汇编阅读

Verilog

- **编译目录的结构**

  请回答

名称	修改日期	类型	大小
include	2018/10/11 16:34	文件夹	
inst	2018/9/18 9:19	文件夹	
obj	2018/9/18 9:19	文件夹	
bin.lds.S	2018/9/17 17:36	S 文件	3 KB
convert.c	2018/9/17 17:36	C 文件	2 KB
Makefile	2018/9/18 9:17	文件	2 KB
Readme_first.txt	2018/9/18 9:18	文本文档	2 KB
rules.make	2018/9/17 17:36	MAKE 文件	1 KB
start.S	2018/9/18 9:18	S 文件	7 KB

- **宏指令：**

  bal、b、

  li=lui+ori/addiu

- **伪指令（指示语句）：**

  .set、.text

  .org n

  .align n

  .set noreorder：gcc4.3-ls232缺陷

- **数字标号：**

  1f、1b

```
##s4, exception pc
    .set    noreorder
    .globl  _start
    .globl  start
    .globl  __main
_start:
start:

    b   locate
    nop

##avoid "j locate" not taken
    lui   t0, 0x8000
    addiu t1, t1, 1
    or    t2, t0, zero
    addu  t3, t5, t6
    lw    t4, 0(t0)
    nop

##avoid cpu run error
.org 0x0ec
```

- **数字标号：**

  1f、1b

- **注释：**

  #或//或//**

  #define、#if、#include不是注释

```
    sll t0, t0, 16      #t0 = switch<<16
1:
    addiu t0, 1
2:
    addiu t0, -1
    bne t0,zero, 2b
    nop
    jr ra
    nop
```

- **条件编译：**

  #if 条件1

  #elif 条件2

  #else

  #endif

```
run_test:
#if CMP_FUNC==1
    bal shell1
    nop
#elif CMP_FUNC==2
    bal shell2
    nop
#elif CMP_FUNC==3
    bal shell3
    nop
#elif CMP_FUNC==4
    bal shell4
#else
    b go_finish
    nop
#endif
go_finish:
```

- **汇编传参：**

  a0、a1、a2、a3

- **参数多路怎么办？**

  使用“无参数函数”中转

```
    nop
#endif
#if 1
    li      a0, 100 //LOOP
    bal     shell5  //dhrystone
    nop
    sw      v0, 0(sp)
    addiu   sp, sp, -4

    li      a0, 100 //LOOP
    bal     matrix_test
    sw      v0, 0(sp)
    addiu   sp, sp, -4
    nop
  //printf pass or fail
    lw      t0, -4(sp)
```

```
void shell(void)
{
    write_confreg_cr(1,0x1234);
    printf("coremark test begin.\n");
    int r = core_mark(0,0,0x66,ITERATIONS,7,1,2000);
    if(r == 0){
        printf("coremark PASS!\n");
        *((volatile int *)LED_RG1_ADDR) = 1;
        *((volatile int *)LED_RG0_ADDR) = 1;
        *((volatile int *)LED_ADDR)     = 0xffff;
    }else{
        printf("coremark ERROR!!!\n");
        *((volatile int *)LED_RG1_ADDR) = 1;
        *((volatile int *)LED_RG0_ADDR) = 2;
        *((volatile int *)LED_ADDR)     = 0;

    }

    return;
}
```

- **反汇编文件：**

  mipsel-liunx-objdump –alD elf文件 > test.s

- **反汇编格式：**

  地址:　二进制值　汇编码

```
Disassembly of section .text:

bfc00000 <_ftext>:
/media/sf_xjz/class/ucas/ucas18_19/develop/lab_all/ucas_CDE/soft/func_lab2_3/start.S:23
bfc00000:   100000e2    b   bfc0038c <locate>
/media/sf_xjz/class/ucas/ucas18_19/develop/lab_all/ucas_CDE/soft/func_lab2_3/start.S:24
bfc00004:   00000000    nop
/media/sf_xjz/class/ucas/ucas18_19/develop/lab_all/ucas_CDE/soft/func_lab2_3/start.S:27
bfc00008:   3c088000    lui t0,0x8000
/media/sf_xjz/class/ucas/ucas18_19/develop/lab_all/ucas_CDE/soft/func_lab2_3/start.S:28
bfc0000c:   25290001    addiu   t1,t1,1
/media/sf_xjz/class/ucas/ucas18_19/develop/lab_all/ucas_CDE/soft/func_lab2_3/start.S:29
bfc00010:   01005025    move    t2,t0
/media/sf_xjz/class/ucas/ucas18_19/develop/lab_all/ucas_CDE/soft/func_lab2_3/start.S:30
bfc00014:   01ae5821    addu    t3,t5,t6
/media/sf_xjz/class/ucas/ucas18_19/develop/lab_all/ucas_CDE/soft/func_lab2_3/start.S:31
bfc00018:   8d0c0000    lw  t4,0(t0)
```

