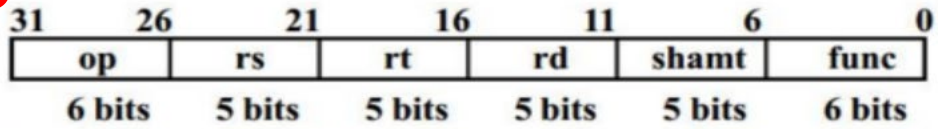

# 硬件综合设计注意事项

# 目录

# 目录

# 1、MIPS指令集

## 1. R型指令




OP：操作码，R型指令的OP字段都是“000000”，由功能码的值定义不同的运算操作！

func：功能码，能给出不同的操作。

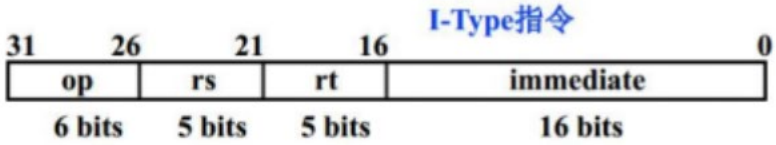
例如：当 op = “000000”，func=“100000”，即32时，表示“加法”运算。

当 op = “000000”，func=“100010”，即34 时，表示“减法”运算。

rs：第一个源操作数所保存的寄存器名称

rt：第二个源操作数所在的寄存器名称

## 2. I型指令



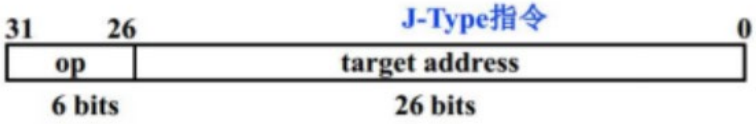

OP：操作码

rs：第一个源操作数寄存器

rt：第二个源操作数寄存器

immediate：立即数或load/store指令和分支指令的偏移地

## 3. J型指令




OP：操作码

target address：无条件转移地址的低26位。将PC高4位（放在高4位）拼上26位直接地址，最后添2个“0”就是32位目标地址。

# 1、MIPS指令集

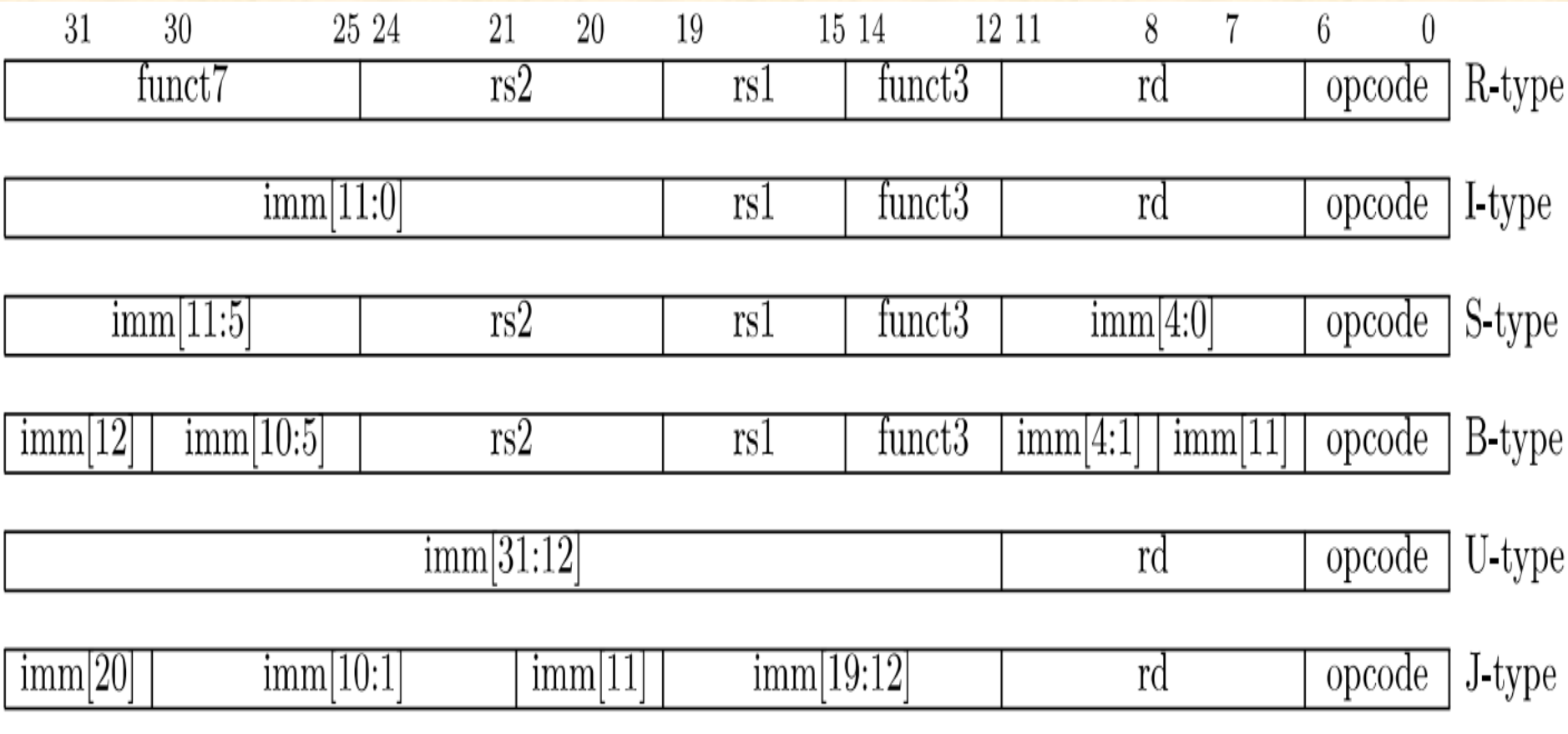
| MIPS 指令集(共 31 条） | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 助 记符 | 指令格式 | | | | | | 示例 | 示例含义 | 操作及其解释 |
| Bit # | 31..26 | 25..21 | 20..16 | 15..11 | 10..6 | 5..0 | | | |
| R-type | op | rs | rt | rd | shamt | func | | | |
| add | 000000 | rs | rt | rd | 00000 | 100000 | add $1,$2,$3 | $1=$2+$3 | rd <- rs + rt ；其中 rs＝$2，rt=$3, rd=$1 |
| addu | 000000 | rs | rt | rd | 00000 | 100001 | addu $1,$2,$3 | $1=$2+$3 | rd <- rs + rt ；其中 rs＝$2，rt=$3, rd=$1,无符号数 |
| sub | 000000 | rs | rt | rd | 00000 | 100010 | sub $1,$2,$3 | $1=$2-$3 | rd <- rs - rt ；其中 rs＝$2，rt=$3, rd=$1 |
| subu | 000000 | rs | rt | rd | 00000 | 100011 | subu $1,$2,$3 | $1=$2-$3 | rd <- rs - rt ；其中 rs＝$2，rt=$3, rd=$1,无符号数 |
| and | 000000 | rs | rt | rd | 00000 | 100100 | and $1,$2,$3 | $1=$2 & $3 | rd <- rs & rt ；其中 rs＝$2，rt=$3, rd=$1 |
| or | 000000 | rs | rt | rd | 00000 | 100101 | or $1,$2,$3 | $1=$2 \| $3 | rd <- rs \| rt ；其中 rs＝$2，rt=$3, rd=$1 |
| xor | 000000 | rs | rt | rd | 00000 | 100110 | xor $1,$2,$3 | $1=$2 ^ $3 | rd <- rs xor rt ；其中 rs＝$2，rt=$3, rd=$1(异或） |
| nor | 000000 | rs | rt | rd | 00000 | 100111 | nor $1,$2,$3 | $1=~($2 \| $3) | rd <- not(rs \| rt) ；其中 rs＝$2，rt=$3, rd=$1(或非） |

# 1、MIPS指令集

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| I-type | op | rs | rt | immediate | | | |
| addi | 001000 | rs | rt | immediate | addi $1,$2,100 | $1=$2+100 | rt <- rs + (sign-extend)immediate ; 其中 rt=$1,rs=$2 |
| addiu | 001001 | rs | rt | immediate | addiu $1,$2,100 | $1=$2+100 | rt <- rs + (zero-extend)immediate ; 其中 rt=$1,rs=$2 |
| andi | 001100 | rs | rt | immediate | andi $1,$2,10 | $1=$2 & 10 | rt <- rs & (zero-extend)immediate ; 其中 rt=$1,rs=$2 |
| ori | 001101 | rs | rt | immediate | andi $1,$2,10 | $1=$2 \| 10 | rt <- rs \| (zero-extend)immediate ; 其中 rt=$1,rs=$2 |
| xori | 001110 | rs | rt | immediate | andi $1,$2,10 | $1=$2 ^ 10 | rt <- rs xor (zero-extend)immediate ; 其中 rt=$1,rs=$2 |
| lui | 001111 | 00000 | rt | immediate | lui $1,100 | $1=100*65536 | rt <- immediate*65536 ; 将 16 位立即数放到目标寄存器高 16 位，目标寄存器的低 16 位填 0 |

| | | | | | |
|---|---|---|---|---|---|
| J-type | op | address | | | |
| j | 000010 | address | j 10000 | goto 10000 | PC <- (PC+4)[31..28],address,0,0 ; address=10000/4 |
| jal | 000011 | address | jal 10000 | $31<-PC+4; goto 10000 | $31<-PC+4 ; PC <- (PC+4)[31..28],address,0,0 ; address=10000/4 |

# 2、RISC V指令集




RISC-V 指令格式

## 32-bit RISC-V instruction formats

| Format | Bit | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Register/register** | funct7 | | | | | | | rs2 | | | | | rs1 | | | | | funct3 | | | rd | | | | | opcode | | | | | | |
| **Immediate** | imm[11:0] | | | | | | | | | | | | rs1 | | | | | funct3 | | | rd | | | | | opcode | | | | | | |
| **Upper immediate** | imm[31:12] | | | | | | | | | | | | | | | | | | | | rd | | | | | opcode | | | | | | |
| **Store** | imm[11:5] | | | | | | | rs2 | | | | | rs1 | | | | | funct3 | | | imm[4:0] | | | | | opcode | | | | | | |
| **Branch** | [12] | imm[10:5] | | | | | | rs2 | | | | | rs1 | | | | | funct3 | | | imm[4:1] | | | | [11] | opcode | | | | | | |
| **Jump** | [20] | imm[10:1] | | | | | | | | | | [11] | imm[19:12] | | | | | | | | rd | | | | | opcode | | | | | | |

- **opcode (7 bits):** Partially specifies which of the 6 types of *instruction formats*.
- **funct7, and funct3 (10 bits):** These two fields, further than the opcode field, specify the operation to be performed.
- **rs1 (5 bits):** Specifies, by index, the register containing first operand (i.e., source register).
- **rs2 (5 bits):** Specifies the second operand register.
- **rd (5 bits):** Specifies the destination register to which the computation result will be directed.

# 2、RISC V指令集

| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 | |
|---|---|---|---|---|---|---|
| imm[31:12] | | | | rd | 0110111 | U lui |
| imm[31:12] | | | | rd | 0010111 | U auipc |
| imm[20\|10:1\|11\|19:12] | | | | rd | 1101111 | J jal |
| imm[11:0] | | rs1 | 000 | rd | 1100111 | I jalr |
| imm[12\|10:5] | rs2 | rs1 | 000 | imm[4:1\|11] | 1100011 | B beq |
| imm[12\|10:5] | rs2 | rs1 | 001 | imm[4:1\|11] | 1100011 | B bne |
| imm[12\|10:5] | rs2 | rs1 | 100 | imm[4:1\|11] | 1100011 | B blt |
| imm[12\|10:5] | rs2 | rs1 | 101 | imm[4:1\|11] | 1100011 | B bge |
| imm[12\|10:5] | rs2 | rs1 | 110 | imm[4:1\|11] | 1100011 | B bltu |
| imm[12\|10:5] | rs2 | rs1 | 111 | imm[4:1\|11] | 1100011 | B bgeu |
| imm[11:0] | | rs1 | 000 | rd | 0000011 | I lb |
| imm[11:0] | | rs1 | 001 | rd | 0000011 | I lh |
| imm[11:0] | | rs1 | 010 | rd | 0000011 | I lw |
| imm[11:0] | | rs1 | 100 | rd | 0000011 | I lbu |
| imm[11:0] | | rs1 | 101 | rd | 0000011 | I lhu |
| imm[11:5] | rs2 | rs1 | 000 | imm[4:0] | 0100011 | S sb |
| imm[11:5] | rs2 | rs1 | 001 | imm[4:0] | 0100011 | S sh |

