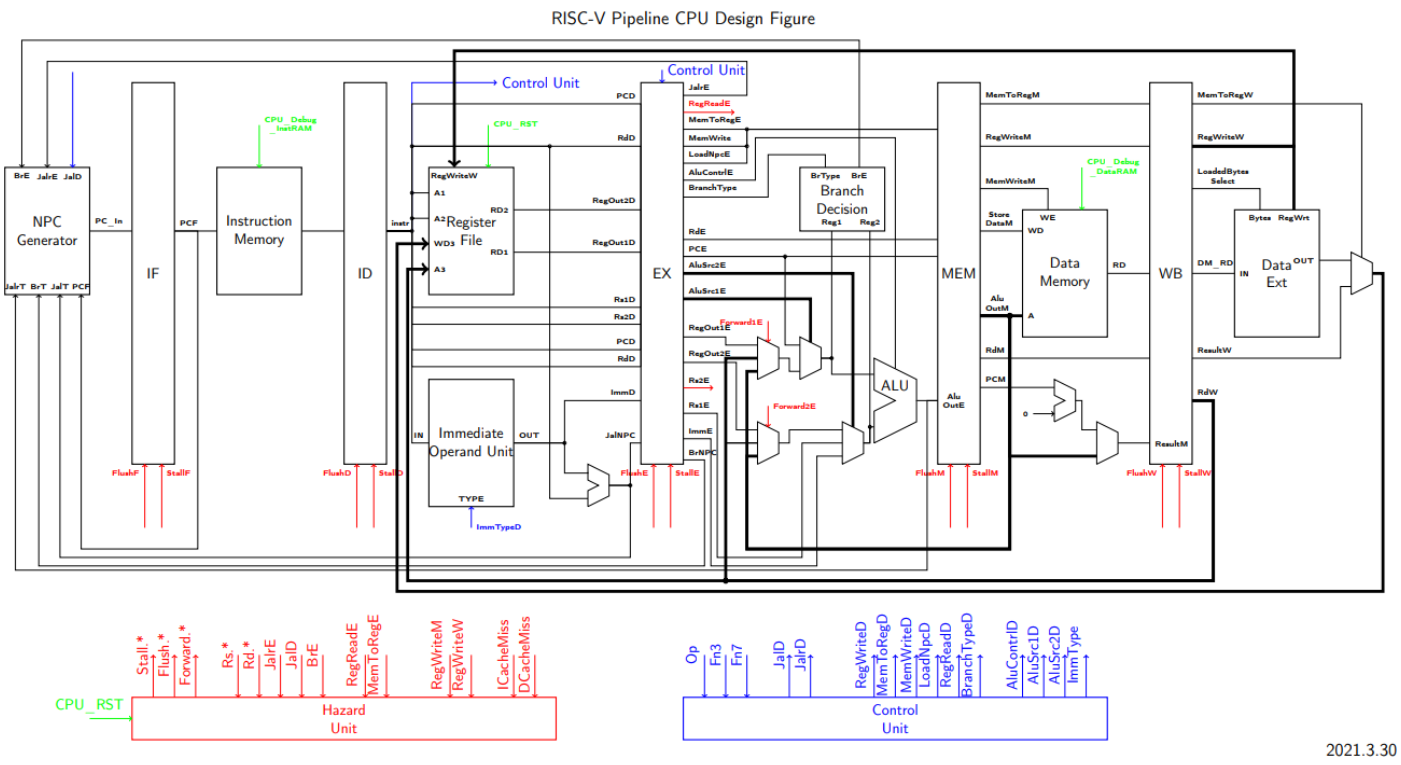
# 计算机体系结构 实验 1

2021.4.2-4.18　　PB18151866 龚小航

## 根据 RISC-V Pipeline CPU Design Figure 回答以下问题：



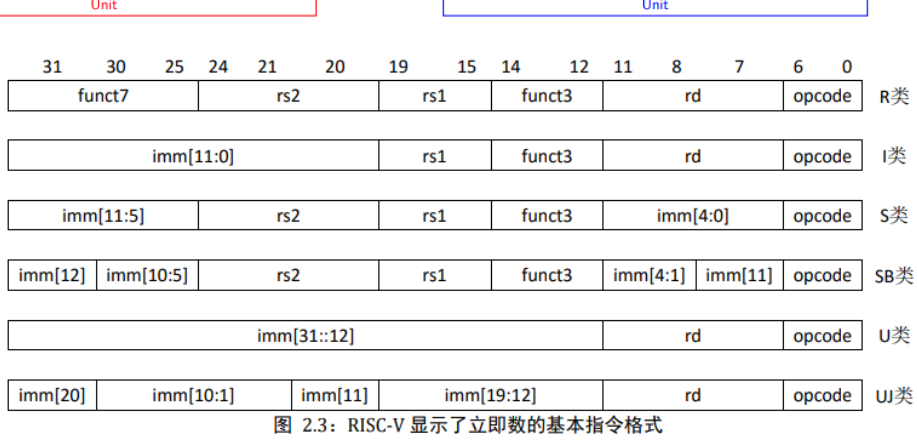




图 2.3：RISC-V 显示了立即数的基本指令格式

### 1、描述执行一条 XOR 指令的过程（数据通路、控制信号等）。[R-Type]

以每个时钟周期系统的操作分别列于下：

① NPC Generator 确定下一条指令的地址，选中这条 XOR 指令对应的地址。此时处于预取指阶段。

② IF 段：时钟边沿来临，IF 段间寄存器刷新，将这条 XOR 的地址写入 IF 段间寄存器。同时 Instruction Memory 模块的输入变成这条 XOR 指令的地址，因此 Instruction Memory 模块的输出为这条 32 位的 XOR 指令。

③ ID 段：时钟边沿来临，ID 段间寄存器刷新，这条 XOR 指令进入 ID 段间寄存器。之后 32 位的 instr 的不同部分进入各模块进行不同处理：instr[0:6]被送入 Control Unit 作为 Opcode 产生控制信号；instr[15:19]作为 Rs 送入 Register File 的 A1 口中成为第一个源操作数寄存器地址，instr[20:24]作为 Rt（Rs2）送入 Register File 的 A2 口中成为第二个源操作数寄存器地址，同时 RD1,RD2 输出 A1,A2 口地址对应的寄存器数据；instr[7:11]是目标寄存器地址，直接送入 