链接脚本：

mipsel-liunx-readelf读elf文件结构

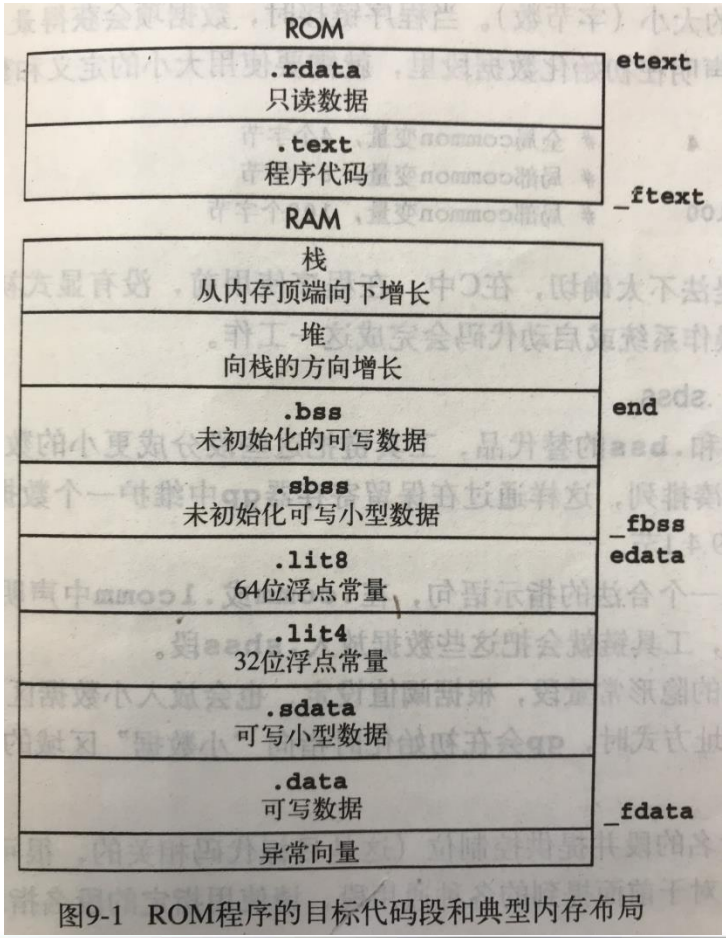


图9-1 ROM程序的目标代码段和典型内存布局

bin.lds.S

```
1  OUTPUT_ARCH(mips)
2  ENTRY(_start)
3  SECTIONS
4  {
5    /* Read-only sections, merged into text s
6    . = 0xbfc00000;
7    .text      :
8    {
9      _ftext = . ;
10     *(.text)
11     *(.rodata*)
12     *(.reginfo)
13     *(.init)
14     *(.stub)
15     /* .gnu.warning sections are handled sp
16     *(.gnu.warning)
17     rodata_end = .;
18   } =0
19   _etext = .;
20   PROVIDE (etext = .);
21   .fini      : { *(.fini)    } =0
22   . = MEMSTART;
23   .data     : AT(rodata_end)
24   {
25     _fdata = . ;
26     _stack = _fdata + MEMSIZE -32;
27     *(.data)
28     *(.data*)
```

