兰州大学

课 程 设 计

(本 科 生)

设 计 标 题 基于 MIPS 架构的单周期 CPU

成 员 姓 名 郑子安 张鑫 朱廷渊 杨宗强 刘波 李恩熠

完 成 时 间 2020 年 6 月 13 日

课 程 名 称 计算机组成原理

学 生 年 级 2018

兰州大学信息科学与工程学院

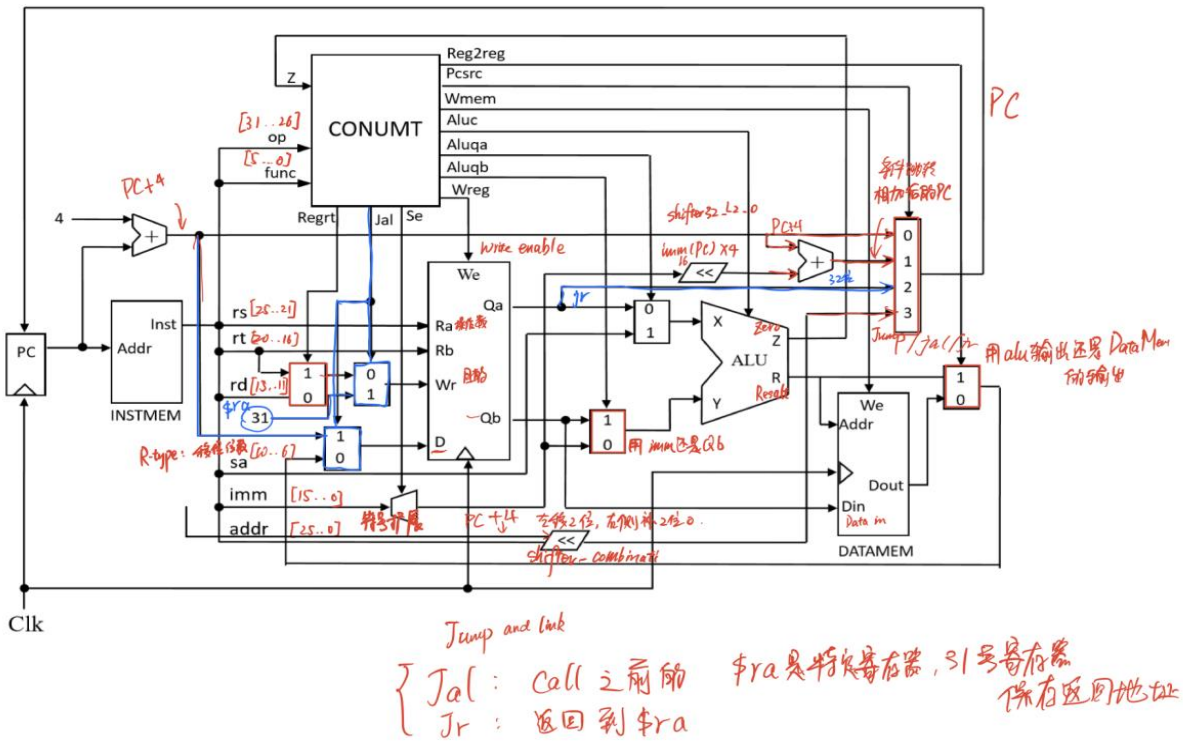
CPU 模块设计方案

```
CPU.v                          -- 顶层模块，负责将所有子模块连接在一起
├──PC_0.v                      -- PC寄存器
├──PCadd4_0.v                  -- 负责PC+4的加法器
├  └──adder.v                  -- 在上一次ALU大作业中实现的32位先行进位加法器
├──INSTMEM_0.v                 -- 指令存储器
├──CONUNIT_0.v                 -- 控制单元
├──MUX2X5_1.v                  -- 根据指令类型（R型和I型），选择rd和rt作为通用寄存器组Wr输入端
├──MUX2X5_2.v                  -- 根据Jal指令控制信号选择将常数31送入通用寄存器组的Wr输入端
├──MUX2X32_3.v                 -- 选择通用寄存器组写入端，是ALU输出还是数据存储器的输出
├──MUX2X32_4.v                 -- 用于Jal指令，选择是否把PC+4的值存入通用寄存器组
├──EXT16T32_0.v                -- I型指令立即数扩展模块，对立即数进行0扩展和符号扩展
├──SHIFTER_COMBINATION_0.v     -- 用于J型指令，实现指令地址的拼接
├──SHIFTER32_L2_0.v            -- 左移两位模块，用于对扩展后的立即数*4
├──REGFILE_0.v                 -- 通用寄存器组
├──ALU_0.v                     -- 逻辑运算单元
├  └──as_32.v                  -- 在上一次ALU大作业中实现的32位先行进位加法器
├  └──shift.v                  -- 32位移位单元，可以实现逻辑/算数左右移位0~31位
├  └──select.v                 -- 根据ALU控制信号Aluc选择输出的结果
├──DATAMEM_0.v                 -- 数据存储器
├──add.v                       -- 用于I型指令条件分支指令跳转，把扩展移位后的imm16与PC+4后的值相加
└──MUX4X32_0.v                 -- 4路选择器，选择更新PC的来源（PC+4/PC+4+imm/jr的PC/jump的PC）
```