EX 段寄存器（此时还未刷新，未写入 EX 段间寄存器）；instr[12:14],instr[25:31]也送入 Control Unit 作为 ALU 功能选择信号；instr[0:11]送入 Immediate Operand Unit 模块，但这是一条 R-type 指令，因此这一部分被控制信号屏蔽（未选择）。

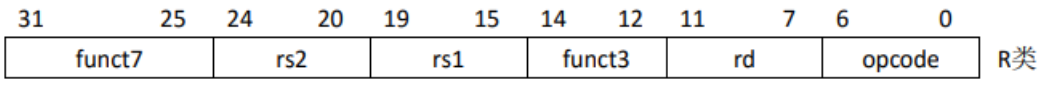
控制信号：（高电平有效，多路选择器从上到下选择信号 00，01，10，11 或 0，1）?表示无关，不改变其值

JalD=0； JalrD=0； RegWriteD=1； MemToRegD=0； MemWriteD=0； LoadNpcD=0； RegReadD=1；

BranchTypeD=?； AliControlD=（XOR）； AluSrc1D=1； AluSrc2D=00； ImmTypeD=?

冲突检测模块信号：

Flush=stall=0； Forward1E=00； Forward2E=00




④ EXE 段：时钟边沿来临，EXE 段间寄存器刷新，RegOut1E,RegOut2E 在控制信号和冲突处理信号的选择下分别通过两个多路选择器进入 ALU，控制单元的 ALUop 使 ALU 做 XOR 运算并将结果输出；其余控制信号和 ALU 结果以及 Rd 一起被传入 MEM 段间寄存器（此时还未刷新，MEM 寄存器未被写入）。

⑤ MEM 段：时钟边沿来临，MEM 段间寄存器刷新。XOR 指令无需读写存储器，仅把 ALU 的运算结果送入 WB 段间寄存器，并且将 rd 和控制信号等传输给 WB 寄存器。

