北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

CPU设计先导

主讲人：蔡建

德以明理 学以精工

CPU时序系统

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

指令周期

一条指令完成所需要的总时间。不同指令的指令周期可能不同。

机器周期

机器周期也称为CPU周期，表示CPU内部完成一个基本操作所需要的时间，比如：取指周期、执行周期、取数周期等等。CPU周期又可以分为定长CPU周期和不定长CPU周期两种。

总线周期

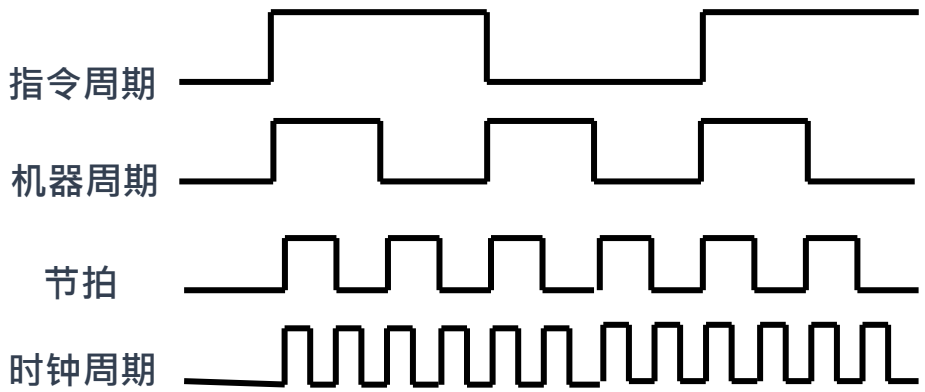
CPU在总线上完成一次读/写操作所需要花费的时间。

节拍

一个机器周期根据操作目的划分为若干微操作，每个微操作使用相同的时间段，称为节拍。

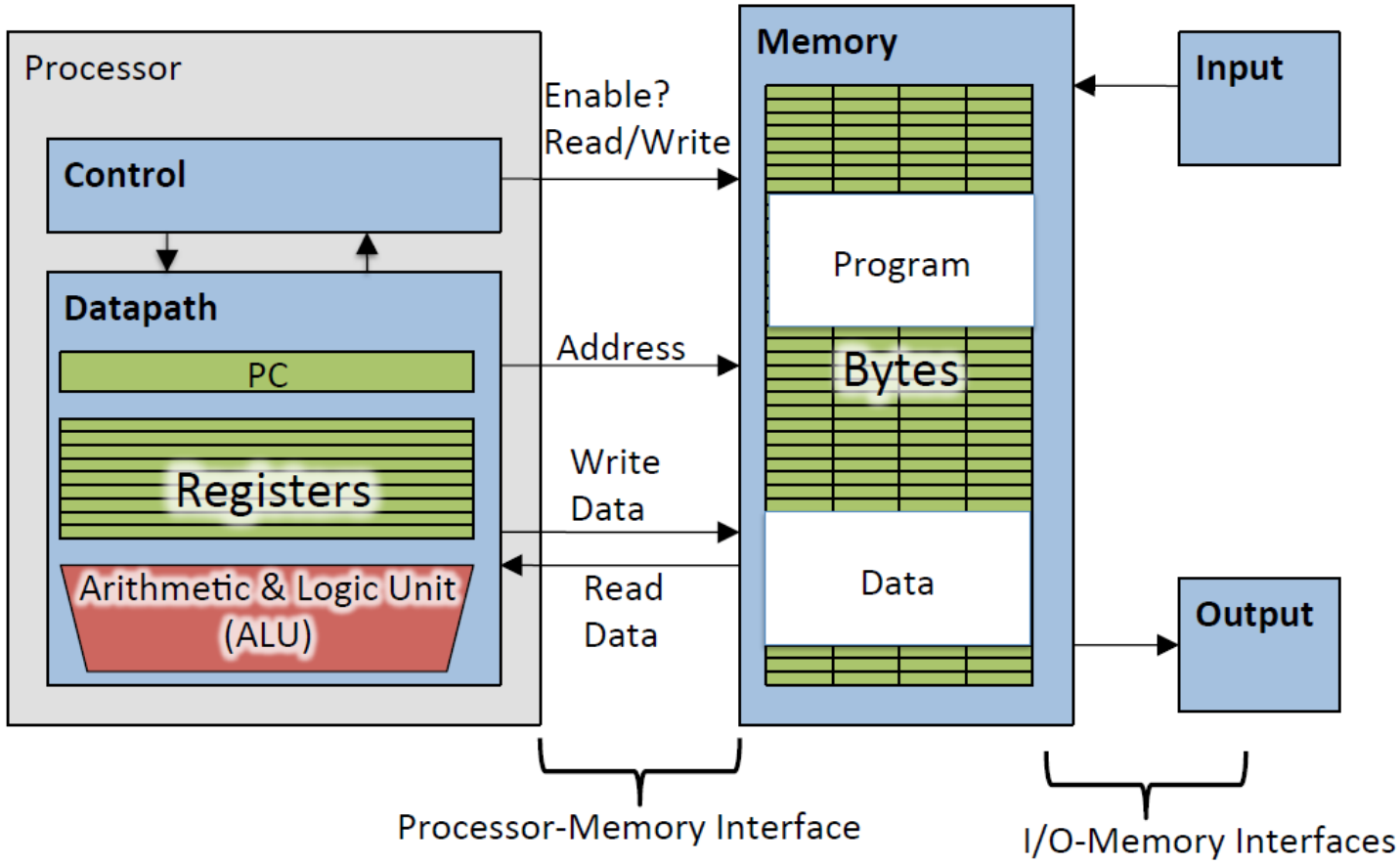
时钟周期

时钟周期也称为振荡周期，片上晶振产生。定义为时钟频率的倒数。时钟周期是计算机中最基本的、最小的时间单位。

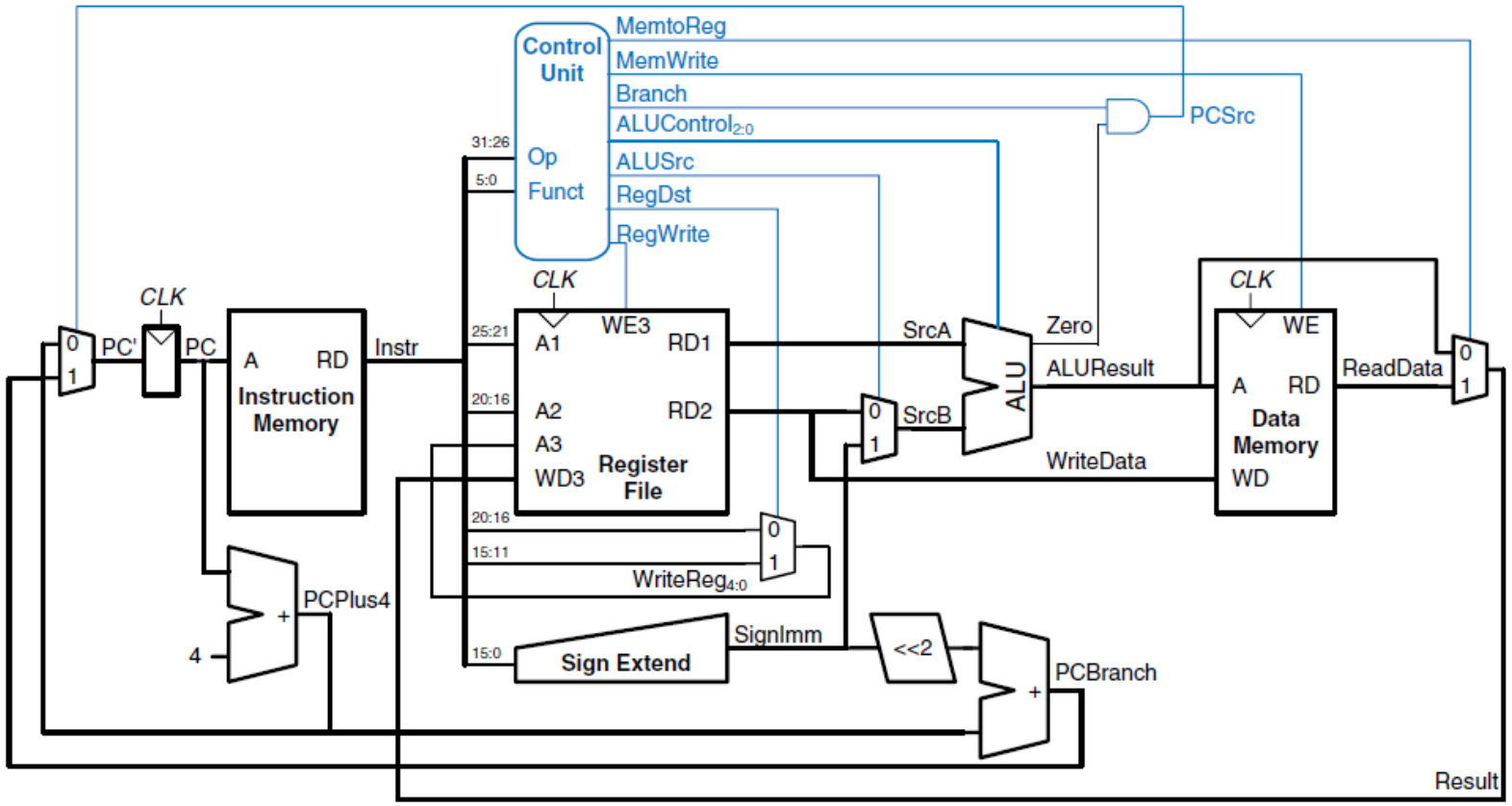


数据通路、控制器与存储元件

- 数据通路：处理器中执行算术操作的部分，以算数逻辑单元ALU为核心。
- 控制器：处理器中根据程序的指令指挥数据通路、存储器和I/O 的部分。
- 存储元件：CPU内部寄存器、外部内存memory、中间cache等。



