# 第六讲 DM（DATA MEM）模块设计

实验目的一：

1、了解Fpga芯片中内存的架构方式、实现原理；

2、掌握Vivado 内存ip核的设计过程；

3、学习和掌握利用verilog 调用、调试ip核的方法。

注释：具体可参见rom模块设计章节

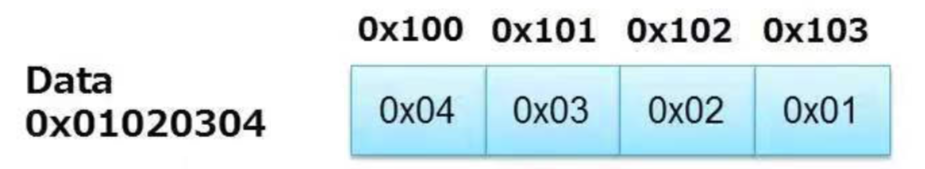
实验目的二：

1、使用verilog语言模拟设计并实现一个DM。它由32×8位的存储单元（即含有32个存储，每个存储单元8位.

2、初步了解使用Fpga实现DM设计的全过程，掌握vivado软件基本功能的使用。

原理：存储多字节数据类型，有两种顺序方式：大端存储（Big Endian）和小端存储（Little Endian），**大端(big endian):低地址存放高位字节。小端(little endian):低地址存放低位字节。**

