编号66

南京航空航天大学

实验报告

题 目 单周期MIPS CPU的设计

学生姓名 李应飞

学 号 161610338

学 院 计算机科学与技术学院

专 业 计算机科学与技术

班 级 1616103

指导教师 施慧彬 副教授

二〇一八年七月

单周期MIPS CPU的设计

摘 要

为了提升对单周期 MIPS CPU 的理解能力和动手能力，以个人为单位，完成一个单周期 CPU 的开发。

研究的结果与主要结论：能写出一个能执行 add，sub，and，or，slt，lw，sw，beq 和 jump

指令，不支持溢出的单周期MIPS CPU。

关键词：单周期，处理器，汇编语言，模拟

The Design of a Single Cycle CPU in MIPS

Instruction Set

Abstract

To improve the understanding single-cycle MIPS CPU and ability of practice, complete a single cycle CPU in MIPS instruction set in personal.

The results of the study and experiment: make a single cycle CPU in MIPS instruction set with 9 instructions: add，sub，and，or，slt，lw，sw，beq and jmp, without overflow process.

Key Words：single cycle, processor, CPU, assembly language, simulation

目 录

摘 要 .................................................................... ⅰ Abstract ................................................................... ⅰ 第一章 引 言 ............................................................. 1

1.1 单周期CPU的理论结构 ................................................ 1

1.1.1 PC模块定义.................................................... 1

1.1.2 NPC模块定义................................................... 2

1.1.3 ALU模块定义................................................... 3

1.1.4 MUXtip模块定义.................................................4

1.1.5 registerFile模块定义...........................................4

1.1.6InstructionMemory模块定义...................................... 5

1.1.7 dataMemory模块定义................................... ........ 5

1.1.8 contrl模块定义.................................................6

1.1.9 signext模块定义............................................... 7

1.2 各个指令与控制信号之间的关系 ........................................ 8

1.2.1 R-type指令与控制信号.......................................... 8

1.2.2 I-type指令与控制信号.......................................... 8

1.2.3 Load指令与控制信号............................................ 9

1.2.4 Store指令与控制信号........................................... 9

1.2.5 分支指令与控制信号 ........................................... 10

1.2.6 各指令控制信号取值 ........................................... 10 第二章 单周期MIPS CPU的具体设计与调试 ................................... 12

2.1 具体Verilog代码实现 ............................................... 12

2.2 ModelSim模拟及测试 ................................................. 23

2.2.1 综合测试用代码 ............................................... 23

2.2.2 各个模块的testbench.......................................... 24

第三章 总结与展望 ........................................................ 30 参考文献 .................................................................. 31 附录 ...................................................................... 31