传统CPU设计的五个基本阶段



将指令的执行结果写回目的寄存器。

写回

访存

load/store类指令访问内存。

ALU执行算数逻辑相关运算，可能伴随着异常检测。

执行

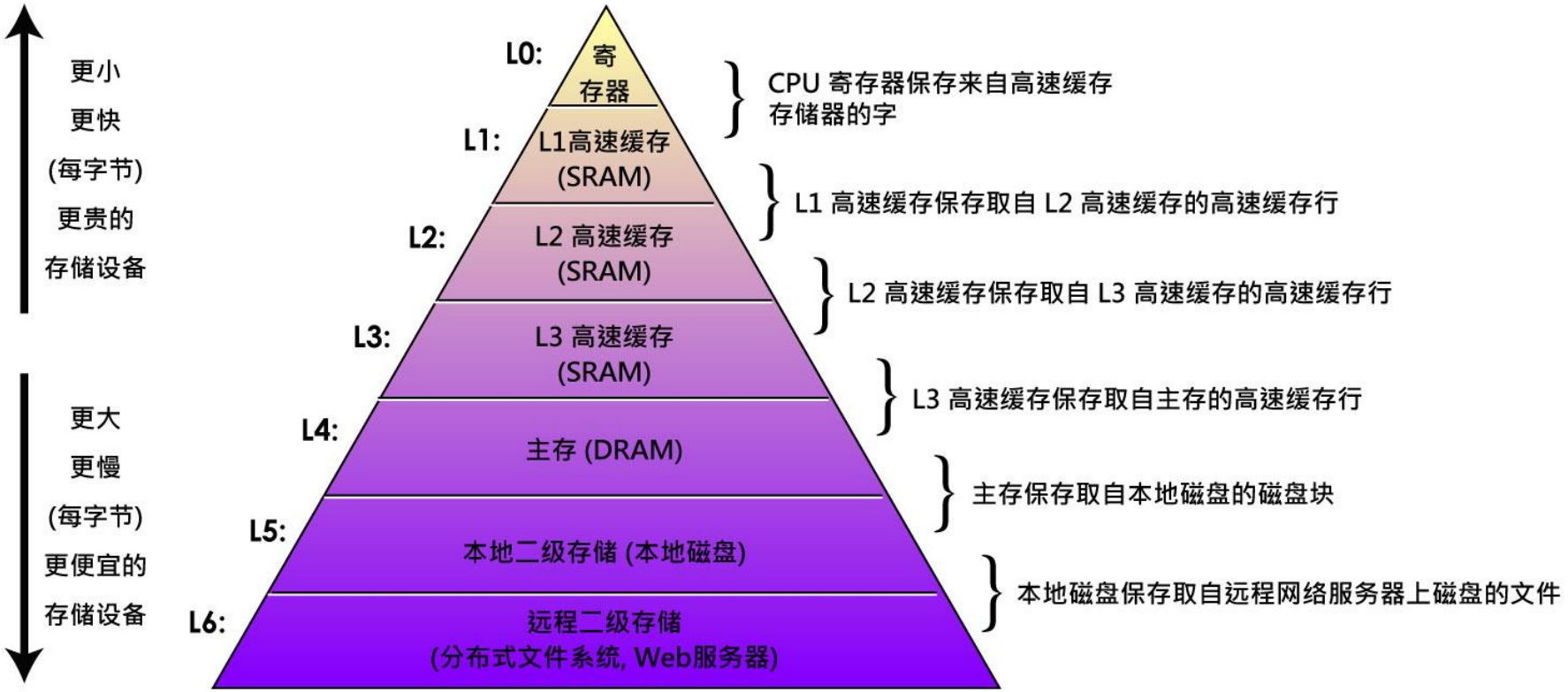
译码

从32位指令读取操作码、寄存器地址、立即数等数据生成操作数与控制信号。

PC给出指令地址，从指令存储器取出指令。

取指

计算机存储体系

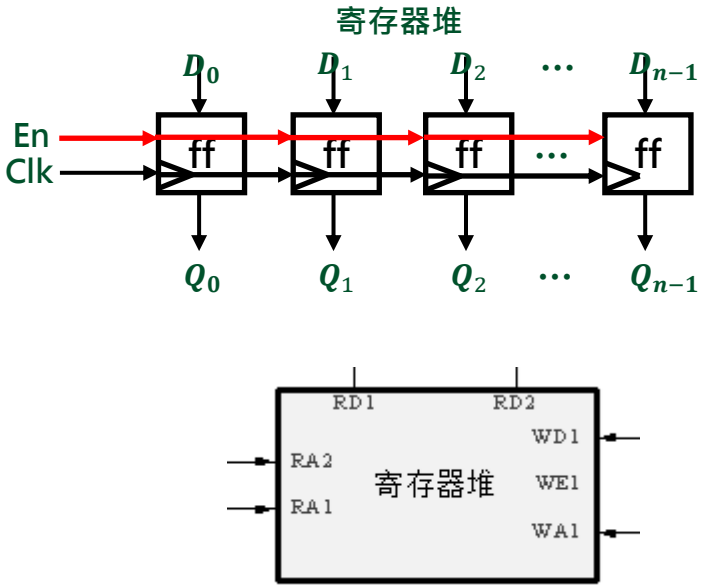


实验建议大家采用指令存储器与数据存储器分开的哈佛结构：

- 简化设计
- 满足后续单周期CPU设计时在一个时钟周期内完成任意指令的任务。
- 避免后续流水线CPU设计时不同流水阶段对内存资源的竞争。

寄存器堆

1个32位寄存器由32个触发器组成，多个寄存器集合成寄存器堆。



REGISTERS		
0	zero	Always equal to zero
1	at	Assembler temporary; used by the assembler
2-3	v0-v1	Return value from a function call
4-7	a0-a3	First four parameters for a function call
8-15	t0-t7	Temporary variables; need not be preserved
16-23	s0-s7	Function variables; must be preserved
24-25	t8-t9	Two more temporary variables
26-27	k0-k1	Kernel use registers; may change unexpectedly
28	gp	Global pointer
29	sp	Stack pointer
30	fp/s8	Stack frame pointer or subroutine variable
31	ra	Return address of the last subroutine call

