

# 实验六 CPU综合设计

## 一、实验目的

1. 掌握复杂系统设计方法。
2. 深刻理解计算机系统硬件原理。

## 二、实验内容

1. 设计一个基于MIPS指令集的CPU，支持以下指令：{add, sub, addi, lw, sw, beq, j, nop};
2. CPU需要包含寄存器组、RAM模块、ALU模块、指令译码模块；
3. 该CPU能运行基本的汇编指令；（D~C+）  
以下为可选内容：
4. 实现多周期CPU（B-~B+）；
5. 实现以下高级功能之一（A-~A+）：
  1. 实现5级流水线CPU；
  2. 实现超标量；
  3. 实现4路组相联缓存；
6. 可基于RISC V、ARM指令集实现。  
如发现代码为抄袭代码，成绩一律按不及格处理。

## 三、实验要求

编写相应测试程序，完成所有指令测试。

## 四、实验代码及结果

### 主要模块简述

1. CU模块

```

21 module CU(State, FuncCode, OP, PCWr, PCWrCond, IorD, MemRd, MemWr, IRWr, MemtoReg, PCSrc, ALUOp, ALUSrcB, ALUSrcA, RegWr, RegDst, NextState);
22 input [3:0] State;
23 input [5:0] FuncCode;
24 input [5:0] OP;
25 output reg PCWr, PCWrCond, IorD, MemRd, MemWr, IRWr, MemtoReg, ALUSrcA, RegWr, RegDst;
26 output reg [1:0] PCSrc, ALUOp, ALUSrcB;
27 output reg [3:0] NextState;
28
29 always@(State)
30 begin
31     case(State)
32     0:NextState<=1;
33     1: begin
34         case(OP)
35         6'b100011:NextState<=2;//lw
36         6'b101011:NextState<=2;//sw
37         6'b000000:
38         begin
39             case(FuncCode)
40             6'b000000:NextState<=0;//nop
41             default:NextState<=6;//R-type
42             endcase
43         end
44         6'b000100:NextState<=8;//bep
45         6'b000010:NextState<=9;//jump
46         6'b001000:NextState<=10;//addi
47         default:NextState<=4'bXXXX;
48         endcase
49     end
50
51     2: begin
52         if(OP == 6'b100011) NextState<=3;//lw
53         else if(OP == 6'b101011) NextState<=5;//sw
54         end
55     3:NextState<=4;4:NextState<=0;5:NextState<=0;6:NextState<=7;
56     7:NextState<=0;8:NextState<=0;9:NextState<=0;10:NextState<=11;
57     11:NextState<=0;default:NextState<=0;
58     endcase
59 end
60
61 always@(State)
62 begin
63     case(State)
64     0: outputs = 13'b1001010000010;
65     1: outputs = 13'b0000000000110;
66     2: outputs = 13'b0000000000101;
67     3: outputs = 13'b0011000000000;
68     4: outputs = 13'b0000001000010;
69     5: outputs = 13'b0010100000000;
70     6: outputs = 13'b0000000010100;
71     7: outputs = 13'b0000000000011;
72     8: outputs = 13'b0100000010100;
73     9: outputs = 13'b1000000100000;
74     10: outputs = 13'b0000000000101;
75     11: outputs = 13'b0000000000010;
76     default: outputs = 13'b0000000000000;
77     endcase
78     {PCWr, PCWrCond, IorD, MemRd, MemWr, IRWr, MemtoReg, PCSrc, ALUOp, ALUSrcB, ALUSrcA, RegWr, RegDst} = outputs;
79 end
80
81 endmodule
82

```

解析：

### 1. 模块声明：

1. 输入：State（当前状态）、FuncCode（功能码）、OP（操作码）。
2. 输出：PCWr、PCWrCond、IorD、MemRd、MemWr、IRWr、MemtoReg、ALUSrcA、RegWr、RegDst等信号。NextState是下一个状态的输出。

