

**实验二**

**DLX/RISC-V指令格式**

学科：计算机体系结构

学期：2024-2025 第一学期

编制日期：2024 年 10 月 28 日

编制人：江家玮

学号：22281188

班级：计科2204

目录

[一、winDLX 1](#_Toc32718)

[1.1 winDLX模拟器的各窗口及各项功能 1](#_Toc19645)

[1.2 PRIM.S程序功能分析 2](#_Toc28330)

[1.3 指令详解 5](#_Toc12194)

[1.4 结果分析 6](#_Toc30130)

[二、RISC-V 6](#_Toc20747)

[2.1 程序功能解析及功能展示 6](#_Toc7441)

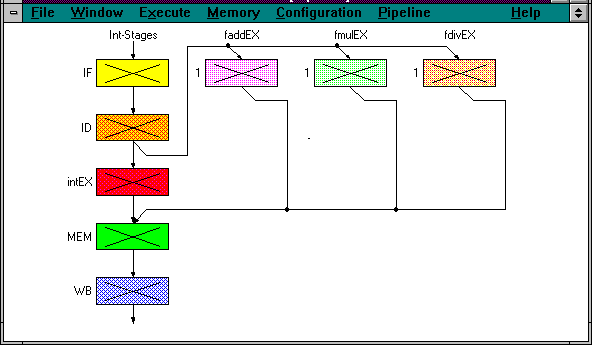
[2.2 指令详解 8](#_Toc10937)

一、 winDLX

* 1. winDLX模拟器的各窗口及各项功能

(1) ***Pipeline*** 窗口

Pipeline窗口展示DLX处理器的内部结构。窗口中用图表形示显示了DLX的五段流水线。



此图显示DLX处理器的五个流水段和浮点操作 (加 / 减, 乘和除)的单元。

(2) ***Code*** 窗口

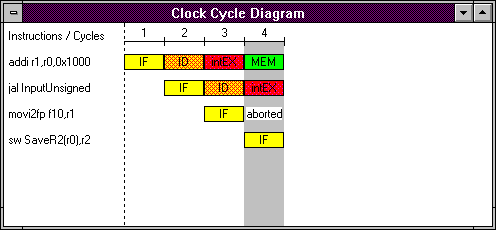
***Code***窗口展示存储器内容的三栏信息，从左到右依次为：地址 (符号或数字)、命令的十六进制机器代码和汇编命令。

$TEXT 0x20011000 addi r1,r0,0x1000

main+0x4 0x0c00003c jal InputUnsigned

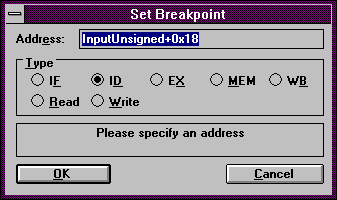
(3) ***Clock Cycle Diagram*** 窗口

***Clock Cycle Diagram*** 窗口显示流水线的时空图。



在窗口中，你将看到模拟正在第四时钟周期，第一条命令正在MEM段，第二条命令在intEX段，第四条命令在IF段。而第三条命令指示为"aborted"。

(4) ***Breakpoint*** 窗口



如果想查看已定义的断点，只要单击图标***Reakpoints***，将弹出一个小窗口，其中显示了所有断点。

(5) ***Register*** 窗口

***Register*** 窗口会显示各个寄存器中的内容。

(6) ***Statistics*** 窗口

***Statistics*** 窗口展示各种统计数字：总的周期数和暂停数，能够计算定向技术带来的加速比。

* 1. PRIM.S程序功能分析

这个程序的主要功能是生成一个素数表，并将前Count个素数存储在Table中。该程序使用了DLX汇编指令，以下是详细分析：

1.2.1 程序功能概述

✱该程序从2开始检查每个整数是否为素数。

✱它逐步生成并存储素数到表Table中，直到找出指定数量的素数。

✱指定的素数数量由变量Count决定，初始化为10，因此程序会生成前10个素数。

✱程序通过逐个检查当前数能否被已有素数整除来判断其是否为素数。

✱如果该数是素数，程序将其添加到Table中并继续检查下一个数，直到找到足够数量素数。

1.2.2 逐行分析

数据段

1. **.data**
2. **;\*\*\* size of table**
3. **.global  Count**
4. **Count:  .word  10**
5. **.global  Table**
6. **Table:  .space  Count\*4**

✱定义了两个全局变量：

✷Count: 保存要生成的素数数量，初始化为10。

✷Table: 一个存储素数的数组，空间大小为Count \* 4字节，每个整数占用4字节。

代码段

1. **.text**
2. **.global main**
3. **main:**
4. **;\*\*\* Initialization**
5. **addi  r1,r0,0  ;Index in Table**
6. **addi  r2,r0,2  ;Current value**

