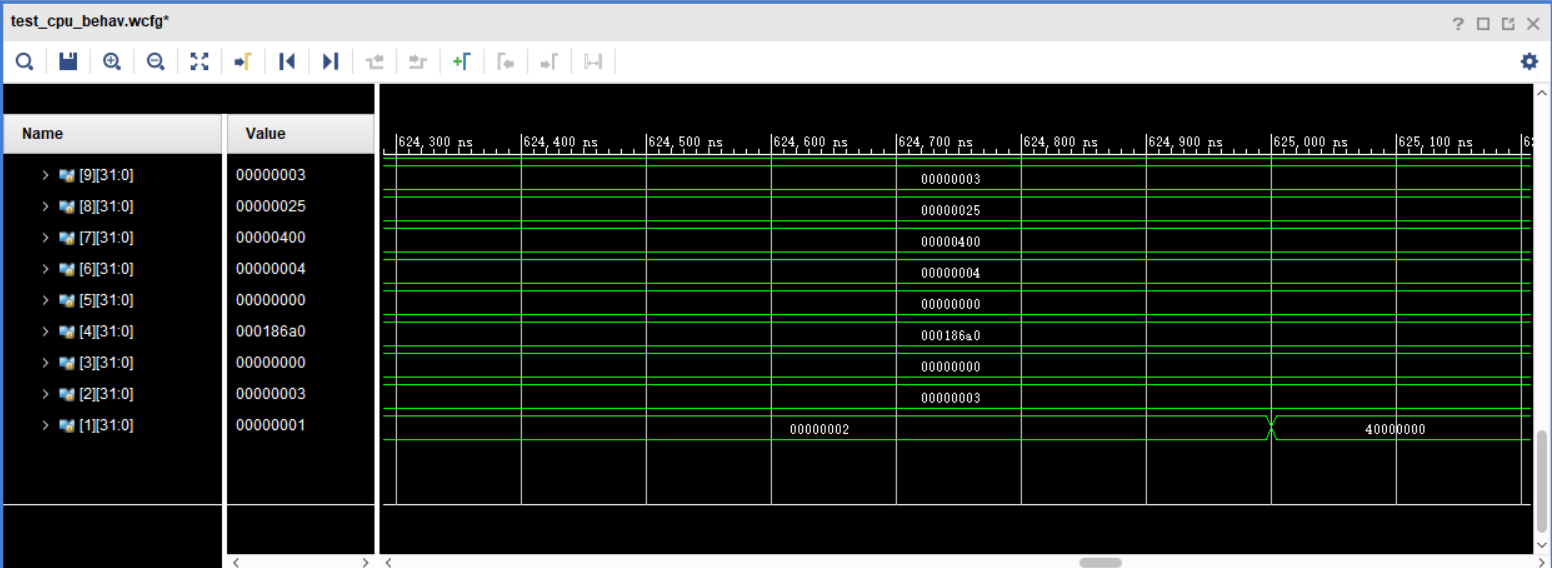
constraints:

·top.xdc

**四、仿真结果及分析**

本实验需要解决的字符串搜索问题，测试数据为“linux is not unix is not unix is not unix”，模式串为“unix”，结果应为3。

