第一章　课程实验一参考资料

对交大软件学院二年级-下的同学（2009 级），为深入理解计算机组成与设计的基本原理，培养从硬件和软件两方面全局考虑计算机针对问题求解所进行的系统设计思想，掌握针对一定规模系统的设计实践能力，配合课堂教学内容开设以下 3 个实验：

1．单周期 CPU 设计实验；

2．五段流水 CPU 设计实验；

3. 外部 I/O 及接口扩展实验。

1.1　实验一单周期 CPU 设计实验参考资料

1.1.1 实验 DE2 板载资源：

（1）Altera-DE2 板载资源的详细介绍，参见 DE2_UserManual.pdf。

（2）DE2 实验板不同于上学期的 JDEE-10K 教学实验箱，其上的 8 个 7 段 LED 数码管的连接电路采用每个段位直连到 FPGA 的 I/O 管脚的静态驱动方法，不是采用动态扫描的方式，如图 1-1 所示。详见 DE2_ UserManual.pdf 文档。

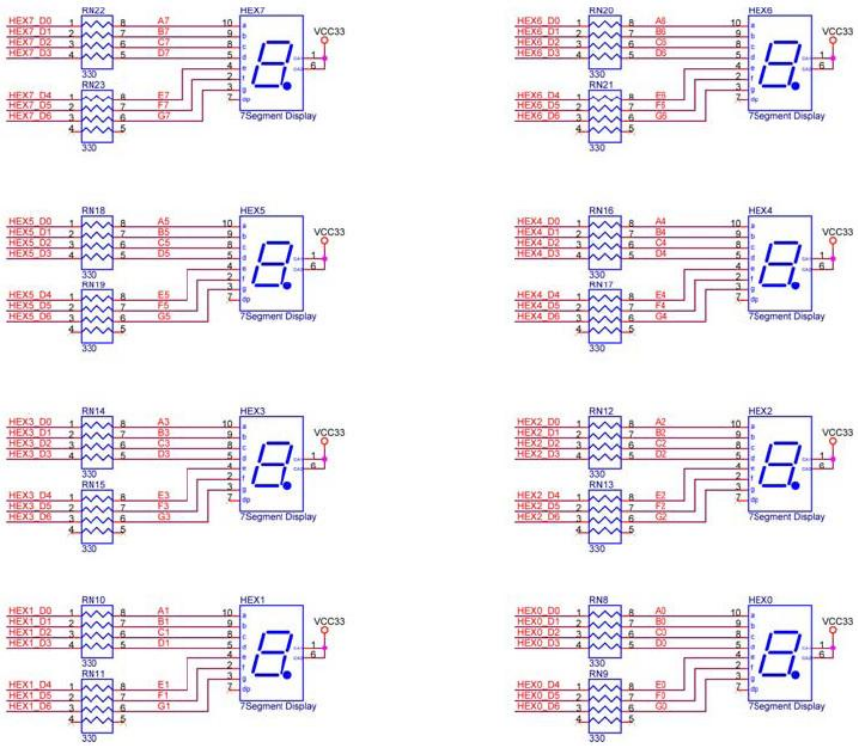


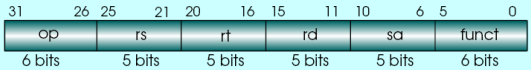
图 1-1. Altera-DE2 实验板上的 7 段 LED 数码管

（3）按键和拨动开关的连接电路详见 DE2_ UserManual.pdf 文档。

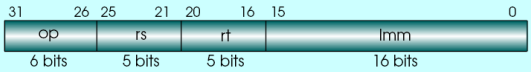
1.1.2. 部分原理介绍及辅助参考

MIPS指令格式

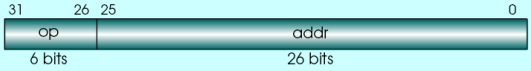
R 類型指令　(rs, rt, rd: 寄存器號, sa: 移位位數)



I 類型指令　(rs, rt: 寄存器號, imm: 立即數)



J 類型指令　(addr: 跳轉目標地址)