**RISC-V 指令格式**

# 2、RISC V指令集

| | | | | | | |
|---|---|---|---|---|---|---|
| imm[11:5] | rs2 | rs1 | 010 | imm[4:0] | 0100011 | S sw |
| imm[11:0] | | rs1 | 000 | rd | 0010011 | I addi |
| imm[11:0] | | rs1 | 010 | rd | 0010011 | I slti |
| imm[11:0] | | rs1 | 011 | rd | 0010011 | I sltiu |
| imm[11:0] | | rs1 | 100 | rd | 0010011 | I xori |
| imm[11:0] | | rs1 | 110 | rd | 0010011 | I ori |
| imm[11:0] | | rs1 | 111 | rd | 0010011 | I andi |
| 0000000 | shamt | rs1 | 001 | rd | 0010011 | I slli |
| 0000000 | shamt | rs1 | 101 | rd | 0010011 | I srli |
| 0100000 | shamt | rs1 | 101 | rd | 0010011 | I srai |
| 0000000 | rs2 | rs1 | 000 | rd | 0110011 | R add |
| 0100000 | rs2 | rs1 | 000 | rd | 0110011 | R sub |
| 0000000 | rs2 | rs1 | 001 | rd | 0110011 | R sll |
| 0000000 | rs2 | rs1 | 010 | rd | 0110011 | R slt |
| 0000000 | rs2 | rs1 | 011 | rd | 0110011 | R sltu |
| 0000000 | rs2 | rs1 | 100 | rd | 0110011 | R xor |
| 0000000 | rs2 | rs1 | 101 | rd | 0110011 | R srl |

**RISC-V 指令格式**

# 2、RISC V指令集

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0100000 | | rs2 | rs1 | 101 | rd | 0110011 | R sra |
| 0000000 | | rs2 | rs1 | 110 | rd | 0110011 | R or |
| 0000000 | | rs2 | rs1 | 111 | rd | 0110011 | R and |
| 0000 | pred | succ | 00000 | 000 | 00000 | 0001111 | I fence |
| 0000 | 0000 | 0000 | 00000 | 001 | 00000 | 0001111 | I fence.i |
| 000000000000 | | | 00000 | 000 | 00000 | 1110011 | I ecall |
| 000000000001 | | | 00000 | 000 | 00000 | 1110011 | I ebreak |
| csr | | | rs1 | 001 | rd | 1110011 | I csrrw |
| csr | | | rs1 | 010 | rd | 1110011 | I csrrs |
| csr | | | rs1 | 011 | rd | 1110011 | I csrrc |
| csr | | | zimm | 101 | rd | 1110011 | I csrrwi |
| csr | | | zimm | 110 | rd | 1110011 | I csrrsi |
| csr | | | zimm | 111 | rd | 1110011 | I csrrci |

**RISC-V 指令格式**

# 3、ARM指令集

## ARM 指令集

### 指令格式

基本格式

<opcode>{<cond>}{S}　<Rd>,<Rn>{,<opcode2>}

其中，<>内的项是必须的，{}内的项是可选的，如<opcode>是指令助记符，是必须的，而{<cond>}为指令执行条件，是可选的，如果不写则使用默认条件 AL(无条件执行)。

| | |
|---|---|
| opcode | 指令助记符，如 LDR，STR 等 |
| cond | 执行条件，如 EQ，NE 等 |
| S | 是否影响 CPSR 寄存器的值，书写时影响 CPSR，否则不影响 |
| Rd | 目标寄存器 |
| Rn | 第一个操作数的寄存器 |
| operand2 | 第二个操作数 |

# 3、ARM指令集

## ARM架构指令集汇总

| | 31 30 29 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MRS | | 0 | 0 | 0 | 1 | 0 | S | 0 | 0 | 1 | 1 | 1 | 1 | Rd | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MSR0 | | 0 | 0 | 0 | 1 | 0 | S | 1 | 0 | a | 0 | 0 | a | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Rm | | | |
| dp0 | | 0 | 0 | 0 | OpCode | | | | S | Rn | | | | Rd | | | | | | | | | | | 0 | Rm | | | |
| BX | | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | Rm | | | |
| dp1 | | 0 | 0 | 0 | OpCode | | | | S | Rn | | | | Rd | | | | Rs | | | | 0 | | | 1 | Rm | | | |
| MULT | | 0 | 0 | 0 | 0 | 0 | 0 | A | S | Rd | | | | Rn | | | | Rs | | | | 1 | 0 | 0 | 1 | Rm | | | |
| MULTL | | 0 | 0 | 0 | 0 | 1 | U | A | S | Rdhi | | | | Rdlo | | | | Rs | | | | 1 | 0 | 0 | 1 | Rm | | | |
| SWP | | 0 | 0 | 0 | 1 | 0 | B | 0 | 0 | Rn | | | | Rd | | | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | Rm | | | |
| LDRH0 | | 0 | 0 | 0 | P | U | 0 | W | L | Rn | | | | Rd | | | | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | Rm | | | |
| LDRH1 | | 0 | 0 | 0 | P | U | 1 | W | L | Rn | | | | Rd | | | | | | | | 1 | 0 | 1 | 1 | | | | |
| LDRSB0 | | 0 | 0 | 0 | P | U | 0 | W | 1 | Rn | | | | Rd | | | | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | Rm | | | |
| LDRSB1 | | 0 | 0 | 0 | P | U | 1 | W | 1 | Rn | | | | Rd | | | | | | | | 1 | 1 | 0 | 1 | | | | |
| LDRSH0 | | 0 | 0 | 0 | P | U | 0 | W | 1 | Rn | | | | Rd | | | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | Rm | | | |
| LDRSH1 | | 0 | 0 | 0 | P | U | 1 | W | 1 | Rn | | | | Rd | | | | | | | | 1 | 1 | 1 | 1 | | | | |
| MSR1 | | 0 | 0 | 1 | 1 | 0 | S | 1 | 0 | a | 0 | 0 | a | 1 | 1 | 1 | 1 | | | | | | | | | | | | |
| dp2 | | 0 | 0 | 1 | OpCode | | | | S | Rn | | | | Rd | | | | | | | | | | | | | | | |
| LDR0 | | 0 | 1 | 0 | P | U | B | W | L | Rn | | | | Rd | | | | | | | | | | | | | | | |
| LDR1 | | 0 | 1 | 1 | P | U | B | W | L | Rn | | | | Rd | | | | | | | | | | | 0 | Rm | | | |
| LDM | | 1 | 0 | 0 | P | U | S | W | L | Rn | | | | | | | | | | | | | | | | | | | |
| B | | 1 | 0 | 1 | L | | | | | | | | | | | | | | | | | | | | | | | | |
| SWI | COND | 1 | 1 | 1 | 1 | | | | | | | | | | | | | | | | | | | | | | | | |

# 3、ARM指令集

## 条件执行

**COND条件码**

| [31:28] | 汇编语言缩写 | 含义 | 条件码状态 |
|---|---|---|---|
| 0000 | EQ | 等于 | cpsr_z == 1'b1 |
| 0001 | NE | 不等于 | cpsr_z == 1'b0 |
| 0010 | CS/HS | 进位标志置位/无符号大于或等于 | cpsr_c== 1'b1 |
| 0011 | CC/LO | 进位标志清零/无符号小于 | cpsr_c==1'b0 |
| 0100 | MI | 小于/负值 | cpsr_n==1'b1 |
| 0101 | PL | 大于/正值或零 | cpsr_n==1'b0 |
| 0110 | VS | 溢出 | cpsr_v==1'b1 |
| 0111 | VC | 无溢出 | cpsr_v==1'b0 |
| 1000 | HI | 无符号大于 | ( cpsr_c==1'b1 )&(cpsr_z==1'b0) |
| 1001 | LS | 无符号小于或等于 | ( cpsr_c==1'b0 )\|(cpsr_z==1'b1) |
| 1010 | GE | 有符号大于或等于 | ( cpsr_n==cpsr_v) |
| 1011 | LT | 有符号小于 | ( cpsr_n!=cpsr_v) |
| 1100 | GT | 有符号大于 | ( cpsr_z==1'b0)&(cpsr_n==cpsr_v) |
| 1101 | LE | 有符号小于或等于 | ( cpsr_z==1'b1)\|(cpsr_n!=cpsr_v) |
| 1110 | | 无条件执行 | 1'b1 |
| 1111 | NV | 无定义 | 1'b0 |

# 3、ARM指令集

## DPO指令——处理方式表（Data Process）

| OpCode | 指令 | 操作方式 |
|---|---|---|
| 0000 | AND | Rd = Rn & sec_operand |
| 0001 | EOR | Rd = Rn ^ sec_operand |
| 0010 | SUB | Rd = Rn – sec_operand |
| 0011 | RSB | Rd = sec_operand – Rn |
| 0100 | ADD | Rd = Rn + sec_operand |
| 0101 | ADC | Rd = Rn + sec_operand + cpsr_c |
| 0110 | SBC | Rd = Rn – sec_operand + cpsr_c – 1 |
| 0111 | RSC | Rd = sec_operand – Rn + cpsr_c – 1 |
| 1000 | TST | 进行Rn & sec_operand操作，结果不写入Rd |
| 1001 | TEQ | 进行Rn ^ sec_operand操作，结果不写入Rd |
| 1010 | CMP | 进行Rn - sec_operand操作，结果不写入Rd |
| 1011 | CMN | 进行Rn + sec_operand操作，结果不写入Rd |
| 1100 | ORR | Rd = Rn \| sec_operand |
| 1101 | MOV | Rd = sec_operand |
| 1110 | BIC | Rd = Rn & ~sec_operand |
| 1111 | MVN | Rd = ~sec_operand |

# 3、ARM指令集

## DP1指令——概述

| 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | OpCode | | | | S | Rn | | | | Rd | | | | Rs | | | | 0 | rot | | 1 | Rm | | | |

## DP2指令——概述

| 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | OpCode | | | | S | Rn | | | | Rd | | | | num | | | | data | | | | | | | |

## MULT指令——概述

| 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | A | S | Rd | | | | Rn | | | | Rs | | | | 1 | 0 | 0 | 1 | Rm | | | |

## MULTL指令——概述

| 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | U | A | S | Rdhi | | | | Rdlo | | | | Rs | | | | 1 | 0 | 0 | 1 | Rm | | | |

# 3、ARM指令集

## MRS指令——概述

| 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | S | 0 | 0 | 1 | 1 | 1 | 1 | Rd | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## B指令——概述

| 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | L | offset | | | | | | | | | | | | | | | | | | | | | | | |

## BX指令——概述

| 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | Rm | | | |

## SWI指令——概述

| 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | label | | | | | | | | | | | | | | | | | | | | | | | |

# 3种指令集的比较

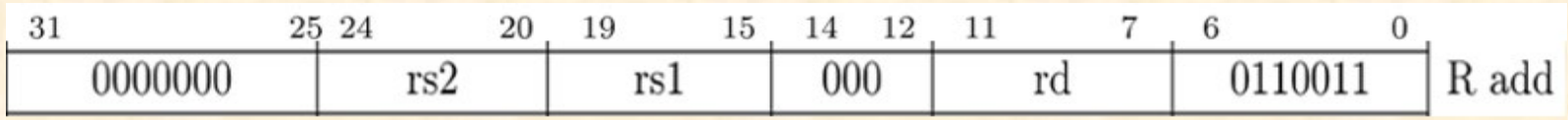
## ADD指令在3种指令集中的机器码

### ADD指令在MIPS指令集中

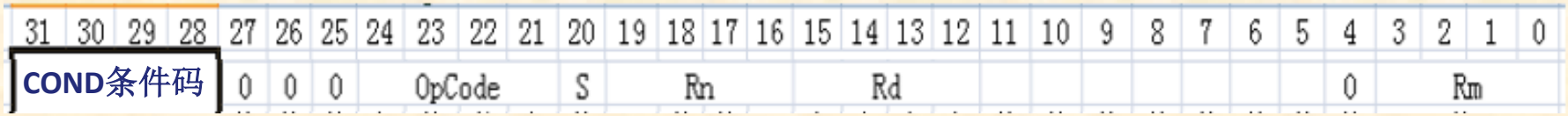
shift　　function

| 31 | …… | 26 | 25 | … | 21 | 20 | … | 16 | 15 | … | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000000 | | | Rs | | | Rt | | | Rd | | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

