



Imagination大学计划

# RVfpga实验17

## 超标量执行

## 1. 简介

如上文所述，Western Digital的SweRV EH1处理器是32位内核，采用9级流水线和双路超标量设计。在实验11-13中，我们分析了通过SweRV处理器的基本指令流以及每个流水线阶段的详情，在实验14-16中，我们介绍了如何在该处理器中处理数据、控制和结构相关性。了解上述基础知识后，就可以分析超标量执行了！

**注：**开始本实验前，建议您先阅读S.Harris和D.Harris所著的《数字设计和计算机体系结构》（RISC-V版本，Morgan Kaufmann出版，简称[DDCARV]）的第7.7.4节。本实验的一些内容受这本书的启发。

超标量执行是一种能够提高处理器性能的微架构技术。超标量处理器包含数据路径硬件的多个副本，可同时执行多条指令。执行单条指令的延时并无变化，但处理器可以在每个周期执行和提交更多的指令，从而提高吞吐量。图1所示为[DDCARV]中提及的双路超标量处理器的示例框图。

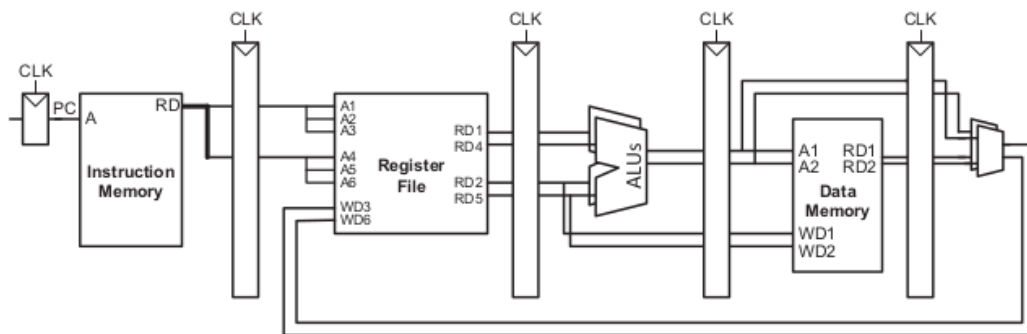


图1. [DDCARV]中的图7.68：双路超标量处理器框图

SweRV EH1是一款类似于图1所示处理器的双路超标量处理器，每个周期最多可以取指、执行和提交两条指令。其数据路径可以一次性对齐两条来自指令存储器的指令。多端口寄存器文件最多可以读取四个源操作数，并在每个周期中写回两个值（加上一个来自非阻塞装载的值，如实验15的分析所示）。SweRV EH1处理器包括两条整数管道、一条乘法管道、一条装载-存储管道和一个非流水线除法器。所有这些管道都是完全独立的，因此处理器可以同时执行一对独立的算术逻辑（AL）指令或任意两条独立的不同指令。但正如实验14中所述，处理器中只有一条乘法管道、一条除法管道和一条装载/存储管道，因此不能在同一周期内执行一对乘法、除法或装载/存储指令，两个顺序相同的非AL指令将引发结构冒险。

除了非阻塞装载的情况之外，SweRV EH1不支持无序执行的动态指令调度。但是，可以静态调整代码顺序，以便更好地利用包括流水线两条通路在内的各种资源。理想情况下，双路超标量处理器（如SweRV EH1）的吞吐量（IPC）应为单发射设计的两倍。遗憾的是，实际程序通常无法达到理想状态：在实际的程序中，从单路处理器升级为双路处理器时，性能通常会提高1.3至1.5倍；但添加第二条通路需要使用更多的硬件。

在第2部分，我们将分析两个简单的程序，比较使用SweRV EH1单发射和双发射配置时的行为。然后，在第3部分，我们会提供几个与超标量执行相关的练习。

## 2. 单发射与双发射

在本部分中，我们使用两个简单的程序：第一个程序（第2.A部分）包含一个由四条AL指令组成的循环，第二个程序（第2.B节）包含一个由两条乘法指令和两条AL指令交错而成的循环。

### A. 四条独立的AL指令

在本部分中，我们将对比图2中的程序在单发射和双发射SweRV EH1内核上运行时的性能。回想一下，在实验11的附录B中，我们介绍了如何配置不同的内核功能（流水线执行、分支预测、超标量等）。

