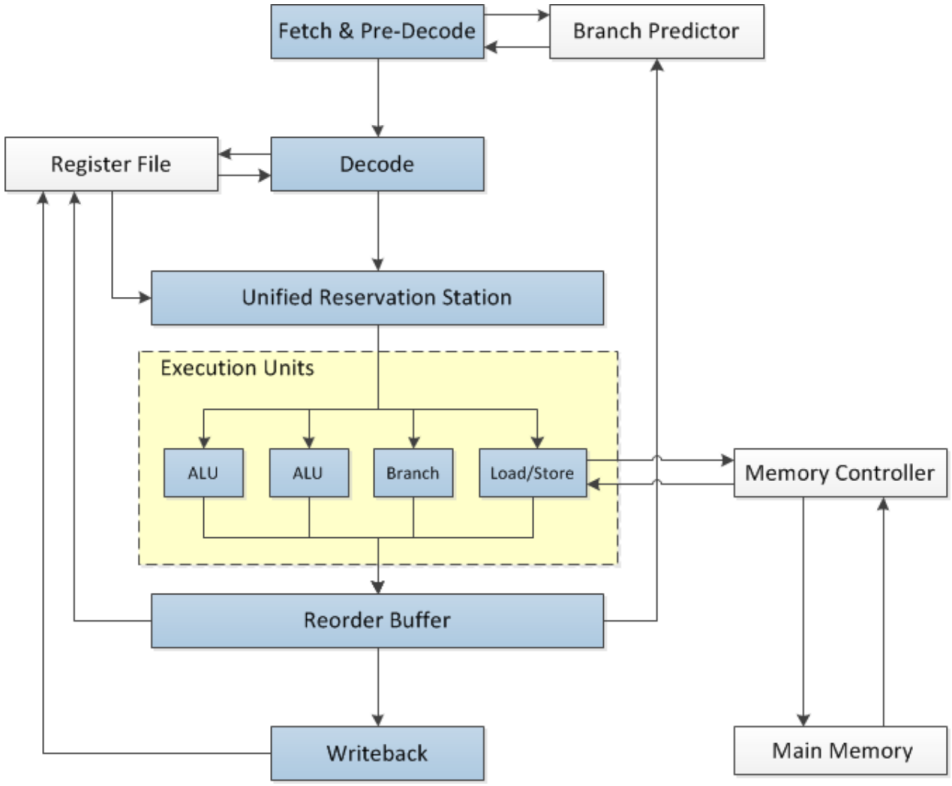
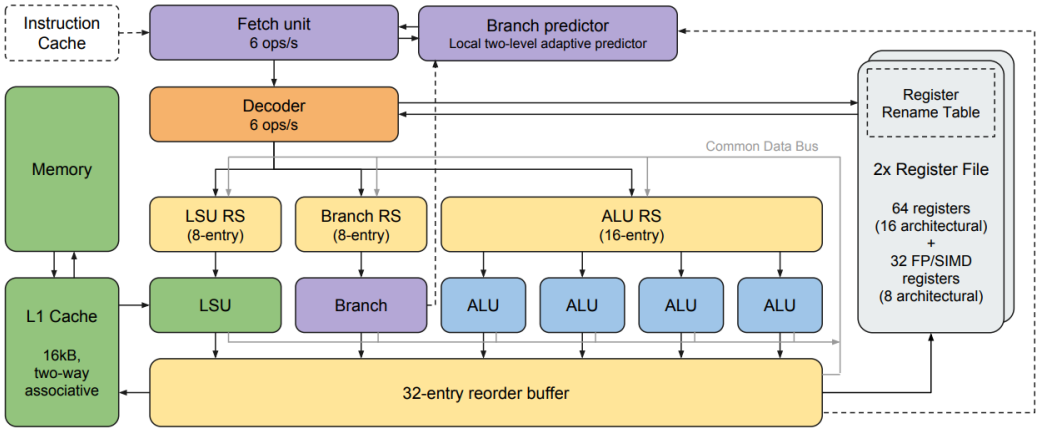