Makefile

```
TOPDIR=$(shell pwd)

all:
    make compile
compile:main.bin main.data convert
    ./convert
    mkdir -p $(OBJDIR)
    mv main.elf  $(OBJDIR)/.
    mv test.s    $(OBJDIR)/.
    mv main.bin  $(OBJDIR)/.
    mv main.data $(OBJDIR)/.
    mv *.coe     $(OBJDIR)/.
    mv *.mif     $(OBJDIR)/.

main.bin:main.elf
    ${CROSS_COMPILE}objcopy -O binary -j .text $< $@

main.data:main.elf
    ${CROSS_COMPILE}objcopy -O binary -j .data $< $@

main.elf: start.o libinst.a
    ${CROSS_COMPILE}gcc -E -P -Umips -D_LOADER -U_MAIN $(CFLAGS) bin.lds.S -o bin.lds
    ${CROSS_COMPILE}ld -g -T  bin.lds  -o $@ start.o -L . -linst
    ${CROSS_COMPILE}objdump -alD $@ > test.s

libinst.a:
    make -C inst $(TOPDIR) $@

convert:convert.c
    gcc  $(ALIGNED) -o convert  convert.c
```

convert.c

lds.S | Makefile | Makefile | convert.c

```
}

int main(void)
{
    FILE *in;
    FILE *out;

    int i,j,k;
    unsigned char mem[32];

    in = fopen("main.bin", "rb");
    out = fopen("inst_ram.coe","w");

    fprintf(out, "memory_initialization_radix = 16;\n");
    fprintf(out, "memory_initialization_vector =\n");
    while(!feof(in)) {
        if(fread(mem,1,4,in)!=4) {
            fprintf(out, "%02x%02x%02x%02x\n", mem[3], mem[2],  mem[1], mem[0]);
        break;
         }
        fprintf(out, "%02x%02x%02x%02x\n", mem[3], mem[2], mem[1],mem[0]);
        }
    fclose(in);
    fclose(out);

    in = fopen("main.data", "rb");
    out = fopen("data_ram.coe","w");

    fprintf(out, "memory_initialization_radix = 16;\n");
```

汇编阅读

Verilog

- Verilog是描述电路的硬件语言
- 所有软件能做的事情，硬件都能做

描述电路的难点

- 不在于写代码
- 而在于结构框图（空间维度）、状态机（时间维度）

就像算法设计师和普通软件编程人员的关系

对CPU实验

- 理清结构是关键！

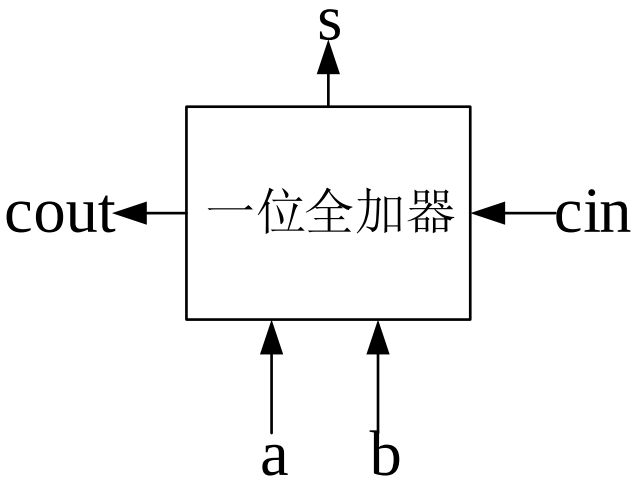
Verilog基本要素

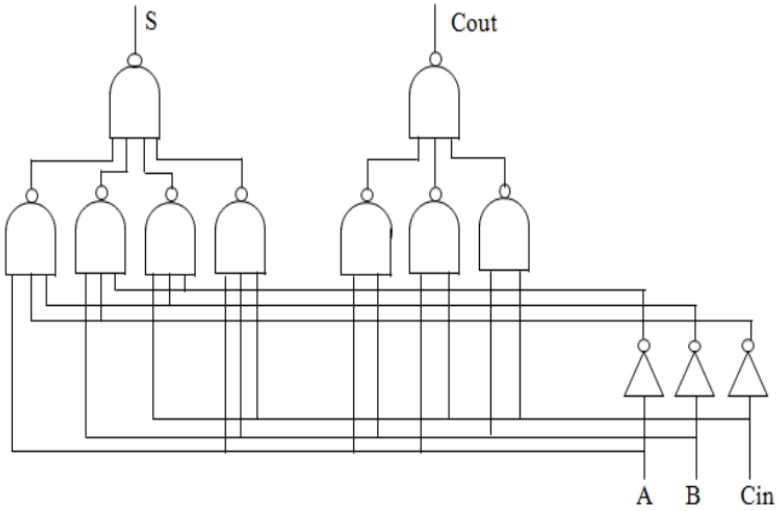
- **完成CPU设计，最少只需要掌握Verilog的三种语句**
  - 组合逻辑：assign
  - 时序逻辑：always @(posedge clock)
  - 模块定义及实例化：module/endmodule