RISC-V 在内存系统中采用小端字节排序，多字节数据的最小有效字节存储在最低内存地址中。如下图：



## 一、DM模块接口定义

**Module dm**(

**Input** clk, //100MHZ CLK

**input** DMWr, //write signal  
**input** [5:0] addr,

**input** [31:0] din,

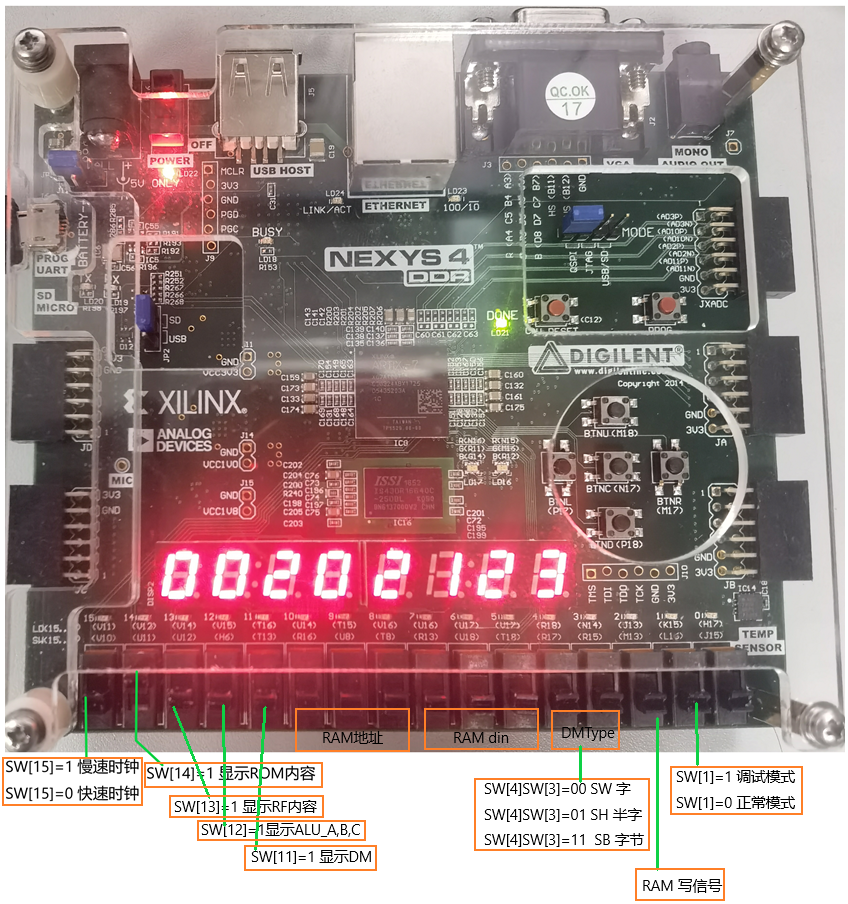
**input** [2:0] DMType,

**output reg** [31:0] dout

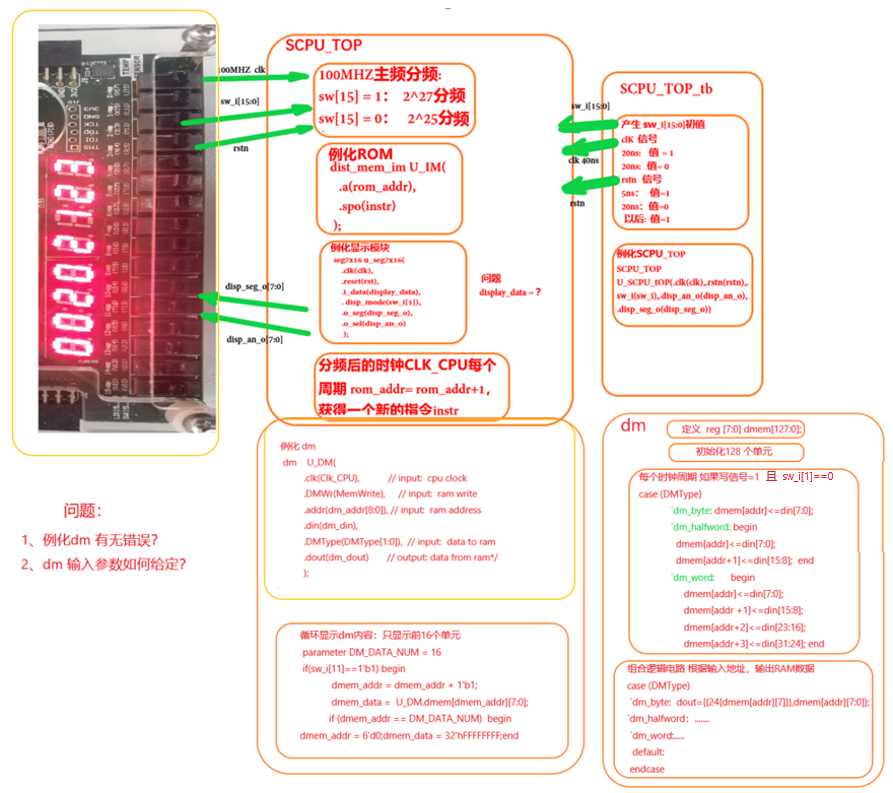
);

