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单周期 CPU，支持 MIPS 指令集中的{beq，gez，bgtz，blez，bltz，bne，addi，addiu，andi，lb，lbu，lh，lhu，lui，lw，ori，sb，sh，slti，sltiu，sw，xori，j，jal，add，addu，and，jalr，jr，nor，or，sll，sllv，slt，sltu，sra，srav，srl，srlv，sub，subu，xor，nop}共 43 条指令，其中 add、addi 和 sub 暂不支持异常。为了实现这些指令，CPU 主要包含了 Controller（控制器）、IFU（取指令单元）、GRF（通用寄存器组，也称为寄存器文件、寄存器堆）、EXT（位扩展器）、ALU（算术逻辑单元）、DM（数据存储器）等基本部件并组成数据通路。

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. IFU

包含 PC（程序计数器）、NPC（Next PC）、IM（指令存储器）及相关逻辑。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。NPC 由计算下一条指令地址的逻辑实现。IM 由容量为 $32\ \text{bit} \times 1024 = 4\ \text{B} \times 1024 = 4\ \text{KB}$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。

2. GRF

由具有同步复位功能的 reg [31:0]实现，复位值为 0。由于 0 号寄存器的值始终为 0，故向 0 号寄存器写入值无效。

3. ALU

支持 32 位加、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。

4. DM

由起始地址为 `0x00000000` 的具有异步复位功能的容量为$32\ \text{bit} \times 1024 = 4\ \text{B} \times 1024 = 4\ \text{KB}$的 `reg [31:0]`实现，复位值为 `0x00000000`。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。

(三) 核心控制模块实现方法

Controller 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 Controller 时，可以“根据每条指令控制输出信号”，也可以“根据输出信号寻找相关指令”。为使添加指令更加直观、便捷和机械化，采用“根据每条指令控制输出信号”的方式。这样甚至可以通过控制信号表1，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 Controller 代码也并不繁琐。

为提升代码可读性与可维护性，将部分控制信号定义为宏，故并不完全与表 1 中的值对应。

单周期CPU.xlsx

表 1 控制信号

Type	Instruction	opcode	funct	NPCop	GRFwr	GRFaddrOp	GRFdataOp	EXTop	ALUop	ALUAop	ALUBop	DMwr	DMstoreOp	DMloadOp
B	**beq**	**000100**		**0b01**	**0**				**0b0010**	**0**	**0**	**0**		
B	bgez	000001		0b01	0				0b0101	0	0	0		
B	bgtz	000111		0b01	0				0b0000	0	0	0		
B	blez	000110		0b01	0				0b0001	0	0	0		
B	bltz	000001		0b01	0				0b0100	0	0	0		
B	bne	000101		0b01	0				0b0011	0	0	0		
I	addi	001000		0b00	1	0b00	0b00	0b01	0b1011	0	1	0		
I	addiu	001001		0b00	1	0b00	0b00	0b01	0b1011	0	1	0		
I	andi	001100		0b00	1	0b00	0b00	0b00	0b0111	0	1	0		
I	lb	100000		0b00	1	0b00	0b01	0b01	0b1011	0	1	0		0b001
I	lbu	100100		0b00	1	0b00	0b01	0b01	0b1011	0	1	0		0b100
I	lh	100001		0b00	1	0b00	0b01	0b01	0b1011	0	1	0		0b000
I	lhu	100101		0b00	1	0b00	0b01	0b01	0b1011	0	1	0		0b011
I	**lui**	**001111**		**0b00**	**1**	**0b00**	**0b00**	**0b10**	**0b1011**	**0**	**1**	**0**		
I	**lw**	**100011**		**0b00**	**1**	**0b00**	**0b01**	**0b01**	**0b1011**	**0**	**1**	**0**		**0b010**
I	**ori**	**001101**		**0b00**	**1**	**0b00**	**0b00**	**0b00**	**0b1000**	**0**	**1**	**0**		
I	sb	101000		0b00	0			0b01	0b1011	0	1	1	0b10	0b001
I	sh	101001		0b00	0			0b01	0b1011	0	1	1	0b01	0b000
I	slti	001010		0b00	1	0b00	0b00	0b01	0b0100	0	1	0		
I	sltiu	001011		0b00	1	0b00	0b00	0b01	0b0110	0	1	0		
I	**sw**	**101011**		**0b00**	**0**			**0b01**	**0b1011**	**0**	**1**	**1**	**0b00**	**0b010**

