# 任务

**任务：**验证这32位（0x0042a303）是否对应于RISC-V架构中的指令lw t1,4(t0)。

**0x0042a303 🡪 000000000100 00101 010 00110 0000011**

**imm11:0 = 000000000100**

**rs1 = 00101 = x5 (t0)**

**funct3 = 010**

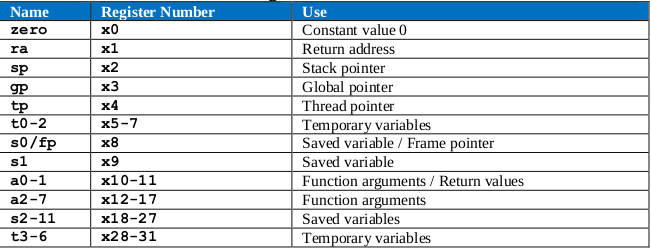
**rd = 00110 = x6 (t1)**

**op = 0000011**

来自DDCARV的附录B：





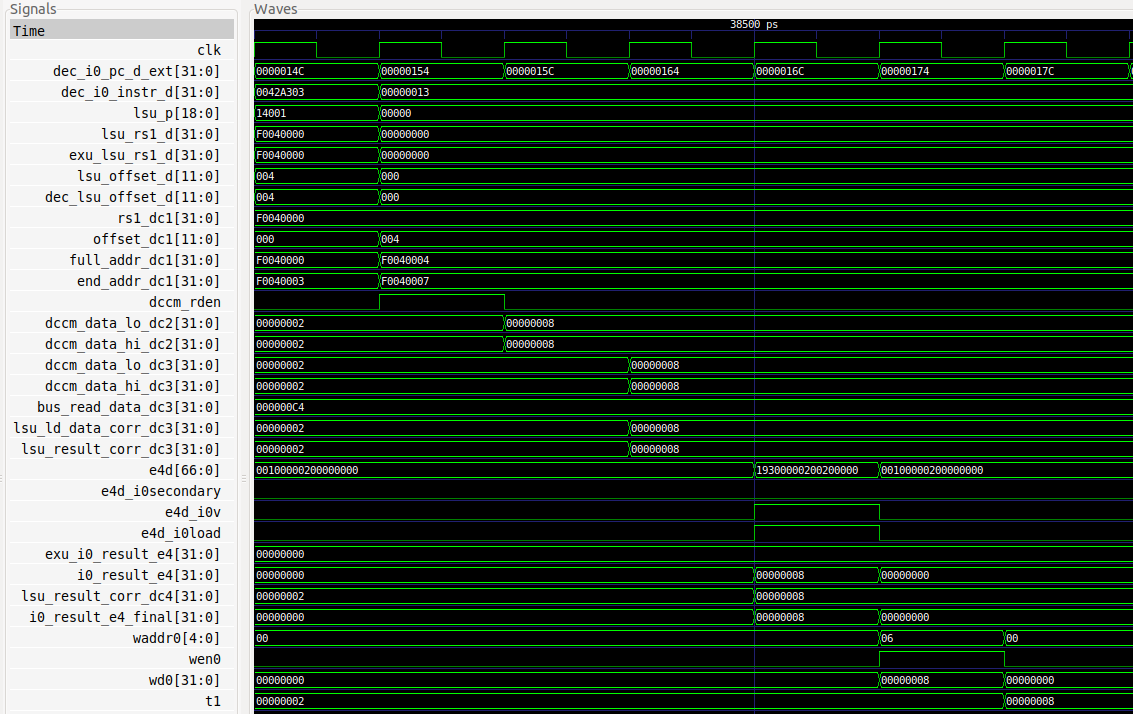


**任务：**在自己的计算机上重复图4中的仿真过程。请按照以下步骤操作（如GSG第7部分所详诉）：

* 必要时生成仿真二进制文件（*Vrvfpgasim*）。
* 在PlatformIO中，打开在以下位置提供的项目：*[RVfpgaPath]/RVfpga/Labs/Lab13/LW\_Instruction\_DCCM*。
* 在文件*platformio.ini*中更正到RVfpga仿真二进制文件（*Vrvfpgasim*）的路径。
* 使用Verilator生成仿真轨迹（生成轨迹）。
* 使用GTKWave打开轨迹。
* 使用文件*scriptLoad.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab13/LW\_Instruction\_DCCM/*中提供）打开与图4所示信号相同的信号。为此，在GTKWave上，单击“*File → Read Tcl Script File*”（文件 → 读取Tcl脚本文件）并选择*scriptLoad.tcl*文件。
* 单击几次“*Zoom In*”（放大）（）移动至18600 ps。

