تحلیل کد کامپیوتر پایه

امیرحسام مرادی و سارا روحانی

مقدمه

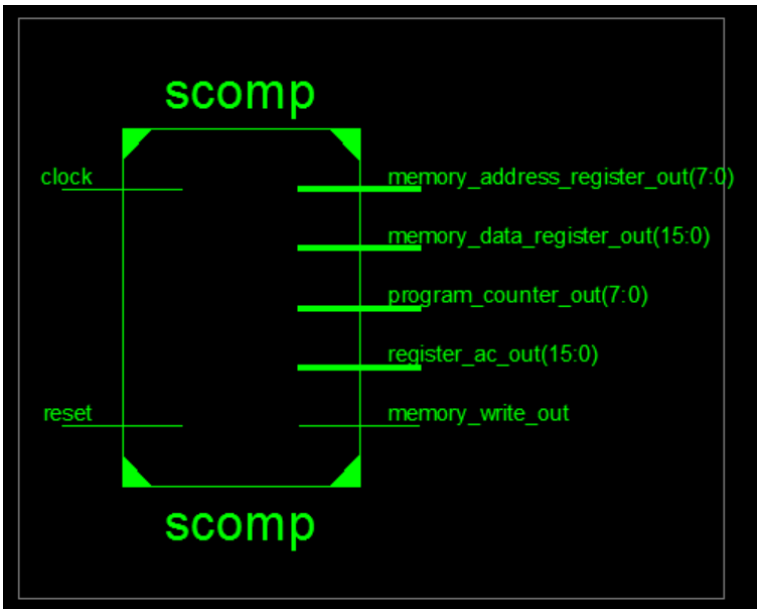
یک کامپیوتر پایه از یک مسیر داده ، تعدادی ثبات و واحد کنترل کننده در کنار یک حافظه تشکیل می شود . مسیر داده انتقال داده بین ثبات ها را ممکن می کند و واحد کنترل کننده با مقدار دهی به پایه های ثبات ها مثل ... , Load, reset و همچین کنترل استفاده ثبات ها از مسیر داده ، الگوریتم فون نیومن را اجرا می کند .

به طور کلی یک کامپیوتر پایه ترکیب یک حافظه و یک پردازنده است ، به طوری که پردازنده دستورات را از حافظه می خواند و پس از اجرای دستورات بر اساس الگوریتم نیومن ، نتایج را در حافظه ذخیره می کند .

تحلیل کد

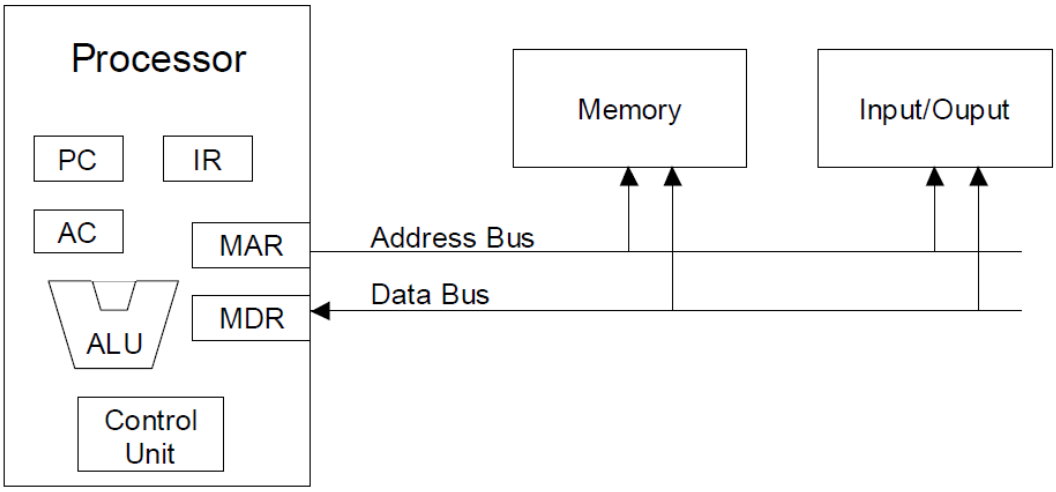
برای این پیاده سازی از حافظه ای با ارتفاع ۲۵۶ که هر کلمه آن ۱۶ بیت دارد ، استفاده شده است ، پس تمام ثبات هایی که آدرس نگه می دارند ۸ بیتی و ثبات های نگه دارنده داده ۱۶ بیتی هستند . در کد داده شده این مقادیر به صورت generic در entity تعریف شده است .