**endmodule**

## 二、NEXYS4主板开关定义



## 三、主模块、仿真模块和DM模块功能定义



注明：1）宏定义如下：

`define dm\_word 3'b000

`define dm\_halfword 3'b001

`define dm\_halfword\_unsigned 3'b010

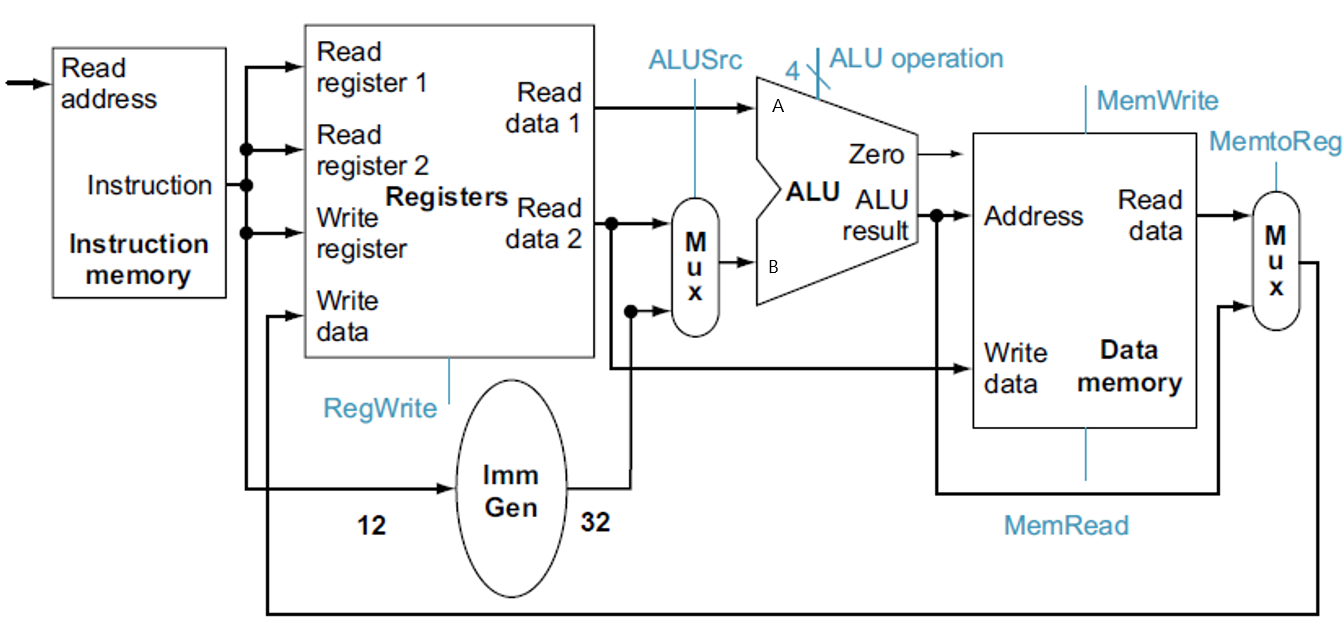
`define dm\_byte 3'b011

`define dm\_byte\_unsigned 3'b100

2）为了调试方便，dmem\_data={demem\_addr,{dmem\_data[27:0]}};这样每个单元号可以显示出来

# 第七讲 四大模块组成串行计算单元

## 一、四大模块连接框图：

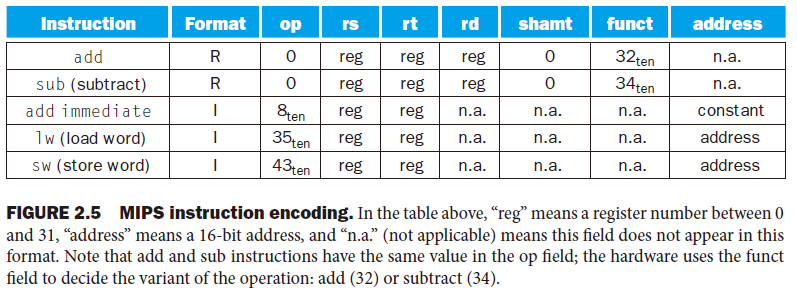


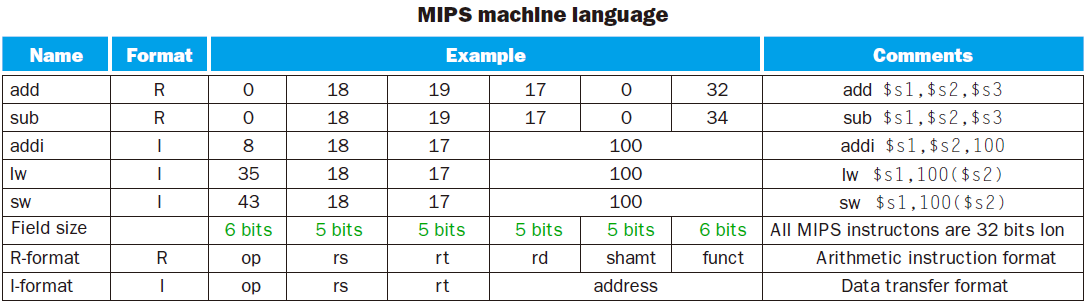
## 二、模拟串行运行一段代码

|  |  |  |
| --- | --- | --- |
| 代 码 | 功 能 | 信 号 |
| **add x1,x0,x0** | **X1=x0+x0 ; x1=0** | **rs1=X0 ; rs2=X0;**  **rd=X1; WD=?; RegWrite=1;**  **A= RD1；AluSrc=上通路; B= RD2 ;**  **Aluop=ADD；aluout=A+B=0**  **MemWrite=0；Memread=X;  MemtoReg=下通路；WD=aluout=0; WD=0 X1=0** |
| **addi x1,x1,1** | **X1=X1+1; X1=1** | **rs1=X1; rs2=XXX;**  **rd=X1; WD= ?; RegWrite=1;**  **A= RD1； AluSrc=下通路; B= IMM ; Aluop=ADD；c=A+imm=0+1=1**  **MemWrite=0；Memread=X;  MemtoReg=下通路；WD=aluout=1; WD=1 X1= 1** |
| **sw x1,1(x0)** | **X1=>MEM( X0+1 )** | **rs1=X0; rs2=X1;**  **rd= ; WD= XXX ; RegWrite=0;**  **A= RD1=0；AluSrc=下通路; B= IMM=1 ; Aluop=ADD；c=A+imm=1;**  **MemWrite=1；Memread=X; WriteData=RD2=X1 MemtoReg=XXX； RF\_WD = MEM(1)=1 ?** |
| **lw x4,1(x0)** | **X4=MEM(x0+1)** | **rs1=X0; rs2=;**  **rd= X4 ; WD= ?; RegWrite=1**  **A= RD1=0；AluSrc=下通路; B= IMM=1 ; Aluop=ADD；c=A+imm=1;**  **MemWrite=0；Memread=XXX; WriteData=XXX MemtoReg=上通道； WD= MEM(1)=1** |
| **add x2,x0,x0** | 学生填写 | 学生填写 |
| **addi x2,x2,2** | ? | ? |
| **sw x2,2(x0)** | ? | ? |
| **lw x4,1(x0)** | ? | ? |
| **add x3,x0,x0** | **?** | **?** |
| **addi x3,x3,3** | **?** | **?** |
| **sw x3,3(x0)** | **?** | **?** |
| **lw x6,3(x0)** | **?** | **?** |