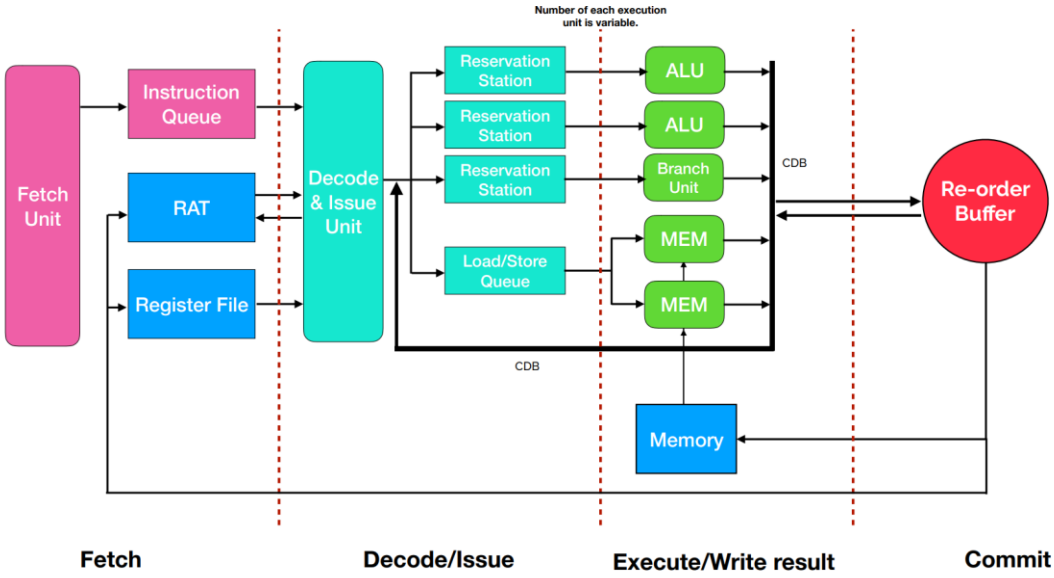
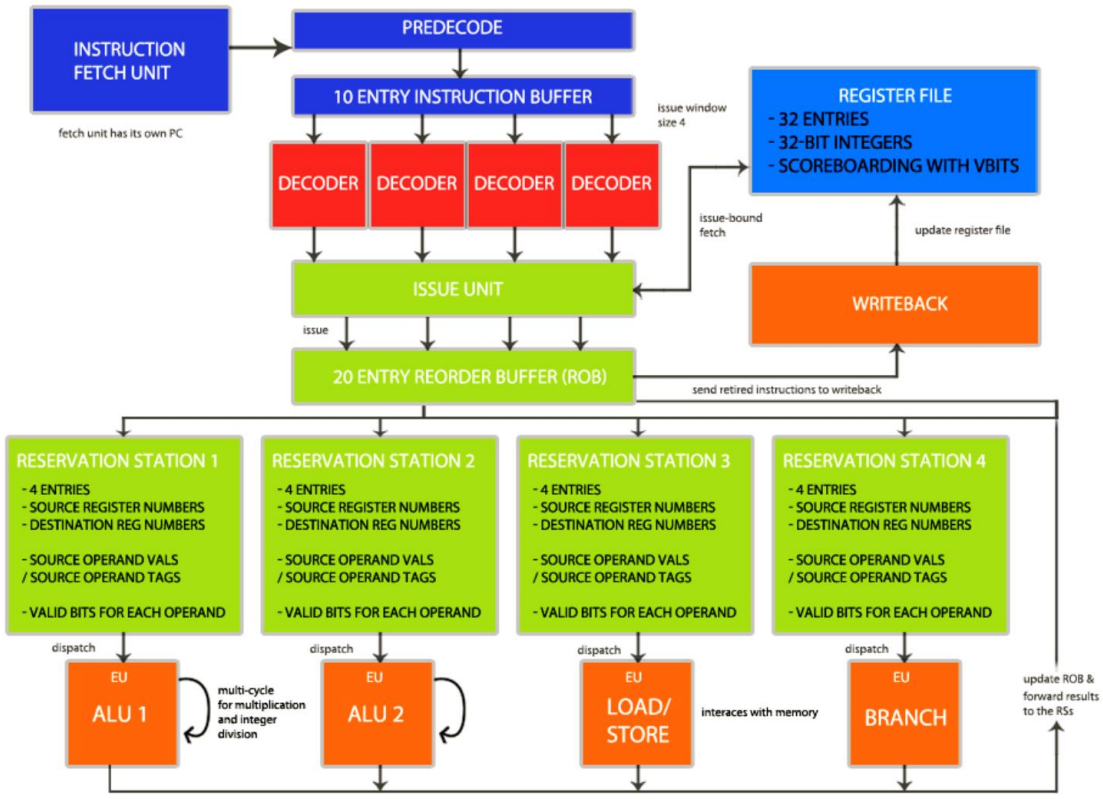
下面这些都是其他人做的:

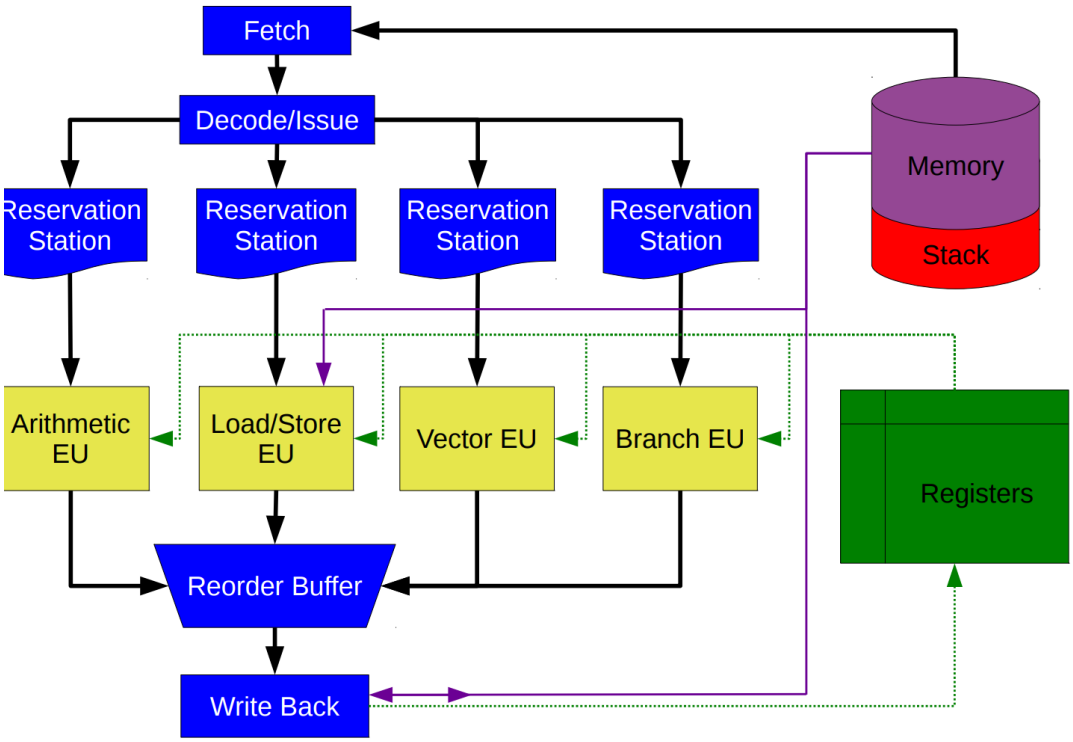


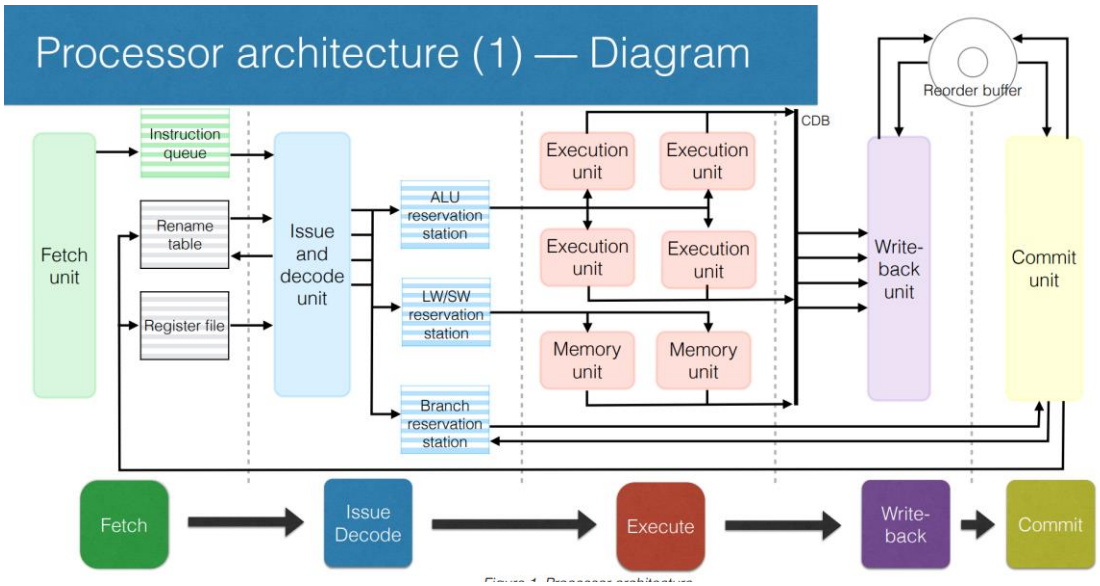



Number of each execution unit is variable.
Fetch Unit
Instruction Queue
RAT
Register File
Decode & Issue Unit
Reservation Station
Reservation Station
Reservation Station
Load/Store Queue
ALU
ALU
Branch Unit
MEM
MEM
CDB
CDB
Re-order Buffer
Memory
Fetch
Decode/Issue
Execute/Write result
Commit

