P4　Verilog 单周期 CPU 设计文档

一、　整体结构

本文档所描述的处理器为 32 位单周期处理器，采用 Verilog HDL 实现。该处理器支持的指令集为{addu, subu, ori, lw, sw, beq, lui, jal, jr, nop}，并进行了适当扩展。

该处理器采用模块化和层次化设计，包含 Controller（控制器）、PC（程序计数器）、IM（指令存储器）、Adder（加法器）、NPC（下一条指令地址计算单元）、GRF（通用寄存器组）、CMP（比较单元）、ALU（算术逻辑单元）、DM（数据存储器）、 EXT（位扩展器）等基本部件。处理器顶层有效驱动信号有时钟信号 clk 和复位信号 reset。

二、　模块规格

1. PC（程序计数器）

1) 基本描述

PC 存储当前指令地址，并在时钟上升沿更新值。PC 的容量为 32bit*1024。

2) 端口说明

表 1 PC 端口说明

信号名	方向	描述
Clk	I	时钟信号。
Reset	I	复位信号（高电平有效）。
En	I	写使能信号（高电平有效）。
In[31:0]	I	输入下一个时钟上升沿 PC 要写入的值。
Out[31:0]	O	输出当前 PC 的值。

3) 功能定义

表 2 PC 功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来时，若复位信号有效，PC 被设置为 0x00003000。
2	输出当前指令地址	存储并由 Out 输出当前指令地址。
3	更新 PC	当时钟上升沿到来时，若写使能信号有效且复位信号无效，则更新 PC 为 In 的值。

2. IM（指令存储器）

1) 基本描述

IM 存储 CPU 要执行的指令，并输出输入地址所对应的指令。IM 的容量为 32bit*1024。

2) 端口说明

表 3 IM 端口说明

信号名	方向	描述
Addr[31:0]	I	输入指令地址。
Instr[31:0]	O	输出 Addr 所对应的指令。

3) 功能定义

表 4 IM 功能定义

序号	功能名称	功能描述
1	输出当前指令	由 Instr 输出 Addr 所对应的指令。

3. Adder（加法器）

1) 基本描述

Adder 输入当前 PC 的值，输出 PC+4 的值。

2) 端口说明

表 5 Adder 端口说明

信号名	方向	描述
In[31:0]	I	数据输入信号，输入当前 PC 的值。
Out[31:0]	O	数据输出信号，输出 PC+4 的值。

3) 功能定义

表 6 Adder 功能定义

序号	功能名称	功能描述
1	PC+4	由 Out 输出 PC+4 的值。

4. NPC（下一条指令地址计算单元）

1) 基本描述

NPC 根据当前指令地址和相应的控制信号计算出下一条指令地址。

2) 端口说明

表 7 NPC 端口说明

信号名	方向	描述
NPCOp	I	指定 NPC 要执行的操作： 0：B 型指令； 1：J 型指令。
PC4[31:0]	I	输入 PC+4。
Imm26[25:0]	I	输入 26 位立即数，用于计算分支或跳转后 PC 的值。
Out[31:0]	O	输出计算出的下一条指令地址。

3) 功能定义

表 8 NPC 功能定义

序号	功能名称	功能描述
1	计算下一条指令地址	当 NPCOp 为 0 时，Out 输出 PC4+SignExt(Imm26[15:0]||0^2)； 当 NPCOp 为 1 时，Out 输出(PC4)[31:28]||Imm26||0^2。

5. GRF（通用寄存器组）

1) 基本描述

GRF 内部包括 32 个具有写使能和复位功能的寄存器。其中，0 号寄存器的值始终保持为 0，其他寄存器初始值均为 0。GRF 提供同时读取 2 个寄存器和写入 1 个寄存器的功能。

2) 端口说明

表 9 GRF 端口说明

信号名	方向	描述

Clk	I	时钟信号。
Reset	I	复位信号（高电平有效）。
A1[4:0]	I	地址输入信号 1，将其对应寄存器中存储的数据输出至 RD1。
A2[4:0]	I	地址输入信号 2，将其对应寄存器中存储的数据输出至 RD2。
A3[4:0]	I	地址输入信号 3，指定写入操作所对应的寄存器。
WD[31:0]	I	数据输入信号，即要写入寄存器中的数据。
RegWrite	I	写使能信号（高电平有效）。
RD1[31:0]	O	数据输出信号，输出 A1 对应寄存器中的 32 位数据。
RD2[31:0]	O	数据输出信号，输出 A2 对应寄存器中的 32 位数据。