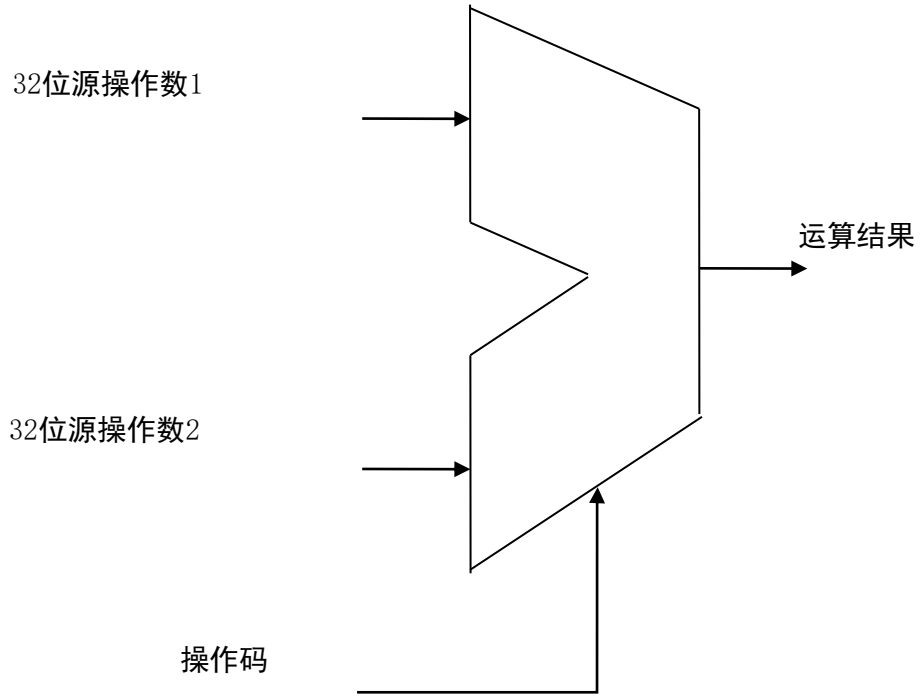
两读一写寄存器堆

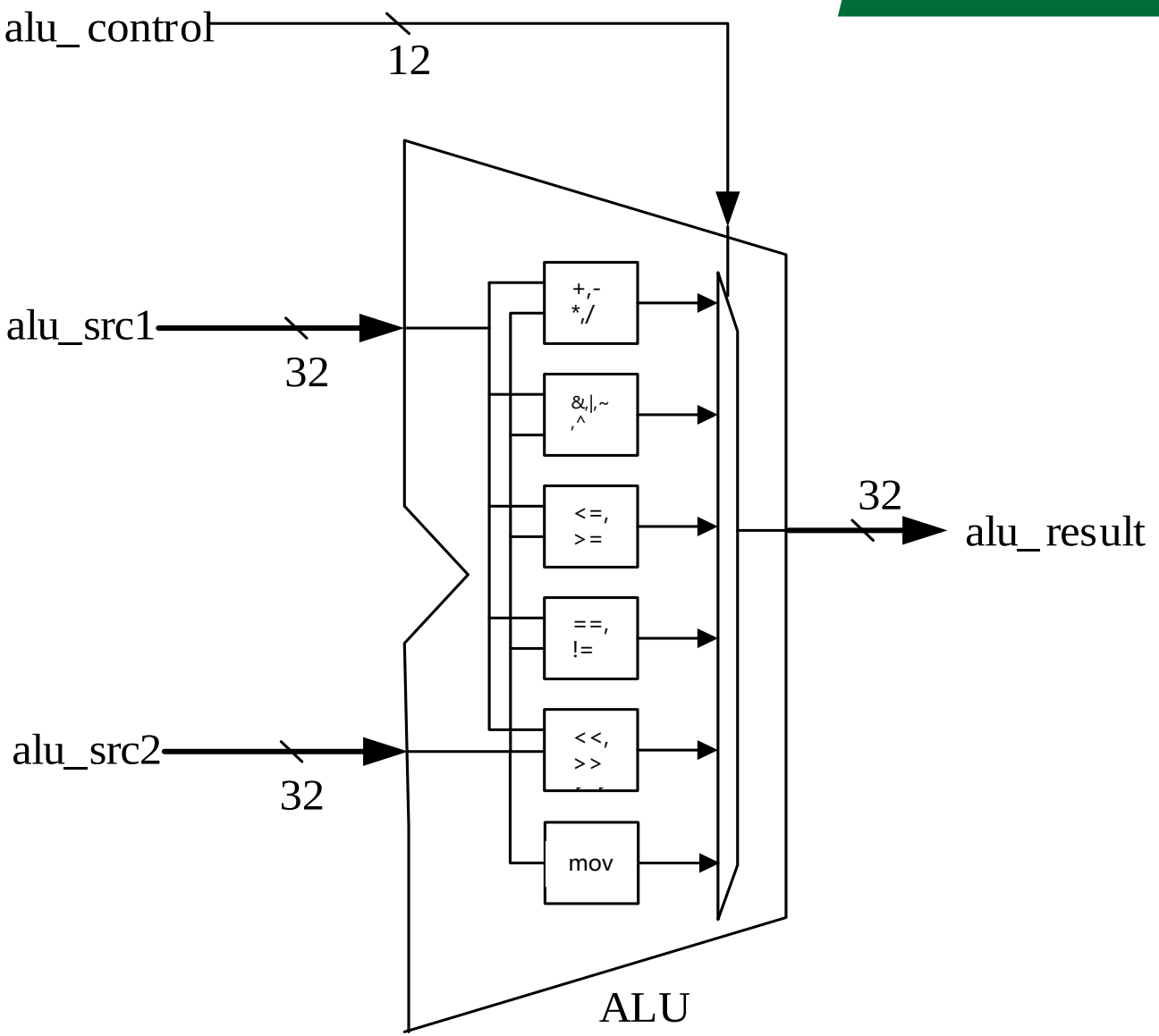
myCPU > regfile.v

```
1   `timescale 1ns / 1ps
2
3   module regfile(
4       input clk,
5       input rst,
6
7       input reg_we,
8       input [4:0] rs_addr,
9       input [4:0] rt_addr,
10      input [4:0] wb_addr,
11      input [31:0] wb_data,
12
13      output [31:0] rs_data,
14      output [31:0] rt_data
15      );
16      reg [31:0] gpr[31:0];
17      integer i;
18      always @(posedge clk or negedge rst) begin
19          if(!rst) begin
20              for(i=0;i<=31;i=i+1) gpr[i] <= 32'b0;
21          end
22          else if(reg_we) gpr[wb_addr] <= wb_data;
23      end
24
25      assign rs_data = gpr[rs_addr];
26      assign rt_data = gpr[rt_addr];
27
28  endmodule
29
```

为了方便演示，因为程序量小，后续也直接用寄存器堆的方式模拟指令存储器和数据存储器，这样每条指令的完成只需一个时钟周期（注意：真实的实际情况不会直接用寄存器当内存）。

