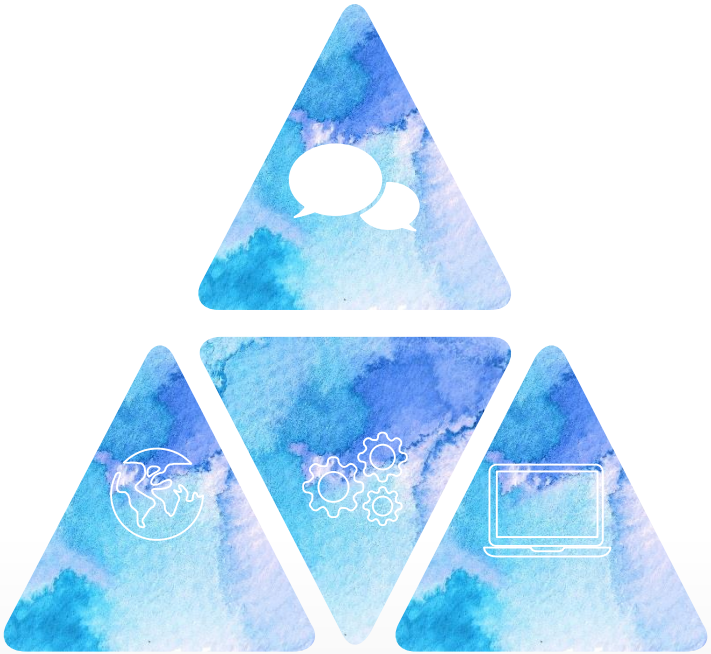
CPU设计汇报

汇报人：智能1601　201608010312 张宁

目录

设计要求



相对简单CPU的规格说明

1．64K字节的存储器，每个存储单元8位宽。

地址引脚A[15..0]

数据引脚D[7..0]

2．CPU的三个内部寄存器

◆ 8位累加器AC：接受任何算术或者逻辑运算的结果，并为使用两个操作数的算术或者逻辑操作指令提供一个操作数。

◆ 寄存器R：一个8位通用寄存器。它为所有的双操作数算术和逻辑运算指令提供第二个操作数。它也可以用来暂时存放累加器马上要用到的数据。（减少存储器访问次数提高CPU的性能）

◆ 零标志位Z：每次执行算术运算或者逻辑运算的时候，它都将被置位。

寄存器

◆ 16位的地址寄存器AR：通过引脚A[15..0]向存储器提供地址。

◆ 16位的程序计数器PC：存放的是将要执行的下一条指令的地址，或者指令需要的下一个操作数的地址。

◆ 8位的数据寄存器DR：通过D[7..0]从存储器中接收指令和数据并且向存储器传送数据。

◆ 8位的指令寄存器IR：存放的是从存储器中取出来的操作码。

◆ 8位的临时寄存器TR：在指令执行过程中，临时存储数据。（程序员不能访问）

指令集结构

指令	指令码	操作
NOP	0000 0000	无
LDAC	0000 0001 Γ	AC←M[Γ]
STAC	0000 0010 Γ	M[Γ]←AC
MVAC	0000 0011	R←AC
MOVR	0000 0100	AC←R
JUMP	0000 0101 Γ	GOTO Γ
JMPZ	0000 0110 Γ	IF（Z＝1） THEN GOTO Γ
JPNZ	0000 0111 Γ	IF（Z＝0） THEN GOTO Γ

指令集结构

ADD	0000 1000	AC←AC＋R，IF（AC＋R＝0） THEN Z←1 ELSE Z←0
SUB	0000 1001	AC←AC－R，IF（AC－R＝0） THEN Z←1 ELSE Z←0
INAC	0000 1010	AC←AC＋1，IF（AC＋1＝0） THEN Z←1 ELSE Z←0
CLAC	0000 1011	AC←0a，Z←1
AND	0000 1100	AC←AC∧R，IF（AC∧R＝0） THEN Z←1 ELSE Z←0
OR	0000 1101	AC←AC∨R，IF（AC∨R＝0） THEN Z←1 ELSE Z←0
XOR	0000 1110	AC←AC⊕R，IF（AC⊕R＝0） THEN Z←1 ELSE Z←0
NOT	0000 1111	AC←AC’，IF（AC’＝0） THEN Z←1 ELSE Z←0

