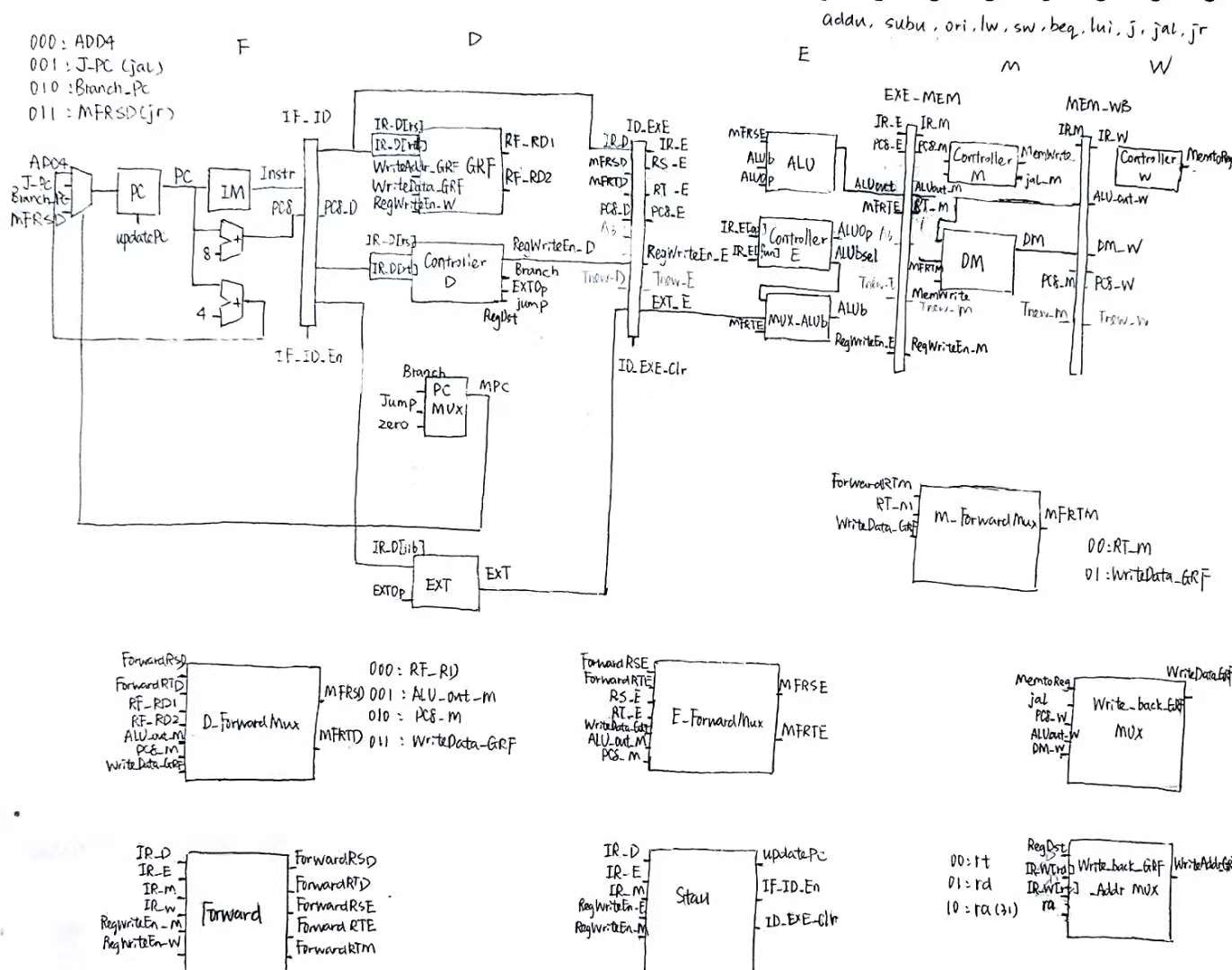
Verilog流水线CPU

1. 顶层设计



1. 模块设计
2. PC

表1 PC接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| PC.v | input [31:0] PC\_in  input clk,  input reset,  input update\_PC,  output reg [31:0] PC\_out |

1. IM

表2 IM接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| IM.v | input [31:0] PC; // instruction address  output [31:0] out; // instruction |

1. IF/ID（F/D级流水线寄存器）

表3 IF/ID接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| IF\_D.v | input clk,  input reset,  input en, //写入使能信号，和stall相关  input [31:0] PC8,  input [31:0] IR\_in,  output [31:0] IR\_out,  output [31:0] PC8\_out |

1. D\_ForwardMux

表4 D转发器接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| MUX.v | input [2:0] ForwardRSD,  input [2:0] ForwardRTD,  input [31:0] RF\_RD1,  input [31:0] RF\_RD2,  input [31:0] ALU\_out\_M,  input [31:0] PC8\_E,  input [31:0] PC8\_M,  input [31:0] WriteData\_GRF,  output [31:0] MFRSD,  output [31:0] MFRTD |