Type	Instruction	opcode	funct	NPCop	GRFwr	GRFaddrOp	GRFdataOp	EXTop	ALUop	ALUAop	ALUBop	DMwr	DMstoreOp	DMloadOp
I	xori	001110		0b00	1	0b00	0b00	0b00	0b1010	0	1	0		
J	j	000010		0b11	0							0		
J	**jal**	**000011**		**0b11**	**1**	**0b10**	**0b10**					**0**		
R	add	000000	100000	0b00	1	0b01	0b00		0b1011	0	0	0		
R	**addu**	**000000**	**100001**	**0b00**	**1**	**0b01**	**0b00**		**0b1011**	**0**	**0**	**0**		
R	and	000000	100100	0b00	1	0b01	0b00		0b0111	0	0	0		
R	jalr	**000000**	001001	0b10	1	0b01	0b10					0		
R	**jr**	**000000**	**001000**	**0b10**	**0**							**0**		
R	nor	**000000**	100111	0b00	1	0b01	0b00		0b1001	0	0	0		
R	or	000000	100101	0b00	1	0b01	0b00		0b1000	0	0	0		
R	sll	**000000**	000000	0b00	1	0b01	0b00		0b1101	1	0	0		
R	sllv	000000	000100	0b00	1	0b01	0b00		0b1101	0	0	0		
R	slt	**000000**	101010	0b00	1	0b01	0b00		0b0100	0	0	0		
R	sltu	000000	101011	0b00	1	0b01	0b00		0b0110	0	0	0		
R	sra	**000000**	000011	0b00	1	0b01	0b00		0b1111	1	0	0		
R	srav	000000	000111	0b00	1	0b01	0b00		0b1111	0	0	0		
R	srl	**000000**	000010	0b00	1	0b01	0b00		0b1110	1	0	0		
R	srlv	000000	000110	0b00	1	0b01	0b00		0b1110	0	0	0		
R	sub	000000	100010	0b00	1	0b01	0b00		0b1100	0	0	0		
R	**subu**	**000000**	**100011**	**0b00**	**1**	**0b01**	**0b00**		**0b1100**	**0**	**0**	**0**		
R	xor	000000	100110	0b00	1	0b01	0b00		0b1010	0	0	0		

二、 测试方案

在编写 Verilog 代码时，首先通过理论分析确保各模块与指令数据通路逻辑的正确性。然后简单改变各模块的输入，将其输出与正确输出比较。

之后使用 C/C++和 Python 3 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，没有明显差别。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.mem`。修改机器码时，只需修改测试专用目录下的 `code.mem` 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.mem` 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。

由于时间和精力所限，目前仅测试了要求的`{addu, subu, ori, lw, sw, beq, lui, jal, jr, nop}`指令，且 `beq`、`jal` 和 `jr` 指令仅进行了手工测试。