另外还有 all_test.v 和 cpu_test.v，all_test.v 对 cpu 内部模块进行了测试，cpu_test.v 进行了实例化测试。

一．实现原理

1.1：单周期 CPU 设计参考图



1.2：控制信号译码：

蓝色部分为 Aluc，黄色部分 I 型、J 型指令不存在

instructio	op[31:26]	func[5:0]	Reg2reg	Pcsrc[1:0]	Wmem	Aluc[3:0]	Aluqa	Aluqb	Wreg	Se(1:sign)	Jal	Regrt
addu	000000	100001	1	00	0	0000	0	1	1	0	0	0
subu	000000	100011	1	00	0	0001	0	1	1	0	0	0
and	000000	100100	1	00	0	0010	0	1	1	0	0	0
or	000000	100101	1	00	0	0011	0	1	1	0	0	0
xor	000000	100110	1	00	0	0100	0	1	1	0	0	0
sll	000000	000000	1	00	0	0101	1	1	1	0	0	0
srl	000000	000010	1	00	0	0111	1	1	1	0	0	0
sra	000000	000011	1	00	0	1000	1	1	1	0	0	0
jr	000000	001000	0	10	0	0000	0	0	0	0	0	0
addi	001000		1	00	0	0000	0	0	1	1	0	1
andi	001100		1	00	0	0010	0	0	1	0	0	1
ori	001101		1	00	0	0011	0	0	1	0	0	1
xori	001110		1	00	0	0100	0	0	1	0	0	1
lw	100011		0	00	0	0000(XX	0	0	1	1	0	1
sw	101011		1	00	1	0000(XX	0	0	0	1	0	1
beq	000100		1	00/01	0	0001	0	1	0	1	0	1
bne	000101		1	00/01	0	0001	0	1	0	1	0	1
lui	001111		1	00	0	0110	0	0	1	0	0	1
j	000010		1	11	0	0000	0	1	0	0	0	0
jal	000011		1	11	0	0000	0	0	1	0	1	0

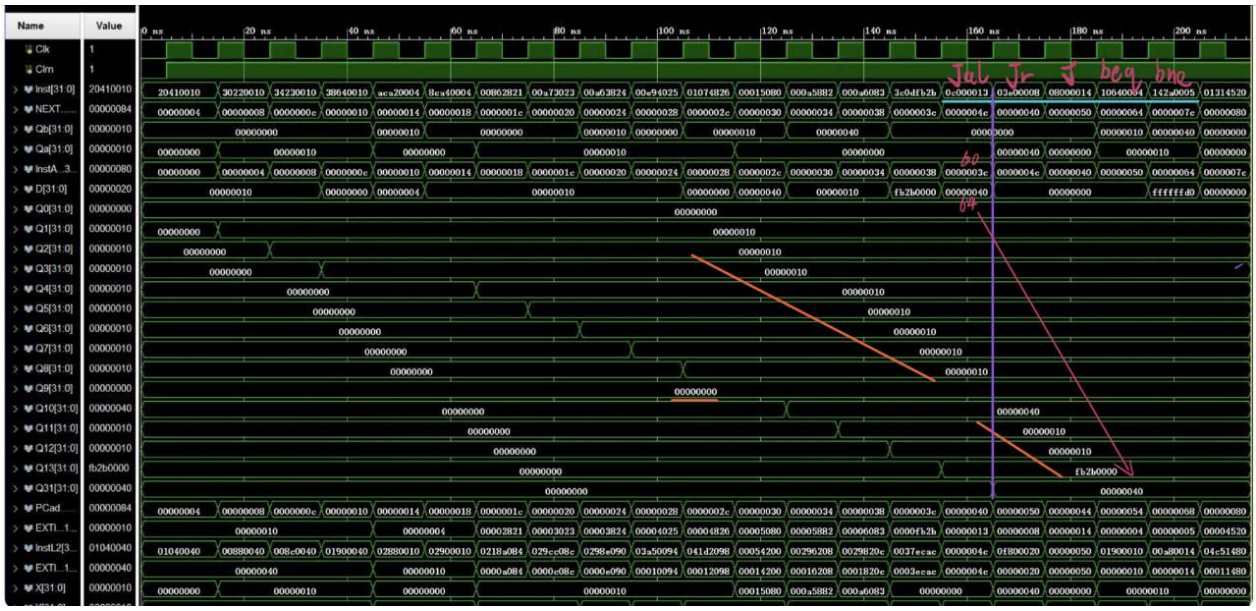
二．数据通路连接及重点指令介绍

我们已经实现了 20 条指令并在测试指令集（见文末）中对所有指令都进行了测试：

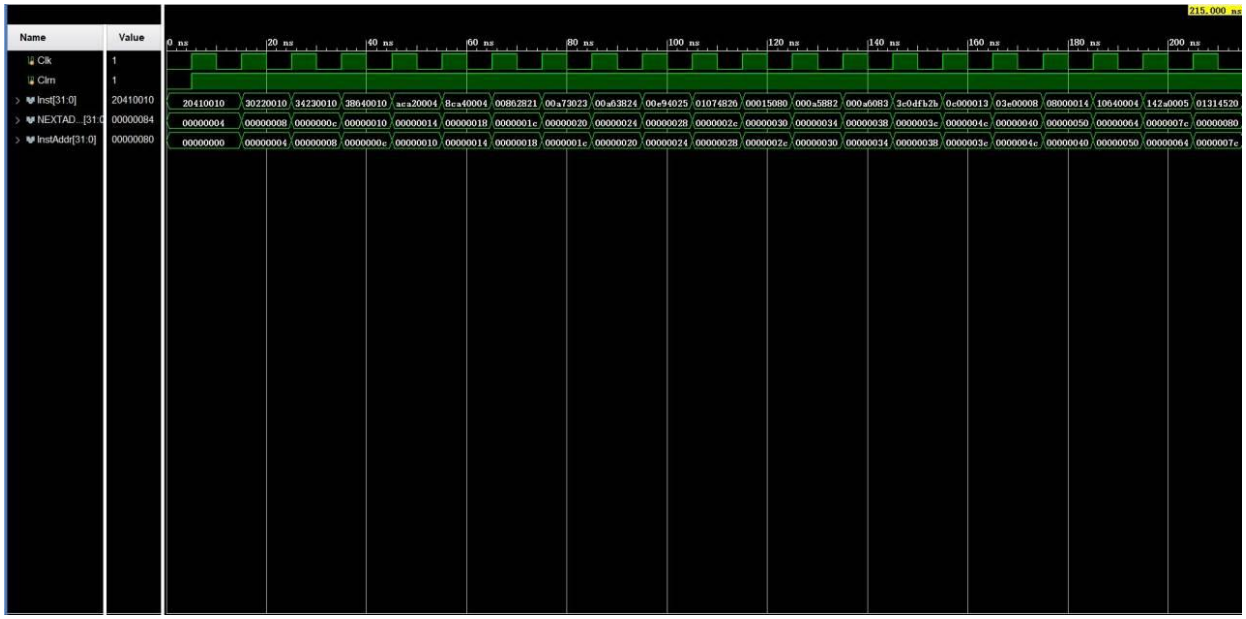
addu、subu、and、or、xor、sll、srl、sra、jr、addi、andi、ori、xori、lw、sw、beq、bne、lui、j、jal