Verilog语言——组合逻辑

- **1位全加器**

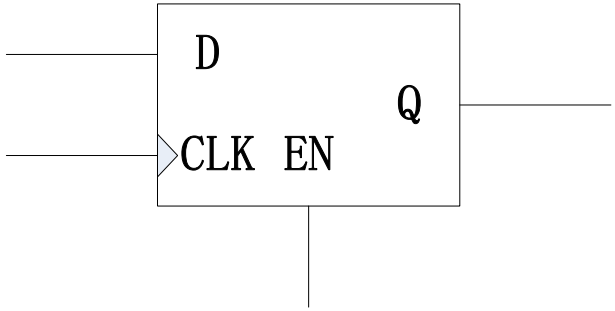
```
module full_adder
(
    input  wire a,
    input  wire b,
    input  wire cin,
    output wire s,
    output wire cout
);

assign s = a^b^cin;
assign cout = (a&b) | (b&cin) | (cin&a);

endmodule
```





- 1位D触发器

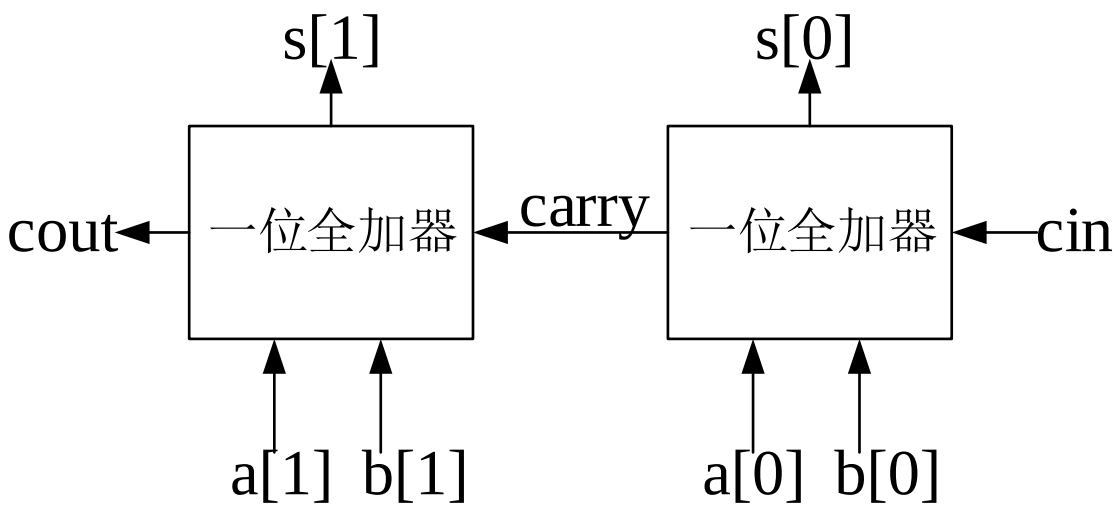
```
module dflipflop
(
    input  wire CLK,
    input  wire D,
    input  wire EN,
    output reg  Q
);

always @(posedge CLK)
begin
    if (EN) begin
        Q <= D;
    end
end
```