(一) C/C++自动测试程序

```
1   #include <cstdio>
2   #include <cstdlib>
3   #include <cstring>
4   #include <ctime>
5
6   #define SPLIT_TEST          0
7   #define TEST_CASE_NUM       10
8   #define FILE_NAME_LEN       16
```

```
9   #define BUFFER_LEN          512
10  #define USELESS_INFO_LEN    17
11  #define QUIT_LEN            7
12  const char simulationTime[] = "1000 ns";
13  const char vivadoDir[] = "C:/Xilinx/Vivado/2020.1/bin";
14  char masterName[FILE_NAME_LEN], queryName[FILE_NAME_LEN], buffer1[BU
    FFER_LEN], buffer2[BUFFER_LEN];
15  bool isAK = true;
16
17  inline int randint(int a, int b) {
18      return a + rand() % (b - a);
19  }
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(0, 27);
27          imm = randint(0, 3070) * 4;
28          fprintf(fp, "ori\t$%d, %d\n", rd, imm);
29      }
30      for (i = 0; i < 5; i++) {
31          target = randint(0, 27);
32          source = randint(0, 27);
33          imm = randint(0, 7) * 4;
34          fprintf(fp, "sw\t$%d, %d($%d)\n", target, imm, source);
35          fprintf(fp, "lw\t$%d, %d($%d)\n", target, imm, source);
36      }
37      for (i = 0; i < 5; i++) {
38          target = randint(0, 27);
39          source = randint(0, 27);
40          rd = randint(0, 27);
41          fprintf(fp, "addu\t$%d, $%d, $%d\n", target, source, rd);
```

```
42      }
43      for (i = 0; i < 5; i++) {
44          target = randint(0, 27);
45          source = randint(0, 27);
46          rd = randint(0, 27);
47          fprintf(fp, "subu\t$%d, $%d, $%d\n", target, source, rd);
48      }
49      for (i = 0; i < 5; i++) {
50          rd = randint(0, 27);
51          imm = randint(0, 0xffff);
52          fprintf(fp, "lui\t$%d, %d\n", rd, imm);
53      }
54      fprintf(fp, "nop\nnop\n");
55      fclose(fp);
56  }
57
58  inline int bufferCmp(char *buf1, char *buf2) {
59      int index1 = 0, index2 = 0;
60      while (buf1[index1] && buf1[index1] != '@') index1++;
61      while (buf2[index2] && buf2[index2] != '@') index2++;
62      return strcmp(buf1 + index1, buf2 + index2);
63  }
64
65  inline void fileCmp(int n) {
66      int cnt = 0;
67      FILE *master = fopen(masterName, "r");
68      FILE *query = fopen(queryName, "r");
69      for (int i = 0; i < USELESS_INFO_LEN; i++) fgets(buffer1, BUFFER
    _LEN, query);
70      while (fgets(buffer2, BUFFER_LEN, master)) {
71          cnt++;
72          if (strlen(buffer2) <= QUIT_LEN) break;
73          if (!fgets(buffer1, BUFFER_LEN, query)) {
74              printf("Test case %d failed at line %d.\n", n, cnt);
```

```
75              printf("Your outputs are fewer than expected outputs.");
76              isAK = false;
77              return;
78          } else if (bufferCmp(buffer1, buffer2)) {
79              printf("Test case %d failed at line %d.\n", n, cnt);
80              printf("Expected answer is %s\n", buffer2);
81              printf("Your answer is %s\n", buffer1);
82              isAK = false;
83              return;
84          }
85      }
86      printf("Test case %d is accepted.\n", n);
87      fclose(master);
88      fclose(query);
89  }
90
91  int main() {
92      char asmFileName[FILE_NAME_LEN];
93      printf("Autotest started...\n");
94      FILE *fp = fopen("cmd.tcl", "w");
95      fprintf(fp, "run %s;\nexit\n", simulationTime);
96      fclose(fp);
97      printf("Initialization succeeded!\n");
98      for (int i = 0; i < TEST_CASE_NUM; i++) {
99          printf("Testing case %d...\n", i);
100         sprintf(asmFileName, "test_%d.asm", i);
101         sprintf(masterName, "master_%d.txt", i);
102         sprintf(queryName, "query_%d.txt", i);
103         genCode(asmFileName);
104         sprintf(buffer1, "java -
    jar MARS.jar %s nc mc CompactDataAtZero a dump .text HexText code.me
    m", asmFileName);
105         system(buffer1);
```

```
106         sprintf(buffer1, "java -
    jar MARS.jar %s nc mc CompactDataAtZero >%s", asmFileName, masterNam
    e);
107         system(buffer1);
108         sprintf(buffer1, "%s/xvlog -prj vlog.prj --
    nolog", vivadoDir);
109         system(buffer1);
110         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);
111         system(buffer1);
112         sprintf(buffer1, "%s/xsim mips_tb -
    key {Behavioral:sim_1:Functional:mips_tb} -tclbatch cmd.tcl -
    log %s", vivadoDir, queryName);
113         system(buffer1);
114         fileCmp(i);
115         if (SPLIT_TEST) system("pause");
116     }
117     if (isAK) printf("\nCongratulations!\nTest cases are all killed!
    ");
118     return 0;
119 }
```