# 引 言

1. 数据通路：

## 1.1 单周期CPU的理论结构

单周期处理器的数据通路结构如图所示。

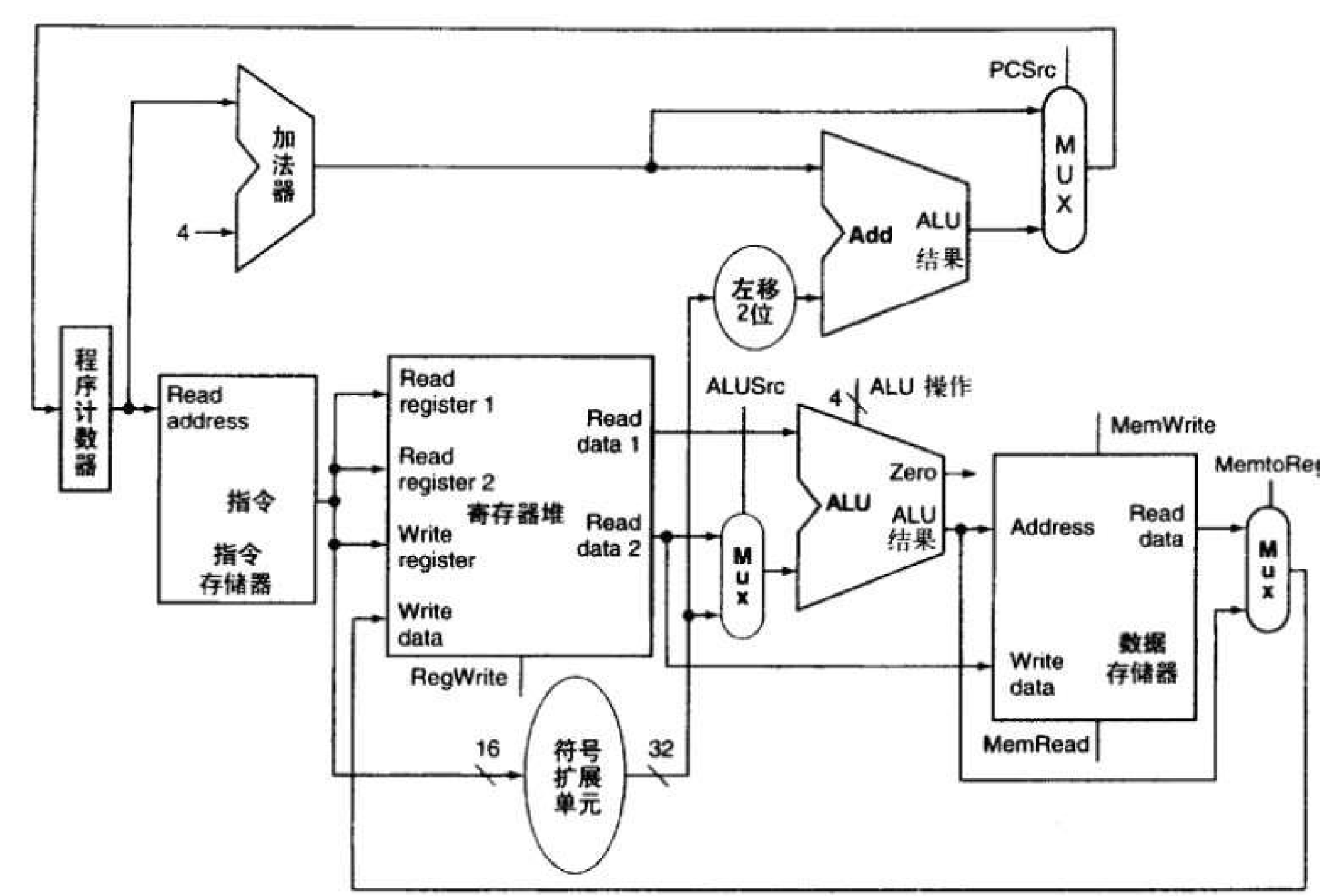


图1.1 单周期处理器的数据通路

### 1.1.1 PC模块定义

1. 基本描述

PC 主要功能是完成输出当前指令地址。复位后，PC指向0x0000\_3000，此处为第

一条指令的地址。

1. 模块接口

表1.1 PC接口的模块定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [31:0] newAddress |  | 输入的指令地址 |
| CLK |  | 时钟信号 |
| Reset |  | 复位信号 |
| [31:0] currentAddress  pcW |  | 输出的指令地址  使其一直能够有pc产生（可无） |
| （3）功能定义 | 表1.2 PC接口的功能定义 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | Reset=1时，将pc置为0X0000\_3000 |
| 2 | 输出指令地址 | 时钟信号到来时，将newAddress赋给currentAddress |

### 1.1.2 NPC模块定义

1. 基本描述

NPC主要功能是根据当前指令是否为beq指令，输出下一条指令的地址。该模块调用了MUX模块。

1. 模块接口

表1.3 NPC接口的模块定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Zero | I | ALU计算结果：1表示当前两寄存  器(rs,rt)值相等;0表示不相等。 |
| Branch | I | 是否为beq指令，1为真，0为假。 |
| [31:0]npc | O | 输出下一条指令地址 |
| [15:0]immediate | I | 输入需要扩展的立即数 |
| [25:0]target | I | 输入需要jump到的地址 |
| Jump | I | Jump指令的控制信号 |
| [31:0]pc | I | 输入当前指令地址 |

1. 功能定义

表1.4 NPC接口的功能定义

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | | 功能名称 | 功能描述 |
| 1 | | 输出指令地址 | 根据Zero与Branch的值输出下一条指令的地址 |
| 2 | | 输出指令地址 | 根据Jump和target的值输出下一条指令的地址 |
| 1.1.3 ALU模块定义   1. 基本描述   实现加、减、与、或、小于   1. 模块接口   表1.5 | | 则赋1五种计算。  ALU接口的模块定义 |  |
| 信号名 |  | 方向 | 描述 |
| [2:0]ALUop |  | I | ALU控制信号 |
| [31:0]busA |  | I | rs寄存器的值 |
| [31:0]B |  | I | rt寄存器或者立即数扩展后的值 |
| zero |  | O | A-B=0,则zero=1; 否则zero=0 |
| [31:0]result |  | O | ALU计算输出的结果 |
| （3）功能定义 | 表1.6 | ALU接口的功能定义 |  |
| 序号 |  | 功能名称 | 功能描述 |
| 1 |  | 输出计算结果 | 根据alu控制信号，输出A与B 的计算结果 |
| 2 |  | 输出zero | 若result=0则输出zero=1，否则  输出zero=0 |

### 1.1.4 MUXtip模块定义

1. 基本描述

实现 32 位和 5 位二选一数据选择器。

1. 模块接口

表1.7 MUX接口的模块定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| control | I | MUX控制信号 |
| [31:0]in1 | I | MUX输入 |
| [31:0]in2 | I | MUX输入 |
| [31:0]out | I | MUX输出 |
| （3）功能定义 | 表1.8 MUX接口的功能定义 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出in1或in2的值 | control为1，result= in1 control为0，result= in2 |

### 1.1.5 registerFile模块定义

1. 基本描述

根据输入的两个寄存器地址，输出相应寄存器的值，根据寄存器写信号和寄存器地址，将输入的数据选择写入寄存器。

1. 模块接口

表1.9 registerFile接口的模块定义

信号名

方向

描述

CLK I 时钟信号

[4:0]rt I Rd寄存器地址

[4:0]rs I Rs寄存器地址

[31:0]busA O Rs寄存器值

[31:0]busB O Rt寄存器值

|  |  |  |
| --- | --- | --- |
| WriteReg | I | 写寄存器地址 |
| WriteData | I | 写入寄存器值 |
| RegWr | I | 写入寄存器的信号 |
| （3）功能定义 | 表1.10 REGFILE接口的功能定义 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读寄存器数据 | 读rs、rt，rd寄存器的数据 |
| 2 | 向寄存器写入数据 | 根据写信号向寄存器选择写数据 |

### 1.1.6 InstructionMemory模块定义

1. 基本描述

初始化从code.txt载入指令。根据输入的指令地址，输出当前位置存储的指令。

1. 模块接口

表1.11 Instructionmemory接口的模块定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [31:0]IAddr  [5:0] op  [4:0] rs  [4:0] rt  [4:0] rd  [5:0] func  [15:0] immediate  [25:0] targe | I  I  I  I  I  I  I  I | 指令地址  op域  rs段  rt段  rd段  func段  立即数  Target  段 |
| （3）功能定义 | 表1.12 InstructionMemory接口的功能定义 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 载入指令 | 初始化载入code.txt中的指令 |
| 2  3 | 输出指令  指令的划分 | 根据输入指令地址，输出当前指令及指令的划分  观察指令的正确性 |