✱addi r1, r0, 0: 初始化r1为0，表示表中的索引，将用来记录当前存储素数的位置。

✱addi r2, r0, 2: 初始化r2为2，表示当前被测试的数，从2开始检查是否为素数。

主循环部分

✱addi r3, r0, 0: 初始化r3为0，用作帮助索引，遍历Table中的已有素数。

✱seq r4, r1, r3: 检查r1（当前存储素数的索引）和r3是否相等。如果相等，说明已经到达表 的末尾，没有任何素数能整除r2，因此r2是素数。

✱bnez r4, IsPrim: 如果r4非零，跳转到IsPrim，说明r2是素数。

✱lw r5, Table(R3): 从Table中取出一个已有的素数。

✱divu r6, r2, r5: 用r5（表中的素数）除以r2，结果存入r6。

✱multu r7, r6, r5: 将商r6乘以r5，得到一个被r5整除的数。

✱subu r8, r2, r7: 将r7从r2中减去，若结果为零，表示r2能被r5整除，不是素数。

✱beqz r8, IsNoPrim: 如果r8为0，跳转到IsNoPrim，r2不是素数。

✱addi r3, r3, 4: 索引增加4（每个整数占4字节），继续检查下一个已有素数。

✱j Loop: 返回Loop继续循环。

是素数时的操作

1. **IsPrim:  ;\*\*\* Write value into Table and increment index**
2. **sw  Table(r1),r2**
3. **addi  r1,r1,4**

✱sw Table(r1), r2: 将r2的值（即当前素数）存入Table中的位置r1。

✱addi r1, r1, 4: 将r1增加4，更新索引以准备存储下一个素数。

检查是否达到素数数量

1. **;\*\*\* 'Count' reached?**
2. **lw  r9, Count**
3. **srli  r10, r1, 2**
4. **sge  r11, r10, r9**
5. **bnez  r11, Finish**

✱lw r9, Count: 从内存中加载Count的值（即素数的数量限制，10）。

✱srli r10, r1, 2: 将r1右移2位，得到存储的素数个数（因为r1表示字节索引，右移2位相当于 除以4，得到实际的整数索引）。

✱sge r11, r10, r9: 检查当前存储的素数数量是否大于或等于Count，如果是，r11将被置为1。

✱bnez r11, Finish: 如果已经存储了足够多的素数，跳转到Finish，结束程序。

不是素数时的操作

1. **IsNoPrim: ;\*\*\* Check next value**
2. **addi  r2,r2,1  ;increment R2**
3. **j  NextValue**

✱addi r2, r2, 1: 如果r2不是素数，则将r2加1，检查下一个数。

✱j NextValue: 跳转到NextValue继续检测下一个数。

程序结束

1. **Finish:  ;\*\*\* end**
2. **trap  0**

✱trap 0: 触发系统调用，结束程序。

* 1. 指令详解
     1. addi r2,r0,1

该指令是I型指令

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Format** | **Bits** | | | | | |
|  | **0     5** | **6     10** | **11**     **15** | **16   20** | **21   25** | **26    31** |
| **I-type** | 0010011 | 00000 | 00010 | 0000000000000001 | | |

✱opcode是7位，用于标识指令类型。对于addi指令，操作码是0010011。

✱Rs1是5位，表示源地址的编号。r0对应的编号是00000。

✱Rd是5位，表示结果存储的目标注册。r2的编号是00010。

✱立即数是12位，表示要加到源寄存器值上的位置。在该指令中，立即数是1，二进制为0000000000000001。

1.3.2 movd f4,f2

该指令是R型指令

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Format** | **Bits** | | | | | |
|  | **0     5** | **6     10** | **11**     **15** | **16   20** | **21   25** | **26    31** |
| **R-type** | 110001 | 00010 | 00000 | 00100 | 00000 | 010001 |

✱opcode (操作码)：操作码决定指令类型。在DLX架构中，浮点寄存器之间的移动指令 movd 的操作码为110001。

✱Rs1 (源寄存器)：f2 是源寄存器，编号为2，二进制表示为00010。

✱Rs2 (源寄存器2)：movd不需要第二个源寄存器，因此设置为00000。

✱Rd (目标寄存器)：f4是目标寄存器，编号为4，二进制表示为00100。

✱shamt (移位量)：movd不需要移位操作，因此移位量为00000。

✱funct (功能码)：movd的功能码为 010001，用于指定浮点寄存器之间的移动操作。

1.3.3 subd f0,f0,f4

该指令是R型指令

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Format** | **Bits** | | | | | |
|  | **0     5** | **6     10** | **11**     **15** | **16   20** | **21   25** | **26    31** |
| **R-type** | 110001 | 00000 | 00100 | 00000 | 00000 | 00010 |

