计算机组成原理实验报告

20231164 张岳霖

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Logisim 实现的流水线 MIPS - CPU，支持的指令集包含{addu、subu、ori、lw、sw、beq、lui、nop}共 8 条。为了实现这些功能，CPU 主要包含了 PC、NPC、IM、Controller、Splitter、RF、EXT、ALU、DM。其中 Splitter 模块采用了和逻辑以及或逻辑分步实现。

（二）关键模块定义

1. PC

模块功能：程序计数器

表一 PC 模块定义

序号	信号名称	数据方向	位数	功能描述
1	Clk	I	1	时钟信号，时钟上升沿更新当前指令地址为下一条指令地址
2	Reset	I	1	异步复位信号，信号为 1 时将指令初始化为
3	Address_in	I	32	输入的地址信号
4	Address_out	O	32	输出的地址信号

PC 用一个 32 位带异步复位信号的寄存器即可实现。

2. NPC

模块功能：下一指令地址计算模块

表二 NPC 模块定义

序号	信号名称	数据方向	位数	功能描述

1	Pc	I	32	当前指令地址
2	Bimm	I	32	b 类指令已处理的跳转地址
3	Zero	I	1	Beq 指令两边是否相等，相等为零
4	Jimm	I	24	J 和 jal 指令未处理的跳转地址
5	Ra	I	32	Jr 和 jalr 得到的来自寄存器的跳转地址
6	NPCOp	I	2	NPC 选择信号，PC+4(0)，b 类指令已处理的跳转地址(1)，j 和 jal 未处理的跳转地址(2)，jr 和 jalr 得到的寄存器地址(3)
7	NPC	O	32	下一条指令的地址

NPC 内部实现逻辑为计算出所有可能的下一个指令地址值，再通过多路选择器选择出适当的地址进行输出，选择信号为 NPCOp。

注：

1. 处理 B 类指令时，输入的信号(Bimm)为已经对立即数做零扩展，并左移两位的值；
2. 处理 J 指令时，输入的信号(Jimm)为直接取自当前指令 0 至 25 位的立即数，在 NPC 内部的处理方法为对该立即数零扩展至 28 位，再左移两位作为低 28 位，以下一条地址的高 4 位作为高 4 位得到的地址。

3. IM

模块功能：指令存储器

表三 IM 模块定义

序号	信号名称	数据方向	位数	功能描述
1	Address	I	32	输入的地址
2	Instruction	O	32	输出地址对应的指令

IM 用 Logisim 中自带的 ROM 即可实现。

4. Controller

模块功能：单周期控制器

表四 Controller 模块定义

序号	信号名称	数据方向	位数	功能描述
1	Opcode	I	6	指令的高六位，用于判断指令类型
2	Funct	I	6	指令的低六位，用于判断指令类型
3	JudgeB	I	5	指令的 16-20 位，用于判断指令类型（处理 bltz 和 bgez）
4	Zero	I	1	判断两个操作数是否相等
5	NPCOp	O	2	NPC 选择信号，用于选择下一条指令，区分是否跳转(具体见 NPC 模块)
6	WRSel	O	2	寄存器写入地址选择[20:16](0),[15:11](1),31 号寄存器(2)
7	WDSel	O	2	寄存器写入数据选择，ALU 计算得到的数据(0),DM 中取出的数据(1),NPC 计算的下一个地址(2)
8	RFWE	O	1	GRF 写入使能信号，1 写入有效
9	EXTOp	O	2	控制对立即数的扩展方式：零扩展(0),符号扩展(1),部分位零扩展(2)
10	BSel	O	2	控制 ALU 中 B 操作数的输入，GRF 输出(0),扩展后的立即数(1)
11	ALUop	O	5	ALU 操作选择信号
12	DMWE	O	1	内存写入使能信号，1 写入有效

13	DMtype	O	2	内存操作类型：0(W),1(H),2(B)

控制器 controller 的设计过程本质是一个译码的过程，其功能是将每一条机器指令包含的信息转化为恰当的 CPU 控制信号，并将各个控制信号输出传递到 CPU 的各个功能单元，通过对功能单元的控制达到控制 CPU 运行进程的效果。本 CPU 中控制器的内部包含和逻辑和或逻辑两个子模块。其中，和逻辑的功能是通过指令判断输入的指令类型，或逻辑的功能是建立指令类型和控制信号的映射。和逻辑采用与门阵列实现，或逻辑采用或门阵列实现。

