中山大学本科生实验报告

（2017 学年秋季学期）

课程名称：**数字 IC 设计**　　任课教师：**王军**　　助教：张恒

年级&班级	2016 级通信工程 2 班	专业（方向）	通信工程
学号	16308149	姓名	张年崧
电话	15625061748	Email	2033203466@qq.com
开始日期	2018/05/12	完成日期	2018/05/21

一、实验目的

1. 根据老师给出的设计完成 cpu_part.v 设计；
2. 实现 operation set 中剩余的操作；
3. 实现 CF 进位标志位的操作；
4. 在 5 个空余操作编码 10011, 10100, 10101, 10110, 10111 中实现自定义操作；
5. 设计 test pattern 进行测试。

二、实验内容

1. 使用 Vivado 完成给出的设计；
2. 根据五级流水线的结构分别完成每一级的设计；
3. 补充 ALU 设计；
4. 补全 operation set 剩余的操作；
5. 在空余编码上实现自定义的操作；
6. 自行设计 test pattern 进行测试

三、实验原理

1. 实验原理图

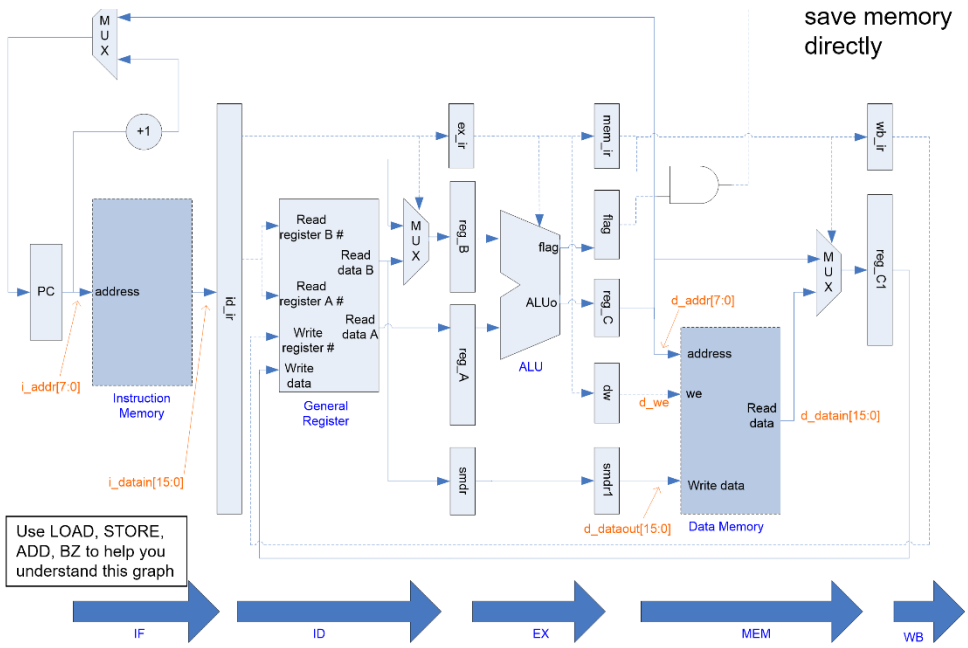

Figure 1. Block Diagram of Processer

如上图所示是一个类 MIPs 结构的五级流水线处理器的原理图。五级流水线分别为 IF, ID, EX, MEM, WB; 在设计过程中主要按照控制模块和流水线的每一级分别设计，进行数据流描述。

(1) CPU Control Logic

CPU 控制模块是一个状态机，分别有 execution 和 idle 两个状态，如下图所示。

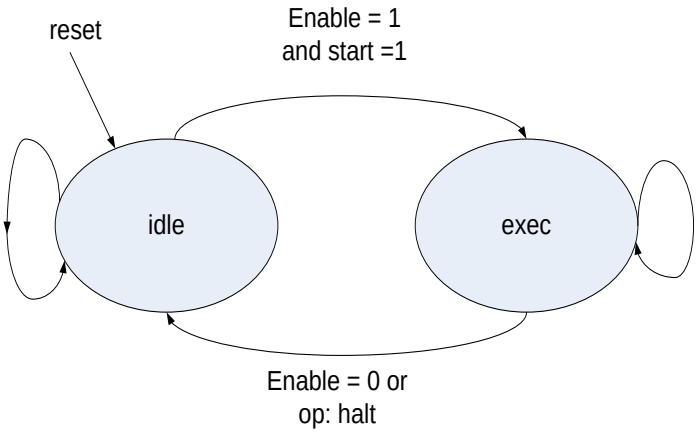

Figure 2. CPU Control Logic State Machine

状态机的输入有 Enable 和 Start, Reset. 另外影响状态机的命令还有停机指令 Halt.