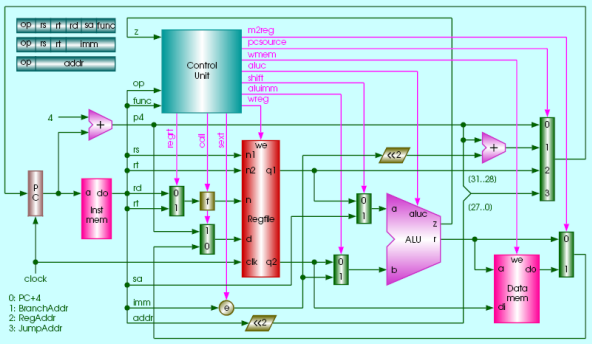
20 MIPS 指令

```
add  rd, rs, rt  ; rd <-- rs + rt
sub  rd, rs, rt  ; rd <-- rs - rt
and  rd, rs, rt  ; rd <-- rs & rt
or   rd, rs, rt  ; rd <-- rs | rt
xor  rd, rs, rt  ; rd <-- rs ^ rt
sll  rd, rt, sa  ; rd <-- rt << sa
srl  rd, rt, sa  ; rd <-- rt >> sa (logical)
sra  rd, st, sa  ; rd <-- rt >> sa (arithmetic)
jr   rs          ; PC <-- rs
addi rt, rs, imm ; rt <-- rs + (sign)imm
andi rt, rs, imm ; rt <-- rs & (zero)imm
ori  rt, rs, imm ; rt <-- rs | (zero)imm
xori rt, rs, imm ; rt <-- rs ^ (zero)imm
lw   rt, imm(rs) ; rt <-- memory[rs + (sign)imm]
sw   rt, imm(rs) ; memory[rs + (sign)imm] <-- rt
beq  rs, rt, imm ; if (rs == rt) PC <-- PC + 4 + (sign)imm << 2
bne  rs, rt, imm ; if (rs != rt) PC <-- PC + 4 + (sign)imm << 2
lui  rt, imm     ; rt <-- imm << 16
j    addr        ; PC <-- (PC+4)[31..28],addr<<2
jal  addr        ; $31 <-- PC+4; PC <-- (PC+4)[31..28],addr << 2
```

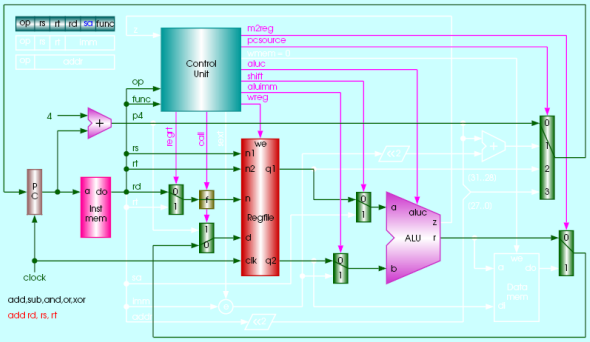

20 MIPS 指令

```
op     rs    rt    rd    sa    func   ; R format instructions
000000 rs    rt    rd    00000 100000 ; add  rd, rs, rt
000000 rs    rt    rd    00000 100010 ; sub  rd, rs, rt
000000 rs    rt    rd    00000 100100 ; and  rd, rs, rt
000000 rs    rt    rd    00000 100101 ; or   rd, rs, rt
000000 rs    rt    rd    00000 100110 ; xor  rd, rs, rt
000000 00000 rt    rd    sa    000000 ; sll  rd, rt, sa
000000 00000 rt    rd    sa    000010 ; srl  rd, rt, sa
000000 00000 rt    rd    sa    000011 ; sra  rd, st, sa
000000 rs    00000 00000 00000 001000 ; jr   rs
op     rs    rt    imm                ; I format instructions
001000 rs    rt    imm                ; addi rt, rs, imm
001100 rs    rt    imm                ; andi rt, rs, imm
001101 rs    rt    imm                ; ori  rt, rs, imm
001110 rs    rt    imm                ; xori rt, rs, imm
100011 rs    rt    imm                ; lw   rt, imm(rs)
101011 rs    rt    imm                ; sw   rt, imm(rs)
000100 rs    rt    imm                ; beq  rs, rt, imm
000101 rs    rt    imm                ; bne  rs, rt, imm
001111 00000 rt    imm                ; lui  rt, imm
op     addr                          ; J format instructions
000010 addr                          ; j    addr
000011 addr                          ; jal  addr
```

