# 2022《数字逻辑与处理器基础》处理器大作业

第二部分

2022/05/16

一、实验目的

1. 掌握 MIPS 单周期处理器的控制通路和数据通路的设计原理和 RTL 实现方法；
2. 掌握 MIPS 多周期处理器的控制通路和数据通路的设计原理和 RTL 实现方法；
3. 深入理解 MIPS 单周期和多周期处理器在资源和性能上的设计折衷。

二、MIPS 指令集

1. MIPS 指令集子集：

lw, sw, lui,

add, addu, sub, subu, addi, addiu,

and, or, xor, nor, andi, sll, srl, sra, slt, sltu, slti, sltiu,

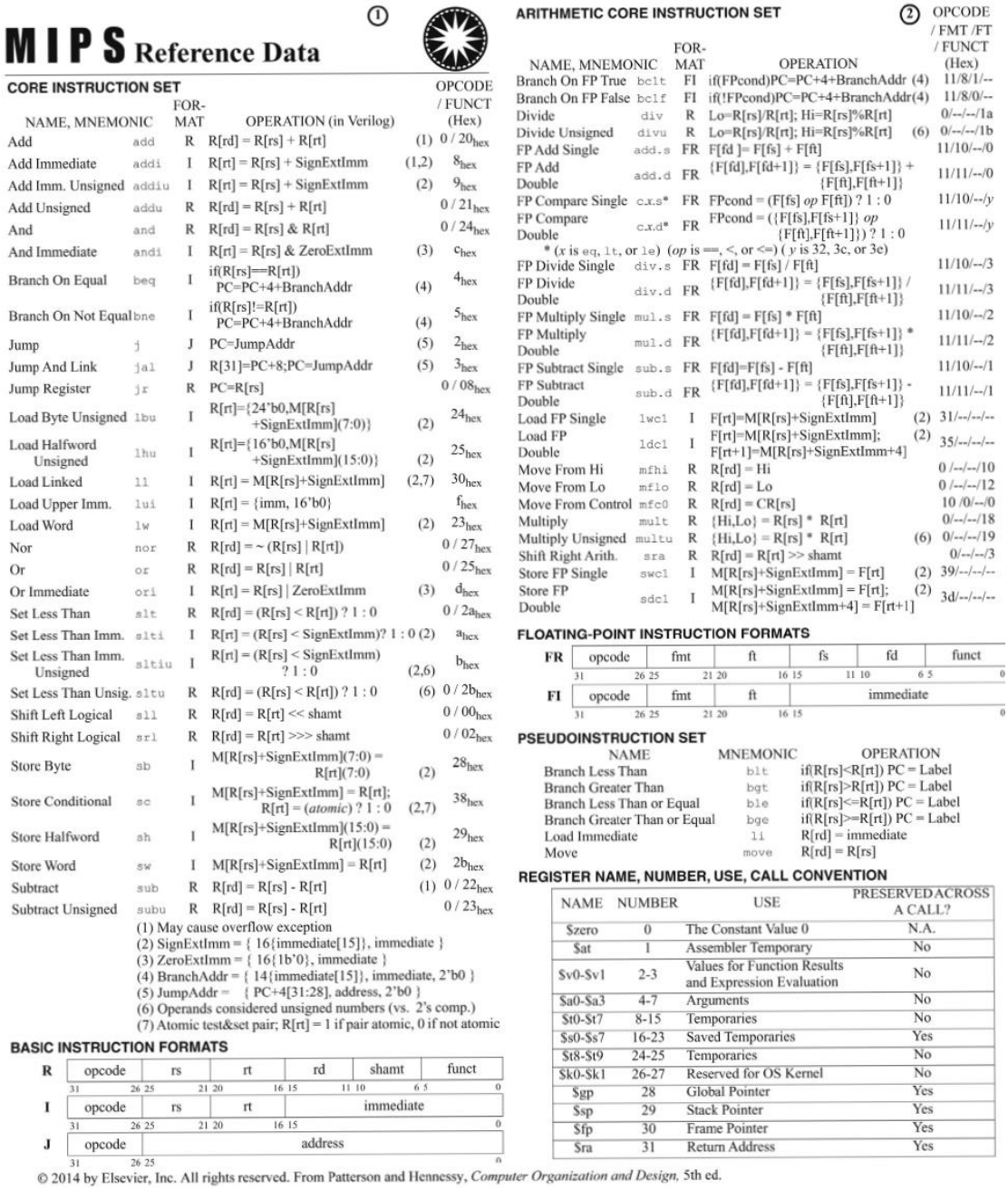
beq, j, jal, jr, jalr

2. MIPS 指令格式：

| Instruction | OpCode[5:0] | Rs[4:0] | Rt[4:0] | Rd[4:0] | Shamt[4:0] | Funct[5:0] |
|---|---|---|---|---|---|---|
| lw rt, offset (rs) | 0x23 | rs | rt | offset | | |
| sw rt, offset (rs) | 0x2b | rs | rt | offset | | |
| lui rt, imm | 0x0f | 0 | rt | imm | | |
| add rd, rs, rt | 0 | rs | rt | rd | 0 | 0x20 |
| addu rd, rs, rt | 0 | rs | rt | rd | 0 | 0x21 |
| sub rd, rs, rt | 0 | rs | rt | rd | 0 | 0x22 |
| subu rd, rs, rt | 0 | rs | rt | rd | 0 | 0x23 |
| addi rt, rs, imm | 0x08 | rs | rt | imm | | |