### ADD指令在RISC-V指令集中




### ADD指令在ARM指令集中



# 硬件综合课程设计注意事项

一、三种不同的指令集

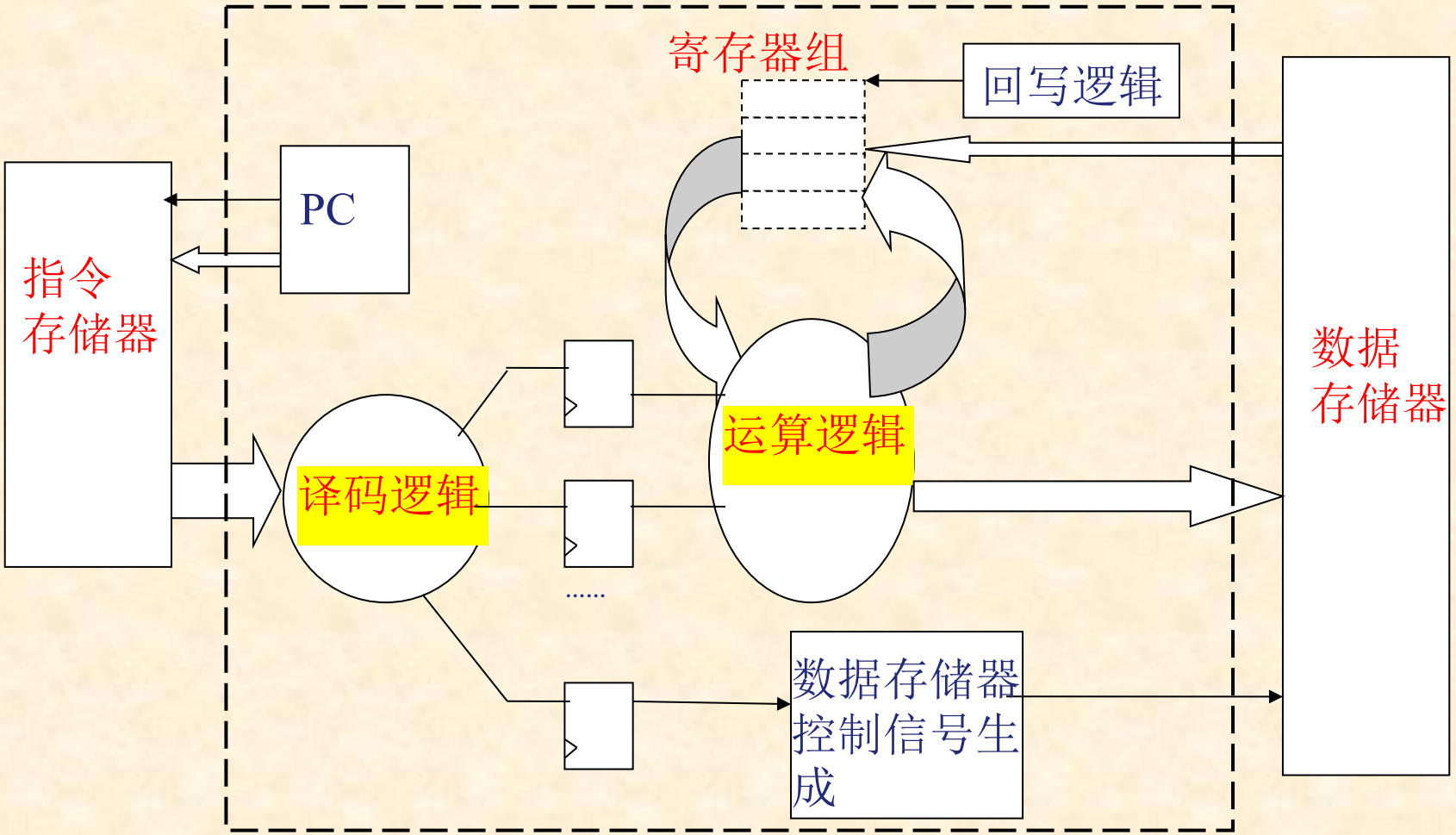
二、CPU的硬件结构与verilog模块

三、单周期CPU结构与多周期流水线结构

四、典型模块结构，verilog代码分析

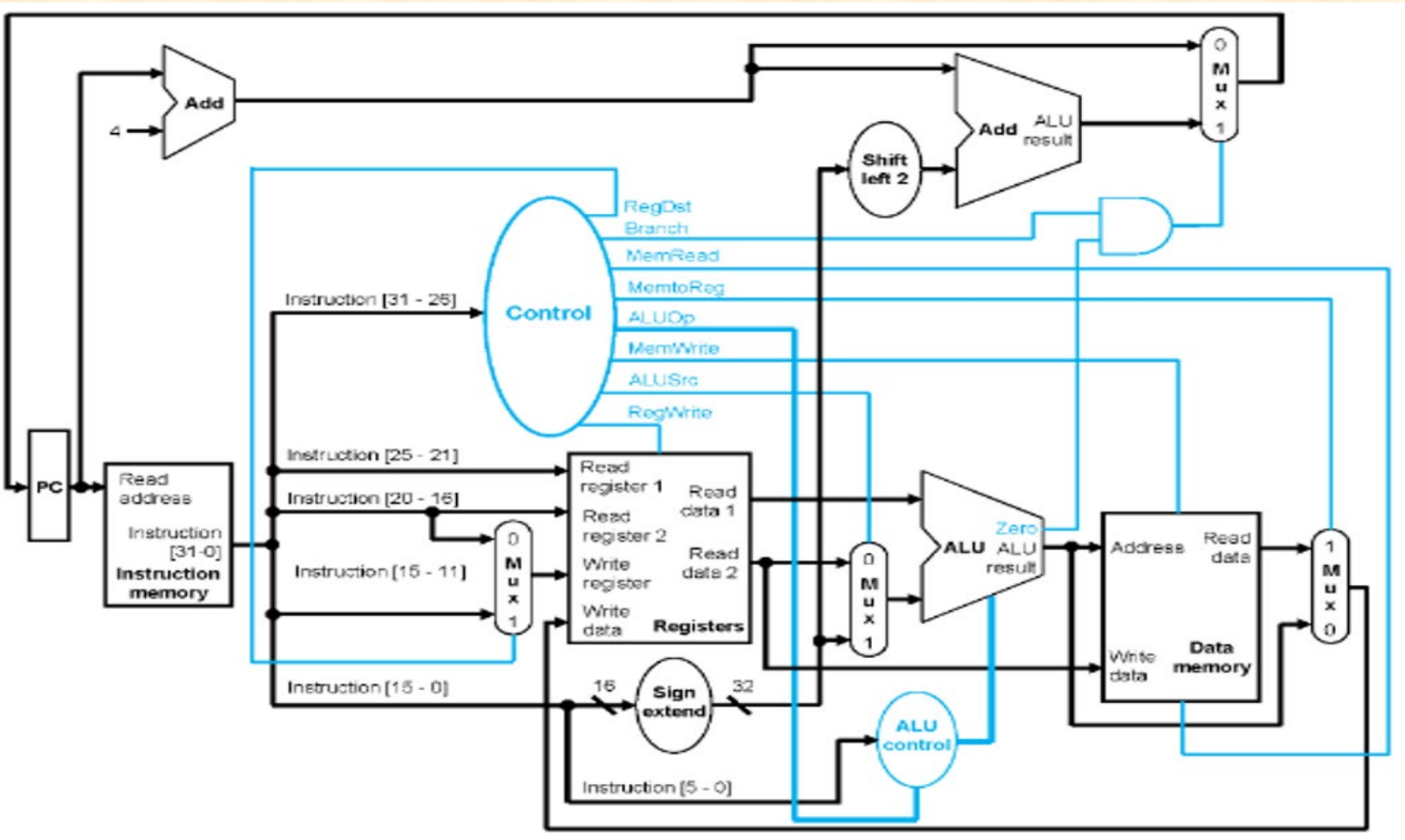
# 1、CPU的硬件组成

## CPU的架构



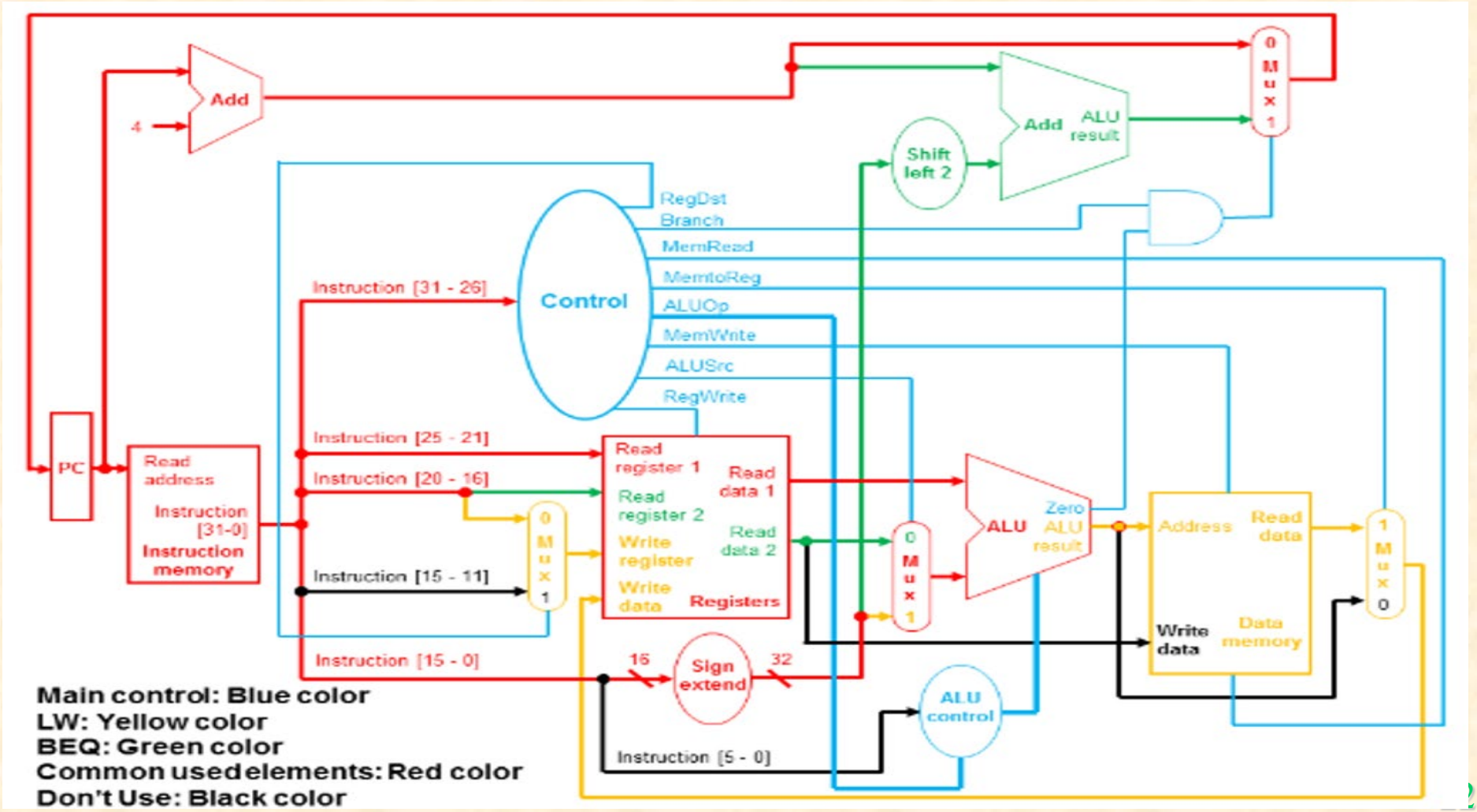
# 1、CPU的硬件组成

## 细化后的单周期CPU硬件结构



# 1、CPU的硬件组成

## 细化后的单周期CPU硬件结构



# 2、CPU的每个硬件模块功能和代码

## 译码控制模块最复杂

译码控制模块主要功能有：

1、根据指令的操作码，生成运算器进行相应的加、减、与、或、直送等运算所需的控制信号。

2、根据指令的地址码，控制寄存器操作数的取值、立即数的符号扩展。

3、根据指令的地址码，发出对存储器进行读、写的控制信号。

## 其次是运算模块

运算模块主要功能有：

1、根据译码模块送来的控制信号，进行相应的加、减、与、或、直送等运算。

2、根据运算结果，设置标志位（零、符号、溢出、进位）。

3、运算结果送给对应的临时寄存器。

# 3、几个重要模块的代码分析

## MIPS指令集中，译码控制模块代码

```
case (op)
  6'b000000:  begin
      case (op2)
          5'b00000:   begin
              case (op3)
                  `EXE_OR:  begin        //or指令
                      wreg_o <= `WriteEnable;     aluop_o <= `EXE_OR_OP;
                      alusel_o <= `EXE_RES_LOGIC;     reg1_read_o <= 1'b1;     reg2_read_
                      instvalid <= `InstValid;
                      end
                  `EXE_AND:   begin        //and指令
                      wreg_o <= `WriteEnable;     aluop_o <= `EXE_AND_OP;
                      alusel_o <= `EXE_RES_LOGIC;   reg1_read_o <= 1'b1;  reg2_read_o <=
                      instvalid <= `InstValid;
                      end
                  `EXE_XOR:   begin        //xor指令
                      wreg_o <= `WriteEnable;     aluop_o <= `EXE_XOR_OP;
                      alusel_o <= `EXE_RES_LOGIC;     reg1_read_o <= 1'b1;     reg2_read_
                      instvalid <= `InstValid;
```

# 3、几个重要模块的代码分析

## MIPS指令集中，译码控制模块代码

```
case(OpCode)
    //R type
    `INSTR_RTYPE_OP:
    begin
        jump = 0;
        RegDst = 0;
        Branch = 0;
        MemR = 0;
        Mem2R = 0;
        MemW = 0;
        RegW = 1;
        Alusrc = 0;
        ExtOp = `EXT_ZERO;
    case(funct)
        `INSTR_SUB_FUNCT:
        begin
            Aluctrl = `ALUOp_SUB;
        end
```