仿真结果示意图：

All_test.v



Cpu_test.v



我们将根据数据通路以及控制信号，着重介绍以下几条指令：

2.1：ADDU

ADDU（无符号加法）指令所需的基本器件有：PC、INSTMEM、CONUMT、REGISTER、ALU。

第一步：取址，得到 PC 值并执行 PC+4。将得到的 PC 地址值输入 INSTMEM，得到指令码 inst。

得到的指令结构为：

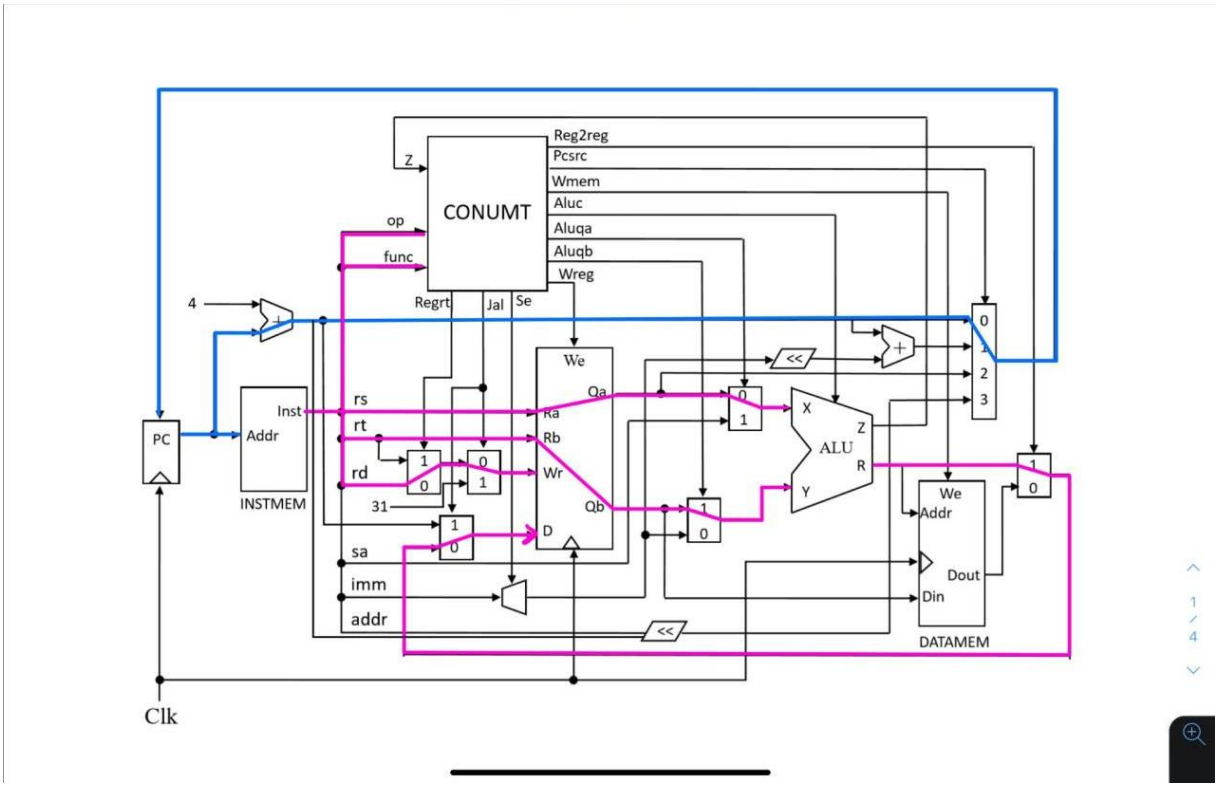
op[31:26]	rs[25:21]	rt[20:16]	rd[15:11]	shamt[10:6]	func[5:0]
000000	xxxxx	xxxxx	xxxxx	00000	100001

第二步：译码，将得到的指令码中的 op 和 func 传入 CONUMT，得到译码后的控制信号，如上图（1.2）。并根据译码后的控制信号将 rs、rt、rd 送入 REGISTER，将 rs，rt 作为输入信号传入 ALU。