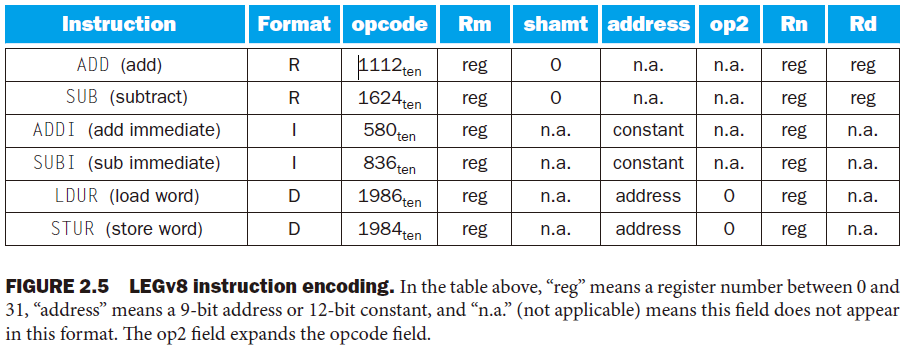
## 三、指令集ISA的分类

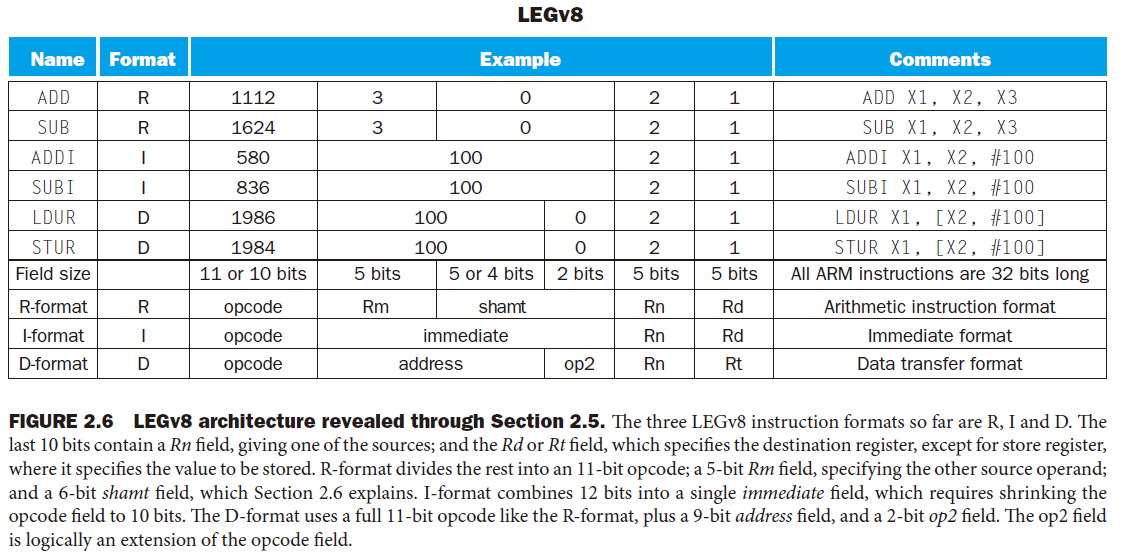
#### 3.1 Mips 指令集





#### 3.2 ARM 指令集



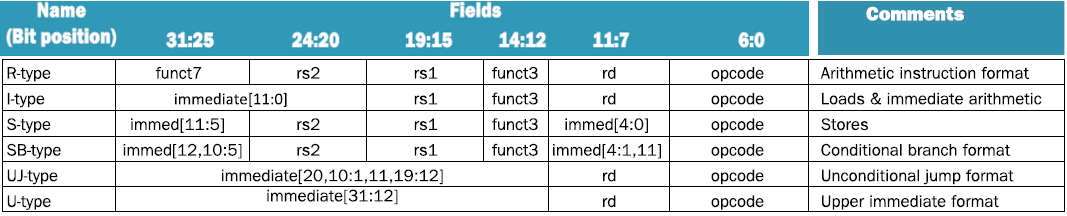


#### 3.3 RISC-V 指令集

##### 1、模块化的指令集满足不同的应用

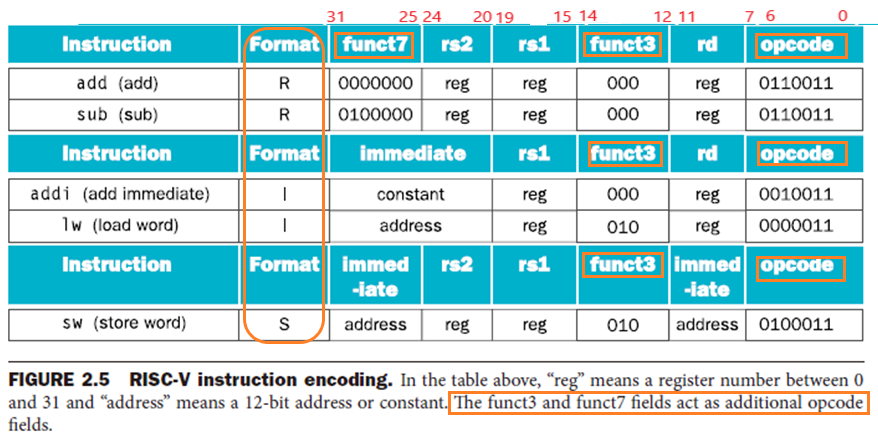
|  |  |  |
| --- | --- | --- |
| **基本指令集名称** | **指令数** | **特 点** |
| **RV32I** | **47** | **整数指令，32位地址空间，32个通用寄存器** |
| RV32E | 47 | 整数指令，RV32E的子集，支持16个通用寄存器 |
| RV64I | 59 | 整数指令，64位地址空间（含少数32位整数指令） |
| RV128I | 71 | 整数指令，128位地址空间 |
|  |  |  |
| **扩展指令集名称** | **指令数** | **特 点** |
| M | 8 | 整数乘除指令 |
| A | 11 | 原子内存操作指令（AMO用于处理器间同步的读-修改-写操作） |
| F | 26 | 单精度浮点运算指令 |
| D | 26 | 双精度浮点运算指令 |
| C | 46 | 压缩指令（16位） |