信号拓展：

Jal：添加 31 号寄存器写入信号，信号为 1 时，31 号寄存器写使能、添加跳转链接信号，为 1 时选择下一条指令地址写入 GRF 实现跳转并链接；

Add Sub：添加判断溢出信号，与 ALU 中得到的溢出信号做与运算；

5. Splitter

模块功能：分线器

表五 Splitter 模块定义

序号	信号名称	数据方向	位数	功能描述
1	Instruction	I	32	输入指令信号
2	Opcode	O	6	指令信号的 31-26 位
3	25-21rs	O	6	指令信号的 25-21 位
4	20-16rt	O	5	指令信号的 20-16 位
5	15-11rd	O	5	指令信号的 15-11 位
6	10-6	O	5	指令信号的 10-6 位
6	Funct	O	5	指令信号的 5-0 位
7	Imm Offset	O	16	指令信号的 15-0 位
8	J_index	O	26	指令信号的 25-0 位

分线器的功能是将从 IM 读入的指令进行分块分类输出，可以是 CPU 的设计更加美观整洁，其内部采用 Splitter 元件实现。

6. RF

模块功能：寄存器堆

表六 RF 模块定义

序号	信号名称	数据方向	位数	功能描述
1	Clk	I	1	时钟信号
2	Reset	I	1	异步复位信号，信号为 1 将 32 个寄存器全部清零
3	A1	I	5	地址输入信号，指定 32 个寄存器中的某一个，将其中存储的数据读入到 RD1
4	A2	I	5	地址输入信号，指定 32 个寄存器中的某一个，将其中存储的数据读入到 RD2
5	A3	I	5	地址输入信号，指定 32 个寄存器中的某一个作为写入的目标寄存器
6	RFWE	I	1	GRF 写入使能信号，1 写入有效
7	WD	I	32	32 位数据输入信号
8	RD1	O	32	输出 A1 指定的寄存器中的 32 位数据
9	RD2	O	32	输出 A2 指定的寄存器中的 32 位数据

寄存器堆是由 32 个 32 位寄存器组成的变量存储类元件，内部通过多路选择器控制输出对应的寄存器数值，多路分配器控制信号输入到恰当的寄存器，

7. EXT

模块功能：位扩展器

表七 位扩展器模块定义

序号	信号名称	数据方向	位数	功能描述

1	Imm Offset	I	16	待扩展数据
2	EXTOp	I	2	扩展方式选择信号：零扩展(0)，符号扩展(1)，部分位零扩展(2)
3	Out	O	32	扩展后的数据

位扩展器内部先分别进行三类扩展，再通过多路选择器选择输出恰当的扩展结果。

8. ALU

模块功能：算术运算单元

表八 ALU 模块定义

序号	信号名称	数据方向	位数	功能描述
1	A	I	32	操作数 1
2	B	I	32	操作数 2
3	ALUop	I	5	ALU 操作类型选择信号
4	Zero	O	1	运算结果是否为零
5	Res	O	32	运算结果

ALU 运算定义表

序号	输入信号	执行的操作
1.	0	两个操作数做加法
2.	1	两个操作数做减法
3.	2	两个操作数按位或
4.	3	第二个操作数的低 16 位作为输出的高 16 位，后面补零
5.	4	以下为预留端口，暂未定义
6.	5	
7.	6	
8.	7	
9.	8	
10.	9	