| addiu rt, rs, imm | 0x09 | rs | rt | imm | | |
| and rd, rs, rt | 0 | rs | rt | rd | 0 | 0x24 |
| or rd, rs, rt | 0 | rs | rt | rd | 0 | 0x25 |
| xor rd, rs, rt | 0 | rs | rt | rd | 0 | 0x26 |
| nor rd, rs, rt | 0 | rs | rt | rd | 0 | 0x27 |
| andi rt, rs, imm | 0x0c | rs | rt | imm | | |
| sll rd, rt, shamt | 0 | 0 | rt | rd | shamt | 0 |
| srl rd, rt, shamt | 0 | 0 | rt | rd | shamt | 0x02 |
| sra rd, rt, shamt | 0 | 0 | rt | rd | shamt | 0x03 |
| slt rd, rs, rt | 0 | rs | rt | rd | 0 | 0x2a |
| sltu rd, rs, rt | 0 | rs | rt | rd | 0 | 0x2b |
| slti rt, rs, imm | 0x0a | rs | rt | imm | | |
| sltiu rt, rs, imm | 0x0b | rs | rt | imm | | |
| beq rs, rt, label | 0x04 | rs | rt | offset | | |
| j target | 0x02 | target | | | | |
| jal target | 0x03 | target | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| jr rs | 0 | rs | 0 | | | 0x08 |
| jalr rd, rs | 0 | rs | 0 | rd | 0 | 0x09 |

