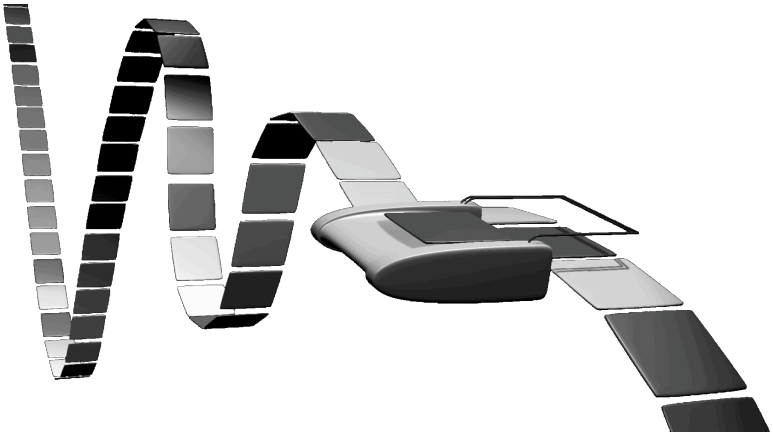
计算机的指令系统

2019年秋

内容概要

- 计算机程序及分类
- 指令系统基本知识
- MIPS指令系统简介
- THINPAD MIPS指令系统
- THINPAD指令模拟器

图灵和图灵机



Turing
Machine
1937

Alan
Turing

计算机程序

- **Computer programs**(also **software programs**, or just **programs**) are instructionsfor a computer. A computer requires programs to function, and a computer program does nothing unless its instructions are executed by a central processor. Computer programs are either executableprograms or the source codefrom which executable programs are derived (e.g., compiled).

- 程序员和计算机硬件之间交互的语言
- 高级语言
- 汇编语言
- 机器语言

程序举例(sum.s)

汇编语言

```
main()
{
  int sum = 0;
  for (int i = 1; i < 11; i++)
     sum += i;
}
```

```
__start:
    addiu   $1, $0, 0x1
    addiu   $2, $0, 0x0
    addiu   $3, $0, 0xA
loop:
    addu    $2, $2, $1
    addiu   $1, $1, 1
    addiu   $3, $3, 0xFFFF
    bgtz    $3, loop
    nop
    jr      $31
    nop
```

程序举例(fib.s)

```
main()
{
  int fibo[10];
  int i;
  fibo[0]=1; fibo[1] =1;
  for ( i=2 ; i<10; i++)
     fibo[i]= fibo[i-1] + fibo[i-
2];
}
```

```
__start:
    addiu   $1, $0, 0x1
    addiu   $2, $0, 0x1
    addiu   $3, $0, 0
    addiu   $4, $0, 10
    addiu   $5, $0, 0x8040
    sll     $5, $5, 16
    sw      $1, 0($5)
    addiu   $5, $5, 4
    sw      $2, 0($5)
loop:
    addu    $3, $2, $1
    addiu   $5, $5, 4
    sw      $3, 0($5)
    addiu   $1, $2, 0
    addiu   $2, $3, 0
    addiu   $4, 0xFFFF
    bgtz    $4, loop
    nop
    jr      $31
    nop
```

Von Newmann结构计算机

- 存储程序计算机
  - 程序由指令构成
  - 程序功能通过指令序列描述
  - 指令序列在存储器中顺序存放
- 顺序执行指令
  - PC指向需要执行的指令
  - 从存储器中读出指令执行(取指)
  - 读取完成之后，PC自增，指向下一条指令

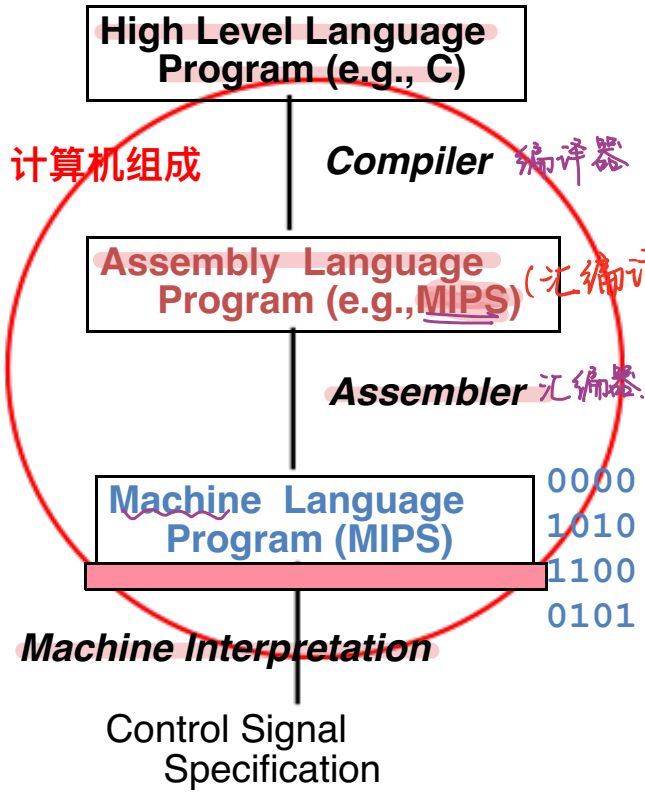
高级语言

- 高级语言又称算法语言，它的实现思路，不是过分地“靠拢”计算机硬件的指令系统，而是着重面向解决实际问题所用的算法，瞄准的是如何使程序设计人员能够方便地写出处理问题和解题过程的程序，力争使程序设计工作的效率更高。
- 用高级语言语言设计出来的程序，需要经过编译程序先翻译成机器语言程序，才能在计算机的硬件系统上予以执行，个别的选用解释执行方案。
- 高级语言的程序通用性强，在不同型号的计算机之间更容易移植。对高级语言进行编译、汇编后得到机器语言在计算机上运行。