注：在 Verilog 实现中增加了 WPC[31:0]输入，用于在线测试时输出 PC 的值。

3) 功能定义

表 10 GRF 功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来时，若复位信号有效，GRF 中的每一个寄存器都被设置为 0x00000000。
2	读取数据	读取 A1 和 A2 所对应寄存器中的数据至 RD1 和 RD2。
3	写入数据	当时钟上升沿到来时，如果 RegWrite 有效且复位信号无效，就将 WD 写入 A3 所对应的寄存器中。

6. CMP（比较单元）

1) 基本描述

CMP 对输入的两个操作数提供相等比较功能，并对输入的第一个操作数提供零比较功能，输出比较结果。

2) 端口说明

表 11 CMP 端口说明

信号名	方向	描述
A[31:0]	I	数据输入信号，输入 CMP 的第一个操作数。
B[31:0]	I	数据输入信号，输入 CMP 的第二个操作数。
Equal	O	相等标志信号（高电平有效），标志两操作数是否相等。
LTZ	O	小于 0 标志信号（高电平有效），标志 A 是否小于 0。
EQZ	O	等于 0 标志信号（高电平有效），标志 A 是否等于 0。

3) 功能定义

表 12 CMP 功能定义

序号	功能名称	功能描述
1	相等比较运算	若 A=B，则 Equal 信号有效；否则无效。
2	零比较运算	若 A<0，则 LTZ 信号有效；否则无效。 若 A=0，则 EQZ 信号有效；否则无效。

7. ALU（算术逻辑单元）

1) 基本描述

ALU 对输入的两个操作数提供 32 位加、减、或、与、或非、异或和移位运算以及小于置位功能，输出运算结果。

2) 端口说明

表 13 ALU 端口说明

信号名	方向	描述
A[31:0]	I	数据输入信号，输入 ALU 的第一个操作数。
B[31:0]	I	数据输入信号，输入 ALU 的第二个操作数。
ALUOp[3:0]	I	指定 ALU 所要进行的操作： 0000：A+B； 0001：A-B； 0010：A|B； 0011：A&B； 0100：~(A|B)； 0101：A^B； 0110：A<<B[4:0]； 0111：A>>B[4:0]； 1000：A>>>B[4:0]； 1001：(A<B)?1:0； 1010：((0|A)<(0|B))?1:0。
Result[31:0]	O	数据输出信号，输出 ALU 的计算结果。

3) 功能定义

表 14 ALU 功能定义

序号	功能名称	功能描述
1	加法	当 ALUOp 为 0000 时，Result 输出 A+B 的值。
2	减法	当 ALUOp 为 0001 时，Result 输出 A-B 的值。
3	或运算	当 ALUOp 为 0010 时，Result 输出 A|B 的值。
4	与运算	当 ALUOp 为 0011 时，Result 输出 A&B 的值。
5	或非运算	当 ALUOp 为 0100 时，Result 输出~(A|B)的值。
6	异或运算	当 ALUOp 为 0101 时，Result 输出 A^B 的值。
7	逻辑左移运算	当 ALUOp 为 0110 时，Result 输出 A<<B[4:0]的值。
8	逻辑右移运算	当 ALUOp 为 0111 时，Result 输出 A>>B[4:0]的值。
9	算术右移运算	当 ALUOp 为 1000 时，Result 输出 A>>>B[4:0]的值。
10	小于比较运算	当 ALUOp 为 1001 时，若 A<B，则 Result 输出 1；否则 Result 输出 0。
11	无符号小于比较运算	当 ALUOp 为 1010 时，若(0|A)<(0|B)，则 Result 输出 1；否则 Result 输出 0。

