COD_LAB6 综合实验——流水线和总线设计

黄业琦 PB17000144

May 7, 2019

Contents

1 实验目的

我们之前的实验里面我们实现了多周期的 CPU，本次实验我们需要设计流水线 CPU，此外，我们还设计了总线和其他组件。增设了 IO 设备，Input 设备为开关，Output 设备为数码管。最后我们再我们的 pipeline-CPU 中运行一个简单的应用。本次实验选择了实现数字密码锁的功能。

2 数字密码锁介绍

密码锁设置为 8 位，操作由上下左右键控制。按 reset 键重启输入。输入方法为：按上下键调整数值大小，左右键控制操作位置。长按 reset 可以修改密码。密码输入错误后四个 LED 亮，成功前四个 LED 亮，其余的 8 个 LED 用于计时，8 个 LED 全部熄灭时表示时间终止。

3 实验环境

- Linux 下编程调试和仿真，使用 IVerilog，GtkWave 系列工具。
- Windows 下用于生成比特流文件，使用 Vivado 2018.2，Verilog HDL。
- 所有下载均在 Nexsy4-DDR 实验板完成。
- 优秀的代码风格和规范的代码格式也很重要，本次实验借助 Vscode 的 verilog-format 插件进行整理代码的工作。
- 汇编、C 等非 Verilog 均在 Vscode 下编程，GNU 库更新至 2019.5.16 最新。
- 获取 MIPS 的 machine code 转换采用李老师教学主页提供的 MIPS-sim 工具实现。

4 数字密码锁代码

4.1 Origin Code Draft

下面的代码仅作为参考，作为我们的思路代码。

Listing 1: lock.v

```
1 #include <stdio.h>
2
```

```
int main()
{
    int input[8];
    int password[8];
    int success;

    int i;
    for (i = 0; i < 8; i++)
    {
        password[i] = i + 1;
        input[i] = 0;
    }

    int bitsselect = 0;
    int nxt, nxt_delay;
    int prv, prv_delay;
    int inc, inc_delay;
    int dec, dec_delay;
    int rst, rst_delay;

    int wait = 0;
    int tag = 0;
    int time_count;
    int rst_count;

    nxt = 0;
    prv = 0;
    inc = 0;
    dec = 0;
    nxt_delay = 0;
    prv_delay = 0;
    inc_delay = 0;
    dec_delay = 0;

    for (;;)
    {
        if (nxt_delay == 0 && nxt == 1)
            bitsselect = bitsselect == 7 ? 7 :bitsselect + 1;

        if (prv_delay == 0 && prv == 1)
            bitsselect = bitsselect == 0 ? 0 :bitsselect - 1;

        if (inc_delay == 0 && inc == 1)
            input[bitsselect] = (input[bitsselect] + 1) % 10;

        if (dec_delay == 0 && dec == 1)
            input[bitsselect] = (input[bitsselect] + 9) % 10;

        if (rst == 1)
        {
            for (i = 0; i < 8; i++)
```

```
54              input[i] = 0;
55          bitsselect = 0;
56      }
57
58      if (rst == 1)
59          rst_count++;
60      else
61          rst_count = 0;
62
63      if (rst_count > 20000000)
64      {
65          for (i = 0; i < 8; i++)
66              password[i] = input[i];
67      }
68
69      int flag = 1;
70
71      for (i = 0; i < 7; i++)
72      {
73          if (input[i] != password[i])
74          {
75              flag = 0;
76              break;
77          }
78      }
79
80      if (flag)
81          success = 1;
82      else
83          success = 0;
84
85      wait++;
86
87      if (success)
88      {
89          wait = 10000000;
90          while (wait--)
91          {
92          }
93          tag = 1;
94          success = 0;
95          for (i = 0; i < 8; i++)
96              input[i] = 0;
97      }
98      else
99      {
100         if (wait > 600000000)
101         {
102             tag = 0;
103             wait = 10000000;
104             while (wait--)
```

```
105                 {
106                 }
107             }
108         }
109 
110         time_count = (127 >> ((600000000 - wait) / (600000000 / 8)));
111     }
112 }
```

C 的代码当然不是最终需要的版本，我们之后再汇编的时候还需要微微调整。比如我们的输入方式就比较特殊，是 IO+ 总线的方式输入的，所以程序中并不可以加 scanf 进行输入。我们再后面的汇编代码中会做更加细致的调整。

4.2 MIPS Origin

相关的 MIPS 代码我们使用相关工具，在C to MIPS网站上输入相关代码，我们就可以进行转换。最终得到：

```
1  .file   1 ""
2      .section .mdebug.abi32
3      .previous
4      .nan    legacy
5      .module fp=32
6      .module nooddspreg
7      .abicalls
8      .text
9      .align  2
10     .globl  main
11     .set    nomips16
12     .set    nomicromips
13     .ent    main
14     .type   main, @function
15 main:
16     .frame  $fp,144,$31     # vars= 128, regs= 1/0, args= 0, gp= 8
17     .mask   0x40000000,-4
18     .fmask  0x00000000,0
19     .set    noreorder
20     .set    nomacro
21     addiu   $sp,$sp,-144
22     sw  $fp,140($sp)
23     move    $fp,$sp
24     sw  $0,12($fp)
25     b   $L2
26     movz    $31,$31,$0
27     nop
28 
29 $L3:
30     lw  $2,12($fp)
31     nop
32     addiu   $3,$2,1
33     lw  $2,12($fp)
34     nop
```

```
35      sll $2,$2,2
36      addiu   $4,$fp,8
37      addu    $2,$4,$2
38      sw  $3,92($2)
39      lw  $2,12($fp)
40      nop
41      sll $2,$2,2
42      addiu   $3,$fp,8
43      addu    $2,$3,$2
44      sw  $0,60($2)
45      lw  $2,12($fp)
46      nop
47      addiu   $2,$2,1
48      sw  $2,12($fp)
49  $L2:
50      lw  $2,12($fp)
51      nop
52      slt $2,$2,8
53      bne $2,$0,$L3
54      nop
55
56      sw  $0,16($fp)
57      sw  $0,20($fp)
58      sw  $0,28($fp)
59      sw  $0,32($fp)
60      sw  $0,36($fp)
61      sw  $0,40($fp)
62      sw  $0,44($fp)
63      sw  $0,48($fp)
64      sw  $0,52($fp)
65      sw  $0,56($fp)
66      sw  $0,60($fp)
67  $L27:
68      lw  $2,48($fp)
69      nop
70      bne $2,$0,$L4
71      nop
72
73      lw  $3,32($fp)
74      li  $2,1            # 0x1
75      bne $3,$2,$L4
76      nop
77
78      lw  $3,16($fp)
79      li  $2,7            # 0x7
80      beq $3,$2,$L5
81      nop
82
83      lw  $2,16($fp)
84      nop
85      addiu   $2,$2,1
86      b   $L6
87      nop
88
89  $L5:
90      li  $2,7            # 0x7
91  $L6:
92      sw  $2,16($fp)
```

```
93  $L4:
94      lw   $2,52($fp)
95      nop
96      bne  $2,$0,$L7
97      nop
98
99      lw   $3,36($fp)
100     li   $2,1               # 0x1
101     bne  $3,$2,$L7
102     nop
103
104     lw   $2,16($fp)
105     nop
106     beq  $2,$0,$L8
107     nop
108
109     lw   $2,16($fp)
110     nop
111     addiu    $2,$2,-1
112     b    $L9
113     nop
114
115 $L8:
116     move     $2,$0
117 $L9:
118     sw   $2,16($fp)
119 $L7:
120     lw   $2,56($fp)
121     nop
122     bne  $2,$0,$L10
123     nop
124
125     lw   $3,40($fp)
126     li   $2,1               # 0x1
127     bne  $3,$2,$L10
128     nop
129
130     lw   $2,16($fp)
131     nop
132     sll  $2,$2,2
133     addiu    $3,$fp,8
134     addu     $2,$3,$2
135     lw   $2,60($2)
136     nop
137     addiu    $3,$2,1
138     li   $2,10              # 0xa
139     bne  $2,$0,1f
140     div  $0,$3,$2
141     break    7
142 1:
143     mfhi     $2
144     move     $4,$2
145     lw   $2,16($fp)
146     nop
147     sll  $2,$2,2
148     addiu    $3,$fp,8
149     addu     $2,$3,$2
150     sw   $4,60($2)
```

```
$L10:
    lw   $2,60($fp)
    nop
    bne  $2,$0,$L11
    nop

    lw   $3,44($fp)
    li   $2,1               # 0x1
    bne  $3,$2,$L11
    nop

    lw   $2,16($fp)
    nop
    sll  $2,$2,2
    addiu    $3,$fp,8
    addu     $2,$3,$2
    lw   $2,60($2)
    nop
    addiu    $3,$2,9
    li   $2,10              # 0xa
    bne  $2,$0,1f
    div  $0,$3,$2
    break    7
1:
    mfhi     $2
    move     $4,$2
    lw   $2,16($fp)
    nop
    sll  $2,$2,2
    addiu    $3,$fp,8
    addu     $2,$3,$2
    sw   $4,60($2)
$L11:
    lw   $3,64($fp)
    li   $2,1               # 0x1
    bne  $3,$2,$L12
    nop

    sw   $0,12($fp)
    b    $L13
    nop

$L14:
    lw   $2,12($fp)
    nop
    sll  $2,$2,2
    addiu    $3,$fp,8
    addu     $2,$3,$2
    sw   $0,60($2)
    lw   $2,12($fp)
    nop
    addiu    $2,$2,1
    sw   $2,12($fp)
$L13:
    lw   $2,12($fp)
    nop
    slt  $2,$2,8
    bne  $2,$0,$L14
```

```
209     nop
210
211     sw   $0,16($fp)
212 $L12:
213     li   $2,1               # 0x1
214     sw   $2,24($fp)
215     sw   $0,12($fp)
216     b    $L15
217     nop
218
219 $L18:
220     lw   $2,12($fp)
221     nop
222     sll  $2,$2,2
223     addiu    $3,$fp,8
224     addu     $2,$3,$2
225     lw   $3,60($2)
226     lw   $2,12($fp)
227     nop
228     sll  $2,$2,2
229     addiu    $4,$fp,8
230     addu     $2,$4,$2
231     lw   $2,92($2)
232     nop
233     beq  $3,$2,$L16
234     nop
235
236     sw   $0,24($fp)
237     b    $L17
238     nop
239
240 $L16:
241     lw   $2,12($fp)
242     nop
243     addiu    $2,$2,1
244     sw   $2,12($fp)
245 $L15:
246     lw   $2,12($fp)
247     nop
248     slt  $2,$2,7
249     bne  $2,$0,$L18
250     nop
251
252 $L17:
253     lw   $2,24($fp)
254     nop
255     beq  $2,$0,$L19
256     nop
257
258     li   $2,1               # 0x1
259     sw   $2,8($fp)
260     b    $L20
261     nop
262
263 $L19:
264     sw   $0,8($fp)
265 $L20:
266     lw   $2,20($fp)
```

```
267     nop
268     addiu    $2,$2,1
269     sw   $2,20($fp)
270     lw   $2,8($fp)
271     nop
272     beq  $2,$0,$L21
273     nop
274
275     li   $2,9961472          # 0x980000
276     ori  $2,$2,0x9680
277     sw   $2,20($fp)
278     nop
279 $L22:
280     lw   $2,20($fp)
281     nop
282     addiu    $3,$2,-1
283     sw   $3,20($fp)
284     bne  $2,$0,$L22
285     nop
286
287     li   $2,1                # 0x1
288     sw   $2,28($fp)
289     sw   $0,8($fp)
290     sw   $0,12($fp)
291     b    $L23
292     nop
293
294 $L24:
295     lw   $2,12($fp)
296     nop
297     sll  $2,$2,2
298     addiu    $3,$fp,8
299     addu     $2,$3,$2
300     sw   $0,60($2)
301     lw   $2,12($fp)
302     nop
303     addiu    $2,$2,1
304     sw   $2,12($fp)
305 $L23:
306     lw   $2,12($fp)
307     nop
308     slt  $2,$2,8
309     bne  $2,$0,$L24
310     nop
311
312     b    $L27
313     nop
314
315 $L21:
316     lw   $3,20($fp)
317     li   $2,599982080            # 0x23c30000
318     ori  $2,$2,0x4601
319     slt  $2,$3,$2
320     bne  $2,$0,$L27
321     nop
322
323     sw   $0,28($fp)
324     li   $2,9961472          # 0x980000
```

```
325     ori $2,$2,0x9680
326     sw  $2,20($fp)
327     nop
328 $L26:
329     lw  $2,20($fp)
330     nop
331     addiu   $3,$2,-1
332     sw  $3,20($fp)
333     bne $2,$0,$L26
334     nop
335
336     b   $L27
337     nop
338
339     .set    macro
340     .set    reorder
341     .end    main
342     .size   main, .-main
343     .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609"
```