### 1.1.7 datamemory模块定义

1. 基本描述

根据输入的地址读出数据内存中的数据，并根据数据写信号，将输入的数据选择写入数据内存中。

1. 模块接口

表1.13 dataMemory接口的模块定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 时钟信号 |
| [31:0]Adr | I | 数据地址 |
| [31:0]DataIn | I | 写入的数据 |
| MenWr | I | 数据内存写信号 |
| [31:0]DataOut | O | 读出的数据 |
| （3）功能定义 | 表1.14 dataMemroy接口的功能定义 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据内存数据 | 根据输入的数据地址，读出数据内存的数据，读出的数据不一定被使用，只有lw指令才会使用 |
| 2 | 向数据内存写入数据 | 在时钟信号到来时，根据写数据信号，将输入的数据选择写入数据内存中 |

### 1.1.8 contrl模块定义

1. 基本描述根据输入的指令高 6 位(op 字段)和指令最低6位（func字段），利用真值表化简，输出 Branch, Jump,RegDst, ALUsrc,funcop，aluop, MemtoReg, RegWr, MemWr,ExtOp，pcW，Rtype控制信号。真值表采用书上166页

图4-22，再加上Jump指令的输入输出。

1. 模块接口

表1.15 contrl和control1接口的模块定义

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 信号名 | | 方向 | | 描述 | |
| [5:0]op  [5:0]func  [2:0] funcop  [2:0] alucop  pcW  Rtype  ExtOp  ALUop | | I  I  I  I  I  I  I | | 指令高6位  指令低6位  由func控制的信号  由op控制的信号  控制一直取指令  R型指令用于调试和选择ALUop由funcop和 alucop决定  符号扩展信号  ALU单元扩展信号 | |
| RegDst | O | | Rd寄存器控制信号 | |
| ALUsrc | O | | ALU数据来源控制信号 | |
| MemtoReg | O | | 数据内存数据写入寄存器控制信号 | |
| RegWr | O | | 寄存器写控制信号 | |
| MemWr | O | | 数据内存写控制信号 | |
| Branch | O | | beq指令控制信号 | |
| Jump | O | | J指令控制信号 | |
| （3）功能定义 | 表1.16 contrl和contrl1接口的功能定义 | |  | |
| 序号 | 功能名称 | | 功能描述 | |
| 1 | 输出各种控制信号 | | 根据输入的OP，利用真值表化简，输出各种控制信号。 | |

### 1.1.9 signext模块定义

1. 基本描述

将输入的16位地址按符号位扩展为32位。

1. 模块接口

表1.17 dm\_4k接口的模块定义

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 |  | 方向 | 描述 |
| [15:0]immediate |  | I | 输入的16位地址 |
| [31:0]imm32 |  | O | 输出的32位地址 |
| ExtOp |  | I | 拓展的信号 |
| （3）功能定义 | 表1.16 | MUX1接口的功能定义 |  |
| 序号 |  | 功能名称 | 功能描述 |
| 1 | 输出扩展的32位地址. | | 根据信号extop将immediate扩展为 32位的imm32. |

## 1.2 各个指令与控制信号之间的关系

### 1.2.1 R-type指令与控制信号

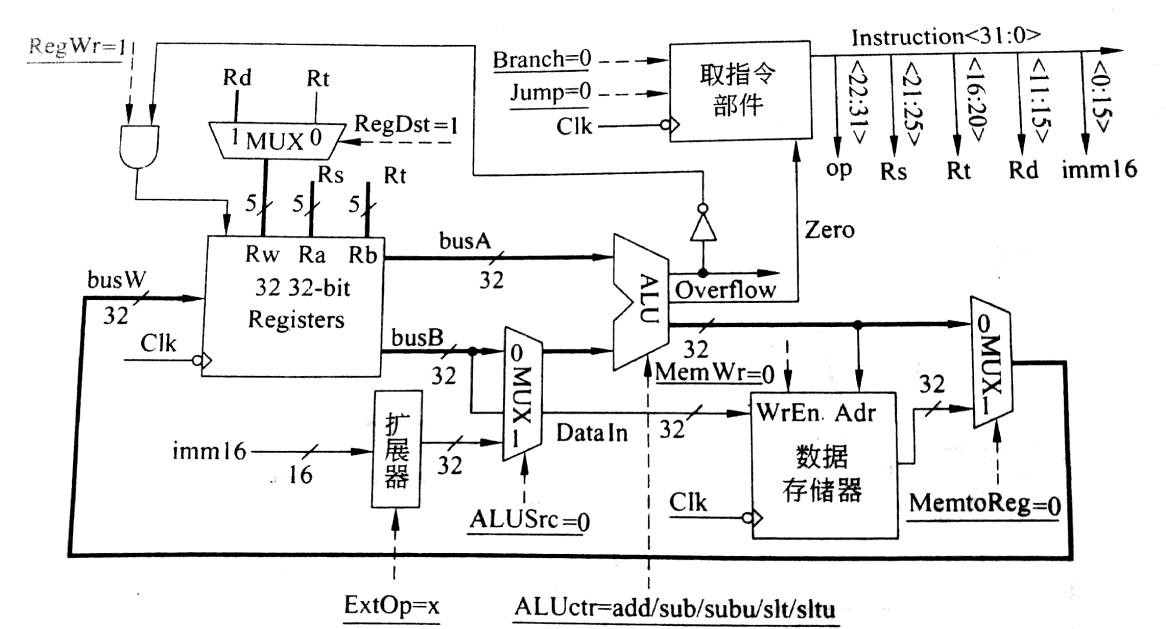


图1.2 R-type指令与控制信号

路径：RegFile(Rs, Rt)->busA, busB->ALU->RegFile(Rd)