(2) IF: Instruction Fetch

IF 是一个取指令的过程，指令存在 Instruction Memory，取出之后给 id_ir 赋值。

指令的地址由 IP (Instruction Pointer) 提供，IP 控制指令的执行顺序。IP 的值受跳转类命令影响，在正常情况下每个时钟周期加一。

(3) ID: Instruction Decoding

这个过程给寄存器 ex_ir, reg_A, reg_B, smdr 赋值。本过程涉及到的寄存器及其含义分别为：

Reg_A: 存放第一个操作数，这个操作数只可来自通用寄存器；

Reg_B: 存放第二个操作数，可来自通用寄存器或立即数；

Ex_ir: Excecution Instruction Register, 为执行过程存放指令；

Id_ir: Instruction Decoding Register, 来自上级，为指令译码存放指令；

Smdr: Data for Save Memory Directly, 执行 STORE 命令时候用来存放将要存入 Data Memory 的数据。

(4) EX: Execution

在这一过程中给寄存器 mem_ir, flag, reg_c, dw, smdr1 寄存器赋值。涉及到的寄存器及其含义分别为：

Mem_ir: 为下一级过程存放指令；

Flag: 包含 CF, NF, ZF 三个标志寄存器，分别为进位标志，负数标志，零标志；

Reg_C: 用来存放 ALU 的运算结果；

Dw: data_write 信号寄存器，用来控制数据写入的开关

Smdr1: smdr 寄存器的下一级，存放等待写入内存的数据。

这一过程中需要自行设计的元件是 ALU，即 Arithmetic Logic Unit, 它用来完成算术运算与逻辑运算，并且运算结果保存在 reg_C，结果影响标志位。

(5) MEM: Memory

这一阶段将需要写入内存的数据写入 data memory, dw 寄存器存放的信号为写入开关，smdr1 存放需要写入的数据，reg_c 提供写入地址。

这一阶段同时也负责从 data memory 读出数据。地址同样由 reg_C 提供，读出的数据被送往 Reg_C1. Reg_C1 中存放上一级 reg_C 中的数据（ALU 运算结果）或者读出的数据，存哪一个由 mem_ir 存放的命令控制。

(6) WB: Write Back

这一阶段负责将 reg_C1 中的内容和 wb_ir 中的指令传回通用寄存器。

2. Top View

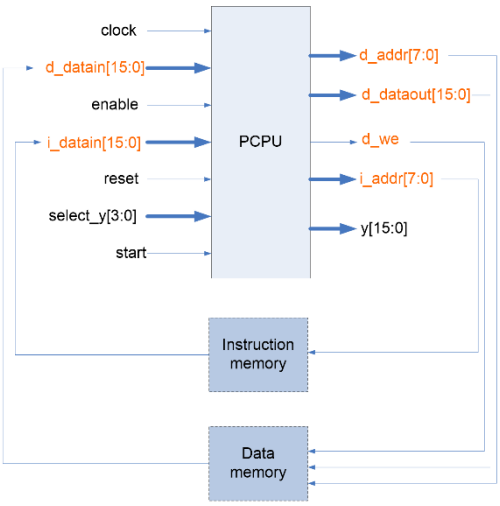

Figure 3. Top View of Processer

如图所示是设计的 Top View，我们可以将设计分成 PCPU 和指令与数据内存三个部分。其中我们主要设计实现的是 PCPU 部分，指令寄存器和数据寄存器使用 IP 核生成配置。在本次实验中主要集中于 PCPU 的设计，因此 Instruction Memory 与 Data Memory 与 PCPU 的数据输入是通过直接赋值生成。