Listing 2: lock.v

显然，代码太长了。

4.3 MIPS Final

当然上面的代码显得还是国语臃肿和累赘，我们为了简化代码，网上找到了相关优化开关：Better MIPS

```
1  .file   1 ""
2      .section .mdebug.abi32
3      .previous
4      .nan    legacy
5      .module fp=32
6      .module nooddspreg
7      .abicalls
8      .section    .text.startup,"ax",@progbits
9      .align  2
10     .globl  main
11     .set    nomips16
12     .set    nomicromips
13     .ent    main
14     .type   main, @function
15 main:
16     .frame  $sp,72,$31      # vars= 64, regs= 0/0, args= 0, gp= 8
17     .mask   0x00000000,0
18     .fmask  0x00000000,0
19     .set    noreorder
20     .set    nomacro
21     addiu   $sp,$sp,-72
22     addiu   $4,$sp,8
23     addiu   $3,$sp,40
24     move    $2,$0
25     move    $9,$4
26     movz    $31,$31,$0
```

```
27      move    $5,$3
28      li  $6,8            # 0x8
29  $L2:
30      addiu   $2,$2,1
31      sw  $2,0($4)
32      sw  $0,0($3)
33      addiu   $4,$4,4
34      bne $2,$6,$L2
35      addiu   $3,$3,4
36
37      li  $4,599982080            # 0x23c30000
38      move    $3,$0
39      li  $10,1           # 0x1
40      li  $8,32           # 0x20
41      li  $11,28          # 0x1c
42      addiu   $4,$4,17921
43      li  $12,9961472         # 0x980000
44  $L3:
45  $L21:
46      beq $10,$0,$L6
47      move    $2,$0
48
49      b   $L22
50      addu    $7,$5,$2
51
52  $L6:
53      addu    $6,$5,$2
54      addiu   $2,$2,4
55      bne $2,$8,$L6
56      sw  $0,0($6)
57
58      b   $L5
59      move    $2,$0
60
61  $L5:
62      addu    $7,$5,$2
63  $L22:
64      addu    $6,$9,$2
65      lw  $7,0($7)
66      lw  $6,0($6)
67      nop
68      bne $7,$6,$L8
69      nop
70
71      addiu   $2,$2,4
72      bne $2,$11,$L22
73      addu    $7,$5,$2
74
75      move    $2,$0
76  $L9:
77      addu    $3,$5,$2
78      addiu   $2,$2,4
79      bne $2,$8,$L9
80      sw  $0,0($3)
81
82      b   $L3
83      li  $3,-1           # 0xffffffffffffffff
84
```

```
$L8:
    addiu   $3,$3,1
    slt $6,$3,$4
    bne $6,$0,$L3
    ori $2,$12,0x9681

    addiu   $2,$2,-1
$L23:
    bne $2,$0,$L23
    addiu   $2,$2,-1

    addiu   $2,$2,1
    b   $L21
    li  $3,-1           # 0xffffffffffffffff

    .set    macro
    .set    reorder
    .end    main
    .size   main, .-main
    .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609"
```

Listing 3: lock.v

最后我们还需要将代码段和数据段分离，选择合适的数据内存位置进行 IO 的分配。存入我们的 COE 文件中，即可进行生成。

5 CPU 指令集设计

本次设计的指令集再上次实验中新增了一些其他指令，扩展 MIPS 指令至 36 条基本指令。

新扩展指令集为:

- R-type:

  type1: ADD, ADDU, SUB, SUBU

  type1: AND, OR, NOR, XOR

  type1: SLT, SLTU, SLLV, SRLV, SRAV

  type2: SLL, SRL, SRA

  JR

- J-type:

  J

- I-type:

  type1: ADDI, ADDIU, ANDI, ORI, XORI, SLTI, SLTIU

  Branch: BEQ, BNE, BGEZ, BGTZ, BLEZ, BLTZ

  LW, LH, LB

  SW

6 模块设计

总设计图参考 PH Processor 设计。

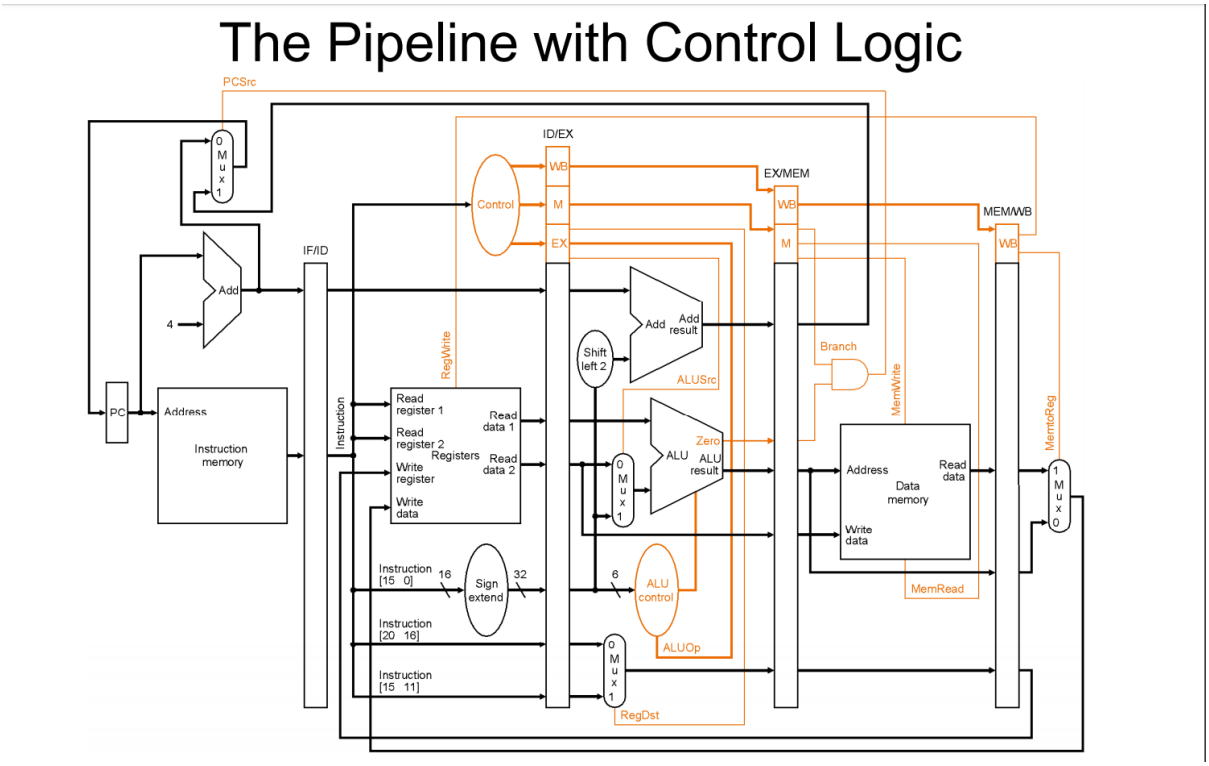

Figure 1: Pipeline

接下来我们依次介绍五级流水的设计结构。

6.1 IF 模块

IF 模块中，我们根据 PC 取对应指令，并设置下一阶段的 PC 的值。

信号	选择
{JR,J,Z}=100	JR 地址
{JR,J,Z}=010	J 地址
{JR,J,Z}=001	Branch 地址
{JR,J,Z}=000	PC+4

Table 1: PC 信号选择

6.2 ID 模块

解析指令的操作码, 产生各种控制信号。流水线冒险检测也在 ID 级进行, 冒险检测电路需要上一条指令的 MemRead, 在检测到冒险条件成立时, 冒险检测电路产生 stall 信号, 清空 ID/EX 寄存器, 插入一个流水线气泡。
我们的控制信号也在这里进行处理。我们相关信号有：(参考代码 Decode.v)

- RegWrite 是否写寄存器

  真条件：opcode 为 LW, R-type1, R-type2, I-type1

- RegDst 目标寄存器是 rt 还是 rd

  rd(RegDst=1): R-type1, R-type2

  rt(RegDst=0): LW, I-type1

- MemWrite 是否写数据存储器

  真条件：opcode 为 LW

- MemRead 是否读数据存储器

  真条件：opcode 为 SW

- MemtoReg 写寄存器的数据来自 Mem 还是 ALU

  ALU(MmetoReg=0): R-type1, R-type2, I-type1

  Mem(MemtoReg=1): LW

- ALUSrcA ALU 第一操作数为 rs 还是 sa

  rs(ALUSrcA=0):LW, R-type1, I-type1

  sa(ALUSrcA=1): R-type2

- ALUSrcB ALU 第二操作数为 rt 还是 Imm

  rt(ALUSrcB=0): R-type1, R-type2

Imm(ALUSrcB=1):LW,SW,I-type1

- PCSrc PC 来源

  见 PC_Source 表 1

- LwByte 写入方式

- ALUCode 决定 ALU 功能, 由指令中的 op 段,rt 段,funct 段共同决定

  我们译码表参考MIPS 指令

6.3 分支检测 (Branch Test)

Zero 检测 Branch 条件是否成立, 其中 BEQ、BNE 两个操作数为 RsData 与 RtData, 而 BGEZ、BGTZ、BLEZ 和 BLTZ 指令则为 RsData 与常数 0 比较。

6.4 寄存器堆

此处的寄存器堆更准确的来说融入了总线的特质，我们在里面增加了 Forwarding 的线路。我们具体的应用中是对四个数求和并输出结果，我们的输入和输出选取了 5 个内存空间进行存储。

6.5 冒险检测 (Hazard Detector)

冒险成立的条件:

1. 上一条指令为 LW, 即 MemRead_ex=1;
2. 在 EX 级的 LW 指令与在 ID 级的指令读写的是同一个寄存器, 即 RegWriteAddr_ex=Rs_Addr_idRtAddr_id;

解决冒险的方法:

1. 插入一个流水线气泡 Stall 清空 ID/EX 寄存器并且阻塞流水线 ID 级、IF 级流水线
   Stall=((RegWriteAddr_ex==RsAddr_id) or (RegWriteAddr_ex==RtAddr_id)) and MemRead_ex
2. 保持 PC 寄存器和 IF/ID 流水线寄存器不变 PC_IFWrite= Stall;

6.6 EX 模块

EX 部分我们与之前的多周期 CPU 并无太大区别，我们的主要设计区别在于 Forwarding 的方法。
操作数 A 和 B 由数据选择器决定，数据选择器的地址信号即为 ForwardA 和 ForwardB。

- FowardA/B=00 操作数取自寄存器堆
- FowardA/B=01 来自二阶数据相关的 Forwarding 数据
- FowardA/B=10 来自一阶数据相关的 Forwarding 数据

7 寄存器设计

流水线寄存器负责将流水线的各部分分开，共有 IF/ID、ID/EX、EX/MEM、MEM/WB 四组。

1. IF/ID: 当发生数据冒险时，需保持 IF/ID 流水线寄存器不变，故 IF/ID 流水线寄存器具有使能信号 PC_IFWrite 输入；当流水线发生分支冒险时，需清空 IF/ID 流水线寄存器，清零信号为 IF_flush。
2. ID/EX: 当流水线发生数据冒险时，需清空 ID/EX 流水线寄存器而插入一个气泡。
3. EX/MEM: 普通寄存器。
4. MEM/WB: 普通寄存器

8 冒险处理方法

8.1 类别一——对应课本 RAW 类型数据冒险

例子，课本代码：

```
1    sub    $2, $1,$3
2    and    $12,$2,$5
3    or     $13,$6,$2
4    add    $14,$2,$2
5    sw     $15,100($2)
```