(二) Python 3 自动测试程序

```
1   import os
2   import subprocess
3   import random
4
5
6   SPLIT_TEST       = 0
7   TEST_CASE_NUM    = 10
8   FILE_NAME_LEN    = 16
9   BUFFER_LEN       = 512
```

```
10 USELESS_INFO_LEN = 17
11 QUIT_LEN         = 7
12 SIMULATION_TIME  = "1000 ns"
13 VIVADO_DIR       = "C:/Xilinx/Vivado/2020.1/bin/"
14 AK = True
15
16
17 def gen_code(file_name):
18     with open(file_name, "w") as f:
19         for i in range(5):
20             rd = random.randint(0, 27)
21             imm = random.randint(0, 3070) * 4
22             f.write(f"ori\t${rd}, {imm}\n")
23         for i in range(5):
24             target = random.randint(0, 27)
25             source = random.randint(0, 27)
26             imm = random.randint(0, 7) * 4
27             f.write(f"sw\t${target}, {imm}(${source})\n")
28             f.write(f"lw\t${target}, {imm}(${source})\n")
29         for i in range(5):
30             target = random.randint(0, 27)
31             source = random.randint(0, 27)
32             rd = random.randint(0, 27)
33             f.write(f"addu\t${target}, ${source}, ${rd}\n")
34         for i in range(5):
35             target = random.randint(0, 27)
36             source = random.randint(0, 27)
37             rd = random.randint(0, 27)
38             f.write(f"subu\t${target}, ${source}, ${rd}\n")
39         for i in range(5):
40             rd = random.randint(0, 27)
41             imm = random.randint(0, 0xffff)
42             f.write(f"lui\t${rd}, {imm}\n")
43         f.write("nop\nnop\n")
```

```
44
45 if __name__ == '__main__':
46     print("Autotest started...")
47     with open("cmd.tcl", "w") as f:
48         f.write(f"run {SIMULATION_TIME};\nexit\n")
49     print("Initialization succeeded!")
50     for i in range(TEST_CASE_NUM):
51         print(f"Testing case {i}...")
52         asm_file_name = f"test_{i}.asm"
53         master_name = f"master_{i}.txt"
54         query_name = f"query_{i}.txt"
55         gen_code(asm_file_name)
56         subprocess.call(f"java -jar MARS.jar {asm_file_name} nc mc
   CompactDataAtZero a dump .text HexText code.mem")
57         master = subprocess.Popen(f"java -jar MARS.jar
   {asm_file_name} nc mc CompactDataAtZero", shell=False,
   stdout=subprocess.PIPE, text="utf-8").communicate()[0].splitlines()
58         with open(master_name, "w") as f:
59             f.writelines(master)
60         os.system(VIVADO_DIR + "xvlog -prj vlog.prj --nolog")
61         os.system(VIVADO_DIR + "xelab -L xil_defaultlib -L
   unisims_ver -L unimacro_ver -L secureip --snapshot mips_tb
   xil_defaultlib.mips_tb xil_defaultlib.glbl --nolog")
62         os.system(VIVADO_DIR + "xsim mips_tb -key
   {Behavioral:sim_1:Functional:mips_tb} -tclbatch cmd.tcl -log " +
   query_name)
63         with open(query_name, "r") as f:
64             query = f.read()
65         line_cnt = 0
66         for line_master, line_query in zip(master,
   query.splitlines()[17:]):
67             line_cnt += 1
68             if line_master == "" or line_master is None:
69                 print(f"Test case {i} is accepted!")
```

```
70                  break
71              elif line_query is None:
72                  print(f"Test case {i} failed at line {line_cnt}!")
73                  print("Your outputs are fewer than expected
    outputs.")
74                  AK = False
75                  break
76              elif line_master != line_query:
77                  print(f"Test case {i} failed at line {line_cnt}!")
78                  print(f"Expected answer is {line_master}.")
79                  print(f"Your answer is {line_query}.")
80                  AK = False
81                  break
82          if SPLIT_TEST:
83              os.system("pause")
84      if AK:
85          print("\nCongratulations!\nTest cases are all killed!")
```