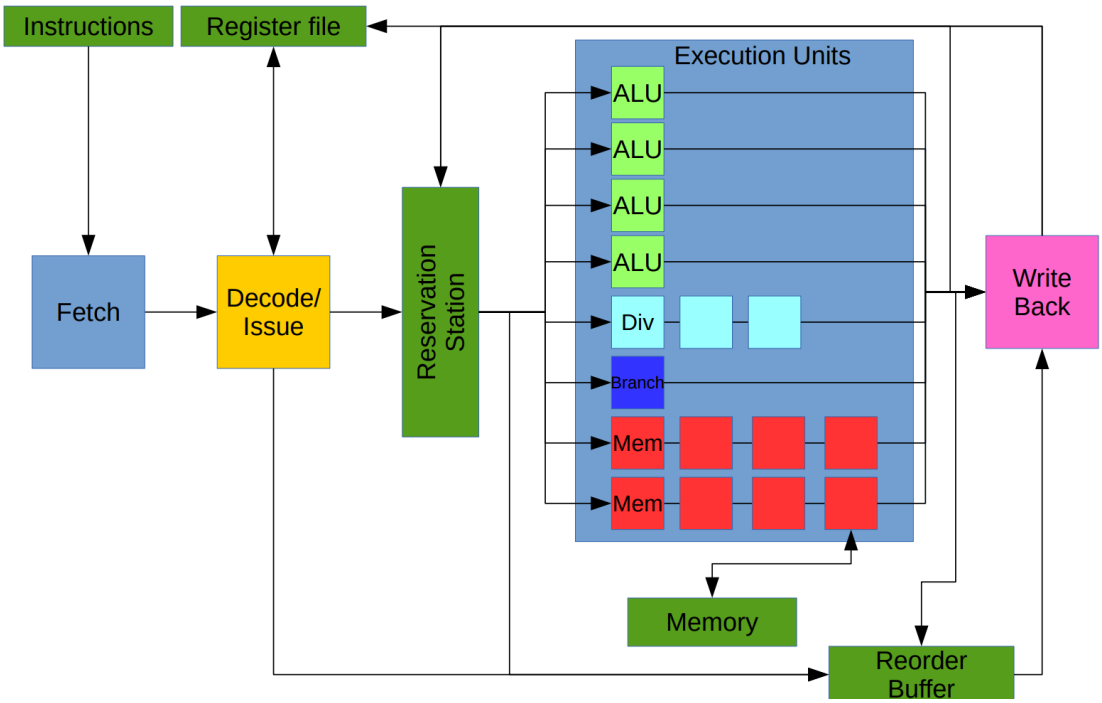

INSTRUCTION FETCH UNIT
fetch unit has its own PC
PREDECODE
10 ENTRY INSTRUCTION BUFFER
issue window size 4
DECODER
DECODER
DECODER
DECODER
REGISTER FILE
- 32 ENTRIES
- 32-BIT INTEGERS
- SCOREBOARDING WITH VBITS
issue-bound fetch
update register file
ISSUE UNIT
WRITEBACK
issue
20 ENTRY REORDER BUFFER (ROB)
send retired instructions to writeback
RESERVATION STATION 1
- 4 ENTRIES
- SOURCE REGISTER NUMBERS
- DESTINATION REG NUMBERS
- SOURCE OPERAND VALS / SOURCE OPERAND TAGS
- VALID BITS FOR EACH OPERAND
RESERVATION STATION 2
- 4 ENTRIES
- SOURCE REGISTER NUMBERS
- DESTINATION REG NUMBERS
- SOURCE OPERAND VALS / SOURCE OPERAND TAGS
- VALID BITS FOR EACH OPERAND
RESERVATION STATION 3
- 4 ENTRIES
- SOURCE REGISTER NUMBERS
- DESTINATION REG NUMBERS
- SOURCE OPERAND VALS / SOURCE OPERAND TAGS
- VALID BITS FOR EACH OPERAND
RESERVATION STATION 4
- 4 ENTRIES
- SOURCE REGISTER NUMBERS
- DESTINATION REG NUMBERS
- SOURCE OPERAND VALS / SOURCE OPERAND TAGS
- VALID BITS FOR EACH OPERAND
dispatch
dispatch
dispatch
dispatch
EU
ALU 1
multi-cycle for multiplication and integer division
EU
ALU 2
EU
LOAD/ STORE
interaces with memory
EU
BRANCH
update ROB & forward results to the RSs