⑥ WB 段：时钟边沿来临，WB 段间寄存器刷新。XOR 指令无需读写存储器，与 DataExt 模块无关。RegWriteW 控制信号回写寄存器堆，RdE 信号回传至寄存器堆写地址端口，而 ResultW 经过多路选择器写回寄存器堆。至此这条指令的所有操作完成，指令执行完毕。

### 2、描述执行一条 BEQ 指令的过程（数据通路、控制信号等）。[SB-Type]

以每个时钟周期系统的操作分别列于下：

① NPC Generator 确定下一条指令的地址，选中这条 BEQ 指令对应的地址。此时处于预取指阶段。

② IF 段：时钟边沿来临，IF 段间寄存器刷新，将这条 BEQ 的地址写入 IF 段间寄存器。同时 Instruction Memory 模块的输入变成这条 BEQ 指令的地址，因此 Instruction Memory 模块的输出为这条 32 位的 BEQ 指令。

③ ID 段：时钟边沿来临，ID 段间寄存器刷新，这条 BEQ 指令进入 ID 段间寄存器。之后 32 位的 instr 的不同部分进入各模块进行不同处理：instr[0:6]被送入 Control Unit 作为 Opcode 产生控制信号；instr[15:19]作为 Rs1 送入 Register File 的 A1 口中成为第一个源操作数寄存器地址，instr[20:24]作为 Rs2 送入 Register File 的 A2 口中成为第二个源操作数寄存器地址，同时 RD1,RD2 输出 A1,A2 口地址对应的寄存器数据；如下图所示的 imm 数据分别被取出再送入 Immediate Operand Unit 模块，用于计算出若分支成功后的目标地址。

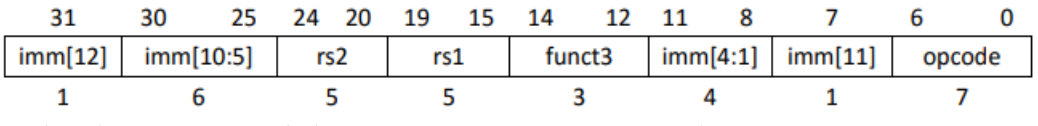
控制信号：（高电平有效，多路选择器从上到下选择信号 00，01，10，11 或 0，1）?表示无关，不改变其值

JalD=0； JalrD=0； RegWriteD=0； MemToRegD=0； MemWriteD=0； LoadNpcD=？； RegReadD=1；

BranchTypeD=(BEQ)； AluControlD=?； AluSrc1D=1； AluSrc2D=00； ImmTypeD=?

冲突检测模块信号：

Flush=stall=0； Forward1E=00； Forward2E=?




④ EXE 段：时钟边沿来临，EXE 段间寄存器刷新，RegOut1E,RegOut2E 在控制信号和冲突处理信号的选择下分别通过两个多路选择器进入 Branch Decision 模块以及 ALU，但这条指令与 ALU 无关因此最后的计算结果只会湮灭在流水线内。Branch Decision 模块判断分支类型以及跳转条件，根据这些生成 BranchE 信号传递给 NPC Generator 模块以生成下一条指令的地址。

特别的，若分支预测失败，则冲突处理模块将 flashI,flashD，flashE 设置为有效，清空段间寄存器。

**3、描述执行一条 LHU 指令的过程（数据通路、控制信号等）。[I-Type]**

从存储器读出半个字（16 位）的无符号数据。LHU R1,0(R2) 以每个时钟周期系统的操作分别列于下：

① NPC Generator 确定下一条指令的地址，选中这条 LHU 指令对应的地址。此时处于预取指阶段。

② IF 段：时钟边沿来临，IF 段间寄存器刷新，将这条 LHU 的地址写入 IF 段间寄存器。同时 Instruction Memory 模块的输入变成这条 LHU 指令的地址，因此 Instruction Memory 模块的输出为这条 32 位的 LHU 指令。