指令执行具体过程

执行指令

LDAC指令

- LDAC是一条多字指令。
- 它包含三个字：
- 操作码　地址的低半部分　地址的高半部分
- 功能：从存储器中获得地址，然后从存储器中获得数据，并把数据装载到累加器中。

F1:AR<-PC

F2:DR<-M,PC<-PC+1

F3:IR<-DR,AR<-PC

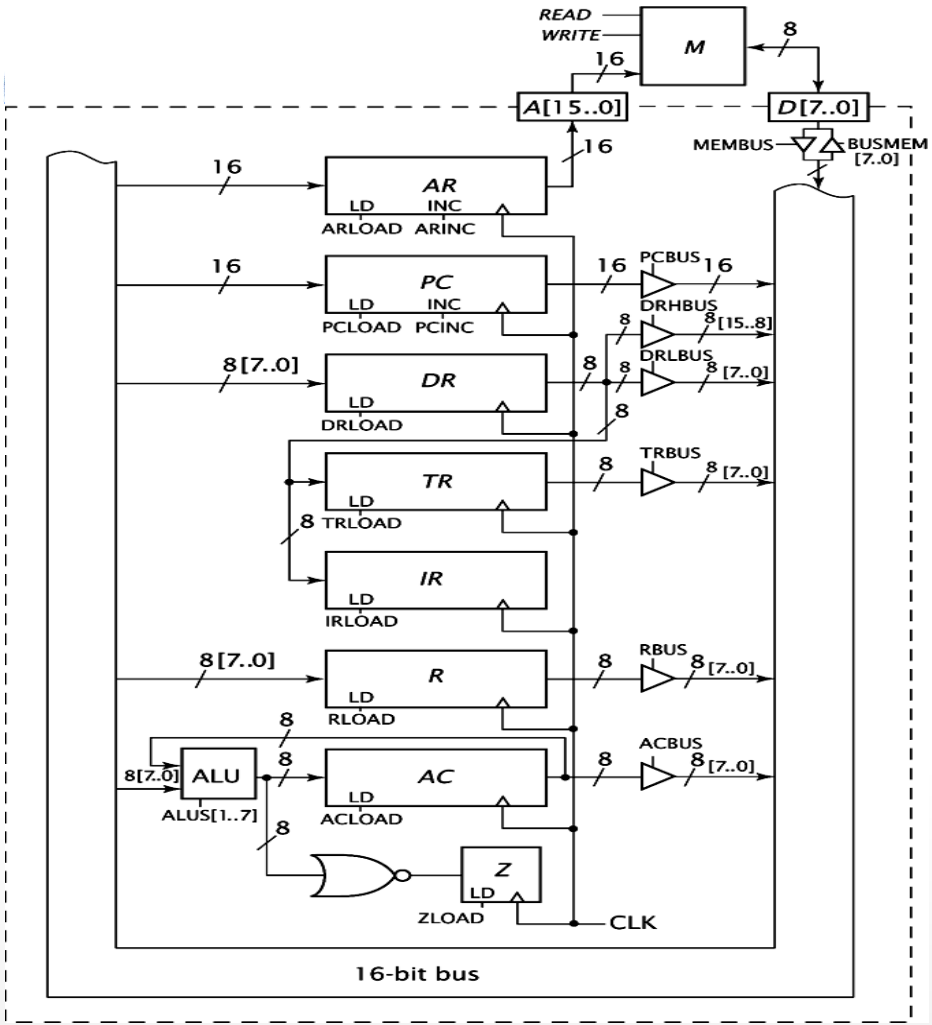
L1:DR<-M, PC←PC＋1，AR←AR＋1

L2:TR←DR，DR←M，PC←PC＋1

L3:AR←DR，TR

L4:DR←M

L5:AC←DR



JUMP指令

JUMP1:　DR←M，AR←AR+1

JUMP2:　TR←DR，DR←M

JUMP3:　PC←DR，TR

JMPZ指令的状态：

JMPZY1:　DR←M，AR←AR＋1

JMPZY2:　TR←DR，DR←M

JMPZY3:　PC←DR，TR

JMPZN1:　PC←PC＋1

JMPZN2:　PC←PC＋1

其余的指令都是在一个状态内完成的。

1、明确CPU的功能、目的和基本规格
2、设计指令集结构
3、取指令、译码（画出状态图）
