计算机组成原理 P4 实验报告

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Verilog 实现的单周期 MIPS - CPU，支持的指令集包含{addu、subu、ori、lw、sw、beq、lui、jal、jr、nop}。为了实现这些功能，CPU 主要包含了 PC、NPC、IM、ALU、DM、EXT、GRF、CONTROL 等模块，其中前七个模块并列置于 DATAPATH 模块下，DATAPATH 与 CONTROL 模块并列置于顶层模块下。

（二）关键模块定义

1. PC

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号（同步复位至 0x00003000）
DI[31:0]	I	32 位输入次指令地址
DO[31:0]	O	32 位输出当前指令地址

2. NPC

信号名	方向	描述
PC[31:0]	I	32 位输入当前指令地址
Imm[25:0]	I	26 位输入用于跳转的立即数
NPCOp[1:0]	I	2 位输入用于选择 NPC
Zero	I	1 位输入辅助选择 NPC
RA	I	32 位输入来自寄存器的指令地址
NPC[31:0]	O	32 位输出次指令地址
PC4[31:0]	O	32 位输出返回值

3. IM

功能描述	指令存储器，内含 32*1024 字存储器，根据输入的地址输出指令	
信号名	方向	描述
ImAddr[31:0]	I	32 位输入当前指令地址
ImData[31:0]	O	32 位输出指令信号

4. ALU

信号名	方向	功能
A[32:0]	I	第一个 32 位操作数
B[32:0]	I	第二个 32 位操作数
ALUOp[3:0]	I	4 位输入 ALU 功能选择
C[31:0]	O	32 位输出计算结果
Zero	O	1 位输出判断结果

5. DM

功能描述	数据存储器，内含 32*1024 字存储器	
信号名	方向	描述
MemAddr[4:0]	I	5 位输入写入内存的地址
MemData[31:0]	I	32 位输入写入内存的数据
WE	I	写入使能
clk	I	时钟信号
reset	I	复位信号（同步复位）
PC[31:0]	I	32 位输入当前指令地址
RD[31:0]	O	32 位读出数据

6. EXT

信号名	方向	功能
Imm[25:0]	I	26 位输入立即数

EXTOp[1:0]	I	2 位输入控制 EXT 功能选择
Ext[31:0]	O	32 位输出扩展结果

7. GRF

信号名	方向	功能
RegA1[4:0]	I	第一个读出寄存器编号
RegA2[4:0]	I	第二个读出寄存器编号
RegA3[4:0]	I	回写寄存器的编号
RegData[31:0]	I	回写寄存器的值
WE	I	写入使能
clk	I	时钟信号
reset	I	复位信号
PC[31:0]	I	32 位输入当前指令地址
RD1[31:0]	O	第一个寄存器编号读出的值
RD2[31:0]	O	第二个寄存器编号读出的值

8. CONTROL

信号名	方向	功能
Func[5:0]	I	6 位输入
OpCode[5:0]	I	6 位输入
NPCOp[1:0]	O	2 位输出控制 NPC 选择
RegWrite	O	1 位输出控制 GRF 写使能
EXTOp[1:0]	O	2 位输出控制 EXT 功能选择
ALUOp[3:0]	O	4 位输出控制 ALU 功能选择
MemWrite	O	1 位输出控制 DM 写使能
RegA3Sel[1:0]	O	2 位输出用于 GRF 写入寄存器的选择
RegDataSel[1:0]	O	2 为输出用于 GRF 写入数据的选择
AluBSel[1:0]	O	2 位输出控制 ALU 第二个数据的选择

（三）数据通路的综合

1. 所有指令的指令级别数据通路

部件	PC	NPC			IM	RF				EXT	ALU		DM	
输入信号	DI	PC	Imm	RS	PC	RegA1	RegA2	RegA3	RegData	Imm	A	B	MemData	MemAddr
addu	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2		
subu	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2		
ori	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]	ALU.C	IM.D[25:0]	RF.RD1	EXT.Ext		
lw	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]	DM.RD	IM.D[25:0]	RF.RD1	EXT.Ext		ALU.C
sw	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]			IM.D[25:0]	RF.RD1	EXT.Ext	RF.RD2	ALU.C
beq	NPC.NPC	PC.DO	IM.D[25:0]		PC.DO	IM.D[25:21]	IM.D[20:16]				RF.RD1	RF.RD2		
lui	NPC.NPC	PC.DO			PC.DO			IM.D[20:16]	EXT.Ext	IM.D[25:0]				
jal	NPC.NPC	PC.DO	IM.D[25:0]		PC.DO			0x1f	NPC.PC4					
jr	NPC.NPC	PC.DO		RF.RD1	PC.DO	IM.D[25:21]								

