

- **整数指令**：算术、逻辑和移位运算
- **访存指令**：内存读取、内存写入
- **转移指令**：分支和跳转

功能测试指令：

<pre>wait_1s: LI (t0,SW_INTER_ADDR) LI (t1, 0xaaaa) lw t2, 0x0(t0) xor t2, t2, t1 sll t3, t2, 9 addiu t3, t3, 1</pre>	—————→ 访存
<pre>sub1: addiu t3, t3, -1 lw t2, 0x0(t0) xor t2, t2, t1 sll t2, t2, 9 sltu t4, t3, t2 bnez t4, 1f nop addu t3, t2, 0</pre>	—————→ 算术运算
	—————→ 逻辑运算
<pre>1: bne t3,zero, sub1 nop</pre>	—————→ 分支
<pre>JR (ra)</pre>	—————→ 跳转

tinymips已经实现的22条指令：

- **算术运算**：ADDU、ADDIU、SUBU、SLT、SLTU
- **逻辑运算**：AND、LUI、OR、XOR
- **移位运算**：SLL、SLLV、SRAV、SRLV
- **条件分支**：BEQ、BNE
- **跳转**：JAL、JALR
- **访存**：LB、LBU、LW、SB、SW

逻辑移位0 算术移位符号位

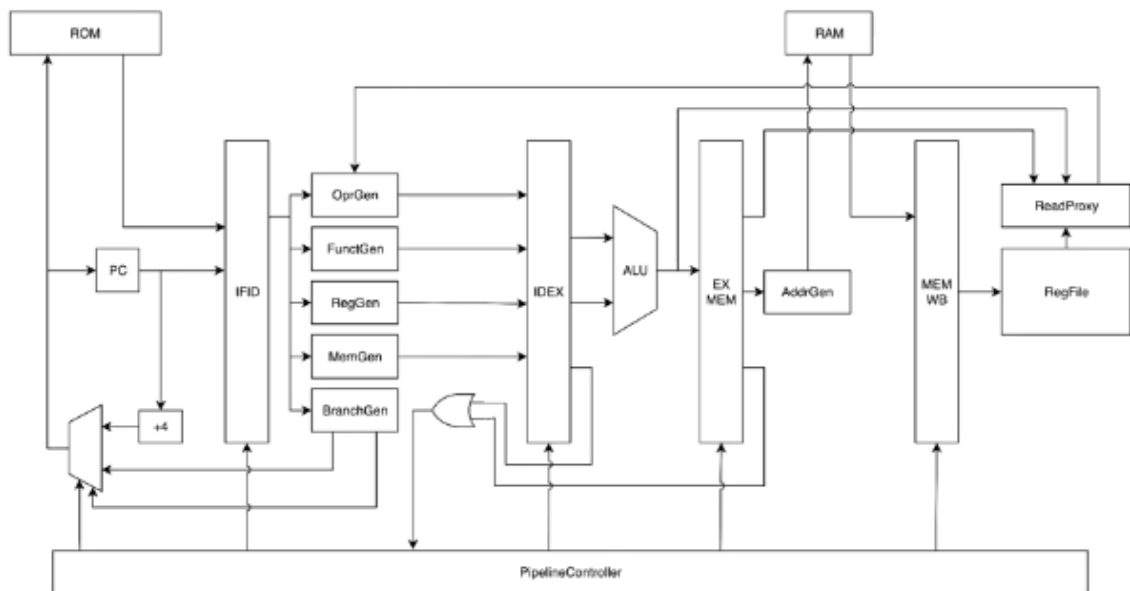
MIPS的寄存器堆：

- **32个**通用寄存器，编号为0~31
- 0号寄存器**始终为0** (hardwired to zero)
- 其他寄存器原则上可以存储**任何数据**，但其中的某些往往会被ABI规定作特定用途

Register Number	Conventional Name	Usage
\$0	\$zero	Hard-wired to 0
\$1	\$at	Reserved for pseudo-instructions
\$2 - \$3	\$v0, \$v1	Return values from functions
\$4 - \$7	\$a0 - \$a3	Arguments to functions - not preserved by subprograms
\$8 - \$15	\$t0 - \$t7	Temporary data, not preserved by subprograms
\$16 - \$23	\$s0 - \$s7	Saved registers, preserved by subprograms
\$24 - \$25	\$t8 - \$t9	More temporary registers, not preserved by subprograms
\$26 - \$27	\$k0 - \$k1	Reserved for kernel. Do not use.
\$28	\$gp	Global Area Pointer (base of global data segment)
\$29	\$sp	Stack Pointer
\$30	\$fp	Frame Pointer
\$31	\$ra	Return Address

