数字逻辑与处理器基础实验
32 位 MIPS 流水线处理器设计实验报告

无 77　廖庄天予　 2017011213

2019 年 9 月 2 日

1 实验目的

1. 了解流水线处理器的基本组成、工作原理；
2. 掌握流水线处理器的设计和实现方法。

2 设计方案

在春季学期完成的单周期 CPU 基础上加以改进，从而完成流水线 CPU 的设计。

2.1 指令扩充

本次设计中实现的指令集主要如下：

- 空指令 nop：0x00000000
- 存储访问指令：lw，sw，lui
- 算术指令：add，addu，sub，subu，addi，addiu
- 逻辑指令：and，or，xor，nor，andi，sll，srl，sra，slt，slti，sltiu
- 分支指令：beq，bne，blez，bgtz，bltz
- 跳转指令：j，jal，jr，jalr

单周期处理器中除了分支指令只实现了 beq 之外，已实现其他所有指令，因此首先将分支指令 bne，blez，bgtz，bltz 完成硬件上的实现。这些指令的格式如表 1所示。

指令	OpCode[5:0]	rs[4:0]	rt[4:0]	rd[4:0]	shamt[4:0]	funct[4:0]
bne rs, rt, label	0x05	rs	rt	offset		
blez rs, label	0x06	rs	0	offset		
bgtz rs, label	0x07	rs	0	offset		
bltz rs, label	0x01	rs	0	offset		

表 1: 部分分支指令格式

每条指令使用与 beq 相似的控制信号，在 EX 阶段的分支判断中，只要有其中一种分支的条件满足，则判断为进行分支。

```
1  //control.v
2  assign Branch_ne = (OpCode == 6'h05)? 1:0;
3  assign Branch_lez = (OpCode == 6'h06)? 1:0;
4  assign Branch_gtz = (OpCode == 6'h07)? 1:0;
5  assign Branch_ltz = (OpCode == 6'h01)? 1:0;
```

```
6    //EX.v
7    assign oBranchJudge = ((iControlSignal[3] && zero) || //beq
8                   (iControlSignal[4] && ~zero) || //bne
9                   (iControlSignal[5] && ~gtz) || //blez
10                  (iControlSignal[6] && gtz) || //gtz
11                  (iControlSignal[7] && ltz))? 1'b1:0; //bltz
```

2.2 完成流水线设计

在实现过程中，将 IF/ID，ID/EX，EX/MEM，MEM/WB 阶段的寄存器依次加入每个对应阶段之间，起到暂存流水线每一级数据的作用；加入数据转发单元和冒险探测单元，解决数据冒险和控制冒险问题；加入简单的中断和异常处理电路，实现中断和未定义指令异常的处理。完成以上设计后，流水线 CPU 的核心部分即基本成型，其核心结构如图 1所示。