汇编语言以及机器语言

- 汇编语言是对计算机机器语言进行符号化处理的结果,再增加一些为方便程序设计而实现的扩展功能。
- 在汇编语言中，可以用英文单词或其缩写替代二进制的指令代码，更容易记忆和理解；还可以选用英文单词来表示程序中的数据(常量、变量和语句标号)，使程序员不必亲自为这些数据分配存储单元，而是留给汇编程序去处理, 达到基本可用标准。
- 若在此基础上，能够在支持程序的不同结构特性（如循环和重复执行结构，子程序所用哑变元替换为真实参数）等方面提供必要的支持，使该汇编语言的实用程度更高。
- 汇编程序要经过汇编器翻译成机器语言后方可运行
- 机器语言是计算机硬件能直接识别和运行的指令的集合,是二进制码组成的指令，用机器语言设计程序基本不可行。
- 程序的最小单元是指令，同时，指令也是计算机硬件执行程序的最小单位。

Levels of Representation



计算机组成

编译器

(汇编语言)

汇编器

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

⇒高级语言

⇓编译器

```
lw   $to,  0($2)
lw   $t1,  4($2)
sw   $t1,  0($2)
sw   $t0,  4($2)
```

⇒汇编语言

⇓汇编器

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

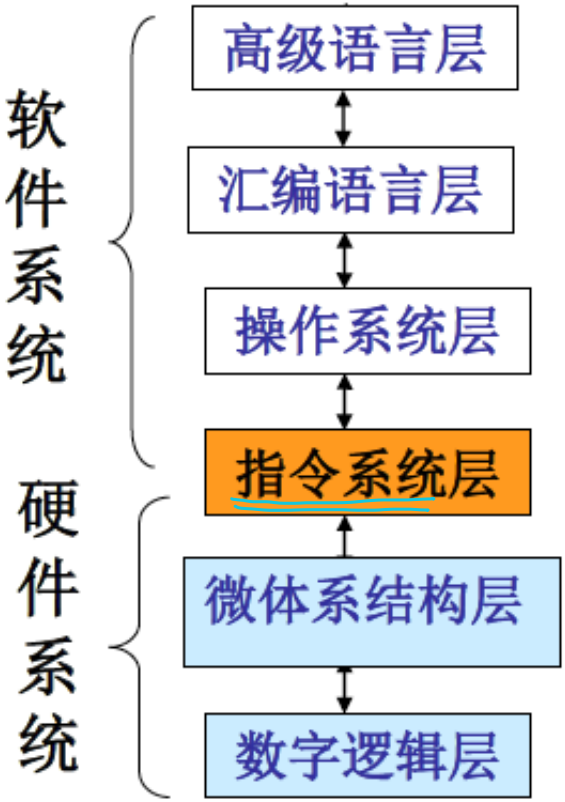
二进制机器语言

指令和指令系统

- 计算机系统由硬件和软件两大部分组成。硬件指由中央处理器、存储器以及外围设备等组成的实际装置。软件是为了使用计算机而编写的各种系统的和用户的程序，程序由一个序列的计算机指令组成。
- 指令是计算机运行的最小的功能单元，是指挥计算机硬件运行的命令，是由多个二进制位组成的位串，是计算机硬件可以直接识别和执行的信息体。指令中应指明指令所完成的操作，并明确操作对象。
- 一台计算机提供的全部指令构成该计算机的指令系统。指令用于程序设计人员告知计算机执行一个最基本运算、处理功能，多条指令可以组成一个程序，完成一项预期的任务

指令系统地位

- 可以从6 个层次分析和看待计算机系统的基本组成。
- 指令系统层处在硬件系统和软件系统之间，是硬、软件之间的接口部分，对两部分都有重要影响。
- 硬件系统用于实现每条指令的功能，解决指令之间的衔接关系；
- 软件由按一定规则组织起来的许多条指令组成，完成一定的数据运算或者事务处理功能。
- 指令系统优劣是一个计算机系统是否成功的关键因素。



计算机系统的层次结构

指令的功能分类

- 数据运算指令
  - 算数运算，逻辑运算 addi
- 数据传输指令 mv
  - 内存/寄存器，寄存器/寄存器
- 控制指令 jump, goto
  - 无条件跳转，条件跳转，子程序的支持（调用和返回）
- 输入输出指令
  - 与输入输出端口的数据传输（输入输出模型如何）
- 其它指令
  - 停机、开/关中断、空操作、特权指令、设置条件码

指令格式

- 指令格式：操作码，操作数地址的二进制分配方案

  - 操作码：指令的操作功能
  - 操作数地址：操作数存放的地址，或者操作数本身
- 指令字：完整的一条指令的二进制表示
- 指令字长：指令字中二进制代码的位数
  - 机器字长：计算机能够直接处理的二进制数据的位数
  - 指令字长（字节倍数）：0.5，1，2，……个机器字长
  - 定长指令字结构 变长指令字结构
  - 定长操作码 扩展操作码

寻址方式

- 如何找寄存器?
- 如何找立即数?
- 如何找内存地址?
  - 直接给出内存地址
  - 通过寄存器进行偏移找出地址
- 如何找输入输出的端口地址?