第三步：执行，ALU 根据译码后的 Aluc 控制信号执行 ADDU 操作，得到运算结果 R。

第四步：写回，根据译码后的 Reg2reg 选择 ALU 的输出结果，写回到寄存器 rd。

数据通路如下图：



2.2：SUBU

SUBU（无符号加法）指令所需的基本器件有：PC、INSTMEM、CONUMT、REGISTER、ALU。

第一步：取址，得到 PC 值并执行 PC+4。将得到的 PC 地址值输入 INSTMEM，得到指令码 inst。

得到的指令结构为：

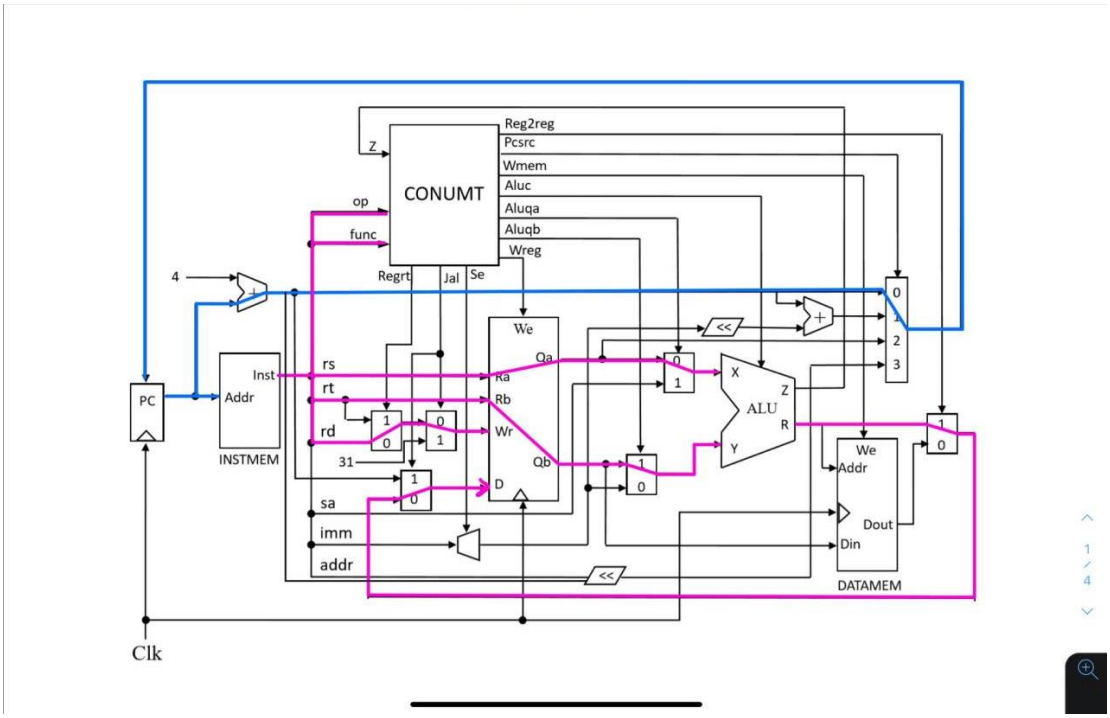
op[31:26]	rs[25:21]	rt[20:16]	rd[15:11]	shamt[10:6]	func[5:0]
000000	xxxxx	xxxxx	xxxxx	00000	100011

第二步：译码，将得到的指令码中的 op 和 func 传入 CONUMT，得到译码后的控制信号，如上图（1.2）。并根据译码后的控制信号将 rs、rt、rd 送入 REGISTER，将 rs，rt 作为输入信号传入 ALU。

第三步：执行，ALU 根据译码后的 Aluc 控制信号执行 SUBU 操作，得到运算结果 R。

第四步：写回，根据译码后的 Reg2reg 选择 ALU 的输出结果，写回到寄存器 rd。

数据通路如下图：（同 ADDU）



2.3：ORI

ORI（立即数或）指令所需的基本器件有：PC、INSTMEM、CONUMT、REGISTER、ALU。

第一步：取址，得到 PC 值并执行 PC+4。将得到的 PC 地址值输入 INSTMEM，得到指令码 inst。

得到的指令结构为：

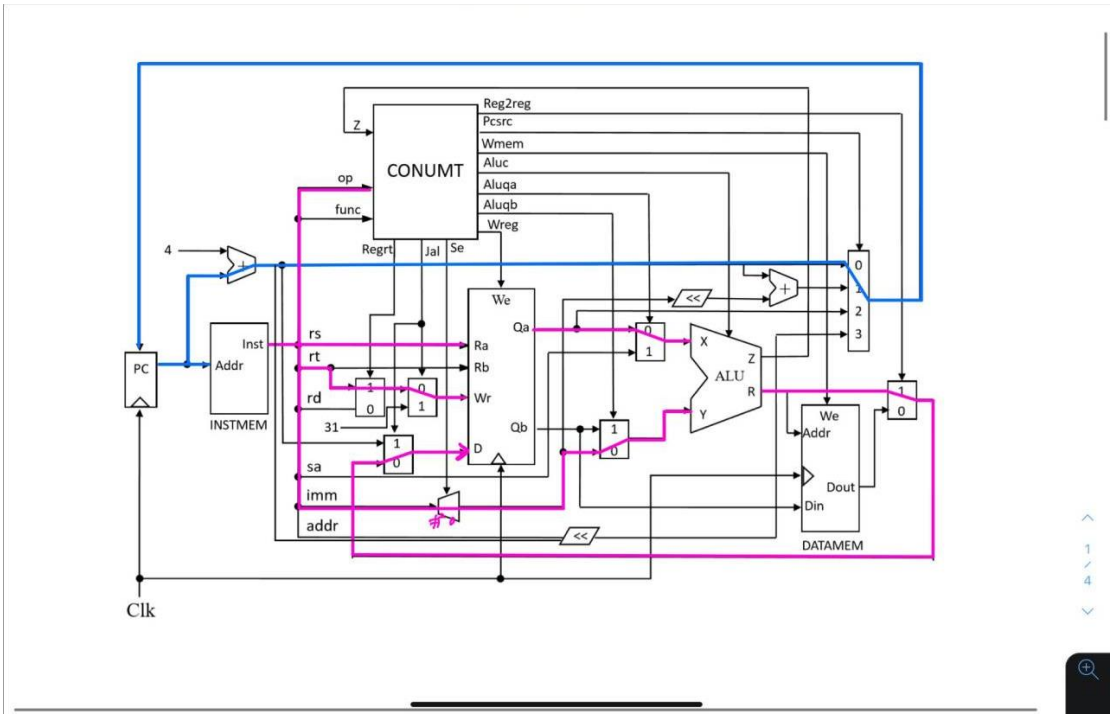
op[31:26]	rs[25:21]	rt[20:16]	imm[15:0]
001101	xxxxx	xxxxx	xxxxxxxxxxxxxxxx

第二步：译码，将得到的指令码中的 op 传入 CONUMT，得到译码后的控制信号，如上图（1.2）。并根据译码后的控制信号将 rs、rt 送入 REGISTER。根据控制信号 Aluqb 和 Se，将 rs 和零扩展后的 imm（立即数）作为输入信号传入 ALU。

