

计算机组成原理实验报告

一、CPU 设计方案综述

(一) 总体设计概述

此 CPU 为通过 Verilog 硬件描述语言实现的 FDEMW 全速转发顺序执行 5 级流水线 CPU，支持 MIPS 指令集中的{ADD, ADDI, ADDIU, ADDU, AND, ANDI, BEQ, BGEZ, BGTZ, BLEZ, BLTZ, BNE, DIV, DIVU, J, JAL, JALR, JR, LB, LBU, LH, LHU, LUI, LW, MFHI, MFLO, MTHI, MTLO, MULT, MULTU, NOR, OR, ORI, SB, SH, SLL, SLLV, SLT, SLTI, SLTIU, SLTU, SRA, SRAV, SRL, SRLV, SUB, SUBU, SW, XOR, XORI}共 50 条指令。“5 级”所指及相应部件如表 1 所示。

表 1 流水阶段及相应部件

阶段	部件
Instruction Fetch	PC, IM
Instruction Decode	NPC, GRF, EXT, CMP
Execution	ALU, MDU
Read Data from Memory Write Data to Memory	DM
Write Data Back to Register	(GRF)

实现此 CPU 时，将其分为“机制”（mechanism）与“策略”（policy）两大部分，即将数据通路相关模块与核心控制模块分开考虑，并采用模块化和层次化设计方法。顶层有效的驱动信号包括且仅包括：时钟信号 `clk` 和同步复位信号 `reset`。

(二) 数据通路相关模块实现方法

1. PC

由起始值为 `0x00003000` 的具有同步复位功能的 `reg [31:0]` 实现，复位值为起始地址。可接收 `enable` 信号，以便在 `nop` 指令时保持原值不变。

2. IM

由容量为 $32 \text{ bit} \times 4096 = 4 \text{ B} \times 4096 = 16 \text{ KB}$ 的 `reg [31:0]` 实现。在这种设计中，每个连续的地址都能取一个 `word`，故将 PC 中储存的地址右移 2 位用于在 IM 中寻址。此处暂时未将 PC、NPC 和 IM 封装在 IFU 这一大模块中；如有需要，后续修改也并不困难。

3. NPC

由计算下一条指令地址的逻辑实现。

4. GRF

由具有同步复位功能的 `reg [31:0]` 实现，复位值为 `0x00000000`。可接收 `writable` 信号，以便在非写入指令时，GRF 中的数据不能被误写。由于 0 号寄存器的值始终为 `0x00000000`，故向 0 号寄存器写入值始终无效。

5. EXT

根据控制信号，将输入的 16 位立即数进行 0 扩展、符号扩展或加载到高位为 32 位整数后输出。

6. CMP

直接连接在 GRF 后的 32 位整数比较分支信号生成器，使在该流水线设计中，b 类指令后只需插入一个 `nop` 指令就能保证正确性。若仍在 ALU 中生成 b 类指令的分支信号，则在该流水线设计中，必须在 b 类指令后插入 3 个 `nop` 指令才能保证正确性。

7. ALU

支持 32 位整数加、减、与、或、或非、异或、移位、比较等功能，不检测溢出。输出结果由 `ALUop` 信号控制。

8. MDU

支持 32 位整数乘、除和取模运算，含 HI 和 LO 两个寄存器，将乘法的低 32 位保存在 LO 寄存器中，高 32 位保存在 HI 寄存器中；将除法的商保存在 LO 寄存器中，取模的余数保存在 HI 寄存器中。出于本实验的性质，使用了*、/、% 运算符，但实际上这些运算需要更长的时钟周期，因此乘法要模拟 6 个周期，除法模拟 11 个周期。通过输出 `start` 信号来表示在该周期内启动了运算，用 `busy` 信号表示在该周期内正在模拟运算中。

9. DM

由起始地址为 `0x00000000` 的具有异步复位功能的容量为 $32\text{ bit} \times 1024 = 4\text{ B} \times 4096 = 16\text{ KB}$ 的 `reg [31:0]` 实现，复位值为 `0x00000000`。可接收 `writable` 信号，以便在非写入指令时，DM 中的数据不能被误写。在这种设计中，每个连续的地址都能取一个 word，故将地址右移 2 位来以 word 为单位寻址，辅以第 1 位来得到 half-word、第 [1:0] 位来得到 byte。