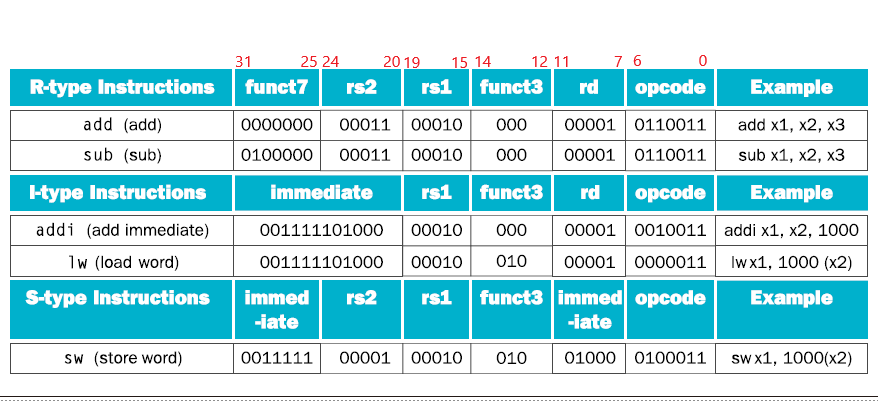
##### 2、指令格式分为六类



备注：jal指令中立即数（或label）为20位， jalr指令中立即数为12位。

#### 3.4 RISC-V 指令集举例：





## 四、RISC\_V指令的译码与控制信号生成

##### 4.1 指令译码单元

//Decode

Op = inst\_in[6:0]; // op

Funct7 = inst\_in[31:25]; // funct7

Funct3 = inst\_in[14:12]; // funct3

rs1 = inst\_in[19:15]; // rs1

rs2 = inst\_in[24:20]; // rs2

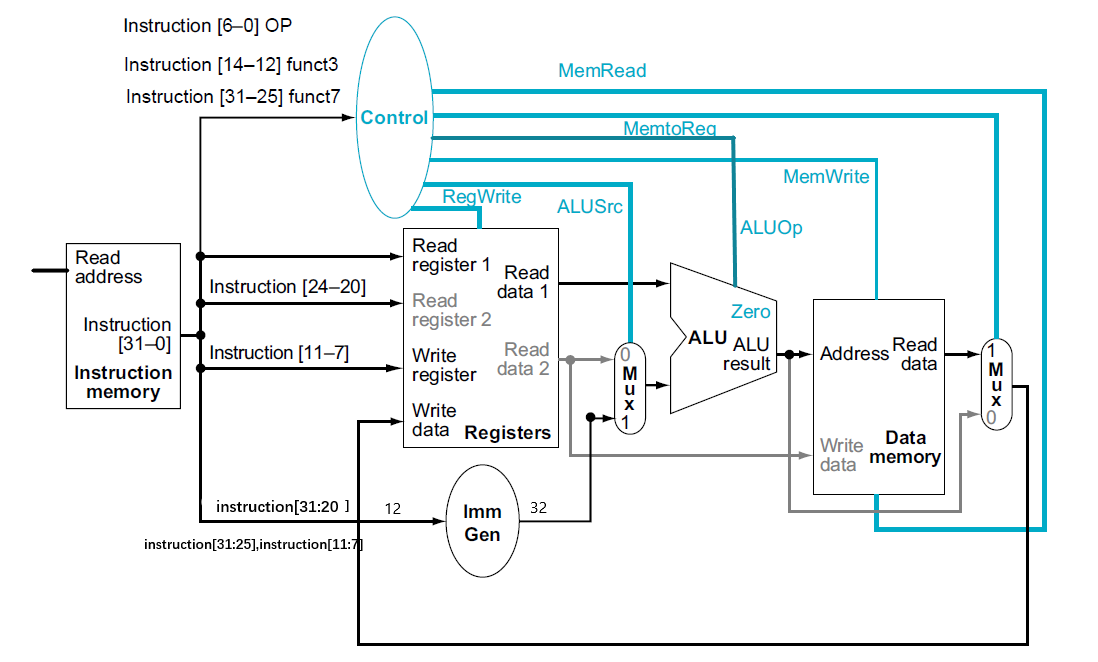
rd = inst\_in[11:7]; // rd

iimm=inst\_in[31:20];//addi 指令立即数，lw指令立即数

simm={inst\_in[31:25],inst\_in[11:7]}; //sw指令立即数

##### 4.2 控制信号生成

**4.2.1 增加Ctrl 模块，集中产生控制信号**，如下图所示：



**4.2.2 控制模块接口定义**

**Module ctrl**(

**input** [6:0] Op, //opcode

**input** [6:0] Funct7, //funct7   
**input** [2:0] Funct3, // funct3

**input** Zero,

**output** RegWrite, // control signal for register write

**output** MemWrite, // control signal for memory write

**output** [5:0]EXTOp, // control signal to signed extension

**output** [4:0] ALUOp, // ALU opertion

**//output** [2:0] NPCOp, // next pc operation

**output** ALUSrc， // ALU source for b

**output** [2:0] DMType, //dm r/w type

**output** WDSel // (register) write data selection (MemtoReg)

); **endmodule**

**4.2.2 Ctrl 模块控制流程：**

**操作码（op funct7 funct3）确定具体操作指令类型**

**R\_type:**

rtype = ~Op[6]&Op[5]&Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0]; //0110011

i\_add=rtype&~Funct7[6]&~Funct7[5]&~Funct7[4]&~Funct7[3]&~Funct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&~Funct3[1]&~Funct3[0]; // add 0000000 000

i\_sub=rtype&~Funct7[6]&Funct7[5]&~Funct7[4]&~Funct7[3]&~Funct7[2]&~Funct7[1]&~Funct7[0]&~Funct3[2]&~Funct3[1]&~Funct3[0]; // sub 0100000 000

**i\_l type**

wire itype\_l = ~Op[6]&~Op[5]&~Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0]; //0000011

wire i\_lb=itype\_l&~Funct3[2]& ~Funct3[1]& ~Funct3[0]; //lb 000

wire i\_lh=itype\_l&~Funct3[2]& ~Funct3[1]& Funct3[0]; //lh 001

wire i\_lw=itype\_l&~Funct3[2]& Funct3[1]& ~Funct3[0]; //lw 010

**// i\_i type**

wire itype\_r = ~Op[6]&~Op[5]&Op[4]&~Op[3]&~Op[2]&Op[1]&Op[0]; //0010011

wire i\_addi = itype\_r& ~Funct3[2]& ~Funct3[1]& ~Funct3[0]; // addi 000 func3

**// s format**

wire stype = ? ;//0100011

wire i\_sw = ? // sw 010

wire i\_sb=stype& ~Funct3[2]& ~Funct3[1]&~Funct3[0];

wire i\_sh=stype&& ~Funct3[2]&~Funct3[1]&Funct3[0];