寻址方式

- 寻址方式（又称编址方式）指的是确定本条指令的操作数地址及下一条要执行的指令地址的方法。
- 不同的计算机系统,使用数目和功能不同的寻址方式，其实现的复杂程度和运行性能各不相同。有的计算机寻址方式较少，而有些计算机采用多种寻址方式。
- 通常需要在指令中为每一个操作数专设一个地址字段，用来表示数据的来源或去向的地址。在指令中给出的操作数（或指令）的地址被称为形式地址，使用形式地址信息并按一定规则计算出来或读操作得到的一个数值才是数据（或指令）的实际地址。在指令的操作数地址字段，可能要指出：
- ①运算器中的累加器的编号或专用寄存器名称（编号）
- ②输入/输出指令中用到的I/O 设备的入出端口地址
- ③内存储器的一个存储单元（或一I/O设备）的地址

评价计算机性能的指标

- 吞吐率
  - 单位时间内完成的任务数量
- 响应时间
  - 完成任务的时间
- 衡量性能的指标
  - MIPS
  - CPI
  - CPU Time
  - CPU Clock
- 综合测试程序

MIPS指令系统

- MIPS
  - Microprocessor without Interlocked Piped Stages
  - 无内部互锁流水级的微处理器
  - RISC芯片 精简
  - 由John L. Hennessy设计
- MIPS
  - Million Instructions per Second
  - 计算机的性能指标之一
- MIPS的历史
  - 见课本或网上各种资料

MIPS处理器

MIPS Processor History

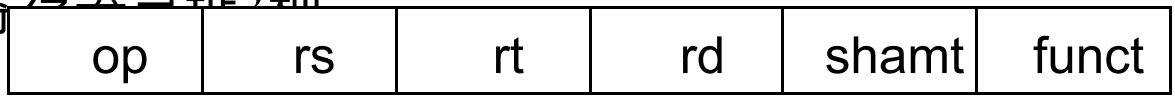
Year	Model - Clock Rate (MHz)	Instruction Set	Cache (I+D)	Transistor Count
1987	R2000-16	MIPS I	External: 4K+4K to 32K+32K	115 thousand
1990	R3000-33		External: 4K+4K to 64K+64K	120 thousand
1991	R4000-100	MIPS III	8K+8K	1.35 million
1993	R4400-150		16K+16K	2.3 million
	R4600-100		16K+16K	1.9 million
1995	Vr4300-133		16K+8K	1.7 million
1996	R5000-200	MIPS IV	32K + 32K	3.9 million
	R10000-200		32K + 32K	6.8 million
1999	R12000-300		32K + 32K	7.2 million
2002	R14000-600		32K + 32K	7.2 million

背 MIPS 的三种指令格式

MIPS指令格式（32位）

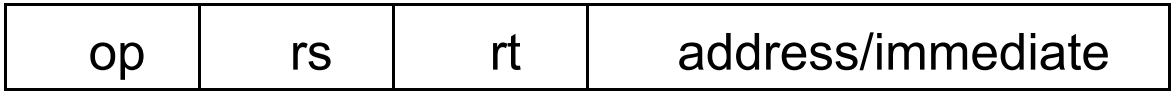
- 所有的指令都是32位字长，有3种指令格式：寄存器型，立即数型，和转移型
- 操作数寻址有基址加16位偏移的访存寻址、立即数寻址和寄存器寻址2种

(R) 寄存器型



add $1, $2, $3　　$3□$1+$2

(I) 立即数型



lw $1, $2, 100　　$2□M[$1+100]　　addi $1, $2, 100　　$2□$1+100

(J) 转移型

j 8000　转移到 PC[31..28] | 8000 X 4

MIPS指令系统

- MIPS 64
  - 面向64位处理器的指令系统
- MIPS 16e
  - 16位字长的MIPS指令集
  - 主要用于嵌入式系统

ThinPAD MIPS指令系统

- 采用与MIPS32兼容的指令格式
  - 32位固定字长
  - 操作码位置及长度固定
  - 寻址方式简单
- 供设计有21+4+4条指令
  - 可根据需要进行扩展
- 作为本课程教学实验的指令系统