10. 流水线寄存器组

4个流水线寄存器组衔接各流水阶段，接收上一阶段部件新产生的数据，并将储存的上一阶段部件的旧数据传给下一阶段部件。指令将在这些寄存器间流动，并且传给相应阶段的控制器来产生控制信号。均可同步复位，复位值为0x00000000。其中，IF/ID级寄存器可接收writeable信号，以便在nop指令时保持原值不变。

(三) 重要机制实现方法

1. 转发

首先假设所有数据冒险均可通过转发解决。也就是说，当某一指令前进到必须使用某一流水阶段中相应寄存器的值时，这个寄存器的值一定已经产生，并存储于后续某个流水线寄存器中，只是还没来得及写入GRF。存在这种需求的部件有：

- (1) D级CMPin1和CMPin2；
- (2) D级NPC中retAddr的输入；
- (3) E级ALUin1、ALUin2、MDUin1和MDUin2；
- (4) M级DMdataIn。

为了实现转发机制，需要对这些输入前加上一个多选器。这些多选器的默认输入来源是上一级中已经转发过的数据。特别地，由于GRF既可被视为在D级中，也可以被视为在W级中，则需对GRF采用内部转发机制，即当前GRF被写入的值会即时反馈到读取端上。

由此可确定截至目前选择信号的生成规则：只要当前位点的读取寄存器地址和某转发输入来源的写入寄存器地址相等且不为0，就选择该转发输入来源；在有多个转发输入来源都满足条件时，最新产生的数据优先级最高。

2. 暂停

事实上，有些数据冒险并不能由转发处理，因为先前指令的新数据还未来得及产生，后续指令就要使用。此时只能暂停流水线，等待先前指令的新数据产生。为了方便处理，需要暂停流水线时，将指令暂停在D级。如何判断某条指令的新数据是否已经产生？需要引入需求时间—供给时间模型。

定义需求时间Tuse为：指令进入D级后，后续某个功能部件必须要使用某寄存器值时，需要经过的时钟周期数。

定义供给时间Tnew为：位于E级及其后续各级的指令，能够产生要写入某寄存器的新值时，需要经过的时钟周期数。在目前的CPU中，W级的指令Tnew恒为0；对于同一条指令，TnewM为TnewE - 1和0之间的最大值。

现在就可以得出需要暂停的条件了：D 级指令读取寄存器的地址与 E 级或 M 级的指令写入寄存器的地址相等且不为 0，且 D 级指令的 Tuse 小于对应 E 级或 M 级指令的 Tnew；或者 MDU 在 busy 或 start，且 D 级有 MDU 相关的指令。在其他情况下，数据冒险均可通过转发机制解决。

暂停时需阻止任何新数据写入 PC、阻止任何新数据写入 IF/ID 级寄存器、复位 ID/EX 级流水寄存器。

3. 控制与抽象（详见思考题）

二、测试方案

(一) 典型测试样例

对于普通运算和 b 类分支指令，除了测试指令功能，只需将冲突指令排列组合即可，此处暂略。需要注意的是，jal 指令允许在延迟槽中改变存入 GRF 中的 \$ra 值，尽管实际很少有程序会这样做。因此特意编写以下死循环程序检验正确性。

```
1  ori    $t0, $t0, 4
2  ori    $t1, $t1, 1
3  jal    label
4  sub    $ra, $ra, $t0
5  ori    $t2, $t2, 1
6  ori    $t3, $t3, 1
7  label:
8  ori    $t4, $t4, 1
9  jr     $ra
10 nop
```