第三步：执行，ALU 根据译码后的 Aluc 控制信号执行 ORI 操作，得到运算结果 R。

第四步：写回，根据译码后的 Reg2reg 选择 ALU 的输出结果，写回到寄存器 rt。

数据通路如下图：



2.4：lw

LW(LOAD)指令所需的基本器件有:PC、INSTMEM、CONUMT、REGISTER、ALU、DATAMEM。

第一步：取址，得到 PC 值并执行 PC+4。将得到的 PC 地址值输入 INSTMEM，得到指令码 inst。

得到的指令结构为：

op[31:26]	rs[25:21]	rt[20:16]	imm[15:0]
100011	xxxxx	xxxxx	xxxxxxxxxxxxxxxx

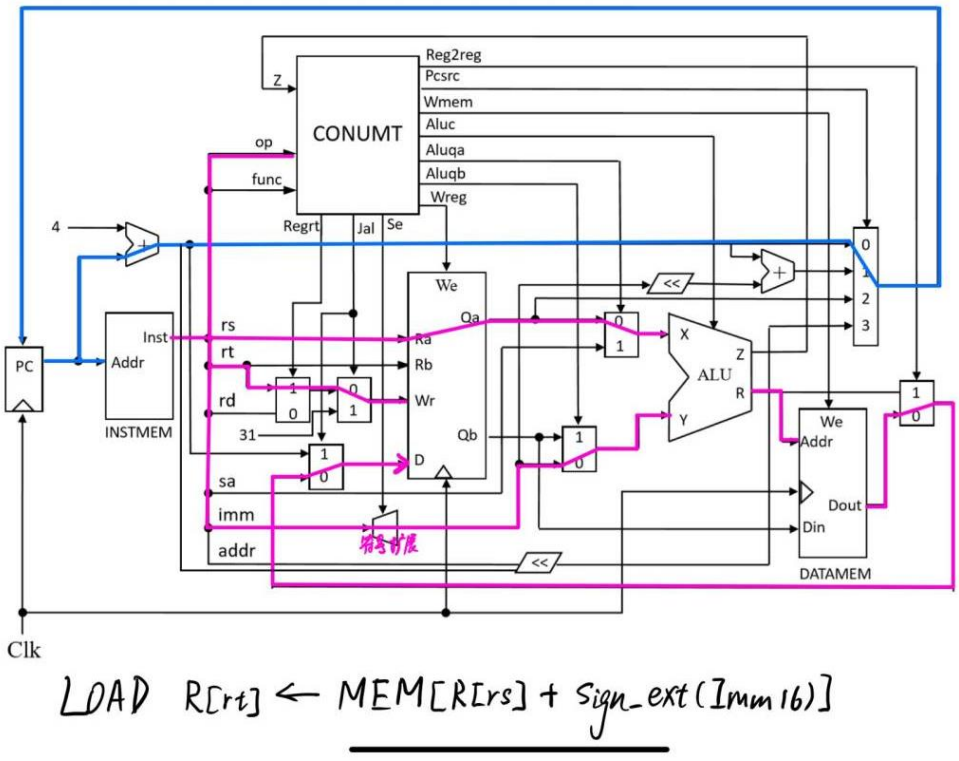
第二步：译码，将得到的指令码中的 op 传入 CONUMT，得到译码后的控制信号，如上图（1.2）。并根据译码后的控制信号将 rs、rt 送入 REGISTER。根据控制信号 Aluqb 和 Se，将 rs 和符号扩展后的 imm（立即数）作为输入信号传入 ALU。

第三步：执行，ALU 根据译码后的 Aluc 控制信号执行 ADDU 操作，得到运算结果 R。

第四步：访存，将 ALU 的输出结果（地址）送入 DATAMEM，取出对应的数据。

第五步：写回，根据译码后的 Reg2reg 选择 DATAMEM 的输出结果，写回到寄存器 rt。

数据通路如下图：



2.5：SW

SW（STORE）指令所需的基本器件有：PC、INSTMEM、CONUMT、REGISTER、ALU、DATAMEM。

第一步：取址，得到 PC 值并执行 PC+4。将得到的 PC 地址值输入 INSTMEM，得到指令码 inst。

得到的指令结构为：

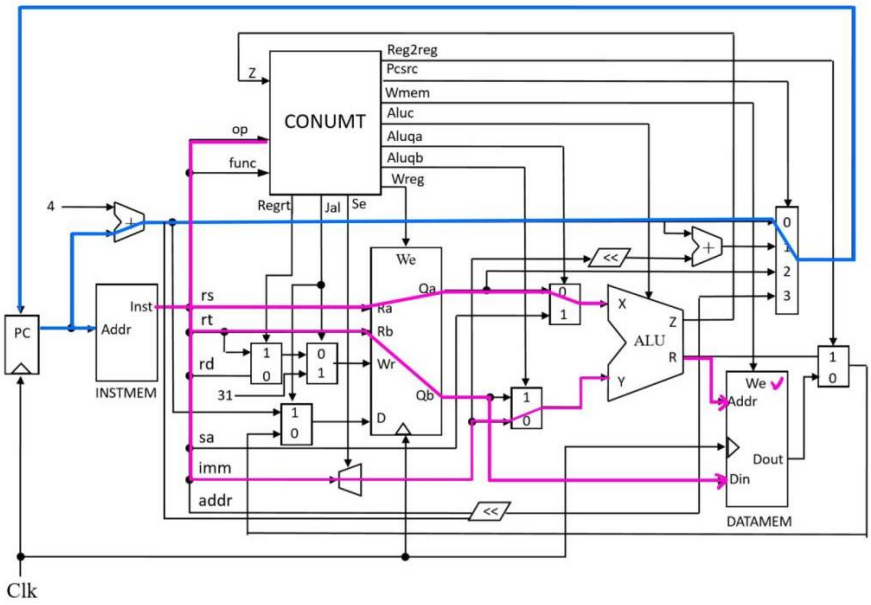
op[31:26]	rs[25:21]	rt[20:16]	imm[15:0]
101011	xxxxx	xxxxx	xxxxxxxxxxxxxxxx