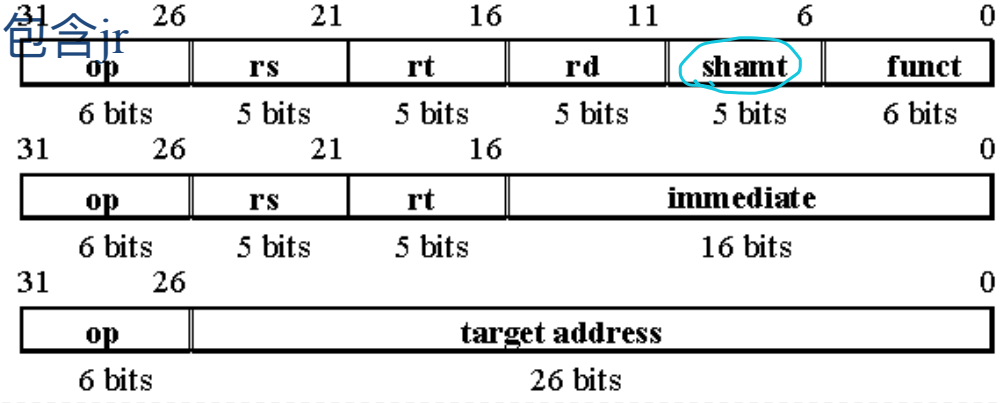
ThinPAD MIPS指令系统概况

所需要实现的基础指令21条，后续只统计这21条的情况

监控程序基础版本	ADDIU, ADDU, AND, ANDI, BEQ, BGTZ, BNE, J, JAL, JR, LB, LUI, LW, OR, ORI, SB, SLL, SRL, SW, XOR, XORI
监控程序支持中断版本	上一行所有指令+ERET, MFC0, MTC0, SYSCALL
监控程序支持TLB版本	上一行所有指令+TLBP, TLBR, TLBWI, TLBWR

MIPS32的指令格式

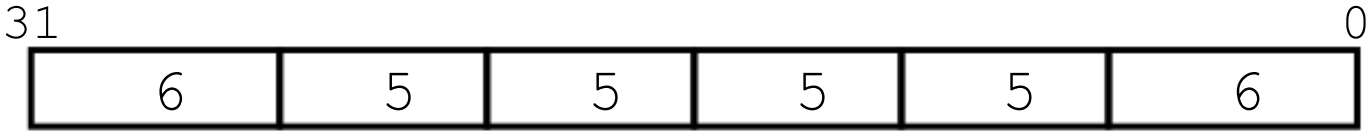
- R-格式：寄存器指令，指定指令中的3个寄存器
- I-格式：指令中包含立即数，lw/sw（offset是立即数），已经beq/bne
  - 不包含移位指令 是R型
- J-格式：j以及jal
  - 不包含jr
  - R
  - I
  - J



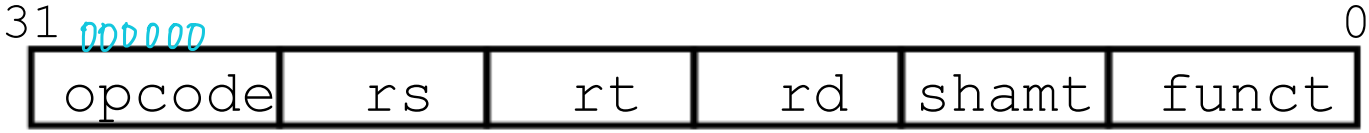
ThinPAD MIPS指令格式（R型指令）

- ADDU,AND,JR,OR,SLL,SLR,XOR共7条

- R型指令每个指令字分成6个域：



- 每个域有自己的名字：

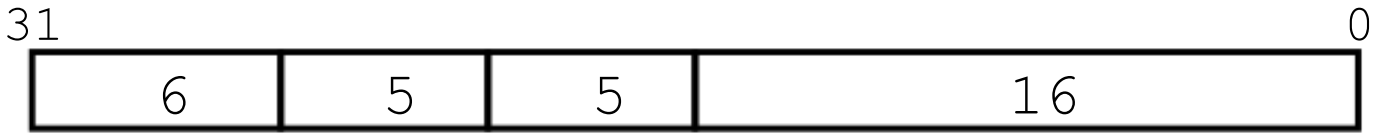


- 每个域自成一个无符号整数
  - 5位域的取值范围0-31，
  - 6位域的取值范围0-63

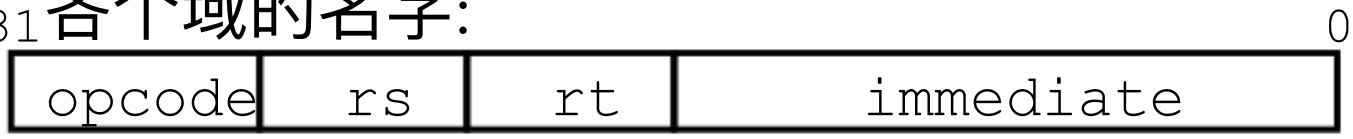
ThinPAD MIPS指令格式（I型指令）

- ADDIU,ANDI,BEQ,BGTZ,BNE,LB,LUI,LW,ORI,SB,SW,XORI,共12条,

- 下面是I类型指令的格式：



- 各个域的名字：

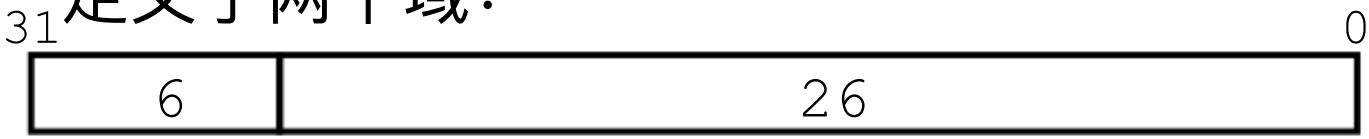


- 三个域与R类型指令是一样的
  - opcode 仍然放置在原来的位置

ThinPAD MIPS指令格式（J型指令）

- J,JAL,共2条

- 定义了两个域：



- 前面为opcode，后面为目标地址：

– opcode 位置和长度和R类型是一样的
– 剩下的位被统一使用，以获得更大的目标地址表示空间

ThinPAD MIPS指令系统(1)