3. Operation Set

Mnemonic	Operand1	Operand2	Operand3	Op. Code	Operation
NOP				00000	No operation
HALT				00001	Halt
LOAD	R1	R2	Val3	00010	gr[r1]<=m[r2+val3]
STORE	R1	R2	Val3	00011	m[r2+val3]<-r1
LDIH	R1	Val2	Val3	10000	r1<=r1+{val2, val3, 0000_0000}
ADD	R1	R2	R3	01000	r1<=r2+r3
ADDI	R1	Val2	Val3	01001	r1<=r1+{val2, val3}
ADDC	R1	R2	R3	10001	r1<=r2+r3+CF
SUB	R1	R2	R3	01010	r1<=r2-r3
SUBI	R1	Val2	Val3	01011	r1<=r1-{val2, val3}
SUBC	R1	R2	R3	10010	r1<=r2-r3-CF
CMP		R2	R3	01100	r2-r3; set CF,ZF and NF
AND	R1	R2	R3	01101	r1<=r2 and r3
OR	R1	R2	R3	01110	r1<=r2 or r3
XOR	R1	R2	R3	01111	r1<=r2 xor r3
SLL	R1	R2	Val3	00100	r1<=r2 shift left logical (val3 bit shift)
SRL	R1	R2	Val3	00101	r1<=r2 shift right logical (val3 bit shift)
SLA	R1	R2	Val3	00101	r1<=r2 shift left arith. (val3 bit shift)
SRA	R1	R2	Val3	11000	r1<=r2 shift left arith. (val3 bit shift)
JUMP		Val2	Val3	11000	jump to {val2, val3}
JMPR	R1	Val2	Val3	11001	jump to r1+{val2, val3}
BZ	R1	Val2	Val3	11010	if ZF=1 branch to r1+{val2, val3}
BNZ	R1	Val2	Val3	11011	if ZF=0 branch to r1+{val2, val3}
BN	R1	Val2	Val3	11100	if NF=1 branch to r1+{val2, val3}
BNN	R1	Val2	Val3	11101	if NF=0 branch to r1+{val2, val3}
BC	R1	Val2	Val3	11110	if CF=1 branch to r1+{val2, val3}

BNC	R1	Val2	Val3	11111	if CF=0 branch to r1+{val2, val3}
NAND	R1	R2	R3	10011	r1<=r2 nand r3
INC	R1	R2		10100	r1<=r2 +1
DEC	R1	R2		10101	r1<=r2 -1
NOT	R1	R2	R3	10110	r1<=~r2
INV	R1	R2		10111	r1<= -r2

4. 内存实现

为了完成一个完整的 CPU，指令内存和数据内存是必不可少的。内存的实现方法有很多种，可以用 IP 核行配置，也可以用寄存器实现。这里采用的是用寄存器实现的方法。设计中数据内存和指令内存的数据线都是 8 位，所以分别使用 256 个寄存器来实现内存的功能。

注：本次实验由于并没有设计 Hazard 防止，所以在复杂指令集中添加了许多 NOP，有可能造成指令数超过 256 条的情况。因此，在实际实现中 instruction memory 的地址总线加宽了 1 为，拓展为 256 位。这种改变是十分灵活的，只需要修改地址线的宽度和指令寄存器的数目即可，因此与原设计不发生冲突。

四、实现过程

1. 宏定义

为了简化代码，增加易读性和方便编写，将命令对应的二进制编码写成宏的形式。对于老师提供的 37 个命令和自己实现的 5 个命令的操作数，将对应的操作时定义为 5 位二进制数，写成宏的形式。特别的，为了简化 CPU 每个流程中的赋值条件语句，将操作分为几个分类。有一些操作在同一个阶段所要求给某一变量的赋值是相同的，有一些操作在某一个阶段对操作码进行的操作是相同的，有一些操作在同一个阶段下一步的流程是相同的，因此按照这些标准进行分类。将操作分为一下几类，并分别写成宏定义的形式：