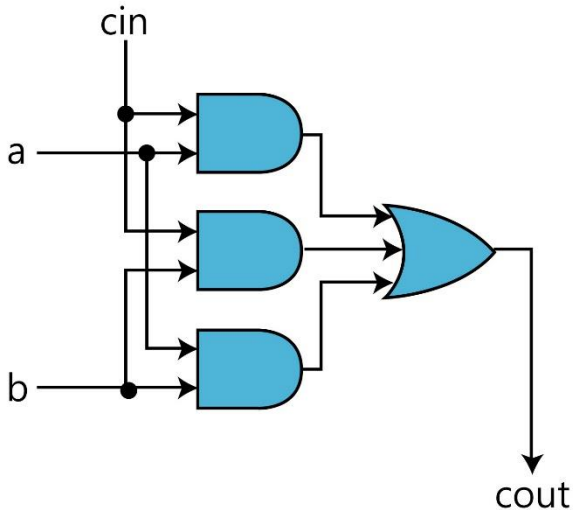
算数逻辑单元ALU

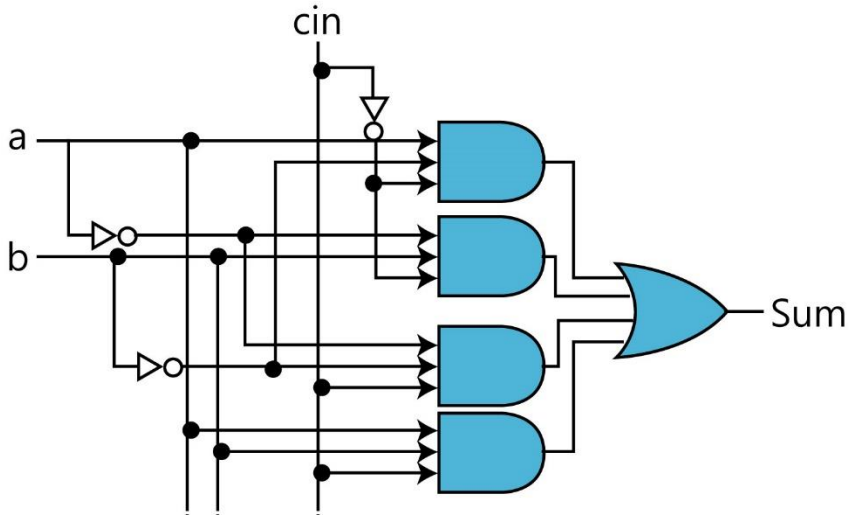

32位源操作数1
32位源操作数2
操作码
运算结果

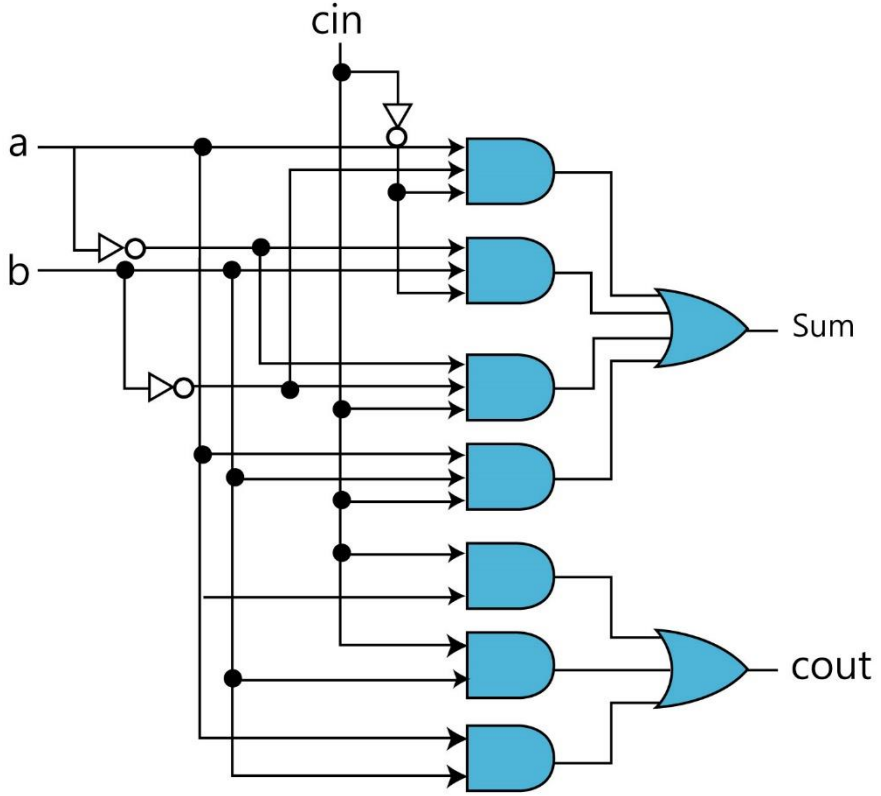

alu_ control
12
alu_src1
32
alu_src2
32
+,-
*,/
&,|,~
,^
<=,
>=
==,
!=
<<,
>>
mov
32
alu_result
ALU

一位全加器

```
module full_adder(a,b,cin,s,cout)
    input a, b, cin;
    output s, cout;
    assign s = a^b^cin;
    assign cout = (a&b) | (b&cin) | (cin&a);
endmodule
```