指令格式	汇编语句	功能说明
001001 rs rt imm	ADDIU rt, rs, immediate	rt □ rs + immediate
000000 rs rt rd 00000 100001	ADDU rd, rs, rt	rd □ rs + rt
000000 rs rt rd 00000 100100	AND rd, rs, rt	rd □ rs AND rt
001100 rs rt imm	ANDI rt, rs, immediate	rt □ rs AND immediate
000100 rs rt offset	BEQ rs, rt, offset	if rs == rt, PC ← PC + sign_extend(offset || 02)
000111 rs 00000 offset	BGTZ rs, offset	if rs > 0, PC ← PC + sign_extend(offset || 02)
000101 rs rt offset	BNE rs, rt, offset	if rs != rt, PC ← PC + sign_extend(offset || 02)
000010 instr_index	J target	jump
000011 instr_index	JAL target	jump and link
000000 rs 00 0000 0000 hint 001000	JR rs	jump register

ThinPAD MIPS指令系统(2)

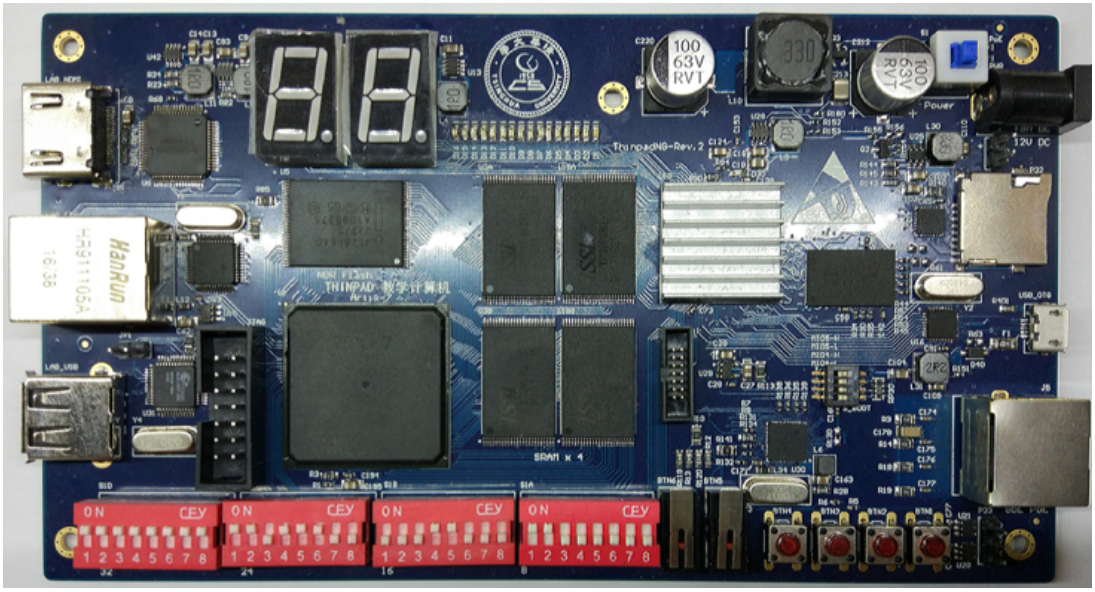
指令格式	汇编语句	功能说明
100000 base rt offset	LB rt, offset(base)	Load Byte
001111 0000 rt imm	LUI rt, immediate	rt ← immediate || 016
100011 base rt offset	LW rt, offset(base)	rt ← memory[base+offset]
000000 rs rt rd 00000 100101	OR rd, rs, rt	rd ← rs or rt
001101rs rt imm	ORI rt, rs, immediate	rt ← rs or zero_extend(immediate)
101000 base rt offset	SB rt, offset(base)	Store byte
000000 00000 rt rd sa 000000	SLL rd, rt, sa	rd ← rt << sa
000000 0000 0 rt rd sa 000010	SRL rd, rt, sa	rd ← rt >> sa
101011 base rt offset	SW rt, offset(base)	Store word
000000 rs rt rd 00000 100110	XOR rd, rs, rt	rd ← rs XOR rt
001110 rs rt imm	XORI rt, rs, immediate	rt ← rs XOR immediate

ThinPAD MIPS指令系统(3)