③ ID 段：时钟边沿来临，ID 段间寄存器刷新，这条 LHU 指令进入 ID 段间寄存器。之后 32 位的 instr 的不同部分进入各模块进行不同处理：instr[0:6]被送入 Control Unit 作为 Opcode 产生控制信号；instr[15:19]作为 Rs1 送入 Register File 的 A1 口中成为第一个源操作数寄存器地址，同时 RD1 输出 A1 口地址对应的寄存器数据；instr[7:11]作为 Rd 直接送入 EX 段寄存器（此时还未刷新未写入）；instr[20:31]送入 Immediate Operand Unit 模块，进行位扩展并成为 ImmD 进入下一阶段成为计算目标地址的偏移量。

控制信号：（高电平有效，多路选择器从上到下选择信号 00，01，10，11 或 0，1）

JalD=0； JalrD=0； RegWriteD=1； MemToRegD=1； MemWriteD=0； LoadNpcD=？； RegReadD=1；

BranchTypeD=?； AliControlD=（ADD）； AluSrc1D=1； AluSrc2D=10； ImmTypeD=(I-Type)

冲突检测模块信号：

Flush=stall=0； Forward1E=00； Forward2E=?

| 31 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|
| imm[11:0] | rs1 | funct3 | rd | opcode |
| 12 | 5 | 3 | 5 | 7 |

④ EXE 段：时钟边沿来临，EXE 段间寄存器刷新，RegOut1E,ImmE 在控制信号和冲突处理信号的选择下分别通过两个多路选择器进入 ALU，控制单元的 ALUop 使 ALU 做 LHU 的地址偏移加运算并将结果输出未 AluOutE；其余控制信号和 ALU 结果以及 Rd 一起被传入 MEM 段间寄存器（此时还未刷新，MEM 寄存器未被写入）。

⑤ MEM 段：时钟边沿来临，MEM 段间寄存器刷新。DataMemory 根据输入的 AluOutM 输出对应地址的 RD 数据，控制信号 MemToRegM，RegWriteM 等与 RdM 都直接传入 WB 段间寄存器。

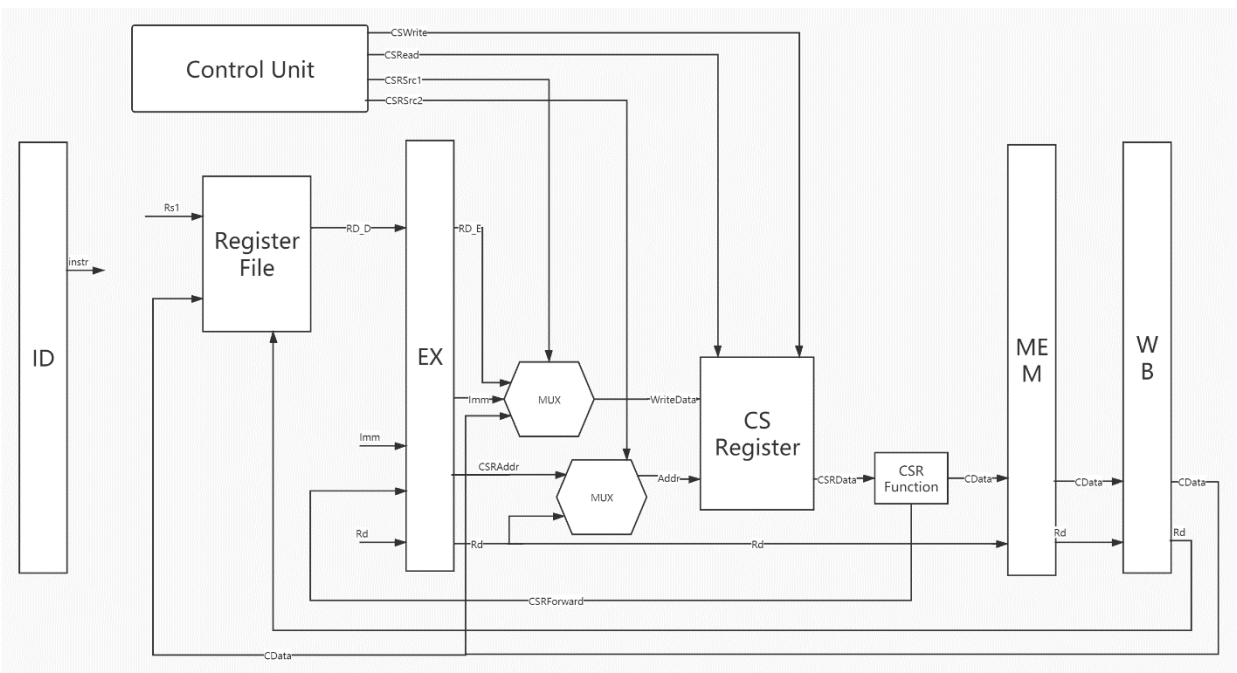
⑥ WB 段：时钟边沿来临，WB 段间寄存器刷新。DataExt 模块选出 32 位数据中的低（或高，与定义有关）16 位，再将空位补 0 得到 32 位数据。RegWriteW 控制信号回写寄存器堆，RdE 信号回传至寄存器堆写地址端口，而 ResultW 经过多路选择器写回寄存器堆。至此这条指令的所有操作完成，指令执行完毕。