解答请参见实验13的主文档。

**任务：**扩展图4中的仿真以包含图6所示的信号（在下文说明）。



**任务：**在SweRV EH1处理器的Verilog文件中找到图6中的结构和信号。

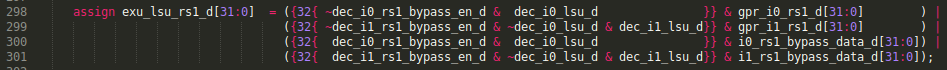
不提供解答。

**任务：**在图4的仿真中包含信号*lsu\_p*并根据上述说明分析其各个位。

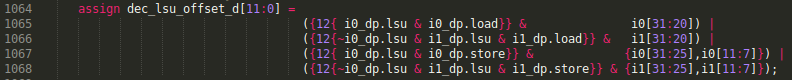
请参见上面的仿真。可以看到，当装载进行译码时，lsu\_p = 0x14001：

* valid = 1。指令有效。
* load = 1。指令为装载指令。
* word = 1。访问的大小是字。

**任务：**在Verilog代码中从LSU的两个输入（exu\_lsu\_rs1\_d和dec\_lsu\_offset\_d）的获取来源分析这两个输入所遵循的路径。此过程涉及几个模块：**dec**、**exu**和**lsu**。为其他指令分析这些信号的行为。