4、执行指令（明确指令的状态）
5、创建数据通路
6、设计ALU等
7、设计控制单元
8、产生CPU的状态

02

设计思想

环境：Quartus II 9.0
仿真：Quartus 下进行波形仿真

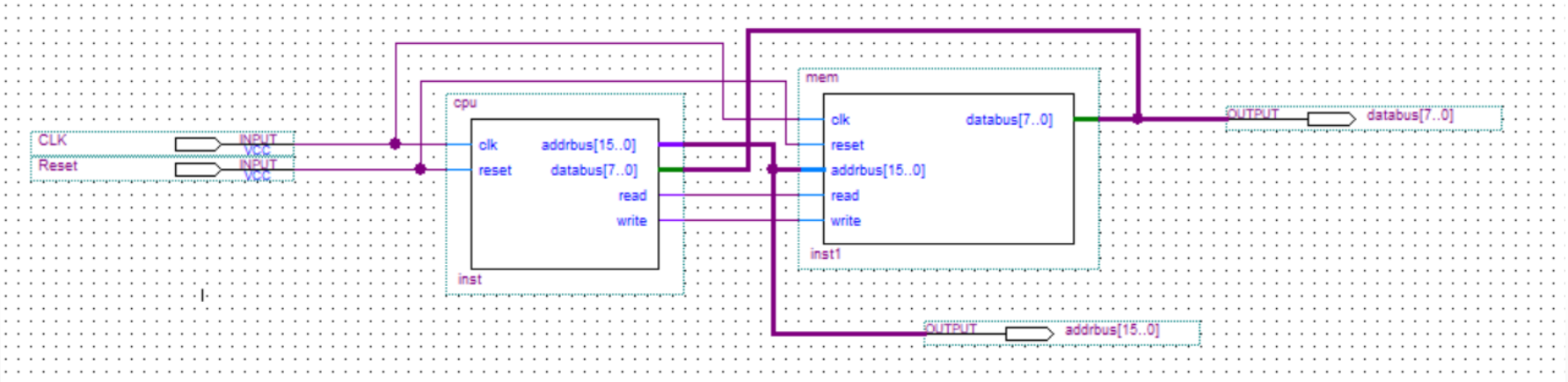
- 根据状态图对指令执行状态译码
  - 其中LDAC、STAC需要5个状态完成，JUMP需要3个状态完成，其余均一个状态完成。

```
constant ldac1:     std_logic_vector(5 downto 0) := "001110";
constant ldac2:     std_logic_vector(5 downto 0) := "001111";
constant ldac3:     std_logic_vector(5 downto 0) := "010000";
constant ldac4:     std_logic_vector(5 downto 0) := "010001";
constant ldac5:     std_logic_vector(5 downto 0) := "010010";
```