(二) 自动测试程序

使用 Python 3 和 C/C++ 各编写了一个自动测试程序，两者功能基本等价，且性能经过优化，没有明显区别。为了更加方便地编写自动测试程序，需修改 MARS 使其运行程序时能输出寄存器和存储器的写入明细字符串。还需通过 Vivado 导出命令行仿真辅助文件到一个测试专用目录下，其中包括了仿真开始时需要自动加载到 IM 中的机器码文件 code.txt。修改机器码时，只需修改测试专用目录下的 code.txt 文件即可自动加载到 IM 中，而 Verilog 代码可以在原工程目录中。Verilog 代码也可被导出到该目录下以增加便携性，但考虑到可

能还会在原工程目录下修改 Verilog 代码，为避免反复修改反复导出或忘记导出浪费时间，暂时不导出 Verilog 代码进行仿真。自动测试程序与执行步骤大致如下：

- (1) 将设定的仿真所需时间写入 `cmd.tcl` 文件。
- (2) 随机生成指令程序文件。
- (3) 调用 MARS 读取该文件并将汇编生成的机器码写入 `code.txt` 文件。
- (4) 调用 MARS 运行该程序并将输出的寄存器和存储器的写入明细字符串写入文件。
- (5) 调用 Vivado 相关组件依次进行 Compile、Elaborate 和 Simulate。仿真过程中，要求 GRF 和 DM 调用系统任务 `$display` 显示在 TCL Console 中的内容会随自动生成的其它仿真信息记录在文件中。
- (6) 比较 MARS 和仿真输出。

程序示意如下。此处并没有粘贴测试所有指令的版本。

1. Python 3 自动测试程序示意

```
1  import os
2  import subprocess
3  import random
4  PAUSE_AT_FAIL      = True
5  TEST_START_NUM     = 34
6  TEST_END_NUM       = 290
7  USELESS_INFO_LEN   = 17
8  TIME_INFO_LEN      = 20
9  SIMULATION_TIME    = "10 us"
10 ASM_FILE_DIR       = "code/290"
11 MARS_DIR            = "MARS_290.jar"
12 VIVADO_DIR          = "C:/Xilinx/Vivado/2020.1/bin/"
13 AK = True
14 def gen_code(file_name):
15     with open(file_name, "w") as f:
16         for _ in range(5):
17             rd = random.randint(1, 27)
18             imm = random.randint(0, 3070) * 4
19             f.write(f"ori\t${rd}, {imm}\n")
```

```

20     for _ in range(5):
21         target = random.randint(1, 27)
22         source = random.randint(0, 27)
23         imm = random.randint(0, 7) * 4
24         f.write(f"sw\t${target}, {imm}({source})\n")
25         f.write(f"lw\t${target}, {imm}({source})\n")
26     for _ in range(5):
27         target = random.randint(1, 27)
28         source = random.randint(0, 27)
29         rd = random.randint(0, 27)
30         f.write(f"addu\t${target}, ${source}, ${rd}\n")
31     for _ in range(5):
32         target = random.randint(1, 27)
33         source = random.randint(0, 27)
34         rd = random.randint(0, 27)
35         f.write(f"subu\t${target}, ${source}, ${rd}\n")
36     for _ in range(5):
37         rd = random.randint(1, 27)
38         imm = random.randint(0, 0xffff)
39         f.write(f"lui\t${rd}, {imm}\n")
40     f.write("nop\nnop\n")
41 if __name__ == '__main__':
42     print("Autotest started...")
43     os.chdir(os.path.dirname(__file__))
44     with open("cmd.tcl", "w") as f:
45         f.write(f"run {SIMULATION_TIME};\nquit;")
46     print("Initialization succeeded!")
47     for i in range(TEST_START_NUM, TEST_END_NUM + 1):
48         print(f"Testing case {i}...")
49         asm_file_name = f"{ASM_FILE_DIR}/testpoint{i}.asm"
50         master_name = f"master/master_{i}.txt"
51         query_name = f"query/query_{i}.txt"
52         # gen_code(asm_file_name)