### 2. 状态转移逻辑：根据当前状态和操作码/功能码确定下一个状态。

### 3. 时钟周期控制：根据当前状态设置控制信号，以控制不同的时钟周期的行为。

### 4. 状态转移和时钟周期控制的详细说明：

状态0是初始状态，下一个状态是1。

状态1表示取指译码，根据操作码进一步判断下一个状态。

状态2表示加载存储器地址，根据操作码判断是lw还是sw，设置相应的下一个状态。  
状态3和状态5表示lw和sw的子状态，根据具体情况设置下一个状态。  
状态6和状态7表示R-type指令（如add、sub）的子状态，分别计算运算结果和将结果写回寄存器组。  
状态8表示bep指令的子状态，进行相减比较。  
状态9表示jump指令的子状态，直接跳转。  
状态10表示addi指令的子状态，计算相加结果。  
状态11表示addi指令的写回寄存器组子状态。

## 5. 信号控制：

控制信号根据状态设置，包括PCWr、PCWrCond、IorD、MemRd、MemWr、IRWr、MemtoReg、ALUSrcA、RegWr、RegDst等。  
这些信号在每个时钟周期控制CPU的不同阶段的操作。

## 2. ALU模块

```
21 module ALU_CU (ALUOp, FuncCode, Out);
22
23 input [1:0] ALUOp;
24 input [5:0] FuncCode;
25 output reg [2:0] Out;
26
27 always @*
28 case (ALUOp)
29     2'b00: Out = 3'b000;
30     2'b01: Out = 3'b001;
31     2'b10:
32     begin
33         case (FuncCode)
34             6'b100000: Out = 3'b000; //add
35             6'b100010: Out = 3'b001; //sub
36             6'b100100: Out = 3'b011; //and
37             6'b100101: Out = 3'b100; //or
38             default: Out = 3'bxxx;
39         endcase
40     end
41 endcase
42
43 endmodule
```

该模块根据输入的 ALUOp 和 FuncCode 生成相应的 ALU 控制信号（Out），以指导 ALU 进行特定的运算操作。ALUOp 为 00 时表示进行加法操作，为 01 时表示进行减法操作，为 10 时表示执行 R-type 操作，根据 FuncCode 进一步选择具体的操作。这个模块通常与 ALU 模块结合使用，以完成指令的运算功能。

## 3. Register File模块

```

23 `define DATA_WIDTH 32
24 module RegFile
25     #(parameter ADDR_SIZE = 5)
26     (input CLK, WE,
27      input [ADDR_SIZE-1:0] RA1, RA2, WA,
28      input [`DATA_WIDTH-1:0] WD,
29      output [`DATA_WIDTH-1:0] RD1, RD2);
30
31     integer i;
32     reg [`DATA_WIDTH-1:0] rf [2 ** ADDR_SIZE-1:0];
33     initial begin
34         for(i = 0; i <= 2 ** ADDR_SIZE-1; i = i+1) rf[i] = 0;
35         rf[28] = 32'h10000000;
36         rf[29] = 32'h7fffffc;
37     end
38
39     always@(posedge CLK)
40     if (WE) rf[WA] <= WD;
41     assign RD1 = (RA1 != 0) ? rf[RA1] : 0;
42     assign RD2 = (RA2 != 0) ? rf[RA2] : 0;
43 endmodule

```

这段Verilog代码实现了一个32位的寄存器文件模块（Register File），包括读写地址、写数据、写使能、时钟信号以及复位信号等输入。其中，初始时将第一个寄存器的值初始化为0，根据读取地址将对应的寄存器数据输出，并在时钟上升沿时，若写使能信号有效，则将写入数据写入指定的寄存器地址。此外，当复位信号触发时，所有寄存器的值被重置为0。