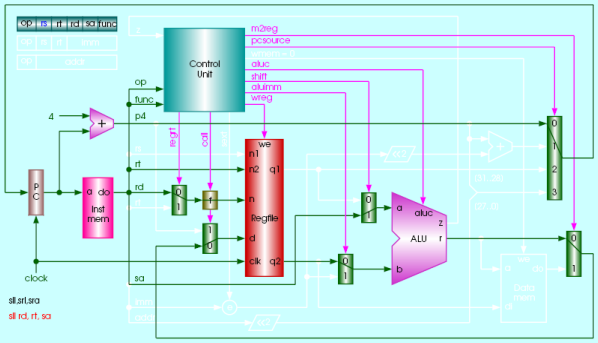

單周期 CPU + 指令存儲器 + 數據存儲器



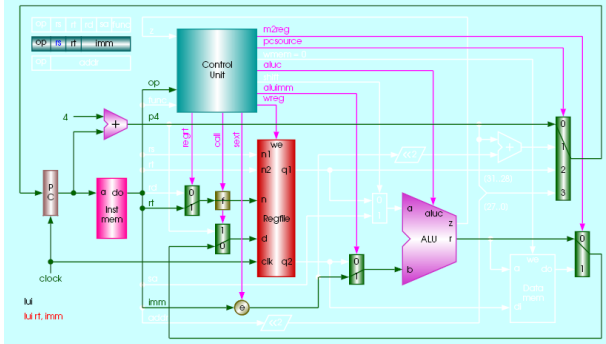
add, sub, and, or, xor 指令



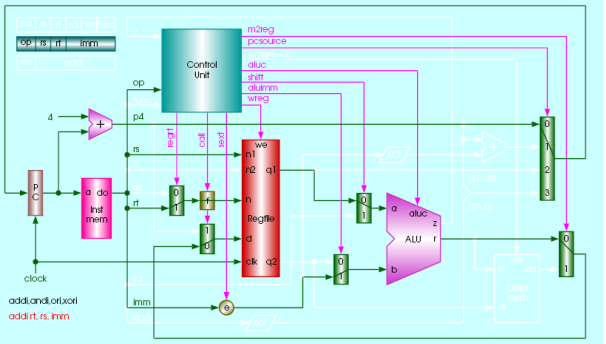
sll, srl, sra 指令



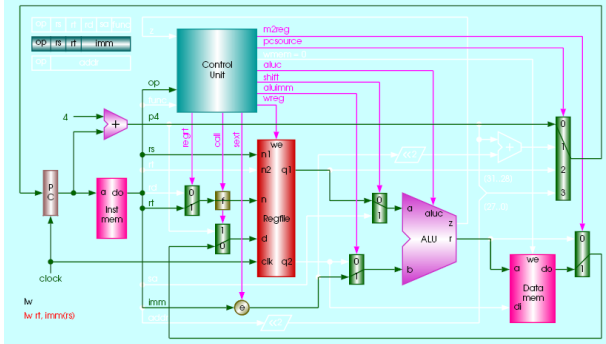
lui 指令



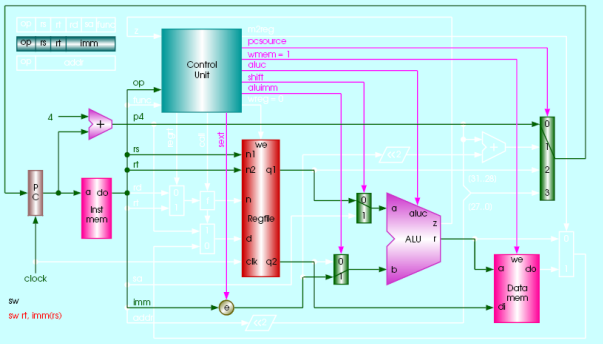
addi, andi, ori, xori 指令



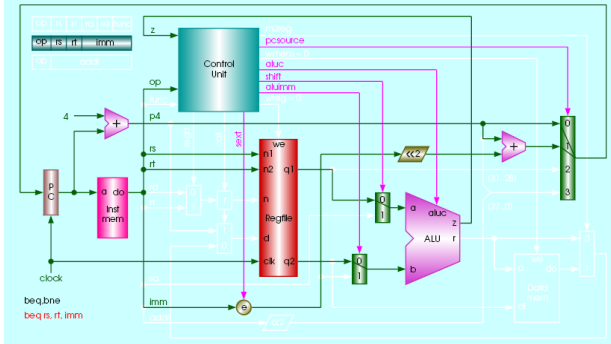
lw 指令



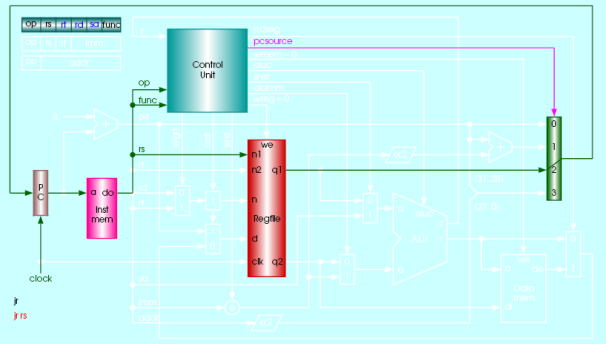
sw 指令



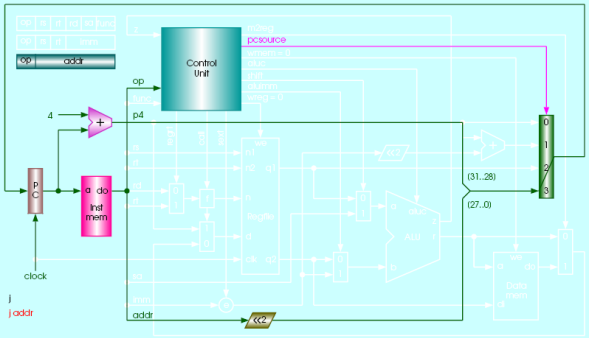
beq, bne 指令



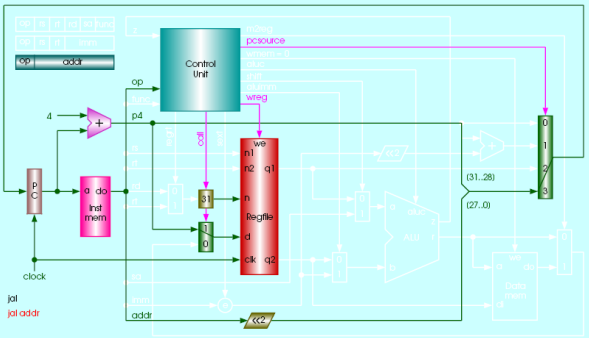
jr 指令



j 指令



jal 指令



控制部件

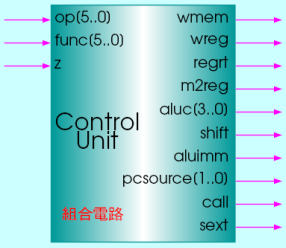
MIPS CPU 指令格式

R 類型指令

I 類型指令

J 類型指令

sc_cu



控制信號的種類

1. 寫使能 (Write Enable)
   - wreg
   - wmem
2. ALU操作控制
   - aluc(3..0)
3. 多路器的選擇信號
   - m2reg
   - pcsource(1..0)