```

```

53         subprocess.call(f"java -
jar {MARS_DIR} {asm_file_name} nc mc CompactDataAtZero a dump .text
HexText code.txt")
54         master = subprocess.run(f"java -
jar {MARS_DIR} {asm_file_name} nc mc CompactDataAtZero db", shell=Fa
lse, stdout=subprocess.PIPE, encoding="utf-8").stdout.splitlines()
55         with open(master_name, "w") as f:
56             f.writelines(master)
57         os.system(VIVADO_DIR + "xvlog -prj vlog.prj --nolog")
58         os.system(VIVADO_DIR + "xelab -debug typical -
L xil_defaultlib -L unisims_ver -L unimacro_ver -L secureip --
snapshot mips_tb xil_defaultlib.mips_tb xil_defaultlib.glbl --
nolog")
59         os.system(VIVADO_DIR + "xsim mips_tb -
key {Behavioral:sim_1:Functional:mips_tb} -tclbatch cmd.tcl -
log " + query_name)
60         with open(query_name, "r") as f:
61             query = f.read()
62             line_cnt = 0
63             master_iter = iter(master)
64             for line_query in query.splitlines()[USELESS_INFO_LEN:]:
65                 line_master = next(master_iter)
66                 line_cnt += 1
67                 while "$ 0" in line_master:
68                     line_master = next(master_iter)
69                 if line_master == "" or line_master is None:
70                     print(f"Test case {i} is accepted!")
71                     break
72                 elif line_query is None:
73                     print(f"Test case {i} failed at line {line_cnt}!")
74                     print("Your outputs are fewer than expected.")
75                     AK = False
76                     break
77                 elif line_master != line_query[TIME_INFO_LEN:]:

```

```

78         print(f"Test case {i} failed at line {line_cnt}!")
79         print(f"Expected answer is {line_master}.")
80         print(f"Your answer is {line_query}.")
81         AK = False
82         break
83     if PAUSE_AT_FAIL and not AK:
84         os.system("pause")
85     if AK:
86         print("\nCongratulations!\nTest cases are all killed!")

```

2. C/C++自动测试程序示意

```

1  #include <stdio>
2  #include <stdlib>
3  #include <string>
4  #include <ctime>
5  #define PAUSE_AT_FAIL      true
6  #define TEST_BEGIN_NUM    1
7  #define TEST_END_NUM      10
8  #define FILE_NAME_LEN     32
9  #define BUFFER_LEN        512
10 #define USELESS_INFO_LEN  17
11 #define QUIT_LEN           7
12 const char simulationTime[] = "1 ms";
13 const char codeDir[] = "code/290";
14 const char MarsDir[] = "MARS_290.jar";
15 const char vivadoDir[] = "C:/Xilinx/Vivado/2020.1/bin";
16 char masterName[FILE_NAME_LEN], queryName[FILE_NAME_LEN], buffer1[BUFFER_LEN], buffer2[BUFFER_LEN];
17 bool AK = true;
18 inline int randint(int a, int b) {
19     return a + rand() % (b - a);
20 }
21 inline void genCode(char *fileName) {
22     FILE *fp = fopen(fileName, "w");

```



```

23     srand(time(nullptr));
24     int i, rd, imm, target, source;
25     for (i = 0; i < 5; i++) {
26         rd = randint(1, 27);
27         imm = randint(0, 3070) * 4;
28         fprintf(fp, "ori\t%d, %d\n", rd, imm);
29     }
30     for (i = 0; i < 5; i++) {
31         target = randint(1, 27);
32         source = randint(0, 27);
33         imm = randint(0, 7) * 4;
34         fprintf(fp, "sw\t%d, %d(%d)\n", target, imm, source);
35         fprintf(fp, "lw\t%d, %d(%d)\n", target, imm, source);
36     }
37     for (i = 0; i < 5; i++) {
38         target = randint(1, 27);
39         source = randint(0, 27);
40         rd = randint(0, 27);
41         fprintf(fp, "addu\t%d, %d, %d\n", target, source, rd);
42     }
43     for (i = 0; i < 5; i++) {
44         target = randint(1, 27);
45         source = randint(0, 27);
46         rd = randint(0, 27);
47         fprintf(fp, "subu\t%d, %d, %d\n", target, source, rd);
48     }
49     for (i = 0; i < 5; i++) {
50         rd = randint(1, 27);
51         imm = randint(0, 0xffff);
52         fprintf(fp, "lui\t%d, %d\n", rd, imm);
53     }
54     fprintf(fp, "nop\nnop\n");
55     fclose(fp);
56 }