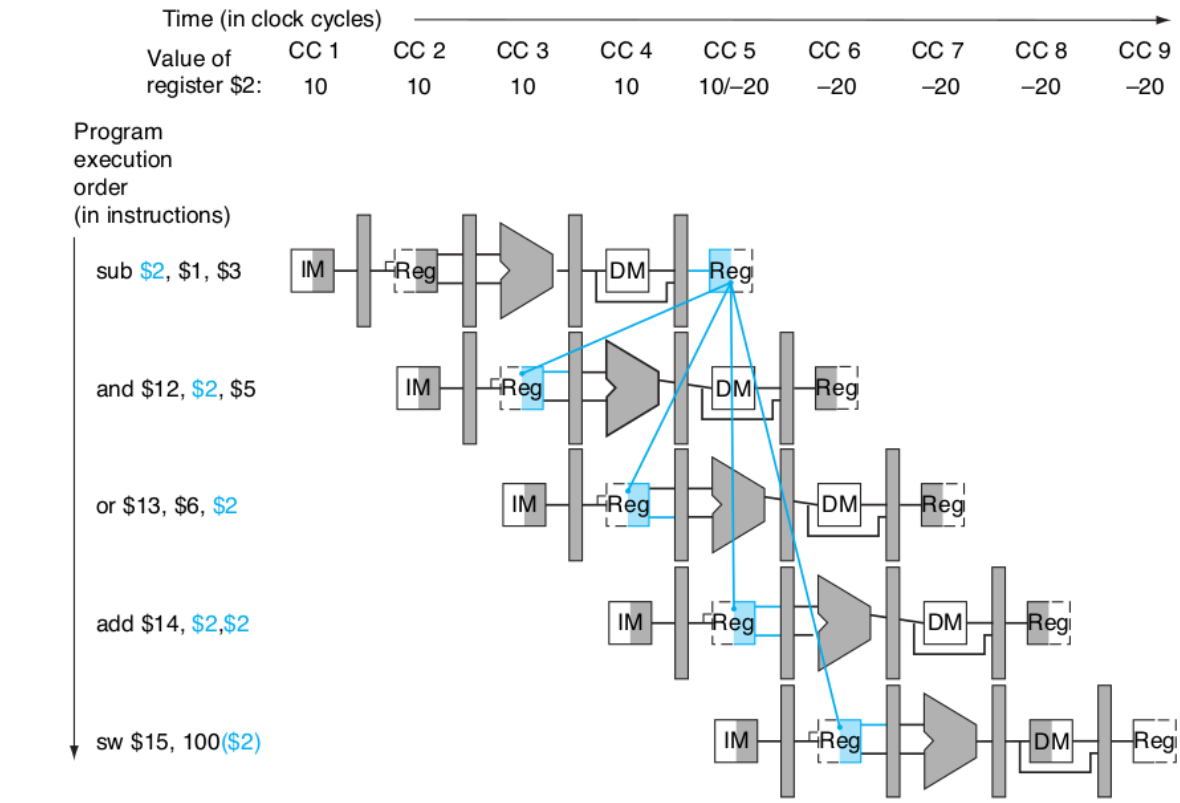

Figure 2: Data Harzard

8.1.1 情况一

第 i 条指令的源操作数与第 $i-1$ 条指令的目标寄存器相同。
例如代码 1,2 两行。sub 的数据结果再第 5 时间周期（第一条指令的 EX）的末尾，产生结果。但是 and 在第 4 周期（第二条指令的 EX）执行前需要用到该数值。

解决方法：Forwarding data = ALUout

Forwarding 条件：

1. MEM 级指令要写回。
2. MEM 级指令写回目标寄存器与 EX 级指令源寄存器是同一寄存器, 即 RegWriteAddr_mem=RsAddr_ex/RtAddr_ex

8.1.2 情况二

第 i 条指令的源操作数与第 $i-2$ 条指令的目标寄存器相同。
例如代码 1,3 两行。sub 的数据结果再第 5 时间周期（第一条指令的 WB）结束后才能写回。但是 or 在第 5 周期（第三条指令的 EX）执行前需要用到该数值。

解决方法：Forwarding data = RegWriteback data

Forwarding 条件：

1. WB 级指令是写操作, 即 RegWrite_wb=1;
2. WB 级指令写回的目标寄存器与在 EX 级指令的源寄存器是同一寄存器, 即 RegWriteAddr_wb=RsAddr_ex 或 RegWriteAddr_wb=RtAddr_ex
3. EX 冒险不成立, 即 RegWriteAddr_mem≠RsAddr_ex 或 RegWriteAddr_mem≠RtAddr_ex

8.1.3 情况三

同一时间周期操作同一个寄存器。
例如代码 1,4 两行。sub 数据在第 5 时间周期（第一条指令的 WB）正在对 $2 进行写回，但 add 指令正在使用该数值。

解决方法：通过 MEM/WB 流水线寄存器, 将前一指令结果转发给后一指令, 转发数据为 RegWriteData_wb。

Forwarding 条件：

1. WB 级指令是写操作, 即 RegWrite_wb=1
2. WB 级指令写回的目标寄存器与在 ID 级指令的源寄存器是同一寄存器, 即 RegWriteAddr_wb=RsAddr_id 或 RegWriteAddr_wb=RtAddr_id

8.2 类别二——对应课本 stall 方法

定义: 当一条指令试图读取一个寄存器, 而它前一条指令是 lw 指令, 并且该 lw 指令写入的是同一个寄存器时,Forwarding 无法解决问题。此时我们将采取 stall 的方法进行处理。

判定条件

1. MemRead_ex = 1

2. EX 级的 lw 和 ID 级指令读同一个寄存器，即 RegWriteAddr_ex=RsAddr_id 或 RegWriteAddr_ex=RtAddr_id

处理方法:

我在判定条件成立的时候将 lw 和之后的指令之间插入 stall，实现时，我们的 Detector 检测到 Hazard 之后我们会输出 Stall 和 PC_IFWrite。Stall 信号将 ID/EX 流水线寄存器中的 EX、MEM 和 WB 级控制信号全部清零。这些信号传递到流水线后面的各级, 由于控制信号均为 0, 所以不会对任何寄存器和存储器进行写操作, 高电平有效。PC_IFWrite 信号禁止 PC 寄存器和 IF/ID 流水线寄存器接收新数据, 低电平有效。

8.3 类别三——分支预测

流水线每个时钟周期都得取指令才能维持运行, 但分支指令必须等到 MEM 级才能确定是否执行分支。这种为了确定预取正确的指令而导致的延迟叫做控制冒险或分支冒险。

判定条件:

将用于判断分支指令成立的 Zero 信号检测电路从 ALU 中独立出来, 并将它从 EX 级提前至 ID 级。

实现方法: 加入一个控制信号 IF_flush, 做为 IF/ID 流水线寄存器的清零信号。当 Z=1, 则 IF_flush=1, 否则 IF_flush=0, 故 IF_flush = Z。考虑到本系统还要实现的无条件跳转指令: J 和 JR, 在执行这两个指令时也必须要对 IF/ID 流水线寄存器进行清空。

9 代码

```
`timescale 1ns / 1ps

module ALU(input [4:0]ALUCode,
           input [31:0]A,
           input [31:0]B,
           output reg[31:0]Result);

    // * R-type
    localparam ALU_ADD = 5'b00000;
    localparam ALU_ADDU = 5'b10101;
    localparam ALU_AND = 5'b00001;
    localparam ALU_XOR = 5'b00010;
    localparam ALU_OR = 5'b00011;
    localparam ALU_NOR = 5'b00100;
```

```
    localparam ALU_SUB = 5'b00101;
    localparam ALU_SUBU = 5'b10110;
    localparam ALU_SLT = 5'b10011;
    localparam ALU_SLTU = 5'b10100;
    localparam ALU_SLL = 5'b10000;
    localparam ALU_SRL = 5'b10001;
    localparam ALU_SRA = 5'b10010;
    // * I-type
    localparam ALU_ANDI = 5'b00110;
    localparam ALU_XORI = 5'b00111;
    localparam ALU_ORI = 5'b01000;
    localparam ALU_LUI = 5'b10111;


    always@(*)
    begin
        case(ALUCode)
            ALU_ADD :Result <= $signed(A) + $signed(B);
            ALU_ADDU :Result <= A + B;
            ALU_AND :Result <= A & B;
            ALU_XOR :Result <= A ^ B;
            ALU_OR  :Result <= A | B;
            ALU_NOR :Result <= ~(A | B);
            ALU_SUB :Result <= $signed(A) - $signed(B);
            ALU_SUBU :Result <= A - B;
            ALU_SLT :Result <= ($signed(A) < $signed(B)) ?1:0;
            ALU_SLTU :Result <= (A < B) ? 1 :0;
            ALU_SLL :Result <= B << A;
            ALU_SRL :Result <= B >> A;
            ALU_SRA :Result <= $signed(B) >>> A;
            ALU_ANDI :Result <= A& {16'd0,B[15:0]};
            ALU_XORI :Result <= A^ {16'd0,B[15:0]};
            ALU_ORI :Result <= A| {16'd0,B[15:0]};
            ALU_LUI :Result <= {B[15:0],16'd0};
            default :Result <= 32'b0;
        endcase
    end

endmodule
```

Listing 4: ALU.v

```
`timescale 1ns / 1ps

module ZeroTest(input [4:0]ALUCode,
                input [31:0]RsData,
                input [31:0]RtData,
                output reg Z);

    localparam alu_beq = 5'b01010;
    localparam alu_bne = 5'b01011;
    localparam alu_bgez = 5'b01100;
```

```
    localparam alu_bgtz = 5'b01101;
    localparam alu_blez = 5'b01110;
    localparam alu_bltz = 5'b01111;

    always@(*)
    begin
        case(ALUCode)
            alu_beq: Z <= &(RsData[31:0]~^RtData[31:0]);
            alu_bne: Z <= |(RsData[31:0]^RtData[31:0]);
            alu_bgez:Z <= ~RsData[31];
            alu_bgtz:Z <= ~RsData[31]&&(|RsData[31: 0]);
            alu_blez:Z <= RsData[31] || ~ (|RsData[31: 0]);
            alu_bltz:Z <= RsData[31];
            default: Z <= 1'b0;
        endcase
    end

endmodule
```