三、 思考题

(一) 根据你的理解，在下面给出的 DM 的输入示例中，地址信号 addr 位数为什么是[11:2]而不是[9:0]？这个 addr 信号又是从哪里来的？

文件	模块接口定义

dm.v	dm(clk, reset, MemWrite, addr, din, dout);
	input clk; // clock
	input reset; // reset
	input MemWrite; // memory write enable
	input [31:0] din; // write data
	input [11:21] addr; // memory's address for write
	output [31:0] dout; // read data;

该 DM 采用了与本文相同的设计，即每个连续的地址都能取一个 word，故只取地址的[11:21]位来以 word 为单位寻址。但这样的设计可扩展性不佳，因为若要写 half-word 或 byte，就要用到地址的后两位。

addr 信号来自 ALU。

(二) 思考 Verilog 语言设计控制器的译码方式，给出代码示例，并尝试对比各方式的优劣。

正如前文所述，实现 Controller 时，可以“根据每条指令控制输出信号”，也可以“根据输出信号寻找相关指令”。

1. 根据每条指令控制输出信号

这样可使添加指令更加直观、便捷和机械化，甚至可以通过控制信号表，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 Controller 代码也并不繁琐。

```
1   `include "constants.vh"
2   module Controller(
3       input [31:0] instruction,
4       input branch,
5       output reg [1:0] NPCop,
6       output reg [1:0] GRFaddrOp,
7       output reg [1:0] GRFdataOp,
8       output reg GRFwr,
9       output reg [1:0] EXTop,
10      output reg ALUAop,
11      output reg ALUBop,
12      output reg [3:0] ALUop,
13      output reg DMwr,
14      output reg [1:0] DMstoreOp,
15      output reg [2:0] DMloadOp
16      );
17
18      wire [5:0] opcode = instruction[31:26];
19      wire [5:0] funct = instruction[5:0];
20      wire [4:0] rt = instruction[20:16];
21
22      always @* begin
23          case (opcode)
24              `bgezORbltz: begin
25                  if (rt == 1) ALUop = `GEZ; else ALUop = `LT;
26                  NPCop = `BRANCH; GRFwr = 0; ALUAop = 0; ALUBop = 0;
27              end
```

```
28              `beq:   begin ALUop = `EQ;  NPCop = `BRANCH;  GRFwr = 0;                                                ALUAop = 0; ALUBop = 0;                                                  end
29              `bgtz:  begin ALUop = `GT;  NPCop = `BRANCH;  GRFwr = 0;                                                ALUAop = 0; ALUBop = 0; DMwr = 0;                                        end
30              `blez:  begin ALUop = `LE;  NPCop = `BRANCH;  GRFwr = 0;                                                ALUAop = 0; ALUBop = 0; DMwr = 0;                                        end
31              `bne:   begin ALUop = `NE;  NPCop = `BRANCH;  GRFwr = 0;                                                ALUAop = 0; ALUBop = 0; DMwr = 0;                                        end
32              `addi:  begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 0; EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 0;                                        end
33              `addiu: begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 0; EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 0;                                        end
34              `andi:  begin ALUop = `AND; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 0; EXTop = `ZERO;   ALUAop = 0; ALUBop = 1; DMwr = 0;                                        end
35              `lb:    begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 1; EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 0; DMloadOp = `BYTE;                   end
36              `lbu:   begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 1; EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 0; DMloadOp = `BYTE_U;                 end
37              `lh:    begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 1; EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 0; DMloadOp = `HALF;                   end
38              `lhu:   begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 1; EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 0; DMloadOp = `HALF_U;                 end
39              `lui:   begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 0; EXTop = `UPPER;  ALUAop = 0; ALUBop = 1; DMwr = 0;                                        end
40              `lw:    begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 1; EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 0; DMloadOp = `WORD;                   end