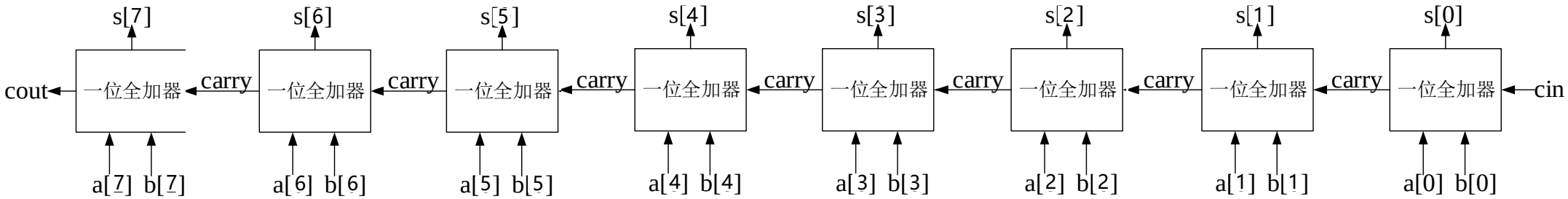




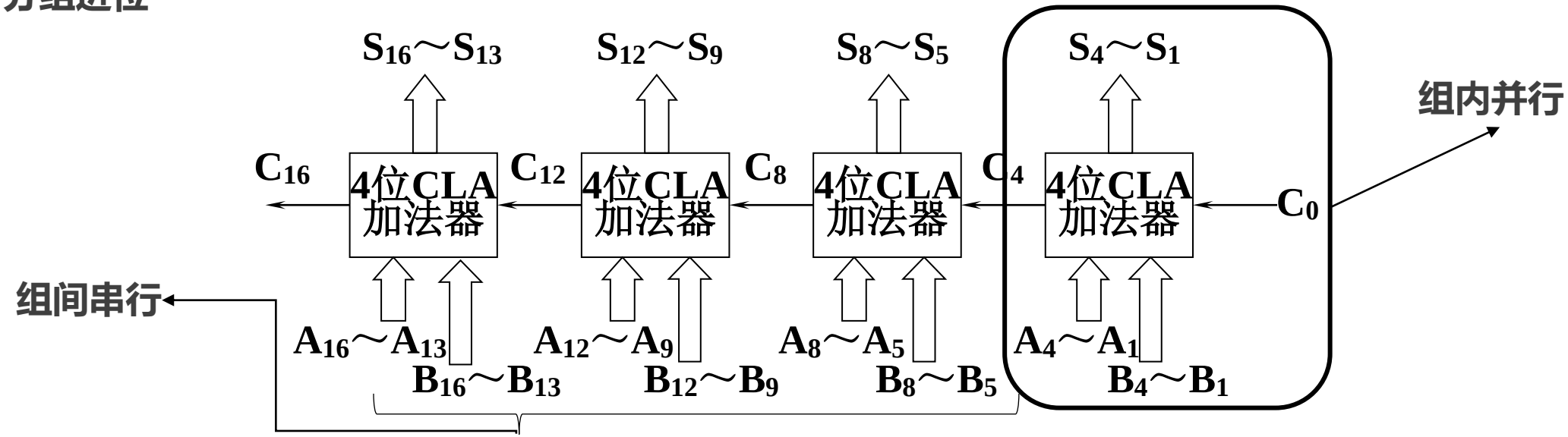


多位加法器

串行进位

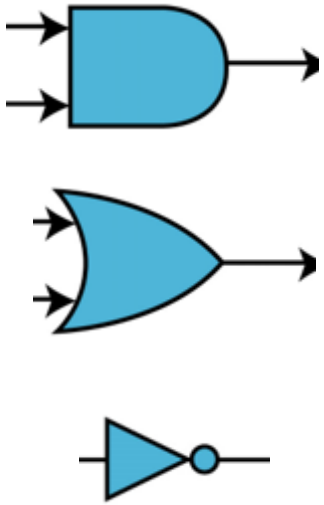


分组进位

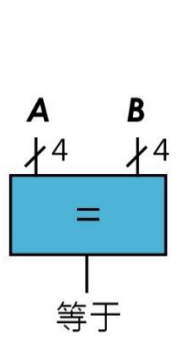


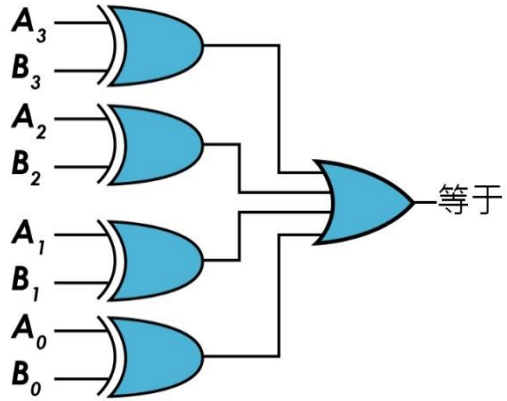
减法、逻辑与比较运算

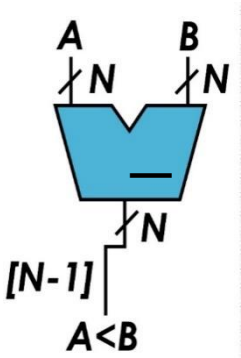
- 减法
  - $A-B = A+[B]_{补}$
  - $[B]_{补} = \sim B + 1$
  - 需要：非门与加法器
- 与
  - 与门
- 或
  - 或门
- 非
  - 非门



- 相等比较
  - 异或门
  - 与门
- 量值比较
  - 减法器
  - 看符号位
- 其它组合逻辑
  - 列真值表
  - 与、或、非门的组合

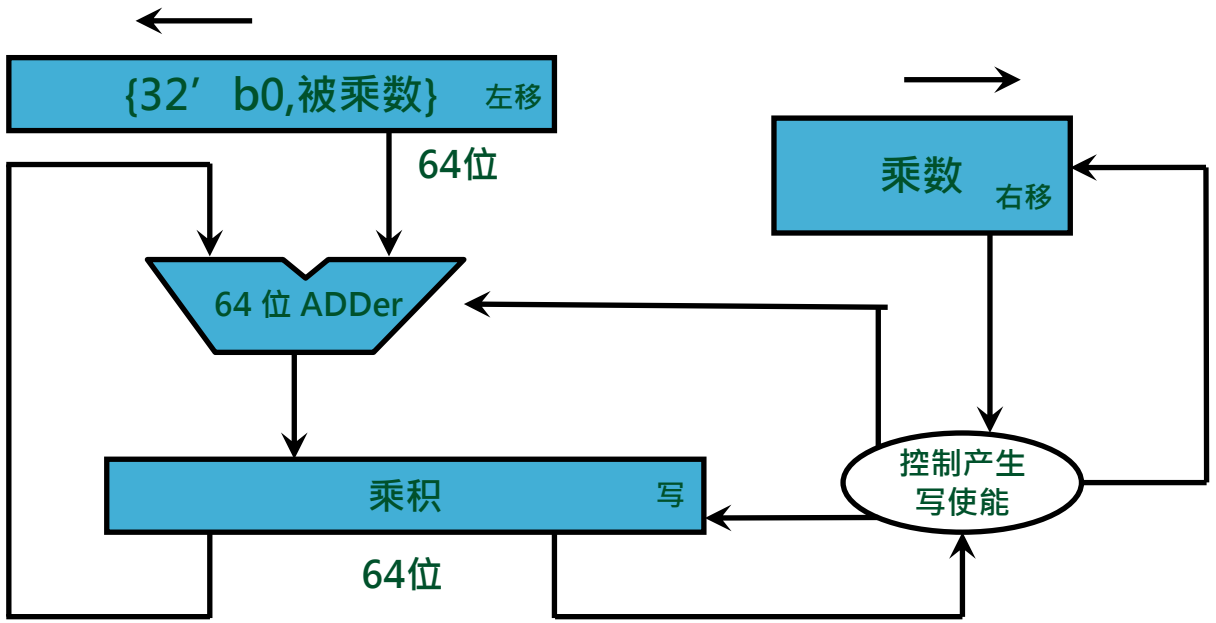






无符号乘法器

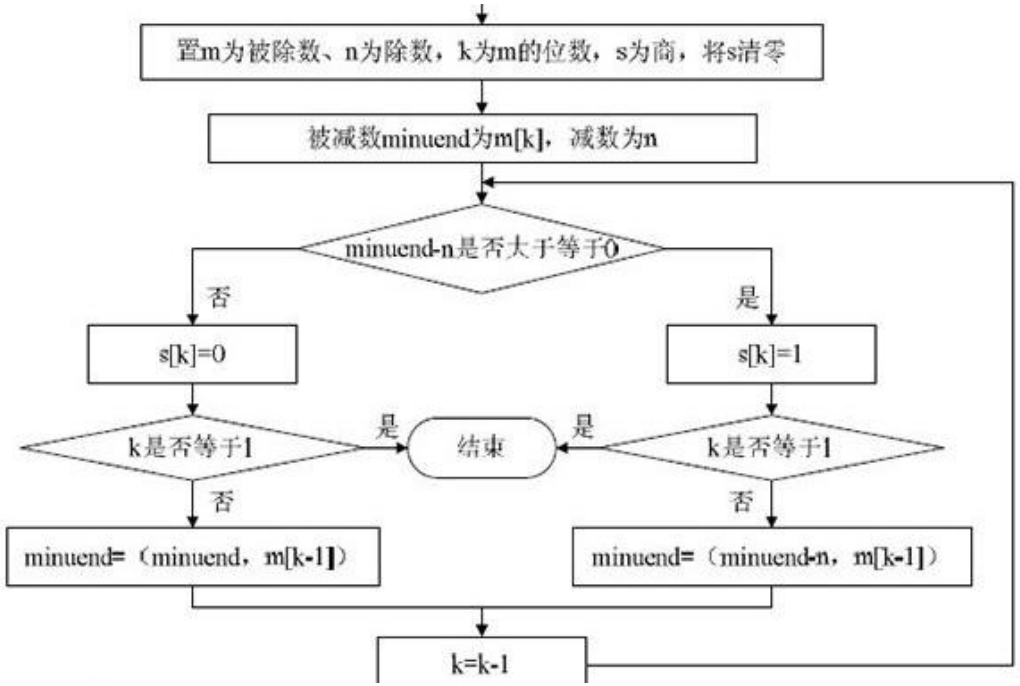
```
        1 0 0 0   被乘数 Multiplicand
  ×     1 0 0 1   乘数 Multiplier
-------------------
        1 0 0 0
      0 0 0 0     ──→ 乘数对应位为0，乘积不变
    0 0 0 0
  1 0 0 0         ──→ 乘数第n位为1，乘积+=被乘数<<n
-------------------
  1 0 0 1 0 0 0   乘积 Product
```




```
myCPU > multiplier.v
1  `timescale 1ns / 1ps
2
3  module multiplier(
4      input clk,
5      input rst,
6      input clear,
7      input stall,
8
9      input mul_en, //乘法使能
10
11     input [31:0] _multiplier, //乘数
12     input [31:0] _multiplicand, //被乘数
13
14     output [63:0] mul_ans, //乘法结果