1. Controller

表5 控制器接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| Controller.v | input [5:0] op,  input [5:0] fun,  output [1:0] RegDst,  output AluSrc,  output MemtoReg,  output RegWrite,  output MemWrite,  output MemRead,  output [2:0] ExtOp,  output [3:0] ALUOp,  output Branch,  output [1:0] jump,  output is\_jal |

1. Write\_back\_GRF\_Addr\_MUX

表6 写入寄存器地址的多路选择器接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| MUX.v | input [4:0] rd,  input [4:0] rt,  input [4:0] ra,  input [1:0] RegDst,  output reg [4:0] WriteAddr\_GRF |

1. PC\_Mux

表7 PC多路选择器信号生成模块接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| ALU\_ctl.v | input Branch,  input [1:0] jump,  input [31:0] MFRSD,  input [31:0] MFRTD,  output reg [2:0] MPC |

1. EXT

表8 EXT接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| EXT.v | input [15:0] imm16,  input [2:0] EXTOp,  output [31:0] ext\_output |

1. ID/EXE（D/E级流水线寄存器）

表9 ID/EXE接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| ALU.v | input [31:0] IR\_in,  input [31:0] RD1\_in,  input [31:0] RD2\_in,  input [31:0] EXT\_in,  input [31:0] PC8,  input [4:0] WriteAddr\_GRF,  input [4:0] A1\_D,  input [4:0] A2\_D,  input [1:0] Tnew\_D,  input RegWriteEn\_in,  output [31:0] IR\_out,  output [31:0] RD1\_out,  output [31:0] RD2\_out,  output [31:0] EXT\_out,  output [31:0] PC8\_out,  output [1:0] Tnew\_E,  output [4:0] A1\_E,  output [4:0] A2\_E,  output [4:0] WriteAddr\_GRF\_E,  output RegWriteEn\_out,  input clk,  input reset,  input clr |

1. ALUSrc\_Mux

表10 ALUSrc\_Mux接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| ALU.v | input ALUSrc,  input [31:0] RD2,  input [31:0] imm32,  output [31:0] ALU\_B |

判断是需要转发后的GRF[rt]还是拓展后的imm。

1. ALU

表11 ALU接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| ALU.v | input [31:0] A,  input [31:0] B,  output reg [31:0] C,  input [3:0] ALUOp |

1. E\_ForwardMux

表12 E级转发器接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| MUX.v | input [2:0] ForwardRSE,  input [2:0] ForwardRTE,  input [31:0] RS\_E,  input [31:0] RT\_E,  input [31:0] WriteData\_GRF,  input [31:0] ALU\_out\_M,  input [31:0] PC8\_M,  output [31:0] MFRSE,  output [31:0] MFRTE |

1. EXE\_MEM（E/M级流水线寄存器）

表13 EXE\_MEM接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| EXE\_MEM.v | input clk,  input reset,  input [31:0] IR\_in,  input [31:0] Add\_in,  input [31:0] RD2\_in,  input [31:0] PC8\_in,  input [1:0] Tnew\_E,  input [4:0] WriteAddr\_GRF\_E,  input [4:0] A1\_E,  input [4:0] A2\_E,  input RegWriteEn\_in,  output [31:0] IR\_out,  output [31:0] Add\_out,  output [31:0] RD2\_out,  output [31:0] PC8\_out,  output [1:0] Tnew\_M,  output [4:0] WriteAddr\_GRF\_M,  output [4:0] A1\_M,  output [4:0] A2\_M,  output RegWriteEn\_out |

1. M\_ForwardMux

表14 M级转发器接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| MUX.v | input [31:0] RT\_M,  input [31:0] WriteData\_GRF,  input [1:0] ForwardRTM,  output [31:0] MFRTM |

1. DM

表15 DM接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| DM.v | input [31:0] A,  input [31:0] WD,  input [31:0] PC,  input WE,  input clk,  input Reset,  output [31:0] RD |

1. MEM\_WB

表16 M/W级流水线寄存器接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| MEM\_WB.v | input clk,  input reset,  input [31:0] IR\_in,  input [31:0] Add\_in,  input [31:0] DR\_in,  input [31:0] PC8,  input [1:0] Tnew\_M,  input [4:0] WriteAddr\_GRF\_M,  input RegWriteEn\_in,  output [31:0] IR\_out,  output [31:0] Add\_out,  output [31:0] DR\_out,  output [31:0] PC8\_out,  output [1:0] Tnew\_W,  output [4:0] WriteAddr\_GRF\_W,  output RegWriteEn\_out |

1. Write\_back\_GRF\_MUX

表17 写回寄存器的值多路选择器接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| MUX.v | input MemtoReg,  input jal,  input [31:0] PC8\_W,  input [31:0] ALU\_out\_W,  input [31:0] DM\_W,  output reg [31:0] WriteData\_GRF |