3. MIPS 指令集参考资料



**MIPS Reference Data** ①

**CORE INSTRUCTION SET**

| NAME, MNEMONIC | | FOR-MAT | OPERATION (in Verilog) | | OPCODE / FUNCT (Hex) |
|---|---|---|---|---|---|
| Add | add | R | R[rd] = R[rs] + R[rt] | (1) | 0 / $20_{hex}$ |
| Add Immediate | addi | I | R[rt] = R[rs] + SignExtImm | (1,2) | $8_{hex}$ |
| Add Imm. Unsigned | addiu | I | R[rt] = R[rs] + SignExtImm | (2) | $9_{hex}$ |
| Add Unsigned | addu | R | R[rd] = R[rs] + R[rt] | | 0 / $21_{hex}$ |
| And | and | R | R[rd] = R[rs] & R[rt] | | 0 / $24_{hex}$ |
| And Immediate | andi | I | R[rt] = R[rs] & ZeroExtImm | (3) | $c_{hex}$ |
| Branch On Equal | beq | I | if(R[rs]==R[rt]) PC=PC+4+BranchAddr | (4) | $4_{hex}$ |
| Branch On Not Equal | bne | I | if(R[rs]!=R[rt]) PC=PC+4+BranchAddr | (4) | $5_{hex}$ |
| Jump | j | J | PC=JumpAddr | (5) | $2_{hex}$ |
| Jump And Link | jal | J | R[31]=PC+8;PC=JumpAddr | (5) | $3_{hex}$ |
| Jump Register | jr | R | PC=R[rs] | | 0 / $08_{hex}$ |
| Load Byte Unsigned | lbu | I | R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)} | (2) | $24_{hex}$ |
| Load Halfword Unsigned | lhu | I | R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)} | (2) | $25_{hex}$ |
| Load Linked | ll | I | R[rt] = M[R[rs]+SignExtImm] | (2,7) | $30_{hex}$ |
| Load Upper Imm. | lui | I | R[rt] = {imm, 16'b0} | | $f_{hex}$ |
| Load Word | lw | I | R[rt] = M[R[rs]+SignExtImm] | (2) | $23_{hex}$ |
| Nor | nor | R | R[rd] = ~ (R[rs] \| R[rt]) | | 0 / $27_{hex}$ |
| Or | or | R | R[rd] = R[rs] \| R[rt] | | 0 / $25_{hex}$ |
| Or Immediate | ori | I | R[rt] = R[rs] \| ZeroExtImm | (3) | $d_{hex}$ |
| Set Less Than | slt | R | R[rd] = (R[rs] < R[rt]) ? 1 : 0 | | 0 / $2a_{hex}$ |
| Set Less Than Imm. | slti | I | R[rt] = (R[rs] < SignExtImm)? 1 : 0 | (2) | $a_{hex}$ |
| Set Less Than Imm. Unsigned | sltiu | I | R[rt] = (R[rs] < SignExtImm) ? 1 : 0 | (2,6) | $b_{hex}$ |
| Set Less Than Unsig. | sltu | R | R[rd] = (R[rs] < R[rt]) ? 1 : 0 | (6) | 0 / $2b_{hex}$ |
| Shift Left Logical | sll | R | R[rd] = R[rt] << shamt | | 0 / $00_{hex}$ |
| Shift Right Logical | srl | R | R[rd] = R[rt] >>> shamt | | 0 / $02_{hex}$ |
| Store Byte | sb | I | M[R[rs]+SignExtImm](7:0) = R[rt](7:0) | (2) | $28_{hex}$ |
| Store Conditional | sc | I | M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0 | (2,7) | $38_{hex}$ |
| Store Halfword | sh | I | M[R[rs]+SignExtImm](15:0) = R[rt](15:0) | (2) | $29_{hex}$ |
| Store Word | sw | I | M[R[rs]+SignExtImm] = R[rt] | (2) | $2b_{hex}$ |
| Subtract | sub | R | R[rd] = R[rs] - R[rt] | (1) | 0 / $22_{hex}$ |
| Subtract Unsigned | subu | R | R[rd] = R[rs] - R[rt] | | 0 / $23_{hex}$ |

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

**BASIC INSTRUCTION FORMATS**

| | | | | | | |
|---|---|---|---|---|---|---|
| R | opcode | rs | rt | rd | shamt | funct |
| | 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |
| I | opcode | rs | rt | immediate | | |
| | 31 26 | 25 21 | 20 16 | 15 0 | | |
| J | opcode | address | | | | |
| | 31 26 | 25 0 | | | | |