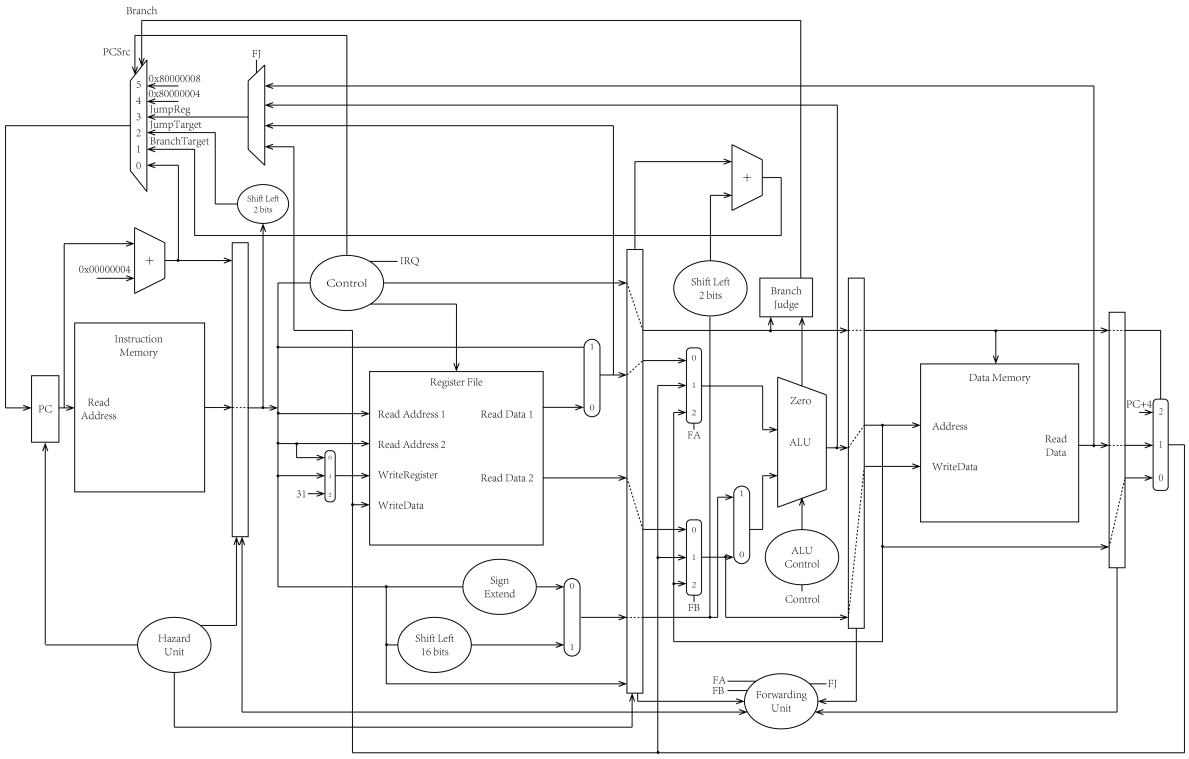

图 1: 示意图

2.3 外设设计

外设主要由定时器、LEDs、七段数码管、系统时钟计数器组成。定时器主要用于定时产生中断信号，LEDs 用于显示排序数据的地址以及指令数显示的高位部分[1]，七段数码管用于显示排序

[1]由于四位数码管无法完全显示排序指令数，使用了 4 个 LED 表示指令数 17-20 位。

数据及指令数，系统时钟计数器用于对排序过程总指令数进行计数。

3 关键代码

3.1 PC 选择器

正常执行时，PC 选择器输出为 PC+4；条件分支发生时，输出为分支目标地址；j，jal 指令直接跳转时，输出为指令的目标地址；jr、jalr 进行寄存器跳转时，输出为来自 JumpRegMux 输出的寄存器内（或转发所得）地址；中断和异常发生时，分别跳到 0x80000004，0x80000008 进入处理例程。

```
assign oPC_next = (iPCSrc == 3'b011)? 32'h80000008:
       (iPCSrc == 3'b100)? iBranchTarget:
       (iPCSrc == 3'b001)? iJumpTgt:
       (iPCSrc == 3'b010)? iJumpReg:
       (iPCSrc == 3'b101)? 32'h80000004:
iPC_plus_4;
```

同时 PC 最高位 PC[31] 作为监督位，用于标记内核态、普通态。只有重置、中断、异常可以将 PC[31] 设置为 1；PC+4，分支和 j，jal 语句应当保证 PC[31] 不变；执行 jr、jalr 指令时，PC[31] 取决于跳转地址的最高位。

```
//IF.v
assign oPC_plus_4 = {PC[31], PC[30:0] + 31'd4}; //PC+4
//EX.v
assign oBranchTarget = {iPC_plus_4[31], (iPC_plus_4[30:0] + {13'b0, iOffset,
    2'b00})};//Branch
//PipelineCore.v
assign PCMUX_iJumpTgt = {ID0.iPC_plus_4[31:28],ID0.iInstruction[25:0],2'b00}; //j, jal
```