**操作指令生成控制信号（写、MUX选择）**

assign RegWrite = rtype | itype\_r|itype\_l ; // register write

assign MemWrite = stype; // memory write

assign ALUSrc = itype\_r | stype | itype\_l ; // ALU B is from instruction immediate

//mem2reg=wdsel ,**WDSel\_FromALU 2'b00 WDSel\_FromMEM 2'b01**

assign WDSel[0] = itype\_l;

assign WDSel[1] = 1'b0;

**操作指令生成运算类型aluop**

//ALUOp\_nop 5'b00000

//ALUOp\_lui 5'b00001

//ALUOp\_auipc 5'b00010

//ALUOp\_add 5'b00011

assign ALUOp[0]= i\_add | i\_addi|stype|itype\_l ;

assign ALUOp[1]= i\_add | i\_addi|stype|itype\_l ;

**操作指令生成常数扩展操作**

//assign EXTOp[0] = stype;

//assign EXTOp[1] = itype\_l | itype\_r ;

assign EXTOp[5] = i\_slli | i\_srai | i\_srli;

assign EXTOp[4] = (itype\_l | itype\_r) & ~i\_slli & ~i\_srai & ~i\_srli;

assign EXTOp[3] = stype;

assign EXTOp[2] = sbtype;

assign EXTOp[1] = i\_lui | i\_auipc;

assign EXTOp[0] = i\_jal;

**根据具体S和i\_L指令生成DataMem数据操作类型编码**

// dm\_word 3'b000

//dm\_halfword 3'b001

//dm\_halfword\_unsigned 3'b010

//dm\_byte 3'b011

//dm\_byte\_unsigned 3'b100

assign DMType[2]=i\_lbu;

assign DMType[1]=i\_lb | i\_sb | i\_lhu;

assign DMType[0]=i\_lh | i\_sh | i\_lb | i\_sb;

##### 4.3 符号扩展模块

**4.3.1 符号扩展模块定义**

**Module EXT**(

**input** [4:0] iimm\_shamt, //

**input** [11:0] iimm, //instr[31:20], 12 bits

**input** [11:0] simm, //instr[31:25, 11:7], 12 bits

**input** [11:0] bimm,//instrD[31],instrD[7], instrD[30:25], instrD[11:8], 12 bits

**input** [19:0] uimm,

**input** [19:0] jimm,

**input** [5:0] EXTOp,

**output** reg [31:0] immout

);**endmodule**

**4.3.2 符号扩展模块扩展**

case (EXTOp)

`EXT\_CTRL\_ITYPE\_SHAMT: immout<={27'b0,iimm\_shamt[4:0]};

`EXT\_CTRL\_ITYPE: immout<={ {20{ iimm[11]}},iimm[11:0]}；

`EXT\_CTRL\_STYPE: immout<={ {20{ simm[11]}},simm[11:0]};

`EXT\_CTRL\_BTYPE: immout<={ {19{ bimm[11]}},bimm[11:0], 1'b0} ;

`EXT\_CTRL\_UTYPE: immout <= {uimm[19:0], 12'b0};

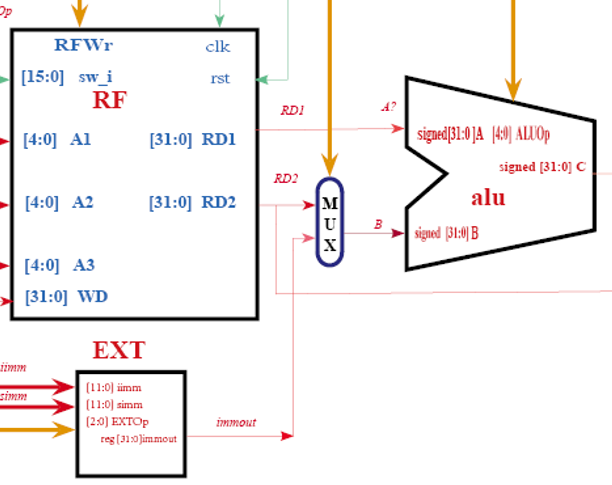
`EXT\_CTRL\_JTYPE: immout<={{11{ jimm[19]}},jimm[19:0],1'b0};

default: immout <= 32'b0;

endcase

4.4 多路选择器

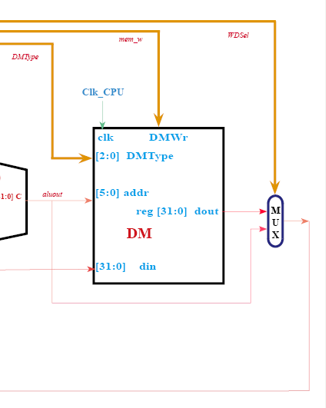
**4.4.1 alu多路选择器**

****

wire[31:0] B ;

assign B = (ALUSrc) ? immout : RD2;

**4.4.2 RF寄存器wd多路选择器**



always @\*

begin

case(WDSel)

`WDSel\_FromALU: WD<=aluout;