```

```

57 inline int bufferCmp(char *buf1, char *buf2) {
58     int index1 = 0, index2 = 0;
59     while (buf1[index1] && buf1[index1] != '@') index1++;
60     while (buf2[index2] && buf2[index2] != '@') index2++;
61     return strcmp(buf1 + index1, buf2 + index2);
62 }
63 inline void fileCmp(int n) {
64     int cnt = 0;
65     FILE *master = fopen(masterName, "r");
66     FILE *query = fopen(queryName, "r");
67     for (int i = 0; i < USELESS_INFO_LEN; i++) fgets(buffer1, BUFFER
        _LEN, query);
68     while (fgets(buffer2, BUFFER_LEN, master)) {
69         if (strstr(buffer2, "$ 0")) continue;
70         cnt++;
71         if (strlen(buffer2) <= QUIT_LEN) break;
72         if (!fgets(buffer1, BUFFER_LEN, query)) {
73             printf("Test case %d failed at line %d.\n", n, cnt);
74             printf("Your outputs are fewer than expected.");
75             AK = false;
76             return;
77         } else if (bufferCmp(buffer1, buffer2)) {
78             printf("Test case %d failed at line %d.\n", n, cnt);
79             printf("Expected answer is %s\n", buffer2);
80             printf("Your answer is %s\n", buffer1);
81             AK = false;
82             return;
83         }
84     }
85     printf("Test case %d is accepted.\n", n);
86     fclose(master);
87     fclose(query);
88 }
89 int main() {

```

```

90     char asmFileName[FILE_NAME_LEN];
91     printf("Autotest started...\n");
92     FILE *fp = fopen("cmd.tcl", "w");
93     fprintf(fp, "run %s;\nexit\n", simulationTime);
94     fclose(fp);
95     printf("Initialization succeeded!\n");
96     for (int i = TEST_BEGIN_NUM; i < TEST_END_NUM; i++) {
97         printf("Testing case %d...\n", i);
98         sprintf(asmFileName, "%s/testpoint%d.asm", codeDir, i);
99         sprintf(masterName, "master/master_%d.txt", i);
100        sprintf(queryName, "query/query_%d.txt", i);
101        genCode(asmFileName);
102        sprintf(buffer1, "java -
        jar %s %s nc mc CompactDataAtZero a dump .text HexText code.mem", Ma
        rsDir, asmFileName);
103        system(buffer1);
104        sprintf(buffer1, "java -
        jar %s %s nc mc CompactDataAtZero db >%s", MarsDir, asmFileName, mas
        terName);
105        system(buffer1);
106        sprintf(buffer1, "%s/xvlog -prj vlog.prj --
        nolog", vivadoDir);
107        system(buffer1);
108        sprintf(buffer1, "%s/xelab -L xil_defaultlib -
        L unisims_ver -L unimacro_ver -L secureip --
        snapshot mips_tb xil_defaultlib.mips_tb xil_defaultlib.glbl --
        nolog", vivadoDir);
109        system(buffer1);
110        sprintf(buffer1, "%s/xsim mips_tb -
        key {Behavioral:sim_1:Functional:mips_tb} -tclbatch cmd.tcl -
        log %s", vivadoDir, queryName);
111        system(buffer1);
112        fileCmp(i);
113        if (PAUSE_AT_FAIL && !AK) system("pause");

```

```

114     }
115     if (AK) printf("Congratulations!\nTest cases are all killed!");
116     return 0;
117 }

```

三、思考题

1. 为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

因为乘除法运算较慢，放进 ALU 会拖慢整体性能。独立的 HI、LO 寄存器可以不影响乘除槽中指令的执行，还可以避免与 GRF 中寄存器混淆，降低 CPU 和程序设计的复杂度。