Processor Architecture

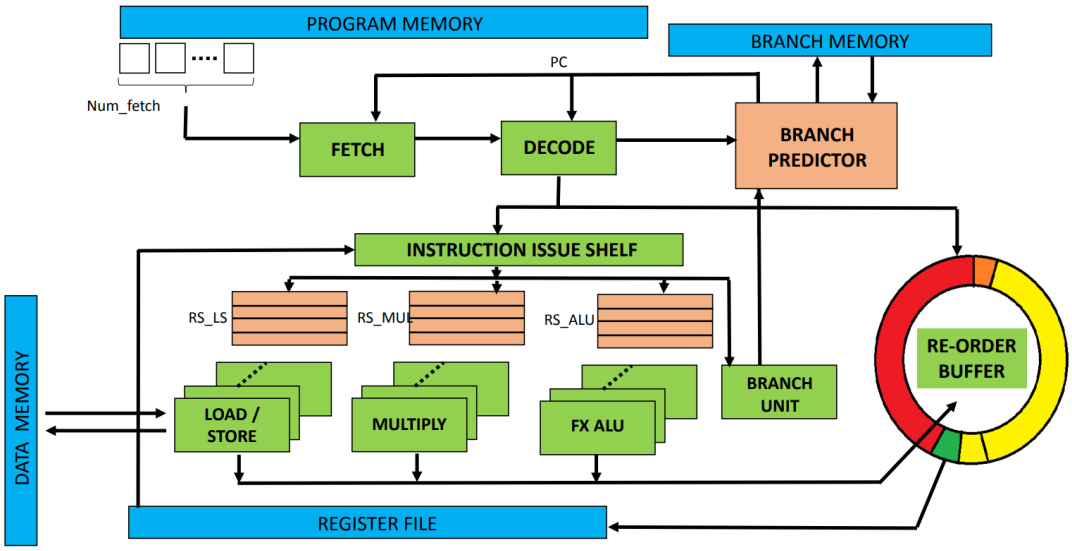




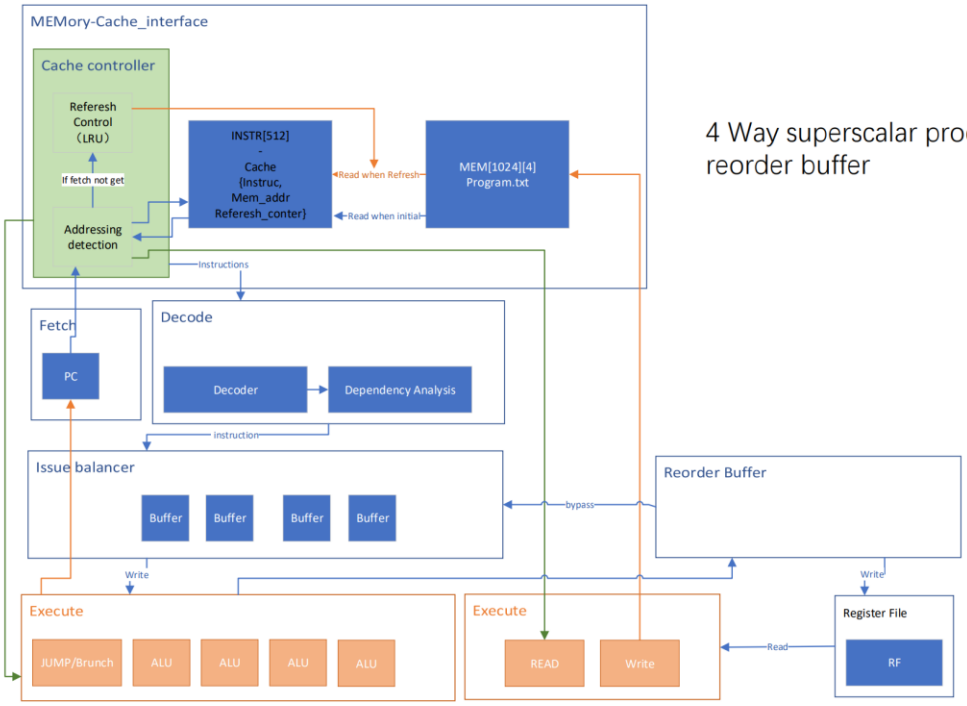
Processor Architecture



Processor Architecture



下面这个是我设计的可能不太对



这是题目要求

Stage 3 （之前的就是设计可以运行的指令集已经有了）

一旦你有了指令集和一些机器可读形式的程序，你就可以开始构建一个简单的处理器模拟器来执行它们。最好的方法是从一个简单的模拟器开始，用越来越多的功能来扩展它，在每一步之后测试它是否能正确执行你的基准程序:

1. 为一个非管道式的标量处理器编写一个模拟器，该处理器按顺序执行指令；这基本上是一个循环，它通过使用程序计数器，并酌情使用寄存器文件和（单一）ALU 来迭代获取-执行步骤。你可以假设处理器使用一个 "哈佛 "内存层次结构，数据和指令通过单独的、简单的接口访问内存：你不需要考虑缓存内存。（这一步在 notpipeline 里面已经实现了）
2. 对你的初始模拟器进行 pipeline 处理，以便对不同的 in-flight instructions 并行地进行获取-解码-执行的步骤。这要求你建立一个时钟模型来推进流水线，以及保存部分执行指令的流水线寄存器。在这个阶段，你可以忽略依赖性和危险性的问题（即假设程序员避免了这些问题）。(流水线化)

3. 复制执行阶段，使你的仿真现在有几个执行单元（例如，两个整数 ALU，一个负载存储单元和分支单元）。你不需要对执行单元的内部进行建模（例如，对加法或乘法的实际电路进行建模），并且在这个阶段可以忽略保留站，使用阻塞问题。请注意，尽管有效的第一步是假设每个执行单元有一些固定的延迟，但更合理的方法是使用子管道。