   - shift
   - aluimm
   - regrt
   - call
   - sext

控制信號的意義

- wreg
  - 1: 寫寄存器堆
  - 0: 不寫
- wmem
  - 1: 寫存儲器
  - 0: 不寫
- m2reg
  - 1: 選擇從存儲器中讀出的數據
  - 0: 選擇ALU的運算結果
- shift
  - 1: 選擇移位位數(sa)
  - 0: 選擇寄存器堆的數據

控制信號的意義

- aluimm
  - 1: 選擇擴展後的立即數
  - 0: 選擇寄存器堆的數據
- regrt
  - 1: 選擇rt
  - 0: 選擇rd
- call
  - 1: 選擇PC+4
  - 0: 選擇ALU或者存儲器數據
- sext
  - 1: 符號擴展
  - 0: 零擴展

控制信號的意義

- pcsource(1..0)
  - 0 0: 選擇PC+4
  - 0 1: 選擇轉移地址
  - 1 0: 選擇寄存器地址
  - 1 1: 選擇跳轉地址
- aluc(3..0)
  - x 0 0 0 : ADD (指令: add, addi, lw, sw)
  - x 1 0 0 : SUB (指令: sub, beq, bne)
  - x 0 0 1 : AND (指令: and, andi)
  - x 1 0 1 : OR (指令: or, ori)
  - x 0 1 0 : XOR (指令: xor, xori)
  - x 1 1 0 : LUI (指令: lui)
  - 0 0 1 1 : SLL (指令: sll)
  - 0 1 1 1 : SRL (指令: srl)
  - 1 1 1 1 : SRA (指令: sra)