این ماژول دو ورودی تک بیتی کلاک و ریست دارد و مقادیر موجود در ثبات های PC,AC,MAR,MDR و سیگنال دستور نوشتن در حافظه را در خروجی قرار می دهد



```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity scomp is
generic( address_width : integer := 8;
data_width : integer := 16);
   port(
   clock, reset : in std_logic:= '1';
program_counter_out : out std_logic_vector(address_width-1 downto 0);
register_ac_out : out std_logic_vector(data_width-1 downto 0);
memory_data_register_out : out std_logic_vector(data_width-1 downto 0);
memory_address_register_out : out std_logic_vector(address_width-1 downto 0);
memory_write_out : out std_logic
);
```

برای پیاده سازی کامپیوتر پایه معمولا از دو نوع ثبات عام منظوره و خاص منظوره استفاده می شود ، اما در این کد صرفا از ثبات های خاص منظوره استفاده شده است .



ثبات PC برای ذخیره کردن آدرس دستوری که در نوبت بعدی اجرا است

ثبات IR برای ذخیره کردن دستوری که نوبت اجرای آن است

ثبات AC برای ذخیره حاصل جمع مقدار خوانده شده از حافظه و مقدار قبلی AC

ثبات MAR برای فرستادن آدرس به حافظه

ثبات MDR برای ذخیره داده برای گرفتن از یا فرستادن به حافظه

در ادامه کد ثبات ها و سیگنال دستور نوشتن در حافظه و همچنین حافظه با ابعاد خواسته شده به صورت یک آرایه دو بعدی تعریف شده است .

```
16  architecture rtl of scomp is
17  type ram is array(0 to 2 ** address_width-1) of unsigned(data_width-1 downto 0);
18  signal ram_block : ram;
19  attribute ram_init_file : string;
20  attribute ram_init_file of ram_block : signal is "program.mif";
21  type scomp_fsm is ( reset_pc, fetch, decode, execute_add, execute_load,
22  execute_jneg,execute_jneg2, execute_store,
23  execute_store2, execute_jump );
24  signal state : scomp_fsm;
25  signal instruction_register : unsigned(data_width-1 downto 0);
26  signal memory_data_register : unsigned(data_width-1 downto 0);
27  signal register_ac : signed(data_width-1 downto 0);
28  signal program_counter : unsigned(address_width-1 downto 0);
29  signal memory_address_register : unsigned(address_width-1 downto 0);
30  signal memory_write : std_logic;
```

در کد بالا برای پیاده سازی ترتیب الگوریتم فون نیومن حالت هایی در نظر گرفته شده است . حالت reset برای تنظیمات اولیه ، fetch برای گرفتن دستور ، decode برای فهمیدن دستور و انواع execute برای اجرای دستورات .

در بین دستورات ، دستور add, load, jump را می توان در یک کلاک اجرا کرد ، اما برای اجرای دستورات store و jneg به دو کلاک نیاز است ، به همین دلیل برای این ۲ دستور ، ۲ حالت در نظر گرفته شده است .

در ادامه ثبات هایی که مقادیر آن ها باید در خروجی نمایش داده شود به خروجی های مورد نظر متصل شده است .

```
32  -- Output major signals
33  program_counter_out <= std_logic_vector(program_counter);
34  register_AC_out <= std_logic_vector(register_AC);
35  memory_data_register_out <= std_logic_vector(memory_data_register);
36  memory_address_register_out <= std_logic_vector(memory_address_register);
37  memory_write_out <= memory_write;
```

در ادامه اولین process برای خواندن و نوشتن از حافظه پیاده سازی شده است . به طوری که با آمدن لبه بالارونده کلاک اگر سیگنال memory_write منطق یک داشته باشد ، مقدار موجود در ثبات AC در خانه ای از حافظه که آدرس آن در ثبات MAR قرار دارد ، ذخیره می شود و اگر این سیگنال منطق صفر داشته باشد ، مقدار موجود در خانه ای از حافظه که آدرس آن در MAR قرار دارد ، در ثبات MDR ریخته می شود .