3.2 数据转发单元

数据转发单元采取完全的 Forwarding 电路解决 EX 阶段和 jr，jalr 的数据关联问题。如果数据转发单元发现 EX 阶段需要使用还在 MEM、WB 阶段，需要写入但还没有写入寄存器堆的数据，写入寄存器相同并且不是 $0，则发出进行转发的信号，2 代表从 MEM 阶段转发，1 代表从 WB 阶段转发，0 代表不进行转发。对于 jr，jalr 转发源选择器 JumpRegMux，3 代表从 WB 转发，2 代表从 MEM 转发，1 代表从 ID 转发，0 代表不转发。

```
if(EXMEM_RegWrite == 1'b1 && // Register writing enabled
EXMEM_RegWriteAddr != 5'b0 && // Not writing $0
EXMEM_RegWriteAddr == IDEX_Rs // Use the same register immediately in EX
)
```

```
    ForwardA = 2'b10;
    else if(MEMWB_RegWrite == 1'b1 && // Register writing enabled
    MEMWB_RegWriteAddr != 5'b0 && // Not writing $0
    MEMWB_RegWriteAddr == IDEX_Rs // Use the same register immediately in EX
    )
    ForwardA = 2'b01;
    else
    ForwardA = 2'b00;
```

EX 阶段和 JumpRegMux 根据控制信号决定是否使用转发数据。

```
    //EX.v
    assign FA = (iForwardA == 2'b10)? iEXMEM_forward_data:
         (iForwardA == 2'b01)? iMEMWB_forward_data: iDatabusA;
    assign FB = (iForwardB == 2'b10)? iEXMEM_forward_data:
         (iForwardB == 2'b01)? iMEMWB_forward_data: iDatabusB;
    //JumpRegMux.v
    assign oJumpReg = (iEXForwardJ == 2'b00)? iIDRegReadData:
         (iEXForwardJ == 2'b01)? iEXALUResult:
         (iEXForwardJ == 2'b10)? iMEMReadData: iWBRegWriteData;
```

3.3 冒险检测单元

冒险检测单元用于解决 Load-use 竞争、分支和跳转的控制冒险。对于 Load-use 类竞争，采用阻塞一个周期 +Forwarding 的方法解决；分支指令在 EX 阶段判断，分支发生时刻取消 ID 和 IF 阶段的两条指令；J 类指令在 ID 阶段判断，并取消 IF 阶段指令。

在冒险检测单元中，当 Load-use、分支、跳转都不发生时，才允许 PC 和 IF/ID 寄存器按原顺序写入指令；当其中任一冲突发生时，对应的 IF/ID 和 ID/EX 寄存器需要 flush。

```
    assign PCWrite = PCWrite_request[0] & PCWrite_request[1] & PCWrite_request[2];
    assign IFID_write = IFID_write_request[0] & IFID_write_request[1] &
        IFID_write_request[2];
    assign IFID_flush = IFID_flush_request[0] | IFID_flush_request[1] |
        IFID_flush_request[2];
    assign IDEX_flush = IDEX_flush_request[0] | IDEX_flush_request[1] |
        IDEX_flush_request[2];
```

Load-use 竞争

发生 Load-use 竞争时，EX 阶段指令需从存储读取数据至寄存器，而 ID 阶段恰好使用 EX 阶段存入的寄存器。此时 ID 阶段应插入一条空指令，同时 PC-4。

```
    if(IDEX_MemRead == 1'b1 &&
```

```
    (IDEX_Rt == IFID_Rs || IDEX_Rt == IFID_Rt))
    begin
      PCWrite_request[0] = 1'b0;
      IFID_write_request[0] = 1'b0;
      IDEX_flush_request[0] = 1'b1;
      IFID_flush_request[0] = 1'b0;
    end
```

分支发生

当 EX 阶段为分支相关指令，并且分支判断为真时，应将 IF/ID、ID/EX 寄存器 flush，同时 PC-8。

```
  if(IDEX_PCSrc == 3'b100 && EX_need_branch == 1'b1)
    begin
      PCWrite_request[2] = 1'b1;
      IFID_write_request[2] = 1'b1;
      IDEX_flush_request[2] = 1'b1;
      IFID_flush_request[2] = 1'b1;
    end
  else
```

跳转发生

在 ID 阶段检测到 J 型指令时，应将 IF/ID 寄存器 flush，同时 PC-4。

```
  if(ID_PCSrc == 3'b001 || ID_PCSrc == 3'b010 || ID_PCSrc == 3'b011 || ID_PCSrc ==
     3'b101)
    begin
      PCWrite_request[1] = 1'b1;
      IFID_write_request[1] = 1'b1;
      IDEX_flush_request[1] = 1'b0;
      IFID_flush_request[1] = 1'b1;
    end
  else
```