本程序包含一个执行1,000,000（0xF4240）次迭代的循环；循环主体包含四条独立的AL指令（add、sub、or和xor），并由nop指令包围，以便用户能够单独查看每次迭代。文件夹 *[RVfpgaPath]/RVfpga/Labs/Lab17/Four\_AL\_Instructions* 中提供PlatformIO项目，可根据需要对其进行分析、仿真和修改。

```
.globl Test_Assembly

.text

Test_Assembly:

li t2, 0x400          # Disable Dual-Issue Execution
csrrs t1, 0x7F9, t2

li t0, 0x0
li t1, 0x1
li t2, 0x1
li t3, 0x3
li t4, 0x4
li t5, 0x5
li t6, 0x6

lui t2, 0xF4
add t2, t2, 0x240

REPEAT:
    add t0, t0, 1
    INSERT_NOPS_10
    INSERT_NOPS_4
    add t3, t3, t1
    sub t4, t4, t1
    or  t5, t5, t1
    xor t6, t6, t1
    INSERT_NOPS_10
    INSERT_NOPS_3
    bne t0, t2, REPEAT # Repeat the loop

.end
```

图2. 包含四条AL指令的程序

在第2.A.i部分，我们将根据Verilator中的仿真结果以及Nexys A7开发板上的执行结果，分析图2中的程序在单发射SweRV EH1处理器中的执行情况。为此，我们在程序开始时使用以下两条指令来禁止双发射功能：

```
li t2, 0x400
csrrs t1, 0x7F9, t2
```

在第2.A.ii部分，我们将根据Verilator中的仿真结果以及Nexys A7开发板上的执行结果，分析图2中的程序在双发射SweRV EH1处理器中的执行情况。为此，只需注释掉先前使用的两条指令。

### i. 单发射SweRV EH1处理器中的执行情况

在单发射配置中，SweRV EH1会直接忽略第二条通路，在每个周期只执行一条指令。图3所示为图2中的程序在此种SweRV EH1配置下的仿真结果。我们随机选取了REPEAT循环中的一次迭代（非第一次迭代，假定每次迭代均相同）。