15     output calculating //运算中
16     );
17
18     reg [4:0] cnt;//计数运算次数，cnt由0变为1则进行第一次运算，32位共需运算32次
19     always @(posedge clk or negedge rst) begin
20       if(!rst||clear) cnt<=0;
21       else if(mul_en==0&&!stall) cnt<=0;
22       else if(cnt==5'b11111&&!stall) cnt<=0;
23       else if(!stall) cnt<=cnt+1;
24     end
25     assign calculating = (cnt==0)? 0: 1; // cnt = 0时还未开始计算，cnt=1->cnt=31时正在计算中，cnt重新等于0时计算已经完成
26
27     reg [31:0] multiplier;
28     reg [31:0] multiplicand;
29     reg [63:0] ans_tmp;
30     always @(posedge clk or negedge rst) begin
31       if(!rst||clear) begin
32         ans_tmp<=0;
33         multiplicand<=0;
34         multiplier<=0;
35       end
36       else if(mul_en&&!stall) begin
37         if (cnt==0) begin //初始化部分
38           multiplicand <= {32'b0,_multiplicand};
39           multiplier <= _multiplier;
40           if (_multiplier[0]) ans_tmp <= {32'b0,_multiplicand};
41           else ans_tmp <= 64'b0;
42         end
43         else if(multiplier[cnt]) ans_tmp <= ans_tmp+(multiplicand<<cnt);
44       end
45     end
46
47     assign mul_ans = ans_tmp;
48
49 endmodule
50
```