第二步：译码，将得到的指令码中的 op 传入 CONUMT，得到译码后的控制信号，如上图（1.2）。并根据译码后的控制信号将 rs、rt 送入 REGISTER。根据控制信号 Aluqb 和 Se，将 rs 和符号扩展后的 imm（立即数）作为输入信号传入 ALU。

第三步：执行，首先，ALU 根据译码后的 Aluc 控制信号执行 ADDU 操作，得到运算结果 R。

第四步：访存，首先，将 ALU 的输出结果（地址）送入 DATAMEM 的 addr 端。其次，将 rt 的内容送入 DATAMEM 的 Din 端。将 Din 的值写入 addr 对应的内存单元。

数据通路如下图：



2.6：BEQ

BEQ(Branch on equal)指令所需的基本器件有：PC、INSTMEM、CONUMT、REGISTER、ALU、MUX4。

第一步：取址，得到 PC 值并执行 PC+4。将得到的 PC 地址值输入 INSTMEM，得到指令码 inst。

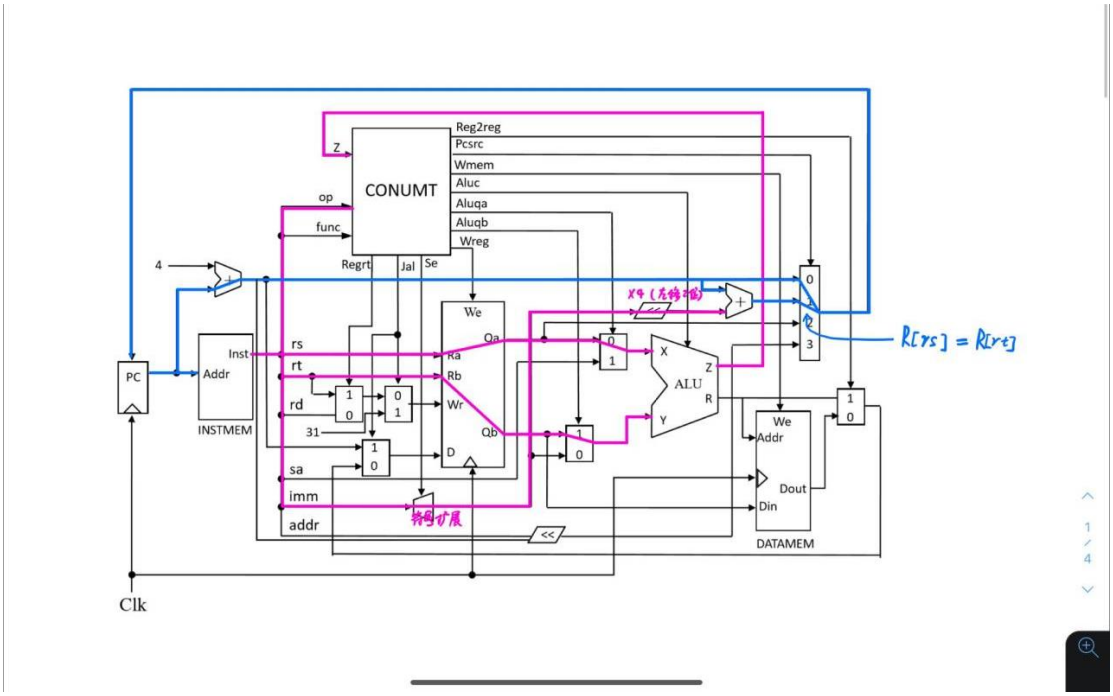
得到的指令结构为：

op[31:26]	rs[25:21]	rt[20:16]	imm[15:0]
000100	xxxxx	xxxxx	xxxxxxxxxxxxxxxx

第二步：译码，将得到的指令码中的 op 传入 CONUMT，得到译码后的控制信号，如上图（1.2）。并根据译码后的控制信号将 rs、rt 送入 REGISTER。根据控制信号 Aluqa 和 Aluqb，将 rs 和 rt 作为输入信号传入 ALU。

第三步：执行，首先，ALU 根据译码后的 Aluc 控制信号执行 SUBU 操作，得到运算结果 Z。将 PC+4 后的值与符号扩展后的立即数相加，得到新的 PC 值。如果 Z=0，条件不成立，Pcsrc 选 00，不进行跳转，继续将 PC+4 更新 PC。如果 Z=1，条件成立，Pcsrc 选 01，进行跳转，将计算到的新的 PC 值更新 PC。