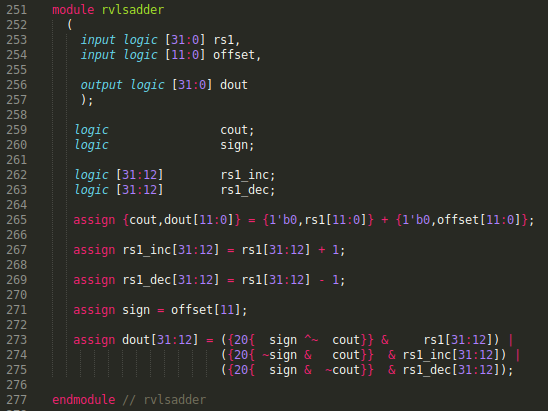
基址可以来自寄存器文件或旁路（来自通路0或通路1）。



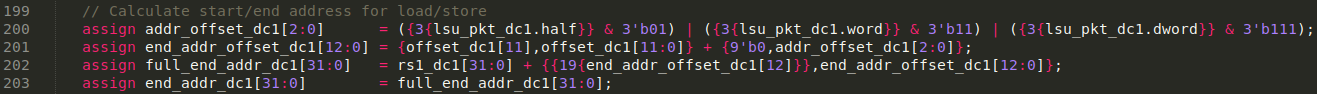
偏移量来自通路0或通路1指令的32位。

**任务：**分析DC1阶段中两个加法器的实现，这两个加法器在模块**lsu\_lsc\_ctl**中实例化。我们通过通过展示这些加法器的实现在下面的图7中提供指导。

**文件*beh\_lib.sv*：**

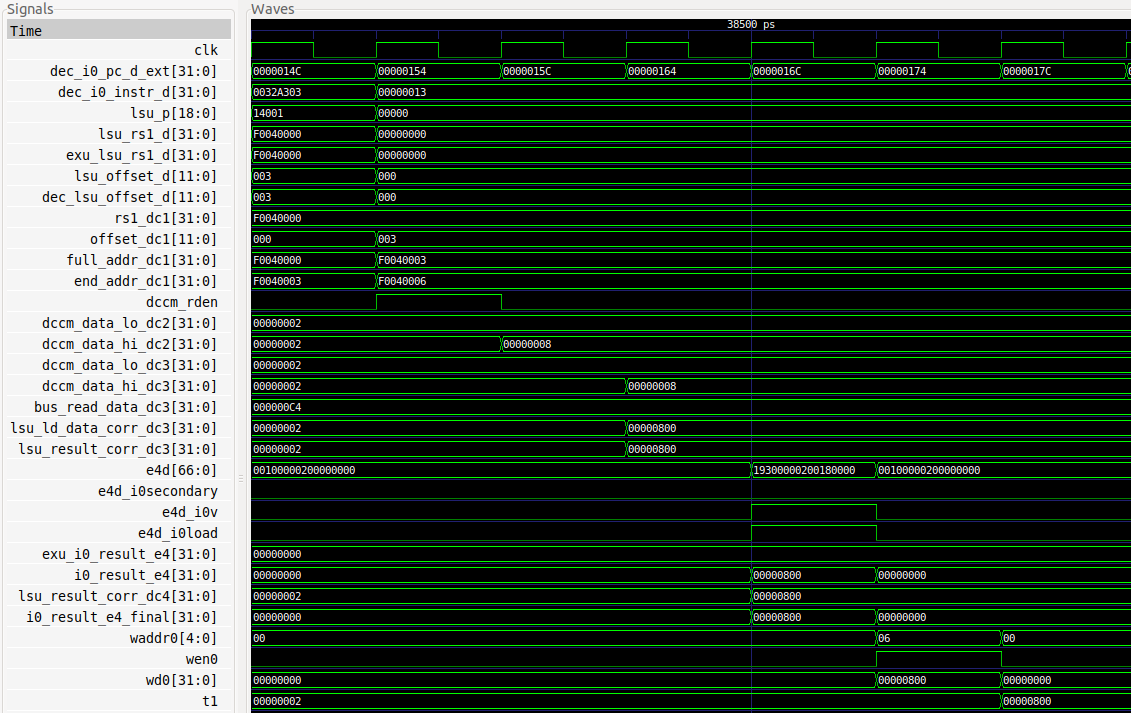


**文件*lsu\_lsc\_ctl.sv*：**



**任务：**在图2的程序中，尝试不同的访问大小（字节和半字）和未对齐访问。为此，请更改偏移量或将访问类型从lw更改为lb（*装载字节*）或lh（*装载半字*）。例如，如果将偏移量从4更改为3，则装载字指令将对从地址0xF0040003开始的32位执行未对齐访问，如图8所示。分析上述不同情况下信号lsu\_addr\_dc1[31:0]（或full\_addr\_dc1[31:0]）和end\_addr\_dc1[31:0]的值。

在实验20中，我们将从DCCM的内部分析这种情况。



信号lsu\_addr\_dc1[31:0]和end\_addr\_dc1[31:0]的值将访问的起始地址和结束地址传达给存储器：0xF0040003和0xF0040007。读取两个字（0x00000002和0x00000008），并在对齐器中提取最后一个字（0x00000800）。

