计算机组成原理 CPU 设计实验报告

王嘉禾 2193211079 计算机试验班 001

单周期 CPU

(代码见文件夹“单周期 cpu”)

1. 指令集设计

指令集分为四个类型，分别为 1/2/3/4 型指令，格式如下：

占用 bit	2b	5b	5b	5b	11b	4b
1 型指令	00	Rs	Rd	Rt	/	Func
2 型指令	01	Rs	Rt	Addr_16/Imm_16		Func
3 型指令	10	Rs	/	Addr_16/Imm_16		Func
4 型指令	11	/		Addr_16/Imm_16		Func

其中 Rs/Rd/Rt 分别代表两个源寄存器和一个目标寄存器的编号

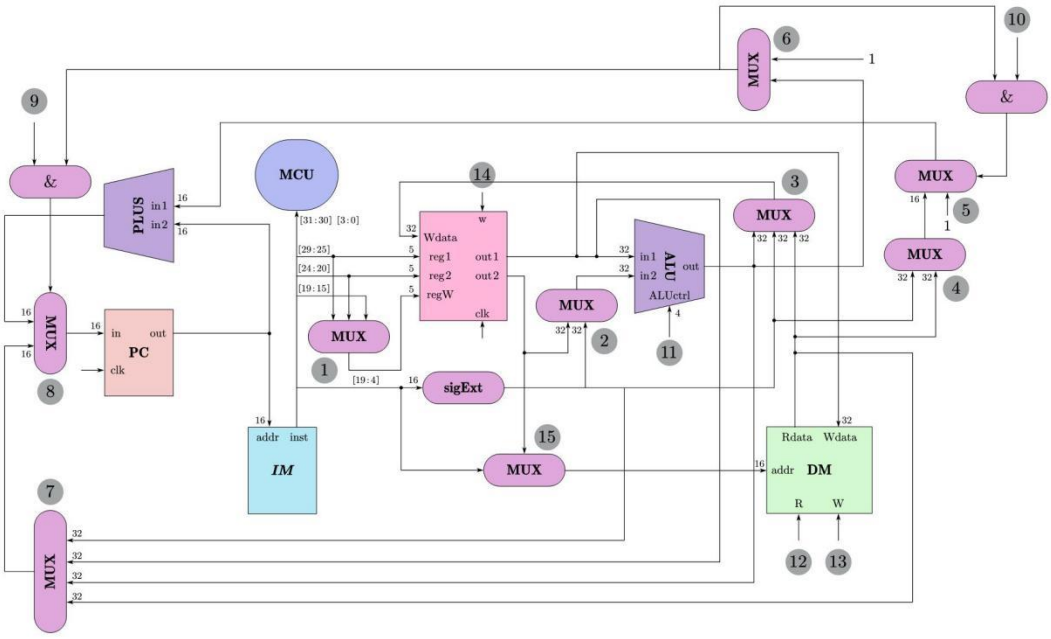
Addr_16 代表 16 位地址，Imm_16 代表 16 位立即数

详细指令设计如下：

1 型指令（共 10 条）	
操作码	功能
0000	Rt <- (Rs) + (Rd)
0001	Rt <- (Rs) - (Rd)
0010	Rt_32 <- (Rs_16low) *(Rd_16low)
0011	Rt_16low <- (Rs_32) / (Rd_16low)
0100	Rt <- (Rs) & (Rd)
0101	Rt <- (Rs) | (Rd)
0110	Rt <- (Rs) ^ (Rd)
0111	if (Rs) == (Rd) Rt<-1 else Rt<-0
1000	Rt <- (Rs) << (Rd)
1001	Rt <- (Rs) >> (Rd)
2 型指令（共 6 条）	
操作码	功能
0000	if (Rs) == (Rt) PC <- Addr_16 else PC <- PC + 1
0001	if (Rs) == (Rt) PC = PC + Imm_16 else PC <- PC + 1
0010	if (Rs) == (Rt) PC <- DM[Addr_16] else PC <- PC + 1
0011	if (Rs) == (Rt) PC = PC + DM[Addr_16] else PC <- PC + 1
0111	If (Rs) == Imm_16 Rt <= 1 else Rt <- 0
1000	DM[(Rt)] <- (Rs)
3 型指令（共 14 条）	
操作码	功能
0000	Rs <- (Rs) + Imm_16
0001	Rs <- (Rs) - Imm_16
0010	Rs <- (Rs_16low) * Imm_16
0011	Rs_16low <- (Rs) / Imm_16
0100	Rs <- (Rs) & Imm_16
0101	Rs <- (Rs) | Imm_16

0110	Rs <- (Rs) ^ Imm_16
1000	Rs <- (Rs) << Imm_16
1001	Rs <- (Rs) >> Imm_16
1010	Rs <- DM[Addr_16]
1011	Rs <- Imm_16
1100	DM[Addr_16] <- Rs
1101	PC <- (Rs)
1110	PC <- (Rs) + Imm_16
4 型指令（共 4 条）	
操作码	功能
0000	PC <- Addr_16
0001	PC <- PC + Imm_16
0010	PC <- DM[Addr_16]
0011	PC = PC + DM[Addr_16]