- 时钟上升沿时，根据指令更新寄存器的值
- 产生下一状态的进程（根据当前指令产生下一状态）
- 对每个状态译码（信号如何变化）

在mem.vhd文件中，验证1+2+…+n。（没有过多改动）

在comp工程中，把cpu.vhd、rsisa.vhd、mem.vhd三个文件添加进来，通过画图的方式连接。



03

代码实现

cpu.vhd:

```
signal pc: std_logic_vector(15 downto 0);
signal ac: std_logic_vector(7 downto 0);
signal r: std_logic_vector(7 downto 0) ;
signal ar: std_logic_vector(15 downto 0);
signal ir: std_logic_vector(7 downto 0);
signal dr: std_logic_vector(7 downto 0);
signal tr: std_logic_vector(7 downto 0);
signal z: std_logic;

signal arinc: std_logic;
signal writel: std_logic;
signal pcbus: std_logic;
signal membus: std_logic;
signal rbus: std_logic;
signal acbus: std_logic;
signal trbus: std_logic;
signal drbus: std_logic;
signal pcload: std_logic;
signal arload: std_logic;
signal drload: std_logic;
signal irload: std_logic;
signal acload: std_logic;
signal rload:  std_logic;
signal trload: std_logic;
signal nextpc: std_logic_vector(15 downto 0);
signal state: std_logic_vector(5 downto 0);
signal nextstate: std_logic_vector(5 downto 0)
```

```
constant fetch1:    std_logic_vector(5 downto 0) := "000000";
constant fetch2:    std_logic_vector(5 downto 0) := "000001";
constant fetch3:    std_logic_vector(5 downto 0) := "000010";

constant clacl:     std_logic_vector(5 downto 0) := "000100";
constant incacl:    std_logic_vector(5 downto 0) := "000101";
constant addl:      std_logic_vector(5 downto 0) := "000110";
constant subl:      std_logic_vector(5 downto 0) := "000111";
constant andl:      std_logic_vector(5 downto 0) := "001000";
constant orl:       std_logic_vector(5 downto 0) := "001001";
constant xorl:      std_logic_vector(5 downto 0) := "001010";
constant notl:      std_logic_vector(5 downto 0) := "001011";
constant mvacl:     std_logic_vector(5 downto 0) := "001100";
constant movrl:     std_logic_vector(5 downto 0) := "001101";
constant ldacl:     std_logic_vector(5 downto 0) := "001110";
constant ldac2:     std_logic_vector(5 downto 0) := "001111";
constant ldac3:     std_logic_vector(5 downto 0) := "010000";
constant ldac4:     std_logic_vector(5 downto 0) := "010001";
constant ldac5:     std_logic_vector(5 downto 0) := "010010";
constant stacl:     std_logic_vector(5 downto 0) := "010011";
constant stac2:     std_logic_vector(5 downto 0) := "010100";
constant stac3:     std_logic_vector(5 downto 0) := "010101";
constant stac4:     std_logic_vector(5 downto 0) := "010110";
constant stac5:     std_logic_vector(5 downto 0) := "010111";
```

cpu.vhd:

```
-- update pc, state and other registers
update_regs: process(clk)
if(arload='1')  then    ar<=bus16;
end if;
if(drload='1')  then    dr<=bus16(7 downto 0);
end if;
if(irload='1')  then    ir<=dr;
end if;
for_nextstate: process(state, ir, z)
begin

    if(state=fetch1)    then    nextstate<=fetch2;
    elsif(state=fetch2) then    nextstate<=fetch3;
    elsif(state=fetch3) then
        if(dr=RSCLAC)       then    nextstate<=clac1
        elsif(dr=RSINAC)    then    nextstate<=incac
        elsif(dr=RSADD)     then    nextstate<=add1;
        elsif(dr=RSSUB)     then    nextstate<=sub1;
        elsif(dr=RSAND)     then    nextstate<=and1;
        elsif(dr=RSOR)      then    nextstate<=or1;
        elsif(dr=RSXOR)     then    nextstate<=xor1;
        elsif(dr=RSNOT)     then    nextstate<=not1;
        elsif(dr=RSMVAC)    then    nextstate<=mvac1
        elsif(dr=RSMOVR)    then    nextstate<=movr1;
        elsif(dr=RSLDAC)    then    nextstate<=ldac1;
        elsif(dr=RSSTAC)    then    nextstate<=stac1;
        elsif(dr=RSJUMP)    then    nextstate<=jump1;
```