#### 4. PC模块

```

module reg_PC(clock, rst, ena_W, In, Out);

input clock;
input rst;
input ena_W;
input [31:0] In;
output [31:0] Out;

reg [31:0] PC;

always @(posedge rst) PC<=32'h00400000;

always @(posedge clock)
begin
    if (ena_W) PC<=In;
end

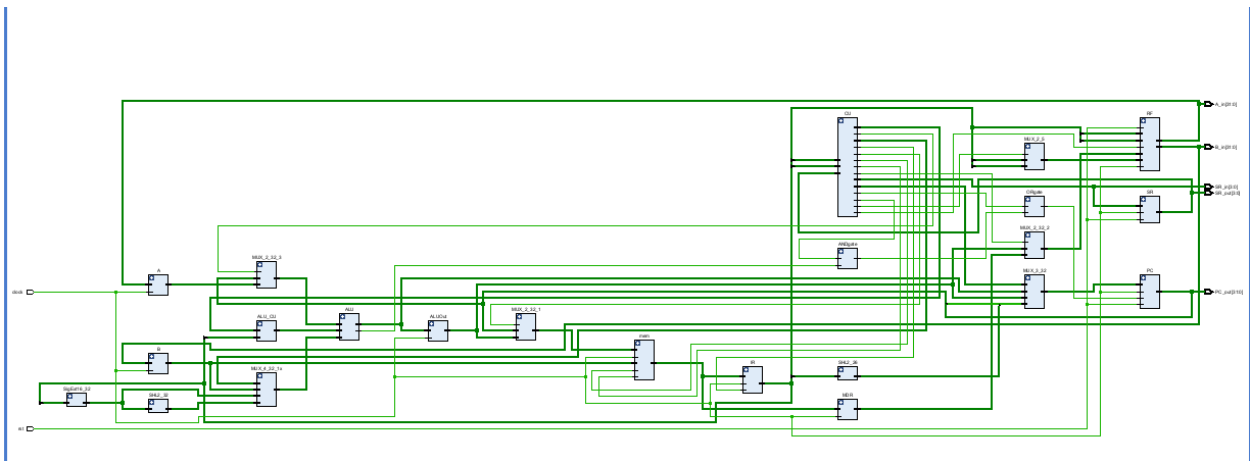
assign Out=PC;

endmodule

```

该模块实现了一个32位的PC模块，包括读写地址、写数据、写使能、时钟信号以及复位信号等输入。其中，初始时将PC的值初始化为0，根据读取地址将对应的PC数据输出，并在时钟上升沿时，若写使能信号有效，则将写入数据写入指定的PC地址。此外，当复位信号触发时，PC的值被重置为0。指令存储在内存中地址为0x00400000的位置，因此PC的初始值为0x00400000。

#### 5. RTL仿真



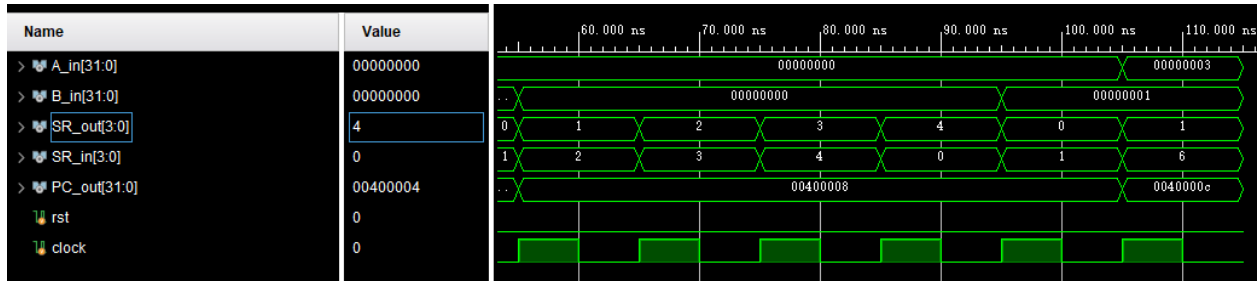
## 指令测试

### 1. 指令:

```
lw, r[4]=mem[3]=3
lw, r[1]=mem[1]=1
sub, r[2]=r[4]-r[1]=2
```






### 2. 具体以sub, r[2]=r[4]-r[1]=2句来分析

### 3. 指令周期0



在此指令周期执行取指令操作






Name	Value
ena_W	0
ena_R	1
clock	0
> addr[31:0]	00400008
> data_W[31:0]	00000000
> data_R[31:0]	00811022
> mem[1023:0][31:0]	XXXXXXXX,XXXX
> i[31:0]	512

Name	Value	Data Type
 clock	1	Logic
 ena_W	1	Logic
>  In[31:0]	00811022	Array
>  Out[31:0]	8c010001	Array
>  IR[31:0]	8c010001	Array

上图为mem模块和IR模块，

mem: input的addr为0x00400008，即PC的值，output的data为指令的值，即sub指令的值：0x00811022

IR: input的data为mem模块的output的data，即sub指令的值：0x00811022，output会在下一个时钟周期输出，即指令周期1输出。

Name	Value	Data Type
>  ALUctl[2:0]	0	Array
>  A[31:0]	00400008	Array
>  B[31:0]	00000004	Array
>  ALUout[31:0]	0040000c	Array
 Zero	0	Logic