2. CPU 元件和连线设计

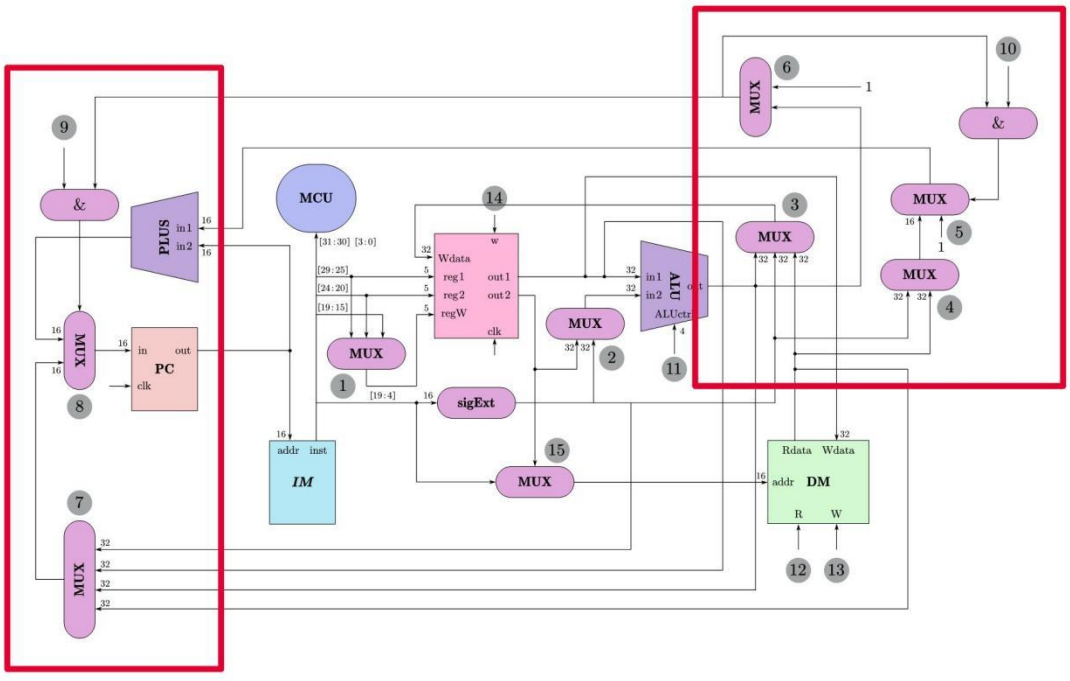


图中的编号代表没有画出的主控制单元生成的主控制信号，以下给出详细解释

编号	信号	作用解释
1	多路选择器控制信号	由 MCU 根据指令生成，决定指令中那一个字段是要写入数据的寄存器编号。
2	多路选择器控制信号	由 MCU 根据指令生成，决定是两个寄存器的运算还是一个寄存器和一个立即数的运算，对于 ALU，in1 一定为寄存器，in2 可能是寄存器或立即数。
3	多路选择器控制信号	由 MCU 根据指令生成，决定写入寄存器的数据来自指令中的立即数/ALU 的运算结果/主存中取出的值。
4	多路选择器控制信号	由 MCU 根据指令生成，当涉及无条件跳转指令且为 PC 跳转一个特定距离时，决定该距离是一个立即数/存储空间的值。

5/10	多路选择器控制信号来自与门的输出信号	多路选择器的控制信号由 MCU 或 MCU 和 ALU 共同控制，当不涉及条件转移时，6 号多路选择器输出 1，不影响与门的结果，由 MCU 控制信号控制 PC 的变化；当涉及条件转移时，mux6 输出 ALU 的结果，若为 1，同上，若为 0，则不转移，传输 1，PC=PC+1（和另一个与门 9 和多路选择器 6 联合作用）
6	多路选择器控制信号	由 MCU 根据指令生成，ALU 是否参与决定 PC 的变化方式，即判断该指令是否是条件转移指令，当是条件转移指令且 ALU 结果为 0 时（不满足条件），该多路选择器输出 0，控制两个与门都输出 0，此时会选择 PC 加上值的模式，且该值为 1；当不是条件转移指令或条件满足时，多路选择器输出 1，由 MCU 的信号完全控制 PC 的转移。（与两个与门联合作用）
7	多路选择器控制信号	由 MCU 根据指令生成，涉及转移指令时生效，当 PC 的转移方式为直接赋成特定值时，决定改值来自立即数/寄存器/ALU 运算结果/主存内容。
8/9	多路选择器控制信号来自与门的输出信号	与 6 号多路选择器联合作用，8 号多路选择器决定 PC 的转移方式为加上特定的值/被赋成特定的值，信号 9 位 MCU 控制信号，当非 PC=PC+1 的情况下做出上述决定，当存在条件转移且条件不满足时，与门的另一输入为 0，强制选择加上特定值的方式且特定值为 1，即 PC=PC+1。
11	ALU 控制信号	由 MCU 生成，决定 ALU 作何种运算，在此 cpu 设计中，ALU 控制信号在绝大多数情况下与指令中的操作码字段一致。
12	主存读信号	控制主存读 addr 字段对应地址的内容并输出。
13	主存写信号	控制主存将 Wdata 端口输入的数据存入主存的 addr 地址。
14	寄存器堆写信号	控制寄存器堆将传入的值写入 regW 端口输入指定编号的寄存器。
15	多路选择器控制信号	控制主存写入的地址时立即数还是目标寄存器中的值（低 16 位）

3. 自主设计：cpu 指令转移通路



整个指令转移通路的核心部件为五个多路选择器（其中四个二路选择器，一个四路选择器）和两个与门构成。总的来说，**转移通路有两条支路通过一个二路选择器输入进 PC，分别对应位移量或直接赋值的转移方式。两条支路各有多路选择器控制相关的量的输入来源。并且，ALU 的计算结果通过一个二路选择器和两个与门干预整个转移通路，对应条件转移指令的情况。**以下给出核心部件的作用描述：