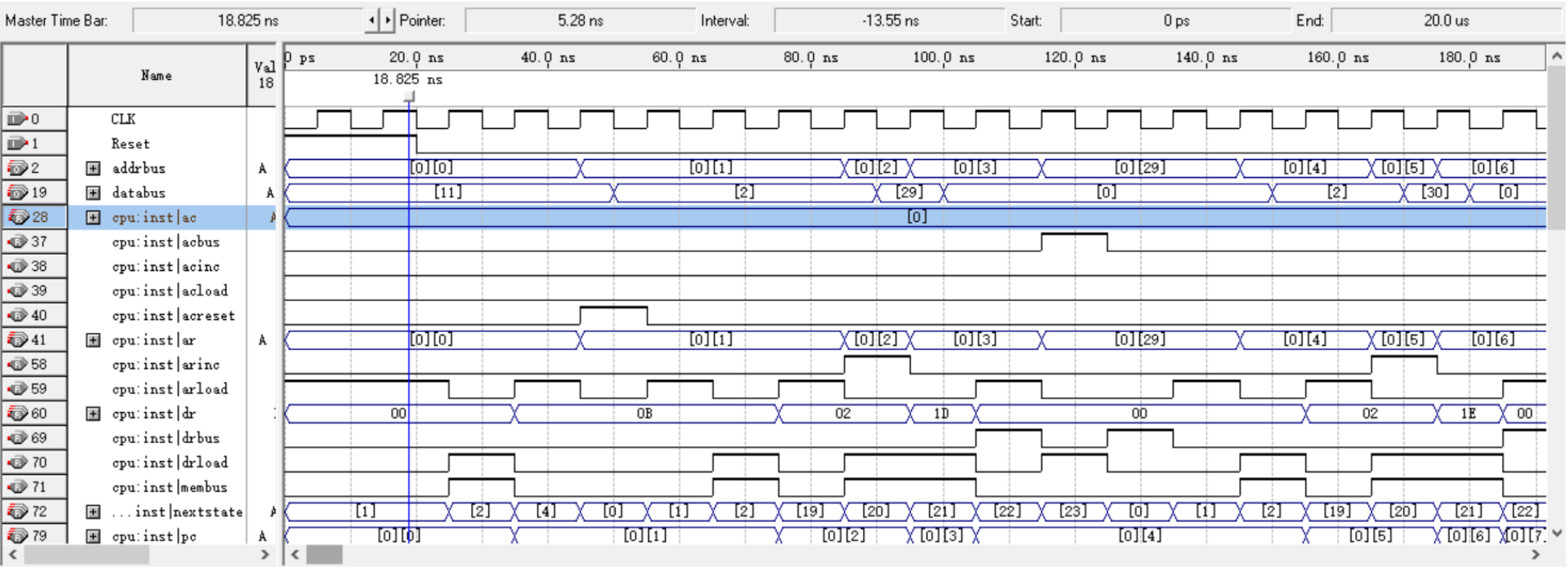
```
-- generate control signals for each state
gen_controls: process(state)
begin
    if(state=fetch1)    then
    arload<='1';
    pcbus<='1';
    pcinc<='0';
    drload<='0';
    membus<='0';
    irload<='0';
    acreset<='0';
    acinc<='0';
    rbus<='0';
    s<="0000";acload<='0';
    rload<='0';acbus<='0';
    arinc<='0';trbus<='0';
    drbus<='0';trload<='0';
    read<='0';write1<='0';
    write<='0';pcload<='0';
```