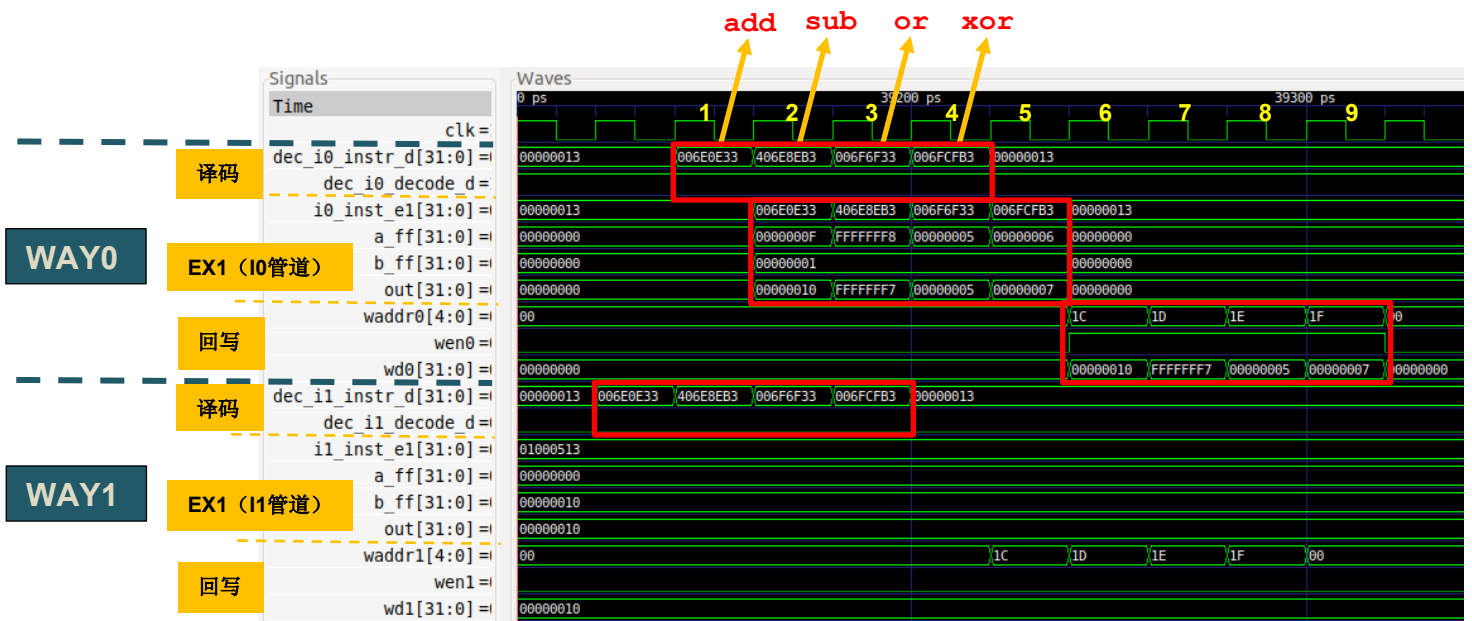


图3. 图2中的程序在单发射SweRV EH1处理器上的仿真结果

译码时，两条通路都会接收指令（参见信号dec\_i0\_instr\_d[31:0]和dec\_i1\_instr\_d[31:0]），但仅通路0会将指令发送至执行阶段，因为通路1的发送功能已禁止。信号dec\_i0\_decode\_d和dec\_i1\_decode\_d用于确定指令是否会从译码阶段传播到EX1阶段，信号i0\_inst\_e1[31:0]和i1\_inst\_e1[31:0]分别包含E1阶段的通路0和通路1中的指令。

#### - 通路0:

- 在我们的示例中，信号dec\_i0\_decode\_d始终为1；具体地说，对于我们所分析的四条AL指令，其值均为1。

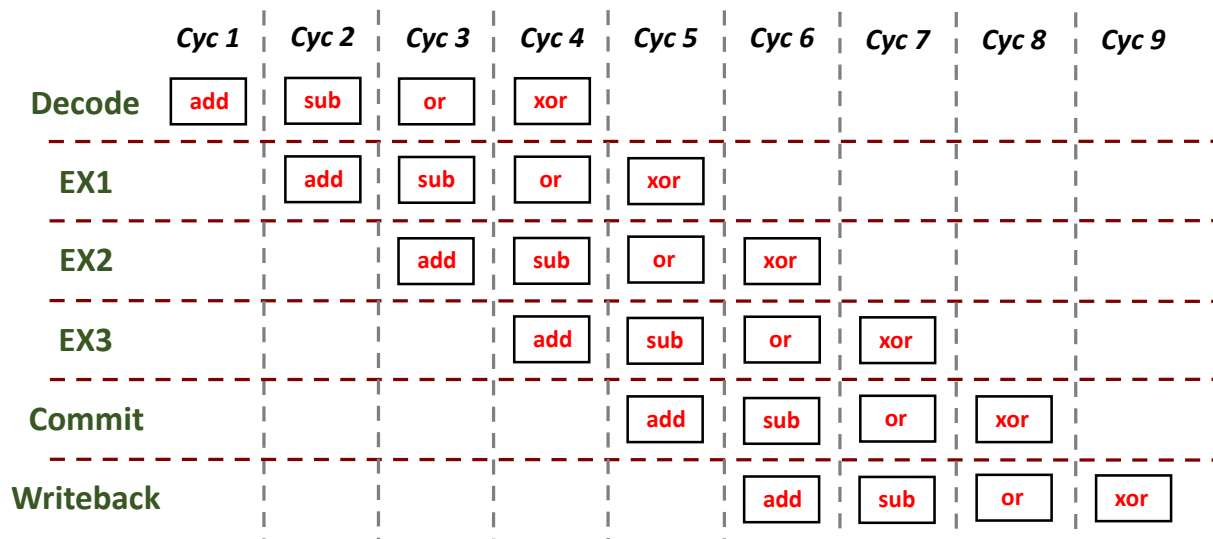
- 译码阶段的指令 (`dec_i0_instr_d[31:0]`) 会传播到I0管道 (`i0_inst_e1[31:0]`)

**- 通路1:**

- 在我们的示例中，信号`dec_i1_decode_d`始终为0；具体地说，对于我们分析的四条AL指令，其值均为0。
- 译码阶段的指令 (`dec_i1_instr_d[31:0]`) 不会传播到 (`i1_inst_e1[31:0]`) 执行阶段。