对于有符号的两个操作数可以先取绝对值算出乘积绝对值再根据乘数与被乘数符号定积的符号。

无符号除法器



Divider模块

1. 计数器cnt 每三十二个时钟周期记一轮，从divider被使能开始计数，计数结束后divider完成信号置1，使能信号置0.
2. divisor 寄存器用来维持被除数信号，支持每次迭代。

算法参考

试商法

初始：被减数为被除数最高位；商为零；k为被除数位数（迭代的总次数）

每次迭代：
1. tmp=被减数-除数
2. if(tmp>=0) 商1,新的被减数={tmp,被除数[k-1]};
   else 商0,新的被减数={当前被减数，被除数[k-1]};
3. k=k-1;
4. if(k!=1) 继续迭代
   else 迭代完成

结果：商为商；余数为新的被减数。

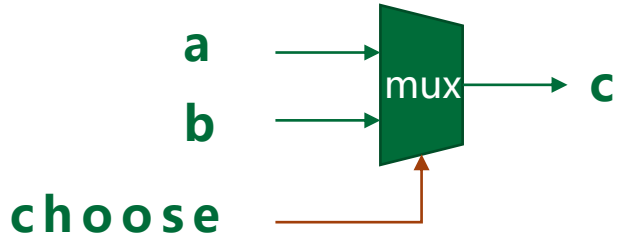
divider

名称	宽度	方向	描述
时钟/复位/控制信号			
clk	1	input	时钟信号，来自 clk_pll 的输出时钟
rst	1	input	复位信号，低电平有效
div_en	1	input	模块使能信号。外部输入 div_en=1 的时钟周期同时获取被除数与除数
数据输入/输出			
dividend	32	input	32 位无符号被除数
_divisor	32	input	32 位无符号除数
calculating	1	output	=1，除法计算中
div_ans	64	output	除法运算结果，前 32 位为余数，后 32 位为商

Verilog运算符

- 可以直接使用Verilog硬件描述语言中的运算符完成算数、逻辑、位移、关系、比较等运算。
- Verilog的运算符在进行设计综合时会自动转化成对应的硬件电路。
- ALU中的多路选择器可以用条件运算符实现：

```
assign c = (choose)? a: b;
```



类 别	运 算 符	优先级
逻辑非、按位取反	! ~	高
算术运算符	* / %	
	+ -	
移位运算符	<< >>	
关系运算符	< <= > >=	
等式运算符	= = ! = === !==	
缩减运算符	& ~&	
	^ ^~	
	| ~|	
逻辑运算符	&&	
	||	低
条件运算符	? :	

一个ALU模块实例


```
1  `timescale 1ns / 1ps
2
3  module alu(
4      input clk,
5      input rst,
6      input [3:0] ctrl,//0-add,1-addiu,2,liu,3-lw,4-sw,5-beq,6-j,7-srav
7
8      input [31:0] alu_num1,
9      input [31:0] alu_num2,
10
11     output [31:0] ans,
12     output error,
13     output [1:0] error_message_,//0-没错，1-add溢出，2-lw/sw地址出错,3-没有这个指令
14     output done_//done=1,程序执行完毕
15     );
16     reg [1:0] error_message;
17     reg done;
18     wire [32:0] e_alu_num1={alu_num1[31],alu_num1};
19     wire [32:0] e_alu_num2={alu_num2[31],alu_num2};
20     wire [32:0] e_add_ans=e_alu_num1+e_alu_num2;
21     wire [31:0] lui_ans={alu_num2[15:0],16'b0};
22
23     wire [31:0] sw_lw_addr=e_add_ans[31:0];
24     always @(posedge clk or negedge rst) begin
25     if(!rst) begin
26     done<=1'b0;
27     error_message<=2'b0;
28     end
29     else if((ctrl==0)&&(e_add_ans[32]!=e_add_ans[31])) error_message<=2'b01;
30     else if((ctrl==3||ctrl==4)&&((sw_lw_addr[0]!=0)||(sw_lw_addr[1]!=0))) error_message<=2'b10;
31     else if(ctrl==8) error_message<=2'b11;
32     else if(ctrl==4'b1111) done<=1'b1;
33     end
34
35     assign error= (error_message==0)? 0:1;
36     assign done_=done;
37     assign error_message_=error_message;
38
39     assign ans= (ctrl==0||ctrl==1)? e_add_ans[31:0]:
40                 (ctrl==2)? lui_ans:
41                 (ctrl==3||ctrl==4)? sw_lw_addr:
42                 (ctrl==7)? srav_ans:32'b0;
43
44 endmodule
45
```