**ARITHMETIC CORE INSTRUCTION SET** ②

| NAME, MNEMONIC | | FOR-MAT | OPERATION | | OPCODE / FMT /FT / FUNCT (Hex) |
|---|---|---|---|---|---|
| Branch On FP True | bc1t | FI | if(FPcond)PC=PC+4+BranchAddr | (4) | 11/8/1/-- |
| Branch On FP False | bc1f | FI | if(!FPcond)PC=PC+4+BranchAddr | (4) | 11/8/0/-- |
| Divide | div | R | Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] | | 0/--/--/1a |
| Divide Unsigned | divu | R | Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] | (6) | 0/--/--/1b |
| FP Add Single | add.s | FR | F[fd ]= F[fs] + F[ft] | | 11/10/--/0 |
| FP Add Double | add.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]} | | 11/11/--/0 |
| FP Compare Single | c.x.s* | FR | FPcond = (F[fs] *op* F[ft]) ? 1 : 0 | | 11/10/--/*y* |
| FP Compare Double | c.x.d* | FR | FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0 | | 11/11/--/*y* |
| * (*x* is eq, lt, or le) (*op* is ==, <, or <=) ( *y* is 32, 3c, or 3e) | | | | | |
| FP Divide Single | div.s | FR | F[fd] = F[fs] / F[ft] | | 11/10/--/3 |
| FP Divide Double | div.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]} | | 11/11/--/3 |
| FP Multiply Single | mul.s | FR | F[fd] = F[fs] * F[ft] | | 11/10/--/2 |
| FP Multiply Double | mul.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]} | | 11/11/--/2 |
| FP Subtract Single | sub.s | FR | F[fd]=F[fs] - F[ft] | | 11/10/--/1 |
| FP Subtract Double | sub.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]} | | 11/11/--/1 |
| Load FP Single | lwc1 | I | F[rt]=M[R[rs]+SignExtImm] | (2) | 31/--/--/-- |
| Load FP Double | ldc1 | I | F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4] | (2) | 35/--/--/-- |
| Move From Hi | mfhi | R | R[rd] = Hi | | 0 /--/--/10 |
| Move From Lo | mflo | R | R[rd] = Lo | | 0 /--/--/12 |
| Move From Control | mfc0 | R | R[rd] = CR[rs] | | 10 /0/--/0 |
| Multiply | mult | R | {Hi,Lo} = R[rs] * R[rt] | | 0/--/--/18 |
| Multiply Unsigned | multu | R | {Hi,Lo} = R[rs] * R[rt] | (6) | 0/--/--/19 |
| Shift Right Arith. | sra | R | R[rd] = R[rt] >> shamt | | 0/--/--/3 |
| Store FP Single | swc1 | I | M[R[rs]+SignExtImm] = F[rt] | (2) | 39/--/--/-- |
| Store FP Double | sdc1 | I | M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1] | (2) | 3d/--/--/-- |

**FLOATING-POINT INSTRUCTION FORMATS**

| | | | | | | |
|---|---|---|---|---|---|---|
| FR | opcode | fmt | ft | fs | fd | funct |
| | 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |
| FI | opcode | fmt | ft | immediate | | |
| | 31 26 | 25 21 | 20 16 | 15 0 | | |

**PSEUDOINSTRUCTION SET**

| NAME | MNEMONIC | OPERATION |
|---|---|---|
| Branch Less Than | blt | if(R[rs]<R[rt]) PC = Label |
| Branch Greater Than | bgt | if(R[rs]>R[rt]) PC = Label |
| Branch Less Than or Equal | ble | if(R[rs]<=R[rt]) PC = Label |
| Branch Greater Than or Equal | bge | if(R[rs]>=R[rt]) PC = Label |
| Load Immediate | li | R[rd] = immediate |
| Move | move | R[rd] = R[rs] |