MIPS

```
assign {RegDstD, RegWriteD, ALUSrcD, BranchD, MemWriteD, ALUControlD, MemToRegD,
        ExtOpD, IsJJalD, IsJrJalrD, CompOpD, IsLbSbD, IsLhShD, IsUnsignedD,
        MdOpD, HiLoWriteD, HiLoD, IsMdD, IsShamtD} = CtrlCode;

always @(*)
begin
  casex(OpD)
    LB:     CtrlCode <= 27'b1_1_01_0_0_0000_1_00_0_0_000_1_0_0_00_0_0_0_0;
    LBU:    CtrlCode <= 27'b1_1_01_0_0_0000_1_00_0_0_000_1_0_1_00_0_0_0_0;
    LH:     CtrlCode <= 27'b1_1_01_0_0_0000_1_00_0_0_000_0_1_0_00_0_0_0_0;
    LHU:    CtrlCode <= 27'b1_1_01_0_0_0000_1_00_0_0_000_0_1_1_00_0_0_0_0;
    LUI:    CtrlCode <= 27'b1_1_01_0_0_0000_0_10_0_0_000_0_0_0_00_0_0_0_0;
    LW:     CtrlCode <= 27'b1_1_01_0_0_0000_1_00_0_0_000_0_0_0_00_0_0_0_0;
    SB:     CtrlCode <= 27'b0_0_01_0_1_0000_0_00_0_0_000_1_0_0_00_0_0_0_0;
    SH:     CtrlCode <= 27'b0_0_01_0_1_0000_0_00_0_0_000_0_1_0_00_0_0_0_0;
    SW:     CtrlCode <= 27'b0_0_01_0_1_0000_0_00_0_0_000_0_0_0_00_0_0_0_0;
    BEQ:    CtrlCode <= 27'b0_0_00_1_0_0000_0_00_0_0_000_0_0_0_00_0_0_0_0;
    BNE:    CtrlCode <= 27'b0_0_00_1_0_0000_0_00_0_0_001_0_0_0_00_0_0_0_0;
    BGTZ:   CtrlCode <= 27'b0_0_00_1_0_0000_0_00_0_0_011_0_0_0_00_0_0_0_0;
    BLEZ:   CtrlCode <= 27'b0_0_00_1_0_0000_0_00_0_0_100_0_0_0_00_0_0_0_0;
    BB:
    begin
        casex(RtD)
            BGEZ:     CtrlCode <= 27'b0_0_00_1_0_0000_0_00_0_0_010_0_0_0_00_0_0_0_0
            BLTZ:     CtrlCode <= 27'b0_0_00_1_0_0000_0_00_0_0_101_0_0_0_00_0_0_0_0
            default: CtrlCode <= 27'b0;
        endcase
    end
    J:      CtrlCode <= 27'b0_0_00_0_0_0000_0_00_1_0_000_0_0_0_00_0_0_0_0;
    JAL:    CtrlCode <= 27'b0_1_00_0_0_0000_0_00_1_0_000_0_0_0_00_0_0_0_0;

    ADDI:   CtrlCode <= 27'b1_1_01_0_0_0001_0_00_0_0_000_0_0_0_00_0_0_0_0;
    ADDIU:  CtrlCode <= 27'b1_1_01_0_0_0000_0_01_0_0_000_0_0_0_00_0_0_0_0;
    ANDI:   CtrlCode <= 27'b1_1_01_0_0_1100_0_01_0_0_000_0_0_0_00_0_0_0_0;
    ORI:    CtrlCode <= 27'b1_1_01_0_0_1101_0_01_0_0_000_0_0_0_00_0_0_0_0;
    XORI:   CtrlCode <= 27'b1_1_01_0_0_1110_0_01_0_0_000_0_0_0_00_0_0_0_0;
    SLTI:   CtrlCode <= 27'b1_1_01_0_0_0101_0_00_0_0_000_0_0_0_00_0_0_0_0;
    SLTIU:  CtrlCode <= 27'b1_1_01_0_0_0100_0_01_0_0_000_0_0_0_00_0_0_0_0;

    RType:
```

# 3、几个重要模块的代码分析

## RISC-V指令集中，译码控制模块代码

```
 wd_o <= inst_i[11:7];                    // 写寄存器地址
reg1_addr_o <= inst_i[19:15];             // 读寄存器 A 地址
reg2_addr_o <= inst_i[24:20];             // 读寄存器 B 地址
// 单值控制信号
 branch_o <= op[6]&op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0];
   // B-type
jump_o <= op[6]&op[5]&~op[4]&op[3]&op[2]&op[1]&op[0];              // J-type
mem_to_reg_o <= ~op[6]&~op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0];  // Load
reg_wr_o <= (~op[6]&op[5]&op[4]&~op[3]&~op[2]&op[1]&op[0])       // R-type
| (~op[6]&~op[5]&op[4]&~op[3]&~op[2]&op[1]&op[0])   // I-type-ALU
| (~op[6]&op[5]&op[4]&~op[3]&op[2]&op[1]&op[0])     // lui
| (~op[6]&~op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0])  // Load
| (op[6]&op[5]&~op[4]&op[3]&op[2]&op[1]&op[0]);     // J-type
mem_wr_o <= ~op[6]&op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0];        // Store
alu_asrc_o <= op[6]&op[5]&~op[4]&op[3]&op[2]&op[1]&op[0];         // J-type
```

# 3、几个重要模块的代码分析

## RISC-V指令集中，译码控制模块代码

```
// 多值控制信号
 alu_bsrc_o[1] <= (~op[6]&~op[5]&op[4]&~op[3]&~op[2]&op[1]&op[0])   // I-type-ALU
| (~op[6]&op[5]&op[4]&~op[3]&op[2]&op[1]&op[0])       // lui                          |
(~op[6]&~op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0])    // Load                         |
(~op[6]&op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0]);    // Store           alu_bsrc_o[0] <=
op[6]&op[5]&~op[4]&op[3]&op[2]&op[1]&op[0];       // J-type                        ext_op_o[2] <=
op[6]&op[5]&~op[4]&op[3]&op[2]&op[1]&op[0];       // J-type           ext_op_o[1] <=
(op[6]&op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0])    // B-type                        |
(~op[6]&op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0]); // Store           ext_op_o[0] <=
(~op[6]&op[5]&op[4]&~op[3]&op[2]&op[1]&op[0])     // lui                           |
(op[6]&op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0]);   // B-type
alu_ctr_o[3] <= (~op[6]&op[5]&op[4]&~op[3]&op[2]&op[1]&op[0])    // lui                          |
(op[6]&op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0]);   // B-type           alu_ctr_o[2] <=
((~op[6]&op[5]&op[4]&~op[3]&~op[2]&op[1]&op[0])  // R-type                        |
(~op[6]&~op[5]&op[4]&~op[3]&~op[2]&op[1]&op[0]))  // I-type-ALU                        & fn[2]
| (~op[6]&op[5]&op[4]&~op[3]&op[2]&op[1]&op[0]);    // lui           alu_ctr_o[1] <=
((~op[6]&op[5]&op[4]&~op[3]&~op[2]&op[1]&op[0])  // R-type                        |
(~op[6]&~op[5]&op[4]&~op[3]&~op[2]&op[1]&op[0]))  // I-type-ALU                        & fn[1]
| (~op[6]&op[5]&op[4]&~op[3]&op[2]&op[1]&op[0]);    // lui             alu_ctr_o[0] <=
((~op[6]&op[5]&op[4]&~op[3]&~op[2]&op[1]&op[0])  // R-type                        |
(~op[6]&~op[5]&op[4]&~op[3]&~op[2]&op[1]&op[0]))  // I-type-ALU                        & fn[0]
| (~op[6]&op[5]&op[4]&~op[3]&op[2]&op[1]&op[0]);    // lui
```

# 3、几个重要模块的代码分析

## 运算模块代码

```
always @(*) begin
 case (OP)
 0:  RESULT <= X + Y; // add
   1:  RESULT <= X - Y; // sub
   2:  RESULT <= X & Y; // and
   3:  RESULT <= X | Y; // or
   4:  RESULT <= X ^ Y; // xor
   5:  RESULT <= X << Y; // shift left logical
   6:  RESULT <= X >> Y; // shift right logical
   7:  RESULT <= X_signed >>> Y; // shift right arithmetic
   8:  RESULT <= X * Y; // mul
   9:  RESULT <= X * Y; // mulh
   10: RESULT <= X / Y; // div
   11: RESULT <= X % Y; // rem
   12: RESULT <= (X_signed < Y_signed ? 1 : 0); // set less than (slt)
   13: RESULT <= (X < Y ? 1 : 0);       // set less than (sltu)
endcase
```