41              `ori:   begin ALUop = `OR;  NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 0; EXTop = `ZERO;   ALUAop = 0; ALUBop = 1; DMwr = 0;                                        end
42              `sb:    begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 0;                                EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 1; DMstoreOp = `BYTE;                    end
43              `sh:    begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 0;                                EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 1; DMstoreOp = `HALF;                    end
44              `slti:  begin ALUop = `LT;  NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 0; EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 0;                                        end
45              `sltiu: begin ALUop = `LTU; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 0; EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 0;                                        end
46              `sw:    begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 0;                                EXTop = `SIGNED; ALUAop = 0; ALUBop = 1; DMwr = 1; DMstoreOp = `WORD;                    end
47              `xori:  begin ALUop = `XOR; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 0; GRFdataOp = 0; EXTop = `ZERO;   ALUAop = 0; ALUBop = 1; DMwr = 0;                                        end
48              `j:     begin               NPCop = `JMP_26;  GRFwr = 0;                                                                 DMwr = 0;                                                end
49              `jal:   begin               NPCop = `JMP_26;  GRFwr = 1; GRFaddrOp = 2; GRFdataOp = 2;                                   DMwr = 0;                                                end
50              0: case (funct)
51              `add:   begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
52              `addu:  begin ALUop = `ADD; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
53              `and:   begin ALUop = `AND; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
54              `jalr:  begin               NPCop = `JMP_REG; GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 2;                       DMwr = 0; end
55              `jr:    begin               NPCop = `JMP_REG; GRFwr = 0;                                                     DMwr = 0; end
56              `nor:   begin ALUop = `NOR; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
57              `or:    begin ALUop = `OR;  NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
58              `sll:   begin ALUop = `SL;  NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 1; ALUBop = 0; DMwr = 0; end
59              `sllv:  begin ALUop = `SL;  NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
60              `slt:   begin ALUop = `LT;  NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
61              `sltu:  begin ALUop = `LTU; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
62              `sra:   begin ALUop = `SRA; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 1; ALUBop = 0; DMwr = 0; end
63              `srav:  begin ALUop = `SRA; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
64              `srl:   begin ALUop = `SRL; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 1; ALUBop = 0; DMwr = 0; end
65              `srlv:  begin ALUop = `SRL; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
66              `sub:   begin ALUop = `SUB; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
67              `subu:  begin ALUop = `SUB; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
68              `xor:   begin ALUop = `XOR; NPCop = `NORMAL;  GRFwr = 1; GRFaddrOp = 1; GRFdataOp = 0; ALUAop = 0; ALUBop = 0; DMwr = 0; end
69              `nop:   begin               NPCop = `NORMAL;  GRFwr = 0;                                                     DMwr = 0; end
70              endcase
71          endcase
72      end
73  endmodule
```

2. 根据输出信号寻找相关指令

这样设计编写出来的代码不如上面的整齐、机械，且由于添加指令时是通过指令来确定输出信号的，很容易出错。由于用 Logisim 实现 Controller 时，实际上采用的就是这种方法（如图 1 所示），此处暂不给出等价的 Verilog 代码了。