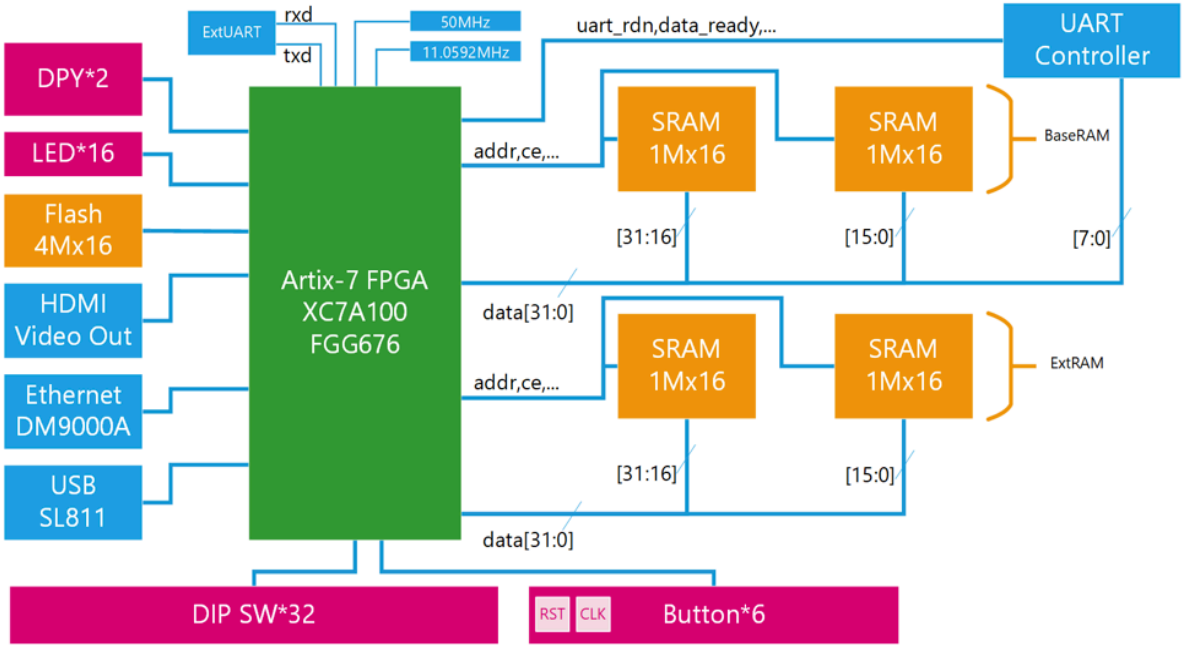
指令格式	汇编语句	功能说明
010000 1 000 0000 0000 0000 0000 011000	ERET	Exception Return
010000 00000 rt rd 00000000 sel	MFC0 rt, rd MFC0 rt, rd, sel	GPR[rt] ← CPR[0,rd,sel]
010000 00100 rt rd 0000 000 sel	MTC0 rt, rd MTC0 rt, rd, sel	CPR[0, rd, sel] ← GPR[rt]
000000 code 001100	SYSCALL	System call
010000 1 000 0000 0000 0000 0000 001000	TLBP	TLB probe
010000 1 000 0000 0000 0000 0000 000001	TLBR	TLB read
010000 1 000 0000 0000 0000 0000 000010	TLBWI	TLB write or invalid
010000 1 000 0000 0000	TLBWR	TLB write random

ThinPAD的硬件组成

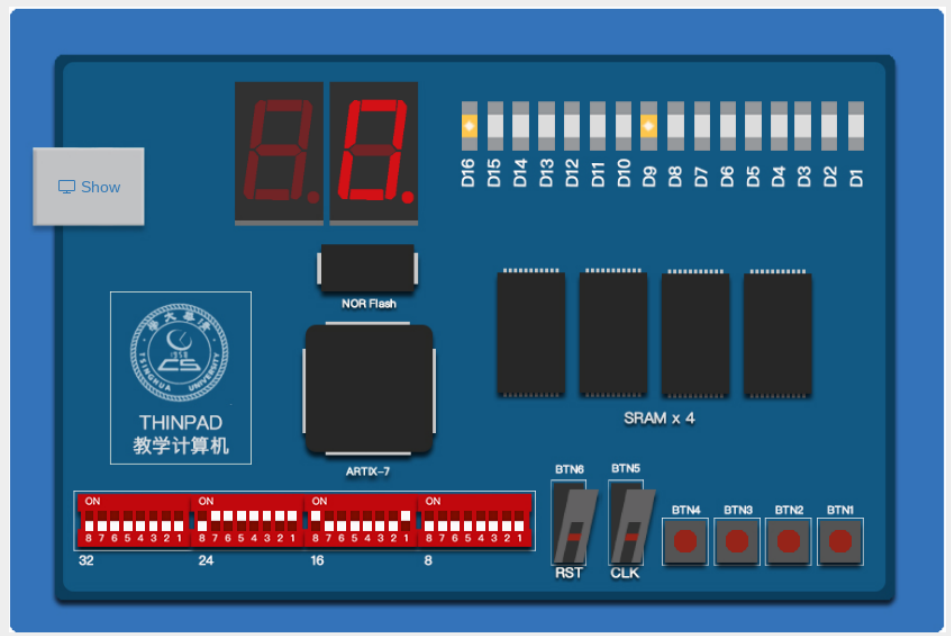
- 主要有一块FPGA，两块SRAM，串口以及其它的一些外围接口电路组成



硬件平台的内部电路构成



网络控制界面


Show
D16 D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1
NOR Flash
THINPAD
教学计算机
ARTIX-7
SRAM x 4
ON
8 7 6 5 4 3 2 1
32 24 16 8
BTN6 BTN5
RST CLK
BTN4 BTN3 BTN2 BTN1