因此，系统仅使用来自I0管道的ALU（参见两条通路中的信号`aff`、`bff`和`out`），并且仅使用寄存器文件的写端口0（参见两条通路中的信号`waddr`、`wen`和`wd`）。

图4所示为I0管道中从译码阶段到回写（EX5）阶段的四条AL指令流图中展示了图3中指定的9个周期（1至9）。留空的方格对应于四条AL指令周围的nop指令，为简单起见，图中未显示这些指令。



**图4. I0管道中的4条AL指令流**

**任务：**在图3的仿真中加入跟踪信号，并仿照图4的形式高亮显示流水线中从译码阶段到回写阶段的指令流。可以使用以下位置提供的.tcl文件：

`[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions/test_task1.tcl`。

**任务：**删除图2所示的循环主体中的所有nop指令。

重复图3中的仿真。该程序的预期IPC是多少？

在开发板上执行程序，验证获得的IPC是否符合您的预期。

## ii. 双路超标量SweRV EH1中的执行情况

注释掉图2中的两条指令，以允许SweRV EH1的双发射功能。现在，只要符合条件，SweRV EH1将在每个周期向执行阶段发送两条指令。图5所示为该程序的仿真结果。与之前一样，我们随机选取了*REPEAT*循环中的一次迭代（非第一次迭代，假定每次迭代均相同）。

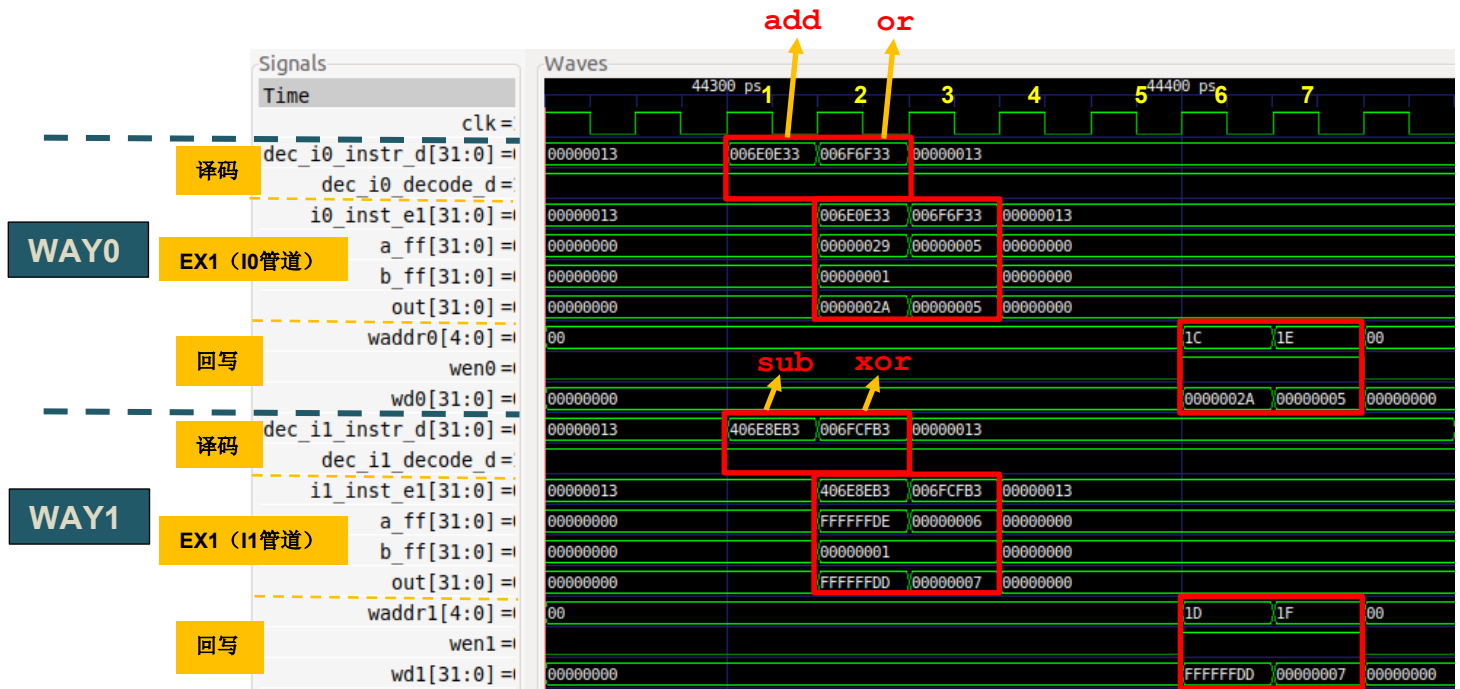


图5. 图2中的程序在双发射SweRV EH1处理器上的仿真结果

译码时，每条通路各接收一条指令（参见信号`dec_i0_instr_d[31:0]`和`dec_i1_instr_d[31:0]`），并且在每个周期，I0管道和I1管道各会向执行阶段发送一条指令。