✱opcode (操作码)：操作码决定指令类型。在DLX架构中，浮点寄存器的双精度减法指令 subd 的操作码为 110001。

✱Rs1 (源寄存器1)：f0 是第一个源寄存器，编号为0，二进制表示为 00000。

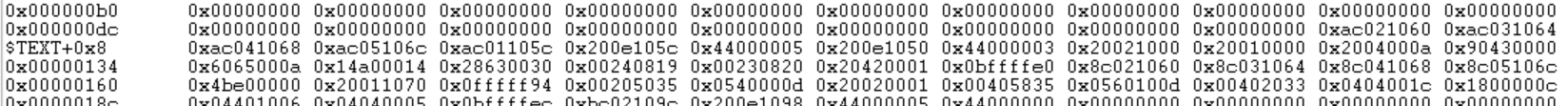
✱Rs2 (源寄存器2)：f4 是第二个源寄存器，编号为4，二进制表示为 00100。

✱Rd (目标寄存器)：f0 是目标寄存器，编号为0，二进制表示为 00000。

✱shamt (移位量)：该指令不需要移位操作，因此移位量为 00000。

✱funct (功能位)：subd 指令的功能码为 000101，用于指示这是一个双精度减法操作。

* 1. 结果分析



上图为执行FACT.S程序的时候内存中的存储情况，可以看到都是最低有效位在高地址处，所以是标准的大端排序。可以明显看出有对齐现象。

二、 RISC-V

2.1 程序功能解析及功能展示

这个程序的功能是计算Fibonacci数列的第n项，其中n是预定义在数据段中的值。

数据段

1. **.data**
2. **.word 2, 4, 6, 8**
3. **n: .word 9**

✱.word 2, 4, 6, 8：这些值未被使用，因此可以忽略。

✱n: .word 9：定义了变量n，其值为9。该值决定Fibonacci数列要计算的项数，即程序将计算并输出Fibonacci数列的第9项。

代码段

1. **.text**
2. **main:   add     t0, x0, x0**
3. **addi    t1, x0, 1**
4. **la      t3, n**
5. **lw      t3, 0(t3)**

✱add t0, x0, x0：将寄存器t0初始化为0。t0用来保存Fibonacci数列的前一项。

✱addi t1, x0, 1：将寄存器t1初始化为 1。t1用来保存Fibonacci数列的当前项。

✱la t3, n：将变量n的地址加载到寄存器t3中。

✱lw t3, 0(t3)：从地址n加载n的值到寄存器t3中，t3的初始值现在是 9。

1. **fib:    beq     t3, x0, finish**
2. **add     t2, t1, t0**
3. **mv      t0, t1**
4. **mv      t1, t2**
5. **addi    t3, t3, -1**
6. **j       fib**

这是 Fibonacci 数列的主要循环部分：

✱beq t3, x0, finish：检查t3是否等于 0，如果是，则跳转到finish，结束循环。t3用作计数器，表示剩余要计算的Fibonacci项数。

✱add t2, t1, t0：将t1和t0的值相加，结果存储在t2中。这一步相当于计算Fibonacci数列中的下一项。

✱mv t0, t1：将t1的值复制到t0，表示将当前的Fibonacci项移动为前一项。

✱mv t1, t2：将t2的值复制到t1，表示将新计算的Fibonacci项赋值为当前项。

✱addi t3, t3, -1：将t3减 1，表示还需要计算的项数减 1。

✱j fib：跳回fib，继续下一个循环。

结束部分

1. **finish: addi    a0, x0, 1**
2. **addi    a1, t0, 0**
3. **ecall *# print integer ecall***
4. **addi    a0, x0, 10**
5. **ecall *# terminate ecall***