数据通路如下图：



2.7:J

J(Jump)指令所需的基本器件有：PC、INSTMEM、CONUMT、MUX4。

第一步：取址，得到 PC 值并执行 PC+4。将得到的 PC 地址值输入 INSTMEM，得到指令码 inst。

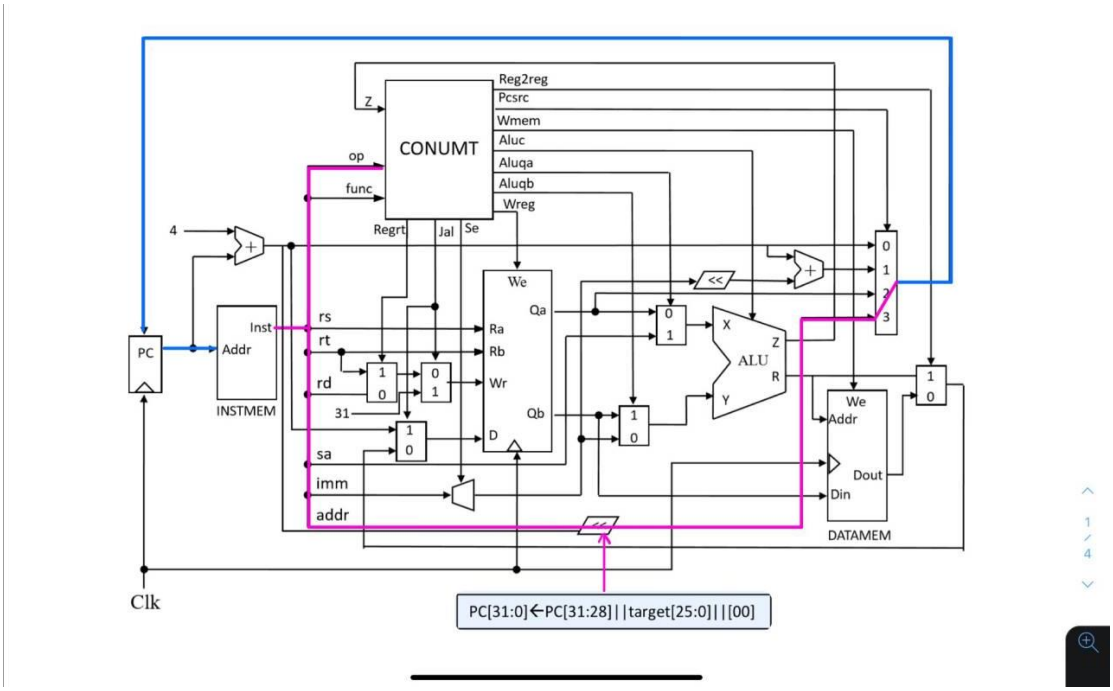
得到的指令结构为：

op[31:26]	addr[25:0]
000010	xxxxxxxxxxxxxxxxxxxxxxxxxx

第二步：译码，将得到的指令码中的 op 传入 CONUMT，得到译码后的控制信号，如上图（1.2）。

第三步：执行，将 addr 左移 2 位的结果与 PC+4 的高四位拼接后的结果作为更新的 PC 值,通过 Pcsrc 控制四路选择器将结果传回 PC。

数据通路如下图：



2.8：Jal

Jal(Jump and link)指令所需的基本器件有：PC、INSTMEM、CONUMT、REGISTER、MUX4。

第一步：取址，得到 PC 值并执行 PC+4。将得到的 PC 地址值输入 INSTMEM，得到指令码 inst。

得到的指令结构为：

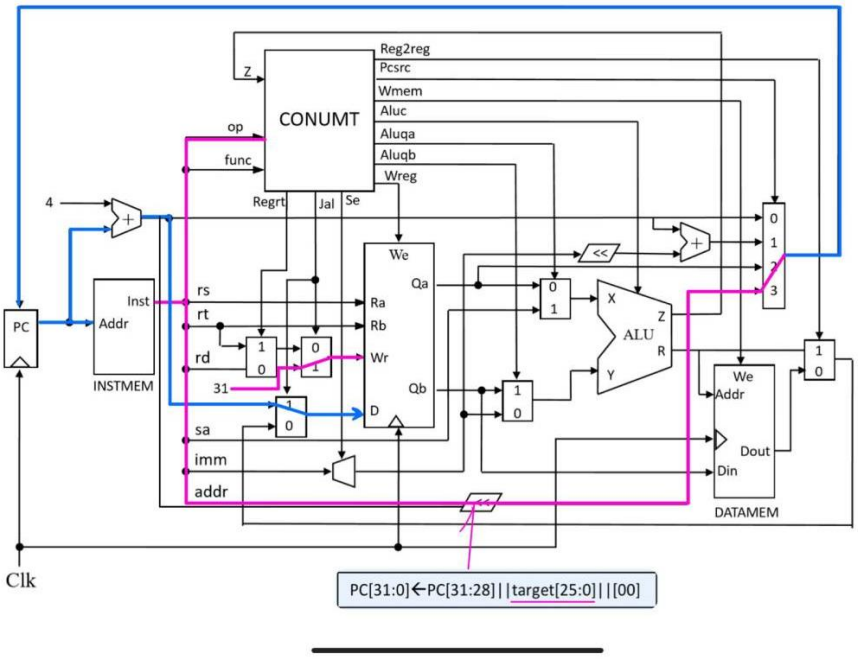
op[31:26]	addr[25:0]
000011	xxxxxxxxxxxxxxxxxxxxxxxxxx

第二步：译码，将得到的指令码中的 op 传入 CONUMT，得到译码

后的控制信号，如上图（1.2）。

第三步：执行，首先将 PC+4 的值作为 D 输入 REGISTER，将 31 号寄存器的地址传入 Wr，将 D（PC+4）存入 Wr（31 号寄存器）中。然后，将 addr 左移 2 位的结果与 PC+4 的高四位拼接后的结果作为更新的 PC 值，通过 Pcsrc 控制四路选择器将结果传回 PC。