Listing 5: BranchTest.v

```
`timescale 1ns / 1ps

`define IO_REGA 23
`define IO_REGB 24
`define IO_REGC 25
`define IO_REGD 26
`define IO_REGS 27

module CPUdl(input clk,
             input switch_en,
             input seq_en,
             input [3:0] add_a,
             input [3:0] add_b,
             input [3:0] add_c,
             input [3:0] add_d,
             output [15:0] add_s);


    // * --- IF ---
    wire [31:0]NextPC_if,BranchAddress,JumpAddress,RsData_id;
    reg [31:0]PC_in;
    wire Z,J,JR;
    wire RegWrite_ex;
    wire [4:0]RegWriteAddr_ex,RsAddr_id;
    wire [31:0]ALUResult_ex;
    wire [31:0]JrAddr;
    wire forward_jr;
    assign forward_jr = RegWrite_ex&&(RegWriteAddr_ex == RsAddr_id);
    assign JrAddr   = forward_jr ? ALUResult_ex :RsData_id;

    // * MUX
```

```
32      always@(*)
33      begin
34          case({JR,J,Z})
35              3'b000:PC_in <= NextPC_if;
36              3'b001:PC_in <= BranchAddress;
37              3'b010:PC_in <= JumpAddress;
38              3'b100:PC_in <= JrAddr;
39              default:PC_in <= 32'b0;
40          endcase
41      end
42
43      // * PC
44      wire [31:0]PC;
45      assign NextPC_if = PC+4;
46      wire PC_IFWrite;
47      Reg #(.width(32))PC_reg_if(.clk(clk),.reset(1'b0),.enable(PC_IFWrite),.in(
            PC_in),.out(PC));
48
49
50      // * InstructionRom
51      wire [31:0]Instruction_if;
52      InstructionROMDl InstrUnit(.addr(PC),.dout(Instruction_if));
53
54      // * FLUSH
55      wire IF_flush;
56      assign IF_flush = JR||J||Z;
57
58
59      // * --- IF/ID ---
60      wire [31:0]NextPC_id;
61      wire [31:0]Instruction_id;
62      Reg #(.width(32))PC_if_id(.clk(clk),.reset(IF_flush),.enable(PC_IFWrite),.
            in(NextPC_if),.out(NextPC_id));
63      Reg #(.width(32))Instr_if_id(.clk(clk),.reset(IF_flush),.enable(PC_IFWrite)
            ,.in(Instruction_if),.out(Instruction_id));
64
65      // * --- ID ---
66
67      wire MemtoReg_id,RegWrite_id,MemWrite_id,MemRead_id,ALUSrcA_id,ALUSrcB_id,
            RegDst_id,Branch_id;
68      wire [1:0]LwByte_id;
69      wire [4:0]ALUCode_id;
70      Decode decoder(.Instruction(Instruction_id),.MemtoReg(MemtoReg_id),.
            RegWrite(RegWrite_id),.MemWrite(MemWrite_id),.MemRead(MemRead_id),.
            ALUCode(ALUCode_id),
71      .ALUSrcA(ALUSrcA_id),.ALUSrcB(ALUSrcB_id),.RegDst(RegDst_id),.J(J),.JR(JR)
            ,.Branch(Branch_id),.LwByte(LwByte_id));
72
73      wire  [4:0]RtAddr_id,RdAddr_id;
74      assign RsAddr_id = Instruction_id[25:21];
75      assign RtAddr_id = Instruction_id[20:16];
```

```
76      assign RdAddr_id = Instruction_id[15:11];
77
78      wire [31:0]Imm_id,Sa_id;
79      assign Imm_id = {{16{Instruction_id[15]}},Instruction_id[15:0]};
80      assign Sa_id = {27'b0,Instruction_id[10:6]};
81
82      assign BranchAddress = NextPC_id+(Imm_id<<2);
83      assign JumpAddress = {NextPC_id[31:28],Instruction_id[25:0],2'b00};
84
85      wire [31:0]RtData_id;
86      wire forward_zero_Rs,forward_zero_Rt;
87      assign forward_zero_Rs = RegWrite_ex&&(RegWriteAddr_ex == RsAddr_id);
88      assign forward_zero_Rt = RegWrite_ex&&(RegWriteAddr_ex == RsAddr_id);
89      wire [31:0]RsData_zero,RtData_zero;
90      assign RsData_zero = forward_zero_Rs ? ALUResult_ex :RsData_id;
91      assign RtData_zero = forward_zero_Rt ? ALUResult_ex :RtData_id;
92      ZeroTest zero_unit(.ALUCode(ALUCode_id),.RsData(RsData_zero),.RtData(
            RtData_zero),.Z(Z));
93
94      wire MemRead_ex;
95      wire Stall;
96      HazardDetector hazard_unit(.RegWriteAddr(RegWriteAddr_ex),.MemRead(
            MemRead_ex),.RsAddr(RsAddr_id),.RtAddr(RtAddr_id),.Stall(Stall),.
            PC_IFWrite(PC_IFWrite));
97
98      wire RegWrite_wb;
99      wire [4:0]RegWriteAddr_wb;
100     wire [31:0]RegWriteData_wb;
101     wire [31:0]RsData_temp,RtData_temp;
102     Registers register_unit(.clk(clk),.RsAddr(RsAddr_id),.RtAddr(RtAddr_id),.
            WriteData(RegWriteData_wb),.WriteAddr(RegWriteAddr_wb),.RegWrite(
            RegWrite_wb),
103     .RsData(RsData_temp),.RtData(RtData_temp));
104
105     wire RsSel,RtSel;
106     assign RsSel    = RegWrite_wb&&(~(RegWriteAddr_wb == 0))&&(RegWriteAddr_wb
            == RsAddr_id);
107     assign RtSel    = RegWrite_wb&&(~(RegWriteAddr_wb == 0))&&(RegWriteAddr_wb
            == RtAddr_id);
108     assign RsData_id = (RsSel == 1)? RegWriteData_wb :RsData_temp;
109     assign  RtData_id = (RtSel == 1)? RegWriteData_wb :RtData_temp;
110
111     // * --- ID/EX ---
112     wire MemtoReg_ex,MemWrite_ex,RegDst_ex,ALUSrcA_ex,ALUSrcB_ex;
113     wire [1:0]LwByte_ex;
114     wire [4:0]ALUCode_ex;
115     wire [31:0]Sa_ex,Imm_ex,RsData_ex,RtData_ex;
116     wire [4:0]RdAddr_ex,RtAddr_ex,RsAddr_ex;
117
118     Reg #(.width(2))LwByte_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in(
            LwByte_id),.out(LwByte_ex));
```

```
119     Reg #(.width(2))WB_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in({
            MemtoReg_id,RegWrite_id}),.out({MemtoReg_ex,RegWrite_ex}));