R2_R3：这些命令的操作数都是寄存器；

R2_V2V3：这些命令的操作数是寄存器和两个立即数；

R2_V3：这些命令的操作数是寄存器和一个立即数；

BRANCHJUMP：这些命令是跳转指令；

SETFLAG：这些命令影响标志位；

WB_ENABLE：这些命令要求写回。

这些宏定义的命令可以更方便地判断每个阶段在什么情况下进行某一特定的操作，同时也使添加操作更加便捷。

2. IF 阶段

主要增加跳转指令的实现。

3. ID 阶段

当执行跳转指令时，ex_ir 要清零；然后分别对 reg_A, reg_B, smdr 进行赋值。首先对 reg_A 赋值。当操作数有 R1 和两个立即数的时候，或者命令为 LDIH 时候（由于 LDIH 的操作数还要补低位的 8 个 0，所以没有列入 R1_V2V3）让 reg_A 读入第一个操作数，或者在命令为 LOAD 比较 id_ir[10:8]与 mem_ir[10:8]，满足时候读入 d_datain；当操作数为两个寄存器或者一个寄存器和一个立即数时，让 reg_A 读入第一个操作数，或者在命令为 LOAD 比较 id_ir[6:4]与 mem_ir[10:8]，满足时候读入 d_datain；在执行跳转指令的时候 reg_A 清零。Reg_B 的赋值方法与 reg_A 类似。

Smdr 一般直接读入寄存器中的数据，其地址由命令的第一个操作数给出。当命令为 LOAD 时，直接读入 d-datain.

4. EX 阶段

主要添加 ALU 和设置 CF 标志位。ALU 执行运算和逻辑操作，使用 CASE 语句实现，对应不同命令的操作码来执行不同的算术逻辑操作；CF 标志位由运算过程产生，根据运算结果设置标志位，另外两个标志由开头设置的宏命令控制，同样根据运算结果设置。

5. MEM 阶段

LOAD 命令时 reg_C1 直接读入 d_datain，其余时候接收 reg_C 的数据，wb_ir 接受来自 mem_ir 的命令。

6. WB 阶段

由前面设置的宏定义控制写回，将 reg_C1 的内容传回通用寄存器，地址由命令的第一个操作数给出。

五、仿真结果

1. pcpu 测试结果

Pcpu 是 processor 除了数据和指令内存以外的部分，是处理器的核心，所有的运算和五级流水线都在这里执行。仿真过程首先使用了老师上课 PPT 给出的测试代码测试 LOAD, NOP, ADD, STORE, HALT 指令，运行结果如下：

```
pc:     id_ir      :regA:regB:regC:da: dd::w:regC1:gr1 :gr2 :gr3
xx:xxxxxxxxxxxxxxxx:xxxx:xxxx:xxxx:xx:xxxx:x:xxxx:xxxx:xxxx:xxxx
00:0000000000000000:xxxx:xxxx:0000:00:0000:0:0000:0000:0000:0000
01:0001000100000000:0000:0000:xxxx:xx:0000:0:0000:0000:0000:0000
02:0001001000000001:0000:0000:0000:00:0000:0:xxxx:0000:0000:0000
03:0000000000000000:0000:0001:0000:00:0000:0:0000:0000:0000:0000
04:0000000000000000:0000:0000:0001:01:0000:0:00ab:0000:0000:0000
05:0000000000000000:0000:0000:0000:00:0000:0:3c00:00ab:0000:0000
06:0100001100010010:0000:0000:0000:00:0000:0:0000:00ab:3c00:0000
07:0000000000000000:00ab:3c00:0000:00:0000:0:0000:00ab:3c00:0000
08:0000000000000000:0000:0000:3cab:ab:0000:0:0000:00ab:3c00:0000
09:0000000000000000:0000:0000:0000:00:0000:0:3cab:00ab:3c00:0000
0a:0001101100000010:0000:0000:0000:00:0000:0:0000:00ab:3c00:3cab
0b:0000100000000000:0000:0002:0000:00:0000:0:0000:00ab:3c00:3cab
0c:0000100000000000:0000:0000:0002:02:3cab:1:0000:00ab:3c00:3cab
0d:0000100000000000:0000:0000:0000:00:3cab:0:0002:00ab:3c00:3cab
0e:0000100000000000:0000:0000:0000:00:3cab:0:0000:00ab:3c00:3cab
0f:0000100000000000:0000:0000:0000:00:3cab:0:0000:00ab:3c00:3cab
```