mem.vhd

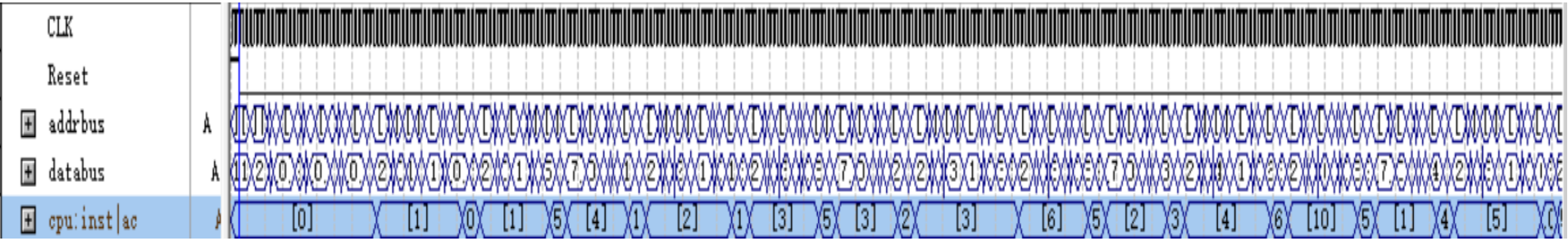
```
architecture mem_behav of mem is
    signal addr: std_logic_vector(15 downto 0);

    type memtype is array(natural range<>) of st
    constant total_addr : integer := 29;
    constant i_addr : integer := 30;
    constant n_addr : integer := 31;
    constant loop_addr : integer := 7;
```

```
2 => std_logic_vector(to_unsigned(total_addr, 8)),
3 => X"00",
4 => RSSTAC,--i=0
5 => std_logic_vector(to_unsigned(i_addr, 8)),
6 => X"00",
7 => RSLDAC,  -- loop
8 => std_logic_vector(to_unsigned(i_addr, 8)),
9 => X"00",
10 => RSINAC,
11 => RSSTAC,
12 => std_logic_vector(to_unsigned(i_addr, 8)),
13 => X"00",
14 => RSMVAC,
15 => RSLDAC,
16 => std_logic_vector(to_unsigned(total_addr, 8)),
17 => X"00",
18 => RSADD, --ac=ac+r
19 => RSSTAC,
20 => std_logic_vector(to_unsigned(total_addr, 8)),
21 => X"00",
22 => RSLDAC,
23 => std_logic_vector(to_unsigned(n_addr, 8)),
24 => X"00",
25 => RSSUB,
26 => RSJPNZ,
27 => std_logic_vector(to_unsigned(loop_addr, 8)),
28 => X"00",
29 => X"00",  -- total
30 => X"00",  -- i
31 => "00000101",  -- n
```

仿真结果


Master Time Bar: 18.825 ns
Pointer: 5.28 ns
Interval: -13.55 ns
Start: 0 ps
End: 20.0 us
Name
Val
18.825 ns
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
100.0 ns
120.0 ns
140.0 ns
160.0 ns
180.0 ns
0 CLK
1 Reset
2 addrbus
19 databus
28 cpu:inst|ac
37 cpu:inst|acbus
38 cpu:inst|acinc
39 cpu:inst|acload
40 cpu:inst|acreset
41 cpu:inst|ar
58 cpu:inst|arinc
59 cpu:inst|arload
60 cpu:inst|dr
69 cpu:inst|drbus
70 cpu:inst|drload
71 cpu:inst|membus
72 ...inst|nextstate
79 cpu:inst|pc

仿真结果



指令	1st Loop	2nd Loop	3rd Loop	4th Loop	5th Loop
CLAC	AC＝0				
STAC total	total＝0				
STAC i	i＝0				
LDAC i	AC＝0	AC＝1	AC＝2	AC＝3	AC＝4
INAC	AC＝1	AC＝2	AC＝3	AC＝4	AC＝5
STAC i	i＝1	i＝2	i＝3	i＝4	i＝5
MVAC	R＝1	R＝2	R＝3	R＝4	R＝5
LDAC total	AC＝0	AC＝1	AC＝3	AC＝6	AC＝10
ADD	AC＝1	AC＝3	AC＝6	AC＝10	AC＝15
STAC total	total＝1	total＝3	total＝6	total＝10	total＝15
LDAC n	AC＝5	AC＝5	AC＝5	AC＝5	AC＝5
SUB	AC＝4, Z＝0	AC＝3, Z＝0	AC＝2, Z＝0	AC＝1, Z＝0	AC＝0, Z＝0
JPNZ Loop	JUMP	JUMP	JUMP	JUMP	NO JUMP

THANKS!