120     Reg #(.width(2))M_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in({
            MemWrite_id,MemRead_id}),.out({MemWrite_ex,MemRead_ex}));
121     Reg #(.width(3))EX_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in({
            RegDst_id,ALUSrcA_id,ALUSrcB_id}),.out({RegDst_ex,ALUSrcA_ex,ALUSrcB_ex
            }));
122     Reg #(.width(5))ALUCode_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in(
            ALUCode_id),.out(ALUCode_ex));
123     Reg #(.width(32))Sa_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in(Sa_id)
            ,.out(Sa_ex));
124     Reg #(.width(32))Imm_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in(Imm_id
            ),.out(Imm_ex));
125     Reg #(.width(5))RdAddr_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in(
            RdAddr_id),.out(RdAddr_ex));
126     Reg #(.width(5))RsAddr_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in(
            RsAddr_id),.out(RsAddr_ex));
127     Reg #(.width(5))RtAddr_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in(
            RtAddr_id),.out(RtAddr_ex));
128     Reg #(.width(32))RsData_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in(
            RsData_id),.out(RsData_ex));
129     Reg #(.width(32))RtData_id_ex(.clk(clk),.reset(Stall),.enable(1'b1),.in(
            RtData_id),.out(RtData_ex));
130
131     // * --- EX ---
132     wire [31:0]ALUResult_mem;
133     wire [1:0]ForwardA,ForwardB;
134     wire [4:0]RegWriteAddr_mem;
135     wire RegWrite_mem;
136
137     // * Forwarding
138     Forwarding forward_unit(.RegWrite_wb(RegWrite_wb),.RegWrite_mem(
            RegWrite_mem),.RegWriteAddr_wb(RegWriteAddr_wb),.RegWriteAddr_mem(
            RegWriteAddr_mem),
139     .RsAddr_ex(RsAddr_ex),.RtAddr_ex(RtAddr_ex),.ForwardA(ForwardA),.ForwardB(
            ForwardB));
140
141     // * MUX
142     wire [31:0]ALUSrcA_d_in,ALUSrcA_d,ALUSrcB_d,MemWriteData_ex;
143     Mux4 #(.width(32))ALUSrcA_mux4(.sel(ForwardA),.in0(RsData_ex),.in1(
            RegWriteData_wb),.in2(ALUResult_mem),.in3(0),.out(ALUSrcA_d_in));
144     assign ALUSrcA_d = (ALUSrcA_ex == 1)? Sa_ex :ALUSrcA_d_in;
145     Mux4 #(.width(32))ALUSrcB_mux4(.sel(ForwardB),.in0(RtData_ex),.in1(
            RegWriteData_wb),.in2(ALUResult_mem),.in3(0),.out(MemWriteData_ex));
146     assign ALUSrcB_d = (ALUSrcB_ex == 1)? Imm_ex :MemWriteData_ex;
147
148     assign RegWriteAddr_ex = (RegDst_ex == 1)? RdAddr_ex :RtAddr_ex;
149
150     // * ALU
151     ALU ALU_unit(.ALUCode(ALUCode_ex),.A(ALUSrcA_d),.B(ALUSrcB_d),.Result(
            ALUResult_ex));
```

```
152
153
154     // * --- EX/MEM ---
155     wire MemtoReg_mem,MemWrite_mem;
156     wire [31:0]MemWriteData_mem;
157     wire [1:0]LwByte_mem;
158
159     Reg #(.width(2))LwByte_ex_mem(.clk(clk),.reset(1'b0),.enable(1'b1),.in(
            LwByte_ex),.out(LwByte_mem));
160     Reg #(.width(3))Signal_ex_mem(.clk(clk),.reset(1'b0),.enable(1'b1),.in({
            MemtoReg_ex,RegWrite_ex,MemWrite_ex}),.out({MemtoReg_mem,RegWrite_mem,
            MemWrite_mem}));
161     Reg #(.width(32))ALUResult_ex_mem(.clk(clk),.reset(1'b0),.enable(1'b1),.in(
            ALUResult_ex),.out(ALUResult_mem));
162     Reg #(.width(32))MemWriteData_ex_mem(.clk(clk),.reset(1'b0),.enable(1'b1),.
            in(MemWriteData_ex),.out(MemWriteData_mem));
163     Reg #(.width(5))RegWriteAddr_ex_mem(.clk(clk),.reset(1'b0),.enable(1'b1),.
            in(RegWriteAddr_ex),.out(RegWriteAddr_mem));
164
165    // * --- MEM ---
166
167     // * DataRam
168
169     wire [31:0]RamOut_mem;
170     wire [4:0] seq_addr;
171
172     Sequence seq_unit(clk,seq_en,seq_addr);
173
174     wire [31:0]Ram_Addr,tmpdata;
175     assign Ram_Addr = seq_en ? {25'b0,seq_addr,2'b00} :(switch_en ? (add_a<<2)
            :ALUResult_mem);
176     DataRamDl data_unit(.clk(clk),.addr(Ram_Addr),.din(MemWriteData_mem),.we(
            MemWrite_mem),.dout(RamOut_mem),.dduaddr(switch_addr<<2),.dduout(
            tmpdata));
177     DataRamDl data_unit(.clk(clk),.addr(`IO_REGA),.din(add_a),.we(1));
178     DataRamDl data_unit(.clk(clk),.addr(`IO_REGB),.din(add_b),.we(1));
179     DataRamDl data_unit(.clk(clk),.addr(`IO_REGC),.din(add_c),.we(1));
180     DataRamDl data_unit(.clk(clk),.addr(`IO_REGD),.din(add_d),.we(1));
181     DataRamDl data_unit(.clk(clk),.addr(`IO_REGS),.din(add_s),.we(1));
182
183     // * --- MEM/WB ---
184     wire MemtoReg_wb;
185     wire [31:0]ALUResult_wb;
186     wire [31:0]RamOut_wb;
187     Reg #(.width(2))WB_mem_ex(.clk(clk),.reset(1'b0),.enable(1'b1),.in({
            RegWrite_mem,MemtoReg_mem}),.out({RegWrite_wb,MemtoReg_wb}));
188     Reg #(.width(32))ALUResult_mem_wb(.clk(clk),.reset(1'b0),.enable(1'b1),.in(
            ALUResult_mem),.out(ALUResult_wb));
189     Reg #(.width(5))RegWriteAddr_mem_wb(.clk(clk),.reset(1'b0),.enable(1'b1),.
            in(RegWriteAddr_mem),.out(RegWriteAddr_wb));
190     Reg #(.width(32))RamOut_mem_wb(.clk(clk),.reset(1'b0),.enable(1'b1),.in(
```

```
        RamOut_mem),.out(RamOut_wb));

    // * --- WB ----

    assign RegWriteData_wb = (MemtoReg_wb == 1)? RamOut_wb :ALUResult_wb;

endmodule
```