监控程序的地址空间划分

虚地址区间	说明
0x80000000-0x800FFFFF	Kernel代码空间
0x80100000-0x803FFFFF	用户代码空间
0x80400000-0x807EFFFF	用户数据空间
0x807F0000-0x807FFFFF	Kernel数据空间
0xBFD003F8-0xBFD003FD	串口数据及状态

串口寄存器位定义

地址	位	说明
0xBFD003F8（数据寄存器）	[7:0]	串口数据，读、写该地址分别表示串口接收、发送一个字节
0xBFD003FC（状态寄存器）	[0]	状态位，只读，为1时表示串口空闲，可发送数据
0xBFD003FC（状态寄存器）	[1]	状态位，只读，为1时表示串口收到数据

模拟器

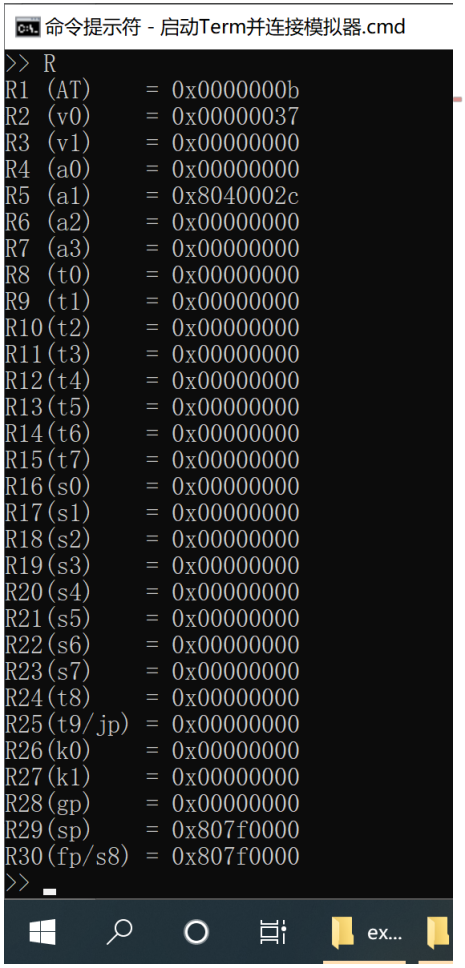
- 模拟器可以模拟ThinPAD硬件的执行，通过QEMU来模拟MIPS处理器
- 监控程序可以直接运行在模拟器上
- 模拟器可以运行在Windows，Linux和Mac三个平台
- Linux是模拟器天然支持的环境
- Windows需要小调整
- Mac需要自行编译汇编器

```
zhang@zhang-mac ~> brew install --build-from-source mipsel-sde-elf-binutils
==> Downloading https://ftp.gnu.org/gnu/binutils/binutils-2.32.tar.gz
Already downloaded: /Users/zhang/Library/Caches/Homebrew/downloads/0d7828e43d0eb3ab5c42e032514f1a8a05fcb4d8e6da855df3cce7b2c4d3d395--binutils-2.32.tar.gz
==> ./configure --target=mipsel-sde-elf --disable-multilib --disable-nls --disable-werror --prefix=/usr/local/Cellar/mipsel
==> make
==> make install
🍺  /usr/local/Cellar/mipsel-sde-elf-binutils/2.32: 186 files, 18.7MB, built in 4 minutes 17 seconds
```

```
zhang@zhang-mac ~/D/s/term> setenv GCCPREFIX mipsel-sde-elf-
zhang@zhang-mac ~/D/s/term> ./term.py -t 166.111.227.235:38823
connecting to 166.111.227.235:38823...connected
MONITOR for MIPS32 - initialized.
>> a
>>addr: 0x80100000
one instruction per line, empty line to end.
[0x80100000] jr $ra
[0x80100004] nop
[0x80100008]
>> u
>>addr: 0x80100000
>>num: 8
0x80100000: jr  ra
0x80100004: nop
>>
```

执行求和程序

```
__start:
    addiu   $1, $0, 0x1
    addiu   $2, $0, 0x0
    addiu   $3, $0, 0xA
loop:
    addu    $2, $2, $1
    addiu   $1, $1, 1
    addiu   $3, $3, 0xFFFF
    bgtz    $3, loop
    nop
    jr      $31
    nop
```



执行斐波那契数列程序

```
MONITOR for MIPS32 - initialized.
>> A
>>addr: 0x80100000
one instruction per line, empty line to end.
[0x80100000] addiu $2,$0,1
[0x80100004] addu $3,$2,$2
[0x80100008] jr $31
[0x8010000c] nop
[0x80100010]
>> U
>>addr: 0x80100000
>>num: 16
0x80100000:  li    v0,1
0x80100004:  addu    v1,v0,v0
0x80100008:  jr    ra
0x8010000c:  nop
>> G
>>addr: 0x80100000

elapsed time: 0.000s
>> R
R1 (AT)     = 0x00000000
R2 (v0)     = 0x00000001
R3 (v1)     = 0x00000002
R4 (a0)     = 0x00000000
R5 (a1)     = 0x00000000
R6 (a2)     = 0x00000000
...
```