8. DM（数据存储器）

1) 基本描述

DM 用于存储数据，其容量为 32bit*1024，起始地址为 0x00000000。DM 支持同步复位功能，并且数据读取和写入端口分离。

2) 端口说明

表 15 DM 端口说明

信号名	方向	描述
Clk	I	时钟信号。
Reset	I	复位信号（高电平有效）。

Addr[31:0]	I	地址信号，指定要操作的存储单元的地址。
WD[31:0]	I	数据输入信号，输入要写入到 Addr 所对应的存储单元的数据。
MemWrite	I	写使能信号（高电平有效）。
OpWidth[1:0]	I	指定操作位宽： 00：Word； 01：Half； 10：Byte。
LoadSigned	I	指定是否进行有符号读取（高电平有效）。
RD[31:0]	O	数据输出信号，输出 Addr 所对应的存储单元的数据。

注：在 Verilog 实现中增加了 WPC[31:0]输入，用于在线测试时输出 PC 的值。

3) 功能定义

表 16 DM 功能定义

序号	功能名称	功能描述
1	同步复位	当时钟上升沿到来时，若复位信号有效，DM 中的每一个存储单元都被设置为 0x00000000。
2	读取	RD 根据 OpWidth 和 LoadSigned 信号输出 Addr 所对应的存储单元的数据。
3	写入	当时钟上升沿到来时，如果 MemWrite 有效且复位信号无效，就根据 OpWidth 信号将 WD 写入 Addr 所对应的存储单元中。

9. EXT（位扩展器）

1) 基本描述

EXT 用于将输入的 16 位立即数根据操作信号扩展成 32 位并输出。

2) 端口说明

表 17 EXT 端口说明

信号名	方向	描述
Imm16[15:0]	I	数据输入信号，输入要进行扩展的数据。
ExtOp[1:0]	I	符号扩展信号（高电平有效）。
Imm32[31:0]	O	数据输出信号，输出扩展后的数据。

3) 功能定义

表 18 EXT 功能定义

序号	功能名称	功能描述
1	无符号扩展	当 ExtOp 为 00 时，将 Imm16 无符号扩展至 32 位并输出至 Imm32。
2	符号扩展	当 ExtOp 为 01 时，将 Imm16 符号扩展至 32 位并输出至 Imm32。
3	左移 16 位	当 ExtOp 为 10 时，将 Imm16 左移 16 位并输出至 Imm32。

三、 数据通路设计

表 19 数据通路

部件	PC	IM	Adder	NPC		GRF				CMP		EXT	ALU		DM	
输入信号	In	Addr	In	PC4	Imm26	A1	A2	A3	WD	A	B	Imm16	A	B	Addr	WD