2. 控制信号真值表

指令	NPCOp	RegWrite	EXTOp	ALUOp	MemWrite	RegA3Sel	RegDataSel	AluBSel
addu	00	1		0000		00	00	00
subu	00	1		0001		00	00	00
ori	00	1	00	0010		01	00	01
lw	00	1	01	0000		01	01	01
sw	00		01	0000	1			01
beq	01							00
lui	00	1	10			01	10	
jal	10	1				10	11	
jr	11							

3. 控制信号状态描述

NPCOp		EXTOp		ALUOp		RegA3Sel		RegDataSel		AluBSel	
控制信号	状态描述	控制信号	状态描述	控制信号	状态描述	控制信号	状态描述	控制信号	状态描述	控制信号	状态描述
00	PC+4	00	0 扩展	0000	加法运算	00	存入寄存器为 IM.D[15:11]	00	存入寄存器数据为 ALU.C	00	ALU 的第二个操作数为 RF.RD2
01	beq 次指令地址计算	01	符号扩展	0001	减法运算	01	存入寄存器为 IM.D[20:16]	01	存入寄存器数据为 DM.RD	01	ALU 的第二个操作数为 EXT.Ext
10	jal 次指令地址计算	10	立即数加载至高位	0010	按位或	10	存入 31 号寄存器	10	存入寄存器数据为 Ext		
11	来自寄存器的次地址							11	存入寄存器数据为 PC4		

（四）重要机制实现方法

1. 跳转

NPC 模块和 ALU 模块协同工作支持指令 x 的跳转机制。

二、测试方案

（一）典型测试样例

1. addu、subu 指令测试

MIPS 汇编指令	机器码
ori $1, $0, 0x1234 ori $2, $0, 0x5678 ori $3, $0, 21	34011234 34025678 34030015

ori $4, $0, 0xffff	3404ffff
ori $31, $0, 5	341f0005
ori $30, $0, 1	341e0001
lui $5, 0xaffc	3c05affc
lui $6, 0xaaac	3c06aaac
addu $7, $5, $6	00a63821
addu $8, $5, $1	00a14021
addu $9, $6, $2	00c24821
addu $10, $7, $3	00e35021
addu $11, $3, $4	00645821
subu $12, $7, $3	00e36023
subu $13, $2, $3	00436823
subu $14, $3, $31	007f7023
subu $15, $1, $31	003f7823
addu $16, $15, $31	01ff8021

Verilog 运行结果：

```
@00003000: $ 1 <= 00001234
@00003004: $ 2 <= 00005678
@00003008: $ 3 <= 00000015
@0000300c: $ 4 <= 0000ffff
@00003010: $31 <= 00000005
@00003014: $30 <= 00000001
@00003018: $ 5 <= affc0000
@0000301c: $ 6 <= aaac0000
@00003020: $ 7 <= 5aa80000
@00003024: $ 8 <= affc1234
@00003028: $ 9 <= aaac5678
@0000302c: $10 <= 5aa80015
```

```
@00003030: $11 <= 00010014
@00003034: $12 <= 5aa7ffeb
@00003038: $13 <= 00005663
@0000303c: $14 <= 00000010
@00003040: $15 <= 0000122f
@00003044: $16 <= 00001234
```

Mips 运行结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00001234
$v0	2	0x00005678
$v1	3	0x00000015
$a0	4	0x0000ffff
$a1	5	0xaffc0000
$a2	6	0xaaac0000
$a3	7	0x5aa80000
$t0	8	0xaffc1234
$t1	9	0xaaac5678
$t2	10	0x5aa80015
$t3	11	0x00010014
$t4	12	0x5aa7ffeb
$t5	13	0x00005663
$t6	14	0x00000010
$t7	15	0x0000122f
$s0	16	0x00001234
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000001
$ra	31	0x00000005
pc		0x00003048
hi		0x00000000
lo		0x00000000

