单周期处理器实现插入排序

2024 年 7 月 28 日

一、 实验目的

使用单周期处理器完成插入排序，并将结果显示在七位字码管上。

二、 设计方案

采用单周期处理器（同理论课）实现 MIPS 指令集子集，将插入排序 c++ 程序翻译为 MIPS 指令后通过单周期处理器运行，最后将排序后结果逐个输出到字码管外设地址，使用循环执行无效指令的方式实现 delay 从而达到每个数据维持显示 1s。

三、 算法指令

MIPS 汇编语言实现插入排序的源代码见附录 1。

四、 关键代码清单

ALU.v: 运算单元
ALUControl.v: 运算控制单元
Control.v:CPU 控制单元
CPU.v: 顶层文件
DataMemory.v: 数据存储器
InstructionMemory.v: 指令存储器
RegisterFile.v: 寄存器

test_cpu.v:testBench

xdc_for_singleCycle.xdc: 约束文件

insert_sort.asm: 插入排序汇编文件

五、 仿真结果及分析

仿真得到 RAM_data[255:0] 和 LED[31:0] 部分结果如图所示：

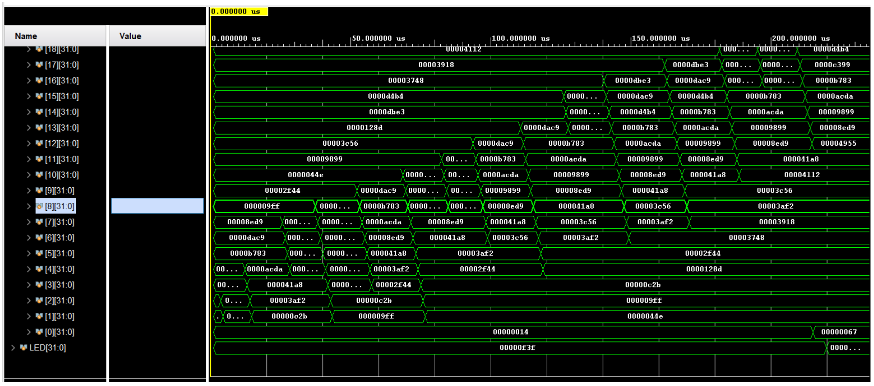

图 1: RAM_data[255:0]

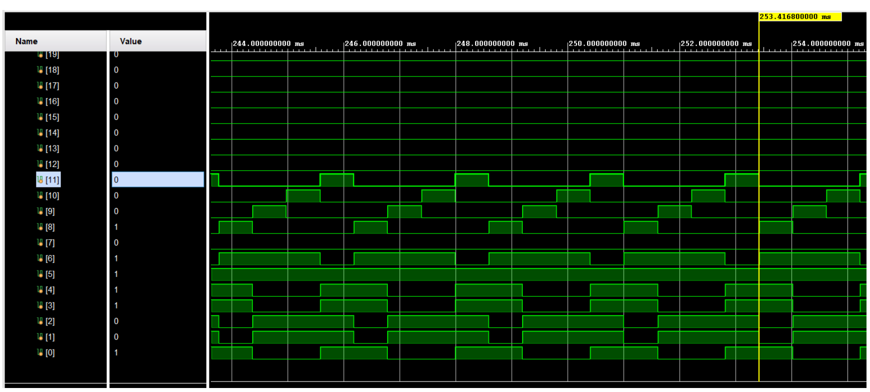

图 2: LED[31:0]

根据图 1 中可以看出插入排序的过程，根据图 2 可以看出 LED[8]、LED[9]、LED[10]、LED[11] 依此为 1，其它为零，从而实现字码管动态显示的效果。

六、 综合情况

硬件资源占用情况和时序报告如图 3 图 4 所示：

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	DSPs (90)	Bonded IOB (250)	BUFGCTRL (32)	MMCME2_ADV (5)
top	4080	9260	1351	512	4096	4080	3	14	2	1
clk_out_0 (clk_wiz_0)	0	0	0	0	0	0	0	0	2	1
inst (clk_wiz_0_clk_\	0	0	0	0	0	0	0	0	2	1
nolabel_line10 (CPU)	4080	9260	1351	512	4096	4080	3	0	0	0
alu1 (ALU)	27	0	3	0	15	27	3	0	0	0
data_memory1 (Data	2261	8236	1090	512	3408	2261	0	0	0	0
register_file1 (Regist	1779	992	258	0	908	1779	0	0	0	0

图 3: 硬件资源占用情况

Setup

Worst Negative Slack (WNS):	22.047 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	18488

Hold

Worst Hold Slack (WHS):	0.321 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	18488

Pulse Width

Worst Pulse Width Slack (WPWS):	3.000 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	9266

All user specified timing constraints are met.