图中编号	部件名称	作用描述
8	二路选择器	控制最终输入 PC 的值是由加法器做加法运算得出（0），还是直接赋值（1），对应于转移指令中是转移一个位移量，还是直接转移到特定地址
7	四路选择器	在 PC 的转移为直接赋值的情况下，控制输入的值是如何得到的，有四种情况，分别为：立即数（00），存储器内容（01），寄存器内容（10）和 ALU 计算结果（通过寄存器内容+立即数得到，对应于基址变址寻址）。当 8 号二路选择器的输入为 1 时，PC 的转移方式为直接赋值，本器件有效运行。
5	二路选择器	在 PC 转移方式为位移量（通过加法器之后赋值）的情况下，控制输入的值是不是 1（顺序转移与否）。当 8 号二路选择器的输入为 0 时，PC 的转移方式为位移量，本器件有效运行。
4	二路选择器	在 PC 转移方式为位移量，且不是直接顺序执行的情况下，控制输入的值是立即数（0）还是存储器读出的值（1）。当 8 号二路选择器的输入为 0 时，PC 的转移方式为位移量，且 5 号二路选择器的输入为 1 时，本器件有效运行。
6	二路选择器	控制 ALU 的计算结果是否影响 PC 的转移。当为条件转移指令时（选择器控制信号为 1），则输出 ALU 的计算结果的最后一位（此时 ALU 输出 0 或 1）；不是条件转移指令时，则输出 1。与 9 号 10 号两个与门配合使用，当该二路选择器输出 0 时（只可能是条件转移且条件不满足），则 5 号二路选择器输出 1，且 8 号二路选择器输入为 0，表示选择加法器运算结果支路，代表强制执行 PC=PC+1；当该二路选择器输出为 1 时（条件转移且条件满足或非条件转移指令），则两个与门输出 9 号和 10 号的信号量，按指令设定的 PC 转移方式运作。
9	与门	输入的信号为指令设定的 PC 转移方式（位移量或直接赋值），与 6 号二路选择器配合使用。若另一输入为 0，则与门输出为 0，这代表此时指令为条件转移指令且指令不满足，强制执行 PC=PC+1，因此 PC 的输入通路为加法器运算结果通路。
10	与门	输入的信号为当 PC 的转移方式为位移量时，是不是 1，与 6 号二路选择器配合使用，当指令为非转移指令时，10 号信号为 0，与门输出 0，代表选择 PC 的位移量为 1,；当为条件转移指令或转移指令时，输出 1，代表 PC 的位移量为立即数或存储器值（当然也可能通过这种方式输出 1），在条件转移下是否成功转移由与门的另一输入（6 号二路选择器的输出）决定。若为条件转移且条件不满足，则为 0，与门输出 0，强制选择位移量 1。

以 2 型指令中的四条条件转移指令为例，分析 CU 部件的控制信号

```
2型：
01|5bit源寄存器编号|5bit目标寄存器编号|16bit地址/立即数|4bit操作码
操作码：
0000 目标=源->PC=16bit地址
0001 目标=源->PC=PC+立即数
0010 目标=源->PC=16bit地址内容
0011 目标=源->PC=PC+16bit地址内容
```

```
4'b0000:begin
    PC_input=1'b1;          PC的转移选择直接赋值的通路
    imm_or_dm1=1'b0;
    assign_PC=2'b00;        在直接赋值通路中，选择输入为立即数
    four_or_not=1'b0;
    reg2_or_imm=1'b0;
    RF_write_data_mux=2'b00;
    cmp_for_leap=1'b1;      ALU的计算结果会影响PC的转移
    RF_write=1'b0;
    DM_write=1'b0;
    DM_read=1'b0;
    reg_write_mux=2'b00;
    DM_addr_in=1'b0;
```

指令1：当源目标寄存器相等是，PC直接转移至指令给出的立即地址

```
4'b0001:begin
    PC_input=1'b0;          PC的输入选择加法器计算通路
    imm_or_dm1=1'b0;        位移量不是1时，在立即数或存储器值之间选择了立即数
    assign_PC=2'b00;
    four_or_not=1'b1;       PC的位移量不是1
    reg2_or_imm=1'b0;
    RF_write_data_mux=2'b00;
    cmp_for_leap=1'b1;      ALU的计算结果影响PC的转移
    RF_write=1'b0;
    DM_write=1'b0;
    DM_read=1'b0;
    reg_write_mux=2'b00;
    DM_addr_in=1'b0;
```

指令2：源目标寄存器相等时，PC转移一个位移量，等于指令中给出的立即数

```
4'b0010:begin
    PC_input=1'b1;          PC的输入选择直接赋值通路
    imm_or_dm1=1'b0;
    assign_PC=2'b01;        在直接赋值通路中，选择输入为存储器读出的数据
    four_or_not=1'b0;
    reg2_or_imm=1'b0;
    RF_write_data_mux=2'b00;
    cmp_for_leap=1'b1;      ALU的计算结果影响PC的转移
    RF_write=1'b0;
    DM_write=1'b0;
    DM_read=1'b1;           存储器读信号有效
    reg_write_mux=2'b00;
    DM_addr_in=1'b0;        存储器读的地址为立即数
```

指令3：源目标寄存器相等时，PC转移至存储器存内容，存储器地址在指令中给出

```
4'b0011:begin
    PC_input=1'b0;          PC的输入选择加法器的输出结果
    imm_or_dm1=1'b1;        在加法器输入不选择1的情况下，选择存储器的值
    assign_PC=2'b00;
    four_or_not=1'b1;       加法器的输入不选择1
    reg2_or_imm=1'b0;
    RF_write_data_mux=2'b00;
    cmp_for_leap=1'b1;      ALU的计算结果影响PC的转移
    RF_write=1'b0;
    DM_write=1'b0;
    DM_read=1'b1;           存储器读信号有效
    reg_write_mux=2'b00;
    DM_addr_in=1'b0;        存储器读的地址为立即数
```