- 通路0:
  - o 对于示例中的四条AL指令中的两条，信号`dec_i0_decode_d`始终为1（另外两条在通路1中进行译码）。
  - o 译码阶段的指令（`dec_i0_instr_d[31:0]`）会传播到I0管道（`i0_inst_e1[31:0]`）。
- 通路1:
  - o 对于示例中的四条AL指令中的两条，信号`dec_i1_decode_d`始终为1（另外两条在通路0中进行译码）。
  - o 译码阶段的指令（`dec_i1_instr_d[31:0]`）会传播到I1管道（`i1_inst_e1[31:0]`）。

因此，系统会使用两条管道（I0和I1）中的ALU（参见两条通路中的信号`a_ff`、`b_ff`和`out`），并且会使用两个寄存器文件写端口（参见两条通路中的信号`waddr`、`wen`和`wd`）。

图6所示为I0和I1管道中从译码阶段到EX5阶段的四条AL指令流。图中展示了图5中指定的7个周期（1至7）。留空的方格对应于四条AL指令周围的nop指令，为简单起见，图中未显示这些指令。

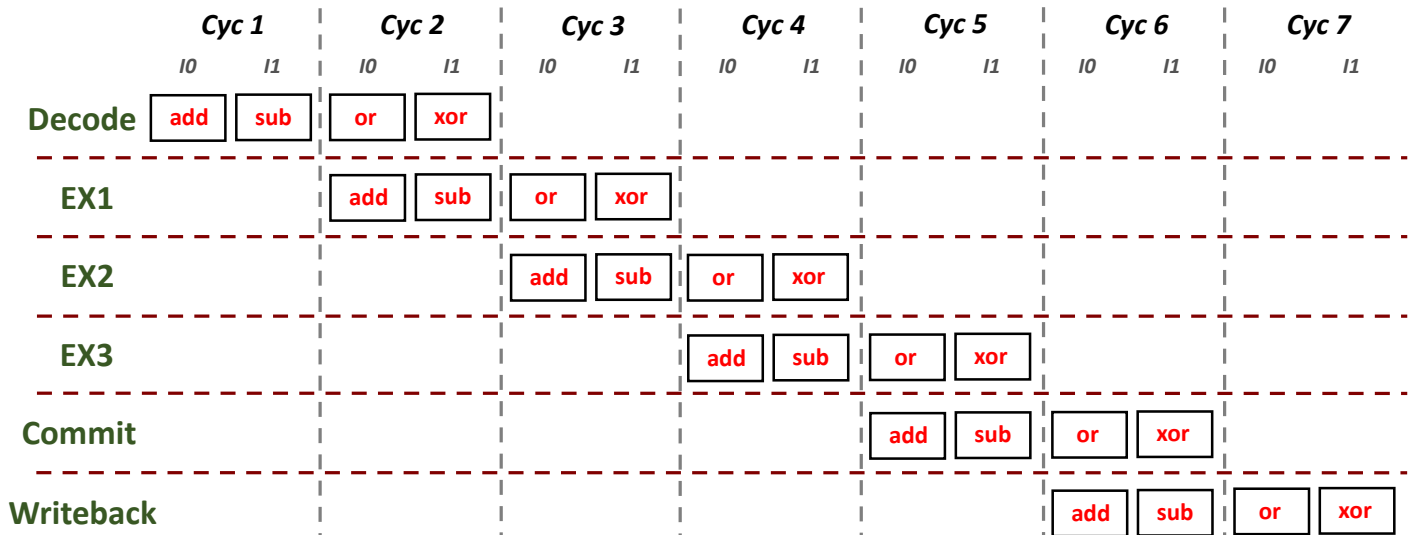


图6. 两条管道（I0和I1）中的4条指令流

**任务：** 在图5的仿真中加入跟踪信号，并仿照图6的形式高亮显示流水线中从译码阶段到回写阶段的指令流。可以使用以下位置提供的.tcl文件：

[RVfpgaPath]/RVfpga/Labs/Lab17/Four\_AL\_Instructions/test\_task2.tcl。

**任务：** 删除图2所示的循环主体中的所有nop指令。

重复图5中的仿真。该程序的预期IPC是多少？

在开发板上执行程序，验证获得的IPC是否符合您的预期。

## B. 互相交错的两条mul指令和两条AL指令

在本部分中，我们将分析图7中的程序在双发射SweRV EH1内核上时的执行情况。本程序包含一个执行1,000,000次迭代的循环；循环主体包含互相交错的两条mul指令和两条AL指令（mul、add、mul和sub），并由nop指令包围，以使用户能够单独查看每次迭代。这四条指令互相独立。我们将在Verilator中进行仿真分析，突出显示主要的信号，并对每种情况下的程序行为进行简要解释。文件夹[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL\_TwoMUL\_Instructions中提供PlatformIO项目，可根据需要对其进行分析、仿真和修改。

```
.globl Test_Assembly

.text

Test_Assembly:
```



```
# li t2, 0x400          # Disable Dual-Issue Execution
# csrrs t1, 0x7F9, t2

li t3, 0x3
li t4, 0x4
li t5, 0x5
li t6, 0x6
li t0, 0x0
lui t1, 0xF4
add t1, t1, 0x240

REPEAT:
    add t0, t0, 1
    INSERT_NOPS_10
    INSERT_NOPS_4
    mul t3, t3, t1
    add t4, t4, t1
    mul t5, t5, t1
    sub t6, t6, t1
    INSERT_NOPS_10
    INSERT_NOPS_3
    bne t0, t1, REPEAT # Repeat the loop

.end
```

图7. 包含2条mul指令和2条AL指令的程序

在此程序中，处理器每个周期将向执行阶段发送一条mul指令和一条AL指令，因此会使用乘法管道以及I0（或I1）管道。图8所示为图7中的程序在双路超标量SweRV EH1处理器上的仿真结果。我们随机选取了REPEAT循环中的一次迭代（非第一次迭代，假定每次迭代均相同）。