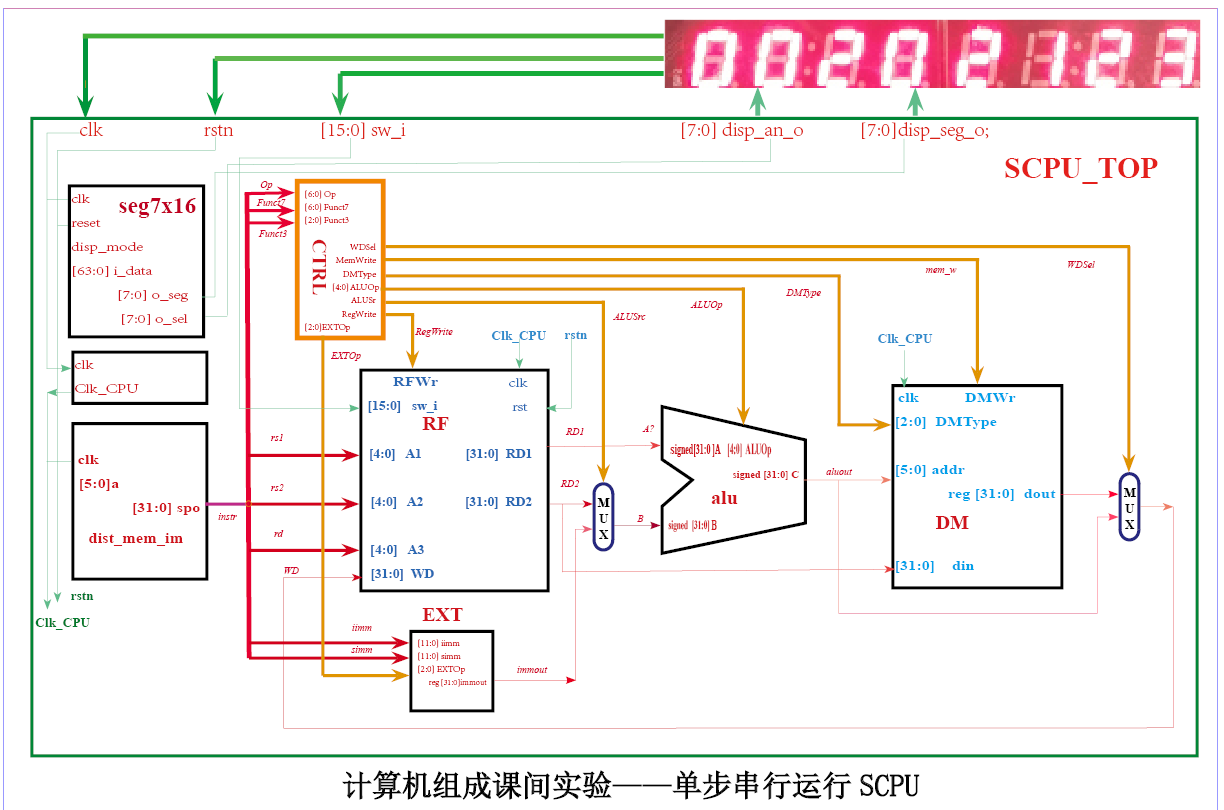
`WDSel\_FromMEM: WD<=dout;

//`WDSel\_FromPC: WD<=PC\_out+4;

endcase

end

## 五、四大模块详细连接框图



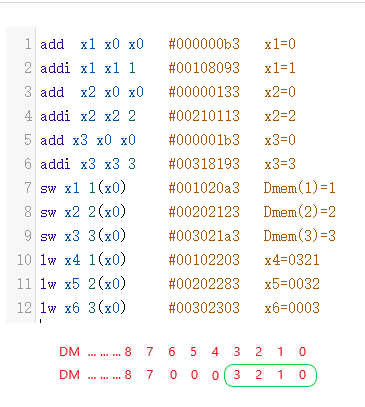
备注：

1）拨动sw\_i[14:11]查看各部件显示内容，系统处要处于调试模式，此时sw\_i[1] =1’b1,此时rom\_addr =rom\_addr,不会移动，只有sw\_i[1]=1’b0时候，rom\_addr = rom\_addr + 1；

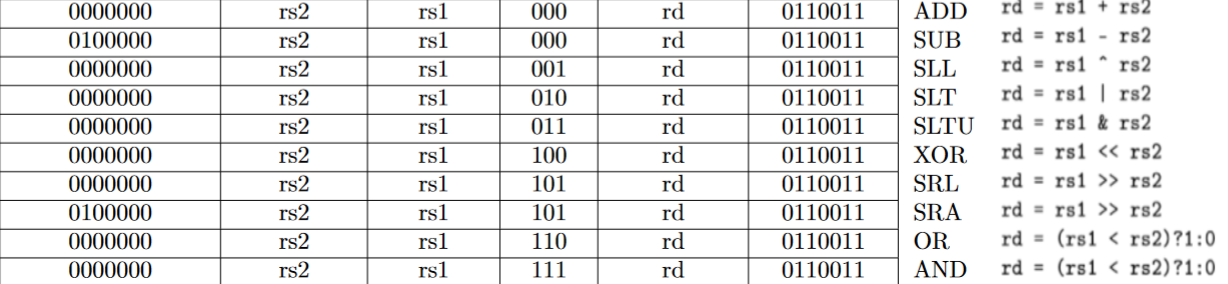
修改 rom\_addr显示模块如下：if (sw\_i[1] == 1’b0) rom\_addr = rom\_addr + 1；