Listing 6: CPUdl.v

```
`timescale 1ns / 1ps
module DataRamDl(
    input clk,
    input [31:0]addr,
    input [31:0]din,
    input we,
    input [31:0] dduaddr,
    output [31:0] dduout,
    output [31:0]dout
    );

    wire [4:0]A;
    assign A=(addr>>2);
    dist_mem_gen_1 Mem(.a(A),.dpra(dduaddr),.d(din),.we(we),.clk(clk),.spo(dout
        ),.dpo(dduout));

endmodule
```

Listing 7: DataRamDl.v

```
`timescale 1ns / 1ps

module Decode(input [31:0]Instruction,
              output MemtoReg,
              output RegWrite,
              output MemWrite,
              output MemRead,
              output reg[4:0] ALUCode,
              output ALUSrcA,
              output ALUSrcB,
              output RegDst,
              output J,
              output JR,
              output Branch,
              output reg[1:0]LwByte);

//Instruction Field
wire [5:0] op;
wire [4:0] rt;
//wire [4:0] rs;
wire [5:0]funct;
assign op   = Instruction[31:26];
```

```
23 assign rt  = Instruction[20:16];
24 assign funct = Instruction[5:0];
25
26
27 //R_type
28 localparam R_type_op = 6'b000000;
29 localparam ADD_funct = 6'b100000;
30 localparam ADDU_funct = 6'b100001;
31 localparam AND_funct = 6'b100100;
32 localparam XOR_funct = 6'b100110;
33 localparam OR_funct = 6'b100101;
34 localparam NOR_funct = 6'b100111;
35 localparam SUB_funct = 6'b100010;
36 localparam SUBU_funct = 6'b100011;
37 localparam SLT_funct = 6'b101010;
38 localparam SLTU_funct = 6'b101011;
39 localparam SLL_funct = 6'b000000;
40 localparam SLLV_funct = 6'b000100;
41 localparam SRL_funct = 6'b000010;
42 localparam SRLV_funct = 6'b000110;
43 localparam SRA_funct = 6'b000011;
44 localparam SRAV_funct = 6'b000111;
45 localparam JR_funct = 6'b001000;
46 //R_type1
47 wire ADD,ADDU,AND,NOR,OR,SLT,SLTU,SUB,SUBU,XOR,SLLV, SRAV, SRLV,R_type1;
48 assign ADD   = (op == R_type_op)&&(funct == ADD_funct);
49 assign ADDU  = (op == R_type_op)&&(funct == ADDU_funct);
50 assign AND   = (op == R_type_op)&&(funct == AND_funct);
51 assign NOR   = (op == R_type_op)&&(funct == NOR_funct);
52 assign OR    = (op == R_type_op)&&(funct == OR_funct);
53 assign SLT   = (op == R_type_op)&&(funct == SLT_funct);
54 assign SLTU  = (op == R_type_op)&&(funct == SLTU_funct);
55 assign SUB   = (op == R_type_op)&&(funct == SUB_funct);
56 assign SUBU  = (op == R_type_op)&&(funct == SUBU_funct);
57 assign XOR   = (op == R_type_op)&&(funct == XOR_funct);
58 assign SLLV  = (op == R_type_op)&&(funct == SLLV_funct);
59 assign SRAV  = (op == R_type_op)&&(funct == SRAV_funct);
60 assign SRLV  = (op == R_type_op)&&(funct == SRLV_funct);
61 assign R_type1 = ADD||ADDU||AND||NOR||OR||SLT||SLTU||SUB||SUBU||XOR||SLLV||
       SRAV||SRLV;
62 //R_type2
63 wire SLL,SRA,SRL,R_type2;
64 assign SLL   = (op == R_type_op)&&(funct == SLL_funct)&&(|Instruction); //巨
       坑，与nop的区
65 assign SRA   = (op == R_type_op)&&(funct == SRA_funct);
66 assign SRL   = (op == R_type_op)&&(funct == SRL_funct);
67 assign R_type2 = SLL||SRA||SRL;
68
69
70 //Branch
71 localparam BEQ_op = 6'b000100;
```

```
72  localparam BNE_op = 6'b000101;
73  localparam BGEZ_op = 6'b000001;
74  localparam BGEZ_rt = 5'b00001;
75  localparam BGTZ_op = 6'b000111;
76  localparam BGTZ_rt = 5'b00000;
77  localparam BLEZ_op = 6'b000110;
78  localparam BLEZ_rt = 5'b00000;
79  localparam BLTZ_op = 6'b000001;
80  localparam BLTZ_rt = 5'b00000;
81
82  wire BEQ,BGEZ,BGTZ,BLEZ,BLTZ,BNE;
83  assign BEQ  = (op == BEQ_op);
84  assign BNE  = (op == BNE_op);
85  assign BGEZ = (op == BGEZ_op)&&(rt == BGEZ_rt);
86  assign BGTZ = (op == BGTZ_op)&&(rt == BGTZ_rt);
87  assign BLEZ = (op == BLEZ_op)&&(rt == BLEZ_rt);
88  assign BLTZ = (op == BLTZ_op)&&(rt == BLTZ_rt);
89  assign Branch = BEQ||BNE||BGEZ||BGTZ||BLEZ||BLTZ;
90
91
92  //I
93  localparam ADDI_op = 6'b001000;
94  localparam ADDIU_op = 6'b001001;
95  localparam ANDI_op = 6'b001100;
96  localparam XORI_op = 6'b001110;
97  localparam ORI_op = 6'b001101;
98  localparam SLTI_op = 6'b001010;
99  localparam SLTIU_op = 6'b001011;
100 localparam LUI_op = 6'b001111;
101 localparam LUI_rs = 5'b00000;
102 wire ADDI,ADDIU,ANDI,XORI,ORI,SLTI,SLTIU,LUI,I_type;
103 assign ADDI  = (op == ADDI_op);
104 assign ADDIU = (op == ADDIU_op);
105 assign ANDI  = (op == ANDI_op);
106 assign XORI  = (op == XORI_op);
107 assign SLTI  = (op == SLTI_op);
108 assign SLTIU = (op == SLTIU_op);
109 assign ORI   = (op == ORI_op);
110 assign LUI   = (op == LUI_op);
111 assign I_type = ADDI||ADDIU||ANDI||XORI||ORI||SLTI||SLTIU||LUI;
112
113
114 //SW,LW
115 localparam SW_op = 6'b101011;
116 localparam LW_op = 6'b100011;
117 wire SW,LW;
118 assign SW = (op == SW_op);
119 assign LW = (op == LW_op) ;
120
121 //LH,LB
122 localparam LH_op = 6'b100001;
```

```
123 localparam LB_op = 6'b100000;
124 wire LH,LB;
125 assign LH = (op == LH_op);
126 assign LB = (op == LB_op) ;
127
128 //J
129 localparam J_op = 6'b000010;
130 assign J      = (op == J_op);
131
132 //JR
133 assign JR = (op == R_type_op)&&(funct == JR_funct);
134
135
136 //输出控制信号
137 assign RegWrite = LB||LH||LW||R_type1||R_type2||I_type;
138 assign RegDst = R_type1||R_type2;
139 assign MemWrite = SW;
140 assign MemRead = LW||LH||LB;
141 assign MemtoReg = LW||LH||LB;
142 assign ALUSrcA = R_type2;
143 assign ALUSrcB = LB||LH||LW||SW||I_type;
144
145 always@(*)
146 begin
147     if (op == LW_op)
148         LwByte = 2'b00;
149     else if (op == LH_op)
150         LwByte = 2'b01;
151     else if (op == LB_op)
152         LwByte = 2'b10;
153     else
154         LwByte = 2'b00;
155 end
156
157
158 //ALUCode
159 localparam alu_add = 5'b00000;
160 localparam alu_and = 5'b00001;
161 localparam alu_xor = 5'b00010;
162 localparam alu_or = 5'b00011;
163 localparam alu_nor = 5'b00100;
164 localparam alu_sub = 5'b00101;
165 localparam alu_andi = 5'b00110;
166 localparam alu_xori = 5'b00111;
167 localparam alu_ori = 5'b01000;
168 localparam alu_beq = 5'b01010;
169 localparam alu_bne = 5'b01011;
170 localparam alu_bgez = 5'b01100;
171 localparam alu_bgtz = 5'b01101;
172 localparam alu_blez = 5'b01110;
173 localparam alu_bltz = 5'b01111;
```

```
174 localparam alu_sll = 5'b10000;
175 localparam alu_srl = 5'b10001;
176 localparam alu_sra = 5'b10010;
177 localparam alu_slt = 5'b10011;
178 localparam alu_sltu = 5'b10100;
179 localparam alu_addu = 5'b10101;
180 localparam alu_subu = 5'b10110;
181 localparam alu_lui = 5'b10111;
182
183
184
185 always@(*)
186 begin
187     if (op == R_type_op)
188     begin
189         case(funct)
190             ADD_funct :ALUCode <= alu_add;
191             ADDU_funct :ALUCode <= alu_addu;
192             AND_funct :ALUCode <= alu_and;
193             XOR_funct :ALUCode <= alu_xor;
194             OR_funct  :ALUCode <= alu_or;
195             NOR_funct :ALUCode <= alu_nor;
196             SUB_funct :ALUCode <= alu_sub;
197             SUBU_funct :ALUCode <= alu_subu;
198             SLT_funct :ALUCode <= alu_slt;
199             SLTU_funct :ALUCode <= alu_sltu;
200             SLL_funct :ALUCode <= alu_sll;
201             SLLV_funct :ALUCode <= alu_sll;
202             SRL_funct :ALUCode <= alu_srl;
203             SRLV_funct :ALUCode <= alu_srl;
204             SRA_funct :ALUCode <= alu_sra;
205             default   :ALUCode <= alu_sra;
206         endcase
207     end
208     else
209     begin
210         case(op)
211             BEQ_op :ALUCode <= alu_beq;
212             BNE_op :ALUCode <= alu_bne;
213             BGEZ_op :begin
214                 if (rt == BGEZ_rt) begin
215                     ALUCode <= alu_bgez;
216                 end
217             end
218             BGTZ_op :begin
219                 if (rt == BGTZ_rt) begin
220                     ALUCode <= alu_bgtz;
221                 end
222             end
223             BLEZ_op :begin
224                 if (rt == BLEZ_rt) begin
```

```
                ALUCode <= alu_blez;
            end
        end
        BLTZ_op :begin
            if (rt == BLTZ_rt) begin
                ALUCode <= alu_bltz;
            end
        end
        ADDI_op :ALUCode <= alu_add;
        ADDIU_op:ALUCode <= alu_addu;
        ANDI_op :ALUCode <= alu_andi;
        XORI_op :ALUCode <= alu_xori;
        ORI_op :ALUCode <= alu_ori;
        SLTI_op :ALUCode <= alu_slt;
        SLTIU_op:ALUCode <= alu_sltu;
        LUI_op :ALUCode <= alu_lui;
        SW_op  :ALUCode <= alu_add;
        LW_op  :ALUCode <= alu_add;
        LH_op  :ALUCode <= alu_add;
        LB_op  :ALUCode <= alu_add;
        default :ALUCode <= alu_add;
    endcase
  end
end
endmodule
```

Listing 8: Decode.v

```
`timescale 1ns / 1ps

module display(clk,
               data,
               seg,
               an);
  input clk;
  input [31:0]data;
  output reg [3:0]an;
  output reg [7:0]seg;

  reg [3:0]disp;

  reg [18:0]count = 0;
  always@(posedge clk)
      count <= count+1;

  always@(posedge clk)
      case(count[17:15])
          3'b000:begin
              an  = 8'b11111110;
              disp = data[3:0];
          end
          3'b001:begin
```

```
25              an   = 8'b11111101;
26              disp = data[7:4];
27          end
28          3'b010:begin
29              an   = 8'b11111011;
30              disp = data[11:8];
31          end
32          3'b011:begin
33              an   = 8'b11110111;
34              disp = data[15:12];
35          end
36          3'b100:begin
37              an   = 8'b11101111;
38              disp = data[19:16];
39          end
40          3'b101:begin
41              an   = 8'b11011111;
42              disp = data[23:20];
43          end
44          3'b110:begin
45              an   = 8'b10111111;
46              disp = data[27:24];
47          end
48          3'b111:begin
49              an   = 8'b01111111;
50              disp = data[31:28];
51          end
52      endcase
53
54  always @(disp)
55      case(disp)
56          0     :seg = 8'b11000000;
57          1     :seg = 8'b11111001;
58          2     :seg = 8'b10100100;
59          3     :seg = 8'b10110000;
60          4     :seg = 8'b10011001;
61          5     :seg = 8'b10010010;
62          6     :seg = 8'b10000010;
63          7     :seg = 8'b11111000;
64          8     :seg = 8'b10000000;
65          9     :seg = 8'b10010000;
66          10    :seg = 8'b10001000;
67          11    :seg = 8'b10000011;
68          12    :seg = 8'b11000110;
69          13    :seg = 8'b10100001;
70          14    :seg = 8'b10000110;
71          15    :seg = 8'b10001110;
72          default:seg = 8'b11000000;
73      endcase
74  endmodule
```