**4、如果要实现 CSR 指令（csrrw，csrrs，csrrc，csrrwi，csrrsi，csrrci），设计图中还需要增 加什么部件和数据通路？给出详细说明。**

CSR 指令是对状态控制寄存器组操作的指令，对状态控制寄存器以及寄存器堆进行操作。

| 31 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|
| csr | rs1 | funct3 | rd | opcode |
| 12 | 5 | 3 | 5 | 7 |
| source/dest | source | CSRRW | dest | SYSTEM |
| source/dest | source | CSRRS | dest | SYSTEM |
| source/dest | source | CSRRC | dest | SYSTEM |
| source/dest | uimm[4:0] | CSRRWI | dest | SYSTEM |
| source/dest | uimm[4:0] | CSRRSI | dest | SYSTEM |
| source/dest | uimm[4:0] | CSRRCI | dest | SYSTEM |

增加的数据通路部分如下所示：




CSR 指令放在 EXE 段进行执行，其中 CSRRW 指令会引起数据相关，因此在计算出 CSR 结果后需要一条 forward 旁路回推至 EX 流水段寄存器。

**5、Verilog 如何实现立即数的扩展？**

立即数扩展可以实施零扩展或是符号扩展。扩展模块的 Verilog 代码如下所示，其中假设了输入数据是 16 位且需要扩展成 32 位数据。Ext_en 信号由外部输入，决定实施零扩展或是符号扩展。

```
`timescale 1ns / 1ps
module Sign_Extend(
    input [15:0] immediate,
    input Ext_en, //用于选择零扩展或符号扩展
    output [31:0] out
    );
    assign out[15:0] = immediate;  // 后16位存储立即数
    assign out[31:16] = Ext_en? (immediate[15]? 16'hffff : 16'h0000) : 16'h0000;
        //此句可以实现零扩展和符号扩展的选择
endmodule
```

**6、如何实现 Data Memory 的非字对齐的 Load 和 Store？**

若需要取出非字对齐的 32 位数据（一个字），实现起来非常简单：我们在实行字对齐的存取时将地址最后两位始终置为 00 以实现按字（32bit）操作，实际上存储器支持按字节寻址。因此这种情况只需要将地址最后两位不为 00，就可以实现取出任意按字节寻址的 32 位数据。




另一方面，若需要取出的数据长度不是一个字（32bit），那么就需要额外的一个模块对取出的 32 位数据做截断操作。如图，DataExt 模块就用于处理需要取出特定位数数据的情况（小于等于 32 位）。LoadedBytes Select 信号输入需要选出的位数，该模块从传入的 32 位 DM_RM 数据中截取若干低位（或高位）再将这个数扩展至 32 位，以 OUT 数据输出。

### 7、ALU 模块中，默认 wire 变量是有符号数还是无符号数？

Verilog 中任何变量都由若干 0、1 表示，默认是无符号整数。在比较大小，计算加减法等操作中默认最高位也是数据而不是代表符号。定义一个有符号型变量的指令如下所示：

```
wire signed [3:0] a=4'b1111;//-1
```