✱addi a0, x0, 1：将系统调用参数1加载到a0，表示需要执行 "print integer" 系统调用。

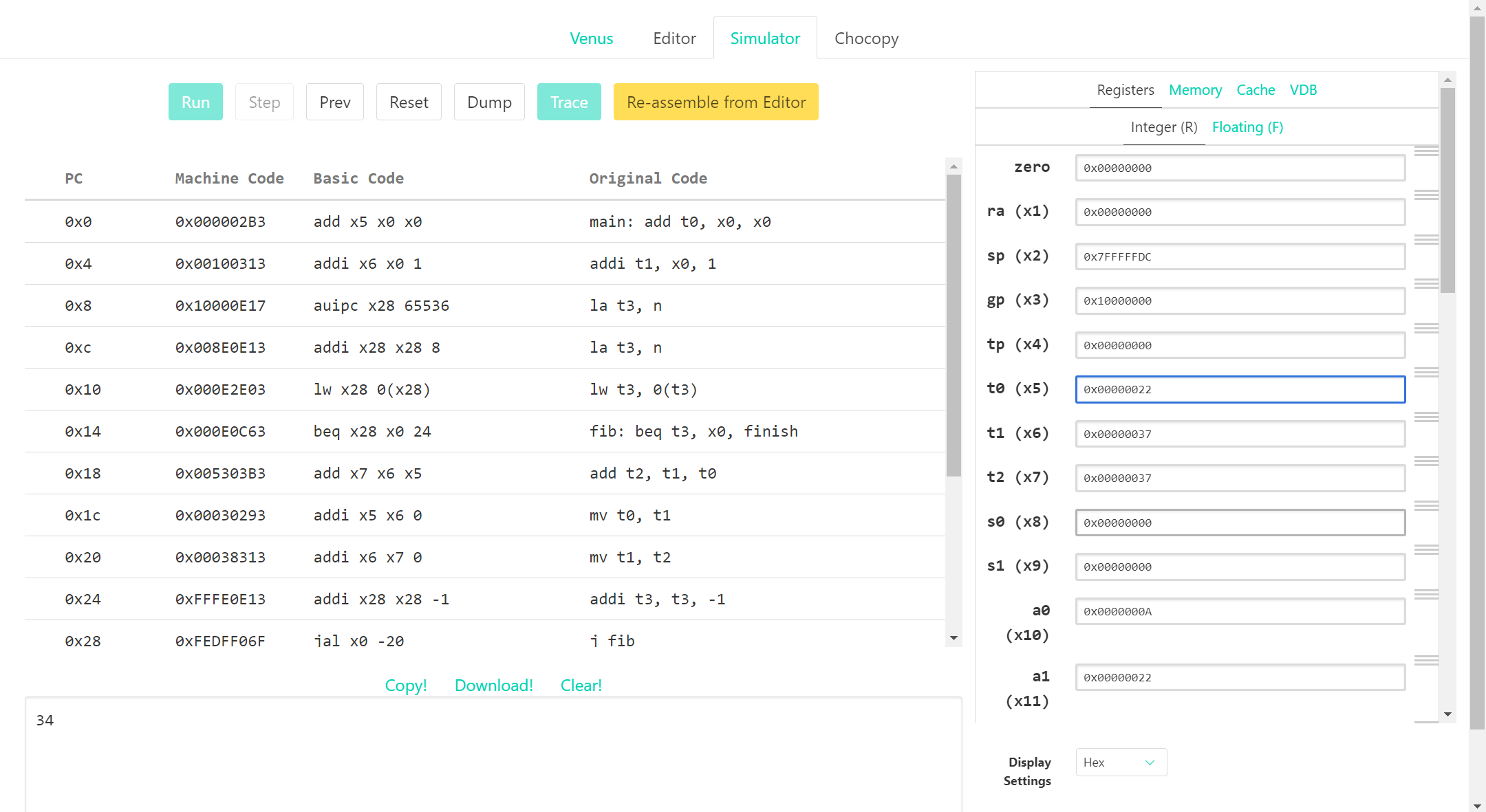
✱addi a1, t0, 0：将寄存器t0中的值（即Fibonacci数列的第n项）加载到寄存器a1，这是即将打印的数值。

✱ecall：执行系统调用，打印a1中的数值。

✱addi a0, x0, 10：将10加载到a0，表示系统调用代码为终止程序。

✱ecall：执行终止程序的系统调用。

结果展示



2.2 指令详解

2.2.1 lw x28 0(28)

该指令是I型指令

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Format** | **Bits** | | | | | | |
|  | **0     6** | **7     11** | **12**     **14** | **15   19** | **20   25** | | **26    31** |
| **I-type** | 0000011 | 11100 | 010 | 11100 | | 000000000000 | |

2.2.2 beq x28 x0 24

该指令是B型指令

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Format** | **Bits** | | | | | | | |
|  | **0     6** | **7     11** | **12**     **14** | **15   19** | **20   25** | | **26    31** | |
| **B-type** | 1100011 | 11000 | 000 | 11100 | | 00000 | | 0000000 |

2.2.3 beq x28 x0 24

该指令是J型指令

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Format** | **Bits** | | | | | |
|  | **0     6** | **7     11** | **12**     **14** | **15   19** | **20   25** | **26    31** |
| **J-type** | 1101111 | 00000 | 1|1111110110|1|11111111 | | | |