自选实验之计算机系统

1. 实验目标

本实验的目标是在 DE10-Standard 开发板的 FPGA 上实现一个简单的计算机系统，能够运行简单的指令，并处理一定量的输入输出。在所有功能开发完毕后，希望能够完成基本的 terminal 功能，即键盘输入命令，并在显示器上输出结果。以下内容只是设计参考，具体实现时可以根据自己的兴趣选做一部分或者进行裁剪和修改。

2. CPU 部分

CPU 部分建议参考 32 位 MIPS 指令集。该指令集是经典的 RISC 指令集，实现起来较为简单。MIPS 32 的所有指令长度都是 32bit，分为三种基本类型：

- R-Type：含 3 个寄存器操作数
- I-Type：2 个寄存器操作数，及一个 16bit 立即数操作数
- J-Type：跳转，26bit 立即数操作数

格式如图 1-1所示：

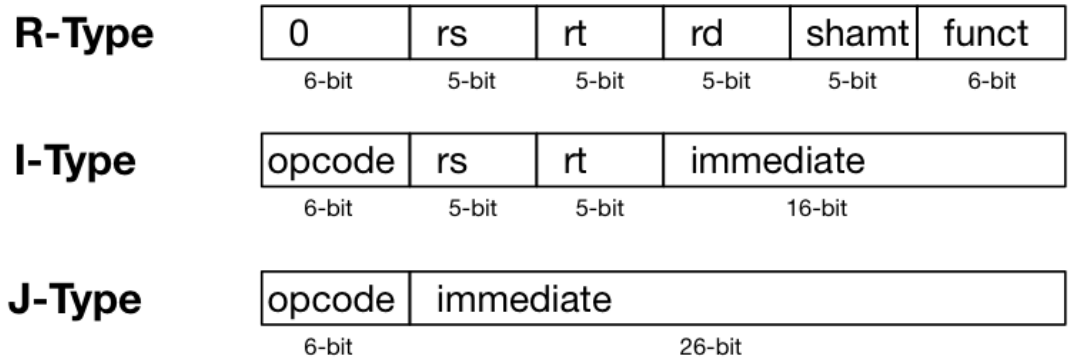

图 1-1: MIPS 指令格式

其中 opcode 必定为指令前 6bit，源、目的寄存器也都在特定位置出现，所以指令解码非常方便。

MIPS 具体包含包含以下几类指令：

- **ALU 指令**：可以是 2 寄存器操作数，结果送入目的寄存器，或一个寄存器一个立即数，结果送入目的寄存器。包括有无符号的加减法，移位，及逻

辑操作等等。

- **Load Store 指令**：一般是寄存器 + 立即数偏移量，读取/写入内存。注意：MIPS 中所有数据都需要先 load 进入寄存器才能进行操作，不能像 x86 一样直接对内存数据进行算术处理。
- **分支与跳转指令**：条件分支包括 BEQ，BNE 等等，根据寄存器内容选择是否跳转（没有 flag 寄存器）。无条件跳转是 26 位立即数。JAL 用于函数调用，自动将返回地址放入 r31 寄存器。注意：MIPS 中有跳转延迟槽的概念，跳转指令下一条指令不管是否跳转都会执行。所以，建议在所有跳转指令后加上一条 NOP(全 0，add r0,r0,r0)。

MIPS 32 共 32 个 32bit 的寄存器（5 bit 寄存器地址），其中寄存器 r0 中的内容总是 0，寄存器 r31 中存储函数调用的返回地址（在 JAL 指令中实现，如果选择实现该指令的话）。