### 1.2.2 I-type指令与控制信号

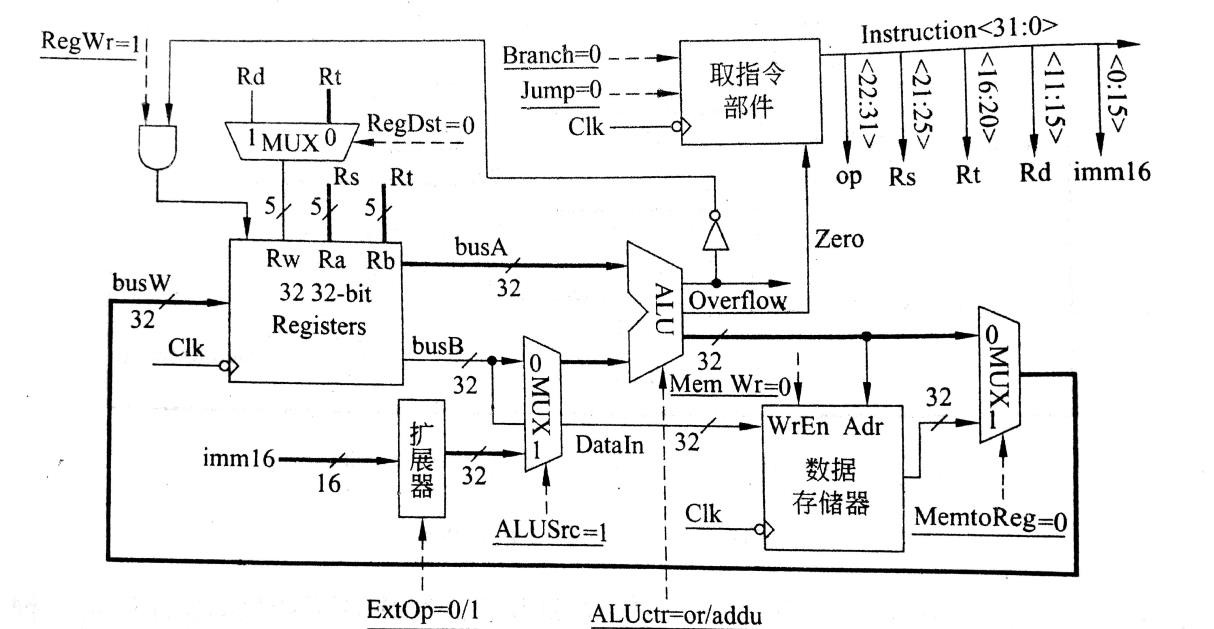


图1.3 I-type指令与控制信号

路径：RegFile(Rs)->busA, 扩展器(imm16)->ALU->RegFile(Rt)

### 1.2.3 Load指令与控制信号

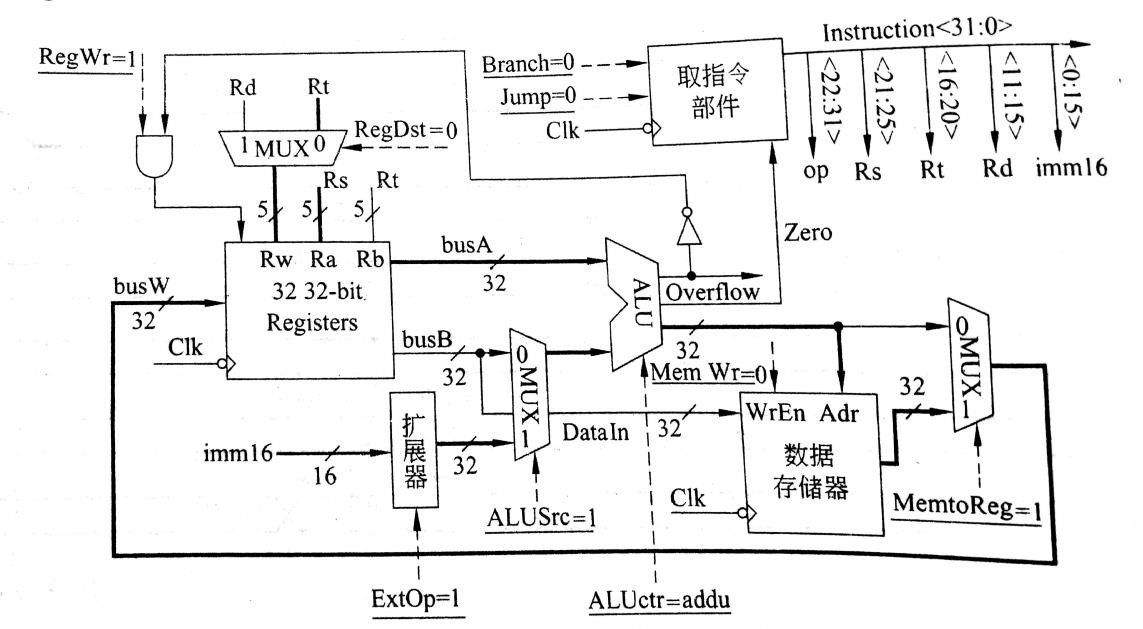


图1.4 Load指令与控制信号

路径：RegFile(Rs)->busA, 扩展器(imm16)->ALU(addu)->数据存储器

->RegFile(Rt)