11.	10	
12.	11	
13.	12	
14.	13	
15.	14	
16.	15	
17.	16	

ALU 内部同时进行所有支持的计算，再通过多路选择器输出我们需要的结果。

9. DM

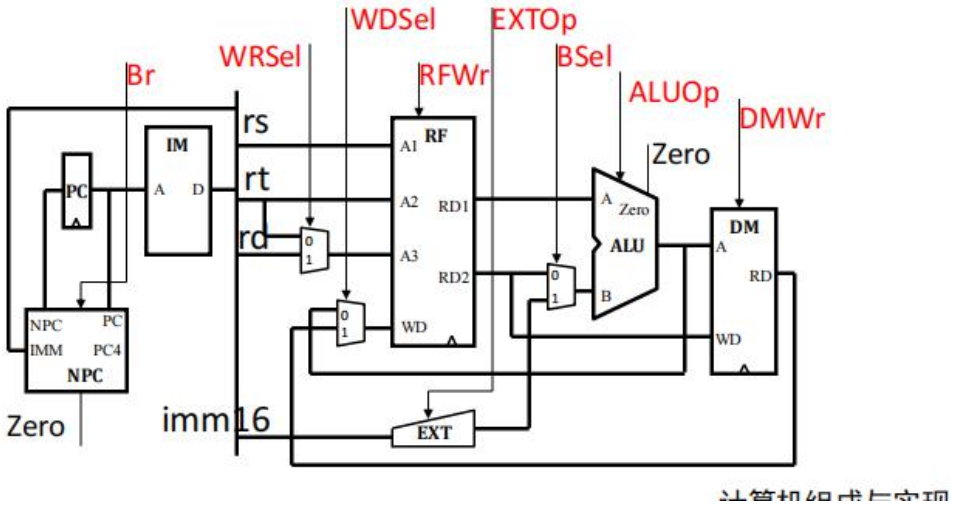
模块功能：数据存储器

表九 数据存储器模块定义

序号	信号名称	数据方向	位数	功能描述
1	Reset	I	1	异步复位信号
2	DMWE	I	1	内存写入使能信号，1 写入有效
3	Address	I	32	32 位地址输入信号
4	WD	I	32	32 位数据输入信号
5	DMtype	I	2	内存操作类型：0(W),1(H),2(B)
6	RD	O	32	输出 Address 指定地址对应的 32 位数据

数据存储器采用 Logisim 中的 RAM 即可实现

（三）控制模块设计



图一 控制信号功能图

控制器的设计本质是译码的过程，下面采取教程中的设计方法，将解码过程分解为和逻辑和或逻辑两个部分来单独进行控制器的构建。

1. 和逻辑真值表

R 型指令（addu subu）：前六位均为 0，可以通过后五位编码

I 型指令（ori lw sw beq lui）：可以直接通过前六位编码

Nop 指令：全零

所以我们可以将真值表分成两部分，第一部分是当前六位不全为零时，仅通过前六位判断指令类型；第二部分是当前六位全为零时，通过后六位判断指令类型，最后通过多路选择器选择应该输出的指令类型。

第一部分：

表十 和逻辑真值表第一部分

	Op5	Op4	Op3	Op2	Op1	Op0
Ori	0	0	1	1	0	1
Lw	1	0	0	0	1	1
Sw	1	0	1	0	1	1
Beq	0	0	0	1	0	0
Lui	0	0	1	1	1	1

第二部分：

表十一 和逻辑真值表第二部分

	Fu5	Fu4	Fu3	Fu2	Fu1	Fu0
Addu	1	0	0	0	0	1
Subu	1	0	0	0	1	1
Nop	0	0	0	0	0	0

2. 与逻辑真值表

表十二 与逻辑真值表

	NPCOp	WRSel	WDSel	RFWE	EXTOp	BSel	ALUop	DMWE
Addu	0	1	0	1	x	0	0	0
Subu	0	1	0	1	x	0	1	0
Ori	0	0	0	1	0	1	2	0
Lw	0	0	1	1	1	1	0	0
Sw	0	x	x	0	1	1	0	1
Beq	1	x	x	0	1	x	x	0
Lui	0	0	0	1	0	1	3	0
Nop	0	x	x	0	x	x	x	0

（三）重要机制实现方法

1. 跳转

采用了控制器分析输入指令，产生跳转信号；NPC 执行跳转信号，计算下一条指令所在位置的实现方法。值得注意的是，对于 B 型指令的跳转，本次实现采用将 EXT 扩展为 32 位的立即数左移两位处理完成后再输入到 NPC 的实现方法，NPC 中无需再次进行输入的立即数信号的处理，也就无需额外添加立即数扩展的功能，在一定程度上达到了元件复用的效果。

二、测试方案

（一）典型测试样例

根据指令的类型和操作数的正负大小的不同等，手工构造了一组测试点，在构造测试点的过程中，每个寄存器仅使用一次，Logisim 中运行完成后，比对 Logisim 中寄存器中的值和 Mars 中运行的结果。