PC 变化

因上述冒险造成的 PC 变化在 IF/ID 寄存器实现。

```
  if(iIFID_flush) begin
    if(iIDEX_flush)
      oPC_plus_4 <= (iPC_plus_4[30:0] == 31'b0)? iPC_plus_4:
```

```
4         (iPC_plus_4[30:0] == 31'h4)? iPC_plus_4 - 4: iPC_plus_4 - 8;
5       else
6         oPC_plus_4 <= (iPC_plus_4[30:0] == 31'b0)? iPC_plus_4: iPC_plus_4 - 4;
7       oInstruction <= 32'h0;
8     end
```

4 文件清单

流水线 CPU 组成文件如表 2所示。

5 仿真结果

5.1 数据转发单元测试

首先使用以下汇编代码对数据转发单元的功能进行测试。

```
1   addi $a0, $0, 1
2   addi $a1, $0, 1
3   add $a0, $a0, $a1
4   add $a1, $a0, $a1
5   add $a0, $a0, $a1
6   add $a1, $a0, $a1
```

结果如图 2所示。可以看出数据转发单元成功转发了每一条需要转发的指令的数据，得到了正确的结果。

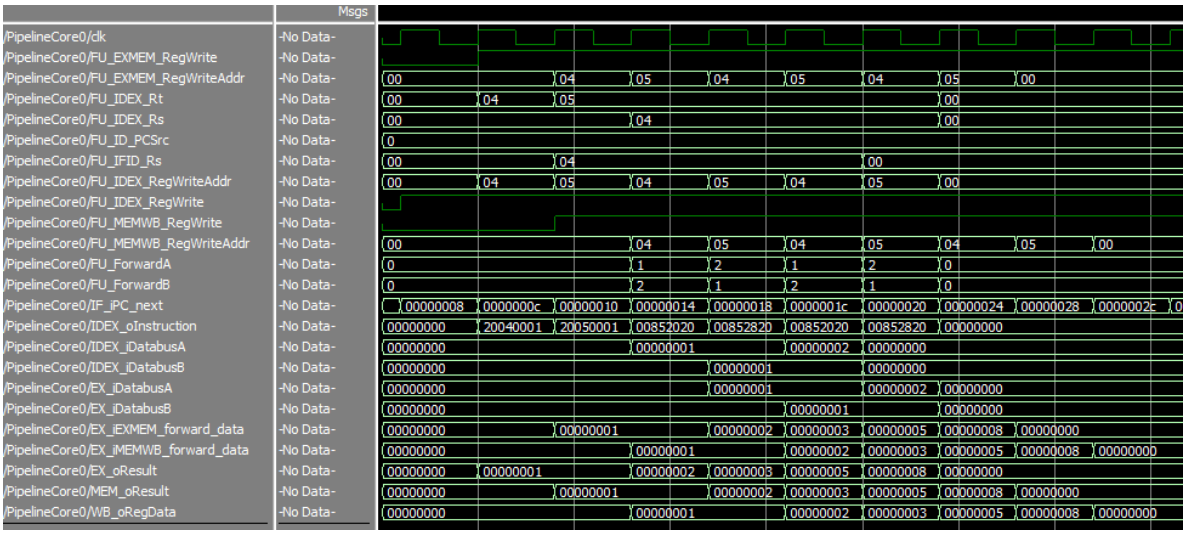

图 2: 数据转发单元测试

文件	描述
Peripheral/BCD7.v	外设七段数码管显示
Peripheral/c_clk.v	驱动 CPU 工作的时钟
Peripheral/debounce.v	按键消抖模块
Peripheral/scan_clk.v	七段数码管扫描时钟
Peripheral/sysclk_counter.v	CPU 时钟计数器
Peripheral/timer.v	中断计时器
Peripheral/unit_clk.v	频率较慢的 CPU 时钟，用于硬件调试
ALU.v	ALU 模块，完成算术和逻辑运算