# 硬件综合课程设计注意事项

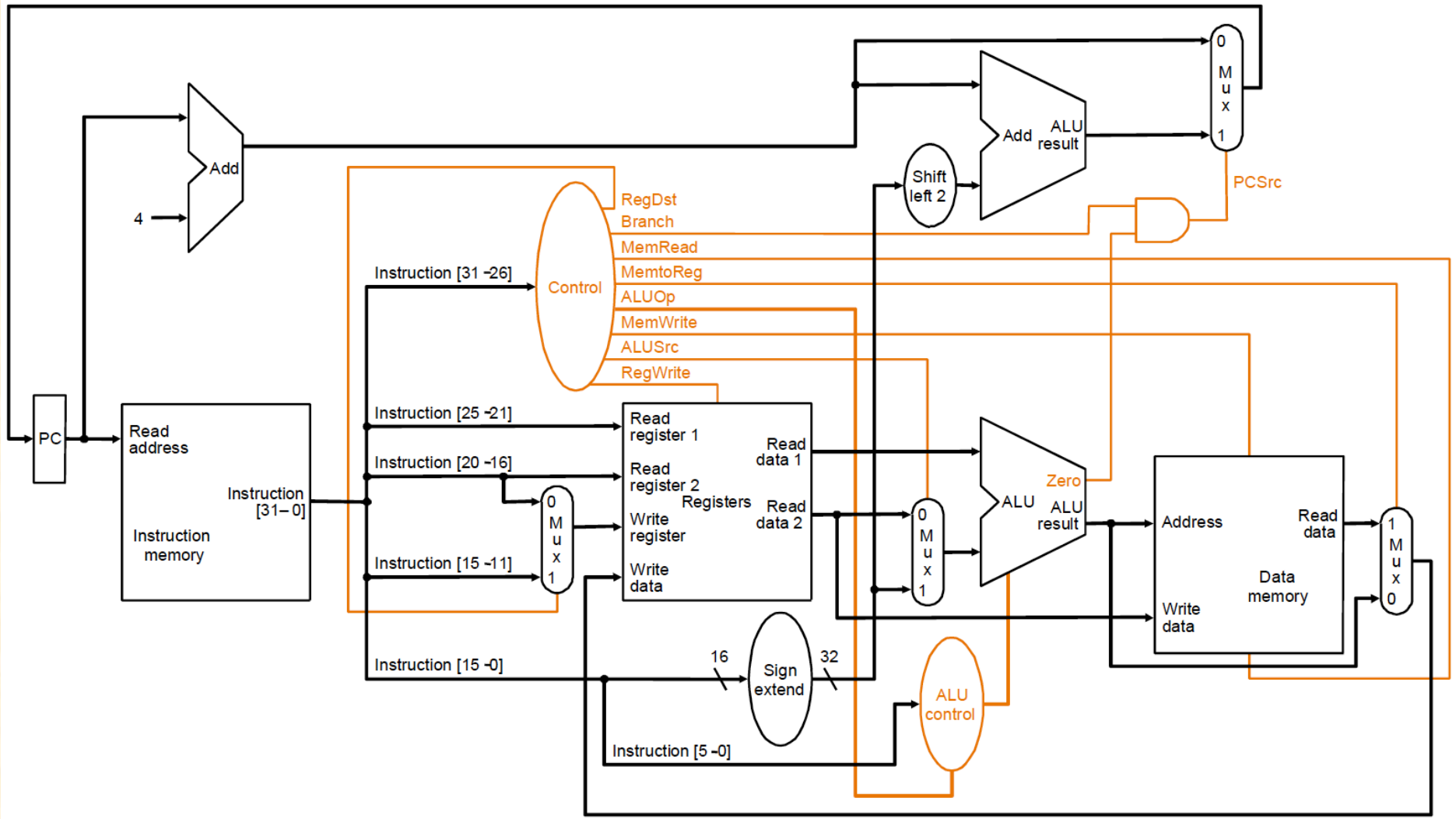
一、三种不同的指令集

二、CPU的硬件结构与verilog模块

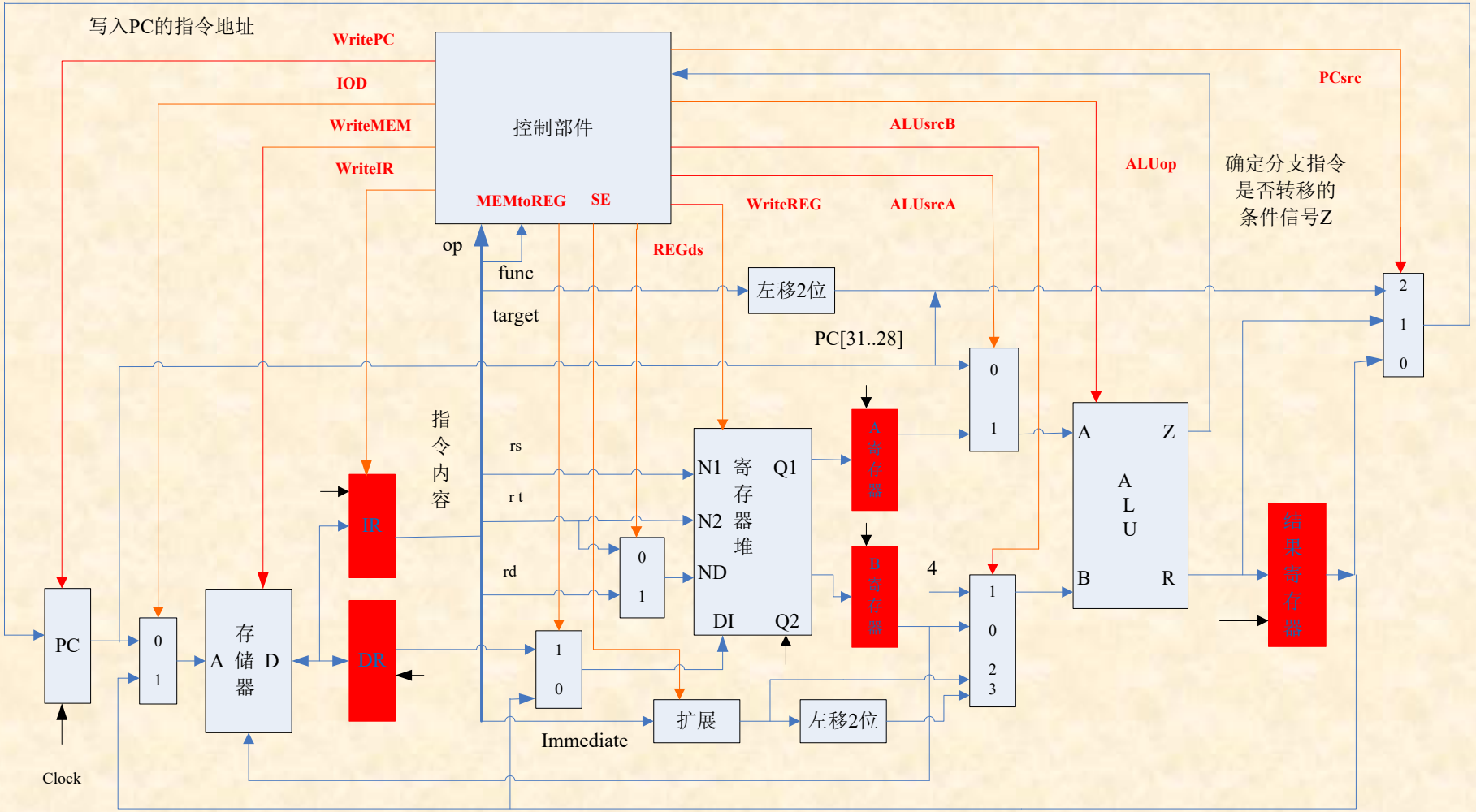
三、单周期CPU结构与多周期流水线结构

四、典型模块结构，verilog代码分析

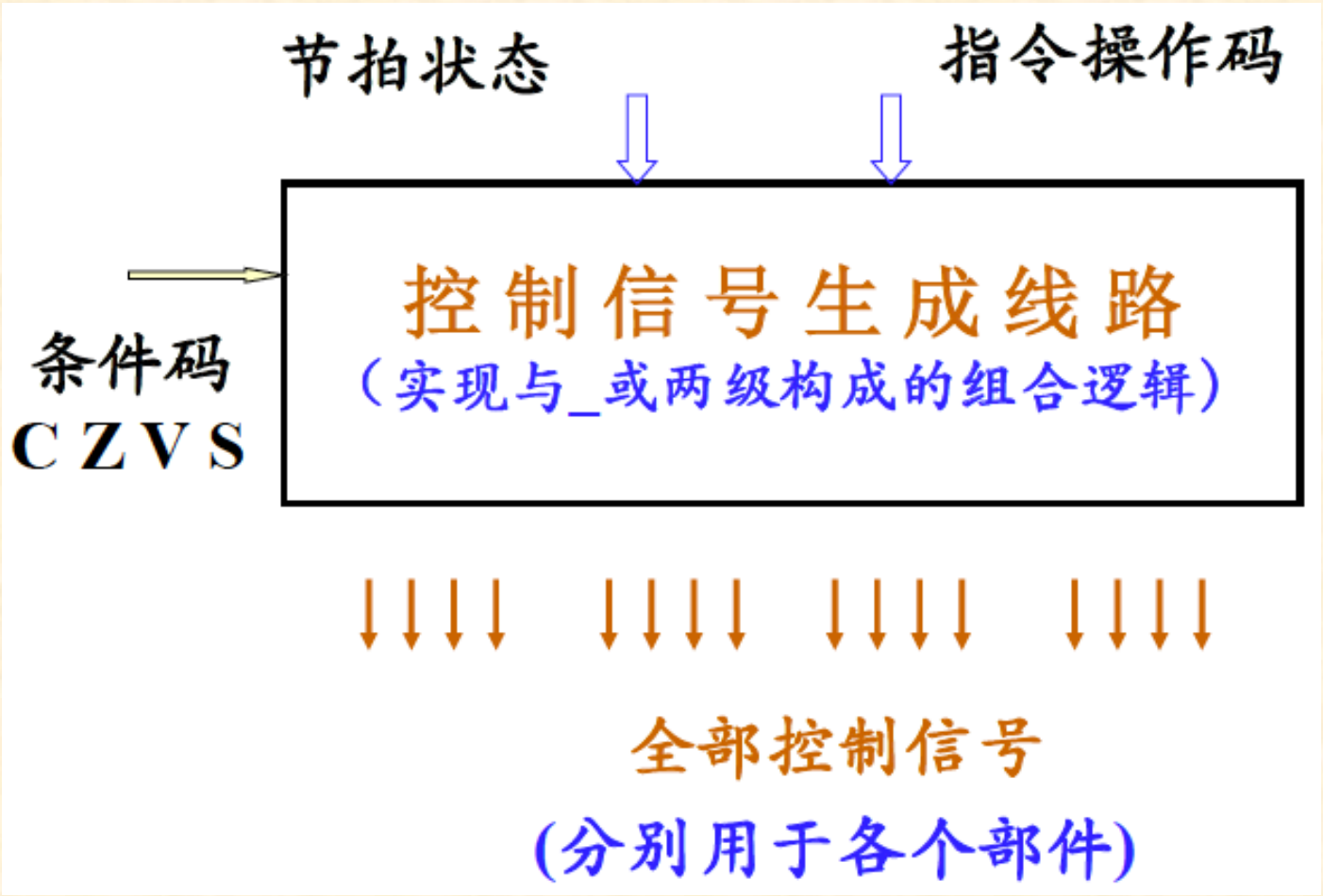
# 1、单周期CPU结构的设计要点



# 2、多周期非流水线CPU结构的设计要点

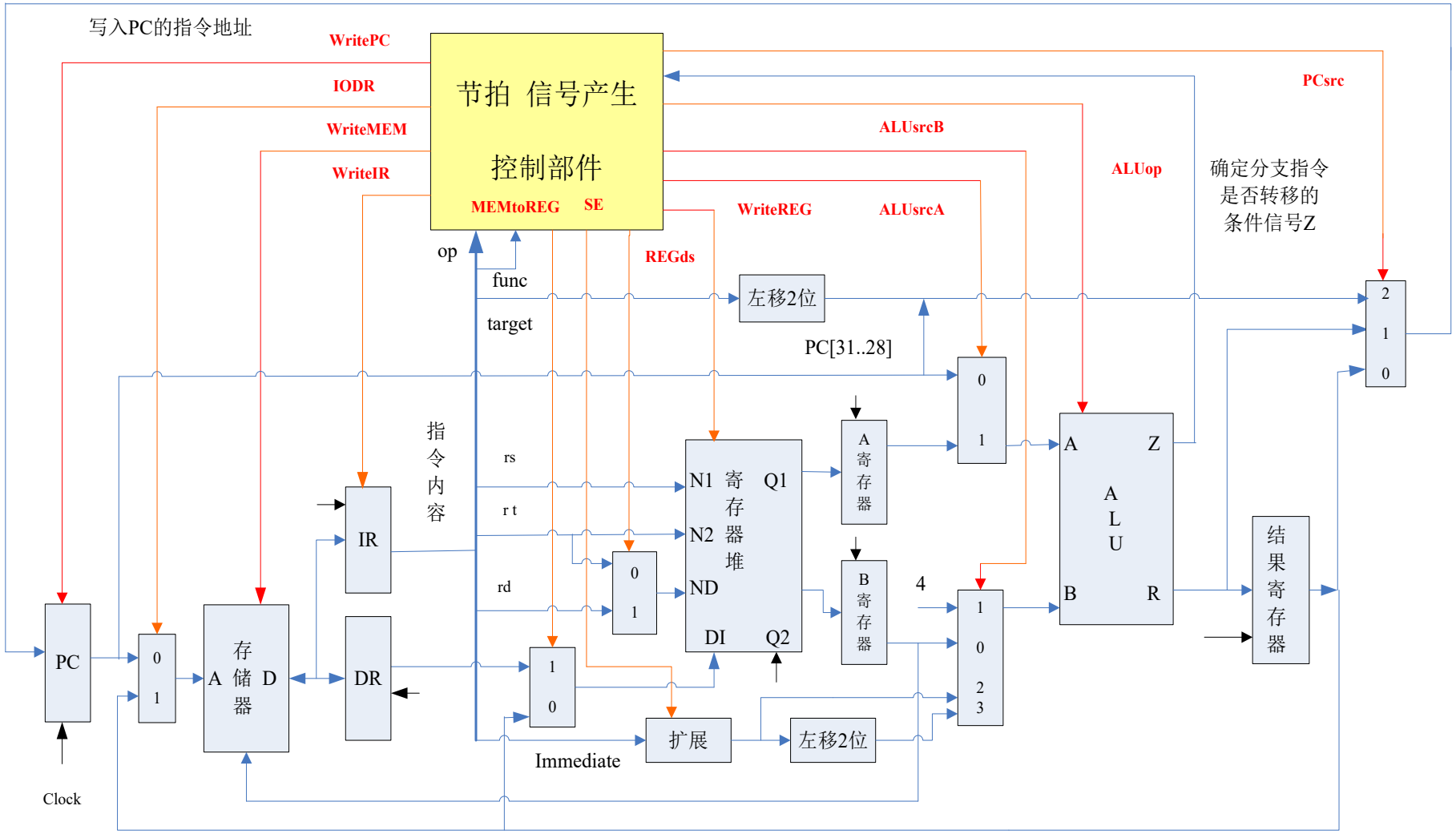


# 2、多周期非流水线CPU结构的设计要点



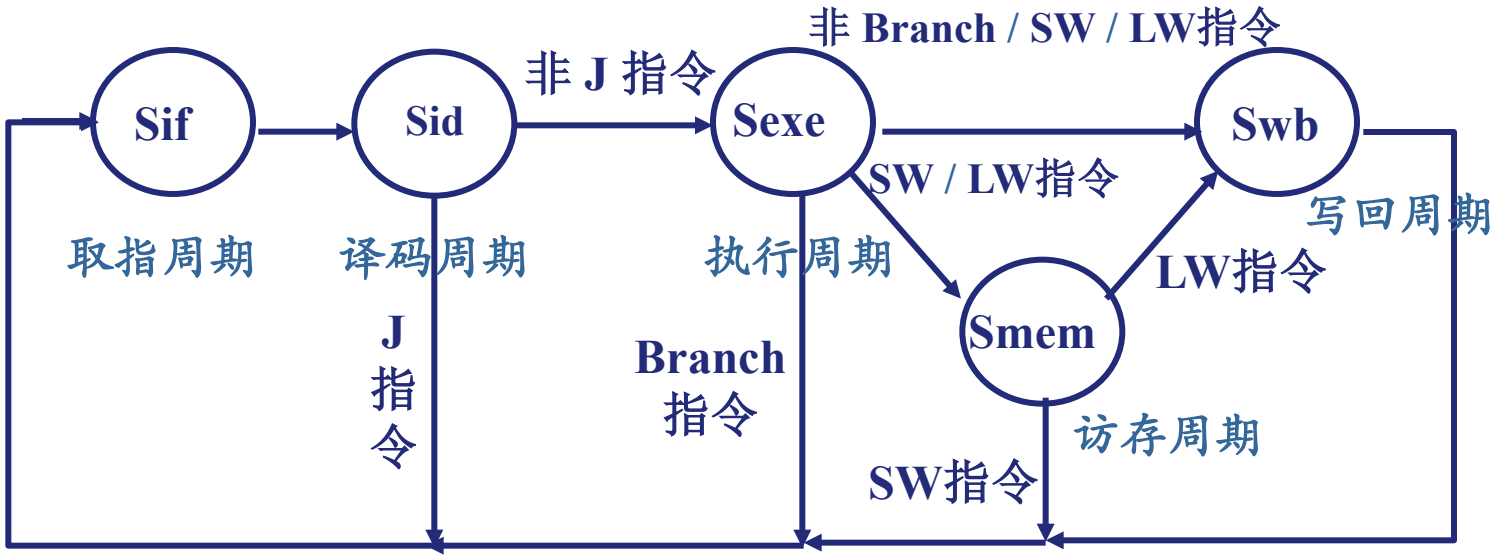
# 2、多周期非流水线CPU结构的设计要点

**控制部件**由**节拍发生器**和**控制信号产生线路**组成，分别完成标明指令执行步骤和向各个部件提供控制信号的功能。