2. 参照你对延迟槽的理解，试解释“乘除槽”。

在乘除槽中，非乘除类指令可以在流水线中继续执行；通过编译优化，可以把无关指令放在乘除槽中，这样减少了乘除法对性能的影响。在编译器不能自动优化时，有时人工改变一些程序中数据的运算顺序（即表达式括号的位置），对性能就能有显著提升。

3. 举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint：考虑 C 语言中字符串的情况）

处理字符串。

4. 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证**覆盖**了所有需要测试的情况；如果你是**完全随机**生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了**特殊的策略**，比如构造连续数据冒险序列，请你描述一下你使用的策略如何**结合了随机性**达到强测的效果。

此思考题请同学们结合自己测试 CPU 使用的具体手段，按照自己的实际情况进行回答。

详见前文。

5. 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

控制器将每一条机器指令中的信息，解码为传输给 CPU 各部分的控制信号。实现控制逻辑需要考虑译码方式和译码风格。

译码方式上，可分为集中式译码与分布式译码。集中式译码指在 F 级或 D 级前，根据该指令将所有控制信号全部解析出，然后让其随着流水往后逐级传递。使用这种方法，只需要在新指令到来时进行一次译码，减少了后续流水级的逻辑复杂度，但流水级之间需要传递的信号数量很大。分布式译码指每一级都部署一个控制器，负责译出当前级指令所需的控制信号。这种方法较为灵活，“现译现用”有效降低了流水级间传递的信号量，但是需要实例化多个控制器，浪费硬件资源。

译码风格上，可分为指令驱动型与控制信号驱动型。指令驱动型指控制逻辑整体在一个 `case` 语句之下，通过判断指令的类型，来对所有的控制信号一一进行赋值。这种方法便于添加指令，不易遗漏控制信号，甚至可以通过控制信号表 2，编写 Python 脚本将 Excel 表格自动转化为相应 Verilog 代码。由于代码编辑器多数都具有多行光标功能，手动编写控制器代码也并不繁琐。控制信号驱动型指为每个指令定义一个 `wire` 型变量，使用或运算描述组合逻辑，对每个控制信号进行单独处理。这种方法可显著减少代码量，缺点是如错添或漏添了某条指令，很难锁定出现错误的位置。为使修改数据通路和添加指令更加直观、便捷和机械化，采用分布式译码的方式和指令驱动型的风格。

Tim Peters 的 *The Zen of Python*（节选）特别能体现本 CPU 采用的控制逻辑实现原则：

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

In the face of ambiguity, refuse the temptation to guess.

There should be one - and preferably only one - obvious way to do it.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

具体地，能由控制器进行译码并抽象的，就不用原始指令数据让多选器实现；能通过列表的形式机械地实现的，就不用条件表达式；能将控制信号定义为宏的，就不用具体数值（因此实际控制信号并不完全与表 2 中的值对应）。例如，转发和暂停机制需要指令的读取寄存器和写入寄存器的地址信息，而这些信息在控制器解码指令时就可以确定并统一抽象为 **Ause1**、**Ause2** 和 **Anew**（比如 **jal** 的 **Anew** 就固定为 **31**），因此在生成选择信号时，无需对指令分类，大大简化了控制逻辑。又例如，由于 **jal** 或 **jalr** 指令产生的数据 **PC + 8** 和 MDU 的 **HI**、**LO** 寄存器数据都并非由 ALU 得出，为了保持原有转发选择逻辑不变，将 **PC + 8** 的值、MDU 与 **ALUoutOriginal** 在 E 级由选择信号 **ALUresultOp** 控制，抽象为 **ALUout** 传给 EX/MEM 寄存器，体现了“*Special cases aren't special enough to break the rules*”的思想。

实现转发需要寄存器地址（A 信息），实现暂停需要寄存器值供需时间（T 信息），称用 A 信息和 T 信息实现转发与暂停的方法为 AT 法。“*There should be one - and preferably only one - obvious way to do it.*”在此 CPU 设计中，这大概就是指 AT 法吧。