### 1.2.4 Store指令与控制信号

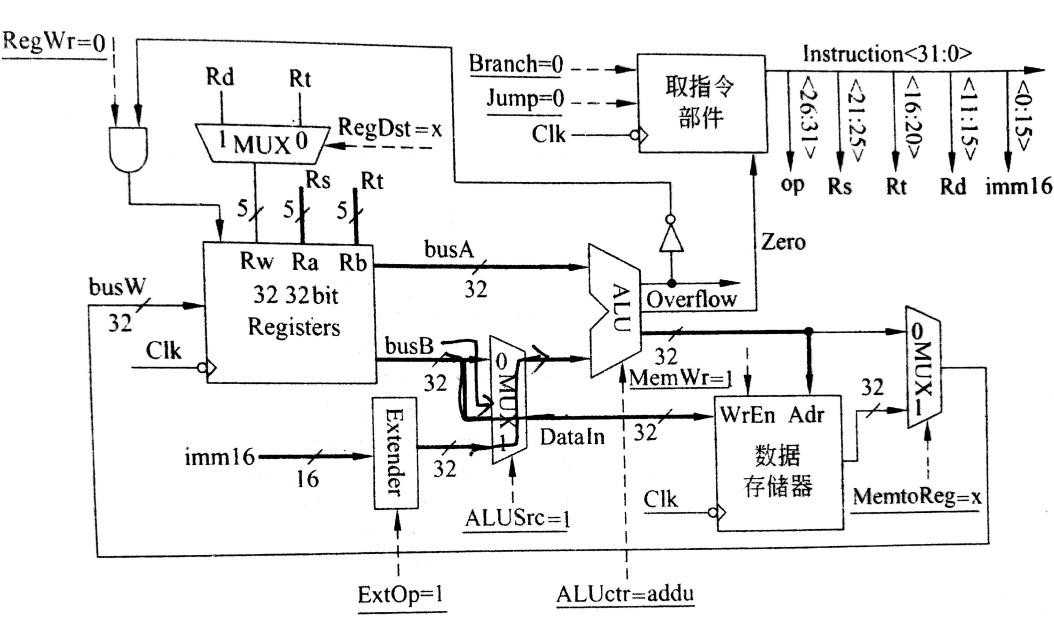


图1.5 Store指令与控制信号

路径：RegFile(Rs, Rt)->busA, 扩展器(imm16), busB->ALU(addu), busB->数

据存储器

### 1.2.5 分支指令与控制信号

路径：RegFile(Rs, Rt)->busA, busB->ALU(subu)->Zero->取指令部件

### 1.2.6 各指令控制信号取值

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | op[5:0] | branch | jump | RegDst | ALUsrc | MemtoReg | RegWr | MemWr | ExtOp | func[5:0] | ALUctr[2:0] |
| add | 000000 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 100000 | 000 |
| sub | 000000 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 100010 | 001 |
| and | 000000 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 100100 | 100 |
| slt | 000000 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 101010 | 001 |
| or | 000000 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 100101 | 011 |
| addi | 001000 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | - | 000 |
| lw | 100011 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | - | 000 |
| sw | 101011 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | - | 000 |
| beq | 000100 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | - | 001 |
| jump | 000010 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | - | 000 |

图1.6 各指令控制信号取值真值表

各个控制信号的逻辑表达式：

funcop[2]=(func==6'b100100)//and

||(func==6'b101010);//slt

funcop[1]=(func==6'b100101);//or

funcop[0]=(func==6'b100101)//or

||(func==6'b100010)//sub

||(func==6'b101010);//slt

Rtype=(op==6'b000000);

pcW=1;

Branch=(op==6'b000100);

Jump=(op==6'b000010);

RegDst=(op==6'b000000);

ExtOp=(op==6'b100011)//lw

||(op==6'b101011);//sw

ALUsrc=(op==6'b100011)||(op==6'b101011)||(op==6'b001000)||(op==6'b001001)||(op==6'b001101);

MemtoReg=(op==6'b100011);

RegWr=(op==6'b000000)//R

||(op==6'b001000)//addi

||(op==6'b100011);//lw

MemWr=(op==6'b101011);//sw

Rtype=op?0:1;

aluop[2]=1'b0;

aluop[1]= 1'b0;