时钟信号、复位信号
以及ALU控制信号

操作数输入

运算结果输出

异常信息输出：加法溢出；地址错误（未对齐）；保留指令

中间信号、寄存器定义

加法运算举例，其它运算类似

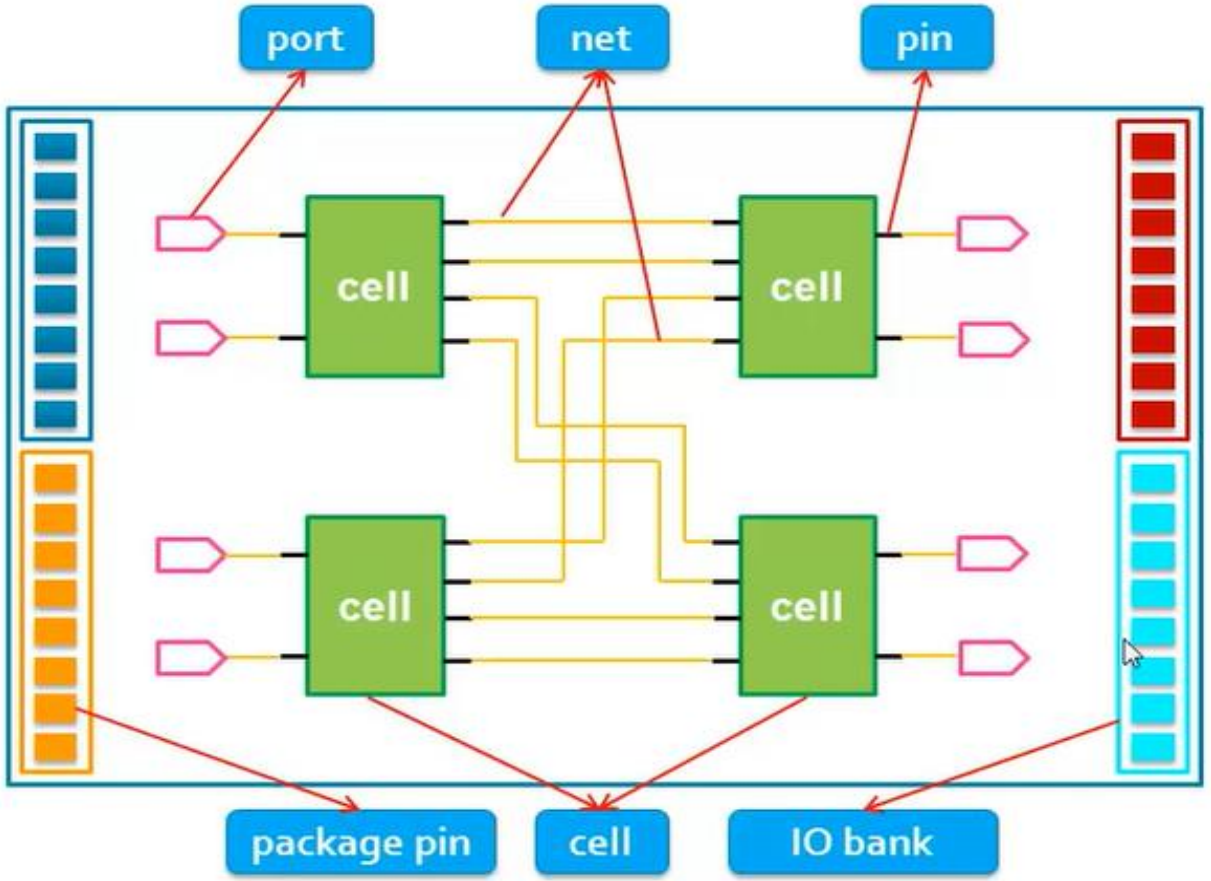
异常检测与异常判断逻辑

ALU结果输出多路复用

CPU分类

单周期CPU	多周期CPU	流水线CPU	超标量流水线
所有指令的指令周期一致，CPU用相同的时钟周期完成所有指令。	根据指令的不同区分不同阶段，CPU可能用不同的时钟周期完成不同的指令。	这多条指令可以同时存在于CPU内流水线的不同阶段，提高吞吐率和频率。	CPU内多条指令流水线同时并行工作。

集成设计环境Vivado



- 每个模块叫cell
- 模块间连接线路叫net
- 每个模块的输入输出端口叫pin
- 整个设计与fpga打交道的接口叫port
- 每个port通过约束连接fpga的管脚（package pin）
- 管脚会被fpga封装到不同的IO bank之内

Vivado 使用技巧：https://github.com/bit-mips/bitmips_experiments_doc/blob/master/others/vivado_skill.md

IP 核简介：https://github.com/bit-mips/bitmips_experiments_doc/blob/master/others/ip.md

指令生成：https://github.com/bit-mips/bitmips_experiments_doc/blob/master/others/cross_compiler.md#%E5%85%B3%E4%BA%8E%E6%8C%87%E4%BB%A4%E7%94%9F%E6%88%90

面向硬件电路的Verilog设计思路

1. CPU本质上是一个数字逻辑电路，所以电路是设计的对象，Verilog只是描述电路的一个工具而已。
2. Verilog语言的很多语法要素与C语言很像，但要注意摒弃这种串行的过程化的思维。
3. 先进行电路结构（结构框图<空间维度>、状态机<时间维度>）设计，再进行Verilog代码编写。包括各个模块和模块之间的连接、模块内部的数据通路和状态机、数据通路中的电路逻辑以及状态机中的状态转换图。
4. 如果要上板，同学们在进行电路设计时只能用到Verilog语言的可综合子集，同时还要注意约束管脚。

Vivado工具仿真

1. 综合和实现是需要花时间的，而且板上也缺乏相应的调试环境。所以在正式上板之前，需要自行编写testbench仿真文件来验证和调试设计的正确性。
2. 同学们可以自行学习一些Verilog系统函数$readmemh、$fscanf、$display、$finish等来帮助更好地完成仿真。
3. 一些经验：

   仿真波形跑到一半不动了：检查设计中是否存在逻辑回环。

   出现x不确定的信号值：检查是否多驱动，是否有浮空的连线。

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

感谢各位

主讲人：蔡建
计算机学院

德以明理　学以精工