2. ori、lui 指令测试

MIPS 汇编指令	机器码
ori $17, $0, 0	34110000
ori $18, $0, 256	34120100
ori $19, $18, 768	36530300
ori $17, $19, 0xffff	3671ffff
ori $28, $0, 1	341c0001
ori $20, $28, 5	37940005
ori $31, $28, 6	379f0006
ori $14, $20, 512	368e0200
ori $28, $14, 0xaffc	35dcaffc
ori $15, $30, 0xcbac	37cfcbac
lui $6, 1	3c060001
lui $7, 0xffff	3c07ffff
lui $8, 0xaffc	3c08affc
lui $9, 256	3c090100
lui $10, 0x1abc	3c0a1abc
lui $11, 0xbbca	3c0bbbca

Verilog 运行结果：

```
@00003000: $17 <= 00000000
@00003004: $18 <= 00000100
@00003008: $19 <= 00000300
@0000300c: $17 <= 0000ffff
@00003010: $28 <= 00000001
@00003014: $20 <= 00000005
@00003018: $31 <= 00000007
@0000301c: $14 <= 00000205
@00003020: $28 <= 0000affd
```

```
@00003024: $15 <= 0000cbac
@00003028: $ 6 <= 00010000
@0000302c: $ 7 <= ffff0000
@00003030: $ 8 <= affc0000
@00003034: $ 9 <= 01000000
@00003038: $10 <= 1abc0000
@0000303c: $11 <= bbca0000
```

MIPS 运行结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00010000
$a3	7	0xffff0000
$t0	8	0xaffc0000
$t1	9	0x01000000
$t2	10	0x1abc0000
$t3	11	0xbbca0000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000205
$t7	15	0x0000cbac
$s0	16	0x00000000
$s1	17	0x0000ffff
$s2	18	0x00000100
$s3	19	0x00000300
$s4	20	0x00000005
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x0000affd
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000007
pc		0x00003040
hi		0x00000000
lo		0x00000000

3. lw、sw 指令测试

MIPS 汇编指令	机器码
ori $1, $0, 12	3401000c
ori $2, $0, 16	34020010
ori $3, $0, 24	34030018
ori $4, $0, 0x0100	34040100
ori $5, $0, 0x0104	34050104
ori $6, $0, 0x0108	34060108
ori $17, $0, 0x1234	34111234
ori $18, $0, 0x5678	34125678
ori $19, $0, 0xbbbb	3413bbbb
ori $20, $0, 0xaffc	3414affc
lui $21, 0x4321	3c154321
lui $22, 0x9876	3c169876
lui $23, 0xcbbc	3c17cbbc
lui $24, 0xddab	3c18ddab
sw $17, 0($1)	ac310000
sw $18, 4($3)	ac720004
sw $19, 0($2)	ac530000
sw $20, 8($3)	ac740008
sw $21, 0($1)	ac350000
sw $22, 12($3)	ac76000c
sw $23, 0($2)	ac570000
sw $24, 16($3)	ac780010
lw $9, 0($1)	8c290000
lw $10, 0($2)	8c4a0000
lw $11, 0($3)	8c6b0000
lw $12, 4($1)	8c2c0004
lw $13, 8($3)	8c6d0008

lw $14, 12($3)	8c6e000c
lw $15, 4($2)	8c4f0004
lw $16, 16($3)	8c700010
sw $17, 0($4)	ac910000
sw $18, 4($6)	acd20004
sw $19, 0($5)	acb30000
sw $20, 8($6)	acd40008
sw $21, 0($4)	ac950000
sw $22, 12($6)	acd6000c
sw $23, 0($5)	acb70000
sw $24, 16($6)	acd80010
lw $9, 0($4)	8c890000
lw $10, 0($5)	8caa0000
lw $11, 0($6)	8ccb0000
lw $12, 4($4)	8c8c0004
lw $13, 8($6)	8ccd0008
lw $14, 12($6)	8cce000c
lw $15, 4($5)	8caf0004
lw $16, 16($6)	8cd00010