图 4: 时序分析报告

最高工作频率：

$$f_{max} = \frac{1}{T - WNS} = \frac{1}{1/(20 \times 10^6) - 22.047 \times 10^{-9}} = 35.77MHz$$

下面计算 CPI（仅统计排序部分）：MARS 统计插入排序总共共执行指令数如下图，总指令数

$$N = 2417$$

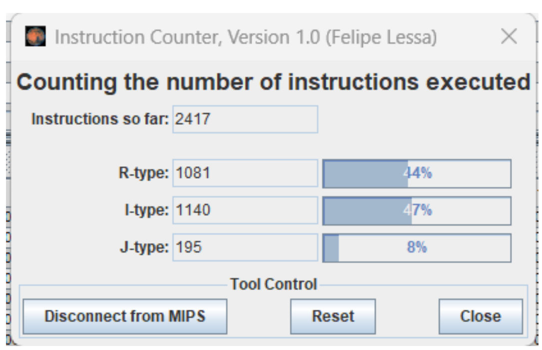

图 5: MIPS 指令数统计

排序部分仿真结果如下图，总用时

$$t = 214.7us$$

周期数

$$C = \frac{t}{T} = \frac{214.7 \times 10^{-6}}{0.05 \times 10^{-6}} = 4294$$

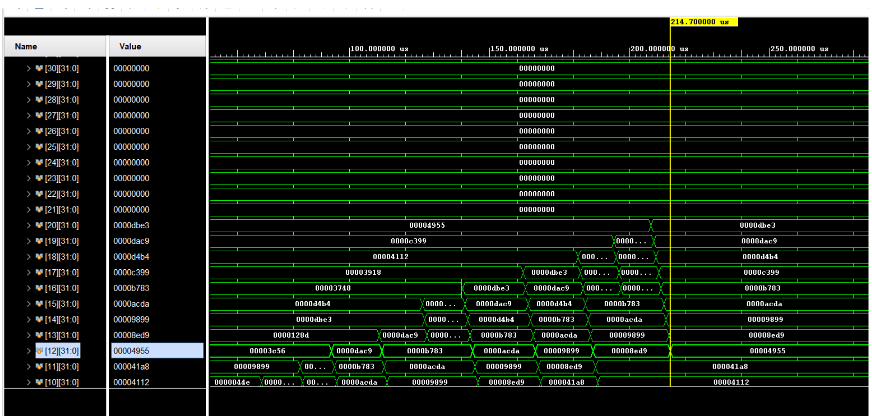

图 6: 排序部分仿真结果

因此插入排序部分 CPI 计算如下：

$$CPI = \frac{C}{N} = \frac{4294}{2417} = 1.7766$$

七、 硬件调试情况

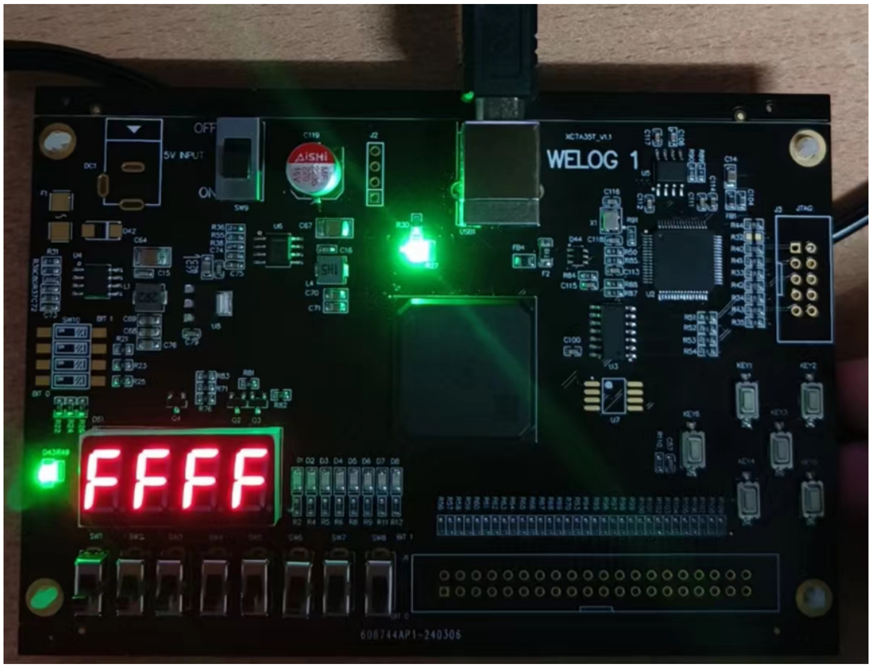

图 7: 实际运行情况

代码可在 FPGA 上正常运行，程序以显示 FFFF 作为结束标志，如图所示。

八、 附录

(一) MIPS 实现插入排序

```
    main:
    addi $s0 $zero 0 #buffer[]
    addi $s2 $zero 0
    lw $s1 0($s0) #($s1)int N = buffer[0]
    addi $v1 $s1 0
insert_sort:? 》
    addi $t0 $s0 4 #$t0 = &(buffer[1])
```

```
    addi $t1 $s1 0 #$t1 = N
    addi $t2 $zero 1 #($t2)int i = 1
insert_sort_loop:
    beq $t1 $t2 end_insert_sort_loop
search:
    addi $t4 $t2 -1 #($t4) i = n - 1
    sll $t5 $t2 2 #$t5 = 4n
    add $t6 $t0 $t5 #($t6)int tmp = v[n];
    lw $t6 ($t6)
search_loop:
    blt $t4 $zero end_search_loop
    addi $s2 $s2 1
    sll $t5 $t4 2 #$t5 = 4i
    add $t7 $t0 $t5 #$t7 = v[i]
    lw $t7 ($t7)
    ble $t7 $t6 end_search_loop #if (v[i] <= tmp) break;
    addi $t4 $t4 -1 #i--
    j search_loop
end_search_loop:
    addi $v0 $t4 1 #return i + 1
insert:
    addi $t4 $t2 -1 #($t4) i = n - 1
    sll $t5 $t2 2 #$t5 = 4n
    add $t6 $t0 $t5 #($t6)int tmp = v[n];
    lw $t6 ($t6)
    addi $t7 $v0 0 #$t7 = k
insert_loop:
    blt $t4 $t7 end_insert_loop
    sll $t5 $t4 2 #$t5 = 4i
    add $t8 $t0 $t5
    lw $s4 ($t8) #$s4 = v[i]
    addi $t9 $t8 4 #$t9 = &v[i+1]
    sw $s4 ($t9) #v[i+1] = v[i]
```

```
    addi $t4 $t4 -1 #i--