# 3、多周期非流水线CPU结构的设计要点

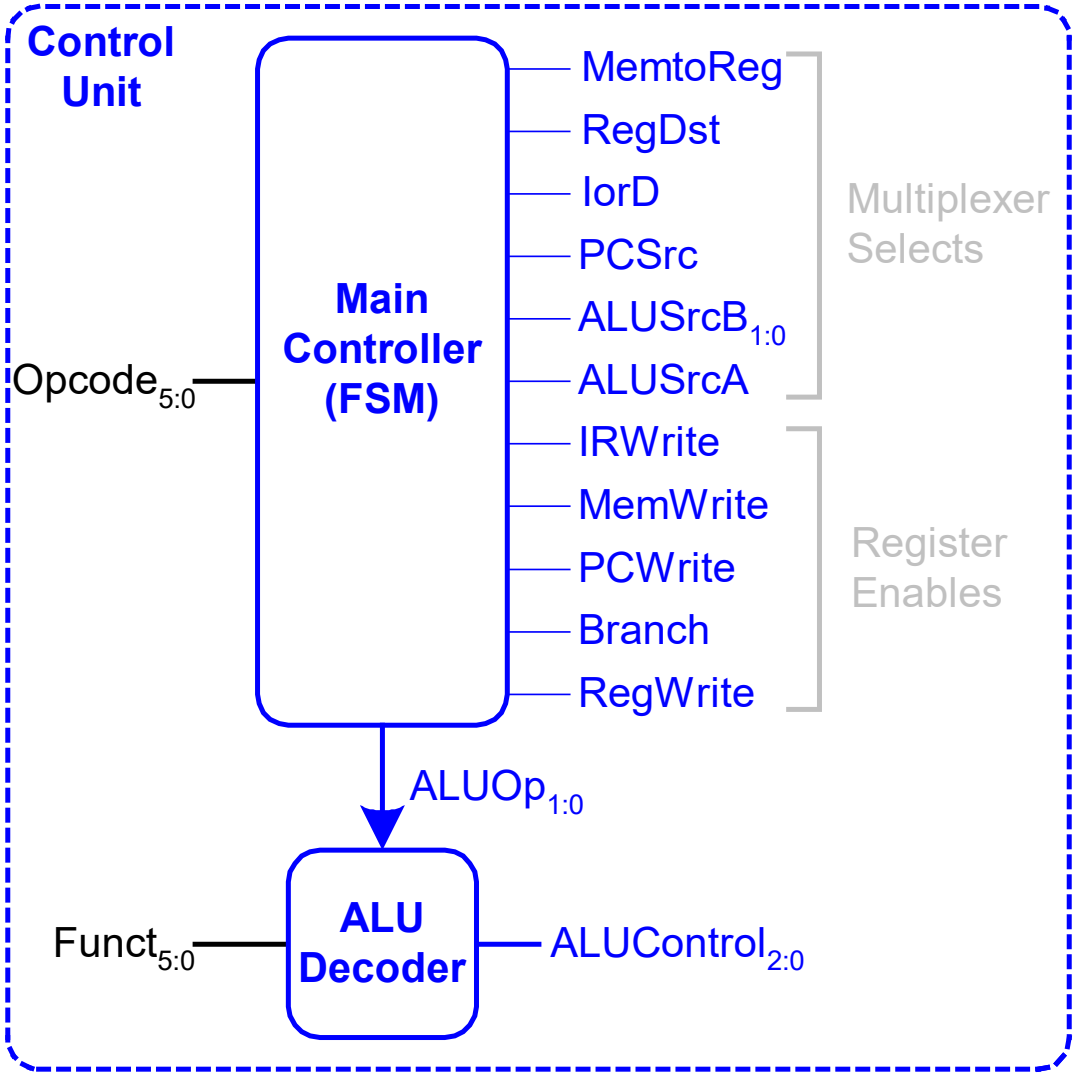
## 状态转移图和指令各执行步骤的操作功能




| 指令步骤 | 读取指令 | 指令译码 | 执行运算 | 内存读写 | 数据写回 |
|---|---|---|---|---|---|
| J 指令 | IR←MEM[PC]<br>PC ← PC+4 | PC←PC[31..28]‖(target<<2) | | | |
| Branch型 | | C←PC+(符号扩展(imm)<<2) | 若条件成立则 PC←R | | |
| R 类型 | | A←Reg[rs] | C←A op B | | Reg[rd]←C |
| Sw指令 | | | C←A + 符号扩展(Imm) | Mem[C]←B | |
| Lw指令 | | B←Reg[rt] | | DR←Mem[C] | Reg[rt]←DR |