ALUControl.v	ALU 控制模块，控制 ALU 进行的运算类型
Control.v	控制单元模块
DataMemory.v	数据内存模块
EX.v	流水线 EX 阶段
EXMEM_reg.v	流水线 EX/MEM 寄存器
ForwardUnit.v	数据转发单元模块
HazardUnit.v	冒险检测单元模块
ID.v	流水线 ID 阶段
IDEX_reg.v	流水线 ID/EX 寄存器
IF.v	流水线 IF 阶段
IFID_reg.v	流水线 IF/ID 寄存器
InstructionMemory.v	指令存储模块
JumpRegMux.v	jr，jalr 指令转发源选择器
MEM.v	流水线 MEM 模块
MEMWB_reg.v	流水线 MEM/WB 寄存器
PCMUX.v	PC 选择器
PipelineCore.v	流水线 CPU 核心部分
PipelineCore_tb.v	流水线 CPU 顶层、测试
RegisterFile.v	寄存器堆
WB.v	流水线 WB 阶段

表 2: 文件清单

5.2 冒险检测单元测试

使用以下汇编代码对冒险检测单元的功能键进行测试。

```
1  main: lw $a0, 0($0) #a0=0x00000132
2       add $a1, $a0, $0
3       addi $a2, $a0, 1
4  tag: beq $a1, $a2, bottom
5       addi $a1, $a1, 1
6       j tag
7  bottom:
```

结果如图 3所示。可以看出 load-use、跳转、分支均在冒险检测单元的控制下正常完成，得到了正确的结果。

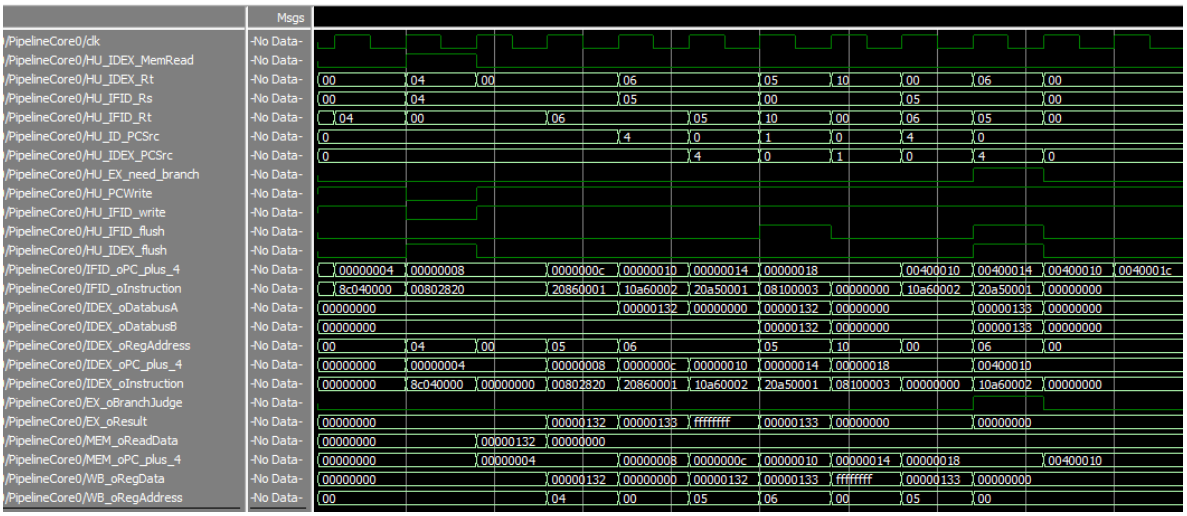

图 3: 冒险检测单元测试

5.3 排序程序测试

使用对 100 个数排序的汇编程序进行仿真测试。排序采用冒泡排序，100 个数提前写在数据存储内，详细汇编代码见附件。

排序前、排序后的数据如图 4和图 5所示。从系统时钟计数器可知，完成上述排序共需 0x00014652 条指令，即 83538 条指令。再通过图 6统计出排序汇编代码的总指令书为 67699 条，则可算得 CPI 为