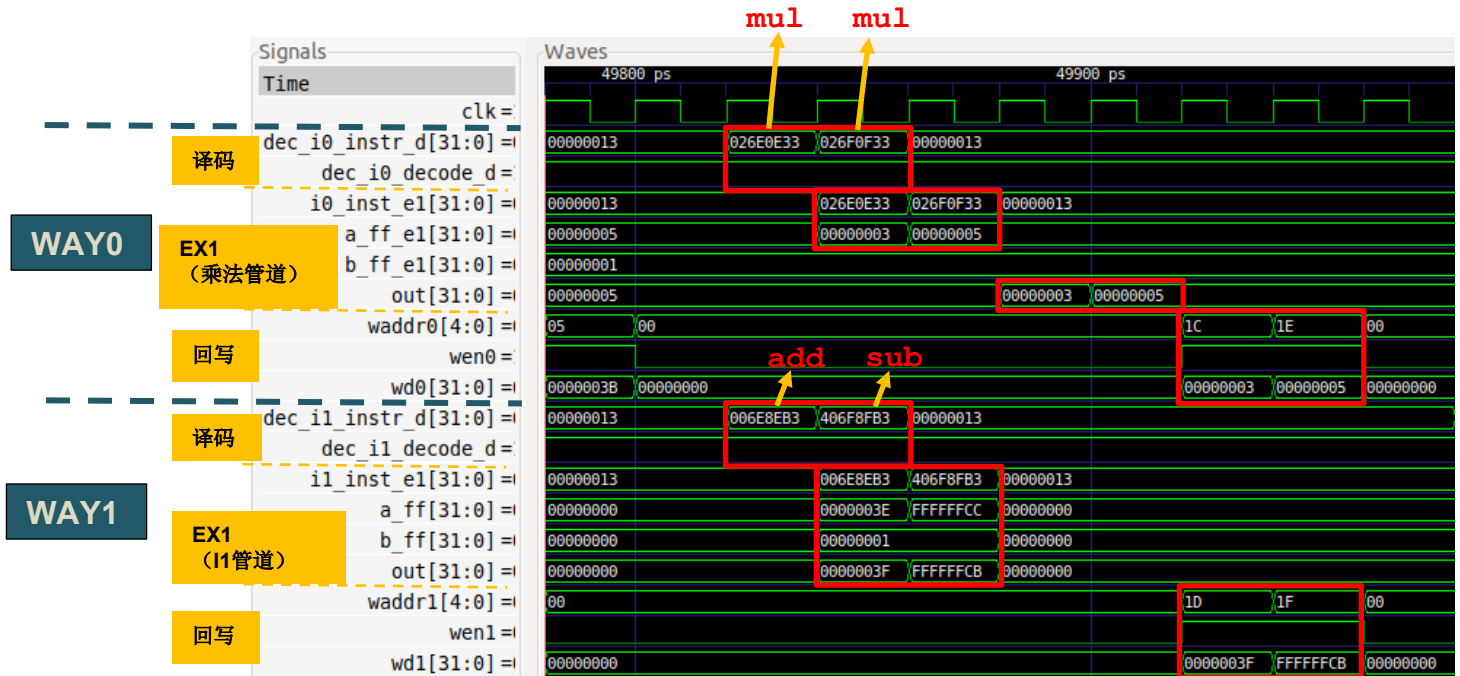


图8. 图7中程序的仿真结果



译码时，两条通路都会接收指令（参见信号`dec_i0_instr_d[31:0]`和`dec_i1_instr_d[31:0]`）并将指令发送至执行阶段。

**- 通路0:**

- 对于示例中所分析的四条指令中的两条，信号`dec_i0_decode_d`始终为1（另外两条在通路1中进行译码）。
- 译码阶段的指令（`dec_i0_instr_d[31:0]`）会传播到乘法管道（`i0_inst_e1[31:0]`）

**- 通路1:**

- 对于示例中所分析的四条指令中的两条，信号`dec_i1_decode_d`始终为1（另外两条在通路1中进行译码）。
- 译码阶段的指令（`dec_i1_instr_d[31:0]`）会传播到I1管道（`i1_inst_e1[31:0]`）

因此，系统会使用I1管道和乘法器中的ALU（参见信号`a_ff_e1`、`b_ff_e1`和`out`，以及信号`a_ff`、`b_ff`和`out`），并且会使用两个寄存器文件写端口（参见两条通路中的信号`waddr`、`wen`和`wd`）。

**任务：**在图8的仿真中加入跟踪信号，并高亮显示流水线中从译码阶段到回写阶段的指令流。可以使用以下位置提供的.tcl文件：

`[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL_TwoMUL_Instructions /test_taskMuls.tcl`。

**任务：**删除图7所示的循环主体中的所有nop指令。

重复图8中的仿真。该程序的预期IPC是多少？

在开发板上执行程序，验证获得的IPC是否符合您的预期。

使用单发射配置重复实验，并对比实验结果。