**任务：**在图2的程序中，当对地址0xF0040004和地址0xF0040003执行lw时，比较信号dccm\_data\_lo\_dc2[31:0]和dccm\_data\_hi\_dc2[31:0]的值。

上文有两个仿真。

* 指向地址0xF0040004的lw

dccm\_data\_lo\_dc2[31:0]：0x00000008

dccm\_data\_hi\_dc2[31:0]：0x00000008

两个信号都包含从请求地址读取的值。

* 指向地址0xF0040003的lw

dccm\_data\_lo\_dc2[31:0]：0x00000002（来自地址0xF0040000的值）

dccm\_data\_hi\_dc2[31:0]：0x00000008（来自地址0xF0040004的值）

**任务：**分析**lsu\_dccm\_ctl**和**lsu\_ecc**模块中的Verilog代码中使用的对齐、合并和错误检查逻辑。

不提供解答。

**任务：**在图2的程序中，当对地址0xF0040004和地址0xF0040003执行lw时，比较信号lsu\_result\_corr\_dc3[31:0]的值。

上文有两个仿真。

* 指向地址0xF0040004的lw

lsu\_result\_corr\_dc3[31:0]：0x00000008

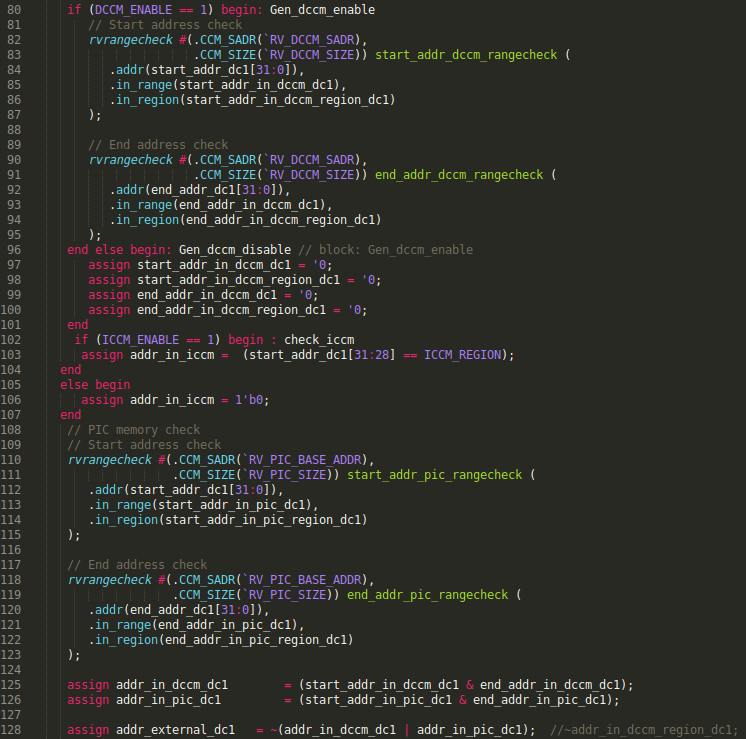
它包含从请求地址读取的值。

* 指向地址0xF0040003的lw

lsu\_result\_corr\_dc3[31:0]：0x00000800

它包含从请求地址读取的值。需考虑RISC-V采用小端模式。

**任务：**在Verilog代码中分析信号addr\_external\_dc1如何于DC1阶段在模块**lsu\_addrcheck**中计算。



模块**rvrangecheck**用于检查请求地址：

* 如果它处于DCCM/ICCM地址范围中（第80-107行），在这种情况下，信号addr\_in\_dccm\_dc1 = 1
* 如果它处于PIC地址范围中（第108-123行），在这种情况下，信号addr\_in\_pic\_dc1 = 1
* 如果它不处于上述任一地址范围中，则处于DDR外部存储器中，在这种情况下：addr\_external\_dc1 = 1