    j insert_loop
end_insert_loop:
    sll $t7 $t7 2
    add $t9 $t0 $t7
    sw $t6 0($t9) #v[k] = tmp;
    addi $t2 $t2 1 #i++
    j insert_sort_loop
end_insert_sort_loop:
    sw $s2 0($s0)

    addi $s3 $zero 0 #count
    addi $t0 $zero 4 #&buffer[i]
    addi $k1 $zero 2000 #delay_times

loop:
    addi $t7 $zero 900 #Times
    addi $t8 $zero 0 #OpTimes
decode_loop:
    addi $a0 $zero 16384 #led
    addi $a2 $zero 1 #led_count
    lw $t1 0($t0) #[15:0]in = buffer[i]
    andi $v0 $t1 15 #[3:0]in
    j decode
led1:
    addi $s7 $zero 256 #an[0]
    or $v0 $v0 $s7 #led
    sw $v0 ($a0)

    addi $k0 $zero 0
delay1:
    addi $k0 $k0 1
    beq $k0 $k1 end1
```

```
    j delay1
end1:

    lw $t1 0($t0)
    srl $v0 $t1 4 #[7:4]in
    andi $v0 $v0 15
    addi $a2 $a2 1 #led_count += 1
    j decode
led2:
    addi $s7 $zero 512 #an[1]
    or $v0 $v0 $s7 #led
    sw $v0 ($a0)

    addi $k0 $zero 0
delay2:
    addi $k0 $k0 1
    beq $k0 $k1 end2
    j delay2
end2:

    lw $t1 0($t0)
    srl $v0 $t1 8 #[11:8]in
    andi $v0 $v0 15
    addi $a2 $a2 1 #led_count += 1
    j decode
led3:
    addi $s7 $zero 1024 #an[2]
    or $v0 $v0 $s7 #led
    sw $v0 ($a0)

    addi $k0 $zero 0
delay3:
    addi $k0 $k0 1
```

```
    beq $k0 $k1 end3
    j delay3
end3:


    lw $t1 0($t0)
    srl $v0 $t1 12 #in[15:12]
    andi $v0 $v0 15
    addi $a2 $a2 1 #led_count +=1
    j decode
led4:
    addi $s7 $zero 2048 #an[3]
    or $v0 $v0 $s7
    sw $v0 ($a0)

    addi $k0 $zero 0
delay4:
    addi $k0 $k0 1
    beq $k0 $k1 end4
    j delay4
end4:


    addi $t8 $t8 1
    bne $t8 $t7 decode_loop #if(OpTimes < Times) decode(in)
    addi $s3 $s3 1 #count++
    addi $t0 $t0 4 #i++
    beq $s3 $v1 end #if(count == N) break

    j loop
decode:
    addi $a1 $zero 0 #decode_count
    beq $v0 $a1 decode0
```

```
    addi $a1 $a1 1
    beq $v0 $a1 decode1
    addi $a1 $a1 1
    beq $v0 $a1 decode2
    addi $a1 $a1 1
    beq $v0 $a1 decode3
    addi $a1 $a1 1
    beq $v0 $a1 decode4
    addi $a1 $a1 1
    beq $v0 $a1 decode5
    addi $a1 $a1 1
    beq $v0 $a1 decode6
    addi $a1 $a1 1
    beq $v0 $a1 decode7
    addi $a1 $a1 1
    beq $v0 $a1 decode8
    addi $a1 $a1 1
    beq $v0 $a1 decode9
    addi $a1 $a1 1
    beq $v0 $a1 decodeA
    addi $a1 $a1 1
    beq $v0 $a1 decodeB
    addi $a1 $a1 1
    beq $v0 $a1 decodeC
    addi $a1 $a1 1
    beq $v0 $a1 decodeD
    addi $a1 $a1 1
    beq $v0 $a1 decodeE
    addi $a1 $a1 1
    beq $v0 $a1 decodeF
decode0:
    addi $v0 $zero 63
    j decode_jump
```

```
decode1:
    addi $v0 $zero 6
    j decode_jump
decode2:
    addi $v0 $zero 91
    j decode_jump
decode3:
    addi $v0 $zero 79
    j decode_jump
decode4:
    addi $v0 $zero 102
    j decode_jump
decode5:
    addi $v0 $zero 109
    j decode_jump
decode6:
    addi $v0 $zero 125
    j decode_jump
decode7:
    addi $v0 $zero 7
    j decode_jump
decode8:
    addi $v0 $zero 127
    j decode_jump
decode9:
    addi $v0 $zero 111
    j decode_jump
decodeA:
    addi $v0 $zero 119
    j decode_jump
decodeB:
    addi $v0 $zero 124
    j decode_jump
```

```
decodeC:
    addi $v0 $zero 88
    j decode_jump
decodeD:
    addi $v0 $zero 94
    j decode_jump
decodeE:
    addi $v0 $zero 121
    j decode_jump
decodeF:
    addi $v0 $zero 113
    j decode_jump
decode_jump:
    addi $t9 $zero 1
    beq $a2 $t9 led1
    addi $t9 $t9 1
    beq $a2 $t9 led2
    addi $t9 $t9 1
    beq $a2 $t9 led3
    addi $t9 $t9 1
    beq $a2 $t9 led4
end:
    addi $t0 $zero 3953
    sw $t0 ($a0)
    j end
```