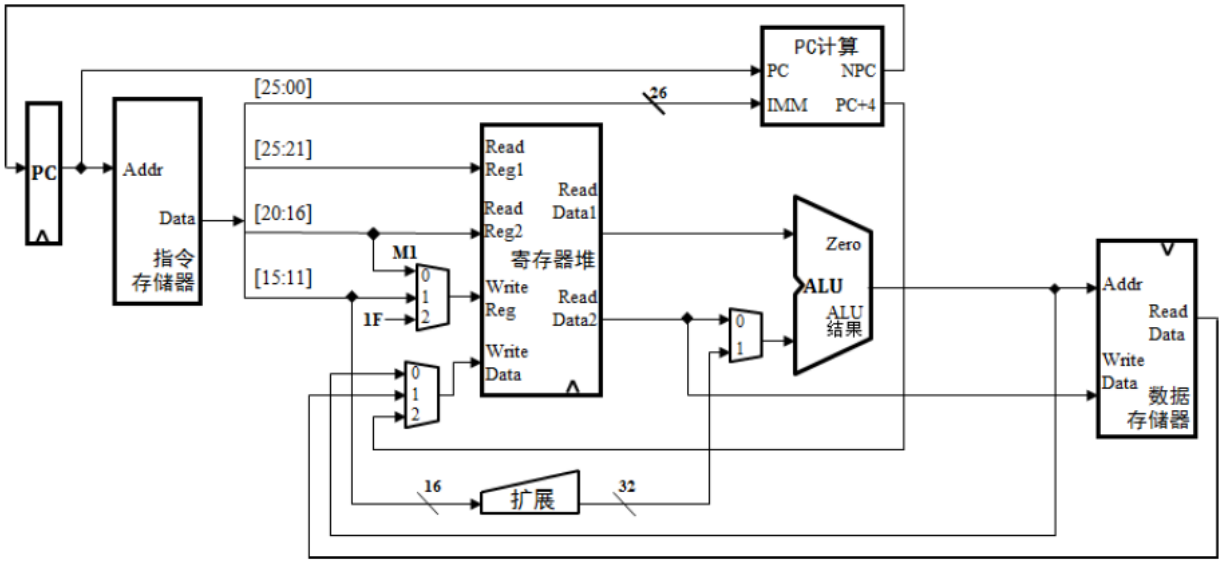
addu	Adder.Out	PC.Out	PC.Out			IM.Instr[25:21]	IM.Instr[20:16]	IM.Instr[15:11]	ALU.Result				GRF.RD1	GRF.RD2		
subu	Adder.Out	PC.Out	PC.Out			IM.Instr[25:21]	IM.Instr[20:16]	IM.Instr[15:11]	ALU.Result				GRF.RD1	GRF.RD2		
ori	Adder.Out	PC.Out	PC.Out			IM.Instr[25:21]		IM.Instr[20:16]	ALU.Result			IM.Instr[15:0]	GRF.RD1	EXT.Imm32		
lw	Adder.Out	PC.Out	PC.Out			IM.Instr[25:21]		IM.Instr[20:16]	DM.RD			IM.Instr[15:0]	GRF.RD1	EXT.Imm32	ALU.Result	
sw	Adder.Out	PC.Out	PC.Out			IM.Instr[25:21]	IM.Instr[20:16]					IM.Instr[15:0]	GRF.RD1	EXT.Imm32	ALU.Result	GRF.RD2
beq	Adder.Out NPC.Out	PC.Out	PC.Out	Adder.Out	IM.Instr[15:0]	IM.Instr[25:21]	IM.Instr[20:16]			GRF.RD1	GRF.RD2					
lui	Adder.Out	PC.Out	PC.Out			IM.Instr[25:21]		IM.Instr[20:16]	ALU.Result			IM.Instr[15:0]	GRF.RD1	EXT.Imm32		
jal	NPC.Out	PC.Out	PC.Out	Adder.Out	IM.Instr[25:0]			31	Adder.Out							
jr	GRF.RD1	PC.Out	PC.Out			IM.Instr[25:21]										
nop	Adder.Out	PC.Out	PC.Out													
综合	Adder.Out NPC.Out GRF.RD1	PC.Out	PC.Out	Adder.Out	IM.Instr[25:0]	IM.Instr[25:21]	IM.Instr[20:16]	IM.Instr[20:16] IM.Instr[15:11] 31	ALU.Result DM.RD Adder.Out	GRF.RD1	GRF.RD2	IM.Instr[15:0]	GRF.RD1	GRF.RD2 EXT.Imm32	ALU.Result	GRF.RD2

表 20 MUX 连接关系

名称	控制信号	输入 0	输入 1	输入 2	输出

MPC	PCSrc[1:0]	Adder.Out	NPC.Out	GRF.RD1	PC.In
MRegDst	RegDst[1:0]	IM.Instr[20:16]	IM.Instr[15:11]	31	GRF.A3
MRegSrc	RegSrc[1:0]	ALU.Result	DM.RD	Adder.Out	GRF.WD
MALUSrc	ALUSrc	GRF.RD2	EXT.Imm32		ALU.B

四、 数据通路参考示意图



五、 控制器（Controller）设计

1. 基本描述

控制器通过输入的 Op 和 Funct 信号以及 CMP 产生的比较信号产生数据通路所需的控制信号。

2. 真值表

表 21 控制器真值表

指令 (Op/Funct)	NPC Op	Reg Write	ALUOp [3:0]	Mem Write	OpWidth[1:0]	LoadSigned	ExtOp [1:0]	PCSrc[1:0]	RegDst [1:0]	RegSrc [1:0]	ALU Src
addu (000000/100001)	x	1	0000	0	xx	x	xx	00	01	00	0
subu (000000/100011)	x	1	0001	0	xx	x	xx	00	01	00	0
ori (001101)	x	1	0010	0	xx	x	00	00	00	00	1
lw (100011)	x	1	0000	0	00	x	01	00	00	01	1
sw (101011)	x	0	0000	1	00	x	01	00	xx	xx	1
beq (000100)	0	0	xxxx	0	xx	x	01	CMP.Equal ?01:00	xx	xx	x
lui (001111)	x	1	0000	0	xx	x	10	00	00	00	1

jal (000011)	1	1	xxxx	0	xx	x	xx	01	10	10	x
jr (000000/001000)	x	0	xxxx	0	xx	x	xx	10	xx	xx	x
nop (000000/000000)	x	0	xxxx	0	xx	x	xx	00	xx	xx	x