Listing 9: download.v

```
`timescale 1ns / 1ps

module Forwarding(input RegWrite_wb,
                  input RegWrite_mem,
                  input [4:0]RegWriteAddr_wb,
                  input [4:0]RegWriteAddr_mem,
                  input [4:0]RsAddr_ex,
                  input [4:0]RtAddr_ex,
                  output [1:0]ForwardA,
                  output [1:0]ForwardB);

    assign ForwardA[0] = RegWrite_wb&&(RegWriteAddr_wb! = 0)&&(!((
        RegWriteAddr_mem == RsAddr_ex)&&(RegWrite_mem)))&&(RegWriteAddr_wb ==
        RsAddr_ex);
    assign ForwardA[1] = RegWrite_mem&&(RegWriteAddr_mem! = 0)&&(
        RegWriteAddr_mem == RsAddr_ex);
    assign ForwardB[0] = RegWrite_wb&&(RegWriteAddr_wb! = 0)&&(!((
        RegWriteAddr_mem == RtAddr_ex)&&(RegWrite_mem)))&&(RegWriteAddr_wb ==
        RtAddr_ex);
    assign ForwardB[1] = RegWrite_mem&&(RegWriteAddr_mem! = 0)&&(
        RegWriteAddr_mem == RtAddr_ex);
endmodule
```

Listing 10: Forwarding.v

```
`timescale 1ns / 1ps

module HazardDetector(input [4:0]RegWriteAddr,
                      input MemRead,
                      input [4:0]RsAddr,
                      input [4:0]RtAddr,
                      output Stall,
                      output PC_IFWrite);

    assign Stall    = (MemRead&&((RegWriteAddr == RsAddr)||(RegWriteAddr ==
        RtAddr)));
    assign PC_IFWrite = ~Stall;
endmodule
```

Listing 11: HazardDetector.v

```
`timescale 1ns / 1ps

module InstructionROMDl(input [31:0]addr,
                        output [31:0]dout);

    wire [4:0]A;
    assign A = (addr>>2);
    dist_mem_gen_0 Instr(.a(A),.spo(dout));
```

```
endmodule
```

Listing 12: InstructionRomDl.v

```
`timescale 1ns / 1ps

module Mux4 #(parameter width = 32)
           (  input [1:0]sel,
            input [width-1:0]in0,
            input [width-1:0]in1,
            input [width-1:0]in2,
            input [width-1:0]in3,
            output reg[width-1:0]out);
    always@(*)
    begin
        case(sel)
            2'b00:out <= in0;
            2'b01:out <= in1;
            2'b10:out <= in2;
            2'b11:out <= in3;
            default:out <= 0;
        endcase
    end
endmodule
```

Listing 13: Mux4.v

```
`timescale 1ns / 1ps

module Reg #(parameter width = 32)
           (   input clk,
            input reset,
            input enable,
            input [width-1:0]in,
            output reg[width-1:0]out);
    always@(posedge clk)
        if (reset)
            out <= {width{1'b0}};
        else if (enable)
            out <= in;
        else
            out <= out;

    initial
        out = 0;
endmodule
```

Listing 14: Reg.v

```
`timescale 1ns / 1ps
module Registers(
    input clk,
```

```
    input [4:0]RsAddr,
    input [4:0]RtAddr,
    input [31:0]WriteData,
    input [4:0]WriteAddr,
    input RegWrite,
    output [31:0]RsData,
    output [31:0]RtData
    );

    reg [31:0]Regs[0:31];

    assign RsData=(RsAddr==5'b0) ? 32'b0 :Regs[RsAddr];
    assign RtData=(RtAddr==5'b0) ? 32'b0 :Regs[RtAddr];

    always@(posedge clk)
    begin
        if(RegWrite)
            Regs[WriteAddr]<=WriteData;
    end

    initial
    begin
       Regs[0]=0;
        Regs[1]=0;
        Regs[2]=0;
        Regs[3]=0;
        Regs[4]=0;
        Regs[5]=0;
        Regs[6]=0;
        Regs[7]=0;
        Regs[8]=0;
        Regs[9]=0;
        Regs[10]=0;
        Regs[11]=0;
        Regs[12]=0;
        Regs[13]=0;
        Regs[14]=0;
        Regs[15]=0;
        Regs[16]=0;
        Regs[17]=0;
        Regs[18]=0;
        Regs[19]=0;
        Regs[20]=0;
        Regs[21]=0;
        Regs[22]=0;
        Regs[23]=0;
        Regs[24]=0;
        Regs[25]=0;
        Regs[26]=0;
        Regs[27]=0;
        Regs[28]=0;
```

```
55          Regs[29]=0;
56          Regs[30]=0;
57          //Regs[31]=0;
58      end
59
60
61  endmodule
```

Listing 15: Registers.v

```
1   `timescale 1ns / 1ps
2   module Sequence(
3       input clk,
4       input en,
5       output [4:0]out
6       );
7
8       reg [4:0]addr=0;
9       assign out=addr;
10      reg [25:0]count=0;
11
12      always@(posedge clk)
13      if(en)
14          count<=count+1;
15
16      always@(posedge count[25])
17          addr<=addr+1;
18
19  endmodule
```

Listing 16: Sequence.v

```
1   `timescale 1ns / 1ps
2
3   module top(
4       input clk,
5       input switch_en,
6       input seq_en,
7       input [3:0] adda,
8       input [3:0] addb,
9       input [3:0] addc,
10      input [3:0] addd,
11      output [3:0] an,
12      output [7:0] seg
13      );
14
15      wire [31:0] data;
16
17      CPUdl  cpu (clk,switch_en,seq_en,adda,addb,addc,addd,data);
18      display segs (clk,data,seg,an);
19  endmodule
```

Listing 17: top.v

10 性能评测

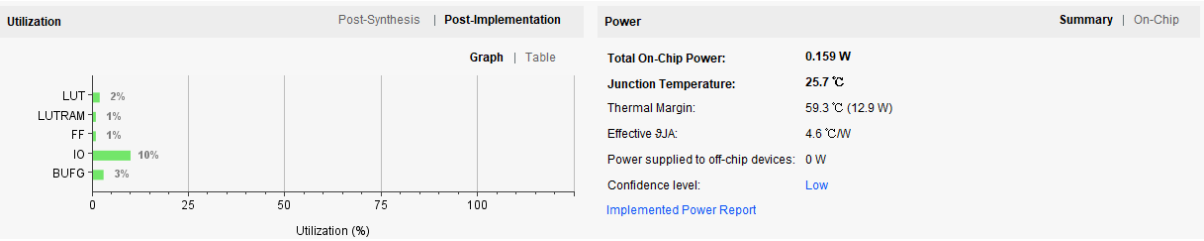

Figure 3: Performance1

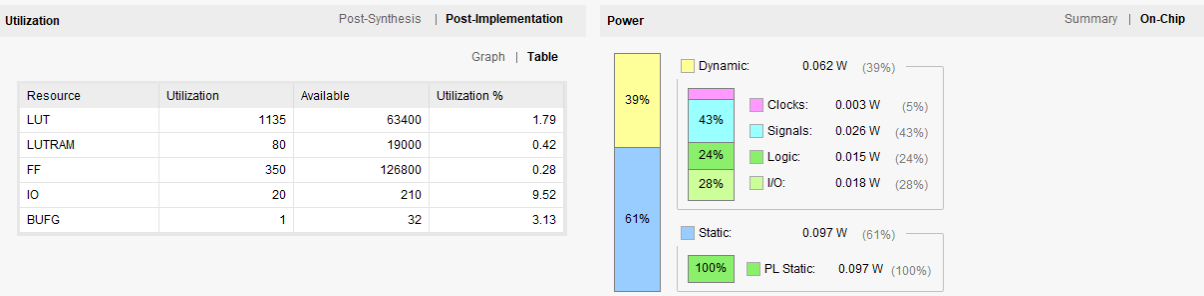

Figure 4: Performance2

11 自我总结与建议

经过一个学期的磨炼……终于完成了最后一个实验，当然在经过这个学期的学习，确实学到了不少东西。特别是 Lab4 和 Lab5 收货尤其大，这两次实验我基本采取了和其他同学不同的试验方法。Lab4 不是采取状态机的写法，而是采用了纯逻辑的写法，代码量大大增加，但是具有很强的移植能力，可以稍加转化就可以在我 Lab6 的实验程序上运行。Lab5 则是采用由果到因倒序编程，先假设我写好了模块，并且完成连接，最后再一一填充模块。体会了一下自顶向下设计的感受。本次实验我和大部分同学一样采用了自底向上的设计方法。

总体来说我还是非常喜欢 COD 实验的，这个学期的实验，好玩，有趣，有区分度。当然我写报告写得也很痛快，不想上个学期，写个报告就是超超要求，贴贴代码，这个学期老师要求写写思路，写写设计，不仅给我们提供了很好的思路整理机会，还间接带我们一起对只是进行了复习，本学期再写报告中也得到了不少收获！