而后，我采用了讨论区中同学的代码生成器，生成了 10 组 Mars 代码，并编写了自动测试工具与同学的 CPU 进行对拍测试。

测试样例附在附录一中（#1 为自行构造的测试样例，#2 为一组自动生成的测试样例）。

（二）自动测试工具

1. 测试样例生成器

采用了讨论区中“吴湛宇 19374006”同学的测试样例生成器自动生成了 10 组数据对构建的小型 CPU 进行强测。

2. 自动执行脚本

```
import os
import re


filename = input('请输入文件名：')

CPUname = input('请输入 CPU 名字：')

os.system("java -jar mars.jar " + filename + ".asm nc mc CompactTextAtZero a dump .text HexText " + filename + ".txt")
instruction = open(filename + ".txt").read()
cur = open(CPUname + ".circ", encoding='utf-8').read()
cur = re.sub(r'addr/data: 24 32([\s\S]*)</a>', "addr/data: 5 32\n" + instruction + "</a>", cur)
with open(CPUname + "test" + filename + ".circ", "w",
```

```
encoding='utf-8', ) as file:
        file.write(cur)
    os.system("java -jar logisim.jar" + CPUname + "test" +
filename + ".circ -tty table >resultof" + CPUname + filename
+ ".txt")
```

该脚本能够实现将 mars 中的代码指令自动以 16 进制形式导出并输入到 Logisim 中 IM 模块的 ROM 中，并将 Logisim 中输出的信号导出，方便我们进行对拍程序验证。

3. 正确性判定脚本

```
    import filecmp


    filename = input('请输入 MARS 文件名：')
    res = filecmp.cmp("resultofCPU0" + filename + ".txt",
"resultofCPU1" + filename + ".txt")
    print(res)
```

编写了一个简单的 Python 程序来对比两个 CPU 输出的结果进行正确性判定。

三、思考题

（一）

现在我们的模块中 IM 使用 ROM，DM 使用 RAM，GRF 使用 Register，这种做法合理吗？请给出分析，若有改进意见也请一并给出。

我认为这种实现方法是合理的。IM 的功能是存储指令，在 CPU 的运行过程中，IM 仅需要输出指令而无需写入指令，同时 ROM 在断电后其内部数据不会消失，能够保护指令。DM 是数据存储器，在 CPU 的整个运行过程中涉及到数据从内存中的读取和写入，这要求 DM 中的存储器能够支持读写操作，因此我们选择 RAM。对于 GRF，CPU 对几乎所有指令的操作都会涉及到寄存器，所以提高 GRF 的读取和写入速度是十分重要的，同时，GRF 仅需要保存我们经常使

用到的数据，32 个寄存器组几乎可以满足我们的需求，所以采用多个处理数据较快的寄存器搭建 GRF。

（二）

事实上，实现 nop 空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由。

首先，就我本身设计的 CPU 而言，使能端的默认值均为 0，NPC 的默认操作也为 PC+4，所以对于 nop 空指令，无需在控制信号中进行任何操作，我们即可实现 nop；

其次，nop 可以看作是特殊的 sll 指令，即将 0 号寄存器中的值左移 0 位存储到 0 号寄存器中，若 CPU 已经支持 sll 指令，我们并不需要将其加入真值表。

（三）

上文提到，MARS 不能导出 PC 与 DM 起始地址均为 0 的机器码。实际上，可以通过为 DM 增添片选信号，来避免手工修改的麻烦，请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

对于 PC 或者 DM 初始地址不为零的情况，我们仅需要在其输入信号输入到 ROM 或者 RAM 之前将其减去初始地址即可。比如当 DM 初始值不为零时，我们可以在 DM 的输入信号输入到 RAM 之前将其减去 DM 的初始值，并选择减去 DM 初始值之后的数据作为 RAM 地址的输入信号。

查阅资料得知，片选信号是指当很多芯片挂在同一总线上的时候，需要有一个信号来区别总线上的数据和地址由哪个芯片处理，这个信号被称为是片选信号。我们可以采用类似的设计，将 DM 可能的地址初始值通过不同的组合逻辑计算出来，得到多组处理后的数据，再添加片选信号作为多路选择器的选择信号，选择出适当的数据传递给 RAM。