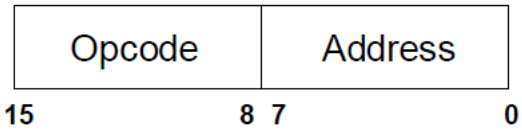
```
38  process (clock)
39  begin
40  if rising_edge(clock) then
41  if (memory_write = '1') then
42  ram_block(to_integer(memory_address_register)) <= unsigned(register_ac);
43  end if;
44  memory_data_register <= ram_block(to_integer(memory_address_register));
45  end if;
46  end process;
```

در دومین process نحوه تغییر حالت سیستم توصیف شده است . در این process ابتدا سیگنال ورودی reset چک می شود ، اگر ۱ باشد ، سیستم به حالت اولیه می رود و اگر صفر باشد ، حالت بعدی سیستم بر اساس حالت فعلی و با در نظر گرفتن الگوریتم فون نیومن تعیین می شود .

```
47  process (clock,reset)
48  begin
49  if reset = '1' then
50  state <= reset_pc;
51  elsif rising_edge(clock) then
52  case state is
53  -- reset the computer, need to clear some registers
54  when reset_pc =>
55  program_counter <= (others => '0');
56  register_ac <= (others => '0');
57  state <= fetch;
58  -- fetch instruction from memory and add 1 to pc
59  when fetch =>
60  instruction_register <= memory_data_register;
61  program_counter <= program_counter + 1;
62  state <= decode;
```

اگر سیستم در حالت ریست باشد ، در ثبات های PC و MAR مقدار صفر ریخته می شود و سیستم به حالت fetch می رود تا دستور بعدی را بخواند و بعد از این حالت دستگاه به حالت تفسیر دستور می رود .

این تفسیر بر اساس قالب دستور العمل انجام می شود و قالب دستور العمل به صورت زیر است .



در این قالب ۱۶ بیتی ، از ۸ بیت سمت چپ آن برای مشخص کردن نو عملیات و از ۸ بیت سمت راست آن برای نگه داری آدرس عملوند استفاده شده است .

تمام عملیات های در نظر گرفته برای این کامپیوتر پایه یک عملوند احتیاج دارد .

Instruction Mnemonic	Operation Preformed	Opcode Value
ADD address	AC <= AC + contents of memory address	00
STORE address	Contents of memory address <= AC	01
LOAD address	AC <= contents of memory address	02
JUMP address	PC <= address	03
JNEG address	IF AC<0 THEN PC <= address	04

مقدار opcode عملیات های این کامپیوتر پایه در سمت راست جدول آورده شده است . پس حالت بعد از حالت fetch بر اساس مقدار opcode موجود در دستور به صورت زیر تعیین می شود .

```
63  -- decode instruction and send out address of any data operands
64  when decode =>
65  case instruction_register( 15 downto 8 ) is
66  when "00000000" =>
67  state <= execute_add;
68  when "00000001" =>
69  state <= execute_store;
70  when "00000010" =>
71  state <= execute_load;
72  when "00000011" =>
73  state <= execute_jump;
74  when "00000100" =>
75  state <= execute_jneg;
76  when others =>
77  state <= fetch;
78  end case;
```