Fig 4. PPT 中测试代码验证

由测试结果对比标准答案，可以看出这里额测试结果是正确的。Pcpu 测试的第二个部分是自己定义 texture simulation，这里

```
pc:     id_ir      :regA:regB:regC:da: dd::w:regC1:gr1 :gr2 :gr3 :gr4 :gr5 :gr6 :gr7 :gr8
xx:xxxxxxxxxxxxxxxx:xxxx:xxxx:xxxx:xx:xxxx:x:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx
00:0000000000000000:xxxx:xxxx:0000:00:0000:0:0000:0000:0000:0000:0000:0000:0000:0000:0000
01:0001000100000000:xxxx:xxxx:xxxx:xx:xxxx:0:0000:0000:0000:0000:0000:0000:0000:0000:0000
02:0000000000000000:0000:0000:xxxx:xx:0000:0:xxxx:0000:0000:0000:0000:0000:0000:0000:0000
03:0000000000000000:0000:0000:0000:00:0000:0:xxxx:0000:0000:0000:0000:0000:0000:0000:0000
04:0000000000000000:0000:0000:0000:00:0000:0:00ab:0000:0000:0000:0000:0000:0000:0000:0000
05:0001001100000010:0000:0000:0000:00:0000:0:0000:0000:00ab:0000:0000:0000:0000:0000:0000
06:0000000000000000:0000:0002:0000:00:0000:0:0000:0000:00ab:0000:0000:0000:0000:0000:0000
07:0001010100000100:0000:0002:0002:02:0000:0:0000:0000:00ab:0000:0000:0000:0000:0000:0000
08:0000000000000000:0000:0004:0002:02:0000:0:3c00:0000:00ab:0000:0000:0000:0000:0000:0000
09:0000000000000000:0000:0004:0004:04:0000:0:0002:0000:00ab:0000:3c00:0000:0000:0000:0000
```