$$CPI = 83538 \div 67699 \approx 1.234 \tag{1}$$

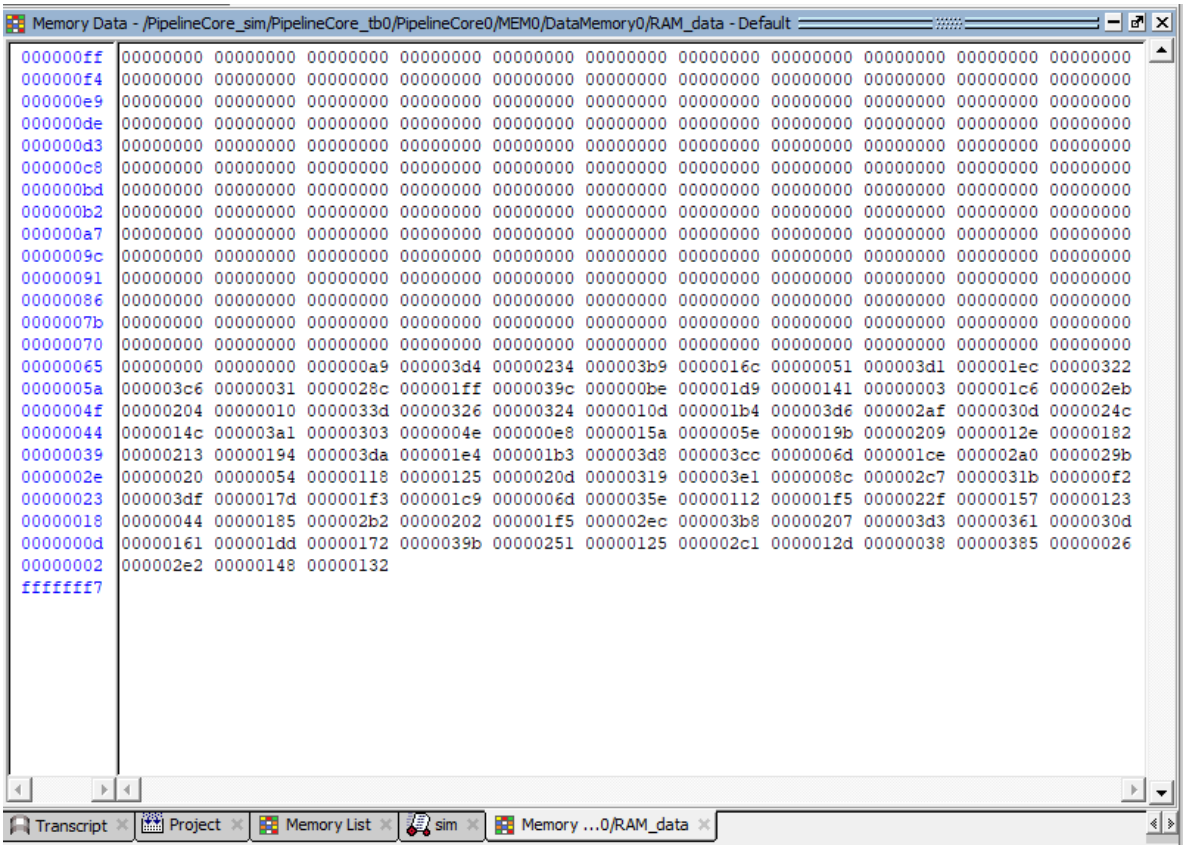
Memory Data - /PipelineCore_sim/PipelineCore_tb0/PipelineCore0/MEM0/DataMemory0/RAM_data - Default
000000ff 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000000f4 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000000e9 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000000de 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000000d3 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000000c8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000000bd 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000000b2 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000000a7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000009c 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000091 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000086 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000007b 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000070 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000065 00000000 00000000 000000a9 000003d4 00000234 000003b9 0000016c 00000051 000003d1 000001ec 00000322
0000005a 000003c6 00000031 0000028c 000001ff 0000039c 000000be 000001d9 00000141 00000003 000001c6 000002eb
0000004f 00000204 00000010 0000033d 00000326 00000324 0000010d 000001b4 000003d6 000002af 0000030d 0000024c
00000044 0000014c 000003a1 00000303 0000004e 000000e8 0000015a 0000005e 0000019b 00000209 0000012e 00000182
00000039 00000213 00000194 000003da 000001e4 000001b3 000003d8 000003cc 0000006d 000001ce 000002a0 0000029b
0000002e 00000020 00000054 00000118 00000125 0000020d 00000319 000003e1 0000008c 000002c7 0000031b 000000f2
00000023 000003df 0000017d 000001f3 000001c9 0000006d 0000035e 00000112 000001f5 0000022f 00000157 00000123
00000018 00000044 00000185 000002b2 00000202 000001f5 000002ec 000003b8 00000207 000003d3 00000361 0000030d
0000000d 00000161 000001dd 00000172 0000039b 00000251 00000125 000002c1 0000012d 00000038 00000385 00000026
00000002 000002e2 00000148 00000132
ffffffff
Transcript
Project
Memory List
sim
Memory ...0/RAM_data