**REGISTER NAME, NUMBER, USE, CALL CONVENTION**

| NAME | NUMBER | USE | PRESERVED ACROSS A CALL? |
|---|---|---|---|
| $zero | 0 | The Constant Value 0 | N.A. |
| $at | 1 | Assembler Temporary | No |
| $v0-$v1 | 2-3 | Values for Function Results and Expression Evaluation | No |
| $a0-$a3 | 4-7 | Arguments | No |
| $t0-$t7 | 8-15 | Temporaries | No |
| $s0-$s7 | 16-23 | Saved Temporaries | Yes |
| $t8-$t9 | 24-25 | Temporaries | No |
| $k0-$k1 | 26-27 | Reserved for OS Kernel | No |
| $gp | 28 | Global Pointer | Yes |
| $sp | 29 | Stack Pointer | Yes |
| $fp | 30 | Frame Pointer | Yes |
| $ra | 31 | Return Address | Yes |



注：jal 指令的 R[31]=PC+8 是针对流水线处理器的情形，这是因为发生跳转时下一条指令(PC+4)已经在执行了，所以需要保存的是接下来的第二条指令(PC+8)。而对于多周期处理器来说，由于不存在指令同时执行的情况，因此需要保存的是下一条指令(PC+4)。

三、实验内容

1. （MIPS 单周期 CPU 实现，3 分）：
   a) **控制器模块设计**：根据对各个控制信号的理解，完成 MIPS 指令集子集与控制信号的真值表（如下表所示，填 0、1、2、x 等），并根据真值表完成 *single-cycle*

文件夹中控制器模块 **Control.v** 的 Verilog 代码实现。

| Instruction | PCSrc [1:0] | Branch | RegWrite | RegDst [1:0] | MemRead | MemWrite | MemtoReg [1:0] | ALUSrc1 | ALUSrc2 | ExtOp | LuOp |
|---|---|---|---|---|---|---|---|---|---|---|---|
| lw | | | | | | | | | | | |
| sw | | | | | | | | | | | |
| lui | | | | | | | | | | | |
| add | | | | | | | | | | | |
| addu | | | | | | | | | | | |
| sub | | | | | | | | | | | |
| subu | | | | | | | | | | | |
| addi | | | | | | | | | | | |
| addiu | | | | | | | | | | | |
| and | | | | | | | | | | | |
| or | | | | | | | | | | | |
| xor | | | | | | | | | | | |
| nor | | | | | | | | | | | |
| andi | | | | | | | | | | | |
| sll | | | | | | | | | | | |
| srl | | | | | | | | | | | |
| sra | | | | | | | | | | | |
| slt | | | | | | | | | | | |
| sltu | | | | | | | | | | | |
| slti | | | | | | | | | | | |
| sltiu | | | | | | | | | | | |
| beq | | | | | | | | | | | |
| j | | | | | | | | | | | |
| jal | | | | | | | | | | | |
| jr | | | | | | | | | | | |
| jalr | | | | | | | | | | | |

b) **数据通路设计：**阅读 *single-cycle* 文件夹中的各个基础功能模块的 Verilog 代码，理解每个模块的输入输出接口和实现的基本功能。将这些基础功能电路模块进行实例化和连接（参考教材第 VI 章图 VI-19），基于处理器大作业第一部分设计的 **ALU.v** 和 **ALUController.v** 模块（自行替换相应文件），完成单周期处理器整体数据通路 **CPU.v** 的 Verilog 代码实现。请写出数据通路中包含的多路选择器，结合代码简要说明它们的功能。

c) **汇编程序分析-1**：阅读 **InstructionMemory.v** 中的指令代码，结合注释理解存储器中的指令程序，对应的 MIPS 汇编指令如下所示。

| MIPS Assembly 1 | |
|---|---|
| 0 | addi $a0, $zero, 12123 |
| 1 | addiu $a1, $zero, -12345 |
| 2 | sll $a2, $a1, 16 |
| 3 | sra $a3, $a2, 16 |
| 4 | beq $a3, $a1, L1 |
| 5 | lui $a0, 22222 |
| | L1: |
| 6 | add $t0, $a2, $a0 |
| 7 | sra $t1, $t0, 8 |
| 8 | addi $t2, $zero, -12123 |
| 9 | slt $v0, $a0, $t2 |
| 10 | sltu $v1, $a0, $t2 |
| | Loop: |
| 11 | j Loop |

i. 这段程序运行足够长时间后，寄存器$a0 到$a3，$t0 到 t2，$v0 到$v1 中的值应该是多少？写出计算过程。

ii. 使用ModelSim或Vivado等仿真软件进行仿真，顶层仿真模块为**test_cpu.v**。请给出仿真波形图，验证计算结果和所设计的单周期处理器的功能正确性。