（四）

除了编写程序进行测试外，还有一种验证 CPU 设计正确性的办法——形式验证。形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其

正确性或非正确性。请搜索“形式验证（Formal Verification)”了解相关内容后，简要阐述相比于测试，形式验证的优劣之处。

形式验证是通过数学证明的方式验证程序设计的正确性，它主要可以分为三大类：等价性检查，形式模型检查和定理证明。

形式验证的优势：形式验证是使用的是严格的数学推导来证明设计的正确性。其优势主要体现在三点，其一，形式验证能够达到无死角完全检验数据的效果。即使是高强度的测试也未必能够达到对全部输入数据的完全检查，而形式验证通过严谨的数学证明，能够实现对所有可能输入信号的完全检查；其二，对于大部分的 CPU 设计，输入的指令是多种多样不计其数的，我们采用枚举验证的方法是不切实际而且十分低效的，形式验证能够相对快速的得到一种普适的推到结果，节省我们的测试时间；其三，我们在进行测试的过程中，总是会费尽脑汁思考程序数据的边界条件，而且一些边界特殊情况往往难以察觉，而形式验证则避开了构造测试数据的过程，通常情况下操作较为简便。

形式验证的缺点：形式验证的本质是一个静态的逻辑推导方法，其缺点主要体现在两个方面，第一，形式验证无法验证一些动态行为，对于含有时序逻辑的模块，较难应用形式验证方法。同时，形式验证的静态特征还使得它无法有效分析电路的性能，如 CPU 的指令处理速度和功耗等等；第二，随着模块设计复杂度的增加，形式验证的困难程度将以指数函数呈爆炸性增长，逻辑推导将变得十分困难。

总而言之，形式验证是一种严谨便捷的正确性验证方法，对于一些中小型的静态逻辑验证比数据测试方便得多。然而对于大型模块的逻辑推理和动态模块的验证，形式验证可能不是一个合适的方法。在 CPU 设计和验证的过程中，我们可以采用形式验证和程序测试相互结合的方法，发挥出这两种方法各自的长处，从而高效严谨的对我们设计的 CPU 进行正确性验证。

附录一：测试样例

#1

```
ori $at,123
ori $v0,456  # 构造正数

lui $v0,0xffff
ori $v1,$v0,456
lui $a0,0xffff
ori $a1,$a0,795  # 构造负数

addu $a2,$at,$v0  # 正正
addu $a3,$v1,$a1  # 负负
addu $t0,$a2,$a3  # 正负
addu $t1,$a3,$a2  # 负正

subu $t2,$a2,$at  # 正正
subu $t3,$a1,$a3  # 负负
subu $t4,$a2,$a3  # 正负
subu $t5,$a3,$t4  # 负正

ori $t6,0
ori $t7,1
nop
sw $t2,0($t6)
sw $t3,8($t6)
sw $t4,12($t6)
```

```
sw $t5,4($t6)

nop

lw $s0,12($t6)
lw $s1,8($t6)
lw $s2,4($t6)
lw $s3,0($t6)
loop:
   addu $t6,$t6,$t7
   addu $s4,$s0,$s1
   subu $s5,$t0,$s2
   addu $s7,$t9,$s3
   beq $t6,$t7,loop
```

机器码：

```
v2.0 raw
3421007b
344201c8
3c02ffff
344301c8
3c04ffff
3485031b
00223021
00653821
00c74021
00e64821
00c15023
00a75823
00c76023
```

```
00ec6823
35ce0000
35ef0001
00000000
adca0000
adcb0008
adcc000c
adcd0004
00000000
8dd0000c
8dd10008
8dd20004
8dd30000
01cf7021
0211a021
0112a823
0333b821
11cffffb
```