仿真结果如下：



可以看到，最终寄存器$v0的值为3。

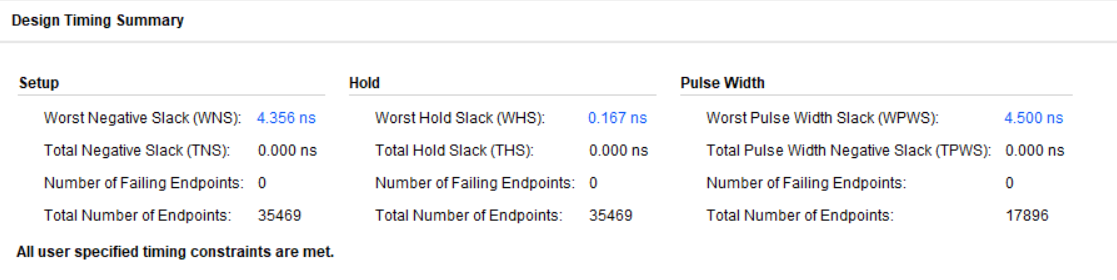


数码管显示为0003，仿真结果正确。

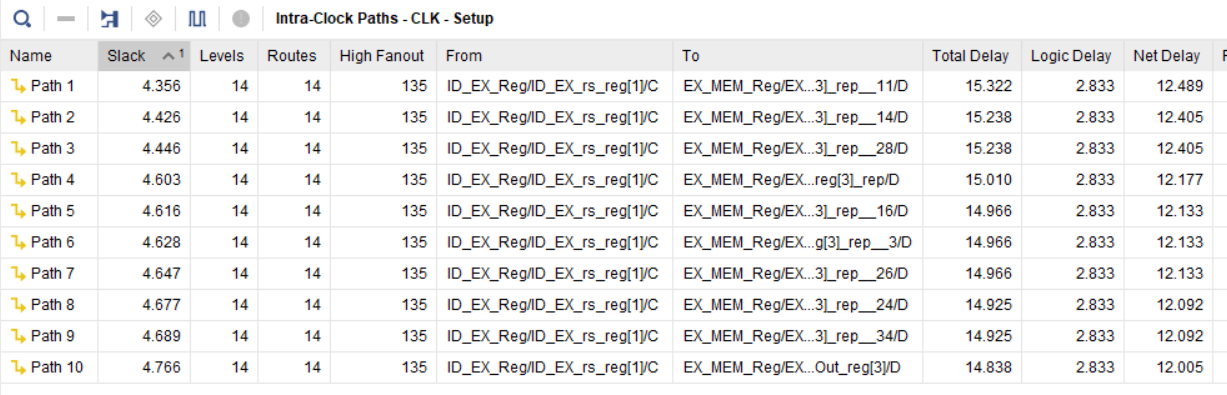
**五、综合情况**

（一）时序性能

虚拟时钟周期设为20ns，观察implementation之后的时序裕量为4.356ns。



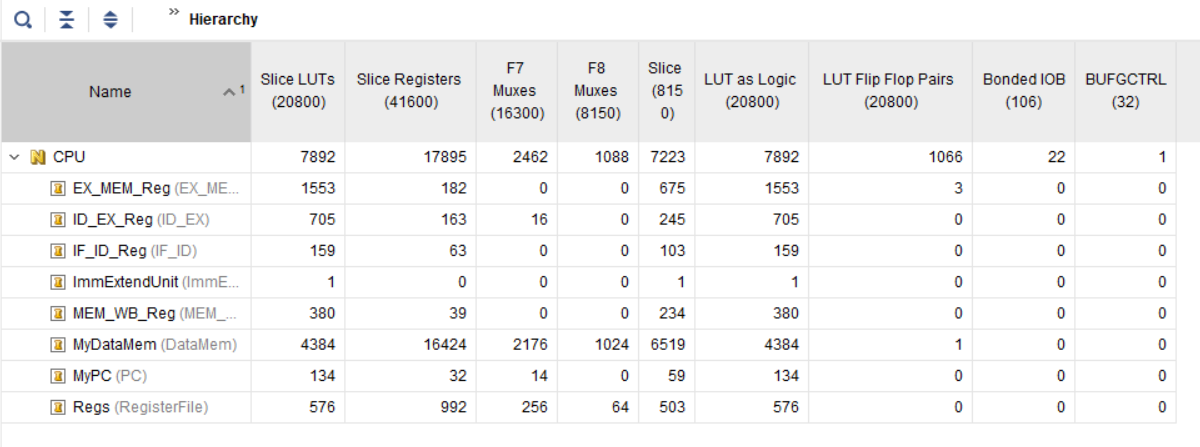
时钟周期为15.644ns，时钟频率为63.9MHz。

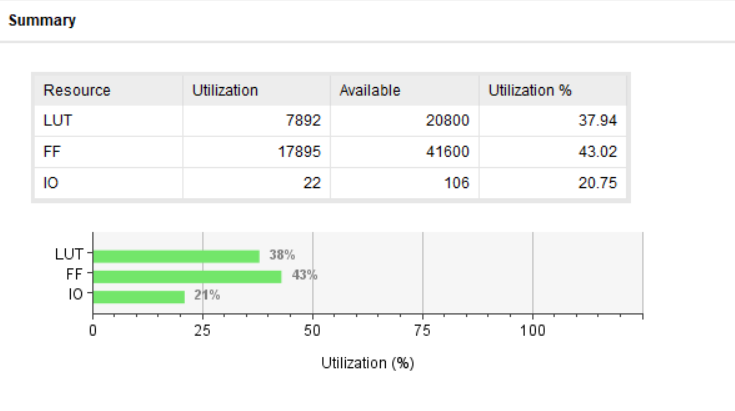


可以看到关键路径是从ID\_EX\_Reg到EX\_MEM\_Reg的路径。

（二）逻辑资源占用情况

处理器的逻辑资源占用情况如下：





**六、硬件调试情况**

一开始我在综合之后会报这个warning：[Synth 8-3295] tying undriven pin ControlUnit:OpCode[5] to constant 0。虽然不影响前仿，但是会影响后仿和上板结果。后来我发现是由于我使用了类似ID\_EX\_Reg.Instruction的结构，可能导致布线出现问题。于是我将其模块的自身属性改为output，就可以正常运行了。

另外，用指令控制数码管显示时，需要注意控制数字切换频率不能太高，否则可能看不清。

**七、实验小结**

通过本次实验，我对流水线有了更深的理解。由于我在写代码之前先对整个流水线结构进行了规划，画了数据通路图，因此在实现的过程中比较顺利。但是还是遇到了一些细节上的问题，这些在后续仿真验证的过程中也都成功解决了。这次实验是我使用仿真debug最多的一次，用起来非常方便，帮助我找到了许多小错误。

不过，这次试验也有一些不足之处。比如，处理器的频率较低，没有进行更深入的优化；分支冒险未在ID阶段判断，造成一定时间损耗。

总而言之，本次实验还是让我有很多收获的。最后感谢老师在这一学期的倾情讲授，感谢助教的悉心指导，没有老师和助教的帮助，我也难以得到这么多的收获。