2. （MIPS 多周期 CPU 实现，3 分）：

a) **多周期状态机控制器：**阅读并理解 *multi-cycle* 文件夹中控制器模块**Controller.v**的 Verilog 实现，画出状态转移图。要求状态转移列出具体的指令类型（结合二.1 中给定的MIPS 指令集子集），并且需要在状态图中写出相关的控制信号。

b) **多周期 CPU 的 ALU 控制逻辑与功能实现：**基于 *multi-cycle* 文件夹中控制器模块 **Controller.v**，完成 **Controller.v** 和 **ALUControl.v** 文件中关于 ALU 控制逻辑的 RTL 实现，以及 **ALU.v** 文件中 32 位 ALU 的行为级 RTL 实现。请写出你的设计思路和代码逻辑，并说明与单周期 CPU 的 ALU 实现的差异。

c) **数据通路设计：**阅读 *multi-cycle* 文件夹中的各个基础功能模块的Verilog代码，理解每个模块的输入输出接口和实现的基本功能。将这些基础功能电路模块进行实例化和连接（参考教材第 VI 章图 VI-28），完成多周期处理器整体数据通路 **MultiCycleCPU.v** 的 Verilog 代码实现。请写出数据通路中包含的寄存器（除寄存器堆之外）和多路选择器，结合代码简要说明它们的功能。

d) **功能验证（汇编程序分析-1）：**基于 InstAndDataMemory.v 中的指令代码（即 MIPS Assembly 1 的汇编程序），结合仿真波形图验证所设计的多周期处理器的功能正确性。

3. （MIPS 单周期和多周期 CPU 的性能对比，2 分）：

a) **汇编程序分析-2：**阅读并理解下面这段汇编程序。

| MIPS Assembly 2 | |
|---|---|
| 0 | addi $a0, $zero, 5 |
| 1 | xor $v0, $zero, $zero |

```
2           jal sum
      Loop:
3           beq $zero, $zero, Loop
      sum:
4           addi $sp, $sp, -8
5           sw $ra, 4($sp)
6           sw $a0, 0($sp)
7           slti $t0, $a0, 1
8           beq $t0, $zero, L1
9           addi $sp, $sp, 8
10          jr $ra
      L1:
11          add $v0, $a0, $v0
12          addi $a0, $a0, -1
13          jal sum
14          lw $a0, 0($sp)
15          lw $ra, 4($sp)
16          addi $sp, $sp, 8
17          add $v0, $a0, $v0
18          jr $ra
```

i. 如果第 0 行的 5 是任意正整数 n，这段程序能实现什么功能？Loop，sum，L1 各有什么作用？为每一句代码添加注释。

ii. 将这段汇编代码翻译成机器码并写出，分别完成单周期和多周期 CPU 的指令存储器文件的修改（请另外保存一份指令存储器文件）。

iii. 使用 ModelSim 或 Vivado 等仿真软件对单周期和多周期 CPU 进行仿真。运行足够长时间后，寄存器$a0，$v0 的值是多少？和你预期的程序功能是否一致？请给出仿真波形图。

b) **资源与性能对比：**面向“数逻实验课”所采用的 FPGA 板卡（参见其他说明-6），基于 Vivado 工具进行综合并开展静态时序分析，根据 Vivado 的资源和时序分析报告，对比单周期与多周期 CPU 的资源开销与时序性能。提交实验报告并分析说明两种 CPU 所可能达到的最高时钟频率、单次计算（针对汇编程序-2）所需要的最低延时、和所使用的硬件资源开销，并附上 Vivado 的综合分析报告截图。