Verilog 运行结果：

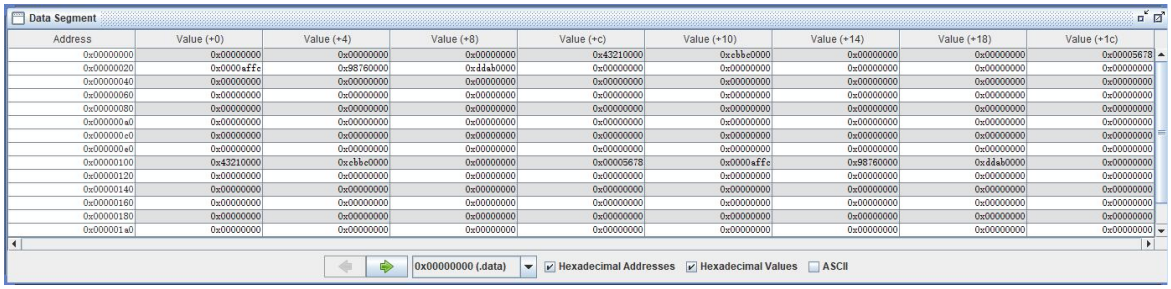
```
@00003000: $ 1 <= 0000000c
@00003004: $ 2 <= 00000010
@00003008: $ 3 <= 00000018
@0000300c: $ 4 <= 00000100
@00003010: $ 5 <= 00000104
@00003014: $ 6 <= 00000108
@00003018: $17 <= 00001234
@0000301c: $18 <= 00005678
```

```
@00003020: $19 <= 0000bbbb
@00003024: $20 <= 0000affc
@00003028: $21 <= 43210000
@0000302c: $22 <= 98760000
@00003030: $23 <= cbbc0000
@00003034: $24 <= ddab0000
@00003038: *0000000c <= 00001234
@0000303c: *0000001c <= 00005678
@00003040: *00000010 <= 0000bbbb
@00003044: *00000020 <= 0000affc
@00003048: *0000000c <= 43210000
@0000304c: *00000024 <= 98760000
@00003050: *00000010 <= cbbc0000
@00003054: *00000028 <= ddab0000
@00003058: $ 9 <= 43210000
@0000305c: $10 <= cbbc0000
@00003060: $11 <= 00000000
@00003064: $12 <= cbbc0000
@00003068: $13 <= 0000affc
@0000306c: $14 <= 98760000
@00003070: $15 <= 00000000
@00003074: $16 <= ddab0000
@00003078: *00000100 <= 00001234
@0000307c: *0000010c <= 00005678
@00003080: *00000104 <= 0000bbbb
@00003084: *00000110 <= 0000affc
@00003088: *00000100 <= 43210000
@0000308c: *00000114 <= 98760000
@00003090: *00000104 <= cbbc0000
@00003094: *00000118 <= ddab0000
```

```
@00003098: $ 9 <= 43210000
@0000309c: $10 <= cbbc0000
@000030a0: $11 <= 00000000
@000030a4: $12 <= cbbc0000
@000030a8: $13 <= 0000affc
@000030ac: $14 <= 98760000
@000030b0: $15 <= 00000000
@000030b4: $16 <= ddab0000
```

MIPS 运行结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x0000000c
$v0	2	0x00000010
$v1	3	0x00000018
$a0	4	0x00000100
$a1	5	0x00000104
$a2	6	0x00000108
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x43210000
$t2	10	0xcbbc0000
$t3	11	0x00000000
$t4	12	0xcbbc0000
$t5	13	0x0000affc
$t6	14	0x98760000
$t7	15	0x00000000
$s0	16	0xddab0000
$s1	17	0x00001234
$s2	18	0x00005678
$s3	19	0x0000bbbb
$s4	20	0x0000affc
$s5	21	0x43210000
$s6	22	0x98760000
$s7	23	0xcbbc0000
$t8	24	0xddab0000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x000030b8
hi		0x00000000
lo		0x00000000