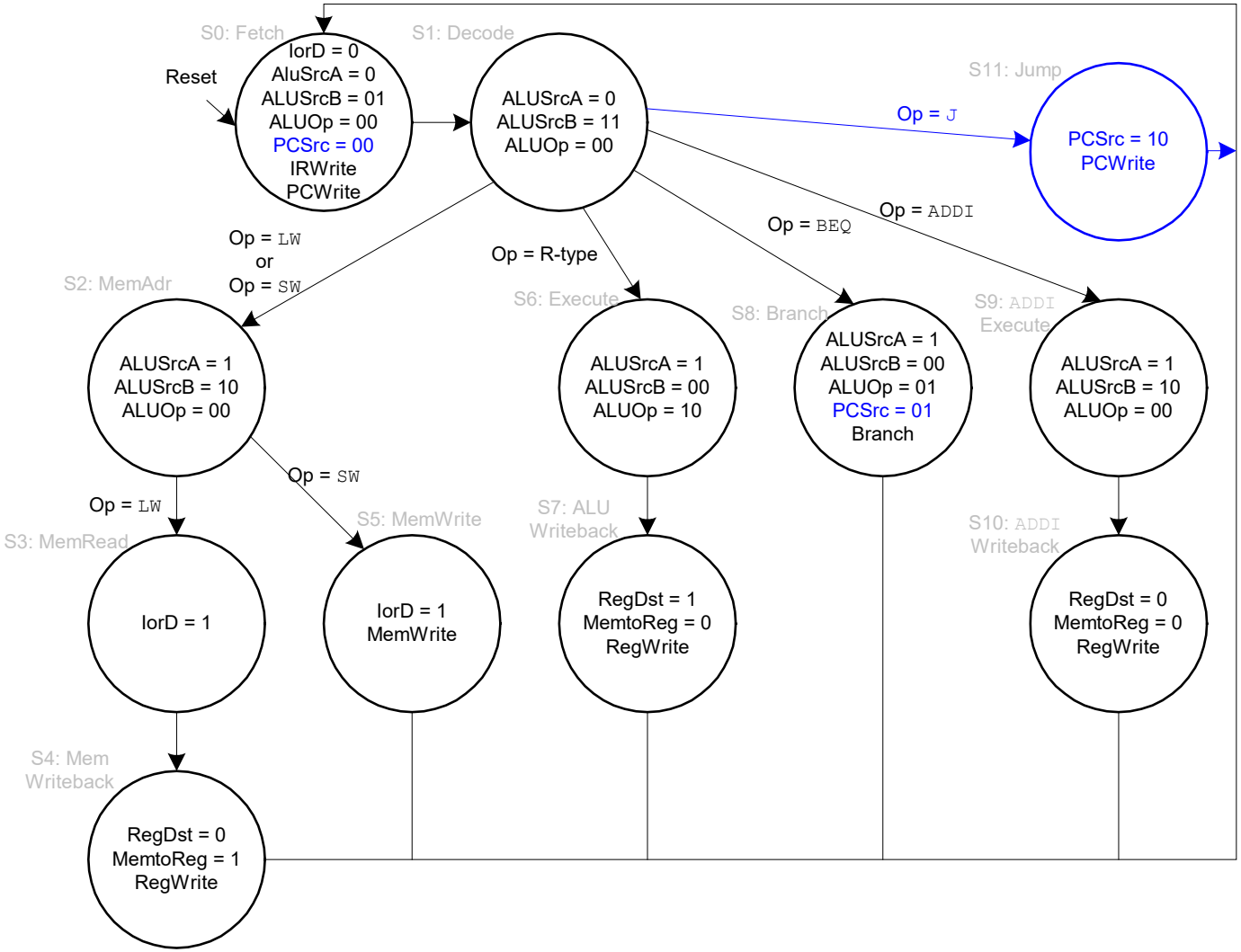
# 2、多周期非流水线CPU结构的设计要点

## Control Unit



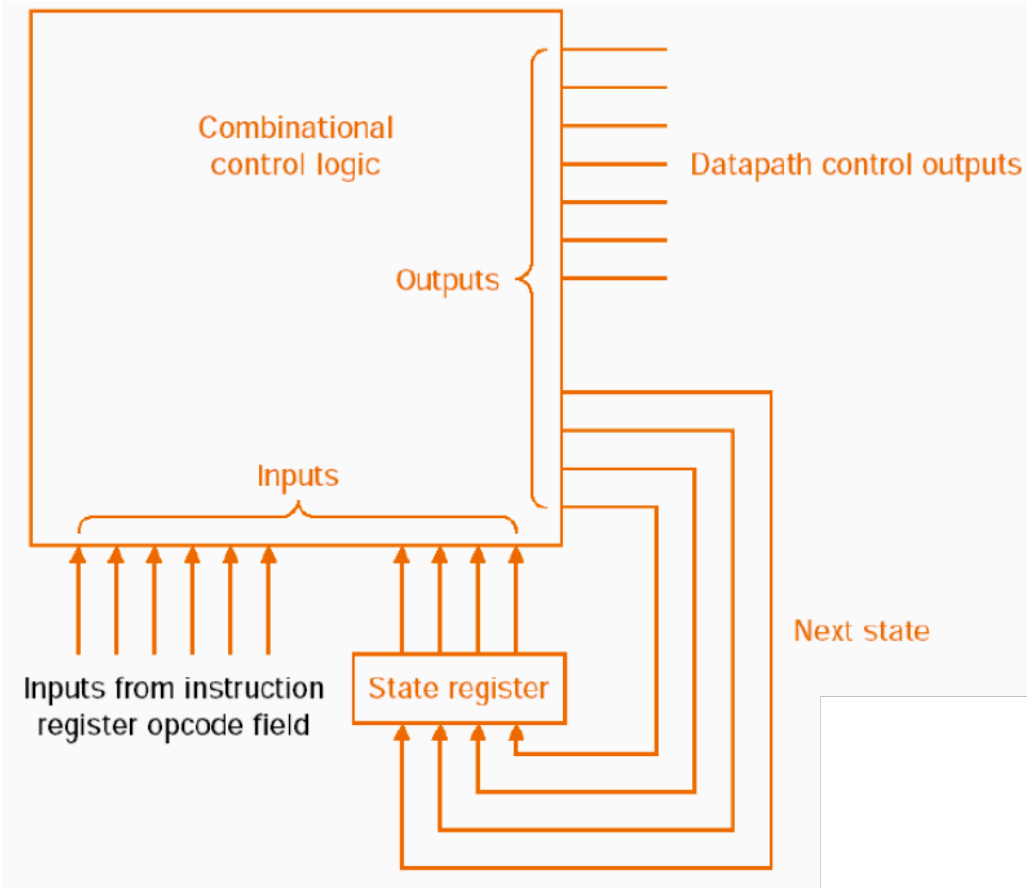
# 2、多周期非流水线CPU结构的设计要点

## Control FSM



# 3、多周期非流水线CPU结构的设计要点

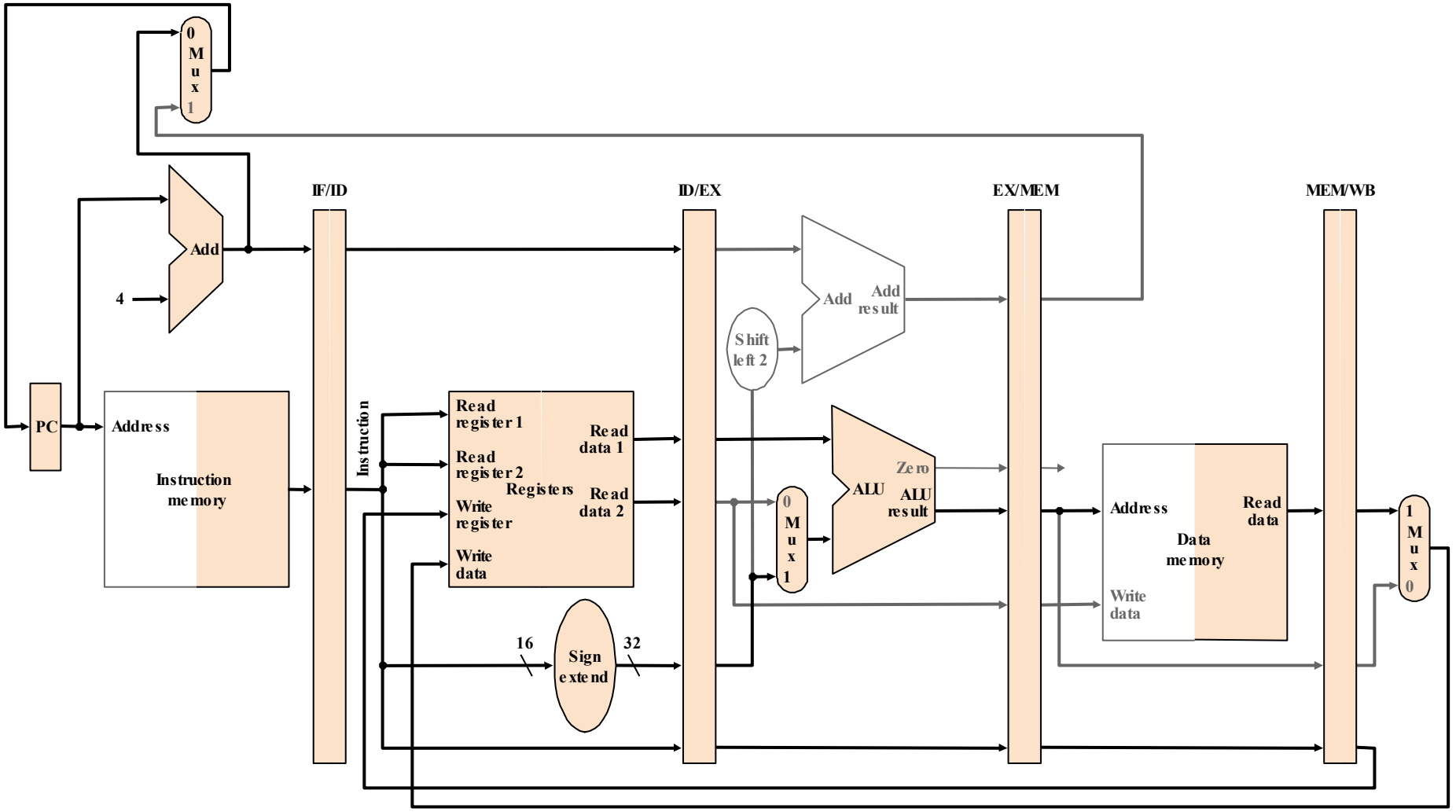
## FSM Implementation



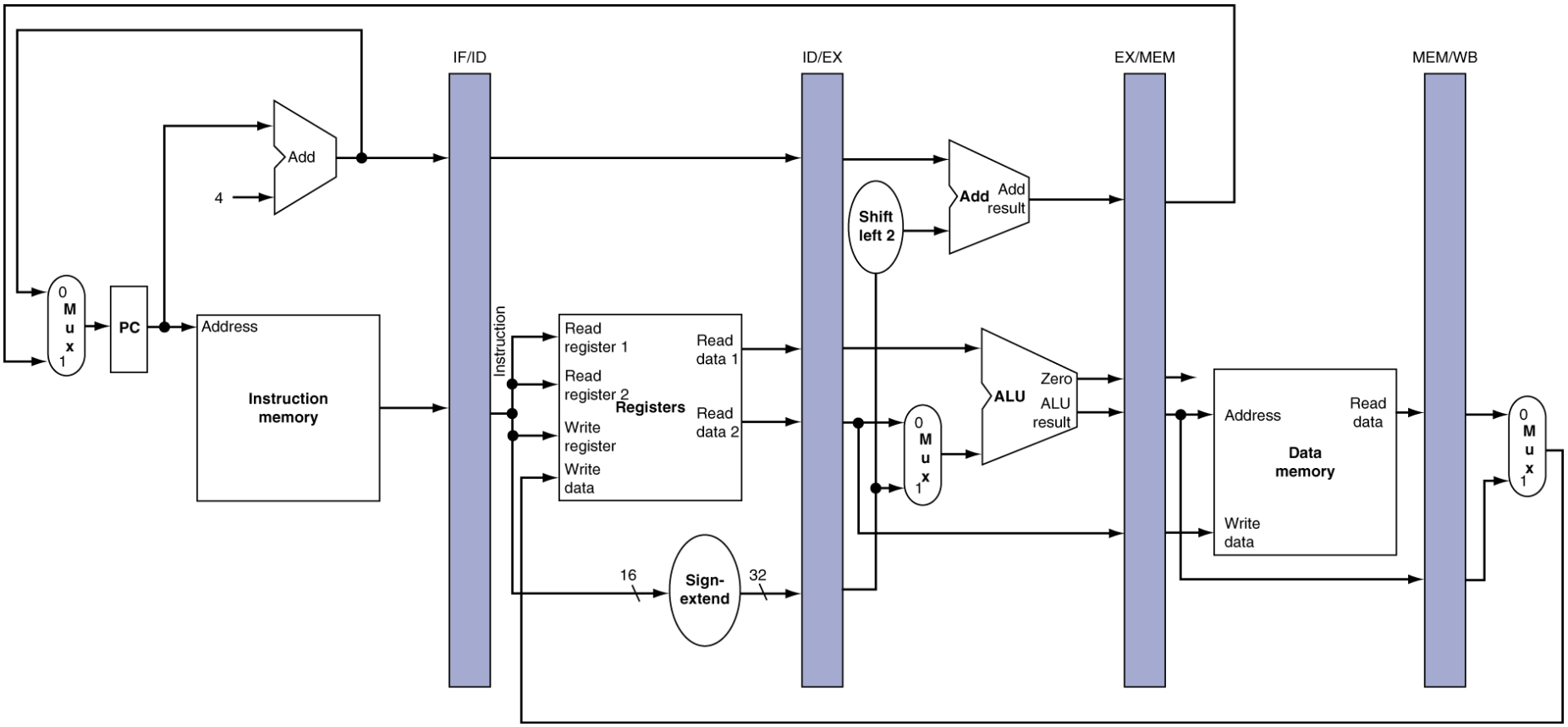

使用有限状态机
标志执行步骤

# 3、多周期流水线CPU结构的设计要点

## 流水线的（各个阶）段寄存器

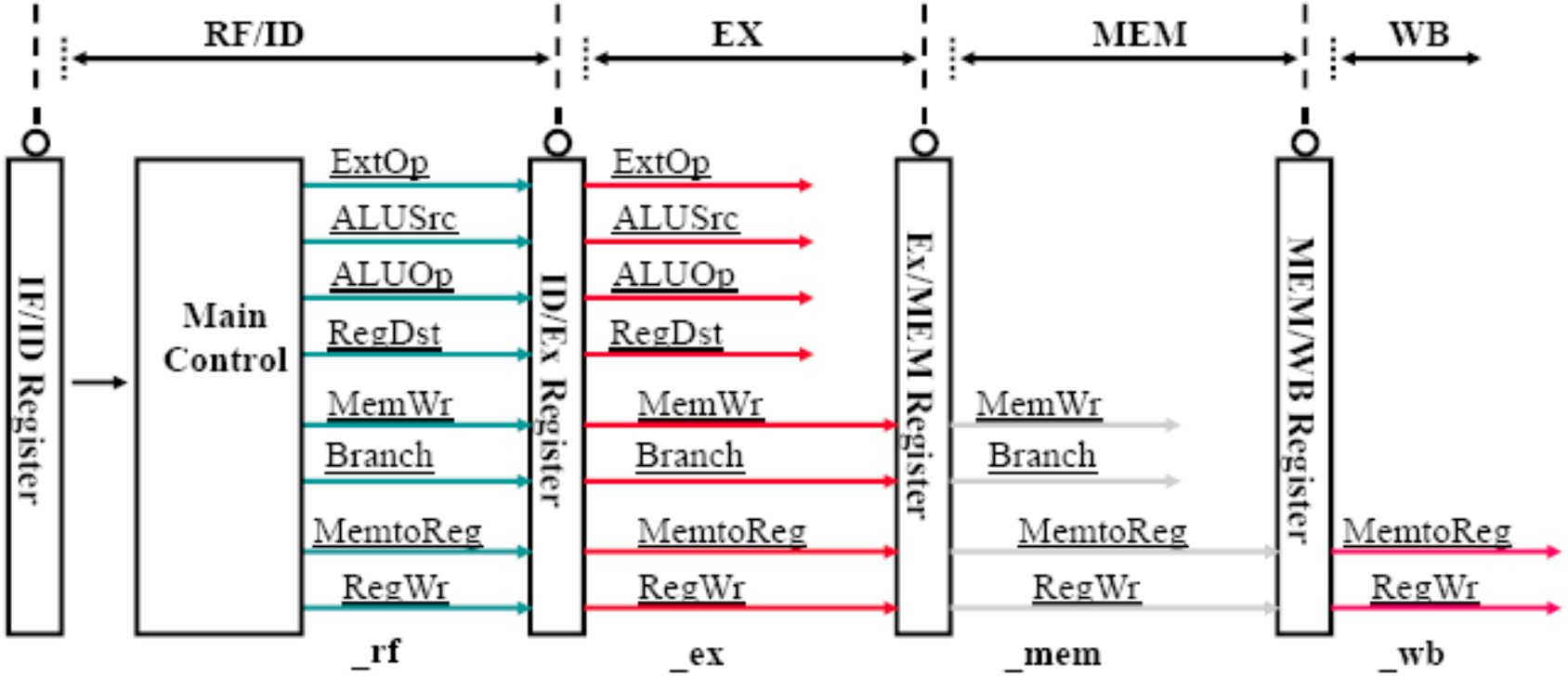


# 设计时注意事项


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign-extend
32
Address
Read data
Data memory
Write data