转发与暂停的核心逻辑具体实现如下。

```

1  wire retAddrFwdOp = (Ause1D == AnewM && Ause1D != 0 && TnewM == 0) ? 1 : 0;
2  wire CMPin1fwdOp = (Ause1D == AnewM && Ause1D != 0 && TnewM == 0) ? 1 : 0;
3  wire CMPin2fwdOp = (Ause2D == AnewM && Ause2D != 0 && TnewM == 0) ? 1 : 0;
4  wire [1:0] ALUin1fwdOp = (Ause1E == AnewM && Ause1E != 0 && TnewM == 0) ? 2 :
5                          (Ause1E == AnewW && Ause1E != 0 && TnewW == 0) ? 1 :
6                          0;
7  wire [1:0] ALUin2fwdOp = (Ause2E == AnewM && Ause2E != 0 && TnewM == 0) ? 2 :
8                          (Ause2E == AnewW && Ause2E != 0 && TnewW == 0) ? 1 :
9                          0;
10 wire [31:0] ALUresult = (ALUresultOp == 0) ? ALUresultOriginal :
11                        (ALUresultOp == 1) ? PCplus4E + 4 :
12                        (ALUresultOp == 2) ? HI :
13                        (ALUresultOp == 3) ? LO :
14                        ALUresultOriginal;
15 wire stall = (Ause1D == AnewE && Ause1D != 0 && Tuse1D < TnewE) ||
16             (Ause2D == AnewE && Ause2D != 0 && Tuse2D < TnewE) ||
17             (Ause1D == AnewM && Ause1D != 0 && Tuse1D < TnewM) ||
18             (Ause2D == AnewM && Ause2D != 0 && Tuse2D < TnewM) ||
19             (MDUopD > 0 && (MDUstartE || MDUbusy));

```



流水线CPU.xlsx

表 2 控制信号

Type	Instruction	opcode	funct	Ause1	Ause2	Anew	Tuse1	Tuse2	Tnew	NPCop	GRFwr	GRFdataOp	EXTop	ALUop	ALUAop	ALUBop	DMwr	DMstoreOp	DMloadOp
B	beq	000100		rs	rt	0	0	0	0	0b01	0			0b0010	0	0	0		
B	bgez	000001		rs	rt	0	0	0	0	0b01	0			0b0101	0	0	0		
B	bgtz	000111		rs	rt	0	0	0	0	0b01	0			0b0000	0	0	0		
B	blez	000110		rs	rt	0	0	0	0	0b01	0			0b0001	0	0	0		
B	bltz	000001		rs	rt	0	0	0	0	0b01	0			0b0100	0	0	0		
B	bne	000101		rs	rt	0	0	0	0	0b01	0			0b0011	0	0	0		
I	addi	001000		rs	0	rt	1	0	2	0b00	1	0b00	0b01	0b1011	0	1	0		
I	addiu	001001		rs	0	rt	1	0	2	0b00	1	0b00	0b01	0b1011	0	1	0		
I	andi	001100		rs	0	rt	1	0	2	0b00	1	0b00	0b00	0b0111	0	1	0		
I	lb	100000		rs	0	rt	1	0	3	0b00	1	0b01	0b01	0b1011	0	1	0		0b001
I	lbu	100100		rs	0	rt	1	0	3	0b00	1	0b01	0b01	0b1011	0	1	0		0b100
I	lh	100001		rs	0	rt	1	0	3	0b00	1	0b01	0b01	0b1011	0	1	0		0b000
I	lhu	100101		rs	0	rt	1	0	3	0b00	1	0b01	0b01	0b1011	0	1	0		0b011
I	lui	001111		0	0	rt	0	0	2	0b00	1	0b00	0b10	0b1011	0	1	0		
I	lw	100011		rs	0	rt	1	0	3	0b00	1	0b01	0b01	0b1011	0	1	0		0b010
I	ori	001101		rs	0	rt	1	0	2	0b00	1	0b00	0b00	0b1000	0	1	0		
I	sb	101000		rs	rt	0	1	2	0	0b00	0		0b01	0b1011	0	1	1	0b10	0b001
I	sh	101001		rs	rt	0	1	2	0	0b00	0		0b01	0b1011	0	1	1	0b01	0b000
I	slti	001010		rs	0	rt	1	0	2	0b00	1	0b00	0b01	0b0100	0	1	0		
I	sltiu	001011		rs	0	rt	1	0	2	0b00	1	0b00	0b01	0b0110	0	1	0		
I	sw	101011		rs	rt	0	1	2	0	0b00	0		0b01	0b1011	0	1	1	0b00	0b010
I	xori	001110		rs	0	rt	1	0	2	0b00	1	0b00	0b00	0b1010	0	1	0		
J	j	000010		0	0	0	0	0	0	0b11	0						0		
J	jal	000011		0	0	31	0	0	0	0b11	1	0b10					0		
R	add	000000	100000	rs	rt	rd	1	1	2	0b00	1	0b00		0b1011	0	0	0		
R	addu	000000	100001	rs	rt	rd	1	1	2	0b00	1	0b00		0b1011	0	0	0		