4. beq、nop 指令测试

MIPS 汇编指令	机器码
ori $1, $0, 0x1111	34011111
ori $2, $0, 0xaffc	3402affc
ori $31, $0, 4	341f0004
lui $3, 0x1234	3c031234
lui $4, 0xaffc	3c04affc
addu $8, $1, $2	00224021
nop	00000000
addu $10, $2, $1	00415021
beq $3, $4, label1	10640004
subu $20, $8, $1	0101a023
sw $20, 0($31)	aff40000
nop	00000000
beq $8, $10, label2	110a0001
label1:	00235821
addu $11, $1, $3	8c1e0004
label2:	3419affc
lw $30, 4($0)	13220002
ori $25, $0, 0xaffc	00000000
beq $25, $2, label3	033fc021
nop	3c171234
addu $24, $25, $31	12e20002
label3:	00000000
lui $23, 0x1234	3c101123

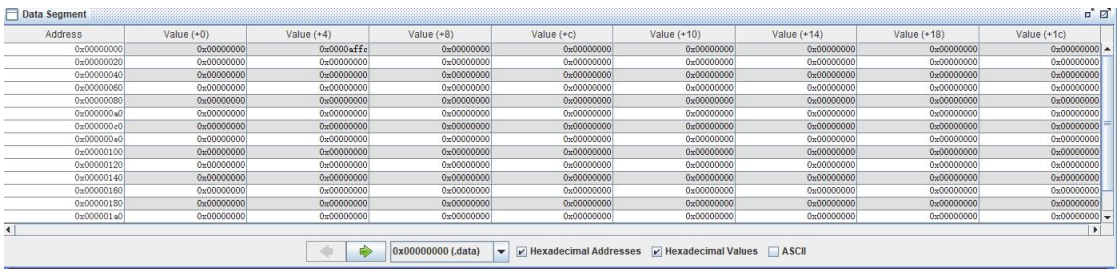
beq $23, $2, label4 nop lui $16, 0x1123 label4: nop lui $27, 0x1000	00000000 3c1b1000

Verilog 运行结果：

```
@00003000: $ 1 <= 00001111
@00003004: $ 2 <= 0000affc
@00003008: $31 <= 00000004
@0000300c: $ 3 <= 12340000
@00003010: $ 4 <= affc0000
@00003014: $ 8 <= 0000c10d
@0000301c: $10 <= 0000c10d
@00003024: $20 <= 0000affc
@00003028: *00000004 <= 0000affc
@00003038: $30 <= 0000affc
@0000303c: $25 <= 0000affc
@0000304c: $23 <= 12340000
@00003058: $16 <= 11230000
@00003060: $27 <= 10000000
```

MIPS 运行结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00001111
$v0	2	0x0000affc
$v1	3	0x12340000
$a0	4	0xaffc0000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x0000c10d
$t1	9	0x00000000
$t2	10	0x0000c10d
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x11230000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x0000affc
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x12340000
$t8	24	0x00000000
$t9	25	0x0000affc
$k0	26	0x00000000
$k1	27	0x10000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x0000affc
$ra	31	0x00000004
pc		0x00003064
hi		0x00000000
lo		0x00000000



5. jal、jr 指令测试

MIPS 汇编指令	机器码
ori $1, $0, 0x3048	34013048
ori $2, $0, 256	34020100
ori $3, $0, 0xffff	3403ffff

ori $4, $0, 1	34040001
addu $8, $2, $4	00444021
addu $10, $8, $8	01085021
subu $11, $10, $2	01425823
addu $12, $8, $4	01046021
jal label	0c000c0d
beq $11, $12, label2	116c0001
ori $21, $0, 145	34150091
label2:	3c191853
lui $25, 0x1853	00200008
jr $1	340f007b
label:	3c101234
ori $15, $0, 123	03e00008
lui $16, 0x1234	36315678
jr $ra	02119821
ori $17, 0x5678	0324d023
addu $19, $16, $17	0044d823
subu $26, $25, $4	0084e021
subu $27, $2, $4	
addu $28, $4, $4	

Verilog 运行结果：

```
@00003000: $ 1 <= 00003048
@00003004: $ 2 <= 00000100
@00003008: $ 3 <= 0000ffff
@0000300c: $ 4 <= 00000001
@00003010: $ 8 <= 00000101
@00003014: $10 <= 00000202
@00003018: $11 <= 00000102
```

```
@0000301c: $12 <= 00000102
@00003020: $31 <= 00003024
@00003034: $15 <= 0000007b
@00003038: $16 <= 12340000
@0000302c: $25 <= 18530000
@00003048: $26 <= 1852ffff
@0000304c: $27 <= 000000ff
@00003050: $28 <= 00000002
```