指令譯碼

生成表示指令的中間變量, 變量名與指令名相同[a]

$$Rtype = \overline{op(5)} \cdot \overline{op(4)} \cdot \overline{op(3)} \cdot \overline{op(2)} \cdot \overline{op(1)} \cdot \overline{op(0)}$$

$$add = Rtype \cdot func(5) \cdot \overline{func(4)} \cdot \overline{func(3)} \cdot \overline{func(2)} \cdot \overline{func(1)} \cdot \overline{func(0)}$$

$$sub = Rtype \cdot func(5) \cdot \overline{func(4)} \cdot \overline{func(3)} \cdot \overline{func(2)} \cdot func(1) \cdot \overline{func(0)}$$

$$\cdots = \cdots$$

$$addi = \overline{op(5)} \cdot \overline{op(4)} \cdot op(3) \cdot \overline{op(2)} \cdot \overline{op(1)} \cdot \overline{op(0)}$$

$$\cdots = \cdots$$

$$j = \overline{op(5)} \cdot \overline{op(4)} \cdot \overline{op(3)} \cdot \overline{op(2)} \cdot op(1) \cdot \overline{op(0)}$$

$$jal = \overline{op(5)} \cdot \overline{op(4)} \cdot \overline{op(3)} \cdot \overline{op(2)} \cdot op(1) \cdot op(0)$$

[a]在Verilog HDL中不能這樣幹, 因爲有些指令名與Verilog HDL關鍵字相同

控制部件設計

控制信號的邏輯表達式

pcsource(1) =
pcsource(0) =
aluc(3) =
aluc(2) =
aluc(1) =
aluc(0) =
... =
wmem = st;
wreg = add + sub + and + or + xor + sll + srl + sra + addi + andi + ori + xori + lw + lui + jal;
... =
call = jal;

控制部件設計

首先確認是什麼指令, 即對指令進行譯碼

R 類型			I 類型	
指令	op(5..0)	func(5..0)	指令	op(5..0)
add	000000	100000	addi	001000
sub	000000	100010	andi	001100
and	000000	100100	ori	001101
or	000000	100101	xori	001110
xor	000000	100110	lw	100011
sll	000000	000000	sw	101011
srl	000000	000010	beq	000100
sra	000000	000011	bne	000101
jr	000000	001000	lui	001111

J 類型			
指令	op(5..0)	指令	op(5..0)
j	000010	jal	000011

控制部件設計

然後, 寫出控制信號的邏輯表達式

輸入		輸出									
指令	z	pcsource(1..0)	aluc(3..0)	shift	aluimm	sext	wmem	wreg	m2reg	regrt	call
add	x	00	0000	0	0	x	0	1	0	0	0
...	x										
sll	x	00	0011	1	0	x	0	1	0	0	0
...	x										
beq	0	00						0			
	1	01									
...	x										
j	x	11	xxxx	x	x	x	0	0	x	x	x
jal	x	11	xxxx	x	x	x	0	1	x	x	1

测試程序

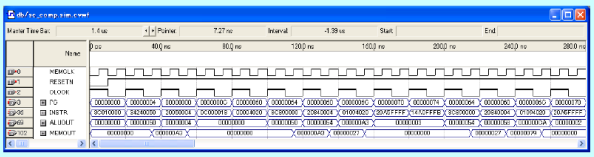
```
DEPTH = 64;           % Memory depth and width are required    %
WIDTH = 32;           % Enter a decimal number                 %
ADDRESS_RADIX = HEX;  % Address and value radixes are optional %
DATA_RADIX = HEX;     % Enter BIN, DEC, HEX, or OCT; unless    %
                      % otherwise specified, radixes = HEX     %
CONTENT
BEGIN
 0 : 3c010000; % (00)  main:   lui  $1, 0          # address of data[0]      %
 1 : 34240050; % (04)          ori  $4, $1, 80     # address of data[0]      %
 2 : 20050004; % (08)          addi $5, $0,  4     # counter                 %
 3 : 0c000018; % (0c)  call:   jal  sum            # call function           %
 4 : ac820000; % (10)          sw   $2, 0($4)      # store result            %
 5 : 8c890000; % (14)          lw   $9, 0($4)      # check sw                %
 6 : 01244022; % (18)          sub  $8, $9, $4     # sub: $8 <- $9 - $4      %
 7 : 20050003; % (1c)          addi $5, $0,  3     # counter                 %
 8 : 20a5ffff; % (20)  loop2:  addi $5, $5, -1     # counter - 1             %
 9 : 34a8ffff; % (24)          ori  $8, $5, 0xffff # zero-extend: 0000ffff  %
 A : 39085555; % (28)          xori $8, $8, 0x5555 # zero-extend: 0000aaaa  %
 B : 2009ffff; % (2c)          addi $9, $0, -1     # sign-extend: ffffffff  %
 C : 312affff; % (30)          andi $10, $9, 0xffff # zero-extend: 0000ffff  %
```