图 4: 排序前数据

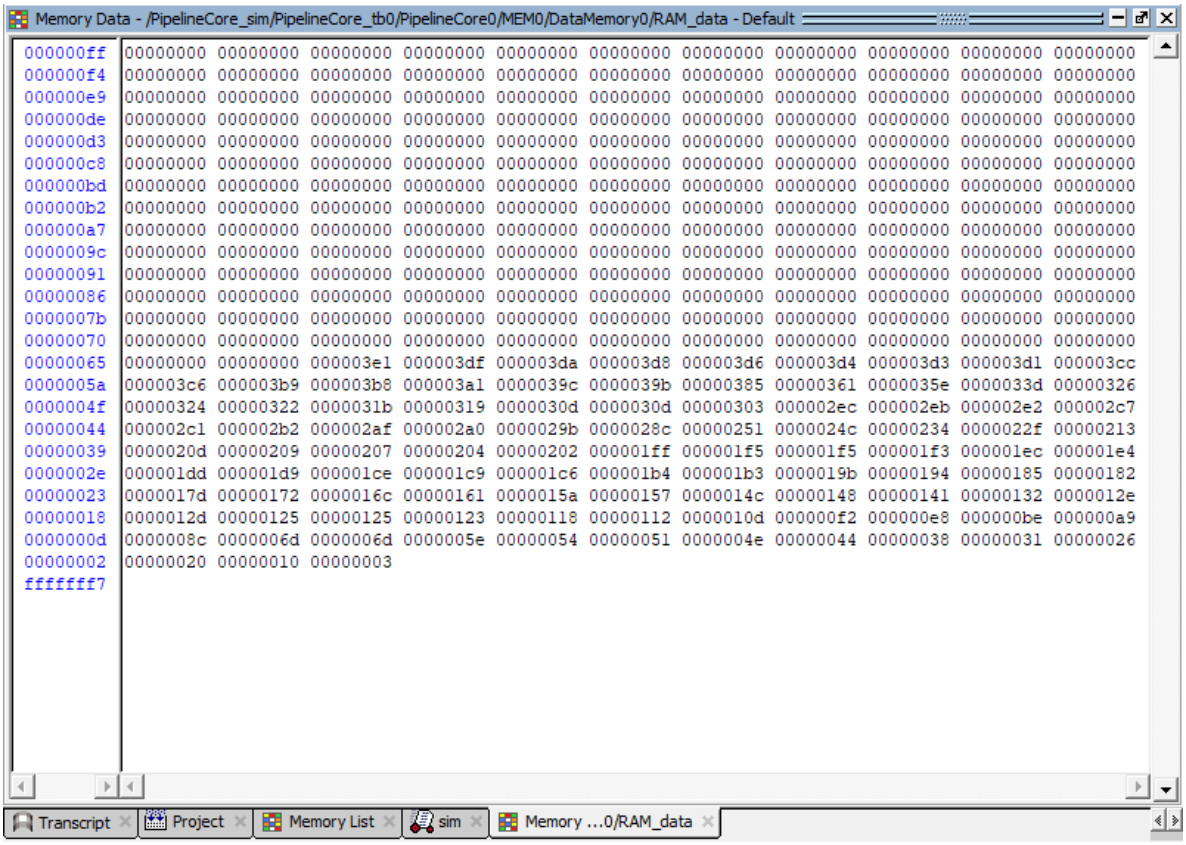

图 5: 排序后数据

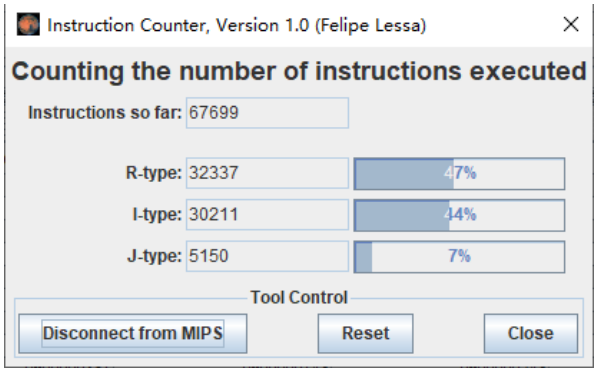

图 6: 汇编程序总指令数

6 综合情况

6.1 时序性能

通过 Vivado 的 Implementation 中的时序报告来评价流水线 CPU 的性能。为了使其达到可能的最大主频，Vivado 的 Synthesis 采用 Flow_PerfOptimized_high 策略，Implementation 采用 Performance_ExploreWithRemap 策略。同时为了比较流水线的性能提升，使用同样的标准对春季学期单周期 CPU 进行分析。

对于单周期 CPU，当时钟设定为周期 18.000ns 时，Setup 对应时序余量为 0.023ns，如图 7所示，近似认为达到最大主频，此时主频为 55.56MHz。

对于流水线 CPU，当时钟周期设定为 12.600ns 时，Setup 对应时序余量为 0.024ns，Hold 对应时序余量为 0.017ns，如图 8所示，可近似认为达到最大主频，此时主频为 79.37MHz。

由上比较可以看出，流水线的引入将主频提高了约 42.9%，由此可见流水线对于提高效率起到了显著作用。

结合式 1的计算结果，可以算出流水线 CPU 最快情况下每秒能执行指令约 64.32 万条，而单周期最快每秒能执行 55.56 万条，通过流水线优化，CPU 每秒执行指令数提高了大概 15.8%。

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.023 ns	Worst Hold Slack (WHS):	0.384 ns	Worst Pulse Width Slack (WPWS):	8.500 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:	17378	Total Number of Endpoints:	17378	Total Number of Endpoints:	8707

All user specified timing constraints are met.

图 7: 单周期时序分析

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.024 ns	Worst Hold Slack (WHS):	0.017 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:	14288	Total Number of Endpoints:	14288	Total Number of Endpoints:	9894