1. AT

表18 解码器接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| AT.v | input [31:0] IR,  output Tuse\_RS0,  output Tuse\_RS1,  output Tuse\_RT0,  output Tuse\_RT1,  output Tuse\_RT2,  output reg [1:0] Tnew\_D,  output [4:0] A1,  output [4:0] A2 |

1. Forward

表19 转发信号控制接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| AT.v | input RegWriteEn\_E,  input RegWriteEn\_M,  input RegWriteEn\_W,  input [4:0] A1\_D,  input [4:0] A2\_D,  input [4:0] A1\_E,  input [4:0] A2\_E,  input [4:0] A2\_M,  input [4:0] A3\_E,  input [4:0] A3\_M,  input [4:0] A3\_W,  input [1:0] Tnew\_E,  input [1:0] Tnew\_M,  input [1:0] Tnew\_W,  input jal,  output [2:0] ForwardRSD,  output [2:0] ForwardRTD,  output [2:0] ForwardRSE,  output [2:0] ForwardRTE,  output [1:0] ForwardRTM |

1. Stall

表20 暂停信号控制接口定义表

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| stall.v | input [31:0] IR\_D,  input Tuse\_RS0,  input Tuse\_RS1,  input Tuse\_RT0,  input Tuse\_RT1,  input Tuse\_RT2,  input [1:0] Tnew\_E,  input [1:0] Tnew\_M,  input [4:0] A3\_E,  input [4:0] A3\_M,  input RegWriteEn\_E,  input RegWriteEn\_M,  output UpdatePC,  output IF\_ID\_EN,  output ID\_EXE\_Clr |

Tuse为：这条指令位于D级的时候，再经过多少个时钟周期就必须要使用相应的数据。

Tnew为：位于某个流水级的某个指令，它经过多少个时钟周期可以算出结果并且存储到流水级寄存器里。

Tnew=0，说明结果已经算出，如果指令处于WB级，则可以通过寄存器的内部转发设计解决（关于内部转发请自行查阅资料思考或与同学讨论），不需要任何操作。如果指令不处于WB级，则可以通过转发结果来解决。

Tnew<=Tuse，说明需要的数据可以及时算出，可以通过转发结果来解决。（与1有何区别？）

Tnew>Tuse，说明需要的数据不能及时算出，必须暂停流水线解决。

1. 思考题
2. 为什么需要有单独的乘除法部件而不是整合进ALU？为何需要有独立的HI、LO寄存器？

CPU运算乘除法延迟时间较长，而且位数会超32。为了提升CPU效率，在进行乘法时也可以进行addu等其他操作，因此需要单独模块。

HI/LO寄存器不能被编码，在乘法超出32位范围时使用。避免在简单流水线中从高延迟指令回写寄存器文件的问题。比如当mult产生计算值后，其需要写入的有两个值，同时也有可能流水线其他指令也要写入寄存器，此时会产生冲突。而建立HI、LO寄存器，则可以避免这样的问题。

1. 参照你对延迟槽的理解，试解释“乘除槽”。

乘除槽是指在进行乘除法运算时，所有和HI/LO有关的指令全部被冻结在D，然而其他的指令可以顺利向前进行。这是为了保证HI/LO寄存器的值稳定。

1. 为何上文文末提到的lb等指令使用的数据扩展模块应在 MEM/WB 之后，而不能在 DM 之后?

放在DM之后一定会使该级延迟增加Tc（拓展器时间），放在MEM/WB之后对于一些不需要写寄存器的指令，相对于前一种情况，延时减少Tc。

1. 举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑C语言中字符串的情况）

对于C语言中字符串的情况，如果是按照字节寻址，则可以直接找到地址不是4的倍数的字符。如果按照字寻址，则还需要首先将寻找地址右移两位后细找，增加了运算时间并且更加麻烦了。

1. 如何概括你所设计的CPU的设计风格？为了对抗复杂性你采取了哪些抽象和规范手段？

Planner型，对所有指令产生的可能情况提前进行统筹规划。将暂停转发问题的讨论变成对Tnew、Tuse情况的分析。采用将指令类型进行分类，按照指令使用的寄存器类型进行分类。也可以使用宏定义。

1. 你对流水线CPU设计风格有何见解？

在指令条数多，指令类型复杂的时候，应当在暂停、转发时不细致考虑指令类型，反而转化为对Tnew、Tuse的讨论，否则代码量极大。这样做的另一个好处是测试时候只需要检测每一级之间的转发是否正确即可，（在保证Tew，Tuse运算正确的条件下）不用测试所有指令类型。但是，对于初学者，按照具体指令类型进行编程更加容易理解，但是测试时候需要测试所有的指令类型产生的冲突，工程量很大，不太容易。