### 3. 练习

1) 使用能够展示双发射执行相关新情境的指令组合，创建与图2和图7所示程序类似的程序。

2) 分析（双发射）SweRV EH1处理器与S.Harris和D.Harris所著教材《数字设计和计算机体系结构》（RISC-V版本，简称[DDCARV]）第7.7.4节中所示的超标量处理器（为方便起见，该处理器已在图1中给出）之间的差异。

3) 分析DDCARV第7.7.4节的图7.70中的程序，该程序位于文件夹  
`[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_SuperscalarExample`下的PlatformIO项目中。使用仿真器和开发板两种方式，在SweRV EH1上运行该程序（对于后一种方式，请删除nop指令）。解释得到的结果。如有必要，可调整程序以获得最理想的IPC。

接下来，根据本实验及SweRVref.docx（第2部分）中所述，禁止双发射执行。将仿真结果和开发板上的执行结果与使能双发射功能时的相应结果进行比较。

- 4) 修改练习3中的程序，将指令add s9, s8, t1替换为add t2, s8, t1。解释得到的结果。如有必要，可调整程序以获得最理想的IPC。

然后，根据本实验及SweRVref.docx（第2节）中所述，禁止双发射执行。将仿真结果和开发板上的执行结果与使能双发射功能时的相应结果进行比较。

- 5) （以下练习基于《计算机组织结构和设计》（RISC-V版本，Patterson & Hennessy ([HePa])）中的练习4.31。）

在本练习中，我们将比较单发射处理器和双发射处理器的性能，并考虑可优化双发射执行的程序修改。本练习中的问题涉及以下循环（用C语言编写）：

```
for(i=0;i!=j;i+=2) b[i]=a[i]-a[i+1];
```