All user specified timing constraints are met.

图 8: 流水线时序分析

6.2 面积性能

单周期和流水线 CPU 的 FPGA 单元占用情况如表 3和表 4所示，可以看出流水线 CPU 比单周期占用更多的 FPGA 单元。

Name	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Slice	LUT as Logic
CPU	3944	8706	1281	547	3499	3944
ALU	44	0	0	0	42	44
DataMemory	2220	8192	1088	543	3380	2220
RegisterFile	1685	480	193	4	539	1685

表 3: 单周期面积性能

Name	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Slice	LUT as Logic
PipelineCore	14687	9891	1728	832	4593	14687
EX	21	0	0	0	33	21
EXMEM_reg	9071	326	0	0	2691	9071
ID	821	992	128	64	596	821
IDEX_reg	648	182	0	0	246	648
IF	71	32	0	0	57	71
IFID_reg	76	63	0	0	46	76
JumpRegMUX	62	0	0	0	40	62
MEM	2713	8192	1056	512	3540	2713
MEMWB_reg	40	104	0	0	60	40
PCMUX	64	0	0	0	21	64
WB	32	0	0	0	24	32

表 4: 流水线面积性能

7 思想体会

流水线 CPU 或许是夏季学期的任务中最让人头疼的，尤其在今年首次要求单人完成的情况下，更令我无时无刻不感受到压力。说实话，在完成实验之前，我即使已经结束了数字逻辑与处理器理论课的学习，也感觉没有对流水线结构有足够清晰的认识。在压力之下，我埋着头一遍又一遍翻着课件，找着资料，从最开始的毫无头绪，到逐渐完成框架，再到一遍遍痛苦的 debug，最终终于完成了流水线 CPU 的设计。不过也正是在这样高强度的学习和实践下，我对流水线的理解逐渐清晰并更加深刻，这样也算是对为期一学期的理论课和实验课学习的一个交代了。