Verilog语言——模块调用

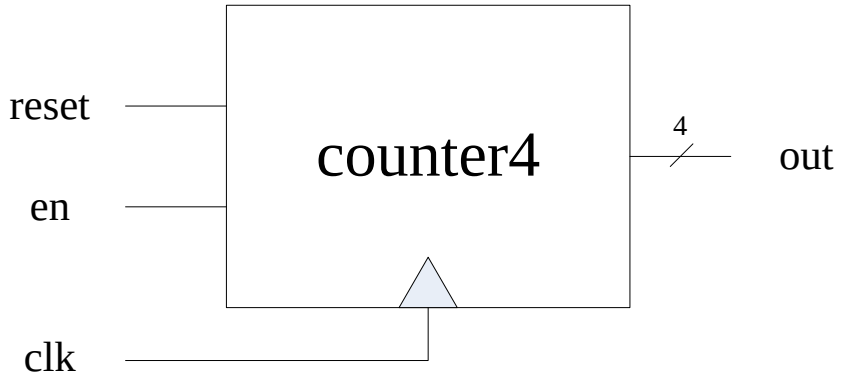
- **2位串行加法器**

```
module adder2
(
    input  wire [1:0] a,
    input  wire [1:0] b,
    input  wire       cin,
    output wire [1:0] s,
    output wire       cout
);

wire carry;
full_adder  bit0(.a(a[0]),.b(b[0]),.c(cin  ),.s(s[0]),.cout(carry));
full_adder  bit1(.a(a[1]),.b(b[1]),.c(carry),.s(s[1]),.cout(cout ));

endmodule
```



练习——四位计数器

- **功能规格定义：**
  - 输入clk、reset、en，输出out[3:0]；
  - 复位信号reset为同步复位，复位值为0；
  - 仅当en为1时，该counter每时钟周期自增1



myCPU中

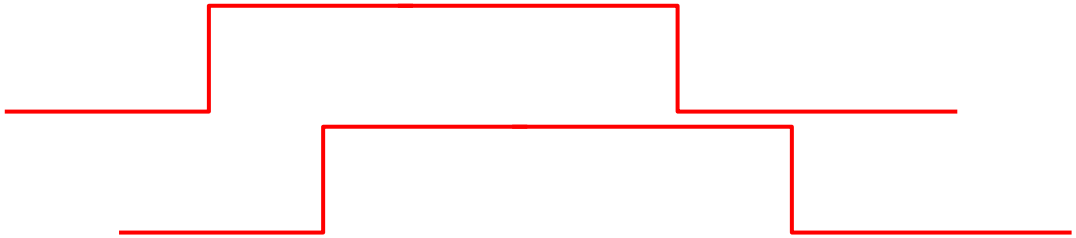
- 禁止initial语句
- 禁止使用always @（*)
- 禁止使用always @（a)
- 禁止使用always @（posedge clk or negedge resetn)
- 只允许always @(posedge clk)　——时序电路
- assign　——组合电路
- 建议：一个always只对一个变量赋值
- 请多多使用中间变量
- 常量加宽度：　1’b0

理清逻辑，不要重复

```
always @(posedge clk)
begin
        if (!resetn)
                a<=1'b0;
        else if(…)
                a<=!resetn ? 1'b0 :  ……
        else
                a<=a;
end

assign b = a&!resetn;
```

resetn请只用在复位端，组合逻辑慎用！！！

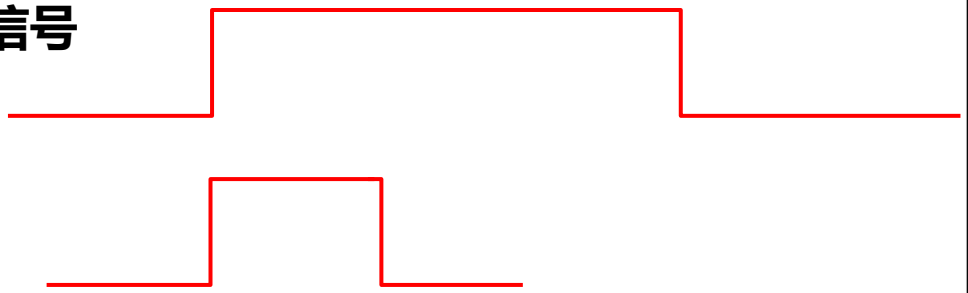
- 如何检测一个信号的上升和下降?



- 如何实现一个信号第一拍为1，第二拍为0?

```
always @（posedge clk）
        if (!resetn) a<=1’b0;
        else if(set)   a<=1’b1;
        else if (a)  a<=1’b0;
```

- 如何实现第一拍不运行，第二拍再运行?
  - 创造一个一拍的no_run信号
  - run & ~no_run
  - 多拍呢?



- 其他?

谢谢！