有些学生喜欢采取 "面向对象 "的方法，通过定义类来表示处理器中的各种组件（例如，register file, reservation stations 等）来进行模拟。这不是一个要求，当然也不是唯一的选择，但可以使产生的程序更有模式，更容易理解和开发。对于处理器状态中的每个组件，定义--"当前 "值（在每个仿真周期开始时），--"下一个 "值（在每个仿真周期结束时）是有用的。然后，每个组件的行为可以由一个函数来模拟，该函数在被调用时，从 "当前 "值导出 "下一个 "值。通过调用所有模拟组件行为的函数，模拟每次都在一个周期内前进。当它们都被调用后，它将 "下一个 "值复制到 "当前 "值中，然后准备重复这个过程，执行下一个仿真周期。提供一种方法，以交互的方式进行单步仿真，在每一步都在屏幕上显示所有的状态，可能会有帮助。这类似于调试器在现实生活中的处理器上的工作方式，使你能够根据程序和处理器的行为来检查和调试模拟器。你应该确保你的模拟器能输出有用的指标，比如说:

1. 执行的指令数量
2. 整个运行所需的周期数
3.每个周期的指令数（到合理的小数位数）
4. 任何其他有用的/有趣的相关信息，如正确的分支预测率等。

Stage 4

在这个阶段，你应该有一个可以执行 "有效 "程序的模拟器（在这个意义上，他们尊重任何潜在的危险），并利用多个执行单元的优势。然而，仍然缺失真正的超标量处理器的许多允许高性能的特性。下一阶段的目标是实现这些功能。这是一个特意开放的问题，因为有许多潜在的功能你可以包括在内。作为例子，你可以选择:

1.增加分支预测或预测执行的机制，
2.增加保留站和非阻塞问题，
3.增加重排序缓冲器，
4.增加寄存器重命名（使用重排序缓冲器），
5.包括一个支持矢量指令的执行单元。