aluop[0]=(op==6'b000100);//beq

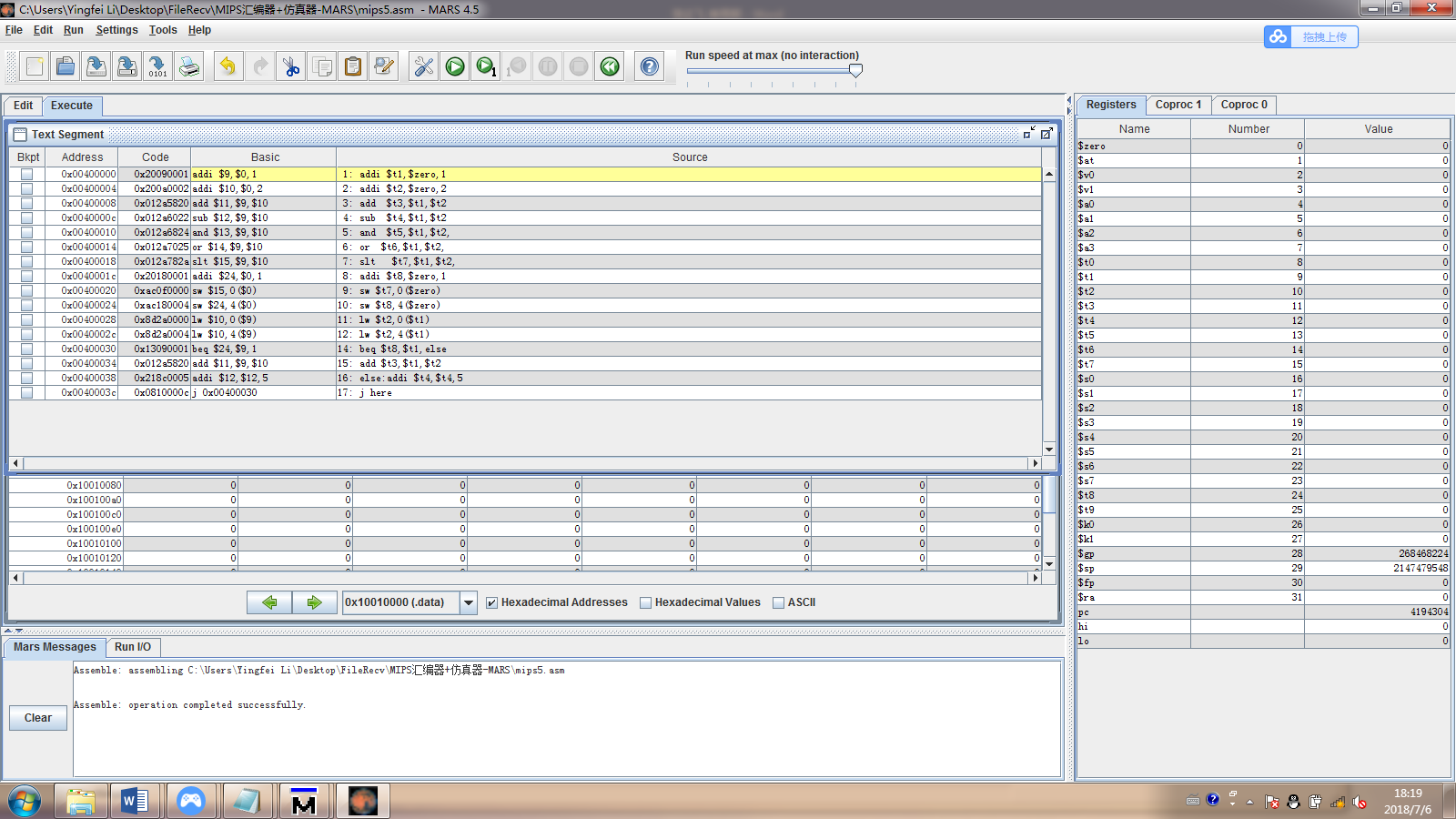
-