指令4：源目标寄存器相等时，PC转移一个位移量，值为存储器中读出，地址为指令中的立即数

4. 实验测试

在程序存储器中初始化一端小程序，如下所示，程序的作用在注释中给出

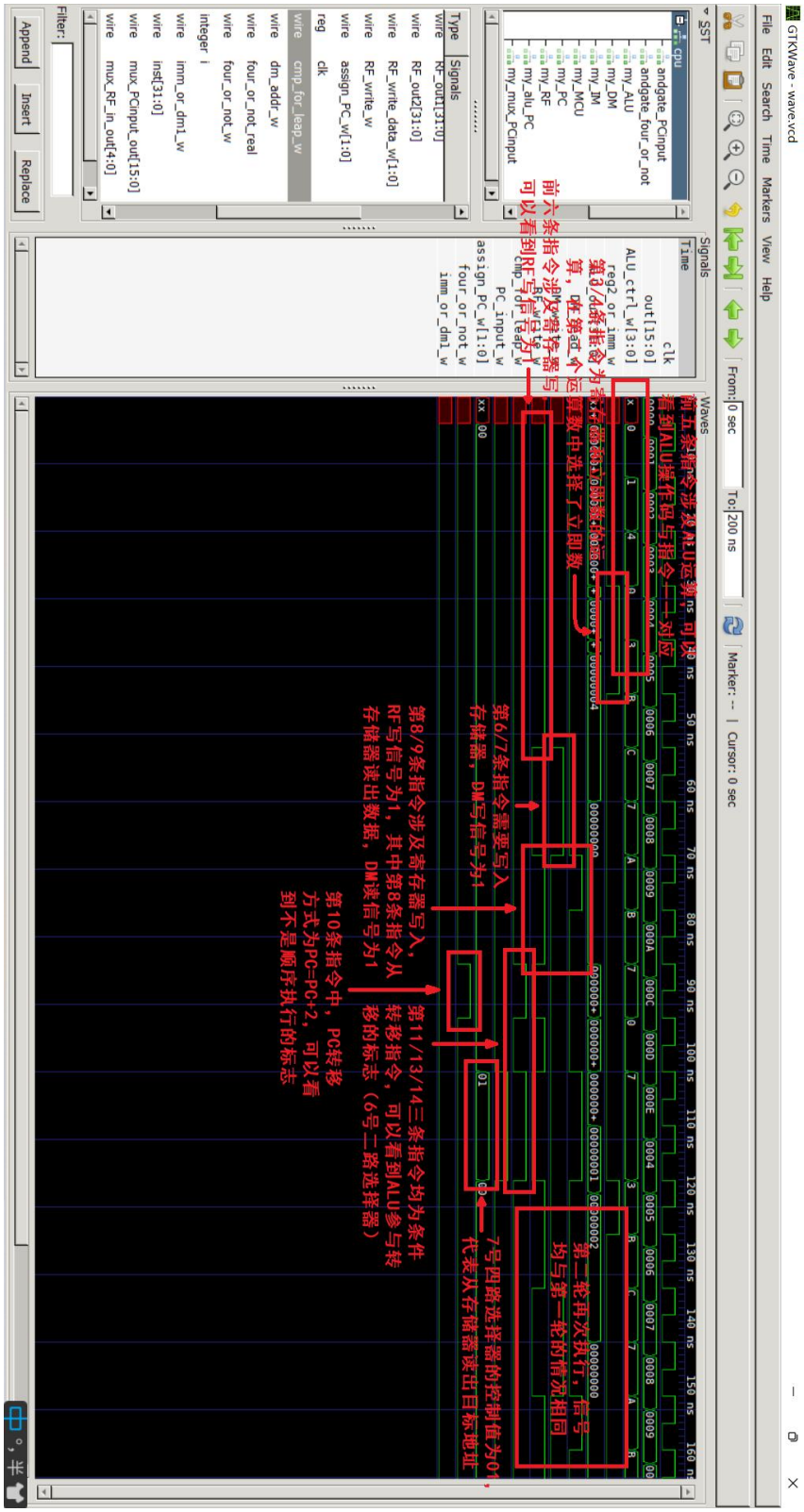
```
memory[0]=32'b00000100000100000000000000000000; //reg0=reg2+reg1=3
memory[1]=32'b00000000000100100000000000000001; //reg4=reg0-reg1=2
memory[2]=32'b00011000001100000000000000010100; //reg0=reg12&reg30=0
memory[3]=32'b10000000000000000000000010000000; //reg0+=8=8
memory[4]=32'b10000000000000000000000001000011; //reg0/=4=2
memory[5]=32'b10000010000000000000000000101011; //reg1=2
memory[6]=32'b10000010000000000000000000001100; //DM[0]=reg1=2
memory[7]=32'b01000010011100000000000000001000; //DM[reg7(7)]=reg1=2
memory[8]=32'b10001100000000000000001100001010; //reg6=DM[48]=48
memory[9]=32'b10000000000000000000000000101011; //reg0=2
memory[10]=32'b01000000000100000000000000100001; //reg0==reg1?->PC+=2(=12)(triggered)
//memory[10]=32'b01000000000100000000000011000000; //reg0==reg1?->PC=12(triggered)
memory[11]=32'b01000000000100000000000000100001; //reg0==reg1?->PC+=2(skipped)
memory[12]=32'b00000100000100000000000000000000; //reg2+reg1->reg0=4
memory[13]=32'b01000000000100000000000000100010; //reg0==reg1?->PC=DM[2]=2(untriggered)
//memory[14]=32'b01000100000100000000000001000010; //reg2==reg1?->PC=DM[4]=4(triggered)
//memory[14]=32'b11000000000000000000000001000000; //jmp 4
//memory[14]=32'b10000000000000000000000000001101; //jmp (reg0=4)
memory[14]=32'b10000100000000000000000000101110; //jmp (reg3+2=4)
memory[15]=32'b11000000000000000000000001010011; //PC+=DM[5](=9)(invalid)
```

在每个时钟周期打印 0-7 号寄存器的值，0-7 号主存的值和 PC 的值，如下图所示

```
E:\files\计算机组成原理\实验\单周期cpu>vvp a.out
--------------------------------PC=    0
reg[0-7]:         0         1         2         3         4         5         6         7
DM[0-7]           0         1         2         3         4         5         6         7
VCD info: dumpfile wave.vcd opened for output.
--------------------------------PC=    1
reg[0-7]:         3         1         2         3         4         5         6         7
DM[0-7]           0         1         2         3         4         5         6         7
--------------------------------PC=    2
reg[0-7]:         3         1         2         3         2         5         6         7
DM[0-7]           0         1         2         3         4         5         6         7
--------------------------------PC=    3
reg[0-7]:         0         1         2         3         2         5         6         7
DM[0-7]           0         1         2         3         4         5         6         7
--------------------------------PC=    4
reg[0-7]:         8         1         2         3         2         5         6         7
DM[0-7]           0         1         2         3         4         5         6         7
--------------------------------PC=    5
reg[0-7]:         2         1         2         3         2         5         6         7
DM[0-7]           0         1         2         3         4         5         6         7
--------------------------------PC=    6
reg[0-7]:         2         2         2         3         2         5         6         7
DM[0-7]           0         1         2         3         4         5         6         7
--------------------------------PC=    7
reg[0-7]:         2         2         2         3         2         5         6         7
DM[0-7]           2         1         2         3         4         5         6         7
--------------------------------PC=    8
reg[0-7]:         2         2         2         3         2         5         6         7
DM[0-7]           2         1         2         3         4         5         6         2
--------------------------------PC=    9
reg[0-7]:         2         2         2         3         2         5        48         7
DM[0-7]           2         1         2         3         4         5         6         2
--------------------------------PC=   10
reg[0-7]:         2         2         2         3         2         5        48         7
DM[0-7]           2         1         2         3         4         5         6         2
--------------------------------PC=   12
reg[0-7]:         2         2         2         3         2         5        48         7
DM[0-7]           2         1         2         3         4         5         6         2
--------------------------------PC=   13
reg[0-7]:         4         2         2         3         2         5        48         7
DM[0-7]           2         1         2         3         4         5         6         2
--------------------------------PC=   14
reg[0-7]:         4         2         2         3         2         5        48         7
DM[0-7]           2         1         2         3         4         5         6         2
--------------------------------PC=    4
reg[0-7]:         4         2         2         3         2         5        48         7
```