MIPS 运行结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00003048
$v0	2	0x00000100
$v1	3	0x0000ffff
$a0	4	0x00000001
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000101
$t1	9	0x00000000
$t2	10	0x00000202
$t3	11	0x00000102
$t4	12	0x00000102
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x0000007b
$s0	16	0x12340000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x18530000
$k0	26	0x1852ffff
$k1	27	0x000000ff
$gp	28	0x00000002
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00003024
pc		0x00003054
hi		0x00000000
lo		0x00000000

三、思考题

（一）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？

文件	模块接口定义

dm.v	dm(clk,reset,MemWrite,addr,din,dout);
	input clk; //clock
	input reset; //reset
	input MemWrite; //memory write enable
	input [11:2] addr; //memory's address for write
	input [31:0] din; //write data
	output [31:0] dout; //read data

答：在指令中，地址的偏移单位是 4，即每个单位存储一个字节，而 dm 的偏移单位是 1，即每个单位存储一个字，因此指令地址偏移是 dm 中地址的四倍，与此同时，在二进制表示中，四倍相当于左移两位，即 dm 中地址选取[11:2]位。

（二）思考 Verilog 语言设计控制器的译码方式，给出代码示例，并尝试对比各方式的优劣。

答：

（1） 宏定义、三位运算符、或运算相结合（本 CPU 设计所用方法）

```
`define R_op 6'b000000
`define ADDU_fu 6'b100001
`define BEQ 6'b000100
`define JAL 6'b000011
`define JR_fu 6'b001000
`define LUI 6'b001111
`define LW 6'b100011
`define ORI 6'b001101
`define SUBU_fu 6'b100011
`define SW 6'b101011
`define NOP 6'b000000
```

```
addu = (OpCode == `R_op && Func == `ADDU_fu) ? 1 : 0;
beq = (OpCode == `BEQ) ? 1 : 0;
jal = (OpCode == `JAL) ? 1 : 0;
jr = (OpCode == `R_op && Func == `JR_fu) ? 1 : 0;
lui = (OpCode == `LUI) ? 1 : 0;
lw = (OpCode == `LW) ? 1 : 0;
ori = (OpCode == `ORI) ? 1 : 0;
subu = (OpCode == `R_op && Func == `SUBU_fu) ? 1 : 0;
sw = (OpCode == `SW) ? 1 : 0;
nop = (OpCode == `R_op && Func == `NOP) ? 1 : 0;

NPCOp[1] = jal || jr;
NPCOp[0] = beq || jr;
RegWrite = addu || subu || ori || lw || lui || jal;
EXTOp[1] = lui;
EXTOp[0] = lw || sw;
ALUOp[3] = 0;
ALUOp[2] = 0;
ALUOp[1] = ori;
ALUOp[0] = subu;
MemWrite = sw;
RegA3Sel[1] = jal;
RegA3Sel[0] = ori || lw || lui || jal;
RegDataSel[1] = jal || lui;
RegDataSel[0] = lw || jal;
AluBSel[1] = 0;
AluBSel[0] = ori || lw || sw;
```

（2）if-else 句式

```
//addu
if(OpCode == 6'b000000 && Func == 6'b100001)begin
   NPCOp = 2'b00;
   RegWrite = 1;
   EXTOp = 2'b00;
   ALUOp = 4'b0000;
   MemWrite = 0;
   RegA3Sel = 2'b00;
   RegDataSel = 2'b00;
   AluBSel = 2'b00;
end
//lw
if(Func == 6'b100011)begin
   NPCOp = 2'b00;
   RegWrite = 1;
   EXTOp = 2'b01;
   ALUOp = 4'b0000;
   MemWrite = 0;
   RegA3Sel = 2'b01;
   RegDataSel = 2'b01;
   AluBSel = 2'b01;
end
//jal
if(OpCode == 6'b000000 && Func == 6'b100011)begin
   NPCOp = 2'b10;
   RegWrite = 1;
   EXTOp = 2'b00;
   ALUOp = 4'b0000;
   MemWrite = 0;
   RegA3Sel = 2'b10;
   RegDataSel = 2'b11;
   AluBSel = 2'b00;
end
```

