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

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

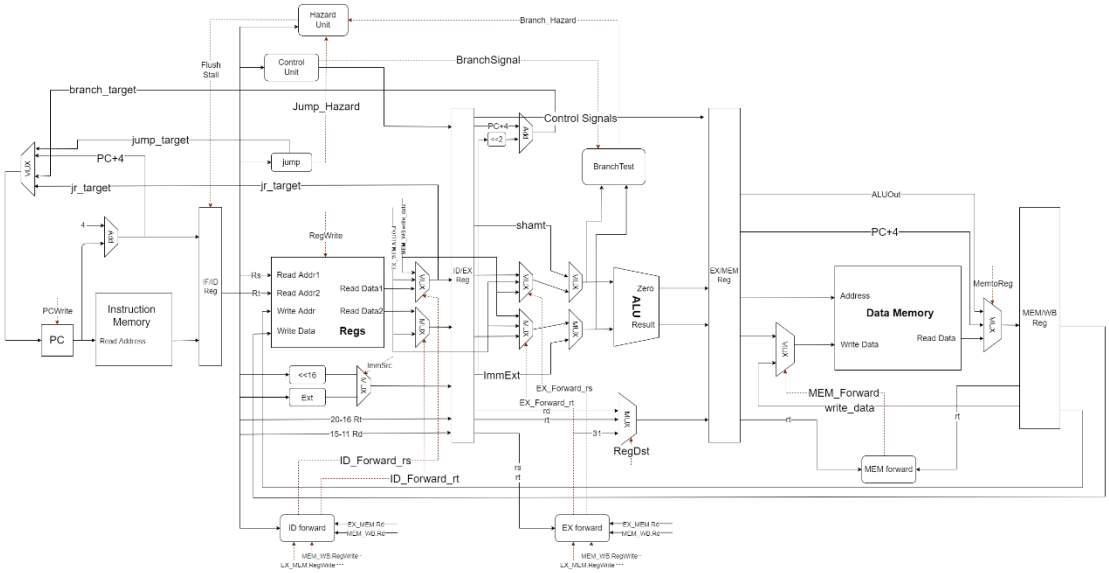
1、R 型指令：add、addu、sub、subu、and、or、xor、nor、slt、sltu、sll、srl、sra、jr、jalr；

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

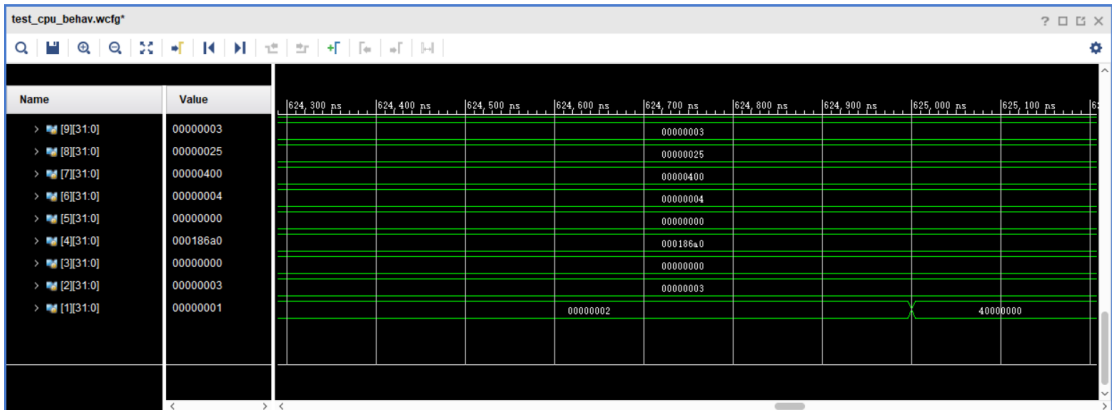
constraints:

- top.xdc

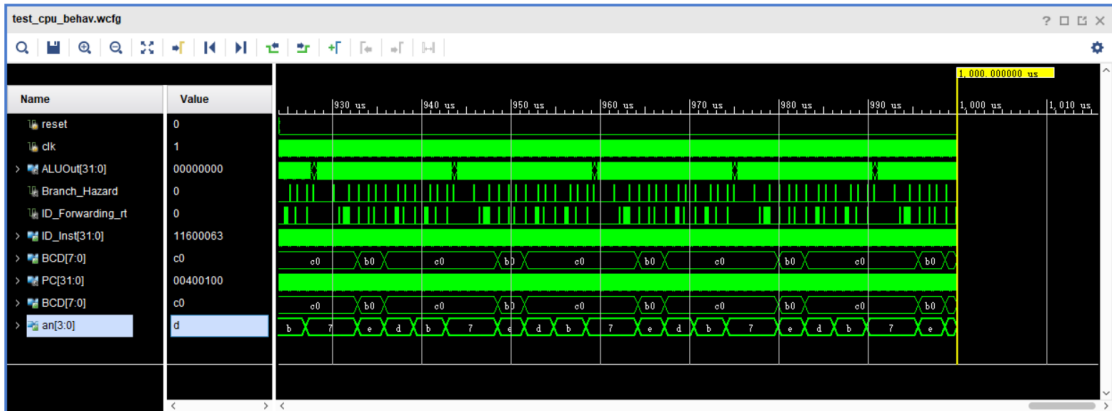
四、仿真结果及分析

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。

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.356 ns	Worst Hold Slack (WHS):	0.167 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	35469	Total Number of Endpoints:	35469	Total Number of Endpoints:	17896

All user specified timing constraints are met.

时钟周期为 15.644ns，时钟频率为 63.9MHz。

Intra-Clock Paths - CLK - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	4.356	14	14	135	ID_EX_Reg/ID_EX_rs_reg[1]/C	EX_MEM_Reg/EX...3]_rep__11/D	15.322	2.833	12.489
Path 2	4.426	14	14	135	ID_EX_Reg/ID_EX_rs_reg[1]/C	EX_MEM_Reg/EX...3]_rep__14/D	15.238	2.833	12.405
Path 3	4.446	14	14	135	ID_EX_Reg/ID_EX_rs_reg[1]/C	EX_MEM_Reg/EX...3]_rep__28/D	15.238	2.833	12.405
Path 4	4.603	14	14	135	ID_EX_Reg/ID_EX_rs_reg[1]/C	EX_MEM_Reg/EX...reg[3]_rep/D	15.010	2.833	12.177
Path 5	4.616	14	14	135	ID_EX_Reg/ID_EX_rs_reg[1]/C	EX_MEM_Reg/EX...3]_rep__16/D	14.966	2.833	12.133
Path 6	4.628	14	14	135	ID_EX_Reg/ID_EX_rs_reg[1]/C	EX_MEM_Reg/EX...g[3]_rep__3/D	14.966	2.833	12.133
Path 7	4.647	14	14	135	ID_EX_Reg/ID_EX_rs_reg[1]/C	EX_MEM_Reg/EX...3]_rep__26/D	14.966	2.833	12.133
Path 8	4.677	14	14	135	ID_EX_Reg/ID_EX_rs_reg[1]/C	EX_MEM_Reg/EX...3]_rep__24/D	14.925	2.833	12.092
Path 9	4.689	14	14	135	ID_EX_Reg/ID_EX_rs_reg[1]/C	EX_MEM_Reg/EX...3]_rep__34/D	14.925	2.833	12.092
Path 10	4.766	14	14	135	ID_EX_Reg/ID_EX_rs_reg[1]/C	EX_MEM_Reg/EX...Out_reg[3]/D	14.838	2.833	12.005

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

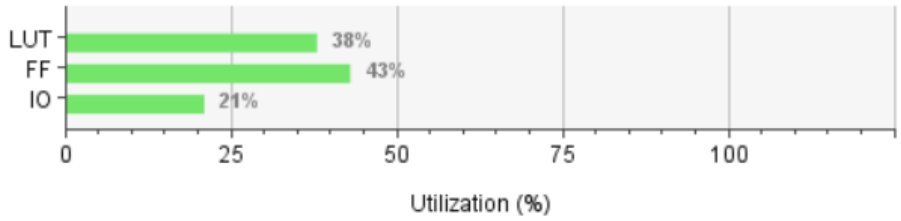
处理器的逻辑资源占用情况如下：

Hierarchy

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (106)	BUFGCTRL (32)
CPU	7892	17895	2462	1088	7223	7892	1066	22	1
EX_MEM_Reg (EX_ME...	1553	182	0	0	675	1553	3	0	0
ID_EX_Reg (ID_EX)	705	163	16	0	245	705	0	0	0
IF_ID_Reg (IF_ID)	159	63	0	0	103	159	0	0	0
ImmExtendUnit (ImmE...	1	0	0	0	1	1	0	0	0
MEM_WB_Reg (MEM_...	380	39	0	0	234	380	0	0	0
MyDataMem (DataMem)	4384	16424	2176	1024	6519	4384	1	0	0
MyPC (PC)	134	32	14	0	59	134	0	0	0
Regs (RegisterFile)	576	992	256	64	503	576	0	0	0

Summary

Resource	Utilization	Available	Utilization %
LUT	7892	20800	37.94
FF	17895	41600	43.02
IO	22	106	20.75



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。