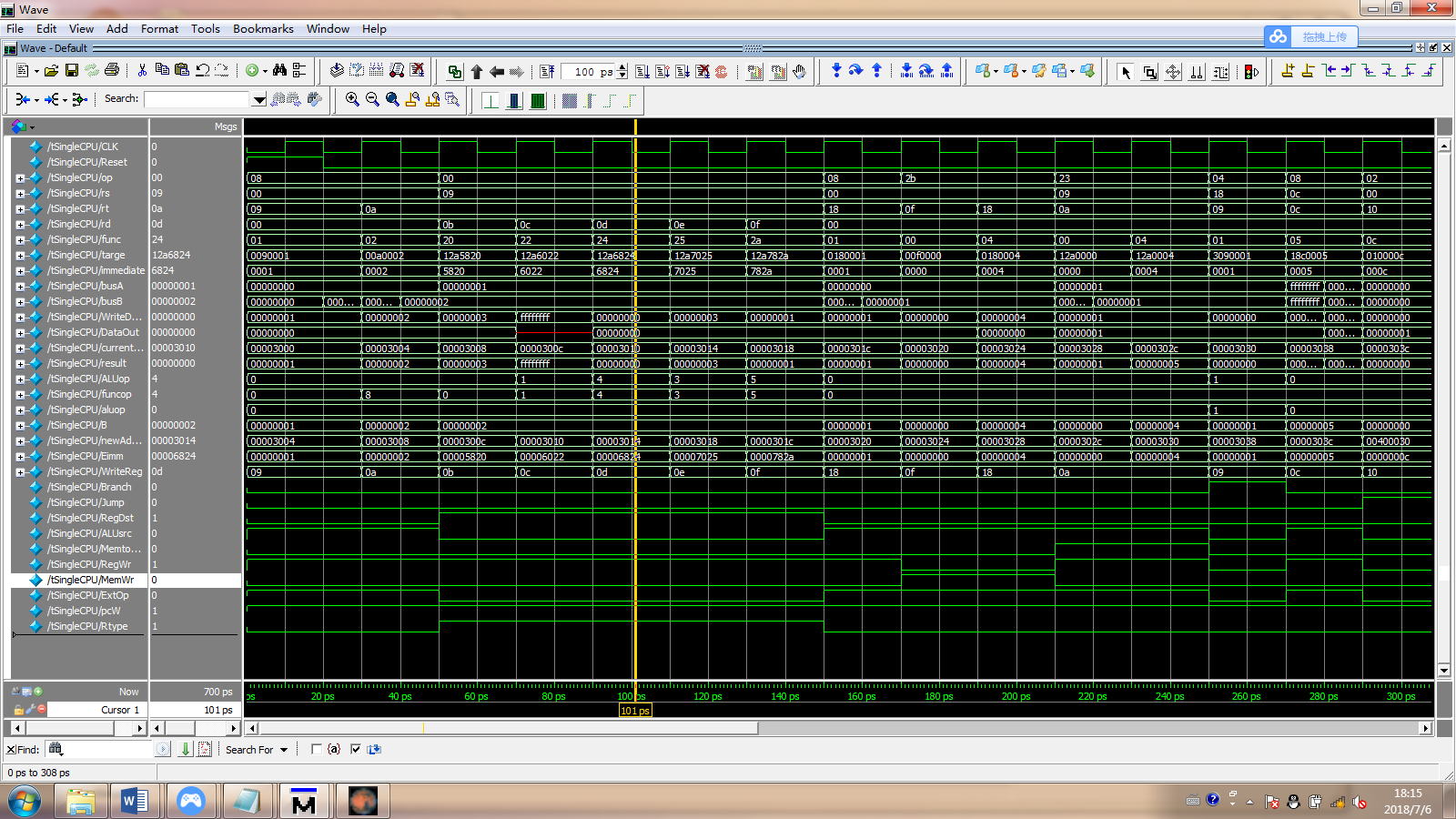
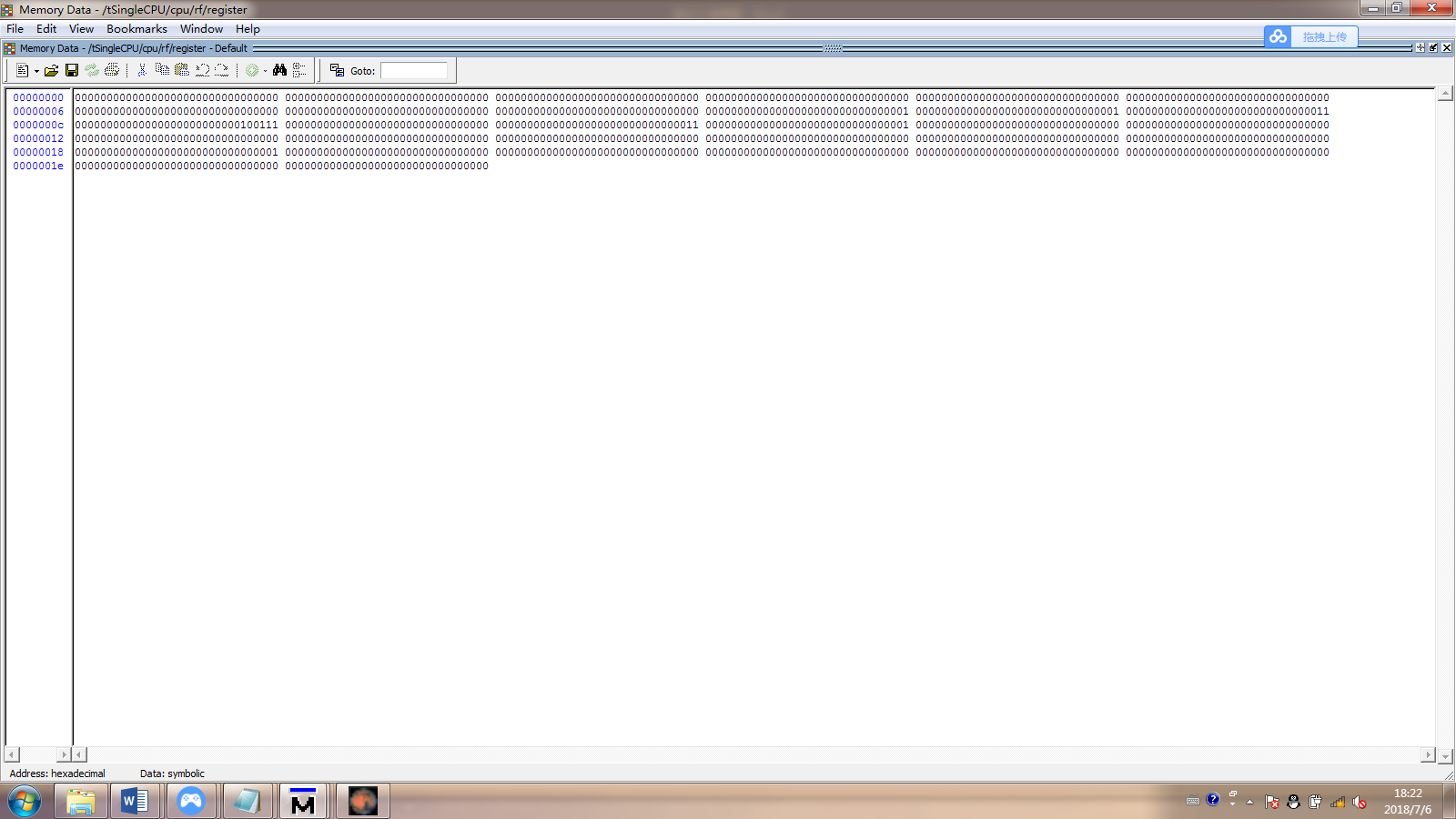
# 第二章 单周期MIPS CPU的具体设计与调试