**任务：**验证这32位（0x0062a023）是否对应于RISC-V架构中的指令sw t1,0(t0)。

**0x0062a023 🡪 0000000 00110 00101 010 00000 0100011**

**imm11:0 = 000000000000**

**rs2 = 00110 = x6 (t1)**

**rs1 = 00101 = x5 (t0)**

**funct3 = 010**

**op = 0100011**

来自DDCARV的附录B：

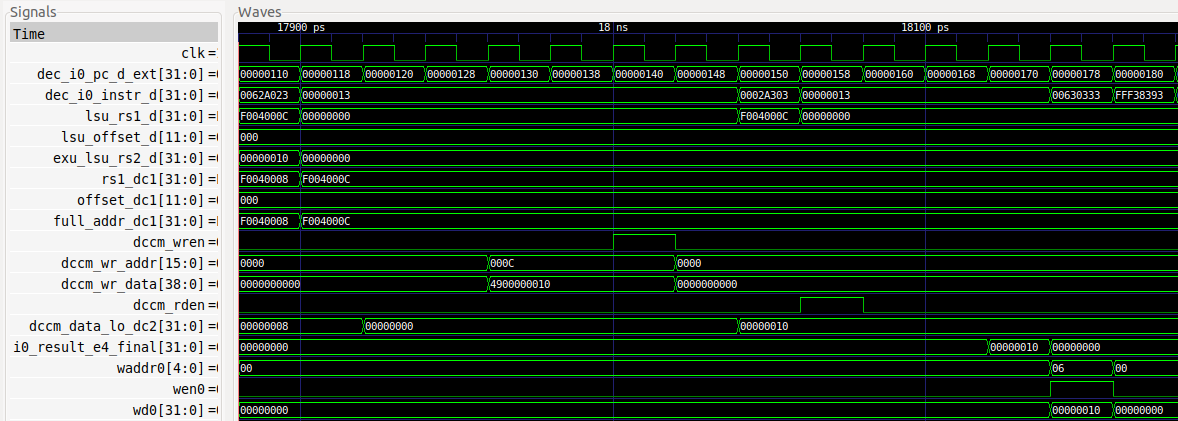


**任务：**在自己的计算机上重复图12中的仿真过程。请按照以下步骤操作（如GSG第7部分所详诉）：

* 必要时生成仿真二进制文件（*Vrvfpgasim*）。
* 在PlatformIO中打开在以下位置提供的项目：*[RVfpgaPath]/RVfpga/Labs/Lab13/SW\_Instruction\_DCCM*。
* 在文件*platformio.ini*中更新到RVfpga仿真二进制文件（*Vrvfpgasim*）的路径。
* 使用Verilator生成仿真轨迹（生成轨迹）。
* 在GTKWave上打开轨迹。
* 使用文件*scriptStore.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab13/SW\_Instruction\_DCCM/*中提供）显示与图4所示信号相同的信号。为此，在GTKWave上，单击“*File → Read Tcl Script File*”（文件 → 读取Tcl脚本文件）并选择*scriptStore.tcl*文件。
* 单击几次“*Zoom In*”（放大）（）移动至17900 ps。

解答请参见实验13的主文档。

**任务：**在仿真中分析存储指令之后的装载指令，以验证值是否已正确写入DCCM。需要添加图4和图6中的一些信号来分析装载。



**任务：**按照与第2.B部分中对lw指令执行的高级分析类似的方式扩展本部分中对sw指令执行的基础分析。

不提供解答。

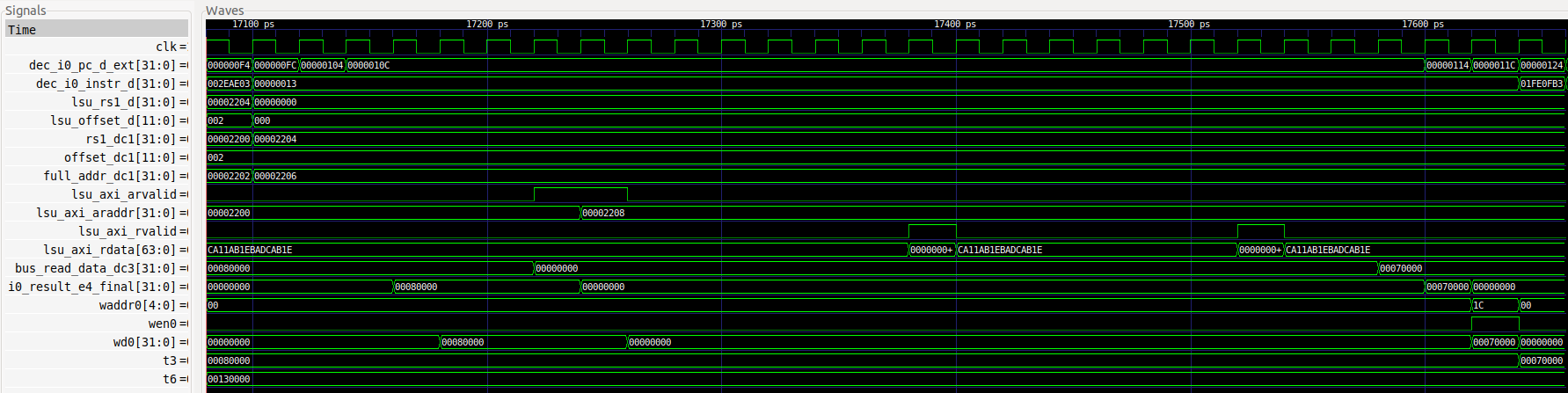
**任务：**分析针对DCCM的未对齐存储以及子字存储：存储字节（sb）或*存储半字*（sh）。

不提供解答。

**任务：**在自己的计算机上重复图17中的仿真过程。使用文件*test\_Blocking.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab13/LW\_Instruction\_ExtMemory*中提供）。单击几次“*Zoom In*”（放大）（）移动至16940 ps。