仅经过少量优化或未经优化的编译器可能会生成以下RISC-V汇编代码：

```
li x12, 0
li x13, 8000
li x14, 0
TOP:
    slli x5, x12, 2
    add x6, x10, x5
    lw x7, 0(x6)
    lw x29, 4(x6)
    sub x30, x7, x29
    add x31, x11, x5
    sw x30, 0(x31)
    addi x12, x12, 2
    ENT: bne x12, x13, TOP
```

该代码使用以下寄存器：

i	j	a	b	Temporary values
x12	x13	x10	x11	x5-x7, x29-x31

该代码位于[RVfpgaPath]/RVfpga/Labs/Lab17/PaHe\_SuperscalarExample路径下，与本书练习中提供的代码相比，存在一些小的修改，但不会影响程序的行为：

- 将寄存器x13初始化为8000，以使程序执行4000次迭代。
- 删除了jal指令。
- 将ld和sd指令替换为lw和sw指令。这表示访问从4个字节宽变为8个字节宽。

假设有一个具备以下属性的双发射静态调度处理器：

1. 一条指令必须是访存操作；另一条必须是算术/逻辑指令或分支指令。
2. 处理器的各阶段之间支持所有可能的转发路径。
3. 处理器可实现完美分支预测。
4. 如果两条指令相关，则可能无法同时发出。
5. 如果某一阶段需要暂停指令，则必须同时暂停两条指令。

- a) 比较该示例处理器与SweRV EH1处理器的属性。
- b) 绘制流水线图和仿真图，显示上述RISC-V代码的任意一次循环迭代（第一次迭代除外）在双发射SweRV EH1处理器上的执行情况。假设在进行4000次迭代后退出循环（上述代码即为如此）。
- c) 从单发射SweRV EH1处理器变为双发射SweRV EH1处理器时，加速比为多少？解释得到的结果。在开发板上测试程序并允许/禁止双发射执行。
- d) 重新排列/重写上述RISC-V代码，以在双发射SweRV EH1处理器上实现更高的性能。（但请勿展开循环。）
- e) 接下来展开RISC-V代码，使展开后循环的每次迭代等同于原循环的两次迭代。然后，重新排列/重写展开的代码，以在双发射SweRV EH1处理器上实现更高的性能。

6) （以下练习基于DDCARV第7章的练习7.30、7.32和7.34。）

假设SweRV EH1处理器正在运行以下代码片段。回想一下，SweRV EH1带有冒险单元。可以假设一个能够在一个周期内返回结果的存储器系统（为此，我们使用DCCM，将代码片段插入一个循环中，并避免使用第一次迭代，以免发生IS\$未命中）。

```
addi s1, t0, 11    # t0 contains the base address of the DCCM
lw    s2, 25(s1)
lw    s5, 16(s2)
add   s3, s2, s5
or    s4, s3, t4
and   s2, s3, s4
```

- a) 使用Verilator和GTKWave仿真该程序。分析结果，并针对每个周期指明以下内容：
  - \* 对哪些指令进行译码？将哪些指令发送至执行阶段？提交哪些指令？
  - \* 对哪些寄存器进行写操作和读操作？
  - \* 发生哪些转发和暂停情况？
- b) 对于该程序，处理器的CPI是多少？先给出理论解答，然后在开发板上执行程序，确认您的答案。
- c) 在单发射处理器上执行相同的分析，将结果与使用双发射处理器时的结果进行比较。

PlatformIO项目的路径如下：**[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV\_Exercises-30-32-34**。所分析的程序将插入一个循环中，以使仿真更易理解（可使用除第一次迭代外的任何迭代进行分析）并且可以使用性能计数器进行测定。

7) (以下练习基于DDCARV第7章的练习7.31、7.33和7.35。)

使用以下代码片段重复练习7。

```
addi s1, t0, 52
addi s0, s1, -4
lw    s3, 16(s0)
sw    s3, 20(s0)
xor   s2, s0, s3
or    s2, s2, s3
```

PlatformIO项目的路径如下：

`[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_Exercises-31-33-35。`