4. （附加题-多周期 CPU 的状态修改，3 分）参考教材中“单周期与多周期处理器”章节习题的第 20 小题，完成以下实验：

a) 在原始多周期 CPU 状态转移图的基础上，如果寄存器更新操作可以和读内存或是 ALU 操作在同一个时钟周期内完成，哪些状态是可以合并的（结合你画的状态转移图来说明）？请完成多周期 CPU 控制器 **Controller.v** 的修改（请思考数据通路是否也需要修改），并针对汇编程序-2 验证功能正确性。请对比修改前后的最高时钟频率、单次计算所需要的最低延时、和所使用的硬件资源开销，并附上 Vivado 的综合分析报告截图。

b) 在原始多周期 CPU 状态转移图的基础上，如果有效地址计算和内存访问在同

一个时钟周期内完成，哪些状态是可以合并的（结合你画的状态转移图来说明）？请完成多周期 CPU 控制器 **Controller.v** 的修改（请思考数据通路是否也需要修改），并针对汇编程序-2 验证功能正确性。请对比修改前后的最高时钟频率、单次计算所需要的最低延时、和所使用的硬件资源开销，并附上 Vivado 的综合分析报告截图。

c) 对比三种多周期 CPU（原始的和上述两种实现）的实现，针对汇编程序-2 来说，哪一种实现更快？是否存在不同的汇编程序可以使得另外一种实现更快，如果是请给出相应的汇编程序实例，并附上仿真波形图和延时分析说明。

四、实验结果与提交材料

1. 针对实验内容 1、2、3，请完成所有的题目并撰写实验报告（word 或者 pdf 版本）。单周期 CPU 和多周期 CPU 的代码文件（包括 Verilog 源代码文件、testbench 测试代码文件和 xdc 约束文件）分别存放在 *single-cycle* 和 *multi-cycle* 文件夹下。如果有重名的代码文件，比如对应不同汇编程序的指令存储器文件，请以不同后缀（比如-1/-2）来区分命名，并在实验报告中说明。
2. 针对实验内容 4（附加题），请完成相应的题目并撰写实验报告（word 或者 pdf 版本）。所有相关的代码文件请存放在 *extra-bonus* 文件夹下。如果有重名的代码文件，比如对应不同汇编程序的指令存储器文件以及多周期状态控制器文件，请以不同后缀（比如-1/-2）来区分命名，并在实验报告中说明。

五、其他说明

1. 本学期处理器大作业分为两部分，第一部分占比 7 分，第二部分占比 8 分（附加题 3 分），总分为 15 分（超过 15 分按 15 分计）。
2. 处理器大作业第一部分的提交时间为 5 月 2 日至 **5 月 22 日 23 点 59 分**，第二部分的提交时间为 5 月 16 日至 **6 月 5 日 23 点 59 分**，晚交将直接影响最终成绩（直接在总分上扣除），最终扣分规则以网络学堂公告为准。
3. 我们鼓励讨论，但是要求所有代码与实验报告均独立完成，严禁抄袭！如发现抄袭现象，将上报学校教务处进行处理。
4. 我们将在大作业提交时间截止之后安排随机抽查，要求与助教当面解释代码设计思路，具体时间安排以网络学堂公告为准。
5. 如对本次处理器大作业有任何问题或建议，请发送邮件至曾书霖助教邮箱（zengsl18@mails.tsinghua.edu.cn），或在答疑时间进行答疑。
6. 针对部分没有选修“数逻实验课”的同学，我们提供了数逻实验课的相关资料（如下清华云盘链接所示），供各位同学**自行学习** Verilog 语法和相关工具的使用。
   链接：https://cloud.tsinghua.edu.cn/f/cec824dad54b4dcf9c2a/