Fig 5. 自定义测试代码验证结果（部分）

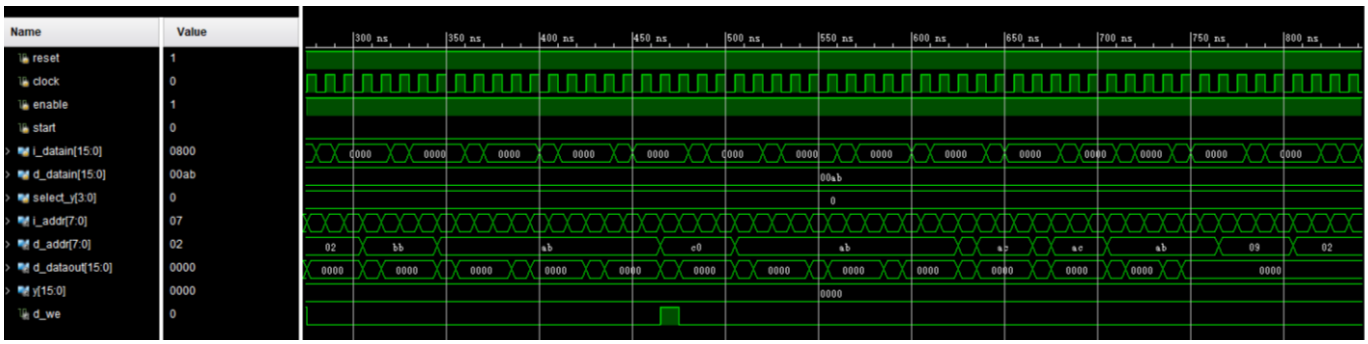

Fig 6. 自定义测试代码波形图（部分）

自定义测试代码验证主要验证了除了 LOAD, ADD, STORE, HALT 指令之外的，后来实现的指令。经过验证我们可以确定后来实现代码的正确性。

2. processor 测试结果

Processor 测试是验证的主要部分。Processor 是指 pcpu + data memory + instruction memory，全部实现之后的处理器系统。内存我们采用寄存器的方法实现，并且实现时候就给内存中初始化变量。这里采用的测试命令和数据是添加了 Nop 指令之后的复杂指令测试集 instr_test. 值得注意的一点是由于没有实现 Hazard，所以要在合适的地方添加 Nop。测试的结果作为文件输出，输出文件为 dmem.mem.master，默认路径为 D 盘根目录。

```
Add:00000000, Cell:fffd  Add:00000017, Cell:ebebAdd:00000031, Cell:xxxx
Add:00000001, Cell:0004  Add:00000018, Cell:aaaaAdd:00000032, Cell:xxxx
Add:00000002, Cell:0005  Add:00000019, Cell:c369Add:00000033, Cell:xxxx
Add:00000003, Cell:c369  Add:0000001a, Cell:86d2Add:00000034, Cell:xxxx
Add:00000004, Cell:69c3  Add:0000001b, Cell:3690Add:00000035, Cell:xxxx
Add:00000005, Cell:0041  Add:0000001c, Cell:8000Add:00000036, Cell:xxxx
Add:00000006, Cell:ffff  Add:0000001d, Cell:c369Add:00000037, Cell:xxxx
Add:00000007, Cell:0001  Add:0000001e, Cell:61b4Add:00000038, Cell:0030
Add:00000008, Cell:xxxx  Add:0000001f, Cell:00c3Add:00000039, Cell:xxxx
Add:00000009, Cell:xxxx  Add:00000020, Cell:0001Add:0000003a, Cell:xxxx
Add:0000000a, Cell:xxxx  Add:00000021, Cell:c369Add:0000003b, Cell:xxxx
Add:0000000b, Cell:xxxx  Add:00000022, Cell:86d2Add:0000003c, Cell:xxxx
Add:0000000c, Cell:xxxx  Add:00000023, Cell:6900Add:0000003d, Cell:xxxx
Add:0000000d, Cell:xxxx  Add:00000024, Cell:8000Add:0000003e, Cell:xxxx
Add:0000000e, Cell:xxxx  Add:00000025, Cell:69c3Add:0000003f, Cell:xxxx
Add:0000000f, Cell:xxxx  Add:00000026, Cell:d386Add:00000040, Cell:xxxx
Add:00000010, Cell:b600  Add:00000027, Cell:c300Add:00000041, Cell:xxxx
Add:00000011, Cell:0001  Add:00000028, Cell:8000Add:00000042, Cell:xxxx
Add:00000012, Cell:0005  Add:00000029, Cell:c369Add:00000043, Cell:xxxx
Add:00000013, Cell:0001  Add:0000002a, Cell:e1b4Add:00000044, Cell:xxxx
Add:00000014, Cell:ffff  Add:0000002b, Cell:ffc3Add:00000045, Cell:xxxx
Add:00000015, Cell:fffe  Add:0000002c, Cell:ffffAdd:00000046, Cell:xxxx
Add:00000016, Cell:4141  Add:0000002d, Cell:69c3Add:00000047, Cell:xxxx
                         Add:0000002e, Cell:34e1Add:00000048, Cell:xxxx
                         Add:0000002f, Cell:0069Add:00000049, Cell:xxxx
                         Add:00000030, Cell:0000
```

Fig 6. 复杂指令测试结果

上图为复杂指令测试的结果，Add 为 data memory 的地址，Cell 为其中的数据内容。这里结果是所有的有值的输出，省略了一部分 xxx 的值。由结果与标准答案对比可知，输出的结果是正确的。