#2

```
ori $1,$0,67
ori $18,$0,27
ori $13,$0,42
ori $15,$0,104
ori $20,$0,21
ori $25,$0,116
ori $21,$0,16
ori $7,$0,28
lw $14,380($15)
```

```
beq $27,$16,branch1
nop
ori $11,$17,57570
lui $0,35002
ori $22,$7,18261
subu $17,$23,$0
subu $26,$5,$25
subu $23,$6,$20
lui $3,12903
lw $14,469($18)
lw $16,-56($15)
lui $23,7317
lui $26,13604
beq $20,$19,branch2
subu $5,$20,$24
sw $18,104($25)
lw $0,425($1)
subu $22,$4,$0
nop
beq $25,$25,branch3
lw $0,308($25)
addu $9,$19,$27
branch3:
lw $0,241($1)
subu $0,$27,$17
subu $2,$7,$7
ori $0,$5,57692
ori $9,$17,54733
branch2:
beq $16,$11,branch4
```

```
sw $0,137($1)
addu $11,$0,$17
sw $10,96($21)
beq $19,$22,branch5
nop
beq $10,$21,branch6
lw $10,315($20)
lw $3,-16($25)
lw $0,-17($20)
addu $6,$3,$21
addu $14,$21,$23
lui $16,10529
addu $6,$14,$9
sw $25,-23($18)
nop
subu $24,$22,$25
nop
lw $27,133($1)
lui $26,52715
subu $27,$0,$15
sw $17,404($7)
beq $22,$24,branch7
subu $16,$14,$19
nop
sw $27,286($13)
sw $18,-38($13)
sw $0,-13($20)
subu $14,$0,$26
ori $19,$11,25968
nop
```

```
nop
lui $23,21366
addu $24,$24,$9
branch5:
subu $22,$10,$16
addu $3,$5,$25
beq $20,$21,branch8
lw $16,199($20)
lw $5,-2($13)
sw $7,28($21)
ori $0,$10,42770
lui $14,40557
branch4:
sw $25,312($21)
ori $22,$18,39501
branch7:
beq $27,$21,branch9
beq $6,$21,branch10
nop
subu $12,$23,$25
lw $6,278($13)
addu $27,$11,$19
addu $17,$22,$0
subu $17,$21,$3
branch1:
lui $6,36307
branch9:
lw $17,329($1)
subu $14,$17,$20
beq $4,$0,branch11
```

```
branch6:
addu $23,$6,$6
lui $14,21831
beq $2,$25,branch12
sw $4,112($25)
sw $24,288($21)
addu $16,$18,$14
nop
nop
lui $17,35845
subu $23,$5,$2
branch12:
sw $22,374($13)
addu $6,$13,$19
nop
lw $11,53($1)
sw $25,136($15)
beq $21,$20,branch13
ori $0,$17,8134
nop
lw $2,276($7)
beq $26,$24,branch14
ori $9,$0,21820
addu $23,$5,$0
branch8:
nop
branch11:
lw $22,52($7)
sw $23,383($20)
branch14:
```

```
sw $27,-32($15)
subu $0,$4,$26
addu $5,$18,$13
ori $0,$27,7369
lw $22,467($20)
branch10:
lui $17,5347
branch13:
subu $14,$6,$14
sw $16,157($1)
addu $16,$23,$27
ori $27,$19,64468
nop
```

Logisim输出结果：

```
0011 0100 0001 1100 0000 0000 0000 0000  1  1 1100   0000
0000 0000 0000 0000 0000 0000 0000 0  0 0000   0000 0000
0000 0000 0000 0000 0000 0000
0011 0100 0001 1101 0000 0000 0000 0000  1  1 1101   0000
0000 0000 0000 0000 0000 0000 0000 0  0 0000   0000 0000
0000 0000 0000 0000 0000 0000
0011 0100 0000 0000 1000 0000 1111 1000  1  0 0000   0000
0000 0000 0000 1000 0000 1111 1000 0  1 1110   0000 0000
0000 0000 0000 0000 0000 0000
0011 0110 1110 1110 0000 0011 0110 0111  1  0 1110   0000
0000 0000 0000 0000 0011 0110 0111 0  1 1001   0000 0000
0000 0000 0000 0000 0000 0000
```

0011 0101 1001 0011 0111 0110 1001 1100 1 1 0011 0000
0000 0000 0000 0111 0110 1001 1100 0 0 0111 0000 0000
0000 0000 0000 0000 0000 0000

0011 0100 1101 1101 1010 0101 0011 1001 1 1 1101 0000
0000 0000 0000 1010 0101 0011 1001 0 0 1110 0000 0000
0000 0000 0000 0000 0000 0000