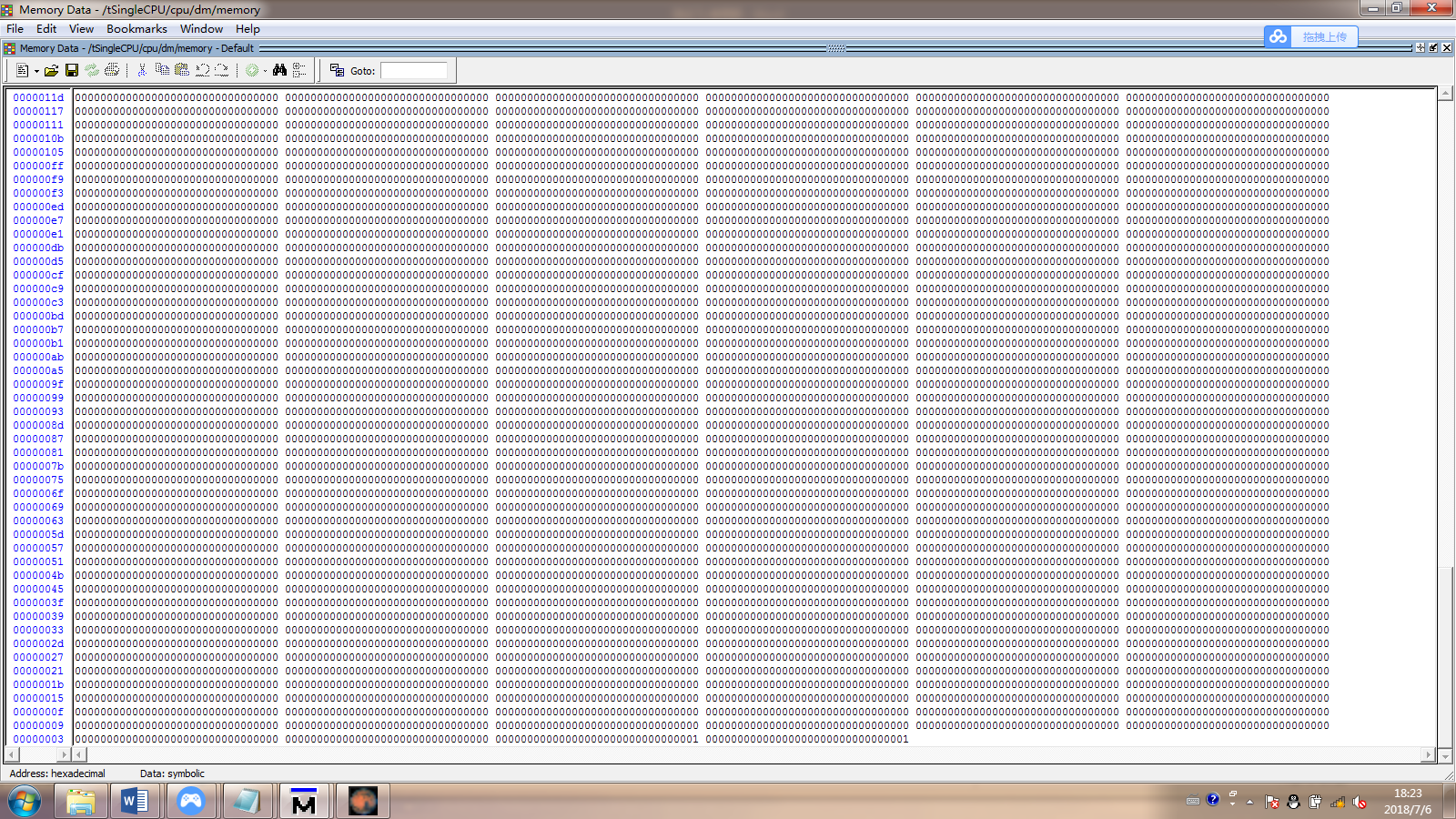
## 2.1 具体Verilog代码实现

#### 2.2 ModelSim模拟及测试

##### 2.2.1 综合测试用代码

1. 汇编代码

运行结果如下

从波形图的各项参数中，可以看到包括数据输入/输出、ALU、指令跳转等部分都在正常工作。并且会进入不断的循环之中.从两个寄存器也可以看出，我们需要的Lw、sw都成功写入与成功读出,其余指令也成功写入目的寄存器。Beq指令成功跳到下一条指令 之后又到J指令。从而形成一个循环。对我们研究有很好地帮助。

# 第三章 总结和展望

进行处理器设计的大致步骤：第一步：分析每条指令的功能，并用RTL(Register Transfer Language) 来表示。 第二步：根据指令的功能给出所需的元件，并考虑如何将他们互连。 第三步：确定每个元件所需控制信号的取值。 第四步：汇总所有指令所涉及到的控制信号，生成一张反映指令与控制信 号之间关系的表。 第五步：根据表得到每个控制信号的逻辑表达式，据此设计控制器电路。9条指令的（我增加了addi）单周期CPU设计和实现，是我第一次使用Verilog语言写项目。并且 ModelSim工具的调试功能不同于面向软件的IDE，偏向数电基础，需要对数字电路有所熟悉。虽然这个软件各种不知名的错误都出现，但只要我们用久了，发现还是不错的。

当我开始写单周期的时候，我是先看了几遍书。然后上网查了一下别人怎么写的。开始也是不知所措。慢慢的就会了。由于我数电基础还比较好吧，写代码的思路倒是很清晰。但是就是调试的时候，感觉一下就蒙了。不知道怎么调，以至于我花了很长时间调试和运行。逐渐的我也发现了自己的问题所在。我对这些指令不怎么熟悉，每个过程都不一定一样。因为每条指令有每条指令的功能。如果不提前对这些指令了如指掌，看信号的时候就有些不知所措。因此花费了我很多的时间。除此之外，我对寄存器不怎么理解，还有就是modelsim如何显示寄存器里的值不会，因为我都是忙着写，群里的消息没看的有好多。验收时，也由于对指令的不熟悉和在后面验收，一度被怀疑代码抄别人。这个我理解。就希望给分公平一些。

9条指令的（单周期CPU设计和实现，也是我们后面36条指令和课设的基础和入门槛。我们只有熟练的掌握了，我们才能够更好、更快的设计出流水线。并且能够很好地理解cpu是如何工作的，对我们了解计算机有很大的帮助。

参 考 文 献

1. MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set[M]. Revision 2.50, Mountain View, CA: MIPS Technologies Inc., July 1, 2005.
2. 袁春风，等. 计算机组成与系统结构[M]. 第 2 版，北京：清华大学出版社，2010.

附 录

1．指令格式及其意义

表1 指令格式及其意义

|  |  |  |
| --- | --- | --- |
| 指令名称 | 汇编举例 | 含义 |
| add | add $s1, $s2, $s3 | $s1 = $s2 + $s3 |
| sub | sub $s1, $s2, $s3 | $s1 = $s2 - $s3 |
| and | and $s1, $s2, $s3 | $s1 = $s2 & $s3 |
| or | or $s1, $s2, $s3 | $s1 = $s2 | $s3 |
| slt | slt $s1, $s2, $s3 | if($s2<$s3) $s1=1; else $s1=0; |
| lw | lw $s1, 0($s2) | $s1 = Mem[$s2+0] |
| sw | sw $s1, 0($s2) | Mem[$s2+0] = $s1 |
| beq | beq $s1, $s2, L | if($s1==$s2) goto L; |
| jump | j L | goto L |