部分波形如下图所示，可以看到，第 10/11/13/14 条指令为条件转移指令，第 10 条指令条件成立，转移至第 12 条指令，因而第 11 条指令没有执行，而第 13 条质指令条件不成立，第 14 条指令条件成立，转移至第 4 条指令，因而第 15 条指令没有执行。而波形中第 10/13 条指令执行时，波形显示 ALU 计算结果影响 PC 转移,即多路选择器 mux 输出了 ALU 的计算

结果。第 3/4 条指令调用立即数参与运算，从波形中可看出 ALU 的第二个输入 in2 选择了立即数。第 6/7 条指令涉及写入主存内容，可以看到主存的写信号 DM_w 变为 1，但是寻址方式不同，可以看到 15 号多路选择器在第 6 条指令中选择了立即数而在第 7 条中选择了寄存器。在所有的寄存器运算指令中，也可以看到寄存器堆 RF 的写指令变为 1。其余一些波形也均符合程序调用，说明程序调用正确。（图中还给出了另外两种使 PC 转移到位置 4 的方法，也均测试正确）以下以部分关键信号我为例分析波形：



多周期 CPU（单总线模式）

1. 指令集设计（MIPS32）

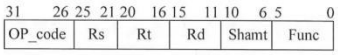
表 6-1　10 条 MIPS 32 指令描述

指令类型	指令	指令格式类型	指令功能描述	
			取指令阶段	执行指令阶段
存储器访问指令	lw Rt,Imm16(Rs)	I-型	(1) M[PC]或者 IR←M[PC] (2) PC←(PC)+4	(3) addr←(Rs)+SigExt(Imm16) (4) Rt←(M)[addr]
	sw Rt,Imm16(Rs)			(3) addr←(Rs)+SigExt(Imm16) (4) M[addr]←(Rt)
算术/逻辑运算指令	add Rd,Rs,Rt	R-型		(3) Rd←(Rs)+(Rt),判溢出
	sub Rd,Rs,Rt			(3) Rd←(Rs)−(Rt),判溢出
	addu Rd,Rs,Rt			(3) Rd←(Rs)+(Rt)
	and Rd,Rs,Rt			(3) Rd←(Rs)∧(Rt)
	or Rd,Rs,Rt			(3) Rd←(Rs)∨(Rt)
	slt Rd,Rs,Rt			(3) if(Rs)<(Rt) Rd←1 else Rd←0
程序转移指令	beq Rs,Rt,Addr16	I-型		(3) if(Rs)==(Rt) PC←(PC)+SigExt(Addr16)×4
	j Addr26	J-型		PC←PC[31～28]‖(Addr26)×4

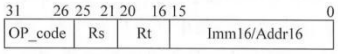
注：① M[PC]表示以 PC 内容作为存储器地址来读取存储单元内容；
② addr 表示存储器地址；
③ Rs、Rt、Rd 分别表示 2 个源寄存器和 1 个目的寄存器，(Rs)、(Rd)、(Rt)分别表示对应寄存器的内容；
④ SigExt(Imm16)表示对指令中的立即数 Imm16 进行 32 位符号扩展；
⑤ ‖表示拼接。

指令格式描述如下

MIPS 32 指令格式如图 6-13 所示。



(a) R-型指令



(b) I-型指令

(c) J-型指令

注：
①OP_code：基本操作码；Shamt：移位量；Func：函数码，与OP_code配合使用；
②Rs：第一个源操作数寄存器；Rt：第二个源操作数寄存器；Rd：目的操作数寄存
③Imm16：16位立即数；Addr16：16位地址；Addr26：26位地址。

图 6-13　MIPS 32 指令格式

OP_code	对应指令
100011	Lw
101011	rw
000000	beq
000010	jmp

ALU 操作码如下表所示

100	加法（判断溢出）	C <- (A) + (B) (save Z)
110	减法（判断溢出）	C <- (A) - (B) (save Z)
101	加法（不判断溢出）	C <- (A) + (B)
000	与运算	C <- (A) & (B)
001	或运算	C <- (A) | (B)
011	异或运算	C <- (A) ^ (B)
010	判断大小	If (A) < (B) C <- 1 else C <- 0
111	判断相等	If (A) == (B) C <- 1 else C <- 0