بعد از مشخص شدن نوع عملیات مورد نظر ، سیستم به حالت اجرای آن عملیات می رود .

```
79  -- execute the add instruction
80  when execute_add =>
81  register_ac <= register_ac + signed(memory_data_register);
82  state <= fetch;
83  -- execute the store instruction
84  -- (needs two clock cycles for memory write and fetch mem setup)
85  when execute_store =>
86  -- enable memory write, write register_ac to memory
87  -- load memory address and data registers for memory write
88  state <= execute_store2;
89  -- finish memory write operation and load memory registers
90  -- for next fetch memory read operation
91  when execute_store2 =>
92  state <= fetch;
93  -- execute the load instruction
94  when execute_load =>
95  register_ac <= signed(memory_data_register);
96  state <= fetch;
97  -- execute the jump instruction
98  when execute_jump =>
99  program_counter <= instruction_register(address_width-1 downto 0);
100 state <= fetch;
```

برای عملیات add مقداری که از حافظه در ثبات MDR ذخیره شده است با مقدار موجود در ثبات AC جمع و حاصل در خود ثبات AC قرار می گیرد و سیستم به حالت fetch می رود .

برای عملیات store ابتدا در کلاک اول سیگنال memory_write یک می شود و در ثبات MAR آدرس خانه مورد نظر برای ذخیره و در ثبات MDR مقدار مورد نظر برای ذخیره در حافظه قرار می گیرد و در کلاک دوم این مقدار در حافظه ذخیره شده و سیستم به حالت fetch می رود .

برای عملیات load مقدار موجود در ثبات MDR به ثبات AC ریخته می شود که این مقدار از خانه ای از حافظه که آدرس آن در ۸ بیت سمت راست دستور المعل نوشته شده است به ثبات MDR ریخته شده است و در انتها سیستم به حالت fetch می رود .

برای عملیات jump آدرس موجود در ۸ بیت سمت راست دستورالعمل در ثبات PC ریخته می شود و سیستم به حالت fetch می رود .

برای عملیات jneg در کلاک اول مقدار موجود در ثبات AC بررسی می شود اگر بیشتر از صفر باشد ، در کلاک بعدی کاری انجام نخواهد شد (تعریف سیگنال ثبات AC به صورت singed انجام شده است) و اگر این مقدار کمتر از صفر باشد ، در کلاک آدرس موجود در ۸ بیت سمت راست دستور المعل در ثبات PC ریخته شده و سیستم به حالت fetch می رود

با تعیین تکلیف تمام حالت های تعریف شده برای سیستم این process تمام می شود .

```
101  when execute_jneg =>
102  if (register_ac < 0) then
103  program_counter <= instruction_register(address_width-1 downto 0);
104  end if;
105  state <= execute_jneg2;
106  when execute_jneg2 =>
107  state <= fetch;
108  when others =>
109  state <= fetch;
110  end case;
111  end if;
112  end process;
```

در آخر با استفاده از ساختار with/select مقدار ذخیره شده در ثبات MAR و مقدار سیگنال memory_write بر اساس حالت دستگاه مشخص می شود .

```
113  -- memory address register is already inside synchronous memory unit
114  -- need to load its value based on current state
115  -- (no second register is used - not inside a process here)
116  with state select
117  memory_address_register <= (others => '0') when reset_pc,
118  program_counter when fetch,
119  instruction_register(address_width-1 downto 0) when decode,
120  program_counter when execute_add,
121  instruction_register(address_width-1 downto 0) when execute_store,
122  program_counter when execute_store2,
123  program_counter when execute_load,
124  instruction_register(address_width-1 downto 0) when execute_jump,
125  program_counter when execute_jneg,
126  program_counter when execute_jneg2;
```

سیستم در حالت های fetch, execute_add, execute_store2, execute_load, execute_jneg, execute_jneg2 مقدار موجود در ثبات PC را در MAR می ریزد و در حالت های decode, execute_store, execute_jump قرار موجود در ۸ بیت سمت راست ثبات IR را در MAR قرار می دهد .

```
127  with state select
128  memory_write <= '1' when execute_store,
129  '0' when others;
130  end rtl;
```

سیستم فقط در حالت execute_store مقدار سیگنال memory_write را یک می کند و در سایر حالت مقدار صفر را برای این سیگنال در نظر می گیرد .

با این مقدار دهی ها عملیات های درخواستی به درستی انجام خواهد شد .