（3）assign 语句（跟学长学的，想不到想不到）

```
assign
{NPCOp,RegWrite,EXTOp,ALUOp,MemWrite,RegA3Sel,RegDataSel,AluBSel} =
(OpCode == 6'b000000 && Func == 6'b100001) ? 16'b0010000000000000 :  //addu
(OpCode == 6'b000000 && Func == 6'b100011) ? 16'b0010000010000000 :  //subu
(OpCode == 6'b000000 && Func == 6'b001000) ? 16'b1100000000000000 :  //jr
(OpCode == 6'b001101) ? 16'b0010000100010001 :  //ori
(OpCode == 6'b100011) ? 16'b0010100000010101 :  //lw
(OpCode == 6'b101011) ? 16'b0000100001000001 :  //sw
(OpCode == 6'b000100) ? 16'b0100000000000000 :  //beq
(OpCode == 6'b001111) ? 16'b0011000000011000 :  //lui
                        16'b1010000000101100    //jal
```

比较：第一种方式便于添加指令但相对混乱（或许可以通过增添空格等方式手动对齐），第二种方式逻辑很清晰但代码过长可读性差，第三种方式代码相对简洁但可读性更差（16 位的编码 debug 直接眼瞎），总的来说宏定义能节省大量时间、不易出错且便于修改在三中译码方式中都建议使用，相比之下我更偏好第一种方式，一定程度上可能也是因为其与 logisim 的逻辑是一脉相承的。

（三）题目：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 所驱动的部件具有什么共同特点？

答：reset 驱动的部件本质上都是记忆性部件（logisim 里对应的 Memory Library），且都支持写入和读取，reset 信号为 1 时，在时钟上升沿部件内所有数据返回到初始值（对于 PC 来说是 0x00003000，对于其余部件来说是 0）。

（四）题目：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。

答：从相关指令的具体操作来看，add 和 addi 判断溢出的具体方法是在 32 位数前再接一位符号位并将其与加法执行后的第 31 位比较（从 0 位计），若不溢出，计算结果仍取 0-31 位。我的理解是判断溢出的功能只是增添了一位用于判断的数，而并没有对原数据计算结果的 0-31 造成影响，因此在不考虑溢出即程序员可以保证程序不会有溢出时 add 和 addi 均可正常产生结果且结果分别与 addu 和 addiu 相同。即可以认为在忽略溢出的前提下，addi 与 addiu 是等价的，add 与 addu 是等价的。

（五）题目：根据自己的设计说明单周期处理器的优缺点。

答：优点：设计简单，各模块内聚程度高，模块间耦合程度小。大部分模块内部是简单的组合逻辑，只需要建立好逻辑清晰的数据通路，进而完成各模块内部的逻辑即可保证较低的错误率，增添指令时只需考虑各模块内部真值表的补充，实现简单。缺点：时钟频率受各部分组合逻辑的制约，为保证在一个时钟周期内完成任一条指令的执行，需设置时钟周期大于等于执行时间最长的一条指令的延迟时间（本设计中该指令是 lw），这将导致在执行其他指令时大部分时间被浪费，CPU 速度较慢，性能不高。

四、debug 记录

（一）display 放置问题

最开始我懒于在 GRF 和 DM 模块中添加 PC 端口，只是将 display 代码放置在了 DATAPATH 模块，这导致在 reset 高位时本不应有写入功能（但此时可能出现写使能为 1 的情况），自然不应有 display 的执行，但在我的设计中 display 的执行与否只依赖于写使能的判断，这导致在 reset 为 1 的时候出现了一些多余的输出。

之后我将 display 功能分别放入了 GRF 与 DM 模块内，在模块内部 reset 具有最高优先级，display 是在 reset 为 0 与写使能为 1 的双重前提下执行的，因此问题得到解决。

（二）控制信号设置问题

在解决了 display 的问题后，我又遇到了三个 bug，一个是因为设计文档中控制信号打错，两个是因为 verilog 代码中控制信号敲错，不得不说，仔细真的很重要。