同时ALU模块进行加法操作，将PC的值加4，即0x0040000c，作为下一条指令的地址默认值，如果下一条指令是跳转指令，会通过一个多路选择器修改PC的值。

#### 4. 指令周期1

在此指令周期执行译码操作

Name	Value	Data Type
> State[3:0]	1	Array
> FuncCode[5:0]	22	Array
> OP[5:0]	22	Array
PCWr	0	Logic
PCWrCond	0	Logic
IorD	0	Logic
MemRd	0	Logic
MemWr	0	Logic
IRWr	0	Logic
MemtoReg	0	Logic
ALUSrcA	0	Logic
RegWr	0	Logic
RegDst	0	Logic
> PCSrc[1:0]	0	Array
> ALUOp[1:0]	0	Array
> ALUSrcB[1:0]	3	Array
> NextState[3:0]	6	Array












FuncCode为0x22，即sub指令的功能码，根据FuncCode和OP，可以确定下一个状态为6，即执行运算操作。

Name	Value	Data Type
/sim_CPU/CPU/MDR/clock		Logic
> In[31:0]	00811022	Array
> Out[31:0]	00811022	Array
> reg32[31:0]	00811022	Array






上图为MDR模块，input的data为mem模块的output的data，即sub指令的值：

## 5. 指令周期6

在此指令周期执行取数和运算操作

>  Read1[4:0]	02
>  Read2[4:0]	02
>  WriteReg[4:0]	01
>  WriteData[31:0]	00404094
 RegWr	0
 clock	1
 RESET	0
>  Data1[31:0]	00000003
>  Data2[31:0]	00000001
>  i[31:0]	32
>  RF[31:0][31:0]	00000000,00000000












上图为RegFile模块，两个寄存器已经将数据读出，分别为r[4]和r[1]，即3和1

Name	Value	Data Type ^
>  ALUctl[2:0]	1	Array
>  A[31:0]	00000003	Array
>  B[31:0]	00000001	Array
>  ALUout[31:0]	00000002	Array
 Zero	0	Logic

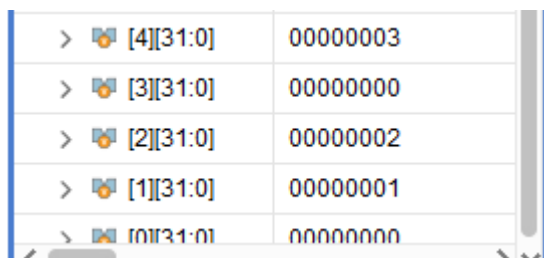
上图为ALU模块，将r[4]和r[1]相减，得到r[2]的值，即2

## 6. 指令周期7

该指令周期执行写回操作

>  Read1[4:0]	02
>  Read2[4:0]	02
>  WriteReg[4:0]	02
>  WriteData[31:0]	00000002
 RegWr	1
 clock	1
 RESET	0
>  Data1[31:0]	00000003
>  Data2[31:0]	00000001
>  i[31:0]	32
✓  RF[31:0][31:0]	00000000,00000000





> [4][31:0]	00000003
> [3][31:0]	00000000
> [2][31:0]	00000002
> [1][31:0]	00000001
> [0][31:0]	00000000

上图为RegFile模块，将r[2]的值写入r[2]寄存器中，即2

## 五、调试和心得体会

1. 本次实验的难点在于对CPU的整体结构的理解，以及对每个模块的功能的理解，以及各个模块之间的联系。
2. 在实验过程中，我遇到了一些问题，例如在理解CPU的整体结构和各个模块的功能时，我感到有些困难。我通过查阅相关资料，反复阅读实验指导书，以及和同学、老师的讨论，逐渐理解了这些概念和原理。在实际操作过程中，我也遇到了一些问题，例如在编写代码时，我发现有些部分的逻辑不够清晰，导致我无法得到预期的结果。我通过反复检查代码，找出了问题所在，并进行了修正。
3. 通过这次实验，我深入理解了CPU的结构和工作原理，以及各个模块的功能和相互之间的联系。我也学到了如何通过调试来找出和解决问题。这些都是非常宝贵的经验。
4. 对于这个实验，我认为可以在实验指导书中提供更多的示例和解释，以帮助我们更好地理解 and 掌握这些知识。
5. 总的来说，这次实验虽然有些难度，但是我从中学到了很多。我会继续努力，提高自己的理论知识和实践能力。