延迟槽：执行分支指令时自动执行分支指令后的一条指令，不论是否跳转，可以在分支指令后自动加一条nop空操作

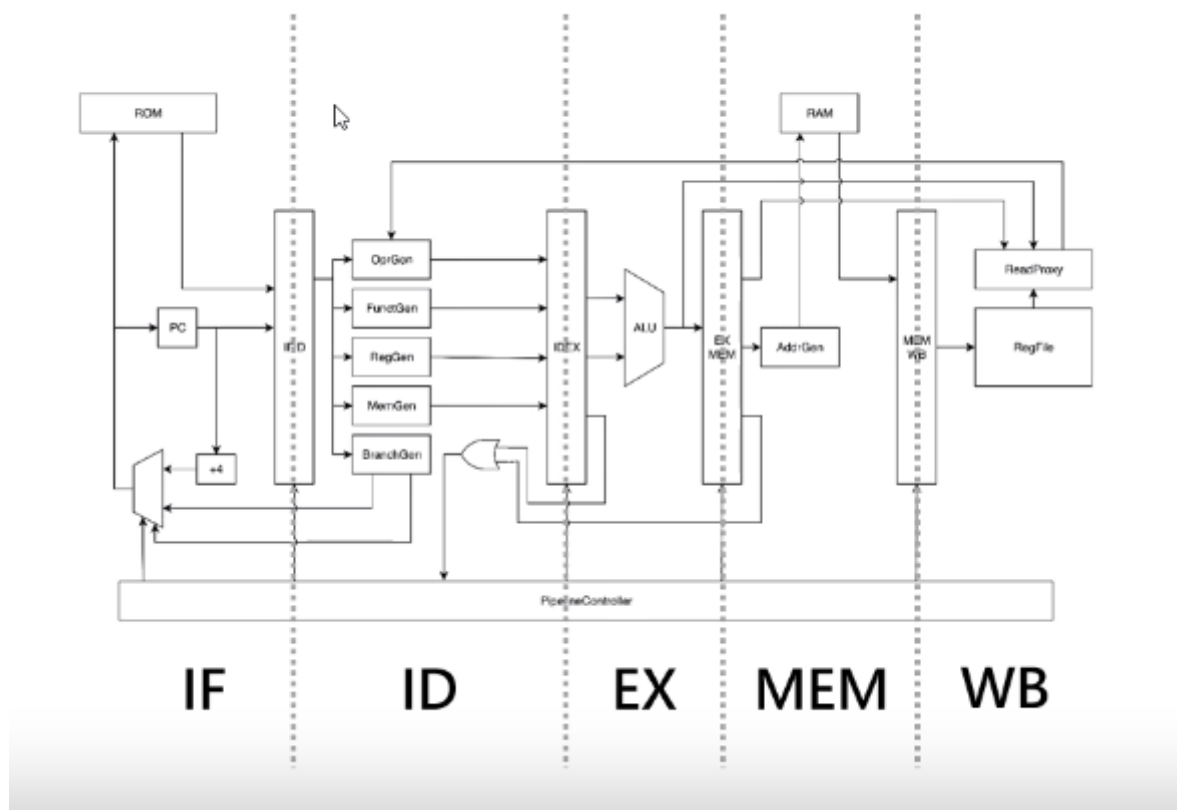
- 汇编器把汇编代码翻译成二进制数据
- 二进制数据被使用某种方式放在了存储器里
- CPU上电后从存储器取出指令并执行



五级流水：取指、译码、执行、访存、写回

经典五级流水线

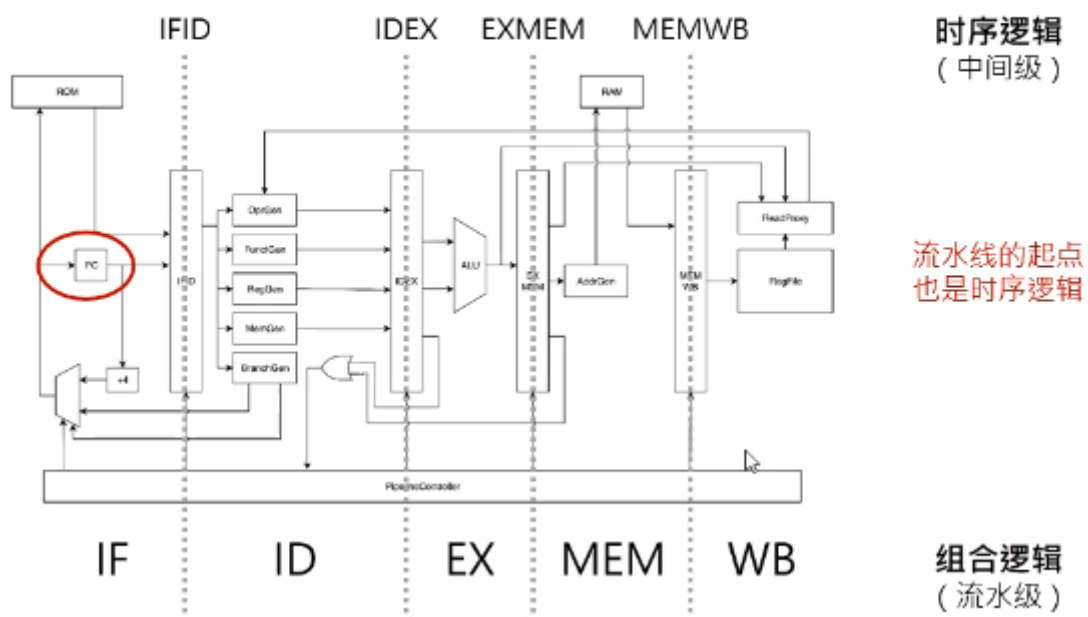
- 取指 (IF)：根据PC从存储器取指交给下一级，更新PC
- 译码 (ID)：获取指令的操作数，生成相关控制信号
- 执行 (EX)：根据操作数和控制信号，计算指令的结果
- 访存 (MEM)：如果需要访存，则从/向内存读取/写入数据
- 写回 (WB)：将指令执行的最终结果写回寄存器堆