建议实现表 1-1中的指令，如果有需要，可以按 MIPS 指令集自行扩充

表 1-1: MIPS 部分常用指令功能

指令	功能	编码
add rd,rs,rt	rs+rt->rd, 无溢出才装入	0x0/6,rs/5,rt/5,rd/5,0/5,0x20/6
addu rd,rs,rt	rs+rt->rd, 不判溢出	0x0/6,rs/5,rt/5,rd/5,0/5,0x21/6
addi rt,rs,imm	rs+imm->rt, 无溢出装入	0x8/6,rs/5,rt/5,imm/16
addiu rt,rs,imm	rs+imm->rt，不判溢出，符号扩展	0x9/6,rs/5,rt/5,imm/16
sub rd,rs,rt	rs-rt->rd, 无溢出装入	0x0/6,rs/5,rt/5,rd/5,0/5,0x22/6
subu rd,rs,rt	rs-rt->rd, 不判溢出	0x0/6,rs/5,rt/5,rd/5,0/5,0x23/6
nor rd,rs,rt	rs nor rt ->rd	0x0/6,rs/5,rt/5,rd/5,0/5,0x27/6
xori rt,rs,imm	rs 与 imm 无符号扩展 xor -> rd	0xe/6,rs/5,rt/5,imm/16
slt rd,rs,rt	rs 比 rt 小，rd 置一，有符号数	0x0/6,rs/5,rt/5,rd/5,0/5,0x2a/6
sltu rd,rs,rt	rs 比 rt 小，rd 置一，无符号数	0x0/6,rs/5,rt/5,rd/5,0/5,0x2b/6
slti rt,rs,imm	rs 与符号扩展 imm 比较，置 rt	0xa/6,rs/5,rt/5,imm/16
sltiu rt,rs,imm	rs 与符号扩展 imm 比较，置 rt	0xb/6,rs/5,rt/5,imm/16
blez rs,imm	rs<0 跳转，imm*4	0x6/6,rs/5,0/5,imm/16
j target	无条件跳转 target*4	0x2/6,imm/26
lw rt,offset(base)	Base 加上 offset 地址装入 rt	0x23/6, base/5,rt/5,offset/16
sw rt,offset(base)	rt 写入 Base 加上 offset 地址	0x2b/6, base/5,rt/5,offset/16

可以考虑使用最简单的单周期实现，即每条指令在 1 个周期内完成读指

令，解码，计算和写回操作。时钟频率可以调整到系统能够在一个 CPU 时钟周期内完成所有操作。

总体框图如图 1-2所示：

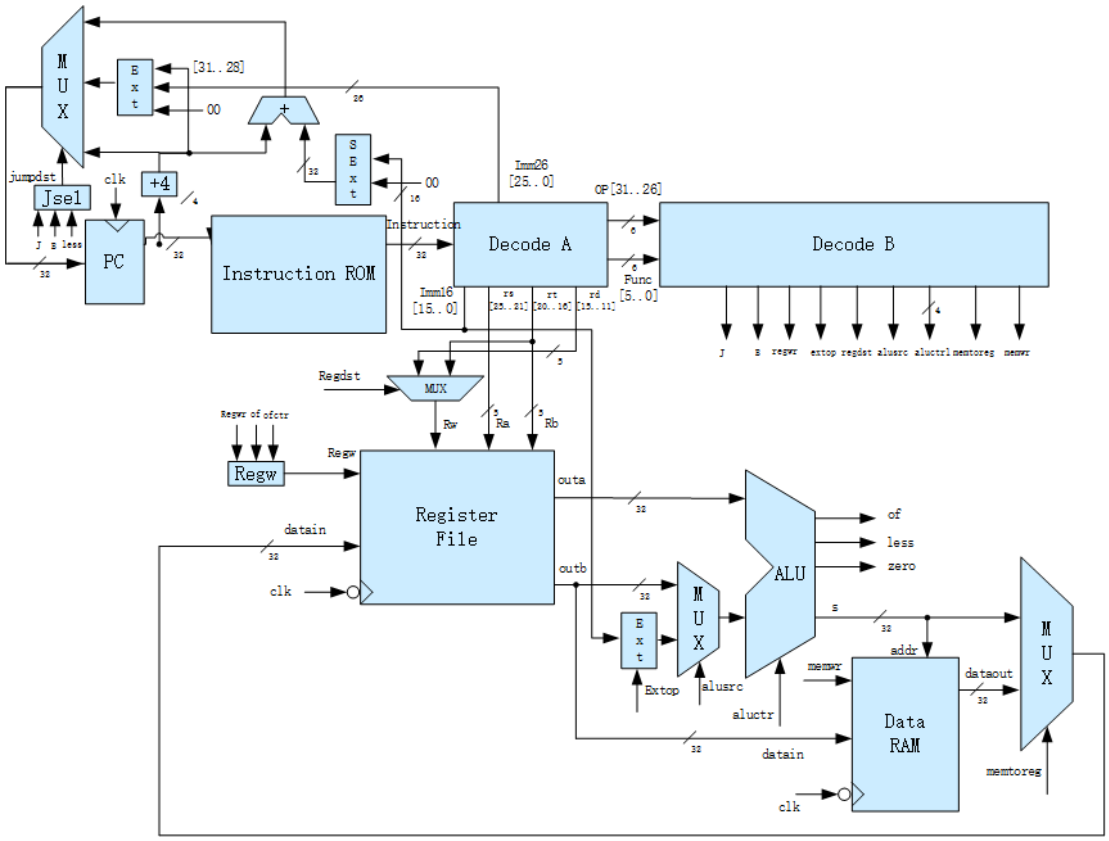

图 1-2: MIPS 单周期 CPU 框图

主程序可以分成以下几个大的子模块:

- **ALU**: 参考之前的 ALU 实现，如表 1-2所示。
- **寄存器组**：采用之前的寄存器实现，包括读和写，地址译码等。对于单周期完成的指令，请注意寄存器读取和写入的时序，建议可以读取采用不受时钟控制的直通方式，写入在上升沿。
- **指令 ROM**：采用 RAM 实现，利用 mif 文件初始化。编译方式需要自己设计。可以使用 MARS MIPS 模拟器编程 http://courses.missouristate.edu/KenVollmar/mars/ ，编译后通过 mem dump 将 Code 段（及数据段）转换为二进制，然后自编 Python 小程序将 MARS 的二进制格式转为 mif 格式固化在 RAM 中。具体指令地址起始范围自行确定。**注意：该编译器只支持最多 1024 条指令，如果软件规模大，请采用其他方式编译。**

表 1-2: ALU 操作功能表

ALUCtr<3:0>	操作类型	Subctr 加减	ALUout<2:0>	输出选择
0 0 0 0	addu	0	0 0 0	输出加法器结果但不判溢出
0 0 0 1	add	0	0 0 0	输出加法器结果
0 0 1 0	and	x	0 1 0	输出按位与结果
0 0 1 1	or	x	0 1 1	输出按位或结果
0 1 0 0	not	x	1 0 0	输出按位取反结果
0 1 0 1	nor	x	1 0 1	输出按位或非结果
0 1 1 0	xor	x	1 1 0	输出按位异或结果
0 1 1 1	neg	x	1 1 1	输出取负结果
1 0 0 0	subu	1	0 0 0	输出加法器结果但不判溢出
1 0 0 1	sub	1	0 0 0	输出加法器结果
1 0 1 0	sltu	1	0 0 1	输出小于置位结果
1 0 1 1	slt	1	0 0 1	输出小于置位结果

- **数据 RAM**：使用 RAM 实现，读取可以使用下降沿，写入可以在上升沿。数据 RAM 可以用 mif 文件初始化。具体 RAM 地址分配自行确定，需要考虑给外设：如键盘和显示器分配特定内存空间。
- **PC 部分:** 如上图所示，采用一个 PC 寄存器实现，在时钟上升沿更新。PC 更新源包括 3 个，Branch，jump，和 PC+4，利用控制信号选择。PC 部分单独一个模块。由于在本示例中仅实现了 blez，所以只用了一位 branch 信号，同 Less，Zero 联合判断是否要 branch。具体扩展其他分支方式请自行考虑。
- **指令解码部分:** 分成两块。一块是将指令 rom 输出信号直接分成 op,func,rs,rt,rd 和 imm 等。另一块是通过 op/func 信号输出各类控制信号。
  具体的控制信号如表 1-3所示，控制信号的含义与教材一致

  建议在 CPU 调试过程中先分块实现，分别仿真模拟，再整合到一起去。

表 1-3: MIPS 指令译码表

指令	OP	Func	Jump	Branch	Regwr	Extop	Regdst	ALUsrc	ALUctrl	Memtoreg	Memwr
add	00h	20h	0	0	1	0	1	0	0001	0	0
addu	00h	21h	0	0	1	0	1	0	0000	0	0
addi	08h	-	0	0	1	1	0	1	0010	0	0
addiu	09h	-	0	0	1	0	0	1	0010	0	0
sub	00h	22h	0	0	1	0	1	0	1001	0	0
subu	00h	23h	0	0	1	0	1	0	1000	0	0
nor	00h	27h	0	0	1	0	1	0	0101	0	0
xori	0eh	-	0	0	1	0	0	1	0110	0	0
slt	00h	2ah	0	0	1	0	1	0	1011	0	0
sltu	00h	2bh	0	0	1	0	1	0	1010	0	0
slti	0ah	-	0	0	1	1	0	1	1011	0	0
sltiu	0bh	-	0	0	1	0	0	1	1010	0	0
blez	06h	-	0	1	0	0	0	0	0000	0	0
j	02h	-	1	0	0	0	0	0	0000	0	0
lw	23h	-	0	0	1	1	0	1	0000	1	0
sw	2bh	-	0	0	0	1	0	1	0000	0	1

3. 输入输出

输入输出分为以下功能，每个模块可以是独立的，与 CPU 并行运行。

- **显示器**：实现标准字符显示界面。显示器部分独立于 CPU，自动进行扫描控制和字符显示。实现方式是，规定字符界面宽度和高度，如 70 列 *30 行字符，在 CPU 内存中划出 70*30 Byte 区域作为显存，存储要显示字符的 ASCII 码。CPU 可以对该区域写 ASCII 码，确定要显示什么字符。显示器模块只是每次扫描时读取这块显存，显示对应的字符。具体方式是在扫描时扫到对应位置，读取显存的 ASCII 码，根据 ASCII 码计算每个点对应的字模点阵地址，读取字模的 RGB 值。需要在 RAM 中预先存储好一套 ASCII 字库，内存组织方式请自行确定。附件为一简单的 9*16 ASCII 字库。字符覆盖范围是 ascii[6:0]，忽略 ascii 最高位。每个字符是 9*16 点阵，每一行以 12bit 表示（对应 bit 为 1 为亮，为 0 是暗，低 bit 位在显示器左边），每个字符 16 行。共 128 个字符。