六、RTL 原理图

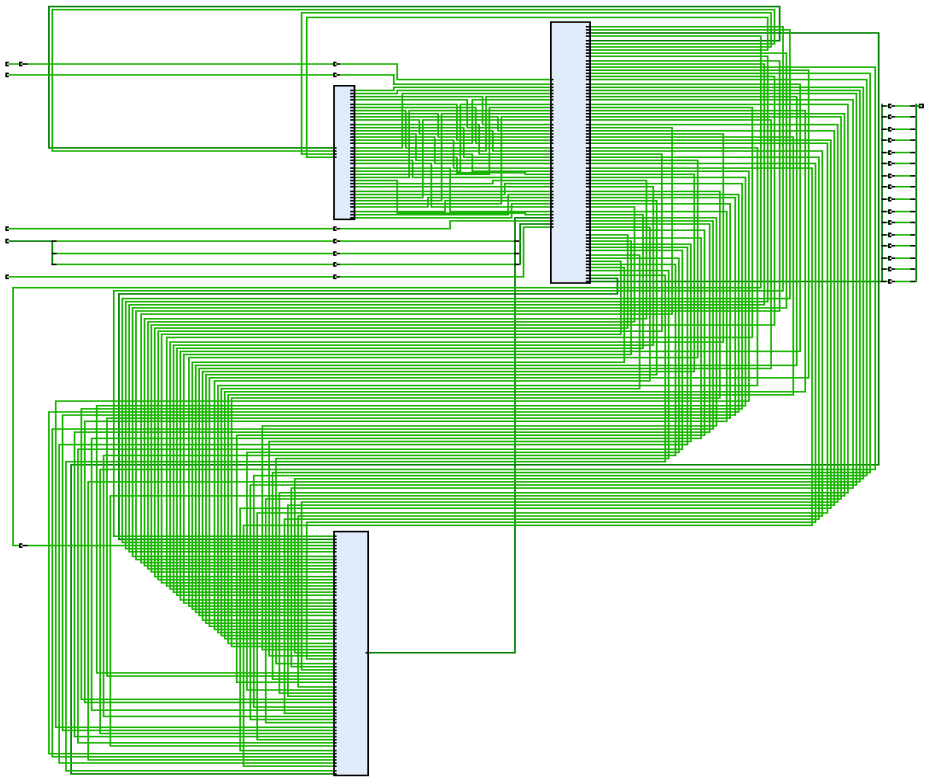

Fig 7. Synthesize 之后得到的原理图

如图所示是 Synthesize 之后得到的原理图，这里的原理图指的是 RTL 图。图中两个比较长的模块是 data memory

和 instruction memory，比较小的模块是 pcpu.

七、时序报告

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.646 ns	Worst Hold Slack (WHS):	0.154 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:	590	Total Number of Endpoints:	590	Total Number of Endpoints:	282

All user specified timing constraints are met.

Fig 8. Design Timing Summary

七、实验感想

这次实验是我第一次动手写一个完整的较大型的系统。通过这次实验，我对 CPU 的工作方式有了一个全面的把握和系统的了解，同时通过自己动手的方式验证了自己的想法。这次实验中我最大的收获是对 CPU 流水线的工作方式有了较为细致的了解，以及 CPU 中数据是传送的，指令是如何执行的，每一级流水线是如何相互配合的。在 CPU 的实际编写中我也碰到了一些问题，并进行了自己的思考，得到了以下几点感想：

1. 我发现逐条添加指令非常繁琐，并且直接在每一级流水线的判断条件修改会非常臃肿并且不好控制。之后我学会了使用宏定义的方法来将指令分类，如下一步操作相同的指令分为一类，都要影响标志位的指令分到一类，等等。

2. pcpu 的测试花去了大量时间。助教给出的复杂指令测试集有一下两点问题：首先所有的两条指令之间都添加了三个 Nop 语句，占去了大量的空间，因此指令超过 256 条，不得不拓展地址线；后来我找出了不必要的 Nop 并且删除，最终控制语句数目在 256 条以内；第二点，由于直接加入 Nop 语句，导致原来 JUMP 和 BRANCH 语句的地址与实际地址不符合，会出现死循环的情况。因此又逐条改正跳转语句的地址。这些问题是在测试中发现的，最终将指令更改或删减得以解决。

3. 在决定 NOP 是否必须时，我总结的方法是判断后一条命令是否需要用到前一条命令的结果，或者寄存器的使用是否造成冲突，这样在不必要的时候即可不需要加入 Nop。

4 .测试的时候发现有时地址 0 － 7 的数据也变了，后来发现是跳转语句的跳转地址有错误，改正之后解决。

5 .提供的测试文件中 Vivado 输出文件的语句并不正确，后来添加了$fclose 命令和绝对地址才使文件得以正确输出。另外学会了定义 task 的写法。

6. 有时输出不完整，是因为仿真没有执行到 finish 命令，可以将仿真延长至 100000ns, 确保仿真执行到结束步骤。

7. 老师在 PPT 中讲的算术左移要求保留符号位，但从助教给出的测试标准答案来看应该是不保留符号位的，这一点是在仿真结果 debug 时发现的。

8. 查看频率的方法是，添加（时序）约束文件，例如内容是 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clock]，“clock”改成代码里时钟的名字，这句话表示 100MHz 时钟，然后综合，综合以后查看时序报告 Vivado － Synthesis － Report Timing Summary，弹出对话框直接点 OK，然后如果没有出现红色，就表示这个频率下可以正常工作，可以把频率调高，再综合。