# 3、多周期流水线CPU结构的设计要点

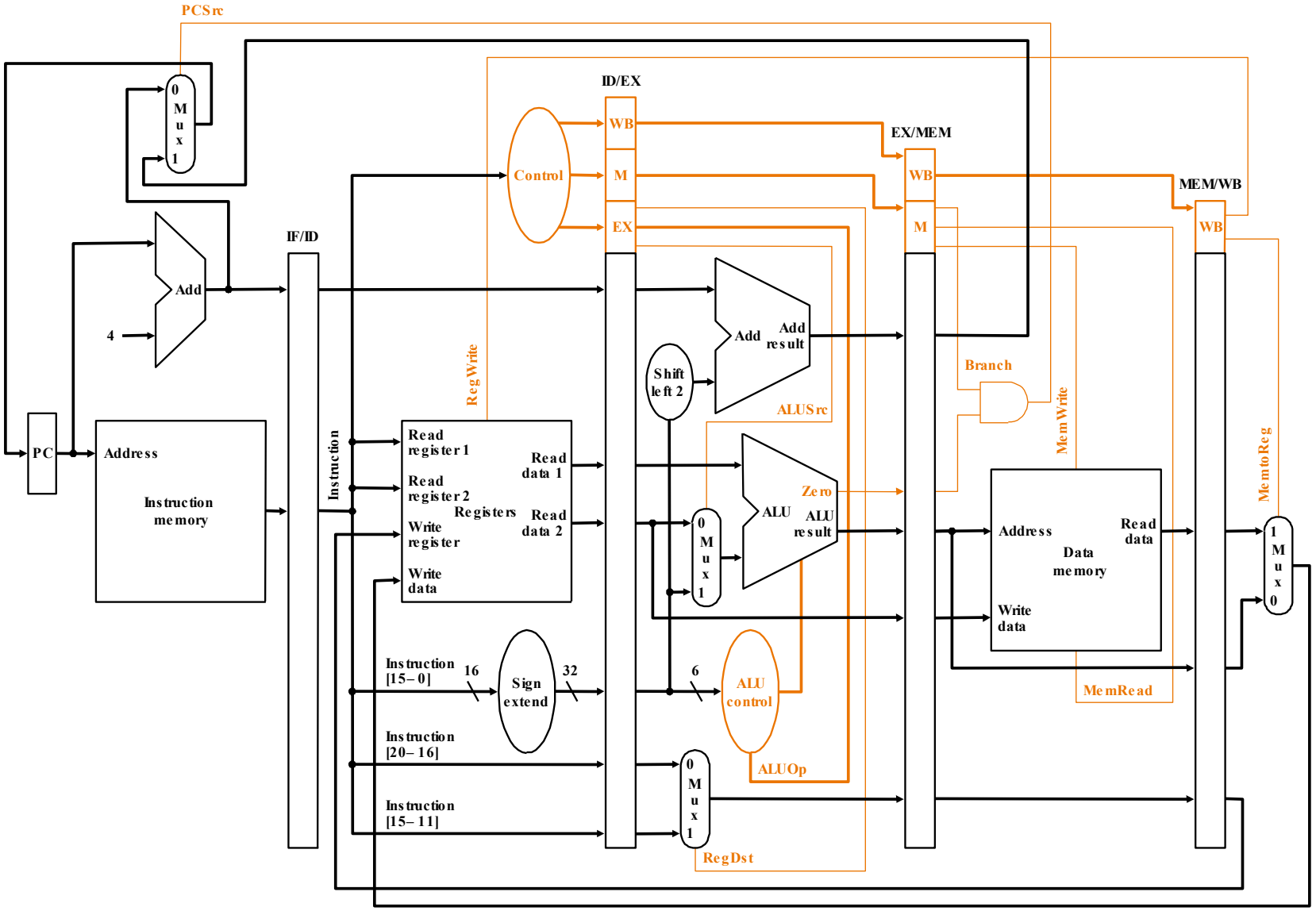
## 流水线控制的实现

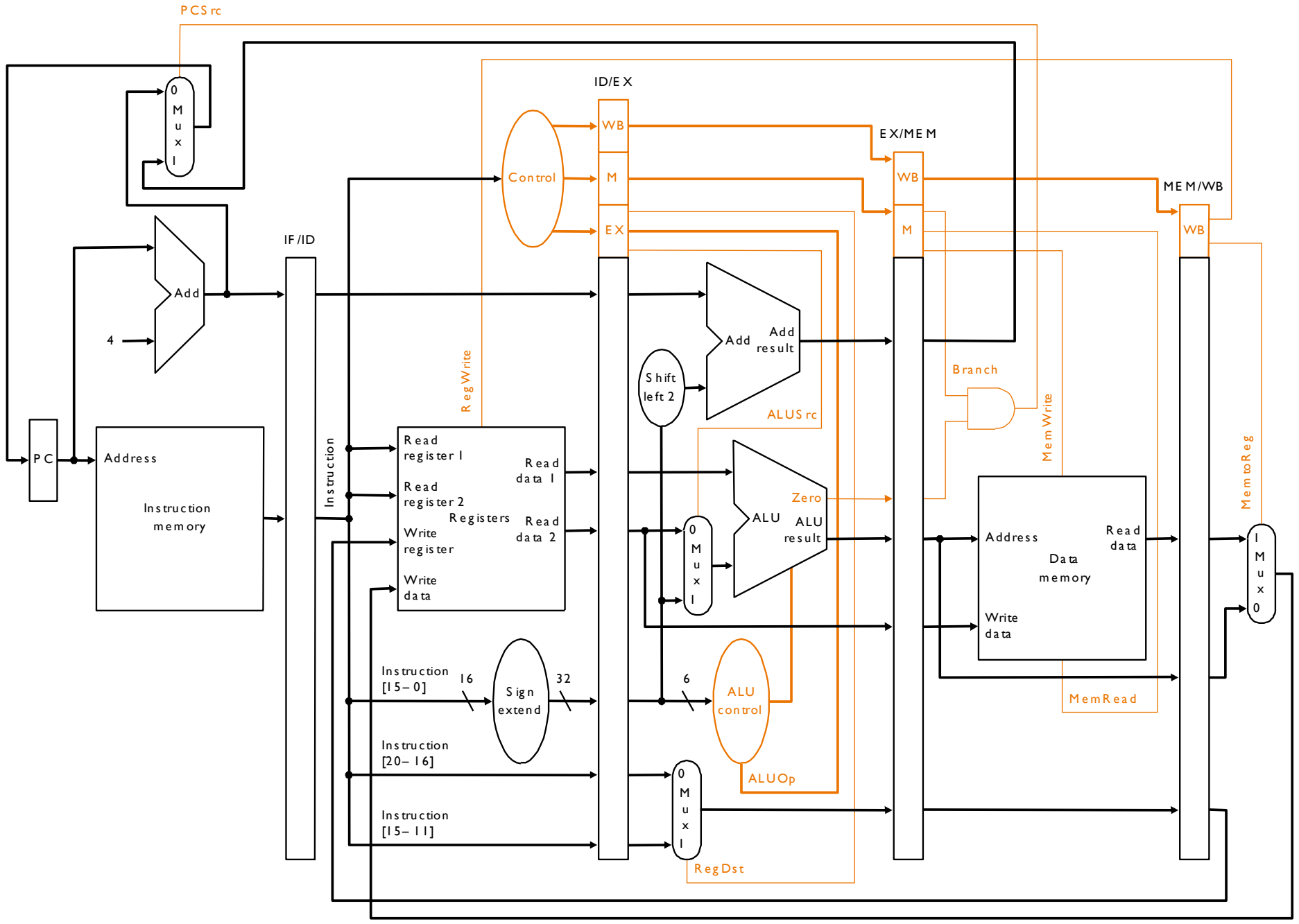



- 在RF/ID阶段生成控制信号
  - 1个时钟周期后使用EX要用的控制信号
  - 2个时钟周期后使用MEM要用的控制信号
  - 3个时钟周期后使用WB要用的控制信号

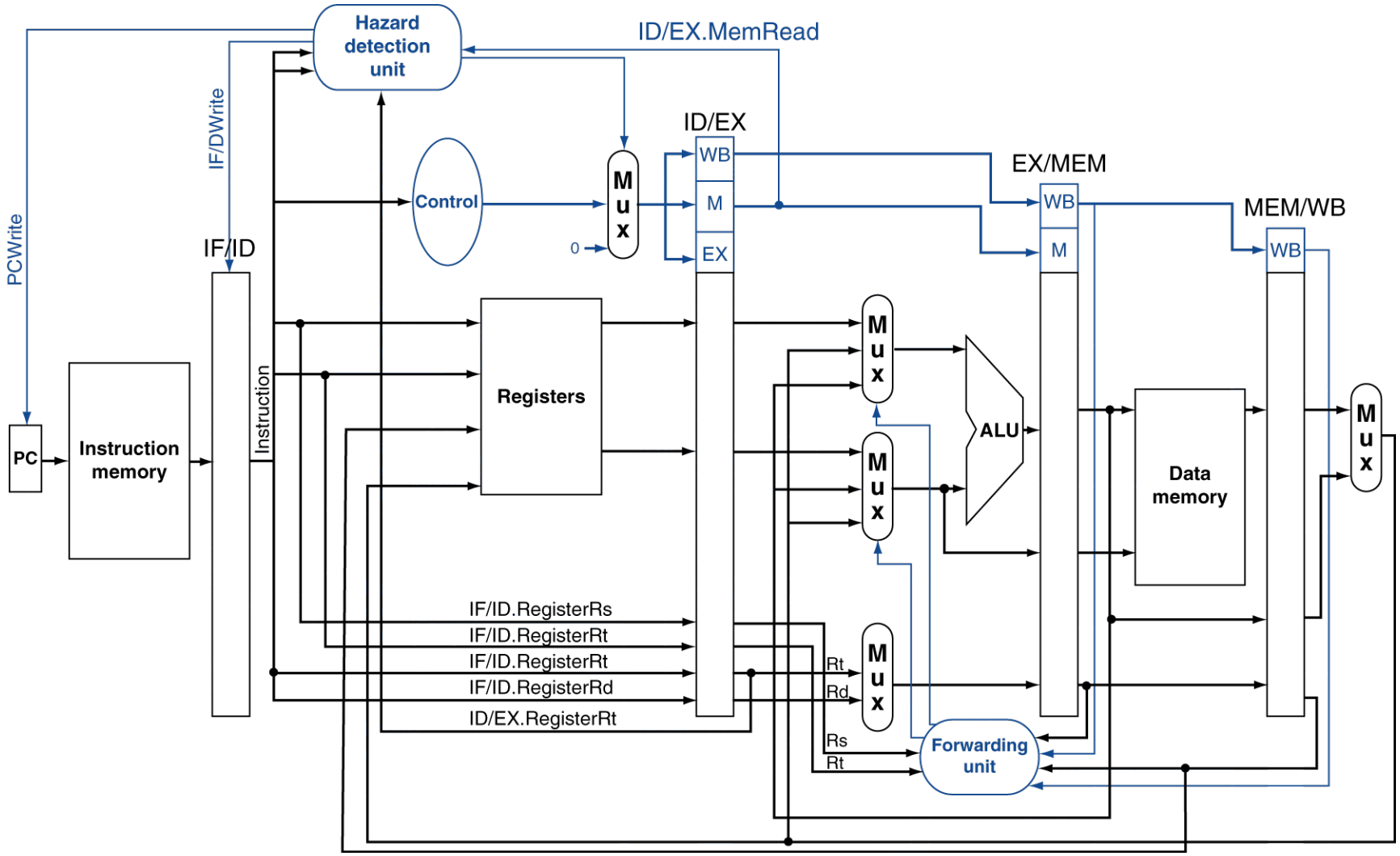
# 3、多周期流水线CPU结构的设计要点

## 支持流水的CPU

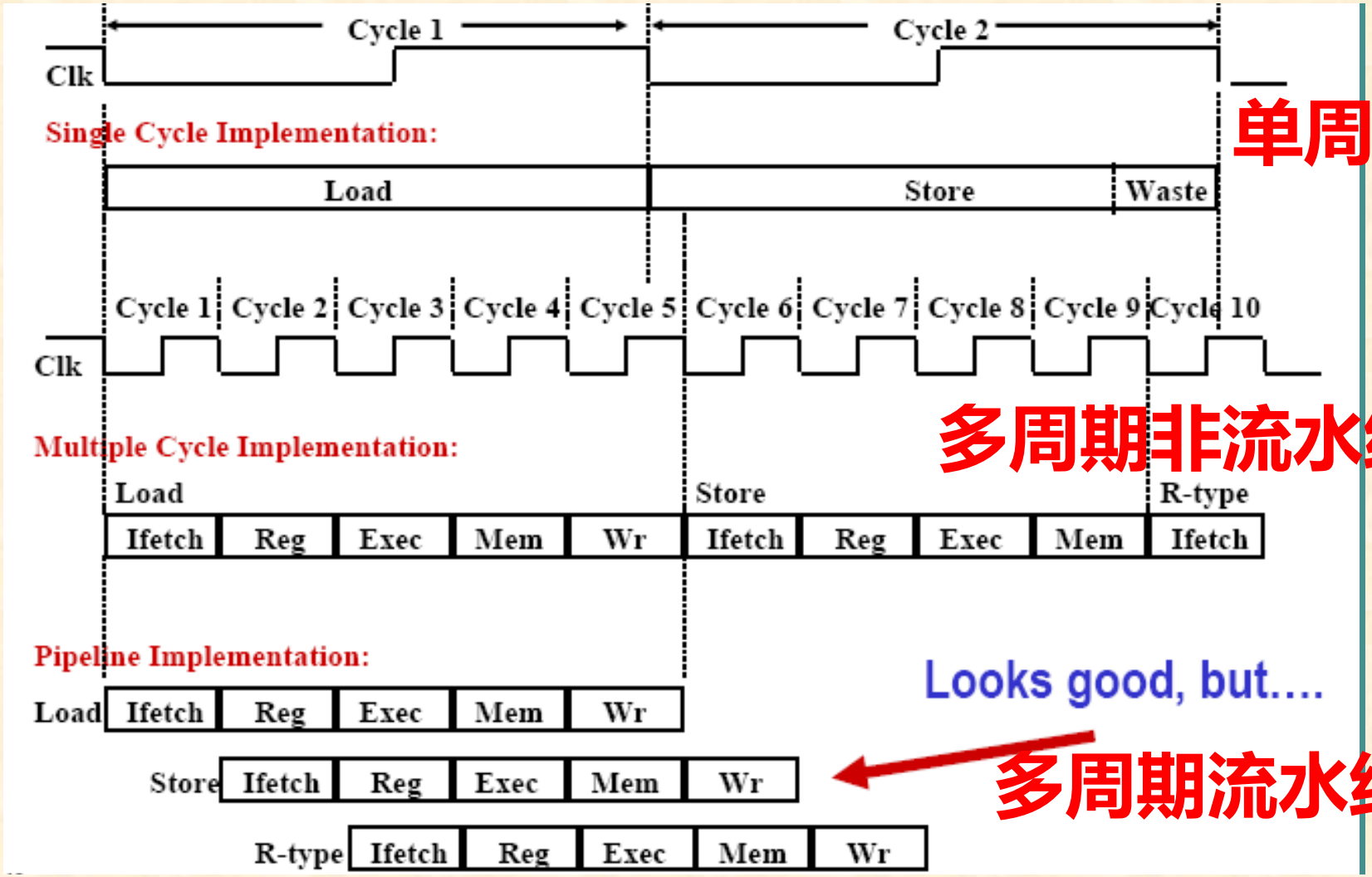



PCSrc
0
M
u
x
1
ID/EX
WB
M
EX
Control
EX/MEM
WB
M
MEM/WB
WB
Add
4
IF/ID
RegWrite
Add
Add result
Shift left 2
ALUSrc
Branch
MemWrite
MemtoReg
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
MemRead
Instruction [15–0]
16
Sign extend
32
6
ALU control
ALUOp
Instruction [20–16]
Instruction [15–11]
RegDst

# 能处理数据冲突的数据通路



# 单周期、多周期非流水线与多周期流水线比较



# 硬件综合课程设计注意事项

一、三种不同的指令集

二、CPU的硬件结构与verilog模块

三、单周期 多周期非流水线与多周期流水线结构

四、典型模块结构，verilog代码分析

# 1、单周期CPU设计

# 2、多周期非流水线CPU设计

# 3、多周期流水线CPU设计


# Thank You !