2. 数据通路如下图所示

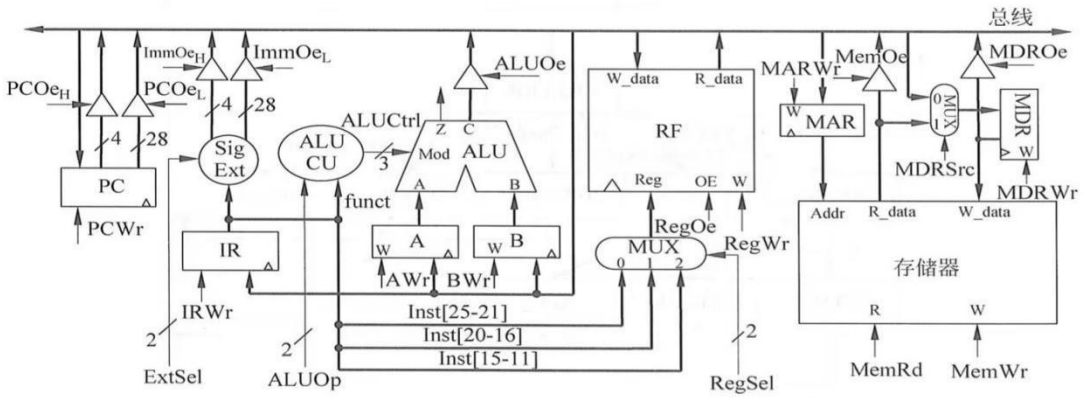


图 6-29　基于通用寄存器的单总线 CPU 数据通路

3. CU 单元控制信号表如下所示

信号名称	含　义	信号名称	含　义
CLK	时钟信号	ALUOe	ALU 输出总线
$PCOe_H$	PC 高 4 位输出总线	RegOe	通用寄存器输出总线
$PCOe_L$	PC 低 28 位输出总线	RegWr	通用寄存器写
PCWr	PC 写	RegSel	通用寄存器选择
IRWr	IR 写	MARWr	MAR 写
$ImmOe_H$	立即数高 4 位输出总线	MemRd	存储器读
$ImmOe_L$	立即数低 28 位输出总线	MemWr	存储器写
ExtSel	符号扩展方式选择	MDRSrc	MD 源选择
ALUOp	ALU 操作	MDROe	MDR 输出总线
ALUctrl	ALU 操作控制	MDRWr	MDR 写
AWr	A 暂存器写	MemOe	存储器输出总线
BWr	B 暂存器写		

以下给出控制信号随始终周期变化的逻辑

取指令阶段

第一个时钟周期，送指令地址：MAR<-(PC);PCOeH,PCOeL,MARWr/A<-(PC);AWr

第二个时钟周期，取指令：IR<-M[MAR];MemRd,MemOe,IRWr

第三个时钟周期，送 PC 修正量：B<-1;ExtSel=10,ImmOeH,ImmOeL,BWr

第四个时钟周期，修正 PC：PC<-(PC)+4;ALUOp=00,ALUCtrl=100,ALUOe,PCWr

第五个时钟周期，指令译码和读寄存器：A<-(RF[IR[25-21]]);RegSel=00,RegOe,AWr

指令执行阶段

（1） 取数指令(lw)

第六个时钟周期，计算存储器地址：B<-(SigExt(IR[15-0]));ExtSel=00,ImmOeH,ImmOeL,BWr

第七个时钟周期，送存储器地址：MAR<-(A)+(B);ALUOp=00,ALUCtrl=100,ALUOe,MARW

第八个时钟周期，读存储器：MDR<-M[MAR];MemRd,MDRSrc,MDRWr

第九个时钟周期，写寄存器：RF[IR[20-16]]<-(MDR);MDROe,RegSel=01,RegWr

（2） 存数指令(sw)

第六个时钟周期，计算存储器地址：B<-(SigExt(IR[15-0]));ExtSel=00,ImmOeH,ImmOeL,BWr

第七个时钟周期，送存储器地址：MAR<-(A)+(B);ALUOp=00,ALUCtrl=100,ALUOe,MARWr

第八个时钟周期，读寄存器 MDR<-(RF[IR[20-16]];RegSel=01,RegOe,MDRSrc=0,MDRWr

第九个时钟周期，写存储器：M[MAR]<-(MDR); MemWr

（3） 算数/逻辑运算指令(R 型)

第六个时钟周期，读寄存器：B<-(RF[IR[20-16]]);RegSel=01,RegOe,BWr

第七个时钟周期，运算并写寄存器：RF[IR[15-11]]<-(A)OP(B);ALUOp=10,ALUCtrl=xxx,ALUOe,RegSel=10,RegWr

（4） 分支指令(beq)

第六个时钟周期，程序转移：B<-(RF[IR[20-16]]);RegSel=01,RegOe BWr

第七个时钟周期，送 PC：If Z==0 then 指令周期结束;ALUOp=01,ALUCtrl=110 A<-(PC);PCOeH,PCOeL,AWr

第八个时钟周期，送 PC 修正量：B<-(SigExt(IR[15-0])<<2);ExtSel=01.ImmOeH,ImmOeL,BWr

第九个时钟周期，计算转移地址：PC<-(A)+(B);ALUOp=00,ALUCtrl=100,ALUOe,PCWr

（5） 跳转指令(j)

第六个时钟周期，转移：PC<-(PC[31-28])||(IR[25-0]);PCOeH,ExtSel=11,ImmOeL,PCWr

4. CU 组件的组织形式

在 CU 组件中，设置一个 period 变量，控制当前所处的时钟周期数，初始值设置为 0，每遇到 CLK 信号上沿，则根据当前指令的操作码输出相应的控制信号，并进行 period（时钟周期）的转移。当遇到 CLK 信号下沿时，则将所有控制信号清零，程序片段如下图所示：

```
    integer period=0; //clock period
    integer ex=0;
    reg [31:0] OP_code;

always@(posedge clk)begin
    case(period)
        0:begin
            PCOeH=1;
            PCOeL=1;
            MARWr=1;
            AWr=1;
            period=period+1;
        end
        1:begin
            MemRd=1;
            MemOe=1;
            IRWr=1;
            period=period+1;
            #2
            OP_code=MCU_in;
        end
        2:begin
            ExtSel=2'b10;
            ImmOeH=1;
            ImmOeL=1;
            BWr=1;
            period=period+1;
        end
        3:begin
            ALUOp=2'b00;
```

```
            6'b000000:begin //R
                RegSel=2'b01;
                RegOe=1;
                BWr=1;
                period=period+1;
            end
            6'b000100:begin //beq
                RegSel=2'b01;
                RegOe=1;
                BWr=1;
                period=period+1;
            end
            6'b000010:begin //j
                PCOeH=1;
                ExtSel=2'b11;
                ImmOeL=1;
                PCWr=1;
                OP_code=0;
                period=0;
            end
        endcase
    end
    6:begin
        case(OP_code[31:26])
            6'b100011:begin //lw
                ALUOp=2'b00;
                //ALUCtrl=3'b100;
                ALUOe=1;
                MARWr=1;
                period=period+1;
```