测試程序

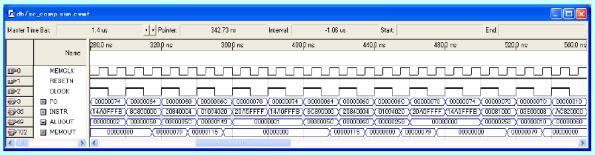
```
 D : 01493025; % (34)          or   $6, $10, $9     # or:  ffffffff          %
 E : 01494026; % (38)          xor  $8, $10, $9     # xor: ffff0000          %
 F : 01463824; % (3c)          and  $7, $10, $6     # and: 0000ffff          %
10 : 10a00001; % (40)          beq  $5, $0, shift   # if $5 = 0, goto shift  %
11 : 08000008; % (44)          j    loop2           # jump loop2             %
12 : 2005ffff; % (48)  shift:  addi $5, $0, -1      # $5   = ffffffff        %
13 : 000543c0; % (4c)          sll  $8, $5, 15      # <<15 = ffff8000        %
14 : 00084400; % (50)          sll  $8, $8, 16      # <<16 = 80000000        %
15 : 00084403; % (54)          sra  $8, $8, 16      # >>16 = ffff8000 (arith) %
16 : 000843c2; % (58)          srl  $8, $8, 15      # >>15 = 0001ffff (logic) %
17 : 08000017; % (5c)  finish: j    finish          # dead loop              %
18 : 00004020; % (60)  sum:    add  $8, $0, $0      # sum                    %
19 : 8c890000; % (64)  loop:   lw   $9, 0($4)      # load data              %
1A : 20840004; % (68)          addi $4, $4,  4      # address + 4            %
1B : 01094020; % (6c)          add  $8, $8, $9      # sum                    %
1C : 20a5ffff; % (70)          addi $5, $5, -1      # counter - 1            %
1D : 14a0fffb; % (74)          bne  $5, $0, loop    # finish?                %
1E : 00081000; % (78)          sll  $2, $8, 0       # move result to $v0     %
1F : 03e00008; % (7c)          jr   $ra             # return                 %
END ;
```


測試數據和模擬波形

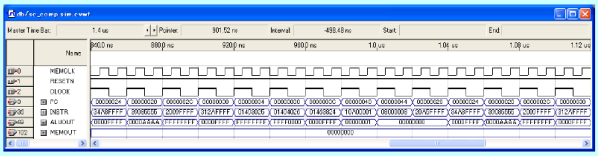
```
DEPTH = 32;           % Memory depth and width are required    %
WIDTH = 32;           % Enter a decimal number                 %
ADDRESS_RADIX = HEX;  % Address and value radixes are optional %
DATA_RADIX = HEX;     % Enter BIN, DEC, HEX, or OCT; unless    %
CONTENT               % otherwise specified, radixes = HEX     %
BEGIN
  14 :    000000A3; % (50) data[0] %
  15 :    00000027; % (54) data[1] %
  16 :    00000079; % (58) data[2] %
  17 :    00000115; % (5C) data[3] %
END ;
```



模擬波形



模擬波形



顶层代码结构，实验中具体指令 ROM 存储器和数据 RAM 存储器的实现方法有所不同。注意采用 Altera 宏模块进行实现时的差异。

Example: sc_comp.v

```
module sc_comp (clock,resetn,inst,pc,aluout,memout,mem_clk);
    input clock,resetn,mem_clk;
    output [31:0] inst,pc,aluout,memout;
    wire [31:0]   data;
    wire          wmem;
    sc_cpu cpu (clock,resetn,inst,memout,pc,wmem,aluout,data);
    scinstmem imem (pc,inst);
    scdatamem dmem (clock,memout,data,aluout,wmem,mem_clk,mem_clk);
endmodule
```


软件学院 2011-05