2）如果rf，dm模块的写信号是posedeg 时钟周期有效，那么wd，din信号的写入是下个周期开始的上升沿。

3）附上12条指令coe的原码



附录：1）[R型指令介绍](https://blog.csdn.net/qq_70829439/article/details/129565365#I_8)



注释：

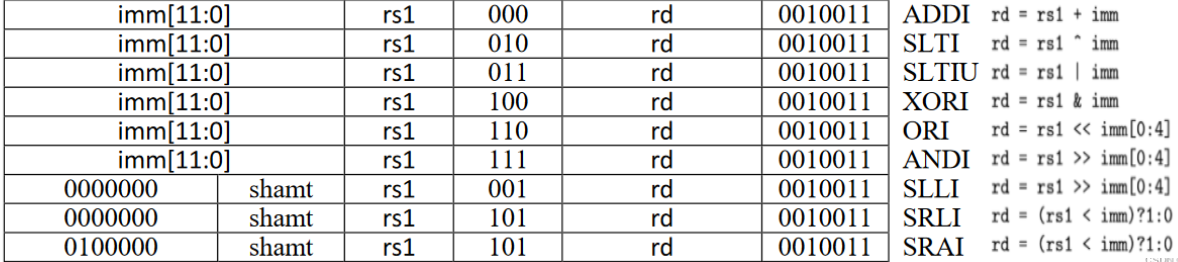
A)SLT：x[rd] = (x[rs1] < x[rs2])? 1:0

B)SLL rd,rs1,rs2（shift left logical），x[rd] = x[rs1] << x[rs2]，rs1左移rs2位(低5位)，空出的位补0，结果写入rd寄存器

C) SRA rd,rs1,rs2（shift right arithmetic） rs1右移rs2位(低5位)，空出的位用rs1的最高位补充，结果写入rd寄存器

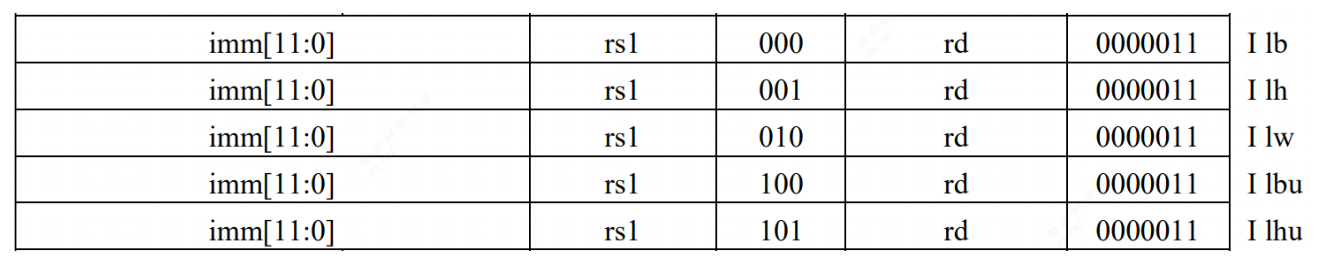
D）sra a0, a1, a2 //a0 = a1 >> a2 (算术右移，高位补原来的符号位)

2）[I型指令介绍](https://blog.csdn.net/qq_70829439/article/details/129565365#I_8)



注释：SLLI（shift left logical imm），其中Imm（立即数，占12位），shamt（位移次数，占5位）；

L型指令



注释：

lb 内存取出的是8位数

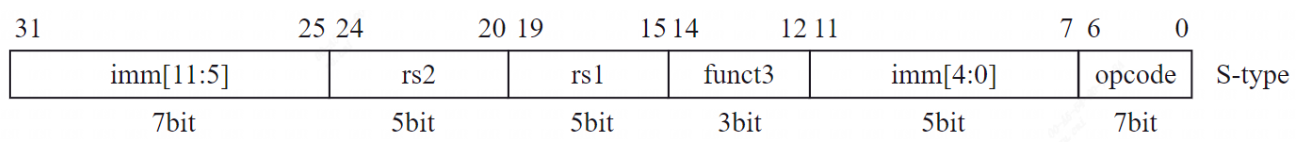
lh 内存取出16位数

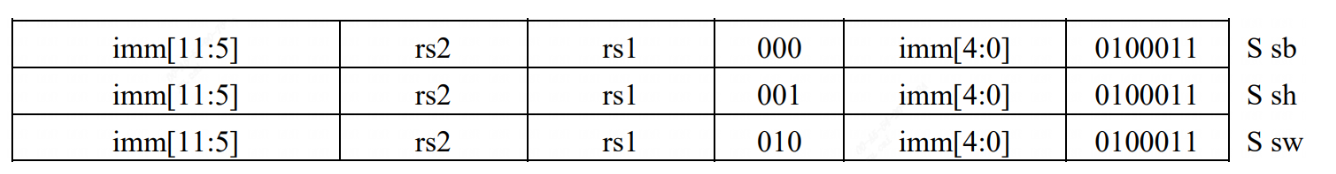
lw 内存取出32位数

lbu内存取出8位无符号数

lhu内存取出16位无符号数

3）[S型指令介绍](https://blog.csdn.net/qq_70829439/article/details/129565365#I_8)





注释：A）imm[11:5]+ imm[4:0] B）sb将rs1的低8位写入；sh将rs1的低16位写入；sw将rs1的32位写入

4）<https://venus.kvakil.me/> 仿真编译器

最后附37条指令图