对于无条件跳转指令，一个指令周期仅包含六个时钟周期

对于每个时钟周期，若不是该指令周期中的最后一个时钟周期，则信号设置结束后执行 period=period+1，否则 period 直接跳转回 0，开始下一个指令周期

5. 实验测试

在程序存储器中初始化一端小程序，如下所示，程序的作用在注释中给出，其中，在最后一条指令中 PC 被设置为 1，从指令 DM[0]开始重新执行，实验结果中仅给出额外执行一轮的结果。

```
memory[0]=32'b00000000000000010001000000100001; //RF[2]=RF[0]+RF[1]=1
memory[1]=32'b00000000100000110010100001100010; //RF[5]=RF[4]-RF[3]=1
memory[2]=32'b10001100001001000000000000011000; //DM[24+1](9)->RF[4]=9
memory[3]=32'b10101100010111110000000000001111; //RF[31](31)->DM[15+1]=31
memory[4]=32'b00000000000000010011100000101011; //RF[0]<RF[1]?->RF[7]=1(triggered)
memory[5]=32'b00010000001000100000000000000001; //RF[1]==RF[2]?->PC+=1(triggered)
memory[6]=32'b10001100100000000000000000000111; //DM[7+9](31)->RF[0]=31(skipped)
memory[7]=32'b10001100100000000000000000001000; //DM[8+9](1)->RF[0]=1
memory[8]=32'b00001000000000000000000000000000; //PC=0
                                                 //PC=0 RF[2]=RF[0]+RF[1]=2
                                                 //PC=1 RF[5]=RF[4]-RF[3]=6
                                                 //PC=2 DM[24+1](9)->RF[4]=9
                                                 //PC=3 RF[31](31)->DM[15+2]=31
                                                 //PC=4 RF[0]<RF[1]?->RF[7]=1(not triggered)
                                                 //PC=5 RF[1]==RF[2]?->PC+=1(not triggered)
                                                 //PC=6 DM[7+9](31)->RF[0]=31
                                                 //PC=7 DM[8+9](31)->RF[0]=31
```

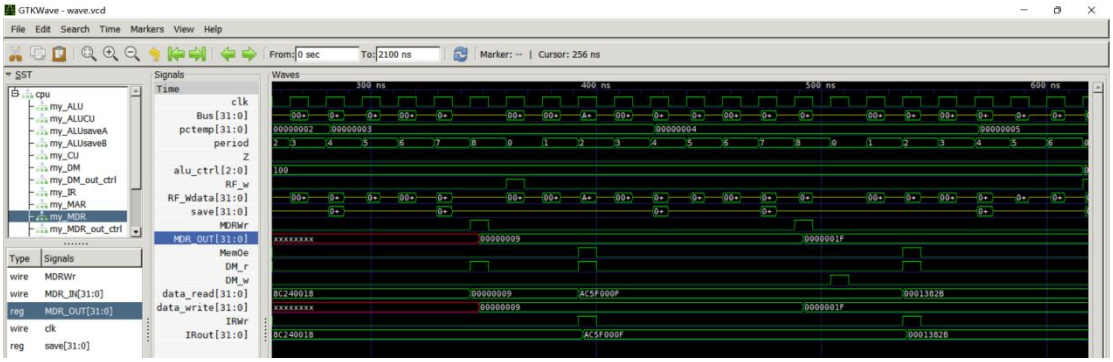
初始化时，将 reg0~reg7 分别赋值为 0~7，将 DM[16]~DM[24]分别赋值为 0~7