六、 CPU 测试

1. 测试程序

```
lui $t0, 49
ori $t0, 18
ori $s0, $0, 16
sw $t0, 4($0)
lw $t1, -12($s0)
sw $t0, -8($s0)
lw $t2, 8($0)
addu $t3, $t1, $0
subu $t4, $0, $t1
Label:
beq $t2, $0, Skip
lui $t5, 1
beq $t0, $0, Label
beq $t1, $t2, Skip
lui $t6, 1
Skip:
nop
jal Funct
ori $t9, $0, 0x3054
ori $s1, $0, 64
jr $t9

Funct:
ori $t8, $0, 16
jr $ra
```

2. 期望结果

@00003000: $ 8 <= 00310000

@00003004: $ 8 <= 00310012

@00003008: $16 <= 00000010

@0000300c: *00000004 <= 00310012

@00003010: $ 9 <= 00310012

@00003014: *00000008 <= 00310012

@00003018: $10 <= 00310012

@0000301c: $11 <= 00310012

@00003020: $12 <= ffceffee

@00003028: $13 <= 00010000

@0000303c: $31 <= 00003040

@0000304c: $24 <= 00000010

@00003040: $25 <= 00003054

@00003044: $17 <= 00000040

思考题

一、 L0.T2

1. 根据你的理解，在下面给出的 DM 的输入示例中，地址信号 addr 位数为什么是[11:2]而不是[9:0]？这个 addr 信号又是从哪里来的？

因为 DM 字长为 4 个字节，内部实现时采用一个 32 位 reg 来实现一个字的存储，故选择 reg 时应将地址左移两位。addr 信号来自 ALU 的计算结果。

2. 在相应的部件中，reset 的优先级比其他控制信号（不包括 clk 信号）都要高，且相应的设计都是同步复位。清零信号 reset 是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

PC，GRF 和 DM。PC 需要从指令存储器的起始地址开始计数，GRF 和 DM 需要全部初始化为 0。若不清零，这些部件内存储的值不确定，可能会使 CPU 无法正常工作。

二、 L0.T4

1. 列举出用 Verilog 语言设计控制器的几种编码方式（至少三种），并给出代码示例。

1) 利用 if-else（或 case）完成操作码和控制信号的值之间的对应。

例：

```
if (Op == 6'b000011) NPCOp = 1;
else NPCOp = 0;
```

2) 利用 assign 语句完成操作码和控制信号的值之间的对应。

例：

```
assign NPCOp = (Op == 6'b000011) ? 1 : 0;
```

3) 利用宏定义。

例：

```
'define JAL 6'b000011
assign NPCOp = (Op == JAL) ? 1 : 0;
```

2. 根据你所列举的编码方式，说明他们的优缺点。

1) 优点：结构清晰，易于理解；

缺点：可读性较差，编码较长；需要借助 always 块实现。

2) 优点：可独立于过程块，代码相对较短；

缺点：可读性较差。

3) 优点：可读性好，便于维护；

缺点：宏定义名称需要避免与变量冲突。

三、 L0.T5

1. C 语言是一种弱类型程序设计语言。C 语言中不对计算结果溢出进行处理，这意味着 C 语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持 C 语言，MIPS 指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi 与 addiu 是等价的，add 与 addu 是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的 Operation 部分。

   若忽略溢出，则 addi 与 add 均不会触发异常，指令的其余操作与 addiu 和 addu 分别等价。

2. 根据自己的设计说明单周期处理器的优缺点。

   优点：设计简单，不会产生冒险。

   缺点：所有指令的时钟周期等长，故时钟周期只能由最长的关键路径所决定，这对很多指令产生了时间浪费；在同一个时钟周期内，部件的空闲时间较长，没有得到充分利用。

3. 简要说明 jal、jr 和堆栈的关系。

   jal 主要用于调用过程，调用前需要在栈中保存相关的变量寄存器和返回地址；jr 主要用于过程返回，返回后需要从栈中取出保存的相关寄存器变量和返回地址。