四个中间级：时序逻辑

五个流水级：组合逻辑

流水线的起点pc也是时序逻辑



在理想情况下，可以简单视为D触发器相连，有统一的时钟信号触发，其状态转化与五级流水线类似（移位寄存器）

那么五个流水级相当于跟五个D触发器对应

最早可以在译码级ID确定是转移指令，但是此时IF已经取出了转移指令的后一条指令（PC+4）（延迟槽），（所以可以在跳转指令后面加一个NOP），相当于一种妥协，为了提高cpu效率的妥协

转移指令：

- **取指级：**正常取出指令，PC加4
- **译码级：**发现是转移指令，生成控制信号告诉取指级进行转移
- **执行级：**什么也不干
- **访存级：**什么也不干
- **写回级：**可能会写回数据（JAL、JALR）

实际的硬件系统往往是冯诺依曼结构，可以从rom和ram中取指令

cpu操作外设实际上也就是cpu向总线请求，“访存”请求

axi是cpu与总线交互的接口

流水线解决冲突：数据相关、控制相关、结构相关

重点考虑数据相关

写后读

regproxy进行数据转发

EX、MEM--》ID进行转发

但是涉及到访存时数据转发无法解决

因为如果写后读中写的对象是寄存器的话，在wb才会写回

但是如果写的对象是内存的话，在mem才知道自己要写回mem，如果后一条指令要读相应的内存，处于id，但是前一条指令处于ex，并不知道自己要写回mem，因此跟数据转发不一样。可以暂时暂停前面的流水及，让后面的流水及运行

验收问题：

Q：存储器冲那个文件读取功能测试文件？

A：soft文件夹下，starts的文件修改，后编译为二进制文件，然后在vivado工程中读取.coe文件，其中为指令的十六进制

Q：为什么cpu能够读取波荡开关的值？

A：通过访存指令，

Q: 为什么cpu执行功能测试的延迟随波荡开关实时变化

A: 多次调用wait1s

Q: cpu的写回级会写回pc寄存器嘛?

A: 不会, pc的更新要么+4, 要么分支, 与wb级无关

Q: trace比对是每隔10000ns比对一次嘛?

A: 不是, trace比对频率跟时钟信号频率有关 (always参数), 只是每隔10000ns打印一次信息, 两个不在一个always块中

Q: 为什么某些pc在trace文件中找不到? 为什么有时输出0x0000000

A:

```
begin
    debug_wb_err <= 1'b0;
end
else if(|debug_wb_rf_wen && debug_wb_rf_wnum!=5'd0 && !debug_end && `CONFREG_OPEN_TRACE)
begin
    if ( (debug_wb_pc!=ref_wb_pc) || (debug_wb_rf_wnum!=ref_wb_rf_wnum)
        || (debug_wb_rf_wdata_v!=ref_wb_rf_wdata_v) )
    begin
        $display("-----");
        $display("[%t] Error!!!", $time);
        $display("    reference: PC = 0x%8h, wb_rf_wnum = 0x%2h, wb_rf_wdata = 0x%8h",
            ref_wb_pc, ref_wb_rf_wnum, ref_wb_rf_wdata_v);
        $display("    mycpu      : PC = 0x%8h, wb_rf_wnum = 0x%2h, wb_rf_wdata = 0x%8h",
            debug_wb_pc, debug_wb_rf_wnum, debug_wb_rf_wdata_v);
        $display("-----");
        debug_wb_err <= 1'b1;
        #40;
        $finish;
    end
end
end
```

并不是每一条指令都会进行trace比对, 只有满足一定条件才会进行比对, 如需要写入寄存器堆、写入寄存器不是\$0, 都实行完毕、open_trace变量控制防止由于性能上的差异导致比对失败

axi协议, 在一个周期内取不到数据, 会将自己暂停, pc设置为0

Q: 功能仿真时为什么输出pass

A: 每过一个功能点都会输出一次pass

Q: 上班时cpu是否还执行trace比对

A: 不进行, trace比对发生在仿真时, 上板时另一种方法发现错误。cpu自己与答案进行比较,

Q: 延迟槽里也是一条分支指令?

A: 处理器行为不可预知, 这是自行规定的