R	and	000000	100100	rs	rt	rd	1	1	2	0b00	1	0b00		0b0111	0	0	0		
R	jalr	000000	001001	rs	0	rd	0	0	0	0b10	1	0b10					0		
R	jr	000000	001000	rs	0	0	0	0	0	0b10	0						0		
R	nor	000000	100111	rs	rt	rd	1	1	2	0b00	1	0b00		0b1001	0	0	0		
R	or	000000	100101	rs	rt	rd	1	1	2	0b00	1	0b00		0b1000	0	0	0		
R	sll	000000	000000	0	rt	rd	0	1	2	0b00	1	0b00		0b1101	1	0	0		
R	sllv	000000	000100	rs	rt	rd	1	1	2	0b00	1	0b00		0b1101	0	0	0		
R	slt	000000	101010	rs	rt	rd	1	1	2	0b00	1	0b00		0b0100	0	0	0		
R	sltu	000000	101011	rs	rt	rd	1	1	2	0b00	1	0b00		0b0110	0	0	0		
R	sra	000000	000011	0	rt	rd	0	1	2	0b00	1	0b00		0b1111	1	0	0		
R	srav	000000	000111	rs	rt	rd	1	1	2	0b00	1	0b00		0b1111	0	0	0		
R	srl	000000	000010	0	rt	rd	0	1	2	0b00	1	0b00		0b1110	1	0	0		
R	srlv	000000	000110	rs	rt	rd	1	1	2	0b00	1	0b00		0b1110	0	0	0		
R	sub	000000	100010	rs	rt	rd	1	1	2	0b00	1	0b00		0b1100	0	0	0		
R	subu	000000	100011	rs	rt	rd	1	1	2	0b00	1	0b00		0b1100	0	0	0		
R	xor	000000	100110	rs	rt	rd	1	1	2	0b00	1	0b00		0b1010	0	0	0		
R	nop	000000	000000	0	rt	rd	0	1	2	0b00	0						0		

Instruction	opcode	funct	Ause1	Ause2	Anew	Tuse1	Tuse2	Tnew	NPCop	ALUAop	ALUBop	ALUresultOp	GRFwr	GRFdataOp	MDUwr	MDUstart	MDUop	DMwr
mfhi	000000	010000	0	0	rd	0	0	2	NORMAL			HI	1	ALU	0	0	MFHILO	0
mflo	000000	010010	0	0	rd	0	0	2	NORMAL			LO	1	ALU	0	0	MFHILO	0
mthi	000000	010001	rs	0	0	1	0	0	NORMAL	0			0		1	0	MTHI	0
mtlo	000000	010011	rs	0	0	1	0	0	NORMAL	0			0		1	0	MTLO	0
mult	000000	011000	rs	rt	0	1	1	0	NORMAL	0	0		0		1	1	MULT	0
multu	000000	011001	rs	rt	0	1	1	0	NORMAL	0	0		0		1	1	MULTU	0
div	000000	011010	rs	rt	0	1	1	0	NORMAL	0	0		0		1	1	DIV	0
divu	000000	011011	rs	rt	0	1	1	0	NORMAL	0	0		0		1	1	DIVU	0