在运行时，每过一个指令周期,打印这 8 个寄存器和 8 个主存的内容，运行结果如下图所示

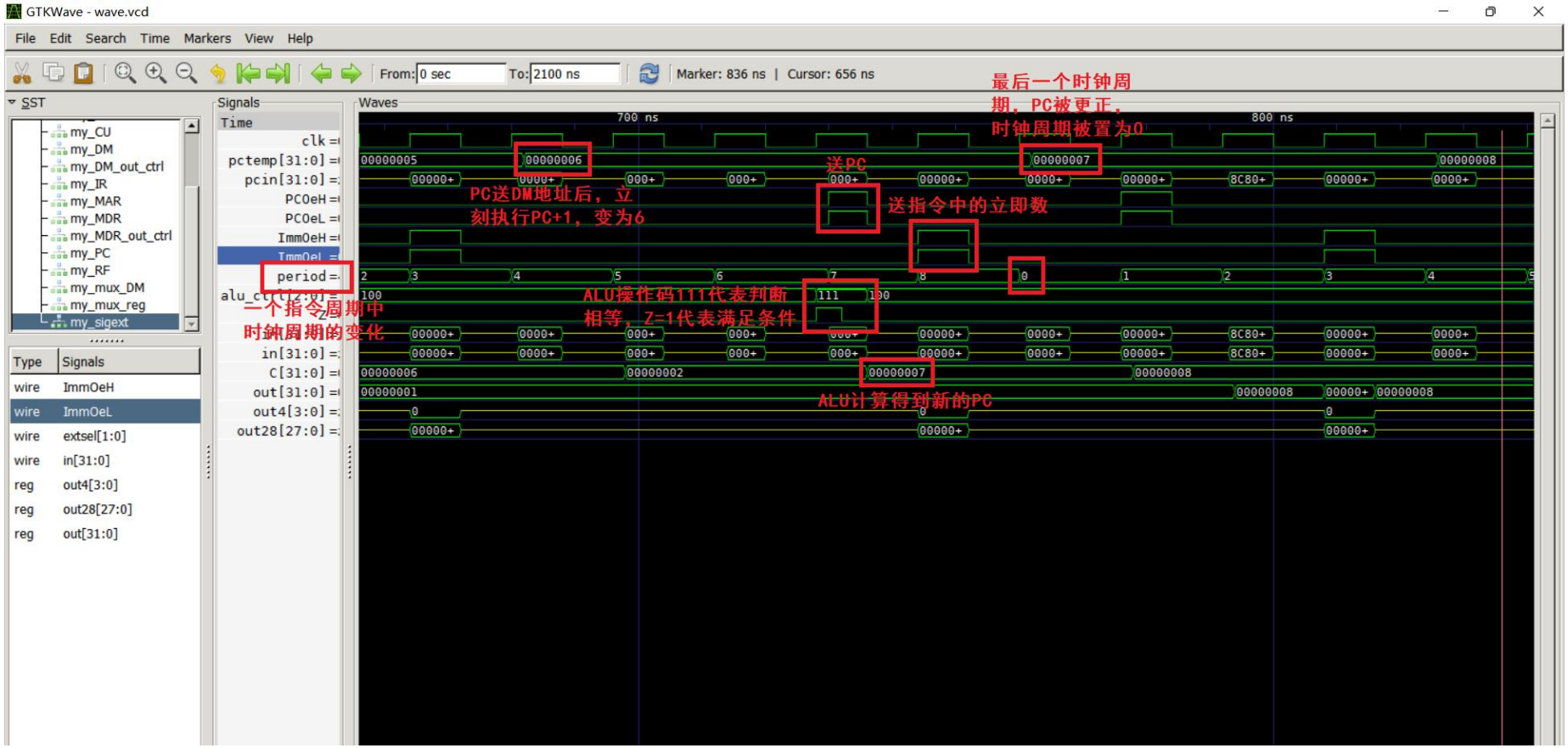
```
C:\WINDOWS\system32\cmd.exe
E:\files\计算机组成原理\实验\多周期cpu>vvp a.out
----------
reg[0-7]        0        1        2        3        4        5        6        7
dm[16-23]       0        1        2        3        4        5        6        7
VCD info: dumpfile wave.vcd opened for output.
----------
reg[0-7]        0        1        1        3        4        5        6        7
dm[16-23]       0        1        2        3        4        5        6        7
----------
reg[0-7]        0        1        1        3        4        1        6        7
dm[16-23]       0        1        2        3        4        5        6        7
----------
reg[0-7]        0        1        1        3        9        1        6        7
dm[16-23]       0        1        2        3        4        5        6        7
----------
reg[0-7]        0        1        1        3        9        1        6        7
dm[16-23]      31        1        2        3        4        5        6        7
----------
reg[0-7]        0        1        1        3        9        1        6        1
dm[16-23]      31        1        2        3        4        5        6        7
----------
reg[0-7]        0        1        1        3        9        1        6        1
dm[16-23]      31        1        2        3        4        5        6        7
----------
reg[0-7]        1        1        1        3        9        1        6        1
dm[16-23]      31        1        2        3        4        5        6        7
----------
reg[0-7]        1        1        1        3        9        1        6        1   <-PC=0，重新开始执行
dm[16-23]      31        1        2        3        4        5        6        7
----------
reg[0-7]        1        1        2        3        9        1        6        1
dm[16-23]      31        1        2        3        4        5        6        7
----------
reg[0-7]        1        1        2        3        9        6        6        1
dm[16-23]      31        1        2        3        4        5        6        7
----------
reg[0-7]        1        1        2        3        9        6        6        1
dm[16-23]      31        1        2        3        4        5        6        7
----------
reg[0-7]        1        1        2        3        9        6        6        1
dm[16-23]      31       31        2        3        4        5        6        7
----------
reg[0-7]        1        1        2        3        9        6        6        0
dm[16-23]      31       31        2        3        4        5        6        7
----------
reg[0-7]        1        1        2        3        9        6        6        0
dm[16-23]      31       31        2        3        4        5        6        7
----------
reg[0-7]       31        1        2        3        9        6        6        0
```

可以看到，两轮执行的算数运算指令由于参与运算的值发生变化，得到的结果有所不同，产生了链锁反应，导致后续的存取数指令的地址有所变化，条件指令（判断大小和条件转移）的结果也有所不同。例如，在第一轮的指令中，reg0 和 reg1 的值分别为 0 和 1，reg2 的值计算后为 1，这导致第 3 条指令中的寻址过程发生变化（第一次是 DM[16]被写入，这一次是 DM[17]被写入）第 4 和第 5 条判断指令和条件转移指令都满足条件，而条件转移指令跳

过了第 6 条指令，执行第 7 条指令，将 reg0 设置为 1，随后在第 8 条指令中 PC 被设置为 0，重新开始执行。这一次，reg2=reg0+reg1 计算后值为 2，再加上第一轮最后 reg0 的值被设为 1，导致这一次判断大小指令和条件转移指令均不满足，执行第 6 条指令，将 reg0 设置为 31，随后在第 7 条指令中，由于存储器中 DM[17]的值已经在之前被设置为 31，因此 reg0 的值仍然为 31。其他所有的执行也都正确符合程序，一些关键环节的波形如下图所示。



图中截获了一些关键信息，例如 ck 信号，PC 的值，period 表示指令中期中时钟周期的序号，主存的读写时间等等，一下以第一次执行第 5 条指令为例，分析波形，见下页：


GTKWave - wave.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 2100 ns
Marker: 836 ns | Cursor: 656 ns
SST
my_CU
my_DM
my_DM_out_ctrl
my_IR
my_MAR
my_MDR
my_MDR_out_ctrl
my_PC
my_RF
my_mux_DM
my_mux_reg
my_sigext
Type Signals
wire ImmOeH
wire ImmOeL
wire extsel[1:0]
wire in[31:0]
reg out4[3:0]
reg out28[27:0]
reg out[31:0]
Signals
Time
clk
pctemp[31:0]
pcin[31:0]
PCOeH
PCOeL
ImmOeH
ImmOeL
period
alu_ctrl[2:0]
in[31:0]
C[31:0]
out[31:0]
out4[3:0]
out28[27:0]
Waves
700 ns
800 ns
最后一个时钟周期，PC被更正，时钟周期被置为0
送PC
PC送DM地址后，立刻执行PC+1，变为6
送指令中的立即数
一个指令周期中时钟周期的变化
ALU操作码111代表判断相等，Z=1代表满足条件
ALU计算得到新的PC