数据通路如下图：



2.9：Jr

Jr(Jump register)指令所需的基本器件有：PC、INSTMEM、CONUMT、REGISTER、MUX4。

第一步：取址，得到 PC 值并执行 PC+4。将得到的 PC 地址值输入 INSTMEM，得到指令码 inst。

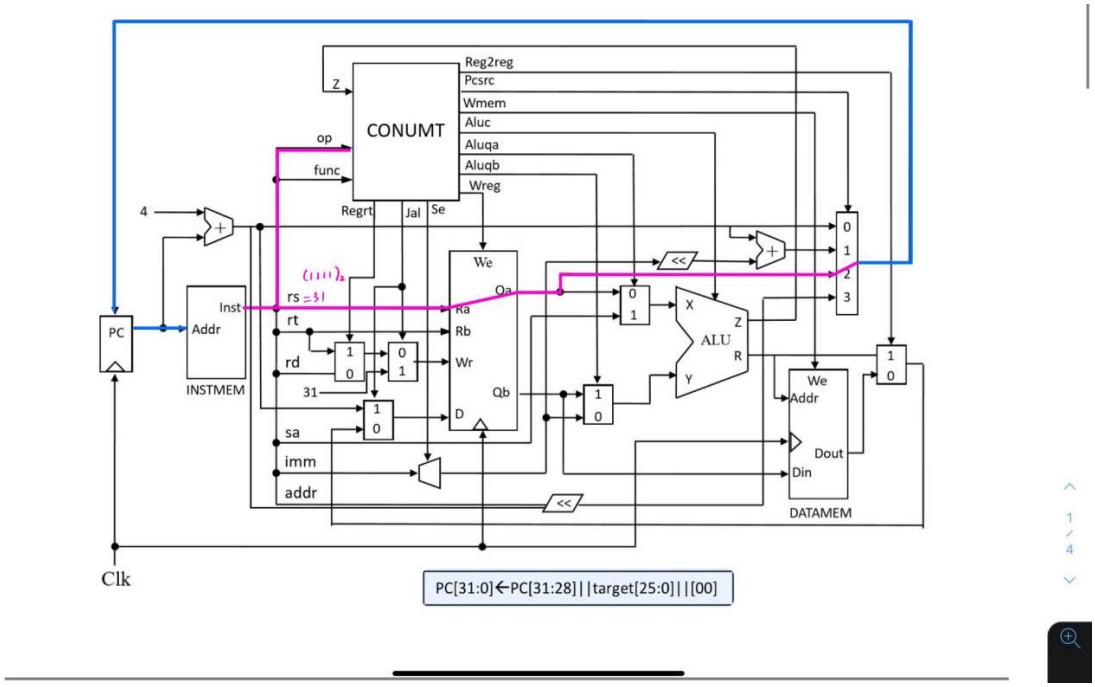
得到的指令结构为：

op[31:26]	rs[25:21]	rt[20:16]	rd[15:11]	shamt[10:6]	func[5:0]
000000	11111	xxxxx	xxxxx	00000	001000

第二步：译码，将得到的指令码中的 op 传入 CONUMT，得到译码后的控制信号，如上图（1.2）。

第三步：执行，将 31 号寄存器地址作为 Ra 输入到 REGISTER，拿出对应寄存器中的数据 Qa 作为新的 PC。通过 Pcsrc 控制的四路选择器，将新的 PC 值更新到 PC。

数据通路如下图：



测试指令集

```
//命令 ->预计结果 二进制机器码
//addi $1,$2,16 ->$1=$2+16=16 001000 00010 00001 0000000000010000
assign Rom[5'h0]=32'h20410010;
//andi $2,$1,16 ->$2=$1&&16=16 001100 00001 00010 0000000000010000
assign Rom[5'h1]=32'h30220010;
//ori $3,$1,16 ->$3=$1||16=16 001101 00001 00011 0000000000010000
assign Rom[5'h2]=32'h34230010;
//xori $4,$3,16 ->$4=0 001110 00011 00100 0000000000010000
assign Rom[5'h3]=32'h38640010;
//sw $2,4($5) ->mem[$5+4]=$2=16 101011 00101 00010 0000000000000100
assign Rom[5'h4]=32'haca20004;
//lw $4,4($5) ->$4=mem[$5+4]=16 100011 00101 00100 0000000000000100
assign Rom[5'h5]=32'h8ca40004;
//addu $5,$4,$6 ->$5=$4+$6=16 000000 00100 00110 00101 00000 100001
assign Rom[5'h6]=32'h862821;
//subu $6,$5,$7 ->$6=$5-$7=16 000000 00101 00111 00110 00000 100011
assign Rom[5'h7]=32'ha73023;
//and $7,$5,$6 ->$7=$5&$6=16 000000 00101 00110 00111 00000 100100
assign Rom[5'h8]=32'ha63824;
//or $8,$7,$9 ->$8=&7||&9=16 000000 00111 01001 01000 00000 100101
assign Rom[5'h9]=32'he94025;
//xor $9,$8,$7 ->$9=$8xor$7=0 000000 01000 00111 01001 00000 100110
assign Rom[5'ha]=32'h1074826;
//sll $10,$1,2 ->$10=$1<<2=64 000000 00000 00001 01010 00010 000000
assign Rom[5'hb]=32'h15080;
//srl $11,$10,2 ->$11=$10>>2=16 000000 00000 01010 01011 00010 000010
assign Rom[5'hc]=32'ha5882;
//sra $12,$10,2 ->$12=$10>>2=16 000000 00000 01010 01100 00010 000011
assign Rom[5'hd]=32'ha6083;
//lui $13,0xfb2b ->$13=0xfb2b0000 001111 00000 01101 1111101100101011
assign Rom[5'he]=32'h3c0dfb2b;
//jal 19 ->go jr($31=&j) 000011 00000 00000 0000000000010011
assign Rom[5'hf]=32'hc000013;
//j 20 -> go beq 000010 00000000000000000000010100
assign Rom[5'h10]=32'h8000014;
assign Rom[5'h11]=32'hXXXXXXXX;
assign Rom[5'h12]=32'hXXXXXXXX;
//jr go j 000000 11111 00000 00000 00000 001000
assign Rom[5'h13]=32'h3e00008;
//beq $3,$4,4 ->go bne 000100 00011 00100 0000000000000100
```

```
assign Rom[5'h14]=32'h10640004;
assign Rom[5'h15]=32'hXXXXXXXX;
assign Rom[5'h16]=32'hXXXXXXXX;
assign Rom[5'h17]=32'hXXXXXXXX;
assign Rom[5'h18]=32'hXXXXXXXX;
//bne $1,$10,5 ->go love 000101 00001 01010 0000000000000101
assign Rom[5'h19]=32'h142a0005;
assign Rom[5'h1a]=32'hXXXXXXXX;
assign Rom[5'h1b]=32'hXXXXXXXX;
assign Rom[5'h1c]=32'hXXXXXXXX;
assign Rom[5'h1d]=32'hXXXXXXXX;
assign Rom[5'h1e]=32'hXXXXXXXX;
//love~
assign Rom[5'h1f]=32'h1314520;
```