### 8、简述 BranchE 信号的作用。

BranchE 信号由 Branch Decision 模块生成，当一条分支指令到达 EX 段时 BranchType 信号告诉该模块这条分支指令的类型，规则等，该模块根据进入的 RegOut1E 与 RegOutE2 生成 BranchE。若输入满足这条分支指令的跳转规则，则 BranchE=1，传入 NPC Generator 使下一条指令产生分支跳转。对于 NPC Generator 模块来说，BranchE 信号的优先级高于 pc+4，若 BranchE=1 则下一条指令地址取为 BrT；若 BranchE=0 则下一条指令地址取为 pc+4。

同时 BranchE 信号有效是分支指令的标志，因此它还作为 Hazard Unit 的输入产生相应的冲突处理信号以清空已经在流水线内但不执行的分支上的指令。

### 9、NPC Generator 中对于不同跳转 target 的选择有没有优先级？

由 NPC Generator 模块的输入输出信号引脚可知，这个流水线共有四种指令地址产生方式：

pc+4；Branch；JAL；JALR

其中 JALR 指令格式为 jalr rs rd，无条件跳转到由寄存器 rs 指定的指令，并将下一条指令的地址保存到寄存器 rd（默认为 31）中；JAL 指令格式为 jal target，无条件跳转到目标指令，并将下一条指令地址保存到寄存器$ra 中。

根据流水线图，JALR 和 Branch 指令在 EXE 阶段产生是否跳转的信号，而 JAL 指令（无条件跳转）在 ID 阶段就开始无条件跳转。Branch 指令和 JALR 指令不可能同时出现在 EXE 段，因此这两条指令不会有冲突，可以认为它们优先级相同；而 Branch/JALR 指令与 JAL 指令由于产生控制信号的流水阶段不同，有可能同时产生改变 NPC 的使能信号：一条 Branch/JALR 指令紧跟着一条 JAL 指令，这样在它们分别执行到 EXE、ID 段的时候将同时产生跳转信号。按照 CPU 设计逻辑，先产生的指令具有高优先级，因此 Branch/JALR 指令优先级更高，若下一条指令不是紧跟着的那条 JAL 指令，那么 JAL 将被从流水段间寄存器中清空。

这四种跳转的优先级为： Branch/JALR > JAL > PC + 4。

### 10、Harzard 模块中，有哪几类冲突需要插入气泡，分别使流水线停顿几个周期？

假设使用了静态分支预测，分支跳转指令都默认分支不发生，此时控制冲突不需要气泡而是清空段间寄存器。CSR 指令可认为是一条 R 型指令，跟在 Load 后且需要用到相同寄存器时停顿一个周期。

| 冲突类型 | 需要插入气泡的冲突指令 | 使流水线停顿的周期 |
|---|---|---|
| 结构冲突 | 设计避免了结构冲突的产生 | 无 |
| 数据冲突 | Load+R-Type 指令使用 load 的值 | 1 |
| | Load+Branch/JALR 使用 load 的值 | 1 |
| 控制冲突 | 只需要清空流水线段间寄存器 | 无 |

### 11、Harzard 模块采用静态分支预测器，即默认不跳转，遇到 branch 指令时，如何控制 flush 和 stall 信号？

若预测成功，分支条件不成立，则 Flush 和 Stall 信号都无效，流水线按照预测继续正常运行。

若预测失败，分支条件成立，则将 FlushD 和 FlushE 信号置为有效，清空 Branch 指令之后的两条已经在流水线中的指令。

另一方面，若这一条 Branch 指令前一条指令是 Load 并且这条 Branch 依赖于上一条 Load 指令取出的数据，则静态分支预测器会在取指阶段就分析出这个冲突并将 stall 信号置为有效，使这条 Load 指令的后一条指令为一个空指令。

### 12、0 号寄存器值始终为 0，是否会对 forward 的处理产生影响？

这只是 CPU 设计中的一条人为规则，只需要在定义的时候考虑到这条规则即可。即对寄存器堆中的寄存器进行任何写操作时，检测目标寄存器。若目标寄存器是 0 号寄存器则不执行赋值操作。因此这条特性不会影响前推。