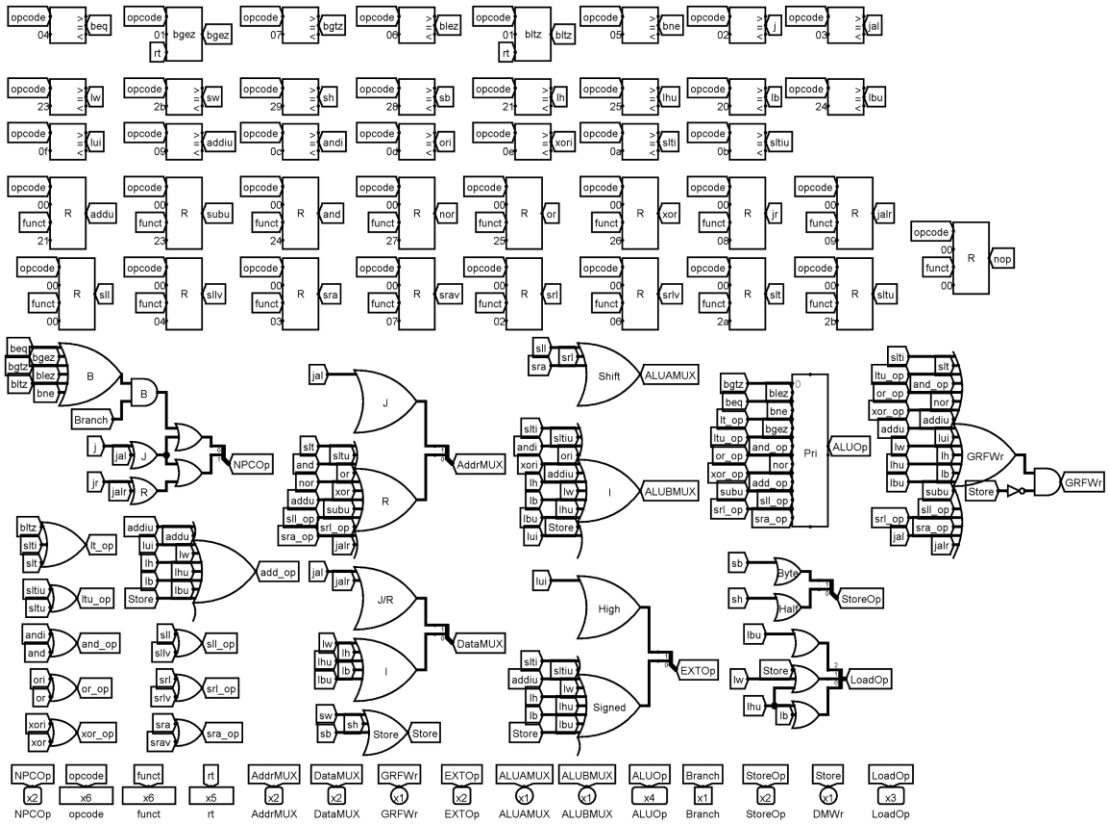

图 1 Logisim Controller 根据输出信号寻找相关指令

(三) 在相应的部件中，reset 的优先级比其他控制信号（不包括 clk 信号）都要高，且相应的设计都是同步复位。清零信号 reset 所驱动的部件具有什么共同特点？

是时序逻辑电路。

(四) C 语言是一种弱类型程序设计语言。C 语言中不对计算结果溢出进行处理，这意味着 C 语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持 C 语言，MIPS 指令的所有计算指令均可以忽略溢出。请说明为什么在忽略溢出的前提下，addi 与 addiu 是等价的，add 与 addu 是等价的。

因为唯一的区别是结尾不含 u 的指令在溢出时会抛出异常，而含 u 的不会。

(五) 根据自己的设计说明单周期处理器的优缺点。

优点是相对简单，逻辑清晰。缺点是闲置部件太多，效率较低。