0011 0100 0110 0101 1100 0101 1101 1100 1 0 0101 0000
0000 0000 0000 1100 0101 1101 1100 0 1 0111 0000 0000
0000 0000 0000 0000 0000 0000

0011 0100 1011 1110 0000 1111 1111 0001 1 1 1110 0000
0000 0000 0000 1100 1111 1111 1101 0 1 1111 0000 0000
0000 0000 0000 0000 0000 0000

0011 0100 0111 0010 1010 1111 0001 1101 1 1 0010 0000
0000 0000 0000 1010 1111 0001 1101 0 0 0111 0000 0000
0000 0000 0000 0000 0000 0000

0011 0101 0101 0110 0110 0110 0000 0010 1 1 0110 0000
0000 0000 0000 0110 0110 0000 0010 0 0 0000 0000 0000
0000 0000 0000 0000 0000 0000

0011 0101 1111 1110 0111 0010 1010 0111 1 1 1110 0000
0000 0000 0000 0111 0010 1010 0111 0 0 1001 0000 0000
0000 0000 1100 1111 1111 1101

0011 0100 0011 0111 1111 1111 0010 0011 1 1 0111 0000
0000 0000 0000 1111 1111 0010 0011 0 0 1000 0000 0000
0000 0000 0000 0000 0000 0000

0011 0111 1011 1001 0110 0100 1001 1110 1 1 1001 0000
0000 0000 0000 1110 0101 1011 1111 0 0 1111 0000 0000
0000 0000 0000 0000 0000 0000

0011 0110 1001 0000 1011 1111 1001 1011 1 1 0000 0000
0000 0000 0000 1011 1111 1001 1011 0 0 0110 0000 0000
0000 0000 0000 0000 0000 0000

0011 0111 1111 1110 0100 1110 0000 1100 1 1 1110 0000
0000 0000 0000 0100 1110 0000 1100 0 0 0011 0000 0000
0000 0000 0111 0010 1010 0111
0011 0111 0001 0001 1111 0100 1001 1101 1 1 0001 0000
0000 0000 0000 1111 0100 1001 1101 0 0 0111 0000 0000
0000 0000 0000 0000 0000 0000
0011 1100 0001 0001 0011 1011 1111 1110 1 1 0001 0011
1011 1111 1110 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 1111 0100 1001 1101
0011 1100 0000 0101 0111 0100 0011 0011 1 0 0101 0111
0100 0011 0011 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 1100 0101 1101 1100
0011 1100 0000 0011 1100 0100 0101 0001 1 0 0011 1100
0100 0101 0001 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 0000 0000 0000 0000
0011 1100 0000 0100 1100 0100 0011 0001 1 0 0100 1100
0100 0011 0001 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 0000 0000 0000 0000
0011 1100 0001 1100 0010 0100 1101 1001 1 1 1100 0010
0100 1101 1001 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 0000 0000 0000 0000
0011 1100 0001 1011 1111 0000 1101 1010 1 1 1011 1111
0000 1101 1010 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 0000 0000 0000 0000
0011 1100 0000 1100 1100 1100 0110 1000 1 0 1100 1100
1100 0110 1000 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 0000 0000 0000 0000
0011 1100 0000 1010 0001 0001 1000 1101 1 0 1010 0001
0001 1000 1101 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 0000 0000 0000 0000

0011 1100 0001 0000 0010 0001 1110 1010 1 1 0000 0010
0001 1110 1010 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 1011 1111 1001 1011
0011 1100 0000 0000 0000 0100 1101 0011 1 0 0000 0000
0100 1101 0011 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 0000 0000 0000 0000
0011 1100 0001 0110 0001 0001 0101 1011 1 1 0110 0001
0001 0101 1011 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 0110 0110 0000 0010
0011 1100 0001 1000 0100 1111 0000 0110 1 1 1000 0100
1111 0000 0110 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 0000 0000 0000 0000
0011 1100 0001 0010 0001 0101 1101 1110 1 1 0010 0001
0101 1101 1110 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 1010 1111 0001 1101
0000 0000 0000 0000 0000 0000 0000 0000 0 0 0000 0000
0000 0000 0000 0000 0000 0000 0000 0 0 0000 0000 0000
0000 0000 0000 0000 0000 0000