解答请参见实验13的主文档。

**任务：**修改图15中的程序以分析需要通过AXI总线向外部存储器发送两个地址的未对齐装载访问。

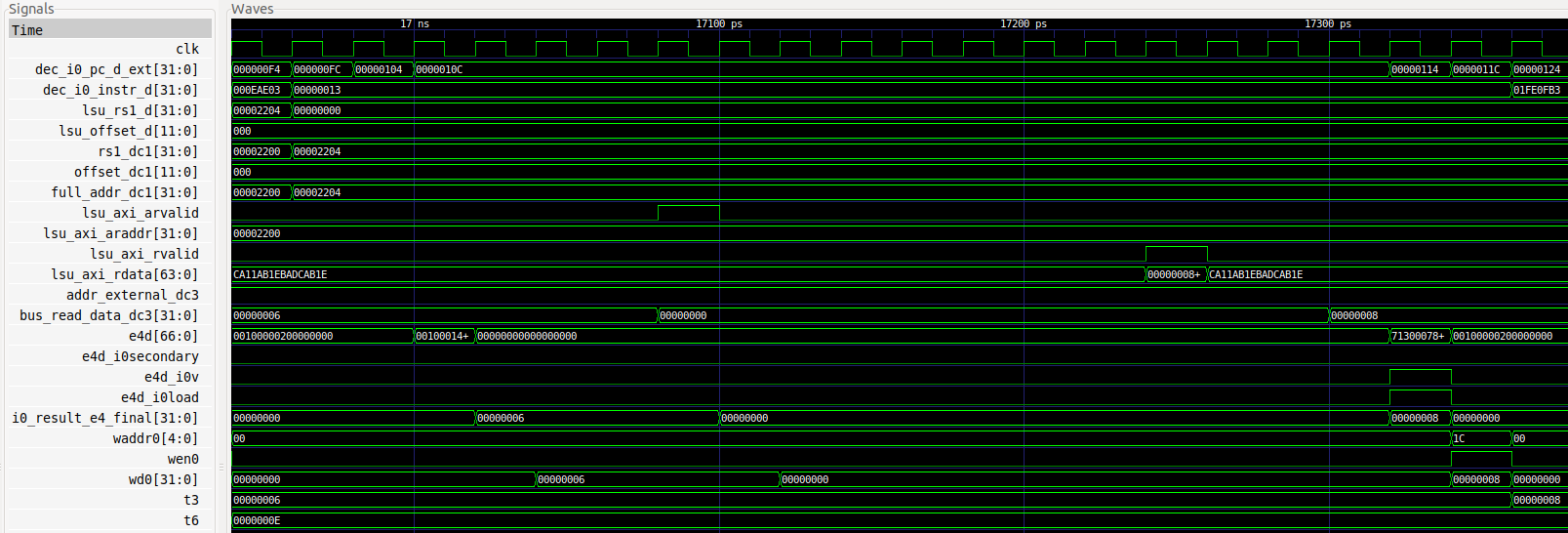


**任务**：将控制多路开关的信号添加到仿真中（在图16的DC3和提交阶段），其中多路开关选择由DDR外部存储器提供的数据。可以在Verilog代码的以下几行中找到这些多路开关：

- 2:1多路开关：模块**lsu\_lsc\_ctl**的第264行。

- 3:1多路开关：模块**dec\_decode\_ctl**的第2277行。

可以使用的*.tcl*文件位于：*[RVfpgaPath]/RVfpga/Labs/Lab13/LW\_Instruction\_ExtMemory/test\_Blocking\_Extended.tcl*



**任务：**分析用于访问DRAM控制器的AXI总线实现也很有趣，为此可以检查**lsu\_bus\_intf**模块。

不提供解答。

**任务：**在自己的计算机上重复图18中的仿真过程。使用文件*scriptStoreBuffer.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab13/SW\_Instruction\_DCCM*中提供）。单击几次“*Zoom In*”（放大）（）移动至17900 ps。

解答请参见实验13的主文档。

**任务：**修改图11中的程序以实现两个出色的存储操作，并执行与图18中的分析类似的分析。

不提供解答。