  可以参考图 1-3在显示区域中自行划出一块区域用直接设置的方式来显示 CPU 当前 PC、指令和寄存器状态等等。
- **键盘**：复用之前的键盘程序，在内存中开辟一块键盘缓存区（可以仅单个 byte），将键盘输入转换为 ASCII 码存入，并将键盘输入指示置 1。
- **七段数码管和 LED**：可根据自己的调试需要输出。
- **拨动开关和按钮**：自行设计。为方便调试，可以设置 reset 按钮，重置 CPU

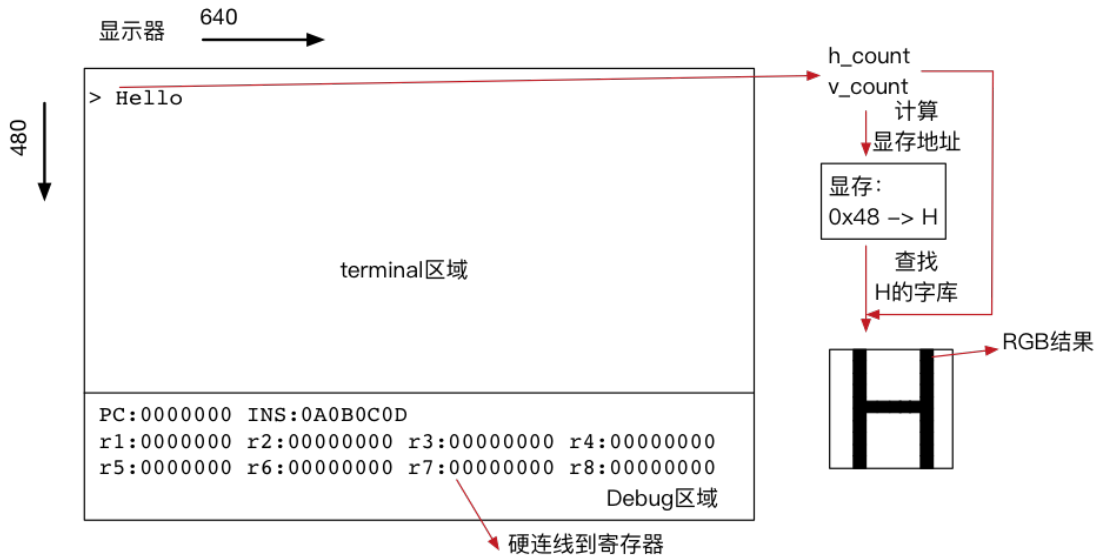

图 1-3: 显示器界面布置示意图

的 PC 和寄存器。可以设置 CPU 时钟来源，在单步调试时，用按钮作为 CPU 时钟，每按一下运行一条指令，正常运行时使用内部时钟。

4. 软件部分

软件部分用 MIPS 汇编编写，可以下载 MIPS 模拟器http://courses.missouristate.edu/KenVollmar/mars/。**注意！此汇编器最多支持 1024 条指令，如果程序规模较大，建议自行完成汇编功能。**

软件编写完成后，可以用模拟器提供的 mem dump 功能将代码段机器码存储下来，并用 python 转换成 mif 格式。**注意：本次使用的开发板没有和 FPGA 相连的串口，因此不能通过串口写入 MIPS 代码。如何高效地进行软件改写需要仔细考虑。**

软件可选实现的功能：

1. 主循环：不停轮询接口状态。如果只处理键盘输入，只需轮询键盘缓冲器和标志位。在有键盘输入时，跳转到键盘输入处理。
2. 键盘处理：将键盘输入存入命令缓冲区，如果有回车键，跳转到命令分析部分。
3. 命令分析：匹配字符串，根据命令执行对应的子程序，并将执行结果输出到屏幕上。执行完成后跳回主循环。扩展需求，需要处理显示换行，滚动等功能。

- 打入 hello，显示 Hello World!
- 打入 LED 1 on，打开 LED 1，打入 LED 1 off，关闭 LED 1.
- 打入 time，显示时间
- 打入简单表达式，如 (9+6*6)-3，输出结果
- 打入未知命令，输出 Unknown Command。