命令提示符 - 启动Term并连接模拟器.cmd

```
0x80400030: 0x00000000
0x80400034: 0x00000000
0x80400038: 0x00000000
0x8040003c: 0x00000000
0x80400040: 0x00000000
0x80400044: 0x00000000
0x80400048: 0x00000000
0x8040004c: 0x00000000
>> U
>>addr: 0x80100000
>>num: 80
0x80100000: li  at,1
0x80100004: li  v0,1
0x80100008: li  v1,0
0x8010000c: li  a0,10
0x80100010: li  a1,-32704
0x80100014: sll a1,a1,0x10
0x80100018: sw  at,0(a1)
0x8010001c: addiu       a1,a1,4
0x80100020: sw  v0,0(a1)
0x80100024: addu        v1,v0,at
0x80100028: addiu       a1,a1,4
0x8010002c: sw  v1,0(a1)
0x80100030: addiu       at,v0,0
0x80100034: addiu       v0,v1,0
0x80100038: addiu       a0,a0,-1
0x8010003c: bgtz        a0,0x80100024
0x80100040: nop
0x80100044: jr  ra
0x80100048: nop
0x8010004c: nop
>>
```

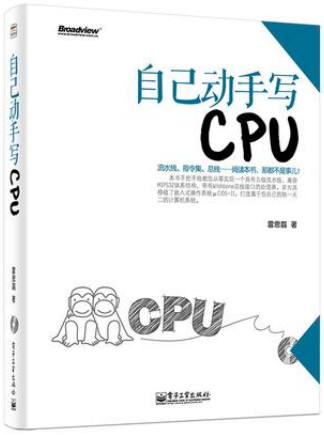
命令提示符 - 启动Term并连接模拟器.cmd

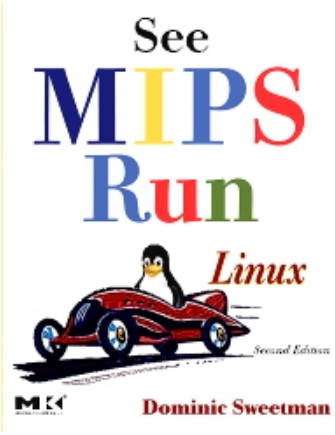
```
[0x80100048] nop
[0x8010004c]
>> g
>>addr: 0x80100000

elapsed time: 0.001s
>> D
>>addr: 0x80400000
>>num: 80
0x80400000: 0x00000001
0x80400004: 0x00000001
0x80400008: 0x00000002
0x8040000c: 0x00000003
0x80400010: 0x00000005
0x80400014: 0x00000008
0x80400018: 0x0000000d
0x8040001c: 0x00000015
0x80400020: 0x00000022
0x80400024: 0x00000037
0x80400028: 0x00000059
0x8040002c: 0x00000090
0x80400030: 0x00000000
0x80400034: 0x00000000
0x80400038: 0x00000000
0x8040003c: 0x00000000
0x80400040: 0x00000000
0x80400044: 0x00000000
0x80400048: 0x00000000
0x8040004c: 0x00000000
>>
```

实验参考书

- 《自己动手写CPU》 详细给出了如何使用verilog写一个CPU的例子，可以作为实验的基础
- 《See MIPS Run》详细介绍了MIPS处理器的指令集体系结构，如果需要实现中断以及TLB，可以参考





实验提示

- 监控程序在kern下，命令行程序在term下
- 大部分预先需要知道的信息在README.md中，务必先阅读这（两）个文件。在kern/README.md有另外一个。
- 两个README.md文件内容有重复，可以都阅读一下
- 按照试验指导材料运行监控程序，编写汇编代码，熟悉监控程序

监控程序阅读提示

- .set noreorder，不允许汇编器移动指令。一般指令顺序移动只会发生在编译器里。MIPS的汇编器可能会把指令放到延迟槽中。这里禁止汇编器做这个事情。
- .set noat就是不使用at寄存器。写指令的时候避免使用会用到at的那些伪指令。
- .p2align 2，按照2^2对齐，即4字节对齐。 .p2align 12 按照2^12对齐。
- .global current，导出符号current，模块外部可见，外部可引用。监控程序是单一模块，程序中可以不使用global。

MIPS中的伪指令（合成指令）

- move dst, src □ addi dst, src, 0
- li dst, imm □ addi(u) dst, $0, imm
- li $t1, 40 □ addi $t1, $zero, 40
- li $t1, -4000000 □ lui $at, 0xffc2　ori $t1, $at, 0xf700

Makefile

- .PHONY
- 忽略工作目录下的接在.PHONY后面的文件，强制去执行build命令
- 有开关：ON_FPGA，EN_INT，EN_TLB，分别对应不同版本的监控程序（ON_FPGA，处理串口的方式不同）
- make ON_FPGA=y EN_INT=y EN_TLB=y

小结

- 计算机程序语言
  - 高级语言
  - 汇编语言
  - 机器语言
- 指令系统
  - 硬件/软件接口
  - 指令功能/指令格式
- ThinPAD MIPS指令系统

阅读和思考

- 阅读
  - 教材：第2章 指令系统
- 思考
  - 指令系统的作用和地位?
  - 为实现ThinPAD指令系统，ALU应该具备哪些功能
  - 数据在计算机内部如何表示？应表示哪几类数据?
- 练习
  - 分析ThinPAD MIPS指令系统的特点

谢谢